Ατομική Διπλωματική Εργασία

ΠΡΟΒΛΕΨΗ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΠΡΩΤΕΙΝΩΝ ΜΕ ΧΡΗΣΗ ΣΥΝΕΛΙΚΤΙΚΩΝ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΣΕ ΣΥΝΔΥΑΣΜΟ ΜΕ ΦΙΛΤΡΑ GABOR ΚΑΙ SUPPORT VECTOR MACHINES

Αντρέας Διονυσίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2018

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών με Χρήση Συνελικτικών Νευρωνικών Δικτύων σε Συνδυασμό με Φίλτρα Gabor και Support Vector Machines

Αντρέας Διονυσίου

Επιβλέπων Καθηγητής
Δρ. Χρίστος Χριστοδούλου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2018

Ευχαριστίες

Ένας μεγάλος αγώνας έφτασε στο τέλος του. Ένας φοιτητής εκπονώντας την διπλωματική του εργασία, μπορεί να συνειδητοποιήσει το πώς η έρευνα και το επιστημονικό κομμάτι της σπουδής του, μπορεί να αποβεί χρήσιμο στην κοινωνία. Η κοινωνία που ζούμε σήμερα αλλά και η εξέλιξη του ανθρώπινου είδους, βασίζεται εξολοκλήρου στην πρόθεση κάποιων ανθρώπων μιας κοινωνίας να διαθέσουν χρόνο και μόχθο, έτσι ώστε να προσπαθήσουν να ανακαλύψουν κάτι καινούριο, διαφορετικό, το οποίο στο μέλλον θα θεωρείται δεδομένο. Ο εν κατακλείδι σκοπός μιας διπλωματικής εργασίας, είναι να δώσει στον κάθε φοιτητή τα κατάλληλα ερεθίσματα για την επιστημονική του επαγρύπνηση.

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω μέσα από τα βάθη της καρδιάς μου, τον αναπληρωτή καθηγητή Δρ. Χρίστο Χριστοδούλου, για την καθοδήγηση, την παροχή ερεθισμάτων και την εμπιστοσύνη που μου έδειξε κατά την διάρκεια εκπόνησης της διπλωματικής μου εργασίας, αφού αποτέλεσε τον πυρήνα και το μεγαλύτερο στήριγμα, για την επιτυχή ολοκλήρωση της έρευνας μου. Επίσης, θα ήθελα να ευχαριστήσω τον επίκουρο καθηγητή Δρ. Βασίλη Προμπονά, του οποίου οι γνώσεις στο βιολογικό κυρίως κομμάτι, έπαιξαν εξαιρετικά σημαντικό ρόλο για την επιτυχή ολοκλήρωση της έρευνας μου.

Ιδιαίτερες ευχαριστίες θα ήθελα επίσης να δώσω, στο διδακτορικό φοιτητή Μιχάλη Αγαθοκλέους, ο οποίος υπήρξε τεράστιο στήριγμα σε θέματα γνώσεων, αφού ανατροφοδοτούσε τις διάφορες προσπάθειες μου για επίλυση του συγκεκριμένου προβλήματος, με βάση τις δικές του γνώσεις και εμπειρίες στο συγκεκριμένο θέμα.

Τέλος, ίσως οι σημαντικότερες ευχαριστίες κατανέμονται στην οικογένεια μου, στην οποία οφείλω όλα όσα έχω, και είμαι, σήμερα. Η αμέριστη στήριξη που είχα από την οικογένεια μου, με οδήγησε στο να καταφέρω να ολοκληρώσω έναν από τους μεγαλύτερους κύκλους, αν όχι το μεγαλύτερο κύκλο της ζωής του ανθρώπου, τις προπτυχιακές σπουδές. Αφιερώνω λοιπόν, την συγκεκριμένη διπλωματική εργασία στην οικογένεια μου, αφού είναι το ελάχιστο ευχαριστώ που μπορώ να προσφέρω, απέναντι στην ανιδιοτελή τους υποστήριξη καθ’ όλη τη διάρκεια των σπουδών μου.

Περίληψη

Αυτή η διπλωματική εργασία, αφορά το θέμα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών με χρήση συνελικτικών νευρωνικών δικτύων σε συνδυασμό με φίλτρα Gabor και support vector machines. Αρχικά, οι πρωτεΐνες αποτελούν ένα από τα σημαντικότερα στοιχεία του ανθρωπίνου σώματος, αλλά και κάθε ζωντανού οργανισμού. Η παρουσία των πρωτεϊνών στα κύτταρα είναι απαραίτητη προκειμένου αυτά, να εκτελέσουν κανονικά τις λειτουργίες του οργανισμού. Η δομή των πρωτεϊνών χωρίζεται σε τέσσερις κατηγορίες, την πρωτοταγή, τη δευτεροταγή, την τριτοταγή και την τεταρτοταγή δομή. Ιδιαίτερης σημασίας χρίζει ο καθορισμός της ακριβούς δομής των πρωτεϊνών στον τρισδιάστατο χώρο, αφού γνωρίζοντας την ακριβή δομή των πρωτεϊνών στον τρισδιάστατο χώρο (τριτοταγής δομή), μπορούμε να μελετήσουμε τις πρωτεΐνες αυτές σε λειτουργικό και συνεπώς ιατροφαρμακευτικό επίπεδο.

Οι πειραματικές μέθοδοι τις οποίες μπορούμε να χρησιμοποιήσουμε σήμερα για τον καθορισμό της τριτοταγούς δομής των πρωτεϊνών, είναι εξαιρετικά χρονοβόρες, πολύπλοκες και έχουν υψηλό κόστος. Η δευτεροταγής δομή μιας πρωτεΐνης εξαρτάται από την αλληλουχία των αμινοξέων (πρωτοταγής δομή) μιας πρωτεΐνης. Για τον καθορισμό της τριτοταγούς δομής μιας πρωτεΐνης, μπορεί να χρησιμοποιηθεί ως ενδιάμεσο βήμα η δευτεροταγής δομή της συγκεκριμένης πρωτεΐνης, συνεπώς αναπτύχθηκαν αλγόριθμοι πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, χρησιμοποιώντας σαν είσοδο, την πρωτοταγή δομή της πρωτεΐνης. Η έρευνα αυτή ασχολείται αρχικά με την μελέτη των πρωτεϊνών, ως επίσης και διαφόρων τεχνικών που χρησιμοποιήθηκαν για την πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών (Protein Secondary Structure Prediction - PSSP). Στην συνέχεια, η έρευνα αυτή, ασχολείται με το πώς τα νευρωνικά δίκτυα (Neural Networks - NNs), μπορούν να δώσουν λύση στο PSSP πρόβλημα, με την επέκταση των αλγορίθμων/τεχνικών που χρησιμοποιήθηκαν, συμπεριλαμβάνοντας και τη χρήση Συνελικτικών Νευρωνικών Δικτύων (Convolutional Neural Networks - CNNs), σε συνδυασμό με τα φίλτρα Gabor και Support Vector Machines (SVMs). Το μεγαλύτερο ποσοστό επιτυχίας στην πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών Q3 (Εξίσωση 1.1) το οποίο πετύχαμε στη συγκεκριμένη έρευνα είναι ***80.40%*** και Segment Overlap (Rost et al., 1994) ***0.736*** χρησιμοποιώντας το CB513 dataset (Cuff and Barton, 1999).

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή 1

1.1 Πρωτεΐνες και Αμινοξέα 2

1.2 Το πρόβλημα 5

1.3 Μηχανική Μάθηση, Νευρωνικά Δίκτυα και Φίλτρα Gabor 6

1.4 Προηγούμενη Σχετική Έρευνα 9

Κεφάλαιο 2 Βιολογικό Υπόβαθρο 20

2.1 Πρωτεΐνες και Αμινοξέα 21

2.2 Ο Βιολογικός Ρόλος των Πρωτεϊνών 26

2.3 Επίπεδα Οργάνωσης Πρωτεϊνών 27

2.3.1 Πρωτοταγής Δομή 28

2.3.2 Δευτεροταγής Δομή 29

2.3.3 Τριτοταγής Δομή 30

2.3.4 Τεταρτοταγής Δομή 30

2.4 Σημασία Γνώσης της Δομής των Πρωτεϊνών 31

2.5 Πρόβλεψη της Δομής των Πρωτεϊνών 32

Κεφάλαιο 3 Τεχνητή Νοημοσύνη και Τεχνητά Νευρωνικά Δίκτυα 35

3.1 Τεχνητή Νοημοσύνη (Artificial Intelligence) 37

3.2 Μηχανική Μάθηση (Machine Learning) 39

3.2.1 Γενικά 39

3.2.2 Είδη Μάθησης 40

3.2.2.1 Επιβλεπόμενη Μάθηση (Supervised Learning) 40

3.2.2.2 Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning) 41

3.2.2.3 Ενισχυτική Μάθηση (Reinforcement Learning) 42

3.3 Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks) 44

3.3.1 Πηγή Έμπνευσης Τεχνητών Νευρωνικών Δικτύων 44

3.3.2 Αρχιτεκτονική και Λειτουργία Τεχνητών Νευρωνικών Δικτύων 46

3.3.3 Μοντέλο Νευρώνα McCulloch και Pitts 49

3.3.4 Πολυστρωματικά Δίκτυα Perceptron Εμπρόσθιου Περάσματος 51
3.3.4.1 Εισαγωγή στα Δίκτυα Perceptron 51
3.3.4.2 Αλγόριθμος Μάθησης Perceptron 53
3.3.4.3 Κανόνας Δέλτα 55
3.3.4.4 Μέθοδος Κατάβασης Κλίσης 55
3.3.4.5 Παράγοντας Ορμής 56
3.3.4.6 Αλγόριθμος Ανάστροφης Μετάδοσης Σφάλματος 58
3.3.5 Συνελικτικά Νευρωνικά Δίκτυα 60
3.3.5.1 Εισαγωγή στα Συνελικτικά Νευρωνικά Δίκτυα 60
3.3.5.2 Αρχιτεκτονική Συνελικτικών Νευρωνικών Δικτύων 61
3.3.5.3 Επεξήγηση Λειτουργίας Συνελικτικών Νευρωνικών Δικτύων 66
3.3.6 Support Vector Machines (SVMs) 71
3.3.6.1 Εισαγωγή στα SVMs 71
3.3.6.2 Επεξήγηση Βασικής Λειτουργίας του SVM 72
3.3.6.3 Συνδυασμός CNNs και SVMs 73
3.3.7 Συναρτήσεις Ενεργοποίησης 73
3.3.7.1 Βηματική Συνάρτηση – Συνάρτηση Σκαλί 73
3.3.7.2 Σιγμοειδής Συνάρτηση 74
3.3.7.3 Συνάρτηση Rectifier 76
3.3.7.4 Softmax Συνάρτηση 77
3.3.8 Vanishing και Exploding Gradient Problem 78
3.3.9 Updaters για Διαχείριση της Μεταβολής του Ρυθμού Μάθησης 79
3.3.9.1 Γενικά 79
3.3.9.2 Ορμή 80
3.3.9.3 Adagrad (Adaptive Gradient Algorithm) 81
3.3.9.4 RMSProp 82
3.3.9.5 AdaDelta 82
3.3.9.6 ADAM (Adaptive Moment Estimation) 82

Κεφάλαιο 4 Φίλτρα Gabor 83
4.1 Εισαγωγή στα Φίλτρα Gabor 84
4.2 Μαθηματικός Ορισμός Φίλτρου Gabor 85
4.3 Σύνδεση Φίλτρων Gabor με Συνελικτικά Νευρωνικά Δίκτυα 86
Κεφάλαιο 5 Δεδομένα Εισόδου 88
5.1 Δεδομένα Εκπαίδευσης και Επαλήθευσης 89
5.1.1 Γενικά 89
5.1.2 Μεθοδολογία Επιλογής Δεδομένων Εισόδου 89
5.2 Βάσεις Δεδομένων Πρωτεϊνών και DSSP Κωδικοποίηση 91
5.3 Αρχεία Δεδομένων Πρωτεϊνών 92
5.4 Αρχεία Multiple Sequence Alignment (MSA) 93
5.5 Κατασκευή Δεδομένων Εισόδου με Βάση τα Αρχεία MSA 94
5.6 Χρήση Batches για Εκπαίδευση Νευρωνικών Δικτύων 97
5.7 Εφαρμογή Φίλτρων Gabor στα Αρχεία Εισόδου 99
5.8 Τεχνική Σημαντικότερων Γειτονικών Αμινοξέων 101
5.9 CB513 και PISCES Datasets 104
5.10 Segment Overlap (SOV) 106
5.11 Τεχνική Συνδυασμού CNN με BRNN 107
Κεφάλαιο 6 Πειράματα και Ανάνυση Αποτελεσμάτων 108
6.1 Βιβλιοθήκη DeepLearning4j – DL4J 111
6.1.1 Γενικά 111
6.1.2 Προσαρμογή Βιβλιοθήκης για το PSSP πρόβλημα 111
6.2 Πειράματα με Χρήση Αρχείων MSA 113
6.2.1 Περιγραφή 113
6.2.2 Πειράματα με Χρήση Διαφορετικού Αριθμού Κρυφών Επιπέδων 114
6.2.3 Πειράματα με Χρήση Διαφορετικών Μεγεθών Kernel 118
6.2.4 Πειράματα με Χρήση Διαφορετικής Τιμής Ορμής 122
6.2.5 Πειράματα με Χρήση Διαφορετικού Stride 125
6.2.6 Πειράματα με Χρήση Διαφορετικού Αριθμού Παράλληλων Φίλτρων 129

6.2.7 Πειράματα με Χρήση Διαφορετικών Πλάνων Ενημέρωσης Ρυθμού Μάθησης 133
6.2.8 Πειράματα με Χρήση Max Pooling Layers 136
6.2.9 Πειράματα με Χρήση Διαφορετικών Μεθόδων Ενημέρωσης του Ρυθμού Μάθησης 140
6.2.10 Συμπεράσματα για Πειράματα με Χρήση Αρχείων MSA 143
6.3 Πειράματα με Convolution Μεγάλων Gabor Φίλτρων 147
6.3.1 Περιγραφή 147
6.3.2 Πειραματικό Μέρος 147
6.3.3 Συμπεράσματα 150
6.4 Πειράματα με Convolution Kernels Gabor Φίλτρων 150
6.4.1 Περιγραφή 150
6.4.2 Πειραματικό Μέρος 151
6.4.3 Συμπεράσματα 154
6.5 Πειράματα με Χρήση Μέσου Όρου Γειτονικών Κελιών 154
6.5.1 Περιγραφή 154
6.5.2 Πειραματικό Μέρος 155
6.5.3 Συμπεράσματα 157
6.6 Πειράματα με Χρήση Τεχνικής Σημαντικότερων Γειτονικών Αμινοξέων 158
6.6.1 Περιγραφή 158
6.6.2 Πειραματικό Μέρος 159
6.6.3 Συμπεράσματα 161
6.7 Πειράματα με Χρήση της Μεθόδου Conjugate Gradient 162
6.7.1 Περιγραφή 162
6.7.2 Πειραματικό Μέρος 163
6.7.3 Συμπεράσματα 166
6.8 Πειράματα με Χρήση του PISCES Dataset 167
6.8.1 Περιγραφή 167
6.8.2 Πειραματικό Μέρος 167
6.8.3 Συμπεράσματα 170

6.9 Πειράματα με Συνδυασμό CNN και BRNN 170
6.9.1 Περιγραφή 170
6.9.2 Πειραματικό Μέρος 170
6.9.3 Συμπεράσματα 172
6.10 Πειράματα με Συνδυασμό CNN και SVM 173
6.10.1 Περιγραφή 173
6.10.2 Πειραματικό Μέρος 173
6.10.3 Συμπεράσματα 175
Κεφάλαιο 7 Συμπεράσματα και Μελλοντική Εργασία 176
7.1 Συμπεράσματα 177
7.2 Μελλοντική Εργασία 186
Αναφορές 188
Γενική Βιβλιογραφία 195
Παράρτημα Α Α-1
Παράρτημα Β Β-1
Παράρτημα Γ Γ-1
Παράρτημα Δ Δ-1
Παράρτημα Ε Ε-1
Παράρτημα Ζ Ζ-1
Παράρτημα Η Η-1
Παράρτημα Θ Θ-1
Παράρτημα Ι Ι-1

Κεφάλαιο 1

Εισαγωγή

1.1	Πρωτεΐνες και Αμινοξέα	2
1.2	Το πρόβλημα	5
1.3	Μηχανική Μάθηση, Νευρωνικά Δίκτυα και Φίλτρα Gabor	6
1.4	Προηγούμενη Σχετική Έρευνα	9

1.1 Πρωτεΐνες και Αμινοξέα

Αρχικά, οι πρωτεΐνες αποτελούν ένα από τα σημαντικότερα στοιχεία του ανθρωπίνου σώματος, αλλά και κάθε ζωντανού οργανισμού. Η παρουσία των πρωτεϊνών στα κύτταρα είναι απαραίτητη προκειμένου αυτά, να εκτελέσουν κανονικά τις λειτουργίες του οργανισμού. Συγκεκριμένα, μπορούμε να πούμε ότι τα διαφορετικά είδη πρωτεϊνών, είτε σαν δομικές είτε σαν λειτουργικές μονάδες, διασφαλίζουν τις περισσότερες ανάγκες για τη βιολογική λειτουργία κάθε βιολογικά ζωντανού οργανισμού. Οι πρωτεΐνες αποτελούνται από αμινοξέα τα οποία ενώνονται μεταξύ τους, σχηματίζοντας έτσι τις πολυπεπτιδικές αλυσίδες. Οι πρωτεΐνες ανήκουν στην κατηγορία των μακρομορίων, και αποτελούνται από μία ή περισσότερες αλυσίδες αμινοξέων.

Η δομή των πρωτεϊνών χωρίζεται σε τέσσερις κατηγορίες, την πρωτοταγή, τη δευτεροταγή, την τριτοταγή και την τεταρτοταγή δομή. Η πρωτοταγής δομή κάθε πρωτεΐνης είναι πάντα σταθερή (σε συγκεκριμένο περιβάλλον), ενώ οι άλλες δομές, προκύπτουν από τις αλληλεπιδράσεις μεταξύ των αμινοξέων της πρωτεϊνικής αλληλουχίας. Η πρωτοταγής δομή των πρωτεϊνών, είναι ουσιαστικά η αλληλουχία των αμινοξέων, και φανερώνει τη σειρά με την οποία βρίσκονται τα αμινοξέα στην πρωτεΐνη, αν την ξεδιπλώσουμε. Η δευτεροταγής δομή, είναι ο τρόπος, με τον οποίο η αλυσίδα αυτή των αμινοξέων, προσανατολίζεται ή αναδιπλώνεται στο χώρο, δημιουργώντας έτσι κάποιες τοπικές αναδιπλώσεις (δομές), τις α-έλικες και τις β-πτυχωτές επιφάνειες. Η τριτοταγής δομή μιας πρωτεΐνης, είναι και η τελική τρισδιάστατη της μορφή, η οποία τρισδιάστατη μορφή, καθορίζει και την βιολογική λειτουργία της πρωτεΐνης. Η τριτοταγής δομή μιας πρωτεΐνης, οφείλεται στην αναδίπλωση της πολυπεπτιδικής της αλυσίδας, λόγω των αλληλεπιδράσεων των αμινοξέων και των πλευρικών αλυσίδων που την αποτελούν, μεταξύ τους. Πέραν αυτού, η τρισδιάστατη μορφή μιας πρωτεΐνης, επηρεάζεται από διάφορους παράγοντες, όπως θερμοκρασία, pH κ.τ.λ. Η μεταφορά μιας πρωτεΐνης σε ένα διαφορετικό περιβάλλον ύπαρξης, προκαλεί την αναδιάταξη της τρισδιάστατης μορφής της πρωτεΐνης. Πιο απλά, χάνεται η προηγούμενη βιολογική της λειτουργία, αφού η τρισδιάστατή της μορφή αλλάζει. Είναι εξαιρετικά βοηθητικό να γνωρίζουμε την δευτεροταγή δομή μιας πρωτεΐνης, για να μπορέσουμε έτσι να μεταβούμε

από την πρωτοταγή στην τριτοταγή δομή της πρωτεΐνης, με μεγαλύτερη σχετικά ευκολία. Η τεταρτοταγής δομή μιας πρωτεΐνης, είναι η συνένωση δύο ή περισσότερων πολυπεπτιδικών αλυσίδων της τριτοταγής δομής.

Ιδιαίτερης σημασίας χρίζει ο καθορισμός της ακριβούς δομής των πρωτεϊνών στον τρισδιάστατο χώρο, όπου αλληλουχίες είκοσι διαφορετικών αμινοξέων, αναδιπλώνονται με μοναδικό τρόπο, προσδίδοντας σε αυτές μοναδικό σχήμα και κατ' επέκταση μοναδική λειτουργία. Γνωρίζοντας την ακριβή δομή των πρωτεϊνών στον τρισδιάστατο χώρο, δηλαδή γνωρίζοντας την τριτοταγή τους δομή, μπορούμε να μελετήσουμε τις πρωτεΐνες αυτές σε ιατροφαρμακευτικό επίπεδο, μπορώντας έτσι να προχωρήσουμε στην κατασκευή φαρμάκων/ενζύμων ή και άλλων προϊόντων που αφορούν την λειτουργία αυτών των μακρομορίων. Συνεπώς, η ακριβής γνώση της δομής, αλλά και της λειτουργίας μιας πρωτεΐνης, έχει τεράστια σημασία για τις επιστήμες της βιολογίας, της ιατρικής, αλλά και της φαρμακευτικής. Παρόλα αυτά, η λεπτομερής γνώση της δομής των πρωτεϊνών εμπλέκεται και με άλλους τομείς που αφορούν την γενικότερη ποιότητα ζωής, όπως είναι: η κατασκευή φυτοφαρμάκων για καταπολέμηση ασθενειών στα διάφορα φυτά, η κατασκευή ουσιών οι οποίες βοηθούν τα φυτά να μεγαλώνουν με ταχύτερο ρυθμό και να παράγουν περισσότερους καρπούς, η κατασκευή τεχνητών ή και βιολογικών πρωτεϊνών οι οποίες συντείνουν στην αύξηση της μυϊκής μάζας, και η αναστολή της λειτουργείας συγκεκριμένων πρωτεϊνών, οι οποίες μεταφέρουν ανίατες ασθένειες.

Εξαιτίας των όσων αναφέραμε πιο πάνω, ο ακριβής καθορισμός της δομής των πρωτεϊνών, αποτελεί αρκετά σημαντικό θέμα προς μελέτη, για τη σύγχρονη εποχή. Οι δυσκολίες που παρουσιάζονται στην έρευνα του συγκεκριμένου θέματος είναι πολλές. Γνωρίζοντας μόνο την πρωτοταγή δομή μιας πρωτεΐνης, δηλαδή την αλληλουχία των αμινοξέων τα οποία αποτελούν την δομική μονάδα των πρωτεϊνών, και την οποία εύκολα μπορούμε να καταγράψουμε για κάθε πρωτεΐνη, δεν μπορούμε να εξαγάγουμε τις απαραίτητες πληροφορίες έτσι ώστε να καθορίσουμε με ακρίβεια την αντίστοιχη τρισδιάστατη μορφή της πρωτεΐνης αυτής, δηλαδή την τριτοταγή της δομή. Συγκεκριμένα, στις βάσεις δεδομένων πρωτεϊνών, υπάρχουν

καταγεγραμμένες περίπου 3,500,000 πρωτεϊνικές αλληλουχίες, από τις οποίες μόνο για 140,109 γνωρίζουμε τη δευτεροταγή τους δομή.

Το πρόβλημα το οποίο προκύπτει για τον καθορισμό τις τριτοταγούς δομής των πρωτεϊνών, είναι και το ότι, οι πειραματικές μέθοδοι οι οποίες μπορούν να χρησιμοποιηθούν σήμερα για τον καθορισμό της τριτοταγούς δομής των πρωτεϊνών, είναι εξαιρετικά χρονοβόρες, πολύπλοκες και έχουν επίσης πολύ υψηλό κόστος. Ως αποτέλεσμα αυτού, η επιστημονική κοινότητα έχει ελλιπείς γνώσεις, για τις τριτοταγείς δομές συγκεκριμένων πρωτεϊνών άρα και τη λειτουργία τους, όπως επίσης και καθόλου γνώση για τις τριτοταγείς δομές αρκετών πρωτεϊνών αφού όπως αναφέραμε προηγουμένως, μόλις μερικές χιλιάδες πρωτεΐνες (από τα εκατομμύρια που υπάρχουν), έχουν μελετηθεί, για τον καθορισμό της τριτοταγούς τους δομής, και με απώτερο σκοπό τον καθορισμό της λειτουργίας τους. Η δευτεροταγής και η τριτοταγής δομή μιας πρωτεΐνης, εξαρτάται από την αλληλουχία των αμινοξέων (πρωτοταγής δομή) που απαρτίζουν την πρωτεΐνη, και είναι πάντα σταθερή σε συγκεκριμένο περιβάλλον. Με άλλα λόγια, η πρωτοταγής δομή μιας πρωτεΐνης, δηλαδή η αλληλουχία των αμινοξέων που την αποτελούν, καθορίζει τη δευτεροταγή, αλλά και την τριτοταγή δομή της πρωτεΐνης. Όπως έχουμε επισημάνει προηγουμένως, για τον σχηματισμό της δευτεροταγούς δομής μιας πρωτεΐνης, τα αμινοξέα τα οποία αποτελούν την πρωτεΐνη αλληλεπιδρούν μεταξύ τους, προκαλώντας έτσι κάποιες τοπικές αναδιπλώσεις τις οποίες ονομάζουμε α-έλικες και β-κλώνους (δευτεροταγής δομή πρωτεΐνης). Για τον καθορισμό της τριτοταγούς δομής μιας πρωτεΐνης, μπορεί να χρησιμοποιηθεί ως ενδιάμεσο βήμα, η δευτεροταγής δομή της συγκεκριμένης πρωτεΐνης, αφού με τον καθορισμό της δευτεροταγούς δομής μιας πρωτεΐνης, μπορούμε να πάρουμε αρκετές χρήσιμες πληροφορίες, για τον ακριβή καθορισμό της τρισδιάστατης της μορφής (τριτοταγής δομή). Τέλος, με βάση τα όσα λέχθηκαν, καθορίζοντας επακριβώς την τρισδιάστατη μορφή μιας πρωτεΐνης, μπορούμε να αποφανθούμε για την ακριβή λειτουργία της, καθώς επίσης και να την μελετήσουμε σε ιατροφαρμακευτικό επίπεδο.

1.2 Το Πρόβλημα

Η πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών (Protein Secondary Structure Prediction - PSSP), αποτελεί ίσως ένα από τα σημαντικότερα προβλήματα της βιοπληροφορικής, λόγω του ότι μια πετυχημένη πρόβλεψη, θα αποτελέσει κυρίαρχο βήμα και στην πρόβλεψη της τριτοταγούς δομής των πρωτεϊνών. Υπάρχουν διάφοροι αλγορίθμοι/τεχνικές που χρησιμοποιήθηκαν για την πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών (θα μελετηθούν στη συνέχεια), παίρνοντας έτσι χρήσιμες πληροφορίες για τον καθορισμό της τριτοταγούς δομής των πρωτεϊνών (άρα και της λειτουργίας τους), χρησιμοποιώντας σαν είσοδο την αλληλουχία των αμινοξέων των πρωτεϊνών, δηλαδή την πρωτοταγή τους δομή.

Το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, ξεκίνησε το 1951 όταν οι Pauling, Corey και Branson προέβλεψαν ελικοειδής διαμορφώσεις και διαμορφώσεις φύλλου, για τη πολυπεπτιδική ραχοκοκαλιά των πρωτεϊνών, προτού ακόμα να καθοριστεί η δομή οποιασδήποτε πρωτεΐνης (Pauling et al., 1951). Εξήντα πέντε χρόνια μετά, νέες, δυναμικές μέθοδοι πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, έχουν δημιουργηθεί. Τα υψηλότερα ποσοστά επιτυχίας Q3 (Εξίσωση 1.1) τα οποία έχουν επιτευχθεί, χωρίς να βασίζονται σε δομικά πρότυπα πρωτεϊνών, είναι στο 82-86% (Heffernan et al., 2017; Wang et al., 2016; Fang et al., 2018), ένα ποσοστό απίστευτο μόλις λίγα χρόνια προηγουμένως. Αυτή η βελτίωση, προήλθε κυρίως από τα μεγαλύτερα σύνολα δεδομένων (datasets) πρωτεϊνικών αλληλουχιών και δομών για εκπαίδευση, τη χρήση προτύπων πληροφοριών δευτεροταγούς δομής πρωτεϊνών και τις πιο ισχυρές τεχνικές βαθιάς μάθησης (Yang et al., 2016).

Σε αυτή την έρευνα, σε μια απόπειρα να λύσουμε το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, απευθυνθήκαμε στη Μηχανική Μάθηση (Machine Learning - ML) και συγκεκριμένα στα Νευρωνικά Δίκτυα (Neural Networks - NNS). Ο λόγος για αυτή την ενέργεια, είναι και το ότι, δεν υπάρχει μια ξεκάθαρη συσχέτιση ή κάποια μαθηματική συνάρτηση, που να συνδέει την πρωτοταγή δομή μιας πρωτεΐνης με τη δευτεροταγή της δομή, λόγω του τεράστιου αριθμού των πιθανών συνδυασμών μεταξύ των αμινοξέων που απαρτίζουν μια πρωτεΐνη. Πιο συγκεκριμένα, σε αυτή την έρευνα, καλούμαστε να

υλοποιήσουμε ένα Συνελικτικό Νευρωνικό Δίκτυο (Convolutional Neural Network - CNN), το οποίο θα εκπαιδευτεί με τη μέθοδο της ανάστροφης μετάδοσης σφάλματος, και θα δέχεται στην είσοδό του, ένα συγκεκριμένο αμινοξύ, μαζί με κάποια άλλα αμινοξέα τα οποία προηγούνται και έπονται αυτού του αμινοξέος (δηλαδή μια σειρά από αμινοξέα), και τέλος θα προβλέπει στην έξοδο του, σε ποια από τρεις κατηγορίες δευτεροταγούς δομής ανήκει το συγκεκριμένο αμινοξύ. Συγκεκριμένα, μέσα από την διαδικασία αυτή, θα χρησιμοποιήσουμε ένα συνελικτικό νευρωνικό δίκτυο, το οποίο θα προβλέπει την δευτεροταγή δομή μιας πρωτεΐνης για κάποιο αμινοξύ, συσχετίζοντάς το με τα γειτονικά αμινοξέα, τα οποία βρίσκονται «κοντά» στο αμινοξύ που μελετούμε, και τα οποία γειτονικά αμινοξέα είναι πολύ πιθανόν να αλληλεπιδρούν με το αμινοξύ που μελετούμε, άρα και να καθορίζουν εν μέρει, την δευτεροταγή δομή της πρωτεΐνης, στην συγκεκριμένη θέση, για το συγκεκριμένο αμινοξύ.

Στην συνέχεια, θα προσπαθήσουμε να εισάγουμε τα φίλτρα Gabor στην έρευνα μας, όπως και κάποιες άλλες τεχνικές οι οποίες θα αναφερθούν στη συνέχεια, με στόχο να εξάγουμε χαρακτηριστικά από τα ήδη έτοιμα δεδομένα εισόδου, έτσι ώστε να λάβουμε κάποια άλλα, νέα/εμπλουτισμένα δεδομένα εισόδου, τα οποία θα περιέχουν περισσότερες πληροφορίες για την πρωτεΐνη της οποίας προσπαθούμε να προβλέψουμε την δευτεροταγή δομή. Στη συνέχεια, θα δημιουργήσουμε και θα εκπαιδεύσουμε κάποια συνελικτικά νευρωνικά δίκτυα, χρησιμοποιώντας τα νέα/εμπλουτισμένα δεδομένα εισόδου, έτσι ώστε να πετύχουμε υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών.

Στο τελευταίο στάδιο της έρευνας αυτής, θα χρησιμοποιήσουμε τα Support Vector Machines (SVMs) για να φιλτράρουμε/διορθώσουμε τα αποτελέσματα πρόβλεψης τα οποία λάβαμε από τα συνελικτικά νευρωνικά δίκτυα, για περαιτέρω βελτίωση της ποιότητας των αποτελεσμάτων.

1.3 Μηχανική Μάθηση, Νευρωνικά Δίκτυα και Φίλτρα Gabor

Η Μηχανική μάθηση (Machine Learning) είναι ένα υποπεδίο της επιστήμης των υπολογιστών που αναπτύχθηκε από τη μελέτη της αναγνώρισης προτύπων και της υπολογιστικής θεωρίας μάθησης στην τεχνητή νοημοσύνη. Το 1959, ο Arthur Samuel ορίζει τη μηχανική μάθηση ως «Πεδίο μελέτης που δίνει στους υπολογιστές την ικανότητα να μαθαίνουν, χωρίς να έχουν

προγραμματιστεί ρητά» (Samuel, 1959). Η μηχανική μάθηση ασχολείται με τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα που τους παρέχονται, αλλά και να κάνουν προβλέψεις σχετικά με αυτά. Τέτοιοι αλγόριθμοι λειτουργούν κατασκευάζοντας μοντέλα από πειραματικά δεδομένα, προκειμένου να κάνουν προβλέψεις βασιζόμενες στα δεδομένα εισόδου ή να εξάγουν αποφάσεις/συναρτήσεις που εκφράζονται ως το αποτέλεσμα των δεδομένων εισόδου. Η μηχανική μάθηση είναι στενά συνδεδεμένη και συχνά συγχέεται με την υπολογιστική στατιστική, ένας κλάδος που επίσης επικεντρώνεται στην πρόβλεψη μέσω της χρήσης των υπολογιστών. Έχει ισχυρούς δεσμούς με την μαθηματική βελτιστοποίηση, η οποία παρέχει τις μεθόδους, τη θεωρία και τους τομείς εφαρμογής. Η Μηχανική μάθηση εφαρμόζεται σε μια σειρά από υπολογιστικές εργασίες, όπου τόσο ο σχεδιασμός όσο και ο ρητός προγραμματισμός των αλγορίθμων είναι ανέφικτος.

Το Νευρωνικό Δίκτυο (Neural Network - NN) είναι ένα δίκτυο από απλούς υπολογιστικούς κόμβους (νευρώνες, νευρώνια), διασυνδεδεμένους μεταξύ τους. Είναι εμπνευσμένο από το Κεντρικό Νευρικό Σύστημα (ΚΝΣ), το οποίο και προσπαθεί να προσομοιώσει.

Οι νευρώνες είναι τα δομικά στοιχεία του δικτύου. Κάθε τέτοιος κόμβος δέχεται ένα σύνολο αριθμητικών εισόδων από διαφορετικές πηγές (είτε από άλλους νευρώνες, είτε από το περιβάλλον), επιτελεί έναν υπολογισμό με βάση αυτές τις εισόδους και παράγει μία έξοδο. Η εν λόγω έξοδος είτε κατευθύνεται στο περιβάλλον, είτε τροφοδοτείται ως είσοδος σε άλλους νευρώνες του δικτύου. Υπάρχουν τρεις τύποι νευρώνων: οι νευρώνες εισόδου, οι νευρώνες εξόδου και οι υπολογιστικοί νευρώνες ή κρυμμένοι νευρώνες. Οι νευρώνες εισόδου δεν επιτελούν κανέναν υπολογισμό, μεσολαβούν απλώς ανάμεσα στις περιβαλλοντικές εισόδους του δικτύου και στους υπολογιστικούς νευρώνες. Οι νευρώνες εξόδου διοχετεύουν στο περιβάλλον τις τελικές αριθμητικές εξόδους του δικτύου. Οι υπολογιστικοί νευρώνες πολλαπλασιάζουν κάθε είσοδό τους με το αντίστοιχο συναπτικό βάρος και υπολογίζουν το ολικό άθροισμα των γινομένων. Το άθροισμα αυτό τροφοδοτείται ως όρισμα στη συνάρτηση ενεργοποίησης, την οποία υλοποιεί εσωτερικά κάθε κόμβος. Η τιμή που παράγει η συνάρτηση για το εν λόγω όρισμα είναι και η έξοδος του νευρώνα για τις τρέχουσες εισόδους και βάρη.

Τα (Convolutional Neural Networks – CNNs), ανήκουν στην οικογένεια των Βαθιών Νευρωνικών Δικτύων (Deep Neural Networks), και σύμφωνα με μελέτες που έχουν γίνει, φαίνεται να είναι κατάλληλα σε περιοχές όπως ανάλυση και αναγνώριση αντικειμένων σε εικόνες, ανάλυση και εντοπισμό χαρακτηριστικών σε ηχητικά σήματα (π.χ. φωνή) και γενικά σε προβλήματα που σχετίζονται με την αναγνώριση χαρακτηριστικών, και κατηγοριοποίηση δεδομένων (Krizhevsky et al., 2012; Karpathy et al., 2014; Kim, 2014; Karpathy, 2016;). Για να μπορέσουμε επιτυχώς να εκπαιδεύσουμε ένα CNN πρέπει να επιχειρήσουμε να μετασχηματίσουμε/οπτικοποιήσουμε τα δεδομένα εισόδου έτσι ώστε το δίκτυο να μπορέσει να εντοπίσει χαρακτηριστικά και να προβλέψει την δευτεροταγή δομή μιας πρωτεΐνης.

Στην επεξεργασία εικόνας, ένα φίλτρο Gabor, το οποίο ονομάστηκε από τον Dennis Gabor, είναι ένα γραμμικό φίλτρο που χρησιμοποιείται για ανάλυση υφής, πράγμα που σημαίνει ότι βασικά αναλύει αν υπάρχει συγκεκριμένο περιεχόμενο συχνότητας στην εικόνα σε συγκεκριμένες κατευθύνσεις σε μια τοπική περιοχή γύρω από το σημείο ή την περιοχή της ανάλυσης. Πολλοί σύγχρονοι επιστήμονες που ασχολούνται με την όραση ισχυρίζονται ότι, οι παραστάσεις συχνότητας και προσανατολισμού των φίλτρων, είναι παρόμοιες με εκείνες του ανθρώπινου οπτικού συστήματος, αν και δεν υπάρχουν εμπειρικά στοιχεία και λειτουργικό σκεπτικό για την υποστήριξη αυτής της ιδέας. Στο χωρικό πεδίο, ένα δισδιάστατο (2D) φίλτρο Gabor, είναι μια Γκαουσιανή (Gaussian) συνάρτηση πυρήνα, διαμορφωμένη από ένα ημιτονοειδές επίπεδο κύμα (Jones et al., 1987).

Επιπρόσθετα, τα Support Vector Machines (SVMs) είναι κάποιοι αλγόριθμοι μηχανικής μάθησης οι οποίοι φαίνεται να διαχωρίζουν διάφορα προβλήματα με την αναγωγή των σημείων του προβλήματος σε υψηλότερες διαστάσεις έτσι ώστε το πρόβλημα να γίνει πλέον γραμμικά διαχωρίσιμο. Ακολούθως βρίσκει την ιδανική επιφάνεια διαχωρισμού με το να βρίσκει τα οριακά πιο κοντινά, διαφορετικών κλάσεων σημεία (support vectors), και παίρνει τέλος την μεσαία γραμμή διαχωρισμού ως την ιδανική επιλογή για διαχωρισμό των κλάσεων (Cortes and Vapnik, 1995).

Συνδέοντας όλα τα πιο πάνω, σε αυτή την έρευνα, για να καταφέρουμε να αυξήσουμε τα ποσοστά επιτυχίας στην πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών, θα

επιχειρήσουμε αρχικά να χρησιμοποιήσουμε διαφορετικά φίλτρα Gabor, έτσι ώστε να εξάγουμε χαρακτηριστικά σε συγκεκριμένες κατευθύνσεις και συχνότητες, και να πάρουμε ως αποτέλεσμα κάποια νέα/εμπλουτισμένα δεδομένα εισόδου που θα χρησιμοποιήσουμε έτσι ώστε να εκπαιδεύσουμε τα CNNs μας, χρησιμοποιώντας τον αλγόριθμο μάθησης ανάστροφης μετάδοσης σφάλματος.

Στην συνέχεια, θα προχωρήσουμε με το να εκπαιδεύσουμε τα CNNs μας με τα αρχικά δεδομένα εισόδου και να φιλτράρουμε/διορθώσουμε τα αποτελέσματα την πρόβλεψης των CNNs με χρήση των SVMs. Τα CNNs, μας δίνουν την δυνατότητα να εντοπίσουμε συγκεκριμένα χαρακτηριστικά μέσα από πολύπλοκα δεδομένα εισόδου, αφού παρέχουν βελτιστοποίηση πολλαπλών επιπέδων.

1.4 Προηγούμενη Σχετική Έρευνα

Υπάρχουν αρκετές προηγούμενες σχετικές έρευνες που έγιναν στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, λόγω της προφανής σοβαρότητας αλλά και της ανάγκης για επίλυση του συγκεκριμένου προβλήματος για τους λόγους που αναφέραμε προηγουμένως (Ενότητα 1.2). Το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών κατατάχθηκε ως ένα από τα 125 μεγαλύτερα άλυτα προβλήματα από το περιοδικό Science (Dill et al., 2008). Τα ποσοστά επιτυχίας Q3 (Εξίσωση 1.1) στην πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών, είναι σε αρκετά ψηλά επίπεδα, χωρίς ωστόσο να θεωρούνται ικανοποιητικά στην ολική εύρεση λύσης για το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Κάποιες από τις έρευνες που έγιναν στο συγκεκριμένο θέμα αναφέρονται και επεξηγούνται πιο κάτω, με την έρευνα των Heffernan et al. (2017) να πετυχαίνει τα υψηλότερα ποσοστά επιτυχίας.

$$Q3 = \frac{1}{TotalNumberOfAminos} * 100 * \sum_{n=1}^{TotalNumberOfAminos} \text{Aij}$$

Εξίσωση 1.1: Εξίσωση εύρεσης ποσοστού επιτυχίας Q3, όπου TotalNumberOfAminos ο συνολικός αριθμός των αμινοξικών καταλοίπων που αποτελούν την πρωτεΐνη και το A_{ij} να

παίρνει την τιμή 1 αν η πραγματική έξοδος i είναι η ίδια με την επιθυμητή έξοδο j, και την τιμή 0 διαφορετικά.

Πιο κάτω παραθέτουμε μια ιστορική αναδρομή των πιο σημαντικών ερευνητών που ξεχώρισαν με τις μεθόδους που χρησιμοποίησαν για την επίλυση του προβλήματος αυτού, και οι οποίες αφορούν κυρίως υλοποιήσεις νευρωνικών δικτύων αλλά και στατιστικών μεθόδων.

1. Feedforward Fully Connected NN (Qian and Sejnowski, 1988): Στην συγκεκριμένη έρευνα χρησιμοποιήθηκε ένα απλό πλήρως συνδεδεμένο νευρωνικό δίκτυο με παράθυρο εισόδου (local input window) 13ων αμινοξέων ορθογώνιας κωδικοποίησης (orthogonal encoding) και με μόνο ένα κρυφό επίπεδο. Η έξοδος του δικτύου, είναι μία από τις τρείς κατηγορίες της δευτεροταγούς δομής (H: α - έλικας, E: β - πτυχωτή επιφάνεια, C: αμινοξέα που δεν ανήκουν σε μια από τις άλλες κατηγορίες των πρωτεϊνών) και αναφερόταν στο αμινοξύ που βρισκόταν στο κέντρο του παραθύρου εισόδου. Στην συγκεκριμένη υλοποίηση χρησιμοποιήθηκε επιπρόσθετα, ένα δεύτερο διαδοχικό νευρωνικό δίκτυο, το οποίο ουσιαστικά βελτίωνε τα δεδομένα εξόδου του προηγούμενου δικτύου. Η τεχνική που χρησιμοποιήθηκε στη συγκεκριμένη έρευνα αντιμετώπιζε προβλήματα υπερεκπαίδευσης (overfitting).

2. PHD (Rost and Sander, 1993): Η αρχιτεκτονική του συγκεκριμένου δικτύου, είναι η ίδια με την αρχιτεκτονική του «Feedforward Fully Connected NN» («Εμπρόσθιο πλήρως συνδεδεμένο νευρωνικό δίκτυο»), με την διαφορά ότι χρησιμοποιήθηκαν κάποιες επιπλέον τεχνικές για την επιτυχή αντιμετώπιση του προβλήματος της υπερεκπαίδευσης. Η πρώτη τεχνική που χρησιμοποιήθηκε, είναι η τεχνική του γρήγορου σταματήματος (early stopping), κατά την οποία η εκπαίδευση του δικτύου σταματά αυτόματα, αν το σφάλμα εκπαίδευσης συνεχίζει να μειώνεται ενώ στο σφάλμα επαλήθευσης συνεχίζει να αυξάνεται και η δεύτερη τεχνική που χρησιμοποιήθηκε είναι η τεχνική του συνολικού μέσου όρου (ensemble average), κατά την οποία παίρνουμε ως τελικό αποτέλεσμα εξόδου πρόβλεψης τον μέσο όρο διαφορετικών νευρωνικών δικτύων, τα οποία εκπαιδεύουμε ταυτόχρονα με διαφορετικά δεδομένα εισόδου και μεθόδους μάθησης. Στην συγκεκριμένη έρευνα, χρησιμοποίησαν τεχνικές για επεξεργασία των δεδομένων εισόδου για εκπαίδευση και επαλήθευση, όπως η τεχνική της πολλαπλής

στοίχισης αλληλουχιών (Multiple Sequence Alignment - MSA), για εκμετάλλευση της εξελικτικής πληροφορίας διαφορετικών πρωτεϊνών.

3. NNSSP (Salamov and Soloveyev, 2000): Στην συγκεκριμένη έρευνα, το τεχνητό νευρωνικό δίκτυο που έχουν δημιουργήσει οι Salamov και Soloveyev, χρησιμοποιεί την μέθοδο του κοντινότερου-γείτονα (nearest-neighbor method) ούτως ώστε αρχικά, να ομαδοποιήσει τις πρωτεϊνικές ακολουθίες ανάλογα με τις ομοιότητές τους, και στη συνέχεια να τις συγκρίνει με άλλες πρωτεϊνικές ακολουθίες των οποίων η δευτεροταγής δομή τους είναι ήδη γνωστή. Τέλος, έχοντας υπόψιν τα πιο πάνω, το νευρωνικό δίκτυο προσπαθεί να προβλέψει την δευτεροταγή δομή αγνώστων πρωτεϊνών.

4. DSC (King and Sternberg, 1996): Στην έρευνα των King και Sternberg (1996), φαίνεται ότι αρχικά, ο αλγόριθμος που ανάπτυξαν, ομαδοποιεί τα δεδομένα εξόδου του νευρωνικού δικτύου σε διάφορες κατηγορίες και στη συνέχεια, χρησιμοποιώντας απλές και γραμμικές στατιστικές μεθόδους προσπαθεί να προσεγγίσει την ακριβή δευτεροταγή δομή των πρωτεϊνών που μελετά.

5. PREDATOR (Frishman and Argos, 1995): Σε αυτήν την έρευνα, οι Fisherman και Argos (1995), έχουν αναπτύξει έναν αυτόματο αλγόριθμο, ο οποίος προσπαθεί να προβλέψει τη δευτεροταγή των πρωτεϊνών που μελετά, χρησιμοποιώντας τις ατομικές συντεταγμένες του κάθε αμινοξέος και λαμβάνοντας υπόψιν την ενέργεια των υρδογονικών δεσμών και διάφορες στατιστικές πληροφορίες που προέρχονται από τη στρεπτική γωνιακή σπονδυλική στήλη.

6. Consensus (Cuff and Barton, 1999): Η συγκεκριμένη μέθοδος αφορά τεχνητό νευρωνικό δίκτυο το οποίο παίρνει σαν είσοδο, δεδομένα που έχουν υποστεί πολλαπλή στοίχιση ακολουθιών (MSA), και στην οποία φαίνονται επιπλέον πληροφορίες και στοιχεία για την πρωτεΐνη που μελετούμε αντί για μόνο μια ακολουθία αμινοξέων. Το νευρωνικό δίκτυο στη συγκεκριμένη έρευνα, προσπαθούσε να εντοπίσει διάφορες ομοιότητες των δεδομένων εισόδου με άλλες πρωτεϊνικές ακολουθίες, να μεταφράσει αυτές τις ομοιότητές σε γενετικό κώδικα, εξελικτική ιστορία και κοινές βιολογικές λειτουργίες και ανάλογα να πραγματοποιήσει πρόβλεψη της δευτεροταγούς τους δομής, λαμβάνοντας υπόψιν όλες αυτές τις πληροφορίες.

7. Bidirectional Recurrent Neural Network (BRNN) – Backpropagation (Baldi et al., 1999; Baldi et al., 2000): Ο αλγόριθμος ο οποίος κατασκευάστηκε στη συγκεκριμένη έρευνα, είχε σχετικά την καλύτερη απόδοση στην πρόβλεψη δευτεροταγούς δομής πρωτεϊνών (76% επιτυχές αποτέλεσμα), το συγκεκριμένο χρονικό διάστημα. Ο αλγόριθμος αυτός, χρησιμοποιεί τεχνητό νευρωνικό δίκτυο, το οποίο δέχεται στην είσοδό του ένα παράθυρο με μια ακολουθία από αμινοξέα. Έπειτα, το νευρωνικό δίκτυο προσπαθεί να προβλέψει τη δευτεροταγή δομή του αμινοξέος που βρίσκεται στο κέντρο του παραθύρου εισόδου, με βάση τα αμινοξέα που προηγούνται και έπονται αυτού στην αλυσίδα εισόδου, χρησιμοποιώντας αμφίδρομη αναδρομή.

8. LAD (Blazewicz et al., 2005): Η συγκεκριμένη έρευνα, μελετά εκτός των άλλων, την δομή των πρωτεϊνών και τις ιδιότητες των αμινοξέων για προσπάθεια εξεύρεσης λύσης στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής. Στην έρευνα αυτή, υλοποιήθηκε ο αλγόριθμος LAD, ο οποίος αποτελεί έναν αλγόριθμο μηχανικής μάθησης, ο οποίος έχει ως σκοπό να αναγνωρίσει ιδιότητες των αμινοξέων, που μπορεί να παρέχουν διάφορες σχετικές πληροφορίες για την ομοιογένεια των πρωτεϊνών, κάτι το οποίο μπορεί να βοηθήσει στην πρόβλεψη της δευτεροταγούς τους δομής. Η συγκεκριμένη μέθοδος εκτός του ότι είχε σχετικά καλά αποτελέσματα, οδήγησε και στην εξαγωγή συμπερασμάτων που αφορούν τις ιδιότητες των αμινοξέων και τη σχέση που έχουν αυτά στην πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών. Για παράδειγμα, ένα συμπέρασμα που λήφθηκε από την πραγματοποίηση της συγκεκριμένης έρευνας, είναι και το ότι, μία από τις πιο σημαντικές ιδιότητες που επηρεάζουν την κλάση των α ελίκων, είναι τα μοριακά βάρη, ενώ για την κλάση των β πτυχωτών επιφανειών, είναι η μέση περιβαλλόμενη υδροφοβία. Τέλος, μια από τις πιο σημαντικές ιδιότητες που επηρεάζουν οποιαδήποτε άλλη μορφή/κλάση είναι η πολικότητα.

9. MASSP3 (Armano et al., 2006): Ένα σύστημα που καταφεύγει σε πολλούς εμπειρογνώμονες για την αντιμετώπιση του προβλήματος της πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, των οποίων οι επιδόσεις είναι συγκρίσιμες με εκείνες που λαμβάνονται από άλλους state-of-the-art predictors. Το σύστημα εκτελεί μια συνολική επεξεργασία που βασίζεται σε δύο βασικά βήματα: πρώτον, πραγματοποιείται μια πρόβλεψη «ακολουθίας προς δομή», προσφεύγοντας

σε έναν πληθυσμό υβριδικών γενετικά-νευρικών εμπειρογνωμόνων, και στη συνέχεια πραγματοποιείται μια πρόβλεψη «δομής προς δομή», με την προσφυγή σε τεχνητά νευρωνικά δίκτυα εμπρόσθιου περάσματος. Για να διερευνηθεί η απόδοση της προτεινόμενης προσέγγισης, το σύστημα δοκιμάστηκε με το σύνολο των πρωτεϊνών RS126. Τα πειραματικά αποτελέσματα (περίπου 76% ποσοστό επιτυχίας Q3), αναδεικνύουν την εγκυρότητα της προσέγγισης.

10. Two-Stage method (Yuksektepe et al., 2008): Στόχος της συγκεκριμένης μεθοδολογίας ήταν να προβεί σε πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών σε δυο στάδια. Το πρώτο στάδιο εντοπίζει αστάθειες σχετικά με τον τρόπο αναδίπλωσης μιας πρωτεΐνης στο χώρο και προσπαθεί να κατατάξει τα διάφορα σημεία της πρωτεΐνης σε κατηγορίες, ενώ στο δεύτερο στάδιο οι πρωτεΐνες διασπώνται σε ακολουθίες τριών μέχρι επτά κατάλοιπων και ο αλγόριθμος προσπαθεί να εντοπίσει την δευτεροταγή δομή των συγκεκριμένων ακολουθιών που λαμβάνονται ως αποτέλεσμα αυτών των διασπάσεων που γίνονται.

11. Evolutionary method for learning HMM structure (Won et al., 2007): Λόγω της δυσκολίας και πολυπλοκότητας δημιουργίας ενός Hidden Markov Model (HMM), στη συγκεκριμένη έρευνα, χρησιμοποιήθηκαν γενετικοί αλγόριθμοι που έχουν την δυνατότητα να αλλάζουν δυναμικά τις παραμέτρους ενός HMM και ως αποτέλεσμα να μπορούν να το κτίζουν δυναμικά, με απώτερο σκοπό να μπορεί να προβλέπει την δευτεροταγή δομή των πρωτεϊνικών αλληλουχιών που δέχεται στην είσοδο του.

12. Cascade Bidirectional Recurrent Neural Network (BRNN) (Chen and Chaudhari, 2007): Η συγκεκριμένη έρευνα, είναι μια από τις πιο πρόσφατες υλοποιήσεις τεχνητών νευρωνικών δικτύων, η οποία χρησιμοποιήθηκε για την επίλυση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών και αφορά τη δημιουργία ενός ή δύο επιπέδων νευρωνικού δικτύου. Η υλοποίηση αυτή, συμπεριέλαβε στα δεδομένα εισόδου και την έννοια των μακρινών αλληλεξαρτήσεων (long range dependencies) μεταξύ των αμινοξικών καταλοίπων, κάτι το οποίο είναι απαραίτητο να ληφθεί υπόψιν, αφού παίζει σημαντικό ρόλο στην αναδίπλωση μιας οποιασδήποτε πρωτεΐνης, και της συσχέτισης που υπάρχει μεταξύ γειτονικών δομών δευτεροταγούς δομής. Συγκεκριμένα, για το τελευταίο, οι Chen και

Chaudhari (2007), αναφέρονται στο άρθρο τους, στη σχέση υπάρχει μεταξύ της δευτεροταγούς δομής ενός αμινοξέος και της δευτεροταγούς δομής των γειτονικών του αμινοξέων. Η μέθοδος που προτείνεται στην συγκεκριμένη έρευνα, είναι η δημιουργία δυο BRNN με την έξοδο του πρώτου να αποτελεί την είσοδο του δεύτερου. Η μέθοδος αυτή είχε αρκετά καλά αποτελέσματα, αλλά όχι τα καλύτερα μέχρι την συγκεκριμένη στιγμή και σε σχέση πάντα, με άλλες προσεγγίσεις της εποχής.

13. Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields (Wang et al., 2016): Σε αυτή την έρευνα εξετάζεται το δίκτυο DeepCNF (Deep Convolutional Neural Fields) για πρόβλεψη δευτεροταγούς δομής πρωτεϊνών. Το DeepCNF είναι μια επέκταση του Deep Learning από Conditional Neural Fields (CNF), η οποία είναι μια συνένωση των Conditional Random Fields (CRF) και των shallow neural networks. Το DeepCNF μπορεί να μοντελοποιήσει όχι μόνο τη σύνθετη σχέση αλληλουχίας-δομής από μια βαθιά ιεραρχική αρχιτεκτονική, αλλά και αλληλεξάρτηση μεταξύ γειτονικών δευτεροταγούς δομής ετικετών, έτσι είναι πολύ πιο ισχυρό από το CNF. Πειραματικά αποτελέσματα έδειξαν ότι το DeepCNF μπορεί να αποκτήσει ακρίβεια ~84% Q3[1], ~85 βαθμολογία SOV[2] και ~72% ακρίβεια Q8[1], αντίστοιχα, στις πρωτεΐνες δοκιμής CASP και CAMEO, υπερβαίνοντας έτσι τις υφιστάμενες μεθόδους πρόβλεψης δευτεροταγούς δομής πρωτεϊνών. Ως ένα γενικό πλαίσιο, τα DeepCNF δίκτυα μπορούν να χρησιμοποιηθούν για την πρόβλεψη άλλων ιδιοτήτων της δομής μιας πρωτεΐνης, όπως τα contact number, disorder regions, and solvent accessibility.

14. Πρόβλεψη Δευτεροταγούς Δομής Των Πρωτεϊνών Με Τη Χρήση Των Convolutional Neural Networks Για Οπτική Αναγνώριση Αντικειμένων (Παυλίδης, 2016): Η εκπόνηση αυτής της έρευνας είχε σκοπό να εξετάσει το πώς οι Convolutional Neural Networks (CNN) αλγόριθμοι μπορούν να βοηθήσουν το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Πιο συγκεκριμένα, τέτοιου είδους νευρωνικά δίκτυα εκμεταλλεύονται τη χωρική δομή των δεδομένων εισόδου, γεγονός που μας γεμίζει με ελπίδα για καλύτερα αποτελέσματα από ότι τα BRNN δίκτυα. Επιπλέον, θεωρητικά, τα συνελικτικά νευρωνικά δίκτυα κάνουν

[1] Οι Q3 και Q8 μετρικές συγκρίνουν ένα προς ένα τα αμινοξέα για να εντοπίσουμε το ποσοστό επιτυχίας.
[2] Η βαθμολογία SOV βασίζεται στη μέση αλληλεπικάλυψη μεταξύ των παρατηρούμενων και των προβλεπόμενων τμημάτων.

καλύτερη διαχείριση των δεδομένων εισόδου σε προβλήματα που έχουν να κάνουν με αλληλουχίες δεδομένων ή γενικότερα προβλήματα, τα οποία λαμβάνουν ως παράμετρο το χώρο (όπως επεξεργασία εικόνας). Η μέθοδος αυτή, κατάφερε να φτάσει ακρίβεια μόνο ~40% Q3 επιτυχία στην πρόβλεψη.

15. LSTM-BRNN (Heffernan et al., 2017): Η συγκεκριμένη έρευνα χρησιμοποιεί τα Long Short-Term Memory (LSTM) Bidirectional Recurrent Neural Networks (BRNNs), τα οποία είναι ικανά να εντοπίσουν long range αλληλεπιδράσεις μεταξύ των αμινοξέων, χωρίς να χρησιμοποιούν κάποιο παράθυρο (window). Αυτή η μέθοδος κατάφερε να πετύχει ποσοστά επιτυχίας 84% Q3.

16. MUFold-SS (Fang et al., 2018): Στην συγκεκριμένη έρευνα, προτείνεται μια αρχιτεκτονική βαθιών νευρωνικών δικτύων, με όνομα Deep inception-inside-inception (Deep3I) δίκτυο, για το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Η τεχνική αυτή υλοποιήθηκε σαν εργαλείο λογισμικού με όνομα MUFOLD-SS. Η είσοδος στο MUFOLD-SS είναι ένας προσεκτικά σχεδιασμένος πίνακας από χαρακτηριστικά που απευθύνεται στην πρωτοταγή δομή μιας πρωτεΐνης, και αποτελείται από ένα πλούσιο σετ πληροφοριών αντλούμενες από κάθε αμινοξύ της πρωτεΐνης καθώς και από το γενικό περιεχόμενο της πρωτεΐνης. Η αρχιτεκτονική του MUFOLD-SS επιτρέπει την αποδοτική επεξεργασία τοπικών και γενικών αλληλεπιδράσεων μεταξύ των αμινοξέων, για πιο ακριβής προβλέψεις. Το MUFOLD-SS φαίνεται να ξεπερνά κατά πολύ τις καλύτερες μεθόδους που ξέρουε μέχρι σήμερα, καθώς και άλλες μεθόδους βαθιών νευρωνικών δικτύων.

Όπως είδαμε, υπήρξαν αρκετές προσεγγίσεις στην προσπάθεια επίλυσης του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, χωρίς ωστόσο να υπάρχει μια μελέτη/έρευνα, η οποία αντιμετώπισε το συγκεκριμένο πρόβλημα με μια συγκεκριμένη μεθοδολογία και η οποία να δίνει αποτελέσματα επιτυχίας στην πρόβλεψη, έτσι ώστε το πρόβλημα αυτό, να θεωρηθεί πλέον λυμένο για την επιστημονική κοινότητα.

ΑΡ.	ΜΕΘΟΔΟΣ	Q3 ΕΠΙΤΥΧΙΑ (%)
1	Feedforward Fully Connected NN (Qian και Sejnowski, 1988)	63.30
2	PHD (Rost, 2001; Rost και Sander, 1993)	71.40
3	NNSSP (Salamov και Soloveyev, 1997)	68.41
4	DSC (King και Sternberg, 1996)	71.95
5	PREDATOR (Frishman και Argos, 1997)	68.60
6	Consensus (Cuff και Barton, 1999)	72.70
7	BRNN – Backpropagation (Baldi et al., 1999)	76.00
8	LAD (Jacek et al., 2005)	70.60
9	MASSP3 (Giuliano et al., 2005)	76.10
10	Evolutionary method for learning HMM structure (Won et al., 2007)	65.00
11	Two-Stage method (Fadime et al., 2007)	74.10
12	Cascade BRNN (Jinmiao και Narendra, 2007)	74.38
13	Deep Convolutional Neural Fields (Wang et al., 2016)	83.00
14	Convolutional Neural Networks (Παυλίδης, 2016)	40.00
15	LSTM-BRNN (Heffernan et al., 2017)	84.00
16	MUFold-SS (Fang et al., 2018)	**86.49**

Πίνακας 1.1: Οι βασικές μέθοδοι που χρησιμοποιήθηκαν για πρόβλεψη δευτεροταγούς δομής πρωτεϊνών και τα ποσοστά επιτυχίας τους.

Στον πιο πάνω πίνακα βλέπουμε κάποιες από τις καλύτερες θεωρητικά τεχνικές για επίλυση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Κάποιες έρευνες έχουν πετύχει αρκετά υψηλά αποτελέσματα επιτυχίας, χωρίς

ωστόσο να έχουμε κάποια έρευνα η οποία να επιλύει το πρόβλημα ολοκληρωτικά πετυχαίνοντας ένα ποσοστό επιτυχίας το οποίο δεν μπορεί να τύχει βελτίωσης. Τα καλύτερα αποτελέσματα μέχρι στιγμής, πήρε η έρευνα των Fang et al. (2018), με ποσοστά επιτυχίας Q3 86.49%.

Για να μπορέσει να επιτευχθεί ένα υψηλό ποσοστό επιτυχίας για το συγκεκριμένο πρόβλημα θα πρέπει η τεχνική που θα χρησιμοποιηθεί, να ανταποκρίνεται στη φύση του προβλήματος. Η τεχνική που χρησιμοποιήθηκε στην έρευνα των Qian και Sejnowski (1988), δεν θα μπορούσε να πετύχει μεγαλύτερα ποσοστά επιτυχίας αφού η τεχνική αυτή δεν έχει εφαρμογή στη φύση του προβλήματος. Ο λόγος είναι ότι ένα από νευρωνικό δίκτυο εμπρόσθιου περάσματος δεν παρέχει μηχανισμούς για την συσχέτιση της σειράς της ακολουθίας των αμινοξέων. Τέλος η χρήση παραθύρου αλλά η έλλειψη ανάδρασης στο συγκεκριμένο δίκτυο δεν επιτρέπει την

συσχέτιση των αμινοξέων για ακριβέστερο καθορισμό της δευτεροταγούς δομής των πρωτεϊνών (Elman, 1990).

Την ίδια τύχη έχει και η έρευνα των Rost και Sander (1993), αφού ήταν απλά μια βελτιστοποίηση της προηγούμενης. Στην έρευνα αυτή, σημαντική είναι η ιδέα χρησιμοποίησης αλγορίθμων πολλαπλής στοίχισης (multiple alignment), για ομαδοποίηση των πρωτεϊνικών ακολουθιών αφού έτσι μπορεί η ιδιότητα αυτή να χρησιμοποιηθεί στην δημιουργία των συνόλων δεδομένων, για κάλυψη όλου του εύρους πρωτεϊνικών ακολουθιών.

Οι μετέπειτα έρευνες που έγιναν από τους Salamov και Soloveyev (1997) και τους King και Sternberg (1996), ήταν κυρίως στατιστικές μέθοδοι και στηρίζονταν στις ομοιότητες με πρωτεΐνες που γνώριζαν ήδη την δευτεροταγή δομή τους. Συνεπώς ο κυρίως λόγος που απέτυχαν αυτές οι μέθοδοι μπορεί να ήταν και ο ελάχιστος αριθμός των πρωτεϊνών για τις οποίες γνωρίζαμε ήδη την δευτεροταγή τους δομή, εκείνη την εποχή (Μερικές χιλιάδες από τις περίπου 5300 πρωτεϊνικές ακολουθίες για τις οποίες γνωρίζαμε την δευτεροταγή τους δομή).

Ως συνέχεια στις έρευνες του PSSP προβλήματος έχουμε και την έρευνα των Frishman και Argos (1995), η οποία βασιζόταν στην δομή του πρωτεϊνικού σκελετού και στους δεσμούς υδρογόνου, οι οποίοι παίζουν σημαντικό ρόλο μόνο στην κατηγορία δευτεροταγούς δομής των α-ελίκων. Συνεπώς αυτή η μέθοδος, δεν είχε αποτελέσματα πρόβλεψης μεγαλύτερα από 70%, λόγω του ότι ολόκληρη η έρευνα και η λογική των Frishman και Argos στηριζόταν σε ελλιπή στοιχεία, αν και θεωρείται από τις θεμελιώδεις μεθόδους πρόβλεψης δευτεροταγούς δομής πρωτεϊνών.

Το δίκτυο των Cuff και Barton (Cuff and Barton, 1999), ήταν από τις πρώτες μεθόδους (αν όχι η πρώτη) που πέτυχε αποτελέσματα πρόβλεψης μεγαλύτερα από 72%. Το κύριο σημείο αυτής της έρευνας ήταν ότι χρησιμοποιούσε τον αλγόριθμο πολλαπλής στοίχισης (multiple alignment), για προεπεξεργασία των δεδομένων, κάτι που όντως αποδείχθηκε εξαιρετικά χρήσιμο αφού σχηματίζονταν έτσι, νέα δεδομένα εισόδου εμπλουτισμένα σε πληροφορία. Η μέθοδος αυτή χρησιμοποιούσε εξωγενή στοιχεία των πρωτεϊνών που ίσως ξέφευγαν από την ιδέα της φύσης του προβλήματος, δηλαδή από το γεγονός ότι η δευτεροταγής δομή οφείλεται καθαρά στην ακολουθία των αμινοξέων (πρωτοταγής δομή). Αντίθετα η έρευνα του Baldi et al.

(1999, 2000) εφαρμοζόταν πλήρως στην φύση του προβλήματος αφού το δίκτυο αυτό, προσπαθούσε να υπολογίσει την δευτεροταγή δομή μιας πρωτεϊνικής ακολουθίας με βάση τα αμινοξέα που προηγούνται και έπονται του κάθε αμινοξέος. Είναι προφανές συνεπώς και ο λόγος για τον οποίο αυτή η έρευνα είχε αρκετά υψηλά αποτελέσματα επιτυχίας της τάξης του 76% Q3 στο συγκεκριμένο πρόβλημα.

Οι έρευνες γύρω από το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών των Armano et al. (2006) και των Chen και Chaudhari (2007), οι οποίοι χώρισαν τη διαδικασία πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, σε δύο στάδια. Το πρώτο στάδιο προβλέπει τη δευτεροταγή δομή με βάση την ακολουθία των αμινοξέων δηλαδή την πρωτοταγή δομή της πρωτεΐνης, κάτι το οποία ανταποκρίνεται απόλυτα στην φύση του προβλήματος, και ακολούθως το δεύτερο στάδιο χρησιμοποιεί τα δεδομένα του πρώτου σταδίου για να υπολογίσει την τελική δευτεροταγή δομή με βάση την αρχική πρόβλεψη. Υποστηρίζουν στην έρευνα τους ότι η δευτεροταγής δομή ενός αμινοξέος, εξαρτάται σε μεγάλο βαθμό από τη δευτεροταγή δομή των γειτονικών αμινοξέων του συγκεκριμένου αμινοξέος υπό μελέτη, κάτι το οποίο φαίνεται να ισχύει σε μεγάλο βαθμό, απλά και μόνο παρατηρώντας τις διάφορες ήδη γνωστές δευτεροταγής δομές διαφόρων πρωτεϊνών (Αγαθοκλέους, 2009).

Ένα από τα καλύτερα ποσοστά επιτυχίας που επιτεύχθηκαν μέχρι σήμερα, προέρχεται από την έρευνα των Wang et al. (2016), και είναι της τάξης του 83%. Η συγκεκριμένη μεθοδολογία χρησιμοποιεί βαθιά νευρωνικά πεδία με συνέλιξη και μειωμένες συνδέσεις μεταξύ των κρυφών επιπέδων που περιορίζονται στη σύνδεση ενός νευρώνα σε ένα κρυφό επίπεδο με μόνο τους τρείς νευρώνες στο προηγούμενο επίπεδο. Η μέθοδος αυτή όπως αναφέρεται και στην έρευνα τους συνδυάζει τα πλεονεκτήματα των Περιοδικών Νευρωνικών Πεδίων (Conditional Neural Fields - CNF) και των Βαθιών Συνελικτικών Νευρωνικών Δικτύων (Deep Convolutional Neural Networks – DCNN) με μετατροπές στην αρχιτεκτονική. Υποστηρίζεται ότι η χρήση του CNF μέρους στο στην μεθοδολογία τους εντοπίζει της γειτονικές συσχετίσεις των γειτονικών αμινοξέων που επηρεάζουν την δευτεροταγή δομή του αμινοξέος που μελετάται, ενώ με την αντικατάσταση του τυπικού νευρωνικού δικτύου με το DCNN δίκτυο εντοπίζονται όλες οι πολύπλοκες συσχετίσεις μεταξύ των δεδομένων εισόδου και των ετικετών εξόδου.

Είναι φανερό ότι η μεθοδολογία και η αρχιτεκτονική που επιλέχθηκαν έχουν πλήρη εφαρμογή στη φύση του προβλήματος εξού και τα πολύ υψηλά σχετικά αποτελέσματα πρόβλεψης.

Τέλος, η έρευνα του Παυλίδης (2016), προσπαθεί να λύσει το PSSP πρόβλημα με το να χρησιμοποιήσει την κλασική υλοποίηση των Συνελικτικών Νευρωνικών Δικτύων (Convolutional Neural Networks - CNN), για την αντιμετώπιση του PSSP προβλήματος. Η δυσκολία στην συγκεκριμένη έρευνα ήταν και ο τρόπος με τον οποίο θα έπρεπε να γίνει η αναπαράσταση των δεδομένων εισόδου στο δίκτυο, έτσι ώστε το δίκτυο να μπορέσει να εντοπίσει χαρακτηριστικά και να πραγματοποιήσει σωστές προβλέψεις. Η συγκεκριμένη έρευνα πέτυχε μόλις 40% επιτυχία και ο λόγος είναι ότι φαίνεται να υπήρχαν λάθη στην υλοποίηση του συνελικτικού νευρωνικού δικτύου, καθώς επίσης και στην αναπαράσταση των δεδομένων εισόδου, λόγω του ότι ένα ποσοστό της τάξης του 40% δηλώνει ότι το δίκτυο δεν εκπαιδεύεται σωστά.

Συνδέοντας όλες τις πιο πάνω σχετικές μελέτες, θα προσπαθήσουμε σε αυτή την έρευνα να συνδυάσουμε στοιχεία και ευρήματα από διαφορετικές έρευνες έτσι ώστε να προσπαθήσουμε να πετύχουμε μεγαλύτερα ποσοστά επιτυχίας στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

Κεφάλαιο 2

Βιολογικό Υπόβαθρο

2.1	Πρωτεΐνες και Αμινοξέα	21
2.2	Ο Βιολογικός Ρόλος των Πρωτεϊνών	26
2.3	Επίπεδα Οργάνωσης Πρωτεϊνών	27
	2.3.1 Πρωτοταγής Δομή	28
	2.3.2 Δευτεροταγής Δομή	29
	2.3.3 Τριτοταγής Δομή	30
	2.3.4 Τεταρτοταγής Δομή	30
2.4	Σημασία Γνώσης της Δομής των Πρωτεϊνών	31
2.5	Πρόβλεψη της Δομής των Πρωτεϊνών	32

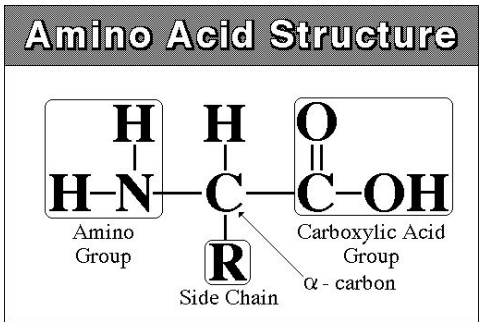
2.1 Πρωτεΐνες και Αμινοξέα

Στο κόσμο των βιομορίων υπάρχει μια ιεραρχία, την οποία πρέπει να αντιληφθούμε για να είμαστε σε θέση να καταλάβουμε καλύτερα τι είναι ακριβώς οι πρωτεΐνες. Αρχικά το κάθε κύτταρο αποτελείται από οργανίδια τα οποία δημιουργούνται από συμπλέγματα μακρομορίων. Τα μακρομόρια χωρίζονται σε τέσσερις κατηγορίες. Η πρώτη κατηγορία αποτελείται από τα νουκλεϊκά οξέα τα οποία σχηματίζονται από τα νουκλεοτίδια, η δεύτερη κατηγορία αποτελείται από τους πολυσακχαρίτες που δημιουργούνται από τους μονοσακχαρίτες, στη τρίτη κατηγορία ανήκουν τα λιπίδια τα οποία σχηματίζονται από τη γλυκερίνη και τα λιπαρά οξέα, και τέλος στην τέταρτη κατηγορία ανήκουν οι πρωτεΐνες οι οποίες δημιουργούνται από τα αμινοξέα (Johnson and Raven, 2001).

Οι πρωτεΐνες αποτελούν τα πιο διαδεδομένα και πολυδιάστατα, τόσο στη μορφή όσο και στη λειτουργία τους, μακρομόρια. Σχηματίζονται στα ριβοσώματα, με τη διαδικασία της πρωτεινοσύνθεσης, με τη διαδοχική συνένωση απλούστερων χημικών ενώσεων, των αμινοξέων. Ακόμη και σ΄ ένα απλό κύτταρο των βακτηρίων εντοπίζονται εκατοντάδες διαφορετικές πρωτεΐνες με την καθεμία εξ αυτών να έχει ιδιαίτερο ρόλο. Οι πρωτεΐνες αποτελούν είτε δομικά συστατικά των μεμβρανών του κυττάρου, είτε συνεργούν σε κάποια συγκεκριμένη λειτουργία, όπως η δημιουργία πρωτεϊνικών συμπλόκων. Είναι μεγάλα σύνθετα βιομόρια, με μοριακό βάρος από 10.000 μέχρι πάνω από 1 εκατομμύριο, αποτελούμενα από αμινοξέα (Σχήμα 2.1), τα οποία ενώνονται μεταξύ τους με πεπτιδικούς δεσμούς σχηματίζοντας μια γραμμική αλυσίδα, καλούμενη αλυσίδα πολυπεπτιδίων. Όλες οι πρωτεΐνες περιέχουν άνθρακα, οξυγόνο και άζωτο και οι περισσότερες εξ αυτών και θείο.

Αμινοξέα ονομάζονται οι χημικές ενώσεις που περιέχουν μία τουλάχιστον καρβονική ομάδα (από τα καρβονικά οξέα και μία τουλάχιστον αμινομάδα). Τα αμινοξέα αποτελούν τα βασικά δομικά στοιχεία των πρωτεϊνών που καθορίζουν και τις χαρακτηριστικές ιδιότητές τους. Τα περισσότερα αμινοξέα αποτελούνται από ένα ασύμμετρο άτομο άνθρακα συνδεδεμένο ομοιοπολικά με ένα άτομο υδρογόνου, μια καρβοξυλομάδα, μια αμινομάδα και μια πλευρική ομάδα (Σχήμα 2.1). Η μεταβλητότητα των αμινοξέων εμφανίζεται στην πλευρική ομάδα (R), προσδίδοντας στο αμινοξύ διαφορετικές ιδιότητες και κατ’ επέκταση κατηγοριοποίηση του σε

μια από τις επτά ομάδες (αμινοξέα με αλειφατικές - μη πολικές πλευρικές αλυσίδες, αμινοξέα με αλειφατικές - πολικές πλευρικές αλυσίδες, αμινοξέα με αρωματικές πλευρικές αλυσίδες, αμινοξέα με βασικές πλευρικές αλυσίδες, αμινοξέα με όξινες πλευρικές αλυσίδες, αμινοξέα με καρβοξυλαμίδια στις πλευρικές αλυσίδες τους, ειδικά αμινοξέα).



Σχήμα 2.1: Η δομή των αμινοξέων (Promponas, 2004).

Η ακολουθία αμινοξέων σε μια πρωτεΐνη, καθορίζεται από ένα γονίδιο και κωδικοποιείται κατά τον γενετικό κώδικα DNA. Παρόλο που ο γενετικός κώδικας κωδικοποιεί 20 αμινοξέα, τα αμινοξέα που συνιστούν την πρωτεΐνη, συχνά, υφίστανται χημικές αλλαγές κατά τη μετα-μεταγραφική τροποποίηση: προτού η πρωτεΐνη να μπορέσει να λειτουργήσει είτε στο κύτταρο, είτε ως τμήμα των μηχανισμών ελέγχου. Η αλληλουχία/ακολουθία των αμινοξέων καθορίζει τη μοναδική τρισδιάστατη δομή κάθε πρωτεΐνης και την ειδική λειτουργία της.

Πιο συγκεκριμένα, οι πρωτεΐνες στους περισσότερους οργανισμούς συντίθενται με τον πολυμερισμό 20 διαφορετικών τύπων L-a-αμινοξέων (Πίνακας 2.1). Οποιαδήποτε αλλαγή στη σειρά, τον αριθμό και την ποσότητα των αμινοξέων, οδηγεί στη δημιουργία μιας διαφορετικής πρωτεΐνης, η οποία έχει διαφορετικές ιδιότητες και εκτελεί διαφορετικές λειτουργίες. Η καταγραφή της αλληλουχίας μιας πρωτεΐνης μπορεί να γίνει πολύ εύκολα γράφοντας λέξεις με 20 διαφορετικά γράμματα τα οποία θα αντιστοιχούν το κάθε ένα σε ένα αμινοξύ (Πίνακας 2.1).

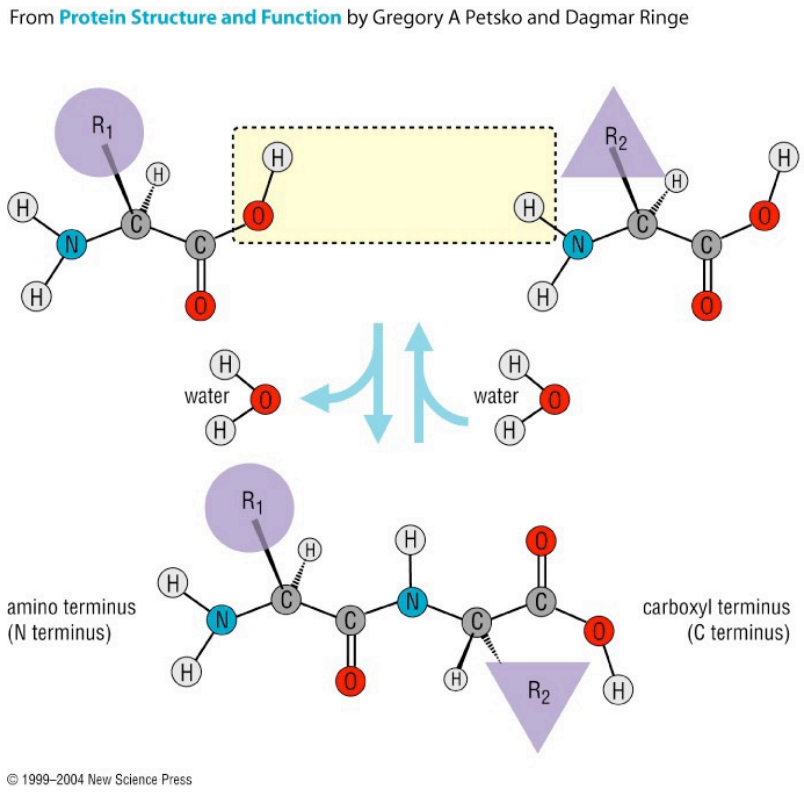
ΑΜΙΝΟΞΥ	ΣΥΜΒΟΛΙΣΜΟΣ		ΣΥΝΤΑΚΤΙΚΟΣ ΤΥΠΟΣ
Γλυκίνη	GLY	G	NH2-CH2-COOH
Αλανίνη	ALA	A	CH3-CH(NH2)-COOH
Βαλίνη	VAL	V	(CH3)2-CH-CH(NH2)-COOH
Ισολευκίνη	ILE	I	CH3-CH2-CH(CH3)-CH(NH2)-COOH
Λευκίνη	LEU	L	(CH3)2-CH-CH2-CH(NH2)-COOH
Φαινυλαλανίνη	PHE	F	Ph-CH2-CH(NH2)-COOH
Προλίνη	PRO	P	NH-(CH2)3-CH-COOH |_____|
Μεθειονίνη	MET	M	CH3-S-(CH2)2-CH(NH2)-COOH
Τρυπτοφάνη	TRP	W	Ph-NH-CH=C-CH2-CH(NH2)-COOH |_____|
Κυστεΐνη	CYS	C	HS-CH2-CH(NH2)-COOH
Σερίνη	SER	S	HO-CH2-CH(NH2)-COOH
Θρεονίνη	THR	T	CH3-CH(OH)-CH(NH2)-COOH
Ασπαραγίνη	ASN	N	H2N-CO-CH2-CH(NH2)-COOH
Γλουταμίνη	GLN	Q	H2N-CO-(CH2)2-CH(NH2)-COOH
Τυροσίνη	TYR	Y	HO-p-Ph-CH2-CH(NH2)-COOH
Ιστιδίνη	HIS	H	NH-CH=N-CH=C-CH2-CH(NH2)-COOH |_____|
Ασπαρτικό οξύ	ASP	D	HOOC-CH2-CH(NH2)-COOH
Γλουταμικό οξύ	GLU	E	HOOC-(CH2)2-CH(NH2)-COOH
Λυσίνη	LYS	K	H2N-(CH2)4-CH(NH2)-COOH
Αργινίνη	ARG	R	HN=C(NH2)-NH-(CH2)3-CH(NH2)-COOH

Πίνακας 2.1: Τα 20 φυσικά L-α-αμινοξέα που συμμετέχουν στο σχηματισμό των πρωτεϊνών (Αγαθοκλέους, 2009).

