

Thesis Dissertation

---

**Leveraging Vision Transformers for Early  
Breast Cancer Detection: A Study on Thermal  
Infrared Imaging with the use of DINOv2**

---

**Dimitriana Georgiou**

**University of Cyprus**



**Computer Science Department**

**May 2025**

# Acknowledgements

Completing this thesis has been a journey of growth, learning and perseverance. Over the course of the four years I have spent studying Computer Science, I have gained invaluable knowledge that has shaped my ability to conduct research, analyse complex problems, and apply programming skills effectively. This thesis has challenged me in many ways, pushing me to refine my problem-solving approach, improve my time management, and develop a structured workflow to meet deadlines successfully. Even though this journey was far from easy, it has shaped me to the person I have become today and for that I am beyond grateful.

I would like to express my sincere gratitude to my advisor, Dr. Chris Christodoulou, for his guidance and support throughout this research. His expertise and feedback have been instrumental in shaping my understanding of the topic and ensuring the quality of my work.

I am also deeply grateful to Marios Pafitis, whose continuous assistance and dedication greatly contributed to the development of this thesis. As the founder of MammoCheck, a company that focuses on early breast cancer examinations at home using affordable thermal cameras, AI algorithms, and your smartphone, Marios played a crucial role in helping me collect the necessary data and refine the core idea of my research. His availability, insightful discussions, and willingness to provide guidance at every stage made a significant difference in my ability to navigate the challenges of this project. His support has been invaluable, and I truly appreciate the time and effort he devoted to helping me bring this thesis to completion.

Additionally, I would like to thank Valentinos Pariza for his help in understanding DINOv2, which was a key aspect of my research. His insights and explanations allowed me to grasp the technical complexities and apply them effectively.

Beyond the academic support, I am deeply thankful to my family for their unwavering encouragement and patience throughout this journey, especially to my mother Claire Kouppas, who had to listen to my unlimited complaints and frustration.

I also owe a very special debt of gratitude to my grandfather and grandmother, Andreas and Maro Kouppas, who generously provided me with the laptop on which this entire thesis was completed. My family's support, both material and emotional, has played a significant role in making this work possible.

To claim that the virtues of a piece of work are largely due to the help of others and that the sin of omission is very much one's own is a cliché. There is, however, a core truth in clichés.

**UNIVERSITY OF CYPRUS**  
**COMPUTER SCIENCE DEPARTMENT**

**Leveraging Vision Transformers for Early Breast Cancer Detection: A  
Study on Thermal Infrared Imaging with the use of Dinov2**

**DIMITRIANA GEORGIOU**

Supervisor

Dr. Chris Christodoulou

A thesis submitted in partial fulfillment of the requirements for the award  
of Bachelor's degree in Computer Science at the University of Cyprus

May 2025

## Abstract:

Breast cancer is one of the leading causes of mortality in women worldwide, making its early diagnosis critical for improving survival rates. Although mammography remains the established detection method, it presents significant limitations, such as false positives, high cost, and challenges in analysis using Convolutional Neural Networks (CNNs).

This paper proposes an alternative method for detecting breast cancer through thermal imaging, utilizing Vision Transformers (ViTs), and specifically DINOv2. ViTs outperform CNNs in analyzing spatial relationships and preserving information at a global level, making them particularly suitable for medical imaging. The purpose of this study is to apply DINOv2 to classify thermal images of breasts as healthy or cancerous, with the aim of improving early diagnosis.

For training and evaluation of the model, thermal breast images from two public databases were used: "A New Database for Mastology Research with Infrared Image" and "Thermal Infrared Breast Screening Database (TIBSDB)". From these, a total of 329 frontal images were selected (257 from healthy cases and 72 from cancer patients), while after removing duplicates, the final number of images reached 323. The images come from static and dynamic acquisition protocols, including information on vascular patterns and temperature fluctuations.

In the study, experiments were introduced using two different classifiers (classification heads): a linear classifier (Linear Head) and a multi-layer perceptron (MLP Head), in order to compare their performance in classifying thermal images. Additionally, the analysis includes training via Stratified K-Folds Cross-Validation, ensuring fair evaluation of the model.

The results showed that both classifiers achieved relatively high sensitivity, a key priority in medical screening, but the MLP head consistently outperformed the linear head in accuracy, specificity, and F1-score. Despite these promising outcomes, the number of false positives and false negatives remained too high for real-world clinical deployment. This highlights the need for further research to improve stability, reduce misclassifications, and enhance generalization.

Overall, this research presents a novel approach to breast cancer detection using DINOv2-based ViTs on thermal images. It offers a non-invasive, low-cost, and potentially more effective alternative to traditional screening methods, while also laying the groundwork for future improvements and more clinically viable AI solutions in medical imaging.

# Contents

<b>Chapter 1.....</b>	<b>1</b>
<b>Introduction.....</b>	<b>1</b>
<b>1.1    Breast Cancer: Significance of Early Detection and Its Impact on Diagnosis and Treatment .....</b>	<b>1</b>
<b>1.2    Past Attempts .....</b>	<b>3</b>
<b>1.2.1    Past Achievements with CNNs in Breast Cancer Detection: Achievements and Limitations: .....</b>	<b>3</b>
<b>1.2.2    Past Achievements with MobileNetV2 in Breast Cancer Detection: Limitations .....</b>	<b>5</b>
<b>1.2.3    Past Attempts using Vision Transformers for Breast Cancer Detection .....</b>	<b>6</b>
<b>Chapter 2.....</b>	<b>8</b>
<b>Background .....</b>	<b>8</b>
<b>2.1    Proposed Solution: Vision Transformers and DINOv2 for Breast Cancer Detection.....</b>	<b>8</b>
<b>2.1.1    Exploring DINOv2 With and Without Register Tokens .....</b>	<b>9</b>
<b>2.2    Vision Transformers.....</b>	<b>11</b>
<b>2.3    DINOv2.....</b>	<b>15</b>
<b>2.3.1    What is DINOv2 .....</b>	<b>18</b>
<b>2.3.2    Comparative Analysis: DINOv2 vs. Other Self-Supervised Learning Methods .....</b>	<b>20</b>
<b>2.3.3    Justification for Using DINOv2 in this Thesis.....</b>	<b>21</b>
<b>2.4    Classifier Architectures.....</b>	<b>21</b>
<b>2.4.1    Linear Classifier .....</b>	<b>22</b>
<b>2.4.2    Multi Layer Perceptron Classification (MLP) .....</b>	<b>23</b>
<b>Chapter 3.....</b>	<b>24</b>
<b>Dataset Handling .....</b>	<b>24</b>
<b>3.1    Dataset Overview.....</b>	<b>24</b>
<b>3.2    Dataset Size and Class Distribution .....</b>	<b>28</b>
<b>3.3    Image Dimensions and Format.....</b>	<b>29</b>
<b>3.4    Preprocessing for DINOv2 Compatibility.....</b>	<b>30</b>
<b>3.5    Dataset Splitting Strategy.....</b>	<b>33</b>
<b>3.6    Class Distribution Per Fold .....</b>	<b>35</b>

<b>Chapter 4.....</b>	<b>37</b>
<b>Implementation.....</b>	<b>37</b>
<b>4.1 Overview.....</b>	<b>37</b>
<b>4.2 Image Reading and Labelling.....</b>	<b>38</b>
<b>4.3 Image Preprocessing.....</b>	<b>38</b>
<b>4.4 Dataset Preparation – Splitting .....</b>	<b>39</b>
<b>4.5 Loading the DINOv2 ViT-s/14 Model .....</b>	<b>39</b>
<b>4.6 Feature Extraction Strategy .....</b>	<b>40</b>
<b>4.7 Classification Heads.....</b>	<b>40</b>
<b>4.7.1 Linear Classification Head .....</b>	<b>41</b>
<b>4.7.2 MLP Classification Head .....</b>	<b>47</b>
<b>4.7.3 MLP Head: Thresholding and Training Constraints Matched to Linear         Baseline .....</b>	<b>53</b>
<b>Chapter 5.....</b>	<b>57</b>
<b>Experiments, Results &amp; Discussion .....</b>	<b>57</b>
<b>5.1 Overview of Evaluation Setup .....</b>	<b>57</b>
<b>5.2 Linear Classification Head Results .....</b>	<b>61</b>
<b>5.2.1 Performance at 50 Epochs.....</b>	<b>61</b>
<b>5.2.2 Performance at 150 Epochs .....</b>	<b>76</b>
<b>5.2.3 Performance at 30 Epochs.....</b>	<b>82</b>
<b>5.2.4 Analysis of Misclassified Cases Affecting Sensitivity with Linear Head .....</b>	<b>88</b>
<b>5.2.5 Hyperparameter and Architectural Exploration for the Linear Head .....</b>	<b>92</b>
<b>5.3 MLP Classification Head Results .....</b>	<b>94</b>
<b>5.3.1 Performance at 50 Epochs.....</b>	<b>94</b>
<b>5.3.2 Performance at 150 Epochs .....</b>	<b>112</b>
<b>5.3.3 Performance at 30 Epochs.....</b>	<b>117</b>
<b>5.3.4 Analysis of Misclassified Cases Affecting Sensitivity with MLP Head.....</b>	<b>123</b>
<b>5.3.5 Hyperparameter and Architectural Exploration for the MLP Head.....</b>	<b>126</b>
<b>5.4 Modified MLP Head Results: Aligned with Linear Head Constraints .....</b>	<b>130</b>
<b>5.4.1 Performance at 50 Epochs.....</b>	<b>130</b>
<b>5.4.2 Comparison Between Original and Aligned MLP Head Results .....</b>	<b>135</b>
<b>5.5 Comparative Analysis: Linear Classification vs MLP Classification at 50     Epochs 140</b>	
<b>5.6 Results and Discussion .....</b>	<b>154</b>
<b>5.6.1 Generalization and Model Behaviour.....</b>	<b>154</b>

5.6.2	Sensitivity vs Specificity Trade Off .....	156
5.6.3	Per-Fold Performance Instability .....	157
5.6.4	Clinical Implications .....	159
5.7	Comparative Summary to Teachable Machine Model .....	161
5.8	Further Validation Using a Balanced Dataset .....	164
5.8.1	Comparative Evaluation: Balanced vs. Unbalanced Dataset .....	167
Chapter 6.....		169
Conclusion and Future work .....		169
6.1	Summary of Findings.....	169
6.2	Limitations .....	170
6.3	Future Work .....	171
7	Appendix .....	172
References :	.....	193

## Acronyms:

CNNs : Convolutional Neural Networks  
ViTs: Vision Transformers  
TIBSDB: Thermal Infrared Breast Screening Database  
MLP : Multi-Layer Perceptron  
EU: European Union  
CAD: Computer-Aided Diagnosis  
DINOv2: Distillation with NO labels version  
AI: Artificial Intelligence  
SSL: Self Supervised Learning  
iBot: Image BERT pretext task  
NeCo: Neighbour Consistency Optimization  
CLS: Class Token  
EMA: Exponential Moving Average  
FSDP: Fully Sharded Data Parallelism  
ROI: Region of Interest  
LVD-142M: Large Visual Dataset of 142 million curated images  
BCE: Binary Cross-Entropy  
GPU: Graphics Processing Unit  
CSV: Comma-Separated Values  
xFormers: Efficient Transformer operations library  
KoLeo: Kozachenko-Leonenko Regularization  
AAT: American Academy of Thermology  
MAE: Masked Autoencoder  
RGB: Red , Green , Blue  
CV: Cross Validation  
AUC: Area Under the Curve  
DMR: Database for Mastology Research  
VGG: Visual Geometry Group  
UCY: University of Cyprus  
TPR: True Positive Rates  
FPR: False Positive Rates  
TP: True Positive  
FP: False Positive  
TN: True Negative  
FN: False Negative  
ReLu: Rectified Linear Unit  
GELU: Gaussian Error Linear Unit  
MHSA: Multi-Head Self-Attention  
LayerNorm: Layer Normalization  
MemEffAttention: Memory-Efficient Attention



# Chapter 1

## Introduction

---

1.1 Breast Cancer: Significance of Early Detection and Its Impact on Diagnosis

1.2 Past Attempts

1.2.1 Past Achievements with CNNs in Breast Cancer Detection: Achievements and Limitations

1.2.2 Achievements with MobileNetV2 in Breast Cancer Detection: Limitations

1.2.3 Past Attempts using Vision Transformers for Breast Cancer Detection

---

### **1.1 Breast Cancer: Significance of Early Detection and Its Impact on Diagnosis and Treatment**

Breast cancer is a malignant tumour that originates in the cells of the breast representing the most commonly diagnosed cancer among women worldwide, accounting for 30% of all new female cancer cases annually <sup>[1]</sup> and responsible for one in six cancer deaths globally<sup>[2]</sup>. In 2025, it is estimated that approximately 316,950 women in the United States will be diagnosed with invasive breast cancer <sup>[3]</sup>. In the European Union (EU), the incidence of breast cancer was estimated at 355,500 cases in 2020, with one in eleven women developing the disease before the age of 74 <sup>[2]</sup>.

In Cyprus, breast cancer possesses a significant health concern. The country aligns with the broader EU statistics, reflecting the high prevalence of this disease among women. Notably, approximately 21% of breast cancer cases in Europe occur in women under 50, underscoring the importance of awareness and screening across various age groups <sup>[2]</sup>.

Early detection of breast cancer is crucial for improving survival rates and reducing the severity of treatment <sup>[2]</sup>. When diagnosed at an early stage, the five-year survival

rate is significantly higher compared to late-stage detection <sup>[4]</sup>. Early-stage breast cancers are often confined to the breast tissue and can be effectively treated with surgical interventions, reducing the need for more aggressive therapies. Regular screening methods, such as mammography and breast self-examinations, play a vital role in identifying the tumor(s) before they progress to advanced stages <sup>[4]</sup>. Breast thermography is a non-invasive and contact-free technique that does not involve radiation or painful breast compression. Diagnosing breast cancer typically requires expert radiologists and pathologists, a process that can be time-consuming. Their conclusions are based on various visual features, which may differ from one individual to another <sup>[5]</sup>. Additionally, the widespread adoption of mammography screening has contributed to a rise in breast cancer diagnoses; however, it has also significantly lowered mortality rates in many countries by enabling timely intervention. <sup>[2]</sup>

Despite advancements in treatment, unfortunately breast cancer remains a leading cause of cancer-related deaths among women. In Europe, it accounts for one in six cancer fatalities <sup>[2]</sup>.

All of these statistics mentioned above, highlight the aggressive nature of certain breast cancer subtypes and the challenges associated with treating advanced-stage disease. Therefore, emphasizing early detection strategies especially at an early stage is essential to reduce mortality rates, improve the quality of life of those affected <sup>[4]</sup> as well as to increase the likelihood of successful treatment, preventing cancer from spreading to vital tissues and organs <sup>[2]</sup>.

The remainder of this thesis is structured as follows. The next chapter provides an overview of relevant background knowledge, including Vision Transformers (ViTs), the DINOv2 architecture, and the classification strategies employed. This is followed by a chapter detailing the dataset characteristics, acquisition protocols, preprocessing steps, and data splitting methodology. The implementation chapter then outlines the complete experimental pipeline, including feature extraction, model design, and training procedures. Subsequently, the experimental results are presented and analyzed, with comparisons between classifier performance and an in-

depth evaluation of key metrics. The thesis concludes with a summary of findings, a discussion of limitations, and suggestions for future research directions.

## **1.2 Past Attempts**

### **1.2.1 Past Achievements with CNNs in Breast Cancer Detection: Achievements and Limitations:**

Convolutional Neural Networks (CNNs) have been widely employed for breast cancer detection, particularly in mammographic image analysis. Their ability to automatically extract hierarchical features has significantly advanced computer-aided diagnosis (CAD) systems, whilst also reducing the workload of radiologists and improving detection accuracy <sup>[1]</sup>. Various CNN architectures have been tested in this field, with models such as ResNet and EfficientNet demonstrating promising performance in classifying malignant and benign breast lesions <sup>[6]</sup>. Despite these advancements, CNNs still face several limitations that affect their real-world applicability.

One of the primary challenges with CNNs is their high computational cost. Due to the multiple convolutional layers and complex feature extraction processes, CNN models require substantial processing power making them inefficient for deployment in resource-limited clinical settings <sup>[1]</sup>. Additionally, CNN-based models often use a patch-based approach, where sections of mammograms are analyzed separately. While this method improves local feature detection, it can lead to a loss of global contextual information, resulting in a significantly high rate of false positives and false negatives <sup>[6]</sup>. Furthermore, CNNs struggle with generalization across different datasets due to variations in imaging protocols and patient demographics, making their performance highly dependent on large and diverse training datasets <sup>[1]</sup>.

Another significant drawback of CNNs is their sensitivity to image quality. Poor contrast, noise, and variations in mammogram scans can negatively impact model accuracy, requiring extensive preprocessing techniques to improve image clarity <sup>[1]</sup>. Moreover, CNNs often require large amounts of annotated data to achieve robust performance, but medical image datasets, particularly for rare breast cancer subtypes, are often limited. This leads to imbalanced training data, where CNNs

may favor the majority class, reducing sensitivity to any rare malignant cases <sup>[6]</sup> . These limitations have prompted researchers to explore alternative deep learning models such as Vision Transformers (ViTs), which can capture global dependencies more effectively and address some of the fundamental weaknesses of CNNs <sup>[1]</sup> .

A notable recent contribution is the work by Abunasser et al. <sup>[7]</sup> , who proposed a deep learning-based framework using a custom CNN architecture called BCCNN for the detection and classification of breast cancer. Their model was trained on a histopathological image dataset sourced from Kaggle and further enhanced using Generative Adversarial Networks (GANs) to synthetically balance the classes. This approach aimed to address a common problem in medical datasets , which is class imbalance. Imbalance in dataset classes, can significantly hold back model performance in distinguishing between rare malignant subtypes and more common benign cases.

The BCCNN architecture was evaluated across various image magnifications (40×, 100×, 200×, and 400×) and compared to five widely-used pre-trained CNN models: Xception, InceptionV3, VGG16, MobileNet, and ResNet50. The results demonstrated that BCCNN outperformed these models in key metrics such as precision, recall, and F1-score, achieving a top F1-score of 98.28% which is an extremely high percentage especially in comparison to the results of the pre-trained CNN models. This performance highlights the potential benefits of task-specific custom architectures over general-purpose pre-trained models especially when combined with data augmentation strategies like GANs <sup>[7]</sup> .

Moreover, the study by Abunasser et al. emphasized the impact of multi-class classification in breast cancer detection. Unlike binary classification systems that only distinguish between benign and malignant tumors, BCCNN was trained to classify eight distinct categories, including multiple malignant and benign subtypes making it a more general model. This kind of highly detailed classification, mirrors clinical reality more closely and provides richer diagnostic information. It also underlines a growing trend in deep learning for medicine: moving from binary outputs to more nuanced, clinically relevant predictions.

However, even with these achievements, the study reflects broader challenges within CNN-based systems <sup>[1] [7]</sup>. The reliance on annotated data, the difficulty in generalizing across diverse clinical datasets, and the need for high computational resources still remain persistent limitations. Additionally, while BCCNN showed excellent performance on the dataset it was trained and tested on, the authors acknowledge that further validation on external datasets and real clinical settings is essential for proving its robustness <sup>[7]</sup>. Therefore, taking into consideration these challenges, limitations and weaknesses that CNN models face, it is extremely important to introduce new methods using different models, such as ViTs, that potentially have the ability to outperform the existing ones.

### **1.2.2 Past Achievements with MobileNetV2 in Breast Cancer Detection: Limitations**

Beyond CNNs, MobileNetV2 has also been investigated as a potential architecture for early breast cancer detection. MobileNetV2 is a lightweight deep learning model which has supposedly been designed for efficient computation, particularly in mobile and edge devices. It utilizes depth wise separable convolutions to reduce the number of parameters and computational cost while maintaining reasonable accuracy in image classification tasks <sup>[8]</sup>. However, despite these advantages, MobileNetV2 has demonstrated several limitations that impact its effectiveness in medical imaging applications <sup>[8]</sup>.

One of the key challenges faced by MobileNetV2 is overfitting, which is particularly noticeable when trained on limited datasets. This occurrence has led to poor generalization when applied to new mammograms <sup>[8]</sup>. Additionally, its feature extraction capability is not as strong as other deep learning models which affects its ability to detect subtle abnormalities in the breast tissue, potentially increasing false negatives in early stage cancer detection <sup>[8]</sup>. Another drawback is its lower validation accuracy, especially when compared to other architectures like MobileNetV1 which therefore, suggests instability in training and classification performance <sup>[8]</sup>.

These limitations underline the importance of exploring more advanced models that can provide better feature representation, improved accuracy, and greater generalizability for breast cancer detection.

### **1.2.3 Past Attempts using Vision Transformers for Breast Cancer Detection**

ViTs have emerged as a powerful alternative to CNNs in image classification tasks including medical imaging. In a recent study, Muthusamy et al. applied a ViT architecture for classifying breast cancer from thermal images <sup>[5]</sup>. The ViT model was trained from scratch using thermal breast thermograms all taken from the DMR dataset, with images of individuals aged between 29 and 85 captured using a high-resolution FLIR SC-620 infrared camera.

The main approach involved dividing each thermal image into smaller patches ( $16 \times 16$  and  $32 \times 32$ ), which were then flattened and fed into the Transformer encoder along with a learnable classification token. This allowed the self-attention mechanism to focus on the most relevant parts of the input image during classification. The ViT-Base model used in the study consisted of 12 encoder layers with 12 attention heads, a 768-dimensional embedding size, and a 3072-dimensional feed-forward layer. The network was optimized using the Adam optimizer with a learning rate of  $1e-2$  (which equals to  $1 \times 10^{-2}$ ) and 10% of test data was used for validation purposes.

The experiment's results demonstrated that the ViT model achieved impressive performance reaching a maximum accuracy of 95.78% and an AUC which is a metric that shows the overall performance of the ViTs, of 0.957 when using  $32 \times 32$  patches and a 90/10 train-test split. This performance was comparable to and in some cases even exceeded, that of CNN-based models such as ResNet and VGG variants. However, the authors also highlighted key limitations of the approach.

More specifically, ViTs require large datasets to fully leverage their capacity and due to the relatively small size of the DMR dataset (460 cancerous and 490 healthy patients), a significant portion of the available data had to be used for training. This

raises concerns about overfitting and generalizability. Furthermore, the computational demands of training ViTs from scratch can pose challenges, especially when dealing with high-resolution medical images or limited hardware resources.

Despite these challenges, the study provides strong evidence for the applicability and potential use of Transformer-based architectures in the medical domain. The ability of ViTs to model long-range dependencies and attend to crucial features in thermal imaging makes them a compelling choice for breast cancer detection.

The aforementioned study, served as a key motivation for selecting ViTs as the foundational model in this thesis. It demonstrated the potential of attention-based architectures to yield high classification performance in breast cancer detection tasks, which aligns with the goals of this research.

# Chapter 2

## Background

---

### 2.1 Proposed Solution: Vision Transformers and DINOv2 for Breast Cancer Detection

#### 2.1.1 Exploring DINOv2 With and Without Register Tokens

#### 2.2 Vision Transformers

#### 2.3 DINOv2

##### 2.3.1 What is DINOv2

##### 2.3.2 Comparative Analysis: DINOv2 vs. Other Self-Supervised Learning Methods

##### 2.3.3 Justification for Using DINOv2 in this Thesis

#### 2.4 Classifier Architectures

##### 2.4.1 Linear Classifier

##### 2.4.2 MLP Classifier

---

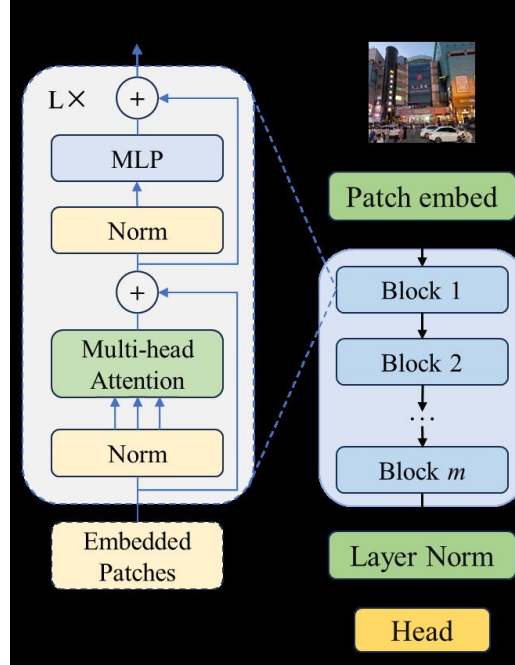
### **2.1 Proposed Solution: Vision Transformers and DINOv2 for Breast Cancer Detection**

The goal of this thesis is to explore the application of ViTs and DINOv2 in breast cancer detection. Traditional deep learning approaches, particularly CNNs, have played a significant role in medical imaging tasks, including mammography analysis. However, CNNs come with several inherent limitations, such as a restricted receptive field, sensitivity to spatial locality, and difficulty in capturing long-range dependencies <sup>[1]</sup>. These drawbacks hinder their effectiveness in detecting subtle patterns associated with early-stage breast cancer. To address these challenges, this study leverages ViTs, a novel deep learning architecture that utilizes self-attention mechanisms, and DINOv2, a self-supervised learning approach that enhances feature extraction without labeled data <sup>[9]</sup>.

By adopting ViTs and DINOv2, we aim to improve the generalization, robustness, and interpretability of breast cancer detection models. Unlike CNNs, which primarily focus on local features through convolutional layers, ViTs process entire images as sequences of patches, enabling them to capture global context and long-range dependencies <sup>[6]</sup>.



This shift in image processing has demonstrated significant improvements in classification tasks, particularly in complex datasets such as medical imaging. Additionally, DINOv2 eliminates the dependency on large-scale annotated datasets, making it an ideal choice for medical applications where labelled data is limited <sup>[9]</sup>.



**Figure 2.1:** Structural diagram of the DINOv2 model. The input image is divided into patches, which are embedded and passed through a stack of transformer blocks comprising multi-head self-attention, normalization layers, and MLP layers. This architecture builds on Vision Transformers and is trained using a self-supervised approach <sup>[10]</sup>.

### 2.1.1 Exploring DINOv2 With and Without Register Tokens

Building upon the adoption of DINOv2 for breast cancer detection, this section delves into a recent architectural enhancement to the DINOv2 framework. This enhancement includes register tokens, and the following section evaluates its performance within the context of this thesis.

DINOv2 as discussed, is a self-supervised vision transformers known for its strong transferable features across a variety of vision tasks <sup>[9],[11]</sup>. However, Darcet et.al (2024) recently identified a structural issue in the standard DINOv2 architecture: the emergence of outlier tokens, which are patch tokens with unusually high activation norms often located in uninformative background regions <sup>[11]</sup>. These outliers tend to encode global scene-level information in a manner that can interfere with the local semantic interpretation of individual image patches.

To mitigate this, the authors proposed the integration of register tokens, which are a set of learnable tokens appended to the input sequence during training. These tokens act as a form of persistent memory, explicitly designed to absorb global information and internal computations that would otherwise manifest as outlier behaviour in patch tokens. This modification not only stabilizes the training process but also yields smoother attention maps and enhances performance on tasks such as object discovery, segmentation, and dense prediction <sup>[11]</sup>.

Despite the theoretical and empirical benefits shown in the article <sup>[11]</sup>, this thesis employs the standard DINOv2 architecture without register tokens. This decision was based on careful experimental evaluation during the development of two downstream classifiers for binary classification of breast thermographic images.

While models utilizing register tokens achieved slightly lower average training and validation losses, they consistently underperformed in terms of sensitivity and specificity, two metrics that are critical in medical diagnosis <sup>[5]</sup>, especially when the cost of a false negative is high. Sensitivity, which reflects the model's ability to correctly identify sick cases, is prioritized in this work due to the potential consequences of missing a cancer diagnosis. As shown in the graphs comparing the performance metrics of the model with and without registers, sensitivity (see Appendix ii.) was notably better for the model without registers. Specificity was also more balanced for the model *without* registers (see Appendix iii.), and accuracy was better as well (see Appendix iv.). The only consistent advantage observed for the model *with* registers was that it achieved lower loss values across all folds, each remaining under 1.0, whereas the loss values for the model *without* registers were much higher (see Appendix v.).

Moreover, the improvements offered by register tokens in "Vision Transformers Need Registers," <sup>[11]</sup> were most significant in dense prediction and object discovery tasks, which differ in nature from the binary classification problem targeted in this thesis. The addition of registers appeared to alter the internal representation of patch tokens in ways that were not beneficial for maximizing per-sample diagnostic accuracy in this medical context.

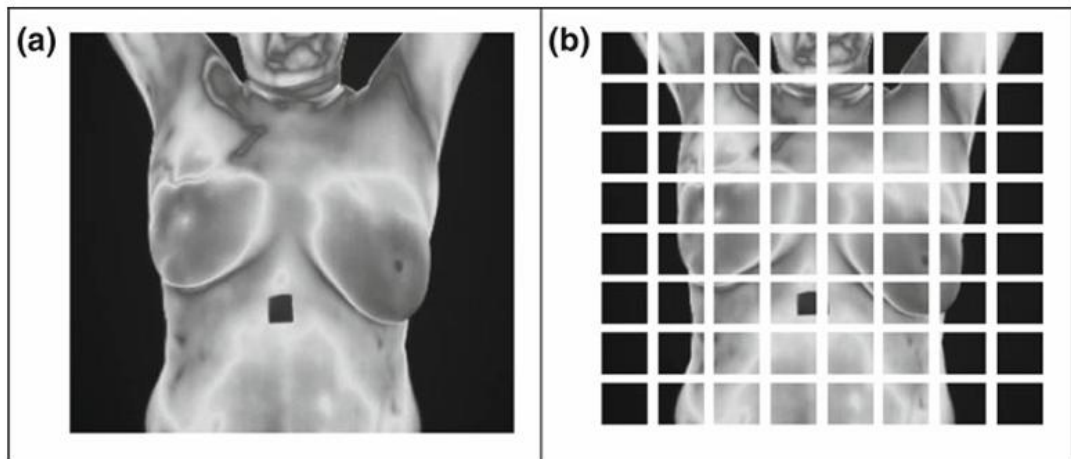
Consequently, the choice to utilize DINOv2 without registers ensures alignment with the core objective of this work which is achieving high sensitivity and specificity in breast cancer detection from thermal imaging. This aligns with prior findings that

architectural modifications must be evaluated in task-specific contexts, as performance gains in one domain may not generalize universally <sup>[11]</sup> .

## 2.2 Vision Transformers

ViTs are a groundbreaking class of deep learning models that apply the Transformer architecture, originally developed for natural language processing (NLP), to computer vision tasks <sup>[6]</sup> . While CNNs have long been the dominant architecture for visual tasks, their hierarchical and spatially localized nature makes it challenging to capture long-range dependencies and global context. In contrast, ViTs treat an image as a sequence of patches and use self-attention mechanisms to model relationships between these patches across the entire image <sup>[5]</sup> .

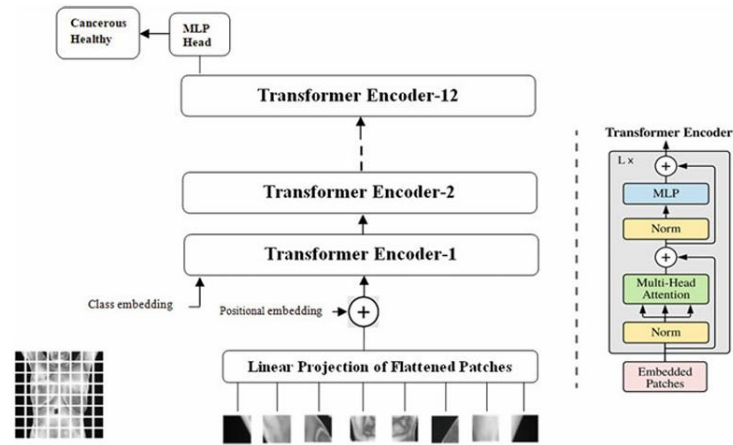
Unlike CNNs, which utilize fixed-size convolutional filters to extract local features, ViTs divide the input image into fixed-size non-overlapping patches, typically  $16 \times 16$  or  $32 \times 32$  pixels. Each patch is flattened into a 1D vector, then linearly projected into a higher-dimensional embedding space. These patch embeddings, along with a learnable classification token (CLS token) and positional encodings, are input to a Transformer encoder composed of multi-head self-attention layers and feedforward neural networks, following the same architecture used in NLP models like BERT <sup>[6][5]</sup> .



<sup>[5]</sup> **Figure 2.2:** Patch extraction process applied to a breast thermogram for a ViT input.  
**a.** Original thermal image of the breast **b.** The same image divided into  $32 \times 32$  pixel patches as used in ViT architectures.

Figure 2.2 provides a visual example of how an input thermogram is divided into fixed-size patches ( $32 \times 32$ ) before being processed by the ViT <sup>[5]</sup> . This patching mechanism

enables ViTs to model long-range dependencies across the image, which is particularly valuable in identifying subtle patterns in medical images such as breast thermograms.



[5] **Figure 2.3:** ViT architecture for classifying breast thermograms.

The ViT pipeline utilized for breast thermogram classification is illustrated in Figure 2.3. Initially, the input thermal image is divided into non-overlapping patches, in this particular case of size  $32 \times 32$  pixels, to convert the image into a sequence of smaller image tokens. Each patch is then flattened into a 1D vector and passed through a linear projection layer to obtain patch embeddings in a higher-dimensional feature space. To retain spatial ordering, positional embeddings are added to each patch embedding, and a special [CLS] token is prepended to the sequence. This [CLS] token is designed to aggregate global information across the entire image during training.

The resulting sequence of embeddings is processed by a series of Transformer Encoder blocks, each consisting of multi-head self-attention layers and feedforward neural networks, interleaved with normalization layers and residual connections. As the sequence passes through these encoder layers, self-attention allows the model to capture long-range dependencies and contextual relationships between patches, which is considered a critical advantage over CNNs for subtle pattern detection in breast thermograms.

Finally, the transformed representation of the [CLS] token, which now encodes a comprehensive understanding of the entire image, is passed through a Multi Layer

Perceptron (MLP) classification head. This head outputs the final prediction, indicating whether the thermogram corresponds to a cancerous or healthy breast.

The attention mechanism in ViTs allows every patch to attend to every other patch, enabling global reasoning from the very first layer. This contrasts sharply with CNNs, where the receptive field grows slowly with depth, and global context is only captured in deeper layers. As a result, ViTs can effectively model spatial dependencies between distant parts of an image, which is especially important in medical imaging applications like breast cancer detection, where subtle and dispersed patterns may indicate pathology [5] .

ViTs also introduce greater interpretability compared to CNNs. Attention maps generated by the Transformer layers can highlight which regions of the image the model focuses on when making decisions, offering explainable AI (XAI) potential in clinical scenarios [5] .

Aspect	CNNs	Vision Transformers
Locality Bias	Strong (via convolution)	None (learns spatial relationships)
Global Context	Weak unless deep layers are used	Strong due to self-attention
Data Requirement	Moderate	High (can overfit on relatively small datasets, needs pretraining)
Interoperability	Limited	Higher (attention maps can be visualized)

[5],[9] **Table 2.1:** Comparative Analysis of CNNs and ViTs in the context of image understanding.

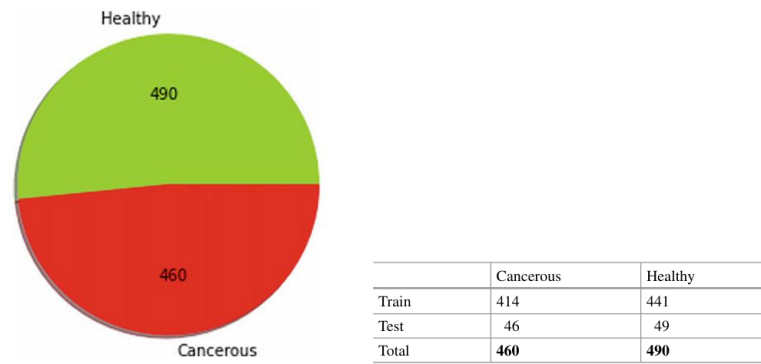
While CNNs have been dominant in medical image analysis, they are inherently limited by their localized receptive fields and lack of global context awareness in early layers. ViTs by contrast, leverage self-attention to enable each patch to interact with all others from the very first layer, facilitating global reasoning and enhanced interpretability through attention maps [9],[5] .

A side-by-side comparison of these two architectures is presented in Table 2.1, summarizing the key differences relevant to medical imaging applications such as breast cancer detection.

Moreover, transfer learning plays a crucial role in the application of ViTs to medical domains. Since ViTs are typically data-hungry, training them from scratch on limited

datasets (like thermal breast images) often leads to overfitting. To overcome this, pretrained ViTs on large datasets such as ImageNet are used and then fine-tuned on the target domain <sup>[5]</sup>. This approach retains the general visual knowledge learned during pretraining and helps adapt it to specialized medical tasks. In particular, self-supervised pretrained models, such as DINOv2, have been shown to improve representation quality without requiring labeled data <sup>[9]</sup>.

In the study "Vision Transformers for Breast Cancer Classification from Thermal Images" <sup>[5]</sup>, ViTs achieved superior performance in identifying cancerous tissues from thermal images compared to CNNs. This performance is attributed to their ability to model subtle asymmetries, vascular changes, and diffuse thermal anomalies, all considered features that are not always locally prominent and might be missed by CNNs.



<sup>[5]</sup> **Fig. 2.4:** Distribution of Thermograms Across Cancerous and Healthy Classes

<sup>[5]</sup> **Table 2.2:** Table representation of Thermograms Across Cancerous and Healthy Cases

The Transformer encoder in ViTs is built using:

- Multi-head self-attention layers: Each attention head computes a weighted representation of all patches, allowing the model to attend to multiple aspects of the input in parallel.
- Feedforward neural networks: Apply non-linear transformations to the output of the attention layers.
- Residual connections and layer normalization: Help with gradient flow and model stability <sup>[5]</sup>.

Mathematically, each self-attention layer in a ViT computes the output as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{dk}} \right) \mathbf{V}$$

Where:

- $\mathbf{Q}, \mathbf{K}, \mathbf{V}$  are the query, key, and value matrices derived from the input embeddings,
- $dk$  is the dimension of the key vectors.

The result is that each patch can gather information from all others, regardless of spatial distance, making ViTs especially powerful for medical diagnostics where global coherence and symmetry analysis are crucial.

In conclusion, ViTs represent a significant advancement in medical image analysis. Their self-attention mechanism, ability to handle global dependencies, and flexibility to model irregular, non-local patterns make them an ideal architecture for detecting subtle and dispersed anomalies in breast thermal images, which is a challenge where conventional CNNs fall short <sup>[5]</sup>.

### 2.3 DINOv2

In this thesis, transfer learning was applied to perform classification on thermal breast images. Transfer learning is a machine learning technique in which a model trained on one task is reused or adapted to a different but related task, allowing for efficient use of learned representations when labelled data is limited <sup>[12]</sup>. According to Pan and Yang <sup>[12]</sup>, transfer learning is particularly effective when the source domain has abundant data but the target domain lacks labelled examples, which is a scenario that aligns with this thesis, where we apply a DINOv2 model pretrained on a large-scale dataset (LVD-142M) to a small, domain-specific task involving just 323 thermal breast images, where acquiring extensive labelled data is difficult or impractical.

For the purpose of this particular project, the DINOv2 ViT-s/14 model was employed, which is a 21 million parameter ViT released by Meta AI, trained with a self-supervised learning (SSL) approach that requires no human labels or captions <sup>[13]</sup>. DINOv2 stands out among recent vision foundation models for its ability to produce robust, transferable frozen features that generalize well across a variety of tasks, including classification,

segmentation, instance retrieval, and depth estimation, all without task-specific fine-tuning <sup>[13]</sup> .

### **Model variant used:**

In this thesis, the DINOv2 ViT-s/14 model was specifically used, which is the smallest in the DINOv2 family and contains approximately 21 million parameters <sup>[13]</sup> . The model uses 12 transformer blocks, each with 384-dimensional token embeddings, and a patch size of 14×14. The input images are resized to 224×224, which aligns with the model's architecture. Internally, this resolution yields 16×16 grid patches, producing 256 patch tokens per image. A special [CLS] token is also prepended, resulting in a total sequence length of 257 per image , all of which are embedded into a 384-dimensional space.

The model begins with a Conv2D-based patch embedding layer and passes these tokens through a transformer stack. Each transformer block consist of Multi-Head Self-Attention (MHSA) layers, with 6 attention heads in ViT-s/14, followed by Gaussian Error Linear Unit (GELU) activation, Layer Normalization (LayerNorm) and a feedforward MLP.

GELU is a smooth non-linear activation function, used instead of ReLu, It enables better gradient flow, and it is defined as:

$$\text{GELU}(\mathbf{x}) = \mathbf{x} \cdot \Phi(\mathbf{x})$$

Where  $\Phi(\mathbf{x})$  is the cumulative distribution function of the standard normal distribution <sup>[14]</sup> . GELU has no tuneable hyperparameters and it is widely used in Transformer models for its smoother activation curve.

LayerNorm is applied before attention and MLP sub-layers. It standardizes the inputs across the features for each token as opposed to across the batch, thus stabilizing training <sup>[15]</sup> . It has two hyperparameters: a learnable scale and bias, both applied after normalization.

Memory Efficient Attention (MemEffAttention) is a customized version of FlashAttention <sup>[13]</sup> , used in DINOv2 to speed up computation and reduce memory



usage during the attention operation. While not a conceptual change to the attention mechanism itself, it enables efficient training especially for large-scale ViT models. It relies on hardware-aligned parameters for optimized matrix operations.

No classification head is used as the model outputs the raw feature representations directly. For the purpose of this thesis, custom classification heads (Linear and MLP) are applied on top of these frozen features.

The model is downloaded using the following command:

```
model = torch.hub.load('facebookresearch/dinov2', 'dinov2_vits14')
```

This automatically downloads the pretrained model weights from Meta AI’s official DINOv2 ViT-s/14 checkpoint: `dinov2_vits14_pretrain.pth`. These weights were trained on the LVD-142M dataset using the DINOv2 self-supervised method <sup>[13]</sup>. The `.pth` file contains the full set of trained parameters, including weights and biases for:

- All 12 transformer blocks
- The Conv2d patch embedding layer
- Multi-head attention layers
- MLP (feedforward) layers
- LayerNorm layers

These weights are used in this thesis as a frozen backbone, as they are not modified or updated during training.

On top of this, a NeCo-based post-training enhancement <sup>[16]</sup> was applied, that encourages local spatial consistency in the extracted features by aligning neighbouring patches in the latent space. This post-processing step occurs externally, without modifying the original DINOv2 parameters, by refining the spatial alignment of patch-level features.

The NeCo refinement applied in this thesis is based on the official open-source implementation provided by Valentin Pariz <sup>[16]</sup>. Specifically, the “Student” model checkpoint from the NeCo repository was downloaded and used for feature extraction, while maintaining the DINOv2 ViT-s/14 weights in a frozen state.

### 2.3.1 What is DINOv2

DINOv2 (DIstillation with NO labels v2) is a discriminative self-supervised learning framework developed by Meta AI. It leverages the Vision Transformer (ViT) architecture previously discussed in Section 2.2 and is trained using a self-distillation framework without labels, where a student network learns to match the output representations of a teacher network. The teacher is not trained directly; instead, it is updated as an exponential moving average (EMA) of the student’s weights, which stabilizes the learning targets and prevents collapse (i.e., trivial representations). During training, both networks receive differently augmented views of the same image, and the student is encouraged to produce similar high-level features as the teacher. This process enables the model to learn semantically consistent representations across augmentations, making it highly effective for self-supervised learning <sup>[13]</sup>.

DINOv2 employs a combination of complementary training objectives to learn robust and semantically meaningful visual representations. These objectives not only serve individual purposes but also interact synergistically to guide the model toward both global and local understanding of images:

- **Global Consistency (DINO Loss):**

The DINO loss encourages the student model to produce feature representations that are consistent with those of the teacher, even when the input views are drastically different (e.g., different crops or augmentations of the same image). This global alignment helps the model focus on the semantic content of the image rather than low-level details. By enforcing similarity in the output distributions of the student and teacher, the DINO loss fosters the learning of high-level, transformation-invariant features.

