

DOG RECOGNITION AND IDENTIFICATION USING MACHINE LEARNING

Eleni Aristidou

A Thesis

Submitted in Partial Fulfillment of the
Requirements for the Degree
of Bachelor of Science at the
University of Cyprus

University of Cyprus



Computer Science Department

JUNE 2021

UNIVERSITY OF CYRUS
COMPUTER SCIENCE DEPARTMENT

APPROVAL PAGE

Bachelor of Science Thesis

**DOG RECOGNITION AND IDENTIFICATION
USING MACHINE LEARNING**

Presented by Eleni Aristidou

Research Supervisor

Andreas Aristidou

University of Cyprus

June, 2021

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to Dr. Andreas Arisitdou, post-doc researcher associated with the Department of Computer Science at the University of Cyprus, for being my mentor and helping me in every stage of my thesis. I am extremely thankful and indebted to him for sharing his expertise and giving me the chance to further develop my skills in the field of machine learning. His excitement over his work pushed me to become a better researcher, to always accumulating as many information I can and to find ways to overcome any obstacles I face.

I am also grateful to Tassos Yiannakides, the assistant of my mentor at the Computer Science Department of University of Cyprus, for providing me with all the necessary knowledge needed to begin learning about the technologies that were necessary for the implementation of the tools in my study. He was always on duty to help me and to clear out any information I wanted to analyze. He also helped me to build a proper environment to work on and use all the computing units I needed to train my network. Without his help I would not be able to understand so well the network.

I would also like to thank Ioustina Harasim, an undergraduate student who is working on the same topic of research as me. She is going to optimize my research and create a full representation of the results by creating a mobile application. She involved in almost every process of this study and helped me gather more images for my dataset. Furthermore she was always delighted to help me if I was facing any difficulties. With their help, we were able to solve our problem that was to recognize dogs so pet owners can find their lost dogs.

Finally, I would like to thank my friends and family for always supporting me and encouraging me during my final year at the University of Cyprus.

Eleni Aristidou – University of Cyprus, 2021

ABSTRACT

As we all know, the field of Machine learning has a huge growth the recent years. Deep Neural Networks is an approach in Machine Learning. A Neural Network can be built with layers of neurons and it consists data that the network learns, allowing it to make predictions. Face recognition is now an application of our everyday lives based on deep convolutional networks. Many implementations have been made, so that face recognition and face verification is integrated in many systems such as the facial recognition used on smartphones and in other forms of technology, such as robotics. What about pet facial recognition? Our research is based on FaceNet implementation using deep convolutional neural networks so we can detect, verify and identify individual pet faces in digital images. Analysing the problem of human recognition, we extract important techniques to apply them in pet verification and identification. Processing numerous images of pets, specifically dog pictures, we construct a data set of dog images sourced from internet pet adoption profiles. We use a pretrained model that can detect a dog's face and pre-process the images so that dogs will be aligned properly for the training. Afterwards, using triplet loss to generate 64-dimensional embeddings, we can use multiple methodologies to verify and identify almost pet. We relied on previous researches to understand the concept of building a neural network and tries to enhance the ability of the network to recognise different dogs. In order to succeed, we need to take into consideration various algorithms to study how good results, our approach can accomplish. Many combinations and extensions of the algorithms have been made. Finally, we have designed a simple and modern website to represent our results. It is important to say that Flask has been used to show some of the results we get, not only with data from training but also data that the algorithm has not seen before. We have been able to establish a connection between our web application and run machine learning algorithms that use TensorFlow running in the background of Flask.

Table of contents

Table of contents	5
List of Figures.....	8
List of abbreviations and acronyms	11
Chapter 1: Introduction	12
1.1 Motivation.....	12
1.2 Contributions.....	13
Chapter 2: Literature and Related Work.....	15
2.1 Introduction.....	15
2.2 FaceNet: A Unified Embedding for Face Recognition and Clustering.....	15
2.3 Deep Learning on Animal Biometrics.....	16
2.4 Building a dog search engine with FaceNet.....	17
2.5 DogFaceNet : Dog Identification	18
Chapter 3: Data Acquisition	20
3.1 Data Collection	20
3.2 Image Preprocessing.....	22
3.2.1 Dog Points Detection	22
3.2.2 Stage 2 : Preprocessing	24
Chapter 4: Methodology.....	25
4.1 Overview	26
4.2 History.....	27
4.2.1 Neural Networks	27
4.2.2 Deep Convolutional Neural Networks.....	28
4.3 Network Architecture.....	29
4.4 Open Source Software for Machine Learning	29
4.5 Model Definition.....	30
4.5.1 The process of Model Definition	30
4.5.2 The Layers of our Network.....	31
4.5.2.1 Convolutional Layers and Activation Function.....	31

4.5.2 .2 Pooling Layers	33
4.5.3 Adding more layers to our network	34
4.5.3.1 Applying many Convolutional Networks	34
4.5.3.2 Global Average Pooling.....	36
4.5.3.3 Flatten Layer	36
4.5.3.4 Dropout Layer.....	36
4.5.3.5 Dense Layer	37
4.5.4 Batches.....	38
4.5.4.1 Define the Batches	38
4.5.4.2 Batches Normalization.....	39
4.6 Training	40
4.6.1 Organization of the Data	40
4.6.1.1 Splitting the Dataset.....	40
4.6.1.2 Npy files:.....	41
4.6.2 Optimizer Algorithm.....	41
4.6.2.1 Overview	41
4.6.2.2 Learning Rate.....	42
4.6.2.3 SGD – Adam Optimizer to minimize the error.....	42
4.6.3.1 Triplet Loss	43
4.6.3.2 The margin.....	45
4.6.3.3 Metrics Triplet and Accuracy: Pairwise Ranking Loss	46
4.6.4 Triplet Sampling	47
4.6.4.1 The problem of Overfitting.....	47
4.6.4.2 Hard Triplet Mining Strategy: Hard sampling.....	48
4.6.5 Adaptive Hard Image Generator	49
4.7 Testing.....	50
4.7.2 Introduction.....	50
4.7.3 Useful Functions	50
4.7.3.1 Finding the Best Threshold.....	50
4.7.3.2 Create Embeddings	51
4.7.3.3 Find Distance	51
4.7.4 Algorithms	52
4.7.4.1 Brute Force Nearest Images.....	52
4.7.4.2 Agglomerative Clustering.....	52
4.7.4.3 Hybrid Solution: Cluster the Nearest Images	54

Chapter 5: Evaluation	56
5.1 Overview	56
5.2 Results	57
Chapter 6: Web Application	60
6.1 Motivation.....	60
6.2 Used Technologies.....	60
6.3 Methodology	61
6.3.1 Basic Start for a Flask Application	61
6.3.2 Website Setup	62
6.3.3 Background Algorithm	62
6.4 Web Demonstration and Results	62
Chapter 7: Conclusions, Limitations & Future Work	66
6.1 General Conclusions	68
6.2 Limitations.....	69
6.3 Future work.....	70
Bibliography	71
Appendix A.....	74
Triplet Loss.....	74
Appendix B.....	75
1. Find Nearest Embedding Vectors.....	75
2. Cluster the nearest embeddings.....	76

List of Figures

Chapter 1: Introduction	12
Chapter 2: Literature and Related Work.....	15
Figure 1: Model structure of [1].	16
Figure 2: Overview of Triplet Loss in Human Faces	
Figure 3: Model structure of [1].	16
Figure 4: Dog Breed Identification	17
Figure 5: Animal Biometrics (Beads and ridges)	17
Figure 6: FaceNet Implementation on DogFaceNet - Some Results of his work.....	18
Figure 7: Architecture definition.....	19
Chapter 3: Data Acquisition	20
Figure 8: An example of the dog face dataset	20
Figure 9: Selecting Dog Images through Pet Profiles on Social Media.	21
Figure 10: Visiting Pet Profiles through Social Media.....	21
Figure 11: Dog's Landmark Detection	22
Figure 12: Some saved landmarks in excel file	22
Figure 13: Example image that detector can not Detect Dog's Face.	23
Figure 14: Manual Marking of Nose and Eyes with OpenCV	23
Figure 15: Dog Image Example with its landmarks.....	24
Figure 16: Dog Face Alignment	24
Chapter 4: Methodology.....	25
Figure 17: Passing inputs images into convolutional neural networks	26
Figure 18: Images mapping though DCNN to embedding space.....	27
Figure 21: Artificial Neural Network	27
Figure 21: a neuron.....	27
Figure 21: Deep Neural Network	27
Figure 22: : Visualization of the Network	29
Figure 23 : Each input image will pass it through a series of layers.	30
Figure 24: Relu Function	31
Figure 25:Extraction of features	32
Figure 26: Keras Conv2D	32
Figure 27: Some of the convolved images using many different kernels.....	32
Figure 28:: Example of a Conv2D layer	33
Figure 29: Application of Max Pooling with stride=2	33

Figure 30: Adding various convolutional layers example	34
Figure 31: Example of kernel and its output result	34
Figure 32: Example 2 of kernel and its output result.	35
Figure 33: Shapes that the filters on the left detected from the images	35
Figure 34: Application of Global Average Pooling with stride = 2	36
Figure 35: Flatten Layer	36
Figure 36: Dropout Neural Net Model.....	36
Figure 37: A simple Neural Network with only 1 hidden layer that is fully connected.....	37
Figure 38: A more complex Neural Network with only 3 hidden layers that are fully connected.	37
Figure 39: The architecture of our Network.....	38
Figure 40: Batch Normalization Process	39
Figure 41: For each dimension it is calculated the mean and standard deviation.....	40
Figure 42: Batch Normalization	40
Figure 43: Mean and Variance	42
Figure 44: Backward propagation and forward propagation	43
Figure 45: Example of a triplet ranking loss setup.....	44
Figure 46: Minimization of the distance between the positive and anchor. Maximization of the distance between negative and the anchor.....	44
Figure 47: Example of Triplet Loss learning process.....	45
Figure 48: Triplet loss example 2	45
Figure 49: Final Triplet Loss Formula	45
Figure 50: Triplet Loss learning process.....	46
Figure 51: Loss for Negative and Positive Pairs.....	46
Figure 52: Triplet Loss example for our dog network.....	47
Figure 53: Pairwise Ranking Loss	47
Figure 54: Hard Triplet: Hard Positive and Hard Negative	48
Figure 55: Loss Value to adjust ratio of the hard dataset.....	48
Figure 56: Hard Triplet Selection: Hard Positive and Hard Negative.....	49
Figure 57: Euclidean Distance	51
Figure 58: Results of Clustering	53
Figure 59: Four dogs with their characteristics. Our Network will be able to extract some of them like patterns, eyes, fur texture etc.	54
Figure 60: Clustering Nearest Embeddings.....	54

Figure 61: Elbow Method for Optimal k	55
Figure 62: Clustered Images and Centroids of each cluster	55
Chapter 5: Evaluation	56
Figure 63: SGD algorithm to minimize the Loss and upgrade the Weights.....	56
Figure 64: Batch Loss	57
Figure 65: Batch Triple Accuracy.....	58
Figure 66: Learning Rate based on Loss.....	58
Figure 67: Training and Validation Lost for 100 epochs	59
Figure 68: Training and Validation Accuracy for 100 epochs	59
Chapter 6: Web Application	60
Figure 69: Communication of Flask with the Web page and ML Algorithms	61
Figure 70: Home Page	63
Figure 71: Library Page	64
Figure 72: Search a Dog Page.....	65
Figure 73: Calculating Results	65
Figure 74: Example Result 1.....	66
Figure 75: Example Result 2.....	66
Figure 76: Example Result 3.....	67
Figure 77: Example Result 4.....	67
Chapter 7: Conclusions, Limitations & Future Work	66

List of abbreviations and acronyms

SVM – Support Vector Machine

DCNN – Deep Convolutional Networks

NN – Neural Networks, **CNN** – convolutional Neural Networks

ReLU Rectified Linear Unit

t-SNE - t-Distributed Stochastic Neighbor Embedding

KNN – K Nearest Neighbours

ML – Machine Learning

Chapter 1

Introduction

Contents

1.1 Motivation	12
1.2 Contributions	13

1.1 Motivation

Pets hold a very special place in our hearts and every owner fears the day their beloved furry friend go missing. Usually an owner that lost his dog, will do anything is possible that could help find it. The most frequent way to search for a lost dog is usually looking though social media. Many Facebook pages have been made for this reason. The purpose of these pages is to allow people interact and communicate with others to help him find his lost dog. Commonly when someone loses his dog, he will post a picture of it and write details about the location or any other important information like the collar it last wore, or any special characteristics it may has. The dog may have a unique colour, or very characteristic features that anyone could recognize such a dog by seeing him somewhere. If the owner is lucky enough, people who have seen his dog going around their neighborhood will come in contact with him. In some other cases people who see a dog walking down the street and have not seen him any other day around, will consider that the dog is lost. They may take the dog in their house, or they will just notice and continue their day. People who are willing to help, post the dog, found in their way, in these pages we were talking about previously and wait for someone to inform them if this dog was reported as lost from their owner or they know who their owner is. Another solution for finding the owner of a lost dog is checking if they have a microchip. They go to their nearest vet clinic and ask the vet doctor to check if the dog has an owner. But what happens if the dog does not have a microchip? Or maybe in some other cases, people will not think that checking the microchip would help returning the lost dog to its home. The least scenario that could play nowadays, is to print posters with photos of the dog with a contact number and some other information. The posters are usually tucked to electric poles around the area or they are even given to local shops so that people passing by would see the posters.

In all these scenarios, we have seen that images are really important to the human eye because it is the only way to remember and recognize things. People are trying to remember if

they have seen those lost dogs and try to compare with any other dog they have seen around and thought it was a lost one. It would be much more easier for the pet owners to build an application that will automize the process of matching a lost with a found dog. Our society is more technologically advanced than ever, so why don't we take advantage of it? Computer Vision and Artificial Intelligence combined are here to analyze and understand images. These applications manage to find a meaning behind any patterns and can benefit us to build the system we vision: make the procedure of searching for a lost pet, or reporting a found one, waiting for his owner to look up for him, an automatic process.

We want a system that will be able take an image of a dog and it will be able to track amongst hundreds or thousands of images in a database, the most similar dogs. The process we are going through, uses advanced-level algorithms to conduct deep analysis of the data for our recognition problem. This analysis further facilitates decision-making. Image recognition enables a machine to identify and then categorize any elements detected in an image. The image acts as input to this technique, which in turn offers labels as the output. Based on this output, the model is trained to automatically find patterns by looking into the classes from a predefined list. An Artificial Intelligence system that processes the visual info, that we going to give it, depends on computer vision. Image recognition includes processes like object detection and visual search. We are going to discuss the prime steps of our system of image recognition that is the collection and organization of the data. This data is then utilized to prepare predictive models, which further offers precise outputs. We are also going to represent a real time application of this process. We have already seen in action machine learning techniques in many applications such as Human Face Recognition [1] and proved us that we can build an adequate system. These achievements provide many services and facilities in authentication systems. The human face recognition model is based on finding embeddings for each image using a deep convolutional network. The network is trained such that the images of same person will have small distances and faces of dissimilar people will have large distances. Having the produced embeddings, the model can verify faces calculating the thresholding the distance between the embeddings. By studying the neural network designed in the facial recognition model, we adapt the approach to dog identification. A method for dog face identification and recognition was developed in [2] that could help us solve our problem using animal biometrics.

1.2 Contributions

The ultimate goal of this study is to not only analyse how dog face identification system works but to find ways to improve its results and make a visual representation of an active system, that will be able to help pet owners that have lost their dog. Starting off with the very beginning of the research made in [29] we will observe that the dataset is small compared to

other big systems of machine learning. We find a way to collect more data and add it to our dataset. Its important to pre-process those images so they could work in our solution. Dog detection system is being studied and applied to our images. We need to find the landmarks of the dogs to make changes to the images. Continuingly, we dive into the training of the system which is the most important process. It is a crucial matter to absorb any knowledge that refers to computer vision being used for artificial intelligence methods. Thus, we study the general idea of deep convolutional neural networks and investigate in depth the functions thar are used to build this architecture[2]. We examine the layers added to the network to be able to extract features. Images that the network if being fed with, are several images of dogs and we need to obtain as many characteristics and patterns of theirs. Triplet loss is being firstly introduced in [1] and we experience an another triplet loss sampling : hard triplet loss. This method is also being used in [2] but it needs to being performed in our new data. Moving on to the evaluation procedure, we make observations about the Network and the losses of it and decide which models we will be using. Putting to the test many algorithms such as the agglomerative clustering, the pair verification method using embeddings and a hybrid solution of the previous two, we decided the algorithm that will be able to show us results in an actual problem solving. The implementation of this project[2,3] was developed as a research of recognition appliance to dogs. Until now, it wasn't been developed as an actual application. Our aim is to build a simple and modern web application that can represent the automate procedure of finding your lost friend. In order to do that, we find ways to run machine learning algorithms and the trained system into the background of the website. Our last purpose is to set the start for new researchers to find ways to improve more and more this system and build a final visualisation of the project. Our contribution to dog identification problem using machine learning must be effective on the web demonstration for the people that are going to extend and use this platform.

Chapter 2

Literature and Related Work

Contents

2.1 Introduction	15
2.2 FaceNet: A Unified Embedding for Face Recognition and Clustering.....	15
2.3 Deep Learning on Animal Biometrics.....	16
2.4 Building a dog search engine with FaceNet.....	17
2.5 DogFaceNet : Dog Identification	18

2.1 Introduction

In the fields of deep learning and machine learning, studies for face identification were mainly performed on the human face. The newly proposed model and loss function were evaluated with the human face dataset. Also, statistics in [5,6], showed that the number of companion animals has increased dramatically, and so the number of abandoned or lost animals. To solve this problem, studies were conducted in which the deep learning models for human biometrics were applied to animals [7]. In this section, we will mention the previous studies on human face identification and some other previous studies on animal biometrics [8].

2.2 FaceNet: A Unified Embedding for Face Recognition and Clustering

Deep learning and machine learning models can be trained for human face recognition using large-scale datasets of human faces, leading to high-performance computing resources that have improved. Metric learning has been widely used to train deep learning models for face identification [10]. FaceNet is a general-purpose system that can be used for face verification to see if it is the same person, recognition to check who is this person and clustering that we are looking for similar people. The method adopted by FaceNet[1] is to map the image to Euclidean space through convolutional neural network learning. The spatial distance is directly related to the image similarity: different images of the same person have a small distance in space, and images of different people have a larger distance in space. As long as the mapping is determined, the related face recognition task becomes very simple. FaceNet

achieved 99.63%, and 95.12% accuracy levels for the LFW and YouTube Faces DB datasets, respectively. The inspiration of triplet loss is that the traditional loss function tends to map face images with one type of feature to the same space.[11] Triplet loss attempts to separate an individual's face image from other face images. We will talk about Triplet loss in more details later, in our implementation.



Figure 1: Model structure of [1].

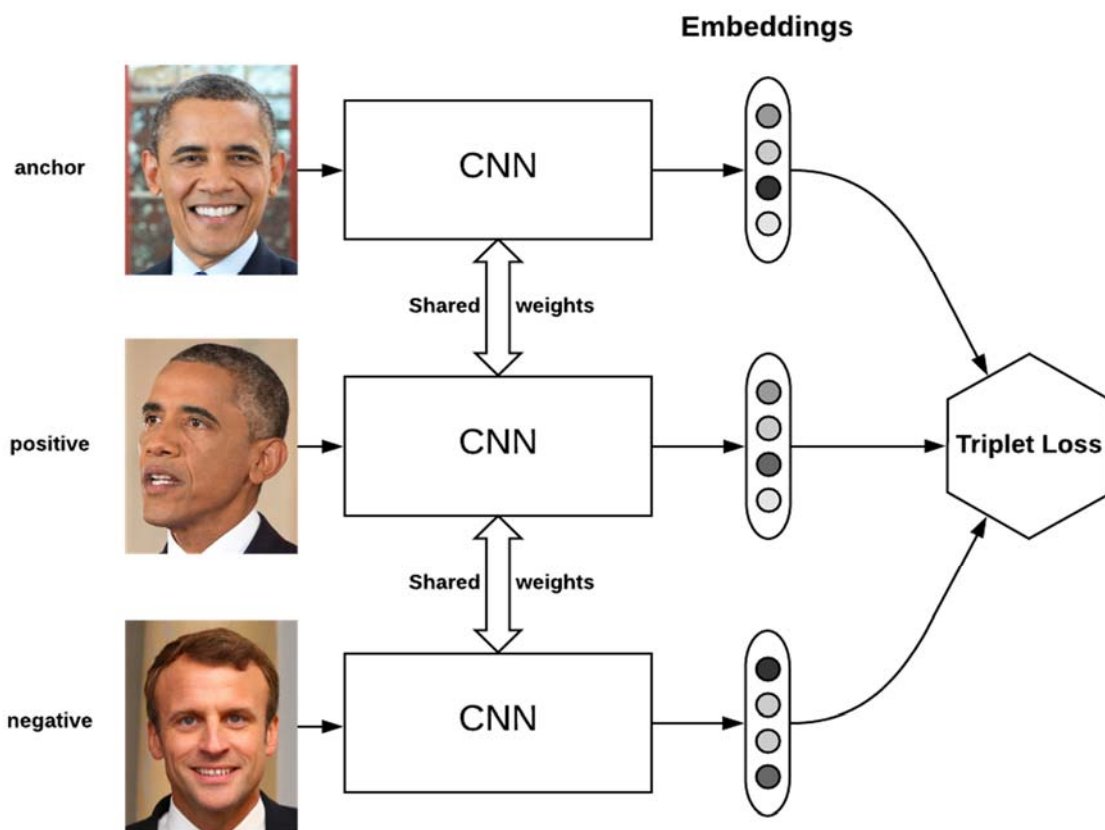


Figure 2: Overview of Triplet Loss in Human Faces [34]

2.3 Deep Learning on Animal Biometrics

In machine learning, many researches on animal biometrics have been made, specifically on cattle [12–15], horses [16], pigs [17] and endangered animals [18,19]. In the Individual Cattle Identification study, they were extracting biometrics using muzzle points [20,22]. In the case of the other animals identification researches like horses, pigs and endangered animals, studies on biometrics-based face recognition have been conducted. [23–26]. However, dogs have mainly been studied for breed classification [20-23]. In [31], animal

biometrics were divided in categories like is the muzzle point, iris pattern, retinal vascular, and face images. In the field of face identification, many existing studies have been made on the human face such as FaceNet. However, when low-level of normality datasets, such as dog faces, are applied in the human face identification models [1], we face the problem of rapid overfitting. A few studies have been made in this regard [24,25,26], to solve this problem and finally construct a network for dog face identification. In [26], the authors collected Flickr dog dataset of 42 dog faces consisting only husky and pug images. They used SVM to classify the features of those photos, using a CNN. Additionally, they attempted breed classification. Breed Classification was trained on a pre-trained GoogleNet [28].