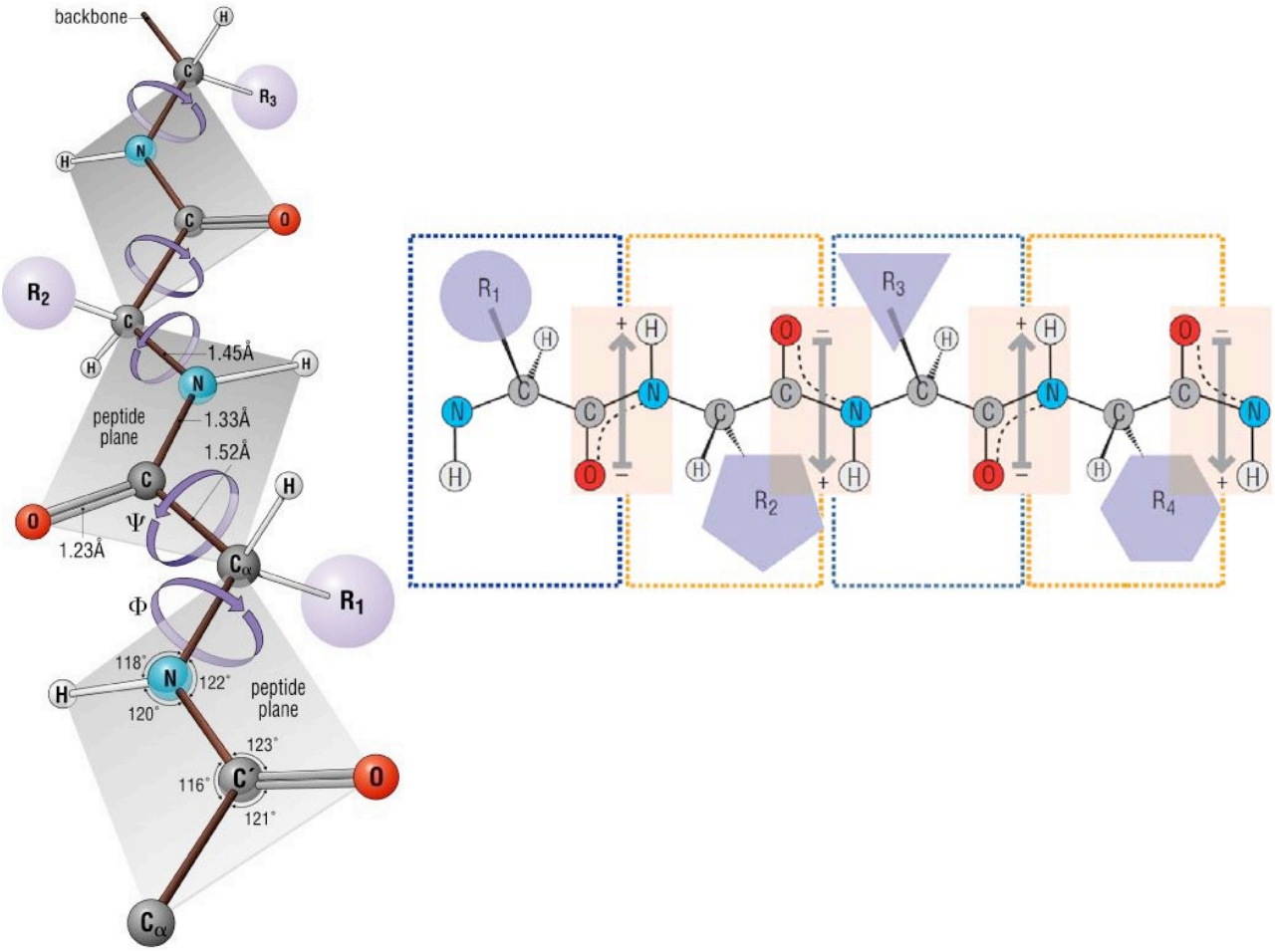
Όλα τα αμινοξέα έχουν φορτισμένη θετικά την αμινομάδα τους και φορτισμένη αρνητικά την καρβοξυλομάδα τους με αποτέλεσμα να συνδέονται εύκολα μεταξύ τους σε σειρές, με ομοιοπολικούς δεσμούς. Οι συνδέσεις αυτές των αμινοξέων σε μια σειρά, οδηγούν στην δημιουργία των διαφορετικών πρωτεϊνών. Ο πεπτιδικός δεσμός (Σχήμα 2.2), που σχηματίζουν δυο αμινοξέα όταν ενώνονται μεταξύ τους ονομάζεται δυπεπτίδιο, και με την πραγματοποίηση του αποβάλλεται ένα μόριο νερού. Αρχικά ένα πεπτίδιο, αμέσως μετά τη σύνθεσή του είναι ανενεργό, ανίκανο δηλαδή να εκδηλώσει το βιολογικό του ρόλο. Για να γίνει ενεργό πρέπει πρώτα να αποκτήσει την οριστική του στερεοδιάταξη, η οποία οργανώνεται σε τέσσερα επίπεδα: την πρωτοταγή, τη δευτεροταγή, την τριτοταγή και την τεταρτοταγή δομή (Σχήμα 2.4). Τα αμινοξέα, μετά την συνένωση τους, αναφέρονται ως αμινοξικά κατάλοιπα. Όλα τα αμινοξέα που ενώνονται με αυτό τον τρόπο σχηματίζουν ένα πολυπεπτίδιο ή μια πολυπεπτιδική αλυσίδα. Ο πεπτιδικός δεσμός είναι επίπεδος και σχεδόν άκαμπτος περιορίζοντας την περιστροφική ελευθερία μιας πολυπεπτιδικής αλυσίδας (Σχήμα 2.3). Παρατηρώντας τον πεπτιδικό δεσμό (Σχήμα 2.2), μπορούμε να παρατηρήσουμε την

επίπεδη διάταξη των πεπτιδικών δεσμών καθώς και την ελευθερία περιστροφής των επιπέδων αυτών γύρω από το μόριο άνθρακα. Η περιστροφές των επιπέδων, δεξιά και αριστερά από το μόριο άνθρακα, περιγράφονται με δυο δίεδρες γωνίες, φ και ψ. Αυτές οι ιδιότητες των πεπτιδικών δεσμών έχουν ως αποτέλεσμα να είναι αρκετά σταθεροί και έτσι τα αμινοξικά κατάλοιπα να μην μπορούν να περιστραφούν γύρω από τον εαυτό τους, ούτε να έχουν μεγάλο εύρος κινήσεων.

Η ποικιλομορφία που μπορεί να εμφανιστεί, στο σχηματισμό της πρωτεΐνης, από μία ή περισσότερες πολυπεπτιδικές αλυσίδες, αυξάνουν σημαντικά τον αριθμό διαφορετικών σε σχήμα και λειτουργία πρωτεϊνών. Ο μοναδικός τρόπος που αναδιπλώνονται οι πολυπεπτιδικές αλυσίδες στο χώρο, προσδίδουν στην πρωτεΐνη τη μοναδική τρισδιάστατη της μορφή και σχήμα. Η αλληλουχία, επομένως, των αμινοξέων στο πολυπεπτίδιο διαδραματίζει σημαντικό ρόλο ως προς το τελικό σχήμα που θα πάρει η πρωτεΐνη στο χώρο, άρα κατ' επέκταση και της λειτουργίας της πρωτεΐνης.



Σχήμα 2.2: Ο πεπτιδικός δεσμός (Promponas, 2004).



Σχήμα 2.3: Τα επίπεδα και οι γωνίες που σχηματίζουν οι πεπτιδικοί δεσμοί (Promponas, 2004).

Εκτός από την σημαντικότητα της παρουσίας των πρωτεϊνών στον οργανισμό μας, ιδιαίτερης σημασίας χρίζει και η κατανόηση της δομής και της λειτουργίας τους. Διάφορες πληροφορίες που αφορούν την δομή των πρωτεϊνών, μπορούν να βοηθήσουν στην κατασκευή συμπληρωμάτων διατροφής, φαρμάκων και αντιβιοτικών. Με την τροποποίηση της αλληλουχίας μιας πρωτεΐνης, μπορούμε να διορθώσουμε σοβαρές ασθένειες και παθήσεις, ενώ υπάρχει η δυνατότητα δημιουργίας νέων πρωτεϊνών με νέες ιδιότητες. Επιπρόσθετα, συσχετίζοντας την πρωτοταγή δομή μιας πρωτεΐνης με την αναδίπλωση της στο χώρο, μπορούμε να καθορίσουμε κανόνες που καθορίζουν τις αναδιπλώσεις για κάθε ξεχωριστή αλληλουχία αμινοξέων. Πρωτεΐνες που έχουν παρόμοια αλληλουχία αμινοξέων (πρωτοταγή δομή), μας δίνουν πληροφορίες για τη εξελικτική ιστορία των πρωτεϊνών.

2.2 Ο Βιολογικός Ρόλος των Πρωτεϊνών

Οι διάφορες λειτουργίες που παρατηρούνται στους οργανισμούς γίνονται χάρη των πρωτεϊνών. Ο δε βιολογικός τους ρόλος καθορίζεται κάθε φορά από την τρισδιάστατη δομή τους, που είναι συνέπεια της αλληλουχίας των αμινοξέων, ή αλλιώς, της πρωτοταγής δομής της πρωτεΐνης.

Όπως και διάφορα άλλα βιολογικά μακρομόρια (π.χ. οι πολυσακχαρίτες, τα λιπίδια, και νουκλεϊκά οξέα) έτσι και οι πρωτεΐνες είναι απαραίτητες για όλους τους ζωντανούς οργανισμούς, και συμμετέχουν σε κάθε διαδικασία μέσα στα κύτταρα. Πολλές πρωτεΐνες δρουν ως ένζυμα που καταλύουν τις βιοχημικές αντιδράσεις, και είναι ζωτικής σημασίας στο μεταβολισμό. Άλλες πρωτεΐνες έχουν δομικές ή μηχανικές λειτουργίες, όπως οι πρωτεΐνες του κυτταρικού σκελετού οι οποίες συμβάλλουν στη διατήρηση της μορφής των κυττάρων. Οι πρωτεΐνες είναι επίσης σημαντικές στη διακυτταρική επικοινωνία, τη δράση του ανοσοποιητικού συστήματος, τον σχηματισμό κυτταρικών ιστών, και τον κυτταρικό κύκλο.

Οι πρωτεΐνες είναι απαραίτητα συστατικά στη διατροφή μας, δεδομένου ότι τα ζώα, όπως και οι άνθρωποι δεν μπορούν να συνθέσουν όλα τα αμινοξέα, αλλά πρέπει να τα λάβουν από τα τρόφιμα. Μέσω της διαδικασίας της πέψης, τα ζώα και οι άνθρωποι αποικοδομούν την πρωτεΐνη στα ελεύθερα αμινοξέα που μπορούν να χρησιμοποιηθούν για την πρωτεϊνική σύνθεση.

Στο ανθρώπινο σώμα υπάρχουν περισσότερες από 30.000 διαφορετικές πρωτεΐνες, ενώ ακόμη και σε απλά κύτταρα βακτηρίων μπορεί κανείς να συναντήσει εκατοντάδες πρωτεΐνες. Έχουν δομικό και λειτουργικό ρόλο, δηλαδή αποτελούν δομικά στοιχεία του κυττάρου και συμβάλουν στην ανάπτυξη και τη λειτουργία όλων των οργανισμών. Είναι υπεύθυνες για την άμυνα και την προστασία του οργανισμού, τη μεταφορά θρεπτικών ουσιών, την αποκατάσταση των φθορών στους ιστούς, συμβάλουν στην κίνηση και τη στήριξη του σώματος, καθώς και αμέτρητων άλλων λειτουργιών.

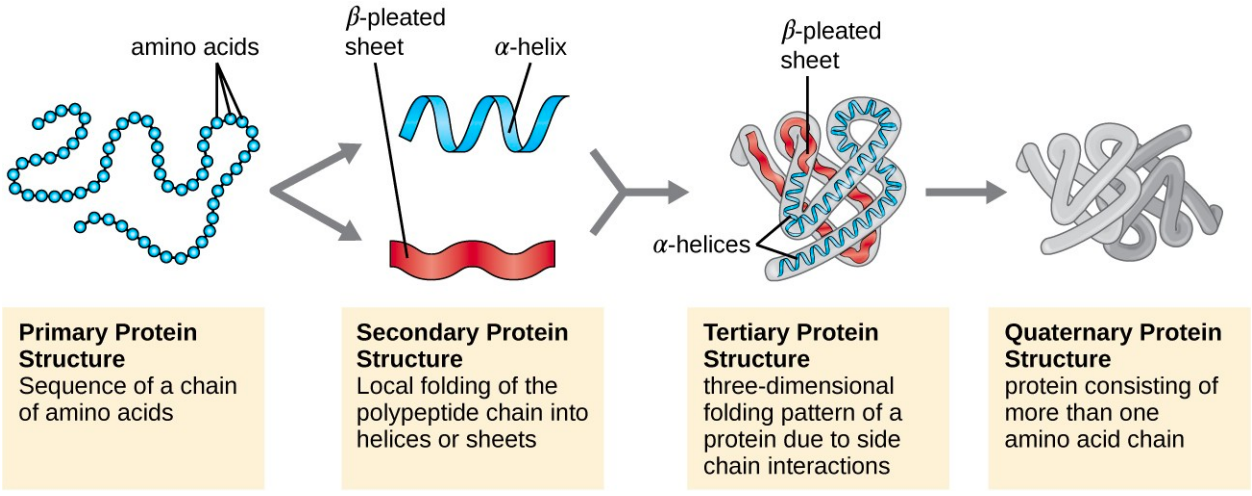
Οι πρωτεΐνες μπορούν να περιγραφούν σύμφωνα με τη μεγάλη ποικιλία των λειτουργιών τους στον οργανισμό, με αλφαβητική σειρά, κάποιες από τις οποίες βλέπουμε στον πίνακα 2.2.

Λειτουργία	Περιγραφή
Αντίσωμα	Τα αντισώματα συνδέονται με συγκεκριμένα ξένα σωματίδια, όπως ιούς και βακτήρια, για να βοηθήσουν στην προστασία και ενίσχυση του σώματος.
Ένζυμο	Τα ένζυμα πραγματοποιούν σχεδόν όλες τις χιλιάδες χημικές αντιδράσεις που λαμβάνουν χώρα στα κύτταρα. Βοηθούν επίσης στο σχηματισμό νέων μορίων διαβάζοντας τις γενετικές πληροφορίες που είναι αποθηκευμένες στο DNA.
Αγγελιαφόρος	Οι πρωτεΐνες «Αγγελιαφόροι», όπως ορισμένοι τύποι ορμονών, μεταδίδουν σήματα για τον συντονισμό βιολογικών διεργασιών μεταξύ διαφορετικών κυττάρων, ιστών και οργάνων.
Δομικό στοιχείο	Αυτές οι πρωτεΐνες παρέχουν δομή και υποστήριξη για τα κύτταρα. Σε μεγαλύτερη κλίμακα, επιτρέπουν επίσης στο σώμα να μετακινηθεί.
Μεταφορά /αποθήκευση	Αυτές οι πρωτεΐνες συνδέουν και μεταφέρουν άτομα και μικρά μόρια μέσα στα κύτταρα και σε ολόκληρο το σώμα.

Πίνακας 2.2: Βιολογικοί ρόλοι πρωτεϊνών

2.3 Επίπεδα Οργάνωσης των Πρωτεϊνών

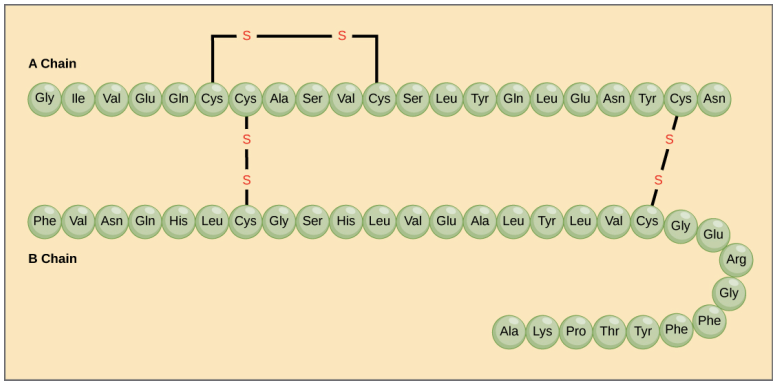
Όπως αναφέραμε προηγουμένως, η δομή των πρωτεϊνών αποτελείται από τέσσερα επίπεδα και περιλαμβάνει την πρωτοταγή, δευτεροταγή, τριτοταγή και τεταρτοταγή δομή. Η γνώση του αριθμού και του είδους των αμινοξέων από τα οποία αποτελείται μια πρωτεΐνη δεν είναι αρκετή, αφού τεράστιο ρόλο παίζει η σειρά και διάταξη των αμινοξέων που αποτελούν μια πρωτεΐνη, αφού η αλληλουχία αυτή καθορίζει την τρισδιάστατη δομή και συνεπώς τη λειτουργία της εν λόγω πρωτεΐνης. Πιο κάτω φαίνονται (Σχήμα 2.4) και περιγράφονται αναλυτικά τα τέσσερα επίπεδα της δομής μιας πρωτεΐνης.



Σχήμα 2.4: Τα τέσσερα επίπεδα δομής των πρωτεϊνών (CNX OpenStax, 2016).

2.3.1 Πρωτοταγής δομή

Το απλούστερο επίπεδο της πρωτεϊνικής δομής, η πρωτοταγής δομή, είναι απλά η αλληλουχία των αμινοξέων σε μια πολυπεπτιδική αλυσίδα. Οι θέσεις που κατέχουν τα αμινοξέα δεν είναι τυχαίες, αλλά καθορίζονται από το γενετικό υλικό, καθώς επίσης είναι γνωστό ότι το περιβάλλον που εκτίθενται τα διάφορα αμινοξέα επηρεάζει τη σειρά αυτή. Για παράδειγμα, η ορμόνη ινσουλίνη έχει δύο πολυπεπτιδικές αλυσίδες, Α και Β, που παρουσιάζονται στο παρακάτω διάγραμμα. (Το μόριο ινσουλίνης που παρουσιάζεται εδώ είναι ινσουλίνη αγελάδας, παρόλο που η δομή του είναι παρόμοια με εκείνη της ανθρώπινης ινσουλίνης). Κάθε αλυσίδα έχει τη δική της ομάδα αμινοξέων, συναρμολογημένη με συγκεκριμένη σειρά ή διάταξη. Για παράδειγμα, η αλληλουχία της αλυσίδας Α αρχίζει με γλυκίνη, τελειώνει με ασπαραγίνη και είναι διαφορετική από την αλληλουχία της Β αλυσίδας.



Σχήμα 2.5: Πρωτοταγής δομή πρωτεΐνης. Πρωτεΐνη που αποτελείται από δύο αλυσίδες διαφόρων αμινοξέων (CNX OpenStax, 2016).

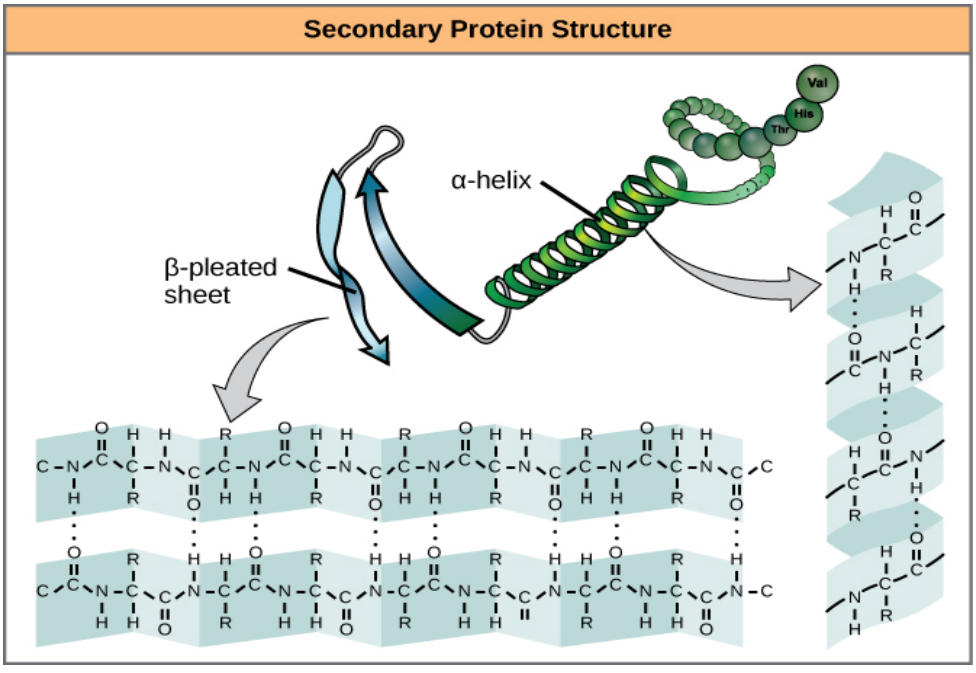
Η αλληλουχία μιας πρωτεΐνης προσδιορίζεται από το DNA του γονιδίου που κωδικοποιεί την πρωτεΐνη (ή που κωδικοποιεί ένα τμήμα της πρωτεΐνης, για πρωτεΐνες πολλαπλών υπομονάδων). Μία αλλαγή στην αλληλουχία DNA του γονιδίου μπορεί να οδηγήσει σε μια αλλαγή στην αλληλουχία αμινοξέων της πρωτεΐνης. Ακόμα και η αλλαγή μόνο ενός αμινοξέος σε μια αλληλουχία πρωτεΐνης μπορεί να επηρεάσει τη συνολική δομή και λειτουργία της πρωτεΐνης.

Για παράδειγμα, μια αλλαγή ενός αμινοξέος σχετίζεται με τη δρεπανοκυτταρική αναιμία, μια κληρονομική ασθένεια που επηρεάζει τα ερυθρά αιμοσφαίρια. Στην αναιμία των

δρεπανοκυττάρων, μία από τις πολυπεπτιδικές αλυσίδες που συνιστούν την αιμοσφαιρίνη, η πρωτεΐνη που μεταφέρει οξυγόνο στο αίμα, έχει μια μικρή αλλαγή στην ακολουθία των αμινοξέων. Το γλουταμινικό, που είναι κανονικά το έβδομο αμινοξύ της β αλυσίδας αιμοσφαιρίνης (ένας από τους δύο τύπους πρωτεϊνικών αλυσίδων που αποτελούν την αιμοσφαιρίνη), αντικαθίσταται από βαλίνη.

2.3.2 Δευτεροταγής δομή

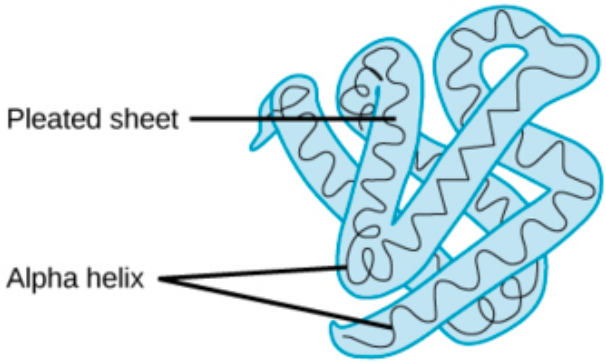
Η δευτεροταγής δομής αναφέρεται σε τοπικές αναδιπλούμενες δομές που σχηματίζονται σε ένα πολυπεπτίδιο, λόγω των αλληλεπιδράσεων μεταξύ των ατόμων της ραχοκοκαλιάς (Η ραχοκοκαλιά αναφέρεται στην πολυπεπτιδική αλυσίδα εκτός από τις ομάδες R - οπότε το μόνο που εννοούμε εδώ είναι ότι η δευτεροταγής δομή δεν περιλαμβάνει άτομα της ομάδας R). Μελέτες που διεξήχθησαν με χρήση κρυσταλλογραφιών ακτινών Χ, έδειξαν ότι οι αναδιπλώσεις αυτές χωρίζονται σε δυο κύριες κατηγορίες, την κατηγορία α - έλικας και την κατηγορία β - πτυχωτή επιφάνεια. Οι δύο αυτές δομές διατηρούν το σχήμα τους με δεσμούς υδρογόνου, οι οποίοι σχηματίζονται μεταξύ του μορίου οξυγόνου (Ο) του ενός αμινοξέος και του μορίου υδρογόνου (Η) του άλλου.



Σχήμα 2.6: Δευτεροταγής δομικά στοιχεία πρωτεινων. Αριστερά: Παράδειγμα αναδίπλωσης β – πτυχωτής επιφάνειας. Δεξιά Παράδειγμα αναδίπλωσης α – έλικας (CNX OpenStax, 2016).

2.3.3 Τριτοταγής δομή

Η τριτοταγής δομή μιας πρωτεΐνης αναφέρεται στην τρισδιάστατη δομή της, η οποία αναπαρίσταται με τις τριών διαστάσεων συντεταγμένες του κάθε ατόμου της πρωτεΐνης. Η τριτοταγής δομή μιας πρωτεΐνης εξαρτάται από πολλαπλά στοιχεία της δευτεροταγούς δομής της, και είναι γενικά το αποτέλεσμα των αλληλεπιδράσεων των πλευρικών αλυσίδων μεταξύ των διαφόρων αμινοξέων. Συνεπώς, η διάταξη των αμινοξέων μιας πρωτεΐνης (πρωτοταγής δομή), σχηματίζει την δευτεροταγή δομή της πρωτεΐνης και στη συνέχεια οι τοπικές αναδιπλούμενες δομές της δευτεροταγούς δομής της πρωτεΐνης καθορίζουν την τριτοταγή δομή της πρωτεΐνης, δηλαδή το τελικό σχήμα και λειτουργία της πρωτεΐνης.



Σχήμα 2.7: Τριτοταγής δομή πρωτεινης, αποτελούμενη απο αναδιπλώσεις α – ελίκων και β – πτυχωτών επιφανειών (CNX OpenStax, 2016).

2.3.4 Τεταρτοταγής δομή

Η τεταρτοταγής δομή μιας πρωτεΐνης αντιπροσωπεύει την αλληλεπίδραση μεταξύ πολλαπλών πολυπεπτιδικών αλυσίδων. Η αλληλεπίδραση αυτή μεταξύ των διαφόρων πολυπεπτιδικών αλυσίδων οφείλεται στις μη ομοιοπολικές αλληλεπιδράσεις μεταξύ των ατόμων διαφορετικών πολυπεπτιδικών αλυσίδων. Κάποια παραδείγματα αυτών των αλληλεπιδράσεων, περιλαμβάνουν μεταξύ άλλων τους δεσμούς υδρογόνου, τις Van Der Walls αλληλεπιδράσεις, τους δεσμούς ιόντων, και τους δεσμούς δισουλφιδίου.

Σχήμα 2.8: Τεταρτοταγής δομή πρωτεινών. Είναι η αναδίπλωση δύο ή περισσοτέρων πολυπεπτιδικών αλυσίδων (CNX OpenStax, 2016).

2.4 Σημασία Γνώσης της Δομής των Πρωτεϊνών

Όπως έχουμε αναφέρει προηγουμένως, η πρωτοταγής δομή μιας πρωτεΐνης, δηλαδή η αλληλουχία των αμινοξέων που αποτελούν την πρωτεΐνη, καθορίζει την δευτεροταγή αλλά και τριτοταγή δομή της πρωτεΐνης αυτής. Η δομή μιας πρωτεΐνης εξαρτάται, εκτός από τις φυσικές και χημικές ιδιότητες των αμινοξέων και των μεταξύ τους αλληλεπιδράσεων (δηλαδή τους πεπτιδικούς δεσμούς που δημιουργούνται), και από το περιβάλλον στο οποίο εκτίθεται η πρωτεΐνη (θερμοκρασία, pH κτλ.). Αυτό αποδείχθηκε πειραματικά αφού μια πρωτεΐνη με συγκεκριμένη τρισδιάστατη μορφή σε ένα συγκεκριμένο περιβάλλον, μεταφέρθηκε σε διαφορετικό περιβάλλον από το προηγούμενο, και ως αποτέλεσμα είχαμε την αναδιάταξη της πρωτεΐνης, δηλαδή την αλλαγή της τρισδιάστατης της μορφής. Ως αποτέλεσμα αυτής της αναδιάταξης της πρωτεΐνης, έχουμε και την αλλαγή της βιολογικής της λειτουργίας. Στην συνέχεια, η ίδια πρωτεΐνη μεταφέρθηκε πίσω στο αρχικό της περιβάλλον και ως αποτέλεσμα αυτού, έχουμε την επαναφορά της πρωτεΐνης στην αρχική της τρισδιάστατη μορφή. Με την αναδιάταξη της τρισδιάστατης μορφής της πρωτεΐνης στη μορφή που είχε αρχικά, έχουμε και την επαναφορά της αρχικής της βιολογικής λειτουργίας. Από τα πιο πάνω φαίνεται ξεκάθαρα η άμεση συσχέτιση του περιβάλλοντος και της τρισδιάστατης μορφής της πρωτεΐνης, καθώς επίσης και η άμεση συσχέτιση της τρισδιάστατης μορφής της πρωτεΐνης και της βιολογικής της λειτουργίας. Η βιολογική λειτουργία μιας πρωτεΐνης εξαρτάται αποκλειστικά από την τρισδιάστατης της μορφή στο χώρο, δηλαδή την τριτοταγή της δομή.

Η σημαντικότητα της γνώσης της ακριβούς δομής των πρωτεϊνών, διαφαίνεται στο γεγονός ότι, γνωρίζοντας την τρισδιάστατη μορφή (τριτοταγής δομή) μιας πρωτεΐνης μπορούμε να καθορίσουμε επακριβώς την λειτουργία της συγκεκριμένης πρωτεΐνης, όπως επίσης και να είμαστε σε θέση να την μελετήσουμε σε βάθος.

2.5 Πρόβλεψη της Δομής των Πρωτεϊνών

Η έρευνα στο τομέα της πρόβλεψης της δομής των πρωτεϊνών, βασίζεται κυρίως στην πρόβλεψη της δευτεροταγούς και τριτοταγούς δομής της πρωτεΐνης, από την πειραματικά γνωστή πρωτοταγή δομή (διάταξη/ακολουθία των αμινοξέων) της πρωτεΐνης. Το γεγονός αυτό, οφείλεται κυρίως στο ότι είναι σχετικά εύκολο να προσδιορίσουμε την πρωτοταγή δομή μιας πρωτεΐνης, στο ότι οι υπόλοιπες δομές της πρωτεΐνης εξαρτώνται κυρίως από την πρωτοταγή δομή της πρωτεΐνης, και τέλος, λόγω της πολυπλοκότητας που υπάρχει στο να καθορίσουμε επακριβώς την τριτοταγή και τεταρτοταγή δομή μιας πρωτεΐνης.

Όπως αναφέραμε προηγουμένως, η αλληλουχία των αμινοξέων μιας πρωτεΐνης (πρωτοταγής δομή), καθορίζει τόσο τη δευτεροταγή, όσο και την τριτοταγή δομή της πρωτεΐνης. Η σειρά με την οποία βρίσκονται διατεταγμένα τα αμινοξέα μιας πρωτεΐνης, σχηματίζουν τις τοπικές αναδιπλώσεις (δευτεροταγής δομή), και αυτές οι τοπικές αναδιπλώσεις με την σειρά τους αλληλεπιδρούν μεταξύ τους, σχηματίζοντας έτσι την τρισδιάστατη μορφή (τριτοταγής δομή) της πρωτεΐνης, η οποία καθορίζει και την βιολογική λειτουργία της πρωτεΐνης. Πέραν αυτού, η τρισδιάστατη μορφή μιας πρωτεΐνης, επηρεάζεται από διάφορους παράγοντες όπως θερμοκρασία, pH κτλ. Η μεταφορά μιας πρωτεΐνης σε ένα διαφορετικό περιβάλλον ύπαρξης, προκαλεί την αναδιάταξη της τρισδιάστατης μορφής της πρωτεΐνης. Πιο απλά, χάνεται η προηγούμενη βιολογική της λειτουργία, αφού η τρισδιάστατή της μορφή αλλάζει. Αξίζει να σημειωθεί ότι είναι εξαιρετικά βοηθητικό να γνωρίζουμε τη δευτεροταγή δομή μιας πρωτεΐνης για να μεταβούμε από την πρωτοταγή στην τριτοταγή δομή της πρωτεΐνης.

Όπως εξηγήσαμε ήδη, η μελέτη και ο πειραματικός προσδιορισμός της δευτεροταγούς και τριτοταγούς δομής μιας πρωτεΐνης, είναι μια χρονοβόρα αλλά και εξαιρετικά δαπανηρή διαδικασία. Συγκεκριμένα, για την εξαγωγή της τρισδιάστατης μορφής (τριτοταγής δομής) των πρωτεϊνών, απαιτούνται κάποιες διαδικασίες, όπως η μέθοδος της κρυσταλλογραφίας ακτινών

X, και του πυρηνικού μαγνητικού συντονισμού (Nuclear Magnetic Resonance - NMR) (Αγαθοκλέους, 2009). Επιπρόσθετα γνωρίζουμε ότι η μορφή μιας πρωτεΐνης καθορίζεται πλήρως από την πρωτοταγή της δομή, ενώ περίπου το 70% της δευτεροταγούς της δομής επηρεάζεται από τις αλληλεπιδράσεις των κοντινών αμινοξέων της ραχοκοκαλιάς, και το υπόλοιπο 30% επηρεάζεται από πιο μακρινές αλληλεπιδράσεις (Berg et al., 2002). Εξαιτίας αυτού, οι διάφορες τεχνικές και υλοποιήσεις που προσπαθούν να προβλέψουν αντί να εξάγουν πειραματικά, τη δευτεροταγή και τριτοταγή δομή μιας πρωτεΐνης, είναι πολύ διαδεδομένες αφού δίνουν αρκετά καλά αποτελέσματα επιτυχίας στην πρόβλεψη της δευτεροταγούς και τριτοταγούς δομής των πρωτεϊνών, σε σχετικά σύντομο χρονικό διάστημα, έχοντας ακόμα το πλεονέκτημα του χαμηλού κόστους. Μια τέτοια μέθοδος πρόβλεψης, είναι αυτή της εξ αρχής πρόβλεψης (ab initio prediction), η οποία προσπαθεί να προβλέψει οποιαδήποτε από τις τρείς (3) υπόλοιπες δομές της πρωτεΐνης χρησιμοποιώντας μόνο την διάταξη των αμινοξέων της πρωτεΐνης (πρωτοταγής δομή), χωρίς να λαμβάνει υπόψη οποιοδήποτε πρότυπο. Αυτή η μέθοδος χωρίζεται σε δύο διαφορετικές περιπτώσεις. Για την πρώτη περίπτωση, η μέθοδος αυτή προσομοιώνει την διεργασία αναδίπλωσης ή προσπαθεί να ελαχιστοποιήσει την ελεύθερη ενέργεια του πολυπεπτιδίου. Σε αυτή την περίπτωση δεν χρησιμοποιούνται άλλες γνωστές δομές αλλά μόνο η πρωτοταγής δομή της πρωτεΐνης. Η δεύτερη μέθοδος στηρίζεται σε προηγούμενη γνώση, προσπαθώντας έτσι να προβλέψουμε την δομή μιας πρωτεΐνης με βάση ήδη γνωστές και υπάρχουσες δομές πρωτεϊνών (Berg et al., 2002). Η χρήση τεχνητών νευρωνικών δικτύων για την πρόβλεψη της δομής μιας πρωτεΐνης, ανήκει στη δεύτερη κατηγορία και στα πλαίσια αυτής της διπλωματικής εργασίας θα επικεντρωθούμε εξολοκλήρου σε αυτή.

Συνοψίζοντας τα πιο πάνω, το πρόβλημα βρίσκεται στην αδυναμία πειραματικού καθορισμού της δομής όλων των πρωτεϊνών, λόγω του κόστους που έχουν αυτές οι μέθοδοι σε χρόνο και χρήμα. Στα πλαίσια αυτής της διπλωματικής εργασίας θα προσπαθήσουμε να χρησιμοποιήσουμε τα συνελικτικά νευρωνικά δίκτυα (Convolutional Neural Networks - CNNs) σε συνδυασμό με φίλτρα Gabor και τα Support Vector Machines (SVMs), έτσι ώστε να μπορέσουμε να προβλέψουμε την δευτεροταγή δομή των πρωτεϊνών με τα μεγαλύτερα δυνατά ποσοστά επιτυχίας.

Η σημαντικότητα της επίλυσης του συγκεκριμένου προβλήματος δηλαδή της πρόβλεψης της δευτεροταγούς δομής μιας πρωτεΐνης (Protein Secondary Structure Prediction - PSSP), φαίνεται ξεκάθαρα στο ότι η δευτεροταγής δομή μιας πρωτεΐνης λειτουργεί ως ενδιάμεσο βήμα για τον καθορισμό/προσδιορισμό και της τριτοταγούς δομής της εν λόγω πρωτεΐνης. Η δευτεροταγής δομή μιας πρωτεΐνης, μας δίνει πάρα πολλά στοιχεία για την καθορισμό της τριτοταγούς δομής της πρωτεΐνης, άρα και τη γνώση της λειτουργίας της. Οι πειραματικές μέθοδοι προσδιορισμού της τριτοταγούς δομής είναι χρονοβόρες και ασύλληπτα δαπανηρές, αφήνοντας την επιστημονική κοινότητα με μερική μόνο γνώση των λειτουργιών των πρωτεϊνών, αφού μόλις μερικές χιλιάδες (από τις εκατομμύρια που υπάρχουν) πρωτεΐνες έχουν μελετηθεί. Συνεπώς, καθορίζοντας την δευτεροταγή δομή μιας πρωτεΐνης μπορούμε να αποφανθούμε με μεγαλύτερη ακρίβεια και ευκολία για την τριτοταγή δομή της πρωτεΐνης. Αξίζει να σημειωθεί, ότι όλες οι βιολογικές λειτουργίες που μπορεί να εκτελέσει μια πρωτεΐνη, βασίζονται στην διάταξη των είκοσι αμινοξέων που υπάρχουν και μπορούν να συνθέσουν μια πρωτεΐνη. Αυτός είναι και ο πραγματικός λόγος για τον οποίο η μελέτη των πρωτεϊνών, της δομής, της σύνθεσης και της λειτουργίας τους, χρίζει υψίστης σημασίας. Πρέπει να είμαστε σε θέση να μπορούμε να κατανοήσουμε πως αυτά τα μόρια αναδιπλώνονται στο χώρο, πώς αλληλεπιδρούν μεταξύ τους, πώς συναρμολογούνται σε συγκροτήματα και πώς λειτουργούν, εάν θέλουμε να απαντήσουμε σε ερωτήματα όπως γιατί κάποιοι πάσχουν από καρκίνο, γιατί γερνάμε, γιατί αρρωσταίνουμε, πώς μπορούμε να βρούμε θεραπείες για διάφορες ασθένειες, γιατί η ζωή όπως γνωρίζουμε έχει εξελιχθεί με αυτόν τον τρόπο σε αυτόν τον πλανήτη και όχι οπουδήποτε αλλού, τουλάχιστον προς το παρόν.

Όλες οι λειτουργίες των πρωτεϊνών εξαρτώνται από τη δομή τους, η οποία με τη σειρά της, εξαρτάται από φυσικές και χημικές παραμέτρους. Αυτό είναι ένα σημαντικό γεγονός για τη μελέτη αυτών των μορίων. Οι κλασσικές βιολογικές, φυσικές, χημικές, μαθηματικές και πληροφοριακές επιστήμες, συνεργάζονται σε ένα νέο τομέα γνωστό ως ο τομέας της βιοπληροφορικής, για να μπορέσουν δημιουργήσουν ένα νέο επίπεδο γνώσης και να δώσουν απαντήσεις στα τεράστια ερωτήματα γύρω από τις πρωτεΐνες και γενικότερα την οργάνωση της ζωής.

Κεφάλαιο 3

Τεχνητή Νοημοσύνη και Τεχνητά Νευρωνικά Δίκτυα

3.1 Τεχνητή Νοημοσύνη (Artificial Intelligence) 37

3.2 Μηχανική Μάθηση (Machine Learning) 39

3.2.1 Γενικά 39

3.2.2 Είδη Μάθησης 40

3.2.2.1 Επιβλεπόμενη Μάθηση (Supervised Learning) 40

3.2.2.2 Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning) 41

3.2.2.3 Ενισχυτική Μάθηση (Reinforcement Learning) 42

3.3 Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks) 44

3.3.1 Πηγή Έμπνευσης Τεχνητών Νευρωνικών Δικτύων 44

3.3.2 Αρχιτεκτονική και Λειτουργία Τεχνητών Νευρωνικών Δικτύων 46

3.3.3 Μοντέλο Νευρώνα McCulloch και Pitts 49

3.3.4 Πολυστρωματικά Δίκτυα Perceptron Εμπρόσθιου Περάσματος 51

3.3.4.1 Εισαγωγή στα Δίκτυα Perceptron 51

3.3.4.2 Αλγόριθμος Μάθησης Perceptron 53

3.3.4.3 Κανόνας Δέλτα 55

3.3.4.4 Μέθοδος Κατάβασης Κλίσης 55

3.3.4.5 Παράγοντας Ορμής 56

3.3.4.6 Αλγόριθμος Ανάστροφης Μετάδοσης Σφάλματος 58

3.3.5 Συνελικτικά Νευρωνικά Δίκτυα 60
3.3.5.1 Εισαγωγή στα Συνελικτικά Νευρωνικά Δίκτυα 60
3.3.5.2 Αρχιτεκτονική Συνελικτικών Νευρωνικών Δικτύων 61
3.3.5.3 Επεξήγηση Λειτουργίας Συνελικτικών Νευρωνικών Δικτύων 66
3.3.6 Support Vector Machines (SVMs) 71
3.3.6.1 Εισαγωγή στα SVMs 71
3.3.6.2 Επεξήγηση Βασικής Λειτουργίας του SVM 72
3.3.6.3 Συνδυασμός CNNs και SVMs 73
3.3.7 Συναρτήσεις ενεργοποίησης 73
3.3.7.1 Βηματική Συνάρτηση - Συνάρτηση Σκαλί 73
3.3.7.2 Σιγμοειδής Συνάρτηση 74
3.3.7.3 Συνάρτηση Rectifier 76
3.3.7.4 Softmax Συνάρτηση 77
3.3.8 Vanishing και Exploding Gradient Problem 78
3.3.9 Updaters για Διαχείριση της Μεταβολής του Ρυθμού Μάθησης 79
3.3.9.1 Γενικά 79
3.3.9.2 Ορμή 80
3.3.9.3 Adagrad (Adaptive Gradient Algorithm) 81
3.3.9.4 RMSProp 82
3.3.9.5 AdaDelta 82
3.3.9.6 ADAM (Adaptive Moment Estimation) 82

3.1 Τεχνητή Νοημοσύνη (Artificial Intelligence)

Ονομάζουμε τους εαυτούς μας Homo Sapiens (Σοφός Άνθρωπος), επειδή η νοημοσύνη του ανθρώπου χρίζει τεράστιας σημασίας για την ανθρωπότητα. Εδώ και χιλιάδες χρόνια προσπαθούσαμε και προσπαθούμε ακόμα να καταλάβουμε τον τρόπο που σκεπτόμαστε, εννοώντας ότι προσπαθούμε να καταλάβουμε το πώς μια χούφτα ύλης όπως είναι ο άνθρωπος, μπορεί να αντιληφθεί, να καταλάβει, να προβλέψει και τέλος να χειραγωγήσει ένα κόσμο που είναι πολύ μεγαλύτερος και πιο πολύπλοκος από αυτόν. Η μελέτη του τομέα της τεχνητής νοημοσύνης (Artificial Intelligence - AI), προσπαθεί όχι μόνο να καταλάβει, αλλά και να δημιουργήσει όντα με ευφυία.

Το πεδίο της τεχνητής νοημοσύνης είναι ένα από τα νεότερα πεδία των θετικών επιστημών και της μηχανικής. Η μελέτη σχετικά με το AI άρχισε σύντομα μετά τον δεύτερο παγκόσμιο πόλεμο και ο τίτλος τεχνητή νοημοσύνη θεσπίστηκε το 1956. Μαζί με τη μοριακή βιολογία, η τεχνητή νοημοσύνη αναφέρεται συχνά ως το «επιστημονικό πεδίο που θα ήθελα περισσότερο να συμμετάσχω» από διάφορους επιστήμονες σε άλλους κλάδους.

Η τεχνητή νοημοσύνη περιλαμβάνει σήμερα μια τεράστια ποικιλία υποπεδίων, αποδεικνύοντας μαθηματικά θεωρήματα, γράφοντας ποίηση, δημιουργώντας αυτόνομα οχήματα, διαγιγνώσκοντας ασθένειες και πολλά άλλα. Το AI είναι σχετικό με οποιαδήποτε πνευματική εργασία και έχει εφαρμογή σε οποιοδήποτε πεδίο.

Πιο συγκεκριμένα ο όρος τεχνητή νοημοσύνη αναφέρεται στον κλάδο της πληροφορικής ο οποίος ασχολείται με τον σχεδιασμό και την υλοποίηση υπολογιστικών συστημάτων που προσπαθούν να μιμηθούν διάφορα στοιχεία της ανθρώπινης σκέψης και συμπεριφοράς, τα οποία υπονοούν έστω και στοιχειώδη ευφυΐα (μάθηση, προσαρμοστικότητα, εξαγωγή συμπερασμάτων, κατανόηση από συμφραζόμενα, επίλυση προβλημάτων κλπ.). Ένα τέτοιο είδος υπολογιστικών συστημάτων είναι και τα Τεχνητά Νευρωνικά Δίκτυα (Neural Networks - NNs), τα οποία αποτελούν μια απλοποιημένη μορφή του ανθρώπινου εγκεφάλου. Ο Τζον Μακάρθι όρισε τον τομέα αυτόν ως «επιστήμη και μεθοδολογία της δημιουργίας νοούντων μηχανών» (McCarthy, 1989).

Η τεχνητή νοημοσύνη αποτελεί σημείο τομής μεταξύ πολλαπλών επιστημών όπως της πληροφορικής, της φιλοσοφίας, της ψυχολογίας, της νευρολογίας, της γλωσσολογίας και της μηχανικής, με στόχο την προσπάθεια δημιουργίας και σύνθεσης ευφυούς συμπεριφοράς, με διάφορα στοιχεία συλλογιστικής μάθησης και προσαρμογής στο περιβάλλον. Η τεχνητή νοημοσύνη εφαρμόζεται συνήθως σε μηχανές ή υπολογιστές ειδικής κατασκευής.

Η τεχνητή νοημοσύνη χωρίζεται σε δύο κατηγορίες, την συμβολική τεχνητή νοημοσύνη και την υποσυμβολική τεχνητή νοημοσύνη. Η συμβολική τεχνητή νοημοσύνη επιχειρεί να εξομοιώσει την ανθρώπινη νοημοσύνη αλγοριθμικά χρησιμοποιώντας σύμβολα και λογικούς κανόνες υψηλού επιπέδου, ενώ η υποσυμβολική τεχνητή νοημοσύνη, προσπαθεί να αναπαράγει την ανθρώπινη ευφυΐα χρησιμοποιώντας στοιχειώδη αριθμητικά μοντέλα που συνθέτουν επαγωγικά νοήμονες συμπεριφορές με τη διαδοχική αυτο-οργάνωση απλούστερων δομικών συστατικών προσομοιώνοντας πραγματικές βιολογικές διαδικασίες όπως η εξέλιξη των ειδών και η λειτουργία του εγκεφάλου («υπολογιστική νοημοσύνη»), ή αποτελούν εφαρμογή στατιστικών μεθοδολογιών σε προβλήματα τεχνητής νοημοσύνης.

Η διάσημη δοκιμασία Turing η οποία προτάθηκε από τον Alan Turing το 1950, σχεδιάστηκε έτσι ώστε να παρέχει ένα λειτουργικό ορισμό της νοημοσύνης. Ένας υπολογιστής θεωρείται ευφυής και περνά την δοκιμασία Turing, εφόσον ένας άνθρωπος ο οποίος είναι ο κριτής δεν μπορεί να ξεχωρίσει αν οι απαντήσεις στις ερωτήσεις που επέβαλε, προήλθαν από ανθρώπινο ον ή από έναν υπολογιστή (Turing, 2009).

3.2 Μηχανική Μάθηση (Machine Learning)

3.2.1 Γενικά

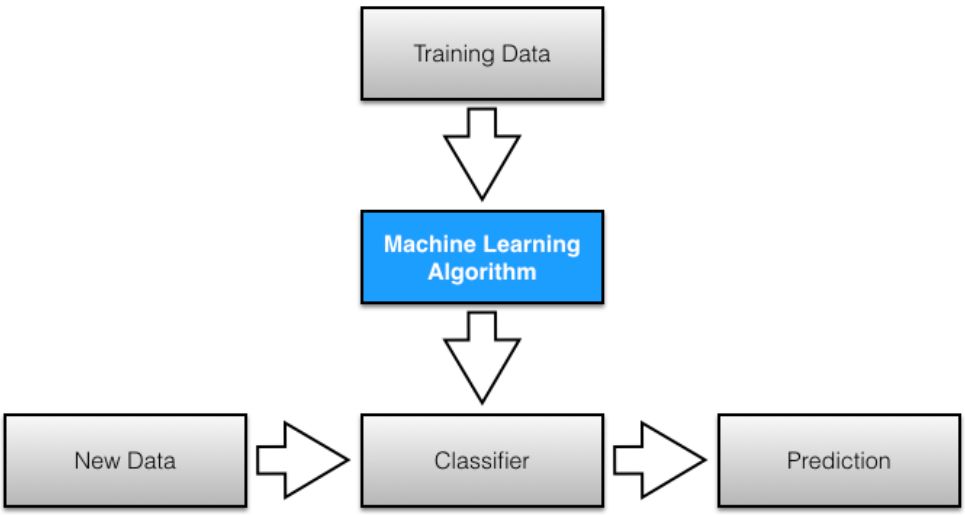
Η μάθηση είναι το σημαντικότερο χαρακτηριστικό για την πρόοδο της ανθρωπότητας. Για να μπορέσει η κοινωνία να προοδεύσει πρέπει οι άνθρωποι που την αποτελούν να μαθαίνουν μέσα από την έρευνα αλλά και τα λάθη τους. Οι διάφορες έρευνες που γίνονται, συμπεριλαμβανομένης και της συγκεκριμένης έρευνας, είναι αποτέλεσμα συνδυασμού αλλά και επέκτασης υφιστάμενων ερευνών και γνώσεων. Η μάθηση είναι το κλειδί για κάθε ερώτηση που μπορεί να υπάρξει. Είναι διαχρονική έννοια αφού η μάθηση συμβαίνει στον άνθρωπο από τη γέννηση μέχρι και το θάνατο του. Ας πάρουμε για παράδειγμα την συγγραφή δοκιμίων. Για να μπορέσει ένας ερευνητής να γράψει ένα δοκίμιο για μια οποιαδήποτε έρευνα θα πρέπει να μάθει το αλφάβητο, τη σύνταξη των προτάσεων καθώς και πολλά άλλα. Η μάθηση μπορεί να ευκολύνει την ζωή του ανθρώπου αν βέβαια χρησιμοποιείται σε ηθικά πλαίσια. Σήμερα, οι διάφορες εταιρείες επενδύουν τεράστια ποσά σε μοντέλα και αλγορίθμους που συλλέγουν και επεξεργάζονται (data analytics) μεγάλα δεδομένα (big data) για να κατασκευάσουν στο τέλος κάποια άλλα, χρήσιμα δεδομένα, τα οποία μπορεί να χρησιμοποιήσει η εταιρεία ή ένας οργανισμός για να «μάθει» πώς πρέπει να κινηθεί στο μέλλον. Αντιλαμβάνεστε λοιπόν τη δύναμη και το αντίκτυπο της μάθησης στην κοινωνία την ανθρωπότητας.

Όπως έχουμε αναφέρει και εξηγήσει στην παρούσα έρευνα, τα τεχνητά νευρωνικά δίκτυα είναι εμπνευσμένα από τον ανθρώπινο εγκέφαλο. Ο εγκέφαλος του ανθρώπου όμως τυγχάνει κάποιας μορφής εκπαίδευσης καθημερινά. Το ίδιο ακριβώς συμβαίνει και με τα τεχνητά νευρωνικά δίκτυα. Η εκπαίδευση που συμβαίνει στα τεχνητά νευρωνικά δίκτυα, είναι ακριβώς μια διαδικασία μάθησης αλλά και εξειδίκευσης ενός νευρωνικού δικτύου για την επίλυση ενός συγκεκριμένου προβλήματος. Τα τεχνητά νευρωνικά δίκτυα λοιπόν, μέσα από την εκπαίδευση τους, προσαρμόζουν τα βάρη του δικτύου ανάλογα, έτσι ώστε να εξειδικευτούν στην επίλυση ενός συγκεκριμένου προβλήματος. Η εκπαίδευση των τεχνητών νευρωνικών δικτύων ανήκει στην κατηγορία της Μηχανικής Μάθησης (Machine Learning - ML).

3.2.2 Είδη Μάθησης

3.2.2.1 Επιβλεπόμενη Μάθηση (Supervised Learning)

Η επιβλεπόμενη μάθηση (Supervised Learning), είναι για την μηχανική μάθηση η προσπάθεια για εύρεση συγκεκριμένης συνάρτησης επίλυσης ενός προβλήματος, από ετικετοποιημένα δεδομένα εκπαίδευσης. Τα δεδομένα εκπαίδευσης αποτελούνται από ένα σύνολο παραδειγμάτων εκπαίδευσης. Στην επιβλεπόμενη μάθηση, κάθε παράδειγμα εισόδου είναι ένα ζεύγος ενός αντικειμένου εισόδου (διάνυσμα εισόδου) και ενός επιθυμητού αποτελέσματος εξόδου (ετικέτα). Ένας αλγόριθμος επιβλεπόμενης μάθησης, μέσα από ανάλυση των δεδομένων εκπαίδευσης, προσπαθεί να εντοπίσει μια συνάρτηση την οποία μπορεί να χρησιμοποιήσει για να κατηγοριοποιήσει νέα παραδείγματα εισόδου. Μετά το πέρας της εκπαίδευσης του δικτύου, ιδανικά, θα είχαμε τον αλγόριθμο επιβλεπόμενης μάθησης να μπορέσει να κατηγοριοποιήσει σωστά δεδομένα εισόδου τα οποία είναι άγνωστα για αυτόν, δηλαδή δεδομένα εισόδου τα οποία δεν έχει «ξαναδεί» στην είσοδο του. Για να γίνει το πιο πάνω, ένας αλγόριθμος επιβλεπόμενης μάθησης θα πρέπει μετά το πέρας της εκπαίδευσης του να είναι σε θέση να γενικεύσει μέσα από τα δεδομένα εκπαίδευσης, για άγνωστα δεδομένα τα οποία δεν έχει ξαναδεί στην είσοδο του έχοντας εντοπίσει μια λογική συνάρτηση και ιδανικά κατηγοριοποιώντας τα άγνωστα δεδομένα εισόδου σωστά.



Σχήμα 3.1: Διάγραμμα μοντέλου επιβλεπόμενης μάθησης (Raschka, 2014).

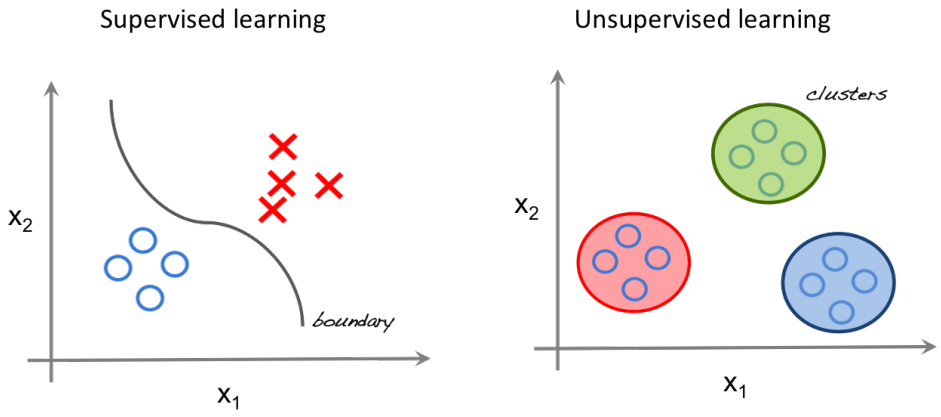
3.2.2.2 Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning)

Πολλές φορές μπορεί να βρεθούμε μπροστά από μεγάλα δεδομένα τα οποία δεν έχουν κατηγοριοποιηθεί. Πιο συγκεκριμένα, μπορεί να έχουμε ένα σύνολο δεδομένων για το οποίο δεν έχουμε τα επιθυμητά αποτελέσματα εξόδου (ετικέτες). Για παράδειγμα, μπορεί να έχουμε κάποια καρκινώματα, για τα οποία θέλουμε να τα ομαδοποιήσουμε σε ομάδες οι οποίες θα έχουν κοινά χαρακτηριστικά έτσι ώστε να είμαστε σε θέση να μελετήσουμε την κάθε ομάδα και να κατασκευάσουμε φαρμακευτικά προϊόντα για την καταπολέμηση τους.

Πώς μπορούμε όμως να βρούμε τα βασικά χαρακτηριστικά ενός συνόλου δεδομένων, έτσι ώστε να μπορέσουμε να διαχωρίσουμε αποδοτικά τα διάφορα παραδείγματα που το αποτελούν σε ομάδες; Πώς μπορούμε αποδοτικά να συμπιέσουμε μεγάλους όγκους δεδομένων και να μειώσουμε τις διαστάσεις ενός προβλήματος; Όλα τα πιο πάνω, είναι κάποια παραδείγματα προβλημάτων τα οποία είναι σε θέση να απαντήσει η μη επιβλεπόμενη μάθηση (Unsupervised Learning), η οποία λέγεται «μη επιβλεπόμενη», λόγω του ότι για κάθε παράδειγμα εισόδου δεν έχουμε την επιθυμητή ετικέτα εξόδου. Οι δύο πιο βασικές εργασίες ενός αλγόριθμου μη επιβλεπόμενης μάθησης είναι η ομαδοποίηση δεδομένων με βάση κοινά (κατά τον αλγόριθμο) χαρακτηριστικά, και η συμπίεση των δεδομένων διατηρώντας την δομή και τα χαρακτηριστικά τους.

Για να ορίσουμε πιο αυστηρά τα πιο πάνω, η μη επιβλεπόμενη μηχανική μάθηση, είναι η μηχανική μάθηση η οποία προσπαθεί να εντοπίσει μια μαθηματική συνάρτηση για να περιγράψει την κρυφή δομή την οποία μπορεί να έχουν κάποια δεδομένα εισόδου για τα οποία δεν έχουμε τα επιθυμητά αποτελέσματα εξόδου. Εφόσον έχουμε παραδείγματα εισόδου χωρίς τα επιθυμητά αποτελέσματα εξόδου, τότε ο «μαθητής» πρέπει κατά κάποιο τρόπο μόνος του να βρει κοινά χαρακτηριστικά και να ομαδοποιήσει τα διάφορα παραδείγματα, κατά έναν τρόπο που νομίζει αυτός σωστό. Αυτή είναι και μια βασική διαφορά της επιβλεπόμενης από τη μη επιβλεπόμενη μάθηση. Ένα από τα πιο σημαντικά προβλήματα που μπορεί να επιλύσει ένας αλγόριθμος μάθησης είναι και η εξαγωγή και εντοπισμός χαρακτηριστικών ‘κλειδιών’ σε ένα σύνολο δεδομένων. Ένας αλγόριθμος μη επιβλεπόμενής μάθησης μπορεί να ομαδοποιήσει διάφορα παραδείγματα με βάση κάποια κοινά

χαρακτηριστικά τα οποία θεωρούνται σημαντικά. Κάποια παραδείγματα δικτύων μη επιβλεπόμενης μάθησης είναι τα δίκτυα Kohonen αλλά και ο αλγόριθμος Hebbian.



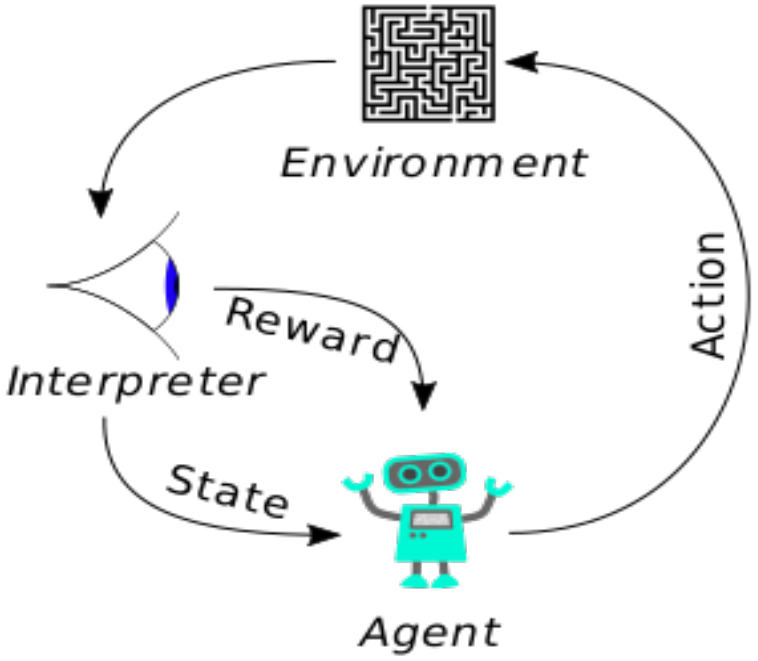
Σχήμα 3.2: Διάγραμμα διαφοράς επιβλεπόμενης και μη επιβλεπόμενη μάθηση (Seif, 2018).

Στο πιο πάνω σχήμα (Σχήμα 3.2), βλέπουμε ότι ο αλγόριθμος επιβλεπόμενης μάθησης εντοπίζει μια συνάρτηση διαχωρισμού των δεδομένων εισόδου μέσα από την εκπαίδευση που πραγματοποιείται, ενώ αντίθετα ο αλγόριθμος μη επιβλεπόμενης μάθησης ομαδοποιεί παραδείγματα με κοινά (κατά τον αλγόριθμο) χαρακτηριστικά, δημιουργώντας τέλος κάποιες συστοιχίες (clusters), με αυτά τα παραδείγματα. Στην συνέχεια για κάθε νέο παράδειγμα ο αλγόριθμος προσπαθεί να το αντιστοιχήσει σε μια (αυτήν που ομοιάζει περισσότερο) από τις ομάδες που δημιούργησε, και να δώσει στην έξοδο του την ετικέτα της ομάδας.

3.2.2.3 Ενισχυτική Μάθηση (Reinforcement Learning)

Η ενισχυτική μάθηση (Reinforcement learning), είναι η περιοχή της μηχανικής μάθησης η οποία είναι εμπνευσμένη από την συμπεριφορική ψυχολογία, και ασχολείται με το πώς ένας υπολογιστικός πράκτορας/αλγόριθμος (agent) αποφασίζει να αναλάβει δράση και να πραγματοποιήσει κάποια ενέργεια, σε κάποιο περιβάλλον, έτσι ώστε να μεγιστοποιήσει (μακροπρόθεσμα ή βραχυπρόθεσμα) κάποιας μορφής αθροιστική ανταμοιβή που θα λαμβάνει για κάθε ενέργεια, από έναν κριτή/διερμηνέα (interpreter). Το πρόβλημα αυτό, λόγω της γενικότητας του, μελετάται και σε πολλούς άλλους κλάδους όπως τη θεωρία παιγνίων, τη θεωρία ελέγχου, τη θεωρία της πληροφορίας, τη βελτιστοποίηση με βάση προσομοιώσεις, τα συστήματα πολλαπλών πρακτόρων, τη στατιστική και τους γενετικούς αλγορίθμους. Στην

ενισχυτική μάθηση δεν υπάρχουν παραδείγματα δεδομένων εκπαίδευσης και επιθυμητά αποτελέσματα, αλλά ο αλγόριθμος/πράκτορας εξακολουθεί να πρέπει να πάρει μια απόφαση για το πώς πρέπει να δράσει. Αφού δεν υπάρχουν δεδομένα εκπαίδευσης ο αλγόριθμος μαθαίνει από την εμπειρία που αποκτά σχετικά με το πρόβλημα. Ο αλγόριθμος δημιουργεί μια δική του λογική στην απόφαση των κινήσεων που διαπράττει, μέσα από την ανατροφοδότηση που πήρε για κάθε μια κίνηση που έκανε (ανατροφοδότηση: αυτή η κίνηση επέφερε θετική επιβράβευση, αυτή η κίνηση επέφερε αρνητική επιβράβευση) από τον κριτή, και αναπτύσσει τέλος μια συμπεριφορά που μεγιστοποιεί την πιθανότητα ανταμοιβής/επιβράβευσης και μειώνει την πιθανότητα τιμωρίας από τον κριτή. Ο απώτερος δηλαδή σκοπός ενός αλγορίθμου ενισχυτικής μάθησης είναι να μεγιστοποιήσει μακροπρόθεσμα ή βραχυπρόθεσμα την (συνολική ή όχι) ανταμοιβή/επιβράβευση που λαμβάνει από τον κριτή.

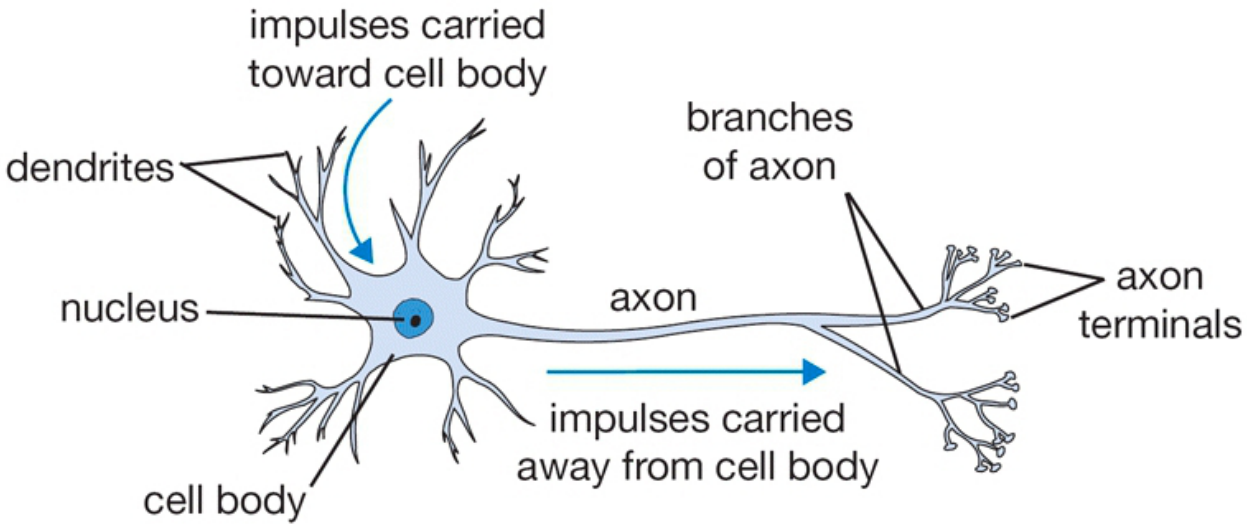


Σχήμα 3.3: Διάγραμμα μοντέλου λειτουργίας ενισχυτικής μάθησης. Στην εικόνα φαίνεται ο κριτής (interpreter) των κινήσεων του πράκτορα/αλγορίθμου (agent), το περιβάλλον (environment) στο οποίο ενεργεί ο πράκτορας, καθώς επίσης και τα κατευθυνόμενα βέλη που δείχνουν την ανταμοιβή (reward) του κριτή στον πράκτορα ανάλογα με την κατάσταση (state) που βρίσκεται ο πράκτορας, και τέλος την ενέργεια (action) που πραγματοποιεί ο πράκτορας στο περιβάλλον που βρίσκεται.

3.3 Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks)

3.3.1 Πηγή Έμπνευσης Τεχνητών Νευρωνικών Δικτύων

Τα τεχνητά νευρωνικά δίκτυα (Artificial neural networks - ANNs), είναι υπολογιστικά συστήματα τα οποία είναι εμπνευσμένα από τα βιολογικά νευρωνικά δίκτυα τα οποία υπάρχουν στον εγκέφαλο των ζώων και των ανθρώπων. Αυτά τα δίκτυα, είναι με απλά λόγια μια απλοποιημένη μορφή του ανθρώπινου εγκεφάλου. Ο όρος «νευρωνικά» προήλθε από τις βασικές λειτουργικές μονάδες του νευρικού συστήματος του ανθρώπου, τους «νευρώνες» ή αλλιώς τα νευρικά κύτταρα, τα οποία βρίσκονται στον ανθρώπινο εγκέφαλο ή και σε άλλα μέρη του ανθρώπινου σώματος. Στον ανθρώπινο εγκέφαλο υπάρχουν 10^{11} νευρώνες, συνδεδεμένοι με 10^4 άλλους νευρώνες.



Σχήμα 3.4: Παράδειγμα βιολογικού νευρώνα (Biological Neuron. Retrieved from http://cs231n.github.io/neural-networks-1/).

Στο σχήμα 3.4 φαίνονται με τόξα, οι κατευθύνσεις των παλμών όταν ένα σήμα έρχεται (αριστερά τόξο), ή φεύγει (δεξιά τόξο), από το νευρώνα. Ο νευρώνας λαμβάνει σήματα από άλλους νευρώνες μέσω των δενδρίτων (dendrites). Το σώμα του νευρώνα (cell body), προσθέτει όλα τα εισερχόμενα σήματα και υπολογίζει την είσοδο στο νευρώνα. Όταν το άθροισμα των εισόδων στο νευρώνα ξεπεράσει μια συγκεκριμένη τιμή κατωφλίου, ο νευρώνας πυροδοτεί, και το σήμα μεταφέρεται μέσω του άξονα (axon) στους άλλους

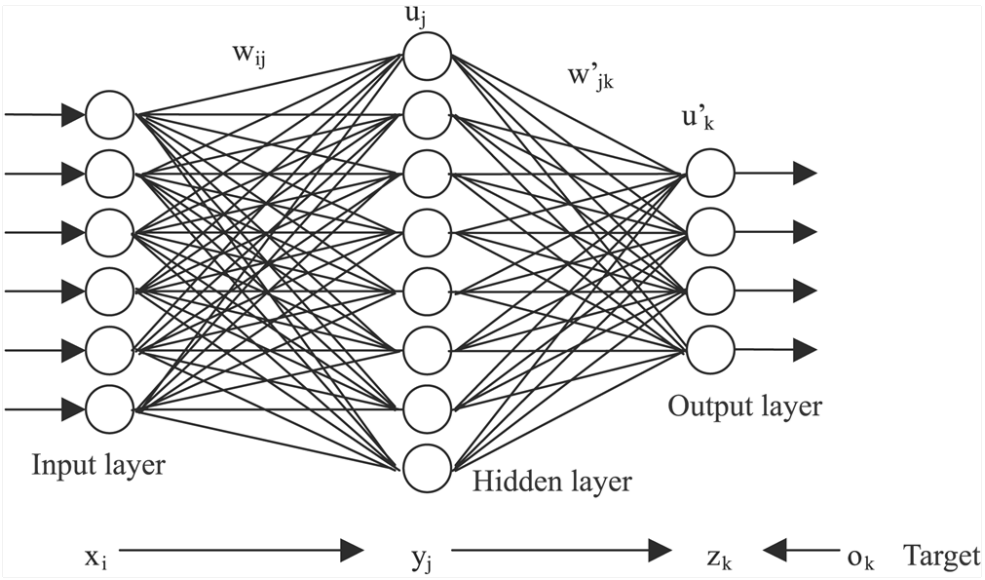
γειτονικούς νευρώνες. Οι συνάψεις (axon terminals), είναι το σημείο διασύνδεσης μεταξύ των νευρώνων του εγκεφάλου. Η δύναμη του σήματος που μεταφέρεται από ένα νευρώνα σε έναν άλλο εξαρτάται από την δύναμη διασύνδεσης (συναπτικό βάρος) των νευρώνων. Οι συνδέσεις μπορεί να είναι συσταλτικές ή ανασταλτικές. Τέλος το νευρωνικό σύστημα του ανθρώπου, είναι ένα εξαιρετικά υψηλής συνδεσιμότητας δίκτυο με τρισεκατομμύρια νευρώνες και δισεκατομμύρια συνδέσεις μεταξύ τους.

Τέτοιου είδους συστήματα, είναι σε θέση να εκπαιδεύονται και κατά συνέπεια να μαθαίνουν (βελτιώνουν την απόδοση τους) σε συγκεκριμένες εργασίες, με το να λαμβάνουν υπόψιν τους κάποια παραδείγματα εισόδου, χωρίς όμως να έχουν προγραμματιστεί συγκεκριμένα για να επιλύουν ένα συγκεκριμένο πρόβλημα. Πιο συγκεκριμένα, τα τεχνητά νευρωνικά δίκτυα, μέσα από τα παραδείγματα που τους δίνονται σαν είσοδο, προσπαθούν να κατηγοριοποιήσουν τα δεδομένα αυτά σε κάποιες ετικέτες εξόδου. Δέχονται τα δεδομένα εισόδου, πραγματοποιούν μια κάποια πρόβλεψη, και στη συνέχει ανάλογα με το αν ήταν λάθος ή σωστή η πρόβλεψη που έκαναν προ ολίγου, ρυθμίζουν κάποιες παραμέτρους τους, τα λεγόμενα βάρη του δικτύου. Για παράδειγμα, στην αναγνώριση εικόνας, τέτοιου είδους συστήματα μπορεί να μάθουν να αναγνωρίζουν εικόνες οι οποίες περιέχουν γάτες, με το να αναλύουν παραδείγματα εικόνων εισόδου στις οποίες έχουν προστεθεί κάποιες ετικέτες ότι περιέχουν ή όχι γάτες, και τέλος χρησιμοποιούν αυτά τα αποτελέσματα και την εκπαίδευση του δικτύου για να μπορέσουν να αναγνωρίσουν αν υπάρχουν ή όχι γάτες σε άλλες εικόνες. Τα συστήματα αυτά, καταφέρνουν να κατηγοριοποιήσουν εικόνες με γάτες ή χωρίς γάτες, χωρίς να έχουν κάποια προηγούμενη γνώση σε σχέση με γάτες (π.χ. ότι έχουν τρίχωμα, ουρά ή νύχια). Αντίθετα, χρησιμοποιούν την δική τους γνώση σε σχέση με τις εικόνες που περιέχουν ή όχι γάτες, και η οποία γνώση αποκτήθηκε κατά την εκπαίδευση του νευρωνικού δικτύου με άλλα παραδείγματα εικόνων εισόδου.

3.3.2 Αρχιτεκτονική και Λειτουργία Τεχνητών Νευρωνικών Δικτύων

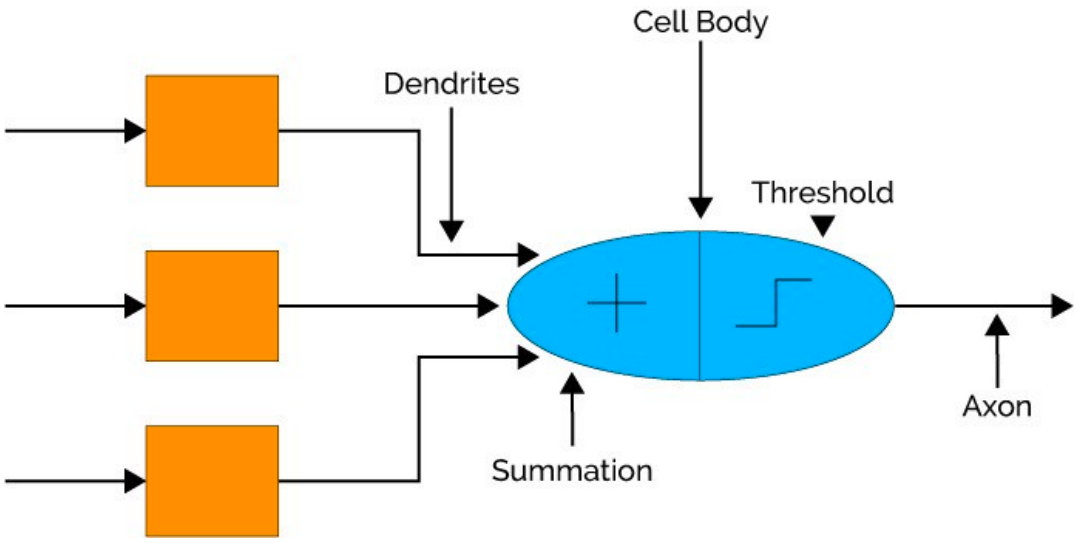
Ένα τεχνητό νευρωνικό δίκτυο, αποτελείται από μια συλλογή από συνδεδεμένες μονάδες ή κόμβους, οι οποίοι ονομάζονται τεχνητοί νευρώνες (αναλογούν με τους βιολογικούς νευρώνες στον εγκέφαλο των ζώων και των ανθρώπων). Κάθε σύνδεση (σύναψη), μεταξύ δύο τεχνητών νευρώνων, μπορεί να μεταφέρει ένα σήμα από τον ένα στον άλλο νευρώνα του δικτύου. Ο τεχνητός νευρώνας που λαμβάνει το σήμα μπορεί να το επεξεργαστεί, και στη συνέχεια να στείλει ένα νέο σήμα στους τεχνητούς νευρώνες που είναι συνδεδεμένοι πάνω του.

Στις περισσότερες υλοποιήσεις των τεχνητών νευρωνικών δικτύων ή νευρωνικών δικτύων όπως θα λέμε αυτά τα συστήματα στη συνέχεια, το σήμα που μεταφέρεται από ένα νευρώνα σε ένα άλλο νευρώνα μέσω μιας σύνδεσης μεταξύ αυτών των δύο νευρώνων είναι ένας πραγματικός αριθμός, όπως επίσης η έξοδος του κάθε νευρώνα του δικτύου, υπολογίζεται με βάση μια μη-γραμμική συνάρτηση πάνω στο άθροισμα των εισόδων του. Οι συνδέσεις μεταξύ των νευρώνων του δικτύου ονομάζονται βάρη και προσαρμόζονται ανάλογα, ενόσω προχωρά η εκπαίδευση του δικτύου. Τα βάρη ενδυναμώνουν ή αποδυναμώνουν την ισχύ του σήματος σε μία σύνδεση μεταξύ δύο νευρώνων. Πολλές φορές οι νευρώνες ενός νευρωνικού δικτύου, έχουν μια τιμή κατωφλίου, η οποία τιμή κατωφλίου πρέπει να ξεπεραστεί, έτσι ώστε ένας τεχνητός νευρώνας να αποστείλει ένα σήμα σε κάποιους άλλους νευρώνες. Τυπικά οι νευρώνες ενός τεχνητού νευρωνικού δικτύου οργανώνονται σε κρυφά επίπεδα, κάθε ένα εκ των οποίων, μπορεί να πραγματοποιεί διαφορετικά είδη διαμόρφωσης των δεδομένων εισόδου. Τα σήματα σε ένα νευρωνικό δίκτυο μεταφέρονται από το πρώτο επίπεδο (επίπεδο εισόδου), στο τελευταίο επίπεδο (επίπεδο εξόδου).



Σχήμα 3.5: Παράδειγμα τεχνητού νευρωνικού δικτύου με ένα κρυφό επίπεδο (one hidden layer). Αριστερά φαίνεται το επίπεδο εισόδου (input layer) με τους νοητούς νευρώνες εισόδου, και δεξιά φαίνεται το επίπεδο εξόδου (output layer). Όπου w_{ij} και w'_{jk}, είναι οι συνδέσεις (βάρη), μεταξύ των νευρώνων του τεχνητού νευρωνικού δικτύου (Templeton, 2015).

Αρχικά, τα νευρωνικά δίκτυα είχαν ως σκοπό να επιλύσουν κάποια προβλήματα, με τον ίδιο τρόπο που θα τα επίλυε και ο ανθρώπινος εγκέφαλος. Στη συνέχεια, οι διάφοροι επιστήμονες που ασχολήθηκαν με το συγκεκριμένο θέμα αντιλήφθηκαν, ότι αυτά τα συστήματα έπρεπε να εξειδικευτούν σε συγκεκριμένους τομείς. Έπρεπε δηλαδή, τα τεχνητά νευρωνικά δίκτυα, να αποκτήσουν «νοητικές» ικανότητες σε πολύ συγκεκριμένες εργασίες, αφού μπορούν να ενσωματώσουν/αφομοιώσουν πολύπλοκα δεδομένα εισόδου για ένα συγκεκριμένο πρόβλημα. Τα νευρωνικά δίκτυα έχουν χρησιμοποιηθεί για να λύσουν μια μεγάλη γκάμα προβλημάτων όπως: υπολογιστική όραση, αναγνώριση ομιλίας, μηχανικές μεταφράσεις, φιλτράρισμα κοινωνικών δικτύων, πρόβλεψη μετοχών στο χρηματιστήριο, να παίζουν διάφορα παιχνίδια, να εντοπίζουν διάφορες ηλεκτρονικές απάτες (ηλεκτρονικό έγκλημα) καθώς επίσης και να πραγματοποιούν ιατρικές διαγνώσεις με τεράστια ακρίβεια σε όσα αναφέρθηκαν αλλά και πολλά άλλα θέματα.



Σχήμα 3.6: Αναλογία ενός τεχνητού νευρώνα με ένα βιολογικό νευρώνα. Οι δενδρίτες (dendrites) στα βιολογικά νευρωνικά δίκτυα, αναλογούν στα διασυνδεδεμένα βάρη από έναν νευρώνα σε έναν άλλο, το σώμα του νευρώνα (cell body) στο βιολογικό νευρώνα, αναλογεί στον τεχνητό κόμβο νευρώνα του τεχνητού νευρωνικού δικτύου, που επίσης αποτελείται δύο βασικά σκέλη, το ένα σκέλος αθροίζει το γινόμενο των εισόδων και των βαρών, και το άλλο σκέλος που είναι η συνάρτηση ενεργοποίησης του τεχνητού νευρώνα. Ο άξονας στο βιολογικό νευρώνα αναλογεί με τη μονάδα εξόδου σε ένα τεχνητό νευρωνικό δίκτυο (Gill, 2017).

Τα νευρωνικά δίκτυα σήμερα είναι πολύ δημοφιλή αφού˙ λόγω της αρχιτεκτονικής τους και γενικότερα του τρόπου λειτουργίας τους, μπορούν να λύσουν προβλήματα τα οποία έχουν εξαιρετικά μεγάλη πολυπλοκότητα και για τα οποία δεν είμαστε σε θέση να καθορίσουμε κανόνες ή μαθηματικές συναρτήσεις που μπορούν να τα επιλύσουν, καθώς επίσης και προβλήματα με τεράστιο όγκο δεδομένων, για τα οποία δεν μπορούμε να βρούμε λύση λόγω ακριβώς αυτού του τεράστιου όγκου πληροφορίας την οποία δεν μπορούμε να διαχειριστούμε. Τα νευρωνικά δίκτυα είναι πολύ ανθεκτικά στα λάθη, αφού το τελικό αποτέλεσμα υπολογίζεται μέσα από την συμβολή όλου γενικά του συστήματος (όλων των νευρώνων), συνεπώς, αν κάποιος νευρώνας καταστραφεί, δεν θα επηρεάσει σε μεγάλο βαθμό την ομαλή λειτουργία και το τελικό αποτέλεσμα του νευρωνικού δικτύου. Θεωρούνται αρκετά γρήγορα συστήματα αφού πραγματοποιούν διάφορες πράξεις σε διαφορετικούς νευρώνες παράλληλα. Επίσης το κόστος για λειτουργία των συγκεκριμένων συστημάτων δεν είναι μεγάλο αφού οι μόνες απαιτήσεις που θα έχει ένα τέτοιο σύστημα, είναι απαιτήσεις όσον

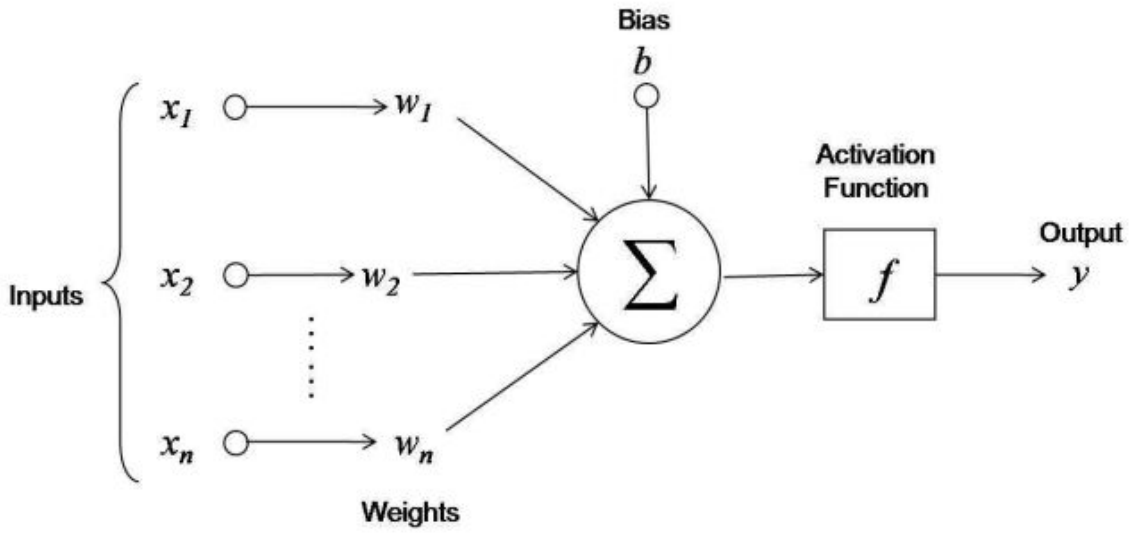
αφορά την προσωρινή μνήμη ενός υπολογιστικού συστήματος στο οποίο θα εγκατασταθεί το τεχνητό νευρωνικό δίκτυο. Ομαδοποιώντας τα όσα λέχθηκαν πιο πάνω, τα τεχνητά νευρωνικά δίκτυα είναι σε θέση να ενσωματώσουν τεράστιο όγκο δεδομένων και πληροφορίας, με το λιγότερο δυνατό κόστος και χωρίς να είναι επιρρεπής σε λάθη ή σφάλματα.