- **Local Consistency (iBOT Loss):**

While the DINO loss focuses on global image-level alignment, the iBOT (Information Bottleneck Objective for Transformers) loss introduces a local, patch-level consistency objective. In this setup, the student receives masked versions of the image (similar to masked language modeling in NLP), and is trained to predict the representations of the unmasked teacher patches. This encourages the model to learn fine-grained spatial details and improves its

ability to reason about local context. iBOT complements the DINO loss by strengthening the model's understanding of object parts and spatial structure <sup>[13]</sup> .

- **Sinkhorn-Knopp Centering:**

One common challenge in self-supervised learning is the collapse of representations, where all inputs are mapped to similar or identical features. Sinkhorn-Knopp centring addresses this by normalizing the output distributions of the teacher using a form of optimal transport. This ensures that feature vectors are balanced across batches and encourages the use of the full representational capacity of the model. It acts as a form of regularization that stabilizes training and preserves diversity in the learned embeddings <sup>[13]</sup> .

- **KoLeo Regularization:**

The KoLeo (Kozachenko–Leonenko) regularization term promotes uniform distribution of feature vectors in the latent space. It explicitly penalizes feature collapse by encouraging the model to spread out representations, thereby increasing entropy and improving generalization. This complements Sinkhorn-Knopp centring by providing an additional mechanism to preserve diversity and avoid degenerate solutions <sup>[13]</sup> .

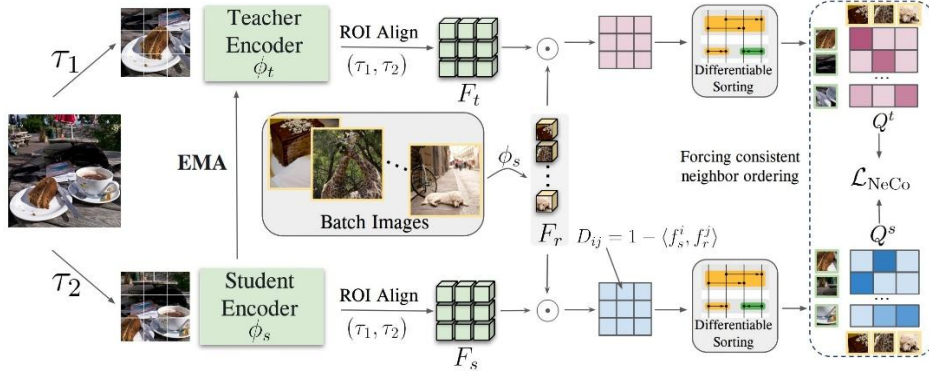
Together, these training objectives interact to achieve both **semantic alignment** and **feature diversity**:

The DINO and iBOT losses focus on making the representations semantically meaningful across different scales (global and local), while Sinkhorn-Knopp and KoLeo regularization ensure that the learned features are diverse, stable, and well-structured in the latent space.

These mechanisms allow DINOv2 to learn semantically meaningful representations entirely without supervision. Unlike prior SSL methods that used uncurated datasets, the original DINOv2 models were trained on LVD-142M, a dataset of 142 million curated images gathered through visual-similarity-based retrieval <sup>[13]</sup> .

However, the NeCo-enhanced DINOv2 model used in this thesis is not trained on LVD-142M directly. Instead, it uses the publicly released pretrained DINOv2

ViT-S/14 weights and applies post-training optimization to further improve the spatial consistency of patch representations <sup>[16]</sup>.



<sup>[16]</sup> **Fig 2.5:** Overview of the NeCo post-training architecture. NeCo takes paired image views passed through frozen student and teacher encoders, aligns region features via ROI Align, and enforces neighbor consistency in the sorted feature space. The student encoder is trained to match the neighbor ordering of the teacher through the NeCo loss

### 2.3.2 Comparative Analysis: DINOv2 vs. Other Self-Supervised Learning Methods

Compared to earlier SSL methods (such as DINOv1 and iBOT) and weakly-supervised models like CLIP/OpenCLIP, DINOv2 exhibits several advantages:

Feature	DINOv2 <sup>[13]</sup>	iBOT / DINOv1	CLIP / OpenCLIP
Supervision	None	None	Image-text supervision
Data	Curated (LVD-142M) via retrieval	ImageNet-22k or uncurated	LAION-2B (noisy captions)
Local Feature Learning	Yes (iBOT loss + KoLeo + NeCo)	iBOT: partial	No
Generalization	Strong across image and pixel tasks	Good	Limited to text-aligned categories
Use in Thesis	ViT-s/14 + NeCo spatial refinement <sup>[16]</sup>	Not used	Not suitable for pixel-level detail

<sup>[13]</sup> **Table 2.3:** DINOv2's architecture is also optimized for large-scale training using FlashAttention, sequence packing, and FSDP, which allows training large models like ViT-g/14 with efficient memory use.

### 2.3.3 Justification for Using DINOv2 in this Thesis

The primary goal of this thesis is to detect early signs of breast cancer in thermal images using robust and spatially aware feature representations. DINOv2 was selected for the following reasons:

**Strong Frozen Representations:** DINOv2 provides state-of-the-art features that support linear classification across multiple domains without requiring model fine-tuning. This is ideal for medical tasks where labelled data is scarce and model retraining is not feasible <sup>[13]</sup>.

**Spatial Feature Awareness:** Thanks to its patch-based learning strategy and iBOT loss, DINOv2 captures fine-grained spatial information. The use of NeCo further enhances these representations by refining spatial coherence through patch-neighbour alignment, which is critical for analysing localized thermal abnormalities in breast images <sup>[16]</sup>.

**Generalization and Robustness:** DINOv2 has been shown to outperform earlier SSL models on domain robustness benchmarks (e.g., ImageNet-A, R, and Sketch), demonstrating that its features generalize well to out-of-distribution data, considered a crucial property for thermal imaging, which can vary across sensors, conditions, or populations <sup>[13]</sup>.

## 2.4 Classifier Architectures

To classify thermal breast images as healthy or cancerous, two different classifier heads were designed and evaluated on top of frozen feature representations extracted by the DINOv2 ViT-s/14 model. These representations were obtained from the 384-dimensional embedding of the [CLS] token, which is commonly used in ViT architectures to summarize the global content of an image. For the linear classifier, the DINOv2 backbone remained entirely frozen throughout training, allowing us to leverage the rich visual semantics already captured by the self-supervised model while ensuring computational efficiency and generalization. In the MLP-based classifier, the final transformer block was selectively fine-tuned alongside the classification head to explore potential gains in performance through limited adaptation.

Training only the classifier head enabled a focused investigation into the quality and linear separability of the DINOv2 features for this domain-specific task. Since the

dataset is relatively small and sensitive to overfitting, freezing the backbone also helped avoid model collapse and preserved the robustness of the pretrained features. Moreover, it allowed for a more interpretable comparison between the two classifier designs, as changes in performance could be directly attributed to the classifier architecture rather than fluctuations in the feature extractor.

The two classification heads implemented include a pure linear model and an MLP with a hidden layer and non-linear activation. Both were trained using supervised learning with binary labels. The linear classifier was evaluated under a sensitivity-first thresholding strategy, where thresholds were selected to maintain a sensitivity of at least 0.90, aligning with clinical needs to minimize false negatives. The MLP classifier was evaluated using a threshold that maximized the difference between true positive and false positive rates (Youden’s index) <sup>[17]</sup>, while still tracking sensitivity and specificity during training.

#### **2.4.1 Linear Classifier**

In alignment with the DINOv2 framework proposed by Facebook Research <sup>[13]</sup>, one of the simplest yet powerful ways to evaluate the quality of self-supervised visual representations is through a linear classification head trained on top of frozen features. A linear classifier is a single fully connected layer (i.e., no hidden layers or non-linearities) that directly maps the representation of an input image to a prediction score. The motivation for using a linear classifier stems from its interpretability and minimal capacity, making it a robust tool for assessing how well the pretrained model has structured the feature space. In the DINOv2 paper and GitHub documentation <sup>[13]</sup>, linear probing is used extensively to benchmark representation quality across various downstream tasks such as classification, retrieval, and segmentation, without modifying the pretrained backbone.

This method is particularly effective in determining if the [CLS] token embedding encodes linearly separable information useful for classification tasks <sup>[13]</sup>. By freezing the DINOv2 ViT-S/14 backbone and training only the linear layer with supervised labels (e.g., healthy vs. cancerous), the fact that classification performance can be attributed solely to the quality of the learned features is ensured, rather than to any fine-tuning of the backbone <sup>[13]</sup>. This setup is also computationally efficient and ideal for

small datasets, as it reduces the risk of overfitting and enables fair comparisons across different classifier heads.

Moreover, practical implementations of this strategy such as those in the open-source tutorial by Rogge <sup>[18]</sup>, demonstrate how linear classifiers can be applied effectively even to tasks like semantic segmentation, using DINOv2 features without additional fine-tuning. This reinforces the versatility of frozen DINOv2 features for diverse computer vision problems. Ultimately, this linear evaluation protocol, as used in DINOv2 benchmarks, shows that frozen representations can outperform other state-of-the-art self-supervised and weakly-supervised models across multiple domains and tasks <sup>[13]</sup>, <sup>[18]</sup>.

#### **2.4.2 Multi Layer Perceptron Classification (MLP)**

While a linear classifier provides a strong baseline for evaluating the linear separability of pretrained representations, it is inherently limited in its capacity to capture complex decision boundaries. To explore whether non-linear combinations of features could improve classification performance, experiments were also made using a multi-layer perceptron (MLP) classifier, a standard architecture that introduces depth and non-linearity to the decision function.

An MLP classifier typically consists of one or more fully connected layers with non-linear activation functions such as ReLU. In this case, the architecture included a hidden layer followed by ReLU activation, batch normalization, dropout, and an output layer for binary prediction. This setup allows the model to learn richer, more flexible mappings from the DINOv2 feature space to the target labels, potentially capturing patterns not linearly separable.

The decision to use an MLP is supported by its widespread application in transfer learning and representation learning tasks, where it is often employed as a projection or classification head on top of the frozen backbones <sup>[19]</sup>. Compared to linear classifiers, MLPs can better adapt to subtle feature differences, especially in medical imaging domains where class distributions may overlap. Moreover, MLPs are commonly used in frameworks like SimCLR and MoCo to separate and refine representations before classification <sup>[20]</sup>, further motivating their inclusion in the evaluation pipeline.

# Chapter 3

## Dataset Handling

- 
- 3.1 Dataset Overview
  - 3.2 Dataset Size and Class Distribution
  - 3.3 Image Dimensions and Format
  - 3.4 Preprocessing for DINOv2 Compatibility
  - 3.5 Dataset Splitting Strategy
  - 3.6 Class Distribution Per Fold
- 

### 3.1 Dataset Overview

The thermal breast images used in this thesis were sourced from two publicly available databases: the “A Database for Mastology Research with Infrared Image (DMR-IR) “<sup>[21]</sup> and the “Thermal Infrared Breast Screening Database (TIBSDB)”<sup>[22]</sup> .

The DMR-IR dataset consists of images collected through a dynamic cooling protocol. Each patient contributed 20 thermal images taken at various angles, along with corresponding temperature matrices. However, for the purposes of this thesis, only frontal-view images were considered, as these are the only images that reliably show both breasts in a single frame. Frontal views are essential for analyzing bilateral temperature symmetry, which plays a critical role in identifying potential abnormalities.

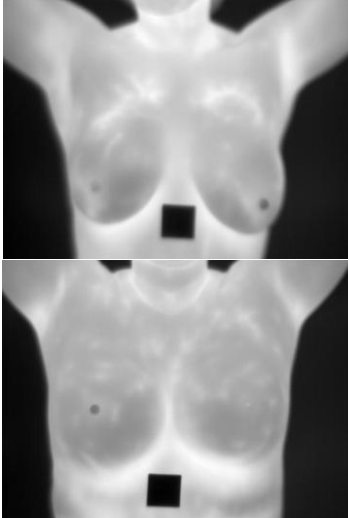
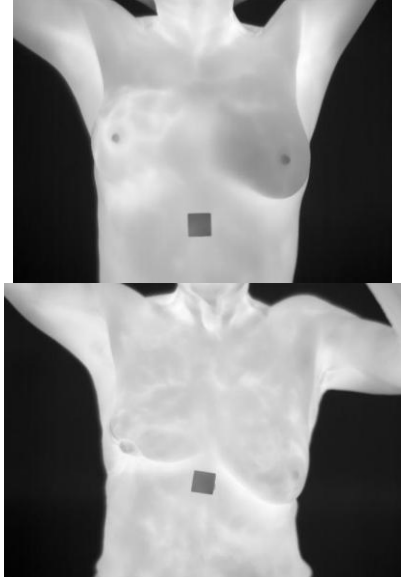
Initially, only the DMR-IR dataset was used. However, after closer inspection, it became clear that most DMR-IR images show only one breast, limiting the ability to assess left–right asymmetry. To address this, the TIBSDB dataset was incorporated into the study. TIBSDB includes three thermal images per subject: frontal, left oblique, and right oblique, but again, only the frontal views were retained to maintain consistency across the combined dataset.

The decision to merge these datasets aimed to increase the total dataset size, thereby enhancing the reliability of the classification process. Both datasets contain high-quality



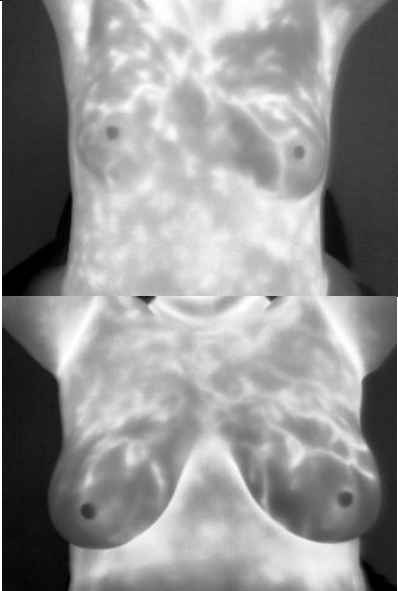
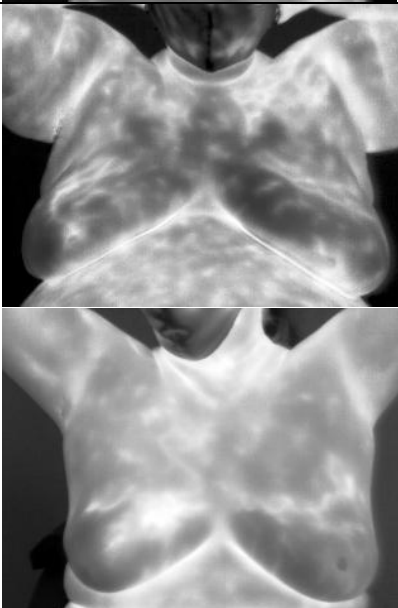
frontal thermal images that clearly show both breasts, enabling symmetry-based analysis and were designed to support non-invasive breast cancer screening and diagnosis through thermal imaging. By combining them, the dataset gains more diversity across patients and imaging settings, which helps reduce overfitting and improves generalization.

Prior to merging, a manual deduplication process was conducted to ensure that no repeated images existed either across or within the datasets. This resulted in the removal of six duplicate images, leading to the final set used in this work.

DMR-IR Images	State
	Healthy
	Sick/Unhealthy

**Table 3.1:** Example frontal thermal healthy and sick images from the DMR-IR dataset <sup>[21]</sup> .

TIBSDB Images	State
---------------	-------

		Healthy
		Sick/Unhealthy

**Table 3.2:** Example frontal thermal healthy and sick images from the TIBSDB dataset <sup>[22]</sup> .

Representative frontal-view images from each dataset are shown in Tables 3.1 and 3.2, respectively. These images demonstrate the consistency in view selection and the visual quality used for symmetry-based breast cancer analysis.

## **Image Acquisition Protocols**

### **DMR-IR Imaging Process**

The DMR-IR dataset employed a dynamic cooling protocol, designed to highlight physiological differences between healthy and abnormal tissues. The process is as follows:

1. **Cooling Phase:** An air stream is applied to the chest area, uniformly lowering skin temperature.
2. **Rewarming Phase:** After cooling, the natural rewarming process is observed.
3. **Image Acquisition:** A series of 20 thermal images are captured at 15-second intervals over a 5-minute period, recording the thermal recovery.

This protocol enhances the contrast between tissues, as abnormal (e.g., malignant) regions typically exhibit different rewarming behaviour compared to healthy tissue. It is particularly effective in highlighting subtle vascular and metabolic irregularities that may not be visible through passive imaging alone.

### **TIBSDB Imaging Standards**

In contrast, the TIBSDB dataset adheres to the imaging standards set by the American Academy of Thermology (AAT), which employs a passive imaging protocol. This means that no external thermal stimulus is applied. Key elements of the AAT protocol include:

1. **Controlled Environment:** Imaging is performed in rooms maintained at 22–24°C with relative humidity between 45–50%.
2. **Patient Preparation:** Patients are allowed time to acclimate to the room environment to stabilize body temperature. Although the dataset documentation references AAT protocols, it does not explicitly state the exact patient acclimatization time prior to imaging.
3. **Standardized Image Angles:** Each patient is imaged from three consistent perspectives: anterior (frontal), left oblique, and right oblique.

By following these strict protocols, TIBSDB ensures the consistency and reliability of its thermal scans, allowing for valid comparisons between subjects and aiding in accurate diagnosis.

### **Relevance for DINOv2-Based Analysis**

The careful adherence to imaging protocols in both datasets ensures thermal consistency across subjects and imaging sessions. This is critical when applying feature-based models like DINOv2, which rely on consistent visual patterns to extract spatial representations. The standardization across datasets enables the model to focus on biological differences rather than imaging artifacts, thereby improving its ability to detect thermal anomalies indicative of breast disease.

All images in the final dataset were acquired using infrared thermal cameras. Furthermore, each image corresponds to a single, unique patient. Only one image per patient was retained in the final dataset to preserve subject independence and prevent data leakage during training and evaluation.

### **3.2 Dataset Size and Class Distribution**

After combining the two thermal imaging datasets (DMR-IR and TIBSDB) and performing manual deduplication, the final dataset used in this thesis consists of 323 frontal thermal breast images, each corresponding to a unique patient. These images were selected based on strict criteria to ensure consistency in view and quality, and to preserve subject independence across all experimental splits.

While the final dataset was carefully curated for consistency and subject independence, its relatively small size, which as mention includes just 323 images, presents certain limitations. Small datasets can increase the risk of overfitting, reduce generalization to unseen cases, and make model performance more sensitive to data imbalance. These constraints highlight the importance of robust cross-validation and careful evaluation strategies, which were applied throughout this thesis to ensure reliable results despite the limited data availability.

The dataset is divided into two classes:

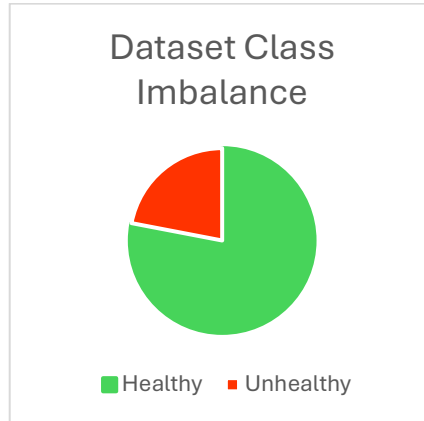
- **Healthy** (patients with no signs of breast abnormalities)

- **Sick** (patients diagnosed with malignant conditions)

The overall class distribution is shown in the table below:

Class	Number of Images
Healthy	252
Unhealthy	71
Total Images	323

**Table 3.3:** Class distribution of the dataset used.



**Figure 3.1:** Visualization of class imbalance in the combined dataset, illustrating the distribution of healthy and cancerous cases. The chart was created using Microsoft Excel.

The final dataset exhibits a noticeable class imbalance, with the healthy class outnumbering the unhealthy class by a ratio of approximately 3.5:1. This imbalance was taken into account during model training, particularly through the use of weighted loss functions and threshold calibration techniques to ensure fair evaluation and maintain sensitivity to the minority (unhealthy) class.

### 3.3 Image Dimensions and Format

The original thermal images in the final dataset were provided in JPG format, as downloaded from the DMR-IR <sup>[21]</sup> and TIBSDB <sup>[22]</sup> repositories. All images were either grayscale with a single channel, or grayscale images encoded with three identical channels (pseudo-RGB), depending on how they were stored and exported. No images used false-colour heatmaps, and no colorized temperature palettes were present.

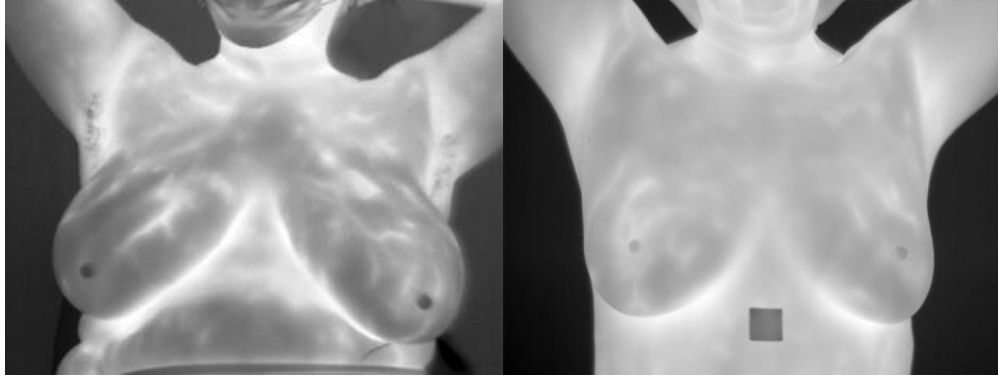
File sizes varied from approximately 14 KB to 218 KB depending on resolution and compression settings.

Across the dataset, two common image resolutions were observed:

- 320×240 pixels

- 640×480 pixels

These resolution variations reflect differences in acquisition settings and equipment used in the original datasets. Some images include embedded metadata indicating use of a FLIR A300 thermal imaging camera and a standardized subject distance of 1 meter, consistent with thermography protocols. All images were later converted to a consistent format as part of preprocessing.



**a. Figure 3.2**

**b. Figure 3.3**

<sup>[22]</sup> **Figure 3.2:** Representative healthy thermal breast image from the TIBSDB dataset, with original dimensions of 320×240 pixels

<sup>[21]</sup> **Figure 3.3:** Representative unhealthy thermal breast image from the DMR-IR dataset, with original dimensions of 640×480 pixels.

### 3.4 Preprocessing for DINOv2 Compatibility

To ensure that the thermal breast images were compatible with the DINOv2 ViT-s/14 architecture, a series of preprocessing steps were applied to standardize the image format, dimensions, and channel configuration. These steps were necessary due to strict architectural requirements of the pretrained DINOv2 models, which were developed using large-scale curated datasets.

According to the DINOv2 publication by Oquab et al. <sup>[13]</sup>, all models including the ViT-s/14 variant used in this thesis, were pretrained on images of fixed resolution and format. The authors specify that input images are resized to 224×224 pixels, which aligns with the patch embedding structure of the Vision Transformer (ViT). As stated in the paper: “The input images are resized to 224×224, which aligns with the model’s architecture” <sup>[13]</sup>.

Additionally, these models were trained on a dataset composed of 142 million curated RGB images sourced from visually diverse domains, including ImageNet-22k and Google Landmarks, using a visual similarity-based retrieval process <sup>[13]</sup>. Although thermal images were not explicitly part of this training set, the scale and diversity of the pretraining data allows DINOv2 to generalize to out-of-distribution domains such as medical thermography:

“Training on our curated data increases the performances on domains that are not used for the curation process... proving that scale and diversity can benefit unseen domains” ( <sup>[13]</sup>. p. 8).

Given that DINOv2 expects 3-channel RGB images of size 224×224, and the thermal breast images included in this particular dataset are acquired in grayscale (with either one or three identical channels), image transformation was required as well.

### **Image Transformation Process**

The first step in the transformation pipeline was to apply colorization to each grayscale thermal image using OpenCV’s COLORMAP\_JET, which maps single-channel intensity values to RGB using a perceptually smooth colour gradient. This was necessary because simply converting 1-channel grayscale images to RGB would result in a flat black-and-white image with no added informational value in the other channels.

The JET colormap was selected because it effectively enhances contrast between different temperature intensities, making subtle variations in heat distribution more visually and numerically distinguishable. This is especially important in medical thermal imaging, where small temperature changes may be clinically meaningful. Although the exact choice of colormap varies across studies, many works employ colormaps to improve feature extraction in thermal-based deep learning tasks.

In the context of DINOv2, which is pretrained on natural RGB images <sup>[9]</sup>, the use of a colormap such as COLORMAP\_JET introduces spatially and chromatically rich patterns into all three channels, making thermal images more visually compatible with the model’s expected input distribution. Although in the official paper <sup>[9]</sup>, DINOv2 was not trained on thermal images, aligning the format of the input with natural RGB patterns allows the model to better leverage its pretrained filters. The JET colormap in particular emphasizes temperature gradients through perceptually distinct colors, which

can help preserve or even enhance relevant thermal features that may otherwise be indistinct in grayscale.

```
# Apply JET colormap
img_color = cv2.applyColorMap(img_np, cv2.COLORMAP_JET)
colorized_img = Image.fromarray(img_color) # Convert to PIL format
```

**Code Snippet 3.1:** Application of colorization to the grayscale thermal images

Then, to prepare the thermal breast images for feature extraction with the DINOv2 ViT-s/14 model, a transformation pipeline was applied using PyTorch using the following Python code:

```
# Define DINOv2-compatible transformations
dinov2_transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to 224x224
    transforms.ToTensor(), # Convert image to tensor (C, H, W)
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406], # Normalize using ImageNet values
        std=[0.229, 0.224, 0.225]
    )
])
```

**Code Snippet 3.2:** Application of DINOv2 compatible transformations to the thermal breast images

First, each image was resized to 224×224 pixels using `transforms.Resize((224, 224))`, in order to match the input resolution expected by DINOv2, which divides each image into 14×14 non-overlapping patches. This resizing ensures architectural compatibility with the model’s patch embedding layer. Next, the image was converted into a PyTorch tensor using `transforms.ToTensor()`, which changes the format to (C, H, W), where C is the number of channels (3 in this case), and also scales pixel values from the 0–255 range to a normalized 0.0–1.0 scale. Finally, `transforms.Normalize()` was applied to each channel using the ImageNet mean and standard deviation values (mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]).

These statistics align the input data distribution with what the DINOv2 model was originally trained on. Normalization ensures that feature extraction is effective and stable, as the pretrained model expects input data with these specific characteristics.

## Visualization

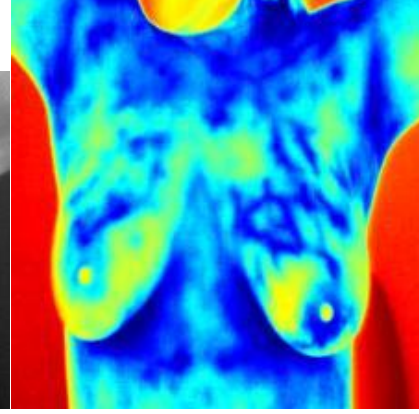
To better illustrate the effect of the preprocessing pipeline described above, an example of a thermal breast image before and after transformation is shown below, in Figure 3



and Figure 4. The original grayscale image is first colorized using the JET colormap, then resized and normalized into a format compatible with DINOv2. This visual demonstration helps highlight the difference between the original grayscale image and its DINOv2-compatible version.



**Figure 3.4**



**Figure 3.5**

**Figure 3.4:** Original grayscale thermal breast image from TIBSDB (IIR0061\_anterior.jpg) before preprocessing with dimensions 320x240.

**Figure 3.5:** The same image from DMR-IR (IIR0061\_anterior.jpg) after colorization, resizing to 224x224, and normalization for DINOv2 compatibility.

### 3.5 Dataset Splitting Strategy

To ensure reliable model evaluation while addressing class imbalance, a Stratified K-fold cross-validation strategy was employed to partition the dataset into training, validation, and test subsets. The splitting was performed using the `train_test_split` and `StratifiedKFold` utilities from Scikit-learn <sup>[23]</sup>.

The choice of `StratifiedKFold` over standard K-Fold was deliberate and essential due to the imbalanced nature of the dataset, which consists of 252 healthy and 71 unhealthy cases. Unlike traditional K-Fold splitting, which divides data without considering label proportions, Stratified K-Fold preserves the relative class distribution in every fold. This is particularly important in medical classification tasks, where underrepresentation of the minority class during training or validation can lead to biased learning, poor generalization, or overestimation of performance. When using the Stratified K-Fold strategy, the folds are made by preserving the percentage of samples for each class <sup>[23]</sup>.

This ensures that each training and validation subset contains representative proportions of healthy and unhealthy samples, thus making evaluation more stable and trustworthy.

In the context of this thesis, where detecting unhealthy cases is critical, maintaining class balance across folds is fundamental to model reliability.

An important function implemented, is that before performing the split, a conditional check using if statements is included to determine whether pre-computed fold files already exist which is the goal as stable folds are important in evaluation of performance metrics. If the folds are found, then they are directly being loaded from the drive to ensure reproducibility and also reduce redundant computation. Otherwise, new folds are being created using Stratified K-Folds and subsequently saved for future use.

### **Fixed 80/20 Initial Split**

Before performing cross-validation, the dataset was first split into two major components:

- 80% (258 images) for training and validation sets
- 20% (65 images) as a fixed final test set, held out for the entire duration of model development, and later on tested as the set with unseen images.

This 80/20 split was stratified and consistent across all five folds. The final test set was used only once, at the end of training and after validation, to provide an unbiased evaluation of model performance on unseen images.

### **Five-Fold Stratified Cross-Validation**

The entire dataset of 323 images was split into five folds using the Stratified K-Fold cross-validation strategy with `shuffle=True` and a fixed random seed for reproducibility. In each fold, a fixed subset of 65 images (20%) was allocated as the final test set and the remaining 258 images (80%) were further split into 206 images for training and 52 images for validation.

This means that in each fold, the model is trained and validated on a unique training/validation split, while being evaluated on the same fixed final test set. This structure allows for robust, stratified cross-validation while maintaining a fully unseen, held-out test set for unbiased final performance evaluation.

Despite the stratification, the class imbalance from the original dataset (252 healthy vs. 71 unhealthy) is still present within each training, validation, and test subset. Stratification only ensures that the proportions are preserved, not that the dataset becomes balanced.

### 3.6 Class Distribution Per Fold

While stratified sampling was employed to maintain the relative proportions of the two classes across all folds, the inherent class imbalance in the original dataset (252 Healthy, 71 Unhealthy) remains evident in each subset.

The table below presents the distribution of Healthy and Unhealthy samples within the training, validation, and final test subsets for each of the five folds. As expected, the proportion of Healthy samples remains significantly higher across all partitions, reflecting the original dataset's imbalance:

Fold	Subset	Total Images	Healthy Images	Unhealthy Images
1	Train	206	160	46
	Validate	52	41	11
	Test	65	51	14
2	Train	206	161	45
	Validate	52	40	12
	Test	65	51	14
3	Train	206	161	45
	Validate	52	40	12
	Test	65	51	14
4	Train	206	161	46
	Validate	52	40	11
	Test	65	51	14
5	Train	206	161	46
	Validate	52	40	11
	Test	65	51	14

**Table 3.4:** Class distribution of Healthy and Unhealthy samples across training, validation and final test sets for each of the five folds, obtained with Stratified-5-Folds.

This figure confirms that the stratified splitting procedure effectively preserved the original class proportions within each training, validation, and test subset. However, it also highlights the persistent class imbalance inherent in the dataset, which must be carefully considered during model training and evaluation. As noted by Luque et al. <sup>[24]</sup>, class imbalance can significantly distort traditional performance metrics such as

accuracy, which can potentially lead to misleading conclusions about a model's effectiveness. Therefore, in order to mitigate these effects, it is essential to incorporate strategies such as class weighting, oversampling of the minority class, or the use of evaluation metrics that are robust to class imbalance, all of which will be discussed in the implementation section (Chapter 4).

# Chapter 4

## Implementation

---

- 4.1 Overview
  - 4.2 Image Reading and Labelling
  - 4.3 Image Preprocessing
  - 4.4 Dataset Preparation-Splitting
  - 4.5 Loading the DINOv2 ViT -s/14 Model
  - 4.6 Feature Extraction Strategy
  - 4.7 Classification Heads
    - 4.7.1 Linear Classification Head
    - 4.7.2 MLP Classification Head
- 

### 4.1 Overview

This section's goal is to provide an overview of the complete implementation pipeline developed for the classification of thermal breast images using a self-supervised ViT model (DINOv2) for breast cancer detection. The entire pipeline was implemented in Python using PyTorch and executed on Google Colab Pro (the subscription of which was kindly provided by UCY) , which provided the necessary GPU acceleration and extended storage space. Google Colab Pro was chosen specifically because the classification experiments involved handling a relatively large dataset of high-resolution thermal images, which required both sufficient computational power and storage capacity to process efficiently, as I personally was struggling to gain access on UCY'S HPC which would also provide the necessary computational power.

The following stages were performed in sequential order as part of the full pipeline which can also be seen in the diagram below:

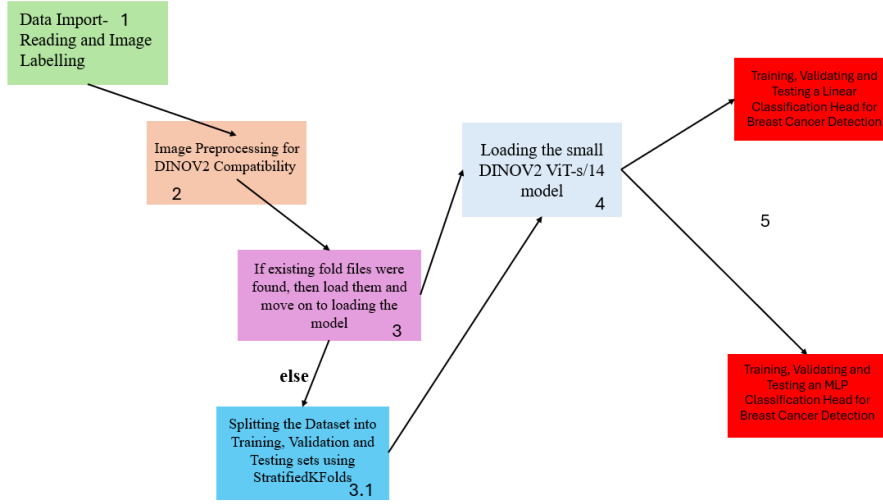
Step 1: Data Import and Image Labelling

Step 2: Image Preprocessing and DINOv2 Compatibility

Step 3: StratifiedKFolds Dataset Splitting and/or Fold Loading

Step 4: Loading the DINOv2 ViT-s/14 Model

Step 5: Classification with a Pure Linear Head and/or Classification with an MLP Head.



**Figure 4.1:** Overview of the implementation pipeline used in this thesis.

The diagram was created using Microsoft PowerPoint to visually represent the sequence of processing stages, from data preparation to classification.

## 4.2 Image Reading and Labelling

As mentioned in Chapter 3, the dataset was formed by combining two different Thermal Breast image datasets, the DMR-IR <sup>[21]</sup> and TIBSDB <sup>[22]</sup> repositories and removing duplicate photos, leading to a final dataset consisting of 323 images. All the Thermal Breast images were then imported in a personal Drive and split into two folders named Healthy and Unhealthy which corresponded to the healthy path and the sick/unhealthy path, respectively. The images were then loaded and combined into a single list, labelled as Healthy and Unhealthy accordingly, forming the final input dataset.

## 4.3 Image Preprocessing

To ensure compatibility with DINOv2, several transformations had to be applied, all of which are detailed in Section 3.4. First, the images were resized to 224x224 which is the required input resolution for the ViT-s/14 architecture. Then, the image tensors were normalized using ImageNet mean and standard deviation values, and finally JET

colormap was applied to the grayscale thermal images to enhance visual features by mapping temperature variations to colour.

#### 4.4 Dataset Preparation – Splitting

As mentioned more specifically in section 3.5, the dataset used in this thesis was split using a Stratified K-Fold cross-validation strategy to ensure a fair and balanced evaluation of model performance. This approach was selected to address the dataset's existing class imbalance, comprising 252 healthy and 71 unhealthy cases by maintaining proportional class distributions across all training validation and test sets.

Before generating new folds, the implementation included a conditional mechanism using if statements to check for the existence of previously saved folds in order to ensure consistency and reproducibility. If these folds did not exist, they were dynamically created and saved for future use.

#### 4.5 Loading the DINOv2 ViT-s/14 Model

A frozen DINOv2 ViT-s/14 model was loaded via the torch.hub utility which simplified accessing pretrained models hosted by research groups. This particular model consists of 12 transformers blocks, each utilizing a 384-dimensional token embedding. It includes a convolutional patch embedding layer with a 14x14 kernel and stride, which converts the input image into a sequence of 256 patch tokens (16x16 grid) before passing them through a stack of standard ViT blocks. Each block employs multi-head self-attention, GELU activations, LayerNorm and MLP feedforward layers. The output of the model is a set of deep visual features extracted without any task-specific classification head. Therefore, only the core feature extractor is used in this work.

In order to enhance the model's spatial awareness, custom pretrained weights which are enhanced by NeCo were manually loaded. These weights were stored in a checkpoint file and were loaded using the torch.load(path) function. The modified state dictionary was then integrated into the base model using `model.load_state_dict(torch.load(path), strict=False)`, which allowed the use of a pretrained self-supervised backbone with enhanced spatial consistency, while keeping the DINOv2 model frozen during training to preserve its learned representations.

In the experiment parameters code block, the model variant is defined using `model_size='s'` which specifies the use of the small DINOv2 model (ViT-s/14). This model is used throughout the pipeline to extract 384-dimensional feature embeddings from each image patch.

#### 4.6 Feature Extraction Strategy

During the early stages of development, the implementation included extracting and storing the DINOv2 patch token embeddings (`x_norm_patchtokens`) for use in classification. However, this approach was removed from the final pipeline as it wasn't necessary for the actual training and added extra complexity. Additionally, the results obtained using the [CLS] token were significantly better for both the Linear and the MLP classifiers compared to earlier experiments using patch token embeddings. So instead, feature extraction now happens on the fly during training, separately in both the Linear Classification head and MLP Classification head. At every epoch, the [CLS] token is extracted directly from the frozen DINOv2 model and fed to the classification head. This way, the pipeline is much simpler and avoids having to store large intermediate feature files.

#### 4.7 Classification Heads

With the purpose of ensuring consistent and fair evaluation across both the linear and the MLP classifiers, the same set of performance metrics was computed at every epoch.

These included Accuracy, Sensitivity, Specificity, Confusion Matrix Components (TP,FP,TN,FN), F1 Score and Binary Cross-Entropy Loss.

These metrics provide a multifaceted view of model performance and they are considered particularly valuable in the context of medical diagnosis, where class imbalance and the consequences of misclassification must be handled.

Accuracy, measures the overall proportion of correct predictions and is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP, TN, FP and FN refer to true positives (correctly classified unhealthy cases), true negatives (correctly classified healthy cases), false positives (healthy cases falsely



classified as unhealthy) and false negatives (unhealthy cases falsely classified as healthy) respectively [5].

Sensitivity, which is otherwise referred to as a recall of the positive class, measures the model's ability to correctly identify unhealthy/cancerous cases. It is especially important in the medical field as its calculation can help minimize false negatives and it is calculated as:

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad [5]$$

Specificity, which is otherwise referred to as a recall of the negative class, measures the model's ability to correctly detect non-cancerous/healthy cases. It helps assess how well the model avoids false alarms, and it is computed as:

$$\text{Specificity} = \frac{TN}{TN + FP} \quad [5]$$

Confusion Matrix Components, which include TP, TN, FP, FN were logged per epoch to provide deeper insight into misclassification behaviour using the confusion\_matrix library from the Scikit-learn library. These raw counts form the basis of the other performance metrics and can help identify systematic prediction errors [5].

F1 score, is the harmonic mean of Precision and Sensitivity, providing a balanced metric that accounts for both FPs and FNs. It is particularly useful when class distribution is imbalanced:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}, \text{ where Precision} = \frac{TP}{TP + FP}$$

Binary Cross-Entropy (BCE) Loss quantifies the model's prediction error for binary classification. It compares the predicted logits with the true labels and penalizes deviations with higher penalties for confident wrong predictions. The BCE loss is calculated per batch and averaged across every epoch.

#### 4.7.1 Linear Classification Head

In order to evaluate the effectiveness of the frozen DINOv2 representations, a linear classification head was chosen as the first classification approach. This method aligns with the evaluation strategy proposed by Facebook Research [13], where a simple linear

layer is used to evaluate the quality of learned features without fine-tuning the backbone. The simplicity of a linear head which consists of a single fully connected layer makes it computationally efficient and especially suitable for relatively small datasets such as the Thermal Breast image dataset that consists of just 323 images. More specifically the linear head allows for a fair assessment of whether the [CLS] token embeddings extracted from the DINOv2 ViT-s/14 model, contain linearly separable information relevant to the task.

The logic and structure of the Linear Classifier implemented in this work were directly inspired by the open-source tutorial by Niels Rogge <sup>[18]</sup> which demonstrates how to apply a linear head on top of DINOv2 features for downstream tasks. By training only the classification layer and keeping the backbone frozen, the setup allows the evaluation to focus entirely on the representational quality of the [CLS] token embeddings produced by DINOv2 ViT-s/14.

With the purpose of preparing the dataset for binary classification, labels of 2 categories were mapped to numerical values. Specifically, the “Healthy” class was mapped to 0 and the “Unhealthy” class to 1 using a dictionary-based approach using:

Label\_mapping= { “Healthy “ : 0, “Unhealthy” :1}. This encoding was necessary for compatibility with the PyTorch loss function BCEWithLogitsLoss which expects numerical targets. The Unhealthy class, was intentionally mapped to the number 1 so that the positive class represented cancerous cases making sensitivity the key metric.

Additionally, before training the linear head, the entire backbone of the DINOv2 model were frozen using param.requires\_grad=False which freezes all the parameters to ensure that the pretrained representations remained unchanged. Freezing the model was essential in order to reduce computational cost and avoid overfitting, especially since the dataset used is relatively small as it consists of just 323 images. A new fully connected layer was appended to the model, mapping the 384-dimensional [CLS] token embedding from the ViT-s/14 variant to a single output logic, suitable for binary classification.

To ensure fair evaluation of the classifier the model was trained using the 5-fold cross-validation scheme. The folds were previously created and saved using a stratified

strategy as described in Section 4.5, which preserved the class distribution across training validation and testing.

By looping through each and every one of the five folds using a for loop, the following steps were executed:

1. The files were loaded from the directory in which they were previously stored. The file contained X\_train, X\_test and X\_final image tensors split into training, validation and final test sets respectively. y\_train, y\_test and y\_final include the corresponding class labels, still in string format (“Healthy” and “Unhealthy”).
2. The next step included label encoding. The string labels were mapped to binary values using a dictionary, Healthy to 0 and Unhealthy to 1. This transformation was necessary to use the BCEWithLogitsLoss loss function, which requires numerical targets.
3. Then, features were extracted from DINOv2. The preprocessed image tensors were passed directly through the frozen DINOv2 model with the sole purpose of extracting the [CLS] token for each image. This was done using `model.forward_features(X)[“x_norm_clstoken”]` and the embeddings were detached from the computational graph with `torch.no_grad()` in order to avoid tracking gradients. A Standard Scaler was applied to normalize the embeddings, which proved to improve numerical stability during training. The scaled features were later converted back to PyTorch tensors and moved to the same device.
4. Then the data loaders were prepared. The feature-label pairs for each split were bundled into TensorDataset objects and passed to Data Loaders with a batch size of 32. The training loader was then shuffled while validation and final test loaders were not.
5. Next, the Loss Function and Optimizer were defined. The BCEWithLogitsLoss function was used, with a positive class weight of 1.35 to counteract class imbalance. The value of class weight was modified multiple times and 1.35 showed the best accuracy results. The selected optimizer was AdamW with a learning rate of 1e-3 which was the best performing value, a variant of Adam that applies weight decay to target only the parameters of the newly added linear head.