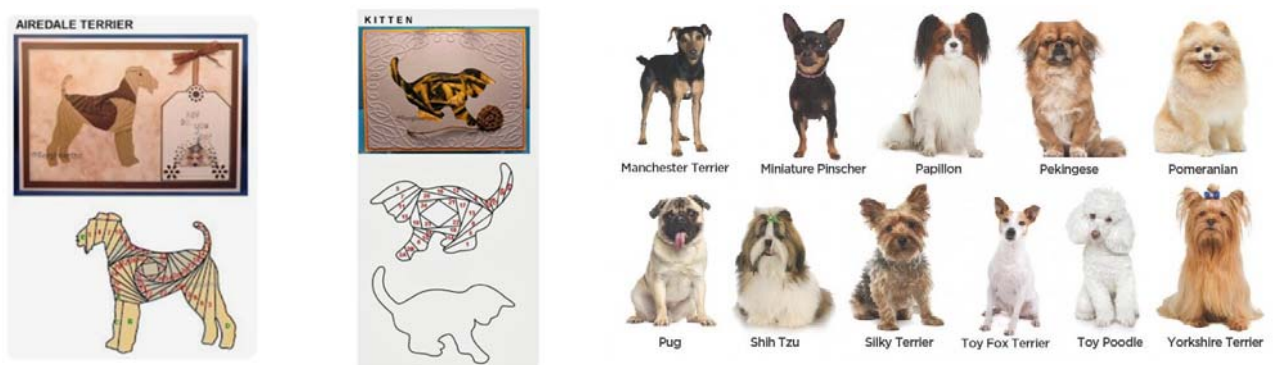


Figure 4: Dog Breed Identification [28]

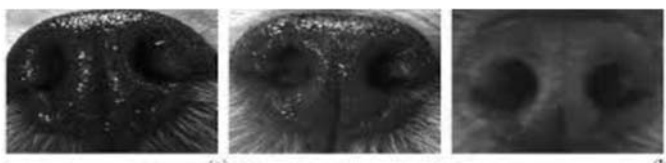


Figure 3: Beads and ridges features of the muzzle point image pattern of cattle from the database. [12-15]

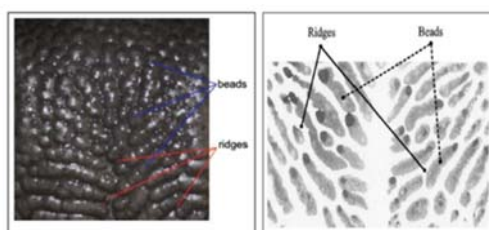


Figure 5: Animal Biometrics (Beads and ridges)

2.4 Building a dog search engine with FaceNet [29]

Two of the most remarkable academic studies that worked on the Dog Identification and Recognition problem[2,29], approached the topic, by training Facenet on DogFaceNet dataset using a custom data loader that implements hard triplet mining. Each implementation

works with a different open-source software. The first study[29], built a dog search engine with FaceNet. It uses PyTorch, a machine learning library based on the Torch library. Their output results are base on KNN. The code is much more simple than the other implementation.

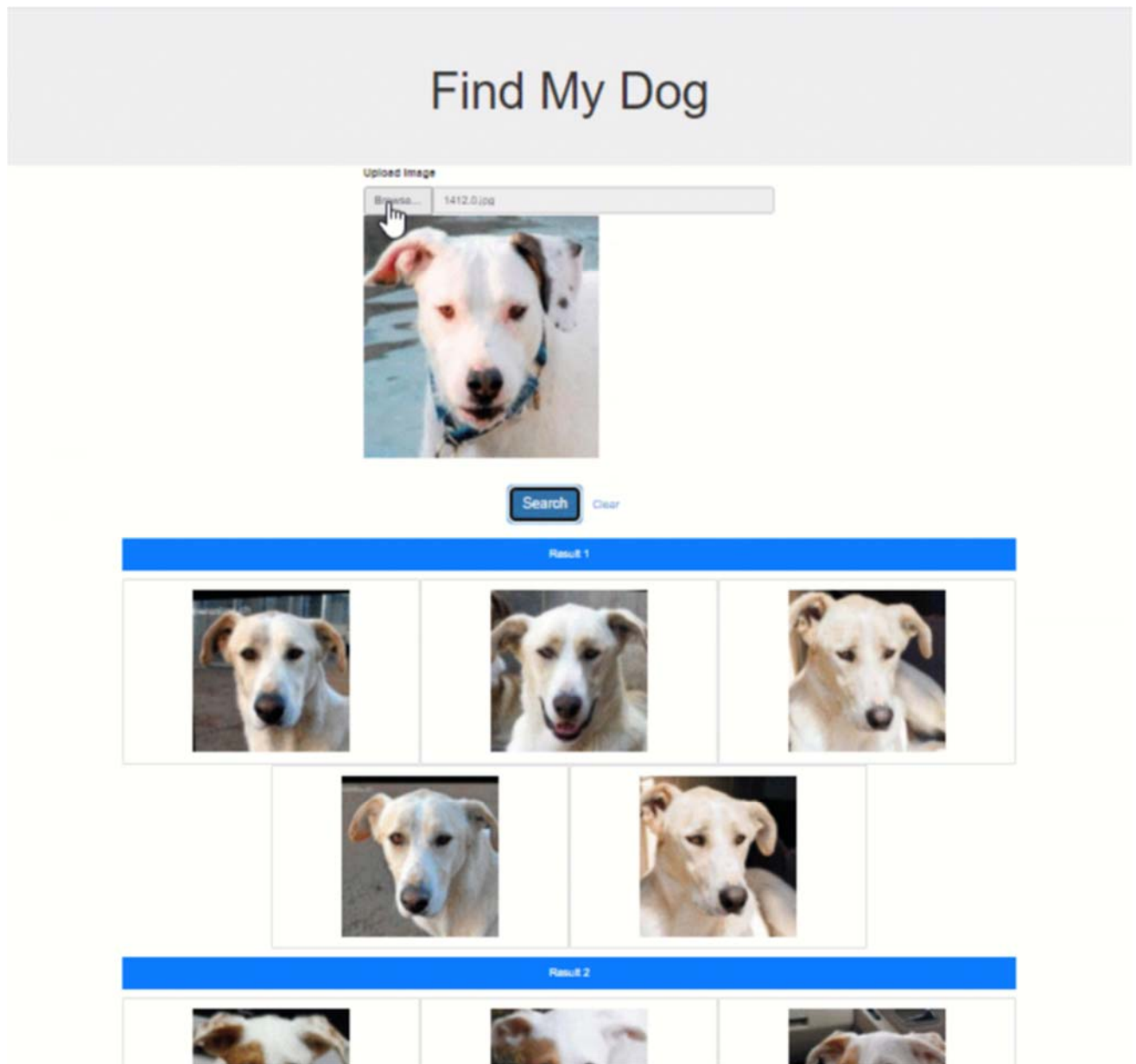


Figure 6: FaceNet Implementation on DogFaceNet - Some Results of his work[29]

2.5 DogFaceNet : Dog Identification [2]

The authors of the other implementation[2], used TensorFlow. This time, they have made this for academic reasons and haven't created a search engine with a simple workflow or a website to test some results. They have only tried out three algorithms for verification, identification and recognition on testing data. The tested data was organized in folders of different dogs and each folder had the same dog in a different place, lighting, pose etc.

The two contributions in our problem helped us analyze the most important configuration used in FaceNet, which is the Triplet Loss. We used many functions from the training of the above studies and we used TensorFlow as it was more understandable for us. In our scenario, we not only want to evaluate our network but to visualize actual results and create a website for lost dogs. We will be discussing the general idea of Convolutional Neural Networks and we will figure out the architecture we will create in our Network.

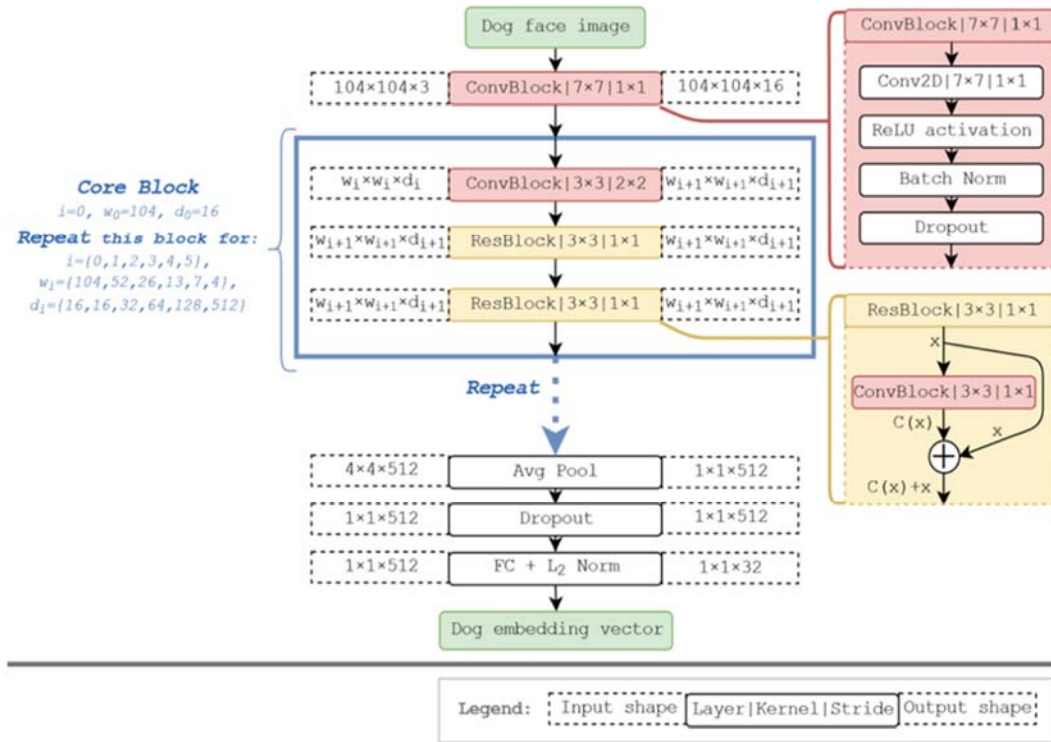


Figure 7: Architecture definition: The architecture takes as input a dog face image of size (224~224~3) and outputs a 32- dimensional embedding vector for the input image. The repeated block is sequentially repeated 5 times. Descriptions of ConvBlock and ResBlock are shown on the right side of the figure. [2]

Chapter 3

Data Acquisition

Contents

3.1 Data Collection	20
3.2 Image Preprocessing.....	22
3.2.1 Dog Points Detection	22
3.2.2 Stage 2 : Preprocessing	24

3.1 Data Collection

Machine learning needs two things to work and that is lots of data and models. When acquiring the data, we have to be sure to have enough features, aspect of data that can help for a prediction, populated to train correctly our learning model. In general, the more data we have the better! We had to collect as many pictures of different dogs as possible and for each dog we needed to get some more pictures individually. Most of the dog face dataset is collected in [2]. The dataset comes from the dog face dataset and it has 1393 classes of dogs, 8363 images and there are at least 2 images per dog. Every image is a .JPG of size 224x224x3. For this project the dataset was split into a training set and testing set. The training classes are automatically selected by the code but there could be difference in the selection depending on the OS we are using (we used Ubuntu).



Figure 8: An example of the dog face dataset

In order to prevent overfitting, we had to increase the size of the dataset. So, we have decided to acquire more data by visiting as many dog profiles we can on social media and gain the best possible images for each dog. We were very careful selecting all those images. It's important to chose images where the same dog is in some cases very similar and there are minor changes in their movement. In other cases we need to pick dog images where the same dog lays in a different position, it smiles, it yawns, it sits etc. Many of them did not make it to the dataset collection. We will discuss this part on the next subchapter where we processed differently the images we collected manually.



Figure 9: Selecting Dog Images through Pet Profiles on Social Media.

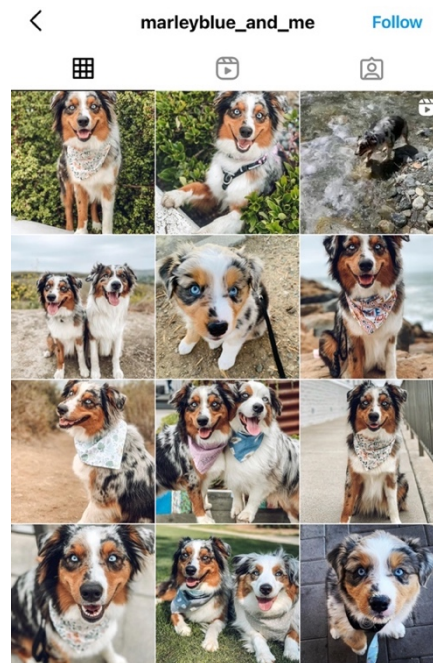


Figure 10: Visiting Pet Profiles through Social Media

3.2 Image Pre-processing

As we said, we went through many Pet Profiles to collect images for our dataset. Due to the reason that our Network is trained for dog faces, we need to detect dog heads and process the images to emphasize on them. We will also need to obtain the position of the dog's eyes and nose so in the following image processing we will align the dog in a decent position in the photo.

3.2.1 Dog Points Detection

To begin with, we will analyse the procedure to detect dog's heads, nose and eyes. In this program we will use .dat files that contain pre-trained models created in [30]. In those trained models dlib is used. It uses this deep learning tool to detect dogs and it saves this file into dogHeadDetector.dat. It also uses dlib shape predictor to identify the positions of the dog's eyes, nose, and top of the head. For every new photo we have collected, we will pass it through the head and landmarks detector. We save the positions of the left eye, right eye and nose into a .csv file to start processing the image.

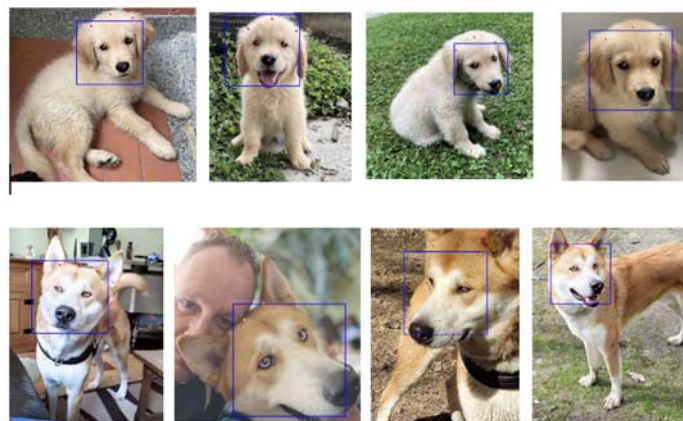


Figure 11: Dog's Landmark Detection

DogPoints.csv							
	filename	lex	ley	rex	rey	nox	noy
0	0.0.jpg	85	80	146	87	114	152
1	0.1.jpg	80	89	143	87	126	152
2	0.2.jpg	95	85	150	85	117	141
3	0.3.jpg	80	92	143	89	87	176
4	1.0.jpg	360	155	455	129	433	235
5	1.1.jpg	179	108	267	113	217	167

Figure 12: Some saved landmarks in excel file

In case our detector didn't catch any dog in the image it will return an empty array. Some images may be too difficult for our program to detect the dog. In some cases it may delays way to much to find the landmarks if the dog lays in a weird position and in some other cases it doesn't seem to find precisely the eyes or the nose as it appears in the following figures . Eventually we decided for those images to manually mark dog's landmarks.

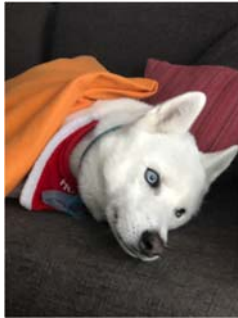


Figure 12: Example image that detector delays way too much to find Dog's Face

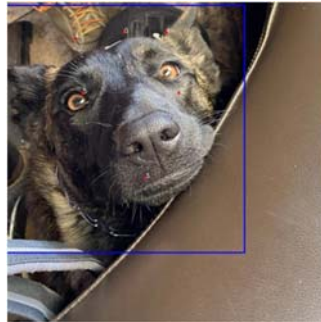


Figure 13: Example Image that detector finds wrong landmarks



Figure 13: Example image that detector can not Detect Dog's Face.

OpenCV is one of the most popular computer vision libraries. It will help us to control and manage different types of mouse events and give us the flexibility to manage them. We need to define the events for the mouse click, and let the user mark only 3 points. The user needs to mark in order the landmarks. First he needs to mark left eye then right eye and finally the dog's nose. After this event we will move on to repeat the whole process for all of our pictures. Finishing this, the three labels are either manually added on the images or our detector have found them: the left and right eye and noise.



Figure 14: Manual Marking of Nose and Eyes with OpenCV

3.2.2 Stage 2 : Pre-processing

Coming to the pre-processing phase, we will take the csv file that we have created before to modify our images. Firstly, we will read all the filenames of the data with their landmarks we have collected.



	filename	lex	ley	rex	rey	nox	noy
0	rex.jpg	756	1087	979	1004	1020	1290

Figure 15: Dog Image Example with its landmarks

After, the program will load the photos and mark the eyes and nose based on their positions in the csv file [Figure 15].



Figure 16: Dog Face Alignment

Dog faces are then aligned using the position of the eyes. Face alignment creates regularities in images and facilitates dog face parts automatic detection. Based on [2]'s alignment, the right and left eye of the dog is placed in position $(0.7/2.4 \times \text{New height}, 0.7/2.4 \times \text{New Width})$. With this metrics, the new aligned images appear to be a good calibration for the picture. The pictures are finally re-sized to $(\text{new height}, \text{new width}, \text{depth}) = (104 \times 104 \times 3)$ pixels. The above Figure [16] represents the example of the dog we gave before to align it properly .