Τα νευρωνικά δίκτυα, μετά το πέρας της εκπαίδευσης τους, και λαμβάνοντας υπόψιν τα διάφορα παραδείγματα εισόδου με τα οποία εκπαιδεύτηκαν, είναι σε θέση να γενικεύσουν αυτά που έμαθαν, και ως αποτέλεσμα να μπορούν να αποφανθούν για νέα δεδομένα εισόδου, τα οποία λαμβάνουν στην είσοδο τους για πρώτη φορά. Η γνώση ενός νευρωνικού δικτύου βρίσκεται αποθηκευμένη, όπως και στον άνθρωπο, στα συναπτικά βάρη του νευρώνα. Ακόμα ένα πλεονέκτημα των νευρωνικών δικτύων, είναι και το γεγονός ότι μπορούν να επιλύσουν διάφορα μη-γραμμικά επιλύσιμα προβλήματα, τα οποία δεν θα μπορούσαν να λυθούν από συστήματα που χρησιμοποιούν γραμμική μοντελοποίηση. Ο λόγος που τα νευρωνικά δίκτυα είναι σε θέση να επιλύσουν τέτοια προβλήματα είναι, εκτός των άλλων, η χρήση μη γραμμικών συναρτήσεων ενεργοποίησης στους νευρώνες τους, καθώς επίσης και η χρήση πολλαπλών κρυφών επιπέδων τα οποία συντείνουν στην επίλυση μη-γραμμικά διαχωρίσιμων προβλημάτων αλλά και προβλημάτων με μεγάλη πολυπλοκότητα.

Λόγω της μεγάλης πολυπλοκότητας των συστημάτων αυτών απαιτείται ένας αριθμός παραμέτρων για ρύθμιση του νευρωνικού δικτύου. Ο μεγάλος αριθμός παραμέτρων, άρα και των συνδυασμών των παραμέτρων ενός νευρωνικού δικτύου, σε συνδυασμό με το κόστος εκπαίδευσης (σε χρόνο), αποτελούν κάποια από τα κύρια μειονεκτήματα αυτών, των κατά τα άλλα, ευφυών συστημάτων.

3.3.3 Μοντέλο Νευρώνα McCulloch και Pitts (1943)

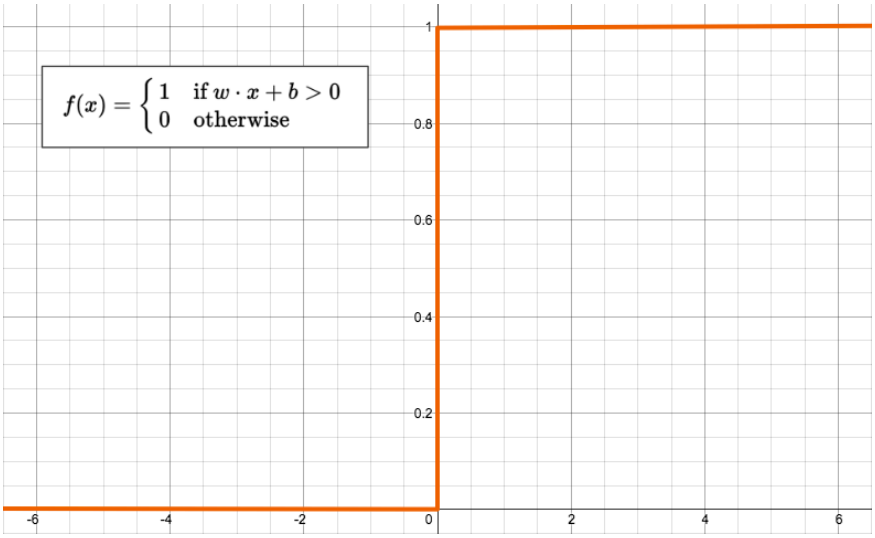
Το πρώτο μοντέλο τεχνητού νευρώνα προτάθηκε από τους McCulloch και Pitts το 1943 (McCulloch and Pitts, 1943). Ο σχεδιασμός του συγκεκριμένου νευρώνα είναι αρκετά απλός και βασισμένος στα όσα έχουμε αναφέρει προηγουμένως. Στην πιο κάτω εικόνα μπορούμε να δούμε τα βασικά μέρη του τεχνητού νευρώνα, καθώς επίσης και μια επεξήγηση της λειτουργίας του κάθε μέρους του τεχνητού νευρώνα ξεχωριστά.



Σχήμα 3.7: Μοντέλο νευρώνα McCulloch and Pitts (1943).

Στην πιο πάνω εικόνα βλέπουμε τα βασικά μέρη και τη λειτουργία ενός τεχνητού νευρώνα. Η είσοδος που θα δοθεί στο νευρώνα μεταφράζεται σε ένα διάνυσμα τιμών και η κάθε τιμή εισόδου αντιστοιχεί σε μια τιμή από τα x1,x2,x3,…,xn. Στην συνέχεια, οι τιμές εισόδου πολλαπλασιάζονται με τα βάρη, τα οποία βάρη προσομοιώνουν/αποθηκεύουν την γνώση του δικτύου για ένα συγκεκριμένο πρόβλημα, και τα οποία χρησιμοποιούνται από το δίκτυο για την επίλυση ενός συγκεκριμένου προβλήματος. Τυπικά, τα βάρη ενός νευρωνικού δικτύου, αρχικοποιούνται με τυχαίες μικρές τιμές (διαφορετικές του μηδενός), αλλά στη συνέχεια αυτές οι τιμές προσαρμόζονται/εκπαιδεύονται ανάλογα με τα παραδείγματα εκπαίδευσης που λαμβάνει το δίκτυο στην είσοδο του, και αντιπροσωπεύουν την δύναμη της διασύνδεσης μεταξύ των νευρώνων μέσα στο νευρωνικό δίκτυο. Κάθε νευρώνας έχει μια επιπλέον είσοδο που είναι πάντα ενεργή (bias). Το πρώτο μέρος του νευρώνα (Σχήμα 3.7 κύκλος με σύμβολο Σ), αθροίζει τα γινόμενα των βαρών και των εισόδων του δικτύου και στη συνέχεια το άθροισμα αυτό, δίνεται σαν είσοδος στη συνάρτηση ενεργοποίησης (activation function f), που θα καθορίσει στην συνέχεια το τελικό αποτέλεσμα εξόδου του νευρώνα (output y). Στο συγκεκριμένο μοντέλο των McCulloch and Pitts, η συνάρτηση ενεργοποίησης είναι η βηματική συνάρτηση ή συνάρτηση σκαλί (Σχήμα 3.8), η οποία δεν μας δίνει αρκετή πληροφορία για το πόσο κοντά ή μακριά είμαστε ως προς την επιθυμητή έξοδο του νευρώνα. Όταν το άθροισμα των γινομένων των βαρών με τις εισόδους του νευρώνα ξεπεράσει μια προκαθορισμένη τιμή

κατωφλίου, τότε, λέμε ότι ο νευρώνας πυροδοτεί θέτοντας ως τιμή εξόδου το 1, διαφορετικά λέμε ότι δεν πυροδοτεί θέτοντας την τιμή εξόδου σε 0.



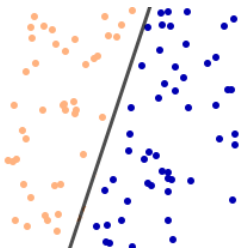
Σχήμα 3.8: Βηματική συνάρτηση ενεργοποίησης (Heaviside Step function).

3.3.4 Πολυστρωματικά Δίκτυα Perceptron Εμπρόσθιου Περάσματος

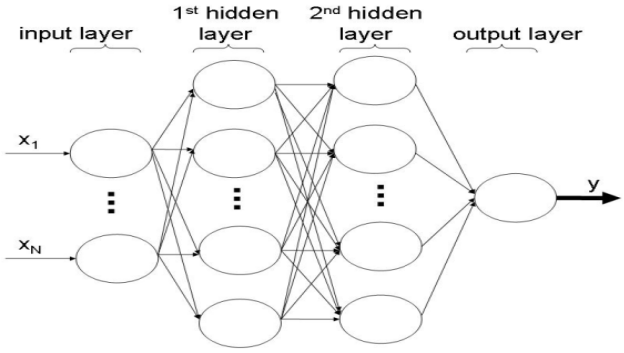
3.3.4.1 Εισαγωγή στα Δίκτυα Perceptron

Ένας νευρώνας McCulloch and Pitts (1943), ο οποίος χρησιμοποιεί σαν συνάρτηση ενεργοποίησης την βηματική συνάρτηση και κάνει χρήση ενός κανόνα μάθησης, λέγεται νευρώνας Perceptron. Ο νευρώνας Perceptron (στα ελληνικά Αντίληπτρο) εφευρέθηκε το 1957 στο Αεροναυτικό Εργαστήριο του Κορνέλλ (Cornell Aeronautical Laboratory) από τον Φρανκ Ρόζενμπλατ (Rosenblatt, 1958). Η δημιουργία ενός νευρωνικού δικτύου με χρήση νευρώνων Perceptron, μπορεί να χαρακτηριστεί ως ένα είδος εμπρόσθια τροφοδοτούμενου (feed-forward) νευρωνικού δικτύου (μπορεί να θεωρηθεί ως ένας γραμμικός ταξινομητής - linear classifier). Με τον όρο εμπρόσθια τροφοδοτούμενο δίκτυο, εννοούμε ότι η φορά μεταφοράς της πληροφορίας για υπολογισμό του τελικού αποτελέσματος εξόδου του δικτύου, είναι μόνο προς τα μπροστά (από το επίπεδο εισόδου στα κρυφά επίπεδα και στη συνέχεια στο επίπεδο εξόδου). Χρησιμοποιώντας και συνδέοντας πολλαπλούς νευρώνες perceptron, μπορούμε να δημιουργήσουμε ένα νευρωνικό δίκτυο πολλαπλών στρωμάτων perceptron. Για τον σχεδιασμό ενός δικτύου πολλαπλών στρωμάτων perceptron, κατατάσσουμε τους νευρώνες σε επίπεδα.

Πιο συγκεκριμένα, όπως αναφέραμε και προηγουμένως, σε μια τέτοια αρχιτεκτονική έχουμε πάντα ένα μη ενεργό επίπεδο εισόδου (χρησιμοποιείται μόνο για τη μεταφορά των δεδομένων), τουλάχιστον ένα κρυφό επίπεδο και ένα επίπεδο εξόδου (ενεργά επίπεδα). Χρησιμοποιώντας ένα μόνο νευρώνα Perceptron, μπορούμε να λύσουμε ένα πρόβλημα το οποίο είναι γραμμικά διαχωρίσιμο.



Σχήμα 3.9: Στο πιο πάνω σχήμα βλέπουμε δύο ομάδες σημείων. Μια ευθεία γραμμή, είναι σε θέση να διαχωρίσει αυτές τις δύο ομάδες σημείων, συνεπώς το πρόβλημα αυτό είναι γραμμικά διαχωρίσιμο. Ένας απλός νευρώνας Perceptron, μπορεί μετά το πέρας της εκπαίδευσης του να κατηγοριοποιήσει ένα σημείο το οποίο δίνεται σαν είσοδος, σε μια από τις δύο ομάδες αυτές.



Σχήμα 3.10: Πολυστρωματικό δίκτυο Perceptron εμπρόσθιου περάσματος, με δύο κρυφά επίπεδα. Στο σχήμα αυτό, φαίνεται ένα πολυστρωματικό νευρωνικό δίκτυο perceptron εμπρόσθιου περάσματος (ομοιάζει με έναν κατευθυνόμενο γράφο), όπου οι κόμβοι αντιστοιχούν στους νευρώνες ενώ οι ακμές αντιστοιχούν στα βάρη/συνδέσεις μεταξύ των νευρώνων (Grzesiak and Zaborski, 2012).

Μια είσοδος η οποία δίνεται σε ένα δίκτυο πολλαπλών στρωμάτων Perceptron, πρέπει να έχει την μορφή ενός διανύσματος έτσι ώστε να μπορεί να γίνει αντιληπτό από ένα νευρωνικό

δίκτυο. Αυτό έχει ως συνέπεια την προεπεξεργασία/προετοιμασία των δεδομένων έτσι ώστε να μεταφραστούν σε διανύσματα. Η προεπεξεργασία των δεδομένων εισόδου είναι ανάλογη του προβλήματος. Τα επεξεργασμένα πλέον δεδομένα, είναι δυνατόν να κανονικοποιηθούν περιορίζοντας τα συνήθως σε διαστήματα όπως (0,1) ή (-1,1), για αποφυγή τυχόν προβλημάτων που έχουν να κάνουν με μεγάλες ή μικρές τιμές στις μεταβλητές του δικτύου (variable overflow, underflow). Τα δεδομένα εισόδου περνούν στην συνέχεια στο πρώτο κρυφό επίπεδο του νευρωνικού δικτύου, όπου υπόκεινται ξανά σε κάποιου είδους επεξεργασία. Στη συνέχεια, τα αποτελέσματα που εξάγονται από τους κρυφούς νευρώνες του πρώτου επιπέδου μπορεί να δοθούν σαν είσοδο σε επόμενα κρυφά επίπεδα ή στο επίπεδο εξόδου (αν υπάρχει μόνο ένα κρυφό επίπεδο). Ο αριθμός των νευρώνων σε κάθε επίπεδο του νευρωνικού δικτύου είναι μεταβλητός, εξαρτάται αποκλείστικα από τη φύση του προβλήματος που θέλουμε να λύσουμε, και αποτελεί μία από τις σημαντικότερες παραμέτρους ρύθμισης ενός νευρωνικού δικτύου. Σημαντικό είναι να αναφέρουμε ότι στα απλά νευρωνικά δίκτυα πολλαπλών στρωμάτων Perceptron, έχουμε πλήρη συνδεσιμότητα των νευρώνων του δικτύου. Με τον όρο πλήρη συνδεσιμότητα εννοούμε ότι ο κάθε νευρώνας σε ένα επίπεδο του δικτύου (έστω x), είναι συνδεδεμένος με όλους τους νευρώνες στο προηγούμενο επίπεδο (x-1). Τέλος, αξίζει να σημειωθεί, ότι η πληροφορία και η γνώση ενός νευρωνικού δικτύου για ένα συγκεκριμένο θέμα βρίσκεται σε τεράστιο βαθμό (αν όχι εξολοκλήρου), στα κρυφά του επίπεδα. Με το πέρας της εκπαίδευσης του νευρωνικού δικτύου, τα βάρη προσαρμόζονται ανάλογα, έτσι ώστε να επιτευχθεί το μεγαλύτερο ποσοστό επιτυχίας στις απαντήσεις του δικτύου, για κάποιες ερωτήσεις για ένα συγκεκριμένο πρόβλημα.

3.3.4.2 Αλγόριθμος Μάθησης Perceptron

Ο αλγόριθμος μάθησης perceptron είναι από τους αρχαιότερους αλγορίθμους επιβλεπόμενης μάθησης/εκπαίδευσης νευρωνικών δικτύων. Ξεκινώντας, ο αλγόριθμος μάθησης perceptron, αρχικοποιεί όλα τα βάρη και κατώφλια του δικτύου με μικρές τυχαίες τιμές. Στη συνέχεια, ο αλγόριθμος τροφοδοτεί το δίκτυο με την είσοδο και την ετικέτα της επιθυμητής εξόδου για το συγκεκριμένο παράδειγμα εισόδου. Στη συνέχεια το νευρωνικό δίκτυο, υπολογίζει την πραγματική έξοδο για την είσοδο που έλαβε προηγουμένως. Ακολούθως, το νευρωνικό δίκτυο προχωρεί στην σύγκριση της πραγματικής εξόδου με την επιθυμητή έξοδο που έλαβε αρχικά.

Αν η πραγματική έξοδος του δικτύου είναι μηδέν (0) ενώ η επιθυμητή έξοδος είναι ένα (1), τότε το δίκτυο χρησιμοποιώντας τη μέθοδο ανάστροφης μετάδοσης του σφάλματος, αλλάζει τις τιμές των συναπτικών βαρών του δικτύου επιβάλλοντας στα βάρη μια μικρή σχετικά αύξηση. Αντίθετα αν η πραγματική έξοδος του δικτύου είναι ένα (1) ενώ η επιθυμητή έξοδος είναι μηδέν (0), τότε το δίκτυο χρησιμοποιώντας και πάλι τη μέθοδο ανάστροφης μετάδοσης του σφάλματος, αλλάζει τις τιμές των συναπτικών βαρών του δικτύου επιβάλλοντας στα βάρη μια μικρή σχετικά μείωση. Τέλος, αν η πραγματική έξοδος του δικτύου είναι η ίδια με την επιθυμητή έξοδο τότε δεν έχουμε καμία αλλαγή στα συναπτικά βάρη του δικτύου.

1. Αρχικοποίηση όλων των βαρών και κατωφλίων με μικρές τυχαίες τιμές.
2. Για κάθε παράδειγμα εκπαίδευσης (έστω j) από το σετ δεδομένων εκπαίδευσης (έστω D) πράξε τα πιο κάτω:
   - 2.1 Υπολόγισε την πραγματική τιμή εξόδου του δικτύου $y(t)$, σύμφωνα με την είσοδο που έλαβες $(x1, x2, x3, \dots, \square n)$ τη χρονική στιγμή t, χρησιμοποιώντας τον Κανόνα (1) για υπολογισμό της εξόδου του κάθε νευρώνα.

     Κανόνας (1): $y(t) = f[w_0(t)x_0(t) + w_1(t)x_1(t) + w_2(t)x_2(t) + \cdots + w_n(t)x_n(t)]$ (όπου $f(x)$ η συνάρτηση ενεργοποίησης του νευρώνα).
   - 2.2 Σύγκρινε την πραγματική έξοδο του δικτύου y(t) με την επιθυμητή έξοδο d(t) που δόθηκε στην αρχή μαζί με τα δεδομένα εισόδου, και προσάρμοσε ανάλογα τα συναπτικά βάρη του δικτύου:
     - 2.2.1 Αν η πραγματική έξοδος $y(t)$ είναι 0 ενώ η επιθυμητή έξοδος είναι 1 τότε: $w_i(t+1) = w_i(t) + n\Delta x_i(t), \text{όπου } \Delta = d(t) - y(t).$
     - 2.2.2 Αν η πραγματική έξοδος $y(t)$ είναι 1 ενώ η επιθυμητή έξοδος είναι 0 τότε: $w_i(t+1) = w_i(t) - n\Delta x_i(t), \text{όπου } \Delta = d(t) - y(t).$
     - 2.2.3 Αν η πραγματική έξοδος $y(t)$ είναι η ίδια με την επιθυμητή έξοδο τότε: $w_i(t+1) = w_i(t).$

   (Όπου $0 \leq \eta \leq 1$ο ρυθμός μάθησης)

Πίνακας 3.1: Αλγόριθμος επιβλεπόμενης μάθησης Perceptron.

3.3.4.3 Κανόνας Δέλτα

Στη μηχανική μάθηση, ο κανόνας δέλτα είναι ένας κανόνας μάθησης βασισμένος στη μέθοδο κατάβασης κλίσης για τη ενημέρωση/αλλαγή των συναπτικών βαρών των εισόδων σε τεχνητούς νευρώνες σε ένα μονοεπίπεδο τεχνητό νευρωνικό δίκτυο. Είναι μια ειδική περίπτωση του πιο γενικού αλγορίθμου ανάστροφης μετάδοσης του σφάλματος. Για ένα νευρώνα έστω j, με συνάρτηση ενεργοποίησης $g(x)$, ο κανόνας δέλτα για το i-οστό συναπτικό βάρος του j-οστού νευρώνα έστω w_{ji}, υπολογίζεται ως εξής:

$$\boldsymbol{\Delta w_{ji} = n(t_j - y_j)g'(h_j)x_i}$$

όπου το n είναι ο ρυθμός μάθησης, το $g'(x)$ είναι η συνάρτηση ενεργοποίησης του νευρώνα, το t_j είναι το επιθυμητό αποτέλεσμα, το h_j είναι το άθροισμα των γινομένων των βαρών και των αντίστοιχων εισόδων, το y_j είναι η πραγματική έξοδος του νευρώνα και το x_i η i-οστή είσοδος του νευρώνα. Η αδυναμία του κανόνα να εφαρμοστεί σε νευρωνικά δίκτυα με περισσότερα από ένα κρυφά επίπεδα περιόριζε την άνθιση της έρευνας στα τεχνητά νευρωνικά δίκτυα για πολλά χρόνια. Η λύση στο συγκεκριμένο πρόβλημα, δόθηκε με την ανακάλυψη και εφαρμογή του αλγορίθμου ανάστροφης μετάδοσης του σφάλματος (Leung and Haykin, 1991).

Ο κανόνας δέλτα συχνά αναγράφεται στην απλοποιημένη του μορφή για ένα νευρώνα με μια γραμμική συνάρτηση ενεργοποίησης ως: $\boldsymbol{\Delta w_{ji} = n(t_j - y_j)x_i}$. Ο κανόνας δέλτα αν και είναι πανομοιότυπος με τον αλγόριθμο μάθησης perceptron, διαφέρουν σε ένα κύριο σημείο το οποίο είναι η πρώτη παράγωγος. Ο νευρώνας perceptron χρησιμοποιεί την βηματική συνάρτηση ως συνάρτηση ενεργοποίησης και αυτό σημαίνει ότι στο σημείο 0 στον άξονα x δεν ορίζεται η πρώτη παράγωγος και οπουδήποτε αλλού στον άξονα των x είναι 0, κάτι που καθιστά την άμεση εφαρμογή του κανόνα δέλτα αδύνατη.

3.3.4.4 Μέθοδος Κατάβασης Κλίσης

Η μέθοδος κατάβασης κλίσης είναι ένας επαναληπτικός αλγόριθμος βελτιστοποίησης πρώτης τάξης, για εύρεση του ολικού ελαχίστου μιας συνάρτησης. Για να μπορέσουμε να προσεγγίσουμε ένα τοπικό ελάχιστο μιας συνάρτησης, χρησιμοποιώντας τη μέθοδο

κατάβασης κλίσης, θα πρέπει να μεταβάλουμε την συνάρτηση ως προς το αρνητικό της πρώτης παραγώγου της συνάρτησης σε ένα συγκεκριμένο σημείο. Αν αντίθετα μεταβάλουμε τη συνάρτηση ως προς το θετικό της πρώτης παραγώγου της συνάρτησης, τότε σημαίνει ότι προσπαθούμε να προσεγγίσουμε ένα τοπικό μέγιστο της συγκεκριμένης συνάρτησης.

Η μέθοδος κατάβασης κλίσης στα νευρωνικά δίκτυα είναι μια μέθοδος επιβλεπόμενης μάθησης η οποία προσπαθεί να προκαλέσει αλλαγές στα συναπτικά βάρη του νευρωνικού δικτύου, έτσι ώστε να καταφέρει να εντοπίσει το ολικό ελάχιστο της συνάρτησης την οποία μελετά. Η κεντρική ιδέα της μεθόδου κατάβασης κλίσης, είναι και το ότι οι αλλαγές που γίνονται στα συναπτικά βάρη του δικτύου είναι ανάλογες του αρνητικού, της πρώτης παραγώγου της συναρτήσεως του σφάλματος, ως προς τα βάρη. Υπολογίζουμε συνεπώς το σφάλμα του δικτύου χρησιμοποιώντας συνήθως την συνάρτηση σφάλματος «Mean square error», και προχωρούμε με το να υπολογίσουμε την πρώτη παράγωγο της συνάρτησης του σφάλματος ως προς τα βάρη. Τέλος, τα βάρη του δικτύου αλλάζουν ανάλογα ως προς το αρνητικό της πρώτης παραγώγου της συναρτήσεως του σφάλματος ως προς τα βάρη (Ενότητα 3.3.4.3).

$$\Delta w_{ij} = -n\frac{dE}{dw_{ij}}$$

Εξίσωση 3.1: Οι αλλαγές που πραγματοποιούνται στα βάρη (Δw_{ij}) είναι ανάλογες, ως προς το αρνητικό της παραγώγου, της συναρτήσεως του σφάλματος, ως προς τα βάρη. Όπου n ο ρυθμός μάθησης, E το σφάλμα εκμάθησης, και w_{ij} το βάρος από το νευρώνα i στον j.

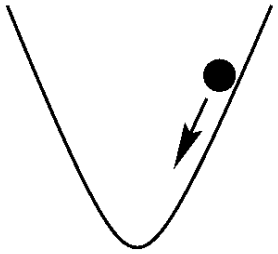
$$\text{Mean Square Error} - \text{MSE} = \frac{1}{n}\sum_{k=1}^{n}(t_k - o_k)^2$$

Εξίσωση 3.2: Εξίσωση υπολογισμού του μέσου σφάλματος. Όπου n το σύνολο των νευρώνων εξόδου, t_k η επιθυμητή έξοδος του νευρώνα k και o_k η πραγματική έξοδος του νευρώνα k.

3.3.4.5 Παράγοντας Ορμής

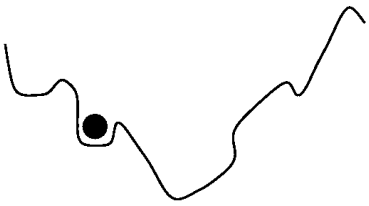
Στα τεχνητά νευρωνικά δίκτυα, χρησιμοποιούμε τη μέθοδο κατάβασης κλίσης έτσι ώστε να ελαχιστοποιήσουμε το αποτέλεσμα της συναρτήσεως του σφάλματος και με απώτερο σκοπό

να φτάσουμε στο ολικό ελάχιστο. Σε έναν ιδανικό κόσμο, η επιφάνεια της συνάρτησης σφάλματος θα μοιάζει όπως πιο κάτω.



Σχήμα 3.11: Παράδειγμα ιδανικής περίπτωσης επιφάνειας συνάρτησης σφάλματος.

Στην πιο πάνω εικόνα, είναι εγγυημένο ότι θα καταφέρουμε να εντοπίσουμε το ολικό ελάχιστο της συνάρτησης και ο λόγος είναι ότι δεν υπάρχουν τοπικά ελάχιστα, στα οποία η συνάρτηση μπορεί να «πέσει μέσα» και να «κολλήσει». Παρόλα αυτά, στην πραγματικότητα, η επιφάνεια σφάλματος είναι πολύ πιο πολύπλοκη, μπορεί να αποτελείται από πολλά τοπικά ελάχιστα και μπορεί να μοιάζει όπως πιο κάτω.



Σχήμα 3.12: Παράδειγμα επιφάνειας πραγματικής συνάρτησης σφάλματος.

Στην συγκεκριμένη περίπτωση του πιο πάνω σχήματος (Σχήμα 3.12), μπορούμε εύκολα να «κολλήσουμε» σε ένα τοπικό ελάχιστο, με αποτέλεσμα ο αλγόριθμος κατάβασης κλίσης να νομίσει ότι φτάσαμε στο ολικό ελάχιστο της συνάρτησης και να έχουμε έτσι μη βελτιστοποιημένα αποτελέσματα. Για να αποφύγουμε αυτή τη κατάσταση, χρησιμοποιούμε μια επιπρόσθετη παράμετρο την οποία λαμβάνουμε υπόψιν κατά την μέθοδο κατάβασης κλίσης και η οποία ονομάζεται ορμή. Η ορμή παίρνει τιμές μεταξύ του μηδέν (0) και του ένα (1), και αυξάνει το μέγεθος των βημάτων που πραγματοποιούνται για εξεύρεση του ολικού ελαχίστου, με αποτέλεσμα να μπορούμε έτσι να «αποδράσουμε» από ένα τοπικό ελάχιστο.

Αν η ορμή είναι μεγάλη (τείνει προς το 1), τότε ο ρυθμός μάθησης πρέπει να διατηρείται μικρότερος. Επιπρόσθετα, αν η ορμή είναι μεγάλη τότε το νευρωνικό δίκτυο αναμένεται να συγκλίνει γρηγορότερα. Αν όμως έχουμε μεγάλη ορμή και μεγάλο ρυθμό μάθησης, τότε μπορεί να ξεφύγουμε από το ολικό ελάχιστο λόγω των μεγάλων βημάτων που θα πραγματοποιούμε. Θέτοντας μια μικρή τιμή στην ορμή, δεν θα μπορούμε αξιόπιστα να αποφύγουμε ένα τοπικό ελάχιστο και θα επιβραδύνουμε επίσης την εκπαίδευση του νευρωνικού δικτύου. Η ορμή συντείνει επίσης στο να ομαλοποιούνται οι αλλαγές των βαρών, αν έχουμε απότομες αλλαγές στην κατεύθυνση στης κλίσης της συνάρτησης του σφάλματος ως προς τα βάρη.

3.3.4.6 Αλγόριθμος Ανάστροφης Μετάδοσης Σφάλματος

Ο αλγόριθμος ανάστροφης μετάδοσης σφάλματος (backpropagation algorithm), είναι μια μέθοδος η οποία χρησιμοποιείται στα τεχνητά νευρωνικά δίκτυα, για να υπολογιστεί μια τιμή (παράγωγος) η οποία χρειάζεται για τον υπολογισμό της αλλαγής που θα εφαρμοστεί στα συναπτικά βάρη του νευρωνικού δικτύου κατά την διάρκεια της εκπαίδευσης. Χρησιμοποιείται συνήθως για την εκπαίδευση βαθιών νευρωνικών δικτύων. Ο όρος «βαθιά νευρωνικά δίκτυα» αναφέρεται στα τεχνητά νευρωνικά δίκτυα με περισσότερα από ένα κρυφά επίπεδα.

Για τη μηχανική μάθηση, ο αλγόριθμος ανάστροφης μετάδοσης σφάλματος χρησιμοποιείται συνήθως από την μέθοδο κατάβασης κλίσης, για να προσαρμόσει τα συναπτικά βάρη των νευρώνων του νευρωνικού δικτύου, με τον υπολογισμό και χρήση της πρώτης παραγώγου (συνήθως) της συναρτήσεως του σφάλματος. Ο αλγόριθμος αυτός, λέγεται αλλιώς και «προς τα πίσω μετάδοση του σφάλματος», και ο λόγος είναι ότι το σφάλμα, υπολογίζεται στο επίπεδο εξόδου και μεταφέρεται προς τα πίσω επίπεδα του τεχνητού νευρωνικού δικτύου.

Ο αλγόριθμος ανάστροφης μετάδοσης σφάλματος θεωρείται αλγόριθμος επιβλεπόμενης μάθησης, αφού απαιτεί ως δεδομένα, ένα επιθυμητό αποτέλεσμα εξόδου για κάθε διάνυσμα εισόδου. Επιπρόσθετα, ο αλγόριθμος αυτός, είναι μια γενίκευση του κανόνα δέλτα (Ενότητα 3.3.4.3) στα πολλαπλών επιπέδων εμπρόσθια τροφοδοτούμενα τεχνητά νευρωνικά δίκτυα, κάτι το οποίο κατέστη εφικτό με τη χρήση του κανόνα αλυσίδας (από το τελευταίο επίπεδο προς το πρώτο επίπεδο του δικτύου), ο οποίος υπολογίζει επαναληπτικά τις παραγώγους της

συναρτήσεως του σφάλματος των νευρώνων του κάθε επιπέδου του δικτύου, για διόρθωση των συναπτικών βαρών τους. Τέλος, ο αλγόριθμος ανάστροφης μετάδοσης του σφάλματος, μπορεί να χρησιμοποιηθεί με οποιονδήποτε αλγόριθμο βελτιστοποίησης που είναι βασισμένος σε παραγώγους συναρτήσεων σφάλματος.

$$\delta_j = y_j(1 - y_j)(y_j - t_j)$$

Εξίσωση 3.3: Εξίσωση υπολογισμού του σφάλματος δ στο επίπεδο εξόδου, για τον νευρώνα j.

$$\delta_j = y_j(1 - y_j)\sum_{i=1}^{N}(w_{ji} \times \delta_i)$$

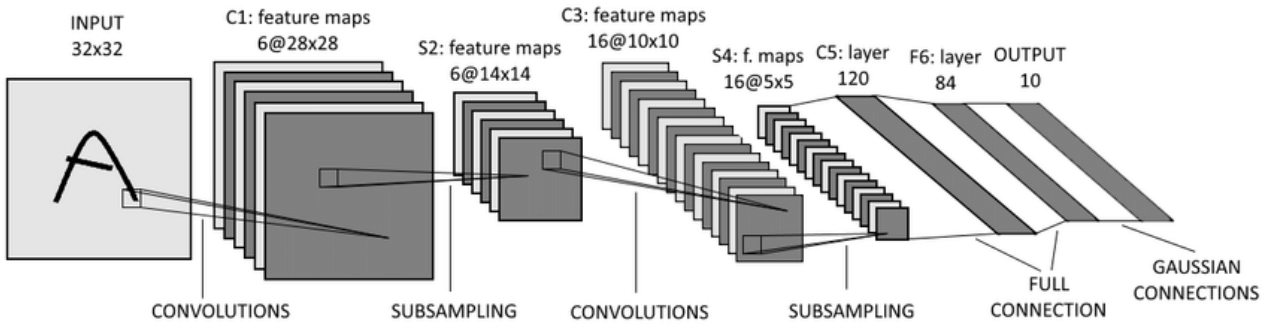
Εξίσωση 3.4: Η εξίσωση υπολογισμού του σφάλματος δ στα κρυφά επίπεδα, για κάθε νευρώνα j. Όπου N το σύνολο των νευρώνων στο επόμενο επίπεδο με τους οποίους είναι συνδεδεμένος ο νευρώνας j και δ_i υπολογίζεται από την εξίσωση 3.3 για το επίπεδο εξόδου.

Μπορούμε να διακρίνουμε δύο κατηγορίες ως προς την συχνότητα εφαρμογής του αλγορίθμου ανάστροφης μετάδοσης του σφάλματος. Η πρώτη κατηγορία φέρει το όνομα on-line ενημέρωση και η άλλη off-line ενημέρωση. Για την πρώτη κατηγορία, η ενημέρωση των βαρών (άρα και η χρήση του backpropagation algorithm), προκύπτει κάθε φορά που παρουσιάζουμε ένα νέο παράδειγμα εισόδου στο νευρωνικό δίκτυο και αφού το δίκτυο πραγματοποιήσει την πρόβλεψη του για το συγκεκριμένο παράδειγμα εισόδου. Για την δεύτερη κατηγορία η ενημέρωση των βαρών του δικτύου (άρα και η χρήση του backpropagation algorithm), γίνεται μετά το πέρας της παρουσίασης ενός καθορισμένου αριθμού παραδειγμάτων (batch), και με βάση τον μέσο όρο του σφάλματος για τα συγκεκριμένα παραδείγματα. Η on-line κατηγορία είναι πιο χρονοβόρα σε σχέση με την off-line κατηγορία, αλλά συγκλίνει γρηγορότερα. Αντίθετα, οι αλλαγές που γίνονται στα βάρη στην off-line κατηγορία, είναι πιο ακριβής και λαμβάνουν υπόψιν όλα τα παραδείγματα του batch, συγκεντρώνοντας έτσι περισσότερη πληροφορία από τα παραδείγματα εισόδου.

3.3.5 Συνελικτικά Νευρωνικά Δίκτυα

3.3.5.1 Εισαγωγή στα Συνελικτικά Νευρωνικά Δίκτυα

Τα συνελικτικά νευρωνικά δίκτυα είναι μια εξέλιξη των τυπικών νευρωνικών δικτύων (Brownlee, 2016). Ένα συνελικτικό νευρωνικό δίκτυο (Convolutional Neural Network - CNN) αποτελείται από ένα ή περισσότερα συνελικτικά στρώματα (συχνά με ένα βήμα υποδειγματοληψίας – subsampling step) και στη συνέχεια ακολουθείται από ένα ή περισσότερα πλήρως συνδεδεμένα στρώματα όπως σε ένα τυπικό νευρωνικό δίκτυο πολλαπλών στρωμάτων. Η αρχιτεκτονική ενός CNN έχει σχεδιαστεί έτσι ώστε να μπορεί να εκμεταλλεύεται τη τρισδιάστατη δομή (3D) μίας εικόνας εισόδου (ή άλλης δισδιάστατης (2D) εισόδου όπως ένα σήμα ομιλίας). Αυτό επιτυγχάνεται με τη χρήση συνελικτικών επιπέδων (Concvolutional Layers - CLs) ακολουθούμενα από κάποια μορφή συγκέντρωσης (pooling) που έχει ως αποτέλεσμα την μετάφραση συγκεκριμένων αμετάβλητων χαρακτηριστικών. Ένα άλλο πλεονέκτημα των CNNs είναι ότι έχουν πολύ πιο αραιή συνδεσιμότητα (λιγότερα συναπτικά βάρη) μεταξύ των νευρώνων του δικτύου σε σχέση με τα πλήρως συνδεδεμένα δίκτυα εμπρόσθιου περάσματος με τον ίδιο αριθμό κρυφών νευρώνων. Πιο συγκεκριμένα, ένας νευρώνας σε ένα κρυφό CL, είναι συνδεδεμένος μόνο με μια μικρή περιοχή των νευρώνων του προηγούμενου επιπέδου. Επιπρόσθετα, ένα από τα σημαντικότερα (αν όχι το πιο σημαντικό) εργαλεία που διαθέτει ένα συνελικτικό νευρωνικό δίκτυο, είναι και η δυνατότητα συνέλιξης/περιελιγμού (convolution) που διαθέτουν, η οποία είναι χρήσιμη όπως αναφέραμε στην επεξεργασία εικόνας. Η δυνατότητα του convolution στα CLs επικεντρώνει, κατά κάποιο τρόπο, την προσοχή του νευρωνικού δικτύου στο ουσιαστικό μέρος των δεδομένων εισόδου, προσανατολίζοντας και κατευθύνοντας με αυτόν τον τρόπο το δίκτυο, έτσι ώστε να μπορέσει να εκπαιδευτεί στο πραγματικό πρόβλημα που αντιμετωπίζουμε.



Σχήμα 3.13: Η αρχιτεκτονική του Le-Net5, ενός συνελικτικού νευρωνικού δικτύου, για την αναγνώριση χαρακτήρων (LeCun et al., 1999).

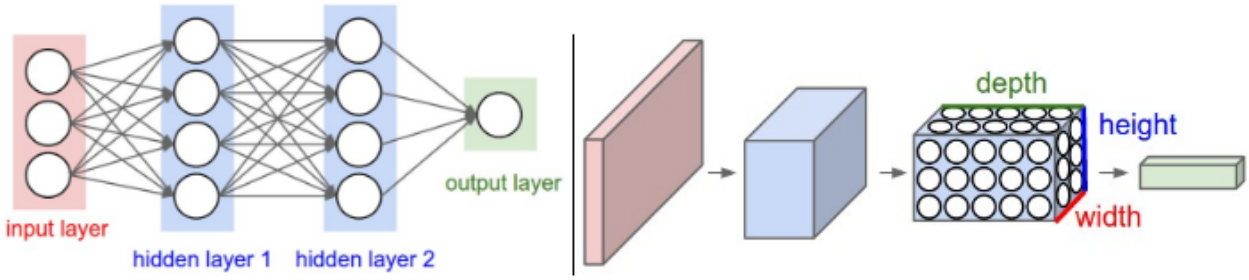
Για παράδειγμα, γνωρίζουμε ότι οι εικόνες περιέχουν πολλές περιττές πληροφορίες και ότι για την αναγνώριση αντικειμένων, οι ακμές (edges) είναι συχνά η σημαντικότερη πληροφορία η οποία χρειαζόμαστε. Συνεπώς, χρησιμοποιούμε αυτή τη γνώση για να δώσουμε στο νευρωνικό μας δίκτυο έναν απλό τρόπο ανίχνευσης ακμών, μέσω του μηχανισμού του convolution στα CLs. Αυτός είναι ο κύριος λόγος για τον οποίο χρησιμοποιούμε συνελικτικά νευρωνικά δίκτυα για τις περισσότερες εργασίες εκμάθησης υπολογιστικής όρασης μέσω βαθιών νευρωνικών δικτύων, αφού χρησιμοποιεί τοπικά πρότυπα των δεδομένων και γενικά ο σχεδιασμός της αρχιτεκτονικής του είναι βασισμένος στην επίλυση τέτοιου είδους προβλημάτων.

3.3.5.2 Αρχιτεκτονική Συνελικτικών Νευρωνικών Δικτύων

Τα συνελικτικά νευρωνικά δίκτυα (CNNs), είναι πολύ όμοια με τα τεχνητά νευρωνικά δίκτυα εμπρόσθιου περάσματος. Αποτελούνται από νευρώνες οι οποίοι έχουν συναπτικά βάρη και κατώφλια τα οποία αλλάζουν/προσαρμόζονται κατά την εκπαίδευση. Ο κάθε νευρώνας λαμβάνει μια είσοδο, υπολογίζει το dot product (γινόμενο αντιστοιχίας) και προαιρετικά, εισάγει το αποτέλεσμα αυτό σε μια μη-γραμμική συνάρτηση ενεργοποίησης. Εξακολουθούν επίσης να υπάρχουν οι συναρτήσεις υπολογισμού του σφάλματος στο τελευταίο επίπεδο εξόδου, και όλες οι τεχνικές για εκπαίδευση ενός τεχνητού νευρωνικού δικτύου που μάθαμε προηγουμένως εξακολουθούν να είναι εφαρμόσιμες.

Όπως είδαμε και στην Ενότητα 3.3.2, τα «απλά» τεχνητά νευρωνικά δίκτυα, λαμβάνουν μια είσοδο (ένα διάνυσμα εισόδου) και το μετατρέπουν, περνώντας το μέσα από μια σειρά κρυφών επιπέδων (διαφορικών εξισώσεων) του δικτύου. Κάθε επίπεδο του νευρωνικού

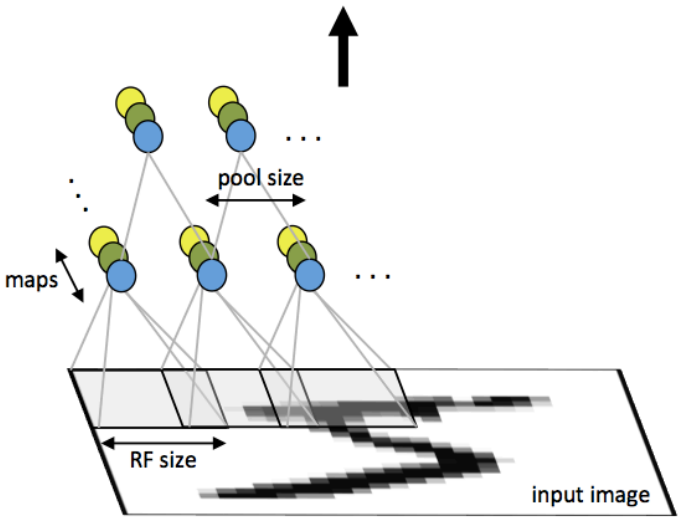
δικτύου, αποτελείται από ένα σύνολο νευρώνων, όπου ο κάθε νευρώνας είναι πλήρως συνδεδεμένος με όλους τους νευρώνες του προηγούμενου επιπέδου. Οι νευρώνες στο ίδιο επίπεδο ενός νευρωνικού δικτύου, δρούν εντελώς ανεξάρτητα ο ένας από τον άλλον και δεν υπάρχει κανένα συναπτικό βάρος που να τους συνδέει. Το τελευταίο επίπεδο του δικτύου ονομάζεται επίπεδο εξόδου, και σε περίπτωση προβλήματος κατηγοριοποίησης επιδεικνύει τα σκορ των κατηγοριών αποτελεσμάτων εξόδου.



Σχήμα 3.14: Αριστερά – Κανονικό νευρωνικό δίκτυο με τρία ενεργά επίπεδα. Δεξιά – Συνελικτικό νευρωνικό δίκτυο (Karpathy, 2016).

Ένα συνελικτικό νευρωνικό δίκτυο, αποτελείται από έναν αριθμό συνελικτικών επιπέδων και υποδειγματοληπτικών επιπέδων (Pooling Layers - PLs), που προαιρετικά ακολουθούνται από πλήρως συνδεδεμένα επίπεδα τυπικών νευρωνικών δικτύων (Σχήμα 3.13). Η είσοδος σε ένα συνελικτικό επίπεδο είναι μια εικόνα μεγέθους $m \times m \times r$, όπου το m είναι το ύψος και το πλάτος της εισόδου και το r είναι ο αριθμός των καναλιών της εισόδου, π.χ. μια εικόνα Red Green Blue (RGB) έχει $r = 3$. Ένα συνελικτικό επίπεδο έχει k φίλτρα (ή kernels) μεγέθους $n \times n \times q$, όπου το n είναι μικρότερο από τις διαστάσεις της εικόνας και το q μπορεί να είναι είτε ίδιο με τον αριθμό καναλιών r ή μικρότερο, και επιπρόσθετα μπορεί να διαφέρει για κάθε kernel. Τα kernels γίνονται convolve με την είσοδο του CNN, κάτι που μας οδηγεί στην δημιουργία k χαρτών χαρακτηριστικών (feature maps) μεγέθους $m - n + 1$. Στη συνέχεια, και πάντα προαιρετικά, σε κάθε feature map πραγματοποιείται υποδειγματοληψία, τυπικά με μέση ή μέγιστη συγκέντρωση πάνω από ίδιες περιοχές στα feature maps μεγέθους $p \times p$, όπου το p κυμαίνεται μεταξύ 2 για μικρού μεγέθους εισόδους και συνήθως δεν υπερβαίνει το 5 για μεγαλύτερου μεγέθους εισόδους. Είτε πριν, είτε μετά από το επίπεδο υποδειγματοληψίας, εφαρμόζεται μια προκατειλημμένη προσθήκη (bias) και μια μη-γραμμική συνάρτηση σε κάθε χάρτη χαρακτηριστικών (feature map). Το παρακάτω σχήμα απεικονίζει

ένα πλήρες στρώμα σε ένα CNN που αποτελείται από συνελικτικά επίπεδα και επίπεδα υποδειγματοληψίας. Μονάδες του ίδιου χρώματος έχουν μεταξύ τους συνδεμένα βάρη, ενώ μονάδες διαφορετικού χρώματος όχι.



Σχήμα 3.15: Ένα πλήρες επίπεδο ενός συνελικτικού νευρωνικού δικτύου (Karpathy, 2016).

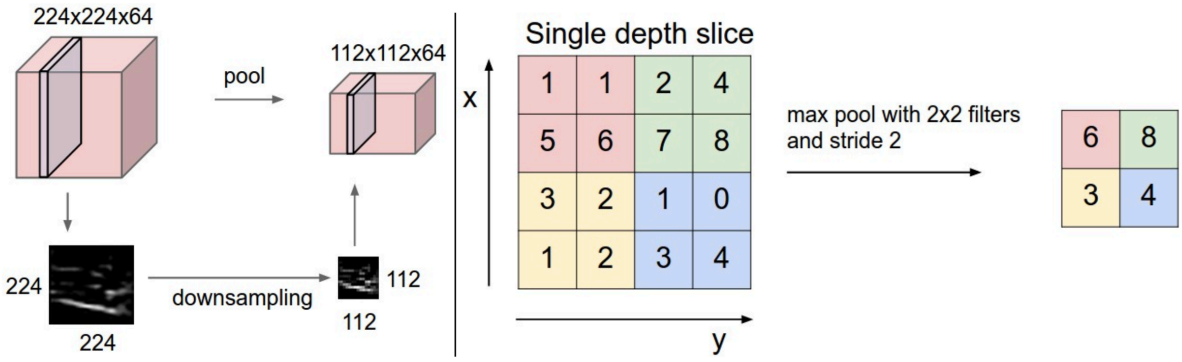
Στο πιο πάνω σχήμα, βλέπουμε ένα πλήρες επίπεδο ενός συνελικτικού νευρωνικού δικτύου με χρήση στρώματος υποδειγματοληψίας (pooling layer). Οι νευρώνες με τα ίδια χρώματα ενώνονται μεταξύ τους με βάρη, και οι νευρώνες με διαφορετικά χρώματα αναπαριστούν διαφορετικούς χάρτες χαρακτηριστικών (feature maps).

Στο πιο πάνω σχήμα (Σχήμα 3.15), φαίνεται η χρήση ενός στρώματος υποδειγματοληψίας ή αλλιώς Pooling Layer (PL). Τα pooling layers είναι ένα από τα βασικά στοιχεία της αρχιτεκτονικής ενός συνελικτικού νευρωνικού δικτύου.

Γενικά, είναι συνηθισμένη τακτική, να εισάγουμε ένα pooling layer μεταξύ διαδοχικών κρυφών συνελικτικών επιπέδων, σε μια κλασσική αρχιτεκτονική συνελικτικών νευρωνικών δικτύων. Ο κύριος λόγος για την χρήση pooling layers, είναι για να μειώσουμε: το μέγεθος της αναπαράστασης των αποτελεσμάτων εξόδου, τον αριθμό των παραμέτρων του νευρωνικού δικτύου, την πολυπλοκότητα του δικτύου, καθώς επίσης και τον συνολικό χρόνο υπολογισμού στο νευρωνικό δίκτυο, αφού μειώνει σημαντικά την ποσότητα των δεδομένων που πρέπει να επεξεργαστεί το δίκτυο. Πράττοντας τα πιο πάνω αποτρέπουμε το δίκτυο από το να

υπερεκπαιδευτεί (προκύπτει όταν το σφάλμα εκπαίδευσης μειώνεται ενώ αντίθετα το σφάλμα επαλήθευσης αυξάνεται - overfitting). Τα pooling layers λειτουργούν ανεξάρτητα από τα άλλα επίπεδα και επεξεργάζονται κάθε αποτέλεσμα εξόδου από κάθε φίλτρο (kernel) που εφαρμόστηκε ξεχωριστά. Η πιο συνήθης μορφή αυτού του είδους επιπέδων, είναι ένα pooling layer με φίλτρα μεγέθους 2×2 και με stride 2,2. Η πιο συνήθης μέθοδος για downsampling που εφαρμόζεται στα περισσότερα pooling layers, είναι η MAX μέθοδος, η οποία παίρνει το μεγαλύτερο όρο από τους όρους που εξετάζει (π.χ. για είσοδο μεγέθους $2 \times 2 = 4$, παίρνει τον μεγαλύτερο από τους 4 όρους), σε κάθε φιλτραρισμένο αποτέλεσμα εξόδου.

Εχτός από τα pooling layers που εφαρμόζουν την MAX μέθοδο (max pooling), τα pooling units μπορούν να χρησιμοποιήσουν και άλλες συναρτήσεις όπως average pooling ή και L2-normalization pooling. Παρόλα αυτά, η τεχνική max pooling φαίνεται να δουλεύει καλύτερα από τις άλλες μεθόδους (Boureau et al., 2010).



Σχήμα 3.16: Παράδειγμα εφαρμογής pooling layer σε μια εικόνα (Karpathy, 2016).

Στο πιο πάνω σχήμα (Σχήμα 3.16), φαίνεται ότι το pooling layer πραγματοποιεί υποδειγματοληψία της εισόδου. Αριστερά: η εικόνα μεγέθους $224 \times 224 \times 64$ γίνεται pooled (περνά από το pooling layer) με φίλτρο μεγέθους δύο και stride δύο σε αποτέλεσμα εξόδου μεγέθους $112 \times 112 \times 64$. Αξίζει να σημειωθεί ότι το βάθος (depth) διατηρείται σταθερό κατά την διαδικασία του pooling. Δεξιά: Φαίνεται η χρήση της πιο κοινής μεθόδου για υποδειγματοληψία που είναι η max pooling τεχνική με stride δύο. Αυτό σημαίνει ότι ο μεγαλύτερος όρος επιλέγεται από αυτούς τους τέσσερις όρους που εξετάζονται (2×2 φίλτρο).

Πολλοί ερευνητές δεν συμπαθούν ιδιαίτερα την pooling λειτουργία και πιστεύουν ότι είναι καλύτερο να μην την χρησιμοποιούμε. Πιο συγκεκριμένα, υποστηρίζουν την απόρριψη της εισαγωγής των PLs και υποστηρίζουν αρχιτεκτονικές που αποτελούνται μόνο από επαναλαμβανόμενα συνελικτικά επίπεδα (CLs). Για να μειώσουμε το μέγεθος της αναπαράστασης των αποτελεσμάτων, προτείνουν τη χρήση μεγαλύτερου stride στα συνελικτικά επίπεδα (Springenberg et al., 2014). Η απόρριψη των pooling layers έχει επίσης βρεθεί ότι είναι σημαντική στην εκπαίδευση καλών γενετικών μοντέλων, όπως οι τροποποιητικοί αυτόματοι κωδικοποιητές ή τα γενετικά δίκτυα αντιπαραθέσεων. Φαίνεται πιθανό ότι οι μελλοντικές αρχιτεκτονικές θα διαθέτουν πολύ λίγα ή καθόλου συγκεντρωτικά στρώματα.

Στην συγκεκριμένη έρευνα, λόγω του μικρού αριθμού δεδομένων εκπαίδευσης που έχουμε, της απαίτησης του δικτύου για όσο περισσότερα δεδομένα εκπαίδευσης, των χαμηλών ποσοστών επιτυχίας (χαμηλότερα αποτελέσματα σε σχέση με νευρωνικά δίκτυα χωρίς pooling layers) Q3 που θα δούμε στα πειράματα που διεξήχθησαν (Κεφάλαιο 6), και η μη ύπαρξη λογικής στην εφαρμογή PLs για το PSSP πρόβλημα (αφού η κάθε τιμή σε ένα feature map κωδικοποιεί μια αλληλεξάρτηση μεταξύ των αμινοξέων), θα δούμε ότι η εισαγωγή PLs στα CNNs δεν βοηθά στην επίλυση του προβλήματος PSSP.

Συνοψίζοντας, ένα Convolutional Layer (CL):

- Δέχεται δεδομένα μεγέθους $W1 \times H1 \times D1$.
- Απαιτεί τέσσερις παραμέτρους: τον αριθμό των φίλτρων - K, το μέγεθος του κάθε φίλτρου - F, το stride - S και το ποσό χρήσης μηδενικών γύρο από τον πίνακα εισόδου - P.
- Παράγει αποτελέσματα εξόδου μεγέθους $W2 \times H2 \times D2$ όπου ισχύουν τα πιο κάτω:
    - $W2 = (W1 - F + 2P)/S + 1$
    - $H2 = (H1 - F + 2P)/S + 1$
    - $D2 = K$
- Δημιουργούνται $F \times F \times D1 + 1$ βάρη για κάθε φίλτρο και συνολικά $F \times F \times D1 \times K$ βάρη και K biases.

- Στο αποτέλεσμα εξόδου το d-th depth slice (μεγέθους $W2 \times H2$) είναι το αποτέλεσμα της πραγματοποίησης του convolution του d-th φίλτρου πάνω στην είσοδο με stride S και με πρόσθεση του d-th bias.

ενώ ένα Pooling Layer (PL):

- Δέχεται δεδομένα μεγέθους $W1 \times H1 \times D1$.
- Απαιτεί δύο παραμέτρους, την χωρική έκταση τους – F και το stride - S.
- Παράγει αποτελέσματα εξόδου μεγέθους $W2 \times H2 \times D2$ όπου ισχύουν τα πιο κάτω:
  - $W2 = (W1 - F + 2P)/S + 1$
  - $H2 = (H1 - F)/S + 1$
  - $D2 = D1$
- Δεν δημιουργεί νέες παραμέτρους αφού υπολογίζει μια σταθερή συνάρτηση της εισόδου.
- Δεν χρησιμοποιείται padding στα pooling layers.

3.3.5.3 Επεξήγηση Λειτουργίας Συνελικτικών Νευρωνικών Δικτύων

Όπως αναφέραμε προηγουμένως, η είσοδος ενός συνελικτικού νευρωνικού δικτύου έχει την μορφή τρισδιάστατου πίνακα. Αν για παράδειγμα χρησιμοποιούσαμε συνελικτικά νευρωνικά δίκτυα για εντοπισμό αντικειμένων σε μια εικόνα μεγέθους 100×100 σε pixels, τότε η είσοδος του συνελικτικού νευρωνικού δικτύου θα είχε την μορφή ενός τρισδιάστατου πίνακα (3D) μεγέθους $100 \times 100 \times 3$, αφού κάθε pixel εικόνας αντιστοιχεί σε τρείς άλλες τιμές red, green και blue. Για το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών (PSSP), η είσοδος του συνελικτικού νευρωνικού δικτύου αρκεί να είναι ένας δυσδιάστατος πίνακας (2D) μεγέθους $N \times 20$ (όπου N, ο συνολικός αριθμός των γραμμών του αρχείου εισόδου και 20 η κάθε πιθανότητα να είναι ένα από τα 20 αμινοξέα που γνωρίζουμε).

Για να επεξηγήσουμε ακριβώς την λειτουργία ενός συνελικτικού νευρωνικού δικτύου, ας πάρουμε για παράδειγμα ένα αρχείο εισόδου μεγέθους 5×5, στο οποίο όμως εφαρμόζουμε padding (χρήση μηδενικών γύρο από τον πίνακα εισόδου) μιας στήλης και μιας γραμμής (padding (1,1)) και παίρνουμε ως αποτέλεσμα ένα πίνακα εισόδου 7×7.

Input Volume (+pad 1) (7x7)

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	2	1	2	2	0
0	2	0	2	0	0	0
0	1	1	1	2	1	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

Σχήμα 3.17: Παράδειγμα αρχείου εισόδου μεγέθους 7×7 (Karpathy, 2016).

Το συγκεκριμένο αρχείο θα τροφοδοτηθεί στο νευρωνικό δίκτυο με την μορφή που φαίνεται πιο πάνω. Ας υποθέσουμε ότι έχουμε ένα συνελικτικό νευρωνικό δίκτυο με ένα μόνο κρυφό συνελικτικό επίπεδο. Το κρυφό επίπεδο του δικτύου έχει ρυθμιστεί με kernel μεγέθους 3×3, stride(2,2) (μετακίνηση του kernel δύο-δύο στήλες μέχρι το τέλος των στηλών (7), και μετά δύο-δύο γραμμές προς τα κάτω, μέχρι το τέλος των γραμμών (7)), και συνάρτηση ενεργοποίησης των νευρώνων την συνάρτηση ReLU (Ενότητα 3.3.6.3). Ας υποθέσουμε ότι θα εφαρμόσουμε μόνο ένα φίλτρο/kernel για feature extraction στην είσοδο του δικτύου. Το φίλτρο παίρνει αρχικά τυχαίες τιμές.

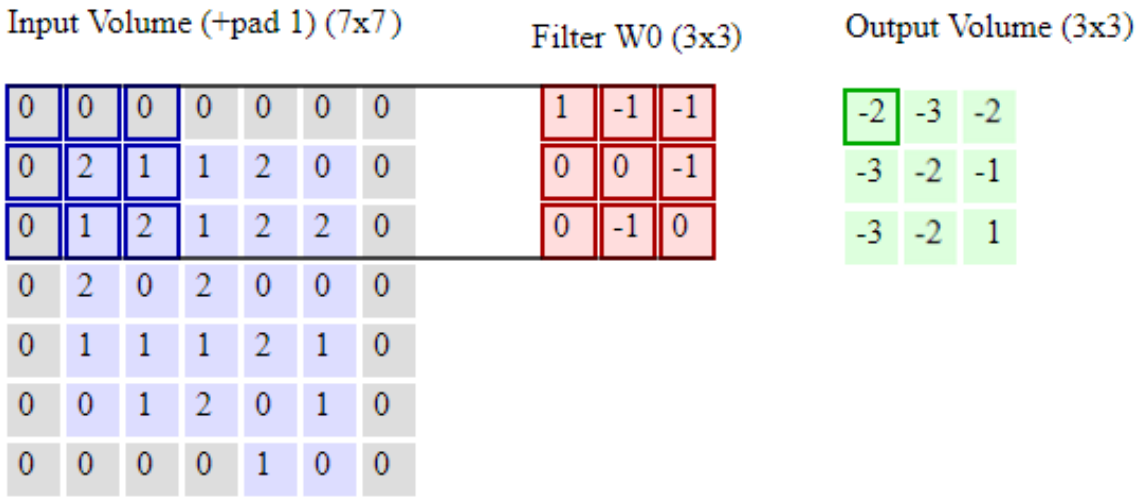
Filter W0 (3x3)

1	-1	-1
0	0	-1
0	-1	0

Σχήμα 3.18: Παράδειγμα kernel 3×3 (Karpathy, 2016).

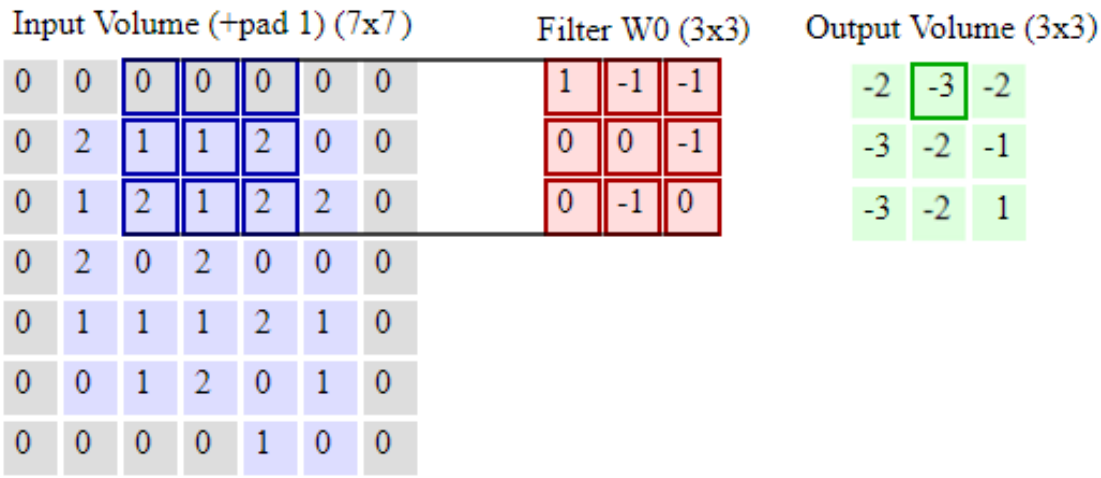
Κατά το πρώτο βήμα εκτέλεσης του αλγορίθμου το νευρωνικό δίκτυο θα επιχειρήσει να πολλαπλασιάσει και στη συνέχεια να αθροίσει τα αντίστοιχα κελιά του αρχείου εισόδου (Input Volume – Σχήμα 3.17) και του kernel (Filter W0 – Σχήμα 3.18). Η διαδικασία αυτή φαίνεται καλύτερα στο πιο κάτω σχήμα (Σχήμα 3.19). Πιο συγκεκριμένα θα γίνει η εξής πράξη: $(0 \times 1) + (0 \times -1) + (0 \times -1) + (0 \times 0) + (2 \times 0) + (1 \times -1) + (0 \times 0) + (1 \times -1) + (2 \times 0) = -2$. Με αυτό τον τρόπο, έχουμε καταφέρει να υπολογίσουμε το πρώτο στοιχείο (-2 σε πράσινο πλαίσιο) από το τελικό αποτέλεσμα εξόδου (Output Volume). Στην συνέχεια, θα

προχωρήσουμε με το να υπολογίσουμε το επόμενο στοιχείο του αποτελέσματος εξόδου μετακινώντας το kernel κατά δύο στήλες δεξιά και πραγματοποιώντας την ίδια πράξη με προηγουμένως (dot product).



Σχήμα 3.19: Πρώτο βήμα εκτέλεσης του αλγορίθμου. Αριστερά φαίνεται το αρχείο εισόδου (input volume), στη μέση το kernel (filter) το οποίο θα εφαρμόσουμε για εξαγωγή χαρακτηριστικών και δεξιά ο πίνακας εξόδου με τα αποτελέσματα (feature map). Το αποτέλεσμα της πράξης φαίνεται σε πράσινο πλαίσιο (Karpathy, 2016).

Για να μπορέσουμε να καταλάβουμε επακριβώς την διαδικασία εκτέλεσης του αλγορίθμου, στο πιο κάτω σχήμα (Σχήμα 3.20) φαίνεται και το δεύτερο βήμα εκτέλεσης του αλγορίθμου, με το οποίο υπολογίστηκε το δεύτερο αποτέλεσμα εξόδου. Η πράξη η οποία γίνεται για υπολογισμό του αποτελέσματος εξόδου είναι: $(0 \times 1) + (0 \times -1) + (0 \times -1) + (1 \times 0) + (1 \times 0) + (2 \times -1) + (2 \times 0) + (1 \times -1) + (2 \times 0) = -3$.



Σχήμα 3.20: Δεύτερο βήμα εκτέλεσης του αλγορίθμου. Αριστερά φαίνεται το αρχείο εισόδου (input volume), στη μέση το kernel (filter) το οποίο θα εφαρμόσουμε για εξαγωγή χαρακτηριστικών και δεξιά ο πίνακας εξόδου με τα αποτελέσματα (feature map). Το αποτέλεσμα που υπολογίστηκε φαίνεται σε πράσινο πλαίσιο (Karpathy, 2016).

Η διαδικασία αυτή συνεχίζεται μέχρι το kernel να διατρέξει όλο το αρχείο εισόδου. Οι διαστάσεις του πίνακα εξόδου εξαρτώνται από το μέγεθος του πίνακα εισόδου και της μεταβλητής μετακίνησης του kernel σε κάθε βήμα (stride). Όσο μεγαλύτερο μέγεθος kernel έχουμε και μεγαλύτερο stride τόσο μειώνεται το μέγεθος του πίνακα εξόδου. Η ρύθμιση zero-padding, η οποία προσθέτει μηδενικά γύρω από τον πίνακα εισόδου, χρησιμοποιείται έτσι ώστε το kernel να μπορεί να μετακινηθεί και να εφαρμοστεί ομαλά σε κάθε βήμα για τον εντοπισμό χαρακτηριστικών.

Η εφαρμογή του kernel σε κάθε περιοχή κάλυψης αντιστοιχεί σε έναν νευρώνα στο κρυφό επίπεδο. Για παράδειγμα, στο πιο πάνω σχήμα (Σχήμα 3.20), το πρώτο κρυφό επίπεδο έχει εννέα (9) νευρώνες. Η πρόσθεση περισσότερων κρυφών συνελικτικών επιπέδων στο δίκτυο προκαλεί την επανάληψη της προαναφερθείσας διαδικασίας αλλά κάθε φορά με είσοδο τα νέα μεγέθη πινάκων που υπολογίζονται μέσω του αλγορίθμου, δηλαδή τα feature maps.

Αθροίζοντας το γινόμενο των αντίστοιχων στοιχείων του πίνακα εισόδου και του kernel σε κάθε επίπεδο, ο κάθε νευρώνας, αναλύει την κάθε γειτονιά που του ανατίθεται. Με άλλα λόγια ο κάθε νευρώνας υπολογίζει μια τιμή, η οποία αντιπροσωπεύει διάφορα χαρακτηριστικά που έχουν εντοπιστεί σε μια συγκεκριμένη περιοχή. Μετακινώντας το kernel

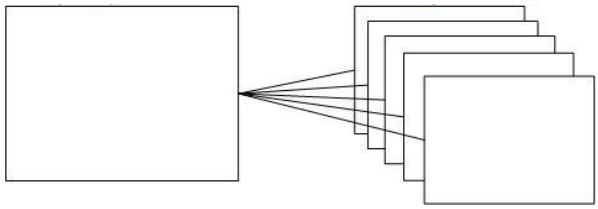
ένα συγκεκριμένο αριθμό θέσεων και εφαρμόζοντας πολλαπλά διαφορετικά kernels στην ίδια είσοδο, έχουμε σαν αποτέλεσμα το τεχνητό νευρωνικό δίκτυο να μπορεί να εντοπίσει διαφορές στις τιμές των αποτελεσμάτων των νευρώνων του κρυφού επιπέδου, άρα και διαφορετικά χαρακτηριστικά σε μια περιοχή. Με αυτό τον τρόπο το νευρωνικό δίκτυο μπορεί να καταλάβει ή καλύτερα να εντοπίσει διάφορα στοιχεία και χαρακτηριστικά που θεωρεί «μαθηματικά» ότι είναι σημαντικά.

Όπως αναφέραμε στην εισαγωγή της ενότητας 3.3.5, ένα από τα πλεονεκτήματα των συνελικτικών νευρωνικών δικτύων είναι το ότι μπορούν να εντοπίσουν χαρακτηριστικά σε πολύπλοκες ακολουθίες λόγω του μειωμένου αριθμού συναπτικών βαρών που υπάρχουν. Εξηγώντας κατάλληλα, δεδομένου μιας εικόνας εισόδου μεγέθους $32 \times 32 \times 3$ (RGB), αν το μέγεθος του kernel που θα εφαρμοστεί για εξαγωγή των χαρακτηριστικών είναι 5×5, τότε ο κάθε νευρώνας στο κρυφό συνελικτικό επίπεδο, θα είναι συνδεδεμένος με μια περιοχή $5 \times 5 \times 3$ του πίνακα εισόδου, και θα έχει συνολικά $5 \times 5 \times 3 = 75 + 1$ (bias) συνδεδεμένα συναπτικά βάρη πάνω του. Συμπερασματικά, λόγω ακριβώς αυτής της αραιής συνδεσμολογίας, ένα συνελικτικό νευρωνικό δίκτυο μπορεί να κωδικοποιήσει και να εντοπίσει πολύπλοκης μορφής χαρακτηριστικά και ιδιότητες από ένα διάνυσμα εισόδου.

Έχουμε εξηγήσει και ορίσει επακριβώς τις συνδέσεις του κάθε νευρώνα σε ένα κρυφό συνελικτικό επίπεδο ανάλογα με την είσοδο που λαμβάνει, αλλά δεν έχουμε ορίσει επακριβώς τους παράγοντες από τους οποίους εξαρτάται το αποτέλεσμα εξόδου ενός οποιουδήποτε κρυφού συνελικτικού επιπέδου. Για να ορίσουμε επακριβώς τον αριθμό και την οργάνωση των νευρώνων του αποτελέσματος εξόδου ενός οποιουδήποτε κρυφού συνελικτικού επιπέδου, θα πρέπει να λάβουμε υπόψιν τις εξής παραμέτρους: το βάθος (depth), τη μετακίνηση του kernel (stride) και το zero-padding που έχουμε.

Αρχικά, το βάθος (depth) καθορίζει και τον αριθμό των διαφορετικών φίλτρων (kernels) που θέλουμε να χρησιμοποιήσουμε, κάθε ένα εκ των οποίων ψάχνει για διαφορετικά χαρακτηριστικά στα δεδομένα εισόδου. Για παράδειγμα, αν το πρώτο κρυφό συνελικτικό

επίπεδο, παίρνει σαν είσοδο μια εικόνα σε μορφή raw[3], τότε οι αντίστοιχοι νευρώνες στα διαφορετικά βάθη, μπορεί να ενεργοποιηθούν στην παρουσία διαφορετικών χαρακτηριστικών στην περιοχή εισόδου που εξετάζουν.



Σχήμα 3.21: Αποτελέσματα (δεξιά πίνακες) εφαρμογής πέντε φίλτρων (kernels) στην είσοδο (αριστερά) (Karpathy, 2016).

Στη συνέχεια, πρέπει να θέσουμε την τιμή μετακίνησης του kernel, η οποία αναφέρεται ως stride. Αν η μεταβλητή stride έχει την τιμή ένα (1), τότε μετακινούμε τα φίλτρα ένα εικονοστοιχείο[4] (pixel) κάθε φορά. Αν η μεταβλητή stride έχει την τιμή δύο (2), τότε τα φίλτρα μετακινούνται δύο pixels κάθε φορά. Αυτό, θα δημιουργήσει αποτελέσματα εξόδου μικρότερου μεγέθους σε σχέση με τα δεδομένα εισόδου στο κρυφό επίπεδο.

Κάποιες φορές, είναι αρκετά χρήσιμο να εισάγουμε μηδενικά (zero-padding) γύρω από τα δεδομένα εισόδου που είναι σε μορφή πίνακα. Ο αριθμός των στηλών και γραμμών των μηδενικών που θα χρησιμοποιηθούν ως περίγραμμα στα δεδομένα εισόδου είναι μεταβλητός. Χρησιμοποιώντας το zero-padding, έχουμε την δυνατότητα να ελέγχουμε το μέγεθος των αποτελεσμάτων εξόδου ενός κρυφού συνελικτικού επιπέδου.

3.3.6 Support Vector Machines (SVMs)

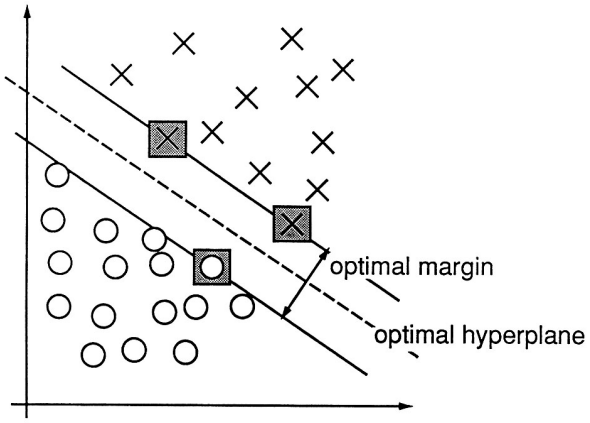
3.3.6.1 Εισαγωγή στα SVMs

Το 1995 οι Cortes & Vapnik (1995), εισήγαγαν για πρώτη φορά στους αλγόριθμους Μηχανικής Μάθησης τα Support Vector Networks (SVMs). Αρχικά τα SVMs έχουν προταθεί για δυαδικά προβλήματα κατηγοριοποίησης. Όπως αναφέρουν στο άρθρο τους, η Ιδέα πίσω από τα SVMs

[3] Raw image file: Περιέχει ελαφρώς επεξεργασμένα δεδομένα σε μορφή διανύσματος από τον αισθητήρα φωτογραφίας.

[4] Εικονοστοιχείο ή pixel: είναι ένα "σημείο" μιας εικόνας που εμφανίζεται στην οθόνη ενός υπολογιστικού συστήματος.

είναι η μη-γραμμική αναγωγή των δεδομένων εισόδου σε υψηλότερες διαστάσεις, όπου θα είναι πλέον γραμμικά διαχωρίσιμα (Cortes and Vapnik, 1995). Σύμφωνα με τον Meyer (2001), τα SVMs αποτελούν μια πολύ ισχυρή τεχνική για γραμμική και μη-γραμμική κατηγοριοποίηση, παλινδρόμηση, και εντοπισμό outliers, με μια διαισθητική αναπαράσταση του μοντέλου.



Σχήμα 3.22: Παράδειγμα γραμμικά διαχωρίσιμου προβλήματος στον δυσδιάστατο (2D) χώρο. Τα support vectors (γκρίζα τετράγωνα), καθορίζουν τα όρια του μεγαλύτερου διαστήματος διαχωρισμού των δύο κλάσεων.

3.3.6.2 Επεξήγηση Βασικής Λειτουργίας του SVM

Για να μπορέσουν τα SVMs να διαχωρίσουν δύο γραμμικά διαχωρίσιμες κλάσεις με την καλύτερη δυνατή επιφάνεια διαχωρισμού (optimal hyperplane), προσπαθούν να μεγιστοποιήσουν την απόσταση μεταξύ των κοντινότερων σημείων, των διαφορετικών κλάσεων. Τα σημεία τα οποία είναι στα όρια αυτού του διαχωρισμού ονομάζονται support vectors, και η μεσαία γραμμή διαχωρισμού ανάμεσα στα σημεία αυτά (support vectors) είναι η ιδανική επιφάνεια διαχωρισμού των δύο αυτών κλάσεων.

Σε περίπτωση που έχουμε σημεία τα οποία βρίσκονται στην περιοχή άλλων κλάσεων (overlapping classes) δεν τα λαμβάνουμε υπόψιν σε μεγάλο βαθμό, για να μειώσουμε έτσι την επιρροή αυτών των σημείων στην γενική λύση. Τα SVMs, όταν δεν μπορούν να διαχωρίσουν γραμμικά ένα πρόβλημα, τότε ανάγουν τα σημεία των κλάσεων προς μελέτη, σε υψηλότερες διαστάσεις, όπου τα σημεία πλέον θα είναι γραμμικά διαχωρίσιμα. Αυτή η διαδικασία

πραγματοποιείται με χρήση τεχνικών με kernels. Ένα πρόγραμμα το οποίο μπορεί να εκτελέσει τις πιο πάνω διαδικασίες, λέγεται Support Vector Machine (SVM).

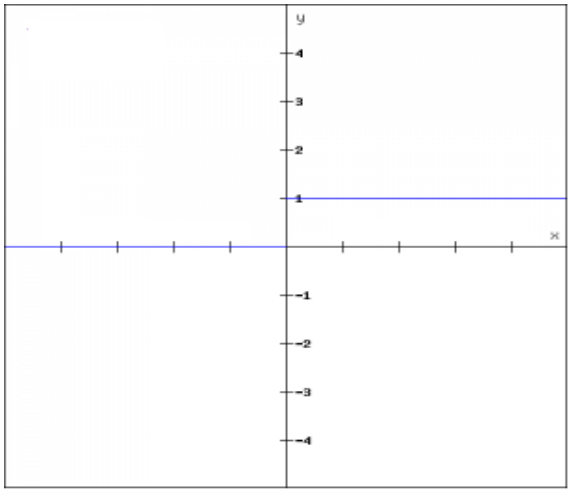
3.3.6.3 Συνδυασμός CNNs και SVMs

Σύμφωνα με την έρευνα των Kountouris et al. (2012), τα SVMs είχαν την καλύτερη απόδοση σε θέματα φιλτραρίσματος για το PSSP πρόβλημα. Βασισμένοι στα αποτελέσματα της έρευνα αυτής, αποφασίσαμε όπως συνδυάσουμε τα συνελικτικά νευρωνικά δίκτυα με τα support vector machines. Πιο συγκεκριμένα, αποφασίσαμε να χρησιμοποιήσουμε τα SVMs για φιλτράρισμα των αποτελεσμάτων πρόβλεψης τα οποία λάβαμε από τα CNNs, έτσι ώστε να διορθώσουμε ως ένα βαθμό, τα αποτελέσματα πρόβλεψης των CNNs πετυχαίνοντας έτσι υψηλότερα ποσοστά επιτυχίας Q3 και υψηλότερες SOV αποδόσεις.

3.3.7 Συναρτήσεις Ενεργοποίησης

3.3.7.1 Βηματική Συνάρτηση – Συνάρτηση Σκαλί

Η βηματική συνάρτηση ή συνάρτηση σκαλί (Σχήμα 3.23), είναι εμπνευσμένη σε μεγάλο βαθμό από τους βιολογικούς νευρώνες του εγκεφάλου. Δοσμένης μιας τιμής κατωφλίου έστω θ, τότε όπως ο βιολογικός νευρώνας, έτσι και ο τεχνητός, αν το άθροισμα των επιμέρους εισόδων του νευρώνα ξεπεράσει την τιμή κατωφλίου θ, τότε ο νευρώνας πυροδοτεί.

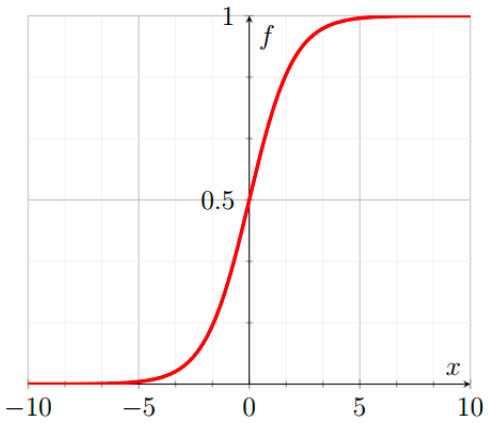


Σχήμα 3.23: Η βηματική συνάρτηση ενεργοποίησης με εξίσωση $\psi = \begin{cases} 1, \varepsilon\acute{\alpha}\nu\ x \geq \theta \\ 0, \varepsilon\acute{\alpha}\nu\ x < \theta \end{cases}$

Η συγκεκριμένη συνάρτηση είναι γραμμική, έχει πεδίο τιμών {0,1}, θεωρείται η πιο απλή συνάρτηση ενεργοποίησης, και δεν είναι τόσο αποδοτική αφού η μόνη πληροφορία που μπορούμε να λάβουμε από την βηματική συνάρτηση, είναι το 1 (ενεργοποίηση) ή το 0 (μη-ενεργοποίηση). Δεν μπορούμε με λίγα λόγια, να ξέρουμε αν το άθροισμα των εισόδων του νευρώνα προκάλεσε ενεργοποίηση ή μη ενεργοποίηση του νευρώνα οριακά (το x κοντά στο θ) ή όχι. Ομαδοποιώντας τα πιο πάνω, η βηματική συνάρτηση ενεργοποίησης δεν παρέχει αρκετή πληροφορία για το πόσο κοντά ή πόσο μακριά είμαστε, από το επιθυμητό αποτέλεσμα εξόδου. Κατά την εκπαίδευση του δικτύου θέλουμε να πραγματοποιούμε αλλαγές στα συναπτικά βάρη, έτσι ώστε στην επόμενη επανάληψη να προσεγγίζουμε την λύση του προβλήματος με μεγαλύτερη ακρίβεια. Ψάχνουμε συνεπώς, για την κατάλληλη αλλαγή στα συναπτικά βάρη έτσι ώστε στην επόμενη επανάληψη να μειωθεί η απόκλιση του νευρωνικού δικτύου από το επιθυμητό αποτέλεσμα, κάτι που η βηματική συνάρτηση με τις ελάχιστες πληροφορίες που παρέχει για το πόσο κοντά ή μακριά είμαστε από την επιθυμητή έξοδο, δεν είναι σε θέση να εντοπίσει. Συμπερασματικά, υλοποιώντας ένα τεχνητό νευρωνικό δίκτυο χρησιμοποιώντας την βηματική συνάρτηση ενεργοποίησης, δεν θα είμαστε σε θέση να πραγματοποιήσουμε ακριβής αλλαγές στα βάρη του νευρωνικού δικτύου με απώτερο σκοπό την επίτευξη μεγαλύτερων ποσοστών επιτυχίας στις προβλέψεις του δικτύου. Τέλος, χρησιμοποιώντας την βηματική συνάρτηση μπορούμε να λύσουμε μόνο γραμμικά διαχωρίσιμα προβλήματα.

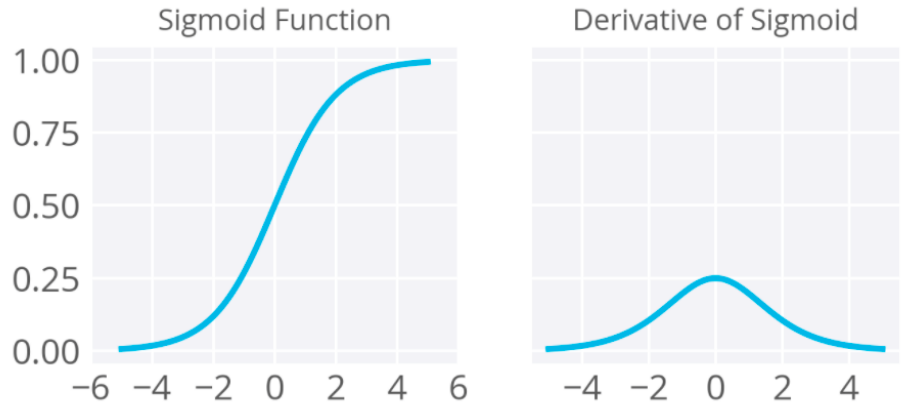
3.3.7.2 Σιγμοειδής Συνάρτηση

Η σιγμοειδής συνάρτηση (Σχήμα 3.24), είναι μια μη-γραμμική συνάρτηση η οποία έχει το χαρακτηριστικό σχήμα «S» και πεδίο τιμών το {0,1}. Πιο συγκεκριμένα, η σιγμοειδής συνάρτηση, είναι οριοθετημένη (παίρνει τιμές από 0-1), κάτι που σημαίνει ότι δεν θα έχουμε «εκρήξεις» (μεγάλες ή μικρές τιμές) στα αποτελέσματα εξόδου του νευρώνα, διαφορική, πραγματικών τιμών εξόδου συνάρτηση, που ορίζεται για όλες τις πραγματικές τιμές εισόδου και έχοντας μια μη-αρνητική παράγωγο σε κάθε της σημείο. Με τον όρο διαφορική, εννοούμε ότι μπορούμε να βρούμε την κλίση της συνάρτησης σε οποιαδήποτε δύο σημεία. Η σιγμοειδής συνάρτηση είναι μονοτονική αλλά η παράγωγος της όχι.



Σχήμα 3.24: Η σιγμοειδής συνάρτηση ενεργοποίησης με εξίσωση $f(x) = \frac{1}{1+e^{-x}}$

Η σιγμοειδής συνάρτηση είναι μια από τις πιο «δημοφιλείς» συναρτήσεις ενεργοποίησης αφού είναι παραγωγίσιμη (Σχήμα 3.25 – Δεξιά γραφική) και για το λόγο αυτό μπορούμε να πάρουμε πληροφορίες για το πόσο κοντά ή μακριά είμαστε από την επιθυμητή έξοδο. Η σιγμοειδής συνάρτηση, αντίθετα με τη βηματική συνάρτηση ενεργοποίησης, είναι μη-γραμμική. Χρησιμοποιώντας μια μη-γραμμική συνάρτηση ενεργοποίησης, είμαστε σε θέση να λύσουμε προβλήματα τα οποία δεν είναι γραμμικά διαχωρίσιμα όπως η συνάρτηση XOR.