6. A nested for loop, looping through 50 epochs was added inside the outer for loop. During each epoch, the linear classification head was trained using the feature-label pairs extracted from the training set. The model was explicitly set to training mode by defining `model.fc.train()` even though the linear head does not contain layers with training-specific behaviors.

A batch of feature vectors representing the features and binary labels (which represent Healthy and Unhealthy cases) were passed from the `train_loader` into the model. The model then works by outputting a raw logic for each sample using `model.fc(features).squeeze(1)` which flattens the output from shape  $(N,1)$  to  $(N,)$  for compatibility with the label tensor.

The loss was computed using `loss=criterion(outputs,labels)` where criterion is `BCEWithLogitsLoss(pos_weight=1.35)` ensuring that emphasis was given on the positive samples, aka Unhealthy cases with label 1 since they are the minority class. Then, gradients are backpropagated with `loss.backward()` and the optimizer updates the weights of the linear layer. For later evaluation, raw predictions and true labels are detached from the computation graph and collected into lists. The average training loss for the epoch is then tracked by adding onto the total train loss as such: `train_loss+= loss.item()`.

7. After the training step, the model's performance is evaluated on the validation set with the purpose of measuring the performance and determining the optimal decision threshold. The model was set to evaluation mode using `model.fc.eval()` to disable gradient computation and enable inference-only behaviour.

Additionally, inference was then performed with `torch.no_grad()` to reduce memory usage and speed up computations. During evaluation, the validation data which consist of [CLS] token embeddings, aka features and labels were passed through the linear head in mini batches producing raw logits again for each sample using the same forward pass structure as during training. These logits which represent the model's confidence scores for the positive/Unhealthy class were stored in a list, while the corresponding ground truth labels were collected into another list.

8. To convert these continuous logits into binary predictions for classification, a custom threshold optimization strategy was implemented, tailored to the domain of breast cancer diagnosis. Instead of using a fixed threshold value which proved

to result in very low accuracy, the threshold was dynamically selected at every epoch. Specifically, thresholds were chosen to maximize sensitivity while maintaining an acceptable balance with specificity. This dynamic strategy was crucial in this case, as false negatives are far more critical than false positives in medical screening tasks.

During the implementation process, it was observed that dynamically optimizing the threshold led to substantially higher sensitivity compared to fixed-threshold methods. This approach is particularly effective and justified in highly imbalanced medical datasets, where one class, in this case healthy, dominates. While not always standard in general classification tasks, dynamic thresholding is increasingly common in medical imaging and clinical AI, where class imbalance and asymmetric risk demand tailored decision boundaries. Therefore, this strategy was adopted in the Linear Head to align model predictions with the practical needs of cancer detection.

Using the `roc_curve` function from Scikit-learn <sup>[26]</sup>, a range of potential threshold was derived along with their corresponding True Positive Rates (TPR) and False Positive Rates (FPR). From this list, only the thresholds that provides a sensitivity value of 90% and above were selected. Among these candidate's the threshold that produced the highest F1-score on the validation set was selected as the best threshold value. If no threshold met the  $\geq 90\%$  sensitivity requirement, a fallback threshold of 0.5 was used. This strategy ensured that model evaluation was aware to sensitivity and that performance comparisons across epochs and folds remained consistent.

9. Once the best performing threshold was selected based on the validation performance, it was applied to evaluate the model on the Final Test set. The final set contains 20% of the entire dataset and was completely held out during the aforementioned training and validation procedures in hopes of ensuring unbiased training.

For each sample in the final test set the [CLS] token features were passed through the frozen model and linear head to produce logits which were thresholded using the selected best threshold value. The resulting binary predictions were compared with the ground truth labels to calculate the

following performance metrics which were logged at every epoch, allowing for detailed tracking of the model's generalization capability:

Accuracy: Which represents the overall proportion of correct predictions

Sensitivity: The ability to correctly detect unhealthy cases (recall of class 1)

Specificity: Ability to correctly detect healthy cases (recall of class 0)

F1-score: Harmonic mean of precision and Sensitivity

Confusion Matrix components: TP,TN,FP,FN

Binary Cross-Entropy Loss on the final test set

Additionally, the misclassified images of each were identified, in order to compare the FNs obtained between the folds.

The following pseudocode outlines the main logic of the linear classification training, evaluation and testing loop described in this section:

*For each epoch:*

*Set model to training mode*

*Initialize loss and prediction lists*

*For each batch in training set:*

*Compute logits*

*Compute loss with class weighting*

*Backpropagate and update linear head*

*Store raw predictions and labels*

*Set model to evaluation mode*

*For each batch in validation set:*

*Compute logits without gradients*

*Store raw predictions and labels*

*Compute ROC curve*

*Filter thresholds with sensitivity  $\geq 0.90$*

*If any:*

*Choose threshold with highest F1*

*Else:*

*Use 0.5 as fallback*

*Apply chosen threshold to raw predictions*

*Compute binary predictions for training and validation*

*Calculate Accuracy, Sensitivity, Specificity, F1-score for both sets*

*Evaluate on final test set:*

*Compute logits*

*Apply threshold*

*Calculate final test metrics (Acc, Sens, Spec, TP, FP, FN, TN, Loss)*

*Identify Misclassified Images*

#### **4.7.2 MLP Classification Head**

While the linear classification head offered a solid baseline for accessing the separability of DINOv2’s learned features, its capacity to capture more complex and non-linear relationships is considered limited. Therefore, to test if results could be improved, a Multi-Layer Perceptron classification head was introduced into the evaluation pipeline. Unlike a linear layer, the MLP architecture includes multiple layers and non-linear activation functions, allowing it to model richer and more abstract patterns in the feature space.

This architectural shift introduces depth into the decision boundary, supposedly enabling the model to uncover relationships that a simple pure linear function cannot capture. In this implementation, the MLP head consisted of a hidden dense layer, followed by ReLu activation, batch normalization for improved convergence, dropout for regularization and a final output layer for binary classification.

The use of MLPs is well-supported in the broader deep learning literature. In representation learning and transfer learning frameworks, MLP heads are commonly employed to project high-dimensional feature vectors into task-specific subspaces <sup>[19]</sup>. Their effectiveness is especially evident in contrastive learning methods where MLP heads play a significantly important role in refining and separating features before classification <sup>[20]</sup>.

Thus, by including an MLP head in the evaluation this study aimed to test whether introducing non-linearity could offer tangible improvements over the linear head and whether the frozen DINOv2 representations could hopefully support more expressive classifiers without needing fine-tuning.

Firstly, all of the DINOv2 model's parameters were frozen by default, using `param.requires_grad=False`. However, the final transformer block, which is block11 and the MLP head (fc) were selectively unfrozen to allow gradient updates. Several attempts were made to unfreeze more than 1 block but results showed that it was causing overfitting and the best accuracy values were given when just block 11 was unfrozen. This was done to enable limited fine-tuning of high level transformer features while keeping majority of the backbone frozen in order to maintain computational efficiency and avoid overfitting.

Then, the MLP architecture was defined, as a simple feedforward network. This network consisted of a linear layer mapping the 384 dimensional [CLS] token embedding to 128 hidden units, batch normalization that stabilizes and accelerates training, ReLu activation to introduce non-linearity, a dropout set to 0.5 to prevent overfitting and a final layer projecting the 128 hidden units to a single output logic for binary classification.

Throughout development, several architectural and training tweaks were explored in hopes of optimizing the performance. These included varying the dropout values between 0.2 to 0.5, modifying the weight decay value which was initially set to  $4e-3$  but showed better performance when set to  $1e-4$ , experimenting with different values of the positive class weighting factors and switching between Traversky loss and focal loss to better emphasize samples which were harder to classify.

The same five stratified folds used in the linear classification experiments were reused in order for consistency to exist between these 2 classification heads. Each fold consisted of training, validation and final test sets. The class labels were mapped using numerical values just like before, by mapping 1 to the Unhealthy class and 0 to the Healthy class.

Once the MLP architecture and fine-tuning strategy were defined, hyperparameters for training were configured. Specifically, the number of epochs was set to 50 therefore the model was trained for a total of 50 epochs per fold. This number was chosen after multiple empirical tests where fewer epochs led to unstable convergence and insufficient generalization. At 50 epochs, the model demonstrated a stable performance allowing sufficient updates to the unfrozen block and MLP head while maintaining



early stopping behavior manually by monitoring metrics. Additionally, a batch size of 32 was selected as it provided a balance between stable gradient estimation and this value was also consistent with the linear head setup to maintain comparability between experiments. The MLP classification head was assigned a relatively high learning rate of 0.0002 ( $2e-4$ ) allowing it to learn quickly. Since this layer is newly utilized and randomly weighted, it required more aggressive updates to learn meaningful representations. In contrast, the final transformer block which is block 11 of the DINOv2 backbone was updated with a much smaller learning rate of  $5e-7$ . This value was selected to prevent the forgetting of pre-trained weights while still allowing higher level representations to be fine-tuned. Lastly, a weight decay of  $1e-4$  was applied to all of the trainable parameters with the purpose of regularizing the model and to prevent overfitting caused by the relatively small size of the dataset.

To ensure consistency with the linear classification head the exact same 5 folds were used for cross-validation. A for loop was used to iterate over each of the five folds and by looping through each and every one of the five folds using a for loop, the following steps were executed :

1. Each dataset split ( $X_{train}$ ,  $X_{test}$  and  $X_{final}$ ) were loaded into memory and immediately cast to float tensors. These were also moved to the same device as the model and the class labels were again transformed using the previously defined label mapping dictionary that mapped them to binary values , Healthy to 0 and Unhealthy to 1 which ensured compatibility with the BCEWithLogitsLoss function that requires numeric values.
2. To handle class imbalance within each of the folds, a positive class weight was computed dynamically using the training labels  $y_{train}$ . This approach was implemented as it gives more weight to the minority class which is Unhealthy cases, ensuring that false negatives are considered more during training. The weight is recalculated per fold to remain adaptive and accurate. The final loss function was defined as `criterion=nn.BCEWithLogitsLoss(pos_weight=pos_weight)`
3. The optimizer used was AdamW, which combines Adam's adaptive learning rate with decoupled weight decay regularization. Parameters were split into 2 groups, with learning rates  $2e-4$  and  $5e-7$  respectively, which enhanced control

over the learning dynamics. To further refine the training process, a Cosine Annealing Warm Restart Scheduler was used, with  $T_0=10$  which sets the initial restart period to 10 epochs and  $T_{mult}=2$  which doubles the restart period after each cycle (10,20,40...). This scheduler cyclically reduces the learning rate and then increases it again, which can help the model escape shallow minima and avoid premature convergence.

4. Before the model training, fold-specific mean and standard deviation statistics were computed. The different values were used to normalize the images and standardize the pixel distribution, in order to ensure numerical stability and consistent gradients. The training transformations included aggressive data augmentation to reduce overfitting.

**Pseudocode of transformations:**

*Apply random horizontal flip to simulate left-right variability*  
*Apply random rotation up to 45° to introduce orientation invariance*  
*Adjust image brightness and contrast using color jitter*  
*Randomly erase a portion of the image to encourage robustness*  
*Normalize image using fold-specific mean and standard deviation*

Each of these operations added robustness against potential noise and variability in real-world thermal imaging. The validation and test sets were only normalized without any augmentation to ensure evaluation on original distributions.

5. The next step included the dataset and data loader preparation. An AugmentedTensorDataset class was used to apply transformations dynamically on each sample during data loading. The training, validation and test datasets were wrapped and fed into PyTorch data loaders. The training loader was shuffled to ensure stochastic gradient updates and the validation and final test loaders preserved order to maintain reproducibility.
6. Once the datasets and data loaders were transformed and prepared, the model entered the core of its training routine, which is a loop running for a total of 50 epochs per fold. At the beginning of each epoch, the model was set to training mode using `model.train()`. This ensured that all components behaved correctly during training. Additionally, each batch from the train loader consisted of preprocessed thermal images and their corresponding labels. For every different batch, gradients were cleared to prevent accumulation from previous batches, [CLS] token embeddings were extracted from the DINOv2 model, the

embeddings were passed through the MLP to produce logits for each image while also removing the singleton dimension and converting shape it from (N,1) to (N,) for compatibility with the label tensor. Moreover, the loss was computed using the weighted BCEWithLogitsLoss and the model weights were updated using optimizer.step(). Thus, raw predictions (logits) and labels were detached from the computation graph and stored for threshold selection and metric evaluation, accumulating the total training loss by adding the loss to the total loss variable.

7. To convert the continuous logits into binary predictions, a threshold needed to be applied. Rather than using a fixed value, a dynamic strategy based on Youden's J statistic <sup>[25]</sup> was adopted. Using Scikit-learn's ROC curve function <sup>[26]</sup>, a range of candidate thresholds was generated, each one associated with its corresponding TPR and FPR. The optimal threshold was chosen by maximizing the value, offering a balance between sensitivity and specificity. This strategy was chosen because it provides an objective and data-driven method for threshold selection, avoiding arbitrary cutoffs. Maximizing Youden's J is a widely used and well-established approach, particularly in biomedical classification problems where both sensitivity and specificity are important. While it does not explicitly prioritize one metric over the other, it offers a balanced compromise, making it especially suitable when both false negatives and false positives carry clinical implications. In our case, it helped ensure that the MLP model achieved a good equilibrium between detecting cancer cases and avoiding false alarms.
8. After the training of the model, it was switched to evaluation mode with model.eval() and inference was performed to disable gradient calculations and reduce memory consumption. For each batch of the tensor loader the same feature extraction and forward pass logic was used. The binary predictions were obtained by thresholding the logits using the best threshold found earlier in the training mode. Several evaluation metrics were calculated, which included Accuracy, Sensitivity, Specificity, F1-score, Confusion matrix and Binary Cross-Entropy Loss.
9. Lastly, the final test set which was held out during training and validation was used. It was evaluated using the same procedure described above, again using

the epoch-specific best threshold. As before, several predictions were generated, and the same set of evaluation metrics was computed, and the misclassified images out of each fold were calculated, with the purpose of comparing the FNs between the folds.

The following pseudocode outlines the core logic of the MLP classification head training, evaluation and testing pipeline as described in this section:

*For each epoch:*

*Set model to training mode*

*Initialize loss and prediction lists*

*For each batch in training set:*

*Compute [CLS] token features*

*Pass features through MLP head to get logits*

*Compute weighted loss*

*Backpropagate and update parameters*

*Store raw predictions and labels*

*Compute ROC curve on training predictions*

*Select threshold that maximizes sensitivity and specificity*

*Set model to evaluation mode*

*For each batch in validation set:*

*Compute logits without gradients*

*Store raw predictions and labels*

*Apply chosen threshold to obtain binary predictions*

*Calculate Accuracy, Sensitivity, Specificity, and Confusion Matrix, F1-score*

*Evaluate on final test set:*

*Compute logits and apply threshold*

*Calculate final test metrics (Accuracy, Sensitivity, Specificity, F1-score, TP, FP, FN, TN, Loss)*

*Identify misclassified images.*

### 4.7.3 MLP Head: Thresholding and Training Constraints Matched to Linear Baseline

In the previous section (4.7.2), the MLP classification head was introduced to evaluate whether a non-linear decision boundary could improve classification performance over the linear head. That implementation focused on balancing sensitivity and specificity using Youden’s J statistic <sup>[25]</sup> for threshold selection, and it included partial fine-tuning by unfreezing the final transformer block (block 11) of the DINOv2 backbone.

To enable a fair and direct comparison with the linear classification head, this section presents a modified implementation of the MLP head that follows the exact same experimental constraints and thresholding strategy as the linear head. Specifically, the entire DINOv2 model remains frozen, with only the MLP head being trained, and the threshold used to convert logits into binary predictions is dynamically selected based on achieving sensitivity  $\geq 0.90$ , followed by maximizing the F1-score among those candidates. If no threshold meets the sensitivity requirement, a fallback value of 0.5 is applied. This design ensures that performance differences between the linear and MLP heads can be attributed solely to their architectural differences, and not to variations in thresholding or fine-tuning.

The same five stratified folds used in the linear classification experiments were reused in order for consistency to exist between these 2 classification heads, and the same values were used as (4.7.2) for the hyperparameters. Each fold consisted of training, validation and final test sets. The class labels were mapped using numerical values just like before, by mapping 1 to the Unhealthy class and 0 to the Healthy class.

Additionally, the number of epochs was set to 50 just like the previous classification head experiments, which showed more stable performance than other epoch numbers. Specifically, this number was chosen after multiple empirical tests where fewer epochs led to unstable convergence and insufficient generalization.

A for loop was used to iterate over each of the five folds and by looping through each and every one of the five folds using a for loop, the following steps were executed:

1. Each dataset split (X\_train, X\_test, and X\_final) was loaded into memory and immediately cast to float tensors. These tensors were also moved to the same device as the model. The class labels were transformed using the previously defined label mapping dictionary, converting “Healthy” to 0 and “Unhealthy” to 1. This transformation ensured compatibility with the BCEWithLogitsLoss function, which expects numeric binary labels.
2. To handle class imbalance within each fold, a positive class weight was computed dynamically using the y\_train labels. This approach assigned greater importance to the minority class (“Unhealthy”), reducing the risk of false negatives. The weight was recalculated per fold to remain adaptive and accurate. The final loss function was defined as `criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)`.
3. The optimizer used was AdamW, which combines Adam’s adaptive learning rate mechanism with decoupled weight decay for better generalization. In contrast to the previous MLP implementation, only the MLP head was trained in this version. The entire DINOv2 backbone remained frozen, and thus only one parameter group (the MLP head) was optimized using a learning rate of  $2e-4$ . A CosineAnnealingWarmRestarts scheduler was also used, with  $T_0=10$  and  $T_{mult}=2$ , to gradually reduce and reset the learning rate cyclically and encourage better convergence behavior.
4. Before model training began, fold-specific mean and standard deviation statistics were computed using the training images. These statistics were then used to normalize all images in the fold, standardizing the pixel distribution for numerical stability and gradient consistency. The training images were also augmented with transformations to reduce overfitting.

**Pseudocode of transformations:**

*Apply random horizontal flip to simulate left-right variability*

*Apply random rotation up to  $45^\circ$  to introduce orientation invariance*

*Adjust image brightness and contrast using color jitter*

*Randomly erase a portion of the image to encourage robustness*

*Normalize image using fold-specific mean and standard deviation*

The validation and final test sets were only normalized (without augmentation) to ensure that evaluation occurred on data distributions that resembled real-world scenarios.

5. Dataset and data loader preparation followed the same structure as before. An `AugmentedTensorDataset` class dynamically applied the defined transformations. Each dataset (train, validation, and final test) was wrapped in a `PyTorch DataLoader`, with shuffling enabled for the training set and disabled for the validation and final test sets to ensure reproducibility.
6. With datasets and data loaders in place, the model entered its training loop, running for 50 epochs per fold. At the beginning of each epoch, the model was set to training mode using `model.train()`. Each batch of preprocessed thermal images and labels was processed by first zeroing out any existing gradients. The [CLS] token embeddings were extracted from the frozen DINOv2 model, then passed through the MLP head to produce raw logits (of shape (N,)). Loss was computed using the weighted `BCEWithLogitsLoss`, and the model's parameters were updated using `optimizer.step()`. Raw logits and labels were detached and stored for metric evaluation and threshold selection.
7. To convert logits into binary predictions, a sensitivity-prioritized threshold selection strategy was employed. Using Scikit-learn's `roc_curve` function <sup>[26]</sup>, candidate thresholds were filtered to retain only those achieving sensitivity  $\geq 0.90$ . From these, the threshold that yielded the highest F1-score was selected as optimal. If no threshold met the sensitivity requirement, a default threshold of 0.5 was used. This approach emphasized minimizing false negatives while maintaining strong predictive performance.
8. After training, the model was switched to evaluation mode with `model.eval()`. Inference was performed on the validation and final test sets without gradient computation. The same forward pass logic was applied, and binary predictions were generated using the previously selected threshold. Key metrics including Accuracy, Sensitivity, Specificity, F1-score, Confusion Matrix values, and BCE Loss were computed.

9. The final test set, which had been withheld from training and validation, was evaluated using the same strategy. All metrics were calculated again, and misclassified images were identified and logged. These results were later used to compare the performance across folds and between classifier heads, particularly focusing on false negatives.

The following pseudocode outlines the core logic of the fully frozen MLP classification head training, evaluation and testing pipeline as described in this section:

*For each epoch:*

*Set model to training mode*

*Initialize loss and prediction lists*

*For each batch in training set:*

*Compute [CLS] token features from frozen DINOv2*

*Pass features through MLP head to get logits*

*Compute weighted loss*

*Backpropagate and update parameters*

*Store raw predictions and labels*

*Compute ROC curve on validation predictions*

*Select threshold with sensitivity  $\geq 0.90$  that maximizes F1-score*

*(Use 0.5 if no threshold meets the sensitivity constraint)*

*Set model to evaluation mode*

*For each batch in validation set:*

*Compute logits without gradients*

*Store raw predictions and labels*

*Apply chosen threshold to obtain binary predictions*

*Calculate Accuracy, Sensitivity, Specificity, and F1-score*

*Evaluate on final test set:*

*Compute logits and apply threshold*

*Calculate final test metrics (Accuracy, Sensitivity, Specificity, F1-score, TP, FP, FN, TN, Loss)*

*Identify misclassified images*



# Chapter 5

## Experiments, Results & Discussion

---

- 5.1 Overview of Evaluation Setup
  - 5.2 Linear Classification Head Results
    - 5.2.1 Performance at 50 Epochs
    - 5.2.2 Performance at 150 Epochs
    - 5.2.3 Performance at 30 Epochs
    - 5.2.4 Analysis of Misclassified Cases Affecting Sensitivity with Linear Head
    - 5.2.5 Hyperparameter and Architectural Exploration with Linear Head
  - 5.3 MLP Classification Head Results
    - 5.3.1 Performance at 50 Epochs
    - 5.3.2 Performance at 150 Epochs
    - 5.3.3 Performance at 30 Epochs
    - 5.3.4 Analysis of Misclassified Cases Affecting Sensitivity with MLP Head
    - 5.3.5 Hyperparameter and Architectural Exploration with MLP Head
  - 5.4 Comparative Analysis: Linear Classification vs MLP Classification at 50 Epochs
  - 5.5 Results and Discussion
    - 5.5.1 Generalization and Model Behaviour
    - 5.5.2 Sensitivity vs. Specificity Trade Off
    - 5.5.3 Per-Fold Performance Instability
    - 5.5.4 Clinical Implications
- 

### 5.1 Overview of Evaluation Setup

The following section presents an overview of the strategy which was adopted in this study to assess the effectiveness of DINOv2 based classification of thermal breast images. The main goal of the evaluation was to determine whether the features extracted by the frozen DINOv2 ViT-s/14 model when paired with the appropriate

classifier heads, can achieve reliable and medically meaningful performance in terms of distinguishing between healthy and unhealthy thermal breast images.

In order to evaluate this two distinct classification heads were trained , evaluated and tested on top of the frozen DINOv2 [CLS] token embeddings, a Linear Classification head consisting of a single fully connected layer used to test the linear separability of extracted features and an MLP Classification head consisting of just one hidden layer, ReLu activation, batch normalization, dropout and an output layer, aimed to capture more complex and non-linear boundaries.

All of the experiments were conducted using a Stratified 5-Fold CV, ensuring that the distribution of healthy and unhealthy cases was preserved for each fold. Exactly 20% of the dataset, more specifically 65 pictures, was held out as a final test set used only once at the end of training and validation with the purpose of providing an unbiased evaluation of the model's performance on unseen data.

Each classification head was evaluated under three training regimes, the first one being 50 epochs which is the final chosen training duration based on stability and generalization and the second and third being 30 and 150 epochs which throughout the testing both acted as an exploratory setting used to access the impact of extended training duration on model performance and overfitting.

Performance was measured using the following key metrics<sup>[5]</sup>:

Accuracy: The overall proportion of correct predictions.

Sensitivity: Measures the model's ability to correctly identify unhealthy/cancerous cases.

Specificity: Measures the model's ability to correctly detect non-cancerous/healthy cases.

Binary Cross-Entropy Loss: Quantifies the model's prediction error for binary classification.

Confusion Matrix Components: TP,FP,TN,FN which are values used for deeper misclassification insights.

In the context of early breast cancer detection, sensitivity is considered the most critical performance metric. Sensitivity which is also referred to as the true positive rate, measures the model's ability to correctly identify individuals who suffer from breast cancer. This is crucial in medical applications where cases in which cancerous images are misclassified as healthy, can delay diagnosis and treatment and potentially result in worsened outcomes or even fatal consequences in some cases. Therefore, this thesis emphasized in achieving a high sensitivity rate, specifically  $\geq 0.9$  even if it comes as the cost of lower specificity and accuracy.

However, while minimizing FN is the main priority, other metrics also play a significant role in evaluating the model's overall performance and clinical reliability. Specificity measures the model's ability to correctly classify healthy individuals and help avoid FP that could lead to unnecessary stress, time consuming follow-up tests and increased healthcare costs. A model with high sensitivity but extremely low specificity can raise too many false alarms and diminish its practical applicability in real world scenarios. Therefore, a balanced trade-off is necessary, and specificity is closely monitored along sensitivity.

Additionally, binary cross-entropy loss provided insights regarding the model's overall confidence in predictions. Even at times where sensitivity or accuracy appear high, high loss values may indicate uncertainty in decision-making especially if the model is producing overconfident but false predictions. So, by tracking the loss over training, validation and testing phases, this thesis ensures that performance improvements are not achieved at the expense of the model's stability or generalization.

As mentioned previously, the dataset used suffers from class imbalance, with 252 healthy images and 71 unhealthy images, thus, to address this, a positive class weight strategy was applied in the loss function and a threshold optimization was conducted dynamically per epoch during validation, which was later used in testing. For the Linear Classification head, thresholds were selected to maintain sensitivity  $\geq 0.90$ . For the MLP Classification, threshold selection prioritized Youden's index<sup>[17], [25]</sup> while still tracking sensitivity closely,

All of the experiments were implemented in PyTorch, using Google Colab Pro with GPU acceleration. Throughout training, the model was evaluated per epoch on

validation data and the final performance was accessed on the test set using the breast threshold identified during validation.

The following hyperparameters were used in each classifier:

**Linear Classification Head contained:**

Learning rate:  $1e-3$

Loss Function: BCEWithLogitsLoss with a positive weight of 1.35

Optimizer: AdamW

Batch Size: 32

Epochs: 50 (final) 30 and 150 (for comparison purposes)

Backbone: DINOv2 ViT-s/14 fully frozen

Threshold Strategy: Sensitivity-aware, keeping only the one's with value 0.9 and above

**MLP Classification Head contained:**

Hidden Layer Size: 128 units

Activation Function: ReLu

Dropout Rate: 0.5

Weight Decay:  $1e-4$

Optimizer: AdamW

Batch Size: 32

Epochs: 50 (final) 30 and 150 (for comparison purposes)

Backbone: DINOv2 ViT-s/14 , all blocks except block 11 were frozen

Learning Rate(Head):  $2e-4$

Learning Rate (Last Transformer Block):  $5e-7$

Loss Function: BCEWithLogitsLoss with a dynamically allocated positive weight

Scheduler: Cosine Annealing Warm Restarts ( $T_0=10$ ,  $T_{mult}=2$ )

Threshold Strategy: Youden's index with sensitivity monitoring

## 5.2 Linear Classification Head Results

### 5.2.1 Performance at 50 Epochs

To evaluate the effectiveness of the Linear Classification head trained on frozen DINOv2 features, this section presents a table summarizing the maximum achieved value, which is the best performance across the 50 epochs, for each fold in terms of key evaluation metrics. These include Accuracy, Sensitivity, Specificity and Loss for training, validation as well as testing.

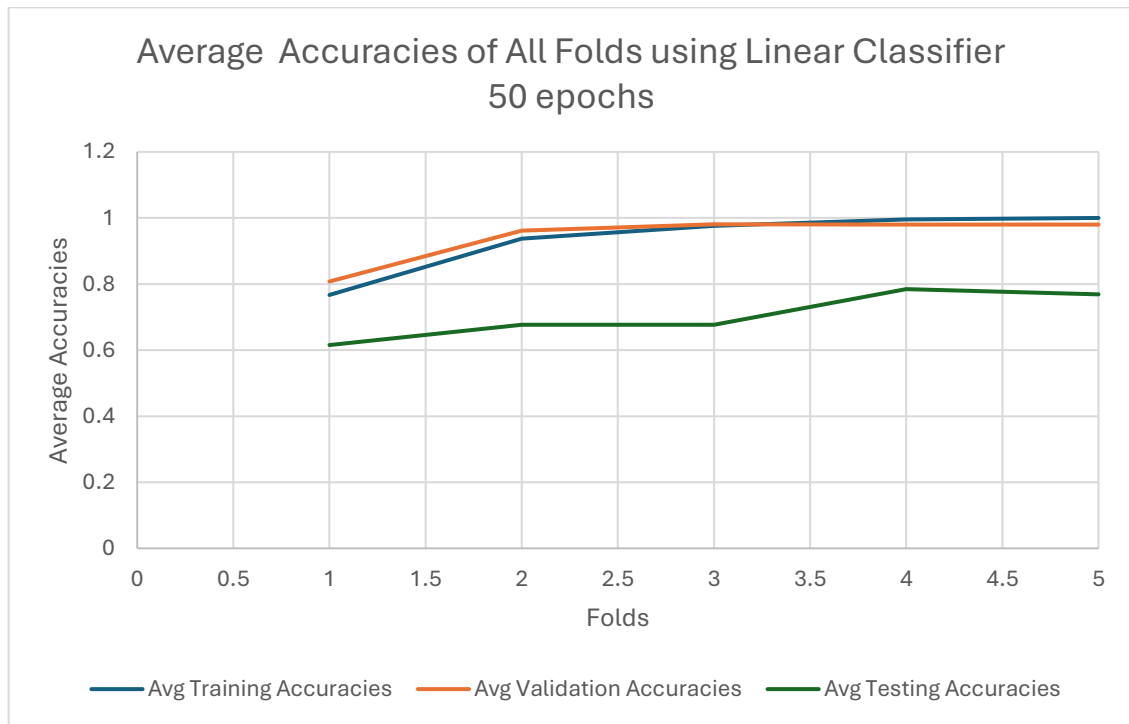
Special attention was paid to sensitivity as the primary goal of this thesis is early breast cancer detection. High sensitivity ensures that the model identifies cancerous cases correctly, minimizing false negatives which is extremely important in medical diagnosis. However, metrics such as specificity and loss are equally as important for understanding the model's ability to distinguish healthy cases and to quantify the overall prediction confidence, respectively. Together, all of these values offer a detailed fold-wise view of the model's behaviour and support reliable CV analysis.

Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss	Test F1-score
1	0.767	1	0.7	5.5	0.8077	0.9091	0.7805	0.6713	0.6154	0.9286	0.5686	1.4299	0.4906
2	0.9369	1	0.9317	3.0824	0.9615	0.9167	0.975	0.477	0.6769	0.8571	0.7451	1.3453	0.4878
3	0.9757	1	0.9814	2.158	0.9809	0.9167	1	0.5004	0.6769	0.9286	0.7461	1.191	0.4583
4	0.9952	0.9783	1	1.7945	0.9804	0.9091	1	0.2138	0.7846	0.6429	0.8824	1.2732	0.4848
5	1	1	1	1.1756	0.9804	0.9091	1	0.4029	0.7692	0.5714	0.9216	1.3588	0.4571

**Table 5.1:** Performance metrics obtained with training, validation and testing, of Linear Classification Head with 50 epochs

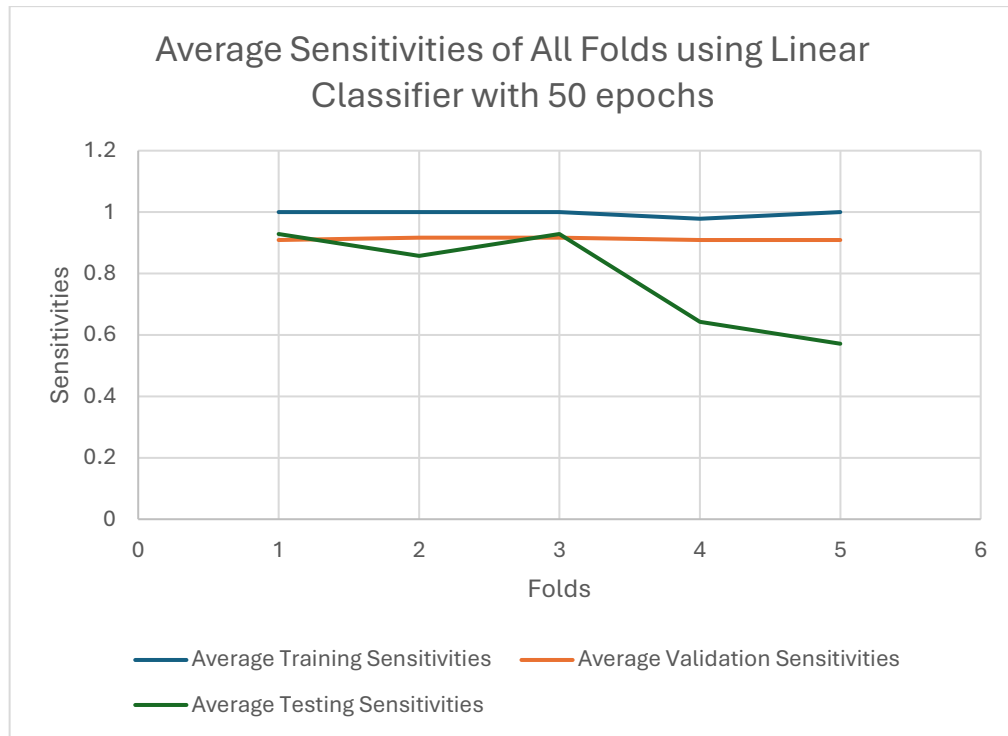
In addition to the summarized table of maximum achieved values, the following linear graphs provide a more detailed and continuous view of the Linear Classification Head's testing performance over all 50 training epochs, across each individual fold. These graphs illustrate the evolution of the four key evaluation metrics, Accuracy, Sensitivity, Specificity, and Loss throughout testing. Unlike the table, which captures only the best point achieved in each fold, these plots reveal the complete learning behaviour and stability of the model over time.

The table and line graphs below, illustrate the trends in average (meaning maximum of each fold) accuracy, sensitivity, specificity and loss across all five folds using the Linear Classification Head trained for 50 epochs. These trends are complemented by the fold-wise numerical values presented in the table.



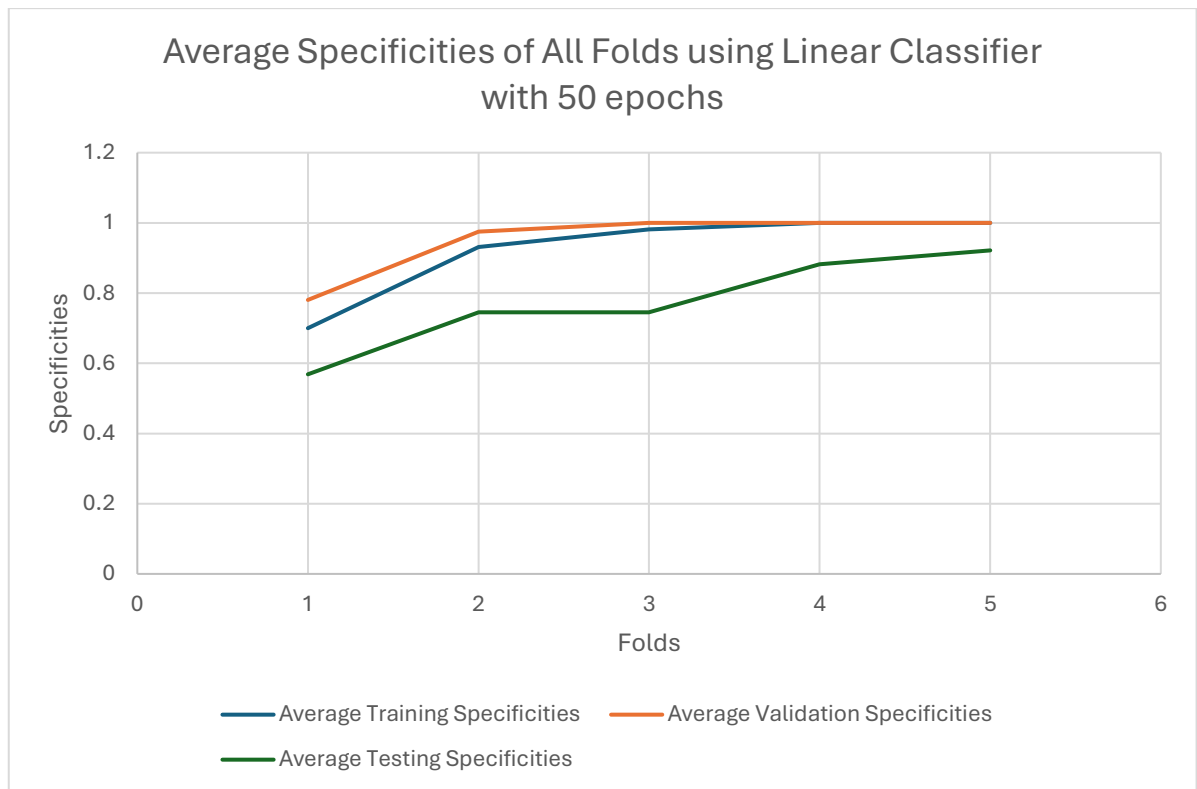
**Figure 5.1:** Average Accuracy Values of All Folds when using the Linear Classifier with 50 epochs.

Starting with accuracy, training accuracy increased progressively across folds, reaching a perfect score of 1.000 in Fold 5, while validation accuracy peaked at 0.9809 in Fold 3. Test accuracy however, remained consistently lower ranging from 0.6145 in Fold 1 to 0.7846 in Fold 4 with an overall average of 0.7046. This noticeable gap between training and validation, and test accuracy suggests mild overfitting which is a pattern that is common in small medical imaging datasets where generalization to unseen data can be challenging.



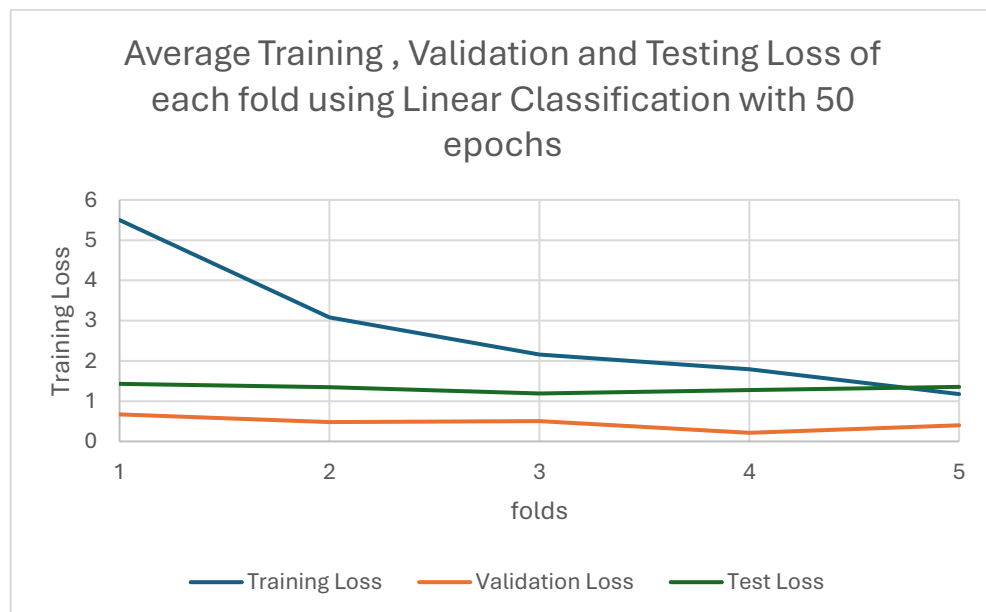
**Figure 5.2:** Average Sensitivity Values of All Folds when using the Linear Classifier with 50 epochs.

Sensitivity which is the metric that is most aligned with this thesis' clinical goal remains exceptionally high in the training phase across all folds, specifically all equal or above of 0.9783 and in validation folds above 0.9 with vales such as 0.9167 in Folds 2 and 3. In the testing phase however, a downward trend was observed. While Folds 1 to 3 achieved strong results all of them being above 0.8571 and fold 3 reaching 0.9286, Fold 4 dropped at 0.6429 and Fold 5 dropped further to 0.5714. These final folds reveal that while the strategy used for thresholding might have been effective in the earlier folds, it failed to preserve sensitivity across all data splits which might be caused to fold-specific class imbalance or subtle domain shifts.



**Figure 5.3:** Average Specificity Values of All Folds when using the Linear Classifier with 50 epochs.

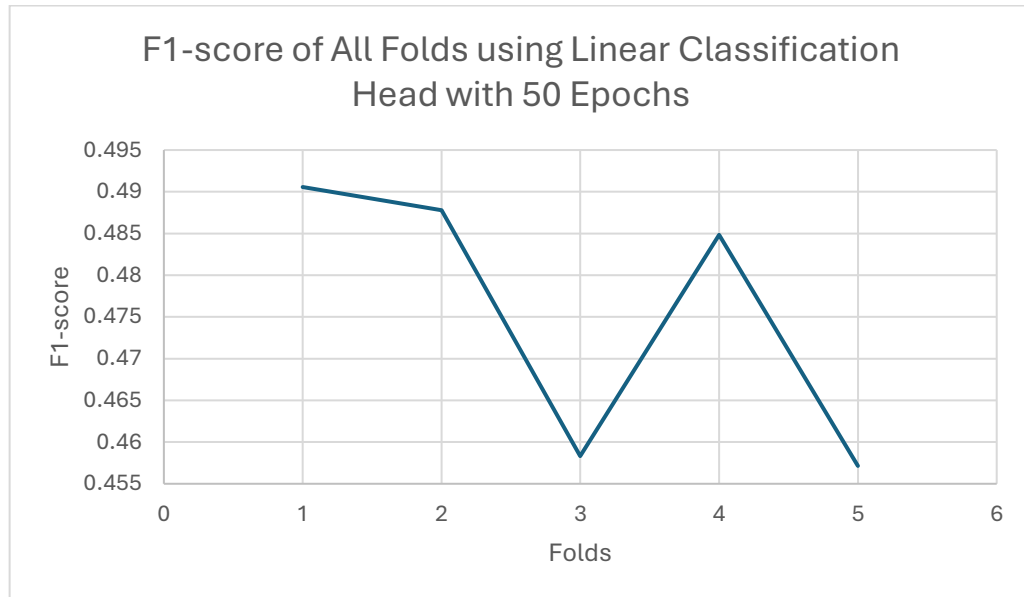
In contrast to sensitivity, specificity followed an upward trend during the testing phase. It started as relatively low at 0.5686 in Fold 1, but gradually increased peaking at 0.9216 in Fold 5. This suggests that in the later folds the model became more confident in correctly classifying healthy images, even though this came at the cost of missing more cancerous cases, thus reducing sensitivity.