Chapter 4

Methodology

Contents

4.1 Overview	26
4.2 History.....	27
4.2.1 Neural Networks	27
4.2.2 Deep Convolutional Neural Networks.....	28
4.3 Network Architecture.....	29
4.4 Open Source Software for Machine Learning	29
4.5 Model Definition.....	30
4.5.1 The process of Model Definition	30
4.5.2 The Layers of our Network.....	31
4.5.2.1 Convolutional Layers and Activation Function.....	31
4.5.2.2 Pooling Layers	33
4.5.3 Adding more layers to our network	34
4.5.3.1 Applying many Convolutional Networks	34
4.5.3.2 Global Average Pooling.....	36
4.5.3.3 Flatten Layer	36
4.5.3.4 Dropout Layer.....	36
4.5.3.5 Dense Layer	37
4.5.4 Batches	38
4.5.4.1 Define the Batches	38
4.5.4.2 Batches Normalization.....	39
4.6 Training	40
4.6.1 Organization of the Data	40
4.6.1.1 Splitting the Dataset.....	40
4.6.1.2 Npy files:.....	41
4.6.2 Optimizer Algorithm.....	41
4.6.2.1 Overview.....	41
4.6.2.2 Learning Rate.....	42
4.6.2.3 SGD – Adam Optimizer to minimize the error.....	42

4.6.3 Loss Function.....	43
4.6.3.1 Triplet Loss	43
4.6.3.2 The margin	45
4.6.3.3 Metrics Triplet and Accuracy: Pairwise Ranking Loss	46
4.6.4 Triplet Sampling	47
4.6.4.1 The problem of Overfitting.....	47
4.6.4.2 Hard Triplet Mining Strategy: Hard sampling.....	48
4.6.5 Adaptive Hard Image Generator	49
4.7 Testing.....	50
4.7.2 Introduction.....	50
4.7.3 Useful Functions	50
4.7.3.1 Finding the Best Threshold.....	50
4.7.3.2 Create Embeddings	51
4.7.3.3 Find Distance	51
4.7.4 Algorithms	52
4.7.4.1 Brute Force Nearest Images.....	52
4.7.4.2 Agglomerative Clustering.....	52
4.7.4.3 Hybrid Solution: Cluster the Nearest Images	54

4.1 Overview

Firstly, we will talk about the general idea of our network and afterwards we will analyze the process of building our model. Our network has as input dog images that will be processed before entering the training process. The input will go through a convolutional neural

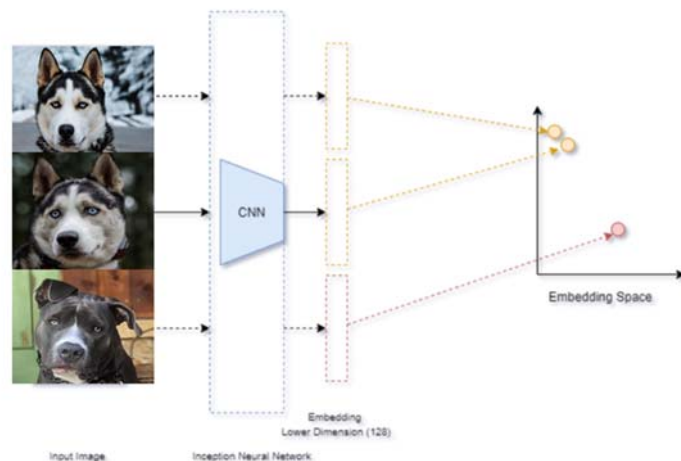


Figure 17: Passing inputs images into convolutional neural networks. Our output is the predicted vectors placed in an Embedding Space

network composed of numerous layers. It is extremely crucial to see how our neural network architecture will be built. Eventually our final output, provides us information on the correlations of the images, according to the embedding space created by the network.

We can think Embedding Space as a multi-dimensional graph that we can position elements such as photos that contain many characteristics. The elements that have significant similarities are close in this space we call to clarify their components. Our network outputs numerous data for each picture and deep learning leverage various ranking losses to learn an object embedding — an embedding where objects from the same class are closer than objects from different classes.

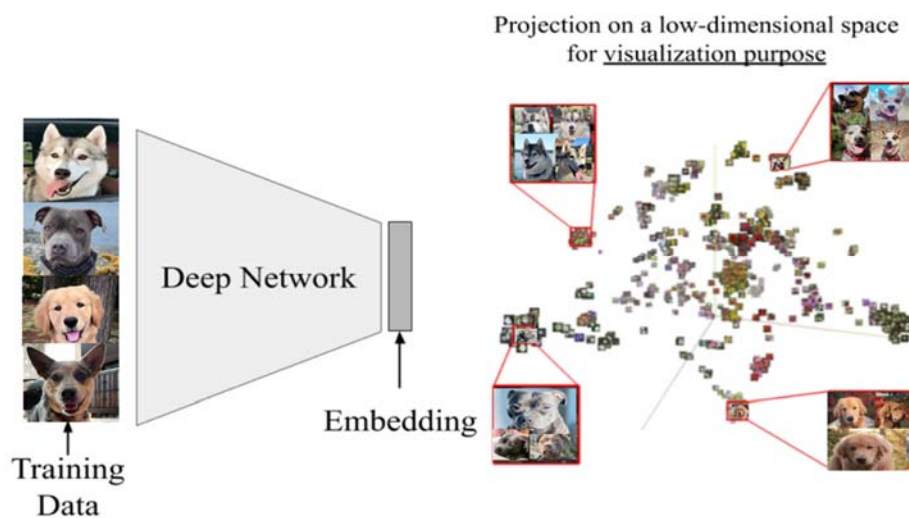


Figure 18: Images mapping through DCNN to embedding space

4.2 History

4.2.1 Neural Networks

Neural Network is an expressive machine learning architecture. It is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons. Basically, the building block of a neural Network is a neuron, specifically a functional unit that takes a signal input that goes into a node and it does some kind of a mathematical operation to give us an output.

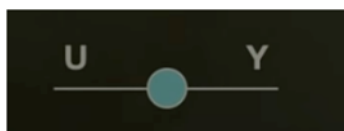


Figure 21: a neuron

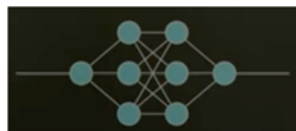


Figure 21: Artificial Neural Network

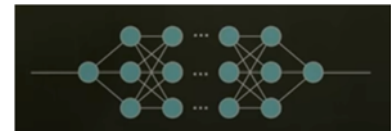


Figure 21: Deep Neural Network

The several computations that a node is structured of, includes input from the data with a set of coefficients, or weights, that either amplify or dampen our input. In this way we are assigning a significance to inputs with regard to the task the algorithm is trying to learn. In our case we are dealing with biometrics of dogs.

4.2.2 Deep Convolutional Neural Networks

We take this unit and start to stack it either in series or in parallel or both so we can do a more complicated function. Building up this complexity we have built what we called an Artificial Neural Network. Each layer is doing some kind of a sequential processing. Stacking several layers that each one of them is doing some kind of a sequential processing is what we call a Deep Neural Network. We will add many hidden layers to our network. They will allow us to complex data thanks to their nodes/neurons. They are “hidden” because the true values of their nodes are unknown in the training dataset. In fact, we only know the input and output. Because of how many hidden layers our network has, is called a deep neural network. The most significant and final type of hidden layer in our network, is the fully-connected layer where each neuron will be connected to all the others in two adjacent layers. It is not connected to the ones in the same layer. The convolutional layers is another type of hidden layers that is very prominent when dealing with dog images. We will use this layer multiple times to extract dog characteristics. Therefore, the first few layers of the network may detect simple features like lines, circles, edges. In each layer, the network is able to combine these findings and continually learn more complex concepts as we go deeper and deeper into the layers of the Neural Network.

Each connection between two nodes has an associated weight, which is just a number. Each weight represents the strength of the connection between the two nodes. Every time the network receives an input at a given node in the input layer, this input is passed to the next node via a connection, and the input will be multiplied by the weight assigned to that connection. These input-weight products are summed : Σ . This sum Σ is then passed through a node’s so-called activation function, which performs some type of transformation on the given sum. For example, an activation function may transform the sum to be a number between zero and one. This function is used to determine whether and to what extent that signal should progress further through the network to affect the ultimate outcome, say, an act of classification. If the signals passes through, the neuron has been “activated.” The actual transformation will vary depending on which activation function is used. The role of the activation function is to buffer the data before it is fed to the next layer. Here’s a diagram of what one node might look like.

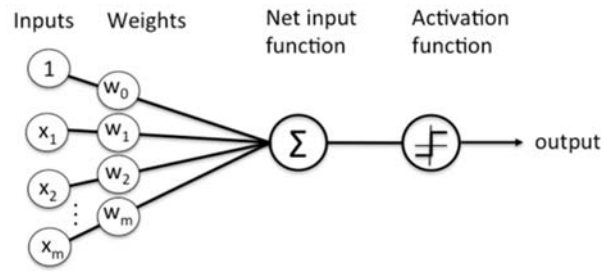


Figure 22: : Visualization of the Network[37]

There is a massive variety of Neural Network Architectures we can design. There are different types of nodes and computations that can be performed. There is also a different topology of whether or not information is getting compressed in a bottleneck or expanded. CNN is used in image recognition likewise in our case. What CNN does, is that it has these convolutional layers that basically take a mask and slide it across the image doing local computations in local patches. We are able to pull out edges or features and we can run that through a convolutional layer and keep doing this process through another convolutional layer, stacking all those layers. Anywhere there is a translation invariant we can use it. For computer vision, this means that regardless of where an object is moved in an image (translation), it doesn't change what that object is (invariance). A dog picture may represent the dog in a different position in the image (top, bottom, top right, etc.) CNN can start to reveal these translations that exist in images.

4.3 Network Architecture

We are going to study the triplet-based network architecture proposed for the ranking loss function[35]. The triplet loss based network architecture has been introduced by the FaceNet [1] paper for face recognition. They describe a new approach to train face embeddings using online triplet mining. Usually in supervised learning we have a fixed number of classes and train the network using the SoftMax cross entropy loss. Having two unknown faces we wouldn't be able to compare them without the triplet loss contribution.

Triplet loss in this case is a way to learn good embeddings for each face. In the embedding space, faces from the same person should be close together and form well separated clusters.

4.4 Open Source Software for Machine Learning

Designing an architecture is becoming easier because of the explosion of open source software of the giants of the Industrial Investment technology companies such as Google and Facebook. There are many incredibly powerful environments like TensorFlow, Keras and

PyTorch where we can design NN architecture and train our data to build an expressive model. In this research we study the architecture of a model that uses the open source software of TensorFlow and Keras, that will be able to help us in this procedure.

4.5 Model Definition

4.5.1 The process of Model Definition

As we mentioned earlier, in neural networks, convolutional neural network is one of the main categories to do images recognition and classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used. CNN image classifications takes an input image, process it and classify it under certain categories (E.g.: Dog, Cat, etc.). A DCNN uses a three-dimensional neural network to process the Red, Green, and Blue elements of the image at the same time. Computers basically see an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see $h \times w \times d$ (h = Height, w = Width, d = Dimension). E.g., An image of $6 \times 6 \times 3$ array of matrix of RGB (3 refers to RGB values). This considerably reduces the number of artificial neurons required to process an image, compared to traditional feed forward neural networks. The architecture of a convolutional network typically consists of four types of layers: convolution, pooling, activation, and fully connected.

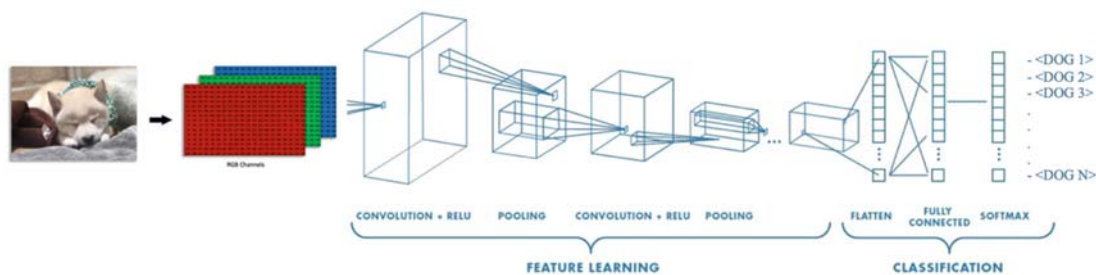


Figure 23 : Each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The above figure is a complete flow of CNN to process an input image and classifies the objects based on values. [38]

Now that we have gone through the idea of the convolutional neural network, we can continue to build our CNN. To build the CNN, we'll use a Keras Sequential model. The network takes an image x of size $(104 \sim 104 \sim 3)$ as input and outputs an embedding vector $f(x)$ of size 32. On the first layer, we specify this input shape, which is the shape of our data. Our images are 104 pixels high and 104 pixels wide and have 3 color channels: RGB. This gives us an input_shape of (104, 104, 3).

4.5.2 The Layers of our Network

4.5.2.1 Convolutional Layers and Activation Function

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel. Based on that, the first layer in the model is a 2-dimensional convolutional layer. This layer will have 16 output filters each with a kernel size of 7x7. We enable zero-padding by specifying padding = 'same'. The activation function we are using is of the most widely used activation functions today called Relu . Relu, which is short for rectified linear unit, transforms the input to the maximum of either 0 or the input itself.

```
// pseudocode
function relu(x) {
  if (x <= 0) {
    return 0;
  } else {
    return x;
  }
}
```

$$\text{relu}(x) = \max(0, x)$$

Figure 24: Relu Function

In simple words, if the input is less than or equal to 0, relu will output 0. If the input is greater than 0, relu will then just output the given input. The idea behind why we are using this activation function is based on the more positive the neuron is, the more activated it is. We specify an activation function in a Keras Sequential model. To achieve this, first we import our classes and then specify an activation function in the constructor of the layer. On-account-of our input images, that are greater than 128×128, we need choose to use a kernel size greater than 3 to help : (1) learn larger spatial filters and (2) to help reduce volume size. So firstly, we need to use 7×7 kernel to learn larger features and then quickly reduce spatial dimensions and start working with 3×3 kernels.

Mandatory Conv2D parameter is the numbers of filters that convolutional layers will learn from. 32 is the number of output filters in the convolution. After this filter has convolved the entire input, we'll be left with a new representation of our input, which is now stored in the output channel. This output channel is called a “feature map”. Each convolutional layer holds a stack of feature maps that build on one another. At the end of the case, the model puts all of these features together. By finishing defining the layers of our network, we can see that each convolutional layer of our network has a set of feature maps that can recognize increasingly complex patterns/shapes in a hierarchal manner like below. The CNN uses pattern recognition of numbers to figure out the most important features of the dog image. As it stacks these patterns on top of each other with more layers, it can build very complex feature maps.

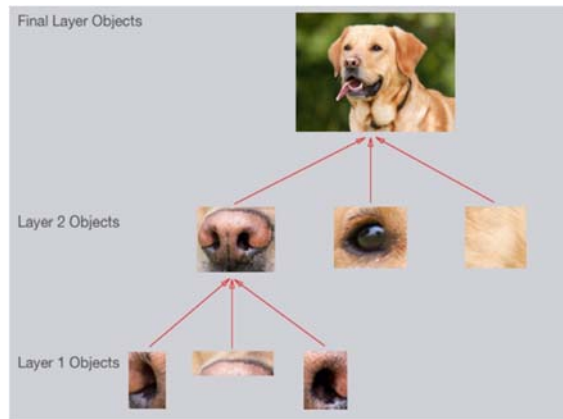


Figure 25: Extraction of features

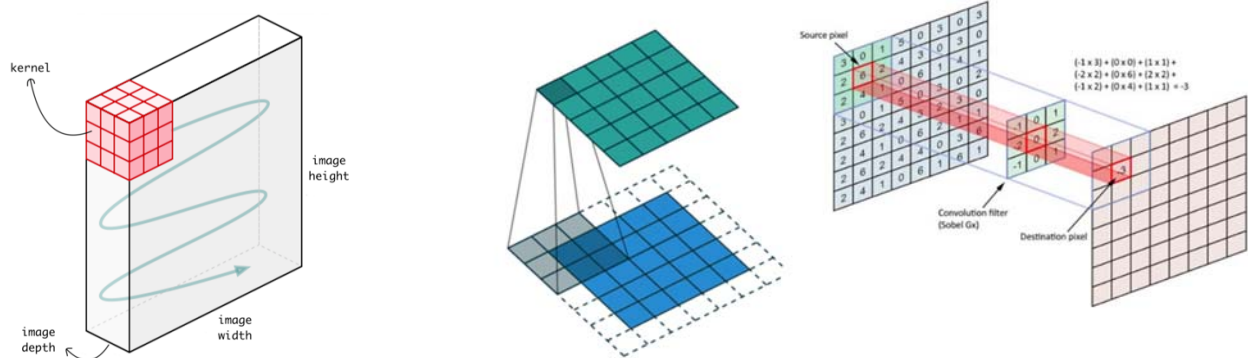


Figure 26: Keras Conv2D: The filter slides over the input and performs its output on the new layer [39].



Figure 27: Some of the convolved images using many different kernels: emboss, sharpen, edge , blur etc.

With CNNs, we look at groups of pixels next to one another which allows the model to learn local patterns like shapes, lines, etc. For example if the CNN saw lots of white pixels around a black circle, it would recognize this pattern as an eye. To get CNNs to accomplish translation variance, we rely on the services of its' feature learning algorithm.

Moreover, we define the stride, which determines how many pixels we want our filter to move as it slides across the image. Stride is the number of pixels shifts over the input matrix. In our case, we move the filters to 2 pixels at a time. The below figure shows convolution would work with a stride of 2.

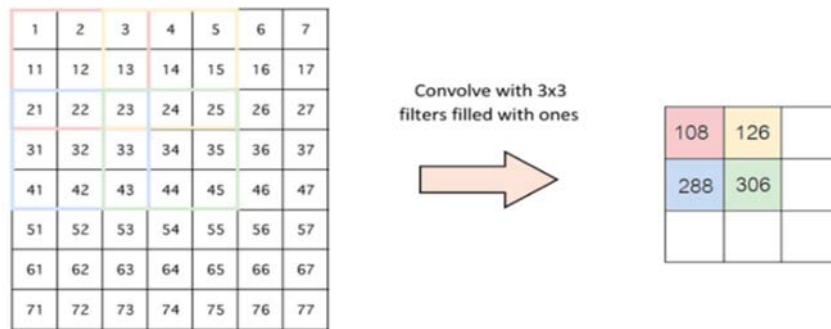


Figure 28:: Example of a Conv2D layer [38]

4.5.2 .2 Pooling Layers

Stepping in the next layer, we need to add Max Pooling to reduce the spatial dimensions of the output volume. By adding this to layer, we reduce the dimensionality of the images by reducing the number of pixels in the output from the previous convolutional layer. Further do, it removes small values by taking the maximum value from a square set of pixels.) As far as choosing the appropriate value for no. of filters, it is always recommended to use powers of 2 as the values. For each block, or “pool”, the operation simply involves computing the *max* value, like is showing in the following figure. Doing so for each pool, we get a nicely down sampled outcome, greatly benefiting the spatial hierarchy we need:

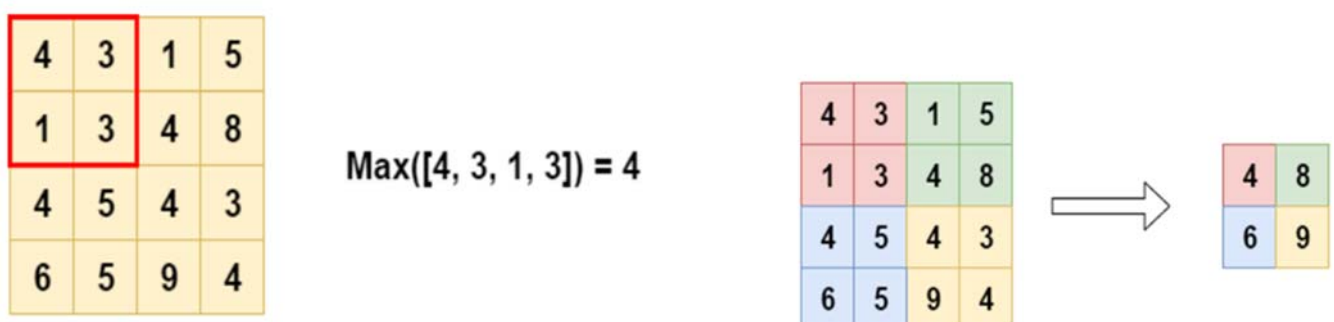


Figure 29: Application of Max Pooling with stride=2 [38]

4.5.3 Adding more layers to our network

4.5.3.1 Applying many Convolutional Networks

Moving forward we start to repeat this process by applying many convolutional layers with different filter sizes each time.

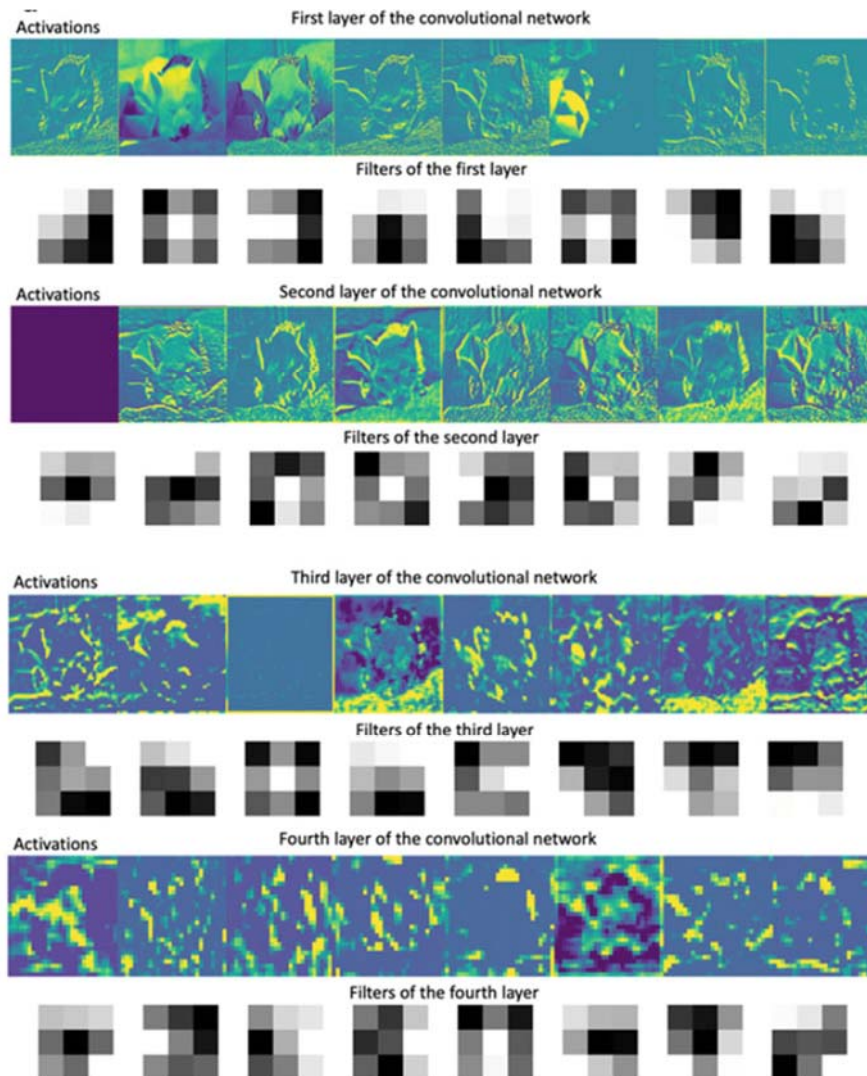


Figure 30: Adding various convolutional layers example

We use filters that can detect edges. In the output channels, the brightest pixels can be interpreted as what the filter has detected. Using the filter in [Figure 31], we can detect top horizontal edges [Figure 32] of the dog, and that's indicated by the brightest pixels (white).