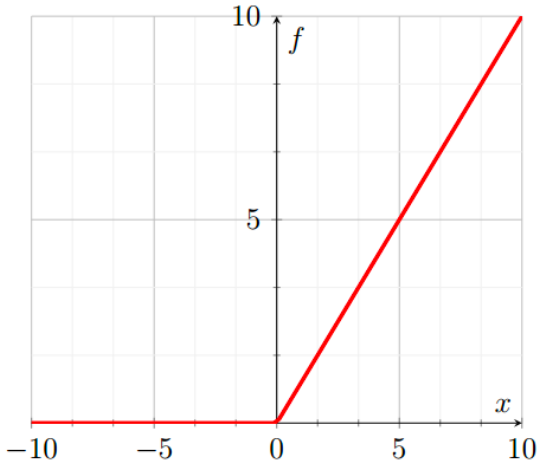
Σχήμα 3.25: Αριστερά η σιγμοειδής συνάρτηση, Δεξιά η πρώτη παράγωγος της σιγμοειδούς συνάρτησης.

Από την γραφική παράσταση της σιγμοειδούς συνάρτησης (Σχήμα 3.24), μπορούμε να δούμε ότι η ψ συντεταγμένη φαίνεται να αλλάζει πάρα πολύ λίγο σε αλλαγές της συντεταγμένης χ στα δύο άκρα της γραφικής παράστασης. Αυτό σημαίνει ότι η πρώτη παράγωγος (Σχήμα 3.25 –

Δεξιά γραφική) σε εκείνες τις περιοχές αναμένεται να είναι εξαιρετικά μικρή, κάτι που προκαλεί το πρόβλημα του «vanishing gradient» (Ενότητα 3.3.8), το οποίο είναι ένα από τις αδυναμίες της σιγμοειδούς συνάρτησης.

3.3.7.3 Συνάρτηση Rectifier

Τα πιο πρόσφατα βαθιά τεχνητά νευρωνικά δίκτυα, χρησιμοποιούν rectifier γραμμικούς νευρώνες (Rectified Linear Units - ReLUs) στα κρυφά επίπεδα, οι οποίοι έχουν σαν συνάρτηση ενεργοποίησης την συνάρτηση rectifier (Σχήμα 3.26) με πεδίο τιμών το {0,∞}, αντί νευρώνες που εφαρμόζουν την σιγμοειδή συνάρτηση (Σχήμα 3.24). Ένας ReLU νευρώνας ο οποίος χρησιμοποιεί σαν συνάρτηση ενεργοποίησης την rectifier συνάρτηση, έχει σαν αποτέλεσμα εξόδου 0 αν το άθροισμα των εισόδων του νευρώνα είναι μικρότερο του μηδενός, διαφορετικά έχει σαν αποτέλεσμα εξόδου το άθροισμα των εισόδων του νευρώνα. Με λίγα λόγια η συνάρτηση ενεργοποίησης rectifier το μόνο που κάνει είναι να βάζει ένα κάτω όριο, το 0, στην έξοδο του ReLU νευρώνα.



Σχήμα 3.26: Η Συνάρτηση ενεργοποίησης rectifier με εξίσωση $f(x) = \max(x, 0)$

Η rectifier συνάρτηση ενεργοποίησης, είναι η πιο απλή μη-γραμμική συνάρτηση που μπορούμε να εφαρμόσουμε σε ένα νευρώνα. Όταν έχουμε θετική είσοδο, το αποτέλεσμα της πρώτης παραγώγου της rectifier συνάρτησης είναι απλά 1, οπότε αποτρέπουμε κατά κάποιο βαθμό το φαινόμενο του «vanishing gradient» (Ενότητα 3.3.8) που έχουμε στην σιγμοειδή συνάρτηση. Έρευνες έχουν δείξει ότι τεχνητά νευρωνικά δίκτυα τα οποία χρησιμοποιούν ReLUs, τείνουν να εκπαιδεύονται με μεγαλύτερες ταχύτητες σε μεγάλα νευρωνικά δίκτυα.

Το πρόβλημα που υπάρχει με τους ReLUs είναι το ότι μπορεί να είναι «νεκροί». Αυτό σημαίνει ότι μεγάλο μέρος αυτών των νευρώνων στο νευρωνικό δίκτυο μπορεί να έχουν την τιμή 0 και συνεπώς να προκαλέσουν βλάβη στο δίκτυο. Μπορούμε να ξεπεράσουμε το συγκεκριμένο πρόβλημα, με το να θέσουμε σωστά τις παραμέτρους του δικτύου καθώς επίσης και χρησιμοποιώντας παραλλαγές των ReLUs, όπως οι Noisy ReLUs, οι Leaky ReLUs και οι ELUs που εγγυώνται ότι το αποτέλεσμα εξόδου των νευρώνων δεν θα είναι ποτέ μηδέν (0), αλλά ελαφρώς διαφορετικό (Xu et al., 2015).

3.3.7.4 Softmax Συνάρτηση

Η σιγμοειδής συνάρτηση μας δίνει αρκετές πληροφορίες για το πόσο κοντά η μακριά είμαστε από το επιθυμητό αποτέλεσμα εξόδου αλλά προκαλεί το πρόβλημα του «vanishing gradient» σε βαθιά νευρωνικά δίκτυα. Αντίθετα, η rectifier συνάρτηση ενεργοποίησης, καταπολεμά το φαινόμενο του «vanishing gradient» και επιταχύνει το χρόνο εκπαίδευσης του νευρωνικού δικτύου. Παρόλα αυτά όταν έχουμε να λύσουμε ένα πρόβλημα κατηγοριοποίησης, δεν φαίνεται να βοηθά ιδιαίτερα.

Σχήμα 3.27: Η κανονικοποίηση των αποτελεσμάτων εξόδου με χρήση της softmax συνάρτησης (Chablani, 2017).

Η softmax συνάρτηση ενεργοποίησης, κανονικοποιεί τα αποτελέσματα εξόδου κάθε νευρώνα έτσι ώστε να είναι στο διάστημα {0,1} όπως ακριβώς και η σιγμοειδής συνάρτηση. Επιπρόσθετα, κανονικοποιεί τα δεδομένα αυτά με τέτοιο τρόπο, έτσι ώστε το συνολικό άθροισμα των αποτελεσμάτων εξόδου ενός νευρώνα να είναι ίσο με 1 (Σχήμα 3.27).

Τα αποτελέσματα εξόδου της softmax συνάρτησης ενεργοποίησης, είναι ανάλογα μιας κατανομής πιθανοτήτων κατηγοριών, δηλαδή κάθε πραγματική τιμή του διανύσματος εξόδου

της συνάρτησης softmax, μας λέει την πιθανότητα, η είσοδος, να ανήκει στη συγκεκριμένη κατηγορία. Μαθηματικά μιλώντας, η συνάρτηση ενεργοποίησης softmax, ορίζεται ως εξής:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

3.3.8 Vanishing και Exploding Gradient Problem

Στα νευρωνικά δίκτυα και γενικότερα στη μηχανική μάθηση, το πρόβλημα του «vanishing gradient» είναι μια δυσκολία η οποία προκύπτει κατά την εκπαίδευση τεχνητών νευρωνικών δικτύων, τα οποία χρησιμοποιούν μεθόδους μάθησης βασισμένες στην πρώτη παράγωγο συναρτήσεων (κλίση συνάρτησης) και στην ανάστροφη μετάδοσης σφάλματος. Σε τέτοιες μεθόδους, κάθε συναπτικό βάρος του δικτύου αλλάζει ανάλογα με το αρνητικό της παραγώγου της συναρτήσεως του σφάλματος ως προς τα βάρη, σε κάθε επανάληψη της εκπαίδευσης του δικτύου. Το πρόβλημα το οποίο προκύπτει σε αυτό το σημείο, είναι το ότι σε κάποιες περιπτώσεις, η πρώτη παράγωγος (κλίση) της συνάρτησης θα έχει τόσο μικρή τιμή που θα προκαλεί απείρως ελάχιστη αλλαγή στα συναπτικά βάρη, και στο χειρότερο σενάριο θα εμποδίσει εντελώς το τεχνητό νευρωνικό δίκτυο από το να εκπαιδευτεί/μάθει. Σαν ένα παράδειγμα της αιτίας του συγκεκριμένου προβλήματος, οι κλασσικές συναρτήσεις ενεργοποίησης (σιγμοειδής, βηματική κτλ.), έχουν τιμές της πρώτης παραγώγου τους στο διάστημα {0,1}, και η ανάστροφη μετάδοση σφάλματος υπολογίζει τις πρώτες παραγώγους χρησιμοποιώντας τον κανόνα αλυσίδας. Αυτό έχει ως συνέπεια να έχουμε εκθετική μείωση της πρώτης παραγώγου (σήμα σφάλματος) καθώς μεταδίδεται το σφάλμα προς τα πίσω στο δίκτυο κατά την μέθοδο ανάστροφης μετάδοσης σφάλματος, κάτι που έχει ως αποτέλεσμα τα «πρώτα» επίπεδα του νευρωνικού δικτύου να εκπαιδεύονται με εξαιρετικά αργούς ρυθμούς.

Σε μεταγενέστερο στάδιο το πρόβλημα αυτό ορίστηκε ως το «vanishing gradient problem», το οποίο επηρεάζει, όχι μόνο νευρωνικά δίκτυα εμπρόσθιου περάσματος με πολλά επίπεδα, αλλά και τα νευρωνικά δίκτυα με ανάδραση (Medsker and Jain, 2001).

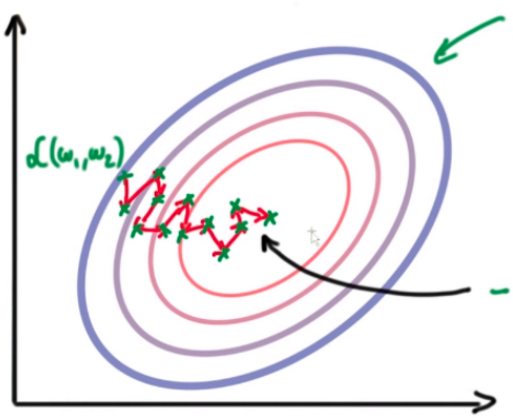
Αντίστοιχα, όταν χρησιμοποιούνται συναρτήσεις ενεργοποίηση των οποίων οι τιμές της πρώτης παραγώγου τους μπορεί είναι πολύ μεγάλες, μπορεί να δημιουργηθεί το πρόβλημα του «exploding gradient» - έκρηξη κλίσης συνάρτησης (Pascanu et al., 2012) .

3.3.9 Updaters για Διαχείριση της Μεταβολής του Ρυθμού Μάθησης

3.3.9.1 Γενικά

Ο ρυθμός μάθησης (learning rate), είναι μια από τις πιο σημαντικές, αν όχι η πιο σημαντική παράμετρος ρύθμισης ενός νευρωνικού δικτύου. Ο ρυθμός μάθησης συμβάλει στην αλλαγή των συναπτικών βαρών (weights) του νευρωνικού δικτύου. Το μυστικό για την επιτυχή εκμάθηση ενός νευρωνικού δικτύου, κρύβεται πολλές φορές στην κατάλληλη διαχείριση της αλλαγής του ρυθμού μάθησης. Στις πιο κάτω υπό-ενότητες, θα μελετήσουμε κάποιες μεθόδους ενημέρωσης (updaters), οι οποίες διαφέρουν στον τρόπο που χειρίζονται την παράμετρο του ρυθμού μάθησης. Έπειτα, στο κεφάλαιο 6, θα πραγματοποιήσουμε κάποια πειράματα, στα οποία θα συγκρίνουμε τις μεθόδους αυτές για το πρόβλημα που μελετούμε σε αυτήν την έρευνα, το οποίο είναι η πρόβλεψη δευτεροταγούς δομής πρωτεϊνών.

Στην ενότητα 3.3.4.4, έχουμε περιγράψει με λεπτομέρεια τη λειτουργία της μεθόδου κατάβασης κλίσης για εύρεση ολικού ελαχίστου μιας συνάρτησης, και συγκεκριμένα της συναρτήσεως του σφάλματος. Η μέθοδος κατάβασης κλίσης ανήκει στις μεθόδους πρώτης τάξης αφού για να ελαχιστοποιήσει τη συνάρτηση του σφάλματος υπολογίζει την πρώτη παράγωγο της συνάρτησης αυτής. Κατά την μέθοδο της κατάβασης κλίσης, τα συναπτικά βάρη του δικτύου αλλάζουν ανάλογα με το αρνητικό της παραγώγου της συναρτήσεως του σφάλματος ως προς τα συναπτικά βάρη. Αν ο ρυθμός μάθησης είναι πολύ μικρός, τότε η σύγκλιση του δικτύου θα είναι σχετικά αργή, ενώ αντίθετα αν ο ρυθμός μάθησης είναι μεγάλος, τότε ο αλγόριθμος μπορεί να μην καταφέρει να εντοπίσει το ολικό ελάχιστο της συνάρτησης, άρα να μην επιτρέψουμε στο νευρωνικό δίκτυο να εκπαιδευτεί ιδανικά.



Σχήμα 3.28: Παράδειγμα σύγκλισης μεθόδου κατάβασης κλίσης (Retrieved form https://deeplearning4j.org/updater).

Στο πιο πάνω σχήμα (Σχήμα 3.28) μπορούμε να δούμε ένα παράδειγμα σύγκλισης της μεθόδου κατάβασης κλίσης. Η πρώτη παράγωγος της συναρτήσεως του σφάλματος αλλάζει αρκετά, μετά από κάθε επανάληψη του νευρωνικού δικτύου, λόγω της διαφορετικότητας των παραδειγμάτων εκπαίδευσης. Στο παράδειγμα του σχήματος 3.28, ο αλγόριθμος ενημέρωσης των βαρών/updater, πραγματοποιεί μικρά ζίκ-ζάκ βήματα, για την πορεία του προς το ολικό ελάχιστο της συνάρτησης. Ο αλγόριθμος ενημερώνει τα βάρη σύμφωνα με την εξίσωση 3.5. Με αυτό τον τρόπο, η μέθοδος κατάβασης κλίσης πραγματοποιεί την προσπάθεια της για ελαχιστοποίηση της συνάρτησης του σφάλματος.

$$w_{i+1} = w_i - a\frac{dE}{dw_i}$$

Εξίσωση 3.5: Εξίσωση ενημέρωσης βαρών. Όπου w_i το βάρος w την χρονική στιγμή i, a ο ρυθμός μάθησης και $\frac{dE}{dw_i}$ η πρώτη παράγωγος της συναρτήσεως του σφάλματος ως προς τα βάρη. Οι πιο κάτω μέθοδοι ενημέρωσης και διαχείρισης του ρυθμού μάθησης έχουν εφαρμογή σε μεθόδους ελαχιστοποίησης του σφάλματος, όπως η μέθοδος κατάβασης κλίσης.

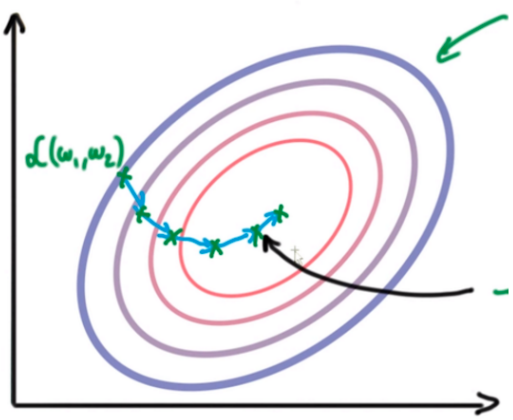
3.3.9.2 Ορμή

Ο όρος ορμή, προέρχεται από την επιστήμη της φυσικής και ορίζεται ως το γινόμενο της μάζας - m ενός αντικειμένου επί την ταχύτητά - u του. Στην μηχανική μάθηση, ο όρος ορμή διαφέρει κατά κάποιο τρόπο αφού είναι απλά ένας παράγοντας που χρησιμοποιείται για να

προσηλώσουμε τις ενέργειες του updater με βάση προηγούμενα βήματα τα οποία έχουν πραγματοποιηθεί. Χρησιμοποιώντας τον παράγοντα της ορμής, ενημερώνουμε τα βάρη του νευρωνικού δικτύου χρησιμοποιώντας την εξίσωση 3.6. Με αυτό τον τρόπο αποφεύγουμε το ζίκ-ζάκ που προκαλείται κατά την μέθοδο κατάβασης κλίσης χωρίς τον παράγοντα της ορμής, οπότε η σύγκλιση του δικτύου συμπεριλαμβάνοντας τον παράγοντα της ορμής κατά την μέθοδο κατάβασης κλίσης, θα πραγματοποιηθεί όπως στο σχήμα 3.29.

$$\Delta w_k(i) = -a\frac{dE}{dw_k} + \mu\Delta w_k(i-1)$$

Εξίσωση 3.6: Ο υπολογισμός της τιμής ενημέρωσης του βάρους w_k την χρονική στιγμή i. Όπου $\frac{dE}{dw_k}$ η πρώτη παράγωγος της συναρτήσεως του σφάλματος ως προς τα βάρη και μ ο παράγοντας ορμής.



Σχήμα 3.29: Παράδειγμα σύγκλισης συνάρτησης με χρήση παράγοντα ορμής (Retrieved form https://deeplearning4j.org/updater).

3.3.9.3 Adagrad (Adaptive Gradient Algorithm)

Η μέθοδος Adagrad (Adaptive Gradient Algorithm), κλιμακώνει το ρυθμό μάθησης σύμφωνα με το ιστορικό των παραγώγων/κλίσεων των προηγούμενων βημάτων. Αυτό γίνεται διαιρώντας την τρέχουσα κλίση στον κανόνα ενημέρωσης, με το άθροισμα των προηγούμενων κλίσεων. Ως αποτέλεσμα, όταν η κλίση/πρώτη παράγωγος είναι πολύ μεγάλη, ο ρυθμός μάθησης μειώνεται και αντίστροφα (Duchi et al., 2011).

Εμπειρικά μιλώντας, η μέθοδος Adagrad δουλεύει αρκετά καλά, δηλαδή η σύγκλιση είναι γρηγορότερη και πιο αξιόπιστη σε σχέση με τη μέθοδο κατάβασης κλίσης. Επιπρόσθετα, η μέθοδος Adagrad, δεν εξαρτάται τόσο από την αρχική τιμή του ρυθμού μάθησης. Πρέπει απλά να θέσουμε μια αρχική τιμή στο ρυθμό μάθησης για την οποία το νευρωνικό δίκτυο συγκλίνει σε ένα αποδεκτό χρονικό πλαίσιο. Η μέθοδος αυτή, έχει ως φυσική ιδιότητα το να μειώνει αποδοτικά το ρυθμό μάθησης συναρτήσει του χρόνου.

3.3.9.4 RMSProp

Η μόνη διαφορά μεταξύ της μεθόδου RMSProp και της μεθόδου Adagrad είναι ότι η παράγωγος/κλίση της συνάρτησης, υπολογίζεται με την εκθετική μείωση του μέσου όρου και όχι του αθροίσματος, των κλίσεων (Hinton et al., 2012).

3.3.9.5 AdaDelta

Η συγκεκριμένη μέθοδος διαχείρισης του ρυθμού μάθησης προσαρμόζεται δυναμικά με την πάροδο του χρόνου, χρησιμοποιώντας μόνο πληροφορίες πρώτης τάξης και έχει ελάχιστη υπολογιστική επιβάρυνση πέραν από τις υπολογιστικές απαιτήσεις της μεθόδου κατάβασης κλίσης. Η μέθοδος αυτή, δεν απαιτεί χειροκίνητη ρύθμιση του ρυθμού μάθησης και φαίνεται να είναι ανθεκτική σε θορυβώδης πληροφορίες κλίσης, διαφορετικές επιλογές αρχιτεκτονικής μοντέλου δικτύου και διαφορετικές μορφές δεδομένων και συνδυασμών παραμέτρων (Zeiler, 2012).

3.3.9.6 ADAM (Adaptive Moment Estimation)

Η μέθοδος ADAM (Adaptive Moment Estimation), είναι μια άλλη μέθοδος που υπολογίζει προσαρμοσμένους ρυθμούς μάθησης για κάθε ξεχωριστή παράμετρο. Εκτός από την αποθήκευση ενός εκθετικά φθίνοντος μέσου όρου των παλιότερων τετραγωνικών κλίσεων της συνάρτησης του σφάλματος, η μέθοδος ADAM, διατηρεί επίσης έναν εκθετικά φθίνοντα μέσο όρο των παλιότερων κλίσεων (Kingma and Ba, 2015).

Κεφάλαιο 4

Φίλτρα Gabor

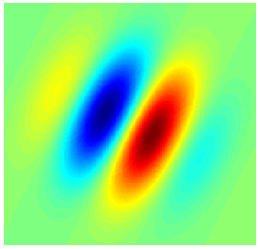
4.1	Εισαγωγή στα Φίλτρα Gabor	84
4.2	Μαθηματικός Ορισμός Φίλτρου Gabor	85
4.3	Σύνδεση Φίλτρων Gabor με Συνελικτικά Νευρωνικά Δίκτυα	86

4.1 Εισαγωγή στα Φίλτρα Gabor

Η υφή αναπαρίσταται από την απόκριση μιας σειράς από φίλτρα που εφαρμόζονται σε μια εικόνα. Η υφή μιας εικόνας είναι η επεξεργασία της εικόνας σε πολλαπλές κλίμακες (multi scale image processing). Αυτή η επεξεργασία της εικόνας σε πολλαπλές κλίμακες με σκοπό να καθορίσουμε την υφή μιας εικόνας, προέρχεται από το γεγονός ότι το ανθρώπινο σύστημα όρασης (Human Visual System - HVS), αποτελείται από πολλαπλά ζωνοδιαβατά, χωρικά φίλτρα συντονισμένα σε διαφορετικές συχνότητες (Multiple Spatial Frequency Channels).

Στην επεξεργασία εικόνας, ένα φίλτρο Gabor το οποίο ονομάστηκε έτσι από λόγω του δημιουργού του Dennis Gabor, είναι ένα γραμμικό φίλτρο που χρησιμοποιείται για ανάλυση υφής, πράγμα που σημαίνει ότι βασικά αναλύει αν υπάρχει περιεχόμενο συγκεκριμένης συχνότητας στην εικόνα σε συγκεκριμένες κατευθύνσεις σε μια τοπική περιοχή γύρω από το σημείο ή την περιοχή της ανάλυσης. Υποστηρίζεται από πολλούς σύγχρονους επιστήμονες της όρασης ότι οι παραστάσεις συχνότητας και προσανατολισμού των φίλτρων Gabor είναι παρόμοιες με εκείνες του ανθρώπινου οπτικού συστήματος, αν και δεν υπάρχουν εμπειρικά στοιχεία και λειτουργικό σκεπτικό για την υποστήριξη της ιδέας. Έχουν βρεθεί ότι είναι ιδιαίτερα κατάλληλες για την αναπαράσταση και διάκριση της υφής. Στον χωρικό τομέα, ένα δισδιάστατο (2D) φίλτρο Gabor είναι μια Γκαουσιανή (Gaussian) συνάρτηση πυρήνα, διαμορφωμένη από ένα ημιτονοειδές επίπεδο κύμα (Jones and Palmer, 1987).

Μερικοί συγγραφείς ισχυρίζονται ότι τα απλά κύτταρα στον οπτικό φλοιό των εγκεφάλων των θηλαστικών μπορούν να μοντελοποιηθούν χρησιμοποιώντας συναρτήσεις Gabor. Έτσι, η ανάλυση εικόνας με φίλτρα Gabor θεωρείται από μερικούς ότι είναι παρόμοια με την αντίληψη στο ανθρώπινο οπτικό σύστημα.



Εικόνα 4.1: Παράδειγμα δισδιάστατου φίλτρου Gabor.

Στην επεξεργασία εικόνας και στην υπολογιστική όραση, τα φίλτρα Gabor χρησιμοποιούνται συνήθως για εντοπισμό ακμών, ανάλυση υφής, εντοπισμό χαρακτηριστικών κτλ. Είναι μια ειδική μορφή bandpass φίλτρων, που σημαίνει ότι εντοπίζουν συγκεκριμένες συχνότητες (bands) ενώ απορρίπτουν κάποιες άλλες.

4.2 Μαθηματικός Ορισμός Φίλτρου Gabor

Η δημιουργία ενός φίλτρου Gabor, θα μας δώσει σαν αποτέλεσμα ένα ημιτονοειδές κύμα (επίπεδο κύμα για δύο διαστάσεων φίλτρα Gabor) πολλαπλασιασμένο με μια Γκαουσιανή συνάρτηση. Εξαιτίας ακριβώς αυτού του πολλαπλασιασμού - συνέλιξης (convolution), ο μετασχηματισμοί Φουριέρ του αποτελέσματος εφαρμογής ενός φίλτρου Gabor είναι η συνέλιξη του μετασχηματισμού Φουριέρ μιας αρμονικής συνάρτησης και ο μετασχηματισμός Φουριέρ μιας Γκαουσιανής συνάρτησης. Το φίλτρο έχει δύο κομμάτια, το πραγματικό (real) και το φανταστικό (imaginary) κομμάτι τα οποία είναι κάθετα μεταξύ τους. Τα δύο αυτά κομμάτια, μπορούν να συμπτυχθούν σε ένα κομμάτι με το όνομα complex ή να χρησιμοποιηθούν ξεχωριστά. Πιο κάτω φαίνονται οι μαθηματικοί ορισμοί των πιο πάνω.

Complex:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

Real:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

Imaginary:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \sin\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

Όπου $x' = x\cos\theta + y\sin\theta$ και $y' = -x\sin\theta + y\cos\theta$

Στις πιο πάνω εξισώσεις το $\boldsymbol{\lambda}$ αναπαριστά το μήκος κύματος της ημιτονοειδούς συνάρτησης, το $\boldsymbol{\vartheta}$ αναπαριστά τον προσανατολισμό των κανονικών σε σχέση με τις παράλληλες γραμμές της

συνάρτησης Gabor, το $\boldsymbol{\psi}$ είναι το phase offset, το $\boldsymbol{\sigma}$ είναι η τυπική απόκλιση της Γκαουσιανής συνάρτησης, και τέλος το $\boldsymbol{\gamma}$ είναι η αναλογία χωρικών διαστάσεων και καθορίζει την ελλειπτικότητα της συνάρτησης Gabor (Movellan, 2008).

4.3 Σύνδεση Φίλτρων Gabor με Συνελικτικά Νευρωνικά Δίκτυα

Η ιδέα που έχουμε για μελέτη και υλοποίηση, είναι η προσπάθεια πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών με τη χρήση των συνελικτικών νευρωνικών δικτύων (Convolutional Neural Networks - CNNs) και των φίλτρων Gabor. Η ιδέα αυτή προέρχεται εν μέρει, από την έρευνα των Wang et al. (2016), η οποία κατάφερε να ξεπεράσει κατά πολύ κάποιους από τους καλύτερους predictors για το πρόβλημα της πρόβλεψης δευτεροταγούς δομής των πρωτεϊνών που είχαμε μέχρι σήμερα, πετυχαίνοντας κάποια από τα υψηλότερα ποσοστά επιτυχίας. Πιστεύουμε ότι με το να χρησιμοποιήσουμε την κλασσική υλοποίηση των συνελικτικών νευρωνικών δικτύων σε συνδυασμό με τα φίλτρα Gabor θα καταφέρουμε να πετύχουμε ένα μεγαλύτερο ποσοστό επιτυχίας αφού τώρα τα δεδομένα εισόδου για το συνελικτικό νευρωνικό δίκτυο θα εμπλουτιστούν και με τα νέα δεδομένα εισόδου και χαρακτηριστικά που θα εντοπίσουμε στις πρωτεΐνες χρησιμοποιώντας τα φίλτρα Gabor σε διάφορες κατευθύνσεις και προσανατολισμούς. Για να πετύχουμε τα πιο πάνω, θα εφαρμόζουμε κάθε φορά τα φίλτρα Gabor πάνω στα δεδομένα εισόδου που έχουμε, έτσι ώστε να παράγουμε κάποια άλλα, νέα δεδομένα εισόδου για το συνελικτικό νευρωνικό δίκτυο, στα οποία φαίνονται και τα μοτίβα/χαρακτηριστικά που έχουν εντοπιστεί από τα φίλτρα Gabor προηγουμένως. Συνεπώς, με αυτό τον τρόπο το συνελικτικό νευρωνικό δίκτυο θα λαμβάνει κάποια νέα, εμπλουτισμένα δεδομένα εισόδου κατά την εκπαίδευση, κάτι που αναμένεται να μας δώσει και μεγαλύτερα ποσοστά ακρίβειας στην πρόβλεψη (Q3) της δευτεροταγούς δομής των πρωτεϊνών.

Η εφαρμογή των φίλτρων Gabor στα δεδομένα εκπαίδευσης θα γίνει με τον εξής τρόπο. Αρχικά, θα χρησιμοποιήσουμε κάποιους συγκεκριμένους συνδυασμούς παραμέτρων (κατεύθυνση, προσανατολισμός, μήκος κύματος κτλ.), για να πάρουμε ως αποτέλεσμα ένα Gabor kernel (πυρήνα), από το πρόγραμμα Python που έχουμε δημιουργήσει (Σχήμα 5.6). Στην συνέχεια, θα εφαρμόσουμε το συγκεκριμένο kernel στα δεδομένα εκπαίδευσης που θα

εξηγηθούν με λεπτομέρεια στο κεφάλαιο 5, πραγματοποιώντας γραμμικό πολλαπλασιασμό του kernel με τα δεδομένα εκπαίδευσης (dot product) που είναι σε μορφή πίνακα (Σχήμα 5.7), και θα πάρουμε ως αποτέλεσμα ένα feature map (πίνακας χαρακτηριστικών) που θα εμπεριέχει μαθηματικά τα χαρακτηριστικά που έχουν εντοπιστεί από την εφαρμογή του συγκεκριμένου φίλτρου (Σχήμα 5.8). Αποφασίσαμε, όπως θα εξηγήσουμε και στη συνέχεια (Ενότητα 5.7), να εφαρμόσουμε ένα σύνολο από συγκεκριμένα φίλτρα Gabor τα οποία θεωρούμε ότι θα εντοπίσουν και τα κυριότερα χαρακτηριστικά των δεδομένων εκπαίδευσης.

Τα αποτελέσματα του συνδυασμού των συνελικτικών νευρωνικών δικτύων με τα φίλτρα Gabor, καθώς και τα συμπεράσματα τα οποία έχουμε λάβει, φαίνονται ξεκάθαρα στο κεφάλαιο με τα πειράματα και τα συμπεράσματα (Κεφάλαιο 6 και Κεφάλαιο 7 αντίστοιχα).

Κεφάλαιο 5

Δεδομένα Εισόδου

5.1	Δεδομένα Εκπαίδευσης και Επαλήθευσης	89
	5.1.1 Γενικά	89
	5.1.2 Μεθοδολογία Επιλογής Δεδομένων Εισόδου	89
5.2	Βάσεις Δεδομένων Πρωτεϊνών και DSSP Κωδικοποίηση	91
5.3	Αρχεία Δεδομένων Πρωτεϊνών	92
5.4	Αρχεία Multiple Sequence Alignment (MSA)	93
5.5	Κατασκευή Δεδομένων Εισόδου με Βάση τα Αρχεία MSA	94
5.6	Χρήση Batches για Εκπαίδευση Νευρωνικών Δικτύων	97
5.7	Εφαρμογή Φίλτρων Gabor στα Αρχεία Εισόδου	99
5.8	Τεχνική Σημαντικότερων Γειτονικών Αμινοξέων	101
5.9	CB513 και PISCES Datasets	104
5.10	Segment Overlap (SOV)	106
5.11	Τεχνική Συνδυασμού CNN με BRNN	107

5.1 Δεδομένα Εκπαίδευσης και Επαλήθευσης

5.1.1 Γενικά

Όπως αναφέραμε αρκετές φορές στα προηγούμενα κεφάλαια, για να είναι σε θέση ένα νευρωνικό δίκτυο να πραγματοποιήσει μια πρόβλεψη ή γενικά να δώσει μια απάντηση σε μια συγκεκριμένη ερώτηση και για ένα συγκεκριμένο πρόβλημα, πρέπει να περάσει μέσα από μια διαδικασία η οποία λέγεται εκπαίδευση. Αντίστοιχα, μετά το πέρας της εκπαίδευσης του νευρωνικού δικτύου, για να είμαστε σε θέση να αποφανθούμε κατά πόσο το νευρωνικό δίκτυο έχει όντως εκπαιδευτεί στο να λύνει ένα συγκεκριμένο πρόβλημα ικανοποιητικά (με μεγάλη σχετικά ακρίβεια), πρέπει να περάσει μέσα από μια διαδικασία η οποία λέγεται επαλήθευση. Οι διαδικασίες της εκπαίδευσης και της επαλήθευσης ενός νευρωνικού δικτύου, είναι άκρως απαραίτητες για την επιτυχή εύρεση λύσης σε ένα πρόβλημα, με χρήση τεχνητών νευρωνικών δικτύων. Για να μπορέσουμε να εκτελέσουμε τις δύο αυτές διαδικασίες πρέπει να χρησιμοποιήσουμε κάποια πειραματικά δεδομένα, τα δεδομένα εκπαίδευσης και επαλήθευσης.

Στόχος της συγκεκριμένης έρευνας, είναι να δημιουργήσουμε και να εκπαιδεύσουμε ένα συνελικτικό νευρωνικό δίκτυο, έτσι ώστε να «λύνει» το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Πιο συγκεκριμένα, θέλουμε να δημιουργήσουμε ένα νευρωνικό δίκτυο, το οποίο δοσμένης της πρωτοταγής δομής (είσοδος) μιας πρωτεΐνης θα μπορεί να προβλέψει την δευτεροταγή της δομή (έξοδος), με μεγάλη ακρίβεια στην πρόβλεψη.

5.1.2 Μεθοδολογία Επιλογής Δεδομένων Εισόδου

Αρχικά, τα δεδομένα εκπαίδευσης τροφοδοτούνται στο δίκτυο κατά την διαδικασία εκπαίδευσης, και αποτελούν συνήθως το μεγαλύτερο ποσοστό του γενικού συνόλου των πειραματικών δεδομένων (~80%), ενώ τα δεδομένα επαλήθευσης με ποσοστό ~20% του γενικού συνόλου των πειραματικών δεδομένων, τροφοδοτούνται στο δίκτυο, μετά το πέρας

μιας εποχής[5] εκπαίδευσης, και χρησιμοποιούνται για να ελέγξουμε την ακρίβεια στην πρόβλεψη του νευρωνικού δικτύου που έχουμε δημιουργήσει. Αξίζει να σημειωθεί, ότι τα πειραματικά δεδομένα επαλήθευσης, δεν χρησιμοποιούνται από το νευρωνικό δίκτυο για διόρθωση των βαρών του κατά τον αλγόριθμο ανάστροφης μετάδοσης σφάλματος.

Η επιλογή των δεδομένων εκπαίδευσης, χρίζει υψίστης σημασίας για την ορθή, αλλά και αποδοτική εκπαίδευση του τεχνητού νευρωνικού δικτύου υπό μελέτη. Ένα νευρωνικό δίκτυο κατά την εκπαίδευση και μέσω των παραδειγμάτων εκπαίδευσης, πρέπει να λαμβάνει υπόψιν την γενική εικόνα του προβλήματος το οποίο καλείται να λύσει. Πρέπει δηλαδή, τα δεδομένα εκπαίδευσης να περιέχουν στοιχεία τα οποία να δίνουν μια γενική εικόνα για τις περιπτώσεις εισόδου που μπορεί να λάβει το δίκτυο αλλά ταυτόχρονα να μην επαναλαμβάνονται πανομοιότυπα παραδείγματα εισόδου (ή αν επαναλαμβάνονται να διατηρείται η αναλογία στις διαφορετικές κατηγορίες δεδομένων). Η κατασκευή ενός αποδοτικού συνόλου δεδομένων εκπαίδευσης, θα οδηγήσει το νευρωνικό δίκτυο στο να εντοπίσει κύρια χαρακτηριστικά στα δεδομένα εισόδου, και να μπορεί να γενικεύσει τους κανόνες που ανάπτυξε εσωτερικά και σε νέα δεδομένα τα οποία δεν έχει ξαναδεί στην είσοδο του. Το ίδιο ισχύει και για την επιλογή των δεδομένων εκπαίδευσης για επίλυση του προβλήματος της πρόβλεψης δευτεροταγούς δομής των πρωτεϊνών. Οι πρωτεΐνες που θα επιλεχθούν για την εκπαίδευση του νευρωνικού δικτύου θα πρέπει να αντικατοπτρίζουν το σύνολο όλων των πρωτεϊνών (όσων γνωρίζουμε). Το ίδιο ακριβώς ισχύει και για τις πρωτεΐνες που θα επιλεχθούν για να συντάξουν το σύνολο των δεδομένων επαλήθευσης του τεχνητού νευρωνικού δικτύου.

Ο κύριος λόγος για τη επιλογή συγκεκριμένων πρωτεϊνών που αντιπροσωπεύουν το μεγαλύτερο σύνολο των πρωτεϊνών, είναι και το ότι, οι διαφορετικές αλληλουχίες των αμινοξέων άρα και οι έλξεις που προκαλούνται μεταξύ τους (Κεφάλαιο 2), έχουν ως αποτέλεσμα να δημιουργείται ένας ασύλληπτα μεγάλος αριθμός πιθανών αναδιπλώσεων. Για αυτόν ακριβώς το λόγο, δεν πρέπει να υπάρχει μεγάλο ποσοστό ομοιότητας μεταξύ των αλληλουχιών που θα επιλεχθούν για να αποτελέσουν τα δεδομένα εκπαίδευσης και επαλήθευσης. Επιπρόσθετα, οι πρωτεΐνες που θα χρησιμοποιηθούν πρέπει να έχουν εξασφαλισμένη την αυθεντικότητά τους.

[5] Μια εποχή εκπαίδευσης αναλογεί σε ένα πλήρες πέρασμα στο νευρωνικό δίκτυο, των δεδομένων εκπαίδευσης.

5.2 Βάσεις Δεδομένων Πρωτεϊνών και DSSP Κωδικοποίηση

Για να μπορέσουμε να συγκεντρώσουμε δεδομένα για εκπαίδευση και επαλήθευση της ακρίβειας στην πρόβλεψη ενός νευρωνικού δικτύου για το PSSP πρόβλημα, θα αποταθούμε σε συγκεκριμένες βάσεις δεδομένων οι οποίες φιλοξενούν δεδομένα και πληροφορίες όπως το όνομα, το μήκος, τον αριθμό των αμινοξέων που την αποτελούν, την πρωτοταγή, τη δευτεροταγή, την τριτοταγή, τον τρόπο εξαγωγής και καθορισμού της τριτοταγούς δομής, την τοποθεσία και ημερομηνία εντοπισμού μιας πρωτεΐνης όπως και άλλες βασικές πληροφορίες, για τις διάφορες πρωτεΐνες που έχουν μελετηθεί. Αξίζει να σημειωθεί ότι μέχρι σήμερα, έχουν ανακαλυφθεί εκατομμύρια πρωτεΐνες, από τις οποίες ωστόσο είμαστε γνώστες της ακριβούς τριτοταγούς δομής, μόνο μερικών χιλιάδων εξ ’αυτών (Αγαθοκλέους, 2009; Χριστοδούλου, 2010). Κάποιες βάσεις οι οποίες περιέχουν τα πιο πάνω στοιχεία, είναι οι εξής: iProClass (Protein Information Resource), η PDBe (Protein Data Bank in Europe), η PDBj (Protein Data Bank in Japan) και η RCSB (RCSB Protein Data Bank).

Η τυποποίηση της δευτεροταγούς δομής των πρωτεϊνών μπορεί να γίνει με αρκετούς τρόπους. Παρόλα αυτά, σε αυτή την έρευνα θα χρησιμοποιήσουμε την πιο διαδεδομένη τυποποίησης της δευτεροταγούς δομής των πρωτεϊνών η οποία λέγεται DSSP (Dictionary for Secondary Structure of Proteins) και χρησιμοποιείται για να περιγράψει τη δευτεροταγή δομή μιας πρωτεΐνης με κωδικοποίηση ενός γράμματος. Ο καθορισμός της δευτεροταγούς δομής μιας πρωτεΐνης, βασίζεται μεταξύ άλλων, στα μοτίβα υδρογονικών δεσμών, όπως αυτοί έχουν προταθεί από τους Pauling et al. το 1951 (προτού έχει μελετηθεί και καθοριστεί οποιαδήποτε δομή πρωτεΐνης). Σύμφωνα με την DSSP κωδικοποίηση, υπάρχουν οκτώ κατηγορίες δευτεροταγούς δομής οι οποίες είναι οι εξής: G: 3-turn helix με ελάχιστο μήκος τριών (3) αμινοξέων, H: 4-turn α helix με ελάχιστο μήκος τεσσάρων (4) αμινοξέων, I: 5-turn helix (π helix) και με ελάχιστο μήκος πέντε (5) αμινοξέων, T: hydrogen bonded turn, E: extended strand με ελάχιστο μήκος δύο (2) αμινοξέων, B: β-bridge, S: Bend, C: Coil (αμινοξέα τα οποία δεν ανήκουν στα πιο πάνω). Παρά την κατηγοριοποίηση αυτή, συνήθως χρησιμοποιούνται μόνο οι τρείς από τις οκτώ αυτές κατηγορίες. Η κατηγορία H (Helix) περιλαμβάνει τις ομάδες H, G και I, η ομάδα E (Extended Strand) περιλαμβάνει τις ομάδες E και B και η τρίτη κατηγορία

περιλαμβάνει όλες τις υπόλοιπες και συμβολίζεται με L/C (Loop/Coil) (Αγαθοκλέους, 2009; Χριστοδούλου, 2010).

Η δυαδική αναπαράσταση των οκτώ αυτών κατηγοριών δευτεροταγούς δομής, την οποία χρησιμοποιήσαμε στην κωδικοποίηση των αμινοξέων και γενικά σε όλη την έρευνα, έχει ως εξής: ***G – 100, H – 011, I – 101, T – 111, E – 010, B – 000, S – 110, C – 001.***

5.3 Αρχεία Δεδομένων Πρωτεϊνών

Τα αρχεία δεδομένων (εκπαίδευσης και επαλήθευσης) τα οποία είχαμε στη διάθεση μας, είχαν εγγραφές με μορφή όπως στο σχήμα 5.1. Στην πρώτη γραμμή βρίσκεται το όνομα της πρωτεΐνης, στη δεύτερη γραμμή η πρωτοταγής δομή της πρωτεΐνης και στην τελευταία γραμμή η τριτοταγής δομή για την συγκεκριμένη πρωτεΐνη.

```
1powB_550-593
LPAEKLRLDSAMSSAADIEAFKQRYEAQDLQPLSTYLKQFGLDD
CCCCCCCCCCCCCCCHHHHHHHHHHCCCCCCCCHHHHHHHCCCCC
```

Σχήμα 5.1: Παράδειγμα πληροφορίας για την πρωτεΐνη 1powB_550-593.

Για την προετοιμασία των δεδομένων εισόδου, έπρεπε να επεξεργαστούμε την πιο πάνω μορφή αρχείων εισόδου (Σχήμα 5.1), και να δημιουργήσουμε δύο νέα αρχεία τύπου CSV[6] (comma-separated-values) ένα αρχείο για τα δεδομένα εκπαίδευσης και ένα αρχείο για τα δεδομένα επαλήθευσης. Για το λόγο αυτό, δημιουργήθηκε ένα πρόγραμμα σε γλώσσα προγραμματισμού Java, το οποίο αποτελείτο από δύο κύρια αρχεία/κλάσεις (Protein.java και Protein_files_creator.java). Το πρώτο αρχείο Protein.java, αναπαριστά ένα αντικείμενο για κάθε πρωτεΐνη με όλες τις πληροφορίες (όνομα, πρωτοταγής και δευτεροταγής δομή) που έχουμε για αυτήν, και το δεύτερο αρχείο Protein_files_creator.java, το οποίο περιλαμβάνει τις κύριες μεθόδους για προετοιμασία των αρχείων δεδομένων εισόδου (εκπαίδευσης και επαλήθευσης) σε CSV μορφή για εκπαίδευση του συνελικτικού νευρωνικού δικτύου. Τα δύο αυτά αρχεία φαίνονται στο παράρτημα Α.

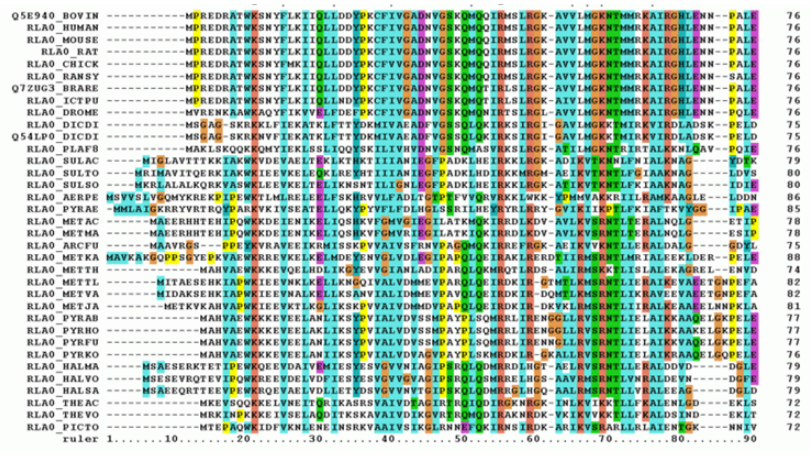
[6] Είδος αρχείου του οποίου, κάθε εγγραφή αποτελείται από τουλάχιστον ένα πεδίο, διαχωρισμένο με κόμμα (,).

5.4 Αρχεία Multiple Sequence Alignment (MSA)

Εκτός από τα αρχεία που περιέχουν τα τρία βασικά στοιχεία για μια πρωτεΐνη (όνομα, πρωτοταγή και δευτεροταγή δομή), έχουμε και κάποια άλλα αρχεία στην διάθεση μας τα οποία ονομάζονται αρχεία ευθυγράμμισης πολλαπλών αλληλουχιών (Multiple Sequence Alignment - MSA).

Με τον όρο ευθυγράμμιση πολλαπλών αλληλουχιών (MSA), εννοούμε την ευθυγράμμιση τριών ή περισσότερων βιολογικών αλληλουχιών οι οποίες γενικά αφορούν ακολουθίες πρωτεϊνών, DNA, ή RNA. Αυτή η μέθοδος είναι εξαιρετικά διαδεδομένη στον τομέα της βιοπληροφορικής. Σε πολλές περιπτώσεις, οι ακολουθίες που επιλέγονται για κατασκευή του MSA αρχείου, υποτίθεται ότι έχουν μια κοινή εξελικτική συσχέτιση και προέρχονται από ένα κοινό πρόγονο. Από το παραγόμενο MSA αρχείο μπορεί να εξαχθεί η ομολογία των ακολουθιών και μπορεί να διεξαχθεί φυλογενετική ανάλυση για να εκτιμηθεί η κοινή εξελικτική προέλευση των ακολουθιών.

Οι οπτικές απεικονίσεις της ευθυγράμμισης των αμινοξέων (Σχήμα 5.2), απεικονίζουν συμβάντα μετάλλαξης, όπως σημειακές μεταλλάξεις (αλλαγές ενός μόνο αμινοξέος ή νουκλεοτιδίων) που εμφανίζονται ως διαφορετικοί χαρακτήρες σε μία στήλη μονής ευθυγράμμισης καθώς επίσης και μεταλλάξεις εισαγωγής ή εξάλειψης που εμφανίζονται ως παύλες σε μία ή περισσότερες από τις ακολουθίες στην ευθυγράμμιση.



Σχήμα 5.2: Οι πρώτες 90 θέσεις μιας ευθυγράμμισης πολλαπλών πρωτεϊνικών αλληλουχιών των καταστάσεων της όξινης ριβοσωματικής πρωτεΐνης Po(L10E) από διάφορους οργανισμούς

(Andrade, M. (2006). Representation of a protein multiple sequence alignment produced with ClustalW. Retrieved from Wikimedia Commons website: https://commons.wikimedia.org/wiki/File:RPLP0_90_ClustalW_aln.gif).

Η πολλαπλή ευθυγράμμιση αλληλουχιών χρησιμοποιείται συχνά για να εκτιμηθεί η διατήρηση αλληλουχίας πρωτεϊνικών περιοχών, τριτοταγών και δευτεροταγών δομών όπως επίσης και μεμονωμένων αμινοξέων ή νουκλεοτιδίων.

Εξαιτίας του ότι, τρείς, ή περισσότερες βιολογικές αλληλουχίες σχετικά ίδιου μήκους, είναι δύσκολο και εξαιρετικά χρονοβόρο να τις ευθυγραμμίσουμε με το χέρι, έχουν δημιουργηθεί υπολογιστικοί αλγορίθμοι για να πραγματοποιήσουν και να αναλύσουν αυτές τις ευθυγραμμίσεις των αμινοξέων. Για την δημιουργία αρχείων MSA απαιτείται πιο έξυπνη και εξειδικευμένη μεθοδολογία από την απλή ευθυγράμμιση ζευγαριών όμοιων αμινοξέων λόγω και της τεράστιας πολυπλοκότητας της φύσης του προβλήματος. Τα περισσότερα προγράμματα για πολλαπλή ευθυγράμμιση αλληλουχιών, χρησιμοποιούν μεθόδους βασισμένες σε ευρετικά αντί για μεθόδους ολικής βελτιστοποίησης, και ο λόγος είναι ότι η εύρεση της βέλτιστης ευθυγράμμισης μεταξύ περισσοτέρων από μερικές αλληλουχίες σχετικά κανονικού μεγέθους είναι υπολογιστικά ακριβό.

Τέλος, τα αρχεία MSA τα οποία δημιουργούνται για κάθε πρωτεΐνη πραγματοποιώντας την πιο πάνω διαδικασία, περιέχουν x-γραμμές (όσα και τα αμινοξέα της πρωτεΐνης) και 20 στήλες, όπου η κάθε στήλη είναι η συχνότητα εμφάνισης του συγκεκριμένου τύπου αμινοξέος (από τα 20 που ξέρουμε) ευθυγραμμισμένων με την αρχική ακολουθία κάθε πρωτεΐνης.

5.5 Κατασκευή Δεδομένων Εισόδου με Βάση τα Αρχεία MSA

Τα συνελικτικά νευρωνικά δίκτυα (CNNs) λαμβάνουν σαν είσοδο δεδομένα σε μορφή πίνακα τριών διαστάσεων, συνεπώς, για να μπορέσουμε να εκπαιδεύσουμε ένα CNN έτσι ώστε να προβλέπει την δευτεροταγή δομή μιας πρωτεΐνης θα πρέπει να παρουσιάσουμε τα δεδομένα εκπαίδευσης στο δίκτυο με την μορφή τρισδιάστατων πινάκων.

Η αξιοποίηση των αρχείων Multiple Sequence Alignment (MSA), τα οποία περιγράψαμε εκτενώς στην προηγούμενη ενότητα (Ενότητα 5.4), φαίνεται να είναι μια από τις πιο

δημοφιλείς τεχνικές, για δημιουργία των αρχείων εκπαίδευσης ενός τεχνητού νευρωνικού δικτύου. Η στοίχιση των αλληλουχιών των αμινοξέων, είναι μια ευρέως γνωστή προσέγγιση στον τομέα της βιοπληροφορικής, η οποία όπως έχουμε ήδη αναφέρει αφορά κυρίως αλληλουχίες DNA, RNA και πρωτεϊνών. Για την κατασκευή των συγκεκριμένων αρχείων, προσπαθούμε να εντοπίσουμε ομοιότητες μεταξύ βιολογικών αλληλουχιών, για να μπορέσουμε να εξάγουμε την ομολογία των αλληλουχιών καθώς επίσης και να μελετήσουμε την εξελικτική σχέση μεταξύ όμοιων βιολογικών αλληλουχιών. Ο λόγος που μελετούμε όμοιες βιολογικές αλληλουχίες, είναι το ότι οι ομοιότητες, ορίζουν συνήθως κάποια βιολογική συσχέτιση, οδηγώντας στην καλύτερη κατανόηση των βιολογικών μηχανισμών.

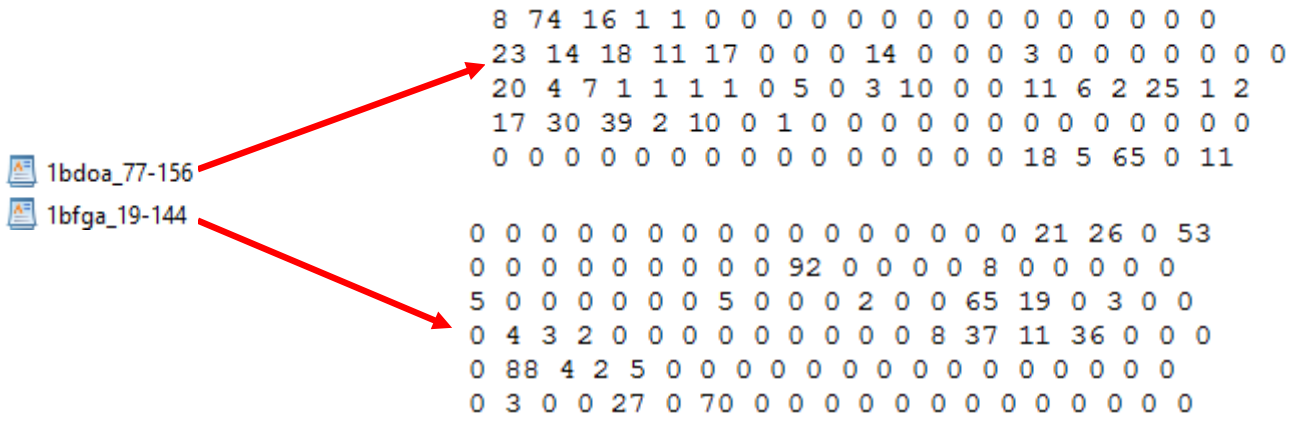
```
0 0 0 100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
29 3 15 0 9 0 0 0 25 3 2 1 0 0 1 7 4 0 1 0
0 0 0 0 0 0 0 0 1 4 1 10 0 2 3 28 3 43 3 2
37 5 58 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
12 4 4 0 8 0 45 0 0 0 0 14 0 0 3 2 0 7 0 0
3 3 31 11 0 0 0 30 0 0 0 15 0 0 7 0 0 0 0 0
2 0 11 7 10 13 47 0 0 0 0 1 0 0 6 2 1 0 0 0
10 1 2 2 0 0 0 1 1 17 23 17 1 0 2 3 5 3 0 11
1 1 2 4 0 0 0 10 1 6 13 34 0 0 1 15 0 9 0 2
5 4 3 1 1 0 1 4 8 6 10 16 0 1 4 3 3 12 6 13
```

Σχήμα 5.3: Παράδειγμα μορφής ενός MSA αρχείου (x - γραμμές, 20 - στήλες). Όπου x, ένας μεταβλητός αριθμός ανάλογα με τον αριθμό των αμινοξέων της πρωτεΐνης που μελετάται.

Για το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, κατά την διαδικασία της στοίχισης πολλαπλών αλληλουχιών, θα πρέπει να γίνει κατάλληλη επιλογή πρωτεϊνών έτσι ώστε το νευρωνικό δίκτυο να παίρνει μια γενική εικόνα για όλο το φάσμα των διαφορετικών πρωτεϊνών που γνωρίζουμε. Αυτό έχει ως αποτέλεσμα, το νευρωνικό δίκτυο να εκτίθεται σε ένα άρτια δομημένο σύνολο δεδομένων, ούτως ώστε να είναι σε θέση να γενικεύει και να παίρνει αποφάσεις για τον καθορισμό της δευτεροταγούς δομής πρωτεϊνών που δεν ξαναείδε στην είσοδο του (άγνωστες προς το δίκτυο), με βάση τη νοημοσύνη που ανέπτυξε κατά την διάρκεια της εκπαίδευσης του για την επίλυση του συγκεκριμένου προβλήματος. Η διαδικασία την στοίχισης πολλαπλών αλληλουχιών στο σύνολο των δεδομένων παρουσιάζει μια μορφή κωδικοποίησης, η οποία μπορεί να γίνει ευκολότερα αντιληπτή από ένα νευρωνικό δίκτυο.

Όπως αναφέραμε προηγουμένως τα CNNs λαμβάνουν σαν δεδομένα εισόδου τρισδιάστατους πίνακες. Για το λόγο αυτό, και έχοντας στη διάθεση μας τα MSA αρχεία (Σχήμα 5.3, Σχήμα 5.4), θα πρέπει να τα ανακατασκευάσουμε με κάποιο τρόπο έτσι ώστε να μπορέσουμε επιτυχώς να εκπαιδεύσουμε ένα CNN δίκτυο με το να οπτικοποιήσουμε κατά κάποιο τρόπο τα MSA αρχεία τα οποία έχουμε στη διάθεση μας.

Για να πετύχουμε τα πιο πάνω και ως αποτέλεσμα να δημιουργήσουμε ένα σύνολο δεδομένων εκπαίδευσης (training set) και ένα σύνολο δεδομένων επαλήθευσης (testing set), επιχειρήσαμε να συμπτύξουμε όλα τα MSA αρχεία που έχουμε σε ένα μόνο αρχείο, προσθέτοντας και την ετικέτα της επιθυμητής εξόδου, στο τέλος της κάθε εγγραφής.



Σχήμα 5.4: Παράδειγμα περιεχομένων δύο αρχείων MSA. Φαίνονται οι τελευταίες 5 γραμμές για το αρχείο 1bdoa_77-156, και οι πρώτες 6 γραμμές για το αρχείο 1bfga_19-144.

Στο πιο κάτω σχήμα (Σχήμα 5.5), βλέπουμε το τελικό αρχείο που δημιουργήθηκε μετά από τη σύμπτυξη των MSA αρχείων. Βλέπουμε ότι έχουν συμπτυχθεί τα δύο MSA αρχεία, έχει αντικατασταθεί το κενό με το κόμμα (,) και ως επίσης έχουμε προσθέσει στο τέλος της κάθε εγγραφής την ετικέτα της επιθυμητής εξόδου του αμινοξέος της δευτεροταγούς δομής της πρωτεΐνης (C:0, E:1, H:2). Σημαντικό είναι να αναφερθεί, ότι η ετικέτα της δευτεροταγούς δομής για κάθε αμινοξύ, εντοπίζεται από τα αρχεία με τα βασικά στοιχεία των πρωτεϊνών (όνομα, πρωτοταγής και δευτεροταγής δομή πρωτεΐνης, Ενότητα 5.4). Συνεπώς, για την προετοιμασία των αρχείων δεδομένων εκπαίδευσης και επαλήθευσης, χρησιμοποιούνται τα αρχεία με τα βασικά στοιχεία των πρωτεϊνών σε συνδυασμό με τα αρχεία MSA, για όλες τις πρωτεΐνες που έχουμε στη διάθεση μας.

```
8,74,16,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
23,14,18,11,17,0,0,0,14,0,0,0,3,0,0,0,0,0,0,0,1
20,4,7,1,1,1,1,0,5,0,3,10,0,0,11,6,2,25,1,2,1
17,30,39,2,10,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,5,65,0,11,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,26,0,53,0
0,0,0,0,0,0,0,0,0,92,0,0,0,0,8,0,0,0,0,0,0
5,0,0,0,0,0,0,5,0,0,0,2,0,0,65,19,0,3,0,0,1
0,4,3,2,0,0,0,0,0,0,0,0,0,0,8,37,11,36,0,0,0,1
0,88,4,2,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
0,3,0,0,27,0,70,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
```

Σχήμα 5.5: Κομμάτι του τελικού αρχείου στο σημείο σύμπτυξης των δύο αρχείων MSA 1bdoa_77-156 και 1bfga_19-144.

Εφαρμόζοντας τα πιο πάνω σε όλα τα αρχεία MSA που έχουμε στη διάθεση μας, έχουμε καταφέρει να δημιουργήσουμε ένα αρχείο το οποίο περιέχει όλα τα MSA αρχεία με την ετικέτα του κάθε αμινοξέος της δευτεροταγούς δομής, για κάθε εγγραφή. Πράττουμε τα πιο πάνω για τα δεδομένα εκπαίδευσης ως επίσης και για τα δεδομένα επαλήθευσης. Έχουμε καταφέρει έτσι, να οπτικοποιήσουμε τα MSA αρχεία έτσι ώστε να μπορούν να ληφθούν ορθά και να γίνουν αντιληπτά, από ένα συνελικτικό νευρωνικό δίκτυο. Τέλος, με την διαδικασία που αναφέραμε προηγουμένως, έχουμε καταφέρει να παρουσιάσουμε τα MSA αρχεία στο CNN με τη μορφή ενός 2D πίνακα.

Το συνελικτικό νευρωνικό δίκτυο κατά την διάρκεια της εκπαίδευσης και επαλήθευσης, θα λαμβάνει μία-μία εγγραφή του τελικού αρχείου, και θα την αναπροσαρμόζει έτσι ώστε να περάσει μέσα από τα κρυφά επίπεδα και να εξαχθεί τέλος η δευτεροταγής δομή του συγκεκριμένου αμινοξέος, από το επίπεδο εξόδου.

5.6 Χρήση Batches για Εκπαίδευση Νευρωνικών Δικτύων

Όπως αναφέραμε αρκετές φορές προηγουμένως (Ενότητα 3.3), κατά την διαδικασία της εκπαίδευσης, αρχικά δίδεται η είσοδος στο νευρωνικό δίκτυο και πραγματοποιείται το εμπρόσθιο πέρασμα έτσι ώστε να υπολογιστεί η πραγματική έξοδος του νευρωνικού δικτύου. Αφού εκτελεστεί το εμπρόσθιο πέρασμα και υπολογιστεί επιτυχώς η πραγματική έξοδος του δικτύου, στη συνέχεια εκτελείται ο αλγόριθμος ανάστροφης μετάδοσης σφάλματος (Ενότητα 3.3.4.6), για αλλαγή των βαρών του νευρωνικού δικτύου. Η διαδικασία εκπαίδευσης, η οποία

εκτελεί τον αλγόριθμο ανάστροφης μετάδοσης σφάλματος, για αλλαγή των συναπτικών βαρών του νευρωνικού δικτύου αμέσως μετά τον υπολογισμό της πραγματικής εξόδου του δικτύου για κάθε παράδειγμα εισόδου εκπαίδευσης, ονομάζεται on-line εκπαίδευση. Όλες οι τεχνικές και αλγόριθμοι εκπαίδευσης που αναφέραμε σε αυτήν την διατριβή μέχρι και αυτό το σημείο, χρησιμοποιούσαν την τεχνική της on-line εκπαίδευσης.

Όπως αναφέραμε και προηγουμένως (Ενότητα 3.3.4.6), υπάρχει ακόμα μια διαδικασία εκπαίδευσης που ονομάζεται off-line εκπαίδευση. Η off-line εκπαίδευση, εκτελεί τον αλγόριθμο ανάστροφης μετάδοσης σφάλματος, μετά από τον υπολογισμό των πραγματικών τιμών εξόδου του νευρωνικού δικτύου για ένα συγκεκριμένο αριθμό παραδειγμάτων, λαμβάνοντας έτσι υπόψιν το μέσο όρο του σφάλματος των πραγματικών τιμών εξόδου του δικτύου. Η τεχνική της off-line εκπαίδευσης ενός νευρωνικού δικτύου, πραγματοποιείται με τη χρήση των batches (παρτίδων). Τα batches δεδομένων είναι μια έννοια η οποία χρησιμοποιείται για να περιγράψει μια ομάδα παραδειγμάτων εισόδου. Πιο συγκεκριμένα, χωρίζουμε τα δεδομένα εισόδου εκπαίδευσης σε κάποιες ομάδες τις οποίες αξιοποιούμε κατά την διαδικασία της εκπαίδευσης. Κατά την διαδικασία της εκπαίδευσης, πραγματοποιείται το εμπρόσθιο πέρασμα στο νευρωνικό δίκτυο για όλα τα παραδείγματα εκπαίδευσης ενός batch, και υπολογίζεται τέλος, ο μέσος όρος του σφάλματος του δικτύου για τα παραδείγματα εισόδου εκπαίδευσης του συγκεκριμένου batch δεδομένων. Στη συνέχεια εκτελούμε τον αλγόριθμο ανάστροφης μετάδοσης σφάλματος, για αλλαγή των συναπτικών βαρών του δικτύου χρησιμοποιώντας το μέσο όρο του σφάλματος για το συγκεκριμένο batch. Αυτή η διαδικασία επαναλαμβάνεται για όλα τα batches δεδομένων εκπαίδευσης και για κάθε εποχή ξεχωριστά. Η χρήση των batches δεδομένων επιταχύνει σημαντικά την διαδικασία εκπαίδευσης αφού ο αλγόριθμος ανάστροφης μετάδοσης σφάλματος με χρήση των batches, εκτελείται πολύ λιγότερες φορές (εποχές $\times$ batches) σε σχέση με την on-line τεχνική εκπαίδευσης (εποχές $\times$ αριθμός παραδειγμάτων εκπαίδευσης). Επιπρόσθετα, ένα ακόμα πλεονέκτημα της χρήσης των batches, είναι το ότι οι αλλαγές που γίνονται στα βάρη είναι πολύ πιο ακριβής σε σχέση με την on-line τεχνική εκπαίδευσης, και ο λόγος είναι ότι κατά τον αλγόριθμο ανάστροφης μετάδοσης σφάλματος, χρησιμοποιείται ο μέσος όρος των επιμέρους σφαλμάτων του νευρωνικού δικτύου, για τις αλλαγές που πραγματοποιούνται στα συναπτικά

βάρη του νευρωνικού δικτύου. Με αυτόν τον τρόπο, το νευρωνικό δίκτυο μπορεί να φτάσει σε πιο ψηλά επίπεδα επιτυχίας στις προβλέψεις του αφού οι αλλαγές που πραγματοποιούνται στα συναπτικά βάρη του δικτύου, λαμβάνουν υπόψιν όλα τα παραδείγματα εκπαίδευσης ενός batch. Η τεχνική χρήσης των batches δεδομένων εισόδου, μπορεί να χρησιμοποιηθεί στην εκπαίδευση όλων των τύπων των νευρωνικών δικτύων για ευνόητους λόγους.

5.7 Εφαρμογή Φίλτρων Gabor στα Αρχεία Εισόδου

Στην ενότητα 4.3 του κεφαλαίου 4, αναφέραμε εκτενώς το πώς θα γίνει η σύνδεση των φίλτρων Gabor με τα συνελικτικά νευρωνικά δίκτυα, για εξαγωγή χαρακτηριστικών από το σύνολο δεδομένων εκπαίδευσης. Συγκεκριμένα, αναφέραμε ότι αρχικά, θα εφαρμοστούν κάποια συγκεκριμένα φίλτρα Gabor (Σχήμα 5.6) στα δεδομένα εκπαίδευσης και θα έχουμε ως αποτέλεσμα κάποια άλλα νέα/εμπλουτισμένα δεδομένα εκπαίδευσης με τα διάφορα χαρακτηριστικά τα οποία έχουν εντοπιστεί από τα φίλτρα Gabor, τα οποία έχουμε εφαρμόσει προηγουμένως πάνω στα δεδομένα εκπαίδευσης. Στην συνέχεια, αφού περιγράψαμε την δομή και τον λόγο ύπαρξης των αρχείων MSA (Ενότητα 4.4), προχωρήσαμε με το να εξηγήσουμε πώς θα κατασκευάσουμε τα αρχεία με τα δεδομένα εκπαίδευσης χρησιμοποιώντας τα αρχεία MSA (Ενότητα 5.5).

Για τον εμπλουτισμό των αρχείων εκπαίδευσης, και με απώτερο σκοπό να δώσουμε μια καλύτερη εικόνα στο νευρωνικό δίκτυο σχετικά με το πρόβλημα που έχουμε να λύσουμε, θα προχωρήσουμε με το να εφαρμόσουμε κάποια φίλτρα Gabor σε συγκεκριμένες κατευθύνσεις και προσανατολισμούς πάνω στα αρχεία δεδομένων εκπαίδευσης που περιγράψαμε στην ενότητα 5.5, έτσι ώστε να πάρουμε κάποια νέα δεδομένα εκπαίδευσης, εμπλουτισμένης μορφής, ευελπιστώντας σε καλύτερα ποσοστά επιτυχίας στις προβλέψεις του δικτύου.

1. gabor_fn(0.4, 0, 2, 0, 1)
2. gabor_fn(0.4, 30, 2, 0, 1)
3. gabor_fn(0.4, 60, 2, 0, 1)
4. gabor_fn(0.4, 90, 2, 0, 1)
5. gabor_fn(0.5, 120, 2, 0, 1)
6. gabor_fn(0.5, 150, 2, 0, 1)

7. gabor_fn(0.5, 180, 2, 0, 1)

Σχήμα 5.6: Παράδειγμα επτά (7) φίλτρων Gabor που θα εφαρμοστούν στα δεδομένα εκπαίδευσης.

[[1.38879439e-11 1.63737713e-07 3.72665317e-06 1.63737713e-07 1.38879439e-11]

[-1.63737713e-07 -1.93045414e-03 -4.39369336e-02 -1.93045414e-03 -1.63737713e-07]

[3.72665317e-06 4.39369336e-02 1.00000000e+00 4.39369336e-02 3.72665317e-06]

[-1.63737713e-07 -1.93045414e-03 -4.39369336e-02 -1.93045414e-03 -1.63737713e-07]

[1.38879439e-11 1.63737713e-07 3.72665317e-06 1.63737713e-07 1.38879439e-11]]

Σχήμα 5.7: Δομή και περιεχόμενα του πρώτου φίλτρου Gabor.

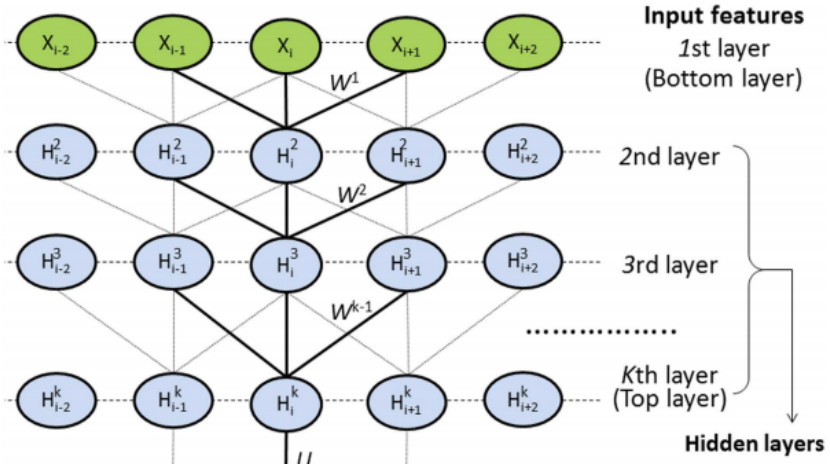
-0.0811834884283,-0.0236026225263,-0.031086993519,-0.027150734982,-0.000104500868014,0.0,-5.97147817222e-05,-0.0158196776249,-0.0966000912766,-0.000403074776625,-0.00778294490131,-0.01547631763,-7.46434771528e-05,-0.00387654375523,0.0211886430327,6.50867792851,6.61938250349,28.6572257827,0.158432680386,12.3497207338,011.3588810166,3.30238803661,4.3495723993,3.7988262656,0.0146213589596,0.0,0.00828041878548,2.19386649949,13.4501756808,0.0517886204415,0.961362161376,2.12627795706,0.0102796121786,0.538514527679,0.538596772013,0.0234153464392,0.0196781180644,0.564732001448,0.020871907312,10.8274240083,0
0.0333645603981,-0.000763919767657,0.0142271078961,0.0124647760155,4.79758619454e-05,-2.98573908611e-05,0.00194186293371,
2.53550228325,9.18327514114,0.640690801459,17.8178796206,5.4750465257,0.0229957609996,0.544170774622,0.544140917232,-0.00358363138,0.566217593697,9.7224828527,3.28987994124,2.73791626744,0
0.54269205881,1.61911913941,0.00625136800155,-5.97147817222e-05,-0.0154464602392,0.00408795895874,1.18555970368,29.1821384264,8.24172374063,0.594487692278,5.43475273502,0.544270663285,0.00171629534294,-0.079314213067,0.00558401693425,1.07140829819,1.62946240331,2.69321856773,-0.0114994402141,1.57560038945,1
0.474600767458,0.528050109621,-0.0211797932729

Σχήμα 5.8: Μέρος αρχείου MSA μιας πρωτεΐνης, μετά την εφαρμογή ενός φίλτρου Gabor.

Τέλος, τα αρχεία κώδικα για την δημιουργία αλλά και εφαρμογή των φίλτρων Gabor, φαίνονται στο παράρτημα Ε.

5.8 Τεχνική Σημαντικότερων Γειτονικών Αμινοξέων

Εξηγήσαμε εκτενώς τη σημασία της ακολουθίας των αμινοξέων που απαρτίζουν μια πρωτεΐνη. Όπως αναφέραμε, η σειρά με την οποία συντάσσονται τα αμινοξέα παίζει καθοριστικό ρόλο στις μεταξύ τους αλληλεπιδράσεις και συνεπώς στη δημιουργία των τοπικών αναδιπλώσεων (δευτεροταγής δομή) μιας πολυπεπτιδικής αλυσίδας, μιας πρωτεΐνης. Η δευτεροταγής δομή ενός αμινοξέος επηρεάζεται άμεσα και σε μεγάλο βαθμό από τα αμέσως γειτονικά αμινοξέα, επηρεάζεται λιγότερο από τα δεύτερα γειτονικά αμινοξέα, επηρεάζεται ακόμα λιγότερα από τα τρίτα γειτονικά αμινοξέα και πάει λέγοντας. Αυτή η ιδέα εκτός από την φύση του προβλήματος, προκύπτει και από την έρευνα των Wang et al. (2016), στην οποία φαίνεται ότι ένας νευρώνας σε ένα οποιοδήποτε κρυφό επίπεδο είναι συνδεδεμένος μόνο με τους τρείς αντίστοιχους νευρώνες στο προηγούμενο επίπεδο (Σχήμα 5.9). Εξηγούν στην συνέχεια ότι η δευτεροταγής δομή ενός αμινοξέος προκύπτει σε μεγάλο βαθμό από την αλληλεπίδραση του με τα αμέσως γειτονικά του αμινοξέα.



Σχήμα 5.9: Αρχιτεκτονική και συνδεσμολογία κρυφών επιπέδων στην έρευνα των Wang et al., (2016).

Ως αποτέλεσμα των όσων αναφέραμε πιο πάνω, αποφασίσαμε όπως προχωρήσουμε με το να τροποποιήσουμε τα δεδομένα εκπαίδευσης του νευρωνικού δικτύου (Σχήμα 5.10), έτσι ώστε να συμπεριλάβουμε σε κάθε εγγραφή του αρχείου εκπαίδευσης, εκτός από τα στοιχεία και το επιθυμητό αποτέλεσμα του συγκεκριμένου αμινοξέος (εγγραφή MSA ενός αμινοξέος και η ετικέτα του επιθυμητού αποτελέσματος στο τέλος) στην συγκεκριμένη θέση, και τα στοιχεία

ενός αριθμού γειτονικών αμινοξέων (Σχήμα 5.11). Συνεπώς, θα έχουμε σε κάθε εγγραφή, τις MSA εγγραφές των αριστερά γειτονικών αμινοξέων (αν υπάρχουν), την MSA εγγραφή του εκάστοτε αμινοξέος που μελετούμε, τις MSA εγγραφές των δεξιά γειτονικών αμινοξέων (αν υπάρχουν), και στο τέλος την ετικέτα επιθυμητού αποτελέσματος για το μεσαίο αμινοξύ. Αξίζει να σημειωθεί, ότι ο αριθμός των γειτονικών αμινοξέων των οποίων τα στοιχεία τους (εγγραφές MSA), θα συμπεριληφθούν στην κάθε εγγραφή, είναι μεταβλητός.

1. 0,0,0,0,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,0,0
2. 0,0,0,0,0,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,0
3. 0,0,0,0,0,0,0,21,49,0,0,31,0,0,0,0,0,0,0,0,1
4. 42,0,0,0,28,0,0,0,31,0,0,0,0,0,0,0,0,0,0,0,1
5. 0,0,0,0,0,0,0,0,0,0,49,51,0,0,0,0,0,0,0,0,1
6. 100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
7. 0,0,0,0,0,0,0,0,0,0,49,51,0,0,0,0,0,0,0,0,1

Σχήμα 5.10: Παράδειγμα πρώτων επτά (7) εγγραφών του κανονικού αρχείου δεδομένων/παραδειγμάτων εκπαίδευσης.

1. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,100,0,0,0,0,0,0,0,0,0,0,0
2. 0,0,0,0,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,21,49,0,0,31,0,0,0,0,0,0,0,0,0
3. 0,0,0,0,0,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,49,0,0,31,0,0,0,0,0,0,0,0,42,0,0,
0,28,0,0,0,31,0,0,0,0,0,0,0,0,0,0,0,1
4. 0,0,0,0,0,0,0,21,49,0,0,31,0,0,0,0,0,0,0,0,42,0,0,0,28,0,0,0,31,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,49,51,0,0,0,0,0,0,0,0,1
5. 42,0,0,0,28,0,0,0,31,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,49,51,0,0,0,0,0,0,0,0,100,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
6. 0,0,0,0,0,0,0,0,0,0,49,51,0,0,0,0,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,49,51,0,0,0,0,0,0,0,0,1
7. 100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,49,51,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,100,0,0,0,0,0,0,0,0,0,0,1

Σχήμα 5.11: Παράδειγμα πρώτων επτά (7) εγγραφών του νέου αρχείου εκπαίδευσης συμπεριλαμβάνοντας σε κάθε εγγραφή τις πληροφορίες των δύο αμέσως γειτονικών αμινοξέων, συμπεριλαμβανομένου και του αμινοξέος που μελετούμε (σύνολο 3 αμινοξέα και $20 \times 3 + 1$ (ετικέτα) αριθμοί σε κάθε εγγραφή).

Πρέπει να τονίσουμε ότι μόλις το δίκτυο λάβει το παράδειγμα εισόδου αναπροσαρμόζει τις διαστάσεις του, δημιουργώντας ένα πίνακα δυσδιάστατης μορφής (2D) στον οποίο η κάθε γραμμή είναι το MSA profile vector για το κάθε αμινοξύ, και με επιθυμητό αποτέλεσμα τον τελευταίο αριθμό ο οποίος συμβολίζει την κατηγορία δευτεροταγούς δομής του μεσαίου αμινοξέος. Το αποτέλεσμα αυτής της αναπροσαρμογής θα τροφοδοτηθεί στη συνέχεια στο δίκτυο ως παράδειγμα εισόδου. Για παράδειγμα, η τέταρτη (4) γραμμή του σχήματος 5.11 θα αναπροσαρμοστεί όπως φαίνεται στο σχήμα 5.12, και μετά θα δοθεί σαν είσοδος στο CNN.

Παράδειγμα Εισόδου Μετά Την Αναπροσαρμογή																				**Επιθυμητό Αποτέλεσμα**
0	**0**	**0**	**0**	**0**	**0**	**0**	**21**	**49**	**0**	**0**	**31**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	1
42	**0**	**0**	**0**	**28**	**0**	**0**	**0**	**31**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**49**	**51**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	

Σχήμα 5.12: Παράδειγμα αναπροσαρμογής του διανύσματος εισόδου της γραμμής 4 του σχήματος 5.11, έτοιμο να τροφοδοτηθεί στο CNN.

Για να ανακατασκευάσουμε τα αρχεία εκπαίδευσης και επαλήθευσης όπως αναφέραμε πιο πάνω, θα δημιουργήσουμε δύο συναρτήσεις σε γλώσσα προγραμματισμού Java και οι οποίες φαίνονται στο παράρτημα Α. Οι δύο αυτές συναρτήσεις με ονόματα create_csv_files_test_Xneighboring_technique(int windowNum) και create_csv_files_train_Xneighboring_technique(int windowNum), δημιουργούν κάποια νέα αρχεία εκπαίδευσης και επαλήθευσης στα οποία η κάθε εγγραφή περιέχει τα MSA records των windowNum DIV 2 αριστερά γειτονικών αμινοξέων (αν υπάρχουν), το MSA record για το αμινοξύ που μελετούμε, τα MSA records των windowNum DIV 2 δεξιά αμινοξέων (αν υπάρχουν), καθώς επίσης και την ετικέτα του αμινοξέος που μελετούμε στο τέλος της κάθε εγγραφής. Το μέγεθος των αρχείων εκπαίδευσης και επαλήθευσης, εξαρτάται αποκλειστικά από το εύρος θέασης (windowNum) που θέτει ο χρήστης σε κάθε εκτέλεση αυτών των συναρτήσεων. Σημαντικό είναι να αναφέρουμε, ότι το εύρος θέασης που θα δίνεται πρέπει να είναι πάντα περιττός αριθμός λόγω του ότι στον αριθμό αυτό, συμπεριλαμβάνεται και το εκάστοτε (κεντρικό) αμινοξύ που μελετούμε.

Η τεχνική σημαντικότερων γειτονικών αμινοξέων φαίνεται ότι όντως δίνει κάποια πολύ καλά αποτελέσματα σχετικά με το ποσοστό επιτυχίας Q3 του συνελικτικού νευρωνικού δικτύου. Πιο συγκεκριμένα, με τη χρήση της συγκεκριμένης τεχνικής κατά την εκπαίδευση, το συνελικτικό νευρωνικό δίκτυο είναι σε θέση να πραγματοποιήσει πιο ακριβείς προβλέψεις, αφού μπορεί να λάβει υπόψιν τα γειτονικά αμινοξέα άρα και τις αλληλεπιδράσεις που προκαλούνται μεταξύ τους. Συμπερασματικά, η χρήση της τεχνικής σημαντικότερων γειτονικών αμινοξέων, επιτρέπει στο νευρωνικό δίκτυο να λάβει υπόψιν περισσότερες παραμέτρους και πληροφορίες (τις εγγραφές MSA των γειτονικών αμινοξέων), πριν να προβεί σε πρόβλεψη της δευτεροταγούς δομής του μεσαίου αμινοξέος. Τα ποσοστά επιτυχίας και τα γενικά συμπεράσματα της χρήσης της συγκεκριμένης τεχνικής αναφέρονται στη συνέχεια (Κεφάλαιο 6 και Κεφάλαιο 7).

5.9 CB513 και PISCES Datasets

Όπως αναφέραμε εκτενώς σε αυτή την έρευνα, τα τεχνητά νευρωνικά δίκτυα περνούν μέσα από μια διαδικασία η οποία λέγεται εκπαίδευση και κατά την οποία το νευρωνικό δίκτυο εξειδικεύεται, έτσι ώστε να λύνει ένα συγκεκριμένο πρόβλημα. Στην συνέχεια, επισημάναμε ότι για να αποφανθούμε κατά πόσο ένα νευρωνικό δίκτυο έχει εκπαιδευτεί επιτυχώς στο να λύνει ένα συγκεκριμένο πρόβλημα, δεν αρκεί μόνο να μειώνεται το σφάλμα εκπαίδευσης (προβλέψεις για παραδείγματα εκπαίδευσης) σε κάθε εποχή (Least Mean Square Error - LMSE), αλλά πρέπει να μειώνεται και το σφάλμα επαλήθευσης (προβλέψεις για παραδείγματα επαλήθευσης).

Ως αρχεία εκπαίδευσης και επαλήθευσης του συνελικτικού νευρωνικού δικτύου που μελετούμε σε αυτή την έρευνα, επιλέξαμε τα ευρέως γνωστά datasets με πρωτεϊνικές ακολουθίες CB513 (Cuff and Barton, 1999) και PISCES (Wang et al., 2003). Ο λόγος που επιλέξαμε αυτά τα δύο datasets είναι ότι χρησιμοποιούνται αρκετά συχνά σαν benchmarks για το PSSP πρόβλημα οπότε θα μπορούμε να έχουμε μια γενική εικόνα και ένα αντικειμενικό μέτρο σύγκρισης για τη λύση που θα προσφέρει το συνελικτικό νευρωνικό δίκτυο μαζί με άλλες, διαφορετικής προσέγγισης λύσεις, για το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Αξίζει να σημειωθεί, ότι για την εκπαίδευση των νευρωνικών μας

δικτύων, θα πραγματοποιούμε ανακάτεμα των πρωτεϊνών, έτσι ώστε να αποτρέψουμε το δίκτυο μας από την απομνημόνευση της σειράς των πρωτεϊνών αλλά και για να πάρουμε καλύτερα ποσοστά επιτυχίας στην πρόβλεψη.