**Figure 5.4:** Average Loss Values of All Folds when using the Linear Classifier with 50 epochs.



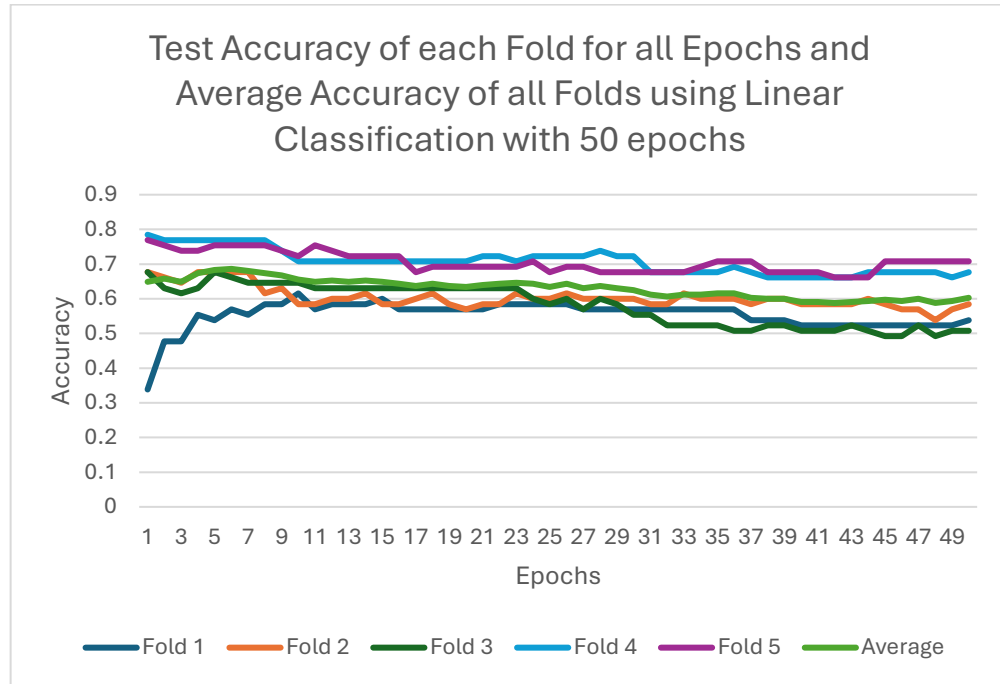
The loss curve further validates the aforementioned patterns. Training loss declined steadily from 5.5 in Fold 1 to 1.1756 in Fold 5, confirming that the model fits the training data well overtime. Validation loss varied between 0.2138 and 0.6713 with the lowest value in Fold 4, while test loss ranged from 1.191 in Fold 3 to 1.4299 in Fold 1. These results show that although the model learned consistently, it did not gain substantially in confidence when exposed to the unseen data.



**Figure 5.5:** Average F1-score Values of All Folds when using the Linear Classifier with 50 epochs.

Figure 5.5, displays the F1-scores obtained on the final test sets across all five cross-validation folds using the Linear Classification Head after 50 training epochs. The results exhibit moderate consistency, with F1-scores ranging from 0.457 to 0.491. The highest F1-score was recorded in Fold 1 (0.491), which also demonstrated a high sensitivity of 0.9286, indicating strong detection of positive (unhealthy) cases. Folds 2 and 4 yielded similar F1-scores (0.488 and 0.485, respectively), with Fold 2 achieving slightly lower sensitivity (0.8571) compared to Fold 4 (0.6429). Notably, Folds 3 and 5 produced the lowest F1-scores (0.458 and 0.457), despite Fold 3 also having a high sensitivity (0.9286), suggesting that precision may have been low in this case, resulting in more false positives. The relatively low F1-score in Fold 5 coincides with its reduced sensitivity (0.5714), indicating weaker detection capability. Overall, the linear classifier showed consistent performance with small variability across folds, but its limited F1-scores suggest a trade-off between maintaining high sensitivity and achieving adequate precision.

Additionally, the following graphs present the evolution of test accuracy, sensitivity, specificity and loss across all 50 training epochs for each fold individual and the average trend across folds. By carefully observing these graphs, specific fold dependent patterns in how the Linear Classifier's generalization performance had evolved during the training phase.



**Figure 5.6:** Test Accuracy per epoch and Average Accuracy, across all folds using the Linear Classification Head with 50 epochs.

As far as test accuracy evolution is concerned, it can be observed from the linear graphs above that for fold 1, test accuracy started extremely low, standing at 0.33 during the first epoch and then rapidly increased by epoch 5, reaching an accuracy of 0.47. Then, a slow and steady rise was observed within accuracy fluctuations between 0.53 and 0.58 during epochs 10 until 20. However, after epoch 20 accuracy stood at around 0.57-0.58 and by epoch 50 it stabilized at 0.53 indicating that no significant generalization improvements occurred after the first 20 epochs.

Fold 2 had started considerably high at 0.67 at the first epoch. However, it showed a decline pretty early on, dropping to 0.61 by epoch 8. Between epochs 10 and 20 accuracy fluctuated mildly between 0.58 and 0.62 and beyond epoch 20 accuracy remained relatively stable at around 0.6, ending at approximately 0.58 when reaching

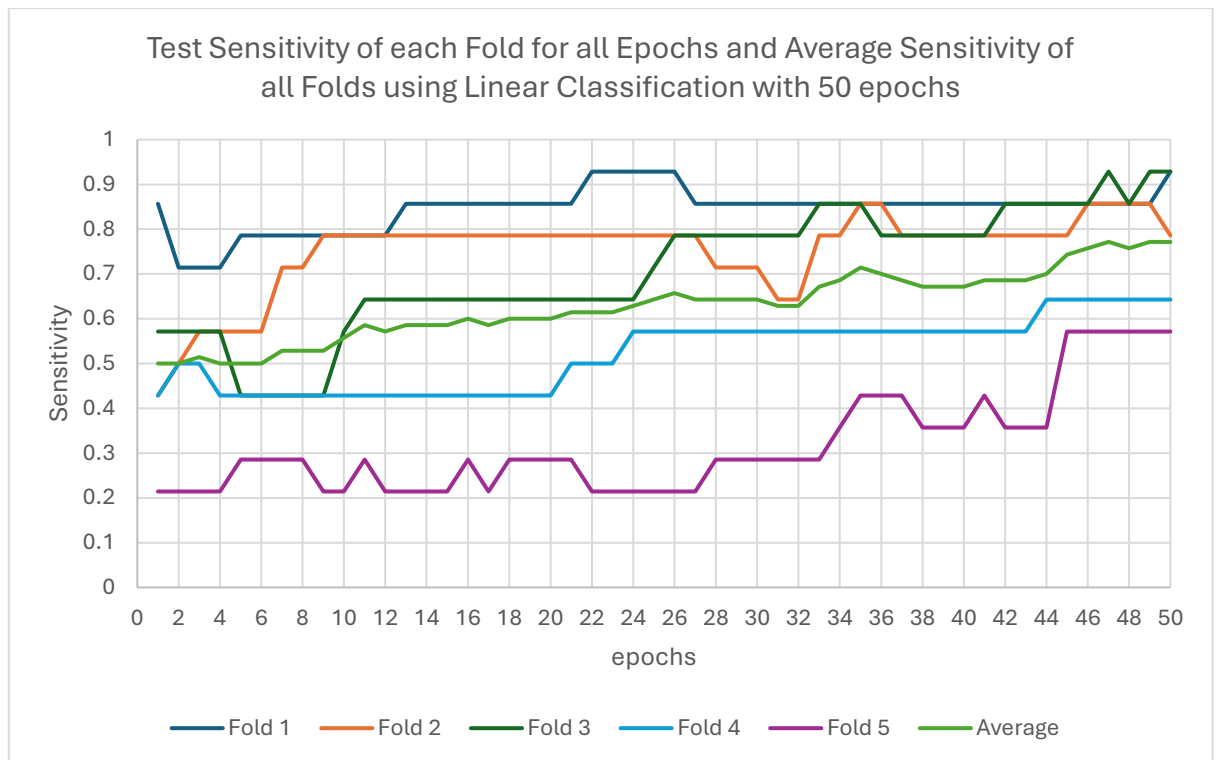
epoch 50. The overall trend was relatively stable but slightly downward throughout the 50 epochs.

Additionally, fold 3 started with a promising test accuracy of 0.67 at epoch 1, maintaining good performance. However, a slight but steady decrease was observed moving from around 0.66 at epoch 5 to 0.60-0.62 by epochs 30-50. More specifically, after epoch 25 a small degradation occurred with the test accuracy hovering around 0.53-0.58 towards the final epochs.

The model in Fold 4, achieved the highest initial accuracy, starting at 0.78 during the first epoch. However, after epoch 10 it experienced a progressive decline with accuracy dropping to 0.73 by around epoch 20. After epoch 30, the performance stabilized slightly lower, at around 0.67-0.70 and remained consistent until epoch 50.

Fold 5 had also started strong, with an accuracy of 0.76. Throughout all 50 epochs it was relatively stable and flat, oscillating slightly but staying between 0.73-0.75. There was a minor decrease around epochs 17-44, dropping to 0.6x but it fortunately recovered quickly, finishing at around 0.70 at epoch 50. Fold 5 maintained the most consistent performance throughout the testing phase.

The average test accuracy of all folds, started at 0.64 at epoch 1, slightly peaked at around epochs 5-10 with a value of 0.68 but then gradually declined after epoch 20. By epoch 30, the average accuracy dropped close to 0.60, fluctuating between 0.59-0.61 until epoch 50. This indicates that the best generalization was reached early on in the testing phase, and prolonged training beyond epochs 20-30 did not yield meaningful gains. Instead, a mild overfitting trend appeared as the model started fitting data better but generalization on the test set weakened.



**Figure 5.7:** Test Sensitivity per epoch and Average Sensitivity, across all folds using the Linear Classification Head with 50 epochs.

As far as test sensitivity evolution is concerned, it can be observed from the linear graphs above that for Fold 1, test sensitivity started very high, standing at 0.85 during the first epoch. Then, a small dip was observed around epoch 2, dropping to 0.71, but fortunately by epoch 5 sensitivity rose back up to 0.78. From epochs 10 until 20, sensitivity remained consistently stable at around 0.78–0.86. After epoch 20, performance slightly improved further, reaching up to around 0.93 by epochs 21–23, and then remained consistently high, fluctuating between 0.85–0.93 all the way until epoch 50. Overall, Fold 1 maintained strong sensitivity across the entire training, achieving the best behaviour in terms of minimizing false negatives.

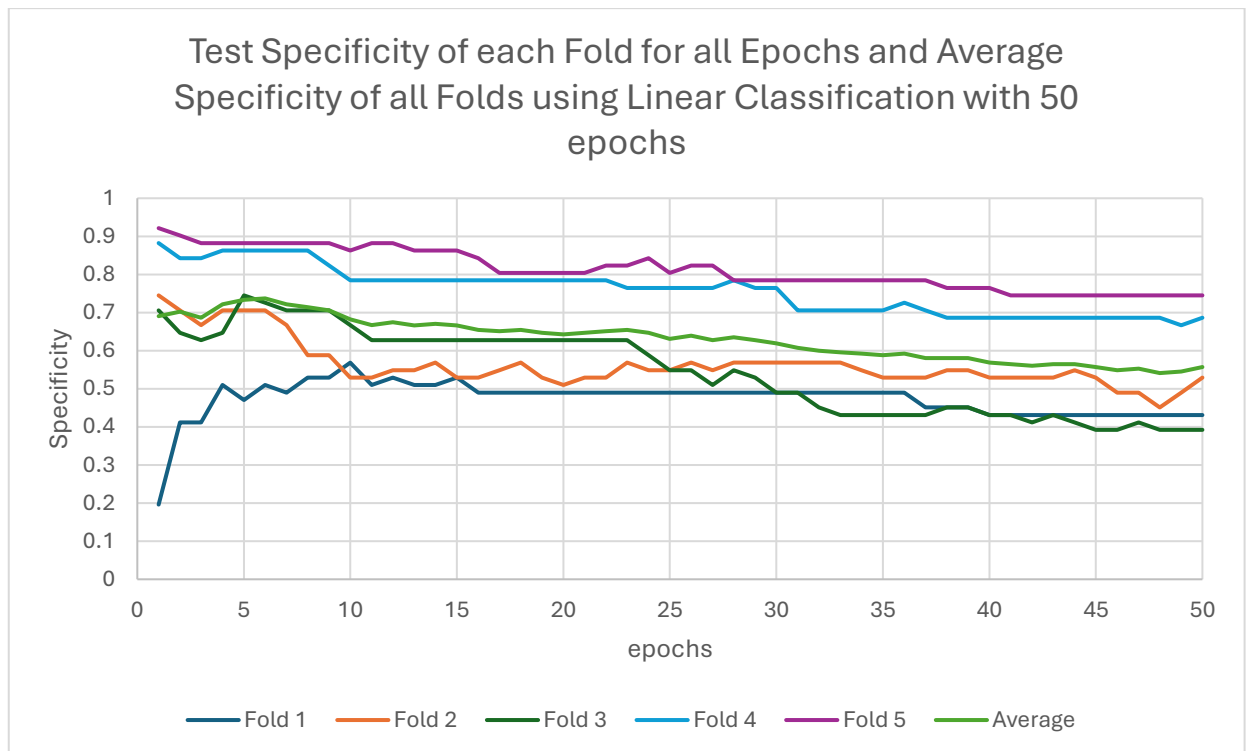
Fold 2 had a more disappointing start, beginning at a much lower sensitivity of 0.42 at epoch 1. However, it can be seen that there was a rapid improvement by epoch 5, rising to 0.57, and further to 0.71 by epoch 7. Between epochs 10 and 30, Fold 2 maintained a relatively steady sensitivity between 0.71 and 0.79, with some minor fluctuations. After epoch 30, sensitivity slightly improved again reaching around 0.85 during the final epochs. Overall, Fold 2 displayed a strong recovery after an initially poor start and stabilized relatively well towards the end.

Additionally, for Fold 3, sensitivity started at 0.57 during epoch 1 and remained stable until epoch 8. From epoch 10 and further, a gradual but clear improvement was observed, with sensitivity climbing from 0.57 to around 0.71 by epoch 20. After epoch 30, Fold 3 showed even stronger performance, achieving 0.78–0.85 sensitivity between epochs 30 and 50, with occasional peaks at 0.93. This indicates that Fold 3 progressively learned better representations for detecting cancerous cases as training progressed.

The model in Fold 4 displayed a different pattern. Sensitivity started at 0.43 at the first epoch and stayed stuck around that value for the first 20 epochs. It was only after epoch 20 that Fold 4 managed a minor boost, reaching around 0.50 sensitivity. After epoch 30, it showed slight improvements, achieving up to 0.57–0.64 sensitivity toward the later epochs. Nonetheless, Fold 4 consistently underperformed compared to the other folds, failing to reach the clinically desired sensitivity threshold ( $\geq 0.90$ ) at any point during training.

Fold 5 had the worst sensitivity evolution among all folds. It started extremely low at 0.21 and remained practically stagnant around 0.21–0.28 for the first 30 epochs. Some small improvements were observed between epochs 30–50, where sensitivity rose to around 0.35–0.57, but overall, performance remained inadequate for reliable cancer detection. Despite minor late improvements, Fold 5 never approached acceptable sensitivity levels.

The average sensitivity across all folds began at around 0.50 during the early epochs. From epoch 5 to epoch 20, it slightly increased to around 0.58–0.60, reflecting the initial improvements seen mostly in Folds 1–3. Beyond epoch 30, a clearer upward trend appeared, and by epochs 40–50 the average sensitivity reached values between 0.71–0.77, showing that although sensitivity improved throughout training, the progress was highly dependent on a few strong folds. This indicates that while the model was eventually able to enhance its detection of positive cases in some folds, it struggled to consistently generalize this ability across all folds, emphasizing the instability of performance regarding cancer detection sensitivity.



**Figure 5.8:** Test Specificity per epoch and Average Specificity, across all folds using the Linear Classification Head with 50 epochs.

Moving on to test specificity evolution, it can be observed from the linear graphs above that for Fold 1, test specificity started extremely low, standing at 0.19 during the first epoch. However, it rapidly increased by epoch 5, reaching around 0.47. After epoch 5, a slow but steady rise was observed, with specificity improving further to 0.50–0.53 around epochs 10 to 20. Nonetheless, after epoch 20, specificity plateaued around 0.49–0.53, and by epoch 50, it slightly dropped to approximately 0.43, indicating a slight deterioration in the model’s ability to correctly identify healthy cases towards the end of training.

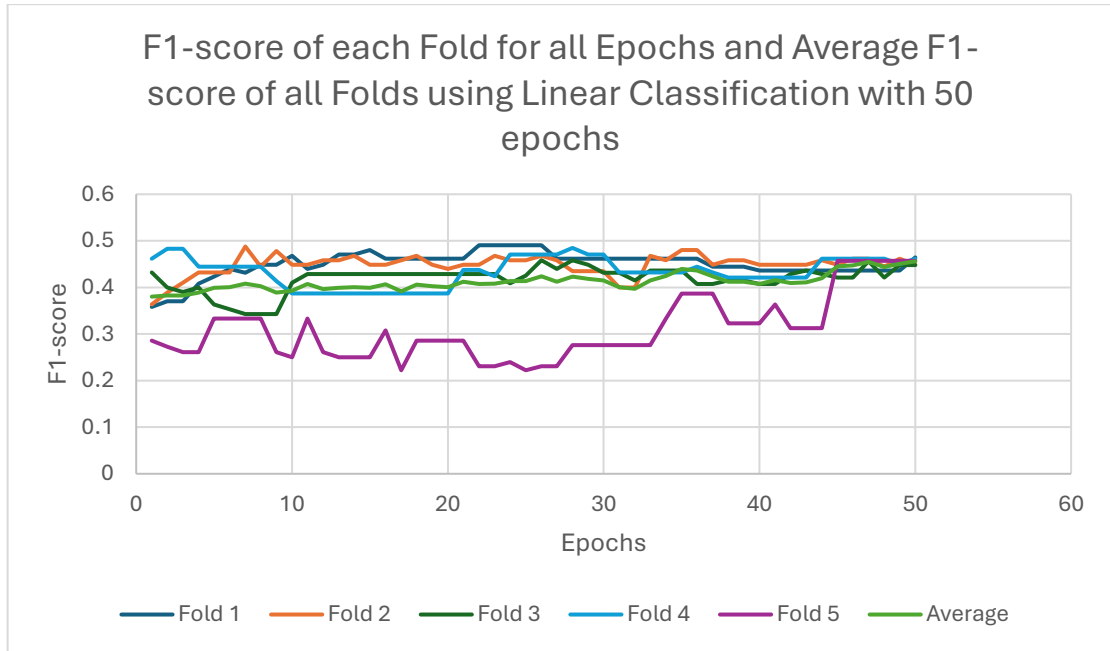
Fold 2 started considerably high at 0.74 during the first epoch. However, a gradual decline was observed early on, with specificity dropping to around 0.66 by epoch 5. Between epochs 10 and 20, Fold 2 fluctuated between 0.52–0.59, showing a steady downward trend. After epoch 20, specificity continued decreasing slightly, fluctuating between 0.49–0.56 until epoch 50. Overall, the trend for Fold 2 was relatively unstable and downward, indicating a weakening performance in detecting healthy cases as training progressed.

Additionally, Fold 3 began with a promising specificity of 0.70 at epoch 1. However, a gradual decline was observed, with specificity dropping to around 0.62–0.66 between epochs 10 to 20. More specifically, after epoch 25, a more significant drop occurred, and by epochs 30 to 50, Fold 3’s specificity hovered around 0.49–0.52, showing a clear degradation in performance as training continued.

The model in Fold 4 displayed the most stable specificity performance among all folds. It started at a high value of 0.88 at epoch 1 and maintained a strong performance throughout the 50 epochs. Although minor fluctuations were present, specificity remained relatively stable, fluctuating between 0.78–0.86 throughout the whole training. This indicates that Fold 4 consistently preserved its ability to correctly classify healthy cases.

Fold 5 had the best overall specificity performance. It started very high at 0.92 during epoch 1 and maintained exceptionally strong results throughout training. Even though slight declines were observed after epoch 20, specificity remained high, consistently staying between 0.74–0.88 by epoch 50. Fold 5 was the most reliable in maintaining high specificity during the entire training phase.

The average test specificity across all folds began at around 0.69 at epoch 1 and remained relatively stable until epoch 10. However, after epoch 10, a gradual downward trend can be seen, with average specificity dropping slowly but steadily toward 0.54–0.57 by epoch 50. This suggests that although some folds, particularly Fold 4 and Fold 5, maintained strong specificity throughout training, the model generally struggled to preserve healthy case identification across all folds, especially in folds with poorer initial performance.



**Figure 5.9:** Test F1-score per epoch and Average F1-score, across all folds using the Linear Classification Head with 50 epochs.

Figure 5.9 presents the evolution of the F1-score for each fold across 50 training epochs, alongside the average F1-score across all five folds. The graph provides a visual representation of model performance consistency and convergence behavior using a linear classification head trained on top of frozen DINOv2 ViT-S/14 features.

From the data, it is evident that the F1-scores generally increase during the early epochs (especially the first 10–15), reflecting the model's learning progression. After this initial phase, most folds demonstrate a stabilization in performance with minor fluctuations. Notably, Fold 5 consistently underperforms relative to the others across majority of the epochs. This suggests that the data distribution in Fold 5 may be more challenging, potentially containing more difficult or ambiguous samples or exhibiting a greater class imbalance.

In contrast, Folds 1 to 4 display relatively stable and higher F1-scores, generally oscillating between 0.42 and 0.48. Fold 1 shows the most stable performance overall, with minimal variation throughout training. Fold 2 reaches the highest F1-score of 0.48 multiple times, showing strong generalization within that subset.

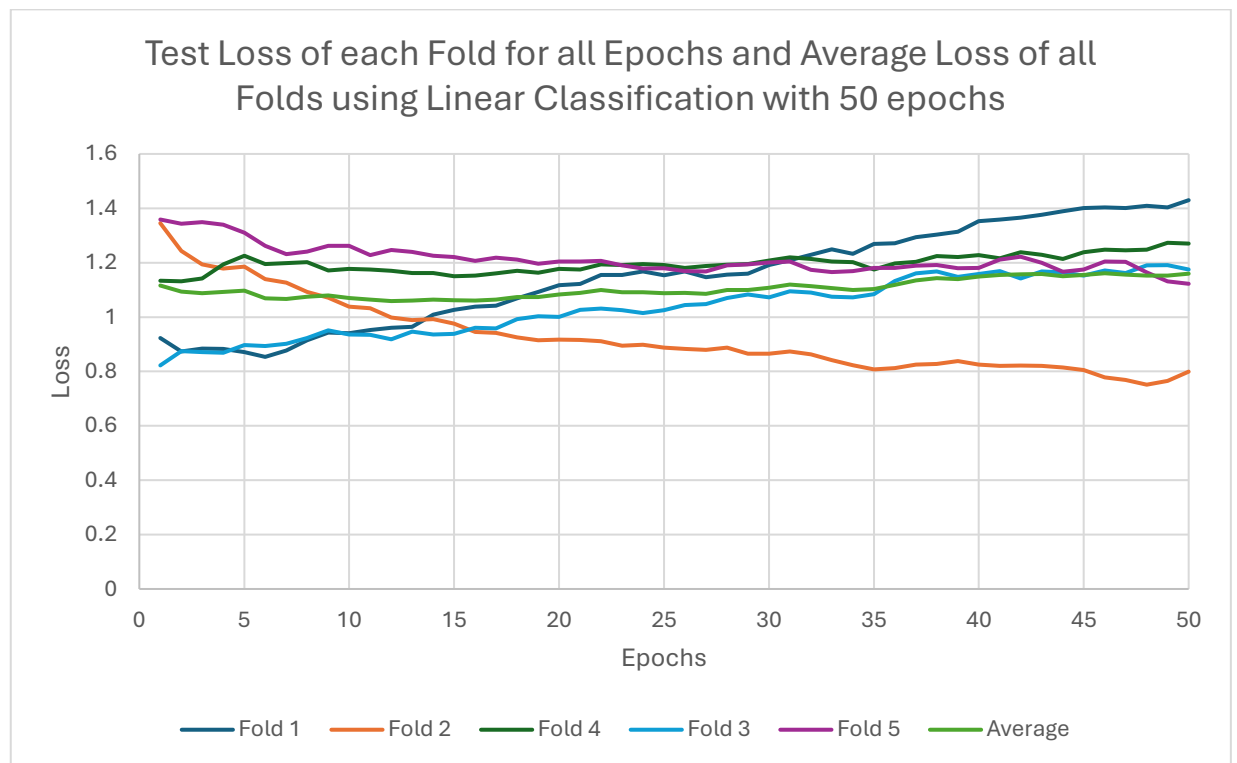
The average F1-score curve (green line) is smoother due to aggregation and steadily improves until around epoch 35. From epoch 35 onward, the average F1-score stabilizes around 0.44–0.45, indicating convergence of the model's learning capacity under the



current configuration. The final average F1-score achieved at epoch 50 is 0.4560, suggesting satisfactory but not perfect class discrimination capability.

Importantly, despite the relatively low variance among Folds 1–4, the consistently lower F1-scores of Fold 5 negatively impact the overall average, highlighting the importance of robust generalization strategies when training on imbalanced or heterogeneous datasets.

This evaluation confirms that the linear classification head, while effective in many cases, may not fully capture complex decision boundaries in more difficult data partitions. This insight motivated further experimentation using non-linear alternatives such MLPs, as explored in subsequent sections.



**Figure 5.10:** Test Loss per epoch and Average Loss, across all folds using the Linear Classification Head with 50 epochs.

Considering test loss evolution, it can be observed from the linear graphs above that for Fold 1, the test loss started relatively low, standing at 0.92 during the first epoch. Then, a small decrease was observed by epoch 5, dropping to around 0.87. Between epochs 10 and 20, loss remained fairly stable, fluctuating between 0.87–0.95. However, after epoch 20, a gradual but clear upward trend appeared, with test loss increasing steadily

to 1.35 by epoch 50. This steady rise in loss indicates that Fold 1 began to overfit the training data after the initial epochs.

Fold 2 started off significantly higher than Fold 1, with a test loss of 1.35 during the first epoch. However, a rapid decrease was observed early on, and by epoch 10 loss dropped to approximately 1.03. After epoch 10, Fold 2 maintained the lowest test loss out of all folds, staying consistently between 0.76–0.99 until epoch 50. Fold 2 displayed the most stable and favourable loss evolution, showing no major overfitting signs even at later stages.

Additionally, for Fold 3, the test loss started quite low at 0.82 during the first epoch. Although a small increase was seen between epochs 5–10 (rising to around 0.87–0.95), Fold 3 maintained relatively stable loss values throughout training. From epochs 20 to 50, the loss gently increased from 0.95 up to about 1.17 by epoch 50. This mild but steady increase suggests that Fold 3 gradually lost some generalization power over time.

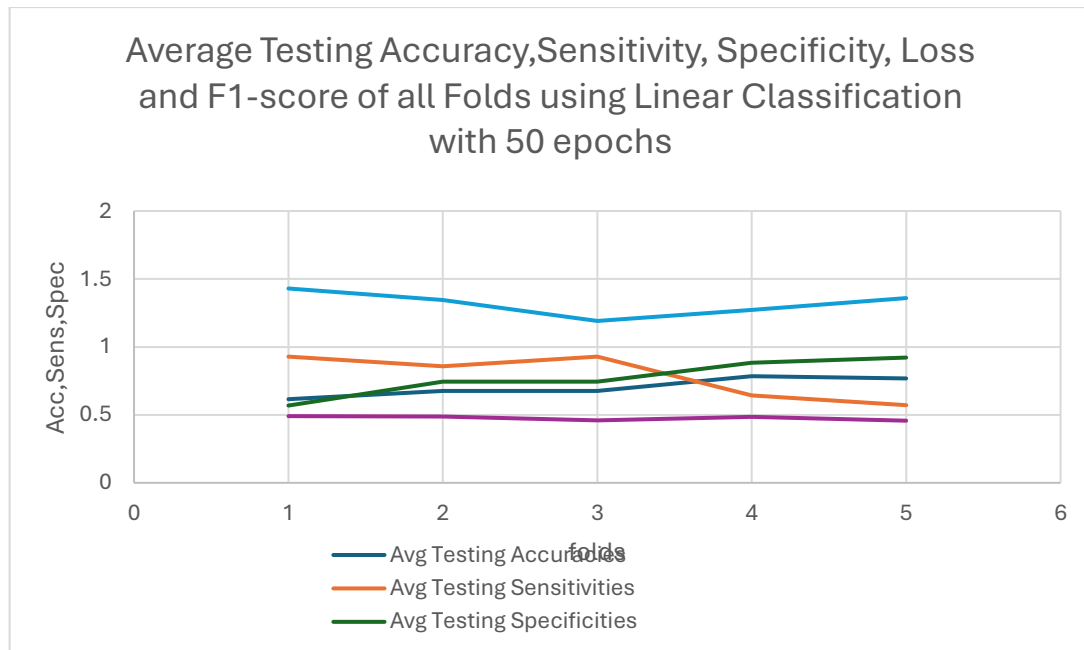
The model in Fold 4 had a different behaviour. Test loss started higher at 1.13 during the first epoch and remained fairly consistent, fluctuating mildly between 1.13–1.22 throughout most of the training phase. After epoch 35, a slight rise was observed, with loss reaching around 1.27 at epoch 50. Nonetheless, compared to the other folds, Fold 4 maintained a fairly controlled test loss across training, with only minor overfitting signs towards the end.

Fold 5 showed the highest initial loss at 1.36 during epoch 1. Although it slightly decreased to about 1.24–1.25 by epoch 10, loss gradually increased again after epoch 20, reaching up to 1.27 at epoch 50. Fold 5 consistently exhibited high uncertainty throughout training, aligning with its poor sensitivity performance, and indicating that the model struggled the most with this specific fold.

The average test loss across all folds started at approximately 1.12 at epoch 1. From epoch 5 to epoch 20, the average loss fluctuated mildly between 1.06–1.08, showing good stability during early training. However, after epoch 20, a gradual and consistent upward trend was observed, with average loss increasing steadily to about 1.15–1.16 by epoch 50. This steady rise reflects that generalization on the unseen test set became weaker over time, especially after 20–30 epochs, suggesting once again that prolonged

training might not have been beneficial, and that earlier stopping could have helped prevent the observed overfitting trends.

An equally important visualization of the performance metrics is given in the following graph, which presents a direct comparison of the average(meaning maximum out of each fold) testing Accuracy, Sensitivity, Specificity, and Loss across all five folds after training the Linear Classification Head for 50 epochs.



**Figure 5.11:** Average Test Accuracy, Sensitivity, Specificity, F1-score and Loss Across All Folds Using the Linear Classification Head with 50 epochs.

It can be observed that testing accuracy remained moderate, ranging from 0.6154 in Fold 1 to a maximum of 0.7846 in Fold 4, reflecting relatively stable but not exceptional generalization to unseen data.

Testing sensitivity, which is the clinical priority of this study, showed strong performance in the early folds, achieving high values of 0.9286 in Folds 1 and 3 and 0.8571 in Fold 2. However, a noticeable drop occurred in Folds 4 and 5, where sensitivity fell to 0.6429 and 0.5714, respectively, indicating that the model struggled to maintain consistent detection of cancerous cases across all splits.

On the other hand, specificity demonstrated a clear upward trend, improving steadily from 0.5686 in Fold 1 to 0.9216 in Fold 5. This suggests that while the model became more confident in correctly classifying healthy cases over time, it did so at the expense of missing more positive cases, highlighting a sensitivity-specificity trade-off.

Finally, testing loss values remained relatively high across folds, fluctuating between

1.191 and 1.4299, further supporting the observation that although the model fit the training data well, it exhibited notable uncertainty when predicting on unseen test samples.

Regarding the F1-score, which balances both sensitivity and precision, the model demonstrated relatively stable but moderate performance across all five folds. The highest F1-score was achieved in Fold 1 at 0.4906, closely followed by Fold 2 at 0.4878 and Fold 4 at 0.4848. These results indicate that the model was generally able to maintain a reasonable balance between correctly identifying cancerous cases and avoiding false positives in these splits. However, the lowest F1-score was observed in Fold 5 (0.4571), which aligns with its previously discussed sensitivity drop. The overall consistency of the F1-scores, with a narrow range between 0.4571 and 0.4906, suggests that while the linear classifier exhibited stable predictive behaviour across different data partitions, it lacked the capacity to reach high performance thresholds.

Given these observations, it can be concluded that the results can be considered partially aligned with the primary goal of this thesis, which was to maintain high sensitivity across all folds. While in Folds 1, 2, and 3 sensitivity values were excellent, achieving 0.9286, 0.8571, and 0.9286 respectively and thus satisfying the clinical objective of minimizing false negatives, in Folds 4 and 5, sensitivity dropped significantly to 0.6429 and 0.5714, falling well below the desired threshold of 0.9.

This inconsistency across folds shows that although the model was capable of reaching high sensitivity in certain splits, it failed to maintain this performance reliably across the entire dataset. As a result, the model's overall behaviour cannot be considered fully satisfactory for clinical application in the real world of medical practice, where consistently high sensitivity is critical for dependable early breast cancer detection.

### **5.2.2 Performance at 150 Epochs**

To further explore the behaviour of the Linear Classification Head and evaluate the impact of prolonged training, an additional experimental setting was conducted by extending the number of training epochs to 150. The following table summarizes the maximum achieved values across the 150 epochs for each fold in terms of key evaluation metrics, Accuracy, Sensitivity, Specificity, and Loss for the training, validation, and testing phases.

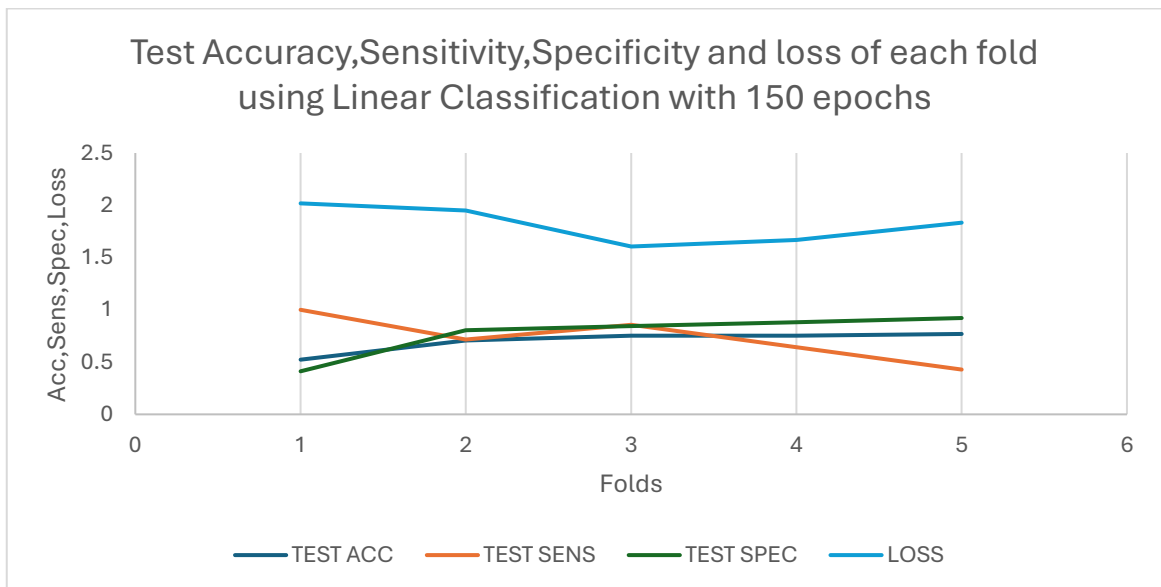
Special attention remains focused on sensitivity as the primary objective, while specificity and loss are also carefully monitored to evaluate the model's ability to correctly identify healthy cases and maintain overall prediction confidence.

By comparing these results to the 50-epoch experiments, insights can be gained into whether extended training helped improve model generalization or instead led to overfitting and degraded performance on the unseen test set.

Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss
1	0.6699	1	0.575	5.6126	0.5769	1	0.4878	0.8726	0.5231	1	0.4118	2.0183
2	1	1	1	2.5604	0.9808	0.9167	1	0.4043	0.7077	0.7143	0.8039	1.9501
3	1	1	1	1.3828	0.9808	0.9167	1	0.5115	0.7538	0.8571	0.8431	1.6056
4	1	1	1	1.18	0.9808	0.9091	1	0.1294	0.7538	0.6429	0.8824	1.6673
5	1	1	1	0.4709	0.9804	0.9091	1	0.2852	0.7692	0.4286	0.9216	1.8339

**Table 5.2:** Performance metrics obtained with training, validation and testing, of Linear Classification Head with 150 epochs

The following graph presents a summary of the final achieved Testing Accuracy, Sensitivity, Specificity, and Loss across all five folds after training the Linear Classification Head for 150 epochs. The focus of this graph is on a compact comparison of the overall best test performances per fold. This visualization helps assess whether prolonged training duration led to improvements or degradation in the model's ability to generalize, particularly focusing on sensitivity, which remains the primary evaluation priority.



**Figure 5.12:** Average Test Accuracy, Sensitivity, Specificity and Loss Across All Folds Using the Linear Classification Head with 150 epochs.

From the graph and the summarized results, it can be observed that testing loss remained the highest value across all folds, fluctuating between 1.6056 and 2.0183, suggesting that uncertainty during testing persisted even after extended training. In terms of testing accuracy, Fold 1 performed the worst, achieving only 0.5231, while Fold 5 achieved the best performance with an accuracy of 0.7692. Nonetheless, accuracy values remained moderate overall and did not indicate substantial generalization improvements compared to the 50-epoch setting.

As far as testing sensitivity is concerned, which is the primary objective of this thesis, it is clear that the model failed to maintain consistently high values. Although Fold 1 reached a perfect 1.0 sensitivity, fully identifying all cancerous cases, a sharp drop was observed in the subsequent folds. Fold 2 achieved a sensitivity of 0.7143, Fold 3 improved slightly to 0.8571, but Folds 4 and 5 deteriorated further, falling to 0.6429 and 0.4286 respectively. These values show that prolonged training did not lead to an overall better sensitivity outcome. In fact, sensitivity stability worsened, with only Folds 1 and 3 approaching the desired  $\geq 0.90$  target.

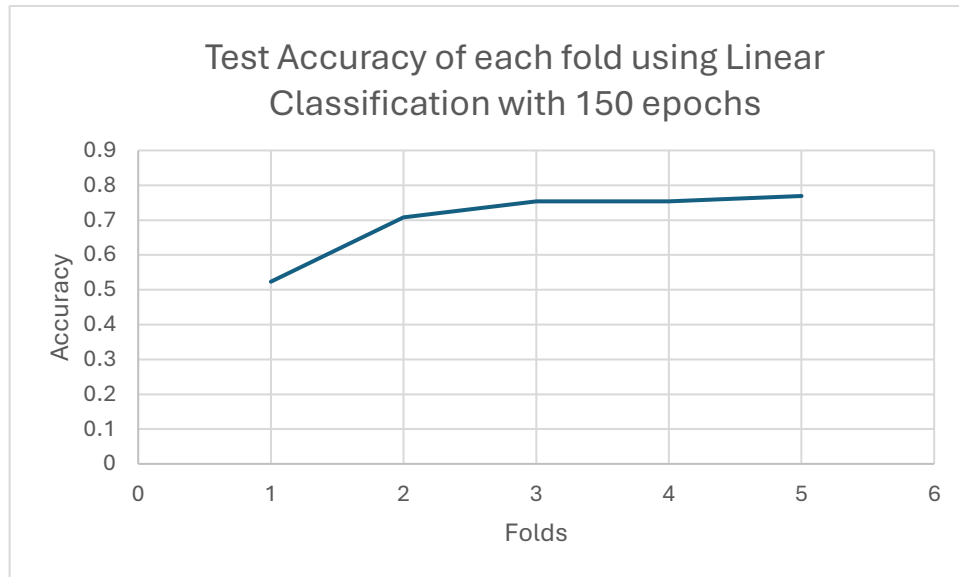
On the other hand, testing specificity demonstrated a continuous upward trend across folds. Starting from 0.4118 in Fold 1, specificity increased steadily, reaching up to 0.9216 in Fold 5. This suggests that the model increasingly prioritized correctly classifying healthy individuals as training continued. However, this improvement in specificity came at the expense of sensitivity, creating a trade-off that is clinically undesirable in early breast cancer detection, where minimizing false negatives is the critical priority.

Overall, while specificity showed improvements and accuracy remained acceptable, sensitivity dropped significantly in multiple folds, particularly in Folds 4 and 5. Thus, the 150-epoch training setting cannot be considered fully satisfactory in relation to the main goal of this study, and signs of overfitting and sensitivity degradation became more apparent as training was prolonged.

#### **Accuracy:**

The following graph illustrates the final Testing Accuracy achieved by the Linear Classification Head across all five folds after 150 epochs. This visualization provides insight into how accurately the model generalized to unseen data in each fold after

prolonged training. It also allows comparisons between folds in terms of generalization stability.



**Figure 5.13:** Average Test Accuracy Across All Folds Using the Linear Classification Head with 150 epochs.

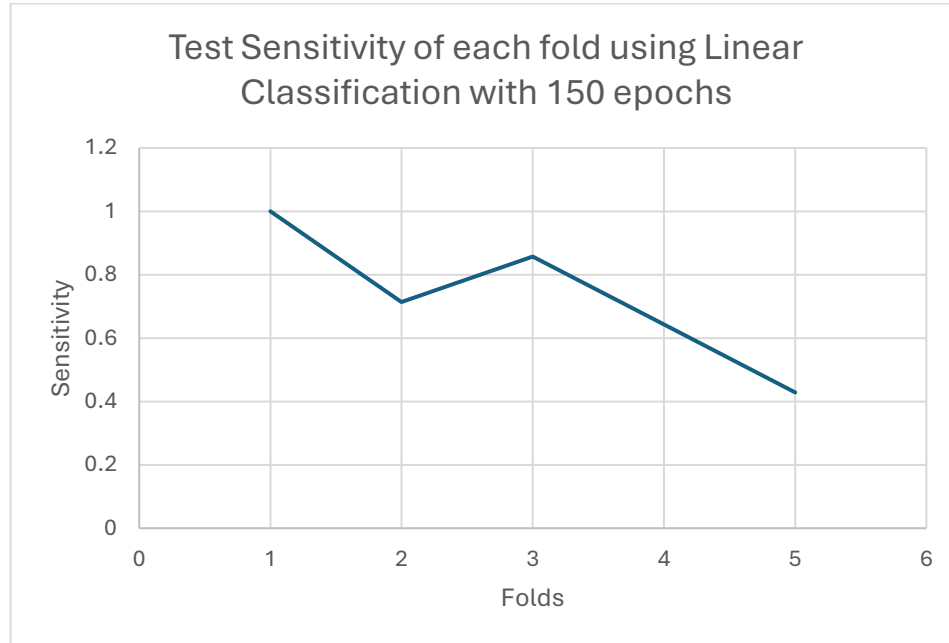
As shown in the graph, Fold 1 exhibited the lowest test accuracy, achieving only 0.5231 after 150 epochs. In contrast, a significant improvement was observed from Fold 2 onwards, with Fold 2 reaching 0.7077, and Folds 3 and 4 both achieving identical values of 0.7538. Fold 5 demonstrated the highest test accuracy overall, finishing at 0.7692.

The clear upward trend from Fold 1 to Fold 5 suggests that the model's ability to correctly classify both healthy and unhealthy cases improved across folds as training progressed. However, the relatively lower performance in Fold 1 highlights fold-dependent variability, which is consistent with small dataset settings where splits can introduce subtle distribution shifts.

While these accuracy values are acceptable and even relatively high in folds 3–5, it is important to note that accuracy alone does not fully reflect the model's medical reliability in this case. High accuracy may still hide critical sensitivity failures, particularly in folds where cancerous cases are under detected. Therefore, while the general trend in accuracy was positive with 150 epochs, it must be interpreted cautiously alongside sensitivity trends to draw meaningful clinical conclusions.

### Sensitivity:

The following graph presents the Test Sensitivity achieved across all five folds when training the Linear Classification Head for 150 epochs. As sensitivity is the primary clinical objective of this thesis, this graph is critical for understanding the model's ability to correctly detect cancerous cases under extended training conditions.