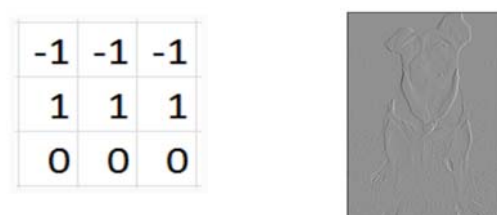


Figure 31: Example of kernel and its output result

The second filter in figure [32] detects left vertical edges, again being displayed with the brightest pixels. The third detects bottom horizontal edges, and the fourth detects right vertical edges.

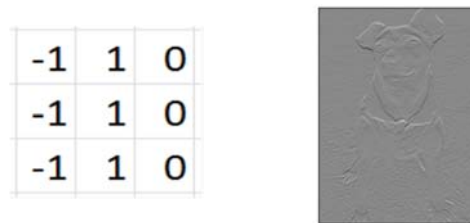


Figure 32: Example 2 of kernel and its output result.

These filters, as we mentioned before, are really basic and just detect edges. These are filters we may see towards the start of a convolutional neural network. More complex filters would be located deeper in the network and would gradually be able to detect more sophisticated patterns like the ones shown here:

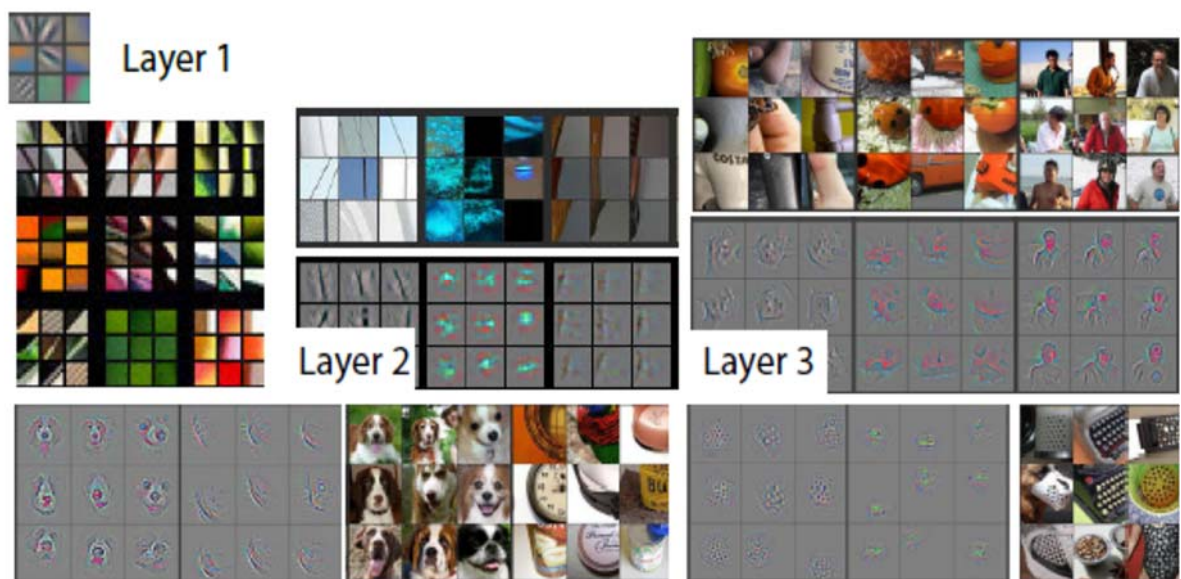


Figure 33: We can see the shapes that the filters on the left detected from the images on the right. We can see circles, curves and corners. As we go further into our layers, the filters are able to detect much more complex patterns like dog faces etc.

The amazing thing is that the pattern detectors are derived automatically by the network. The filter values start out with random values, and the values change as the network learns during training. The pattern detecting capability of the filters emerges automatically.

4.5.3.2 Global Average Pooling

Moreover, we add the Global Average Pooling that downsamples the input along its spatial dimensions (height and width) by taking the average value over an input window that is our previous layer, for each channel of the input. The window is shifted by strides along each dimension.

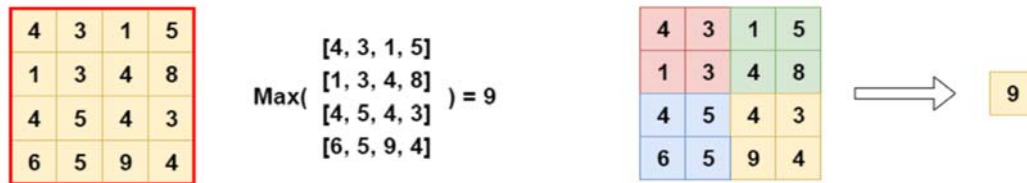


Figure 34: Application of Global Average Pooling with stride = 2

4.5.3.3 Flatten Layer

Continuing to the final layers, we perform a Flatten Layer to convert the data into 1D arrays to create a single feature vector.

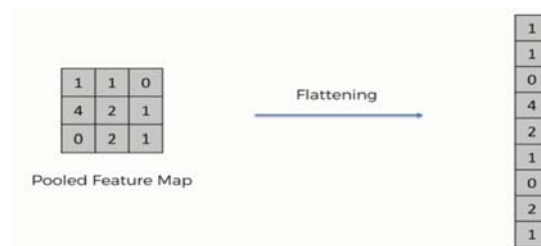


Figure 35: Flatten Layer

4.5.3.4 Dropout Layer

Thereafter, we apply Dropout that consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure 36. The choice of which units to drop is random.

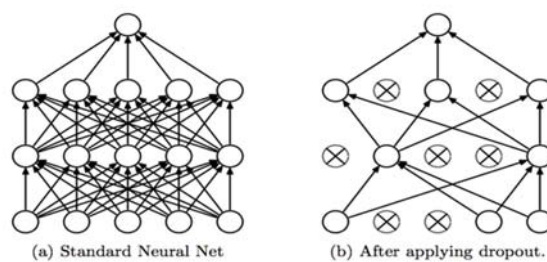


Figure 36: Dropout Neural Net Model. On the Left side, we can see a Standard Neural Network with 2 hidden layers. On the Right side, is the thinned version of the network of our example, produced by applying dropout to the left side. Crossed units have been dropped. [36]

4.5.3.5 Dense Layer

Eventually by adding dense layer, we create a fully connected layer. Each node in this layer is connected to the previous layer. A densely connected layer provides learning features from all the combinations of the features of the previous layer, whereas a convolutional layer relies on consistent features with a small repetitive field. This layer is used at the final stage of CNN to perform classification for our problem. Dense layer does the below operation on the input and returns the output. A densely connected layer provides learning features from all the combinations of the features of the previous layer, whereas a convolutional layer relies on consistent features with a small repetitive field. In the below figure we can clearly see that each Dense Layer receives input from all neurons of previous layers.

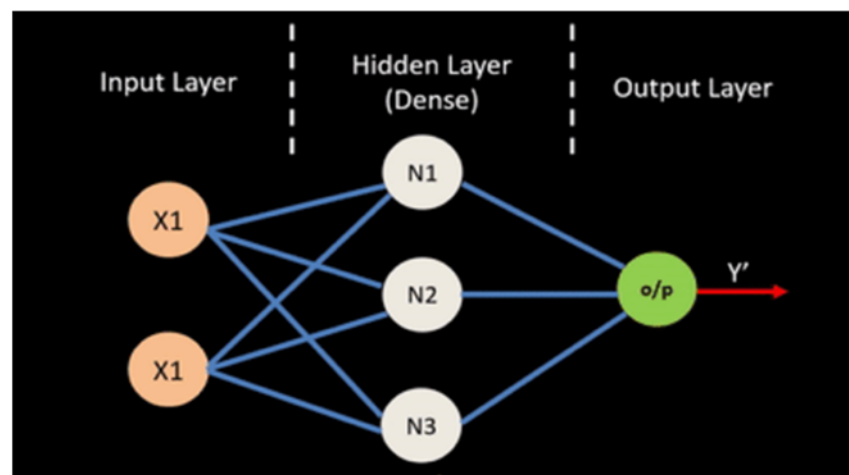


Figure 37: A simple Neural Network with only 1 hidden layer that is fully connected.

We can also see another example in the following figure, that represents a more complex network with more hidden layers, as it happens in our network.

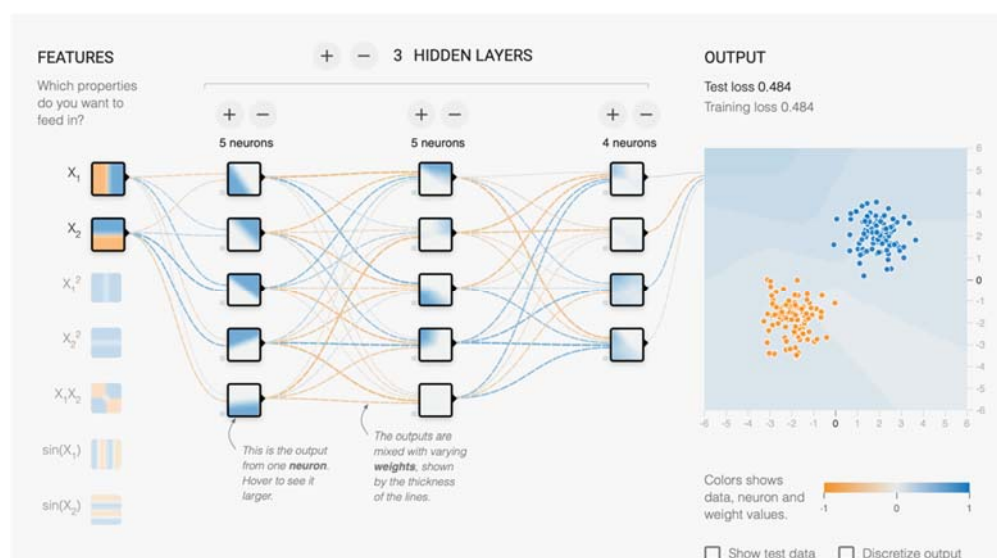


Figure 38: A more complex Neural Network with only 3 hidden layers that are fully connected.

Moving to the final step we need to group layers into an object with training and inference features. In our case, we are going to use "Functional API", where we start from an Input, and chain layer calls to specify the model's forward pass, and finally we create our model from inputs and outputs: Output data will be the result of all these layer transformed though the final layer called Lambda. Lambda is used to transform the input data using an expression or function. In our scenario, Lambda comes with the expression `lambda x: tf.nn.l2_normalize(x, axis=-1)` that is applied to as a final layer. Its input data will be normalized along dimension axis using an L2 norm. This layer will coerce its inputs into a distribution centered around 0 with standard deviation 1. It accomplishes this by precomputing the mean and variance of the data.

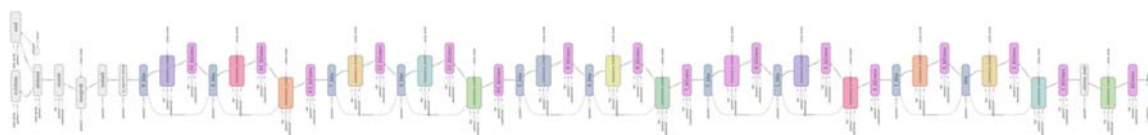


Figure 39: The architecture of our Network.

4.5.4 Batches

4.5.4.1 Define the Batches

To move forward, is important to specify our batches. The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. We can think of a batch as a for-loop iterating over one or more samples and making predictions. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent like we do in our network. In our case, in every step of the training, we set our network to constantly take 30 images that have dimensions 224x224x3. The reason behind setting batch size to 30 is due to the loss that we will use in our training phase. We will refer to triplet loss in the following subchapter but is important to declare that we will be working with 3 images every time (anchor, positive and negative), so batch size is 30 so we can work with 10 triplets of images every time (10x3). So declaring that, the number of batches in each epoch equals to the training set size divided by the batch_size. Generally, the larger the batch size, the quicker our model will complete each epoch during training, due to the fact that our computational recourses will be able to process much more than one single sample at a time. The trade-off, however, is that even if our machine can handle very large batches, the quality of the model may degrade as we

set our batch larger and may ultimately cause the model to be unable to generalize well on data it hasn't seen before.

4.5.4.2 Batches Normalization

Proceeding to the next step we use a technique called Batch Normalization. Keras also provides support for batch normalization via the BatchNormalization layer. We will need to mitigate the effect of unstable gradients within deep neural networks. BN introduces an additional layer to the neural network that performs operations on the inputs from the previous layer. Typically in machine learning, it is common to normalize input data before passing the data to the input layer. We need to normalize to ensure that our model can generalize appropriately. This is achieved by ensuring that the scale of the values is balanced, and also the range of the values are maintained and proportional despite the scale change in the values. The operation standardizes and normalizes the input values. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. The Batch Normalization layer can be used to standardize inputs before or after (in our case), the activation function of the previous layer.

Step	Expression	Description
1	$z = \frac{x - mean}{std}$	Normalize output x from activation function.
2	$z * g$	Multiply normalized output z by arbitrary parameter g .
3	$(z * g) + b$	Add arbitrary parameter b to resulting product $(z * g)$.

Figure 40: Batch Normalization Process

From the above Figure [40] we can come to conclusion that the means of all dimensions are zero and the variances are all 1. Below there are two equations that define the calculations of standard deviation (σ) and mean (μ). They depend on the value of z^i :

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (z^i - \mu)^2}$$

$$\mu = \frac{1}{N} \sum_{i=1}^N z^i$$

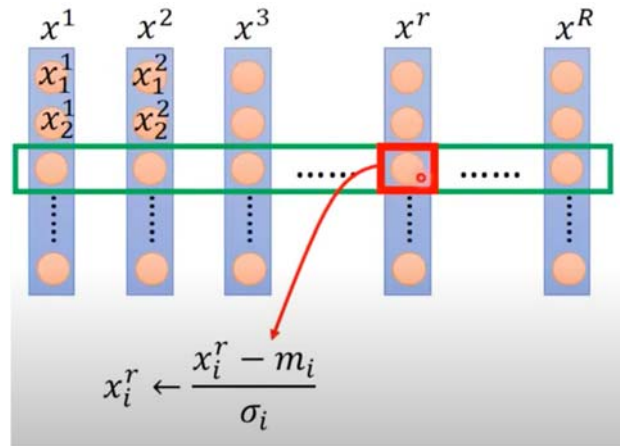


Figure 41: For each dimension it is calculated: mean = m_i and standard deviation = σ_i

When applying batch norm to a layer, the first thing batch norm does is normalize the output from the activation function. Recall from our previous review on our activation functions Relu, that the output from a layer is passed to an activation function, which transforms the output in some way depending on the function itself, before being passed to the next layer as input. After normalizing the output from the activation function, batch norm multiplies this normalized output by some arbitrary parameter and then adds another arbitrary parameter to this resulting product.

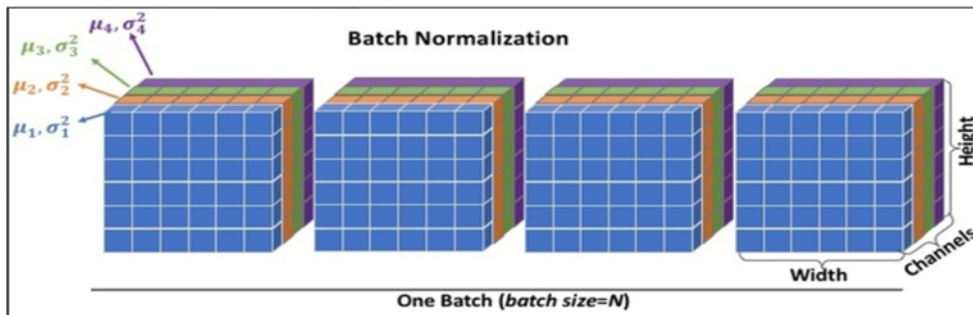


Figure 42: Batch Normalization

4.6 Training

4.6.1 Organization of the Data

4.6.1.1 Splitting the Dataset

Moving on to the next phase to complete our network, we will explore the process of training. Before we dive in to the training phase, we need to split our dataset. We are going to have an open-set dataset, specifically is a bunch of unknown dog pictures that the network can only see for the first time during the testing stage. The open-set problem is a harder problem to solve and closer to a real life problem [2]. We now need to organize the directory structure

on disk to hold the data set. We'll manually do some parts of the organization, and programmatically do the rest. Our code splits train and test pictures that will be used in each procedure. Data.py[2] saves the filenames of testing and training datasets separately in an output folder. We may only run this code only once to save this kind of information onto archives in our network. Its important to split the datasets so we can actually test that our model can do its job. When we run data.py functions in our main code we can see the demarcation between the files we will be using. We have the filenames of testing and training apart and also the labels for them.

4.6.1.2 Npy files:

Anyone who has ever done any kind of data processing in Python has undoubtedly come across Numpy and Pandas. These are the giants of Data Science in Python and stand as the foundation for a lot of other packages, namely Numpy provides the fundamental objects used by the likes of Scikit-Learn and Tensorflow. So why are we referring about these packages and why Numpy in particular? It is well known that the “industry standard” with regard to data-files is .csv files. Now while convenient, these files are highly un-optimized when compared to the alternatives, like the .npy files provides as courtesy of Numpy. Explaining why we use these type of files, we can point out why do we need these files in our program. In our packages we involve many pictures that will be used in training and testing. In filenames.py we obviously save the names of the directories and the photos specifically we are using in each case. For example directory named “Cookie” has a few photos of the same dog named Cookie. This dog will be used in training process and its filenames will be saved in filenames-train.npy. In labels we point out that the photos we are using for the same dog belong only to that specific dog. For example Cookie-Photo1 and Cookie-Photo2 belong to Cookie. In this way, we have splitted our dataset in training and testing and made it clear which files are used where. Our model, is a deep convolutional neural network. It is trained via a triplet loss function that encourages vectors for the same identity to become more similar (smaller distance), whereas vectors for different identities are expected to become less similar (larger distance). The focus on training a model to create embeddings directly (rather than extracting them from an intermediate layer of a model) was an important innovation in this work.

4.6.2 Optimizer Algorithm

4.6.2.1 Overview

For every iteration, an epoch is executed. we showed how each connection between nodes has an arbitrary weight assigned to it. We will configure our model with compile method

of Keras. One of the most important parameters configuring our model is the optimization algorithm in order to succeed that. During training, our weights will be iteratively updated and moved towards their optimal values.

4.6.2.2 Learning Rate

The learning rate is a parameter in our network, that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process. Learning rate may be the most important parameter when configuring our neural network. Therefore it is vital to know how to investigate the effects of the learning rate on model performance and to build an intuition about the dynamics of the learning rate on model behavior.

4.6.2.3 SGD – Adam Optimizer to minimize the error

The most widely known optimizers are called *stochastic gradient descent*, or more simply, SGD. The objective of SGD is to minimize some given function that we call a *loss function*. So, SGD updates the model's weights in such a way as to make this loss function as close to its minimum value as possible. Specifically we have encompassed ‘Adam’ Optimizer, an SGD method that is based on adaptive estimation of first-order (mean) and second-order moments (variance). β_1 and β_2 are the decay rates, that control the relative contribution of past history versus the present gradient.

$$\begin{aligned}m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2\end{aligned}$$

Figure 43: Mean and Variance

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allow us to reduce error rates and to make the model reliable by increasing its generalization. It is a standard method of training artificial neural networks and it helps to calculate the gradient of a loss function with respects to all the weights in the network.