Αρχικά, επικεντρωθήκαμε στη χρήση του μικρότερου από τα δύο datasets, το οποίο είναι το CB513. Το dataset CB513 περιέχει 513 μη ομοιογενείς πρωτεϊνικές ακολουθίες (από τις οποίες έχουν εξερεθεί οκτώ (8) πρωτεΐνες οι οποίες φαίνονται στο παράρτημα Θ, λόγω του ότι τα MSA αρχεία τους ήταν μηδενισμένα), και είναι ιδανικό για να εντοπίσουμε αρχικά κατά πόσο το νευρωνικό δίκτυο φαίνεται να μαθαίνει και να μπορεί να επιλύει σε κάποιο βαθμό το πρόβλημα που μελετούμε (PSSP). Στη συνέχεια, αποφασίσαμε να προχωρήσουμε με το να χρησιμοποιήσουμε το μεγαλύτερο σε μέγεθος dataset το οποίο είναι το PISCES. Το PISCES dataset, περιέχει ένα μεγαλύτερο αριθμό πρωτεϊνικών ακολουθιών της τάξης των οκτώ χιλιάδων πεντακοσίων (8500) πρωτεϊνικών ακολουθιών, από τις οποίες έχουν εξαιρεθεί περίπου τριακόσιες σαράντα ένα (341) πρωτεΐνες (οι οποίες φαίνονται στο παράρτημα Ι), λόγω του ότι τα MSA αρχεία τους ήταν κατεστραμμένα (corrupted) ή μηδενισμένα (zeroed). Ο κύριος λόγος που προχωρήσαμε στην χρήση του μεγαλύτερου σετ δεδομένων, είναι το ότι πιστεύουμε, ότι τροφοδοτώντας το νευρωνικό δίκτυο με ένα μεγαλύτερο αριθμό παραδειγμάτων πρωτεϊνικών ακολουθιών (με κάτω από 25% ομοιογένεια), μετά το πέρας της εκπαίδευσης θα είναι σε θέση να γενικεύσει και να πραγματοποιήσει πιο ακριβείς προβλέψεις αφού θα έχει μια πιο γενική εικόνα για το πρόβλημα.

Τέλος, το PISCES dataset δεν είχε την απαιτούμενη μορφή έτσι ώστε να προβούμε σε εκπαίδευση των νευρωνικών δικτύων της ομάδας του Πανεπιστημίου Κύπρου. Για να επιλύσουμε αυτό το πρόβλημα κατασκευάσαμε ένα Java project με το όνομα PISCES_convert (παράρτημα Δ) και το οποίο περιέχει ένα αρχείο κώδικα με το όνομα Train_Test_FilesCreator.java, το οποίο κατασκευάζει τα αρχεία εκπαίδευσης και επαλήθευσης με την μορφή που χρησιμοποιούμε (Όνομα πρωτεΐνης, πρωτοταγής δομή, δευτεροταγής δομή), βασισμένο στα ακατέργαστα δεδομένα που είχαμε.

Τα νέα έτοιμα προς χρήση αρχεία δόθηκαν σε όλη την ερευνητική ομάδα του Πανεπιστημίου Κύπρου έτσι ώστε να χρησιμοποιηθούν για εκπαίδευση των δικών τους νευρωνικών δικτύων

για εκπλήρωση της έρευνας τους. Τέλος, τα αρχεία αυτά θα αποθηκευτούν σε ένα server, έτσι ώστε να χρησιμοποιούνται σε μεταγενέστερο στάδιο από οποιονδήποτε ερευνητή θέλει να ασχοληθεί με το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

5.10 Segment Overlap (SOV)

Μέχρι αυτό το σημείο, για την αξιολόγηση της ακρίβειας πρόβλεψης της δευτεροταγούς δομής μιας πρωτεΐνης, αναφερόμασταν στην μέθοδο Q3 (Εξίσωση 1.1). Το πρόβλημα που προκύπτει με την συγκεκριμένη μέθοδο, είναι το ότι το δίκτυο μας μπορεί να προβλέπει με μεγάλη επιτυχία μια κατηγορία δευτεροταγούς δομής, αλλά να αποτυγχάνει παταγωδώς να προβλέψει τις άλλες δύο κατηγορίες. Πιο συγκεκριμένα, το νευρωνικό δίκτυο μπορεί να αποτυγχάνει στο να γενικεύει και να διαχωρίσει σωστά, τα δεδομένα που δίνονται στην είσοδο του.

Η μέθοδος SOV είναι βασισμένη στο μέσο όρο επικάλυψης των ακολουθιών της πραγματικής και της προβλεπόμενης δευτεροταγούς δομής, αντί στην κατά αντιστοιχία αμινοξέων ακρίβεια (Q3 ακρίβεια). Πιο συγκεκριμένα, η μέθοδος SOV είναι μια αυστηρή σχετικά αξιολόγηση της προβλεπόμενης δευτεροταγούς δομής, λόγω του ότι δεν συγκρίνει αμινοξικά κατάλοιπα ένα προς ένα ανάμεσα στην πραγματική και την προβλεπόμενη δευτεροταγή δομή μιας πρωτεΐνης, αλλά αντ' αυτού συγκρίνει διαστήματα από αμινοξικά κατάλοιπα τα οποία επικαλύπτονται στις ακολουθίες της προβλεπόμενης και της πραγματικής ακολουθίας αντίστοιχα. Η επικάλυψη διαστημάτων αμινοξικών καταλοίπων ανάμεσα στις δύο ακολουθίες, χρίζει τεράστιας σημασίας αφού είναι και αυτά τα οποία καθορίζουν την «δομή» της πρωτεΐνης. Ως εκ τούτου, η μέθοδος βαθμολόγησης SOV είναι αρκετά σημαντική για να αξιολογήσουμε με πιο μεγάλη ακρίβεια τα αποτελέσματα της εκπαίδευσης του νευρωνικού μας δικτύου. Για παράδειγμα, ο ορισμός της SOV βαθμολογίας για τους α – έλικες είναι όπως στην εξίσωση 5.1.

$$SOV_\alpha = \frac{1}{N_\alpha} \sum_{S_\alpha} \frac{minOV(s_1, s_2) + \delta(s_1, s_2)}{maxOV(s_1, s_2)}.$$

Εξίσωση 5.1: Εξίσωση υπολογισμού SOV βαθμολογίας για τους α – έλικες. Όπου $s1$ και $s2$ είναι τα πραγματικά και τα προβλεπόμενα τμήματα δευτεροταγούς δομής για τους α-έλικες, το s_a είναι ο αριθμός των ζευγών διαστημάτων (s_1, s_2), όπου τα s_1 και s_2 έχουν τουλάχιστον

ένα κοινό κατάλοιπο α-έλικας, το minOV(s_1, s_2) είναι το μήκος της επικάλυψης των s_1 και s_2 και το maxOV(s_1, s_2) είναι το μήκος της συνολικής έκτασης για το οποίο ένα από τα s_1 και s_2 έχουν ένα κατάλοιπο τύπου α-έλικα. Το Na είναι ο συνολικός αριθμός καταλοίπων τύπου α-έλικα. Ο ορισμός του δ(s_1, s_2) είναι όπως πιο κάτω (Zemla et al., 1999).

$$\delta(s_1, s_2) = min \left\{ \begin{array}{c} maxOV(s_1, s_2) - minOV(s_1, s_2) \\ minOV(s_1, s_2) \\ int(0.5 \times len(s_1)) \\ int(0.5 \times len(s_2)). \end{array} \right\}$$

Εξίσωση 5.2: Εξίσωση υπολογισμού όρου δ(s_1, s_2). Όπου len(s_1) είναι ο αριθμός των αμινοξικών καταλοίπων στο τμήμα s_1 (Zemla et al., 1999).

5.11 Τεχνική Συνδυασμού CNN με BRNN

Όπως αναφέραμε και στην αρχή της έρευνας, μια από τις καλύτερες προσεγγίσεις στην επίλυση του προβλήματος της πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, ήταν το τεχνητό νευρωνικό δίκτυο αμφίδρομης ανάδρασης (Bidirectional Recurrent Neural Networks - BRNN) του Baldi (Baldi et al., 1999). Αποφασίσαμε λοιπόν να συνδέσουμε κατά κάποιο τρόπο το συνελικτικό νευρωνικό δίκτυο που δημιουργήσαμε στην συγκεκριμένη έρευνα με ένα νευρωνικό δίκτυο αμφίδρομης ανάδρασης, έτσι ώστε να μπορέσουμε να πετύχουμε υψηλότερα αποτελέσματα στην πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών. Πιο συγκεκριμένα, αποφασίσαμε να εξάγουμε κάποια διανύσματα χαρακτηριστικών (feature vectors), από το τελευταίο κρυφό επίπεδο του συνελικτικού νευρωνικού δικτύου και στη συνέχεια, να τα δώσουμε σαν είσοδο, στο νευρωνικό δίκτυο αμφίδρομης ανάδρασης. Με αυτόν τον τρόπο, το νευρωνικό δίκτυο αμφίδρομης ανάδρασης θα λάβει περισσότερη πληροφορία στην είσοδο του σχετικά με διάφορα χαρακτηριστικά τα οποία έχουν εντοπιστεί από το συνελικτικό νευρωνικό δίκτυο που χρησιμοποιήσαμε προηγουμένως.

Πραγματοποιώντας τα πιο πάνω, θα πραγματοποιήσουμε ένα πρωτοποριακό συνδυασμό αρχιτεκτονικών τεχνικών νευρωνικών δικτύων, στον οποίο η μορφή των δεδομένων εκπαίδευσης που εξάγονται από το συνελικτικό νευρωνικό δίκτυο και εισάγονται στο νευρωνικό δίκτυο αμφίδρομης ανάδρασης χρίζει υψίστης σημασίας.

Κεφάλαιο 6

Πειράματα και Ανάλυση Αποτελεσμάτων

6.1 Βιβλιοθήκη DeepLearning4j – DL4J 111

6.1.1 Γενικά 111

6.1.2 Προσαρμογή Βιβλιοθήκης για το PSSP Πρόβλημα 111

6.2 Πειράματα με Χρήση Αρχείων MSA 113

6.2.1 Περιγραφή 113

6.2.2 Πειράματα με Χρήση Διαφορετικού Αριθμού Κρυφών Επιπέδων 114

6.2.3 Πειράματα με Χρήση Διαφορετικών Μεγεθών Kernel 118

6.2.4 Πειράματα με Χρήση Διαφορετικής Τιμής Ορμής 122

6.2.5 Πειράματα με Χρήση Διαφορετικού Stride 125

6.2.6 Πειράματα με Χρήση Διαφορετικού Αριθμού Παράλληλων Φίλτρων 129

6.2.7 Πειράματα με Χρήση Διαφορετικών Πλάνων Ενημέρωσης Ρυθμού Μάθησης 133

6.2.8 Πειράματα με Χρήση Max Pooling Layers 136

6.2.9 Πειράματα με Χρήση Διαφορετικών Μεθόδων Ενημέρωσης του Ρυθμού Μάθησης 140

6.2.10 Συμπεράσματα για Πειράματα με Χρήση Αρχείων MSA 143

6.3 Πειράματα με Convolution Μεγάλων Gabor Φίλτρων 147

6.3.1 Περιγραφή 147

6.3.2 Πειραματικό Μέρος 147

6.3.3 Συμπεράσματα 150

6.4 Πειράματα με Convolution Kernels Gabor Φίλτρων 150
6.4.1 Περιγραφή 150
6.4.2 Πειραματικό Μέρος 151
6.4.3 Συμπεράσματα 154
6.5 Πειράματα με Χρήση Μέσου Όρου Γειτονικών Κελιών 154
6.5.1 Περιγραφή 154
6.5.2 Πειραματικό Μέρος 155
6.5.3 Συμπεράσματα 157
6.6 Πειράματα με Χρήση Τεχνικής Σημαντικότερων Γειτονικών Αμινοξέων 158
6.6.1 Περιγραφή 158
6.6.2 Πειραματικό Μέρος 159
6.6.3 Συμπεράσματα 161
6.7 Πειράματα με Χρήση της Μεθόδου Conjugate Gradient 162
6.7.1 Περιγραφή 162
6.7.2 Πειραματικό Μέρος 163
6.7.3 Συμπεράσματα 166
6.8 Πειράματα με Χρήση του PISCES Dataset 167
6.8.1 Περιγραφή 167
6.8.2 Πειραματικό Μέρος 167
6.8.3 Συμπεράσματα 170
6.9 Πειράματα με Συνδυασμό CNN και BRNN 170
6.9.1 Περιγραφή 170
6.9.2 Πειραματικό Μέρος 170
6.9.3 Συμπεράσματα 172

6.10 Πειράματα με Συνδυασμό CNN και SVM 173

6.10.1 Περιγραφή 173

6.10.2 Πειραματικό Μέρος 173

6.10.3 Συμπεράσματα 175

6.1 Βιβλιοθήκη DeepLearning4j – DL4J

6.1.1 Γενικά

Η βιβλιοθήκη Deeplearning4j είναι μια βιβλιοθήκη κατασκευής, εκπαίδευσης και εφαρμογής βαθιών τεχνητών νευρωνικών δικτύων η οποία είναι βασισμένη στη γλώσσα προγραμματισμού Java. Η συγκεκριμένη βιβλιοθήκη αποτελείται από κάποιες άλλες βιβλιοθήκες, κάθε μια εκ των οποίων εξυπηρετεί συγκεκριμένους σκοπούς. Για παράδειγμα έχουμε την βιβλιοθήκη DataVec, η οποία χρησιμοποιείται για επεξεργασία, κανονικοποίηση και μετασχηματισμό των δεδομένων εισόδου σε διανύσματα χαρακτηριστικών (feature vectors). Επιπρόσθετα, η βιβλιοθήκη Deeplearning4j, μας παρέχει τα κατάλληλα εργαλεία για να μπορέσουμε να ρυθμίσουμε και να κτίσουμε τεχνητά νευρωνικά δίκτυα καθώς επίσης και υπολογιστικούς γράφους.

Σε αυτή την έρευνα, χρησιμοποιήσαμε την συγκεκριμένη βιβλιοθήκη για την προσαρμόσουμε και να προσπαθήσουμε να επιλύσουμε το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Όλα τα πειράματα τα οποία πραγματοποιήσαμε είναι βασισμένα στην συγκεκριμένη βιβλιοθήκη.