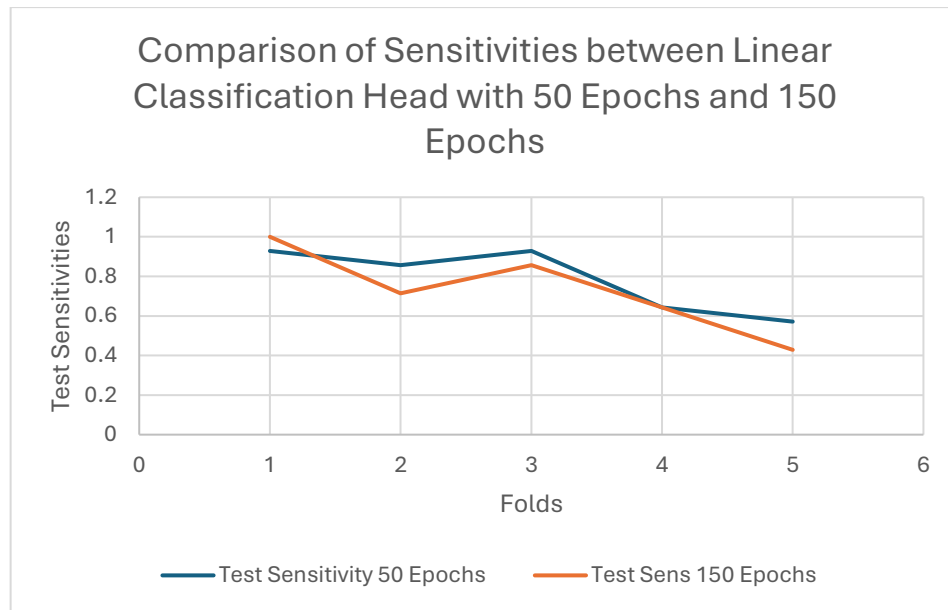


**Figure 5.14:** Average Test Sensitivity Across All Folds Using the Linear Classification Head with 150 epochs.

By observing the graph above which represents the test sensitivity evolution of the 5 folds after 150 epochs, Fold 1 achieved perfect sensitivity, reaching a value of 1.000. This indicates that the model managed to detect all cancerous cases correctly within this fold, fully aligning with the clinical goal of minimizing false negatives. Fold 2, however, demonstrated a notable drop compared to Fold 1, with sensitivity settling at 0.7143. While this value is still moderate, it falls short of the ideal threshold ( $\geq 0.90$ ) desired for clinical application. Fold 3 showed a relatively good recovery, achieving a sensitivity of 0.8571, which, although slightly lower than perfect detection, remains clinically acceptable. Moving on, Fold 4 displayed a sensitivity of 0.6429, which closely mirrors its earlier performance at 50 epochs, indicating that prolonged training did not offer any real improvement for this split. Finally, Fold 5 recorded the lowest sensitivity across all folds, at 0.4286, highlighting a major failure in cancer case detection for this split and raising concerns regarding the model's reliability when exposed to certain data distributions.



Furthermore, a second graph is included to compare the sensitivities obtained after 50 epochs versus 150 epochs, allowing a direct evaluation of how prolonged training influenced sensitivity preservation across folds.



**Figure 5.15:** Sensitivity Comparison Across All Folds between the Linear Classification Head with 50 and 150 epochs.

When directly comparing these results to those obtained after 50 epochs, several important observations emerge. In Fold 1, sensitivity slightly improved, moving from 0.9286 at 50 epochs to a perfect 1.000 at 150 epochs. This shows that prolonged training benefitted Fold 1 specifically. In contrast, Fold 2 experienced a clear decline, with sensitivity dropping from 0.8571 to 0.7143. Similarly, Fold 3's sensitivity decreased slightly from 0.9286 to 0.8571, although it remained relatively strong. For Fold 4, sensitivity stayed identical at 0.6429 regardless of training duration, suggesting that additional epochs did not lead to further gains. Meanwhile, Fold 5 exhibited a considerable deterioration, as sensitivity fell from 0.5714 at 50 epochs to just 0.4286 at 150 epochs, marking the most dramatic sensitivity decline among all folds.

Given these observations, it can be concluded that prolonged training to 150 epochs did not consistently improve sensitivity across folds. Instead, sensitivity either declined, stagnated, or, in the best cases, showed only marginal improvement. Most importantly, the overall goal of maintaining high sensitivity ( $\geq 0.90$ ) across all folds was not achieved. This underlines that simply increasing the number of epochs was not sufficient to guarantee better clinical reliability, and in fact, prolonged training may have contributed to reduced generalization ability in certain folds.

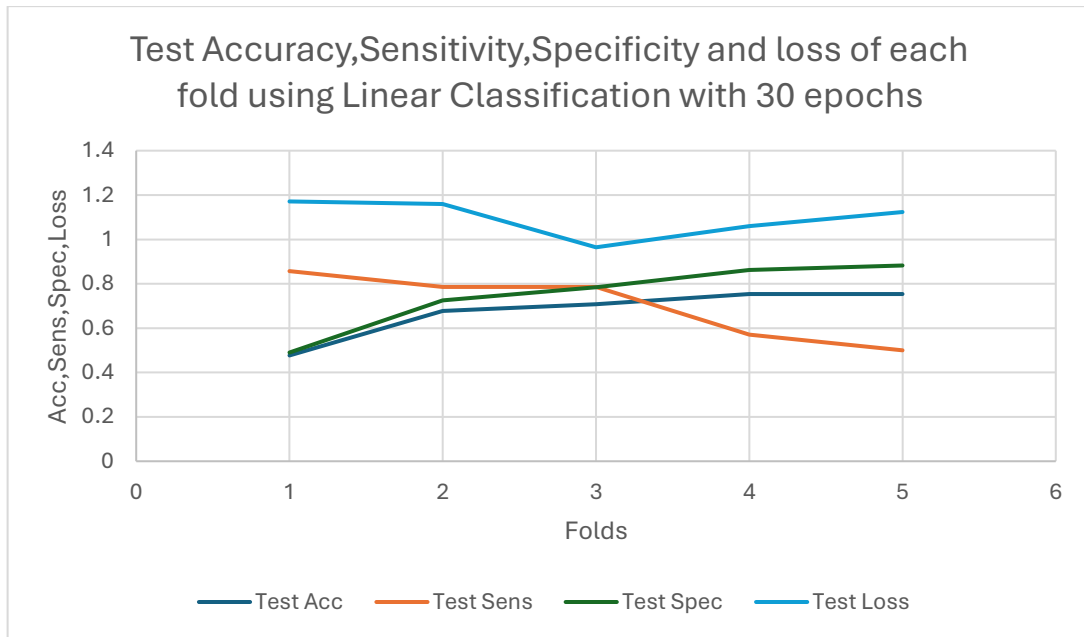
### 5.2.3 Performance at 30 Epochs

Since the previous experiment with prolonged training at 150 epochs did not result in improved performance and, in fact, revealed signs of overfitting and sensitivity degradation, it was decided to explore the opposite approach by reducing the training duration. In this setting, the number of training epochs was decreased from the original 50 to just 30, with the goal of evaluating whether shorter training could enhance stability, reduce overfitting, and potentially preserve sensitivity better across all folds. The following table summarizes the maximum achieved values across the 30 epochs for each fold, presenting key evaluation metrics including Accuracy, Sensitivity, Specificity, and Loss for the training, validation, and testing phases.

Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss
1	0.6553	1	0.5563	5.6089	0.6731	0.9091	0.6341	0.7091	0.4769	0.8751	0.4902	1.1711
2	0.8883	1	0.9068	3.5323	0.9423	0.9167	0.95	0.5027	0.6769	0.7857	0.7255	1.1597
3	0.9563	1	0.9565	2.7999	0.9615	0.8333	1	0.5602	0.7077	0.7857	0.7843	0.9645
4	0.9855	0.9565	1	2.2853	0.9804	0.9091	1	0.2772	0.7538	0.5714	0.8627	1.0596
5	0.9903	0.9565	1	1.8875	0.9804	0.9091	1	0.4471	0.7538	0.5	0.8824	1.1232

**Table 5.3:** Performance metrics obtained with training, validation and testing, of Linear Classification Head with 30 epochs

To further investigate whether earlier stopping could better preserve sensitivity and improve generalization, the number of training epochs was reduced to 30. The following graph summarizes the testing phase results for each fold in terms of Accuracy, Sensitivity, Specificity, and Loss after 30 epochs of training. By examining these trends, it becomes possible to understand whether a shorter training schedule helped the model avoid overfitting and maintain clinically important detection capabilities.



**Figure 5.16:** Average Test Accuracy, Sensitivity, Specificity and Loss Across All Folds Using the Linear Classification Head with 30 epochs.

As seen in the figure and the table above, the results show mixed outcomes.

Starting with Fold 1, the test accuracy was relatively low at 0.4769, but sensitivity remained quite strong at 0.8571, aligning well with the primary goal of minimizing false negatives. However, specificity was poor at 0.4902, indicating that many healthy cases were incorrectly classified as sick. Test loss was relatively high at 1.1711, suggesting that the model struggled to generalize well in this fold.

Fold 2 displayed a noticeable improvement in both accuracy (0.6769) and specificity (0.7255) compared to Fold 1, while still maintaining a relatively good sensitivity of 0.7857. Test loss slightly decreased to 1.1597. This indicates that a better balance between detecting positives and negatives was achieved in Fold 2.

Similarly, Fold 3 further improved in accuracy (0.7077) and specificity (0.7843), while maintaining the same sensitivity (0.7857) as Fold 2. Moreover, Fold 3 had the lowest test loss among all folds at 0.9645, showing that the model achieved better generalization with 30 epochs in this case.

Moving on to Fold 4, accuracy continued to rise to 0.7538 and specificity also improved to 0.8627, which is a strong value. However, sensitivity dropped significantly to 0.5714, indicating that although the model became better at recognizing healthy cases, it missed

many cancerous ones, a trade off that is undesirable for the clinical objective. Test loss remained relatively low at 1.0596.

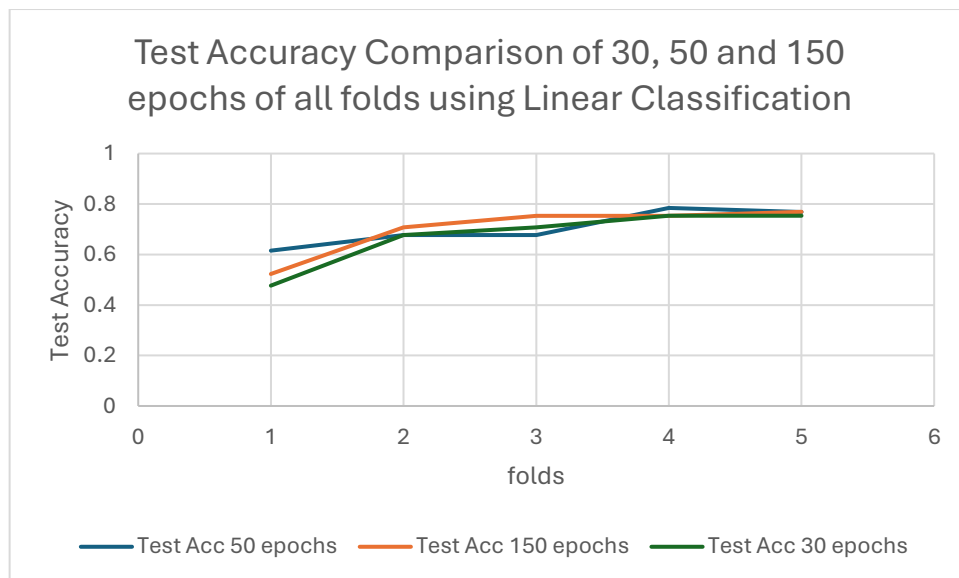
Finally, Fold 5 maintained similar accuracy (0.7538) and even stronger specificity (0.8824) compared to Fold 4. However, sensitivity dropped even further to 0.5000, which is critically low for reliable cancer detection. Test loss slightly increased to 1.1232, suggesting that despite stable accuracy and specificity, the model's overall confidence weakened.

In conclusion, reducing the training duration to 30 epochs preserved high sensitivity in the first few folds (particularly Fold 1), but performance degraded significantly in Folds 4 and 5, where sensitivity became too low to meet clinical reliability standards.

Although test loss values were generally lower than at 50 or 150 epochs, the inconsistency in sensitivity across folds indicates that 30 epochs did not fully solve the issue of balancing sensitivity and specificity.

#### **Accuracy:**

The following graph presents the comparison of test accuracies across all folds for the 30, 50 and 150 epoch experiments conducted using DINOv2. The corresponding numerical results are also summarized in the table below the graph.



**Figure 5.17:** Test Accuracy Comparison Across All Folds Using the Linear Classification Head with 30 , 50 and 150 epochs.

<b>Fold</b>	<b>Test Accuracy 30 Epochs</b>	<b>Test Accuracy 50 Epochs</b>	<b>Test Accuracy 150 Epochs</b>
1	0.4769	0.6154	0.5231
2	0.6769	0.6769	0.7077
3	0.7077	0.6769	0.7538
4	0.7538	0.7846	0.7538
5	0.7538	0.7692	0.7692

**Table 5.4:** Testing Accuracy Comparison for all folds, of Linear Classification Head, obtained with all 3 epoch experiments, 30, 50 and 150 epochs respectively.

Starting with Fold 1, the best accuracy was achieved after 50 epochs with a value of 0.6145. After extending training to 150 epochs, test accuracy dropped significantly to 0.5231 and further decreased to 0.4769 after training for only 30 epochs. Therefore, in Fold 1 neither prolonging nor reducing the number of epochs benefited generalization, and in fact both adjustments resulted in worse performance compared to the original 50 epochs.

In Fold 2, test accuracy improved slightly from 0.6769 at 50 and 30 epochs to 0.7077 at 150 epochs. An interesting observation is that even when training was reduced to 30 epochs the test accuracy remained stable at 0.6769 which is identical to the result achieved with 50 epochs. This shows that Fold 2 was relatively stable across different training durations although minor gains were observed at 150 epochs.

Fold 3 showed similar behaviour to Fold 2. Accuracy rose from 0.6769 at 50 epochs to 0.7538 at 150 epochs, suggesting some benefit from extended training. However, with only 30 epochs, accuracy reached 0.7077, which while slightly lower than 150 epochs, was still better than the initial 50-epoch result. This suggests that Fold 3 may have benefitted somewhat from both extended and reduced training, with longer training yielding slightly higher accuracy.

In Fold 4, the highest test accuracy was achieved at 50 epochs, with a value of 0.7846. Extending training to 150 epochs led to a small drop to 0.7538. Reducing training to 30 epochs also resulted in a test accuracy of 0.7538. Therefore, for Fold 4, prolonged training caused minor overfitting, while early stopping at 30 epochs managed to retain good performance without significant degradation.

Finally, in Fold 5, test accuracy at 50 epochs was 0.7692 which remained unchanged after 150 epochs. When training was reduced to 30 epochs, accuracy slightly dropped to

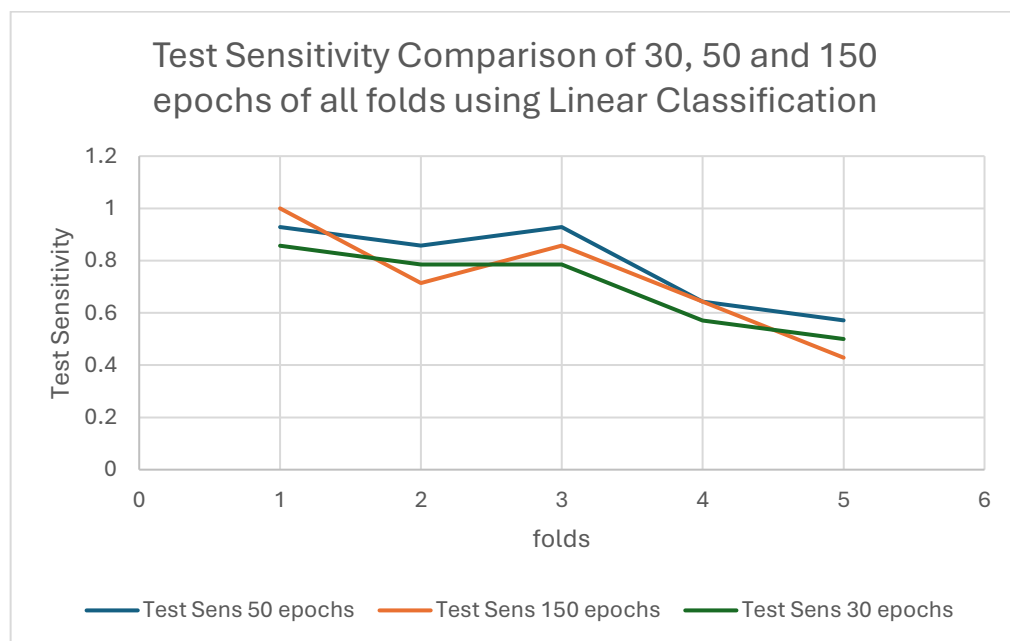
0.7538. This shows that prolonged training neither improved nor hurt Fold 5, but early stopping led to a minor decline.

Overall, comparing all folds, it can be concluded that training for 150 epochs slightly improved test accuracy in Folds 2 and 3, but hurt Folds 1 and 4, and had no effect in Fold 5. Additionally, reducing training to 30 epochs preserved or slightly worsened test accuracy compared to 50 epochs in most folds. Lastly, Fold 1 consistently performed worse under both 30 and 150 epochs, indicating its instability.

Thus, in terms of test accuracy alone 50 epochs appeared to offer the best overall balance, with 150 epochs occasionally helping, but not reliably across all folds. Early stopping at 30 epochs was not significantly beneficial and sometimes slightly hurt accuracy.

### Sensitivity:

As far as sensitivity comparison is concerned, it can be observed from the graph and numerical table below that overall, none of the three epoch settings, meaning 30, 50 and 150 epochs managed to consistently maintain high sensitivity across all folds.



**Figure 5.18:** Test Sensitivity Comparison Across All Folds Using the Linear Classification Head with 30, 50 and 150 epochs.

<b>Fold</b>	<b>Test Sensitivity 30 Epochs</b>	<b>Test Sensitivity 50 Epochs</b>	<b>Test Sensitivity 150 Epochs</b>
1	0.8571	0.9286	1
2	0.7857	0.8571	0.7143
3	0.7857	0.9286	0.8571
4	0.5714	0.6429	0.6429
5	0.5	0.5714	0.4286

**Table 5.5:** Testing Sensitivity Comparison for all folds, of Linear Classification Head, obtained with all 3 epoch experiments, 30, 50 and 150 epochs respectively.

Starting with Fold 1, sensitivity after 50 epochs stood at 0.9286 which is a very strong sensitivity value. Increasing training to 150 epochs, slightly improved the results to a perfect 1.00 successfully achieving zero FNs for this fold. However, decreasing the number of epochs to 30, caused a slight drop to 0.8571 which even though is relatively high and clinically acceptable it is still lower than the values of 50 and 150 epochs.

In Fold 2, sensitivity after 50 epochs was 0.8571. However, when training was extended to 150 epochs, sensitivity degraded notably to 0.7143. Similarly, at 30 epochs sensitivity stood at 0.7857. Thus, shorter training appeared slightly better than prolonged training here, although none of the settings managed to surpass the original 50-epoch sensitivity.

For Fold 3, sensitivity after 50 epochs was very high standing at 0.9286. When training was prolonged to 150 epochs, sensitivity dropped to 0.8571, and after reducing to 30 epochs, it fell further to 0.7857. Therefore, both prolonged and reduced training worsened sensitivity compared to the 50-epoch value.

Fold 4 showed relatively weak sensitivity even at 50 epochs, standing at 0.6429. Training for 150 epochs did not bring any improvements, as sensitivity remained the same at 0.6429. Reducing training to 30 epochs made the situation worse, with sensitivity declining to 0.5714. Thus, Fold 4 remained a persistent challenge under all epoch settings, but shorter training degraded performance even further.

Finally, in Fold 5, sensitivity at 50 epochs was 0.5714, already below the clinically desired threshold. Extending training to 150 epochs led to a further major decline, with sensitivity dropping critically to 0.4286. With 30 epochs, sensitivity was slightly better at 0.5 but remained clinically insufficient. Fold 5 was therefore the weakest fold across all scenarios.

Therefore, comparing the sensitivity values across folds, it is clear that 50 epochs provided the highest and most reliable sensitivity values. Both increasing the epochs to 150 and decreasing to 30 worsened sensitivity in most cases. Particularly, prolonged training (150 epochs) often caused severe sensitivity degradation (e.g., Fold 2 and Fold 5), while shortened training (30 epochs) mildly worsened or stagnated sensitivity performance. Thus, neither longer nor shorter training consistently benefitted the model's ability to detect positive (cancerous) cases, further emphasizing that 50 epochs was overall the most balanced and effective training length for maintaining clinical reliability.

#### **5.2.4 Analysis of Misclassified Cases Affecting Sensitivity with Linear Head**

In order to better understand the causes behind low sensitivity observed in a much larger degree in Folds 4 and 5, a deeper inspection was conducted by extending the code to print and record all misclassified samples, both healthy predicted as unhealthy and unhealthy predicted as healthy, during the testing phase.

After obtaining the results, particular focused was placed on FNs, meaning cases where a cancerous image was incorrectly classified as healthy, since these are the errors that directly impact and lower sensitivity which is the main clinical priority of this thesis.

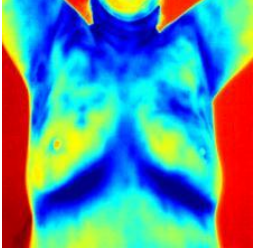
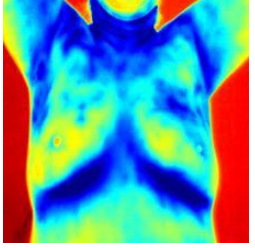
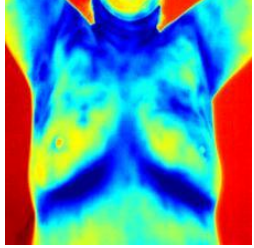
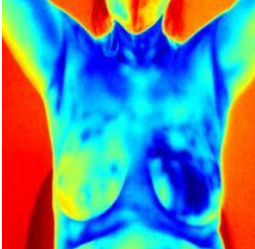
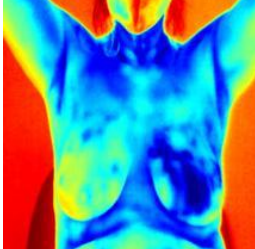
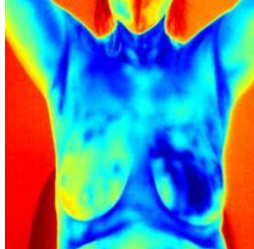
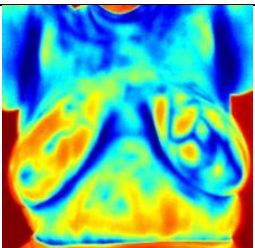
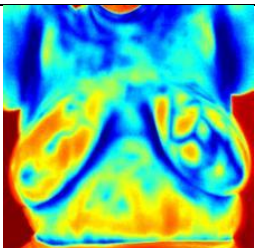
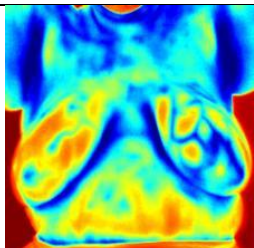
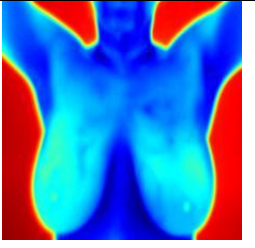
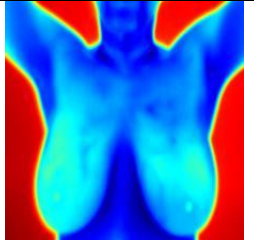
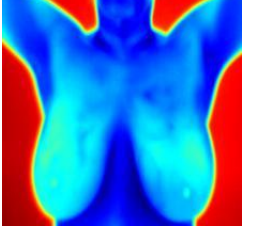
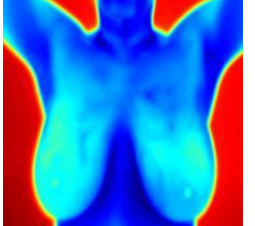

Given that Folds 4 and 5 were consistently showing the worst sensitivity results across the 30, 50 and 150 epoch experiments, they were selected for this targeted investigation. For each setting of epochs, aka 30, 50 and 150, the model was run and all misclassified test samples were extracted. Specifically, all the images predicted as healthy despite being actually unhealthy were collected for each fold and epoch setting.


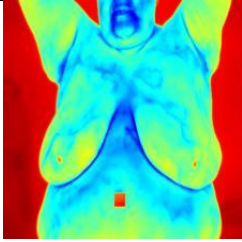
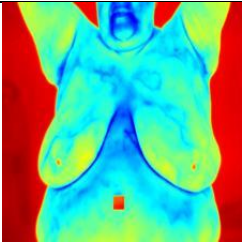
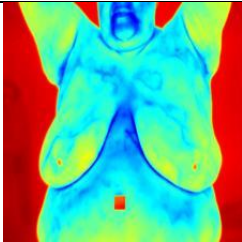
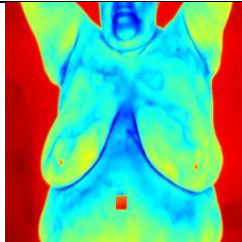
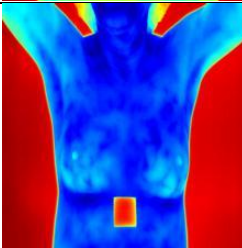
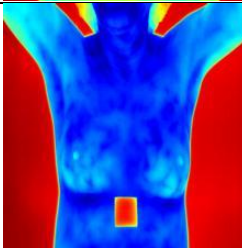
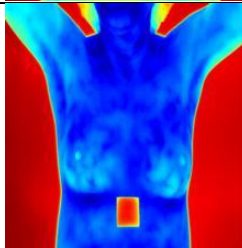
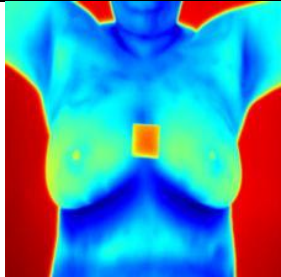
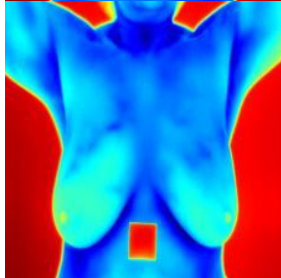
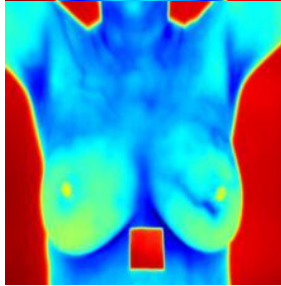
Moreover, the recorded FNs were compared across epoch variations to identify whether certain images were consistently misclassified regardless of the training duration. By identifying and visually inspecting the repeated FN samples, it became possible to assess whether specific characteristics in these images contributes to the model's difficulty in correctly classifying them.

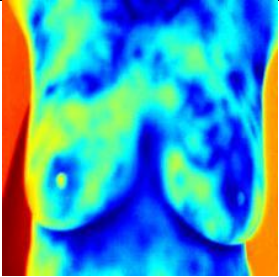
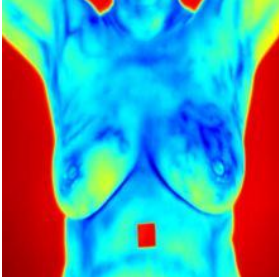


To complement the misclassification analysis described above, this section visualizes the images that were consistently misclassified in Folds 4 and 5 when running the Linear Classification Head across all three training settings.

The images corresponding to each fold and training setting are provided in the table below:

Fold	Misclassified Image using 30 Epochs	Misclassified Image using 50 Epochs	Misclassified Image using 150 Epochs	Correlation
4, 5				Same Image for all epoch settings
4, 5				Same Image for all epoch settings
4, 5				Same Image for all epoch settings
4		Other Image(s)		Same Image for 30 and 150 epochs
5				Same Image for all epoch settings

4			Other Image(s)	Same Image
5				Same Image
5				Same Image
5	Other Image(s)	Other Image(s)	  	Different Images within the same (150) epoch experiment

5		Other Image(s)	Other Image(s)	
4		Other Image(s)	Other Image(s)	

**Table 5.6:** Repeated FN Samples Across Epoch Variations in Folds 4 and 5, with Linear Classification Head.

The table above presents all the misclassified test samples, specifically FNs in Folds 4 and 5 that were consistently misclassified across all three training durations, aka 30, 50, and 150 epochs. These images represent the most problematic cases for the Linear Head, as they were incorrectly predicted as healthy regardless of how long the model was trained. In other words, they reflect samples that the model failed to classify correctly under any condition, highlighting their inherent difficulty.

Where the table shows “Other Image(s)”, it indicates that there were additional false negatives for that specific epoch setting, but these varied from the consistent cases and were therefore omitted for summarization purposes. The focus of the table is on images that repeatedly failed across all settings, as these are the most informative for analysing persistent sensitivity issues.

When analysing the false negatives (FNs) across Folds 4 and 5 for all three training durations (30, 50, and 150 epochs) as seen in the Table 5.6, a consistent pattern was observed. Several specific images were misclassified in almost every epoch setting. These repeated misclassifications suggest that these particular samples are inherently difficult for the model to classify, possibly due to subtle or ambiguous thermal features. Visual inspection reveals that many of these images exhibit symmetry, low contrast, or thermal patterns resembling those typically associated with healthy subjects, which may mislead the model. This trend highlights the limitations of the current linear

classification head in capturing the complexity of certain cases. In contrast, images with more distinct abnormalities may have been easier to classify correctly. These findings emphasize the need for further investigation, potentially incorporating non-linear models or region-specific attention mechanisms to better capture critical diagnostic features.

### 5.2.5 Hyperparameter and Architectural Exploration for the Linear Head

Throughout the development of the linear classification head used for breast cancer detection via DINOv2 features, multiple hyperparameters and configurations were explored in hopes of optimizing the model’s performance. Given the critical importance of minimizing false negatives to ensure high sensitivity, all experiments prioritized sensitivity over specificity, while still aiming to maintain acceptable specificity levels.

The finalized version of the pure linear classifier is based on a single fully connected (nn.Linear) layer attached on top of frozen DINOv2 ViT-S/14 [CLS] token embeddings. Features are normalized using StandardScaler, and the model is trained using the AdamW optimizer with BCEWithLogitsLoss, employing a class imbalance-aware pos\_weight=1.35. During training, threshold optimization is applied to maximize F1-score while ensuring sensitivity  $\geq 0.90$  on the validation set. Performance is monitored per epoch across training, validation, and final test sets.

#### Historical Modifications and Rationale for Discarding

The final design was reached after testing and discarding multiple alternative setups, each of which is detailed below:

Change Attempted	Final Status	Reason for Discarding
Focal Loss	Discarded	Did not significantly improve sensitivity and it complicated optimization
FocalLoss(alpha=0.5, gamma=2)	Discarded	Flattened gradients and increased training instability for minority class.
pos_weight=dynamic value	Discarded	Used $\text{len}(y\_train)/(2*y\_train.sum())$ ; unstable across folds.

Thresholding using Youden's J index <sup>[25]</sup>	Discarded	Prioritized balance between sensitivity/specificity but often violated $\text{sens} \geq 0.90$
Thresholding using $(2 * \text{TPR} - \text{FPR})$	Discarded	Led to increased false negatives.
Threshold = 0.5 fallback	Fallback only	Retained only when no threshold achieves $\geq 0.90$ sensitivity.
Final test set evaluated once	Discarded	Misclassified Samples were not tracked per epoch and it lacked robustness
Final test set evaluated per epoch	Adopted	Allowed per-epoch tracking of generalization and misclassification behaviour
Best threshold optimized using F1 under sensitivity constraint	Adopted	Maintained priority on recall while improving precision and <u>F1-score</u>
Early stopping on validation specificity	Discarded	Prioritized specificity instead of overall validation balance

**Table 5.7:** Summary of Historical Modifications to the Linear Classification Head and Rationale for Discarding.

### Specific Hyperparameters Tested

Loss Function:

Initial tests included Focal Loss with  $\alpha=0.5$ ,  $\gamma=2$ . Although Focal Loss is commonly used in imbalanced settings, it introduced significant training instability, and gradients tended to vanish in cases of ambiguous inputs. The specificity did not improve sufficiently to justify its complexity, leading to its rejection.

The final configuration uses BCEWithLogitsLoss with a fixed positive class weight ( $\text{pos\_weight}=1.35$ ) to counteract class imbalance more robustly and consistently across all folds.

### Optimizer and Learning Rate:

The optimizer remained constant as AdamW, as it provided stable convergence. The learning rate of  $1e-3$  was experimentally confirmed as the optimal value. Lower rates slowed convergence, while higher rates induced oscillation in early epochs.

### Threshold Optimization:

Several methods for determining the classification threshold were considered:

Youden’s J index ( $\text{tpr} + \text{specificity} - 1$ )<sup>[25]</sup>, often failed to preserve the sensitivity constraint of  $\geq 0.90$ , despite good F1.

A custom metric emphasizing sensitivity ( $2 \times \text{TPR} - \text{FPR}$ ) occasionally yielded high sensitivity but at the cost of excessively low specificity.

The final method filters all thresholds with  $\text{TPR} \geq 0.90$  and selects the one that maximizes F1-score among them. This balanced recall and precision under the strict sensitivity constraint and significantly reduced false negatives across folds.

### **Final Test Evaluation Frequency:**

Initially, final test set performance was only assessed once after training completion. This did not allow misclassification dynamics to be studied over time.

The new approach evaluates the final test set at **every epoch**, tracking true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) continuously. This enabled real-time error analysis and refinement.

### **Early Stopping:**

Some intermediate versions introduced early stopping when validation specificity stagnated. However, since the main goal was to maximize sensitivity, this criterion contradicted the primary objective and was discarded.

### **Evaluation and Impact:**

All experimental configurations were benchmarked across 5-fold cross-validation using fixed training/validation/final-test splits. The improvements to thresholding and logging in the final design led to the best trade-off between sensitivity and false positive reduction, which was not possible under earlier configurations.

## **5.3 MLP Classification Head Results**

### **5.3.1 Performance at 50 Epochs**

To evaluate the effectiveness of the MLP Classification Head trained on top of frozen DINOv2 features, this section presents a table summarizing the best achieved performance across 50 training epochs for each fold. The evaluation includes key

metrics such as Accuracy, Sensitivity, Specificity, and Loss, reported for the training, validation, and final test sets.

The primary focus remains on sensitivity, which is clinically the most critical metric in the context of early breast cancer detection. An improved sensitivity indicates the model's enhanced ability to correctly identify cancerous cases, thereby reducing false negatives, which is considered a central objective of this thesis. The inclusion of specificity and loss provides additional insight into the model's discriminative capability and prediction stability.

By analyzing the fold-wise results of the MLP head and comparing them to the earlier linear baseline, this section aims to assess whether the non-linear architecture of the MLP contributes to improved classification performance, especially on difficult cases observed in prior evaluations.

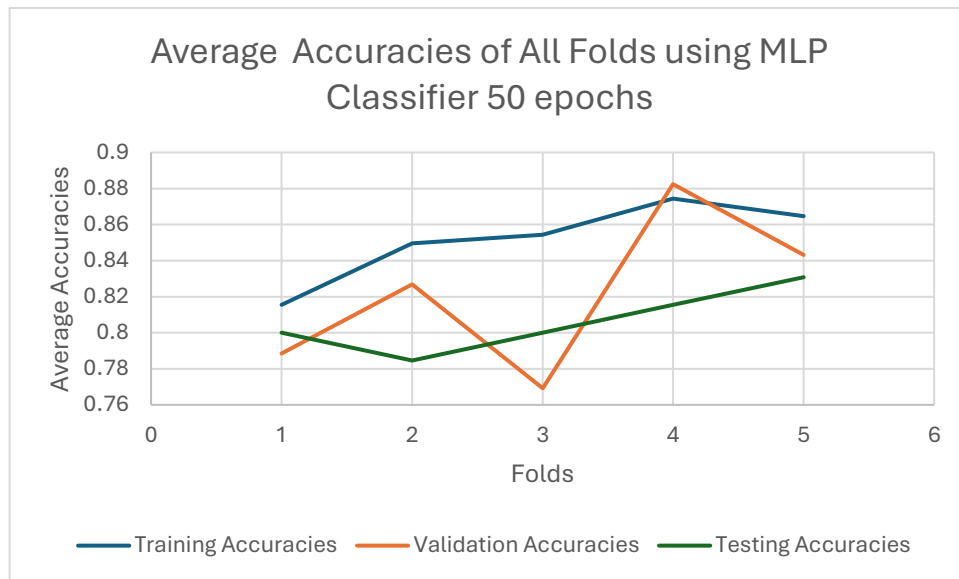
Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss	Test F1-Score
1	0.8155	0.8913	0.9812	6.438	0.7885	0.9091	1	1.9144	0.8	0.9286	1	2.2616	0.4444
2	0.8495	0.8889	0.9379	5.9469	0.8269	1	1	1.6161	0.7846	0.9286	0.9804	1.8563	0.5714
3	0.8544	0.9556	0.8758	4.9854	0.7692	0.5	1	0.20516	0.8	0.8571	0.9804	1.7295	0.5263
4	0.8744	0.9783	0.9068	4.9542	0.8824	0.8182	1	1.6674	0.8154	0.7143	0.9804	1.9845	0.5263
5	0.8647	0.9348	0.8758	4.6421	0.8431	0.3636	1	2.267	0.8308	0.5	0.9804	2.3136	0.5

**Table 5.8:** Performance metrics obtained with training, validation and testing, of MLP Classification Head with 50 epochs

In addition to the summarized table of maximum achieved values, the following linear graphs provide a more detailed and continuous view of the MLP Classification Head's testing performance over all 50 training epochs, across each individual fold. These graphs illustrate the evolution of the four key evaluation metrics, Accuracy, Sensitivity, Specificity, and Loss throughout testing. Unlike the table, which captures only the best point achieved in each fold, these plots reveal the complete learning behaviour and stability of the model over time.

The table and line graphs below, illustrate the trends in average (meaning maximum of each fold) accuracy, sensitivity, specificity and loss across all five folds using the MLP

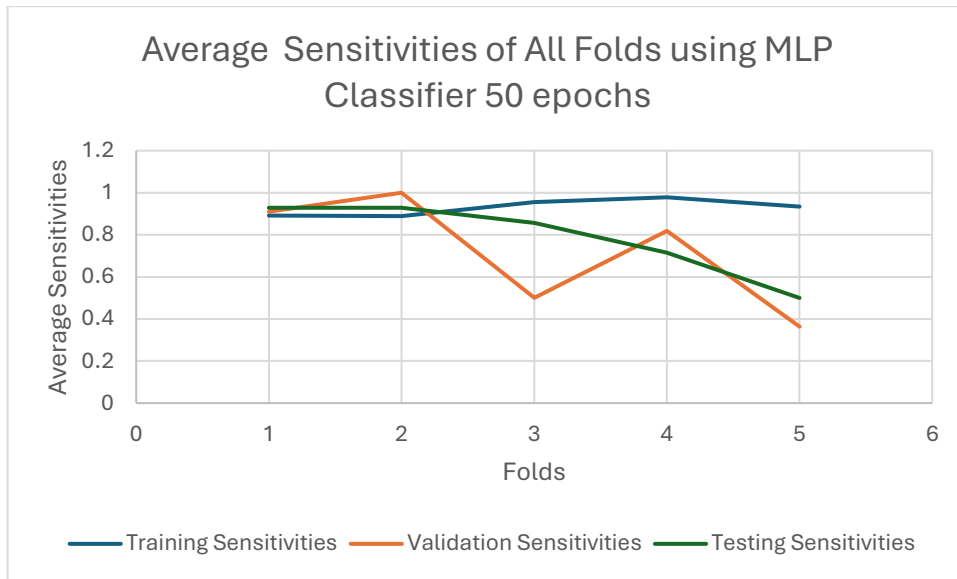
Classification Head trained for 50 epochs. These trends are complemented by the fold-wise numerical values presented in the table.



**Figure 5.19:** Average Accuracy Values of All Folds when using the MLP Classifier with 50 epochs.

Starting with accuracy, training performance steadily improved across folds. More specifically, it began at 0.8155 in Fold 1 and rose to a peak of 0.8744 in Fold 4, then followed closely by 0.8647 in Fold 5. This trend indicates that the MLP model was able to learn effectively across all training splits. Validation accuracy, while generally aligned with training trends, showed more variability. As seen in the table and graph above, it ranged from a low of 0.7692 in Fold 3 to a high of 0.8824 in Fold 4, suggesting that certain folds contained more challenging validation subsets. Test accuracy, which is the most reliable indicator of generalization, ranged from 0.7846 in Fold 2 to 0.8308 in Fold 5, with values in Folds 1 and 3 sitting at 0.8000, and 0.8154 in Fold 4. Unlike the linear model, where test accuracy remained consistently lower, the MLP classifier showed a narrower gap between validation and test performance, with overall higher and more stable test accuracies. This implies better generalization and reduced overfitting, likely due to the MLP's capacity to model non-linear patterns that the linear classifier could not capture. The strongest overall performance appeared in Fold 5, where all three accuracy values (training: 0.8647, validation: 0.8431, test: 0.8308) were among the highest and closely aligned, which implies a strong sign of both fit and generalization.





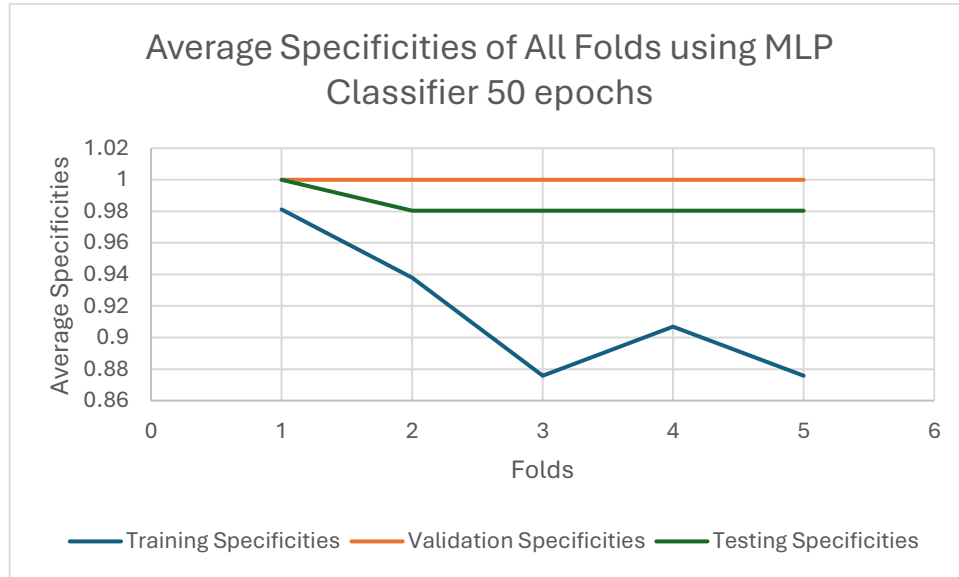
**Figure 5.20:** Average Sensitivity Values of All Folds when using the MLP Classifier with 50 epochs.

Sensitivity, the metric most aligned with this thesis’ clinical objective of minimizing false negatives in cancer detection, demonstrated strong results overall, though with more variability compared to the linear classifier. In the training phase, sensitivity remained high across all folds, ranging from 0.8889 in Fold 2 to a peak of 0.9783 in Fold 4, indicating that the MLP classifier consistently learned to identify unhealthy cases within the training set.

Validation sensitivity, however, showed greater fluctuations. While it reached perfect sensitivity (1.0) in Fold 2 and maintained a strong 0.9091 in Fold 1, it dropped to 0.8182 in Fold 4, 0.5 in Fold 3, and a concerning 0.3636 in Fold 5. This suggests that the MLP struggled to generalize sensitivity performance to certain validation subsets, possibly due to fold-specific sample difficulty or limited representation of cancerous cases in those splits.

In the testing phase, the model achieved excellent sensitivity in Folds 1 and 2 (0.9286 each), and a moderately strong 0.8571 in Fold 3. However, sensitivity declined to 0.7143 in Fold 4 and dropped further to 0.5 in Fold 5, echoing a similar pattern observed with the linear classifier. These consistent underperformances in the final folds highlight potential limitations in the model’s ability to generalize across all test distributions, possibly influenced by subtle domain shifts or class imbalance.