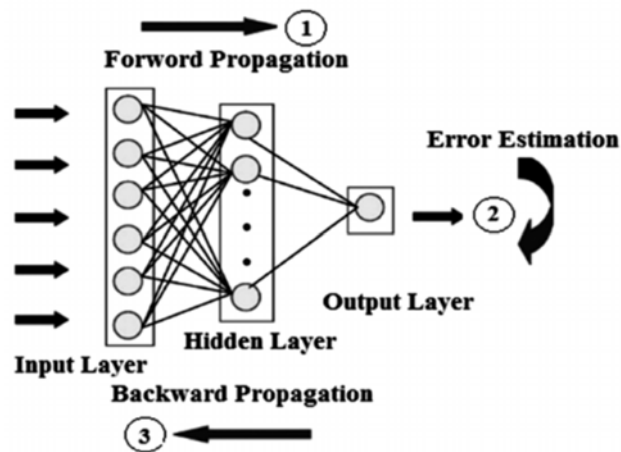


Figure 44: As shown in above diagram, backward propagation is exactly the opposite side of forward propagation. You can view the forward propagation as “forecasting output based on inputs and the ANN”. Backward propagation is to feedback the error and adjust the ANN to make it more accurate.

4.6.3 Loss Function

4.6.3.1 Triplet Loss

Another significant parameter that will help our optimizer, to minimize the error and update the weights, is our loss function (Appendix A). A way to measure whether the algorithm is doing a good job, it is necessary to determine the distance between the algorithm’s current output and its expected output. The measurement is used as a feedback signal to adjust the way the algorithm works. The loss function is the function that computes the distance between the current output of the algorithm and the expected output. It’s a method to evaluate how our algorithm models the data. For our purposes we will primarily focus in Triplet Loss function. Generally, the objective of the model is to generate embeddings that will position same dogs in a close distance to each other in the embedding space and on the other hand, to position different dogs in a larger distance. The triplet loss function will contribute in this idea. A triplet basically contains an anchor, a positive, and a negative image. The positive image is more similar to the anchor image than the negative image (Figure 45 for an illustration).

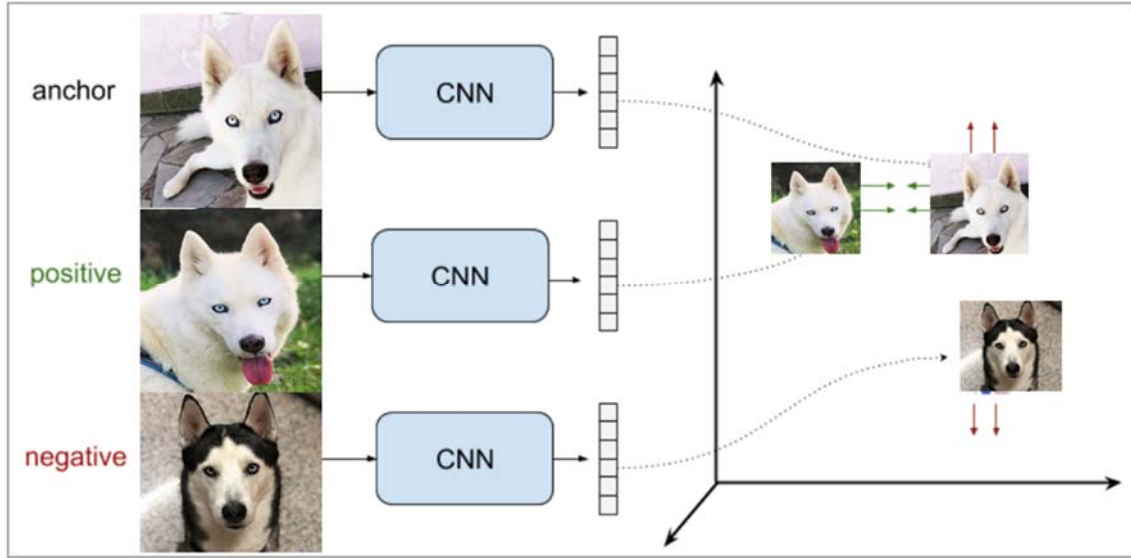


Figure 45: Example of a triplet ranking loss setup to train a network for image face verification. In this setup, the weights of the CNNs are shared. We call it triple nets.

According to the definition of triplets, it is a fact that the distance between the anchor sample and the negative sample representations $d(r_a, r_n)$ is greater (and bigger than a margin) than the distance between the anchor and positive representations $d(r_a, r_p)$. Below we can see, the formula of Triplet loss. We call $T_i = (p_i, p_{+i}, p_{-i})$ a triplet, where p_i, p_{+i}, p_{-i} are the anchor image, positive image, and negative image, respectively. α is a margin that is enforced between positive and negative pairs.

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2, \quad \forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in T$$

Using triplet model selection, our network learns feature embedding by optimizing the relative distance between the samples from the same classes and dissimilar classes.

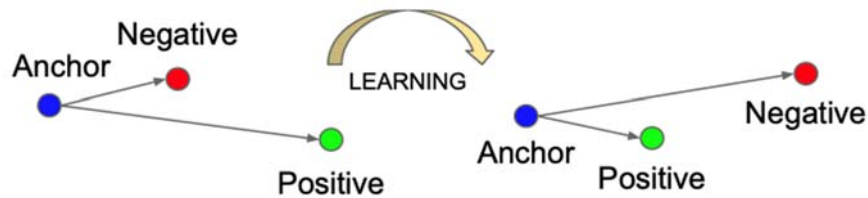


Figure 46: Minimization of the distance between the positive and anchor. Maximization of the distance between negative and the anchor.

Now we will use the triplet loss function over a contrastive loss. This loss minimizes and maximizes the Euclidean distance between similar and different points, respectively. Similar and different points are grouped into positive and negative pairs. The next figure shows its formulation using a pair of points' embeddings.

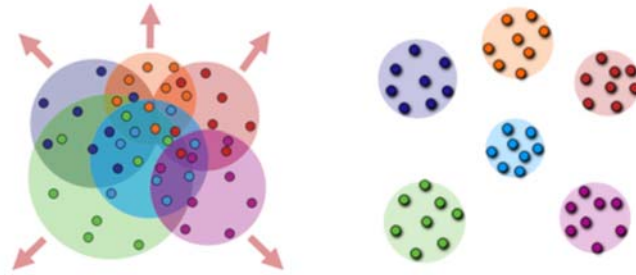


Figure 47: Example of Triplet Loss learning process

Triplet loss pulls the anchor and positive together while pushing the anchor and negative away from each other.



Figure 48: Triplet loss example 2

$$L = \sum_i^{|N|} [\|f(p_i) - f(p_i^+)\|_2 - \|f(p_i) - f(p_i^-)\|_2 + \alpha]$$

Figure 49: Final Triplet Loss Formula

4.6.3.2 The margin

Similar to the contrastive loss, the triplet loss leverages a margin m . Triplet loss is generally superior to the contrastive loss in retrieval applications like Face recognition, Person re-identification, and feature embedding. In our scenario we are dealing with dog face recognition which is very similar to the other applications that are using this loss. In the following formula ' α ' is the parameter representing the margin, which is enforcement between positive and negative pairs. Intuitively, for each dog, triplet loss expects a margin of α between all the combinations of its positive and negative images of dogs. With N represent the set of all possible triplets, the loss function in Metric Layer that is being minimized is shown in the Figure below. [Figure 50].

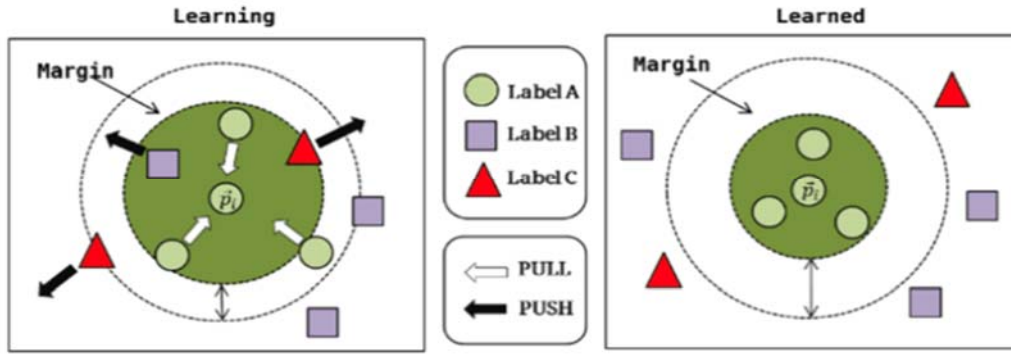


Figure 50: Triplet Loss learning process

4.6.3.3 Metrics Triplet and Accuracy: Pairwise Ranking Loss

Metric values are displayed during `fit()` and logged to the History object returned by `fit()`. They are also returned by `model.evaluate()`. We will set as a metric, the triplet accuracy. As mentioned earlier, in this setup positive and negative pairs of training data points are used. The purpose is to learn representations with a small distance between the for positive pairs, and greater distance than some margin value ' α ' for negative pairs. Pairwise Ranking Loss forces representations to have zero distance for positive pairs, and a distance greater than a margin for negative pairs. Being r_a , r_p and r_n the samples representations and d a distance function, we can write that triplet accuracy will be:

$$L = \begin{cases} d(r_a, r_p) & \text{if } \text{PositivePair} \\ \max(0, m - d(r_a, r_n)) & \text{if } \text{NegativePair} \end{cases}$$

Figure 51: Loss for Negative and Positive Pairs.

For negative pairs, the loss will be zero when the distance between the representations of the two pair elements is greater than the margin m . But when that distance is not bigger than m , the loss will be positive, and net parameters will be updated to produce more distant representation for those two elements. The loss value will be at most m , when the distance between r_a and r_n is zero. The function of the margin is that, when the representations produced for a negative pair are distant enough, no efforts are wasted on enlarging that distance, so further training can focus on more difficult pairs.

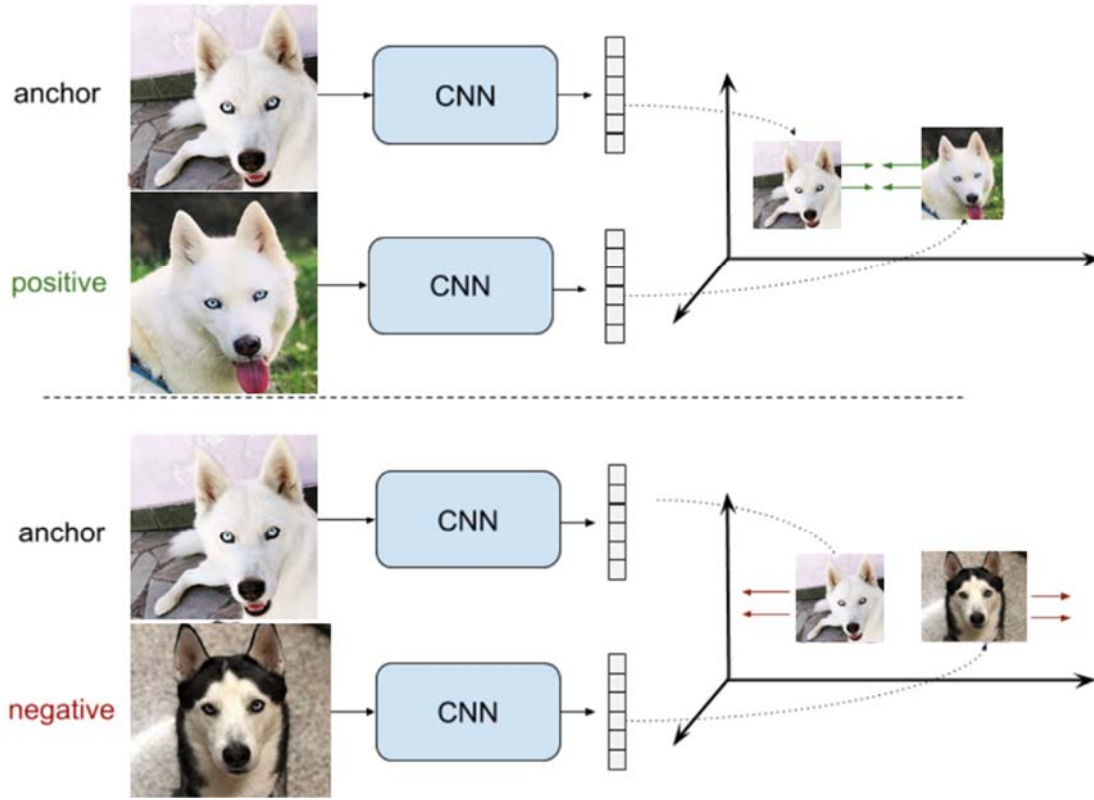


Figure 52: Triplet Loss example for our dog network

If r_0 and r_1 are the pair elements representations, y is a binary flag equal to zero for a negative pair and to one for a positive pair and the distance d is the Euclidian distance, we can equivalently write:

$$L(r_0, r_1, y) = y \|r_0 - r_1\| + (1 - y) \max(0, m - \|r_0 - r_1\|)$$

Figure 53: Pairwise Ranking Loss

4.6.4 Triplet Sampling

4.6.4.1 The problem of Overfitting

Overfitting refers to a network that models the training data too well. This kind of models, happen when they learn the detail and noise in the training data to the extent that it negatively impacts the performance of the model on data it hasn't seen before. To avoid overfitting, it is desirable to utilize a large variety of images. Because of the numerous images of dogs, the number of possible triplets increases cubically. It is computationally prohibitive and sub-optimal to use all the triplets. It is crucial to choose an effective triplet sampling strategy to select the most important triplets for rank learning.

4.6.4.2 Hard Triplet Mining Strategy: Hard sampling

One of the optimizations to the training processes proposed in the paper is the triplet selection process, Hard Triplet Mining. In hard sampling, the farthest positive and closest negative *only* are utilized. For each anchor image, we select a positive image that has embedding farthest from anchor's and we call it Hard Positive. Moreover, we select a negative image that has embedding closest to the anchor's what's so called a Hard Negative. In the next Figure, n_3 is the closest negative for the anchor a . Thus, assuming p is the farthest positive, the loss will be computed using the triplet (a, p, n_3) . In conclusion, the negative sample is closer to the anchor than the positive. The loss is positive and greater than m .

Hard Triplets: $d(r_a, r_n) < d(r_a, r_p)$

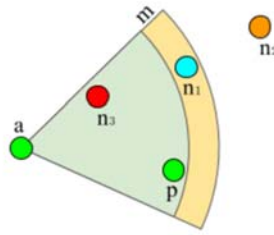


Figure 54: Hard Triplet: Hard Positive and Hard Negative

In our project, adaptive hard triplet was used as a triplet dataset configuration method to sufficiently train the increased number of dogs. Unlike the existing hard triplet configuration, the adaptive hard triplet configuration method adjusted the ratio of the hard dataset according to the loss value of the model. Its values can vary from zero to one. Previously, we have stated that our batch size is going to be 30 (3x10 images for each category) due to the fact that we are dealing with triplets in triplet loss. Therefore, it is anticipated that the maximum number of hard triplets will be 1/3 of the batch size. The number of hard triplets increased as the loss decreased and reached 10 when the loss was zero. The value in this is rounded to an integer and used as the number of hard triplets. The same model was trained on both the base method and the proposed method for a performance evaluation and comparison. The alpha of the triplet loss was 0.3. The model's compiler as we mentioned is Adam optimizer, and the learning rate is 10^{-4} in the entire epoch.

$$G(Loss) = \exp\left(-Loss \times \frac{C}{BS}\right)$$

Figure 55: Loss Value to adjust ratio of the hard dataset

In the making of this procedure we used `global_define_hard_triplets` method that gets all of the positive and negative images for our anchor image and calculates the distances between them. The algorithm decides which images are hard positives and hard negatives. Below, we can observe an example of the hard triplet selection.

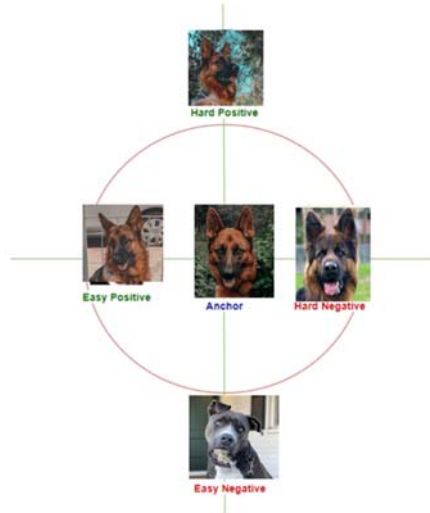


Figure 56: Hard Triplet Selection: Hard Positive and Hard Negative.

4.6.5 Adaptive Hard Image Generator

Moving forward to the main process of training, we need to make as many iterations as our epochs. Number epochs is equal to the number of times the algorithm sees the entire data set. So, each time the algorithm has seen all samples in the dataset, one epoch has completed. We have set epochs to be 100. In this for loop we call from model the `fit_generator` method instead of `fit`, where we just had to give our training generator as one of the arguments. The steps per epoch we are sending, specifies the total number of steps taken from the generator as soon as one epoch is finished and next epoch has started. We decided to set steps per epoch to be 300 our generator can select online hard triplets, at each step, for training. It includes an adaptive control on the number of hard triplets included during the training. We have already analyzed the procedure of hard triplet sampling in the previous subchapter. The filenames of our dataset and their corresponding label are going to be used to create our data generator. After that we have a for loop to loop over as many batches as defined by batch size and then we are loading the current batch of samples in `batch_samples`. To store the image sample and labels we are creating two empty list at the beginning `f_triplet` `y_triplet`. Now we loop over each sample in the current batch for each sample and it generates predictions for the input samples from a data generator. Our generator returns a History object.

Its `History.history` attribute is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values. Keras takes care of the rest. Note that our implementation enables the use of the multiprocessing argument of `fit_generator`, where the number of threads specified in workers are those that generate batches in parallel. We are saving this information for every epoch to h5 files named with the name of our network and their number of epoch.

4.7 Testing

4.7.2 Introduction

After we have built a machine learning model and trained it on some data, in this section we will discuss how to test and evaluate our model. We will put on the test, algorithms that their main target is to classify different dogs and identify input dog images. Also, we will briefly analyze many measurements we have taken during the process that were either decreased significantly at the end of our training or they have elevated. In this way, we will be able evaluate our built network and make some observations. Firstly, we will take a look at various mandatory functions that will help us try out our algorithms.

4.7.3 Useful Functions

4.7.3.1 Finding the Best Threshold

A vital function in this implementation, is to find the best threshold and accuracy when it comes to stating similar images. We need to have in mind that increasing the threshold, we expect from our model to be very sure about its prediction which means we will be filtering out false positives. In this scenario we are targeting **precision**. This might be the case when our model is a part of a mission-critical pipeline where decision made based on positive output of model is costly (in terms of money, time, human resources, computational resources etc...) In our network, time is what makes us doubting a high threshold by virtue of the pet owners waiting for the results in the upcoming website.

On the flip size of having much more false positives , having a higher threshold we are taking accountability for having many of the false negatives results. Following this logic, if we decrease the threshold, our model will say that more examples are positives, which will allow us to explore more examples that are potentially positive. On this scenario we are targeting **recall**. This information is important when a false negative is disastrous e.g. in medical cases (You would rather check whether low-probability patient has cancer rather than ignoring him and find out later that he was indeed sick). We may allow to say that losing a pet

is one of the most saddest and desperate moments for a pet owner. By saying that, it is better to have a high threshold and show more possible results that may happen to be their lost dog, but it needs to be tested several times to be sure about it.

Diving into the process of finding the best threshold an accuracy, we need to calculate the distances between images from the same class and images from another class so we can decide. Randomly, we select if we are going to work with 2 different dogs' images or photos of the same dog. We find again randomly two different photos either, we are looking in the same class or in another. Continuing, we need to find the embedding vectors of the two images in our model space and finally calculate their distance. In every iteration, we considerate a good threshold when two images are the same and their distance is small, but also when two different images have a larger distance than the distance they have with photos of the same class. Accuracy is calculated by the minor average differences in the distances.

4.7.3.2 Create Embeddings

We need to locate our input images to the embedding space we have talked about before. Using TensorFlow's Keras model we can predict their vectors. It is a fact that our inputs will be in a certain shape (224,224,3).

4.7.3.3 Find Distance

There are many ways to calculate the distance between two vectors. We decided to use the common and well known Euclidean Distance that is defined below.

Measure	Meaning	Formula	Relationship to increasing similarity
Euclidean distance	Distance between ends of vectors	$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots + (a_N - b_N)^2}$	Decreases

Figure 57: Euclidean Distance

Having both the embedding vectors and our distance measure formula, we can calculate the distance between two embeddings.

4.7.4 Algorithms

4.7.4.1 Brute Force Nearest Images

In this algorithm we have used all of the above explained methods. As our titled algorithm explains itself, we will find the nearest images of a dog picture by comparing every image in the set. With the library of sklearn we will read all of our images. We will read all of the images that are contained in our database. Noting that we only work with jpeg images. For every image passing through the testing of this algorithm we are calculating its embedding and eventually its distance from the searched dog. If their distance is smaller than the average threshold we have calculated before, it is considered to be a similar photo. In two arrays we save the likely similar images with their distances from the photo we have selected in the start. Because of the wide range of database images (our database can hold limitless lost dogs), we decided to find the top 6 images that are most likely to be similar with our searched dog. In order to find the top 6, we need to sort the likely similar photos array based on their distance indices. In every iteration, we find the smaller distance in the distance array with the function of python called `min()`. We this function we find not only the minimum distance but also we can identify the indices of the smaller distance. Knowing the indices of the smaller distance we know which photo is included in the top 6 similar images and we append it in our `top_6` array. Afterwards, we need to pop out the smaller distance and the photo with the smaller distance and repeat this process for another 5 iterations.