6.1.2 Προσαρμογή Βιβλιοθήκης για το PSSP Πρόβλημα

Η βιβλιοθήκη αυτή έπρεπε να προσαρμοστεί σε μεγάλο βαθμό για να μπορέσουμε να την χρησιμοποιήσουμε για το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Για αυτό το λόγο έγιναν αρκετές αλλαγές στον ανοικτό πηγαίο κώδικα της βιβλιοθήκης και αναπτύχθηκαν κάποια άλλα προγράμματα σε Java (π.χ. Protein.java, Protein_file_creator.java – Παράρτημα Α), τα οποία έχουν ως κύριο σκοπό την προετοιμασία των δεδομένων εκπαίδευσης και επαλήθευσης, τα οποία θα χρησιμοποιήσει η βιβλιοθήκη για την εκπαίδευση και επαλήθευση του τεχνητού νευρωνικού δικτύου που θα κατασκευάσουμε. Στο σχήμα 6.1 φαίνεται ένα μέρος του αρχείου κώδικα μετά τις αλλαγές που πραγματοποιήσαμε στη βιβλιοθήκη, στο οποίο δημιουργούμε και ορίζουμε ένα τεχνητό συνελικτικό νευρωνικό δίκτυο τριών επιπέδων (δύο συνελικτικά επίπεδα και ένα επίπεδο εξόδου), για την επίλυση του προβλήματος της πρόβλεψης δευτεροταγούς δομής των πρωτεϊνών.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
        .iterations(iterations) // Training iterations as above
        .regularization(true).l2(0.005)
        .learningRateDecayPolicy(LearningRatePolicy.Schedule)
        .learningRateSchedule(lrSchedule)
        .weightInit(WeightInit.RELU)
        .optimizationAlgo(OptimizationAlgorithm.GRADIENT_DESCENT)
        .updater(Updater.NESTEROVS).momentum(0.8)
        .list()
        .layer(0, new ConvolutionLayer.Builder()
            .kernelSize(2,2)
            .stride(1,1)
            .nIn(nChannels) //for ConvolutionLayer
            .nOut(10)
            .activation(Activation.LEAKYRELU)
            .build())
        .layer(1, new ConvolutionLayer.Builder()
            .kernelSize(2,2)
            .stride(1,1)
            .nIn(10)
            .nOut(10)
            .activation(Activation.LEAKYRELU)
            .build())
        .layer(2, new        OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
            .nIn(3)
            .nOut(outputNum)
            .activation(Activation.SOFTMAX)
            .build())
        .setInputType(InputType.convolutionalFlat(15,20,1))
        .backprop(true).pretrain(false).build();
//initialize neural model
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
```

Σχήμα 6.1: Κομμάτι κώδικα στο οποίο φαίνεται η ρύθμιση της αρχιτεκτονικής του συνελικτικού νευρωνικού δικτύου.

Στο πιο πάνω σχήμα (Σχήμα 6.1), τα βάρη του δικτύου αρχικοποιούνται με μορφή κατάλληλη για νευρώνες που έχουν ως συνάρτηση ενεργοποίησης την συνάρτηση ReLU. Χρησιμοποιούμε ποινή κανονικοποίησης τύπου L2, συνάρτηση βελτιστοποίησης τη μέθοδο κατάβασης κλίσης, και ως συναρτήσεις ενεργοποίησης έχουμε τη συνάρτηση Leaky Relu, στα δύο πρώτα επίπεδα και τη συνάρτηση softmax στα άλλα δύο επίπεδα του δικτύου. Τα πρώτα δύο συνελικτικά επίπεδα έχουν kernels μεγέθους 2×2, strides (1,1) και χρησιμοποιούνται δέκα παράλληλα φίλτρα (kernels) στα δύο συνελικτικά επίπεδα, για εξαγωγή χαρακτηριστικών από τα δεδομένα εισόδου. Το προσαρμοσμένο αρχείο κώδικα φαίνεται στο παράρτημα Β.

6.2 Πειράματα με Χρήση Αρχείων MSA

6.2.1 Περιγραφή

Τα αρχεία τα οποία θα χρησιμοποιήσουμε για να μπορέσουμε να εκπαιδεύσουμε το νευρωνικό μας δίκτυο, είναι τα αρχεία MSA τα οποία περιγράψαμε εκτενώς στο κεφάλαιο 5. Τα αρχεία MSA φαίνεται να είναι αρκετά δημοφιλή στη χρήση από την επιστημονική κοινότητα, για προσπάθεια επίλυσης του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Ο λόγος είναι ότι μέσω της πολλαπλής στοίχισης αλληλουχιών πρωτεϊνών, δίνουμε στο νευρωνικό δίκτυο στοιχεία για την εξελικτική συσχέτιση τους, παρέχοντας τους έμμεσα την ικανότητα να εντοπίσουν τους παράγοντες που επηρεάζουν την δευτεροταγή δομή των πρωτεϊνών.

Στη συνέχεια, θα προσπαθήσουμε να εφαρμόσουμε στα αρχεία MSA διάφορες τεχνικές, για προσπάθεια εξαγωγής περαιτέρω χαρακτηριστικών έτσι ώστε το νευρωνικό δίκτυο που θα δημιουργήσουμε να μπορεί να επιλύσει με ένα πιο αποδοτικό και ακριβή τρόπο το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Η εφαρμογή των τεχνικών που αναφέραμε στο κεφάλαιο 5, σε συνδυασμό με την επιλογή των βέλτιστων παραμέτρων του δικτύου, αναμένεται να οδηγήσουν στην επιτυχή εκπαίδευση ενός τεχνητού νευρωνικού δικτύου που επιλύει σε αρκετά ικανοποιητικό βαθμό το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

Συνοψίζοντας, χρησιμοποιώντας τα αρχεία κώδικα Protein.java και Protein_files_creator.java, θα πάρουμε σαν αποτέλεσμα τα αρχεία εκπαίδευσης και επαλήθευσης του δικτύου με την MSA κωδικοποίηση. Έπειτα θα εφαρμοστούν όλες οι τεχνικές οι οποίες αναφέρθηκαν σε προηγούμενα κεφάλαια και τέλος θα ελέγξουμε τα ποσοστά επιτυχίας στην πρόβλεψη Q3 καθώς και τα SOV σκορ.

Για να προχωρήσουμε στην πειραματική φάση της έρευνας έπρεπε να δημιουργήσουμε ένα πλάνο το οποίο θα ακολουθούσαμε έτσι ώστε να μπορέσουμε να αναλύσουμε μεθοδικά τα αποτελέσματα επιτυχίας ανάλογα με κάθε συνδυασμό παραμέτρων. Για αυτό το λόγο, αποφασίσαμε αρχικά, να κινηθούμε σύμφωνα με το πιο κάτω πλάνο. Όλα τα πειράματα

πραγματοποιούνται μετά από κανονικοποίηση των δεδομένων εκπαίδευσης και επαλήθευσης στο διάστημα (0,1) ή (-1,1).

Αξίζει να σημειωθεί, ότι σε όλα τα πειράματα που ακολουθούν έχει εφαρμοστεί η μέθοδος γρήγορου σταματήματος, ποινή κανονικοποίησης l2 ή της τεχνικής dropout, έτσι ώστε να μην υπάρχει περίπτωση να υπάρξει το φαινόμενο της υπερεκπαίδευσης - overfit. Στις γραφικές παραστάσεις σφάλματος εκπαίδευσης – εποχής δεν εμφανίζεται το σφάλμα επαλήθευσης αφού χρησιμοποιούνται οι πιο πάνω μέθοδοι που αποφεύγουν το φαινόμενο της υπερεκπαίδευσης. Επιπρόσθετα, η αρχικοποίηση των βαρών έγινε με χρήση συνάρτησης της βιβλιοθήκης DeepLearning4j, η οποία είναι κατάλληλη για αρχικοποίηση των βαρών του συνελικτικού νευρωνικού δικτύου το οποίο θα χρησιμοποιεί ως συνάρτηση ενεργοποίησης την συνάρτηση rectifier (ReLU).

Τέλος, όλα τα πειράματα που ακολουθούν εκτελούνται πέντε (5) φορές το καθένα έτσι ώστε να λάβουμε το μέσο όρο των αποτελεσμάτων εξόδου για κάθε ξεχωριστή περίπτωση συνελικτικού νευρωνικού δικτύου, για εξαγωγή ακριβέστερων και πιο αντικειμενικών συμπερασμάτων.

6.2.2 Πειράματα με Χρήση Διαφορετικού Αριθμού Κρυφών Συνελικτικών Επιπέδων

Ο αριθμός των κρυφών επιπέδων ενός συνελικτικού νευρωνικού δικτύου, είναι μία από τις πιο σημαντικές παραμέτρους ρύθμισης ενός συνελικτικού νευρωνικού δικτύου. Ο αριθμός των κρυφών επιπέδων ενός συνελικτικού νευρωνικού δικτύου, καθορίζει την ικανότητα κωδικοποίησης και εύρεσης πολύπλοκων χαρακτηριστικών στα δεδομένα εισόδου, του συνελικτικού νευρωνικού δικτύου. Αποφασίσαμε λοιπόν, να εκτελέσουμε κάποια πειράματα με συνελικτικά νευρωνικά δίκτυα, τα οποία θα διαφέρουν στον αριθμό των κρυφών επιπέδων. Ο λόγος για αυτή μας την ενέργεια, είναι η εύρεση και καταγραφή του αριθμού κρυφών επιπέδων που οδηγεί στα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3 και SOV. Είναι σημαντικό να αναφέρουμε ότι έχουν πραγματοποιηθεί πολλά πειράματα με διαφορετικούς συνδυασμούς αριθμού κρυφών επιπέδων και των υπόλοιπων παραμέτρων ενός συνελικτικού νευρωνικού δικτύου, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ’

αυτού, θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά με τον αριθμό των κρυφών συνελικτικών επιπέδων, ο οποίος οδηγεί στα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3. Στο πιο κάτω πλάνο, μπορούμε να δούμε τη σειρά διεξαγωγής των πειραμάτων αυτών. Αξίζει να σημειωθεί ότι δεν θα εκτελέσουμε κάποιο πείραμα συνελικτικού νευρωνικού δικτύου, με περισσότερα από πέντε (5) κρυφά επίπεδα, λόγω περιορισμού στη μνήμη του υπολογιστή αλλά και λόγω του μεγάλου χρόνου εκπαίδευσης που απαιτεί αυτή η ενέργεια.

Πλάνο προς εκτέλεση:

1. CNN με ένα (1) κρυφό συνελικτικό επίπεδο – cnn1.
2. CNN με δύο (2) κρυφά συνελικτικά επίπεδα – cnn2.
3. CNN με τρία (3) κρυφά συνελικτικά επίπεδα – cnn3.
4. CNN με τέσσερα (4) κρυφά συνελικτικά επίπεδα – cnn4.
5. CNN με πέντε (5) κρυφά συνελικτικά επίπεδα – cnn5.

Αρχικά, χρησιμοποιήσαμε τις ίδιες παραμέτρους σε κάθε κρυφό συνελικτικό επίπεδο έτσι ώστε να δούμε το πόσο πολύ επηρεάζει το ποσοστό επιτυχίας Q3, ο αριθμός των κρυφών συνελικτικών επιπέδων του δικτύου, κρατώντας τις υπόλοιπες παραμέτρους σταθερές. Οι παράμετροι που θέσαμε σε κάθε κρυφό συνελικτικό επίπεδο του δικτύου φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 200

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

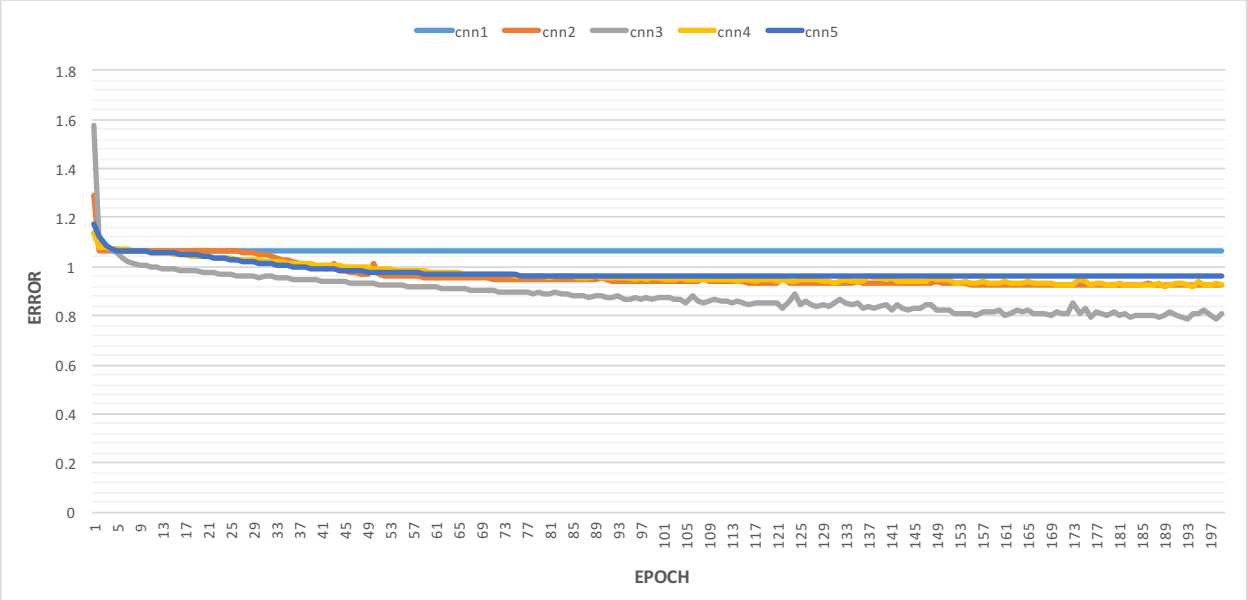
Ορμή (από 0-1): 0.8

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.01
500	0.005
700	0.001
900	0.0005

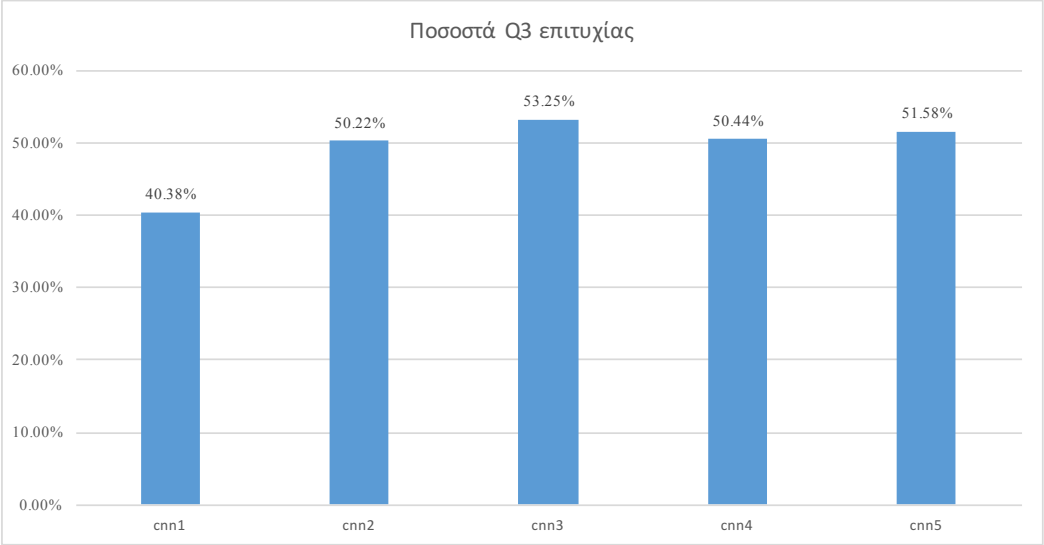
Τιμές παραμέτρων σε κάθε κρυφό συνελικτικό επίπεδο:

Kernel Size: 4x4
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



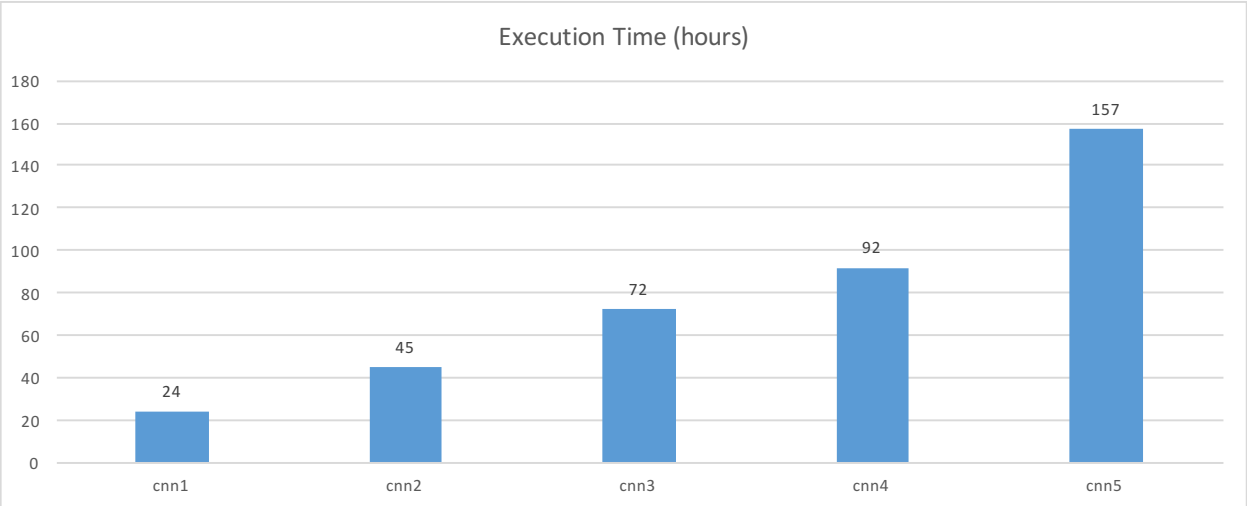
Γραφική Παράσταση 6.1: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο με διαφορετικό αριθμό κρυφών συνελικτικών επιπέδων.

Από την πιο πάνω γραφική παράσταση, μπορούμε να δούμε ότι τα συνελικτικά νευρωνικά δίκτυα φαίνεται να μαθαίνουν, λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι το συνελικτικό νευρωνικό δίκτυο με τρία κρυφά επίπεδα (cnn3) καταφέρνει να μειώσει το σφάλμα εκπαίδευσης περισσότερο από τα άλλα δίκτυα. Στη συνέχεια ακολουθούν τα δίκτυα με 4, 2, 5 και 1 κρυφά επίπεδα.



Γραφική Παράσταση 6.2: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο με διαφορετικό αριθμό κρυφών συνελικτικών επιπέδων.

Στην πιο πάνω γραφική παράσταση μπορούμε να δούμε τα αποτελέσματα επιτυχίας Q3. Όπως ήταν αναμενόμενο, το νευρωνικό δίκτυο με τρία επίπεδα φαίνεται να πετυχαίνει υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη, σε σχέση με τα άλλα νευρωνικά δίκτυα με διαφορετικό αριθμό κρυφών επιπέδων.



Γραφική Παράσταση 6.3: Απαιτούμενος χρόνος εκπαίδευσης για κάθε νευρωνικό δίκτυο με διαφορετικό αριθμό κρυφών συνελικτικών επιπέδων.

Στην πιο πάνω γραφική παράσταση μπορούμε να δούμε τους χρόνους εκτέλεσης για κάθε ένα δίκτυο. Φαίνεται ότι ο χρόνος εκπαίδευσης είναι ανάλογος του αριθμού των κρυφών

συνελικτικών επιπέδων του δικτύου. Με άλλα λόγια, όσο αυξάνουμε τα κρυφά συνελικτικά επίπεδα του δικτύου, αυξάνεται και ο απαιτούμενος χρόνος εκπαίδευσης.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις (Γραφικές παραστάσεις 6.1, 6.2, και 6.3), αποφασίσαμε να προχωρήσουμε τα πειράματα μας με χρήση δύο, τριών ή τεσσάρων κρυφών συνελικτικών επιπέδων, αφού εκτός του ότι πετυχαίνουν τα καλύτερα αποτελέσματα, ο απαιτούμενος χρόνος εκπαίδευσης τους είναι ανεκτός. Αντίθετα, για το δίκτυο με πέντε κρυφά επίπεδα τα αποτελέσματα επιτυχίας Q3 δεν φαίνεται να αυξάνονται ιδιαίτερα σε σχέση με τα άλλα δίκτυα ενώ ο χρόνος εκπαίδευσης είναι αρκετά πιο μεγάλος. Αντίστοιχα, για το νευρωνικό δίκτυο με ένα κρυφό συνελικτικό επίπεδο τα αποτελέσματα επιτυχίας στην πρόβλεψη είναι αρκετά χαμηλά σε σχέση με τα άλλα δίκτυα, κάτι που είναι απόλυτα λογικό, αφού ένα μόνο κρυφό επίπεδο δεν μπορεί να κωδικοποιήσει και να κατανοήσει την πολυπλοκότητα της μορφής του προβλήματος.

6.2.3 Πειράματα με Χρήση Διαφορετικών Μεγεθών Kernel

Για να μπορέσουμε να αποφασίσουμε ποιο είναι το καταλληλότερο μέγεθος kernel, το οποίο θα μας οδηγήσει στα μεγαλύτερα ποσοστά επιτυχίας Q3, αποφασίσαμε να εκτελέσουμε κάποια πειράματα στα οποία θα χρησιμοποιηθεί ένας σταθερός αριθμός κρυφών συνελικτικών επιπέδων, αλλά διαφορετικό μέγεθος από kernels, έτσι ώστε να μπορέσουμε να αποφανθούμε για το καταλληλότερο μέγεθος kernel κρατώντας σταθερές τις υπόλοιπες παραμέτρους. Είναι σημαντικό να αναφέρουμε ότι έχουν πραγματοποιηθεί πολλά πειράματα με διαφορετικά μεγέθη kernel, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού, θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά με το μέγεθος kernel, το οποίο οδηγεί στα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3.

<u>Πλάνο προς εκτέλεση:</u>

1. CNN με 1x1 μέγεθος kernel – cnn1.
2. CNN με 2x2 μέγεθος kernel – cnn2.
3. CNN με 3x3 μέγεθος kernel – cnn3.
4. CNN με 4x4 μέγεθος kernel – cnn4.

5. CNN με 2x10 μέγεθος kernel – cnn5.
6. CNN με 1x20 μέγεθος kernel – cnn6.
7. CNN με 4x5 μέγεθος kernel – cnn7.

Αξίζει να σημειωθεί, ότι δεν μπορούμε να προχωρήσουμε στη χρήση μεγαλύτερου μεγέθους kernel (δηλαδή μέγεθος kernel μεγαλύτερο από είκοσι), αφού για τα απλά αρχεία MSA το νευρωνικό δίκτυο λαμβάνει μία-μία εγγραφή, είκοσι παραμέτρων, που αντιπροσωπεύει ένα αμινοξύ και προβλέπει την δευτεροταγή δομή του συγκεκριμένου αμινοξέος, η οποία ετικέτα δευτεροταγούς δομής βρίσκεται στο τέλος κάθε εγγραφής. Οι παράμετροι που θέσαμε σε κάθε κρυφό συνελικτικό επίπεδο του δικτύου φαίνονται πιο κάτω.

<u>Παράμετροι ρύθμισης των CNNs:</u>

Εποχές: 150

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

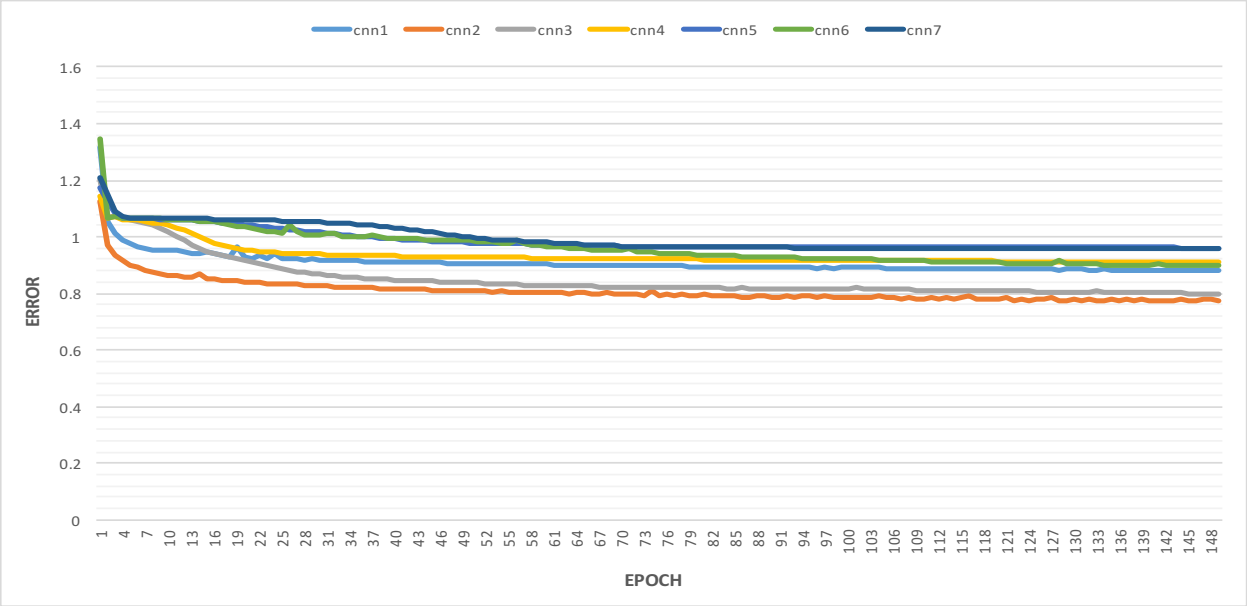
Ορμή (από 0-1): 0.8

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.01
500	0.005
700	0.001
900	0.0005

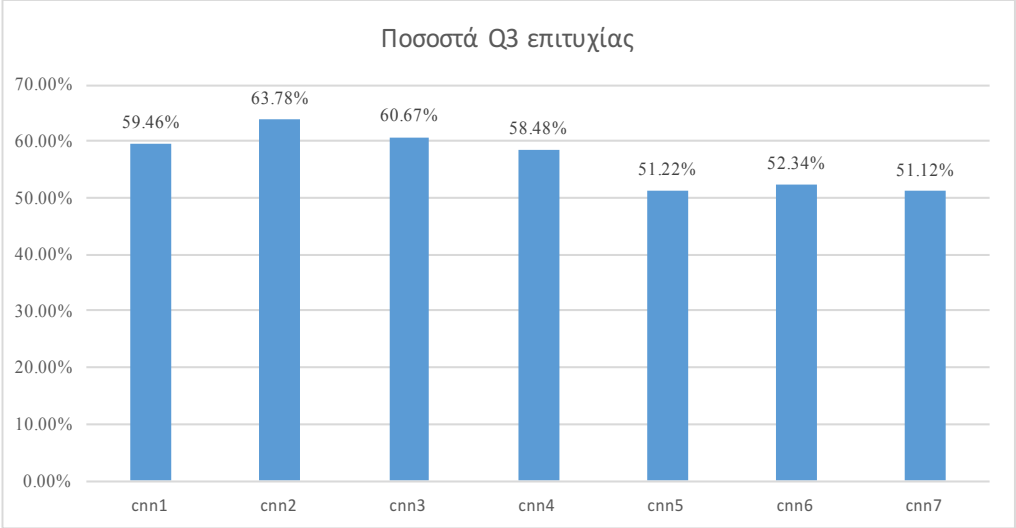
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: όπως καθορίστηκε στο πλάνο για κάθε πείραμα
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.4: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με διαφορετικά μεγέθη kernel.

Από την γραφική παράσταση 6.4, μπορούμε να δούμε ότι τα συνελικτικά νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς, λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι το συνελικτικό νευρωνικό δίκτυο με μέγεθος kernel 2x2, καταφέρνει να μειώσει το σφάλμα εκπαίδευσης περισσότερο από τα άλλα δίκτυα. Στη συνέχεια ακολουθούν τα νευρωνικά δίκτυα με μεγέθη kernel 3x3, 1x1, 1x20, 4x4, 4x5 και 2x10.

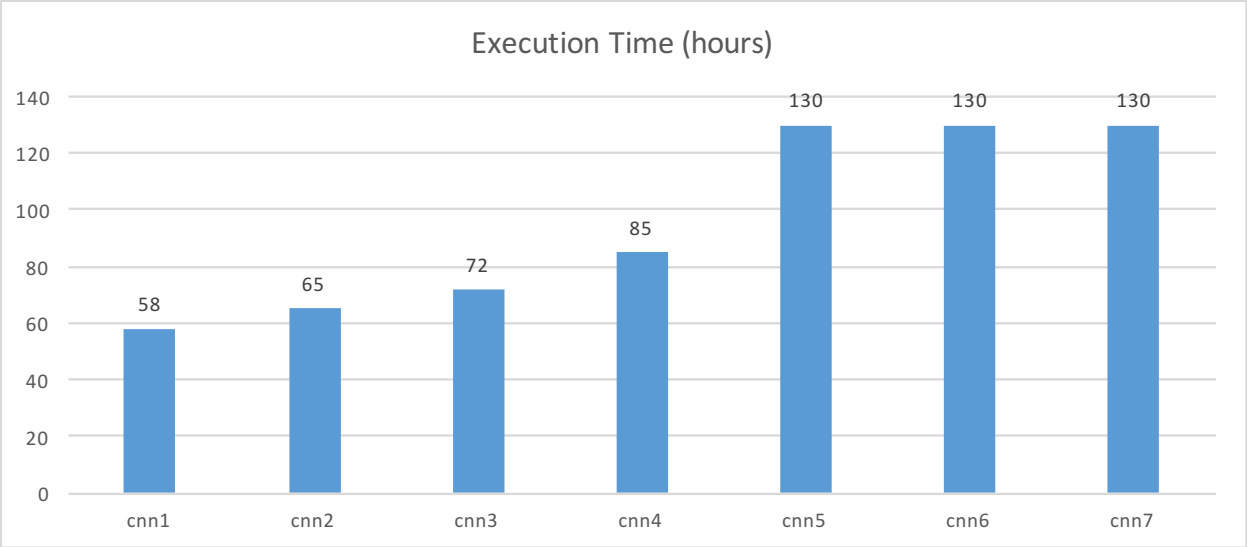


Γραφική Παράσταση 6.5: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο, με διαφορετικά μεγέθη kernel.

Στη γραφική παράσταση 6.5 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής), το νευρωνικό δίκτυο με μέγεθος kernel 2x2 – cnn2, καταφέρνει να πετύχει το υψηλότερο ποσοστό επιτυχίας Q3, ενώ ακολουθούν τα δίκτυα με μεγέθη kernel 3x3, 1x1, 4x4, 1x20, 2x10, και 4x5.

Τα αποτελέσματα των πιο πάνω γραφικών παραστάσεων (Γραφική Παράσταση 6.4 και 6.5), μπορούν να εξηγηθούν εύκολα για τα δίκτυα με μεγέθη kernel 2x10 – cnn5, 1x20 – cnn6 και 4x5 – cnn7. Ο κύριος λόγος που αυτά τα δίκτυα δεν μπορούν να πετύχουν τόσο ψηλά αποτελέσματα σε σχέση με τα άλλα δίκτυα τα οποία έχουν διαφορετικά μεγέθη kernel, είναι ο σχεδιασμός του συνελικτικού νευρωνικού δικτύου, και πιο συγκεκριμένα, η λειτουργία και η αρχιτεκτονική του δικτύου αφού όπως εξηγήσαμε στην ενότητα 3.3.5, τα συνελικτικά νευρωνικά δίκτυα, είναι σχεδιασμένα έτσι ώστε να δουλεύουν καλύτερα χρησιμοποιώντας τετράγωνα σε διαστάσεις φίλτρα – kernels, στα κρυφά συνελικτικά τους επίπεδα για εξαγωγή χαρακτηριστικών από τα δεδομένα εισόδου.

Τα νευρωνικά δίκτυα με μεγέθη kernel 1x1, 3x3 και 4x4 φαίνεται να πετυχαίνουν παρόμοια ποσοστά επιτυχίας Q3, αλλά όχι τόσο ψηλά όσο το νευρωνικό δίκτυο με μέγεθος kernel 2x2. Ο λόγος που συμβαίνει αυτό, είναι το ότι χρησιμοποιώντας πιο μεγάλα σε μέγεθος kernels, οι διαστάσεις των αποτελεσμάτων εξόδου σε κάθε κρυφό συνελικτικό επίπεδο μειώνονται/συμπτύσσονται με αποτέλεσμα να χάνουμε χρήσιμη πληροφορία. Αντίθετα, το δίκτυο με μέγεθος kernel 1x1 δεν καταφέρνει να πετύχει ψηλότερα αποτελέσματα από το δίκτυο με μέγεθος kernel 2x2, λόγω του ότι δεν μπορεί να εντοπίσει τις γειτονικές αλληλεπιδράσεις που προκαλούνται μεταξύ των αμινοξέων ούτε τις πιθανότητες κατηγοριοποίησης του αμινοξέος που μελετάται σε ένα από τα είκοσι αμινοξέα της πρωτοταγούς δομής που ξέρουμε, αφού εφαρμόζουμε κάθε φορά τα φίλτρα/kernel σε μία τιμή της κάθε MSA εγγραφής, η οποία αναφέρεται σε μία μόνο πιθανότητα κατηγοριοποίησης του αμινοξέος που μελετάται σε ένα από τα είκοσι αμινοξέα που γνωρίζουμε.



Γραφική Παράσταση 6.6: Απαιτούμενος χρόνος εκπαίδευσης για κάθε νευρωνικό δίκτυο, με διαφορετικά μεγέθη kernel.

Στη γραφική παράσταση 6.6, μπορούμε να δούμε τους χρόνους εκτέλεσης για κάθε συνελικτικό νευρωνικό δίκτυο. Φαίνεται ότι ο χρόνος εκπαίδευσης είναι ανάλογος του μεγέθους των kernels των κρυφών συνελικτικών επιπέδων του νευρωνικού δικτύου. Αυτό φαίνεται κυρίως από τα cnn5, cnn6 και cnn7 στα οποία χρησιμοποιούνται ίδιου μεγέθους kernels (αλλά με διαφορετικές διαστάσεις), και φαίνεται ότι απαιτούν τον ίδιο χρόνο εκπαίδευσης.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις αποφασίσαμε να προχωρήσουμε τα πειράματα μας χρησιμοποιώντας συνελικτικά νευρωνικά δίκτυα με μέγεθος kernel 2x2 σε κάθε κρυφό συνελικτικό επίπεδο, αφού φαίνεται να πετυχαίνουν τα υψηλότερα ποσοστά επιτυχίας Q3, ενώ ταυτόχρονα απαιτούν το μικρότερο δυνατό χρόνο εκπαίδευσης, σε σχέση με τα άλλα νευρωνικά δίκτυα τα οποία χρησιμοποιούν διαφορετικά μεγέθη kernel.

6.2.4 Πειράματα με Χρήση Διαφορετικής Τιμής Ορμής

Η ορμή είναι ένας εξαιρετικά σημαντικός παράγοντας για την επιτάχυνση του ρυθμού εκπαίδευσης αλλά και την «απόδραση» από τοπικά ελάχιστα κατά την διάρκεια εκπαίδευσης, ενός νευρωνικού δικτύου. Για να μπορέσουμε να αποφασίσουμε την τιμή της ορμής, η οποία θα μας οδηγήσει στα καλύτερα δυνατά αποτελέσματα επιτυχίας στην πρόβλεψη Q3, θα

προχωρήσουμε στην εκτέλεση κάποιων πειραμάτων τα οποία θα αφορούν συνελικτικά νευρωνικά δίκτυα, τα οποία κάνουν χρήση διαφορετικών τιμών ορμής. Τα δίκτυα αυτά, θα έχουν ίδιο αριθμό κρυφών συνελικτικών επιπέδων και ίδιες παραμέτρους (αλλά διαφορετική τιμή στον παράγοντα ορμής), έτσι ώστε να μπορέσουμε να αποφασίσουμε την τιμή του παράγοντα της ορμής, η οποία δίνει την υψηλότερη επιτυχία στην πρόβλεψη Q3. Είναι σημαντικό να αναφέρουμε, ότι έχουν πραγματοποιηθεί πολλά πειράματα με διαφορετικές τιμές ορμής, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού, θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά τον παράγοντα ορμής, ο οποίος οδηγεί στα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3.

Πλάνο προς εκτέλεση:

1. CNN με ορμή 0.2 – cnn1.
2. CNN με ορμή 0.4 – cnn2.
3. CNN με ορμή 0.6 – cnn3.
4. CNN με ορμή 0.7 – cnn4.
5. CNN με ορμή 0.8 – cnn5.
6. CNN με ορμή 0.85 – cnn6.
7. CNN με ορμή 0.9 – cnn7.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 300

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

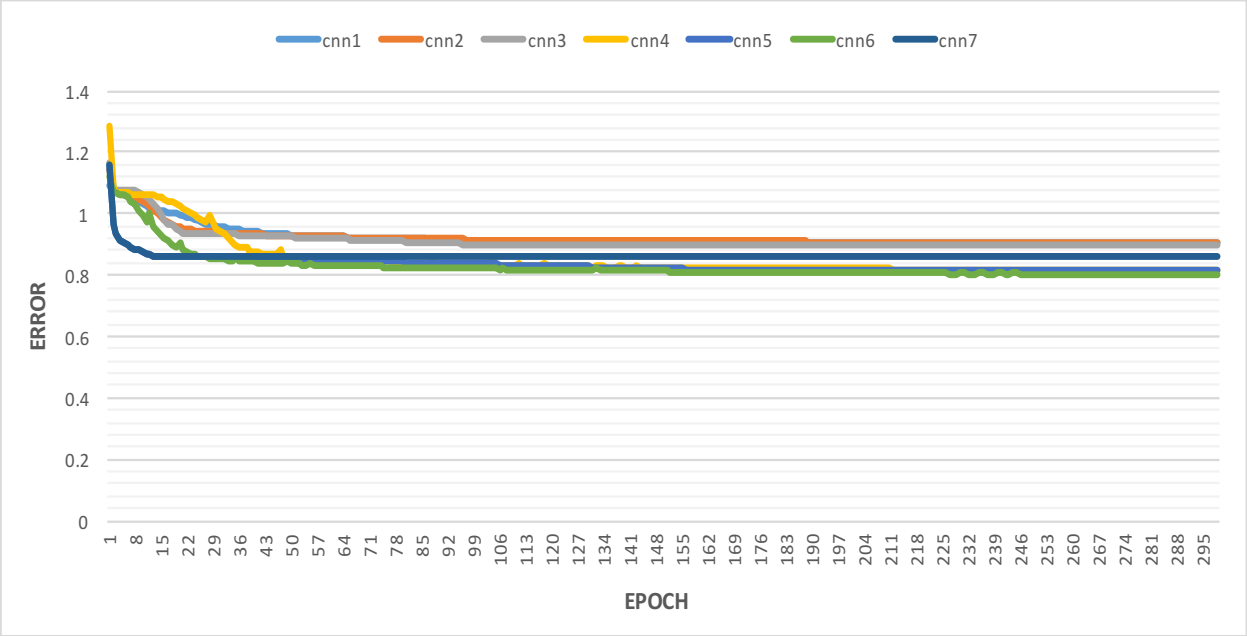
Ορμή (από 0-1): όπως καθορίστηκε στο πλάνο για κάθε πείραμα

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.01
500	0.005
700	0.001
900	0.0005

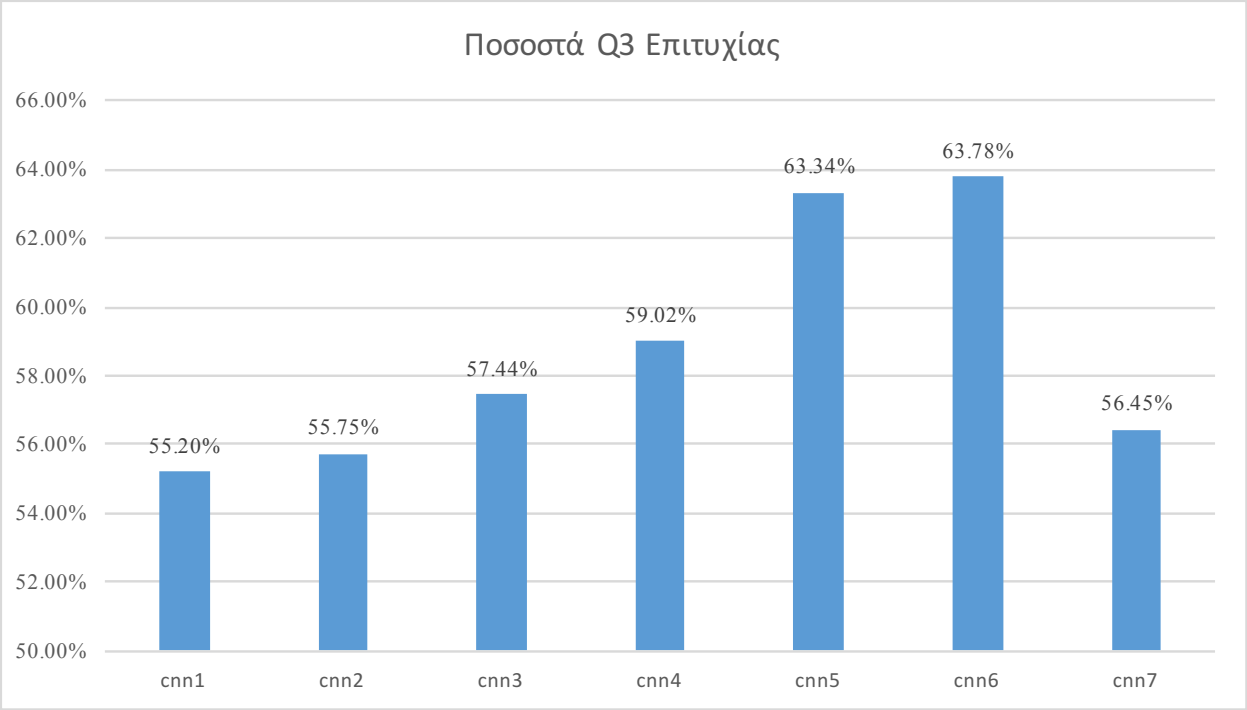
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.7: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με διαφορετικές τιμές ορμής.

Από τη γραφική παράσταση 6.7, μπορούμε να δούμε ότι τα νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς, λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι τα συνελικτικά νευρωνικά δίκτυα cnn5, cnn6 και cnn4, με παράγοντες ορμής 0.8, 0.85 και 0.7 αντίστοιχα, καταφέρνουν να μειώσουν το σφάλμα εκπαίδευσης περισσότερο, σε σχέση με τα άλλα νευρωνικά δίκτυα. Στη συνέχεια, ακολουθούν τα νευρωνικά δίκτυα cnn7, cnn3, cnn2 και cnn1, με παράγοντες ορμής 0.9, 0.6, 0.4 και 0.2 αντίστοιχα.



Γραφική Παράσταση 6.8: Ποσοστά επιτυχίας Q3, για κάθε νευρωνικό δίκτυο, με διαφορετικές τιμές ορμής.

Στη γραφική παράσταση 6.8 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής), το νευρωνικό δίκτυο με παράγοντα ορμής 0.85 – cnn6, καταφέρνει να πετύχει το υψηλότερο ποσοστό επιτυχίας Q3, ενώ ακολουθούν τα δίκτυα cnn5, cnn4, cnn3, cnn7, cnn2 και cnn1 με παράγοντες ορμής 0.8, 0.7, 0.6, 0.9, 0.4 και 0.2 αντίστοιχα.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις, αποφασίσαμε να προχωρήσουμε τα πειράματα μας χρησιμοποιώντας συνελικτικά νευρωνικά δίκτυα με παράγοντα ορμής 0.85 ή 0.8, αφού φαίνεται να πετυχαίνουν τα υψηλότερα ποσοστά επιτυχίας Q3 σε σχέση με άλλα νευρωνικά δίκτυα τα οποία χρησιμοποιούν διαφορετικούς παράγοντες ορμής.

6.2.5 Πειράματα με Χρήση Διαφορετικού Stride

Όπως εξηγήσαμε σε προηγούμενα κεφάλαια, η παράμετρος stride μπορεί να αλλάξει τις διαστάσεις των δεδομένων εξόδου. Η μετακίνηση του φίλτρου/kernel για εξαγωγή

χαρακτηριστικών από τα δεδομένα εισόδου, μπορεί επιτρέψει στο συνελικτικό νευρωνικό δίκτυο να κωδικοποιήσει την πολύπλοκη συσχέτιση και αλληλεπιδράσεις μεταξύ των αμινοξέων στην αρχιτεκτονική του, οδηγώντας το έτσι στην επιτυχή εκπαίδευση του για επίλυση του προβλήματος πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

Για να μπορέσουμε να εντοπίσουμε και να καταγράψουμε την ιδανική τιμή μετακίνησης του kernel (stride), αποφασίσαμε όπως εκτελέσουμε πειράματα συνελικτικών νευρωνικών δικτύων με χρήση διαφορετικών τιμών stride. Ακολούθως, θα χρησιμοποιήσουμε σαν τελική τιμή stride, το stride του δικτύου με τα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3. Οι υπόλοιπες παράμετροι του δικτύου θα παραμείνουν σταθερές, έτσι ώστε να μπορέσουμε να αξιολογήσουμε ορθά τα αποτελέσματα εξόδου με βάση τις διαφορετικές τιμές stride. Είναι σημαντικό να αναφέρουμε, ότι έχουν πραγματοποιηθεί πειράματα με πολλά διαφορετικά strides, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού, θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά με το stride, το οποίο δίνει τα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3.

<u>Πλάνο προς εκτέλεση:</u>

1. CNN με stride 1,1 – cnn1.
2. CNN με stride 1,2 – cnn2.
3. CNN με stride 2,2 – cnn3.
4. CNN με stride 2,1 – cnn4.
5. CNN με stride 2,4 – cnn5.

Αξίζει να σημειωθεί, ότι δεν υπάρχει λογική να χρησιμοποιήσουμε stride μεγαλύτερου μεγέθους από 2,2, αφού το kernel που επιλέξαμε μεγέθους 2x2, θα προσπερνά περιοχές δεδομένων χωρίς να πραγματοποιεί εξαγωγή χαρακτηριστικών άρα θα υπάρχει ελλιπής πληροφόρηση του δικτύου για τα δεδομένα εκπαίδευσης. Για του λόγου του αληθές, εισαγάγαμε στα σχέδια μας ένα πείραμα με stride 2,4 έτσι ώστε να δούμε ότι όντως θα αποτύχει. Οι παράμετροι που θέσαμε σε κάθε κρυφό συνελικτικό επίπεδο του δικτύου φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 200

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

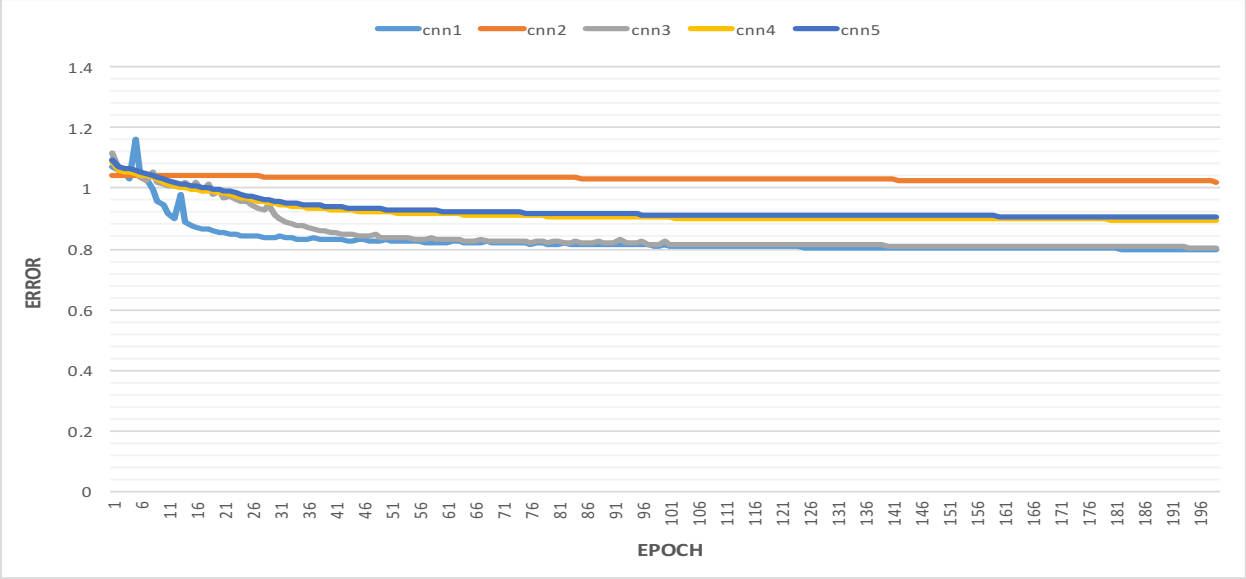
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.01
500	0.005
700	0.001
900	0.0005

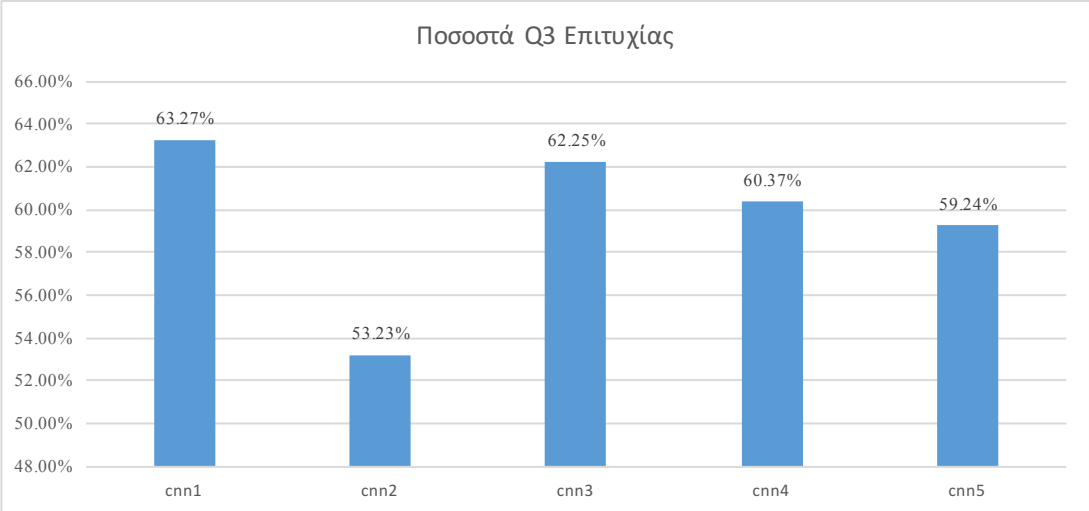
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: όπως καθορίστηκε στο πλάνο για κάθε πείραμα
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.9: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με διαφορετικές τιμές stride.

Από τη γραφική παράσταση 6.9, μπορούμε να δούμε ότι τα νευρωνικά δίκτυα φαίνεται να μαθαίνουν λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι τα συνελικτικά νευρωνικά δίκτυα cnn1 και cnn3, με strides 1,1 και 2,2 αντίστοιχα, καταφέρνουν να μειώσουν το σφάλμα εκπαίδευσης περισσότερο σε σχέση με τα άλλα νευρωνικά δίκτυα. Στη συνέχεια, ακολουθούν τα νευρωνικά δίκτυα cnn4, cnn5 και cnn2, με strides 2,1, 2,4 και 1,2 αντίστοιχα.



Γραφική Παράσταση 6.10: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο, με διαφορετικές τιμές stride.

Στη γραφική παράσταση 6.10, μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής), το νευρωνικό δίκτυο με stride 1,1 – cnn1, καταφέρνει να πετύχει το υψηλότερο ποσοστό επιτυχίας Q3, κάτι που είναι απόλυτα λογικό αφού μετακινώντας ένα-ένα βήμα το φίλτρο στην είσοδο του κάθε συνελικτικού επιπέδου εξάγονται διάφορα χαρακτηριστικά από κάθε ξεχωριστή περιοχή ενώ ταυτόχρονα κωδικοποιούνται τυχόν γειτονικές αλληλεπιδράσεις των αμινοξέων. Στη συνέχεια ακολουθούν τα δίκτυα cnn3, cnn4, cnn5 και cnn2 με stride 2,2, 2,1, 2,4, και 1,2 αντίστοιχα.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις, αποφασίσαμε να προχωρήσουμε τα πειράματα μας χρησιμοποιώντας συνελικτικά νευρωνικά δίκτυα με stride 1,1, αφού φαίνεται να πετυχαίνουν τα υψηλότερα ποσοστά επιτυχίας Q3 σε σχέση με άλλα νευρωνικά δίκτυα τα οποία χρησιμοποιούν διαφορετικά strides.

6.2.6 Πειράματα με Χρήση Διαφορετικού Αριθμού Παράλληλων Φίλτρων

Όπως εξηγήσαμε και στην περιγραφή των συνελικτικών νευρωνικών δικτύων (Ενότητα 3.3.5), ένα φίλτρο/kernel χρησιμοποιείται για εντοπισμό διάφορων χαρακτηριστικών στα δεδομένα εισόδου. Εφαρμόζοντας περισσότερα από ένα παράλληλα φίλτρα σε κάθε στρώμα/επίπεδο του συνελικτικού νευρωνικού δικτύου, είμαστε σε θέση να εντοπίσουμε διαφορετικά χαρακτηριστικά στα δεδομένα εισόδου που μας δίνονται. Αυξάνοντας τα παράλληλα φίλτρα εντοπισμού χαρακτηριστικών σε κάθε κρυφό συνελικτικό επίπεδο, αυξάνονται εκθετικά οι απαιτήσεις σε μνήμη και χρόνο εκπαίδευσης.

Για να μπορέσουμε να εντοπίσουμε και να καταγράψουμε τον κατάλληλο αριθμό παράλληλων φίλτρων εξαγωγής χαρακτηριστικών, αποφασίσαμε όπως εκτελέσουμε κάποια πειράματα συνελικτικών νευρωνικών δικτύων, τα οποία θα χρησιμοποιούν ένα διαφορετικό αριθμό παράλληλων φίλτρων εξαγωγής χαρακτηριστικών. Οι υπόλοιπες παράμετροι του δικτύου θα παραμείνουν σταθερές, έτσι ώστε να μπορέσουμε να αξιολογήσουμε ορθά τα αποτελέσματα εξόδου με βάση τον αριθμό των παράλληλων φίλτρων. Είναι σημαντικό να αναφέρουμε ότι έχουν πραγματοποιηθεί πειράματα με διαφορετικό αριθμό παράλληλων φίλτρων εξαγωγής χαρακτηριστικών, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού, θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά με τον αριθμό παράλληλων φίλτρων, ο οποίος δίνει τα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3.

<u>Πλάνο προς εκτέλεση:</u>

1. CNN με ένα (1) φίλτρο σε κάθε επίπεδο – cnn1.
2. CNN με δύο (2) παράλληλα φίλτρα σε κάθε επίπεδο – cnn2.
3. CNN με πέντε (5) παράλληλα φίλτρα σε κάθε επίπεδο – cnn3.
4. CNN με δέκα (10) παράλληλα φίλτρα σε κάθε επίπεδο – cnn4.
5. CNN με δεκαπέντε (15) παράλληλα φίλτρα σε κάθε επίπεδο – cnn5.
6. CNN με είκοσι (20) παράλληλα φίλτρα σε κάθε επίπεδο – cnn6.

Θα χρησιμοποιήσουμε σε όλα τα νευρωνικά δίκτυα, τρία κρυφά συνελικτικά επίπεδα με μεγέθη kernel 2x2. Οι παράμετροι που θέσαμε σε κάθε κρυφό συνελικτικό επίπεδο του δικτύου φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 300

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

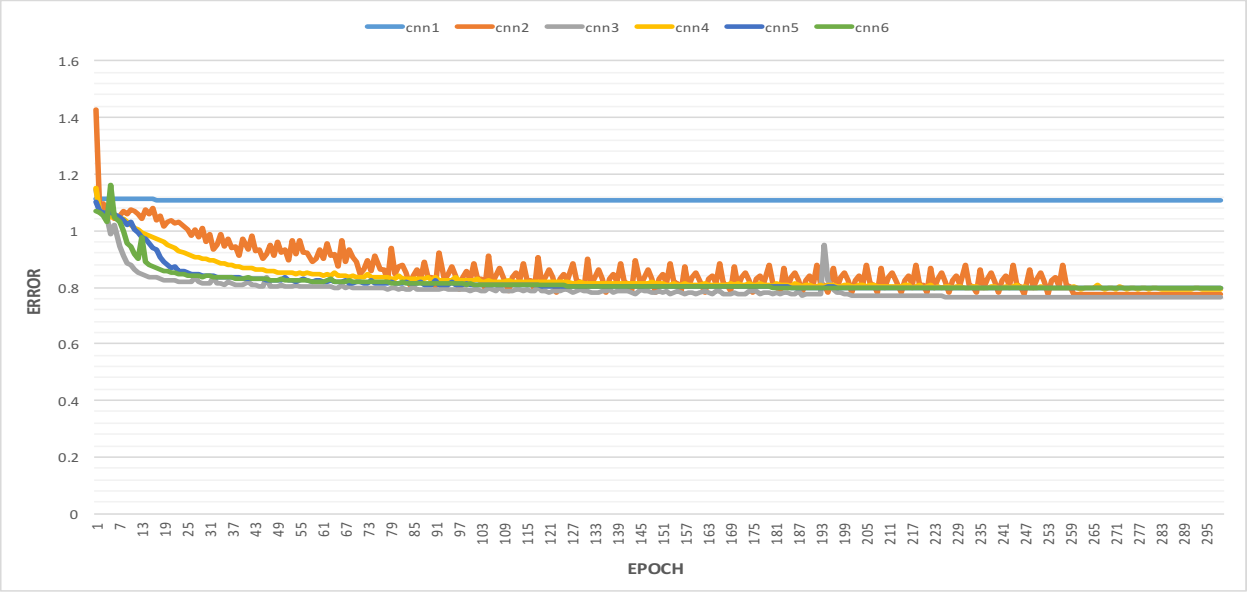
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.01
500	0.005
700	0.001
900	0.0005

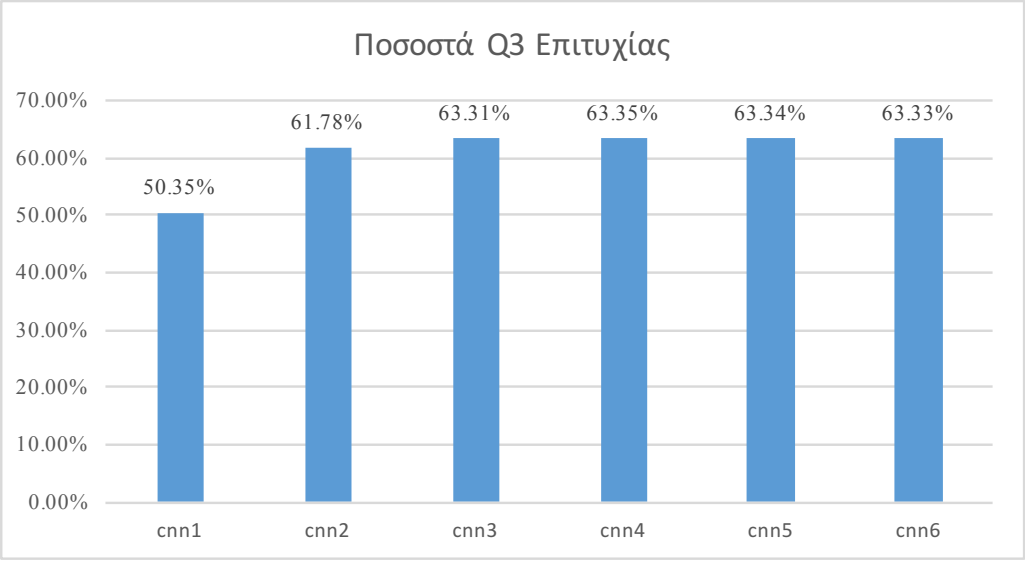
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: όπως καθορίστηκε στο πλάνο για κάθε πείραμα



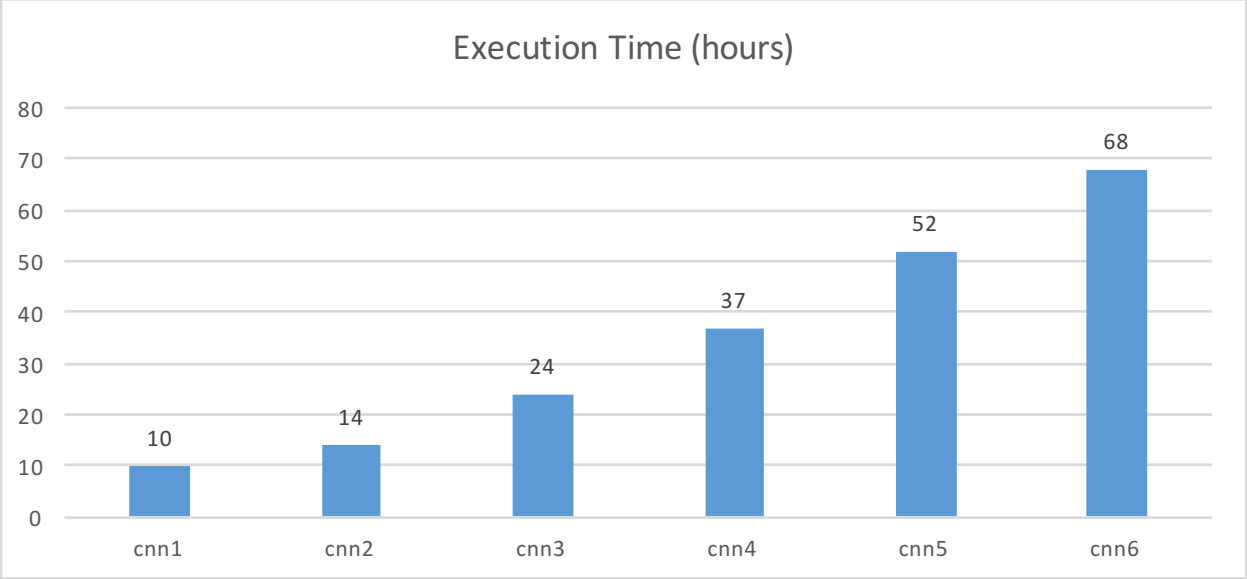
Γραφική Παράσταση 6.11: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με διαφορετικό αριθμό παράλληλων φίλτρων σε κάθε επίπεδο.

Από τη γραφική παράσταση 6.11, μπορούμε να δούμε ότι τα νευρωνικά δίκτυα φαίνεται να μαθαίνουν λόγω του ότι το σφάλμα εκπαίδευσης μειώνεται. Φαίνεται ότι τα συνελικτικά νευρωνικά δίκτυα cnn2, cnn3, cnn4, cnn5 και cnn6 με αριθμό παράλληλων φίλτρων 2, 5, 10, 15 και 20 αντίστοιχα, καταφέρνουν να μειώσουν το σφάλμα εκπαίδευσης περισσότερο σε σχέση με το νευρωνικό δίκτυο cnn1.



Γραφική Παράσταση 6.12: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο, με διαφορετικό αριθμό παράλληλων φίλτρων σε κάθε επίπεδο.

Στη γραφική παράσταση 6.12 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής), τα νευρωνικά δίκτυα με αριθμό παράλληλων φίλτρων 2, 5, 10, 15 και 20, καταφέρνουν να πετύχουν τα υψηλότερα ποσοστά επιτυχίας Q3, ενώ ακολουθεί το δίκτυο cnn1 με 1 φίλτρο σε κάθε επίπεδο.



Γραφική Παράσταση 6.13: Απαιτούμενος χρόνος εκπαίδευσης για κάθε νευρωνικό δίκτυο, με διαφορετικό αριθμό παράλληλων φίλτρων σε κάθε επίπεδο.

Στη γραφική παράσταση 6.13 μπορούμε να δούμε τους απαιτούμενους χρόνους εκπαίδευσης για κάθε νευρωνικό δίκτυο. Φαίνεται ότι ο χρόνος εκπαίδευσης, είναι ανάλογος του αριθμού των παράλληλων φίλτρων που εφαρμόζονται σε κάθε κρυφό επιπέδου του συνελικτικού νευρωνικού δικτύου.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις, αποφασίσαμε να προχωρήσουμε τα πειράματα μας χρησιμοποιώντας συνελικτικά νευρωνικά δίκτυα με πέντε (5) παράλληλα φίλτρα σε κάθε κρυφό επίπεδο, αφού φαίνεται να πετυχαίνουν τα υψηλότερα ποσοστά επιτυχίας Q3, ενώ ταυτόχρονα έχουν τους καλύτερους δυνατούς χρόνους εκτέλεσης σε σχέση με τα άλλα δίκτυα τα οποία χρησιμοποιούν διαφορετικό αριθμό παράλληλων φίλτρων.

6.2.7 Πειράματα με Χρήση Διαφορετικών Πλάνων Ενημέρωσης Ρυθμού Μάθησης

Ο ρυθμός μάθησης είναι μία από τις πιο σημαντικές, αν όχι η σημαντικότερη παράμετρος ρύθμισης ενός συνελικτικού νευρωνικού δικτύου. Ένας μεγάλος ρυθμός μάθησης μπορεί να επιταχύνει την εκπαίδευση ενός νευρωνικού δικτύου αλλά ταυτόχρονα, μπορεί να μην οδηγήσει το δίκτυο στην εύρεση του ολικού ελαχίστου. Αντίθετα, ένας μικρός ρυθμός μάθησης, μπορεί να επιβραδύνει εξαιρετικά την εκπαίδευση ενός νευρωνικού δικτύου και τέλος, να μην οδηγήσει το δίκτυο στην καλύτερη δυνατή λύση, δηλαδή την εύρεση του ολικού ελαχίστου.

Για να μπορέσουμε να εντοπίσουμε και να καταγράψουμε το ιδανικό πρόγραμμα ρυθμού μάθησης (learning rate schedule), αποφασίσαμε όπως εκτελέσουμε πειράματα συνελικτικών νευρωνικών δικτύων με χρήση διαφορετικών learning rate schedules. Ακολούθως, θα χρησιμοποιήσουμε σαν learning rate schedule εκείνο που δίνει τα υψηλότερα ποσοστά στην πρόβλεψη Q3. Οι υπόλοιπες παράμετροι του δικτύου θα παραμείνουν σταθερές, έτσι ώστε να μπορέσουμε να αξιολογήσουμε ορθά τα αποτελέσματα εξόδου με βάση τα διαφορετικά learning rate schedules. Είναι σημαντικό να αναφέρουμε, ότι έχουν πραγματοποιηθεί πειράματα με πολλά διαφορετικά πλάνα εκπαίδευσης, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού, θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά με το πλάνο εκπαίδευσης, το οποίο δίνει τα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3.

<u>Πλάνο προς εκτέλεση:</u>

1. CNN με learning rate 0.5 – cnn1.
2. CNN με learning rate 0.3 – cnn2.
3. CNN με learning rate 0.1 – cnn3.
4. CNN με learning rate schedule ως εξής – cnn4:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.1
500	0.01
700	0.001
900	0.0005

5. CNN με learning rate schedule ως εξής – cnn5:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

6. CNN με learning rate schedule ως εξής – cnn6:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.1
400	0.09
600	0.08
800	0.06
1000	0.04
1500	0.02
2000	0.01
3000	0.008
4000	0.006
6000	0.003
7000	0.001
9000	0.0005
10000	0.0001

Σε όλα τα νευρωνικά δίκτυα, θα χρησιμοποιήσουμε τρία κρυφά συνελικτικά επίπεδα με μεγέθη kernel 2x2. Οι παράμετροι που θέσαμε σε κάθε κρυφό συνελικτικό επίπεδο των νευρωνικών δικτύων φαίνονται πιο κάτω.

<u>Παράμετροι ρύθμισης των CNNs:</u>

Εποχές: 150

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

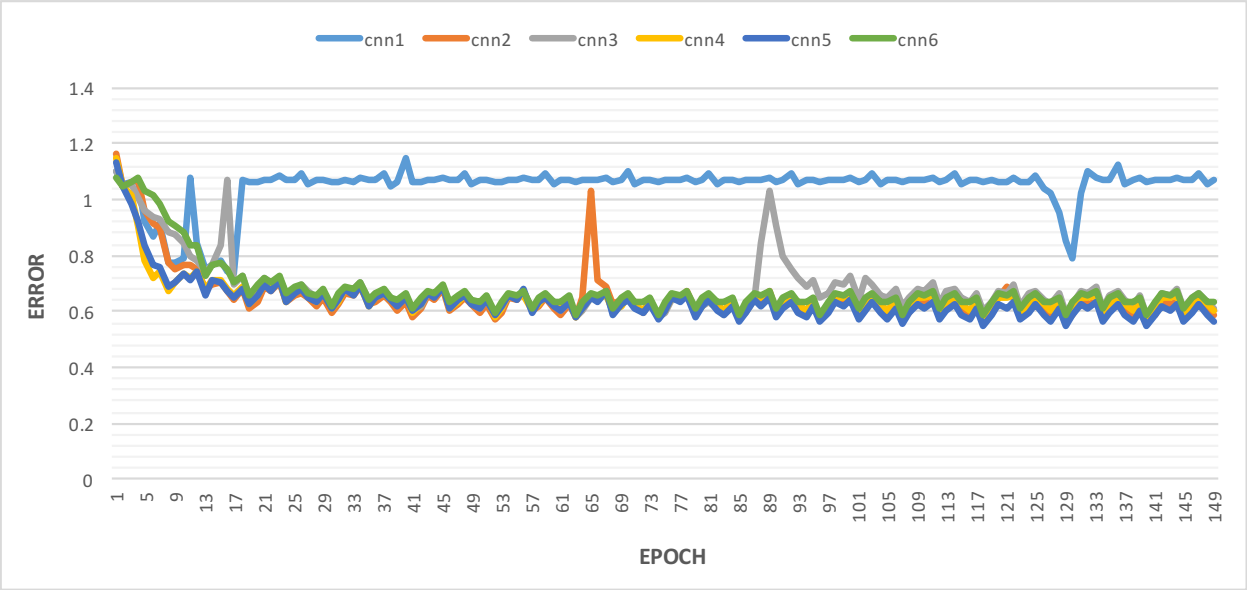
Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη: όπως καθορίστηκε στο πλάνο προς εκτέλεση

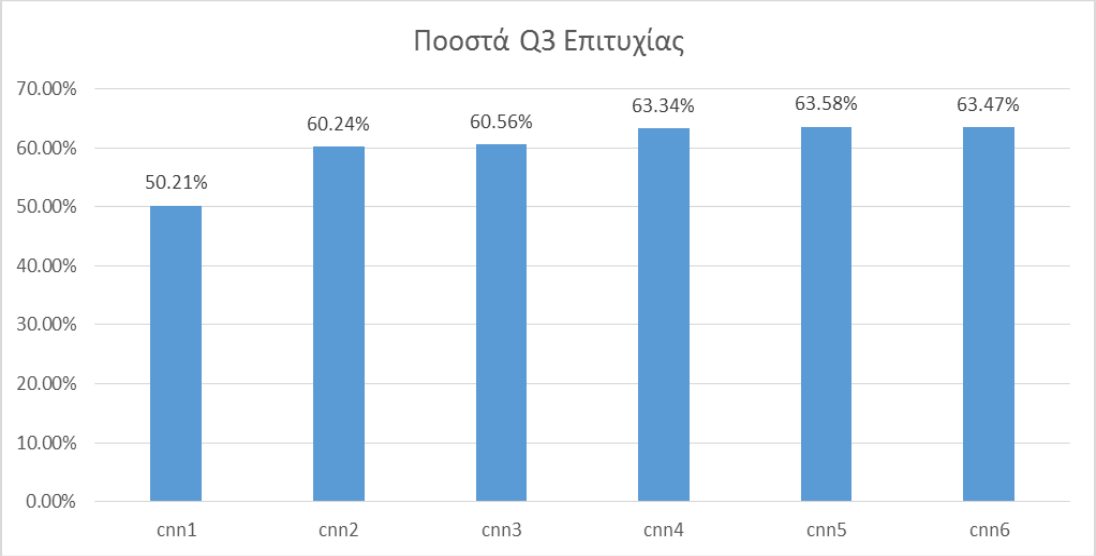
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.14: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με χρήση διαφορετικών πλάνων εκπαίδευσης.

Από τη γραφική παράσταση 6.14, μπορούμε να δούμε ότι τα νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς (εκτός από το cnn1), λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι το συνελικτικό νευρωνικό δίκτυο cnn5, καταφέρνει να μειώσει το σφάλμα εκπαίδευσης περισσότερο σε σχέση με τα άλλα νευρωνικά δίκτυα. Στη συνέχεια, ακολουθούν τα νευρωνικά δίκτυα cnn4, cnn6, cnn3, cnn2 και cnn1. Από τις γραφικές παραστάσεις μπορούμε να πούμε ότι το σφάλμα εκπαίδευσης για τα δίκτυα cnn5, cnn4, cnn6, cnn3, και cnn2, φαίνεται να μειώνεται σχετικά στα ίδια επίπεδα.



Γραφική Παράσταση 6.15: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο, με χρήση διαφορετικών πλάνων εκπαίδευσης.

Στη γραφική παράσταση 6.15 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής), το νευρωνικό δίκτυο cnn5 με το αρκετά πολύπλοκο πρόγραμμα αλλαγής ρυθμού μάθησης, καταφέρνει να πετύχει το υψηλότερο ποσοστό επιτυχίας Q3, ενώ ακολουθούν τα δίκτυα cnn6, cnn4, cnn3, cnn2 και cnn1. Η διαφορά στα ποσοστά επιτυχίας Q3 για τα δίκτυα cnn3, cnn5 και cnn6 φαίνεται να είναι σχετικά μικρή, κάτι που είναι απόλυτα λογικό αφού οι αλλαγές στα learning rate schedules τους είναι αρκετά μικρές.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις, αποφασίσαμε να προχωρήσουμε τα πειράματα μας χρησιμοποιώντας συνελικτικά νευρωνικά δίκτυα με το learning rate schedule του cnn5, αφού φαίνεται να πετυχαίνουν τα υψηλότερα ποσοστά επιτυχίας Q3 σε σχέση με άλλα νευρωνικά δίκτυα, ενώ ταυτόχρονα έχουν σχεδόν την ίδια ή χαμηλότερη πολυπλοκότητα σε σχέση με τα learning rate schedules των cnn4 και cnn6 αντίστοιχα, τα οποία πετυχαίνουν σχετικά παρόμοια ποσοστά επιτυχίας Q3.

6.2.8 Πειράματα με Χρήση Max Pooling layers

Στις πλείστες αρχιτεκτονικές συνελικτικών νευρωνικών δικτύων, εφαρμόζεται η τεχνική του pooling. Η τεχνική pooling εξηγήθηκε ιδιαίτερα στην ενότητα 3.3.5. Όπως έχουμε ήδη

αναφέρει στην προαναφερθείσα ενότητα (Ενότητα 3.3.5), πολλοί ερευνητές δεν υποστηρίζουν την ιδέα εισαγωγής pooling επιπέδων στα συνελικτικά νευρωνικά δίκτυα, αναφέροντας ότι είναι πιο αποτελεσματικό να χρησιμοποιούμε αρχιτεκτονικές που αποτελούνται μόνο από επαναλαμβανόμενα συνελικτικά στρώματα. Επιπρόσθετα, στη συγκεκριμένη έρευνα στην οποία προσπαθούμε να επιλύσουμε το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, ο αριθμός των πρωτεϊνών που έχουμε για εκπαίδευση του δικτύου χρίζει υψίστης σημασίας. Συνεπώς, λόγω ακριβώς του μικρού αριθμού πρωτεϊνών που έχουμε στην διάθεση μας για εκπαίδευση του νευρωνικού δικτύου, η εισαγωγή pooling επιπέδων μάλλον θα είναι καταστροφική.

Για του λόγου του αληθές, αποφασίσαμε να πραγματοποιήσουμε κάποια πειράματα με, και χωρίς την χρήση pooling επιπέδων, έτσι ώστε να μπορέσουμε να προβούμε πειραματικά στην απόφαση του κατά πόσο θα πρέπει να χρησιμοποιήσουμε ή όχι pooling επίπεδα στο συνελικτικό νευρωνικό μας δίκτυο. Αποφασίσαμε λοιπόν να χρησιμοποιήσουμε δύο ειδών συνελικτικά νευρωνικά δίκτυα. Το πρώτο δίκτυο θα διαθέτει τρία (3) κρυφά συνελικτικά επίπεδα, ενώ το δεύτερο δίκτυο θα διαθέτει τρία (3) κρυφά συνελικτικά επίπεδα με τρία (3) max pooling επίπεδα ενδιάμεσα στα διαδοχικά συνελικτικά επίπεδα. Είναι σημαντικό να αναφέρουμε ότι έχουν πραγματοποιηθεί πειράματα με διάφορους συνδυασμούς και είδη pooling επιπέδων, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού, θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά με τη χρησιμότητα αυτής της τεχνικής, το συνδυασμό, αλλά και το είδος των pooling επιπέδων, που οδηγεί στα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3. Το πλάνο που θα ακολουθήσουμε για την διεξαγωγή των πειραμάτων, φαίνεται πιο κάτω.

<u>Πλάνο προς εκτέλεση:</u>

1. CNN με τρία συνελικτικά επίπεδα και τρία max pooling επίπεδα ενδιάμεσα στα διαδοχικά κρυφά συνελικτικά επίπεδα – cnn1.
2. CNN με τρία συνελικτικά επίπεδα – cnn2.

Οι παράμετροι που θέσαμε στο συνελικτικό νευρωνικό δίκτυο καθώς επίσης και σε κάθε κρυφό συνελικτικό επίπεδο των δικτύων φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 300

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

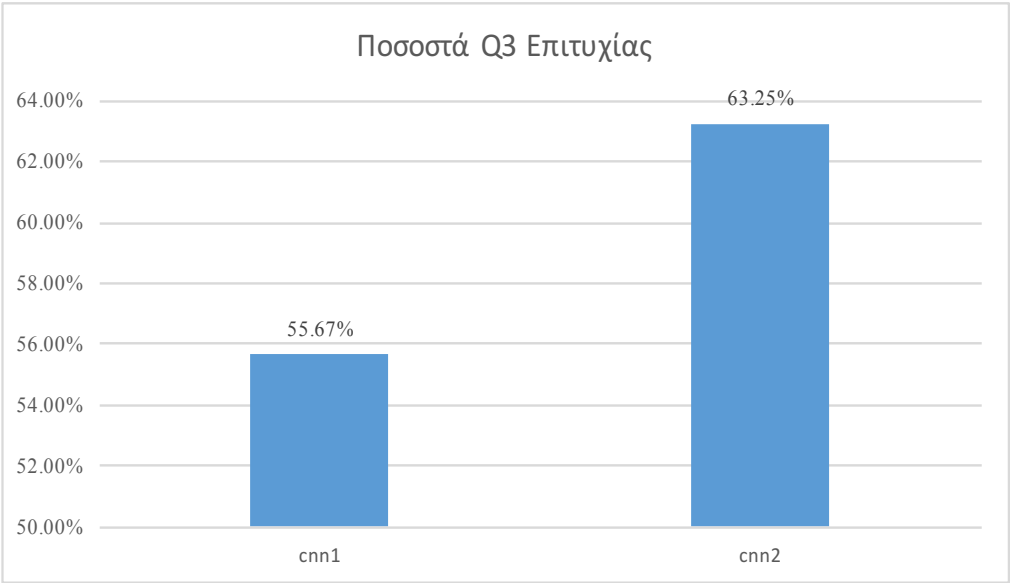
Αρ.Επανάληψης	Ρυθμός μάθησης
1	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5

Γραφική Παράσταση 6.16: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο με (cnn1), και χωρίς max pooling layers (cnn2).

Από τη γραφική παράσταση 6.16, μπορούμε να δούμε ότι τα νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς λόγω του ότι το σφάλμα εκπαίδευσης μειώνεται. Φαίνεται ότι το συνελικτικό νευρωνικό δίκτυο το οποίο ***δεν κάνει*** χρήση pooling επιπέδων – cnn2, καταφέρνει να μειώσει το σφάλμα εκπαίδευσης περισσότερο σε σχέση με το νευρωνικό δίκτυο το οποίο ***κάνει*** χρήση pooling επιπέδων – cnn1 (max pooling layers).



Γραφική Παράσταση 6.17: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο με (cnn1), και χωρίς max pooling layers (cnn2).

Στη γραφική παράσταση 6.17 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής και των απόψεων στο θέμα εισαγωγής ή όχι pooling επιπέδων σε αρχιτεκτονικές συνελικτικών νευρωνικών δικτύων), το νευρωνικό δίκτυο cnn2 το οποίο δεν διαθέτει pooling επίπεδα, καταφέρνει να πετύχει το υψηλότερο ποσοστό επιτυχίας Q3, με αρκετά μεγάλη διαφορά από το νευρωνικό δίκτυο cnn1, το οποίο κάνει χρήση τριών (3) max pooling επιπέδων ενδιάμεσα στα κρυφά συνελικτικά επίπεδα.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις, αποφασίσαμε να προχωρήσουμε τα πειράματα μας χωρίς την χρήση pooling επιπέδων/στρωμάτων, αφού η απουσία των επιπέδων αυτών, φαίνεται να οδηγεί το συνελικτικό νευρωνικό δίκτυο σε επίτευξη υψηλότερων ποσοστών επιτυχίας στην πρόβλεψη Q3.

6.2.9 Πειράματα με Χρήση Διαφορετικών Μεθόδων Ενημέρωσης του Ρυθμού Μάθησης (Updaters)

Στην ενότητα 3.3.8, εξηγήσαμε με λεπτομέρεια κάποιες συγκεκριμένες μεθόδους διαχείρισης του ρυθμού μάθησης (updaters), ο οποίος ρυθμός μάθησης, είναι ένας εξαιρετικά σημαντικός παράγοντας που συμβάλει στην ενημέρωση των συναπτικών βαρών ενός νευρωνικού δικτύου. Ο ρυθμός μάθησης συμβάλει σημαντικά στην ταχύτητα εκπαίδευσης και την τελική «ποιότητα» της εκπαίδευσης του νευρωνικού δικτύου. Αυτές οι μέθοδοι διαχείρισης του ρυθμού μάθησης, έχουν εφαρμογή σε διάφορες μεθόδους εύρεσης ολικού ελαχίστου, όπως η μέθοδος κατάβασης κλίσης. Για να μπορέσουμε να αποφανθούμε για την καλύτερη μέθοδο διαχείρισης του ρυθμού μάθησης, του συνελικτικού νευρωνικού δικτύου που έχουμε δημιουργήσει, αποφασίσαμε να εκτελέσουμε τα ακόλουθα πειράματα που φαίνονται στο πλάνο προς εκτέλεση.

<u>Πλάνο προς εκτέλεση:</u>

1. CNN με χρήση μεθόδου κατάβασης κλίσης – cnn1.
2. CNN με χρήση μεθόδου κατάβασης κλίσης και ορμής – cnn2.
3. CNN με χρήση μεθόδου κατάβασης κλίσης και μεθόδου Adagrad – cnn3.
4. CNN με χρήση μεθόδου κατάβασης κλίσης και μεθόδου RMSProp – cnn4.

5. CNN με χρήση μεθόδου κατάβασης κλίσης και μεθόδου AdaDelta – cnn5.
6. CNN με χρήση μεθόδου κατάβασης κλίσης και μεθόδου ADAM – cnn6.

Οι παράμετροι που θέσαμε στο συνελικτικό νευρωνικό δίκτυο καθώς επίσης και σε κάθε κρυφό συνελικτικό επίπεδο των δικτύων φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 550

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

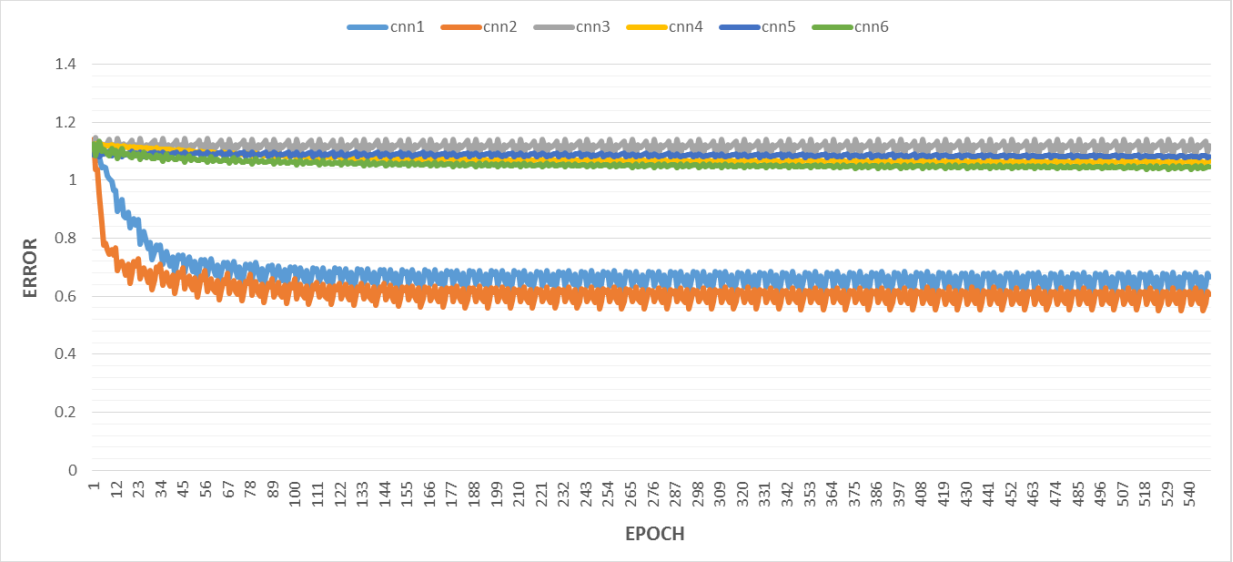
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
2	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

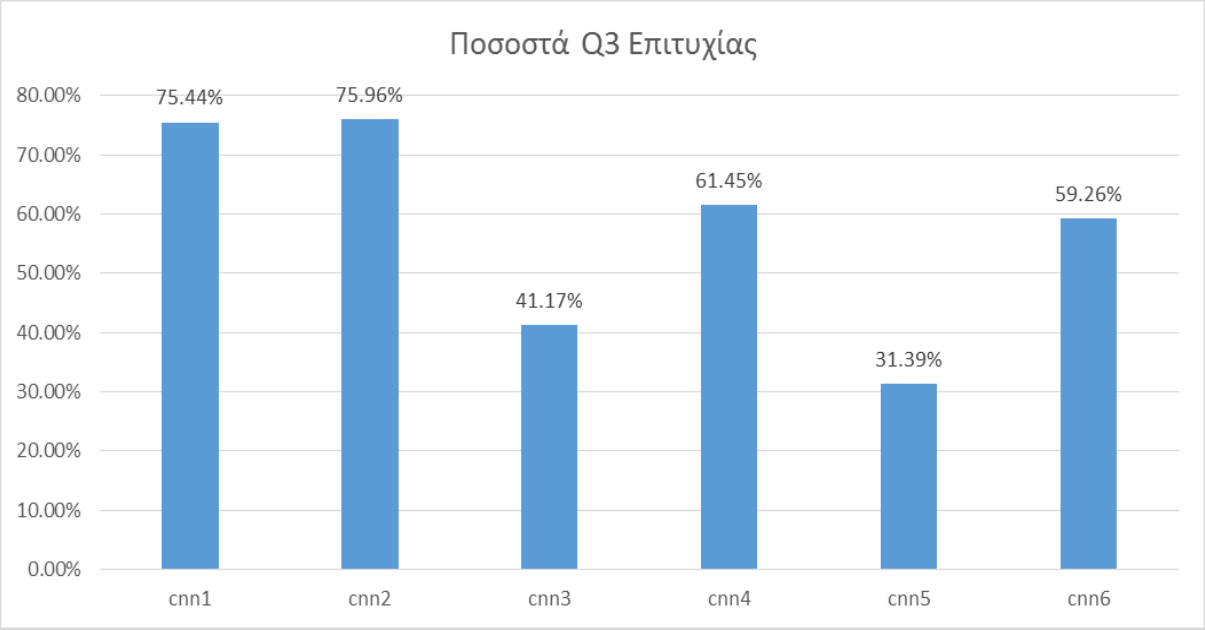
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.18: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με διαφορετικά updaters.

Από τη γραφική παράσταση 6.18, μπορούμε να δούμε ότι τα νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι τα συνελικτικά νευρωνικά δίκτυα cnn1 και cnn2, καταφέρνουν να μειώσουν το σφάλμα περισσότερο από τα υπόλοιπα δίκτυα (cnn3, cnn4, cnn5 και cnn6). Το νευρωνικό δίκτυο που καταφέρνει να μειώσει περισσότερο το σφάλμα φαίνεται να είναι το δίκτυο το οποίο χρησιμοποιεί σαν συνάρτηση ελαχιστοποίησης του σφάλματος τη μέθοδο κατάβασης κλίσης, με χρήση του παράγοντα της ορμής – cnn2. Φαίνεται ότι ***το δίκτυο cnn2***, όχι μόνο ***καταφέρνει να μειώσει περισσότερο το σφάλμα*** αλλά ***έχει και ταχύτερη σύγκλιση*** σε σχέση με το δίκτυο cnn1, το οποίο χρησιμοποιεί μόνο τη μέθοδο κατάβασης κλίσης χωρίς τον παράγοντα της ορμής.



Γραφική Παράσταση 6.19: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο, με διαφορετικά updaters.

Στη γραφική παράσταση 6.19 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής), το νευρωνικό δίκτυο cnn2 το οποίο χρησιμοποιεί σαν συνάρτηση ελαχιστοποίησης του σφάλματος τη μέθοδο κατάβασης κλίσης με χρήση του παράγοντα ορμής, καταφέρνει να πετύχει το υψηλότερο ποσοστό επιτυχίας Q3, με αρκετά μεγάλη διαφορά από τα νευρωνικά δίκτυα cnn3-5 και με μικρή διαφορά από το δίκτυο cnn1, το οποίο κάνει χρήση της μεθόδου κατάβασης κλίσης χωρίς όμως τον παράγοντα της ορμής.

Τέλος, λαμβάνοντας υπόψιν τις πιο πάνω γραφικές παραστάσεις, αποφασίσαμε να προχωρήσουμε τα πειράματα μας με χρήση της μεθόδου κατάβασης κλίσης σαν συνάρτηση ελαχιστοποίησης του σφάλματος και επιπλέον του παράγοντα της ορμής, αφού αυτή η μέθοδος ενημέρωσης του ρυθμού μάθησης φαίνεται να οδηγεί το συνελικτικό νευρωνικό δίκτυο σε επίτευξη υψηλότερων ποσοστών επιτυχίας στην πρόβλεψη Q3.

6.2.10 Συμπεράσματα για Πειράματα με Χρήση Αρχείων MSA

Η διεξαγωγή του κάθε πειράματος που πραγματοποιήσαμε είχε ένα προκαθορισμένο σκοπό. Ο μεγάλος αριθμός παραμέτρων ρύθμισης των συνελικτικών νευρωνικών δικτύων καθιστά το

πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, ένα κατά τα άλλα, παραμετρικό πρόβλημα. Η εύρεση του καλύτερου δυνατού συνδυασμού παραμέτρων ο οποίος οδηγεί στα υψηλότερα ποσοστά επιτυχίας Q3, είναι ένα πρόβλημα εξαιρετικά υψηλής πολυπλοκότητας.

Αρχικά, πραγματοποιώντας το πείραμα της ενότητας 6.2.2, για εύρεση του ιδανικού αριθμού κρυφών συνελικτικών επιπέδων που θα ήταν καλό να εφαρμόσουμε στο συνελικτικό νευρωνικό μας δίκτυο, καταλήξαμε ότι τα δύο, τρία, και τέσσερα κρυφά επίπεδα δίνουν τα καλύτερα αποτελέσματα όσον αφορά ποσοστά επιτυχίας στην πρόβλεψη Q3, χρόνους εκπαίδευσης και απαιτήσεις μνήμης. Αποφασίσαμε λοιπόν, να προσκολληθούμε στη χρήση ***τριών κρυφών συνελικτικών επιπέδων*** για μεταγενέστερα πειράματα στην πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών με συνελικτικά νευρωνικά δίκτυα, αφού φαίνεται να είναι μια καλή επιλογή χρόνου εκπαίδευσης και ποσοστών επιτυχίας Q3.

Προχωρώντας, στην ενότητα 6.2.3 έχουμε προβεί σε πειράματα για καθορισμό του πιο αποδοτικού μεγέθους kernel, για εξαγωγή χαρακτηριστικών από τα δεδομένα εισόδου. Καταλήξαμε ότι το ***μέγεθος kernel 2x2*** σε κάθε κρυφό συνελικτικό επίπεδο, φαίνεται να είναι το ιδανικό σε θέματα ποσοστών επιτυχίας Q3 και χρόνου εκπαίδευσης.

Στην συνέχεια, στην ενότητα 6.2.4 και με τα πειράματα τα οποία πραγματοποιήσαμε, προσπαθήσαμε να μελετήσουμε και να καταγράψουμε την ιδανικότερη τιμή του παράγοντα της ορμής. Μετά από αξιολόγηση των αποτελεσμάτων και των γραφικών παραστάσεων αποφανθήκαμε ότι η τιμή ***ορμής 0.85*** φαίνεται να αποδίδει καλύτερα σε σχέση με άλλες τιμές.

Ακολούθως, στην ενότητα 6.2.5, επιχειρήσαμε να εντοπίσουμε τον καλύτερο παράγοντα μετακίνησης του kernel, δηλαδή το stride. Μετά από εκτενή μελέτη και πειράματα που πραγματοποιήθηκαν, αποφανθήκαμε ότι η τιμή ***stride 1,1*** οδηγεί το συνελικτικό νευρωνικό δίκτυο στην επίτευξη των υψηλότερων ποσοστών επιτυχίας στην πρόβλεψη Q3.

Ο αριθμός των παράλληλων φίλτρων/kernels για εξαγωγή χαρακτηριστικών από τα δεδομένα εισόδου, είναι μια από τις σημαντικότερες παραμέτρους ενός κρυφού συνελικτικού επιπέδου. Κάθε διαφορετικό φίλτρο σε ένα κρυφό επίπεδο ψάχνει για διαφορετικά χαρακτηριστικά στα

δεδομένα εισόδου. Στην ενότητα 6.2.6, ασχοληθήκαμε αποκλειστικά με την προσπάθεια εξεύρεσης του πιο αποδοτικού αριθμού παράλληλων φίλτρων εξαγωγής χαρακτηριστικών σε κάθε κρυφό συνελικτικό επίπεδο, όσον αφορά το ποσοστό επιτυχίας στην πρόβλεψη Q3, τις απαιτήσεις μνήμης και τους χρόνους εκπαίδευσης. Αποφασίστηκε όπως χρησιμοποιηθούν ***πέντε (5) παράλληλα φίλτρα εξαγωγής χαρακτηριστικών*** σε κάθε κρυφό συνελικτικό επίπεδο, αφού φαίνεται να οδηγούν στα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3 έχοντας ταυτόχρονα τους πιο χαμηλούς χρόνους εκπαίδευσης.

Έπειτα, στην ενότητα 6.2.7, προχωρήσαμε με το να πραγματοποιήσουμε πειράματα τα οποία έχουν ως απώτερο σκοπό την εξεύρεση του ιδανικότερου learning rate schedule που μπορεί να εφαρμοστεί. Αφού καθορίσαμε κάποια προγράμματα αλλαγής του ρυθμού μάθησης ανάλογα με τον αριθμό επανάληψης, έχουμε προβεί στην εκτέλεση συγκεκριμένων πειραμάτων. Τα αποτελέσματα των πειραμάτων αυτών έδειχναν ότι ***το learning rate schedule του cnn5,*** φαίνεται να είναι το ιδανικότερο για την προσπάθεια επίλυσης του προβλήματος της πρόβλεψης δευτεροταγούς δομής πρωτεϊνών.

Learning rate schedule του cnn5:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

Προχωρώντας στην πρώτη πειραματική φάση, αποφασίσαμε να εκτελέσουμε κάποια πειράματα για να μπορέσουμε να δούμε αν η προσθήκη επιπέδων υποδειγματοληψίας (pooling layers), βελτιώνει ή όχι τα ποσοστά επιτυχίας στην πρόβλεψη Q3. Αποφανθήκαμε λόγω των τρανταχτών αποτελεσμάτων που πήραμε, ότι ***δεν θα χρησιμοποιήσουμε επίπεδα υποδειγματοληψίας*** στα συνελικτικά νευρωνικά δίκτυα που θα δημιουργήσουμε για την προσπάθεια επίλυσης του συγκεκριμένου προβλήματος το οποίο μελετούμε σε αυτήν την έρευνα.

Τέλος, κλείνοντας την πρώτη πειραματική φάση, επιχειρήσαμε να πραγματοποιήσουμε κάποια πειράματα για εύρεση της καλύτερης μεθόδου (όσων αφορά τα ποσοστά επιτυχίας Q3) ενημέρωσης του ρυθμού μάθησης. Συμπεράναμε ότι η μέθοδος κατάβασης κλίσης συμπεριλαμβάνοντας τον παράγοντα της ορμής, φαίνεται να είναι η καλύτερη τεχνική ενημέρωσης του ρυθμού μάθησης, αφού από τα πειράματα που πραγματοποιήσαμε, αυτή η τεχνική, φαίνεται να συγκλίνει γρηγορότερα σε σχέση με τις άλλες μεθόδους και ταυτόχρονα να πετυχαίνει τα υψηλότερα ποσοστά επιτυχίας Q3.

Συνοψίζοντας τις αποφάσεις μας μετά από τα πειράματα που πραγματοποιήσαμε, σημειώνουμε τα πιο κάτω:

Αριθμός κρυφών συνελικτικών επιπέδων νευρωνικού δικτύου: τρία (3)

Μέγεθος και διαστάσεις kernel σε κάθε κρυφό συνελικτικό επίπεδο: 2x2

Παράγοντας ορμής: 0.85

Παράγοντας stride: 1,1

Αριθμός παράλληλων φίλτρων εξαγωγής χαρακτηριστικών σε κάθε κρυφό επίπεδο: πέντε (5)

Learning rate schedule:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

Χρήση επιπέδων υποδειγματοληψίας: Όχι

Μέθοδος ελαχιστοποίησης του σφάλματος: Μέθοδος κατάβασης κλίσης

Μέθοδος ενημέρωσης του ρυθμού μάθησης: Ορμή

6.3 Πειράματα με Convolution Μεγάλων Gabor Φίλτρων

6.3.1 Περιγραφή

Για τα συγκεκριμένα πειράματα αποφασίσαμε όπως χρησιμοποιήσουμε κάποια μεγάλα φίλτρα Gabor, στο μέγεθος των αρχείων εισόδου του νευρωνικού δικτύου, για εξαγωγή διάφορων χαρακτηριστικών από τα δεδομένα εισόδου. Με αυτό τον τρόπο, χρησιμοποιώντας τα μεγάλα φίλτρα Gabor τα οποία δημιουργήσαμε σε συνδυασμό με την τεχνική του convolution (dot product), θα μπορέσουμε να δημιουργήσουμε κάποια άλλα, νέα αρχεία εισόδου, εμπλουτισμένης μορφής, τα οποία θα περιέχουν στοιχεία και χαρακτηριστικά τα οποία έχουμε εντοπίσει χρησιμοποιώντας τα μεγάλα φίλτρα Gabor (φίλτρα στο μέγεθος των αρχείων εισόδου). Στην συνέχεια, θα προχωρήσουμε με το να εκπαιδεύσουμε και να επαληθεύσουμε το συνελικτικό νευρωνικό δίκτυο το οποίο δημιουργήσαμε, χρησιμοποιώντας τα νέα δεδομένα εισόδου. Είναι σημαντικό να αναφέρουμε ότι έχουμε πραγματοποιήσει πειράματα με πολλά διαφορετικά «μεγάλα» φίλτρα Gabor για εξαγωγή χαρακτηριστικών, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων, σχετικά με τη χρησιμότητα των «μεγάλων» φίλτρων Gabor.

6.3.2 Πειραματικό Μέρος

Η λειτουργία των Gabor φίλτρων καθώς επίσης και οι παράμετροι δημιουργίας ενός Gabor kernel/φίλτρου, έχουν εξηγηθεί με λεπτομέρεια στο κεφάλαιο 4. Για την δημιουργία των νέων εμπλουτισμένων αρχείων εκπαίδευσης, χρησιμοποιήσαμε τα εξής φίλτρα Gabor:

1. gabor_fn(200, 0, 2, 0, 1)
2. gabor_fn(200, 30, 2, 0, 1)
3. gabor_fn(200, 60, 2, 0, 1)
4. gabor_fn(200, 90, 2, 0, 1)
5. gabor_fn(200, 120, 2, 0, 1)
6. gabor_fn(200, 150, 2, 0, 1)
7. gabor_fn(200, 180, 2, 0, 1)

Τα νέα αρχεία εισόδου που λάβαμε μετά την εφαρμογή των συγκεκριμένων Gabor φίλτρων είναι επταπλάσιου μεγέθους σε σχέση με τα αρχικά δεδομένα εισόδου. Ακολούθως,

προχωρήσαμε με το να εκπαιδεύσουμε ένα συνελικτικό νευρωνικό δίκτυο με τα νέα – εμπλουτισμένα δεδομένα εισόδου (cnn1) και το συγκρίναμε με ένα συνελικτικό νευρωνικό δίκτυο εκπαιδευμένο με τα κανονικά δεδομένα εισόδου (cnn2). Το πλάνο εκτέλεσης και τα αποτελέσματα που λάβαμε, φαίνονται πιο κάτω.

Πλάνο προς εκτέλεση:

1. CNN με χρήση νέων/εμπλουτισμένων δεδομένων εισόδου για εκπαίδευση – cnn1.
2. CNN με χρήση αρχικών/κανονικών δεδομένων εκπαίδευσης – cnn2.

Οι παράμετροι που θέσαμε στα δύο συνελικτικά νευρωνικά δίκτυα καθώς επίσης και σε κάθε κρυφό συνελικτικό επίπεδο των δικτύων αυτών, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 150

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Στοχαστική μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

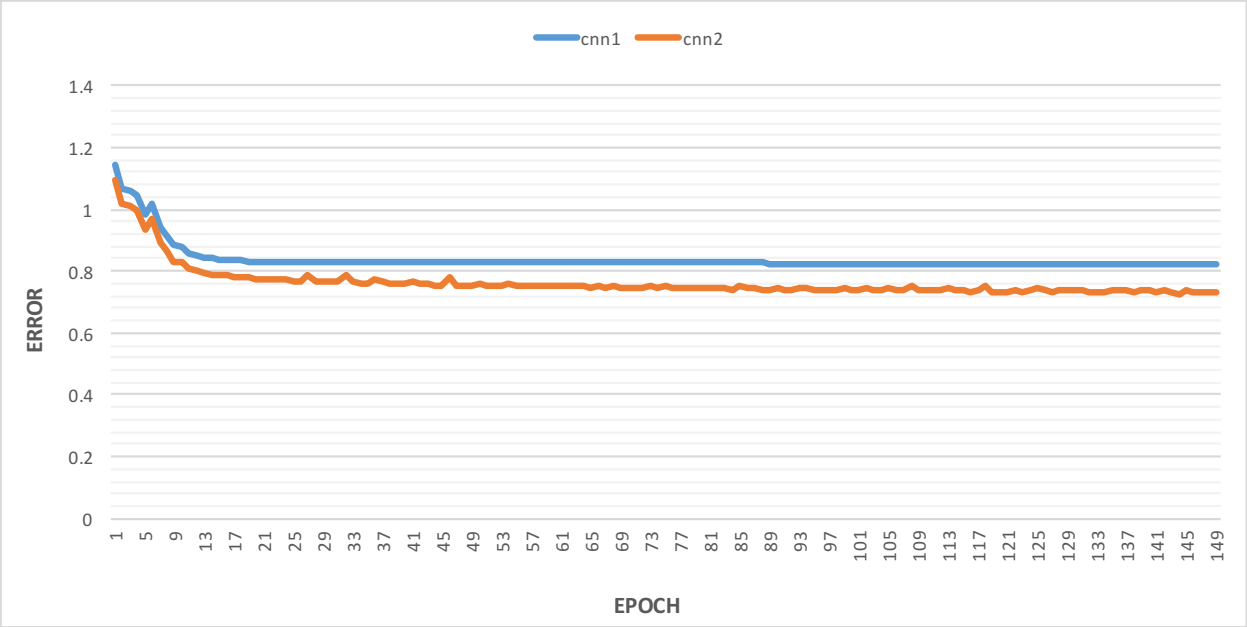
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

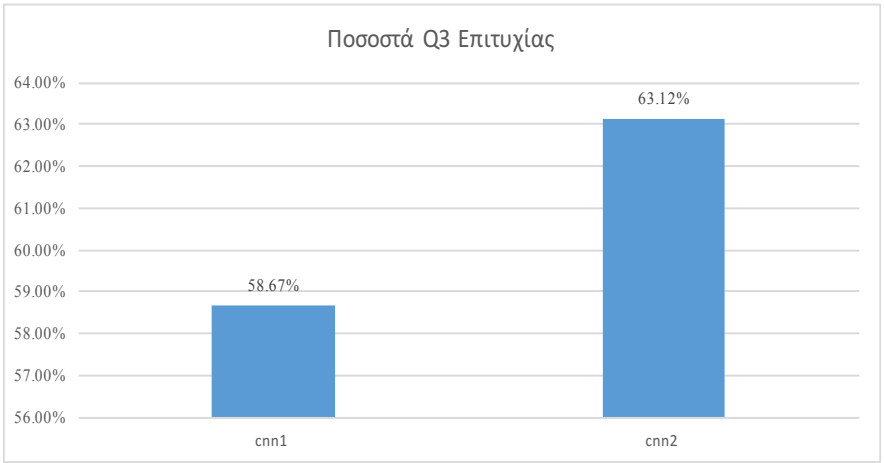
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.20: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο με (cnn1), και χωρίς (cnn2), χρήση μεγάλων φίλτρων Gabor.

Από τη γραφική παράσταση 6.20, μπορούμε να δούμε ότι τα δύο νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι το συνελικτικό νευρωνικό δίκτυο το οποίο χρησιμοποιεί τα αρχικά/κανονικά δεδομένα εκπαίδευσης – cnn2, καταφέρνει να μειώσει το σφάλμα εκπαίδευσης περισσότερο σε σχέση με το νευρωνικό δίκτυο το οποίο χρησιμοποιεί τα νέα/εμπλουτισμένα δεδομένα εκπαίδευσης – cnn1.



Γραφική Παράσταση 6.21: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο με (cnn1), και χωρίς (cnn2), χρήση μεγάλων φίλτρων Gabor.

Στη γραφική παράσταση 6.21 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης), το νευρωνικό δίκτυο cnn2 το οποίο χρησιμοποιεί τα αρχικά/κανονικά δεδομένα εκπαίδευσης, καταφέρνει να πετύχει ψηλότερο ποσοστό επιτυχίας Q3, σε σχέση με το νευρωνικό δίκτυο cnn1, το οποίο χρησιμοποιεί τα νέα/εμπλουτισμένα δεδομένα εκπαίδευσης μετά από την εφαρμογή των μεγάλων φίλτρων Gabor.

6.3.3 Συμπεράσματα

Λαμβάνοντας υπόψιν τα πιο πάνω αποτελέσματα και τις γραφικές παραστάσεις, καταλήξαμε στο ότι η χρήση μεγάλων Gabor φίλτρων (φίλτρων στο μέγεθος των δεδομένων εισόδου) δεν είναι χρήσιμη και δεν θα εφαρμοστεί στη συνέχεια, αφού δεν οδηγεί σε βελτίωση των ποσοστών επιτυχίας στην πρόβλεψη Q3. Αυτό φαίνεται έντονα στη γραφική παράσταση 6.21, αφού το νευρωνικό δίκτυο το οποίο εκπαιδεύεται με τα αρχικά/κανονικά δεδομένα εκπαίδευσης (cnn2), πετυχαίνει υψηλότερα ποσοστά επιτυχίας σε σχέση με το νευρωνικό δίκτυο το οποίο εκπαιδεύεται με τα νέα/εμπλουτισμένα δεδομένα εισόδου μετά από εφαρμογή των Gabor φίλτρων (cnn1).

6.4 Πειράματα με Convolution kernels Gabor Φίλτρων

6.4.1 Περιγραφή

Για τα συγκεκριμένα πειράματα αποφασίσαμε όπως χρησιμοποιήσουμε κάποια kernels Gabor φίλτρων με συγκεκριμένο μέγεθος (μικρότερο από το μέγεθος των δεδομένων εισόδου), για εξαγωγή διάφορων χαρακτηριστικών από τα δεδομένα εισόδου, μετακινώντας τα κάθε φορά ένα συγκεκριμένο αριθμό θέσεων με απώτερο σκοπό την σάρωση όλης της επιφάνειας των δεδομένων εισόδου. Με αυτό τον τρόπο, χρησιμοποιώντας αυτά τα Gabor kernels τα οποία δημιουργήσαμε σε συνδυασμό με την τεχνική του convolution (dot product) και μετακινώντας τα κάποιες θέσεις κάθε φορά, θα μπορέσουμε να δημιουργήσουμε κάποια άλλα, νέα αρχεία εισόδου εμπλουτισμένης μορφής, τα οποία θα περιέχουν στοιχεία και χαρακτηριστικά τα οποία έχουμε εντοπίσει χρησιμοποιώντας κάποια Gabor kernels (φίλτρα Gabor με ένα συγκεκριμένο μέγεθος). Η λογική εφαρμογής αυτών των φίλτρων, είναι παρόμοια με το

παράδειγμα εφαρμογής ενός φίλτρου από ένα κρυφό συνελικτικό επίπεδο (Ενότητα 3.3.5.3). Στην συνέχεια, θα προχωρήσουμε με το να εκπαιδεύσουμε και να επαληθεύσουμε το συνελικτικό νευρωνικό δίκτυο το οποίο δημιουργήσαμε χρησιμοποιώντας τα νέα δεδομένα εισόδου. Είναι σημαντικό να αναφέρουμε ότι έχουμε πραγματοποιήσει πειράματα με πολλά διαφορετικά kernels Gabor φίλτρων για εξαγωγή χαρακτηριστικών, τα οποία όμως δεν θα παρουσιάσουμε στην συγκεκριμένη έρευνα. Αντ' αυτού θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων, σχετικά με τη χρησιμότητα του convolution με kernels Gabor φίλτρων.

6.4.2 Πειραματικό Μέρος

Η λειτουργία των Gabor φίλτρων καθώς επίσης και οι παράμετροι δημιουργίας ενός Gabor kernel/φίλτρου έχουν εξηγηθεί με λεπτομέρεια στο κεφάλαιο 4. Για την δημιουργία των νέων εμπλουτισμένων αρχείων εκπαίδευσης, χρησιμοποιήσαμε τα εξής φίλτρα Gabor/kernels:

1. gabor_fn(0.3, 0, 2, 0, 1)
2. gabor_fn(0.3, 30, 2, 0, 1)
3. gabor_fn(0.3, 60, 2, 0, 1)
4. gabor_fn(0.3, 90, 2, 0, 1)
5. gabor_fn(0.3, 120, 2, 0, 1)
6. gabor_fn(0.3, 150, 2, 0, 1)
7. gabor_fn(0.3, 180, 2, 0, 1)

Τα νέα αρχεία εισόδου που λάβαμε μετά την εφαρμογή των συγκεκριμένων Gabor φίλτρων/kernels είναι επταπλάσιου μεγέθους σε σχέση με τα αρχικά δεδομένα εισόδου. Ακολούθως, προχωρήσαμε με το να εκπαιδεύσουμε ένα συνελικτικό νευρωνικό δίκτυο με τα νέα – εμπλουτισμένα δεδομένα εισόδου (cnn1) και το συγκρίναμε με ένα συνελικτικό νευρωνικό δίκτυο εκπαιδευμένο με τα κανονικά δεδομένα εισόδου (cnn2). Τα αποτελέσματα που λάβαμε, φαίνονται πιο κάτω.

<u>Πλάνο προς εκτέλεση:</u>

1. CNN με χρήση νέων/εμπλουτισμένων δεδομένων εισόδου για εκπαίδευση – cnn1.
2. CNN με χρήση αρχικών/κανονικών δεδομένων εκπαίδευσης – cnn2.

Οι παράμετροι που θέσαμε στα δύο συνελικτικά νευρωνικά δίκτυα καθώς επίσης και σε κάθε κρυφό συνελικτικό επίπεδο των δικτύων αυτών, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 200

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

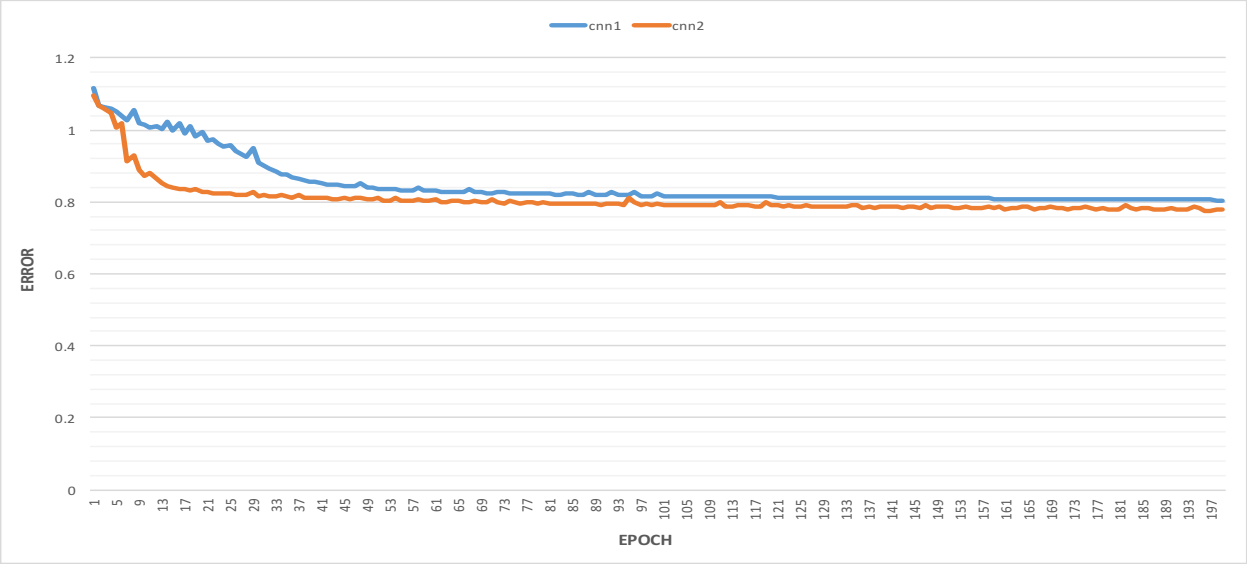
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

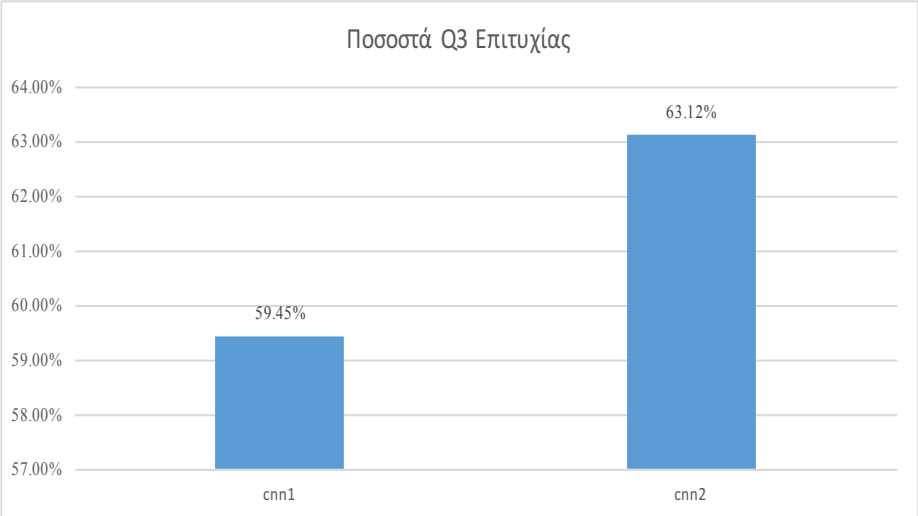
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zerp-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.22: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με (cnn1), και χωρίς (cnn2), χρήση convolution kernels Gabor φίλτρων.

Από τη γραφική παράσταση 6.22, μπορούμε να δούμε ότι τα δύο νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς λόγω του ότι το σφάλμα εκπαίδευσης μειώνεται. Φαίνεται ότι το συνελικτικό νευρωνικό δίκτυο το οποίο χρησιμοποιεί τα αρχικά/κανονικά δεδομένα εκπαίδευσης – cnn2, καταφέρνει να μειώσει ξανά το σφάλμα εκπαίδευσης περισσότερο σε σχέση με το νευρωνικό δίκτυο το οποίο χρησιμοποιεί τα νέα/εμπλουτισμένα δεδομένα εκπαίδευσης – cnn1.



Γραφική Παράσταση 6.23: Ποσοστά επιτυχίας Q3, για κάθε νευρωνικό δίκτυο, με (cnn1), και χωρίς (cnn2), χρήση convolution kernels Gabor φίλτρων.

Στη γραφική παράσταση 6.23 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης – εποχής), το νευρωνικό δίκτυο cnn2 το οποίο χρησιμοποιεί τα αρχικά/κανονικά δεδομένα εκπαίδευσης, καταφέρνει να πετύχει ψηλότερο ποσοστό επιτυχίας Q3, σε σχέση με το νευρωνικό δίκτυο cnn1, το οποίο χρησιμοποιεί τα νέα/εμπλουτισμένα δεδομένα εκπαίδευσης μετά από την εφαρμογή των Gabor φίλτρων. Παρόλα αυτά, αξίζει να σημειωθεί ότι το με την εφαρμογή Gabor kernels με convolution, πήραμε καλύτερα ποσοστά επιτυχίας στην πρόβλεψη Q3 (59.45%), παρά με την εφαρμογή των μεγάλων φίλτρων Gabor της ενότητας 6.3 (58.67%).

6.4.3 Συμπεράσματα

Λαμβάνοντας υπόψιν τα πιο πάνω αποτελέσματα και τις γραφικές παραστάσεις, καταλήξαμε στο ότι ***η χρήση του convolution με kernels Gabor φίλτρων*** (φίλτρων μικρότερου μεγέθους σε σχέση με το μέγεθος των δεδομένων εισόδου) ***δεν είναι χρήσιμη και δεν θα εφαρμοστεί στη συνέχεια***, αφού δεν οδηγεί σε βελτίωση των ποσοστών επιτυχίας στην πρόβλεψη Q3. Αυτό φαίνεται έντονα στη γραφική παράσταση 6.23, αφού το νευρωνικό δίκτυο το οποίο εκπαιδεύεται με τα αρχικά/κανονικά δεδομένα εκπαίδευσης (cnn2), πετυχαίνει υψηλότερα ποσοστά επιτυχίας σε σχέση με το νευρωνικό δίκτυο το οποίο εκπαιδεύεται με τα νέα/εμπλουτισμένα δεδομένα εισόδου μετά από εφαρμογή του convolution με kernels Gabor φίλτρων (cnn1).

6.5 Πειράματα με Χρήση Μέσου Όρου Γειτονικών Κελιών

6.5.1 Περιγραφή

Μια ευρέως γνωστή μέθοδος βελτιστοποίησης της εκπαίδευσης ενός τεχνητού νευρωνικού δικτύου είναι η εισαγωγή θορύβου στα δεδομένα εισόδου. Εισάγοντας ένα είδος θορύβου στα δεδομένα εκπαίδευσης προτρέπουμε το νευρωνικό δίκτυο να προσπαθήσει να ξεχωρίσει και να εντοπίσει διάφορα χαρακτηριστικά καταβάλλοντας μεγαλύτερη προσπάθεια και συνεπώς να εκπαιδευτεί πιο ποιοτικά. Η ιδέα μας για μετασχηματισμό των δεδομένων εισόδου έτσι ώστε ο κάθε όρος των δεδομένων εισόδου, να αντικατασταθεί με τον μέσο όρο των αμέσως γειτονικών του όρων, προήλθε εν μέρει από τα πιο πάνω. Τέλος, χρησιμοποιώντας τα αρχικά

δεδομένα εκπαίδευσης έτσι όπως έχουν περιγραφεί στο κεφάλαιο 5 και προσθέτοντας τα νέα δεδομένα εισόδου με το θόρυβο τον οποίον προσθέσαμε, ευελπιστούμε ότι θα λάβουμε υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3. Η συνάρτηση η οποία δημιουργήσαμε για τη δημιουργία των νέων αρχείων εισόδου φαίνεται στο παράρτημα Ε.

6.5.2 Πειραματικό Μέρος

Αποφασίσαμε να προχωρήσουμε στο πειραματικό μέρος με το να εκπαιδεύσουμε δύο συνελικτικά νευρωνικά δίκτυα με την ίδια ακριβώς αρχιτεκτονική, αλλά με τη διαφορά ότι το πρώτο δίκτυο θα εκπαιδευτεί χρησιμοποιώντας τα αρχικά δεδομένα εισόδου (cnn1), ενώ το δεύτερο δίκτυο θα εκπαιδευτεί χρησιμοποιώντας τα αρχικά δεδομένα εισόδου αλλά και τα νέα δεδομένα εισόδου τα οποία έχουν προκύψει μετά την εισαγωγή του θορύβου (cnn2).

Πλάνο προς εκτέλεση:

1. CNN με χρήση αρχικών/κανονικών δεδομένων εκπαίδευσης – cnn1.
2. CNN με χρήση νέων/εμπλουτισμένων δεδομένων εισόδου (με θόρυβο) για εκπαίδευση – cnn2.

Οι παράμετροι που θέσαμε στα δύο συνελικτικά νευρωνικά δίκτυα καθώς επίσης και σε κάθε κρυφό συνελικτικό τους επίπεδο, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 200

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

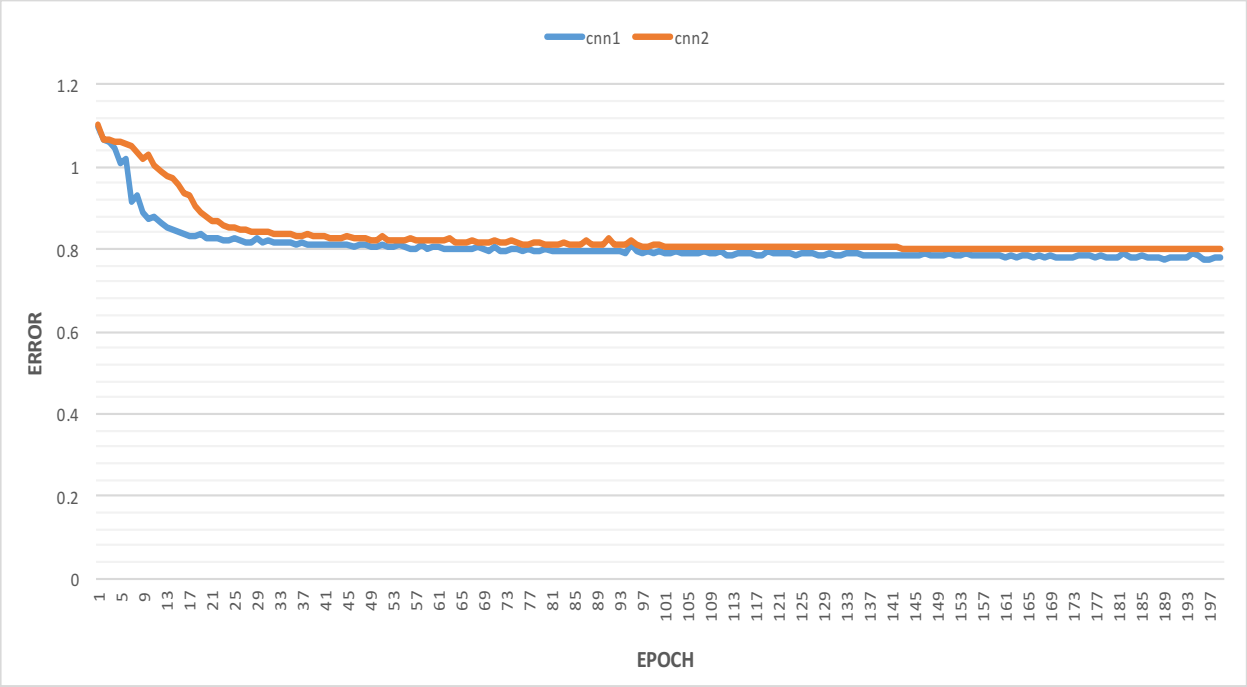
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

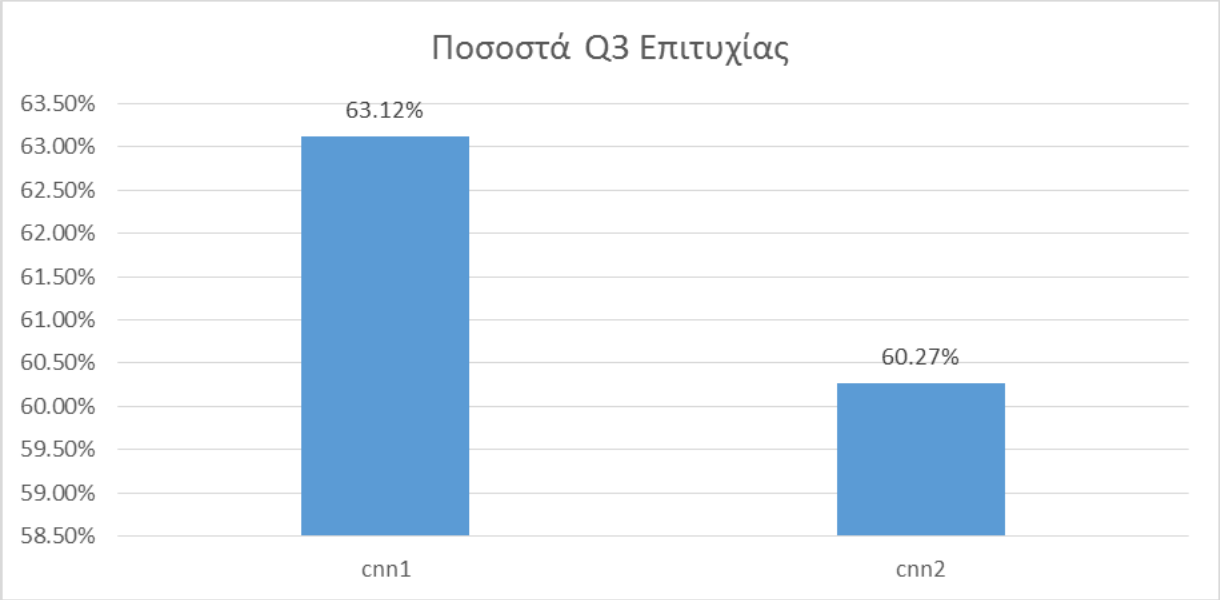
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.24: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο, με χρήση μέσου όρου γειτονικών κελιών.

Από τη γραφική παράσταση 6.24, μπορούμε να δούμε ότι τα δύο νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς λόγω του ότι το σφάλμα εκπαίδευσης μειώνεται και στις δύο περιπτώσεις. Φαίνεται ότι το συνελικτικό νευρωνικό δίκτυο το οποίο χρησιμοποιεί τα αρχικά/κανονικά δεδομένα εκπαίδευσης – cnn1, καταφέρνει να μειώσει το σφάλμα εκπαίδευσης περισσότερο σε σχέση με το νευρωνικό δίκτυο το οποίο χρησιμοποιεί τα νέα/εμπλουτισμένα δεδομένα εκπαίδευσης με θόρυβο – cnn2.



Γραφική Παράσταση 6.25: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο, με χρήση μέσου όρου γειτονικών κελιών.

Στη γραφική παράσταση 6.25 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης), το νευρωνικό δίκτυο cnn1 το οποίο χρησιμοποιεί τα αρχικά/κανονικά δεδομένα εκπαίδευσης, καταφέρνει να πετύχει ψηλότερο ποσοστό επιτυχίας Q3, σε σχέση με το νευρωνικό δίκτυο cnn2, το οποίο χρησιμοποιεί τα νέα δεδομένα εκπαίδευσης με θόρυβο.

6.5.3 Συμπεράσματα

Λαμβάνοντας υπόψιν τα πιο πάνω αποτελέσματα και τις γραφικές παραστάσεις, καταλήξαμε στο ότι ο μετασχηματισμός των δεδομένων εισόδου με αντικατάσταση του κάθε όρου με τον μέσο όρο όλων των αμέσως γειτονικών του όρων δεν είναι τελικά χρήσιμος και δεν θα εφαρμοστεί στη συνέχεια, αφού δεν οδηγεί σε βελτίωση των ποσοστών επιτυχίας στην πρόβλεψη Q3. Αυτό φαίνεται έντονα στη γραφική παράσταση 6.25, αφού το νευρωνικό δίκτυο το οποίο εκπαιδεύεται με τα αρχικά/κανονικά δεδομένα εκπαίδευσης (cnn1), πετυχαίνει υψηλότερα ποσοστά επιτυχίας σε σχέση με το νευρωνικό δίκτυο το οποίο εκπαιδεύεται με τα νέα δεδομένα εισόδου μετά από την εισαγωγή θορύβου (cnn2).

6.6 Πειράματα με Χρήση Τεχνικής Σημαντικότερων Γειτονικών Αμινοξέων

6.6.1 Περιγραφή

Όπως αναφέραμε εκτενώς στην ενότητα 5.8, η έμπνευση μας για την εφαρμογή της τεχνικής των σημαντικότερων γειτονικών αμινοξέων προέρχεται κυρίως από την ιδιότητα που έχουν τα αμινοξέα να επηρεάζουν το ένα το άλλο ανάλογα με την απόσταση που έχουν μεταξύ τους. Αυτές οι αλληλεπιδράσεις μεταξύ των αμινοξέων προκαλούν τη δημιουργία τοπικών αναδιπλώσεων της πολυπεπτιδικής αλυσίδας (δευτεροταγής δομή) μιας πρωτεΐνης. Εφαρμόζοντας τη συγκεκριμένη τεχνική, το νευρωνικό δίκτυο μπορεί να πραγματοποιήσει προβλέψεις για τη δευτεροταγή δομή ενός αμινοξέος, λαμβάνοντας υπόψιν όχι μόνο πληροφορίες για το συγκεκριμένο αμινοξύ, αλλά και για τα γειτονικά του. Αυτή μας η άποψη έχει ενισχυθεί περισσότερο και από την έρευνα των Wang et al. (2016), στην οποία η αρχιτεκτονική του νευρωνικού δικτύου που έχουν δημιουργήσει φαίνεται να λαμβάνει υπόψιν τη συγκεκριμένη ιδιότητα.

Για να μπορέσουμε να εκπαιδεύσουμε ένα συνελικτικό νευρωνικό δίκτυο χρησιμοποιώντας τη συγκεκριμένη τεχνική, πρέπει να μετασχηματίσουμε τα δεδομένα εκπαίδευσης και επαλήθευσης. Για αυτό το σκοπό, έχουμε δημιουργήσει δύο συναρτήσεις σε γλώσσα προγραμματισμού Java, τις create_csv_files_test_Xneighboring_technique(int windowNum) και create_csv_files_train_Xneighboring_technique(int windowNum), οι οποίες δημιουργούν κάποια νέα αρχεία εκπαίδευσης και επαλήθευσης στα οποία η κάθε εγγραφή περιέχει τα MSA records των windowNum DIV 2 αριστερά γειτονικών αμινοξέων (αν υπάρχουν), το MSA record για το αμινοξύ που μελετούμε, τα MSA records των windowNum DIV 2 δεξιά αμινοξέων (αν υπάρχουν), καθώς επίσης και την ετικέτα του αμινοξέος που μελετούμε στο τέλος της κάθε εγγραφής. Το μέγεθος των αρχείων εκπαίδευσης και επαλήθευσης, εξαρτάται αποκλειστικά από το εύρος θέασης (windowNum) που θέτει ο χρήστης σε κάθε εκτέλεση αυτών των συναρτήσεων. Ο κώδικας για τις δύο αυτές συναρτήσεις φαίνεται στο παράρτημα Α.

Είναι σημαντικό να αναφέρουμε ότι έχουμε πραγματοποιήσει πολλά διαφορετικά πειράματα με διαφορετικό εύρος θέασης, τα οποία όμως δεν θα παρουσιάσουμε στη συγκεκριμένη έρευνα. Αντ' αυτού θα παρουσιάσουμε ένα αντιπροσωπευτικό παράδειγμα των πειραμάτων αυτών, για εξαγωγή γενικών συμπερασμάτων σχετικά με τη χρησιμότητα αυτής της τεχνικής και το εύρος θέασης που δίνει τα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3.

Τέλος, τα πειράματα που θα παρουσιάσουμε στην συνέχεια, είναι με **χρήση του dataset CB513**, εκτός και αν γίνει αναφορά ότι χρησιμοποιήθηκε το PISCES dataset ή κάποιο άλλο σύνολο δεδομένων εκπαίδευσης και επαλήθευσης.

6.6.2 Πειραματικό Μέρος

Αποφασίσαμε να προχωρήσουμε στο πειραματικό μέρος με το να εκπαιδεύσουμε ένα συγκεκριμένο αριθμό συνελικτικών νευρωνικών δικτύων με την ίδια ακριβώς αρχιτεκτονική, αλλά με τη διαφορά ότι θα χρησιμοποιούν διαφορετικά αρχεία εκπαίδευσης και επαλήθευσης ανάλογα με το εύρος θέασης του κάθε νευρωνικού δικτύου. Για αυτό το λόγο, αποφασίσαμε να ακολουθήσουμε το πιο κάτω πλάνο για εκτέλεση των πειραμάτων μας.

Πλάνο προς εκτέλεση:

1. CNN με χρήση αρχικών αρχείων εκπαίδευσης (1 αμινοξύ ανά εγγραφή) – cnn1.
2. CNN με εύρος θέασης τριών (3) αμινοξέων – cnn2.
3. CNN με εύρος θέασης πέντε (5) αμινοξέων – cnn3.
4. CNN με εύρος θέασης επτά (7) αμινοξέων – cnn4.
5. CNN με εύρος θέασης εννέα (9) αμινοξέων – cnn5.
6. CNN με εύρος θέασης δεκαπέντε (15) αμινοξέων – cnn6.
7. CNN με εύρος θέασης δεκαεπτά (19) αμινοξέων – cnn7.
8. CNN με εύρος θέασης εικοσιένα (21) αμινοξέων – cnn8.

Αξίζει να σημειωθεί ότι το εύρος θέασης πρέπει να είναι πάντα μονός αριθμός, αφού συμπεριλαμβάνεται στον αριθμό και το εκάστοτε αμινοξύ που μελετούμε. Οι παράμετροι που θέσαμε σε όλα τα συνελικτικά νευρωνικά δίκτυα καθώς επίσης και σε κάθε κρυφό συνελικτικό τους επίπεδο, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 550

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

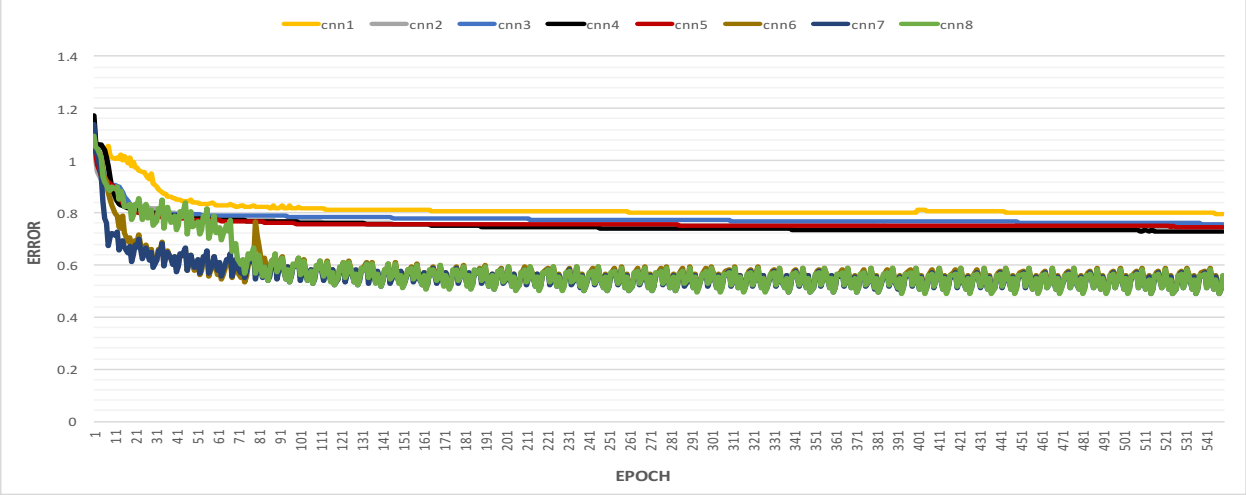
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

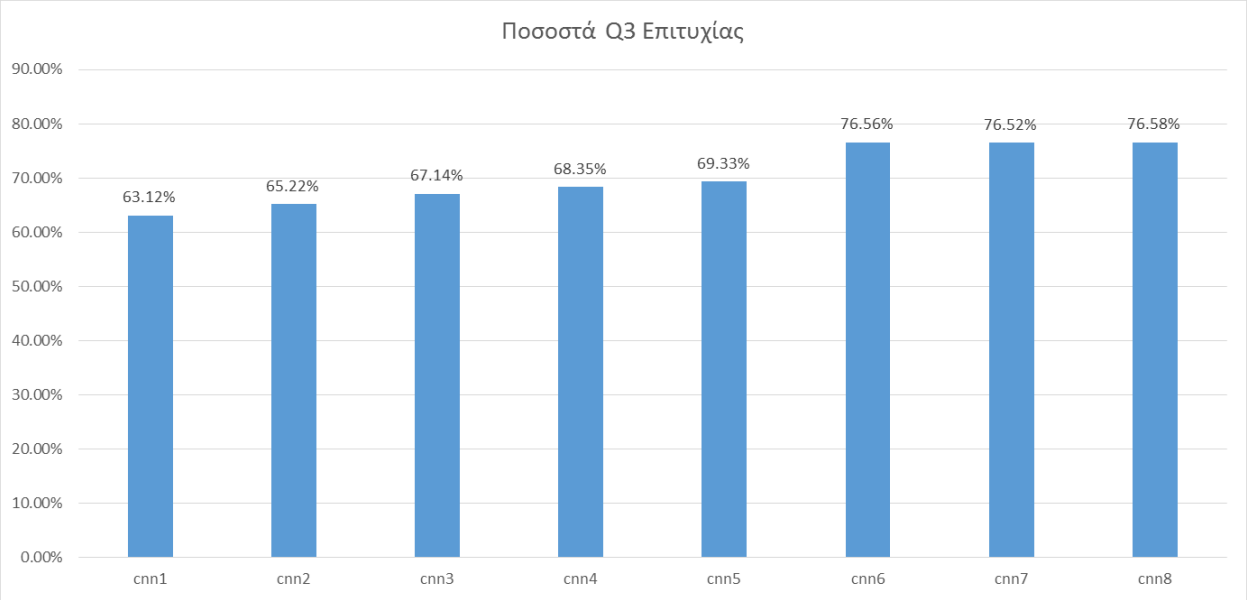
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.26: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο με διαφορετικό εύρος θέασης.

Από τη γραφική παράσταση 6.26, μπορούμε να δούμε ότι όλα τα νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς, λόγω του ότι τα σφάλματα εκπαίδευσης μειώνονται. Φαίνεται ότι τα συγκεκριμένα πειράματα χωρίζονται κάπως σε δύο ομάδες. Η πρώτη ομάδα, η οποία αποτελείται από τα νευρωνικά δίκτυα cnn1, cnn2, cnn3, cnn4 και cnn5, φαίνεται να καταφέρνει να μειώσει το σφάλμα εκπαίδευσης μέχρι ένα συγκεκριμένο σημείο (~0.75). Αντίθετα, η δεύτερη ομάδα η οποία αποτελείται από τα νευρωνικά δίκτυα cnn6, cnn7 και cnn8, φαίνεται να μειώνουν το σφάλμα εκπαίδευσης περισσότερο σε σχέση με τα νευρωνικά δίκτυα της πρώτης ομάδας (~0.58).



Γραφική Παράσταση 6.27: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο με διαφορετικό έυρος θέασης.

Στη γραφική παράσταση 6.27 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης), τα νευρωνικά δίκτυα cnn6, cnn7 και cnn8, καταφέρνουν να πετύχουν τα υψηλότερα ποσοστά επιτυχίας Q3, σε σχέση με υπόλοιπα νευρωνικά δίκτυα (cnn1-5).

6.6.3 Συμπεράσματα

Λαμβάνοντας υπόψιν τα πιο πάνω αποτελέσματα και τις γραφικές παραστάσεις, καταλήξαμε στο ότι τα νέα δεδομένα εκπαίδευσης και επαλήθευσης τα οποία δημιουργήσαμε κάνοντας

χρήση της τεχνικής σημαντικότερων γειτόνων, φαίνεται ότι όντως βοηθούν το δίκτυο να λάβει μια πιο ξεκάθαρη εικόνα για το πρόβλημα, αφού δίνει πολύ καλύτερα αποτελέσματα σε σχέση με το δίκτυο cnn1, το οποίο είναι εκπαιδευμένο με τα αρχικά δεδομένα εκπαίδευσης που αναφέρονται στην ενότητα 5.5. Ο κύριος λόγος για τον οποίο συμβαίνει αυτό, είναι το ότι με τη συγκεκριμένη τεχνική δίνουμε στο δίκτυο περισσότερη πληροφορία έτσι ώστε να μπορεί να προβεί σε μια πιο ακριβή πρόβλεψη για την δευτεροταγή δομή κάθε αμοινοξέος. Η βελτίωση φαίνεται άμεσα μέσα από τα αποτελέσματα του συνελικτικού δικτύου με εύρος θέασης τριών (3) αμινοξέων – cnn2, αφού παίρνουμε ποσοστά επιτυχίας Q3 της τάξης του 65.22%. Φαίνεται ξεκάθαρα από την γραφική παράσταση ποσοστών επιτυχίας Q3 (Γραφική παράσταση 6.27), ότι ενόσω αυξάνουμε το εύρος θέασης του δικτύου τείνουν να αυξάνονται και τα ποσοστά επιτυχίας στην πρόβλεψη Q3.

Αυτή η θεαματική αύξηση φαίνεται ότι διακόπτεται από ένα συγκεκριμένο εύρος θέασης και μετά. Συγκεκριμένα, από το νευρωνικό δίκτυο με εύρος θέασης δεκαπέντε (15) αμινοξέων – cnn6, φαίνεται ότι τα ποσοστά επιτυχίας Q3 παραμένουν τα ίδια. Ένας σοβαρός λόγος μεταξύ άλλων, είναι και το ότι τα αμινοξέα σε απόσταση οκτώ θέσεων από το εκάστοτε αμινοξύ το οποίο μελετούμε, δεν επηρεάζουν σε μεγάλο βαθμό τη δευτεροταγή του δομή. Ένας άλλος λόγος, είναι και το ότι η πολυπλοκότητα των αλληλεπιδράσεων των αμινοξέων από ένα σημείο και έπειτα αυξάνεται σε εξαιρετικά υψηλό βαθμό, έτσι ώστε τα συνελικτικά νευρωνικά δίκτυα που έχουμε δημιουργήσει να μην μπορούν να κωδικοποιήσουν τέτοιου επιπέδου πολύπλοκες συμπεριφορές. Για του λόγου του αληθές, θα πραγματοποιήσουμε στη συνέχεια κάποια πειράματα με χρήση περισσότερων κρυφών συνελικτικών επιπέδων έτσι ώστε να ελέγξουμε αν μπορούμε να λάβουμε κάποια βελτίωση των ποσοστών επιτυχίας Q3.

6.7 Πειράματα με Χρήση της μεθόδου Conjugate Gradient

6.7.1 Περιγραφή

Όπως και η μέθοδος κατάβασης κλίσης, έτσι και η μέθοδος Conjugate Gradient χρησιμοποιείται για την εύρεση του ολικού ελαχίστου της συναρτήσεως του σφάλματος. Ο αλγόριθμος conjugate gradient έχει σχετικά χαμηλό υπολογιστικό κόστος και ανήκει στους

αλγόριθμους δεύτερης τάξης (Burney et al., 2004). Αποφασίσαμε λοιπόν, να χρησιμοποιήσουμε τη συγκεκριμένη μέθοδο έτσι ώστε να δούμε αν μπορούμε να λάβουμε υψηλότερα ποσοστά επιτυχίας Q3, αλλά και να μειώσουμε τον χρόνο εκπαίδευσης του συνελικτικού νευρωνικού δικτύου χρησιμοποιώντας αυτή την τεχνική.

6.7.2 Πειραματικό Μέρος

Αποφασίσαμε να προχωρήσουμε στο πειραματικό μέρος με το να εκπαιδεύσουμε δύο, ίδιας αρχιτεκτονικής, συνελικτικά νευρωνικά δίκτυα, με δύο διαφορετικούς αλγόριθμους βελτιστοποίησης. Το πρώτο νευρωνικό δίκτυο – cnn1, θα εκπαιδευτεί με την μέθοδο κατάβασης κλίσης ενώ το δεύτερο νευρωνικό δίκτυο – cnn2, θα εκπαιδευτεί με την μέθοδο conjugate gradient. Για αυτό το λόγο, αποφασίσαμε να ακολουθήσουμε το πιο κάτω πλάνο για εκτέλεση των πειραμάτων μας.

Πλάνο προς εκτέλεση:

1. CNN με χρήση της μεθόδου κατάβασης κλίσης – cnn1.
2. CNN με χρήση της μεθόδου conjugate gradient – cnn2.

Οι παράμετροι που θέσαμε σε όλα τα συνελικτικά νευρωνικά δίκτυα καθώς επίσης και σε κάθε κρυφό συνελικτικό τους επίπεδο, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 300

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: όπως ορίζεται στο πιο πάνω πλάνο

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

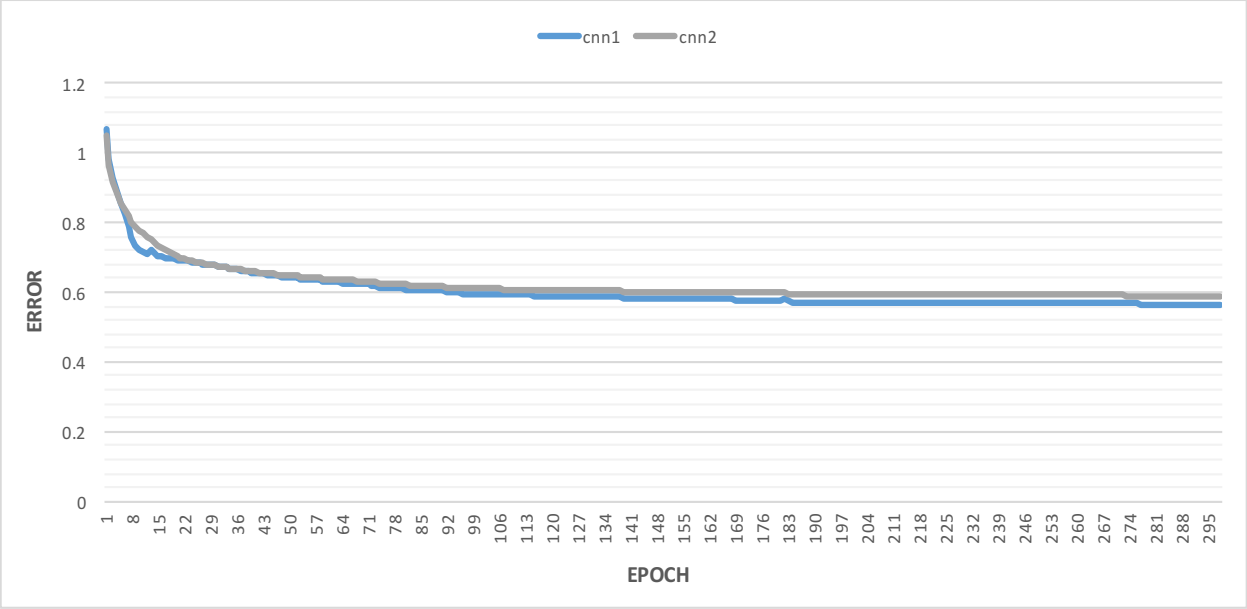
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

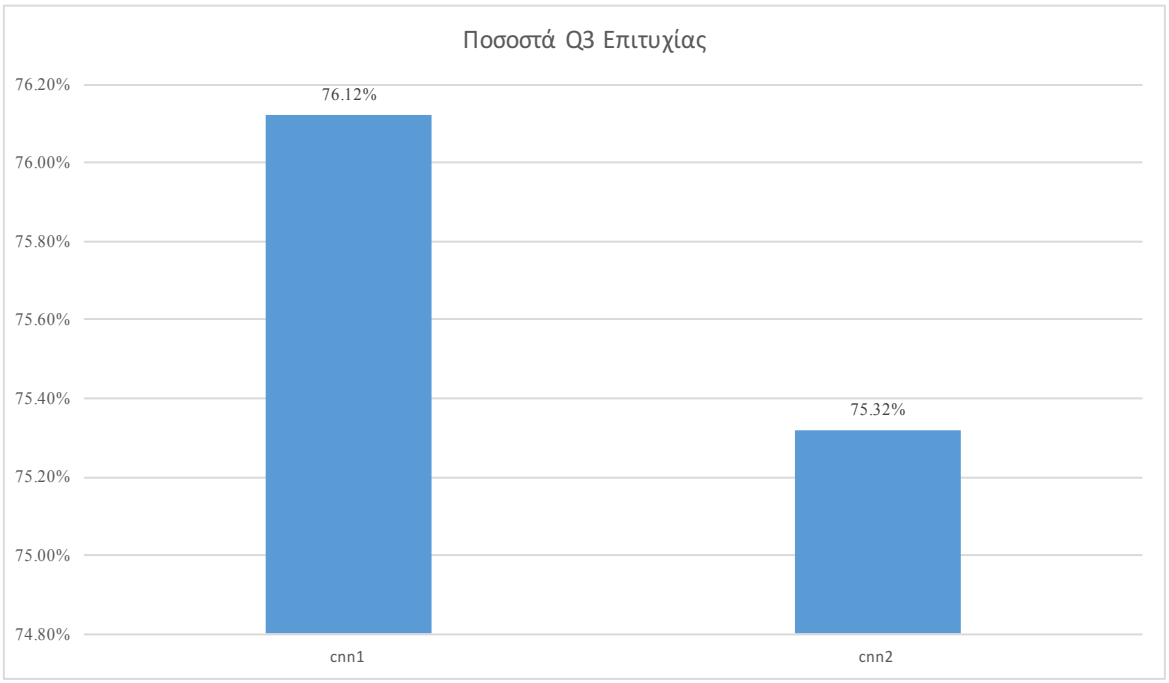
Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.28: Σφάλμα εκπαίδευσης – εποχής για κάθε νευρωνικό δίκτυο με χρήση της μεθόδου κατάβασης κλίσης (cnn1) και της μεθόδου conjugate gradient (cnn2).

Από τη γραφική παράσταση 6.28, μπορούμε να δούμε ότι τα δύο συνελικτικά νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς, λόγω του ότι το σφάλμα εκπαίδευσης μειώνεται και στις δύο περιπτώσεις. Φαίνεται ότι το νευρωνικό δίκτυο που χρησιμοποιεί την μέθοδο κατάβασης κλίσης για ως μέθοδο ελαχιστοποίησης του σφάλματος – cnn1, καταφέρνει να μειώσει το σφάλμα εκπαίδευσης ελαφρώς περισσότερο σε σχέση με το νευρωνικό δίκτυο το

οποίο χρησιμοποιεί τη μέθοδο conjugate gradient ως μέθοδο ελαχιστοποίησης του σφάλματος.



Γραφική Παράσταση 6.29: Ποσοστά επιτυχίας Q3 για κάθε νευρωνικό δίκτυο με χρήση της μεθόδου κατάβασης κλίσης (cnn1) και της μεθόδου conjugate gradient (cnn2).

Στη γραφική παράσταση 6.29 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης - εποχής), το νευρωνικό δίκτυο cnn1, το οποίο χρησιμοποιεί την μέθοδο κατάβασης κλίσης ως μέθοδο ελαχιστοποίησης του σφάλματος, καταφέρνει να πετύχει τα υψηλότερα ποσοστά επιτυχίας Q3, σε σχέση με το νευρωνικό δίκτυο cnn2 το οποίο χρησιμοποιεί την μέθοδο conjugate gradient.

Γραφική Παράσταση 6.30: Απαιτούμενος χρόνος εκπαίδευσης για κάθε νευρωνικό δίκτυο με χρήση της μεθόδου κατάβασης κλίσης (cnn1) και της μεθόδου conjugate gradient (cnn2).

Στη γραφική παράσταση 6.30 μπορούμε να δούμε τους χρόνους εκτέλεσης για κάθε νευρωνικό δίκτυο. Φαίνεται ότι ο χρόνος εκπαίδευσης για το νευρωνικό δίκτυο cnn1 (χρήση μεθόδου κατάβασης κλίσης) είναι μικρότερος σε σχέση με το χρόνο εκπαίδευσης του νευρωνικού δικτύου cnn2 (χρήση μεθόδου conjugate gradient), κάτι που δεν αναμέναμε αφού θεωτηρικά, οι αλγόριθμοι βελτιστοποίησης δεύτερης τάξης εκπαιδεύονται με ταχύτερους ρυθμούς σε σχέση με αλγόριθμους βελτιστοποίησης πρώτης τάξης (Burney et al., 2004). Δύο λόγοι για τους οποίους μπορεί να συμβαίνει αυτό, είναι η χρήση του αλγορίθμου line search ή ο τρόπος υλοποίησης της μεθόδου στη βιβλιοθήκη DL4J.

6.7.3 Συμπεράσματα

Λαμβάνοντας υπόψιν τα πιο πάνω αποτελέσματα που λάβαμε από τις γραφικές παραστάσεις, συνειδητοποιούμε ότι η χρήση της μεθόδου ελαχιστοποίησης του σφάλματος δεύτερης τάξης conjugate gradient, δεν επιφέρει κάποια βελτίωση στα ποσοστά πρόβλεψης Q3 και έχει επίσης μεγαλύτερους χρόνους εκπαίδευσης για το PSSP πρόβλημα. Με βάση τα πιο πάνω καταλήγουμε στο ότι η μέθοδος conjugate gradient δεν θα χρησιμοποιηθεί για την προσπάθεια επίλυσης του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών με χρήση συνελικτικών νευρωνικών δικτύων.

6.8 Πειράματα με Χρήση του PISCES Dataset

6.8.1 Περιγραφή

Τα πειράματα τα οποία πραγματοποιήσαμε μέχρι και αυτό το σημείο, χρησιμοποιούν το σύνολο δεδομένων πρωτεϊνών CB513. Το συγκεκριμένο σύνολο όπως έχουμε αναφέρει περιέχει πεντακόσιες δεκατρείς (513) μη ομοιογενείς πρωτεϊνικές ακολουθίες. Η ικανότητα γενίκευσης και η γνώση ενός νευρωνικού δικτύου για επίλυση ενός συγκεκριμένου προβλήματος είναι ανάλογη της ποιότητας αλλά και της ποσότητας των δεδομένων εκπαίδευσης και επαλήθευσης. Με τον όρο ποιότητα αναφερόμαστε στο πόσο αντιπροσωπευτική είναι η επιλογή των συγκεκριμένων πρωτεϊνών οι οποίες θα αντιπροσωπεύουν το γενικό σύνολο των πρωτεϊνών και θα δώσουν τη γενική εικόνα στο νευρωνικό δίκτυο για το PSSP πρόβλημα. Αποφασίσαμε λοιπόν, να προετοιμάσουμε και να χρησιμοποιήσουμε το PISCES dataset, για εκπαίδευση και επαλήθευση του συνελικτικού νευρωνικού δικτύου, το οποίο διαθέτει οκτώ χιλιάδες πεντακόσιες (8500) πρωτεϊνικές αλληλουχίες από τις οποίες εξαιρέθηκαν οι τριακόσιες πενήντα (350) όπως αναφέραμε στην ενότητα 5.9. Λόγω του μεγάλου χρονικού διαστήματος που απαιτείται για εκπαίδευση ενός συνελικτικού νευρωνικού δικτύου με το PISCES dataset, αποφασίσαμε να πραγματοποιήσουμε μόνο ένα πείραμα έτσι ώστε να μπορέσουμε να αποκτήσουμε τη γενική εικόνα του νευρωνικού δικτύου, όταν τύχει εκπαίδευσης με τα δύο αυτά datasets.

6.8.2 Πειραματικό Μέρος

Στο πειραματικό μέρος θα εκπαιδεύσουμε δύο συνελικτικά νευρωνικά δίκτυα με τα datasets CB513 και PISCES αντίστοιχα, έτσι ώστε να είμαστε σε θέση να συγκρίνουμε τα ποσοστά επιτυχίας στην πρόβλεψη Q3. Το πρώτο νευρωνικό δίκτυο – cnn1, θα εκπαιδευτεί με το dataset CB513, ενώ το δεύτερο νευρωνικό δίκτυο – cnn2, θα εκπαιδευτεί με το dataset PISCES. Για αυτό το λόγο, αποφασίσαμε να ακολουθήσουμε το πιο κάτω πλάνο για εκτέλεση των πειραμάτων μας.

Πλάνο προς εκτέλεση:

1. Cnn με χρήση του CB513 dataset – cnn1.
2. Cnn με χρήση του PISCES dataset – cnn2.

Οι παράμετροι που θέσαμε σε όλα τα συνελικτικά νευρωνικά δίκτυα καθώς επίσης και σε κάθε κρυφό συνελικτικό τους επίπεδο, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης των CNNs:

Εποχές: 300

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

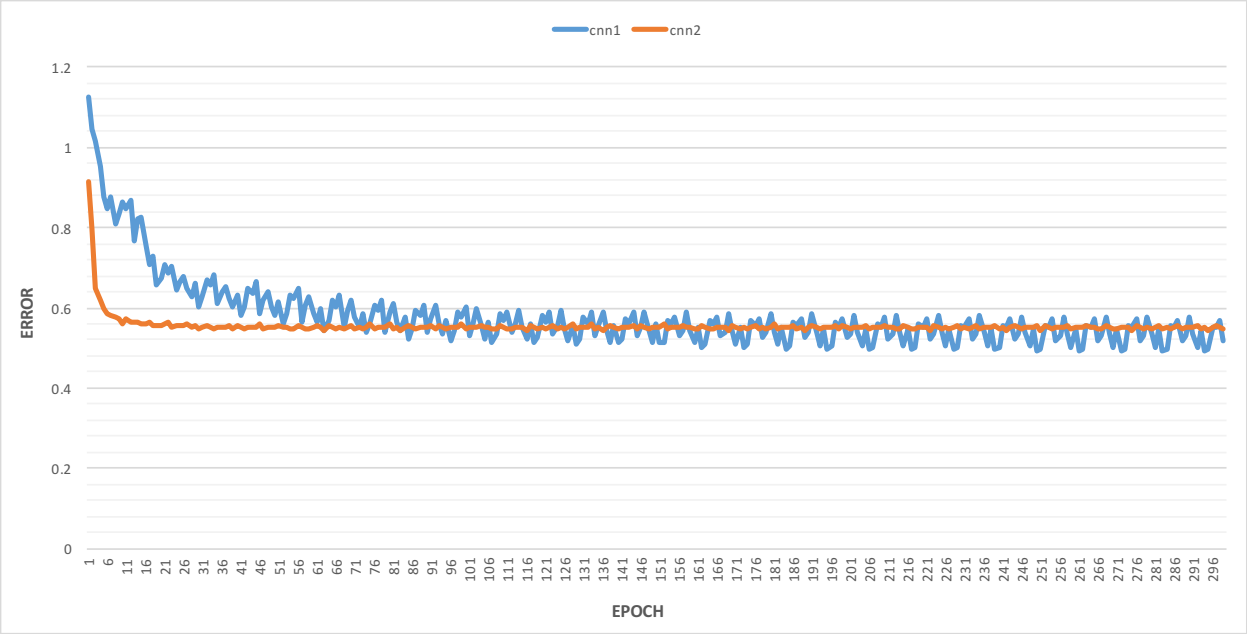
Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

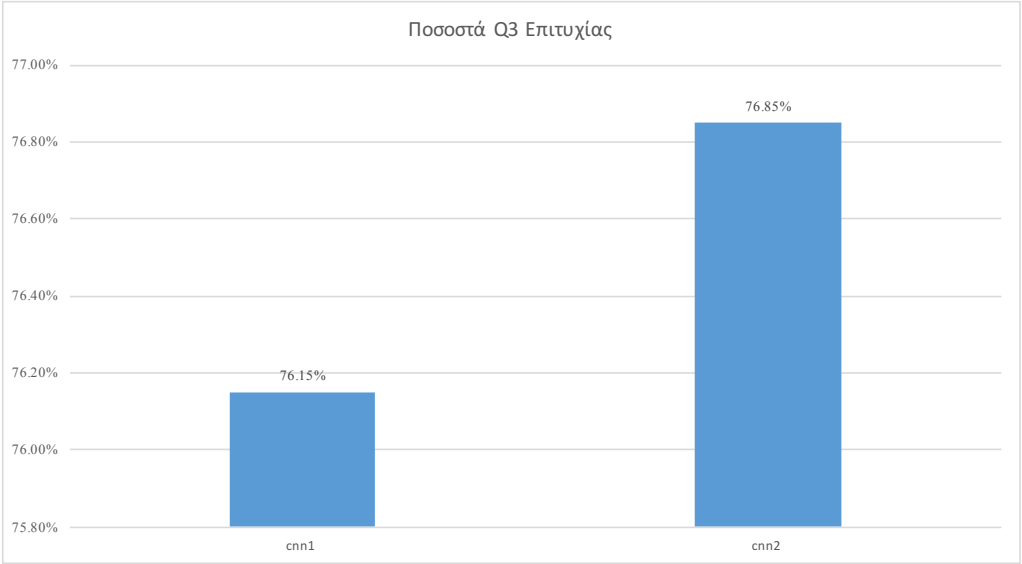
Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5



Γραφική Παράσταση 6.31: Σφάλμα εκπαίδευσης – εποχής για τα δύο δίκτυα cnn1 και cnn2, με datasets εκπαίδευσης τα CB513 και PISCES αντίστοιχα.

Από τη γραφική παράσταση 6.31, μπορούμε να δούμε ότι τα δύο συνελικτικά νευρωνικά δίκτυα φαίνεται να μαθαίνουν επιτυχώς, λόγω του ότι το σφάλμα εκπαίδευσης μειώνεται και στις δύο περιπτώσεις. Φαίνεται ότι τα δύο νευρωνικά δίκτυα cnn1 και cnn2, καταφέρνουν να μειώσουν το σφάλμα, περίπου στα ίδια επίπεδα.



Γραφική Παράσταση 6.32: Ποσοστά επιτυχίας Q3 για τα δύο δίκτυα cnn1 και cnn2, με datasets εκπαίδευσης τα CB513 και PISCES αντίστοιχα.

Στη γραφική παράσταση 6.32 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Όπως ήταν αναμενόμενο (λόγω και της γραφικής παράστασης του σφάλματος εκπαίδευσης), η διαφορά στα ποσοστά επιτυχίας Q3 είναι σχετικά μικρή. Το νευρωνικό δίκτυο cnn2, το οποίο εκπαιδεύτηκε με το σύνολο δεδομένων PISCES, φαίνεται να πετυχαίνει τα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη Q3.

6.8.3 Συμπεράσματα

Χρησιμοποιώντας το μεγαλύτερο σύνολο δεδομένων PISCES φαίνεται να πήραμε καλύτερα ποσοστά επιτυχίας στο δίκτυο κάτι πολύ λογικό και αναμενόμενο αφού δώσαμε στο νευρωνικό δίκτυο μια πιο ευρεία εικόνα του προβλήματος το οποίο προσπαθεί να επιλύσει. Συνεπώς, το νευρωνικό δίκτυο cnn2 αποκτά καλύτερες ικανότητες γενίκευσης σε σχέση με το νευρωνικό δίκτυο cnn1 και καταφέρνει να πετύχει υψηλότερα αποτελέσματα.

6.9 Πειράματα με Συνδυασμό CNN και BRNN

6.9.1 Περιγραφή

Στην προσπάθεια μας να πετύχουμε μεγαλύτερα ποσοστά επιτυχίας στην πρόβλεψη Q3, αποφασίσαμε να συνδυάσουμε τα συνελικτικά νευρωνικά δίκτυα, με μια από τις καλύτερες τεχνικές που έχουν χρησιμοποιηθεί για την επίλυση του PSSP προβλήματος, τα νευρωνικά δίκτυα αμφίδρομης ανάδρασης. Το αρχείο κώδικα για δημιουργία και εκπαίδευση του BRNN, φαίνονται στο παράρτημα Z.

6.9.2 Πειραματικό Μέρος

Αποφασίσαμε στο πειραματικό μέρος να συγκρίνουμε δύο νευρωνικά δίκτυα. Το πρώτο νευρωνικό δίκτυο – cnn1, θα είναι ένα συνελικτικό νευρωνικό δίκτυο το οποίο θα εκπαιδευτεί με τα δεδομένα εισόδου που κατασκευάσαμε στην ενότητα 5.8, ενώ το δεύτερο νευρωνικό δίκτυο – cnn2, θα είναι ένα νευρωνικό δίκτυο αμφίδρομης ανάδρασης και θα εκπαιδευτεί με τα feature vectors που έχουμε εξάγει από το συνελικτικό δίκτυο cnn1. Για αυτό το λόγο, αποφασίσαμε να ακολουθήσουμε το πιο κάτω πλάνο για εκτέλεση των πειραμάτων μας.

Πλάνο προς εκτέλεση:

1. CNN με χρήση των αρχείων εκπαίδευσης σημαντικότερων γειτονικών αμινοξέων (Ενότητα 5.8) – cnn1.
2. BRNN με χρήση των feature vectors που έχουν εξαχθεί από το τελευταίο συνελικτικό επίπεδο του cnn1 – brnn2.

Οι παράμετροι που θέσαμε στα δύο νευρωνικά δίκτυα καθώς επίσης και σε κάθε κρυφό τους επίπεδο, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης του CNN:

Εποχές: 300

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5

Παράμετροι ρύθμισης του BRNN:

Αριθμός νευρώνων πρώτου επιπέδου ανάδρασης: 150

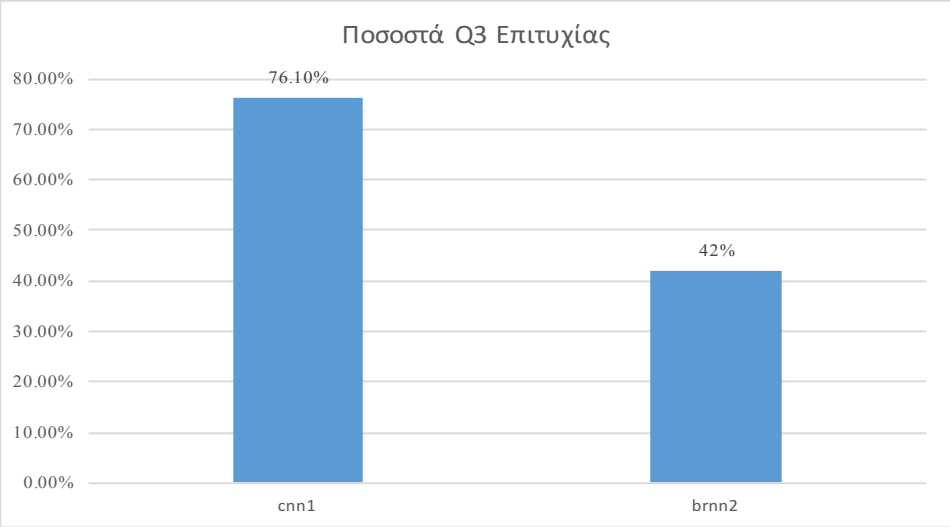
Αριθμός νευρώνων πρώτου επιπέδου ανάδρασης: 150

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Ρυθμός μάθησης: 0.1

Ορμή (από 0-1): 0.8

Συνάρτηση υπολογισμού του σφάλματος: Least Mean Square Error (Εξίσωση 3.2)



Γραφική Παράσταση 6.33: Ποσοστά επιτυχίας Q3 για τα δύο νευρωνικά δίκτυα (cnn1 – CNN, brnn2 - BRNN).

Στη γραφική παράσταση 6.33 μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Το νευρωνικό δίκτυο cnn1 το οποίο χρησιμοποιεί τα κλασικά αρχεία εκπαίδευσης και επαλήθευσης με χρήση της τεχνικής σημαντικότερων γειτονικών αμινοξέων (Ενότητα 5.8), καταφέρνει να πετύχει υψηλότερα ποσοστά επιτυχίας Q3, σε σχέση με το νευρωνικό δίκτυο brnn2 το οποίο χρησιμοποιεί σαν αρχεία εκπαίδευσης τα feature vectors τα οποία έχουν εξαχθεί από το cnn1.

6.9.3 Συμπεράσματα

Βλέποντας την πιο πάνω γραφική παράσταση συνειδητοποιούμε ότι το νευρωνικό δίκτυο αμφίδρομης ανάδρασης πετυχαίνει πολύ πιο χαμηλά ποσοστά επιτυχίας στην πρόβλεψη, κάτι που σημαίνει ότι χρειάζεται να γίνει περαιτέρω έρευνα για εντοπισμό των ιδανικών παραμέτρων ρύθμισης του νευρωνικού δικτύου αμφίδρομης ανάδρασης, όπως επίσης και αναδιαμόρφωση των feature vectors που λαμβάνονται από το τελευταίο κρυφό CL του CNN. Αυτό θα γίνει σε μεταγενέστερο στάδιο λόγω έλλειψης χρόνου.

6.10 Πειράματα με Συνδυασμό CNN και SVM

6.10.1 Περιγραφή

Όπως αναφέραμε στην εισαγωγή αυτής της διατριβής, η τελευταία προσπάθεια για βελτίωση των ποσοστών επιτυχίας στην πρόβλεψη Q3, θα ήταν ο συνδυασμός του συνελικτικού νευρωνικού δικτύου το οποίο έχουμε δημιουργήσει με τα support vector machines. Πιο συγκεκριμένα, αναφέραμε ότι θα χρησιμοποιήσουμε τα SVMs για φιλτράρισμα/διόρθωση των αποτελεσμάτων πρόβλεψης του συνελικτικού νευρωνικού δικτύου, ευελπιστώντας να βελτιώσουμε την ποιότητα των αποτελεσμάτων που λάβαμε. Κατά την εκπαίδευση, η είσοδος του SVM θα είναι ένα παράθυρο των αποτελεσμάτων πρόβλεψης του cnn και η επιθυμητή έξοδος θα είναι η δευτεροταγής δομή του μεσαίου αμινοξέος. Το αρχείο κώδικα για προετοιμασία των αρχείων δεδομένων για φιλτράρισμα από το SVM με χρήση ενός συγκεκριμένου αριθμού παραθύρου θέασης (prepare_SVM_FILES_CSV_FORMAT.py) και το αρχείο κώδικα με την χρήση του SVM (SVM_CORRECTED.py), φαίνονται στο παράρτημα Η.

6.10.2 Πειραματικό Μέρος

Αποφασίσαμε στο πειραματικό μέρος να συγκρίνουμε δύο νευρωνικά δίκτυα. Το πρώτο νευρωνικό δίκτυο – cnn1, θα είναι ένα συνελικτικό νευρωνικό δίκτυο το οποίο θα εκπαιδευτεί με τα δεδομένα εισόδου που κατασκευάσαμε στην ενότητα 5.8, ενώ το δεύτερο νευρωνικό δίκτυο – svm2, θα είναι ένα SVM το οποίο θα λαμβάνει σαν είσοδο τα αποτελέσματα πρόβλεψης του cnn1 και θα τα φιλτράρει/διορθώνει. Για αυτό το λόγο, αποφασίσαμε να ακολουθήσουμε το πιο κάτω πλάνο για εκτέλεση των πειραμάτων μας.

Πλάνο προς εκτέλεση:

1. CNN με χρήση των αρχείων εκπαίδευσης σημαντικότερων γειτονικών αμινοξέων (Ενότητα 5.8) – cnn1.
2. SVM με χρήση αρχείων εκπαίδευσης τα αποτελέσματα πρόβλεψης του cnn1 – svm2.

Οι παράμετροι που θέσαμε στα δύο νευρωνικά δίκτυα, φαίνονται πιο κάτω.

Παράμετροι ρύθμισης του CNN:

Εποχές: 300

Μέγεθος batch δεδομένων εκπαίδευσης: 7000

Μέγεθος batch δεδομένων επαλήθευσης: 7289

Αριθμός batches δεδομένων εκπαίδευσης: 11

Αριθμός batches δεδομένων επαλήθευσης: 1

Μέθοδος ελαχιστοποίησης σφάλματος: Μέθοδος κατάβασης κλίσης

Συνάρτηση υπολογισμού του σφάλματος: Negative Log-Likelihood (Miranda, 2017)

Συνάρτηση ενεργοποίησης νευρώνων: Rectifier συνάρτηση (ReLU)

Ορμή (από 0-1): 0.85

Πλάνο ρύθμισης ρυθμού μάθησης σε κάθε επανάληψη:

Αρ.Επανάληψης	Ρυθμός μάθησης
0	0.08
500	0.005
700	0.001
900	0.0005
1200	0.0001
3000	0.00006

Παράμετροι κάθε κρυφού συνελικτικού επιπέδου:

Kernel Size: 2x2
Stride: 1,1
Zero-Padding: 1,1
Parallel filters: 5

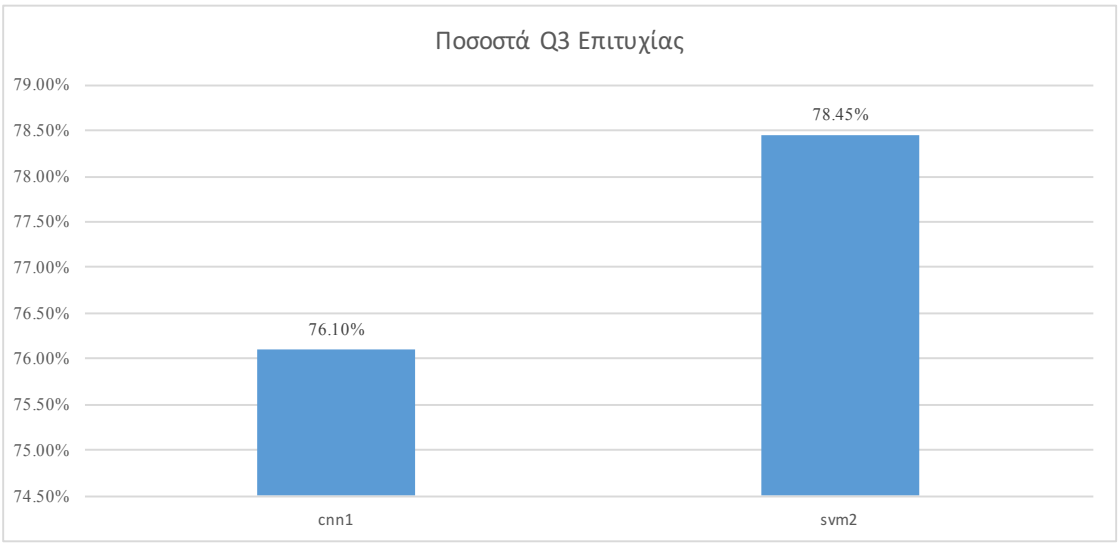
Παράμετροι ρύθμισης του SVM:

Kernel: Radial Basis Function (RBF)

Misclassification penalty parameter (C): 1

Gamma value: 0.001

Παράθυρο θέασης (window): 7



Γραφική Παράσταση 6.33: Ποσοστά επιτυχίας Q3 για τις δύο μεθόδους μηχανικής μάθησης (cnn1 – CNN, svm2 – SVM για φιλτράρισμα των αποτελεσμάτων πρόβλεψης του cnn1).

Στην πιο πάνω γραφική παράσταση μπορούμε να δούμε τα ποσοστά επιτυχίας Q3. Το φιλτράρισμα των αποτελεσμάτων πρόβλεψης του CNN, με χρήση του SVM, φαίνεται ότι όντως βελτιώνει τα ποσοστά επιτυχίας Q3. Αυτό μας χαροποιεί ιδιαίτερα, αφού εκτός του ότι πετύχαμε υψηλότερα ποσοστά επιτυχίας, μπορούμε να ευελπιστούμε και για καλύτερα αποτελέσματα λόγω του ότι στη συνέχεια θα χρησιμοποιήσουμε κάποιες επιπλέον τεχνικές βελτιστοποίησης, όπως είναι τα ensembles και το filtering – external rules, οι οποίες περιγράφονται στο κεφάλαιο 7.

6.10.3 Συμπεράσματα

Βλέποντα την πιο πάνω γραφική παράσταση (Γραφική παράσταση 6.33), μπορούμε να δούμε ότι ο συνδυασμός των δύο αλγορίθμων μηχανικής μάθησης όντως βελτιώνει τα ποσοστά επιτυχίας στην πρόβλεψη. Στο κεφάλαιο 7, θα χρησιμοποιήσουμε κάποιες επιπλέον τεχνικές βελτιστοποίησης όπως είναι τα ensembles και το filtering – external rules, ευελπιστώντας σε ακόμα ψηλότερα ποσοστά επιτυχίας Q3.

Κεφάλαιο 7

Συμπεράσματα και Μελλοντική Εργασία

7.1	Συμπεράσματα	177
7.2	Μελλοντική Εργασία	186

7.1 Συμπεράσματα

Ο κύριος στόχος αυτής της διπλωματικής εργασίας, ήταν η πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών, με τη χρήση συνελικτικών νευρωνικών δικτύων σε συνδυασμό με τα φίλτρα Gabor και τα support vector machines.

Η εφαρμογή και χρήση των συνελικτικών νευρωνικών δικτύων, είναι κατάλληλη για επίλυση προβλημάτων τα οποία έχουν υψηλή πολυπλοκότητα αλληλεξαρτήσεων. Όπως ήδη γνωρίζουμε, τα συνελικτικά νευρωνικά δίκτυα χρησιμοποιούνται κυρίως για ανάλυση και εξαγωγή χαρακτηριστικών από εικόνες. Ο λόγος που συμβαίνει αυτό, είναι κυρίως το ότι τα δίκτυα αυτά μπορούν να κωδικοποιήσουν στην αρχιτεκτονική τους πολύπλοκες συμπεριφορές και ιδιότητες λόγω της χαμηλής συνδεσμολογίας του δικτύου και της εξειδίκευσης του κάθε νευρώνα σε μικρότερους τομείς δεδομένων εισόδου (kernels). Επιπρόσθετα, τα συνελικτικά νευρωνικά δίκτυα ανήκουν στην κατηγορία των βαθιών νευρωνικών δικτύων. Ως αποτέλεσμα, τα δίκτυα αυτά αναλύοντας τα δεδομένα εισόδου σε διαφορετική κλίμακα και εφαρμόζοντας πολλαπλά φίλτρα εξαγωγής δεδομένων σε κάθε επίπεδο, είναι σε θέση να εντοπίσουν πολύπλοκα χαρακτηριστικά στα δεδομένα εισόδου. Προσθέτοντας κρυφά επίπεδα σε ένα συνελικτικό νευρωνικό δίκτυο, αυξάνουμε την ικανότητα του να εντοπίζει πολύπλοκης μορφής χαρακτηριστικά στα δεδομένα εισόδου. Αυτός είναι και ο πραγματικός λόγος, που αυτά τα δίκτυα χρησιμοποιούνται κυρίως σε περιοχές όπως η ανάλυση εικόνας και η υπολογιστική όραση. Εξαιρετικής σημασίας χρίζουν τα αρχεία δεδομένων MSA profiles, τα οποία εισάγουν στα δεδομένα εκπαίδευσης και στο συνελικτικό νευρωνικό δίκτυο ένα είδος εξελικτικής πληροφορίας, με αποτέλεσμα το δίκτυο να είναι σε θέση να γενικεύει καλύτερα και να πραγματοποιεί ακριβέστερες προβλέψεις.

Οπτικοποιώντας τα δεδομένα εκπαίδευσης και επαλήθευσης, είμαστε σε θέση να τα χρησιμοποιήσουμε για να εκπαιδεύσουμε και να επαληθεύσουμε ένα συνελικτικό νευρωνικό δίκτυο για επίλυση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Στη συνέχεια, χρησιμοποιώντας τα φίλτρα Gabor και τις τεχνικές ανακατασκευής των δεδομένων εισόδου τις οποίες αναφέραμε στο κεφάλαιο πέντε (5), δημιουργούμε κάποια νέα, εμπλουτισμένης μορφής δεδομένα εισόδου, τα οποία τα χρησιμοποιήσαμε για

εκπαίδευση των νευρωνικών δικτύων που χρησιμοποιήσαμε, ευελπιστώντας να πετύχουμε υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη.

Η έρευνα μας σχετικά με τα συνελικτικά νευρωνικά δίκτυα, έχει φτάσει σχετικά σε αρκετά καλό σημείο. Τα μεγάλου αριθμού πειράματα τα οποία πραγματοποιήσαμε με διαφορετικούς συνδυασμούς παραμέτρων για τα συνελικτικά νευρωνικά δίκτυά τα οποία έχουμε δημιουργήσει, μας βοήθησαν στο να εντοπίσουμε τον κατάλληλο συνδυασμό τιμών στις παραμέτρους οι οποίες οδηγούν στα υψηλότερα ποσοστά επιτυχίας Q3 και SOV για το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

Το συνελικτικό νευρωνικό δίκτυο που μας οδήγησε στα υψηλότερα ποσοστά επιτυχίας Q3 μετά από 10-fold cross validation, με χρήση του dataset CB513, καθώς επίσης και οι παράμετροι ρύθμισης του κάθε κρυφού του επιπέδου, φαίνονται πιο κάτω:

Εποχές εκπαίδευσης:	250-350
Μέθοδος αποφυγής overfit:	early stopping, l2 regularization (0.005), dropout
Μέθοδος ελαχιστοποίησης σφάλματος:	Μέθοδος κατάβασης κλίσης
Μέγεθος batch εκπαίδευσης:	7000 εγγραφές
Μέγεθος batch επαλήθευσης:	7289 εγγραφές
Αριθμός batches εκπαίδευσης:	11
Αριθμός batches επαλήθευσης:	1
Μέθοδος ενημέρωσης ρυθμού μάθησης:	Ορμή=0.85
Πλάνο ενημέρωσης ρυθμού μάθησης:	Αρ.Επανάληψης Ρυθμός μάθησης 0 0.01 500 0.005 700 0.001 900 0.0005 1200 0.0001 3000 0.00006
Επίπεδο 0:	Είδος επιπέδου: Συνελικτικό κρυφό επίπεδο Μέγεθος kernel: 2x2 Stride: 1,1 Αριθμός παράλληλων φίλτρων: 5 Padding: 1,1 Συνάρτηση ενεργοποίησης νευρώνων: Rectifier (ReLU)
Επίπεδο 1:	Είδος επιπέδου: Συνελικτικό κρυφό επίπεδο Μέγεθος kernel: 2x2 Stride: 1,1

	Αριθμός παράλληλων φίλτρων: 5 Padding: 1,1 Συνάρτηση ενεργοποίησης νευρώνων: Rectifier (ReLU)
Επίπεδο 2:	Είδος επιπέδου: Συνελικτικό κρυφό επίπεδο Μέγεθος kernel: 2x2 Stride: 1,1 Αριθμός παράλληλων φίλτρων: 5 Padding: 1,1 Συνάρτηση ενεργοποίησης νευρώνων: Rectifier (ReLU)
Επίπεδο 3:	Είδος επιπέδου: Επίπεδο εξόδου (MLP) Αριθμός νευρώνων: 3 Συνάρτηση υπολογισμού σφάλματος: Negative log likeliood Συνάρτηση ενεργοποίησης: SOFTMAX

Πίνακας 7.1: Τιμές παραμέτρων οι οποίες οδηγούν στα υψηλότερα ποσοστά επιτυχίας Q3 και SOV σκορ.

Είναι σημαντικό να αναφέρουμε ότι ***τα Gabor φίλτρα φαίνεται να μην είναι ιδιαίτερα αποτελεσματικά και χρήσιμα*** στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, πρώτον λόγω του ότι δεν βελτιώνουν τα ποσοστά επιτυχίας στην πρόβλεψη Q3 και δεύτερον, λόγω του ότι τα συνελικτικά νευρωνικά δίκτυα εκτελούν παρόμοιες λειτουργείες με τα φίλτρα Gabor, σε μεγαλύτερη κλίμακα, πολύ πιο αποδοτικά και πολύ πιο αποτελεσματικά.

Επιπρόσθετα, η τεχνική προσαρμογής των δεδομένων εισόδου η οποία μας οδήγησε στα υψηλότερα ποσοστά επιτυχίας Q3, είναι η τεχνική των ***σημαντικότερων γειτονικών αμινοξέων*** η οποία μας επιτρέπει να πετύχουμε ποσοστά επιτυχίας Q3 της τάξης του 75.15% μετά από πραγματοποίηση 10-fold cross validation (Kohavi, 1995) και SOV 0.713. Στον πίνακα 7.2 φαίνονται αναλυτικά τα υψηλότερα αποτελέσματα Q3 και SOV για κάθε κατηγορία δευτεροταγούς δομής.

Τα αρχεία κώδικα που δημιουργήσαμε για προετοιμασία των αρχείων με την μορφή: όνομα πρωτεϊνης, πρωτοταγής δομή, δευτεροταγής δομή, προβλεπόμενη δευτεροταγής δομή, για την SOV αξιολόγηση φαίνονται στο παράρτημα Γ.

FOLD	Q3%	QH%	QE%	QC%	SOV	SOVH	SOVE	SOVC
0	75.33	72.36	68.24	85.66	0.72	0.72	0.71	0.73
1	75.16	71.23	68.66	84.23	0.71	0.71	0.69	0.73
2	74.54	68.9	67.22	83.37	0.74	0.71	0.67	0.76
3	74.38	69.3	64.01	83.12	0.69	0.68	0.65	0.72
4	76.47	72.33	68.21	84.66	0.67	0.66	0.68	0.72
5	75.36	68.75	66.24	85.67	0.72	0.71	0.67	0.75
6	75.57	68.91	66.55	86.13	0.7	0.67	0.63	0.7
7	74.37	67.32	67.28	83.36	0.7	0.68	0.65	0.71
8	76.35	68.38	69.82	86.57	0.72	0.69	0.65	0.74
9	74.02	67.22	67.16	82.89	0.76	0.73	0.69	0.78
AVERAGE	**75.155%**	69.47	67.339	84.566	**0.713**	0.696	0.669	0.734

Πίνακας 7.2: Ποσοστά επιτυχίας Q3 και SOV, μετά από 10-fold cross-validation με χρήση μόνο του συνελικτικού νευρωνικού δικτύου (CNN).

Κατά την τεχνική του cross validation χωρίζουμε τα δεδομένα εισόδου σε ένα συγκεκριμένο αριθμό υποσυνόλων (στην περίπτωση του 10-fold έχουμε δέκα (10) υποσύνολα). Το πρόγραμμα κάθε φορά εκτελείται έχοντας σαν δεδομένα επαλήθευσης ένα από τα δέκα υποσύνολα και σαν δεδομένα εκπαίδευσης όλα τα υπόλοιπα. Η διαδικασία αυτή επαναλαμβάνεται τόσες φορές όσες και ο αριθμός των υποσυνόλων που έχουμε δημιουργήσει. Ως τελικό αποτέλεσμα παίρνουμε το μέσο όρο των αποτελεσμάτων όλων των εκτελέσεων που πραγματοποιήσαμε. Πραγματοποιώντας την συγκεκριμένη διαδικασία, παίρνουμε πιο αντικειμενικά και με μεγαλύτερη ακρίβεια αποτελέσματα και ποσοστά επιτυχίας.

Ακολούθως, προχωρήσαμε στην εφαρμογή κάποιων αλγορίθμων βελτιστοποίησης των αποτελεσμάτων που λάβαμε. Για παράδειγμα, χρησιμοποιήσαμε την τεχνική ensemble (Dietterich, 2000) κατά την οποία αποφασίσαμε με μεγαλύτερη ακρίβεια, τη δευτεροταγή δομή κάθε πρωτεΐνης, βασιζόμενοι στα αποτελέσματα πρόβλεψης διαφορετικών εκτελέσεων του ίδιου (όσον αφορά τις παραμέτρους) συνελικτικού νευρωνικού δικτύου. Το τελικό σύμβολο δευτεροταγούς δομής για κάθε αμινοξύ ήταν το σύμβολο με το μεγαλύτερο πλήθος εμφανίσεων στις προβλέψεις των επιμέρους διαφορετικών εκτελέσεων. Τα αποτελέσματα τα οποία λάβαμε μετά την εφαρμογή της συγκεκριμένης τεχνικής φαίνονται στον πίνακα 7.3. Στην συνέχεια, προχωρήσαμε στην εφαρμογή της τεχνικής filtering-external rules για επιπρόσθετη

βελτίωση των αποτελεσμάτων που λάβαμε. Πιο συγκεκριμένα, το filtering είναι μια τεχνική κατά την οποία διορθώνουμε κάποια λάθη στις προβλέψεις που λάβαμε, τα οποία είναι βιοχημικά σπάνιο ή και αδύνατο να συμβούν (Salamov and Solovyev, 1995). Για παράδειγμα οι α – έλικες εμφανίζονται σε επαναλαμβανόμενες ακολουθίες τουλάχιστον τριών (3) αμινοξικών καταλοίπων. Ως εκ τούτου, αν παρατηρηθεί εμφάνιση λιγότερων από τρείς (3) α – έλικες σε μια ακολουθία γνωρίζουμε ότι υπάρχει κάποιο λάθος και μπορούμε να προβούμε στη διόρθωση του (Kountouris et al., 2012). Τα αποτελέσματα τα οποία λάβαμε μετά την εφαρμογή της συγκεκριμένης τεχνικής φαίνονται στον πίνακα 7.4.

FOLD	Q3%	QH%	QE%	QC%	SOV	SOVH	SOVE	SOVC
0	78.96	72.89	69.76	84.79	0.75	0.74	0.72	0.74
1	79.19	72.9	67.31	86.57	0.74	0.74	0.71	0.74
2	78.58	72.51	68.34	84.95	0.74	0.73	0.72	0.73
3	78.59	72.83	68.1	85.75	0.74	0.74	0.72	0.74
4	78.44	71.9	69.83	84.79	0.74	0.73	0.73	0.73
5	79.12	74.48	69.99	84.13	0.75	0.76	0.73	0.73
6	79.19	73.63	68.92	85.14	0.75	0.75	0.72	0.74
7	79.06	73.02	69.24	84.96	0.75	0.74	0.73	0.74
8	78.71	72.03	67.73	86.24	0.74	0.73	0.71	0.74
9	79.28	71.29	69.32	86.53	0.74	0.72	0.73	0.74
AVERAGE	**78.91%**	72.748	68.854	85.385	**0.744**	0.738	0.722	0.737

Πίνακας 7.3: Ποσοστά επιτυχίας Q3 και SOV, μετά από τη χρήση της τεχνικής ensemble (6 εκτελέσεις για κάθε υποσύνολο/fold) .

FOLD	Q3%	QH%	QE%	QC%	SOV	SOVH	SOVE	SOVC
0	78.89	70.05	68.12	86.61	0.76	0.73	0.72	0.74
1	78.99	70.72	64.83	88.16	0.76	0.074	0.7	0.73
2	78.33	69.98	66.74	86.5	0.75	0.73	0.71	0.73
3	78.29	70.14	66.02	87.38	0.75	0.74	0.71	0.73
4	78.32	69.26	67.77	86.61	0.75	0.73	0.72	0.73
5	78.92	71.95	68.07	85.82	0.76	0.75	0.72	0.73
6	79.01	70.95	66.85	86.9	0.76	0.74	0.71	0.73
7	78.85	70.64	67.51	86.66	0.76	0.74	0.72	0.73
8	78.39	69.55	65.49	87.77	0.75	0.74	0.7	0.73
9	79	68.23	67.6	88.12	0.76	0.72	0.72	0.73
AVERAGE	**78.69%**	70.147	66.9	87.053	**0.756**	0.6694	0.713	0.731

Πίνακας 7.4: Ποσοστά επιτυχίας Q3 και SOV, μετά από τη χρήση της τεχνικής ensemble (6 εκτελέσεις για κάθε υποσύνολο/fold) και filtering – external rules.

Είναι ενδιαφέρον το ότι με τη χρήστη της τεχνικής filtering – external rules μειώνεται το συνολικό ποσοστό επιτυχίας Q3 αλλά αυξάνεται το SOV, κάτι που αναφέρθηκε ξανά στην έρευνα των Kountouris et al. (2012).

Ως τελευταία προσπάθεια βελτίωσης της ποιότητας των αποτελεσμάτων που λάβαμε, χρησιμοποιήσαμε τον αλγόριθμο μηχανικής μάθησης Support Vector Machine (SVM) (Cortes and Vapnik, 1995; Vapnik, 1998), για να φιλτράρουμε/διορθώσουμε την ποιότητα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών που λάβαμε από το συνελικτικό νευρωνικό δίκτυο, μετά από την εφαρμογή της τεχνικής ensembles. Τα αποτελέσματα τα οποία λάβαμε μετά την εφαρμογή της συγκεκριμένης τεχνικής φαίνονται στον πίνακα 7.5.

FOLD	Q3%	QH%	QE%	QC%	SOV	SOVH	SOVE	SOVC
0	79.69	79.77	70.05	84.75	0.74	0.73	0.71	0.75
1	79.74	78.69	68.06	86.77	0.73	0.73	0.71	0.74
2	78.96	78.64	68.27	84.94	0.72	0.71	0.71	0.73
3	79.55	79.09	67.89	86.12	0.71	0.72	0.7	0.73
4	79.26	78.55	70	84.79	0.73	0.72	0.73	0.72
5	79.7	80.27	70.18	84.31	0.73	0.71	0.72	0.73
6	79.64	79.85	68.87	85.26	0.73	0.73	0.71	0.74
7	83.7	87.68	76.86	83.91	0.76	0.73	0.71	0.77
8	83.91	87.53	76.33	84.62	0.78	0.75	0.74	0.79
9	79.85	79.04	69.27	86.18	0.73	0.71	0.72	0.73
AVERAGE	**80.40%**	80.911	70.578	85.165	**0.736**	0.724	0.716	0.743

Πίνακας 7.5: Ποσοστά επιτυχίας Q3 και SOV, μετά από τη χρήση της τεχνικής ensemble (6 εκτελέσεις για κάθε υποσύνολο/fold) και filtering με χρήση SVMs.

Τα καλύτερα αποτελέσματα τα οποία πήραμε, φαίνεται να είναι αρκετά ικανοποιητικά αφού είναι της τάξης του ***80.40% ποσοστό επιτυχίας Q3 και 0.736 SOV***, χρησιμοποιώντας τα Συνελικτικά Νευρωνικά Δίκτυα σε συνδυασμό με τον αλγόριθμο μηχανικής μάθησης Support Vector Machines για φιλτράρισμα/διόρθωση των αποτελεσμάτων. Τα SVMs φαίνεται να είναι η καλύτερη τεχνική για φιλτράρισμα/διόρθωση των αποτελεσμάτων πρόβλεψης του CNN. Τα αποτελέσματα που λάβαμε μετά την εκπόνηση της συγκεκριμένης έρευνας είναι αρκετά ψηλά αφού κοντεύουν να φτάσουν σχεδόν τα ποσοστά επιτυχίας των καλύτερων μέχρι τώρα τεχνικών που υπάρχουν (Yang et al., 2016).

10 χειρότερες πρωτεΐνες

Protein Name	Q3	SOV	Μήκος
1ednA_1-21	57.1	61	21
1glnA_323-370	60.4	70.5	48
1hupA_88-111	62.5	45.5	24
1bfgA_19-144	63.5	59.9	126
1gkyA_33-82	66	60.5	50
4cpaI_3-38	66.7	52.1	36
1dikA_373-520	68.1	66.3	144
1a45A_1-174	68.8	57.3	173
1azuA_4-127	70.2	60.7	124
1dupA_1-136	75	66.1	136

10 καλύτερες πρωτεΐνες

Protein Name	Q3	SOV	Μήκος
1bdoA_77-156	77.5	70.8	80
1chdA_152-349	77.8	82.8	198
1daaB_120-277	77.8	77.4	158
1aorB_1-211	79.1	83.9	211
1tulA_7-108	79.4	87.6	102
1mlaA_128-197	84.3	91.7	70
1lbuA_1-82	85.4	81.1	82
1ngaA_176-365	85.8	86.5	190
2asrA_38-179	89.4	81.2	142
1edmC_46-84	94.9	95.2	39

Πίνακας 7.6: Ποσοστά επιτυχίας Q3 και SOV των 10 καλύτερων και 10 χειρότερων προβλέψεων πρωτεϊνών με χρήση του συνελικτικού νευρωνικού δικτύου.

Στον πίνακα 7.6 βλέπουμε τις δέκα (10) καλύτερες και τις δέκα (10) χειρότερες προβλέψεις της δευτεροταγούς δομής των πρωτεϊνών που πραγματοποίησε το συνελικτικό νευρωνικό δίκτυο το οποίο έχουμε δημιουργήσει. Φαίνεται ότι από τα συγκεκριμένα αποτελέσματα δεν μπορούμε να εξάγουμε κάποια συσχέτιση για το αν το μήκος των πρωτεϊνών (σε αμινοξέα), επηρεάζει τα ποσοστά επιτυχίας στην πρόβλεψη.

10 χειρότερες πρωτεΐνες

Protein Name	Q3	SOV	Μήκος
1cdlG_796-815	55.3	47.5	20
1bamA_1-213	59.5	54.9	200
1cfrA_1-283	66.4	62.6	283
1fc2C_124-166	67.4	68.3	43
1bbpA_2-178	74.3	73.1	173
1gd1O_0-333	75.1	72.4	334
1cbgA_1-490	75.9	66.1	490
1bmv1_1001-1185	76.2	78.7	185
1hcgB_1-51	78.4	67.9	51
1acxA_1-106	82.2	67.7	107

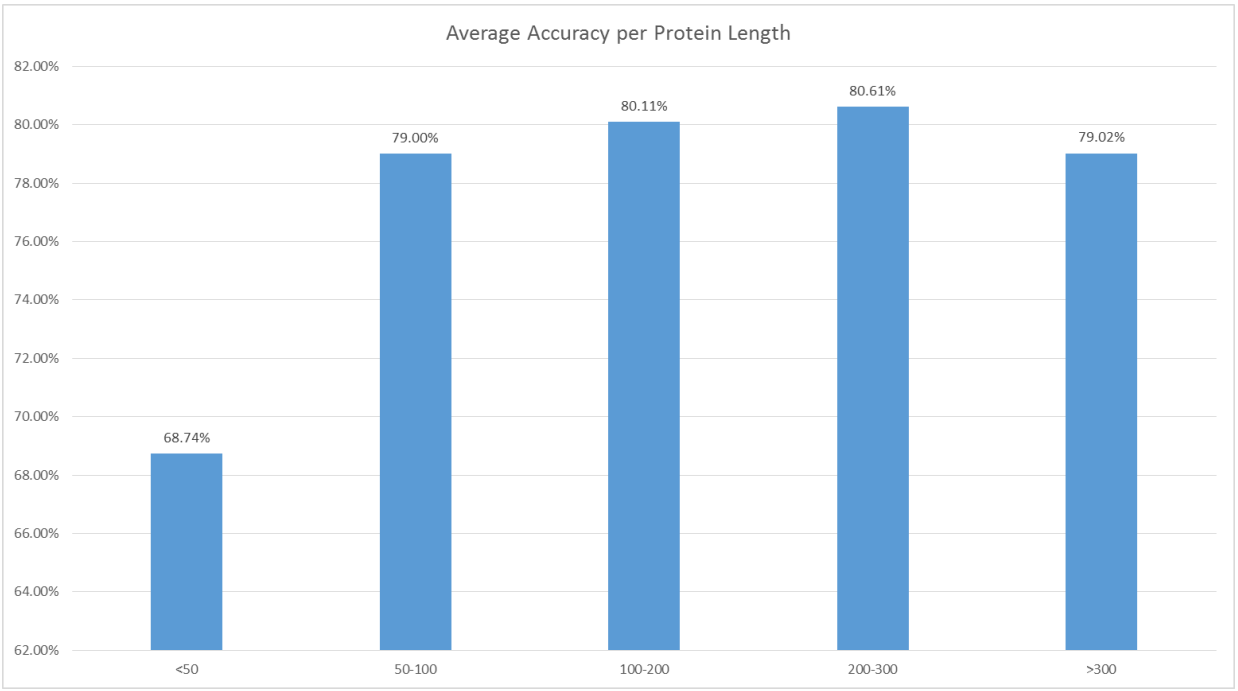
10 καλύτερες πρωτεΐνες

Protein Name	Q3	SOV	Μήκος
1ampA_1-291	83.2	88.3	291
1betA_10-116	83.2	65.1	107
1cemA_33-395	83.5	79.5	363
1bdsA_1-43	83.7	92.6	43
1cqaA_2-133	85.4	93.2	123
1ecaA_1-136	86	89.9	136
1cbhA_1-36	86.1	75.5	36
1cc5A_5-87	86.7	83.7	83
1fkfA_1-107	86.9	80.5	107
1fuaA_1-206	88.8	88	206

Πίνακας 7.7: Ποσοστά επιτυχίας Q3 και SOV των 10 καλύτερων και 10 χειρότερων προβλέψεων πρωτεϊνών με χρήση του συνελικτικού νευρωνικού δικτύου σε συνδυασμό με support vector machine για φιλτράρισμα/διόρθωση των αποτελεσμάτων πρόβλεψης.

Στον πίνακα 7.7 βλέπουμε τις δέκα (10) καλύτερες και τις δέκα (10) χειρότερες προβλέψεις της δευτεροταγούς δομής των πρωτεϊνών που πραγματοποίησε το συνελικτικό νευρωνικό δίκτυο το οποίο έχουμε δημιουργήσει σε συνδυασμό με support vector machine για φιλτράρισμα/διόρθωση των αποτελεσμάτων πρόβλεψης. Φαίνεται ότι τα αποτελέσματα είναι κάπως πιο ομαλά σε σχέση με τα αποτελέσματα του πίνακα 7.6 αλλά και πάλι δεν μπορούμε να αποφανθούμε για το αν το μήκος των πρωτεϊνών παίζει κάποιο ρόλο στα ποσοστά επιτυχίας στην πρόβλεψη.

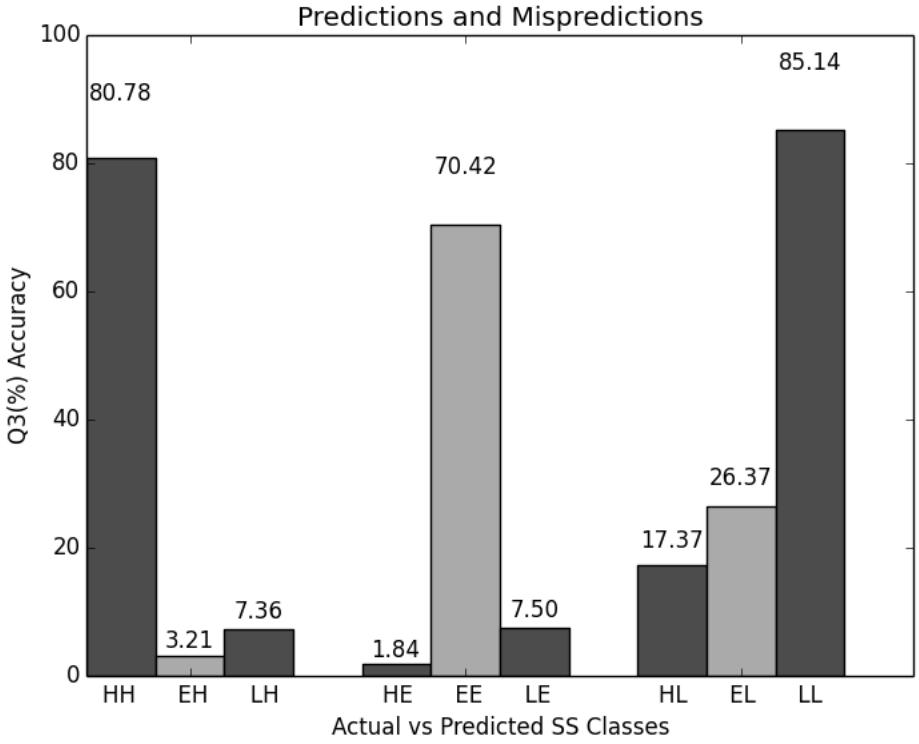
Αξίζει να σημειωθεί ότι οι τα ***fold*** που χρησιμοποιούνται για εξαγωγή των 10 καλύτερων και 10 χειρότερων πρωτεϊνών είναι ***διαφορετικά***. Για να μπορέσουμε να δούμε αν το μήκος των πρωτεϊνών επηρεάζει όντως τα ποσοστά επιτυχίας στην πρόβλεψη της δευτεροταγούς δομής, θα υπολογίσουμε τους μέσους όρους των ποσοστών επιτυχίας Q3, ανάλογα με το μήκος των πρωτεϊνών (σε αμινοξέα).



Γραφική Παράσταση 7.1: Μέσος όρος ποσοστών επιτυχίας στην πρόβλεψη (Q3) ανάλογα με το μήκος των πρωτεϊνών.

Στη γραφική παράσταση 7.1 βλέπουμε το μέσο όρο των ποσοστών επιτυχίας στην πρόβλεψη Q3 ανάλογα με το μήκος των πρωτεϊνών. Τα αποτελέσματα τα οποία φαίνονται,

επιβεβαιώνουν αυτό που είχαμε αρχικά υπόψιν μας, ότι δηλαδή ο συνδυασμός του συνελικτικού νευρωνικού δικτύου με τα support vector machines για filtering, πετυχαίνει τα υψηλότερα ποσοστά επιτυχίας στην πρόβλεψη για πρωτεΐνες με μήκος ακολουθίας μεγαλύτερο ή ίσο των 50 αμινοξέων. Φαίνεται ότι για τις πρωτεΐνες με σχετικά μικρό μήκος ακολουθίας (<50) πετυχαίνει σχετικά χαμηλά ποσοστά επιτυχίας στην πρόβλεψη, της τάξης του 68.74%. Άρα αυτός ο συνδυασμός φαίνεται ότι μειονεκτεί μόνο στις πρωτεΐνες με σχετικά μικρό μήκος ακολουθίας. Αυτό μπορεί να οφείλεται στο μικρό μέγεθος του dataset (CB513) το οποίο χρησιμοποιήσαμε.



Γραφική Παράσταση 7.2: Σωστές και λάθος προβλέψεις για κάθε κατηγορία δευτεροταγούς δομής H, E και C/L, μετά την εφαρμογή της τεχνικής ensembles (winner-take-all) με SVM filtering σε κάθε fold χρησιμοποιώντας το CB513 dataset. Φαίνονται τα ποσοστά επιτυχίας στην πρόβλεψη Q3 για κάθε κατηγορία δευτεροταγούς δομής.

Στη γραφική παράσταση 7.2 φαίνονται τα ποσοστά των σωστών και των λάθος προβλέψεων που πραγματοποίησε το συνελικτικό νευρωνικό δίκτυο μετά από φιλτράρισμα των αποτελεσμάτων πρόβλεψης χρησιμοποιώντας το SVM. Βλέπουμε ότι αυτή η τεχνική

συνδυασμού των δύο μεθόδων μηχανικής μάθησης (CNN και SVM για filtering) είναι μια υψηλής ποιότητας και ανθεκτική μεθοδολογία αφού καταφέρνει να αυξήσει τις σωστές προβλέψεις για κάθε κατηγορία δευτεροταγούς δομής.

7.2 Μελλοντική Εργασία

Για να μπορέσουμε να πετύχουμε υψηλότερα ποσοστά επιτυχίας στο μέλλον, θα μπορούσαμε να χρησιμοποιήσουμε αρχικά κάποιες πιο ειδικές τεχνικές ensembles, οι οποίες θα υπολογίζουν με πιο έξυπνο τρόπο το τελικό σύμβολο δευτεροταγούς δομής για κάθε αμινοξύ μετά από την συγκέντρωση κάποιων εκτελέσεων. Παρόμοια, θα μπορούσαμε να χρησιμοποιήσουμε κάποιες νέες μεθόδους filtering για επίτευξη καλύτερων αποτελεσμάτων.

Προχωρώντας θα μπορούσαμε να δημιουργήσουμε κάποιου είδους υβριδικά νευρωνικά δίκτυα συνδυάζοντας τα συνελικτικά νευρωνικά δίκτυα με κάποιο άλλο είδους νευρωνικό δίκτυο, το οποίο θα λαμβάνει σαν είσοδο τα activations/feature vectors τα οποία μπορούμε να λάβουμε από το τελευταίο κρυφό επίπεδο των συνελικτικών νευρωνικών δικτύων (Medsker, 1994; Li et al., 2016). Πράττοντας το πιο πάνω θα δημιουργήσουμε ένα νέο συνδυασμό νευρωνικών δικτύων κατά τον οποίο το κάθε νευρωνικό δίκτυο θα εντοπίζει διαφορετικά χαρακτηριστικά. Συνεπώς, συνδυάζοντας διαφορετικές αρχιτεκτονικές νευρωνικών δικτύων θα είμαστε σε θέση να συμπεριλάβουμε και να κωδικοποιήσουμε περισσότερη πληροφορία στο νέο υβριδικό νευρωνικό δίκτυο και έτσι θα είναι σε θέση να πραγματοποιήσει πιο ακριβείς προβλέψεις (Wang et al., 2016).

Ακολούθως θα μπορούσαμε να χρησιμοποιήσουμε διαφορετικές μεθόδους βελτιστοποίησης και ελαχιστοποίησης του σφάλματος έτσι ώστε να μπορέσουμε να εκπαιδεύσουμε τα συνελικτικά νευρωνικά δίκτυα με μεγαλύτερη ακρίβεια, προσεγγίζοντας περισσότερο το ολικό ελάχιστο της συνάρτησης (Le et al., 2011).

Επιπρόσθετα, χρησιμοποιώντας μεγαλύτερα datasets όπως το PISCES, θα μπορέσουμε να δώσουμε μια πιο γενική εικόνα στο συνελικτικό νευρωνικό δίκτυο έτσι ώστε να μπορέσει να πραγματοποιήσει καλύτερες προβλέψεις, λόγω του μεγάλου όγκου δεδομένων εκπαίδευσης που θα λάβει. Στη συγκεκριμένη έρευνα, πραγματοποιήσαμε ένα μόνο πείραμα με τη χρήση

του PISCES dataset (αφού έχουμε προετοιμάσει το dataset για εκπαίδευση), λόγω του μεγάλου χρόνου εκπαίδευσης που απαιτείται. Στο μέλλον, θα μπορέσουμε να εκτελέσουμε περισσότερα πειράματα με το συγκεκριμένο dataset για να μπορέσουμε να αξιολογήσουμε στην συνέχεια με μεγαλύτερη ακρίβεια, την απόδοση του συνελικτικού νευρωνικού δικτύου.

Προχωρώντας, θα μπορούσαμε να συνδυάσουμε το συνελικτικό νευρωνικό δίκτυο με support vector machines για φιλτράρισμα των αποτελεσμάτων πρόβλεψης, με απλά δίκτυα εμπρόσθιου περάσματος με χρήση του Hessian free optimization στο τέλος της αρχιτεκτονικής μας, αφού φαίνεται να πετυχαίνουν υψηλά ποσοστά επιτυχίας στην πρόβλεψη ενώ ταυτόχρονα έχουν γρήγορους χρόνους εκπαίδευσης.

Τέλος, θα πρέπει να προσπαθήσουμε να εντοπίσουμε τον πραγματικό λόγο για τον οποίο ο συνδυασμός του CNN με το BRNN πετυχαίνει τόσο χαμηλά ποσοστά επιτυχίας στην πρόβλεψη (42%). Πρέπει να προσπαθήσουμε να αναπροσαρμόσουμε τον τρόπο με τον οποίο εξάγουμε τα feature vectors από το τελευταίο κρυφό επίπεδο του συνελικτικού νευρωνικού δικτύου έτσι ώστε να έχουν νόημα για την μετέπειτα τροφοδότηση τους στο bidirectional recurrent neural network για την εκπαίδευση του νευρωνικού δικτύου. Ακόμα, πρέπει να προχωρήσουμε με το να εντοπίσουμε τις ιδανικές παραμέτρους για αυτό το συνδυασμό νευρωνικών δικτύων.

Αναφορές

Αγαθοκλέους, Μ. (2009). Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών με την χρήση Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης, Προπτυχιακή διπλωματική, Τμήμα Πληροφορικής Πανεπιστήμιο Κύπρου, Λευκωσία.

Παυλίδης, Π. (2016). Πρόβλεψη Δευτεροταγούς Δομής Των Πρωτεϊνών Με Τη Χρήση Των Convolutional Neural Networks Για Οπτική Αναγνώριση Αντικειμένων, Πανεπιστήμιο Κύπρου, Τμήμα Πληροφορικής, Λευκωσία, Ατομική Διπλωματική Εργασία.

Χριστοδούλου, Γ. (2010). Διερεύνηση Μεθόδων Εκπαίδευσης Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης για Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών, Προπτυχιακή διπλωματική, Τμήμα Πληροφορικής Πανεπιστήμιο Κύπρου, Λευκωσία.

Armano, G., Orro, A. & Vargiu, E. (2006). MASSP3: A System for Predicting Protein Secondary Structure, EURASIP Journal on Advances in Signal Processing, Article ID 17195, 1–9.

Baldi, P., Brunak, S., Frasconi, P., Pollastri, G., & Soda, G. (2000). Bidirectional dynamics for protein secondary structure prediction. 80-104, Springer, Berlin, Heidelberg.

Baldi, P., Brunak, S., Frasconi, P., Soda, G. & Pollastri, G. (1999). Exploiting the past and the future in protein secondary structure prediction. Bioinformatics, 15(11), 937-946.

Berg, J. M., Tymoczko, J. L. & Stryer, L. (2002). Biochemistry. 5th edition, New York, NY: WH. Freeman.

Blazewicz, J., Hammer, P. & Lukasiak, P. (2005). Predicting secondary structures of proteins, IEEE engineering in medicine and biology magazine, 24(3), 88-94.

Boureau, Y. L., Ponce, J., & LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In Proceedings of the 27th international conference on machine learning, 111-118.

Brownlee, J. (2016). Crash Course on Multi-Layer Perceptron Neural Networks, Retrieved September 1, 2017, from https://machinelearningmastery.com/neural-networks-crash-course.

Burney, S. M. A., Jilani, T. A., & Ardil, C. (2004). A Comparison of First and Second Order Training Algorithms for Artificial Neural Networks. In International Conference on Computational Intelligence, 12-18.

Chablani, M. (2017). Deep learning concepts, Retrieved September 2, 2017, from https://towardsdatascience.com/deep-learning-concepts-part-1-ea0b14b234c8.

Chen, J. & Chaudhari N.S (2007). Cascaded Bidirectional Recurrent Neural Networks for Protein Secondary Structure Prediction, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 4(4), 572-582.

CNX OpenStax (2016). Protein structures, Retrieved September 3, 2017, from https://cnx.org/contents/5CvTdmJL@4.4:rFziotaH@4/Introduction.

Cortes, C. & Vapnik, V. (1995). Support-vector network. Machine Learning, 20, 1–25.

Cuff, JA. & Barton, GJ. (1999). Evaluation and improvement of multiple sequence methods for protein secondary structure prediction, Proteins: Structure, Function, and Bioinformatics, 34(4), 508–519.

Deeplearning4j. (n.d.). Retrieved September 2, 2017, from https://deeplearning4j.org/updater.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In International workshop on multiple classifier systems, 1-15, Springer, Berlin, Heidelberg.

Dill, K. A., Ozkan, S. B., Shell, M. S., & Weikl, T. R. (2008). The protein folding problem. Annu. Rev. Biophys., 37, 289-316.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159. Retrieved September 10, 2017, from http://jmlr.org/papers/v12/duchi11a.html.

Elman L.J. (1990). Finding Structure in Time, Cognitive Science, 14, 179-211.

Fang, C., Shang, Y., & Xu, D. (2018). MUFOLD-SS: New deep inception-inside-inception networks for protein secondary structure prediction. Proteins: Structure, Function, and Bioinformatics.

Frishman, D. & Argos, P. (1995). Knowledge-based protein secondary structure assignment, Proteins: Structure, Function, and Bioinformatics, 23(4), 566–579.

Gill, S. N. (2017). Overview and Applications of Artificial Neural Networks, Retrieved April 10, 2017, from https://www.xenonstack.com/blog/data-science/overview-of-artificial-neural-networks-and-its-applications.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, 249-256.

Grzesiak, W., & Zaborski, D. (2012). Examples of the use of data mining methods in animal breeding. In Data mining applications in engineering and medicine. InTech, DOI: 10.5772/50893. Available from: https://www.intechopen.com/books/data-mining-applications-in-engineering-and-medicine/examples-of-the-use-of-data-mining-methods-in-animal-breeding.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, 1026-1034.

Heffernan, R., Yang, Y., Paliwal, K., & Zhou, Y. (2017). Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. Bioinformatics, 33(18), 2842-2849.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Lecture 6d - a separate, adaptive learning rate for each connection. Slides of Lecture Neural Networks for Machine Learning.

Johnson, G. B., & Raven, P. H. (2001). Biology: Principles & Explorations. New York, NY: Holt, Rinehart and Winston.

Jones, A. (2015). An Explanation of Xavier Initialization, Retrieved September 10, 2017, from http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization.

Jones, J. P., & Palmer, L. A. (1987). An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. Journal of neurophysiology, 58(6), 1233-1258.

King, RD. and Sternberg, MJ. (1996). Identification and application of the concepts important for accurate and reliable protein secondary structure prediction, Protein Science, 5(11), 2298–2310.

Karpathy, A. (2016). Convolutional Neural Networks for Visual Recognition, Retrieved September 10, 2017, from http://cs231n.github.io/neural-networks-1/.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 1725-1732.

Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. Ijcai, 14(2), 1137-1145.

Kountouris, P., Agathocleous, M., Promponas, V. J., Christodoulou, G., Hadjicostas, S., Vassiliades, V., & Christodoulou, C. (2012). A comparative study on filtering protein secondary structure prediction. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 9(3), 731-739.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 1097-1105.

Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., & Ng, A. Y. (2011). On optimization methods for deep learning. In Proceedings of the 28th International Conference on International Conference on Machine Learning, 265-272, Omnipress.

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object recognition with gradient-based learning. In Shape, contour and grouping in computer vision, 319-345, Springer, Berlin, Heidelberg.

Leung, H., & Haykin, S. (1991). The complex backpropagation algorithm. IEEE Transactions on Signal Processing, 39(9), 2101-2104.

Li, Z., & Yu, Y. (2016). Protein secondary structure prediction using cascaded convolutional and recurrent neural networks. arXiv preprint arXiv:1604.07176.

McCarthy, J. (1989). Artificial intelligence, logic and formalizing common sense. In Philosophical logic and artificial intelligence, 161-190, Springer, Dordrecht.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

Medsker, L. (1994). Design and development of hybrid neural network and expert systems. In Neural Networks, 1994, IEEE World Congress on Computational Intelligence, 1994 IEEE International Conference, 3, 1470-1474, IEEE.

Medsker, L. R., & Jain, L. C. (2001). Recurrent neural networks: Design and Applications. CRC Press, New York US.

Meyer, D., & Wien, F. T. (2001). Support vector machines. R News, 1(3), 23-26.

Miranda, L. (2017). Understanding softmax and the negative log-likelihood. Retrieved January 18, 2018, from https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood.

Pascanu, R., Mikolov, T., & Bengio, Y. (2012). Understanding the exploding gradient problem. CoRR, abs/1211.5063.

Pauling, L., Corey, R. B., & Branson, H. R. (1951). The structure of proteins: two hydrogen-bonded helical configurations of the polypeptide chain. Proceedings of the National Academy of Sciences, 37(4), 205-211.

Promponas B. (2004-2005). Bioinformatics Notes, Athens.

Qian, N. and Sejnowski, TJ. (1988). Predicting the secondary structure of globular proteins using neural network models, Journal of Molecular Biology, 202(4), 865-884.

Raschka, S. (2014). Predictive modeling, supervised machine learning, and pattern classification.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386-408.

Rost, B. and Sander, C. (1993). Improved prediction of protein secondary structure by use of sequence profiles and neural networks, PNAS, 90, 7558-7562.

Salamov, A. A., & Solovyev, V. V. (1995). Prediction of protein secondary structure by combining nearest-neighbor algorithms and multiple sequence alignments.

Salamov, A. and Soloveyev, V. (2000). Ab initio gene finding in Drosophila genomic DNA, Genome Research, 10(4), 516-522.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of research and development, 3(3), 210-229.

Seif, G. (2018). Deep Learning for Image Recognition: why it's challenging, where we've been, and what's next. Retrieved February 1, 2018, from https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.

Templeton, G. (2015). Artificial neural networks are changing the world. What are they, Retrieved September 1, 2017, from https://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they.

Turing, A. M. (2009). Computing machinery and intelligence. In Parsing the Turing Test, 23-65, Springer, Dordrecht.

Vapnik, V. (1998). Statistical learning theory. Wiley, New York.

Wang, G., & Dunbrack Jr, R. L. (2003). PISCES: a protein sequence culling server. Bioinformatics, 19(12), 1589-1591.

Wang, S., Peng, J., Ma, J. and Xu, J. (2016). Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields, Scientific Reports, 6, 18962.

Won, K. J., Hamelryck, T., Prügel-Bennett, A., & Krogh, A. (2007). An evolutionary method for learning HMM structure: prediction of protein secondary structure. BMC bioinformatics, 8(1), 357.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853.

Yang, Y., Gao, J., Wang, J., Heffernan, R., Hanson, J., Paliwal, K., & Zhou, Y. (2016). Sixty-five years of the long march in protein secondary structure prediction: the final stretch. Briefings in bioinformatics, 129.

Yüksektepe, F.U, Yılmaz, O. and Türkay, M. (2008). Prediction of secondary structures of proteins using a two-stage method, Computers & Chemical Engineering, 32(1–2), 78-88.

Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved February 1, 2018, from http://arxiv.org/abs/1212.5701.

Zemla, A., Venclovas, Č., Fidelis, K., & Rost, B. (1999). A modified definition of Sov, a segment-based measure for protein secondary structure prediction assessment. Proteins: Structure, Function, and Bioinformatics, 34(2), 220-223.

Γενική Βιβλιογραφία

Baker, D. and Sali, A. (2001). Protein structure prediction and structural genomics, Science 294, 93–96.

Bishop C. M. (2005). Neural Networks for Pattern Recognition, Oxford UK.

Bengio, Y. (2009). Learning Deep Architectures for AI, Foundations and Trends® in Machine Learning, 2(1), doi: 10.1561/2200000006

Dill, K. A. and MacCallum, J. L. (2012). The protein-folding problem, 50 years on, Science 338, 1042–1046.

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors, arXiv preprint, arXiv: 12070580.

Kabsch, W. and Sander, C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features, Biopolymers, 22(12), 2577-2637.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1989). Gradient-based learning applied to document recognition, Proceedings of the IEEE, 86(11), 2278-2324.

Movellan, R. (2008). Tutorial on Gabor Filters. Retrieved September 1, 2017, from http://mplab.ucsd.edu/tutorials/gabor.pdf.

Petsko, G. A. and Ringe, D. (2004). Protein structure and function, New Science Press: London, UK.

Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9), 2352-2449.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J (1986). Learning representations by back-propagating errors, Nature, 323, 533-536.

Schaffhausen, J. (2012). Advances in structure-based drug design, Trends Pharmacol. Sci. 33, 223.

Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview, Neural Networks, 61, 85–117.

Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U. & Poggio, T. (2007). A quantitative theory of immediate visual recognition, Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function, 165, 33–56.

Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia, Pearson Education Limited.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, 15, 1929-1958

Whisstock, J. C. and Lesk, A. M. (2003). Prediction of protein function from protein sequence and structure, Q. Rev. Biophys. 36, 307–340.

Whittle, P. J. and Blundell, T. L. (1994). Protein structure-based drug design, Annu. Rev. Biophys. Biomol. Struct. 23, 349–375.

Παράρτημα Α

Γενικά Σχόλια για το παράρτημα Α:

Στο συγκεκριμένο παράρτημα βλέπουμε δύο αρχεία κώδικα σε γλώσσα προγραμματισμού Java, με ονόματα Protein.java και Proteins_files_creator.java.

Το πρώτο αρχείο κώδικα (Protein.java) χρησιμοποιείται για αναπαράσταση ενός αντικειμένου πρωτεΐνης το οποίο διαθέτει το όνομα (id), την πρωτοταγή, τη δευτεροταγή (σε χαρακτήρες – String protein_second_order και σε δυαδική μορφή int []second_order_output_repr) και την MSA κωδικοποίηση της πρωτεΐνης.

Το δεύτερο αρχείο κώδικα (Proteins_files_creator.java) χρησιμοποιείται για την κατασκευή των δεδομένων εισόδου για εκπαίδευση και επαλήθευση του CNN. Στο συγκεκριμένο αρχείο κώδικα, βλέπουμε έξι συναρτήσεις οι οποίες είναι οι εξής:

1. create_csv_files_train_allInOne();
2. create_csv_files_test_allInOne();
3. create_csv_train_files_separate();
4. create_csv_test_files_separate();
5. create_csv_files_train_Xneighboring_technique(int windowNum);
6. create_csv_files_test_Xneighboring_technique(int windowNum);

Οι πρώτες δύο συναρτήσεις (1 και 2) χρησιμοποιούνται για κατασκευή των αρχείων εισόδου που περιγράφονται στην ενότητα 5.5. Οι συναρτήσεις 3 και 4 χρησιμοποιούνται για κατασκευή των MSA αρχείων για κάθε πρωτεΐνη ξεχωριστά. Τέλος, οι συναρτήσεις 5 και 6 χρησιμοποιούνται για κατασκευή των δεδομένων εκπαίδευσης και επαλήθευσης που περιγράφονται στην ενότητα 5.8.

Protein.java