Despite the non-linear capacity of the MLP head, these results suggest that improvements in sensitivity are not guaranteed across all folds and achieving reliable sensitivity on difficult or underrepresented subsets remains an ongoing challenge.

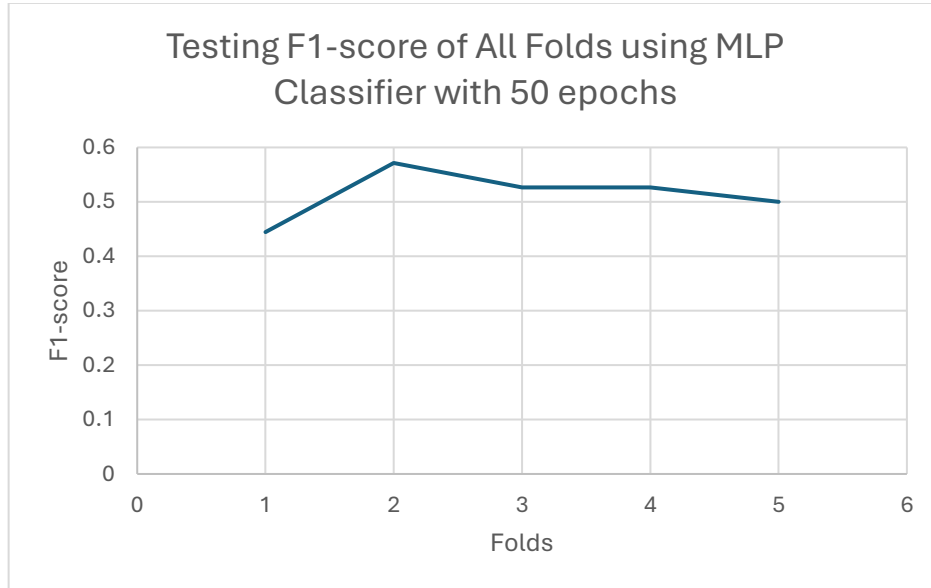


**Figure 5.21:** Average Specificity Values of All Folds when using the MLP Classifier with 50 epochs.

Specificity which measures the model’s ability to correctly identify healthy cases, showed exceptionally strong and stable performance in both the validation and testing phases. In the validation set, specificity was consistently perfect (1.0) across all five folds, indicating that the MLP classifier did not misclassify a single healthy image as unhealthy during validation, regardless of the fold. This trend extended to the test set, where Fold 1 achieved a perfect score of 1.0, and the remaining folds maintained a still-excellent specificity of 0.9804, reflecting a high degree of reliability in avoiding false positives.

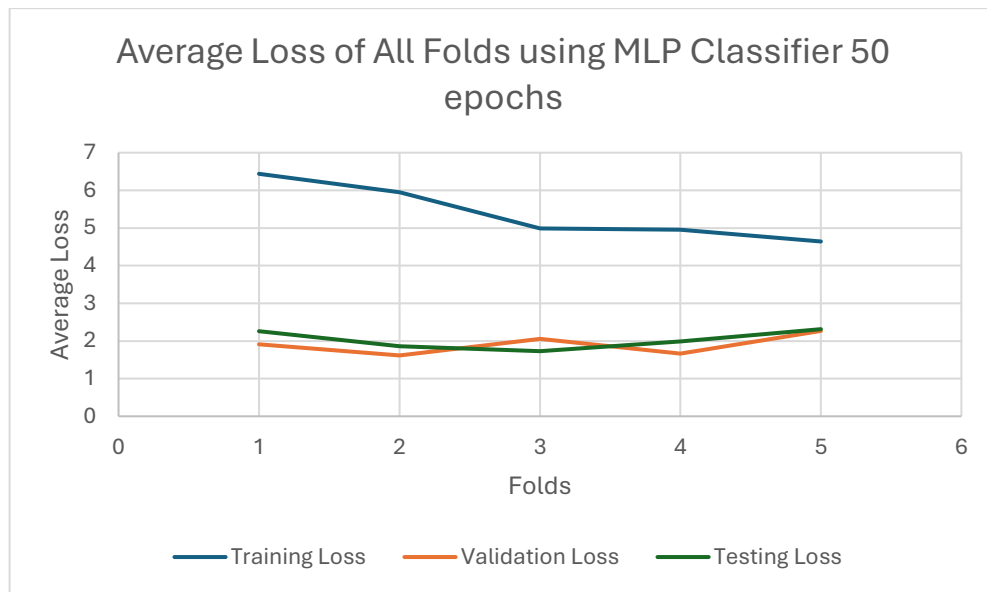
In contrast, training specificity exhibited slightly more variation, starting at 0.9812 in Fold 1 and gradually decreasing in subsequent folds, reaching a low of 0.8758 in Folds 3 and 5. This drop in training specificity may reflect the model’s effort to prioritize sensitivity during training, especially in folds where class balance or sample difficulty posed a challenge. Nevertheless, the consistency of high specificity in both validation and test phases confirms that the MLP classifier was not overfitting to negative (healthy) cases and maintained strong discriminative ability on unseen data.

Overall, the results suggest that while sensitivity was fold-dependent, specificity remained robust and consistently high, which is considered a promising outcome for real-world screening applications where reducing false alarms is essential to avoid unnecessary patient anxiety and additional testing.



**Figure 5.22:** Average F1-score Values of All Folds when using the MLP Classifier with 50 epochs.

The figure above, presents the F1-scores obtained on the final test sets across all five cross-validation folds using the Multi-Layer Perceptron (MLP) classification head after 50 training epochs. The recorded F1-scores range from 0.444 to 0.571, indicating moderate effectiveness in balancing sensitivity and precision. The highest F1-score was achieved in Fold 2 (0.571), suggesting a relatively optimal balance between true positive and false positive rates for that particular data split. Conversely, Fold 1 exhibited the lowest F1-score (0.444), which may be attributed to a larger number of misclassifications, either in the form of false negatives or false positives. The remaining folds (Folds 3 to 5) demonstrated stable and comparable performance, with F1-scores of approximately 0.526 and 0.500, respectively. Overall, the consistency of F1-scores across the majority of folds reflects the MLP classifier's ability to generalize across different subsets of the data. However, the moderate magnitude of these scores suggests that, while high sensitivity was set to be maintained through the thresholding strategy, the precision remains limited, thereby constraining the F1-score.



**Figure 5.23:** Average Loss Values of All Folds when using the MLP Classifier with 50 epochs.

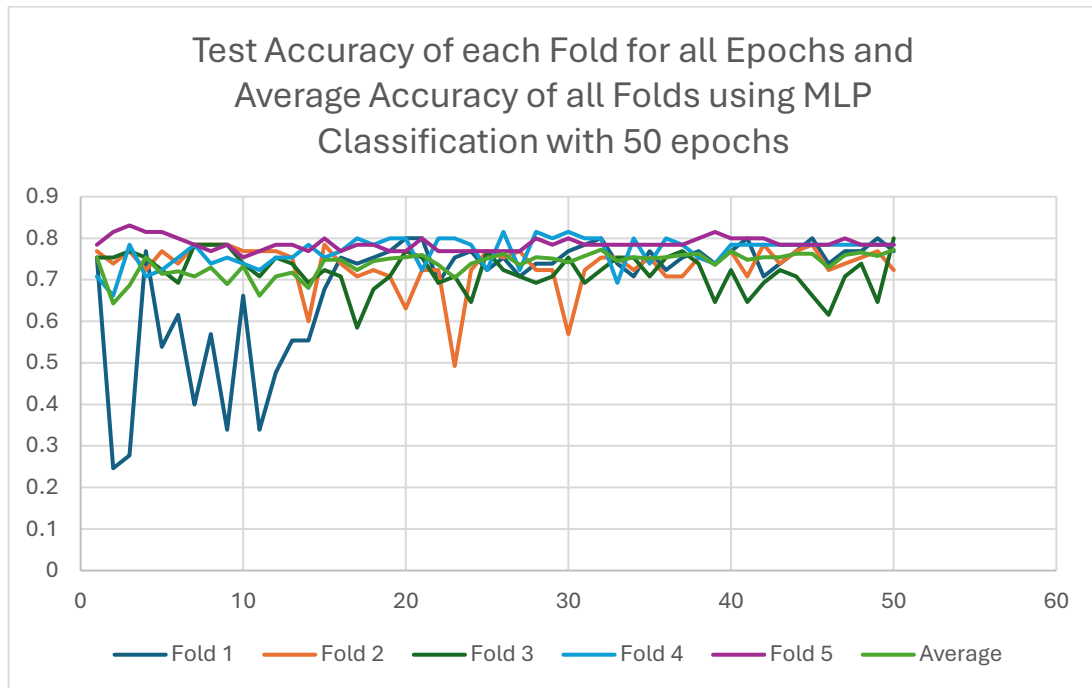
Loss values provide insight into the model's confidence and how well it fits the data. When observing the loss values of the MLP Classification Head it is clear that the training loss consistently decreased across folds, starting at 6.438 in Fold 1 and dropping to 4.6421 in Fold 5. This downward trend suggests a progressive improvement in the model's ability to minimize error on the training data, possibly due to better weight convergence or more favourable fold-specific training splits.

In contrast, validation loss remained relatively stable but fluctuated across folds, ranging from a low of 1.6161 in Fold 2 to a high of 2.2670 in Fold 5. The validation loss stayed within a tighter range than the training loss, indicating that the model's performance on unseen validation samples was more consistent despite some performance dips in sensitivity and accuracy.

Testing loss followed a similar pattern to validation, with values ranging from 1.7295 in Fold 3 (lowest) to 2.3136 in Fold 5 (highest). Folds 3 and 2 showed the lowest test losses, aligning with strong accuracy and decent sensitivity, while Fold 5 had the highest test loss, likely reflecting its poor sensitivity (0.5) and highest validation loss.

Overall, the gap between training and testing losses, especially in earlier folds suggests the model may be overfitting slightly, focusing more on optimizing for the training distribution than generalizing perfectly to unseen data. However, the more balanced and lower validation/test losses in Folds 2 and 3 indicate that under the right data splits, the MLP can generalize effectively.

Additionally, the following graphs present the evolution of test accuracy, sensitivity, specificity and loss across all 50 training epochs for each fold individual and the average trend across folds. By carefully observing these graphs, specific fold dependent patterns in how the MLP Classifier’s generalization performance had evolved during the training phase.



**Figure 5.24:** Test Accuracy per epoch and Average Accuracy, across all folds using the MLP Classification Head with 50 epochs.

As far as test accuracy evolution is concerned for the MLP classification head, several patterns can be observed in how performance evolved across epochs and folds.

In Fold 1, test accuracy began at a fairly strong 0.754 in epoch 1, but immediately dipped to as low as 0.246 in epoch 2, and fluctuated erratically until epoch 10. Between epochs 10 and 25, it showed signs of gradual stabilization with values mostly hovering between 0.53 and 0.67. After epoch 25, accuracy steadily improved and remained above 0.69, eventually peaking again close to its starting point by epoch 50. While initial fluctuations were severe, the model appeared to recover and generalize more effectively later in training.

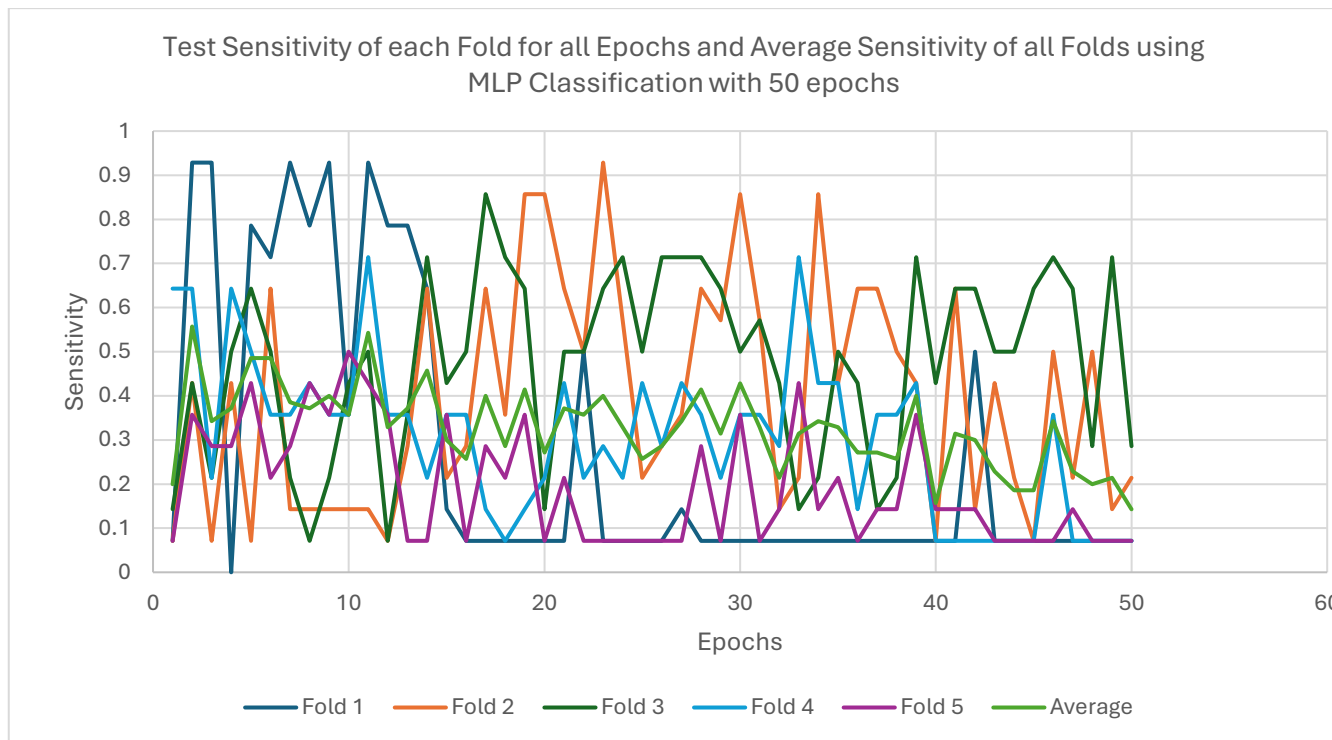
Fold 2 started at 0.76, and although it briefly dipped to 0.63 around epoch 20, and it overall remained highly stable and consistent throughout the 50 epochs, mostly staying between 0.72 and 0.78. This consistency suggests that Fold 2’s data split was well-balanced for the MLP, and the model learned without significant volatility.

For Fold 3, performance was solid and remarkably flat across the entire training timeline. It began at 0.75, never dropped below 0.65, and consistently stayed within the range of 0.69 to 0.77, especially after epoch 10. The stability in this fold reflects good learning and generalization with minimal overfitting signs.

Fold 4, while starting slightly lower at 0.70, followed a smooth upward trend, reaching 0.78 at multiple points. It remained consistent between 0.73 and 0.78 for the majority of the epochs, suggesting that the MLP was able to progressively learn from this split and maintain strong testing performance without sharp declines.

In contrast, Fold 5 achieved one of the strongest and most consistent test accuracy curves. It started at 0.78, peaked at 0.83 by epoch 3, and remained stable between 0.76 and 0.80 across the full 50 epochs. Fold 5's curve was the least volatile, indicating that the model generalized well on this particular fold from the start, and didn't benefit much from longer training as its peak was reached early.

Looking at the average test accuracy across folds, it started at 0.75, briefly dipped to 0.64 in epoch 2, and then gradually recovered. After epoch 10, the average remained relatively stable, consistently ranging between 0.74 and 0.77 until the end. This contrasts with the linear classifier's average trend, where accuracy dropped after epoch 20. In the case of the MLP, generalization performance was more reliable and sustained, and no overfitting signs were evident based on the average test accuracy curve. Overall, this suggests that the MLP head was more robust and stable across folds compared to its linear counterpart.



**Figure 5.24:** Test Sensitivity per epoch and Average Sensitivity, across all folds using the MLP Classification Head with 50 epochs.

As far as test sensitivity evolution is concerned, it can be seen from the line graphs above that for Fold 1, sensitivity began at a very low value of 0.07, but improved drastically by epoch 2, spiking to around 0.93. This was followed by frequent oscillations between high and low values, ranging from 0.3 to 0.92 up until epoch 10. The lowest value appeared in epoch 4 where sensitivity stood at 0, signifying that the model was unable to detect any of the unhealthy cases at all. After this initial volatility, sensitivity gradually decreased and became unstable beyond epoch 20, dipping consistently below 0.4 with multiple flatlines at the minimum score of 0.07, showing that the model's ability to detect unhealthy cases in Fold 1 degraded as training progressed.

Fold 2 presented an erratic pattern with no clear upward or downward trend. Sensitivity jumped between 0.07 and 0.92 throughout the epochs. While Fold 2 reached high peaks such as 0.93 at epoch 23, these spikes were short-lived and were followed by sharp drops. Unlike the linear model which stabilized after epoch 10, the MLP classifier for Fold 2 fluctuated throughout the training timeline, indicating a lack of stability in learning cancerous patterns in this fold.

In Fold 3, sensitivity was relatively more consistent. Although the values remained low between 0.14 and 0.21 during the early epochs, the model gradually improved, reaching 0.64 to 0.71 multiple times from epochs 20 to 40, and even peaking at 0.8571 in epoch 17. Despite this, Fold 3 still demonstrated moderate variability, but overall showed better learning behaviour compared to folds 1 and 2.

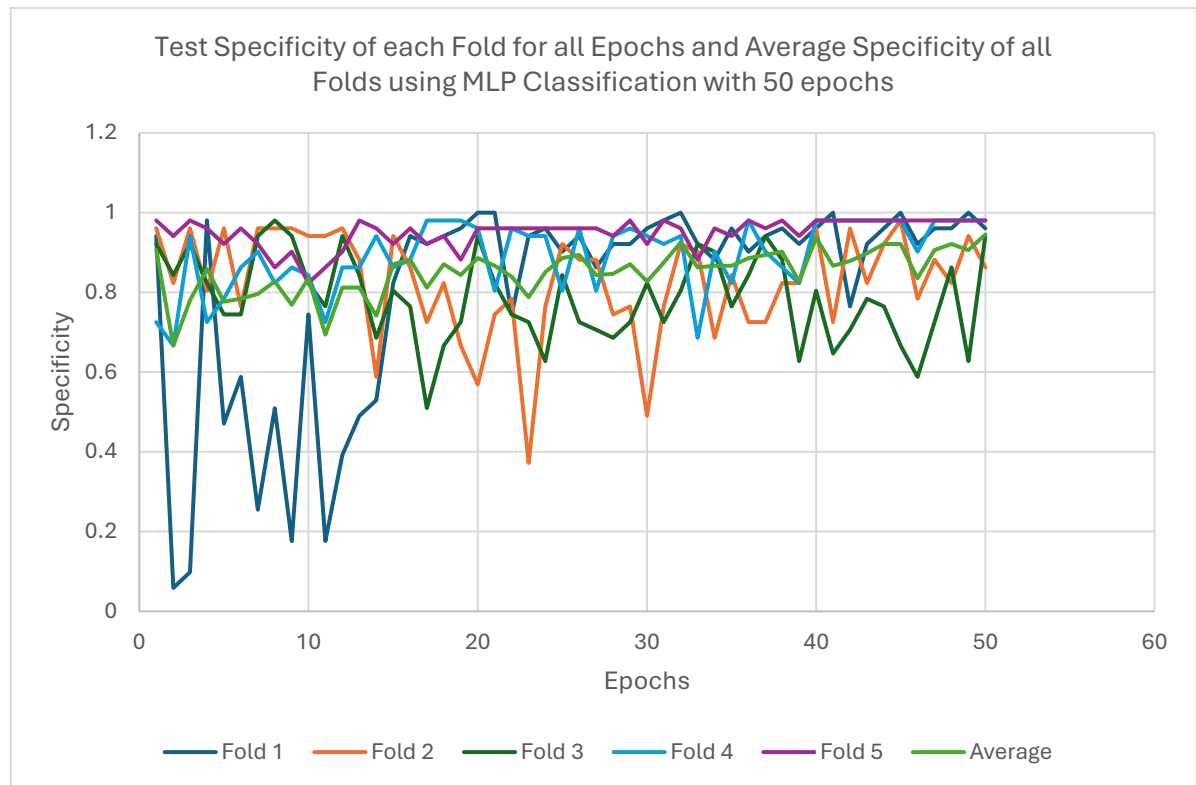
Fold 4 began with strong sensitivity values, starting at 0.6429 in epoch 1 and maintaining similar high values through epochs 2, 4, and 5. In the first 10 epochs, values remained relatively stable between 0.35 and 0.71, suggesting that the model was initially effective in recognizing cancerous cases. However, from epoch 11 onwards, sensitivity started to decline more noticeably. By epoch 18, values had dropped to 0.0714, and for the remainder of the training, sensitivity hovered mostly between 0.0714 and 0.42, with many epochs falling at or near the minimum of 0.0714. This pattern reveals a clear deterioration in sensitivity over time, suggesting that while the model initially learned to detect cancer cases well, it gradually lost its ability to recall positive instances, possibly due to overfitting on the healthy class or failing to generalize well on more difficult samples in later epochs.

In contrast, Fold 5 maintained consistently low sensitivity throughout training. It started at 0.07 and mostly remained below 0.35 across all 50 epochs. There were a few exceptions, such as 0.5 in epoch 10, but these were isolated cases. Overall, Fold 5 demonstrated the weakest performance in detecting cancerous cases, and failed to reach clinically acceptable levels of sensitivity.

The average sensitivity across all folds started at a low 0.20 in the first epoch but saw a significant improvement by epoch 2, reaching a short-lived peak of 0.55. After that, a slow and fluctuating downward trend began. Between epochs 5 and 20, the average sensitivity hovered between 0.30 and 0.48, showing no stable upward momentum. A few scattered improvements were recorded (such as 0.42 in epoch 28 and 0.43 in epoch 30), but these were not sustained. From epoch 35 onward, average sensitivity sharply dropped, reaching critically low values such as 0.22 at epoch 43 and just 0.14 at the final epoch (epoch 50). These trends highlight the MLP classifier's instability in reliably detecting cancerous cases. The volatility and drop in sensitivity over time are especially concerning given the clinical importance of minimizing false negatives. This instability



reinforces the need for enhanced threshold tuning, regularization, or more robust architectures to maintain generalization in sensitive medical prediction contexts.



**Figure 5.25:** Test Specificity per epoch and Average Specificity, across all folds using the MLP Classification Head with 50 epochs.

Regarding test specificity evolution, the MLP Classification Head maintained a remarkably high and consistent performance across all folds, with minimal fluctuations compared to the sensitivity curves. For Fold 1, specificity started strong at 0.9412 in epoch 1 but dipped drastically as low as 0.05 by epoch 2. Then, a highly unstable early period followed until around epoch 15, with values fluctuating between 0.17 and 0.8, indicating a period where the model struggled to confidently identify healthy cases. From epoch 20 onwards, the values began to stabilize and steadily climb, eventually reaching perfect scores of 1.0 in epochs 31 and 32, and remaining consistently above 0.90 by epoch 42 suggesting that the model gradually learned to identify healthy samples with increasing certainty.

Fold 2 on the other hand, showed a very different pattern. From the very beginning, specificity remained consistently high starting strong at 0.96 in epoch 1 and remained impressively consistent throughout the training. Specificity scores rarely dipped below 0.72, with most values clustering between 0.82 and 0.96 and the lowest specificity

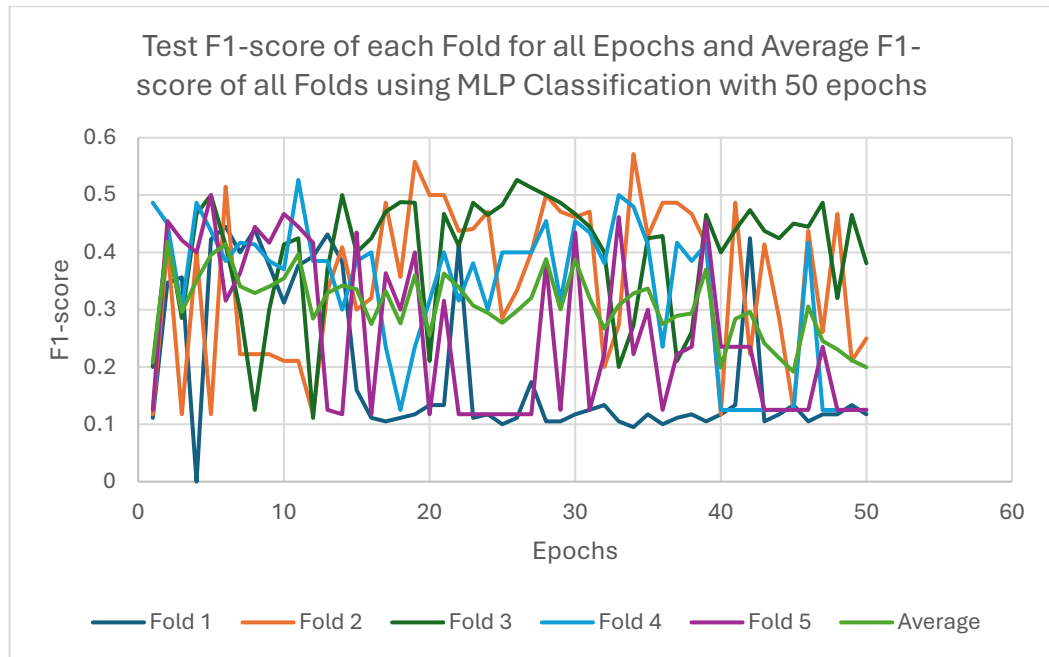
standing at 0.3725 at epoch 23. However, unlike Fold 1, Fold 2 never peaked at 1.0. Its highest values hovered in the 0.96 range. Despite not reaching perfection, Fold 2 demonstrated very stable and high specificity, reflecting robust generalization in identifying negative (healthy) cases.

In Fold 3, performance was also solid. Specificity began at 0.92 in epoch 1, with some fluctuations between 0.74 and 0.94 throughout the epochs. Although a few drops were observed, especially around epochs 14–17 where specificity reached as low as 0.50, the model quickly recovered, finishing above 0.80 in most of the later epochs. This shows that the model remained reliable in classifying healthy cases, though minor inconsistencies were present.

Fold 4 exhibited more variability. It started at 0.72, dropped to 0.66 in epoch 2, and hovered around 0.72–0.86 until about epoch 23. After this point, it significantly improved, peaking at 0.98 in several later epochs. The progression indicates that the model required more time to generalize well for Fold 4 but eventually achieved high specificity.

Fold 5 was the most consistent performer among all folds. It started at 0.98 and sustained specificity values above 0.82 throughout the entire 50 epochs. There were minimal fluctuations, with most of the values surpassing 0.9. This reflects a very strong capability in identifying healthy cases with almost no false positives.

The average specificity across all folds started high at 0.9059, briefly dropped to 0.66 in epoch 2 which was mainly due to Fold 1, and then steadily increased and stabilized. From epoch 12 onwards, the average remained between 0.8 and 0.94, reflecting excellent model performance in distinguishing healthy cases overall. While sensitivity varied widely across folds, specificity remained strong and stable, confirming that the MLP classifier excelled at reducing false positives, even if false negatives were more difficult to control.

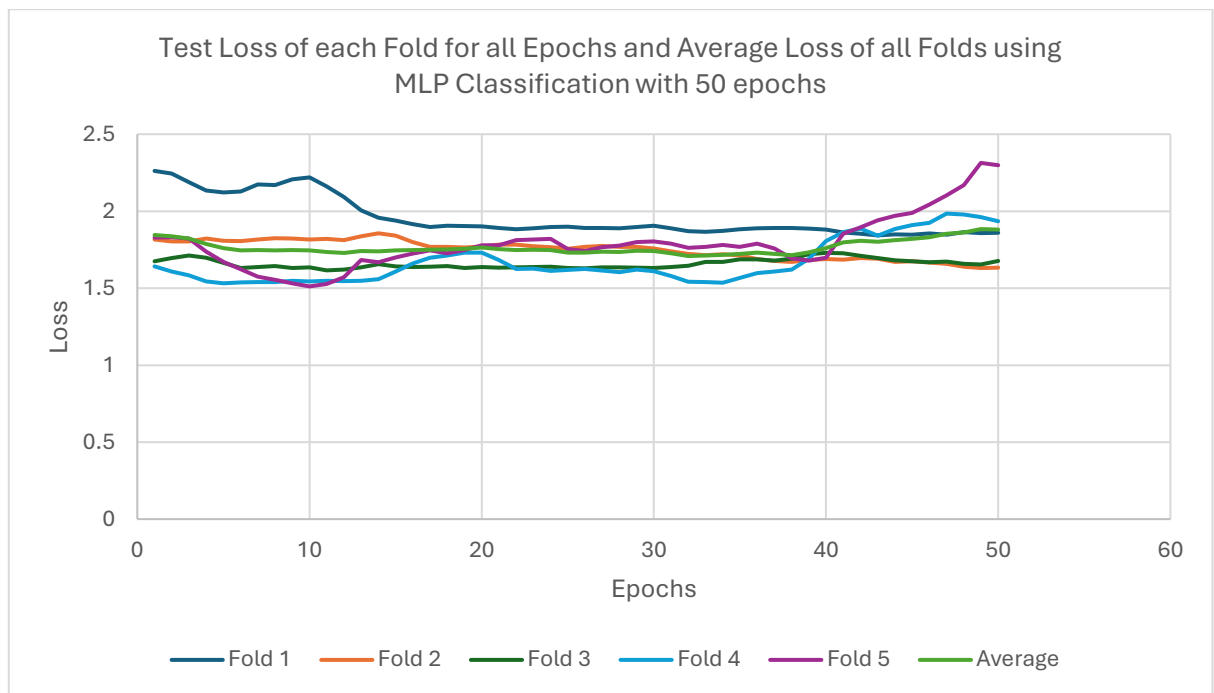


**Figure 5.26:** Test F1-score per epoch and Average F1-score, across all folds using the MLP Classification Head with 50 epochs.

Figure 5.26 illustrates the evolution of the test F1-score across all 50 epochs for each of the five folds using the MLP classification head. Compared to the linear model, the F1-scores here exhibit significantly more volatility, with sharp fluctuations across epochs and folds. Despite this variability, a few patterns emerge. Folds 2 and 3 generally maintained higher F1-scores throughout training, frequently exceeding 0.4 and occasionally reaching peaks above 0.5. This suggests that the MLP classifier was able to learn effective decision boundaries for the data in these folds, achieving a more favorable balance between precision and recall. Conversely, Fold 5 consistently performed poorly, with F1-scores plateauing at or below 0.3 for most epochs, mirroring earlier findings that this fold presented greater classification difficulty.

The average F1-score across all folds, shown in purple, remained moderate and relatively stable throughout training, oscillating around the 0.3–0.35 range. Notably, early epochs (around epoch 2 and 6) saw brief peaks in average F1-score, but the model was unable to sustain these improvements, possibly due to overfitting or the inherent difficulty in reconciling performance across folds with varying levels of difficulty. The lack of a clear upward trend and the persistence of instability even in later epochs indicate that, while the MLP architecture offered a more flexible and expressive model than the linear head, it still struggled to generalize robustly across all test sets without additional regularization or tuning.

These results suggest that although the MLP model shows potential in capturing more complex patterns, it also introduces training instability and inconsistent generalization. As such, further investigation into regularization strategies, early stopping, or ensemble techniques may be warranted to enhance the consistency and reliability of MLP-based classifiers in clinical prediction tasks.



**Figure 5.27:** Test Loss per epoch and Average Loss, across all folds using the MLP Classification Head with 50 epochs.

Fold 1 began with the highest test loss of all folds at 2.2616 during epoch 1. While some improvement was noted in the following epochs, with the loss gradually dropping to around 1.9–1.8 between epochs 14 and 50, the overall trend remained erratic. After epoch 30, the fold experienced a slight rise in loss again, stabilizing at a relatively high level above 1.85 by epoch 50. This behaviour indicates that while the model made some progress in reducing error early on, it struggled to generalize consistently, likely due to instability in detecting both classes under Fold 1’s data distribution.

Fold 2, in contrast, showed a more stable and consistent behaviour. Starting from 1.8168 in epoch 1, the test loss remained within a narrow band of 1.6 to 1.82 for the majority of training. This stability reflects that the MLP classifier performed with reliable generalization on Fold 2. There were no abrupt spikes or concerning increases throughout training, making this fold one of the more well-behaved cases. Although it didn’t achieve the lowest loss, its consistent trend suggests good balance between fitting and generalizing.

Fold 3 began with a relatively lower loss of 1.6754, and continued to show a smooth, albeit gradual, improvement over time. Between epochs 10 and 30, the values fluctuated

gently around 1.63–1.64, indicating relatively strong generalization. The final epochs did see a minor upward creep, peaking at 1.7295 in epoch 40 and 1.6776 at epoch 50. However, this variation remained mild. Overall, Fold 3 demonstrated good stability, and the model maintained reasonably low error throughout.

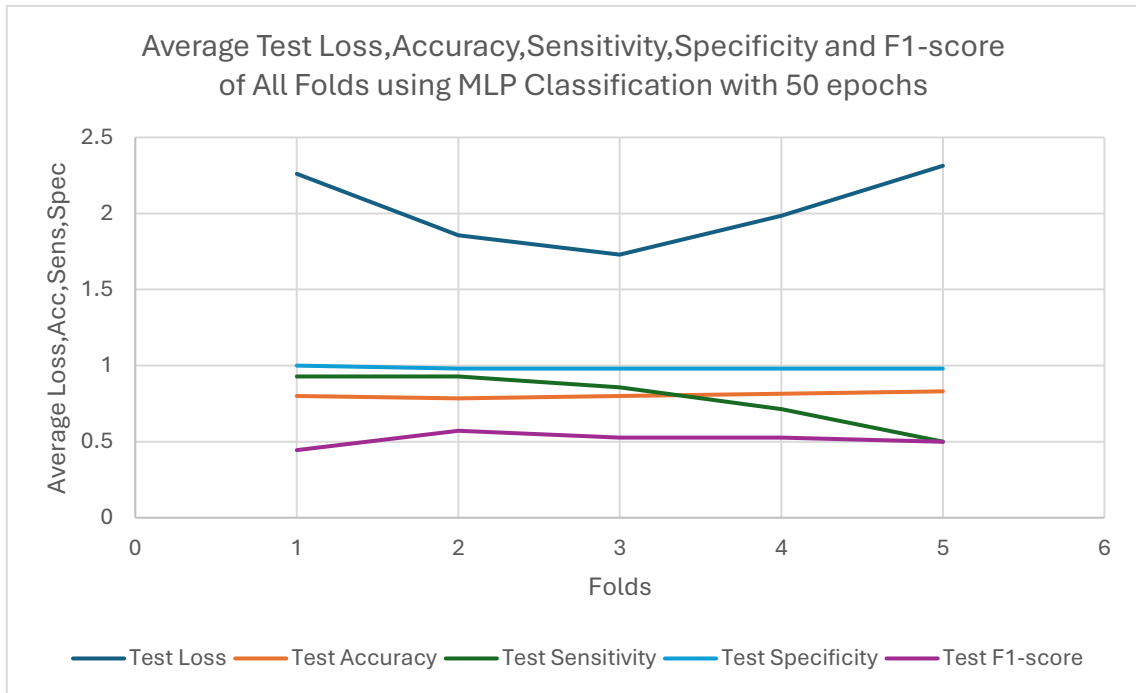
Fold 4 stood out as the fold with the most consistent and lowest test loss values. Beginning at 1.6406, loss dropped quickly below 1.55 by epoch 4 and stayed in the 1.53–1.63 range throughout the majority of training. Unlike other folds, Fold 4 did not suffer from late-epoch overfitting or spikes, and even in the final epochs, loss remained tightly bound, with a final value of 1.9344, which was still better than the late spikes seen in other folds. This suggests that Fold 4's validation and test distribution aligned better with the model's learned features.

Fold 5 followed a very different and concerning trajectory. Starting at 1.8296, the fold experienced a consistent and alarming increase in test loss throughout training. From epoch 20 onward, loss escalated progressively, surpassing 2.0 in epoch 46 and reaching the peak of 2.3136 in epoch 49 which was the highest loss across all folds and epochs. This pattern is a clear indication of severe overfitting, where the model continued to minimize training loss but completely failed to maintain generalization on unseen data in this fold.

The average test loss across all folds began at a high 1.8448 in epoch 1, dropped steadily throughout the first 20–30 epochs, reaching a relative low around 1.71–1.74, particularly between epochs 25–35. However, after epoch 35, a clear upward trend was observed again, climbing to 1.8838 in epoch 49 and 1.8805 in epoch 50. This U-shaped trajectory suggests that the best generalization performance occurred in the middle of training, and that training beyond that point did more harm than good. The rising curve in the later epochs was heavily influenced by deteriorating performance in folds like 1 and 5, indicating that early stopping or regularization strategies could have been beneficial to avoid late-phase overfitting.

An equally important visualization of the performance metrics is given in the following graph, which presents a direct comparison of the average (meaning maximum out of

each fold) testing Accuracy, Sensitivity, Specificity, and Loss across all five folds after training the MLP Classification Head for 50 epochs.



**Figure 5.28:** Average Test Accuracy, Sensitivity, Specificity, F1-score and Loss Across All Folds Using the MLP Classification Head with 50 epochs.

The graph shown above offers a consolidated view of how each fold performed in terms of average test accuracy, sensitivity, specificity, and loss using the MLP classification head trained over 50 epochs. It gives a direct side-by-side comparison that helps highlight generalization consistency and potential weaknesses across folds.

Starting with Fold 1, we observe the highest test loss of 2.2616, indicating that the model struggled significantly in this fold in terms of optimization. However, despite the high loss, both accuracy and sensitivity peaked at 0.8 and 0.9286, respectively, while specificity reached a perfect 1.0. This suggests that although the model had a harder time minimizing the raw loss, it still maintained excellent classification performance on both classes.

In Fold 2, test loss dropped slightly to 1.8563, and accuracy remained strong at 0.7846, with sensitivity again at a high 0.9286. Specificity dropped slightly to 0.9804. This fold shows a well-balanced generalization across both classes and confirms consistent detection of cancerous cases.

Fold 3 exhibited the lowest test loss at 1.7295, indicating the most efficient training performance. Test accuracy remained high at 0.8, sensitivity was still strong at 0.8571, and specificity matched previous folds at 0.9804. This balance makes Fold 3 the most stable and effective fold overall.

In Fold 4, test loss increased to 1.9845, and despite an improved accuracy of 0.8154, sensitivity dropped notably to 0.7143, suggesting a deterioration in the model's ability to detect positive cases. Specificity held constant at 0.9804.

Fold 5 showed the most extreme imbalance: test loss peaked at 2.3136, sensitivity dropped to 0.5 (the lowest among all folds), yet accuracy hit 0.8308, and specificity stayed high at 0.9804. This clearly indicates the model overfit to negative samples, failing to generalize well to the positive class.

It's important to highlight that the worst sensitivity was recorded in Folds 4 and 5, a pattern consistent with the linear classification results. This reinforces the concern that both the linear and MLP models have difficulty generalizing recall performance across certain data splits, particularly when faced with challenging positive samples.

In terms of the F1-score, which represents the harmonic mean between precision and sensitivity, the MLP classification head showed modest but improved balance in performance compared to the linear counterpart. The highest F1-score was achieved in Fold 2 (0.5714), while the lowest appeared in Fold 1 (0.4444). Folds 3 and 4 yielded identical F1-scores of approximately 0.5263, with Fold 5 scoring exactly 0.5. These values suggest that the MLP head maintained more consistent class balance across folds, particularly in Folds 3 to 5, where the F1-scores converged near the 0.5 mark. This reflects better alignment between recall and precision compared to the linear head, which showed more pronounced fluctuations. Although the F1-scores were still relatively moderate overall, the improved stability in MLP results reinforces its effectiveness in handling the class imbalance challenges inherent in the dataset, making it a more reliable alternative for capturing the nuanced patterns necessary for binary medical image classification.

Overall, while accuracy and specificity remained stable and high across all folds, sensitivity exhibited significant volatility, particularly in Folds 4 and 5. This sensitivity instability, coupled with fluctuating loss values, emphasizes the need for better fine-

tuning, whether that is through threshold tuning, class reweighting, or architectural adjustments, in order to ensure more reliable cancer detection across diverse datasets.

### 5.3.2 Performance at 150 Epochs

To further explore the learning capabilities and generalization behavior of the MLP Classification Head, an extended training run was conducted for 150 epochs. This experiment was intended to assess whether prolonged training could lead to improved model performance, especially in terms of test sensitivity and overall robustness across folds. While the previous section focused on results obtained with 50 epochs, the 150-epoch setting provides insight into potential long-term benefits or drawbacks of extended optimization.

Unlike the 50-epoch experiments, which involved detailed visualizations for each metric across epochs, the 150-epoch evaluation is summarized through a smaller, more targeted set of plots. These include performance trends in accuracy, sensitivity, specificity, and loss for each fold, as well as a direct comparison of test sensitivities between the 50- and 150-epoch models.

The following table presents the average (meaning maximum out of every fold) training, validation, and testing metrics (accuracy, sensitivity, specificity, and loss) for each fold after 150 epochs. These values were computed using the best-performing threshold per fold and provide a comprehensive summary of how the model performed across the different phases of evaluation.

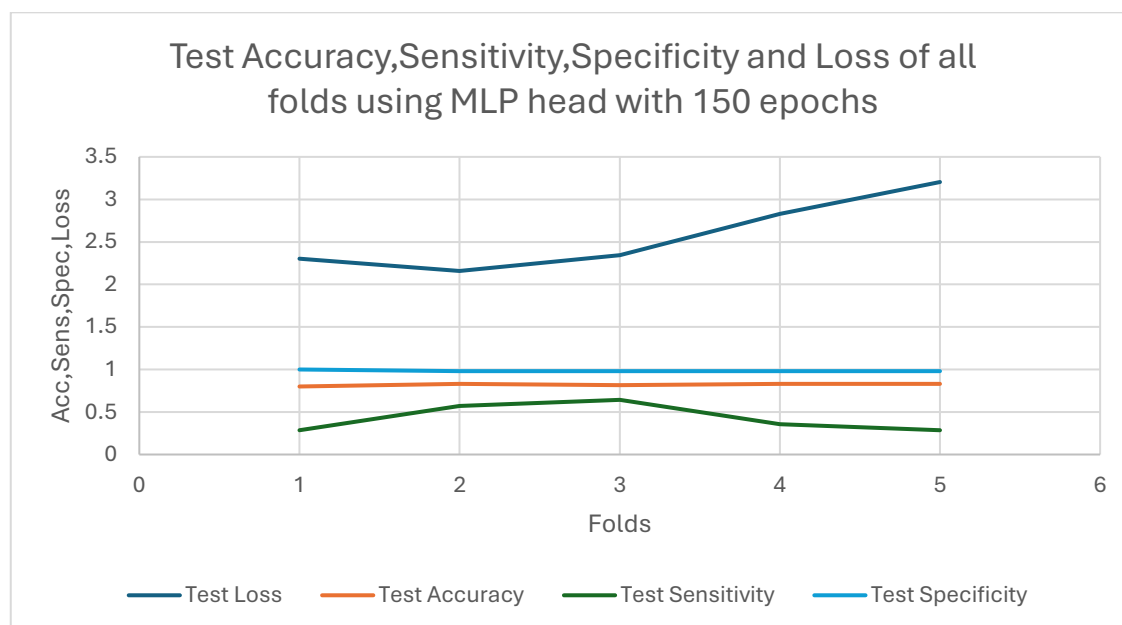
Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss
1	0.8398	0.9348	0.95	6.3572	0.7885	0.1818	1	2.3909	0.8	0.2857	1	1
2	0.8883	0.9556	0.9317	5.263	0.8462	0.75	1	2.0036	0.8308	0.5714	0.9804	0.9804
3	0.932	0.9778	0.9627	4.2386	0.7692	0.25	1	3.376	0.8154	0.6429	0.9804	0.9804
4	0.913	0.9783	0.9627	4.3729	0.902	0.6364	1	2.3658	0.8308	0.3571	0.9804	0.9804
5	0.9565	0.9565	0.9627	4.0954	0.8235	0.2727	1	2.9379	0.8308	0.2857	0.9804	0.9804

**Table 5.9:** Performance metrics obtained with training, validation and testing, of MLP Classification Head with 150 epochs

The first visualization in this section presents the test accuracy, sensitivity, specificity, and loss across all five folds after training the MLP Classification Head for 150 epochs. This graph provides a direct comparison of each fold's final performance, capturing the variability and consistency of the model when evaluated on unseen test data. By



analysing these metrics together, it becomes easier to identify fold-specific weaknesses, particularly in sensitivity, as well as trends in model generalization. This overview is crucial in highlighting whether longer training introduced performance gains, overfitting, or stability across folds.