4.7.4.2 Agglomerative Clustering

We will test our network with images from the testing set. First we will load our trained model and then we will compute the Network's output for our test images. Specifically it will predict our vectors in the embedding space. These vectors, are going to be placed into the KMeans algorithm and KMeans will group those pictures! The output will be the labels for each image! Thereafter we will plot our clusters to check how good they are distributed. It is a quite simple process to check our model but it is not useful to identify the most similar images, given that we need to know how many clusters we have. Basically Kmeans algorithm is used to evaluate our network knowing the number of clusters we have. In our research, our purpose is to find a way to not waste any algorithm just for evaluation.

We want to take advantage of every possible algorithm that can help us in the dog recognition process and this is what we are going to do. Before moving on to the next algorithm, let's take a look to our clustered data. We tried to cluster the some dogs' pictures that our network hasn't seen before, after embedding vector computation using the k-means clustering algorithm. We can see that the results are satisfyingly good, regarding the complexity that there

is behind our pictures. A picture dog may be taken with very different angles, lighting, position and in another landscape. The figure below also shows one of the mistakes made by the algorithm: two different clustered dogs. We can't say that they are badly clustered because the colour patterns that they appear in the dogs are very similar, excluding the fact that the first dog is brown and white and the other one is black and white. Also, if we see closely, in the first cluster on the third column we have a different dog that is very similar to the other one because its colour is fully black too. We understood this by reading the labels. Some images, not even human brain can stand them out.

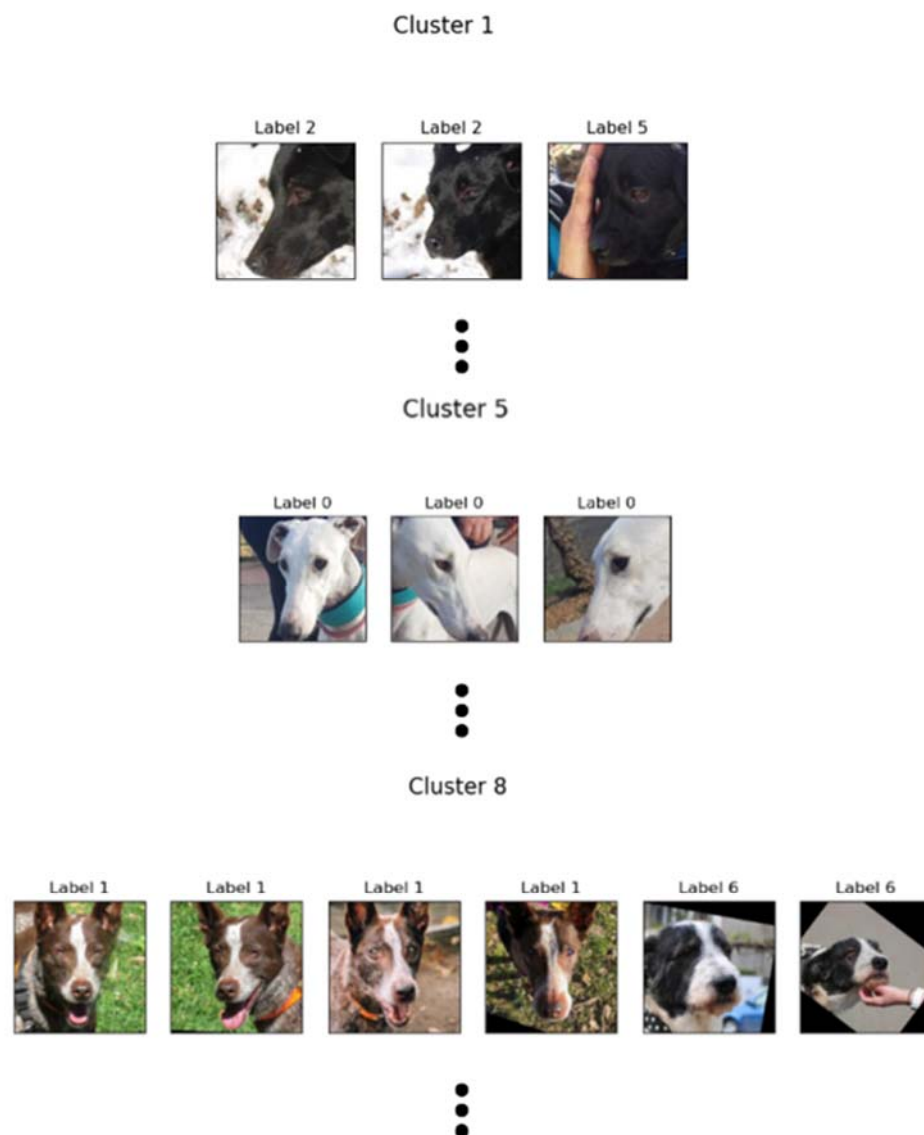


Figure 58: Results of Clustering

4.7.4.3 Hybrid Solution: Cluster the Nearest Images

This is an evolutionary method in our research, that can help us get both of the best worlds. We need to do some changes on the way to combine these two methods that we have mentioned earlier, to customize the algorithm's way to our problem needs (Appendix B). As it is expected, we will start off the process by finding the most likely similar dogs to the dog we have searched for. As we have said in the verification method we will find the embeddings for all the pictures based on their embedding vectors.





Zeus		Rockie		Linda		Lucky	
Brownish orange colour		Light brown, spots of black & white		Brown- a little white		black	
Medium size & brownish eyes		big black eyes		Small black eyes		Medium size black eyes	
Medium size nose		Medium size nose		Very small nose		Big nose	

Figure 59: Four dogs with their characteristics. Our Network will be able to extract some of them like patterns, eyes, fur texture etc. Based on their features, the images are going to be place in the embedding space.

This time we will not take the first 6 similar photos. We will increase our threshold by the accuracy value we have calculated. In this way, we will have more possible false negatives. On the one hand we want to minimize the range of images to look for (and apply clustering later on), but on the other hand we want to make sure to have a wider range of photos that didn't make it through the threshold. Afterwards, we will find out the labels of our testing images that will be our found dogs in our database. Now that we have eliminated many photos from our database, we will cluster the left out images thar some of our possible results.

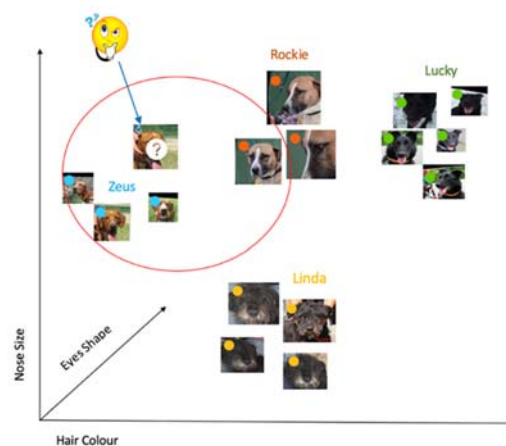


Figure 60: Clustering Nearest Embeddings

As we have pointed out, in K-Means algo, we need to declare how many clusters we want. For this implementation, we will estimate the optimal number of clusters every time we try to cluster data. We make multiple iterations for many values of k starting with 1 cluster and ending placing each data in a different cluster. For every value of k, our algorithm will compute the SSE. SSE is defined as the sum of the squared distance between centroid and each member of the cluster. Then we plot a K against SSE graph. We will observe that as K increases SSE decreases as dissipation will be small. So the idea of this algorithm is to choose the value of K at which the graph decrease abruptly. This sort of produces a “elbow effect” in the picture: In the above picture we can see a elbow occurring around 2- 3 so that’s a good number to choose.

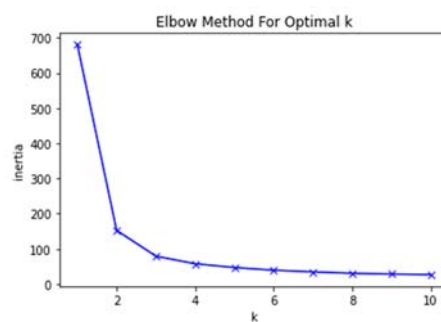
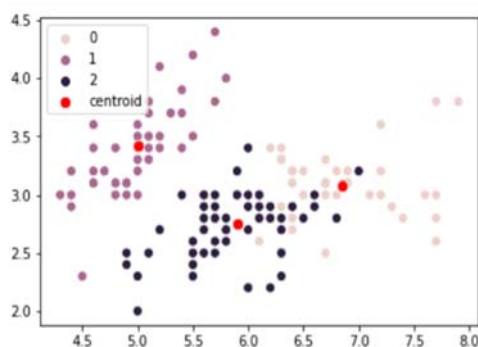


Figure 61: Elbow Method for Optimal k

Kmeans is then performed with the elbow K as the number of clusters, likewise we mentioned earlier. We will only show as a result for our searched dog, the cluster that contains our dog. We can see the following plot how our images positioned in the embedding space applying clustering with K-Means.



	0	1	2	3
0	6.850000	3.073684	5.742105	2.071053
1	5.006000	3.418000	1.464000	0.244000
2	5.901613	2.748387	4.393548	1.433871

Figure 62: Clustered Images and Centroids of each cluster

Chapter 5

Evaluation

Contents

5.1 Overview	56
5.2 Results	57

5.1 Overview

This section describes a trained neural network's evaluation results using a validation set to measure its quality, considering dog images that were not taking part in the training. As we discussed previously, it's important to use new data when evaluating our model to prevent the likelihood of overfitting to the training set. However, sometimes it's useful to evaluate our model as we're building it to find that best parameters of a model - but we can't use the test set for this evaluation or else we'll end up selecting the parameters that perform best on the test data but maybe not the parameters that generalize best. To evaluate the model while still building and tuning the model, we use the subset of the data that are known as the validation set. I'll also note that it's very important to shuffle the data before making these splits so that each split has an accurate representation of the dataset. We carried out an experiment in which we compare pairs of dogs for each class. If it picks two dogs that are the same, it is expected that the Euclidean distance between them should be below a certain small threshold. In addition to that, pairs of dogs that are different were also compared, yielding a distance above that threshold.

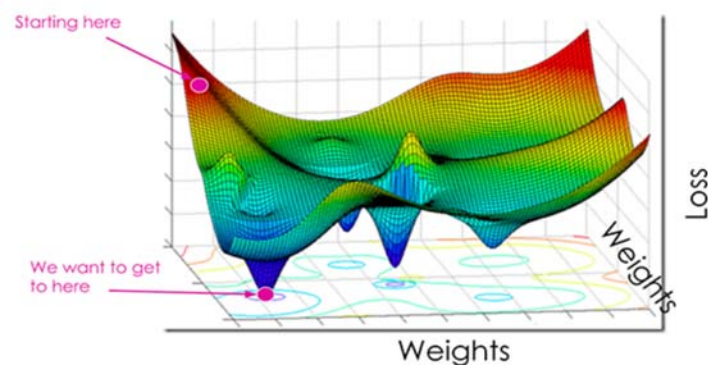


Figure 63: SGD algorithm to minimize the Loss and upgrade the Weights.

At the most basic level, a loss function quantifies how “good” or “bad” a given predictor is at classifying the input data points in a dataset. It is a fact that, the smaller the loss, the better a job the classifier is at modeling the relationship between the input data and the output targets. That said, there is a point where we can overfit our model — by modeling the training data too closely, our model loses the ability to generalize. It’s a balancing act and our choice of loss function and model optimizer can dramatically impact the quality, accuracy, and generalizability of our final model. We chose to work with Adam Optimizer and Triplet Loss.

5.2 Results

In order to evaluate our network we will see how our training and validation is adjusted by the end of all epochs. The triplet loss model was trained using the existing triplet loss learning method with 250 epochs. The second graph [Figure 65], illustrates the training results. We can see the 2 metrics: Loss and accuracy. Training loss is the error on the training set of data. Validation loss is the error after running the validation set of data through the trained network. Neural network is defined as the percentage of triplets satisfying the triplet loss margin condition in a randomly sampled batch of triplets. During an epoch, the loss function is calculated across every data items and it is guaranteed to give the quantitative loss measure at given epochs. We are plotting a curve across iterations to see how loss differs from the beginning to the end of the training. An epoch is an arbitrarily often repeated run over the whole dataset, which in turn is processed in parts, so called batches. After each train on batch, a loss is calculated, the weights are updated and the next batch will be expecting to get better results. These losses are indicators of the quality and learning state of our NN. Note that we can log our losses in two periods: After every Epoch or After every Iteration.

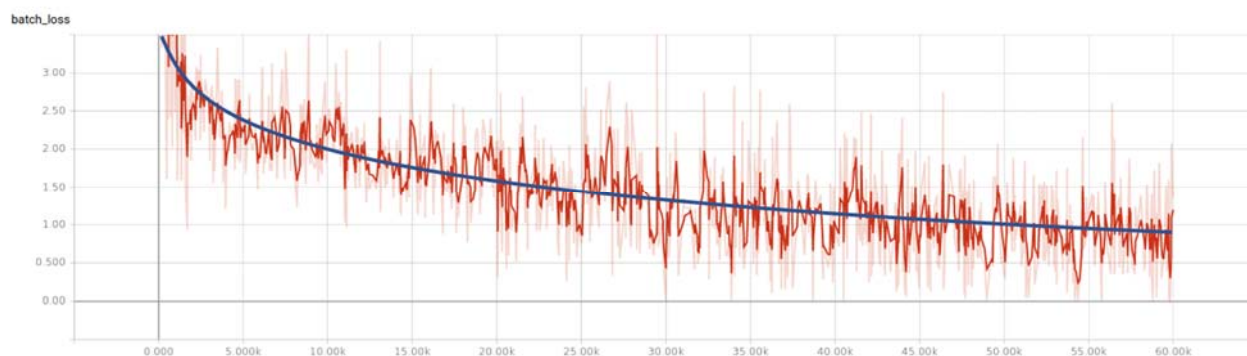


Figure 64: Batch Loss

From the above graph [Figure 64], we can observe our Batch Loss. The x axis is the total number of batches that were created and the y axis is batch loss. Luckily, our loss gets smaller by the end of the iterations. On the other hand, we can see a lot of ups and downs. If we look at the loss (red color) in a particular range of batches, we will see a very noisy estimate of our dataset loss because the batches also stored all the samples our model had trouble with, or all the samples that are trivial to succeed on. Normalized and averaged values are indicated by the curve that hasn't any abrupt changes. Continuing to the next graph [Figure 65], triplet accuracy among the batches is demonstrated. Our accuracy improves during our training. Using more triplets in the batch improved our accuracy. This graphs inherits the conclusions we have made from the previous graph.

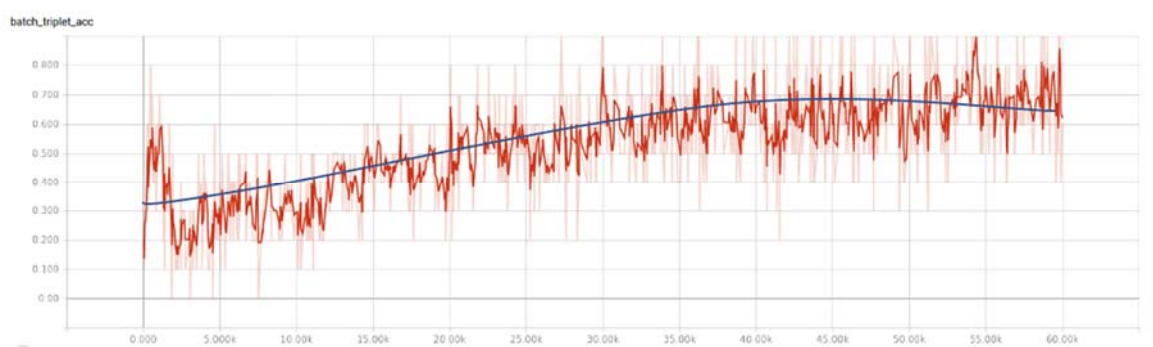


Figure 65: Batch Triple Accuracy

Generally, a network to be considered as a reliable one, loss must decrease by the end of our training and accuracy to be increased. The following figure [66] indicates when are we having a good learning rate based on our loss.

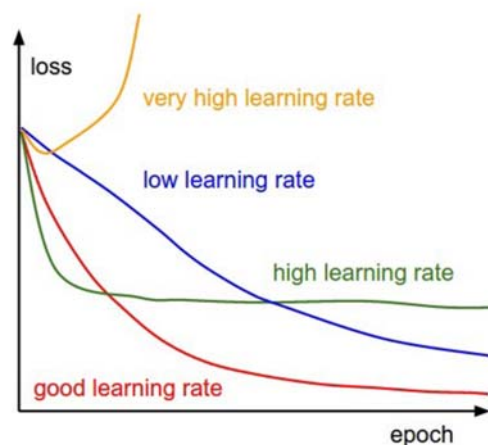


Figure 66: Learning Rate based on Loss

Our network succeeds to lower the loss over iterations and improve our accuracy as it appears in the following figures. It is relatively leveled in the end. We can see that training loss

is slightly larger than the validation. In our case, it is acceptable to consider our network a good fit. Our validation set may consist of "easier" examples than the training set and this leads to be smaller. We can not say it is underfitted because they are very close. To be sure about our results we should try cross validating our model, but because our model takes too much time to train we will just retrain it on a differently mixed train/val sets to see if the trend persists. The answer is that our network is still a good fit.

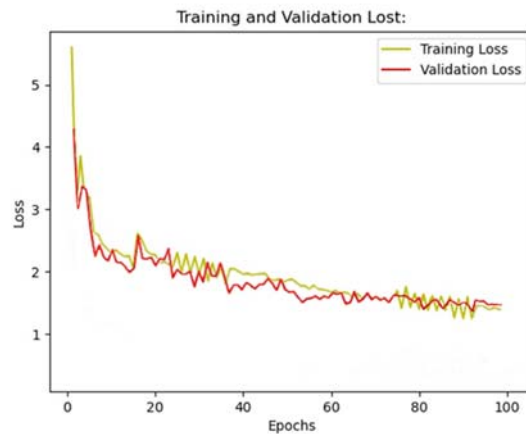


Figure 67: Training and Validation Lost for 100 epochs

The next image [Figure 68] is pretty much self-explanatory. The evolution of prediction accuracy during the training process is shown in Figure 1 for 100 epochs. At some point our graph started to converge so we didn't need to visualize more the losses and accuracies. We can observe that training accuracy and prediction accuracy increases over iterations. There are small local ups and downs around the loss and accuracy curves because of the stochastic gradient descent algorithm. Some ranges that the validation loss is a little larger than training is that we have added a regularization technique, specifically a dropout layer (0.5) to avoid overfitting.

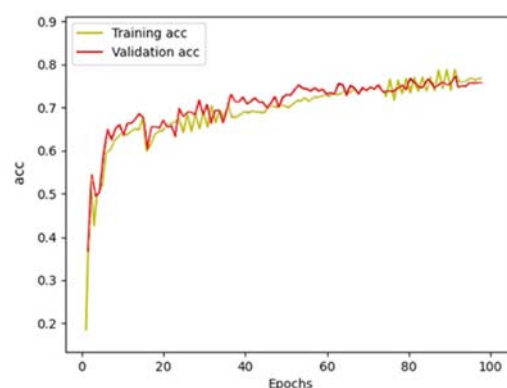


Figure 68: Training and Validation Accuracy for 100 epochs

Chapter 6

Web Application

Contents

6.1 Motivation.....	60
6.2 Used Technologies.....	60
6.3 Methodology.....	61
6.3.1 Basic Start for a Flask Application	61
6.3.2 Website Setup	62
6.3.3 Background Algorithm	62
6.4 Web Demonstration and Results	62

6.1 Motivation

The purpose of our dissertation, as we mentioned, is to help pet owners find their lost pet. It's a real life problem that many people would find interesting. It would help enough people to pin their hopes on such a system to meet their dog again. We have finally built a reliable Network for dog identification using deep convolutional networks, but we want our model to be available for the end-users so that they can make use of it. Model Deployment is one of the last stages of any machine learning project. There are many different things we need to take care of when putting our model into production. We want to show with real data, data that the algorithm has never seen before, the methodology and the results of such a system. Eventually, we designed a modern and at the same time a simple website that will run locally, to show the system that will be developed in the final stage in the future. It is vital to show to someone who has never worked with neural networks before the power of today's abilities. A website is a nice way to give a taste to people what our system is capable of. Later in this section we will represent the design and the functionality of our website.