```
/**
 * Created on 27/07/2017.
 *
 * Author: Antreas Dionysiou
 *
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
public class Protein {

    String protein_id; //protein's name
    String protein_first_order; //primary structure
    String protein_second_order; //secondary structure in chars
representation
    int [] second_order_output_repr; //secondary structure in
binary representation
    int [][] msa_repr;  //protein's msa representation

    public Protein(String id,String first_order,String
second_order,String msa_file_name){
        this.protein_id=id; //protein id i.e. name.
        this.protein_first_order=first_order; //primary
structure.
        this.protein_second_order=second_order; //secondary
structure.
        this.msa_repr= read_msa_file(msa_file_name);
        //construct secondary structure output representation
with 0 and 1.
        this.second_order_output_repr= new
int[3*this.protein_second_order.length()];

create_sec_out_repr(this.protein_second_order,this.second_order_
output_repr);

    }

    /*
    * This function is used to create second order output
representation in 0 and 1 in order
    * to be unsderstood by the neural network.
    * B=000, C=001, E=010, G=011, H=100, I=101, S=110, T=111
```

```
    * */
    private static void create_sec_out_repr(String
second_order,int[] second_order_out_rep){
        int i=0;
        int j=0;
        while (i<second_order.length()){
            if (second_order.charAt(i)=='G'){
                second_order_out_rep[j]=1;
                second_order_out_rep[j+1]=0;
                second_order_out_rep[j+2]=0;
            }
            else if (second_order.charAt(i)=='H'){
                second_order_out_rep[j]=0;
                second_order_out_rep[j+1]=1;
                second_order_out_rep[j+2]=1;
            }
            else if (second_order.charAt(i)=='I'){
                second_order_out_rep[j]=1;
                second_order_out_rep[j+1]=0;
                second_order_out_rep[j+2]=1;
            }
            else if (second_order.charAt(i)=='T'){
                second_order_out_rep[j]=1;
                second_order_out_rep[j+1]=1;
                second_order_out_rep[j+2]=1;
            }
            else if (second_order.charAt(i)=='E'){
                second_order_out_rep[j]=0;
                second_order_out_rep[j+1]=1;
                second_order_out_rep[j+2]=0;
            }
            else if (second_order.charAt(i)=='B'){
                second_order_out_rep[j]=0;
                second_order_out_rep[j+1]=0;
                second_order_out_rep[j+2]=0;
            }
            else if (second_order.charAt(i)=='S'){
                second_order_out_rep[j]=1;
                second_order_out_rep[j+1]=1;
                second_order_out_rep[j+2]=0;
            }
            else if (second_order.charAt(i)=='C'){
                second_order_out_rep[j]=0;
                second_order_out_rep[j+1]=0;
                second_order_out_rep[j+2]=1;
            }
```

```
            else if(second_order.charAt(i)=='!'){
                //if not enough information i.e. ! then
translate the same as left amino. THINK IT AGAIN!!!
                second_order_out_rep[j]=second_order_out_rep[j-
3];

second_order_out_rep[j+1]=second_order_out_rep[j-2];

second_order_out_rep[j+2]=second_order_out_rep[j-1];
            }
            else {
                System.out.println("ERROR!!! WRONG SECONDARY
STRUCTURE!!! THIS AMINO "+second_order.charAt(i)+" DOESNT
EXISTS!!!");
                System.exit(1);
            }
            i++;
            j+=3;
        }
    }


    private static int [][] read_msa_file(String filename){
        int temp [][]= new int[1][1]; //init just for error
avoidance.
        try {
            //specify msa path to next line
            //Scanner s = new Scanner(new
File("./CB513_10_FOLDS/msaFiles/"+filename+".hssp"));
            //OR PISCES DATA
            Scanner s = new Scanner(new
File("./PISCES_DATASET_5_FOLDS/PSSMS/"+filename+".fasta.hssp"));

            ArrayList<String> lines = new ArrayList<String>();
            int column=0;
            String line="";
            while (s.hasNextInt()) {
                if (column==20){
                    lines.add(line);
                    line="";
                    column=0;
                }
                line+=(s.nextInt()+" ");
                column++;

            }
```

```
            //append last line
            lines.add(line);
            //finally close file
            s.close();

            temp = new int[lines.size()][20];
            //test print below. File lines num.
            //System.out.println(lines.size());
            for (int i = 0; i < lines.size(); i++) {
                String[] nums = (lines.get(i)).split(" ");
                for (int j = 0; j < 20; j++) {
                    temp[i][j] = Integer.parseInt(nums[j]);
                    //test print below
                    //System.out.print(temp[i][j]+ " ");
                }
                //test print below
                //System.out.println();
            }
        }
        catch (FileNotFoundException e)
        {
            System.out.println("ERROR ON READING "+filename+ "
msa file!!!\n");
            return null;
            //System.exit(1);
        }
        catch (java.io.IOException e){
            System.out.println("OTHER ERROR\n");
            return null;
        }
        return temp;
    }

    //public static void main(String args[]){
      //  Protein p = new
Protein("a","a","./Data/msaFiles/1a45a_1-174.hssp");
    //}

}
```

Proteins_files_creator.java

```
/**
 * <h1>Training/Testing Proteins arrays class</h1>
 * This class is used to prepare the training and testing
datasets
 * that will be used for training a specific neural netwwork. It
gets
 * two filenames as parameters(training/testing) and then one
Protein
 * object is created for each protein.
 *
 * After loading the data and creating instances for all
proteins,
 * two of the six main methods will be called to create the
final training
 * and testing files.
 *
 * Main Methods:
 * create_csv_files_train_allInOne();
 * create_csv_files_test_allInOne();
 * create_csv_train_files_separate();
 * create_csv_test_files_separate();
 * create_csv_files_train_Xneighboring_technique(int windowNum);
 * create_csv_files_test_Xneighboring_technique(int windowNum);
 *
 * For PISCES dataset preparation uncomment and comment
approrpiate lines
 * with OR in the middle.
 * <p>
 * <b>Note:</b> The files given should have the same format as
the other
 * example files of this project, which is id, primary structure
and secondary
 * structure.
 *
 * @author Antreas Dionysiou
 * @version 1.0
 * @since 26/7/2017
 */

import java.io.File;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;
```

```
public class Proteins_files_creator {
    public static int counter=0;
    ArrayList<Protein> train_proteins = new
ArrayList<Protein>();
    ArrayList<Protein> test_proteins = new ArrayList<Protein>();

    public Proteins_files_creator(String train_filename, String
test_filename) {
        String protein_id;
        String first_order;
        String second_order;
        Protein temp_protein;
        //Create training proteins objects array
        try {
            //SPECIFY TRAINING SET PATH TO FOLDER.
            Scanner s = new Scanner(new File("./CB513_10_FOLDS/"
+ train_filename));
            //OR
            //Scanner s = new Scanner(new
File("./PISCES_DATASET_5_FOLDS/" + train_filename));

            while (s.hasNext()) {
                protein_id = s.next();
                first_order = s.next();
                second_order = s.next();
                temp_protein = new Protein(protein_id,
first_order, second_order, protein_id);
                if (temp_protein.msa_repr==null){
                    //proteins msa not found so dont add it
                    counter++;
                    continue;
                }
                //else protein found add it.
                train_proteins.add(temp_protein);
            }
            s.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE " + train_filename + " NOT
FOUND!!!\n");
            System.exit(1);
        } catch (Exception e) {
            System.out.println("OTHER ERROR ON FILE " +
train_filename + " READING OR OPENING!!!\n");
            System.exit(1);
        }
```

```
        //Create testing proteins objects array
        try {
            //SPECIFY TRAINING SET PATH TO FOLDER.
            Scanner s = new Scanner(new File("./CB513_10_FOLDS/"
+ test_filename));
            //OR
            //Scanner s = new Scanner(new
File("./PISCES_DATASET_5_FOLDS/" + test_filename));

            while (s.hasNext()) {
                protein_id = s.next();
                first_order = s.next();
                second_order = s.next();
                temp_protein = new Protein(protein_id,
first_order, second_order, protein_id);
                if (temp_protein.msa_repr==null){
                    //proteins msa not found so dont add it
                    counter++;
                    continue;
                }
                //else protein found add it.
                test_proteins.add(temp_protein);
            }
            s.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE " + test_filename + " NOT
FOUND!!!\n");
            System.exit(1);
        } catch (Exception e) {
            System.out.println("OTHER ERROR ON FILE " +
test_filename + " READING OR OPENING!!!\n");
            System.exit(1);
        }
        /*Proteins reader has now finished.
        The two arraylists(train_proteins,test_proteins) contain
Protein objects. One for each protein that is in the
        given file(train and testing file). Every object
consists of id,first order,second order and msa representation
        of each protein.
        */

    }
```

```
 private void create_csv_files_test_Xneighboring_technique(int
windowNum) {
        File file = new
File("./CSV_Results/protein_csv_test_all_"+windowNum+"neighbors_
technique_fold9.txt");
        //OR FOR PISCES
        //File file = new
File("./PISCES_CSV_RESULTS/protein_csv_test_all_"+windowNum+"nei
ghbors_technique.txt");


        PrintWriter pW;
        int sec_order_count = 0;
        String s = ""; //for converting binary to decimal.
        try {
            pW = new PrintWriter(file);
            for (int i = 0; i < this.test_proteins.size(); i++)
{
                sec_order_count = 0;
                //prepare first x aminos with 0s at the
beginning.
                int left_window=(windowNum/2); //calc left
window
                int zeros=0; //declaration
                for (int amino=0;amino<left_window;amino++){
                    zeros=left_window*20; //calc initial zeros
                    zeros-=(20*amino);
                    //place zeros
                    for (int zero=0;zero<zeros;zero++){
                        pW.print("0,");
                    }
                    //place 0-amino aminos
                    for (int j=0;j<amino;j++){
                        for (int
f=0;f<test_proteins.get(i).msa_repr[j].length;f++){

pW.print(test_proteins.get(i).msa_repr[j][f]+",");
                        }
                    }
                    //place current and next aminos
                    for (int j=0;j<=left_window;j++){
                        for (int
f=0;f<test_proteins.get(i).msa_repr[amino+j].length;f++){
```

```
pW.print(test_proteins.get(i).msa_repr[amino+j][f]+",");
                        }
                    }
                    //place amino ss label at the end of the
line.
                    s = "" +
test_proteins.get(i).second_order_output_repr[sec_order_count] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
1] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }
                //place all other aminos that there is no
padding.
                int j=1;
                for (j = left_window; j <
test_proteins.get(i).msa_repr.length-left_window; j++) {
                    for (int curr = -left_window; curr <
(left_window+1); curr++) {
                        for (int z = 0; z <
test_proteins.get(i).msa_repr[j + curr].length; z++) {

pW.print(test_proteins.get(i).msa_repr[j + curr][z]);
                            pW.print(",");
                        }
                    }
                    //place amino ss label at the end of the
line.
                    s = "" +
test_proteins.get(i).second_order_output_repr[sec_order_count] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
1] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }
                //output last x amino records with padding.
                zeros=0; //declaration
                for (int amino=1;amino<=left_window;amino++){
                    zeros=amino*20; //calc initial zeros
                    //place current and previous aminos
```

```
                    for (int z=left_window;z>=0;z--){
                        for (int
f=0;f<test_proteins.get(i).msa_repr[j-z].length;f++){

pW.print(test_proteins.get(i).msa_repr[j-z][f]+",");
                        }
                    }
                    //place curr-last aminos
                    for (int
z=j+1;z<test_proteins.get(i).msa_repr.length;z++){
                        for (int
f=0;f<test_proteins.get(i).msa_repr[z].length;f++){

pW.print(test_proteins.get(i).msa_repr[z][f]+",");
                        }
                    }
                    j++; //next amino
                    //next place zeros
                    for (int zero=0;zero<zeros;zero++){
                        pW.print("0,");
                    }
                    //place amino ss label at the end of the
line.
                    s = "" +
test_proteins.get(i).second_order_output_repr[sec_order_count] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
1] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }


            }
            pW.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE NOT FOUND!!!");
            System.exit(1);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("OTHER EXCEPTION FOUND!!!! ");
            System.exit(1);
        }
    }
```

```
    private void
create_csv_files_train_Xneighboring_technique(int windowNum) {
        File file = new
File("./CSV_Results/protein_csv_train_all_"+windowNum+"neighbors
_technique_fold9.txt");
        //OR FOR PISCES DATASET
        //File file = new
File("./PISCES_CSV_RESULTS/protein_csv_train_all_"+windowNum+"ne
ighbors_technique.txt");


        PrintWriter pW;
        int sec_order_count = 0;
        String s = ""; //for converting binary to decimal.
        try {
            pW = new PrintWriter(file);
            for (int i = 0; i < this.train_proteins.size(); i++)
{
                sec_order_count = 0;
                //prepare first x aminos with 0s at the
beginning.
                int left_window=(windowNum/2); //calc left
window
                int zeros=0; //declaration
                for (int amino=0;amino<left_window;amino++){
                    zeros=left_window*20; //calc initial zeros
                    zeros-=(20*amino);
                    //place zeros
                    for (int zero=0;zero<zeros;zero++){
                        pW.print("0,");
                    }
                    //place 0-amino aminos
                    for (int j=0;j<amino;j++){
                        for (int
f=0;f<train_proteins.get(i).msa_repr[j].length;f++){

pW.print(train_proteins.get(i).msa_repr[j][f]+",");
                        }
                    }
                    //place current and next aminos
                    for (int j=0;j<=left_window;j++){
                        for (int
f=0;f<train_proteins.get(i).msa_repr[amino+j].length;f++){

pW.print(train_proteins.get(i).msa_repr[amino+j][f]+",");
```

```
                    }
                }
                //place amino ss label at the end of the
line.
                s = "" +
train_proteins.get(i).second_order_output_repr[sec_order_count]
+ train_proteins.get(i).second_order_output_repr[sec_order_count
+ 1] +
train_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                sec_order_count += 3;
            }
            //place all other aminos that there is no
padding.
            int j=1;
            for (j = left_window; j <
train_proteins.get(i).msa_repr.length-left_window; j++) {
                for (int curr = -left_window; curr <
(left_window+1); curr++) {
                    for (int z = 0; z <
train_proteins.get(i).msa_repr[j + curr].length; z++) {

pW.print(train_proteins.get(i).msa_repr[j + curr][z]);
                        pW.print(",");
                    }
                }
                //place amino ss label at the end of the
line.
                s = "" +
train_proteins.get(i).second_order_output_repr[sec_order_count]
+ train_proteins.get(i).second_order_output_repr[sec_order_count
+ 1] +
train_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                sec_order_count += 3;
            }
            //output last x amino records with padding.
            zeros=0; //declaration
            for (int amino=1;amino<=left_window;amino++){
                zeros=amino*20; //calc initial zeros
                //place current and previous aminos
                for (int z=left_window;z>=0;z--){
```