6.2 Used Technologies

Finding a way to represent real-time results, we have come across the Flask, which is a web application framework written in Python. It has multiple modules that make it easier for a web developer to write applications without having to worry about the details like protocol

management, thread management, etc. It is way much simpler when it comes to use Flask to run in the background Machine Learning Algorithms. Flask is giving us a variety of choices so we can develop a descent web application. There are many useful tools and libraries that allow us to build a web application.

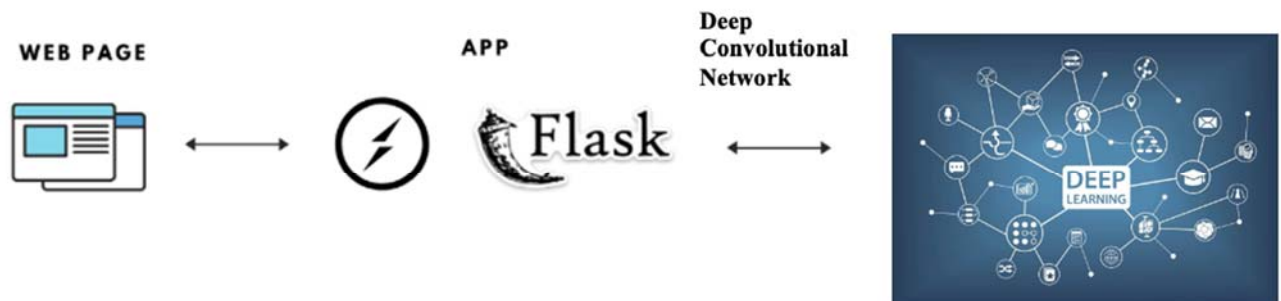


Figure 69: Communication of Flask with the Web page and ML Algorithms

6.3 Methodology

6.3.1 Basic Start for a Flask Application

When we run Flask, it will look in the current directory for anything labeled “app.py” which is what our flask application is named. In a template folder we have the main html file we are using to start off the website. When we have flask running, we are using a development server which only runs on the localhost. When we go to localhost in our browser we will see our index page corresponding to our first route. A note to keep in mind with Flask, is that when we make any changes to the application, they are not reflected until we stop and restart the app itself. We have used flask scripts to start a local development server. We have trained our Network through Flask before running it to set in action our website. We have taken this path because when it came to load our models later to apply algorithms, TensorFlow did not recognize it otherwise. To make possible the procedure of training, we need to import the filename of the code in the application file. When we import files that are not organized in functions and a main, the program will automatically run all the code of the imported files. We have normally set 250 epochs and 300 steps per epoch, the first time we will train the Network. Moving on, we don’t want our application to run training again, so we set epoch to zero. Any files or images are saved on the run, are being saved in a static folder. Writing an html file in Flask is basically very similar to writing a file in normal scenarios. We need to make a few changes so that our app and our website can communicate.

6.3.2 Website Setup

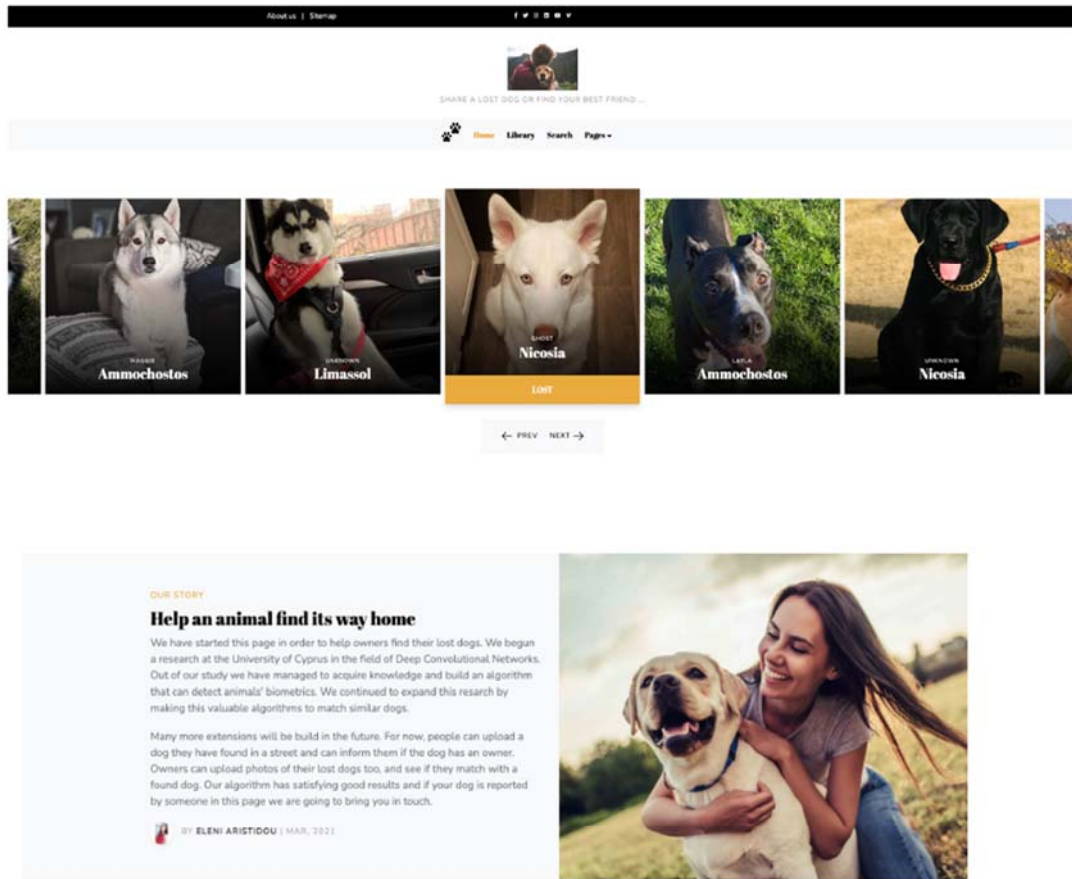
We continue on how our application is built. The user will load our website and the first thing we will see is our main page. In the main page he can see our website's logo and some information about the system. Going to Search button, he will be able to upload a picture of his lost dog. When the user uploads an image of his dog, Flask will save that photo as the searched image in static folder. Every time he uploads a new photo, the searched image will change. We do this so when we move on to run the algorithm, it will be able to find out the searched image. We have a database of photos that are found dogs uploaded from random users. In order to test and simulate the procedure we have fed our algorithm with photos that has not seen before. The algorithm we chose as the final one, will compute the top similar images to our searched dog. It will eventually give us the results below our searched dog.

6.3.3 Background Algorithm

After many tests and changes in our algorithms we have came to the conclusion that we will use our hybrid method as the one to put into operation for our system. The hybrid method consists the verification method firstly proposed in [1], with the difference that our algorithm will be used only to exclude photos that are not possible to be anywhere near our image in the embedding space. It will not be used to find out the top 5, but to remove photos and minimize the range we will look for within reach images. We have explained on the previous chapter how verification works with vector embeddings. Thenceforth, we will expand our method by clustering the remained photos that have left after the verification. Kmeans is a classifier that will cluster our data. If Kmeans decides that some photos are the same, they will be clustered. We have already expounded how clustering partitions images in groups, given that we run multiple times to find the best number of K = number of clusters.

6.4 Web Demonstration and Results

In the following images, we present the website for our findyourdog.com. In Figure 70, we can see the home page. There is an indicating small library for the people who are entering the site. They will see many dogs that some of them are lost and some other are found by someone. After the carousel we can see some information about how we have started this journey and what this page is about.



Part of the of Research



Figure 70: Home Page

In the next image [Figure 71], people can look up to the larger library of dogs. There is a small spot where people can directly press for contribution. Websites gives the opportunity to welcome subscribers of the site to get updates. By entering their email, it is possible for them to upload a photo of a lost or a found dog.

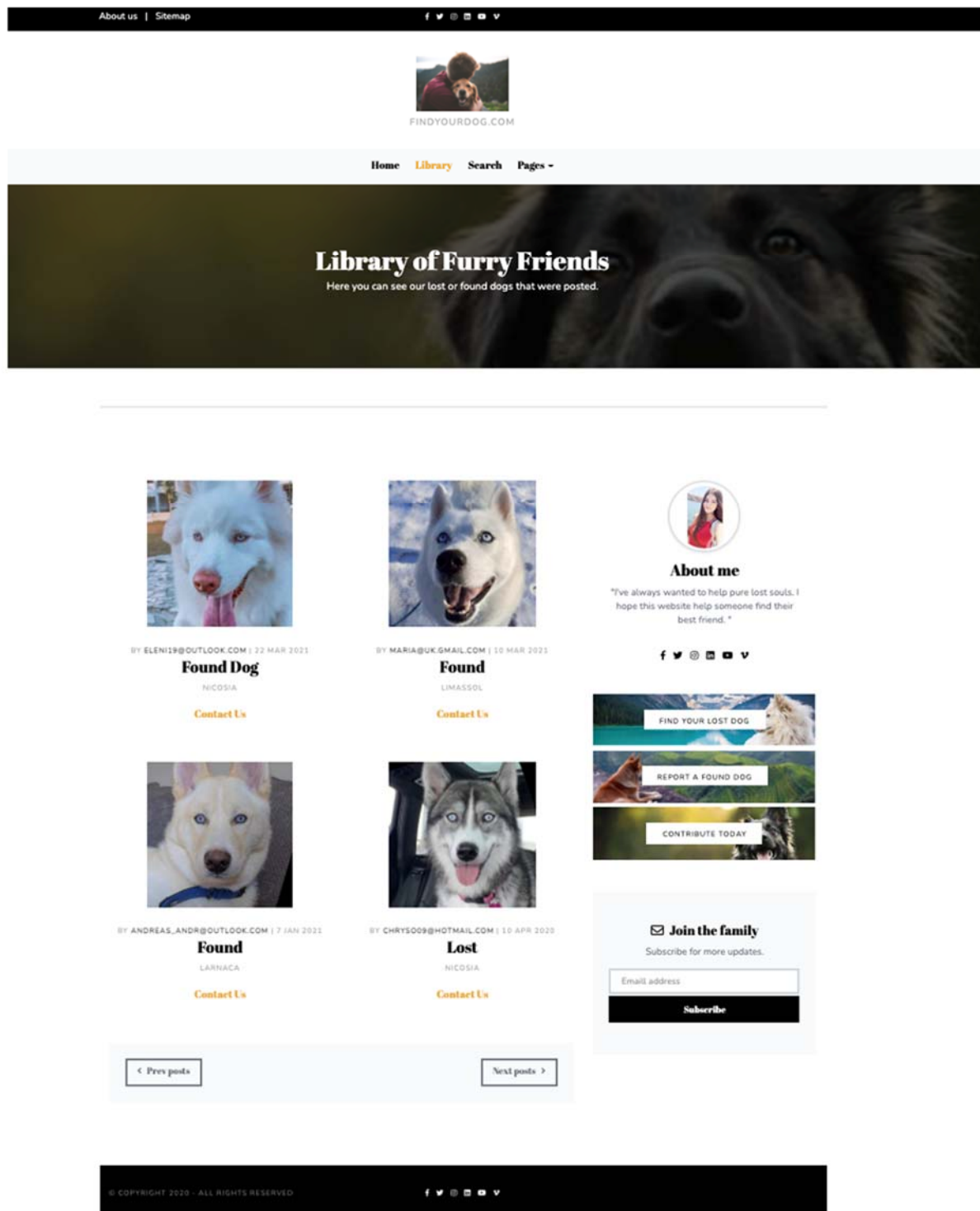


Figure 71: Library Page

In the following images, there are two illustrations of how our search page is. In Figure 73, we can see the available options for Search tab. People can either chose to upload a lost or a found dog. They can click their option in one of the checkboxes. They can upload a photo from their computer. By pressing submit, the algorithm will start searching in the found dogs database or in the lost dog database based on what they marked in the checkbox [Figure 72]. In the next figure we can finally see the results. First we can notice the searched dog and below of it are the results. In the next Figures [74-77] we can see some of our results.

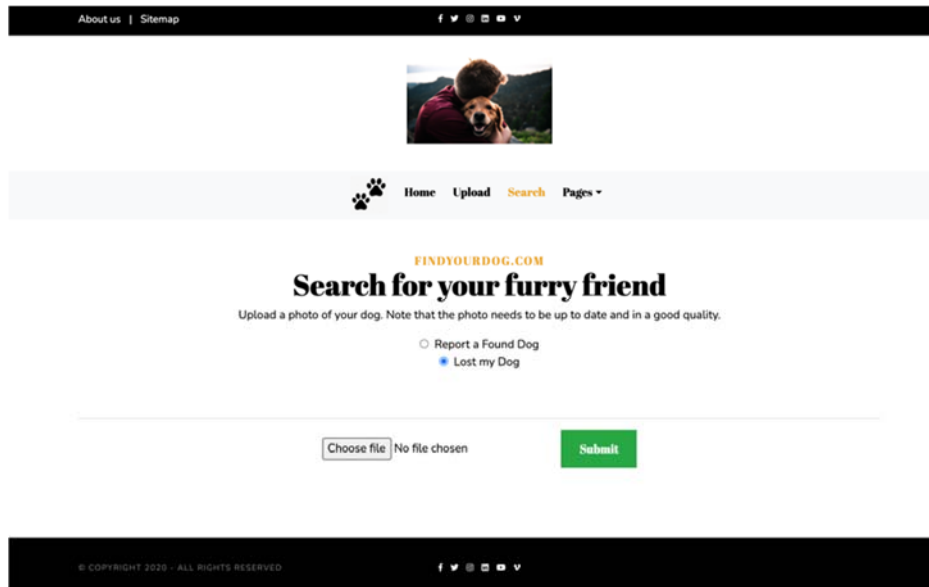


Figure 72: Search a Dog Page

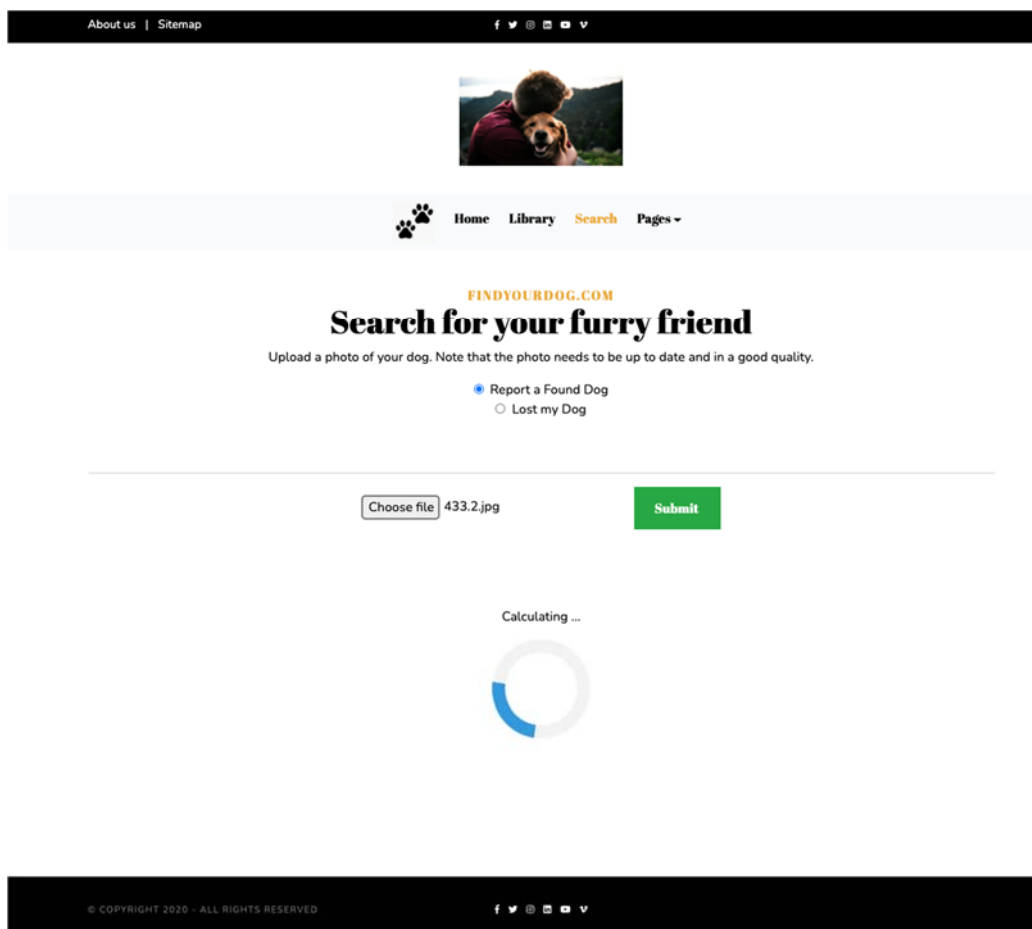


Figure 73: Calculating Results

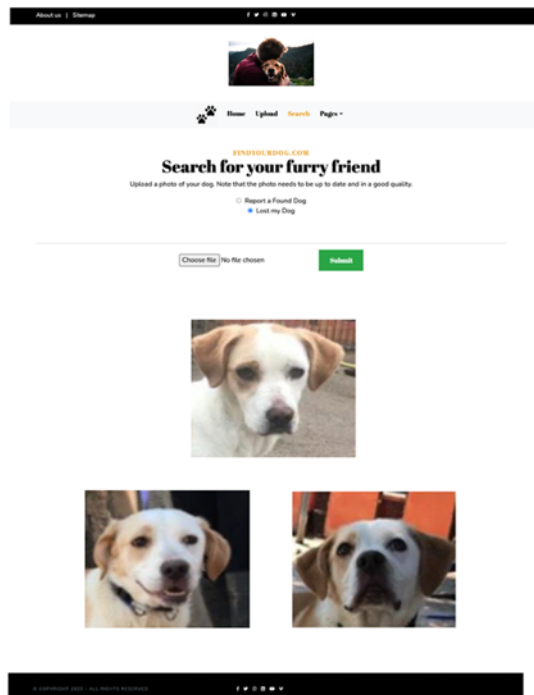


Figure 74: Example Result 1

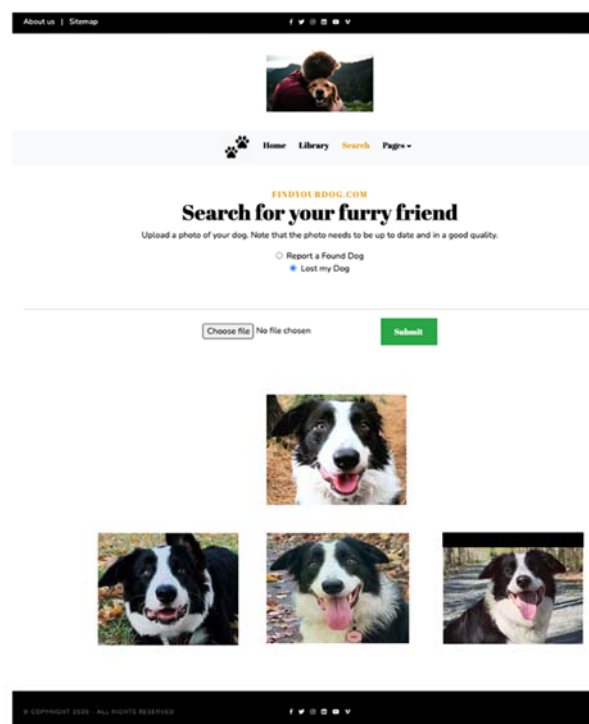


Figure 75: Example Result 2

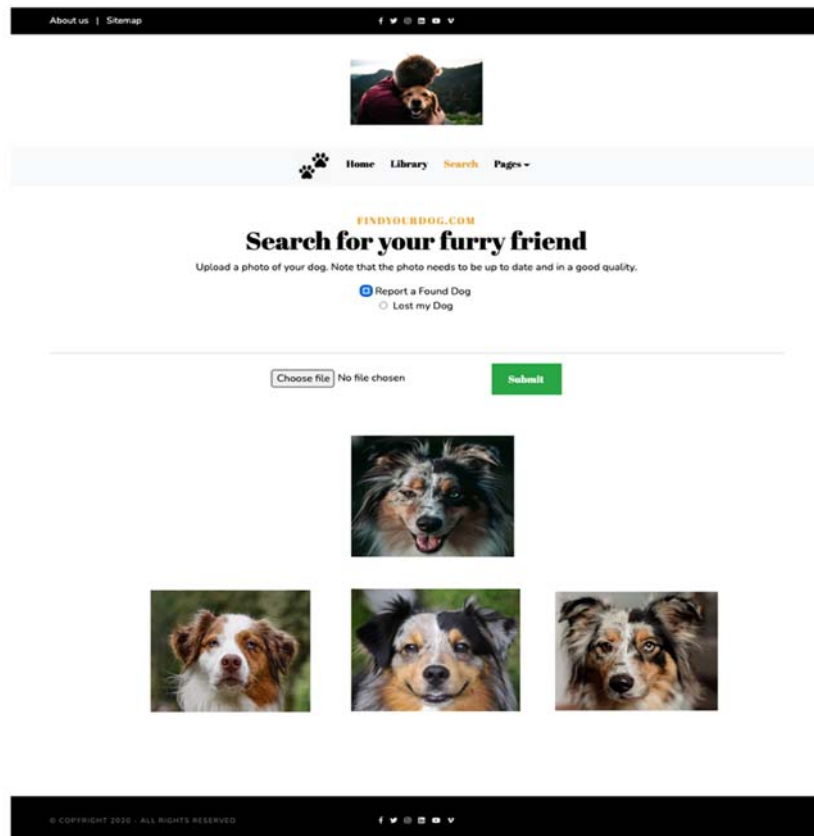


Figure 76: Example Result 3

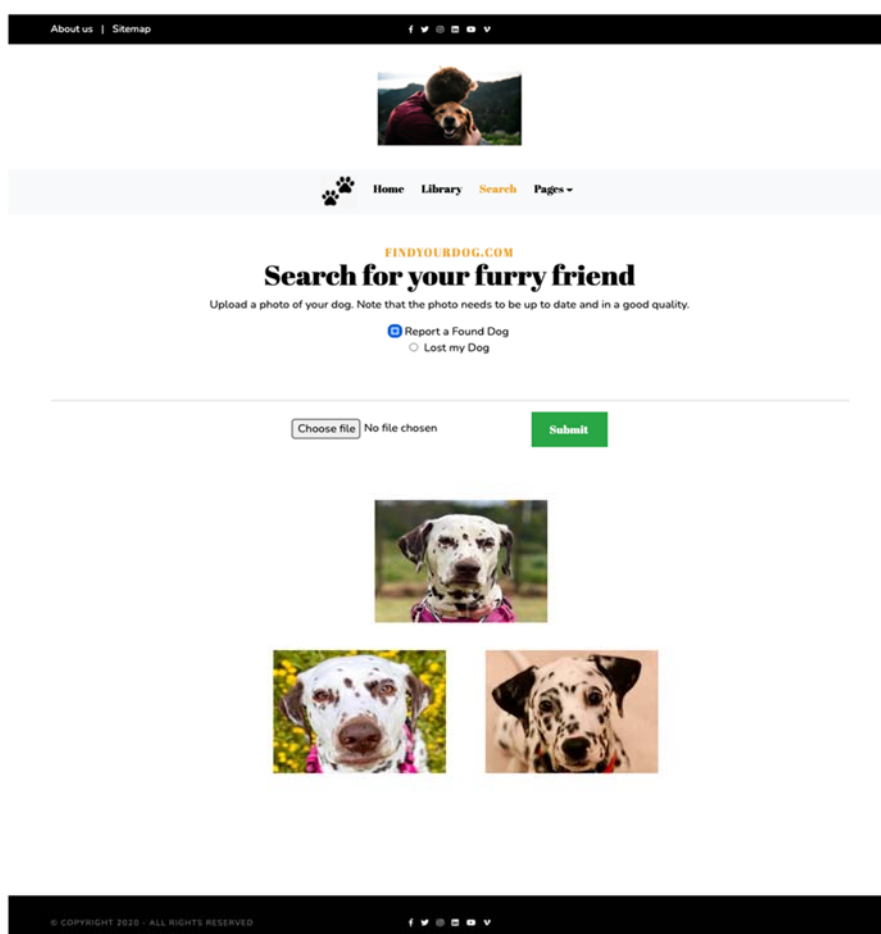


Figure 77: Example Result 4

Chapter 7

Conclusions, Limitations & Future Work

Contents

6.1 General Conclusions	68
6.2 Limitations.....	69
6.3 Future work.....	70

6.1 General Conclusions

Animal identification is a very demanding challenge due to the fact that we have to work with deep neural networks. Deep networks need to be studied in depth to build a model that can make predictions. Many researches have been made mainly in Human Recognition with a very high accuracy, but only few were invested for dog identification. Recognition task in dog faces appears to be more complex than in human faces due to the lack of available data. We have been able to download [2] dataset and add our very own data collected manually. It was a necessity to pre-process the new data we have collected. We have applied a dog face detector to detect and save the found landmarks. In a few cases our algorithm didn't find the face of the dog because of the difficulty of the image. To solve this problem, we used a library that let us mark manually points in the picture. Many photos were rejected as they were considered that they would not fit in our training. To continue, we loaded the landmarks for each photo to align, crop and resize the images to have straight, clear and zoomed dog faces. It is very clear that what makes each dog unique encompasses all of its characteristic such as the texture and length of their fur, the patterns of colours in their body, their height and mass, their smile and look, and many many more variations in dog face pictures. With our research we are now able to understand the process of training a network with so many layers and neurons.

The deep ranking model we present, employs a triplet-based hinge loss ranking function to characterize image similarity relationships. We also used an efficient online triplet sampling method that enables us to learn deep ranking models from very large amount of training data. Instead of choosing simple triplets we used hard triplet loss. Hard sampling seems to have advantages when compared to simple triplet sampling. We try to put high the stakes, by giving

the network images that are very similar but yet different. A major caveat of the triplet loss, though, is that as the dataset gets larger, the more are the possible number of triplets. To make matters worse, our model relatively quickly learns to correctly map most trivial triplets, rendering a large fraction of all triplets uninformative. Thus mining *hard* triplets becomes crucial for learning.

On the other hand, being shown only the hardest triplets would select outliers in the data that may be presented as false negatives. Evaluation gave us the opportunity to observe metrics of our network. Important metrics taken in the training process was the triplet loss and the validation loss. Also we obtained the training accuracy and the validation accuracy. As we saw in the graphs, our model loss decreases during a cycle and increases after hard augmented triplets generation. We can see the convergence of the loss and the accuracy approximately after 70 epochs. To avoid overfitting we will use a trained model under 70 epochs. If we use a later version model we will see that the validation loss will increase as the training loss will decrease, something that will affect our program on new data. With this study coming to an end, we saw how important is to find this real life problem a solution and construct a web application to show the abilities of our network. Flask really helped us configuring out how to run machine learning algorithms in the background of a website.

We can see from the results that the Network understands the similarities in the pattern of shapes and colours of the dog. Our model succeeds this process, with the layers we have added in our Network. Generally, our model is not perfect or at least with a high accuracy to solve a real identification problem but has its becoming very handy to help dog owners finding their lost pet.

6.2 Limitations

The system we have built has its limitations and constraints on practical and theoretical level. To train the net, we need to do an extraction of dog's features by adding many layers to our architecture. Our network does not prosper in the same way when we try it with dog's whole body. There is a need of adding far more layers to extract this kind of features. Also, we have a very limited amount of images in our dataset. We may have thousands of data and still not be able to make our network to be at its peak. Our performance is being limited owing to the low normality of dog faces and the lack of related data. By the same token, when we are testing our program in the website implementation, the user will be able to upload a processed image. For academic purposes, we prepared some images that are aligned and ready to be tested through the program. In addition, the user will have to wait a few moments until the algorithm find similar images. Even if there is no similar image in the database, the algorithm will output a result because it is trying to find the nearest image to ours. We need to take a look at thresholds

and criteria to adapt the system whether it will tell us that we have similar images. To finish our general thoughts about the limitations in this study, we may allow to say that we faced many difficulties with the versions of TensorFlow. It will be much more easier for future works to upgrade the version of this model in the forthcoming releases.

6.3 Future work

Through this application for Dog Identification, we have achieved to test real data and allow to users to check our network. In general, our model produces satisfactory results but there is still room for some improvements. Taking into consideration the limitations we have, we can set as a goal to pass by as many restrictions we can. Starting off with the data acquisition, we could invest into a study for Social media Image Mining. Specifically, we can obtain big data from user-generated content on social media sites and mobile apps in order to extract dog images, based on tweets, hashtags, captions or even comments. Data that is publicly available, can be used on big scale machine learning methodologies. Tools and many techniques on Data Mining could help us reserve images of dogs, posted by their owners in their social media profile. After the dataset collection we move on to pre-processing. Dog detection can be our next field of study for improvement. In our model, we need to zoom on to the dog's head and align it straight. They were times, our detector either failed to find the right position of eyes and nose or it failed to detect a dog at all. Continuing to the training process we know that we need to extracted face characteristics. It will be appealing to study more layers in the convolutional neural network to absorb as many factors as we can. In this way, may a future extend of this application will be able to work with wider range of images that it won't need to feed the network only with dogs faces but their bodies too. Along with that, our model computes the losses with hard triplet selection. An another approach would be semi-hard triplets. These are defined as triplets where the negative is farther from the anchor than the positive, but still produces a positive loss. Because hard triplet sampling will road us to an early convergence we could put to the test semi-hard triplet loss that pairs every anchor with every positive point. This sampling strategy is more robust to the model collapse but converges slower than the hard-mining strategy. Finally, we can say that using Flask helped us represent some of our results but it is a temporary solution. This implementation of the website only covers a small portion of what we would create with an average web app. This does not touch on things like cookies or validation or any other thing that we would nee for a public – facing web app. Concluding to our study, going through a deep convolutional network is an extremely fascinating process that makes us think how far technology has come that can reach and surpass the human brain.

Bibliography

- [1] Schroff, F.; Kalenichenko, D.; Philbin, J. (June 2015). FaceNet: A unified embedding for face recognition and clustering. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 815–823. arXiv:1503.03832. doi:10.1109/CVPR.2015.7298682. ISBN 978-1-4673-6964-0. S2CID 206592766
- [2] Mougeot, G.; Li, D.; Jia, S. A Deep Learning Approach for Dog Face Verification and Recognition. In Proceedings of the Pacific Rim International Conference on Artificial Intelligence, Cuvu, Fiji, 26–30 August 2019; pp. 418–430.
- [3] <https://arxiv.org/abs/1510.02781> , <https://link.springer.com/article/10.1007/s11042-016-3824-1>
- [4] Vlachynska, A., Doggie-smile, GitHub , URL: https://github.com/tureckova/Doggie-smile/blob/master/Final_project_report.pdf
- [5] The American Society for the Prevention of Cruelty to Animals® Statistics, URL: <https://www.asPCA.org/animal-homelessness/shelter-intake-and-surrender/pet-statistics>
- [6] <https://peeVa.co/missing-pet-epidemic-facts-and-figures>
- [7] <https://ieeexplore.ieee.org/document/8851971>
- [8] <https://www.semanticscholar.org/paper/Dog-Identification-using-Soft-Biometrics-and-Neural-Lai-Tu/b734354dfda02ebf55929e1ce74ed47cd8bb8ccde>
- [9] Chechik, G.; Sharma, V.; Shalit, U.; Bengio, S. (2010). "Large Scale Online Learning of Image Similarity Through Ranking" (PDF). Journal of Machine Learning Research. 11: 1109–1135.
- [10] Ailon, Nir; Hoffer, Elad (2014-12-20). "Deep metric learning using Triplet network". arXiv:1412.6622. Bibcode:2014arXiv1412.6622H.
- [11] <https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-020-00510-w>
- [12] Awad, A.I. From classical methods to animal biometrics: A review on cattle identification and tracking. Comput. Electron. Agric. 2016, 123, 423–435. [CrossRef]
- [13] Kumar, S.; Singh, S.K.; Singh, R.S.; Singh, A.K.; Tiwari, S. Real-time recognition of cattle using animal biometrics. J. Real-Time Image Process. 2017, 13, 505–526. [CrossRef] Appl. Sci. 2021, 11, 2074 22 of 22
- [14] Kumar, S.; Pandey, A.; Satwik, K.S.R.; Kumar, S.; Singh, S.K.; Singh, A.K.; Mohan, A. Deep learning framework for recognition of cattle using muzzle point image pattern. Measurement 2018, 116, 1–17. [CrossRef]
- [15] Kumar, S.; Singh, S.K.; Abidi, A.I.; Datta, D.; Sangaiah, A.K. Group sparse representation approach for recognition of cattle on muzzle point images. Int. J. Parallel Program. 2018, 46, 812–837. [CrossRef]

- [16] 23. Jarraya, I.; Ouarda, W.; Alimi, A.M. A preliminary investigation on horses recognition using facial texture features. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Hong Kong, China, 9–12 October 2015; pp. 2803–2808.
- [17] Hansen, M.F.; Smith, M.L.; Smith, L.N.; Salter, M.G.; Baxter, E.M.; Farish, M.; Grieve, B. Towards on-farm pig face recognition using convolutional neural networks. *Comput. Ind.* 2018, 98, 145–152. [CrossRef]
- [18] Crouse, D.; Jacobs, R.L.; Richardson, Z.; Klum, S.; Jain, A.; Baden, A.L.; Tecot, S.R. LemurFaceID: A face recognition system to facilitate individual identification of lemurs. *BMC Zool.* 2017, 2, 2. [CrossRef]
- [19] Deb, D.; Wiper, S.; Gong, S.; Shi, Y.; Tymoszek, C.; Fletcher, A.; Jain, A.K. Face recognition: Primates in the wild. In Proceedings of the IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS), Los Angeles, CA, USA, 22–25 October 2018; pp. 1–10.
- [20] Liu, J.; Kanazawa, A.; Jacobs, D.; Belhumeur, P. Dog breed classification using part localization. In Proceedings of the European Conference on Computer Vision, Florence, Italy, 7–13 October 2012; pp. 172–185.
- [21] Wang, X.; Ly, V.; Sorensen, S.; Kambhamettu, C. Dog breed classification via landmarks. In Proceedings of the IEEE International Conference on Image Processing (ICIP), Paris, France, 27–30 October 2014; pp. 5237–5241.
- [22] Hsu, D. Using Convolutional Neural Networks to Classify Dog Breeds. CS231n: Convolutional Neural Networks for Visual Recognition. 2015. Available online: http://cs231n.stanford.edu/reports/2015/pdfs/fcdh_FinalReport.pdf (accessed on 26 February 2021).
- [23] Ayanzadeh, A.; Vahidnia, S. Modified Deep Neural Networks for Dog Breeds Identification. Preprints 2018. [CrossRef]
- [24] Kumar, S.; Singh, S.K. Monitoring of pet animal in smart cities using animal biometrics. *Future Gener. Comput. Syst.* 2018, 83, 553–563. [CrossRef]
- [25] Moreira, T.P.; Perez, M.L.; de Oliveira Werneck, R.; Valle, E. Where is my puppy? retrieving lost dogs by facial features. *Multimed. Tools Appl.* 2017, 76, 15325–15340. [CrossRef]
- [26] Sermanet, P.; Eigen, D.; Zhang, X.; Mathieu, M.; Fergus, R.; LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv* 2013, arXiv:1312.6229.
- [27] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
- [28] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
- [29] URL: <https://github.com/kvsnoufal/Pytorch-FaceNet-DogDataset>

- [30] Computer Vision for Faces - Final Project, Doggie Smile, 16th October 2017 by Alzbeta Vlachynska,
- [31] SIBI (Sistem Isyarat Bahasa Indonesia) translation using Convolutional Neural Network (CNN), January 2020, IOP Conference Series Materials Science and Engineering 732:012082
- [32] Understanding Deep Convolutional Neural Networks, URL: <https://www.run.ai/guides/deep-learning-for-computer-vision/deep-convolutional-neural-networks/>, <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>,
- [33] Best Place to Learn Neural Network: Interactive Tensorflow Playground, Tinker with Neural Network in Your Browser! by Korkrid Akepanidaworn (Kyle) Korkrid Akepanidaworn (Kyle) Jan 15, 2019, URL: <https://kyleake.medium.com/technical-demo-visualize-neural-network-with-tensorflow-playground-9f6a1d8eb57a>
- [34] A comprehensive guide to facial recognition algorithms – part 2, URL: : <https://www.baseapp.com/computer-vision/a-comprehensive-guide-to-facial-recognition-algorithms-part-2/>
- [35] URL: https://gombru.github.io/2019/04/03/ranking_loss/
- [36] URL: <https://www.tech-quantum.com/implementing-drop-out-regularization-in-neural-networks/>
- [37] URL: <https://medium.com/@anuragrakesh11/activation-functions-6eb6914d18b6>
- [38] URL: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [39] URL: <https://xzz201920.medium.com/conv1d-conv2d-and-conv3d-8a59182c4d6>

Appendix A

Triplet Loss

x_i^a – an *anchor* example. In our context, it is a photograph of a dog's face.

x_i^p – a *positive* example that has the same identity as the anchor. It is a second picture of the same dog as the picture from the anchor example.

x_i^n – a *negative* example that represents a different entity. This would be an image of a second dog—a dog different than the dog represented by the anchor and positive examples.

for an embedding function $f(x) \in \mathbb{R}^d$ that embeds input data X into a d -dimensional vector, we want to satisfy this equation:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha \leq \|f(x_i^a) - f(x_i^n)\|_2^2, \quad \forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in T$$

Where $\|x\|_2^2$ operator is the square Euclidean Norm and the $[x]_+$ operator stands for $\max(0, x)$ – relu function. Below you can see the code that satisfies the above:

```
alpha = 0.3 #margin

def triplet(y_true, y_pred):
    a = y_pred[0::3] #anchor
    p = y_pred[1::3] #positive
    n = y_pred[2::3] #negative

    # Euclidean Norm:
    ap = K.sum(K.square(a - p), -1)
    an = K.sum(K.square(a - n), -1)

    return K.sum(tf.nn.relu(ap - an + alpha)) #relu: max(x,0)
```

This leads to the following loss function over the N possible triplets:

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

Appendix B

Our hybrid Solution welcomes two techniques to find the best possible results. At first we are going to compute Nearest Embeddings. A near embedding with our testing image is defined as the position of the dog images in the embedding space is smaller than a threshold. To find the appropriate threshold we calculate the distances between photos from the training that we know they belong to the same class, and the distances between two random different images. With this method we limit the possible outputs. Now we want the most similar photos from the nearest embeddings. Thus, we apply the agglomerative clustering aka kmeans to decide in which cluster we are going to place our test dog image. The following code is written in much more simple structure to understand the whole procedure.

Load a trained Network

```
model = tf.keras.models.load_model('{:s}/{:s}.{:d}.h5'.format(PATH_MODEL,
NET_NAME, EPOCH),custom_objects={'triplet': triplet, 'triplet_acc': triplet_acc})
```

1. Find Nearest Embedding Vectors

Find best threshold

```
best_t, best = findThresholdAccuracy(labels_test,filenames_test,model)
```

Load the dog the user has searched, the database of the found dogs and check the similarities between them. Return the likely similar array with the dogs.

```
test_dog = str(sys.argv[3])
database_images = str(sys.argv[4])
likely_similar = checkSimilarity(database_images, test_dog)
```

CheckSimilarity(): {

For every dog in the database folder we are testing, make a pair with the test dog image

For test_image in database_images:

```
pairs_test = [test_image] + [test_dog]
```

Find a prediction for their embeddings

```
predict_test=model.predict_generator(predict_generator(pairs_test, 32),
steps=np.ceil(len(pairs_test)/32))
```

Separate the pairs

```
emb1_test = predict_test[0::2]
```

```
emb2_test = predict_test[1::2]
```

Computes the distance between pairs

```
diff_test = np.square(emb1_test-emb2_test)
```

```
dist_test = np.sum(diff_test, 1)
```

```

        # if their distance is below the threshold or (+accuracy) , append to likely similar
        if( (dist_test<=best_t ) or (dist_test<=(best_t+best)) ):
            likely_similar.append(test_image)

    return likely_similar
} end of Check Similarity

# for each photo that is in likely_similar array save it in a directory
Save_Nearest_Embeddings(likely_similar):

    for i in range(len(likely_similar))

        file = likely_similar[i]
        print("Similar: ",file)
        image_name = dirname + "/" +str(i)+".jpg"
        img = Image.open(r""+str(file))
        img.save(image_name, 'JPEG')
        img.close()

# Save the filenames of the database dogs
filenames_verification=findLabels(PATH_MODEL, dirname)

```

2. Cluster the nearest embeddings

Cluster Data

```

clusterData(filenames_verification, SIZE, model)

clusterData():

#We need to check that all images are in jpeg format. We read all of our database images with
# sklearn library

h, w, c = SIZE
images_test = np.empty((len(filenames_test), h, w, c))
for i, f in enumerate(filenames_test):
    res = list(filter(f.endswith, suff_list)) != []
    if res:
        images_test[i] = skimage.io.imread(f)

# Need to make a prediction about the embedding vectors of the database images

predict = model.predict(images_test)

#Find the best number of clusters K:
#A list holds the SSE values for each k

sse = []
for k in range(1, length_file):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs).fit(predict)
    sse.append(kmeans.inertia_)

```

```

kl = KneeLocator(range(1, length_file), sse, curve="convex", direction="decreasing")

#Find the best K
print("Perfect k elbow = ", kl.elbow)

# Apply Kmeans to our data
kmeans = KMeans(n_clusters=kl.elbow, **kmeans_kwargs).fit(predict)

# Find the cluster that contains our test dog image
final_cluster = None
if not os.path.isdir(save_results_path):
    os.makedirs(save_results_path)
for i in range(len(images_cluster)):
    length = len(images_cluster[i])
    if length > 0:
        print('cluster %d: %s' % (i, labels_cluster[i]))
        exists = test_dog in labels_cluster[i]
        if exists == True:
            final_cluster = labels_cluster[i]
            return final_cluster

```