```
                        for (int
f=0;f<train_proteins.get(i).msa_repr[j-z].length;f++){

pW.print(train_proteins.get(i).msa_repr[j-z][f]+",");
                        }
                    }
                    //place curr-last aminos
                    for (int
z=j+1;z<train_proteins.get(i).msa_repr.length;z++){
                        for (int
f=0;f<train_proteins.get(i).msa_repr[z].length;f++){

pW.print(train_proteins.get(i).msa_repr[z][f]+",");
                        }
                    }
                    j++; //next amino
                    //next place zeros
                    for (int zero=0;zero<zeros;zero++){
                        pW.print("0,");
                    }
                    //place amino ss label at the end of the
line.
                    s = "" +
train_proteins.get(i).second_order_output_repr[sec_order_count]
+ train_proteins.get(i).second_order_output_repr[sec_order_count
+ 1] +
train_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }
            }
            pW.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE NOT FOUND!!!");
            System.exit(1);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("OTHER EXCEPTION FOUND!!!! ");
            System.exit(1);
        }
    }


    private void create_csv_files_train_allInOne() {
```

```
        File file = new
File("./CSV_Results/protein_csv_train_all.txt");
        PrintWriter pW;
        int sec_order_count = 0;
        String s = ""; //for converting binary to decimal.
        try {
            pW = new PrintWriter(file);
            for (int i = 0; i < this.train_proteins.size(); i++)
{
                sec_order_count = 0;
                for (int j = 0; j <
train_proteins.get(i).msa_repr.length; j++) {
                    for (int z = 0; z <
train_proteins.get(i).msa_repr[j].length; z++) {

pW.print(train_proteins.get(i).msa_repr[j][z]);
                        pW.print(",");
                    }
                    /*

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
r_count]);

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
r_count+1]);

pW.println(train_proteins.get(i).second_order_output_repr[sec_or
der_count+2]);
                    */
                    //place amino ss label at the end of the
line.
                    s = "" +
train_proteins.get(i).second_order_output_repr[sec_order_count]
+ train_proteins.get(i).second_order_output_repr[sec_order_count
+ 1] +
train_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }

            }
            pW.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE NOT FOUND!!!");
```

```
            System.exit(1);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("OTHER EXCEPTION FOUND!!!! ");
            System.exit(1);
        }
    }


    private void create_csv_files_test_allInOne() {
        File file = new
File("./CSV_Results/protein_csv_test_all.txt");
        PrintWriter pW;
        int sec_order_count = 0;
        String s = ""; //for converting binary to decimal.
        try {
            pW = new PrintWriter(file);
            for (int i = 0; i < this.test_proteins.size(); i++)
{
                sec_order_count = 0;
                for (int j = 0; j <
test_proteins.get(i).msa_repr.length; j++) {
                    for (int z = 0; z <
test_proteins.get(i).msa_repr[j].length; z++) {

pW.print(test_proteins.get(i).msa_repr[j][z]);
                        pW.print(",");
                    }
                    /*

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
r_count]);

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
r_count+1]);

pW.println(train_proteins.get(i).second_order_output_repr[sec_or
der_count+2]);
                    */
                    //place amino ss label at the end of the
line.
                    s = "" +
test_proteins.get(i).second_order_output_repr[sec_order_count] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
1] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
```

```
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }

            }
            pW.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE NOT FOUND!!!");
            System.exit(1);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("OTHER EXCEPTION FOUND!!!! ");
            System.exit(1);
        }
    }

    private void create_csv_train_files_separate() {
        PrintWriter pW = null;
        int sec_order_count = 0;
        String s = ""; //for converting binary to decimal.
        try {
            for (int i = 0; i < this.train_proteins.size(); i++)
{
                sec_order_count = 0;
                //File file = new
File("./CSV_Results/"+train_proteins.get(i).protein_id
+"_csv.txt");
                File file = new File("./CSV_Results/train/file"
+ i + ".csv");
                pW = new PrintWriter(file);
                for (int j = 0; j <
train_proteins.get(i).msa_repr.length; j++) {
                    for (int z = 0; z <
train_proteins.get(i).msa_repr[j].length; z++) {

pW.print(train_proteins.get(i).msa_repr[j][z]);
                        pW.print(",");
                    }
                    /*

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
r_count]);

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
```

```
r_count+1]);

pW.println(train_proteins.get(i).second_order_output_repr[sec_or
der_count+2]);
                    */
                    s = "" +
train_proteins.get(i).second_order_output_repr[sec_order_count]
+ train_proteins.get(i).second_order_output_repr[sec_order_count
+ 1] +
train_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }
                pW.close();

            }
            pW.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE NOT FOUND!!!");
            System.exit(1);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("OTHER EXCEPTION FOUND!!!! ");
            System.exit(1);
        }
    }

    private void create_csv_test_files_separate() {
        PrintWriter pW = null;
        int sec_order_count = 0;
        String s = ""; //for converting binary to decimal.
        try {
            for (int i = 0; i < this.test_proteins.size(); i++)
{
                sec_order_count = 0;
                //File file = new
File("./CSV_Results/"+train_proteins.get(i).protein_id
+"_csv.txt");
                File file = new File("./CSV_Results/test/file" +
i + ".csv");
                pW = new PrintWriter(file);
                for (int j = 0; j <
test_proteins.get(i).msa_repr.length; j++) {
                    for (int z = 0; z <
```

```
test_proteins.get(i).msa_repr[j].length; z++) {

pW.print(test_proteins.get(i).msa_repr[j][z]);
                        pW.print(",");
                    }
                    /*

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
r_count]);

pW.print(train_proteins.get(i).second_order_output_repr[sec_orde
r_count+1]);

pW.println(train_proteins.get(i).second_order_output_repr[sec_or
der_count+2]);
                    */
                    s = "" +
test_proteins.get(i).second_order_output_repr[sec_order_count] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
1] +
test_proteins.get(i).second_order_output_repr[sec_order_count +
2];
                    pW.println(Integer.parseInt(s, 2) - 1); //-1
to start from 0
                    sec_order_count += 3;
                }
                pW.close();

            }
            pW.close();
        } catch (java.io.FileNotFoundException e) {
            System.out.println("FILE NOT FOUND!!!");
            System.exit(1);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("OTHER EXCEPTION FOUND!!!! ");
            System.exit(1);
        }
    }


    public static void main(String args[]) {
        //LOAD DATA FILES INTO PROTEINS_ARRAY OBJECT
        Proteins_files_creator a1 = new
Proteins_files_creator("train_set1_new.txt",
"test_set1_new.txt");
```

```
        //CREATE .CSV FILES TO PASS THEM TO DEEPLEARNING4J
READER LIBRARIES.
        //a1.create_csv_files_train_allInOne();
        //a1.create_csv_files_test_allInOne();
        //a1.create_csv_train_files_separate();
        //a1.create_csv_test_files_separate();
        a1.create_csv_files_train_Xneighboring_technique(15);
        a1.create_csv_files_test_Xneighboring_technique(15);

        System.out.println(counter);
    }
}
```

Παράρτημα Β

Γενικά Σχόλια για το παράρτημα Β:

Στο συγκεκριμένο παράρτημα βλέπουμε το αρχείο κώδικα oneCnn.java, στο οποίο βλέπουμε τον κώδικα για την δημιουργία και εκπαίδευση του συνελικτικού νευρωνικού δικτύου.

Μετά το πέρας της δημιουργία και εκπαίδευσης του συνελικτικού νευρωνικού δικτύου, παίρνουμε τα αποτελέσματα πρόβλεψης για τα δεδομένα εκπαίδευσης και επαλήθευσης (SOV_analysis_tr.txt και SOV_analysis_te.txt αντίστοιχα), καθώς επίσης και τα feature vectors (Feature_vectors.txt), τα οποία εξηγήσαμε στην ενότητα 5.11, για την μετέπειτα εκπαίδευση ενός BRNN.

oneCnn.java

```
/*
*
* Convolutional Neural Network implementation, training and
testing.
*
* After the training and testing of CNN the accuracy results are
shown on the console
* and three files are created.
* - SOV_analysis_tr.txt that is the file holding the predictions
for the training set.
*   This .txt will be used to prepare the files with the name,
primary, secondary and predicted
*   secondary structures for each protein in the training set.
*
* - SOV_analysis_te.txt that is the file holding the predictions
for the testing set.
*   This .txt will be used to prepare the files with the name,
primary, secondary and predicted
*   secondary structures for each protein in the testing set.
*
* - Feature_vectors.txt that holds the responces of the last
hidden convolutional layer of the CNN.
*
* Author: Antreas Dionysiou
*
* Library used: DeepLearning4j
*
* */

import org.datavec.api.records.reader.RecordReader;
import org.datavec.api.records.reader.impl.csv.CSVRecordReader;
import org.datavec.api.split.FileSplit;
import
org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.deeplearning4j.eval.Evaluation;
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.LearningRatePolicy;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.Updater;
import org.deeplearning4j.nn.conf.inputs.InputType;
import org.deeplearning4j.nn.conf.layers.ConvolutionLayer;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
```

```
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import
org.deeplearning4j.optimize.listeners.ScoreIterationListener;
import org.deeplearning4j.ui.standalone.ClassPathResource;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.lossfunctions.LossFunctions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class oneCnn {

    private static Logger log =
LoggerFactory.getLogger(oneCnn.class);

    public static void main(String[] args) throws  Exception {
        //First: get the dataset using the record reader.
CSVRecordReader handles loading/parsing
        int numLinesToSkip = 0;
        String delimiter = ",";
        RecordReader recordReader = new
CSVRecordReader(numLinesToSkip,delimiter);
        recordReader.initialize(new FileSplit(new
ClassPathResource("TOBEFILTERED2/WIN5_FILTERED_FROM_CNN_TR.txt")
.getFile()));
        RecordReader recordReader1 = new
CSVRecordReader(numLinesToSkip,delimiter);
        recordReader1.initialize(new FileSplit(new
ClassPathResource("TOBEFILTERED2/WIN5_FILTERED_FROM_CNN_TE.txt")
.getFile()));

        //Second: the RecordReaderDataSetIterator handles
conversion to DataSet objects, ready for use in neural network
        int labelIndex = 5;     //5 values in each row of the
iris.txt CSV: 4 input features followed by an integer label
(class) index. Labels are the 5th value (index 4) in each row
        int numClasses = 3;     //3 classes (types of iris
```

```
flowers) in the iris data set. Classes have integer values 0, 1
or 2
        int trainbatchSize = 7000;    //31286 examples total. We
are loading all of them into one DataSet (not recommended for
large data sets)
        int testBatchSize=7458; // 15695 total
        int numOfTestBatches=1;
        int numOfTrainBatches=11;
        int epochs=100;
        DataSetIterator iterator = new
RecordReaderDataSetIterator(recordReader,trainbatchSize,labelInd
ex,numClasses,numOfTrainBatches);
        DataSetIterator iterator1 = new
RecordReaderDataSetIterator(recordReader1,testBatchSize,labelInd
ex,numClasses,numOfTestBatches);
        //DataSet trainingData = iterator.next();
        //DataSet testData = iterator1.next();


        //We need to normalize our data. We'll use
NormalizeStandardize (which gives us mean 0, unit variance):
        //DataNormalization scaler = new
NormalizerMinMaxScaler(0,1);
        //scaler.fit(iterator);
        //iterator.setPreProcessor(scaler);
        //iterator1.setPreProcessor(scaler);

        final int numInputs = 20;
        int outputNum = 3;
        int iterations = 1;
        int nChannels = 1;

        log.info("Build model....");
        // learning rate schedule in the form of <Iteration #,
Learning Rate>
        Map<Integer, Double> lrSchedule = new HashMap<>();

        lrSchedule.put(0, 0.1);
        lrSchedule.put(200,0.05);
        lrSchedule.put(300,0.01);
        lrSchedule.put(800,0.006);
        lrSchedule.put(1000, 0.04);
        lrSchedule.put(1500,0.02);
        lrSchedule.put(2000,0.01);
        lrSchedule.put(3000,0.008);
        lrSchedule.put(4000,0.006);
```

```
        lrSchedule.put(6000,0.003);
        lrSchedule.put(7000,0.001);
        lrSchedule.put(9000,0.0005);
        lrSchedule.put(10000,0.0001);

        MultiLayerConfiguration conf = new
NeuralNetConfiguration.Builder()
                .iterations(iterations) // Training iterations
as above
                .regularization(true).l2(0.0005)

.learningRateDecayPolicy(LearningRatePolicy.Schedule)
                .learningRateSchedule(lrSchedule)
                .weightInit(WeightInit.RELU)

.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESC
ENT)
                .updater(Updater.NESTEROVS).momentum(0.85)
                .list()
                .layer(0, new ConvolutionLayer.Builder()
                        .kernelSize(1,1)
                        .stride(1,1)
                        //nIn and nOut specify depth. nIn here
is the nChannels and nOut is the number of filters to be applied
                        .nIn(nChannels) //for ConvolutionLayer
                        //.padding(1,1)
                        .nOut(10)
                        .activation(Activation.LEAKYRELU)
                        .build())
                .layer(1, new ConvolutionLayer.Builder()
                        .kernelSize(1,1)
                        .stride(1,1)
                        //Note that nIn need not be specified in
later layers
                        .nIn(10)
                        //.padding(1,1)
                        .nOut(10)
                        .activation(Activation.LEAKYRELU)
                        .build())
                .layer(2, new ConvolutionLayer.Builder()
                        .kernelSize(1,1)
                        .stride(1,1)
                        //Note that nIn need not be specified in
later layers
                        .nIn(10)
                        //.padding(1,1)
```

```
                        .nOut(10)
                        .activation(Activation.LEAKYRELU)
                        .build())
                .layer(3, new DenseLayer.Builder()
                        .weightInit(WeightInit.XAVIER)
                        .activation(Activation.SOFTMAX)
                        .nOut(3)
                        .build())
                .layer(4, new
OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELI
HOOD)
                        .nIn(3)
                        .nOut(outputNum)
                        .weightInit(WeightInit.XAVIER)
                        .activation(Activation.SOFTMAX)
                        .build())

.setInputType(InputType.convolutionalFlat(1,5,1)) //See note
below
                .backprop(true).pretrain(false).build();


        //run the model
        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();

    /*
        //Initialize the user interface backend
        UIServer uiServer = UIServer.getInstance();

        //Configure where the network information (gradients,
score vs. time etc) is to be stored. Here: store in memory.
        StatsStorage statsStorage = new InMemoryStatsStorage();
//Alternative: new FileStatsStorage(File), for saving and
loading later

        //Attach the StatsStorage instance to the UI: this
allows the contents of the StatsStorage to be visualized
        uiServer.attach(statsStorage);

        //Then add the StatsListener to collect this information
from the network, as it trains
        model.setListeners(new StatsListener(statsStorage));
*/

        //TRAINING
```

```
        model.setListeners(new ScoreIterationListener(10));
//comment for UI VISUALIZATION
        System.out.println("START TRAINING!");
        DataSet trainNextSet=null;
        for (int j=0;j<epochs;j++) {
            System.out.println("EPOCH "+j);
            for (int i = 0; i < numOfTrainBatches; i++) {
                if (iterator.hasNext()) {
                    trainNextSet = iterator.next();
                    model.fit(trainNextSet);
                }

            }
            iterator.reset();
            /*
            //test on j epoch
            //TESTING
            System.out.println("START TESTING!");
            Evaluation a = model.evaluate(iterator1);
            System.out.println(a.stats());
            System.out.println("END OF TESTING!");
            */
        }



        //TESTING
        System.out.println("START TESTING!");
        Evaluation a = model.evaluate(iterator1);
        System.out.println(a.stats());
        System.out.println("END OF TESTING!");


        int i=0;
        //UNCOMMENT TO GET PREDICTED RESULTS FOR CREATION OF
SOV_analysis.txt file for SOV score


        //GET PREDICTED RESULTS FROM TRAINSET
        iterator.reset();
        DataSet trainDataSet = null;
        File out_SOV_train = new File("./SOV_analysis_tr.txt");
        PrintWriter writertrain = new PrintWriter(out_SOV_train,
"UTF-8");
        i =0;
        double max1=0;
```

```
        int max_pos1=0;
        int trainsets=0;
        while (trainsets<numOfTrainBatches) {
            trainDataSet = iterator.next();
            i=0;
            while (i < trainDataSet.numExamples() &&
trainDataSet.get(i) != null) {
                //System.out.println(i);
                INDArray arr =
model.output(trainDataSet.get(i).getFeatureMatrix());
                max1 = arr.getDouble(0);
                max_pos1 = 0;
                if (arr.getDouble(1) > max1) {
                    max1 = arr.getDouble(1);
                    max_pos1 = 1;
                }
                if (arr.getDouble(2) > max1) {
                    max1 = arr.getDouble(2);
                    max_pos1 = 2;
                }
                if (max_pos1 == 0) {
                    writertrain.println("C");
                } else if (max_pos1 == 1) {
                    writertrain.println("E");
                } else if (max_pos1 == 2) {
                    writertrain.println("H");
                }
                i++;
            }
            trainsets++;
        }

        //DO THE SAME FOR LAST REMAINING AMINOS
        numOfTrainBatches =12;
        iterator = new
RecordReaderDataSetIterator(recordReader,trainbatchSize,labelInd
ex,numClasses,numOfTrainBatches);
        trainDataSet = iterator.next();
        System.out.println(trainDataSet.numExamples());
        i=0;
        while (i < 92) { //92aminos remain fold0
            //523 fold 1
            //System.out.println(i);
            INDArray arr =
model.output(trainDataSet.get(i).getFeatureMatrix());
            max1 = arr.getDouble(0);
```

```
            max_pos1 = 0;
            if (arr.getDouble(1) > max1) {
                max1 = arr.getDouble(1);
                max_pos1 = 1;
            }
            if (arr.getDouble(2) > max1) {
                max1 = arr.getDouble(2);
                max_pos1 = 2;
            }
            if (max_pos1 == 0) {
                writertrain.println("C");
            } else if (max_pos1 == 1) {
                writertrain.println("E");
            } else if (max_pos1 == 2) {
                writertrain.println("H");
            }
            i++;
        }

        writertrain.close();

        // GET PREDICTED RESULTS FROM TESTSET
        iterator1.reset();
        DataSet testDataSet = iterator1.next();
        File out_SOV = new File("./SOV_analysis_te.txt");
        PrintWriter writer = new PrintWriter(out_SOV, "UTF-8");
        i =0;
        double max=0;
        int max_pos=0;
        while (i<6858 && testDataSet.get(i)!=null){
            //System.out.println(i);
            INDArray arr =
model.output(testDataSet.get(i).getFeatureMatrix());
            max =arr.getDouble(0);
            max_pos=0;
            if (arr.getDouble(1)>max){
                max= arr.getDouble(1);
                max_pos =1;
            }
            if (arr.getDouble(2)>max){
                max=arr.getDouble(2);
                max_pos=2;
            }
            if (max_pos==0){
                writer.println("C");
            }
```

```
            else if (max_pos==1){
                writer.println("E");
            }
            else if (max_pos==2){
                writer.println("H");
            }
            i++;
        }
        writer.close();

        System.out.println("SOV ANALYSIS FILES COMPLETED!");
        System.out.println("CALCULATING FEATURE VECTORS!");



        //GET FEATURE VECTORS FROM LAST-1 FULL CONNECTED NETWORK
        iterator.reset(); //reset training iterator
        DataSet set;
        List<INDArray> activations =null;
        File feature_vectors = new
File("./Feature_vectors.txt");
        PrintWriter feature_vec = new
PrintWriter(feature_vectors, "UTF-8");
        while(iterator.hasNext()) {
            set =iterator.next();
            i=0;
            INDArray temp=null;
            while (i < trainDataSet.numExamples() && set.get(i)
!= null) {
                activations =
model.feedForward(set.get(i).getFeatureMatrix());
                //na fkalw ta output twn feature vectors se
kathe layer.
                temp=activations.get(2);
                for (int j=0;j<temp.rank();j++){

//System.out.println(temp.getRow(0).getRow(j));
                    //for lines of array
                    for (int
z=0;z<temp.getRow(0).getRow(j).size(0);z++){
                        //for columns of line
                        for (int
z1=0;z1<temp.getRow(0).getRow(j).size(1);z1++){

feature_vec.print(String.format("%.4f",temp.getRow(0).getRow(j).
getRow(z).getDouble(z1)));
```

```
                            feature_vec.print(",");
                        }
                    }
                }
                if
(set.get(i).getLabels().getColumn(0).getInt(0)==1){
                    feature_vec.println("0");
                }
                else if
(set.get(i).getLabels().getColumn(1).getInt(0)==1){
                    feature_vec.println("1");
                }
                else if
(set.get(i).getLabels().getColumn(2).getInt(0)==1){
                    feature_vec.println("2");
                }
                i++;
            }
        }
        feature_vec.close();

    }
}
```

Παράρτημα Γ

Γενικά Σχόλια για το παράρτημα Γ:

Στο παράρτημα Γ βλέπουμε τρία αρχεία κώδικα με ονόματα Sov_file_preparation.java, Sov_file_preparation_perc.java και Sov_file_preparation_PISCES.java.

Το πρώτο αρχείο κώδικα (Sov_file_preparation.java) λαμβάνει σαν είσοδο τα αρχεία SOV_analysis_tr.txt και SOV_analysis_te.txt τα οποία δημιουργούνται μετά την εκπαίδευση και επαλήθευση του συνελικτικού νευρωνικού δικτύου που φαίνεται στο παράρτημα Β (αρχείο κώδικα oneCnn.java), και ετοιμάζει κάποια νέα αρχεία (Sov_ready_files_te.txt και Sov_ready_files_tr.txt) με την πιο κάτω μορφή, για τις πρωτεΐνες εκπαίδευσης και επαλήθευσης για το CB513 dataset:

1. Όνομα πρωτεΐνης.
2. Πρωτοταγής δομή.
3. Δευτεροταγής δομή.
4. Προβλεπόμενη δευτεροταγής δομή από το CNN.

Π.χ.

1pptA_1-36

GPSQPTYPGDDAPVEDLIRFYDNLQQYLNVVTRHRY
CCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHCCCCC
CCCCCCCCCCCCCHHHHHHHHHHHHHHHEEEEECCCC

Το δεύτερο αρχείο Sov_file_preparation_perc.java πραγματοποιεί την ίδια διαδικασία με το πρώτο αρχείο αλλά σαν προβλεπόμενη δευτεροταγή δομή από το CNN βάζει τα ποσοστά ενεργοποίησης για κάθε κατηγορία δευτεροταγούς δομής.

Τέλος, το τρίτο αρχείο Sov_file_preparation_PISCES.java πραγματοποιεί την ίδια δικασία με το πρώτο αρχείο αλλά για το PISCES dataset.

Sov_file_preparation.java

```
/*
*
* Author: Antreas Dionysiou
*
* This code prepares the id, primary, secondary and predicted
secondary structure
* file for each protein in training and testing datasets, given
two input files with
* the predictions for each amino acid and the initial training
and testing datasets.
*
* */


import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class Sov_file_preparation {


    public static void main(String[] args) {

        File predicted_res = new
File("./Predicted_results/SOV_analysis_te_2.txt");
        File  train_set_filename = new
File("./CB513_10_FOLDS/testSet3");
        Scanner scan;
        Scanner scan1;
        PrintWriter write1;
        try {
            write1= new
PrintWriter("./SOV_FILES/Sov_ready_files_te_2.txt");
            scan = new Scanner(predicted_res);
            scan1 = new Scanner(train_set_filename);

            //get Sov_ready_files ready
            String prot_name="";
            String prim_str="";
            String seco_str="";
            String pred_str="";
            int count=0;
            int sinolo =0;
```

```
        int sinolo1=0;
        while (scan1.hasNextLine()){
            prot_name = scan1.nextLine();
            prot_name = prot_name.replaceAll("!","");
            write1.println(prot_name);
            //System.out.println(prot_name);
            if (scan1.hasNextLine()){
                prim_str = scan1.nextLine();
                prim_str = prim_str.replaceAll("!","");
                write1.println(prim_str);
                //System.out.println(prim_str);
            }
            if (scan1.hasNextLine()){
                seco_str= scan1.nextLine();
                seco_str = seco_str.replaceAll("!","");
                write1.println(seco_str);
                //System.out.println(seco_str);
            }
            int numOfAminos = seco_str.length();
            pred_str="";
            //! does not exist in msas so dont consider it.
            for (int i=0;i<numOfAminos;i++){
                if (scan.hasNextLine()){
                    pred_str+=scan.nextLine();
                }
            }
            write1.println(pred_str);
            //System.out.println(pred_str);
            count++;
            sinolo+=seco_str.length();
            sinolo1+=pred_str.length();
        }

        scan.close();
        scan1.close();
        write1.flush();
        write1.close();
    }
    catch (FileNotFoundException e){
        System.err.println("FILE NOT FOUND!!!!");
        System.exit(1);
    }


    //do the same for training
    predicted_res = new
```

```
File("./Predicted_results/SOV_analysis_tr_2.txt");
        train_set_filename = new
File("./CB513_10_FOLDS/trainSet3");
        scan=null;
        scan1=null;
        write1=null;
        try {
            write1= new
PrintWriter("./SOV_FILES/Sov_ready_files_tr_2.txt");
            scan = new Scanner(predicted_res);
            scan1 = new Scanner(train_set_filename);

            //get Sov_ready_files ready
            String prot_name="";
            String prim_str="";
            String seco_str="";
            String pred_str="";
            int count=0;
            int sinolo =0;
            int sinolo1=0;
            while (scan1.hasNextLine()){
                prot_name = scan1.nextLine();
                prot_name = prot_name.replaceAll("!","");
                write1.println(prot_name);
                //System.out.println(prot_name);
                if (scan1.hasNextLine()){
                    prim_str = scan1.nextLine();
                    prim_str = prim_str.replaceAll("!","");
                    write1.println(prim_str);
                    //System.out.println(prim_str);
                }
                if (scan1.hasNextLine()){
                    seco_str= scan1.nextLine();
                    seco_str = seco_str.replaceAll("!","");
                    write1.println(seco_str);
                    //System.out.println(seco_str);
                }
                int numOfAminos = seco_str.length();
                pred_str="";
                //! does not exist in msas so dont consider it.
                for (int i=0;i<numOfAminos;i++){
                    if (scan.hasNextLine()){
                        pred_str+=scan.nextLine();
                    }
                }
                write1.println(pred_str);
```

```
                //System.out.println(pred_str);
                count++;
                sinolo+=seco_str.length();
                sinolo1+=pred_str.length();
            }

            scan.close();
            scan1.close();
            write1.flush();
            write1.close();
        }
        catch (FileNotFoundException e){
            System.err.println("FILE NOT FOUND!!!!");
            System.exit(1);
        }
    }
}
```

Sov_file_preparation_perc.java

```
/*
*
* Author: Antreas Dionysiou
*
* This code prepares the id, primary, secondary and predicted
secondary structure percentages
* for each secondary structure class file for each protein in
training and testing datasets,
* given two input files with the predictions for each amino acid
and the initial training and
* testing CB513 datasets.
*
* */

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class Sov_file_preparation_perc {


    public static void main(String[] args) {

        File predicted_res = new
File("./Predicted_results/SOV_analysus_pers_te.txt");
        File  train_set_filename = new
File("./CB513_10_FOLDS/testSet9");
        Scanner scan;
        Scanner scan1;
        PrintWriter write1;
        try {
            write1= new
PrintWriter("./SOV_FILES/Sov_ready_files_te.txt");
            scan = new Scanner(predicted_res);
            scan1 = new Scanner(train_set_filename);

            //get Sov_ready_files ready
            String prot_name="";
            String prim_str="";
            String seco_str="";
            String pred_str="";
            int count=0;
            int sinolo =0;
```

```
            int sinolo1=0;
            while (scan1.hasNextLine()){
                prot_name = scan1.nextLine();
                prot_name = prot_name.replaceAll("!","");
                write1.println(prot_name);
                //System.out.println(prot_name);
                if (scan1.hasNextLine()){
                    prim_str = scan1.nextLine();
                    prim_str = prim_str.replaceAll("!","");
                    write1.println(prim_str);
                    //System.out.println(prim_str);
                }
                if (scan1.hasNextLine()){
                    seco_str= scan1.nextLine();
                    seco_str = seco_str.replaceAll("!","");
                    write1.println(seco_str);
                    //System.out.println(seco_str);
                }
                int numOfAminos = seco_str.length();
                pred_str="";
                //! does not exist in msas so dont consider it.
                for (int i=0;i<numOfAminos;i++){
                    if (scan.hasNextLine()){
                        pred_str+=scan.nextLine()+",";
                    }
                }
                write1.println(pred_str);
                //System.out.println(pred_str);
                count++;
                sinolo+=seco_str.length();
                sinolo1+=pred_str.length();
            }

            scan.close();
            scan1.close();
            write1.close();
        }
        catch (FileNotFoundException e){
            System.err.println("FILE NOT FOUND!!!!");
            System.exit(1);
        }


        //do the same for training
        predicted_res = new
File("./Predicted_results/SOV_analysus_pers_tr.txt");
```

```
        train_set_filename = new
File("./CB513_10_FOLDS/trainSet9");
        scan=null;
        scan1=null;
        write1=null;
        try {
            write1= new
PrintWriter("./SOV_FILES/Sov_ready_files_tr.txt");
            scan = new Scanner(predicted_res);
            scan1 = new Scanner(train_set_filename);

            //get Sov_ready_files ready
            String prot_name="";
            String prim_str="";
            String seco_str="";
            String pred_str="";
            int count=0;
            int sinolo =0;
            int sinolo1=0;
            while (scan1.hasNextLine()){
                prot_name = scan1.nextLine();
                prot_name = prot_name.replaceAll("!","");
                write1.println(prot_name);
                //System.out.println(prot_name);
                if (scan1.hasNextLine()){
                    prim_str = scan1.nextLine();
                    prim_str = prim_str.replaceAll("!","");
                    write1.println(prim_str);
                    //System.out.println(prim_str);
                }
                if (scan1.hasNextLine()){
                    seco_str= scan1.nextLine();
                    seco_str = seco_str.replaceAll("!","");
                    write1.println(seco_str);
                    //System.out.println(seco_str);
                }
                int numOfAminos = seco_str.length();
                pred_str="";
                //! does not exist in msas so dont consider it.
                for (int i=0;i<numOfAminos;i++){
                    if (scan.hasNextLine()){
                        pred_str+=scan.nextLine()+",";
                    }
                }
                write1.println(pred_str);
                //System.out.println(pred_str);
```

```
                count++;
                sinolo+=seco_str.length();
                sinolo1+=pred_str.length();
            }

            scan.close();
            scan1.close();
            write1.close();
        }
        catch (FileNotFoundException e){
            System.err.println("FILE NOT FOUND!!!!");
            System.exit(1);
        }
    }
}
```

Sov_file_preparation_PISCES.java

```
/*
*
* Author: Antreas Dionysiou
*
* This code prepares the id, primary, secondary and predicted
secondary structure percentages
* for each secondary structure class file for each protein in
training and testing datasets,
* given two input files with the predictions for each amino acid
and the initial training and
* testing PISCES datasets.
*
* */

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class Sov_file_preparation_PISCES {


    public static void main(String[] args) {

        //GET READY TESTING FILE FOR SOV ANALYSIS
        File predicted_res = new
File("./Predicted_results/SOV_analysis_te.txt");
        File  train_set_filename = new
File("./PISCES_DATASET_CORRECTED_BY_ANTREAS/train_test_sets1/tes
t_set1_new.txt");
        Scanner scan;
        Scanner scan1;
        PrintWriter write1;
        try {
            write1= new
PrintWriter("./SOV_FILES/Sov_ready_files_te.txt");
            scan = new Scanner(predicted_res);
            scan1 = new Scanner(train_set_filename);

            //get Sov_ready_files ready
            String prot_name="";
            String prim_str="";
            String seco_str="";
            String pred_str="";
```

```
        int count=0;
        int sinolo =0;
        int sinolo1=0;
        while (scan1.hasNextLine()){
            prot_name = scan1.nextLine();
            prot_name = prot_name.replaceAll("!","");
            write1.println(prot_name);
            //System.out.println(prot_name);
            if (scan1.hasNextLine()){
                prim_str = scan1.nextLine();
                prim_str = prim_str.replaceAll("!","");
                write1.println(prim_str);
                //System.out.println(prim_str);
            }
            if (scan1.hasNextLine()){
                seco_str= scan1.nextLine();
                seco_str = seco_str.replaceAll("!","");
                write1.println(seco_str);
                //System.out.println(seco_str);
            }
            int numOfAminos = seco_str.length();
            pred_str="";
            //! does not exist in msas so dont consider it.
            for (int i=0;i<numOfAminos;i++){
                if (scan.hasNextLine()){
                    pred_str+=scan.nextLine();
                }
            }
            write1.println(pred_str);
            //System.out.println(pred_str);
            count++;
            sinolo+=seco_str.length();
            sinolo1+=pred_str.length();
        }

        scan.close();
        scan1.close();
        write1.close();
    }
    catch (FileNotFoundException e){
        System.err.println("FILE NOT FOUND!!!!");
        System.exit(1);
    }


    //do the same for training
```

```
        predicted_res = new
File("./Predicted_results/SOV_analysis_tr.txt");
        train_set_filename = new
File("./PISCES_DATASET_CORRECTED_BY_ANTREAS/train_test_sets1/tra
in_set1_new.txt");
        scan=null;
        scan1=null;
        write1=null;
        try {
            write1= new
PrintWriter("./SOV_FILES/Sov_ready_files_tr.txt");
            scan = new Scanner(predicted_res);
            scan1 = new Scanner(train_set_filename);

            //get Sov_ready_files ready
            String prot_name="";
            String prim_str="";
            String seco_str="";
            String pred_str="";
            int count=0;
            int sinolo =0;
            int sinolo1=0;
            while (scan1.hasNextLine()){
                prot_name = scan1.nextLine();
                prot_name = prot_name.replaceAll("!","");
                write1.println(prot_name);
                //System.out.println(prot_name);
                if (scan1.hasNextLine()){
                    prim_str = scan1.nextLine();
                    prim_str = prim_str.replaceAll("!","");
                    write1.println(prim_str);
                    //System.out.println(prim_str);
                }
                if (scan1.hasNextLine()){
                    seco_str= scan1.nextLine();
                    seco_str = seco_str.replaceAll("!","");
                    write1.println(seco_str);
                    //System.out.println(seco_str);
                }
                int numOfAminos = seco_str.length();
                pred_str="";
                //! does not exist in msas so dont consider it.
                for (int i=0;i<numOfAminos;i++){
                    if (scan.hasNextLine()){
                        pred_str+=scan.nextLine();
                    }
```

```
                }
                write1.println(pred_str);
                //System.out.println(pred_str);
                count++;
                sinolo+=seco_str.length();
                sinolo1+=pred_str.length();
            }

            scan.close();
            scan1.close();
            write1.close();
        }
        catch (FileNotFoundException e){
            System.err.println("FILE NOT FOUND!!!!");
            System.exit(1);
        }
    }
}
```

Παράρτημα Δ

Γενικά Σχόλια για το παράρτημα Δ:

Στο συγκεκριμένο παράρτημα βλέπουμε το αρχείο κώδικα Train_Test_FilesCreator.java το οποίο χρησιμοποιήσαμε για να ετοιμάσουμε τα σωστά αρχεία εκπαίδευσης με τη μορφή: όνομα πρωτεΐνης, πρωτοταγής και δευτεροταγής δομή, για το PIESCES dataset, έτσι ώστε να δοθούν σε όλη την ερευνητική ομάδα του Πανεπιστημίου Κύπρου για εκπαίδευση των νευρωνικών τους δικτύων και εκπόνηση της έρευνας τους. Τα διορθωμένα αρχεία εκπαίδευσης και επαλήθευσης θα αποθηκευτούν σε ένα server για μελλοντική χρήση τους από οποιονδήποτε ερευνητή θέλει να ασχοληθεί με αυτό το θέμα.

Train_Test_FilesCreator.java

```
/*
*
* Author: Antreas Dionysiou
*
* This file prepares the correct training and testing datasets
(PISCES) with the format
* protein's name (id), primary, secondary structure to be used
by all researchers
* at University of Cyprus.
*
*
* */


import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class Train_Test_FilesCreator {

    //main
    public static void main(String[] args) {
        String trainf = "test_set5";
        File train_file = new File("./pisces/pisces_dataset/" +
trainf + ".txt");
        File pisces_dataset = new
File("./pisces/pisces_dataset/pisces_dataset.txt");
        PrintWriter pw;
        PrintWriter proteins_notfound;
        try {
            String line;
            String protein_name;
            //read all lines of train file given in trainf.
            Scanner scan = new Scanner(train_file);
            //open a reader to pisces_dataset to get primary and
secondary stucture.
            Scanner pisces_data;
            proteins_notfound = new
PrintWriter("./notfoundproteins.txt");
            File excluded = new File("./excluded_proteins.txt");
            Scanner exclude = new Scanner(excluded);
            int count=0;
            while (exclude.hasNextLine()){
```

```
                count++;
                exclude.nextLine();
            }
            String[] ex_files = new String [count];
            exclude.close();
            exclude=new Scanner(excluded);
            count=0;
            while (exclude.hasNextLine()){
                String name = exclude.nextLine();
                ex_files[count]=name;
                count++;
            }
            exclude.close();
            //excluded proteins in ex_Files[] array!!
           //set new writer to final output file
            int flag=0;
            pw = new PrintWriter("./" + trainf + "_new.txt");
            int lines=0;
            while (scan.hasNext()) {
                lines++;
                //System.out.println(lines);
                flag=0;
                line = scan.nextLine();
                protein_name = line;
                //check if corrupted or zeroed
                for (int cor=0;cor<ex_files.length;cor++){
                    if (protein_name.equals(ex_files[cor])){
                        //if found raise flag
                       flag=1;
                    }
                }
                if (flag==1){
                    //should be excluded
                    continue;
                }
                //else protein should be okay so
                //find rest info for protein.
                //first reset scanner.
                pisces_data = new Scanner(pisces_dataset);
                String read_name =
pisces_data.nextLine().substring(1);
                while (pisces_data.hasNext() &&
!read_name.equals(protein_name)) {
                    //skip primary and secondary structures of
not matched protein
                    if (pisces_data.hasNext())
```

```
                    pisces_data.nextLine();
                if (pisces_data.hasNext())
                    pisces_data.nextLine();
                if (pisces_data.hasNext()) {
                    read_name =
pisces_data.nextLine().substring(1);
                }
            }
            if (pisces_data.hasNext()==false &&
!read_name.equals(protein_name)){
                //protein not found in pisces_dataset.txt so
write it in file proteins not found
                proteins_notfound.println(protein_name);
                continue;
            }
            //else write protein's info in new file.
            pw.println(protein_name);
            //protein found
            //output primary
            if (pisces_data.hasNextLine()){
                pw.println(pisces_data.nextLine());
            }
            //output secondary
            if (pisces_data.hasNextLine()){
                pw.println(pisces_data.nextLine());
            }
            pisces_data.close();
        }
        pw.close();
        proteins_notfound.close();
    } catch (FileNotFoundException e) {
        System.err.println("FILE NOT FOUND!!!!");
        System.exit(1);
    }
  }
}
```

Παράρτημα Ε

Γενικά Σχόλια για το παράρτημα Ε:

Στο συγκεκριμένο παράρτημα βλέπουμε το αρχείο κώδικα gabor_fil.py (Python) στο οποίο φαίνονται όλες οι συναρτήσεις που χρησιμοποιήσαμε για τις τεχνικές εφαρμογής Gabor φίλτρων (Ενότητα 5.7, Πειράματα 6.3 και 6.4), την εισαγωγή θορύβου στα δεδομένα εισόδου (Πείραμα 6.5), και διάφορες άλλες συναρτήσεις για ένωση των Gabored αρχείων δεδομένων, διαχωρισμό ενός αρχείου δεδομένων σε δεδομένα εκπαίδευσης και επαλήθευσης ανάλογα με ένα ποσοστό που δίνεται (από 0-1) για το σύνολο δεδομένων εκπαίδευσης κτλ.

gabor_fil.py

```
import numpy as np
import copy
import os
import os.path

"""BY CHANGING SIGMA YOU MAKE SEARCH REGION BIGGER.
BY CHANGING THETA YOU CAN SEARCH FOR FEATURES ORIENTENT IN ANY
PARTICULAR DIRECTION"""
def gabor_fn(sigma, theta, Lambda, psi, gamma):
    sigma_x = sigma
    sigma_y = float(sigma) / gamma

    # Bounding box
    nstds = 3 # Number of standard deviation sigma
    xmax = max(abs(nstds * sigma_x * np.cos(theta)), abs(nstds *
sigma_y * np.sin(theta)))
    xmax = np.ceil(max(1, xmax))
    ymax = max(abs(nstds * sigma_x * np.sin(theta)), abs(nstds *
sigma_y * np.cos(theta)))
    ymax = np.ceil(max(1, ymax))
    xmin = -xmax
    ymin = -ymax
    (y, x) = np.meshgrid(np.arange(ymin, ymax + 1),
np.arange(xmin, xmax + 1))

    # Rotation
    x_theta = x * np.cos(theta) + y * np.sin(theta)
    y_theta = -x * np.sin(theta) + y * np.cos(theta)

    gb = np.exp(-.5 * (x_theta ** 2 / sigma_x ** 2 + y_theta **
2 / sigma_y ** 2)) * np.cos(2 * np.pi / Lambda * x_theta + psi)
    return gb


""" CREATES A BIG GABOR KERNEL ARRAY AND MULTIPLY EACH ELEMENT
OF INPUT WITH
THE APPROPRIATE GABOR KERNEL CELL. PRODUCES ALL1.TXT"""
def filter_file(filename,trainOrTest,filters):
    if trainOrTest==0:
        with open("./train/"+str(filename)+".csv", 'r') as
msafile:
            lines = msafile.readlines()
    elif trainOrTest==1:
        with open("./test/"+str(filename)+".csv", 'r') as
```

```
msafile:
            lines = msafile.readlines()
    msafile.close()
    # Creates a list containing number of lines lists, each of
20 items, all set to 0
    w, h = 20, len(lines);
    Matrix1 = [[0 for x in range(w)] for y in range(h)]
    labels=[0 for x in range(h)]
    #fill 2d array with msa csv file integers
    for i in range(0,len(lines)):
        line = lines[i].translate(None,'\n ')
        splited=line.split(",")
        string = ""
        for j in range(0,len(splited)-1):
            Matrix1[i][j]=splited[j]
            string+=str(Matrix1[i][j])+" "
        labels[i]=splited[len(splited)-1]

    #multiply gabor filters with msa data array and produce new
file.
    for i in range(0,len(filters)):
        #apply i filter
        # choose filter
        filter = filters[i]
        #Copy initial msa data array to a temp list Matrix.
        Matrix = copy.deepcopy(Matrix1)
        for j in range(0, len(Matrix)):
            for z in range(0, len(Matrix[0])):

Matrix[j][z]=float(Matrix[j][z])*float(filter[j][z])
        #write new file with filter i applied.
        if trainOrTest==0:
            path =
"./filteredTrainFiles/"+str(filename)+"_"+str(i)+".csv"
        elif trainOrTest==1:
            path = "./filteredTestFiles/" +
str(filename)+"_"+str(i)+".csv"
        with open(path, 'w') as msafile:
            for row in range(0, len(Matrix)):
                for col in range(0, len(Matrix[0])):
                    msafile.write(str(Matrix[row][col])+",")
                #write label to last column
                msafile.write(str(labels[row]))
                msafile.write("\n")
        msafile.close()
```

```
"""FILTER INPUT BY GETTING THE AVERAGE OF NEIGBOUR CELLS
INCLUDING THE SPECIFIED CELL
PRODUCES ALL2.TXT"""
def mean_sum_neig_filter(filename,trainOrTest):
    if trainOrTest==0:
        with open("./train/"+str(filename), 'r') as msafile:
            lines = msafile.readlines()
    elif trainOrTest==1:
        with open("./test/"+str(filename), 'r') as msafile:
            lines = msafile.readlines()
    msafile.close()
    # Creates a list containing number of lines lists, each of
20 items, all set to 0
    w, h = 20, len(lines);
    Matrix1 = [[0 for x in range(w)] for y in range(h)]
    labels=[0 for x in range(h)]
    #fill 2d array with msa csv file integers
    for i in range(0,len(lines)):
        line = lines[i].translate(None,'\n ')
        splited=line.split(",")
        string = ""
        for j in range(0,len(splited)-1):
            Matrix1[i][j]=int(splited[j])
            string+=str(Matrix1[i][j])+" "
        labels[i]=splited[len(splited)-1]

    #find mean neighbor for all neighbour pixels
    for i in range(0,len(Matrix1)):
        for j in range(0,len(Matrix1[0])):
            if i==0 and j==0:

Matrix1[i][j]=(Matrix1[i][j]+Matrix1[i+1][j]+Matrix1[i][j+1]+Mat
rix1[i+1][j+1])/4
            elif i==len(Matrix1)-1 and j==0:
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i -
1][j] + Matrix1[i][j + 1] + Matrix1[i - 1][j + 1]) / 4
            elif i==0 and j==len(Matrix1[0])-1:
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i][j-1]
+ Matrix1[i+1][j] + Matrix1[i + 1][j - 1]) / 4
            elif i==len(Matrix1)-1 and j==len(Matrix1[0])-1:
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i -
1][j] + Matrix1[i][j - 1] + Matrix1[i - 1][j - 1]) / 4
            elif i==0:
```

```
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i][j-1]
+ Matrix1[i][j + 1] + Matrix1[i + 1][j -
1]+Matrix1[i+1][j]+Matrix1[i+1][j+1]) / 6
            elif i==len(Matrix1)-1:
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i][j-1]
+ Matrix1[i][j + 1] + Matrix1[i - 1][j - 1]+Matrix1[i-
1][j]+Matrix1[i-1][j+1]) / 6
            elif j==0:
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i][j+1]
+ Matrix1[i-1][j] + Matrix1[i + 1][j]+Matrix1[i-
1][j+1]+Matrix1[i+1][j+1]) / 6
            elif j==len(Matrix1[0])-1:
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i][j-1]
+ Matrix1[i-1][j] + Matrix1[i - 1][j -
1]+Matrix1[i+1][j]+Matrix1[i+1][j-1]) / 6
            else:
                Matrix1[i][j] = (Matrix1[i][j] + Matrix1[i-1][j]
+ Matrix1[i+1][j] + Matrix1[i][j - 1]+Matrix1[i][j+1]+Matrix1[i-
1][j-1]+Matrix1[i-1][j+1]+Matrix1[i+1][j-1]+Matrix1[i+1][j+1]) /
9

    #write new file with filter i applied.
    if trainOrTest==0:
        path = "./mean_sum_neig_filter_train/" + str(filename)
    elif trainOrTest==1:
        path = "./mean_sum_neig_filter_test/" + str(filename)
    with open(path, 'w') as msafile:
        for row in range(0, len(Matrix1)):
            for col in range(0, len(Matrix1[0])):
                msafile.write(str(Matrix1[row][col])+",")
            #write label to last column
            msafile.write(str(labels[row]))
            msafile.write("\n")
    msafile.close()

"""SCALES UP MSA VALUES. PRODUCES ALL3.TXT"""
def make_msa_nums_bigger(filename,trainOrTest):
    #READ MS TO MATRIX1 AND LABELS ARRAYS
    if trainOrTest==0:
        with open("./train/"+str(filename), 'r') as msafile:
            lines = msafile.readlines()
    elif trainOrTest==1:
        with open("./test/"+str(filename), 'r') as msafile:
            lines = msafile.readlines()
    elif trainOrTest==2:
        #if all proteins file gived mixes and you want to
```

```
separate them in train and testing
        with open("./ProteinMsaFiles/"+str(filename), 'r') as
msafile:
            lines = msafile.readlines()
    msafile.close()
    # Creates a list containing number of lines lists, each of
20 items, all set to 0
    w, h = 20, len(lines);
    Matrix1 = [[0 for x in range(w)] for y in range(h)]
    labels=[0 for x in range(h)]
    #fill 2d array with msa csv file integers
    for i in range(0,len(lines)):
        line = lines[i].translate(None,'\n ')
        splited=line.split(",")
        string = ""
        for j in range(0,len(splited)-1):
            #here specify how many times to get each value
bigger by multiplying
            Matrix1[i][j]=int(splited[j])*2
            string+=str(Matrix1[i][j])+" "
        labels[i]=splited[len(splited)-1]
    #ARRAYS MATRIX1 AND LABELS CREATED AND FILLED AT THIS POINT
    #NOW MAKE MSA FILES NUMBERS BIGGER BY DOUBLING OR 3* ETC...

    # write new file with bigger msa files.
    if trainOrTest == 0:
        path = "./bigger_msa_numbers_train/" + str(filename)
    elif trainOrTest == 1:
        path = "./bigger_msa_numbers_test/" + str(filename)
    elif trainOrTest==2:
        path="./ProteinMsaFiles/bigger_msa_files/" +
str(filename)
    with open(path, 'w') as msafile:
        for row in range(0, len(Matrix1)):
            for col in range(0, len(Matrix1[0])):
                msafile.write(str(Matrix1[row][col]) + ",")
            # write label to last column
            msafile.write(str(labels[row]))
            msafile.write("\n")
    msafile.close()


"""GETS A GABOR FILTER BANK AND APPLIES THEM TO A GIVEN INPUT.
THEN IT PRODUCES ALL4.TXT"""
def gabor_filter_msa_file(filename, trainOrTest,filters):
    # READ MS TO MATRIX1 AND LABELS ARRAYS
```

```
    if trainOrTest == 0:
        with open("./train/" + str(filename), 'r') as msafile:
            lines = msafile.readlines()
    elif trainOrTest == 1:
        with open("./test/" + str(filename), 'r') as msafile:
            lines = msafile.readlines()
    msafile.close()
    # Creates a list containing number of lines lists, each of
20 items, all set to 0
    #also 2 columns and rows filled with 0 for padding.
    w, h = 22, len(lines)+2;
    Matrix1 = [[0 for x in range(w)] for y in range(h)]
    labels = [0 for x in range(h)]
    # fill 2d array with msa csv file integers
    for i in range(0, len(lines)):
        line = lines[i].translate(None, '\n ')
        splited = line.split(",")
        string = ""
        for j in range(0, len(splited) - 1):
            # here specify how many times to get each value
bigger by multiplying
            Matrix1[i+1][j+1] = int(splited[j])
        labels[i] = splited[len(splited) - 1]
        # ARRAYS MATRIX1 AND LABELS CREATED AND FILLED AT THIS
POINT
    #CONTINUE BY CONVOLVING GABOR KERNEL WITH INPUT MATRIX1.
GABOR FILTER APPLICATION.
    #first copy Matrix1 to create new filtered Matrix2 values
there.
    Matrix2 = copy.deepcopy(Matrix1)
    #set scale factor.STUDY TO FIND THE BEST.!!
    scale=1
    # for each gabor filter in filter bank produce filtered
matrix1 file.
    for a in range(0, len(filters)):
        #choose filter
        filter=filters[a]
        #for each input cell o Matrix1.
        for i in range(1,len(Matrix1)-1): #from 1,len.. and -1
because 1 line of padding exist.
            for j in range(1,len(Matrix1[0])-1): #from 1,len..
and -1 for same as above.
                sum =0;
                #for each gabor kernel cell.
                row=-1 #for multiplying correct input Matrix1
cells.
```

```
                col=-1 # live the line above.
                for ker_row in range(0,len(filter)):
                    for ker_col in
range(0,len(filter[ker_row])):

sum+=(filter[ker_row][ker_col]*Matrix1[i+row][j+col])
                        col+=1
                    col=-1
                    row+=1
                Matrix2[i][j]=sum/scale
        #write new file to gabored_train or test folder.
        #set new filename
        filename1= filename.replace('.csv','_'+str(a)+'.csv')
        if trainOrTest == 0:
            path = "./gabored_flipped_train/" + str(filename1)
        elif trainOrTest == 1:
            path = "./gabored_flipped_test/" + str(filename1)
        with open(path, 'w') as msafile:
            for row in range(1, len(Matrix2)-1):
                for col in range(1, len(Matrix2[0])-1):
                    msafile.write(str(Matrix2[row][col]) + ",")
                # write label to last column
                msafile.write(str(labels[row]))
                msafile.write("\n")
        msafile.close()

"""This function is used to fliped a kernel given i order to be
applied
later for a proper 2d convolution."""
def flip(filter):
    lines = len(filter)
    columns= len(filter[0])
    row=lines-1
    #print "ROWS:"+str(lines)
    #print "COLUMNS:"+str(columns)
    col=columns-1
    temp=0
    for i in range(0,int(lines/2)):
        for j in range(0,columns):
            temp=filter[i][j]
            filter[i][j]= float(filter[row][col])
            filter[row][col]=float(temp)
            col-=1
        row-=1
        col=columns-1
    #finally flip the middle line
```

```
    mid = int(lines/2)
    #print mid
    col =columns-1
    mid_col=columns/2
    #print col
    for i in range(0,mid_col):
        temp = filter[mid][i]
        filter[mid][i]=float(filter[mid][col])
        filter[mid][col]=float(temp)
        col-=1


"""SEPARATES A FILE TO TRAINING AND TESTING SUBFILES ACCORDING
TO
A PRECENTAGE GIVEN FOR TRAINING (0-1). IT PRODUCES FILES
train_all.txt and
test_all.txt"""
def create_tr_te_withpososta(pososto Training,path):
    if os.path.isfile(path+"/test_all.txt"):
        os.remove(path+"/test_all.txt")
    if os.path.isfile(path+"/train_all.txt"):
        os.remove(path+"/train_all.txt")
    #afairese analoga gia na ine swsto to num arxiwn
    num_arxiwn=len(os.listdir(path))-2
    print num_arxiwn
    train_pro_num=(int)(pososto Training*num_arxiwn)
    print train_pro_num
    text=""
    for i in range(0,train_pro_num):
        with open(path+"/file"+str(i)+".csv",'r') as myfile:
            text += myfile.read()
        myfile.close
    with open(path+"/train_all.txt",'w') as resultfile:
        resultfile.write(text)
    resultfile.close
    text = ""
    for i in range(train_pro_num, num_arxiwn):
        with open(path+"/file" + str(i) + ".csv",'r') as myfile:
            text += myfile.read()
        myfile.close
    with open(path+"/test_all.txt",'w') as resultfile:
        resultfile.write(text)
    resultfile.close

"""COMBINES ALL TRAIN AND TEST FILES THAT EXIST IN THE SPECIFIC
DIRECTROTIES
```

```
STATED BELOW."""
def create_all_in_one_files():
    numOfTrainFiles = len(os.listdir("./filteredTrainFiles/"))-1
    numOfTestFiles = len(os.listdir("./filteredTestFiles/"))-1
    print numOfTestFiles
    print numOfTrainFiles
    text=""
    for i in range(0,numOfTrainFiles):
        with open("./filteredTrainFiles/file"+str(i)+".csv",'r')
as myfile:
            text += myfile.read()
        myfile.close
    with open("./protein_csv_train_all1Flipped.txt",'w') as
resultfile:
        resultfile.write(text)
    resultfile.close
    text = ""
    for i in range(0, numOfTestFiles):
        with open("./filteredTestFiles/file" + str(i) +
".csv",'r') as myfile:
            text += myfile.read()
        myfile.close
    with open("./protein_csv_test_all1Flipped.txt",'w') as
resultfile:
        resultfile.write(text)
    resultfile.close

"""COMBINES ALL GABORED TRAIN AND TEST FILES THAT EXIST IN THE
SPECIFIC DIRECTROTIES
STATED BELOW."""
def create_all_in_one_files_GABORED():
    numOfTrainFiles = len(os.listdir("./filteredTrainFiles/"))-1
    numOfTestFiles = len(os.listdir("./filteredTestFiles/"))-1
    print numOfTestFiles
    print numOfTrainFiles
    text=""
    for i in range(0,numOfTrainFiles/7):
        for j in range(0,7):
            with
open("./filteredTrainFiles/file"+str(i)+"_"+str(j)+".csv",'r')
as myfile:
                text += myfile.read()
            myfile.close
    with open("./protein_csv_train_all1Flipped.txt",'w') as
resultfile:
        resultfile.write(text)
```

```
    resultfile.close
    text = ""
    for i in range(0, numOfTestFiles/7):
        for j in range(0,7):
            with
open("./filteredTestFiles/file"+str(i)+"_"+str(j)+".csv",'r') as
myfile:
                text += myfile.read()
            myfile.close
    with open("./protein_csv_test_all1Flipped.txt",'w') as
resultfile:
        resultfile.write(text)
    resultfile.close


"""This function creates protein_csv_test_all.txt and
protein_csv_train_all.txt
by combining training and testing files given the path."""
def combile_train_test():
    numOfTrainFiles = len(os.listdir("./train/")) - 1
    numOfTestFiles = len(os.listdir("./test/")) - 1
    print "NUMBER OF TRAIN FILES IS:"
    print numOfTrainFiles
    print "NUMBER OF TEST FILES IS:"
    print numOfTestFiles
    text = ""
    for i in range(0, numOfTrainFiles):
        with open("./train/file" + str(i) + ".csv", 'r') as
myfile:
            text += myfile.read()
        myfile.close
    with open("./protein_csv_train_all.txt", 'w') as resultfile:
        resultfile.write(text)
    resultfile.close
    text = ""
    for i in range(0, numOfTestFiles):
        with open("./test/file" + str(i) + ".csv", 'r') as
myfile:
            text += myfile.read()
        myfile.close
    with open("./protein_csv_test_all.txt", 'w') as resultfile:
        resultfile.write(text)
    resultfile.close

"""This function reverses files in order to train cnn from both
sides."""
```

```
def reverse_file(filename):
    with open(filename) as f, open(filename+"_REVERSED", 'w') as
fout:
        fout.writelines(reversed(f.readlines()))


def main():
    numOfTrainFiles=len(os.listdir("./train/"))-1
    numOfTestFiles=len(os.listdir("./test/"))-1
    print "NUMBER OF FILES FOR TRAINING:"
    print numOfTrainFiles
    print "NUMBER OF FILES FOR TESTING:"
    print numOfTestFiles
    #uncoment for all1 files creation. comment otherwise.
    """
    #set convolution=1 to flip gabor kernels in order to apply
convolution.
    #set convolution=0 to apply correlation.
    convolution=1
    print "Start filters preparation."
    # parameters explanation:
https://dsp.stackexchange.com/questions/14714/understanding-the-
gabor-filter-function
    filters = []
    filters.append(gabor_fn(200, 0, 2, 0, 1))
    filters.append(gabor_fn(200, 30, 2, 0, 1))
    filters.append(gabor_fn(200, 60, 2, 0, 1))
    filters.append(gabor_fn(200, 90, 2, 0, 1))
    filters.append(gabor_fn(200, 120, 2, 0, 1))
    filters.append(gabor_fn(200, 150, 2, 0, 1))
    filters.append(gabor_fn(200, 180, 2, 0, 1))
    print "Filters successfully prepared."
    if convolution==1:
        print "Flipping kernels 180 degrees to apply
convolution."
        for i in range(0,len(filters)):
            flip(filters[i])
        print "Kernels flipped succesfully. Now 2d convolution
will be applied."
    elif convolution==0:
        print "Applying correlation."
    for i in range(0, numOfTrainFiles):
        print i
        filter_file("file" + str(i), 0,filters)
    for i in range(0, numOfTestFiles):
        print "t" + str(i)
```

```
        filter_file("file" + str(i) , 1,filters)
    """
    #uncoment for all2 files creation. comment otherwise.
    """
    for i in range(0, numOfTrainFiles):
        print i
        mean_sum_neig_filter("file" + str(i) + ".csv", 0)
    for i in range(0, numOfTestFiles):
        print "t" + str(i)
        mean_sum_neig_filter("file" + str(i) + ".csv", 1)
    """
    #uncoment for all3 files creation(files in different
direcorys). comment otherwise.
    """
    for i in range(0,numOfTrainFiles):
        print i
        make_msa_nums_bigger("file"+str(i)+".csv",0)
    for i in range(0,numOfTestFiles):
        print "t"+str(i)
        make_msa_nums_bigger("file"+str(i)+".csv",1)
    """
    #uncoment for all3 files creation(files in same directory).
comment otherwise.
    """
    numofAllFiles= len(os.listdir("./ProteinMsaFiles/"))
    #set limit to number of all files given i ProteinMsaFiles
folder
    for i in range(0,200):
        make_msa_nums_bigger("file"+str(i)+".csv", 2)
    """
    # uncoment for all4_3x3 files creation. comment otherwise.
    """
    #FIRST PREPARE FILTERS TO USE.
    filters = []
    filters.append(gabor_fn(0.3, 0, 2, 0, 1))
    filters.append(gabor_fn(0.3, 30, 2, 0, 1))
    filters.append(gabor_fn(0.3, 60, 2, 0, 1))
    filters.append(gabor_fn(0.3, 90, 2, 0, 1))
    filters.append(gabor_fn(0.3, 120, 2, 0, 1))
    filters.append(gabor_fn(0.3, 150, 2, 0, 1))
    filters.append(gabor_fn(0.3, 180, 2, 0, 1))
    print filters[0]
    for i in range(0,numOfTrainFiles):
        print i
        gabor_filter_msa_file("file"+str(i)+".csv",0,filters)
    for i in range(0,numOfTestFiles):
```

```
        print "t"+str(i)
        gabor_filter_msa_file("file"+str(i)+".csv",1,filters)
    """
    # uncoment for all4_5x5 files creation. comment otherwise.
    #UNDER CONSTRUCTION!!!
    """
    convolution=1
    #FIRST PREPARE FILTERS TO USE.
    filters = []
    filters.append(gabor_fn(0.4, 0, 2, 0, 1))
    filters.append(gabor_fn(0.4, 30, 2, 0, 1))
    filters.append(gabor_fn(0.4, 60, 2, 0, 1))
    filters.append(gabor_fn(0.4, 90, 2, 0, 1))
    filters.append(gabor_fn(0.5, 120, 2, 0, 1))
    filters.append(gabor_fn(0.5, 150, 2, 0, 1))
    filters.append(gabor_fn(0.5, 180, 2, 0, 1))
    if convolution == 1:
        print "Flipping kernels 180 degrees to apply
convolution."
        for i in range(0, len(filters)):
            flip(filters[i])
        print "Kernels flipped succesfully. Now 2d convolution
will be applied."
    elif convolution == 0:
        print "Applying correlation."
    for i in range(0,numOfTrainFiles):
        print i
        gabor_filter_msa_file("file"+str(i)+".csv",0,filters)
    for i in range(0,numOfTestFiles):
        print "t"+str(i)
        gabor_filter_msa_file("file"+str(i)+".csv",1,filters)
    """

if __name__ == "__main__":
    main()
    #reverse_file("protein_csv_train_all.txt")
    #create_all_in_one_files()
    #create_all_in_one_files_GABORED()
    #create_tr_te_withpososta(0.8,"./ProteinMsaFiles")
    #combile_train_test()
```

Παράρτημα Z

Γενικά Σχόλια για το παράρτημα Z:

Στο συγκεκριμένο παράρτημα βλέπουμε το αρχείο κώδικα keras_birnn.py (Python) στο οποίο φαίνεται ο κώδικας δημιουργείας και επαίδευσης ενός BRNN με χρήση των feature vectors τα οποία έχουμε εξάγει από το τελευταίο κρυφό συνελικτικό επίπεδο του CNN (Ενότητα 5.11, Πείραμα 6.9).

Keras_birnn.py

```
'''Trains a Bidirectional LSTM on the IMDB sentiment
classification task.
Output after 4 epochs on CPU: ~0.8146
Time per epoch on CPU (Core i7): ~150s.
'''
from __future__ import print_function
import numpy as np
from keras.optimizers import SGD

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Embedding, LSTM,
Bidirectional
from keras.datasets import imdb
import numpy as np
from tensorflow.contrib.learn.python.learn.estimators._sklearn
import train_test_split


# load pima indians dataset
dataset = np.loadtxt("./Feature_vectors.txt", delimiter=",")
print("DATASET SHAPE:",dataset.shape,"\n")
# split into input (X) and output (Y) variables
X = dataset[:, 0:1496]
Y = dataset[:, [1496]]
print(Y)
print("X SHAPE:",X.shape,"\n")
print("Y SHAPE:",Y.shape,"\n")
#ta x je ta y thkiavazi ta swsta

trainingData, testingData, trainingOutput, testingOutput =
train_test_split(X, Y, test_size=0.30)

trainingData = np.reshape(trainingData,(53963,1496,1))
testingData = np.reshape(testingData,(23127,1496,1))
#print(trainingData.shape)
#print(testingData.shape)
trainingOutput =
np.reshape(trainingOutput,(len(trainingOutput)))
#create new array with 3 nums for each label
y_train_new = np.empty((len(trainingOutput),3))
for i in range(len(trainingOutput)):
    if trainingOutput[i]==0:
        y_train_new[i,0]=1
```

```
        y_train_new[i,1]=0
        y_train_new[i,2]=0
    elif trainingOutput[i]==1:
        y_train_new[i,0]=0
        y_train_new[i,1]=1
        y_train_new[i,2]=0
    elif trainingOutput[i]==2:
        y_train_new[i,0]=0
        y_train_new[i,1]=0
        y_train_new[i,2]=1
    else:
        print("ERROR IT CANT COMPARE TRAIN LABELS PROPERLY!!!!")
        exit(0)

trainingOutput = y_train_new
#trainingOutput =
np.reshape(trainingOutput,(1,len(trainingOutput),3))
print(trainingOutput.shape)


testingOutput = np.reshape(testingOutput,(len(testingOutput)))
#create new array with 3 nums for each label
y_test_new = np.empty((len(testingOutput),3))
for i in range(len(testingOutput)):
    if testingOutput[i]==0:
        y_test_new[i,0]=1
        y_test_new[i,1]=0
        y_test_new[i,2]=0
    elif testingOutput[i]==1:
        y_test_new[i,0]=0
        y_test_new[i,1]=1
        y_test_new[i,2]=0
    elif testingOutput[i]==2:
        y_test_new[i,0]=0
        y_test_new[i,1]=0
        y_test_new[i,2]=1
    else:
        print("ERROR IT CANT COMPARE TEST LABELS PROPERLY!!!!")
        exit(0)

testingOutput= y_test_new
#testingOutput =
np.reshape(testingOutput,(1,len(testingOutput),3))
max_features = 200000000000
maxlen=1496
print('Build model...')
```

```
model = Sequential()
model.add(Bidirectional(LSTM(64, return_sequences=True),
input_shape=(1496, 1)))
model.add(Bidirectional(LSTM(64)))
model.add(Dense(3, activation='softmax'))
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)

# try using different optimizers and different optimizer configs
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
batch_size = 7000
print('Train...')
model.fit(trainingData, trainingOutput,
validation_data=(testingData,testingOutput), epochs=1,
batch_size=53963)
```

Παράρτημα Η

Γενικά Σχόλια για το παράρτημα Η:

Στο συγκεκριμένο παράρτημα βλέπουμε δύο αρχεία κώδικα με ονόματα prepare_SVM_FILES_CSV_FORMAT.py και SVM_CORRECTED.py.

Το πρώτο αρχείο κώδικα (prepare_SVM_FILES_CSV_FORMAT.py) προετιμάζει τα δεδομένα εκπαίδευσης και επαλήθευσης παίρνοντας σαν παράμετρο το εύρος θέασης (window).

Το δεύτερο αρχείο κώδικα (SVM_CORRECTED.py) δημιουργεί, εκπαιδεύει και επαληθεύει ένα SVM παίρνοντας σαν παράμετρο το εύρος θέασης των αρχείων εκπαίδευσης και επαλήθευσης.

prepare_SVM_FILES_CSV_FORMAT.py

```
#open TEST file to read data
with open("./fold1/outPRED_ENSEMBLES_ONLY_te.txt","r") as
testfile:
    lines = testfile.readlines()
testfile.close()
#open TRAIN file to read dat
with open("./fold1/outPRED_ENSEMBLES_ONLY_tr.txt","r") as
trainfile:
    lines_tr = trainfile.readlines()
trainfile.close()
#IT GIVES BETTER RESULTS IF TRAINED WITH THE TESTSET WHICH IS
SMALLER!
linenum=1
window =7
leftwindow =int(window/2)
#create test file
with open("./fold1/fold1_SVM_ready_tr_WIN7.txt", "w") as
svmtrain:
    for line in lines_tr:
        if linenum == 5:
            linenum = 1
        if linenum == 3:
            target_out = line
        if linenum == 4:
            for i in range(leftwindow):
                zeros = leftwindow - i
                for zer in range(zeros):
                    svmtrain.write("0,")
                for rem in range(i):
                    if line[rem] == "C":
                        svmtrain.write("0,")
                    if line[rem] == "E":
                        svmtrain.write("1,")
                    if line[rem] == "H":
                        svmtrain.write("2,")
                #place right aminos
                for j in range(leftwindow+1):
                    if line[i+j] == "C":
                        svmtrain.write("0,")
                    if line[i+j] == "E":
                        svmtrain.write("1,")
                    if line[i+j] == "H":
                        svmtrain.write("2,")
                #place label at the end
```

```
            if target_out[i] == "C":
                svmtrain.write("0")
            if target_out[i] == "E":
                svmtrain.write("1")
            if target_out[i] == "H":
                svmtrain.write("2")
            svmtrain.write("\n")
        #place aminos with no boundary constraints
        for amino in range(leftwindow,len(line)-leftwindow-
1):
            for curr in range(-leftwindow,leftwindow+1):
                if line[amino+curr] == "C":
                    svmtrain.write("0,")
                if line[amino+curr] == "E":
                    svmtrain.write("1,")
                if line[amino+curr] == "H":
                    svmtrain.write("2,")
            #place label
            if target_out[amino] == "C":
                svmtrain.write("0")
            if target_out[amino] == "E":
                svmtrain.write("1")
            if target_out[amino] == "H":
                svmtrain.write("2")
            svmtrain.write("\n")
        #place last aminos with padding
        for i in range(len(line)-leftwindow-1,len(line)-1):
            printed=0
            for left in range(i-leftwindow-1,i):
                if line[left] == "C":
                    svmtrain.write("0,")
                if line[left] == "E":
                    svmtrain.write("1,")
                if line[left] == "H":
                    svmtrain.write("2,")
            for j in range(i,len(line)-1):
                if line[j] == "C":
                    svmtrain.write("0,")
                if line[j] == "E":
                    svmtrain.write("1,")
                if line[j] == "H":
                    svmtrain.write("2,")
                printed=printed+1
            zeros = leftwindow-printed
            for z in range(zeros):
                svmtrain.write("0,")
```

```
                # place label
                if target_out[i] == "C":
                    svmtrain.write("0")
                if target_out[i] == "E":
                    svmtrain.write("1")
                if target_out[i] == "H":
                    svmtrain.write("2")
                svmtrain.write("\n")
        linenum += 1
    svmtrain.flush()
    svmtrain.close()
linenum=1
#create TEST file
with open("./fold1/fold1_SVM_ready_te_WIN7.txt", "w") as
svmtest:
    for line in lines:
        if linenum == 5:
            linenum = 1
        if linenum == 3:
            target_out = line
        if linenum == 4:
            for i in range(leftwindow):
                zeros = leftwindow - i
                for zer in range(zeros):
                    svmtest.write("0,")
                for rem in range(i):
                    if line[rem] == "C":
                        svmtest.write("0,")
                    if line[rem] == "E":
                        svmtest.write("1,")
                    if line[rem] == "H":
                        svmtest.write("2,")
                #place right aminos
                for j in range(leftwindow+1):
                    if line[i+j] == "C":
                        svmtest.write("0,")
                    if line[i+j] == "E":
                        svmtest.write("1,")
                    if line[i+j] == "H":
                        svmtest.write("2,")
                #place label at the end
                if target_out[i] == "C":
                    svmtest.write("0")
                if target_out[i] == "E":
                    svmtest.write("1")
                if target_out[i] == "H":
```

```
                svmtest.write("2")
            svmtest.write("\n")

        #place aminos with no boundary constraints
        for amino in range(leftwindow,len(line)-leftwindow-
1):
            for curr in range(-leftwindow,leftwindow+1):
                if line[amino+curr] == "C":
                    svmtest.write("0,")
                if line[amino+curr] == "E":
                    svmtest.write("1,")
                if line[amino+curr] == "H":
                    svmtest.write("2,")
            #place label
            if target_out[amino] == "C":
                svmtest.write("0")
            if target_out[amino] == "E":
                svmtest.write("1")
            if target_out[amino] == "H":
                svmtest.write("2")
            svmtest.write("\n")
        #place last aminos with padding
        for i in range(len(line)-leftwindow-1,len(line)-1):
            printed=0
            for left in range(i-leftwindow-1,i):
                if line[left] == "C":
                    svmtest.write("0,")
                if line[left] == "E":
                    svmtest.write("1,")
                if line[left] == "H":
                    svmtest.write("2,")
            for j in range(i,len(line)-1):
                if line[j] == "C":
                    svmtest.write("0,")
                if line[j] == "E":
                    svmtest.write("1,")
                if line[j] == "H":
                    svmtest.write("2,")
                printed+=1
            zeros = leftwindow-printed
            for z in range(zeros):
                svmtest.write("0,")
            # place label
            if target_out[i] == "C":
                svmtest.write("0")
            if target_out[i] == "E":
```

```
                svmtest.write("1")
            if target_out[i] == "H":
                svmtest.write("2")
            svmtest.write("\n")
    linenum += 1
svmtest.flush()
svmtest.close()
```

SVM_CORRECTED.py

```
from __future__ import print_function
import numpy as np

import numpy as np
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from tensorflow.contrib.learn.python.learn.estimators._sklearn
import train_test_split


#SKLEAR
from sklearn import svm, pipeline
from sklearn import linear_model


train_dataset =
np.loadtxt("./fold1/fold1_SVM_ready_tr_WIN7.txt", delimiter=",")
win=7
#win=9
X_train = train_dataset[:, 0:win]
y_train = train_dataset[:, [win]]

test_dataset = np.loadtxt("./fold1/fold1_SVM_ready_te_WIN7.txt",
delimiter=",")
X_test = test_dataset[:, 0:win]
y_test = test_dataset[:, [win]]

y_train = np.reshape(y_train,len(y_train))
y_test = np.reshape(y_test,len(y_test))

clf = svm.SVC()
#clf = svm.LinearSVC()

print("Training...")
clf.fit(X_train, y_train)

print("THE SCORE: ", clf.score(X_test, y_test))


#count training samples
ta0=0
ta1=0
ta2=0
for i in range(len(y_train)):
```

```
    if y_train[i] == 0:
        ta0+=1
    elif y_train[i] == 1:
        ta1+=1
    elif y_train[i] ==2:
        ta2+=1
    else:
        print("ERROR!!!")
        exit(0)
#print("TRAINING 0: ",ta0)
#print("TRAINING 1: ",ta1)
#print("TRAINING 2: ",ta2)

#produce manual accuracy report.
all1=0
corr1=0
all2=0
corr2=0
all3=0
corr3=0
c_e=0
c_h=0
e_c=0
e_h=0
h_c=0
h_e=0
for i in range(len(X_test)):
    if y_test[i] == 0:
        all1 += 1
        if clf.predict([X_test[i]]) == 0:
            corr1 += 1
        elif clf.predict([X_test[i]]) ==1:
            c_e+=1
        elif clf.predict([X_test[i]])==2:
            c_h+=1
    elif y_test[i] == 1:
        all2 += 1
        if clf.predict([X_test[i]])==0:
            e_c+=1
        elif clf.predict([X_test[i]]) == 1:
            corr2 += 1
        elif clf.predict([X_test[i]]) ==2:
            e_h+=1
    elif y_test[i] == 2:
        all3 += 1
        if clf.predict([X_test[i]]) ==0:
```

```
            h_c+=1
        elif clf.predict([X_test[i]])==1:
            h_e+=1
        elif clf.predict([X_test[i]]) == 2:
            corr3 += 1
    else:
        print("SOMETHING IS GOING WRONG!!!")
        exit(0)

print("COIL - C:", (100*corr1)/all1)
print("STRAND - E:", (100*corr2)/all2)
print("HELIX - H:", (100*corr3)/all3)
print("Q3 ACCURACY:",
(100*(corr1+corr2+corr3))/(all1+all2+all3))

print("         C           E           H")
print("C    ",corr1,"       ",c_e,"     ",c_h)
print("E    ",e_c,"         ",corr2,"   ",e_h)
print("H    ",h_c,"         ",h_e,"     ",corr3)
```

Παράρτημα Θ

Γενικά Σχόλια για το παράρτημα Θ:

Στο συγκεκριμένο παράρτημα βλέπουμε τα ονόματα των πρωτεινών τις οποίες έχουμε αφαιρέσει από το CB513 dataset λόγω του ότι τα αρχεία MSA τους, ήταν μηδενισμένα (zeroed). Είναι συνολικά 8 πρωτεΐνες.

1. 1coiA_1-29
2. 1mctI_1-28
3. 1tiiC_195-230
4. 2erlA_1-40
5. 1ceoA_202-254
6. 1mrtA_31-61
7. 1wfbB_1- 37
8. 6rlxC_-2-20

Παράρτημα Ι

Γενικά Σχόλια για το παράρτημα Ι:

Στο συγκεκριμένο παράρτημα βλέπουμε τα ονόματα των πρωτεινών τις οποίες έχουμε αφαιρέσει από το PISCES dataset λόγω του ότι αρχεία MSA τους, ήταν κατεστραμμένα (corrupted) ή μηδενισμένα (zeroed). Είναι συνολικά 341 πρωτεΐνες.

1. 3P51A
2. 1V96A
3. 3L7HA
4. 4DHXA
5. 4F2LA
6. 2D7EA
7. 4MTUA
8. 4BSXA
9. 4F87A
10. 4MYVA
11. 1QV9A
12. 3FF5A
13. 4P2VA
14. 1WWPA
15. 2D68A
16. 2R85A
17. 4MO0A
18. 4L3UA
19. 4J5RA
20. 2OL5A
21. 4KTWA
22. 3D33A
23. 3PD7A
24. 3KVPA
25. 3QH6A
26. 3VS8A
27. 3CPTA
28. 3GP6A
29. 3O65A
30. 1TU9A
31. 2OU5A
32. 3L60A
33. 3H0DA
34. 1Q2HA
35. 4I86A
36. 2G7SA
37. 2P63A
38. 3Q18A
39. 2FI1A
40. 2Y5PA
41. 4Q53A
42. 2Q3TA
43. 1VPRA
44. 3DFUA
45. 3DNXA
46. 4GUCA
47. 3Q6CA
48. 4LTBA
49. 3MD9A
50. 3ESMA
51. 3H0NA
52. 4HHVA
53. 3M5QA
54. 4N74A
55. 4KQIA
56. 4IHQA
57. 4K92A
58. 4Q7OA
59. 4J1VA
60. 4X33A
61. 3U5WA
62. 3O6QA
63. 3UV0A
64. 3NS4A
65. 2HQLA
66. 3TE8A
67. 2O38A
68. 4PF3A
69. 4H41A
70. 3H16A
71. 2D59A
72. 1VR4A
73. 2O1QA
74. 2HX5A
75. 2NPTA
76. 3CRYA
77. 2ERWA
78. 3C4RA
79. 2IP6A
80. 3GO9A
81. 1YPYA
82. 3E0RA
83. 4F27A
84. 3PL0A
85. 3IBWA
86. 3TS9A
87. 3D55A
88. 3NOQA
89. 4E6WA
90. 4AQOA

91. 2R19A
92. 3OP6A
93. 2PW9A
94. 4GOFA
95. 3N7XA
96. 2P9XA
97. 2VS0A
98. 2WG8A
99. 4P49A
100. 3ZC0A
101. 3B0FA
102. 2OX7A
103. 4I16A
104. 4PSFA
105. 4AP5A
106. 3K8RA
107. 3D3OA
108. 3HLSA
109. 1WPNA
110. 2O99A
111. 3I76A
112. 4LQBA
113. 4JX0A
114. 4R7RA
115. 4GT9A
116. 4F4WA
117. 2UVPA
118. 1Z4RA
119. 2OU1A
120. 3F67A
121. 1Q9UA
122. 1WV3A
123. 3ONQA
124. 2YF2A
125. 3Q0HA
126. 3TUOA
127. 2E1FA
128. 3C0DA
129. 3IV4A
130. 2CVIA
131. 1SQWA
132. 4BOQA
133. 3W6SA
134. 3TVQA
135. 3LQ9A
136. 3BPQA
137. 2BDRA
138. 3F43A
139. 3G21A
140. 4J91A
141. 4K12A
142. 2Q3SA
143. 4QSGA
144. 2V73A
145. 2WVQA
146. 4NUAA
147. 4G97A
148. 3BRVA
149. 3O12A
150. 3KUPA
151. 2R5SA
152. 2FB0A
153. 3L49A
154. 2O4AA
155. 2NS0A
156. 2ZEXA
157. 3UV1A
158. 3I4UA
159. 3RK6A
160. 4P78A
161. 1UV7A
162. 2HL7A
163. 4U9OA
164. 4BSVA
165. 2C0NA
166. 3I00A
167. 3FH3A
168. 3NR5A
169. 4E6SA
170. 4LKUA
171. 3KTOA
172. 4QRNA
173. 2V9PA
174. 3F95A
175. 1M1QA
176. 3CSXA
177. 4FX7A
178. 3JQOA
179. 4L2WA
180. 3U97A
181. 3ESLA
182. 1U9LA
183. 3H35A
184. 1BB1A
185. 1BB1B
186. 1BB1C
187. 1C94A
188. 1DPJB
189. 1DTDB
190. 1F8VD
191. 1GWMA
192. 1HX6A
193. 1KD8A
194. 1KP6A
195. 1KVEA
196. 1L2WI
197. 1M3WA
198. 1M45B
199. 1M46B
200. 1MCTI
201. 1MQSB
202. 1MW5A
203. 1O06A
204. 1P9IA
205. 1PJMA
206. 1PJNA
207. 1SVFB

208. 1T01B
209. 1TQEX
210. 1TTWB
211. 1U6HB
212. 1UVQC
213. 1ZVZB
214. 1ZW2B
215. 2BPA3
216. 2BPTB
217. 2C5IP
218. 2C5KP
219. 2DS2A
220. 2ERLA
221. 2GWWB
222. 2HZSI
223. 2MLTA
224. 2P06A
225. 2PBDV
226. 2PLXB
227. 2QUOA
228. 2W4YA
229. 2WBYC
230. 2WFUA
231. 2WFVA
232. 2WWXB
233. 2X5CA
234. 2X5RA
235. 2XF7A
236. 2XZEQ
237. 3AJBB
238. 3C5TB
239. 3DT5A
240. 3E8YX
241. 3FBLA
242. 3GP2B
243. 3H0TC
244. 3HE4B
245. 3HE5A
246. 3HE5B
247. 3L9AX
248. 3LCNC
249. 3LJMA
250. 3M6ZA
251. 3NK4C
252. 3OWTC
253. 3P06A
254. 3PLVC
255. 3QKSC
256. 3R46A
257. 3R4AA
258. 3RA3B
259. 3RF3C
260. 3RKLA
261. 3S1BA
262. 3S6PE
263. 3SHPA
264. 3SJHB
265. 3TQ2A
266. 3TWEA
267. 3TZ1B
268. 3U4VA
269. 3U4ZA
270. 3UC7A
271. 3UKXC
272. 3UL1A
273. 3V62C
274. 3V86A
275. 3VU5B
276. 3VU6B
277. 3VVIA
278. 3W8VA
279. 3W92A
280. 3WKNE
281. 3WOEB
282. 3WX4A
283. 3WY9C
284. 3ZTAA
285. 4A94C
286. 4BFHA
287. 4BLQA
288. 4BPLB
289. 4C1AA
290. 4CAYC
291. 4CU4B
292. 4CXFB
293. 4DACA
294. 4EHQG
295. 4F87A
296. 4FBWC
297. 4FTBD
298. 4FZ0M
299. 4G1AA
300. 4GVBB
301. 4H62V
302. 4H7RA
303. 4H8MA
304. 4H8OA
305. 4HB1A
306. 4HBEA
307. 4HLBA
308. 4HR1A
309. 4I7ZE
310. 4IIKA
311. 4J4AA
312. 4JGLA
313. 4JHKC
314. 4KVTA
315. 4KYTB
316. 4LOOB
317. 4M1XA
318. 4M6BC
319. 4MGPA
320. 4MI8C
321. 4N3BB
322. 4N3CB
323. 4OGQE
324. 4OQ9A

325. 4OYDB
326. 4OZKA
327. 4PC0C
328. 4PN8A
329. 4PN9A
330. 4PNAA
331. 4PNBA
332. 4PNDA
333. 4PW1A
334. 4QMFA
335. 4R0RA
336. 4R80A
337. 4R8TA
338. 4RIQC
339. 4TTLA
340. 4UEBB
341. 4W6YA