**Figure 5.29:** Average Test Accuracy, Sensitivity, Specificity and Loss Across All Folds Using the MLP Classification Head with 150 epochs.

As seen in the graph above, the average performance metrics across all folds after training the MLP Classification Head for 150 epochs reveal several important insights. Test accuracy remained quite stable and consistently high across all folds, with four out of five folds reaching the upper bound of 0.8308, and the remaining fold at 0.8. Similarly, test specificity remained exceptionally strong and stable, staying fixed at 0.9804 for four folds and even peaking at 1.0 in Fold 1, confirming that the model maintained a high ability to correctly identify healthy cases.

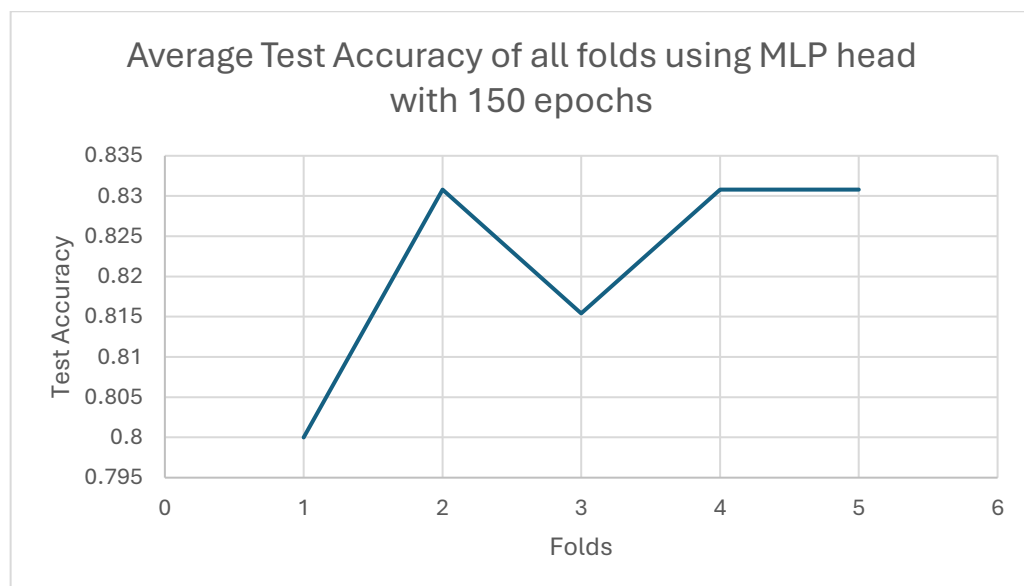
However, the sensitivity values paint a different story. The performance was considerably more inconsistent and significantly weaker compared to accuracy and specificity. Sensitivity peaked at 0.6429 in Fold 3, followed by 0.5714 in Fold 2. Yet, Folds 1 and 5 performed poorly, both recording the lowest sensitivity of 0.2857. Fold 4 followed a similar trend with a mediocre 0.3571. This variability highlights once again that the model's ability to detect positive (cancerous) cases remains unstable and highly fold-dependent, a pattern also previously observed with the 50-epoch setting.

Interestingly, this inconsistency in sensitivity came despite the model achieving its best and most consistent accuracies, reinforcing that high accuracy does not necessarily correlate with clinically meaningful sensitivity in imbalanced medical datasets.

Moreover, the test loss fluctuated notably across folds, ranging from 2.1583 in Fold 2 to as high as 3.2044 in Fold 5, further suggesting a trade-off between the model fitting and generalizing to minority-class instances.

### **Accuracy:**

The following graph illustrates the final Testing Accuracy achieved by the Linear Classification Head across all five folds after 150 epochs. This visualization provides insight into how accurately the model generalized to unseen data in each fold after prolonged training. It also allows comparisons between folds in terms of generalization stability.



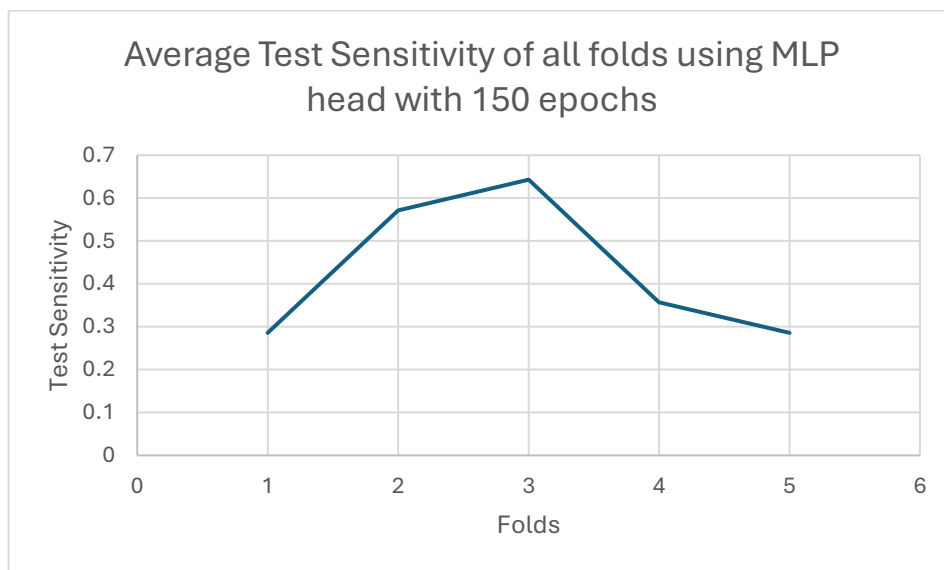
**Figure 5.30:** Average Test Accuracy Across All Folds Using the MLP Classification Head with 150 epochs.

The following graph presents the final testing accuracy of the MLP Classification Head across all five folds after 150 epochs. This visualization provides a clear picture of how well the model generalized to unseen data in each individual fold under extended training. As shown, accuracy remained consistently high in most folds, with Folds 2, 4, and 5 achieving the highest value of 0.8308. Fold 3 followed closely with a solid 0.8154, while Fold 1 had the lowest, albeit still strong, accuracy at 0.8. These results demonstrate that, even with extended training, the MLP head maintained stable performance in terms of test accuracy, indicating strong generalization capabilities on

the majority class (healthy cases). However, as later discussed, this consistent accuracy did not translate into equally consistent sensitivity, which is critical in medical diagnosis contexts.

### Sensitivity:

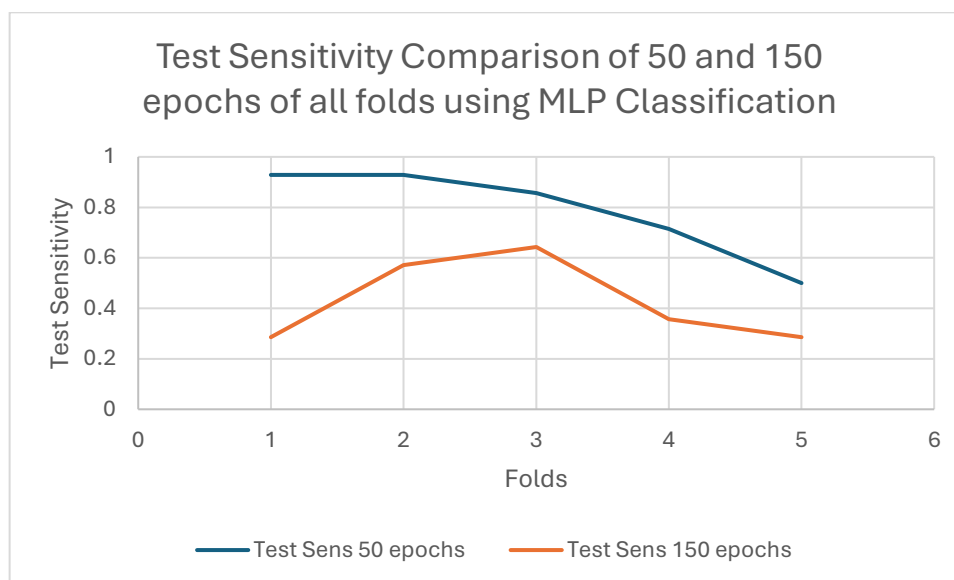
To further examine how well the MLP Classification Head performed in identifying positive cases after extended training, the following graph presents the average test sensitivity across all five folds at 150 epochs. This visualization focuses specifically on the model's recall capabilities per fold, offering insight into its ability to generalize in detecting cancer cases throughout different data splits.



**Figure 5.31:** Average Test Sensitivity Across All Folds Using the MLP Classification Head with 150 epochs.

The graph above presents the average test sensitivity across all five folds after training the MLP Classification Head for 150 epochs. From the visualization, we observe that Fold 3 achieved the highest sensitivity at 0.6429, indicating the model's best ability in that fold to correctly identify cancerous cases. Fold 2 also showed relatively strong performance with a sensitivity of 0.5714, suggesting decent recall on positive samples. However, the sensitivity dropped notably in Folds 1, 4, and 5, with Folds 1 and 5 recording the lowest values at 0.2857. This mirrors the behaviour previously observed at 50 epochs, where these same folds underperformed in sensitivity. The inconsistencies between folds indicate that despite extended training, the model's recall ability remained unstable and highly fold dependent.

To further investigate the effects of extended training on the model’s ability to detect positive (cancerous) cases, a fold-wise comparison of test sensitivity between 50 and 150 training epochs was conducted. The corresponding graph and table offer a clear visualization of how sensitivity evolved with prolonged training for each fold.



**Figure 5.32:** Sensitivity Comparison Across All Folds between the MLP Classification Head with 50 and 150 epochs.

Fold	Test Sensitivity 50 Epochs	Test Sensitivity 150 Epochs
1	0.9286	0.2857
2	0.9286	0.5714
3	0.8571	0.6429
4	0.7143	0.3571
5	0.5	0.2857

**Table 5.10:** Test Sensitivity Comparison of MLP Classification Head, when using 50 and 150 epochs, respectively.

In Fold 1, test sensitivity experienced a sharp decline, dropping from 0.9286 after 50 epochs to 0.2857 after 150 epochs. This represents a substantial degradation of over 69%, suggesting that prolonged training adversely affected the model's recall ability for this fold.

Similarly, in Fold 2, sensitivity decreased from 0.9286 to 0.5714, indicating a reduction of approximately 38%. Although performance remained moderately acceptable, the reduction still points to a sensitivity loss with increased training duration.

Fold 3 displayed a smaller decrease in sensitivity, falling from 0.8571 to 0.6429. While this drop was less pronounced (around 25%), it still reflects the trend of diminishing sensitivity under extended training conditions.

In Fold 4, sensitivity dropped from 0.7143 to 0.3571, representing a 50% reduction. This result underscores a significant performance deterioration, with the model failing to retain its ability to detect positive cases.

Finally, Fold 5 recorded a decrease from 0.5 to 0.2857, once again yielding the weakest sensitivity values across both training durations.

Collectively, these results demonstrate that extending training to 150 epochs consistently led to lower sensitivity scores across all folds. This trend suggests the presence of overfitting to the dominant class (i.e., healthy samples), impairing the model's generalization ability for minority (cancerous) cases. It is also noteworthy that Folds 4 and 5 continued to exhibit the poorest sensitivity, mirroring the behavior observed in the linear classification results and thereby reinforcing their fold-specific difficulty in learning positive representations.

### 5.3.3 Performance at 30 Epochs

To further evaluate the effect of training duration on the model's performance, a final experiment was conducted where the MLP Classification Head was trained for a reduced number of 30 epochs. This setting aimed to investigate whether earlier stopping could prevent overfitting and better preserve sensitivity, particularly in minority cases which are the positive cases. While 50 and 150 epochs provided insights into both a baseline and an extended training performance, the 30 epoch experiment's purpose was to focus on the model's early generalization behaviour. Similarly to the previous section, performance is analysed through key metrics and visualized across folds to assess model stability and sensitivity preservation.

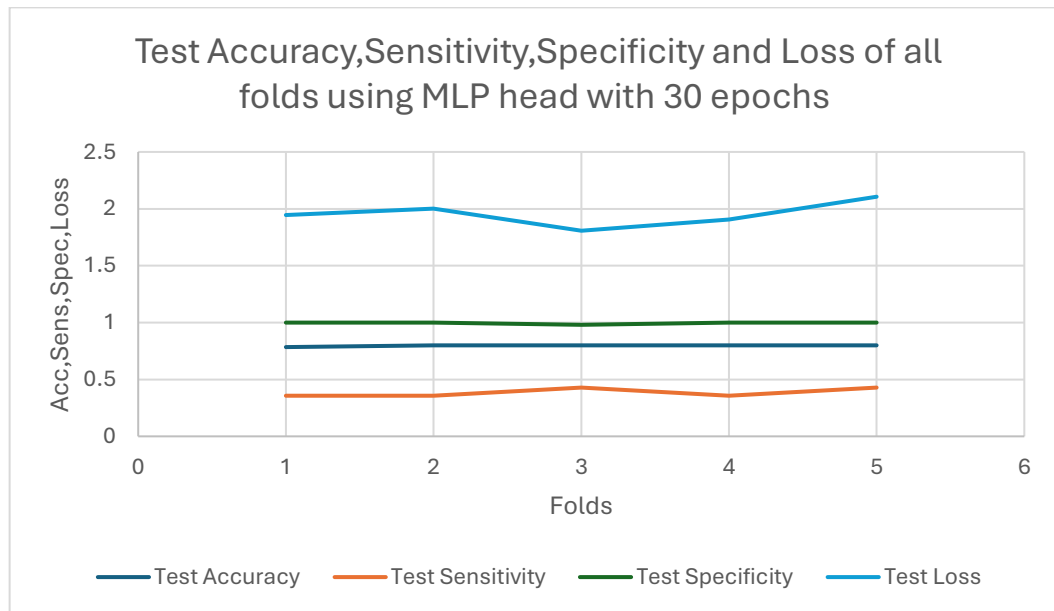
The following table summarizes the best training, validation and testing performance of the MLP classifier trained for 30 epochs across all 5 folds.

Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss
1	0.7573	0.8913	0.8562	6.195	0.8077	0.4545	1	1.9635	0.7846	0.3571	1	1.19463
2	0.8155	0.9111	0.9006	5.8185	0.8462	0.6667	1	1.8269	0.8	0.3571	1	2.0025
3	0.8252	0.9333	0.8758	5.5235	0.8077	0.25	1	2.0158	0.8	0.4286	0.9804	1.8074

4	0.8357	0.8913	0.8882	5.4302	0.8235	0.6364	1	1.5873	0.8	0.3571	1	1.9045
5	0.8937	0.913	0.9317	5.283	0.8627	0.4545	1	2.0139	0.8	0.4286	1	2.1061

**Table 5.11:** Performance metrics obtained with training, validation and testing, of MLP Classification Head with 30 epochs

An important visualization in this section displays the test accuracy, sensitivity, specificity, and loss achieved across all five folds after training the MLP Classification Head for 30 epochs. This graph offers a clear, fold-wise comparison of the model's final performance, highlighting both consistency and variability in its generalization to unseen test data. Evaluating these metrics collectively allows for the identification of fold-specific weaknesses, particularly in sensitivity and provides insight into whether extended training led to performance improvements, signs of overfitting, or stable behaviour across different data splits.



**Figure 5.33:** Average Test Accuracy, Sensitivity, Specificity and Loss Across All Folds Using the MLP Classification Head with 30 epochs.

As seen in the table and graph above, in Fold 1 the model achieved a test accuracy of 0.7846 and a perfect specificity of 1.0, indicating that all healthy samples were correctly identified. However, the sensitivity was relatively low at 0.3571, revealing that a significant number of positive (cancerous) cases were missed. The corresponding test loss was 1.9463, suggesting a moderate level of uncertainty in predictions.

Fold 2 showed a slight improvement in accuracy, reaching 0.8, with a consistent specificity of 1.0. However, similar to Fold 1, the sensitivity remained low at 0.3571. The loss increased slightly to 2.0025, indicating slightly weaker model confidence

compared to Fold 1. Overall, despite accurate classification of healthy cases, cancer detection performance remained limited.

Fold 3, yielded the best overall balance among metrics. Accuracy stood at 0.8, and while specificity was slightly lower than in other folds at 0.9804, sensitivity improved to 0.4286 which is the highest value achieved among all folds. Additionally, the test loss was the lowest at 1.8074, pointing to stronger predictive certainty and better calibration for this subset of the data.

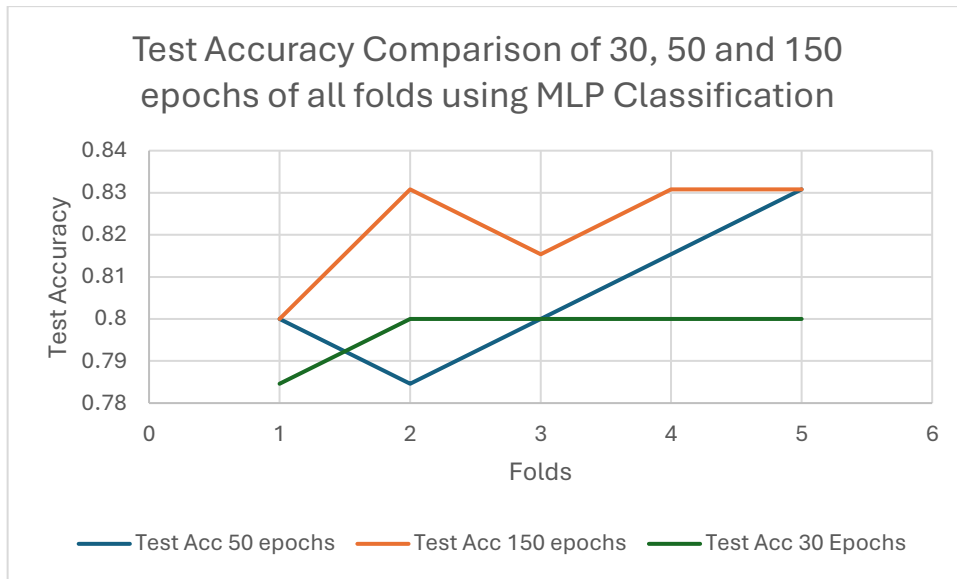
The model in Fold 4 also maintained an accuracy of 0.8 and a perfect specificity of 1.0. However, sensitivity again dropped to 0.3571, matching the lower range observed in Folds 1 and 2. Test loss was 1.9045, which is slightly better than Fold 2 but worse than Fold 3. This reinforces the observation that sensitivity struggled to improve despite high performance on negative cases.

Finally, Fold 5 recorded the highest test loss at 2.1061, implying weaker model confidence or prediction certainty on this fold. Nevertheless, accuracy was stable at 0.8, and specificity remained at 1.0. Sensitivity improved slightly to 0.4286, matching Fold 3. This indicates the model managed to recall more positive samples despite its higher loss.

Across all folds, the model exhibited consistent accuracy and excellent specificity after 30 epochs. However, sensitivity performance was limited and variable, with only Folds 3 and 5 reaching 0.4286. This suggests that while early training ensures general classification reliability, it may not provide the depth required for confident cancer detection, reinforcing the importance of optimizing for sensitivity in medical applications.

### **Accuracy:**

To complement the overall analysis, the following comparison focuses on the classification accuracy achieved across all five folds for the three training durations: 30, 50, and 150 epochs. This graph provides a comprehensive view of the model's ability to correctly classify both healthy and unhealthy cases under varying training lengths. By visualizing and comparing accuracy scores side-by-side, this analysis helps identify which epoch setting led to the most consistent and reliable performance across folds, and whether longer training offered any tangible improvements in generalization.



**Figure 5.34:** Test Accuracy Comparison Across All Folds Using the MLP Classification Head with 30, 50 and 150 epochs.

Folds	Test Accuracy 30 Epochs	Test Accuracy 50 Epochs	Test Accuracy 150 Epochs
1	0.7846	0.8	0.8
2	0.8	0.7846	0.8308
3	0.8	0.8	0.8154
4	0.8	0.8154	0.8308
5	0.8	0.8308	0.8308

**Table 5.12:** Testing Accuracy Comparison for all folds, of MLP Classification Head, obtained with all 3 epoch experiments, 30, 50 and 150 epochs respectively.

The graph and corresponding table present a clear comparison of test accuracy across all five folds under three different training durations: 30, 50, and 150 epochs. The results show how training length influenced the model's ability to generalize to unseen data when using the MLP classification head.

In Fold 1, the test accuracy was consistent between the 50 and 150 epoch settings, both reaching 0.8000. However, the 30-epoch setting resulted in a slightly lower accuracy of 0.7846. This suggests that a shorter training period may have prevented the model from fully capturing the necessary patterns in the data, while extending the training beyond 50 epochs did not further improve performance in this fold.

Fold 2 exhibited a more pronounced improvement with longer training. The model achieved its highest test accuracy of 0.8308 at 150 epochs, outperforming both the 50-epoch (0.7846) and 30-epoch (0.8000) configurations. This indicates that prolonged training helped the model better generalize in this fold, potentially allowing it to learn more complex decision boundaries.



In Fold 3, accuracy remained relatively stable across the three settings. Both the 30 and 50 epoch runs resulted in a test accuracy of 0.8000, while the 150-epoch run slightly improved to 0.8154. Although the improvement was modest, it demonstrates that extended training offered a small benefit in this case without any signs of degradation.

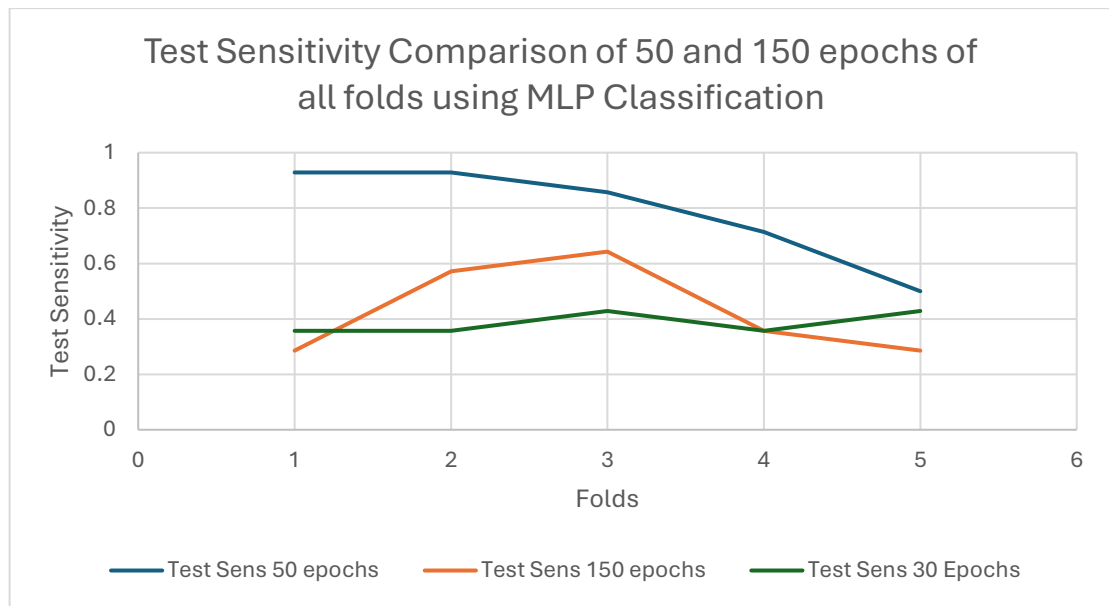
For Fold 4, a similar trend to Fold 2 was observed. The 150-epoch training produced the highest test accuracy of 0.8308, whereas 50 epochs achieved 0.8154 and 30 epochs produced 0.8000. This consistent improvement across increasing training durations reinforces the value of longer training in this fold and may reflect the complexity of the data subset it contains.

Finally, in Fold 5, all settings except the 30-epoch one resulted in the same high test accuracy of 0.8308. The 30-epoch run again fell slightly behind at 0.8000. This consistency across 50 and 150 epochs indicates that the model was able to learn the relevant features effectively within 50 epochs, and further training did not contribute any additional accuracy gains.

Overall, the results suggest that training for 150 epochs generally provided the most stable and highest test accuracy across folds, particularly for folds where shorter training durations were insufficient. The 50-epoch setting also delivered strong performance, while the 30-epoch configuration, although faster, tended to underperform, especially in Folds 1, 2, and 4. This analysis supports the idea that prolonged training contributes positively to model generalization in most cases.

### **Sensitivity:**

The following graph presents a comparative visualization of test sensitivity across all five folds for the MLP Classification Head trained with 30, 50, and 150 epochs. This comparison highlights how varying the number of training epochs impacts the model's ability to correctly identify positive (cancerous) cases in each fold. By analyzing the sensitivity scores side-by-side, it becomes possible to assess whether extended training consistently leads to improved recall or if shorter training durations may offer better generalization in certain scenarios.



**Figure 5.35:** Test Sensitivity Comparison Across All Folds Using the MLP Classification Head with 30 ,50 and 150 epochs.

Fold	Sensitivity 30 Epochs	Sensitivity 50 Epochs	Sensitivity 150 Epochs
1	0.3571	0.9286	0.2857
2	0.3571	0.9286	0.5714
3	0.4286	0.8571	0.6429
4	0.3571	0.7143	0.3571
5	0.4286	0.5	0.2857

**Table 5.13:** Testing Sensitivity Comparison for all folds, of MLP Classification Head, obtained with all 3 epoch experiments, 30, 50 and 150 epochs respectively.

The graph and table above present a comparative overview of test sensitivity across all five folds using the MLP Classification Head, trained under three different epoch settings: 30, 50, and 150 epochs. This visualization enables a direct fold-wise comparison of how prolonged or reduced training influenced the model’s ability to detect positive (cancerous) cases.

From the data, it is clear that training with 50 epochs yielded the best overall sensitivity. In particular, Fold 1 and Fold 2 achieved very high values of 0.9286, with Fold 3 also performing strongly at 0.8571. These results indicate that the model was able to consistently capture cancer-relevant patterns without significant overfitting at this training length.

When the training was extended to 150 epochs, sensitivity notably declined in almost all folds. Fold 1 dropped to 0.2857, and similar decreases were observed in Folds 4 and 5, which fell to 0.3571 and 0.2857, respectively. Although Fold 3 still maintained

relatively better sensitivity at 0.6429, the overall trend suggests that prolonged training beyond 50 epochs may have led to reduced sensitivity, likely due to overfitting on healthy samples and under-identification of positives.

In contrast, the 30-epoch configuration offered slightly better sensitivity than 150 epochs in some folds (e.g., Folds 1 and 5), but remained consistently lower than the 50-epoch configuration across all folds. The highest sensitivity in this setting was 0.4286 (Folds 3 and 5), with Folds 1, 2, and 4 all plateauing at 0.3571, reflecting limited learning capacity due to insufficient training duration.

In summary, the 50-epoch setting provided the most favourable balance, achieving the highest and most consistent test sensitivity across all folds. Both shorter (30) and longer (150) training durations led to a decline in sensitivity, reinforcing that 50 epochs was the optimal training length for maximizing cancer case detection in this MLP configuration.

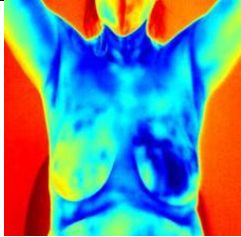
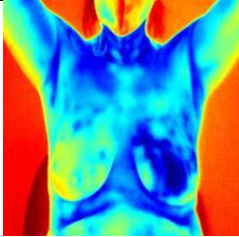
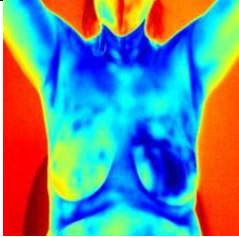
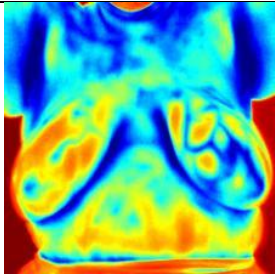
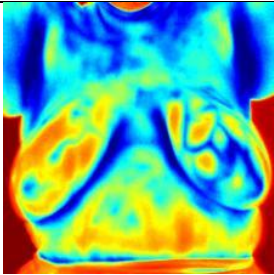
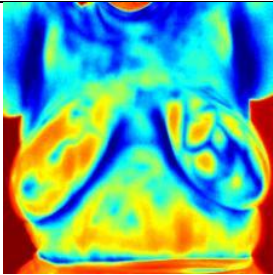
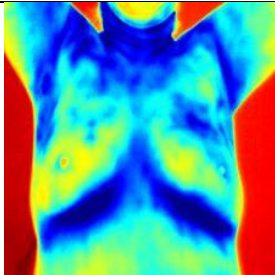
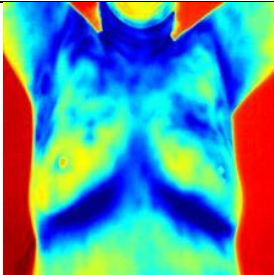
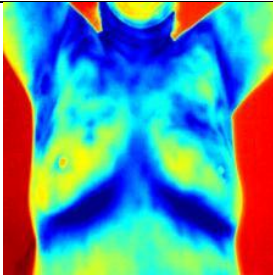
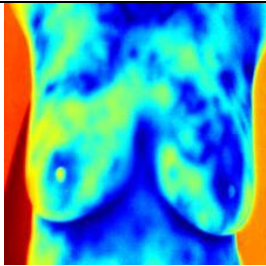
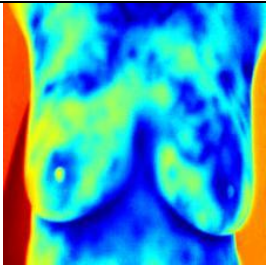
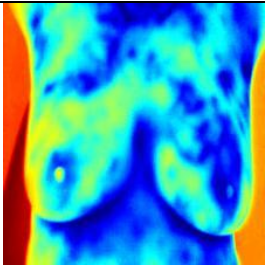
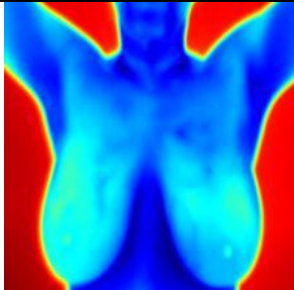
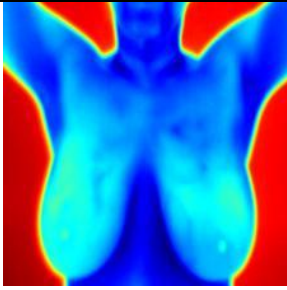
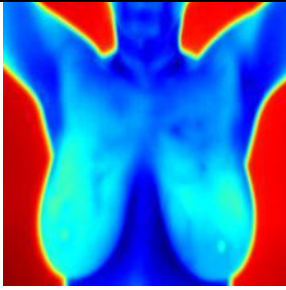
#### **5.3.4 Analysis of Misclassified Cases Affecting Sensitivity with MLP Head**




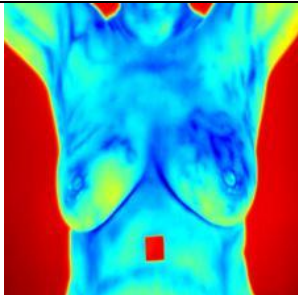
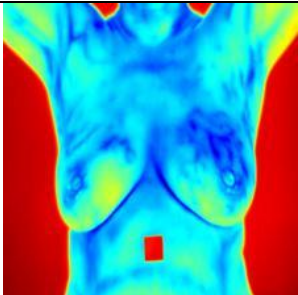
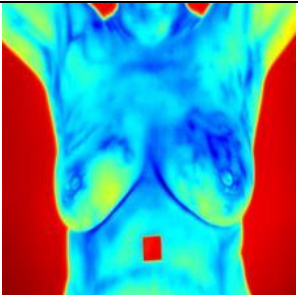
After examining the sensitivity values from the three epoch experiments, it was observed that the worst-performing folds in terms of sensitivity were folds 1, 4, and 5. The main goal of this analysis is to determine if the misclassified images contribute to the observed variations in sensitivity, particularly in these folds. To this end, functionality was added to the code to track and analyse misclassified images for each fold across all three epoch configurations.

The approach included tracking misclassified samples. Specifically in the MLP Classification Head implementation code block, after each training session, the misclassified images were collected to examine whether specific images are misclassified in all epoch configurations. The analysis' purpose was to determine whether these misclassified cases are predominantly from the Unhealthy class (false negatives), which would lower sensitivity. By comparing misclassified cases across different epoch values, the aim is to uncover any consistent patterns in the misclassifications and understand their impact on sensitivity.

This analysis is designed to help pinpoint potential areas for improvement in the model's performance.

The table below presents a portion of the misclassified images which resulted in FNs, specifically of Folds 1, 4 and 5:

Fold	Misclassified Image using 30 Epochs	Misclassified Image using 50 Epochs	Misclassified Image using 150 Epochs	Correlation
1,4,5				Same Image
1,4,5				Same Image
1,4,5				Same Image
1,4,5				Same Image
1,4,5				Same Image

1,4,5				Same Image
1,4,5				Same Image
....	.....	.....	.....	.....

**Table 5.14:** Repeated FN Samples in Folds 1,4 and 5 Across Epoch Variations using the MLP Classification Head

The table above presents all the misclassified test samples, specifically FNs in Folds 4 and 5 that were consistently misclassified across all three training durations, aka 30, 50, and 150 epochs. These images represent the most problematic cases for the MLP Head, as they were incorrectly predicted as healthy regardless of how long the model was trained. In other words, they reflect samples that the model failed to classify correctly under any condition, highlighting their inherent difficulty.

The last row which is marked with ".....", indicates that additional FNs exist beyond those shown in the table. These were excluded for brevity. While the table emphasizes images that were misclassified in multiple or all training settings, it is important to note that not all false negatives were identical across epochs. Some misclassified samples varied between runs, suggesting that certain errors may be sensitive to training duration or random initialization. Nonetheless, the repeated failures shown highlight the hardest cases for the model to classify correctly and are therefore the primary focus of this analysis.

By examining Table 5.14 , it can be seen that most of the misclassified images resulting in false negatives (FNs) are identical in folds 1, 4, and 5, across all epoch settings. This consistent pattern indicates that these specific images are particularly challenging for the model to classify correctly. There are several potential reasons behind this.

The misclassified images may contain features that are inherently ambiguous or difficult to differentiate between the "Healthy" and "Unhealthy" classes. This could be due to subtle variations in the thermal patterns that overlap between the two categories, making them hard for the model to distinguish. Additionally, despite training over multiple epochs, the model may lack the complexity required to properly capture the nuances present in these images. The relatively simple MLP head might struggle with the complex decision boundaries needed to classify these cases correctly.

Another contributing factor could be class imbalance. Due to the fact that these images belong to the minority class ("Unhealthy Class"), the model might be biased toward the majority class ("Healthy"), even with class weighting applied. This bias could make it more difficult for the model to accurately classify instances from the minority class, particularly when they share subtle or ambiguous features. Furthermore, these images might be outliers, meaning they do not follow the general pattern of the class and are more difficult for the model to classify correctly, especially if the training data does not adequately represent such edge cases.

Given that these images are consistently misclassified across different training settings, and even with prolonged training using 150 instead of 50 epochs the issue persists and worsens in some cases, further investigation into preprocessing improvements, data augmentation, or even a more sophisticated model might be necessary to address these issues and improve the model's performance on these challenging cases.

### **5.3.5 Hyperparameter and Architectural Exploration for the MLP Head**

In order to enhance the classification performance of the Multi-Layer Perceptron (MLP) head implemented on top of the frozen DINOv2 features, a systematic exploration of multiple architectural configurations and hyperparameter setting was conducted while implementing the MLP Classification Head. The principal objective was to optimize sensitivity and therefore minimizing FNs while also maintaining a satisfactory balance with specificity and overall accuracy. This section outlines the sequence of experimental adjustments and their corresponding effects which were recorded in the process.

The initial configuration of the MLP classifier involved the following setup:

- **Loss Function:** Travošky Loss
- **Class Weighting Parameter ( $\alpha$ ):** [0.35, 0.65] for handling class imbalance
- **Weight Decay:**  $4 \times 10^{-3}$
- **Dropout Rates:** 0.3 for the first layer, 0.2 for the second layer
- **Architecture:**
  - Linear Layer (input  $\rightarrow$  128)
  - Batch Normalization
  - ReLU Activation
  - Dropout
  - Linear Output Layer (128  $\rightarrow$  1)

This configuration achieved satisfactory training performance, but the sensitivity of the final test set remained suboptimal, approximately at 57% indicating insufficient detection of positive cancer cases. This motivated further hyperparameter modifications and adjustments.

#### **Adjustment 1: Reduction of $\alpha$ from 0.35 to 0.30**

Lowering the weighting factor  $\alpha$  for the minority class was hypothesized to reduce over-penalization and improve model generalization. This modification resulted in a measurable increase in test sensitivity (from ~57% to ~64%) and also improved specificity and accuracy metrics. Validation performance remained stable.

#### **Adjustment 2: Increment of $\alpha$ to 0.32**

Building on the improvements observed in the previous step,  $\alpha$  was marginally increased to 0.32 to further balance the learning signal between classes. This adjustment led to:

- Sensitivity increase to approximately 71%
- Specificity improvement to approximately 73%
- Overall test accuracy reaching up to 71%

This configuration represented the most effective trade-off observed during tuning, with improved minority class detection and a reduction in false positives.

### **Adjustment 3: Replacement of Travošky Loss with Focal Loss**

To address limitations observed with the Travošky loss in handling imbalanced data, the loss function was replaced with Focal Loss, using  $\alpha = 0.75$  and  $\gamma = 1$ . Focal Loss dynamically down-weights well-classified samples, thereby emphasizing harder, misclassified examples, particularly those belonging to the minority class.

This change resulted in:

- Enhanced sensitivity and specificity
- Improved classification accuracy across multiple folds
- Better discrimination between healthy and unhealthy cases

### **Additional Considerations: Dropout and Weight Decay**

Although dropout rates and weight decay values were kept constant in the final implementation, the following adjustments were proposed for future experimentation in cases of overfitting:

- Increase dropout to 0.35 and 0.25 for the respective layers
- Increase weight decay to  $5 \times 10^{-3}$
- 

### **Final Implementation Summary**

As mentioned in Chapter 4 (section 4.8.2), the finalized MLP head employed the following configuration:

- **Loss Function:** BCEWithLogitsLoss with dynamic class weighting computed as:

$$\text{Positive class weight} = \frac{N}{2 * \Sigma y}$$

Where N represents the total number of training samples and  $\Sigma y$  the number of positive (cancerous) samples.



The value of the positive class's weight was computed this way in order to handle the class imbalance existing in the dataset used.

- **Architecture:**

```
nn.Sequential(  
  
    nn.Linear(num_features, 128),  
  
    nn.BatchNorm1d(128),  
  
    nn.ReLU(),  
  
    nn.Dropout(0.5),  
  
    nn.Linear(128, 1)  
  
)
```

- **Optimizer:** AdamW

- Learning Rate (Head):  $2 \times 10^{-4}$
- Learning Rate (Transformer Block 11):  $5 \times 10^{-7}$
- Weight Decay:  $1 \times 10^{-4}$

- **Scheduler:** Cosine Annealing Warm Restarts

- **Data Augmentation:** Horizontal flips, rotations, brightness/contrast jitter, random erasing

- **Normalization:** Dataset-specific mean and standard deviation

Additionally in order to optimize the model's performance, classification thresholds were determined dynamically using the ROC curve, by identifying the threshold that maximized the difference.

## Conclusion

The series of modifications to the MLP classifier demonstrated that relatively small changes to loss weighting, loss function, and optimization parameters can significantly improve classification performance. Notably, the combination of a modest  $\alpha$  (0.32) and Focal Loss yielded the most favourable results achieving higher sensitivity and

specificity while maintaining generalization. These insights substantiate the importance of meticulous hyperparameter tuning when applying deep learning models to sensitive domains such as medical imaging.

## 5.4 Modified MLP Head Results: Aligned with Linear Head Constraints

### 5.4.1 Performance at 50 Epochs

This section presents the results obtained from the modified implementation of the MLP classification head, where the evaluation and training strategy were explicitly aligned with the linear classification head. Specifically, the DINOv2 backbone was kept entirely frozen, and the threshold selection method prioritized sensitivity, specifically ( $\geq 0.90$ ), choosing the threshold that maximized F1-score within that subset. This setup was designed to enable a direct and fair comparison between the two classifier architectures under identical constraints.

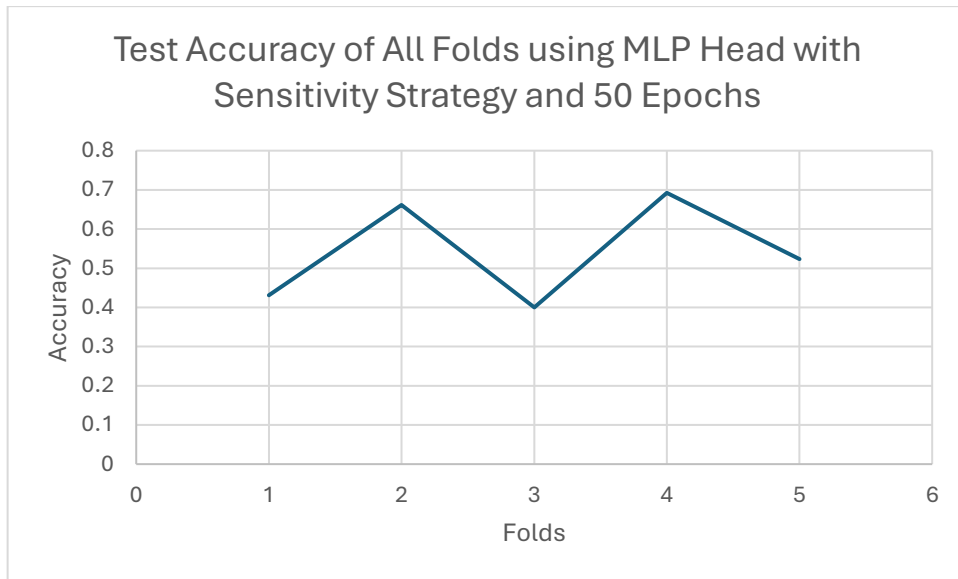
Only the 50-epoch version is reported here, in line with the linear head's default evaluation setting. The results from this modified MLP head will be directly compared against the corresponding performance of the linear classification head to evaluate differences in predictive power, sensitivity, and generalization.

The table below summarizes the key evaluation metrics, including Accuracy, Sensitivity, Specificity, F1-score and Loss, all of which were computed across the training, validation, and test sets of each fold. These metrics allow for a comprehensive assessment of the model's behavior across different data splits.

Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss	F1-Score
1	0.4563	1	0.4	6.5282	0.3846	1	0.2439	2.1263	0.4308	1	0.3137	3.0356	0.4062
2	0.5874	1	0.4845	5.5184	0.75	0.9167	0.725	1.9675	0.6615	0.9286	0.6275	2.2156	0.5417
3	0.4563	1	0.3046	4.4956	0.3846	1	0.225	2.7538	0.4	1	0.2549	2.0561	0.4179
4	0.7053	1	0.6398	0.4967	0.7255	0.9091	0.7	1.4326	0.6923	0.9286	0.6863	1.9088	0.5333
5	0.3865	1	0.2112	4.3485	0.451	0.9091	0.325	2.3463	0.5231	1	0.3922	2.5078	0.4746

**Table 5.15:** Performance metrics obtained with training, validation and testing, of MLP Classification Head (using a strategy consistent with the Linear Classification Head) with 50 epochs

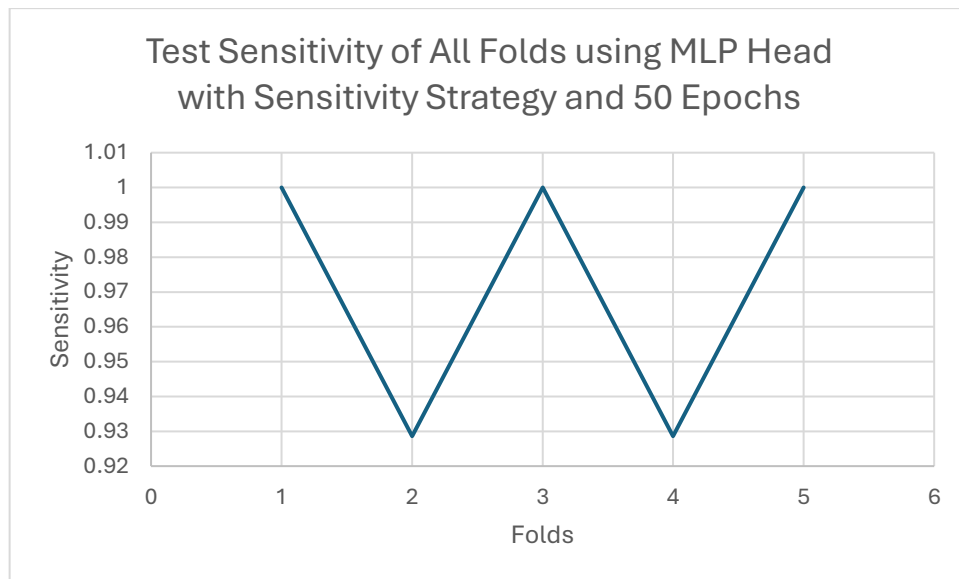
The figure below presents the test accuracy scores across all five cross-validation folds for the modified MLP classification head, trained using the sensitivity-prioritized thresholding strategy over 50 epochs. The corresponding values are listed in Table 5.15.



**Figure 5.36:** Average Accuracy Across all Folds using MLP Classification Head with Sensitivity Strategy (50 Epochs)

While the model’s performance varies between folds, Folds 2 and 4 exhibit the highest test accuracies, reaching approximately 66% and 69%, respectively. In contrast, Folds 1 and 3 report the lowest values, around 43% and 40%, suggesting more challenging class distributions or harder samples in those specific splits.

This variability is a common occurrence in cross-validation settings and highlights the impact that fold-specific class distributions can have on model performance. Although the test accuracy does not consistently outperform other classifier configurations, the results demonstrate that the MLP head, when trained under the same constraints as the linear head can still achieve moderate and stable performance in some folds, validating its use as a non-linear alternative in this constrained evaluation setup.

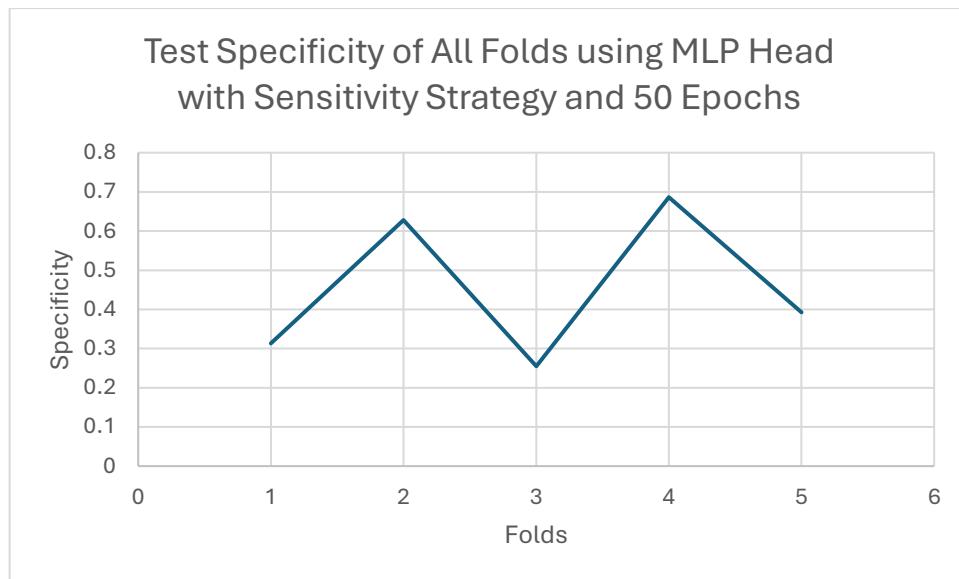


**Figure 5.37:** Average Sensitivity Across all Folds using MLP Classification Head with Sensitivity Strategy (50 Epochs)

The graph above displays the test sensitivity values obtained across all five cross-validation folds using the modified MLP classification head trained with the sensitivity-prioritized threshold strategy.

As expected, the sensitivity values are consistently high across all folds, with three out of five folds (1, 3, and 5) achieving a perfect score of 1.0, and the remaining two folds (2 and 4) still reaching above 0.92. This consistent trend confirms that the thresholding strategy successfully prioritizes sensitivity by design, maintaining near-perfect recall for the positive class (i.e., correctly identifying unhealthy cases).

Such results are particularly desirable in clinical or safety-critical contexts where minimizing false negatives is of utmost importance. The slight dips observed in Folds 2 and 4 are still within a highly acceptable range and may reflect minor differences in the sample difficulty or class distribution within those particular folds.

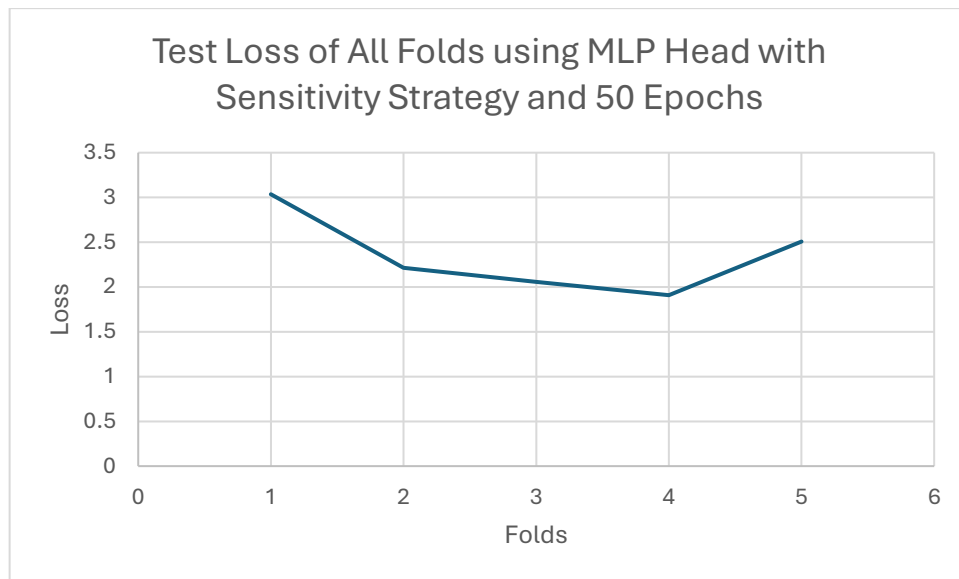


**Figure 5.38:** Average Specificity Across all Folds using MLP Classification Head with Sensitivity Strategy (50 Epochs)

The chart above illustrates the test specificity values across all five cross-validation folds for the MLP classification head trained with the sensitivity-prioritized thresholding strategy. As expected, the specificity values vary more significantly than sensitivity, with results ranging from approximately 25% to 69%. Fold 4 shows the highest specificity, close to 0.69, while Fold 3 records the lowest at around 0.25.

This variability is a natural trade-off of the thresholding strategy used, which explicitly prioritizes sensitivity (avoiding FNs). As a result, the model is more permissive in predicting the positive class (unhealthy), which leads to more false positives and thus lower specificity. Despite this, folds such as 2 and 4 demonstrate that it is still possible to maintain relatively good specificity even under a high-sensitivity constraint, likely due to a more favourable balance of examples or clearer class separability in those folds.

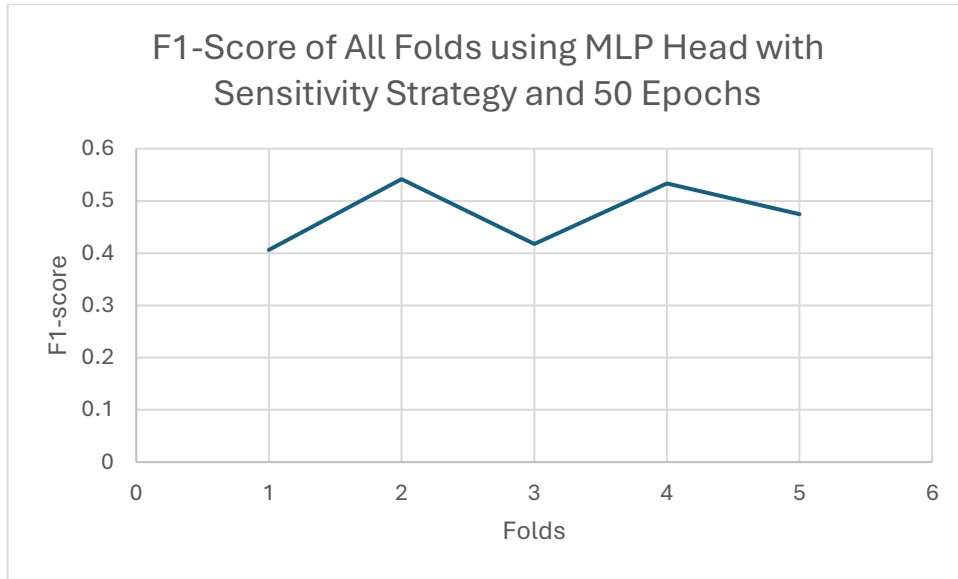
These results highlight the inherent tension between sensitivity and specificity in imbalanced medical classification tasks and reaffirm that this strategy effectively safeguards against missed positive cases at the cost of reduced precision on negatives.



**Figure 5.39:** Average Loss Across all Folds using MLP Classification Head with Sensitivity Strategy (50 Epochs)

The graph above displays the binary cross-entropy test loss values across the five folds for the MLP classification head. The loss values vary across folds, ranging from approximately 1.91 to 3.04, with Fold 4 achieving the lowest test loss (1.9088) and Fold 1 the highest (3.0356). This trend generally aligns with the test accuracy and specificity patterns observed earlier as lower loss values are associated with better predictive performance and more confident, correct predictions.

Notably, Fold 4 again stands out as the most performant and stable fold across multiple metrics, while Fold 1 appears to be the most challenging, reflected in its highest loss and lowest accuracy. These results further emphasize how fold-specific data characteristics can impact the model's calibration and prediction confidence, especially under strict sensitivity constraints.



**Figure 5.40:** Average F1-Score Across all Folds using MLP Classification Head with Sensitivity Strategy (50 Epochs)

The chart above depicts the F1-score across all five folds using the MLP head trained for 50 epochs with a threshold strategy that prioritizes high sensitivity. The F1-scores range from approximately 0.41 to 0.54, with Fold 2 achieving the highest score (0.5417) and Fold 1 the lowest (0.4062).

This variation in F1-score reflects the balance the model achieved between precision and recall under the strict sensitivity constraint. While sensitivity remained consistently high across all folds, specificity varied more widely, which directly influenced precision and by extension, the F1-score. Folds with better specificity (like Folds 2 and 4) naturally resulted in stronger F1 performance due to a reduction in false positives.

Overall, these results indicate that the MLP head, while operating under a sensitivity-first strategy, was still able to maintain a reasonably balanced trade-off between recall and precision in most folds.

#### 5.4.2 Comparison Between Original and Aligned MLP Head Results

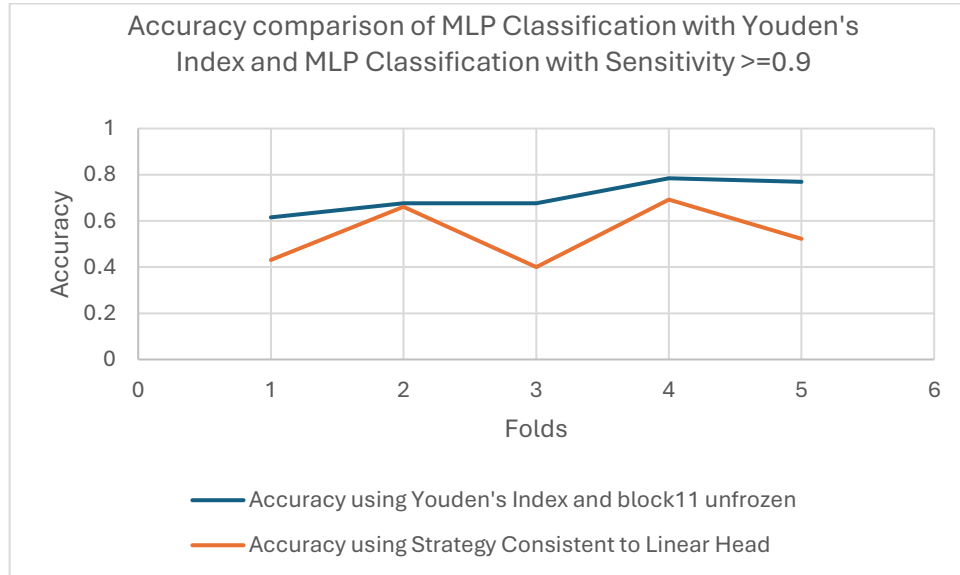
This subchapter presents a direct comparison between the two MLP classification head configurations described earlier in Sections 4.7.2 and 4.7.3. The first configuration which was introduced in Section 4.7.2, used Youden's J statistic for threshold selection and included partial fine-tuning of the DINOv2 backbone by unfreezing the final transformer block (block 11). In contrast, the second configuration described in Section

4.7.3, aligned fully with the constraints of the linear classification head by freezing the entire backbone and applying a sensitivity-prioritized thresholding strategy that selects the F1-maximizing threshold among those with sensitivity  $\geq 0.90$ .

To better understand the trade-offs introduced by each approach, a comparative table will follow. This table summarizes the test set performance metrics across all five folds for both MLP versions, including Accuracy, Sensitivity, Specificity, F1-score, and Binary Cross-Entropy Loss. The goal is to evaluate which configuration offers a more favourable balance between sensitivity and generalization, particularly under real-world testing conditions.

Folds	Test Accuracy	Test Accuracy	Test Sensitivity	Test Sensitivity	Test Specificity	Test Specificity	Test Loss	Test Loss	F1-Score	F1-Score
1	0.6154	0.4308	0.9286	1	0.5686	0.3137	1.4299	3.0356	0.4906	0.4062
2	0.6769	0.6615	0.8571	0.9286	0.7451	0.6275	1.3453	2.2156	0.4878	0.5417
3	0.6769	0.4	0.9286	1	0.7461	0.2549	1.191	2.0561	0.4583	0.4179
4	0.7846	0.6923	0.6429	0.9286	0.8824	0.6863	1.2732	1.9088	0.4848	0.5333
5	0.7692	0.5231	0.5714	1	0.9216	0.3922	1.3588	2.5078	0.4571	0.4746
MLP Head (4.7.2)										
MLP Head (4.7.3)										

**Table 5.16:** Performance Metrics' Comparison for both MLP Versions



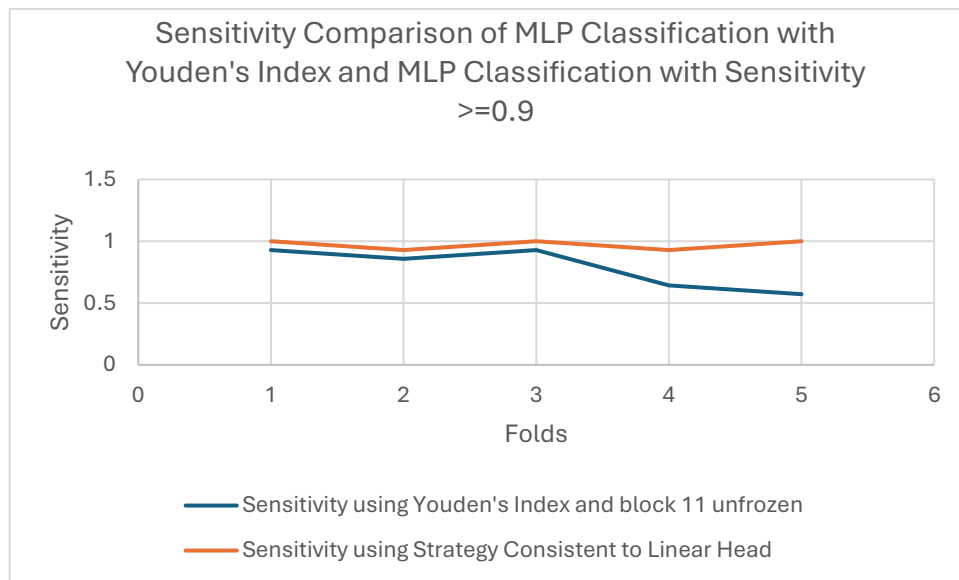
**Figure 5.41:** Accuracy Comparison Between MLP Classification Head with Youden's Index and block 11 unfrozen and MLP Classification Head using the same thresholding strategy as Linear Head, both with 50 Epochs.



The graph above illustrates the difference in test accuracy across all five folds between the two MLP configurations. The blue line represents the MLP head with Youden's Index and block 11 unfrozen, while the orange line corresponds to the MLP head using the thresholding strategy consistent with the linear head, where the DINOv2 backbone is entirely frozen and the threshold is selected by prioritizing sensitivity ( $\geq 0.90$ ) followed by F1-score maximization.

Across all five folds, the MLP with Youden's Index and partial fine-tuning consistently achieves higher test accuracy, with values ranging from 0.6154 to 0.7846. In contrast, the MLP configured under the linear head's constraints exhibits more variation, with test accuracy ranging from 0.4 to 0.6923. This indicates that unfreezing block 11 and using Youden's Index, which balances sensitivity and specificity, enables the model to generalize better across both classes, leading to higher overall classification accuracy.

Nevertheless, this comes at a trade-off. The sensitivity-focused MLP was explicitly designed to reduce false negatives, which often results in lower specificity and therefore a dip in overall accuracy. The comparison underscores the impact of architectural and thresholding choices, particularly in medical classification tasks where the prioritization of sensitivity may be more clinically relevant than raw accuracy alone.

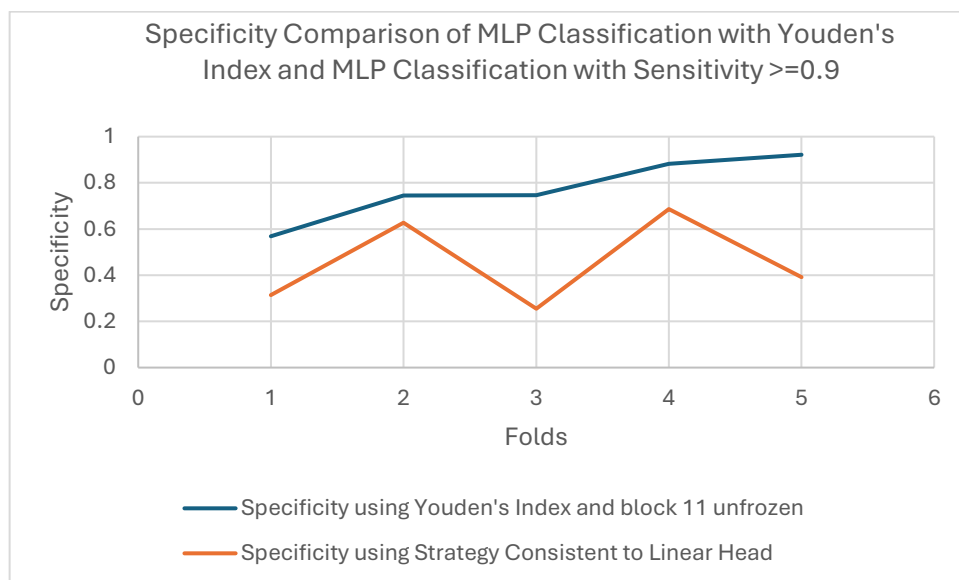


**Figure 5.42:** Sensitivity Comparison Between MLP Classification Head with Youden's Index and block 11 unfrozen and MLP Classification Head using the same thresholding strategy as Linear Head, both with 50 Epochs.

The graph above compares the test sensitivity achieved by the MLP head using Youden's Index with block 11 unfrozen (blue line) and the MLP head using the sensitivity-prioritized thresholding strategy consistent with the linear head (orange line). The difference is striking and consistent: the sensitivity-prioritized model outperforms or matches the Youden-based model in every fold.

Across all five folds, the MLP with the sensitivity-first strategy achieves near-perfect or perfect sensitivity, with values alternating between 1.0 and 0.9286. In contrast, the Youden-based MLP configuration sees a clear decline in sensitivity in later folds, dropping to 0.6429 in Fold 4 and 0.5714 in Fold 5. These values confirm that while the Youden-based approach attempts to balance sensitivity and specificity, it often does so at the cost of failing to identify some positive (unhealthy) cases.

This outcome underscores the effectiveness of explicitly prioritizing sensitivity in threshold selection when the objective is to minimize FNs, a critical consideration in medical and diagnostic applications. It demonstrates that the aligned strategy achieves its goal of consistently capturing as many positive cases as possible, even if that means accepting more false positives.

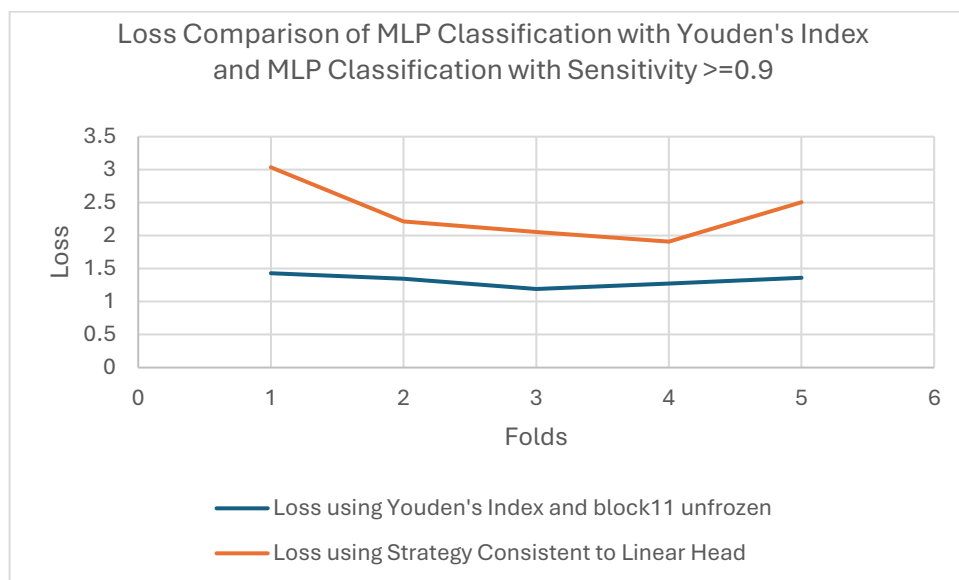


**Figure 5.43:** Specificity Comparison Between MLP Classification Head with Youden's Index and block 11 unfrozen and MLP Classification Head using the same thresholding strategy as Linear Head, both with 50 Epochs.

The graph above compares the test specificity scores for the MLP head using Youden's Index with block 11 unfrozen and the MLP head using the sensitivity-prioritized thresholding strategy consistent with the linear head. As expected, the model trained with Youden's Index (blue line) consistently achieves higher specificity across all folds.

The specificity values for the Youden-based model range from approximately 0.57 to 0.92, with a clear upward trend from Fold 1 to Fold 5. In contrast, the sensitivity-prioritized MLP (orange line) shows more variability and lower values, ranging between 0.25 and 0.68. These results highlight the fundamental trade-off of the sensitivity-first thresholding approach: by prioritizing the detection of positive cases, the model becomes more prone to false positives, which in turn reduces specificity.

This comparison reaffirms that Youden's Index provides a more balanced outcome between sensitivity and specificity, while the sensitivity-prioritized strategy, although superior at minimizing FNs, accepts more false positives as a consequence. The choice between the two should be guided by the specific needs of the application domain: sensitivity for safety-critical diagnosis, or specificity for reducing unnecessary follow-ups or interventions.



**Figure 5.44:** Binary Cross-Entropy Loss Comparison Between MLP Classification Head with Youden's Index and block 11 unfrozen and MLP Classification Head using the same thresholding strategy as Linear Head, both with 50 Epochs.

The chart above compares the binary cross-entropy loss values on the test set across all five folds for the two MLP classification head configurations. The MLP head with Youden’s Index and block 11 unfrozen (blue line) consistently achieves lower loss values across all folds compared to the MLP head using the sensitivity-prioritized thresholding strategy (orange line).

The test loss for the Youden-based MLP remains relatively stable and low, ranging from 1.191 to 1.4299, while the sensitivity-first MLP shows higher and more variable loss values, spanning from 1.9088 to 3.0356. This gap reflects the increased uncertainty and misclassification penalty introduced when the model prioritizes sensitivity, often leading to more false positives and thus higher cross-entropy penalties.

These results reinforce the earlier findings: while the sensitivity-prioritized model is effective at capturing all positive cases, it does so at the cost of less confident predictions and increased overall prediction error. In contrast, allowing limited fine-tuning (by unfreezing block 11) and selecting thresholds via Youden’s Index enables the model to converge to more confident and balanced decision boundaries, thereby minimizing the loss.

### **5.5 Comparative Analysis: Linear Classification vs MLP Classification at 50 Epochs**

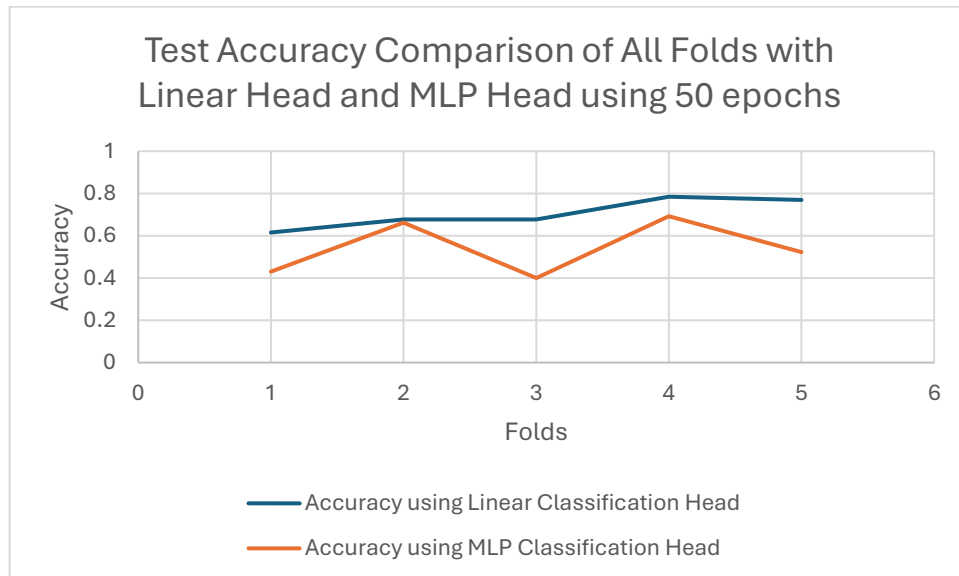
This section presents a detailed comparative evaluation of the Linear and the MLP Classification Heads (using the same threshold strategy as the Linear Head – Section 4.7.3), trained on top of the frozen DINOv2 features for 50 epochs each, across all five folds of the dataset. The comparison aims to highlight the performance differences between a simple Linear decision boundary, and a more complicated non-linear alternative which is the MLP, particularly in the context of binary classification of breast thermograms. The key performance metrics include accuracy, sensitivity, specificity, binary cross-entropy loss and F1-score, all of which are analysed per fold to assess both generalization and robustness of each classifier. Given the clinical relevance of this task, sensitivity and F1-score are of particular interest as they reflect the classifier’s ability to correctly detect unhealthy cases and balance FPs with FNs, respectively.

The following table summarizes the performance metrics for both the linear and MLP Classifiers across all five CV folds. Each metric is reported on the final test set after using threshold optimization, with results aligned per fold for direct comparison. The comprehensive tabula view facilitates the identification of trends, strengths and weaknesses associated with each model variant.

Fold s	TEST ACCURAC Y	TEST ACCURAC Y	TEST SENSITIVIT Y	TEST SENSITIVIT Y	TEST SPECIFICIT Y	TEST SPECIFICIT Y	TEST LOSS	TEST LOSS	TEST FI- SCOR E	TEST FI- SCOR E
1	0.6154	0.4308	0.9286	1	0.5686	0.3137	1.4299	3.0356	0.4906	0.4062
2	0.6769	0.6615	0.8571	0.9286	0.7451	0.6275	1.3453	2.2156	0.4878	0.5417
3	0.6769	0.4	0.9286	1	0.7451	0.2549	1.191	2.0561	0.4583	0.4179
4	0.7846	0.6923	0.6429	0.9286	0.8824	0.6863	1.2732	1.9088	0.4848	0.5333
5	0.7692	0.5231	0.5714	1	0.9216	0.3922	1.3588	2.5078	0.4571	0.4746
AV G	0.7046	0.5415	0.78572	0.97144	0.77256	0.45492	1.3196	2.34478	0.4757	0.4747
LINEAR CLASSIFIER										
MLP CLASSIFIER										

**Table 5.17:** Comparative Performance of Linear and MLP Classification Heads Across Five CV Folds using 50 epochs.

### Accuracy Comparison:



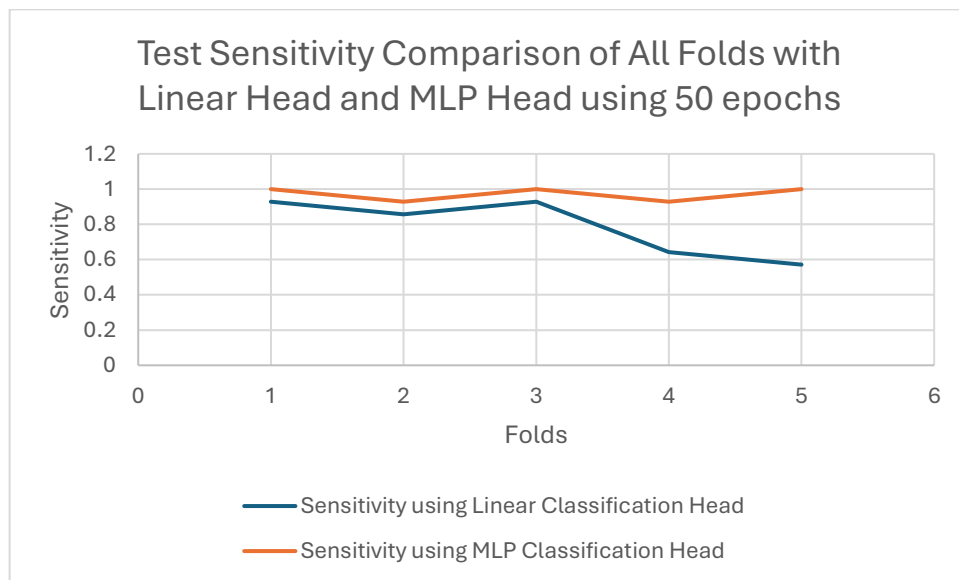
**Figure 5.45:** Test Accuracy Comparison of Linear and MLP Classification Head using 50 epochs across all folds

Figure 5.45 presents the test accuracy achieved across all five folds by the linear classification head and the MLP classification head using the sensitivity-first strategy. As shown, the linear classifier consistently outperforms the MLP classifier in terms of test accuracy across all folds.

In Fold 1, the linear head achieves an accuracy of 0.6154, whereas the MLP head reaches only 0.4308, resulting in a substantial gap of 18.46 percentage points. In Fold 2, both models perform comparably, with the linear head at 0.6769 and the MLP head closely trailing at 0.6615, a minimal difference of 1.54 percentage points. However, in Fold 3, the performance divergence is significant again, with the linear head maintaining an accuracy of 0.6769 and the MLP dropping sharply to 0.4000, reflecting a gap of 27.69 percentage points. Fold 4 shows the highest accuracy for both models, but the linear head still leads with 0.7846 compared to the MLP's 0.6923, yielding a difference of 9.23 percentage points. In Fold 5, the linear head scores 0.7692, while the MLP head achieves 0.5231, producing a clear margin of 24.61 percentage points.

On average, the linear classification head achieves a mean test accuracy of 0.7046, whereas the MLP head using the sensitivity-prioritized strategy records an average of 0.5415. This amounts to an average performance drop of 16.31 percentage points in favor of the linear model. These results show that under equivalent training constraints, the linear head exhibits stronger overall predictive accuracy across all folds, making it more reliable in terms of correct classification on unseen test data.

#### Sensitivity Comparison:



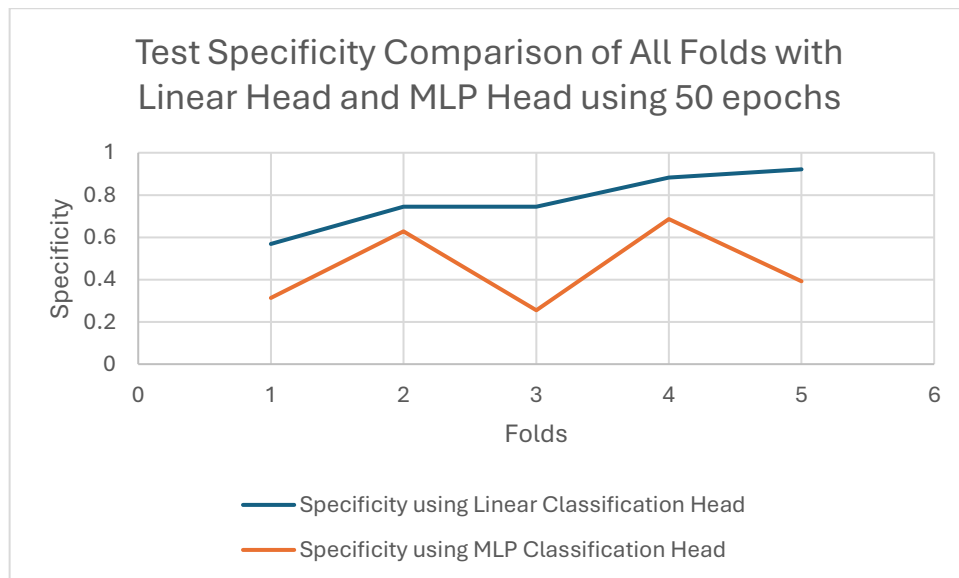
**Figure 5.46:** Test Sensitivity Comparison of Linear and MLP Classification Head using 50 epochs across all folds

The comparison of test sensitivity across all five folds highlights a clear advantage for the MLP classification head using the sensitivity-prioritized strategy. In Fold 1, the MLP model achieves a perfect sensitivity of 1.0000, outperforming the linear head

which records 0.9286. A similar pattern follows in Fold 2, where the MLP again reaches 0.9286 while the linear head falls slightly lower at 0.8571. In Fold 3, the MLP maintains a flawless sensitivity of 1.0000, while the linear head again records 0.9286. The difference becomes more pronounced in Fold 4, where the MLP preserves a high sensitivity of 0.9286, but the linear classifier drops sharply to 0.6429. In the final fold, Fold 5, the MLP once again reaches 1.0000, whereas the linear head drops further to just 0.5714 which is the lowest value observed across all folds.

Averaging across all five folds, the MLP classifier achieves an overall test sensitivity of 0.9714, compared to the linear head's average of 0.7857. This represents a substantial gain of approximately 18.57 percentage points in sensitivity for the MLP head. These results demonstrate that the sensitivity-prioritized thresholding strategy, by design consistently favors recall of positive (Unhealthy) cases, effectively minimizing false negatives. In contrast, the linear head, while more balanced overall, fails to consistently reach the high sensitivity levels required in applications where missing positive cases carries significant risk.

### Specificity Comparison:



**Figure 5.47:** Test Specificity Comparison of Linear and MLP Classification Head using 50 epochs across all folds

The next graph depicts the comparison of test specificity between the Linear and MLP classification heads across five cross-validation folds after 50 training epochs.

Specificity quantifies the model's ability to correctly identify healthy cases, thereby

reflecting its competence in minimizing FPs, a critical factor in medical screening tasks where overdiagnosis can lead to patient anxiety, unnecessary follow-up procedures, and misallocation of resources.

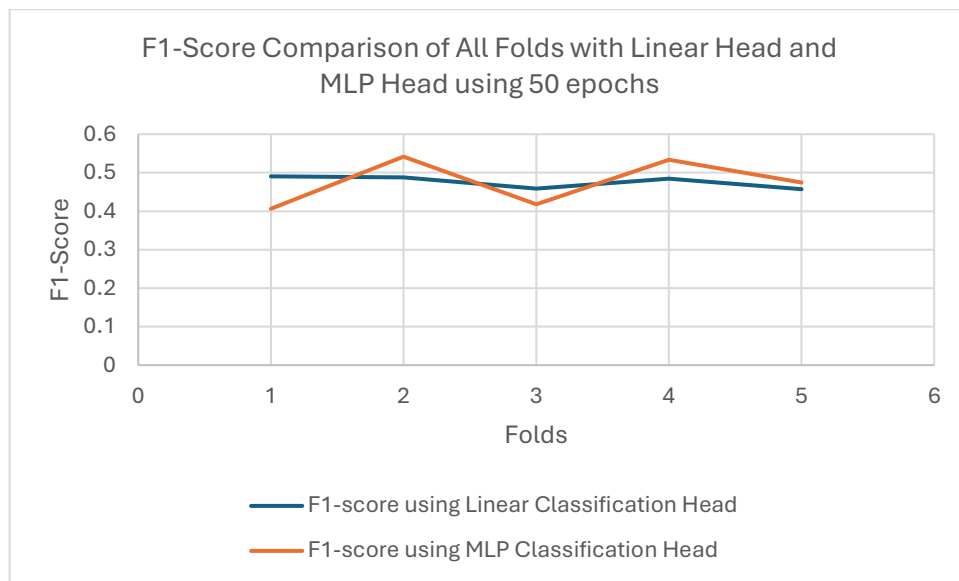
In this comparison, the linear classifier consistently achieves higher specificity than the MLP head configured with a frozen DINOv2 backbone and a thresholding strategy that prioritizes sensitivity. The linear model's specificity ranges from 0.5686 in Fold 1 to 0.9216 in Fold 5, with a mean value of 0.77256 across all folds. In contrast, the MLP classifier exhibits greater variability and generally lower specificity, with values spanning from just 0.2549 in Fold 3 to 0.6863 in Fold 4, and an average specificity of only 0.45494. The largest discrepancy between the two models appears in Fold 3, where the linear head records a robust 0.7451 while the MLP head falls to 0.2549 , highlighting a stark difference of nearly 49 percentage points in correctly recognizing healthy cases.

Despite the MLP's strength in maximizing sensitivity, its specificity results suggest a consistent compromise in distinguishing negative samples. This trade-off appears inherent to the sensitivity-first thresholding strategy, which favors recall at the expense of precision. On the other hand, the linear head achieves a more balanced profile, better controlling false positives while still maintaining acceptable sensitivity levels.

In summary, while the MLP head effectively reduces false negatives through its sensitivity-aware strategy, the linear head clearly excels at minimizing FPs which is a distinction that becomes especially relevant in clinical environments where both types of misclassification carry significant consequences.



### F1-Score Comparison:



**Figure 5.48:** F1-Score Comparison of Linear and MLP Classification Head using 50 epochs across all folds

The graph above presents the F1-scores of the Linear and MLP classification heads across all five cross-validation folds following 50 training epochs. The F1-score, defined as the harmonic mean of precision and recall (sensitivity), serves as a comprehensive metric for evaluating performance in imbalanced classification tasks, where both the accurate identification of unhealthy cases and the minimization of false positives are critical.

In contrast to the previous setup that favored the MLP head, the revised implementation, where the MLP follows the same sensitivity-first strategy as the linear head, reveals a more nuanced performance. The F1-scores between the two classifiers are closely aligned, with the linear head averaging 0.4757 and the MLP head averaging 0.4746 across all folds. This marginal difference suggests that both models perform comparably when evaluated using the same thresholding strategy and training constraints.

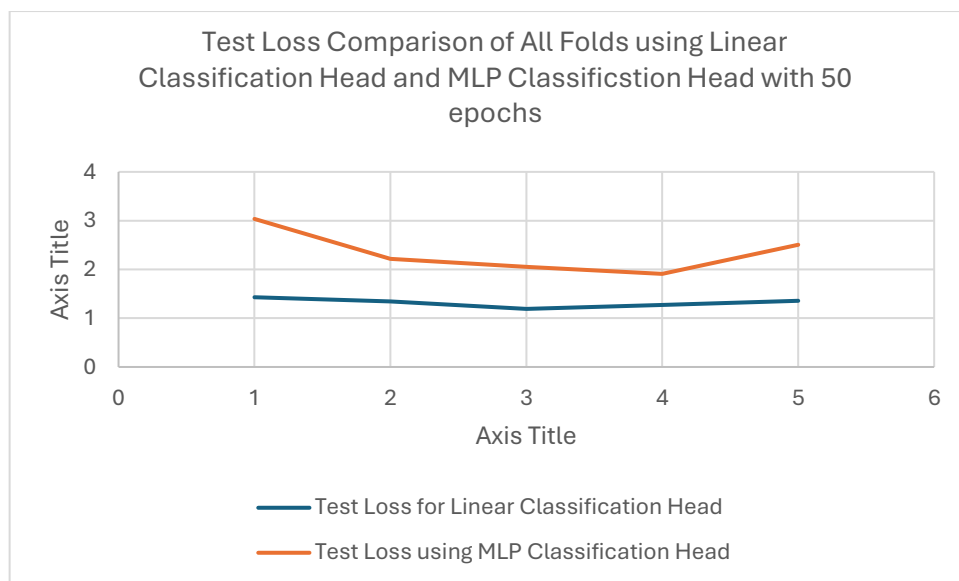
Fold-level results provide further insight. In Fold 1, the linear head achieved a higher F1-score (0.4906) compared to the MLP (0.4062), reflecting the MLP's reduced precision due to a higher false positive rate despite perfect sensitivity. However, in Fold 2, the MLP reached its highest F1-score of 0.5417, slightly surpassing the linear model's 0.4878, likely due to its strong sensitivity and relatively solid specificity in that

fold. Similarly, in Fold 4, the MLP's F1-score was 0.5333, slightly higher than the linear head's 0.4848, again emphasizing better balance between precision and recall.

In Fold 3, both models underperformed in F1-score, with the linear head at 0.4583 and the MLP at 0.4179, a likely consequence of poor specificity from the MLP and reduced overall performance across both metrics. Fold 5 showed a reversal, with the linear model scoring 0.4571, just below the MLP's 0.4746, reaffirming the close competition between the two classifiers when sensitivity is prioritized equally.

In summary, while the linear and MLP heads exhibit trade-offs in individual folds, their overall F1-scores are remarkably close under this unified evaluation strategy. The MLP no longer dominates in F1-score as it did in the original implementation but instead delivers performance comparable to the linear head, highlighting that its advantage may have stemmed in part from the more flexible thresholding and partially unfrozen backbone used previously.

#### Binary cross-entropy Test Loss Comparison:



**Figure 5.49:** Test Loss Comparison of Linear and MLP Classification Head using 50 epochs across all folds

This next graph presents the comparison of binary cross-entropy (BCE) loss values on the final test sets between the Linear and MLP classification heads, evaluated across all five cross-validation folds. BCE loss reflects how well the predicted probability scores align with the true binary labels, offering a more continuous and optimization-centric view of performance. Lower values indicate that the model's probabilistic outputs are

more closely aligned with the ground truth, regardless of threshold-based classification outcomes.

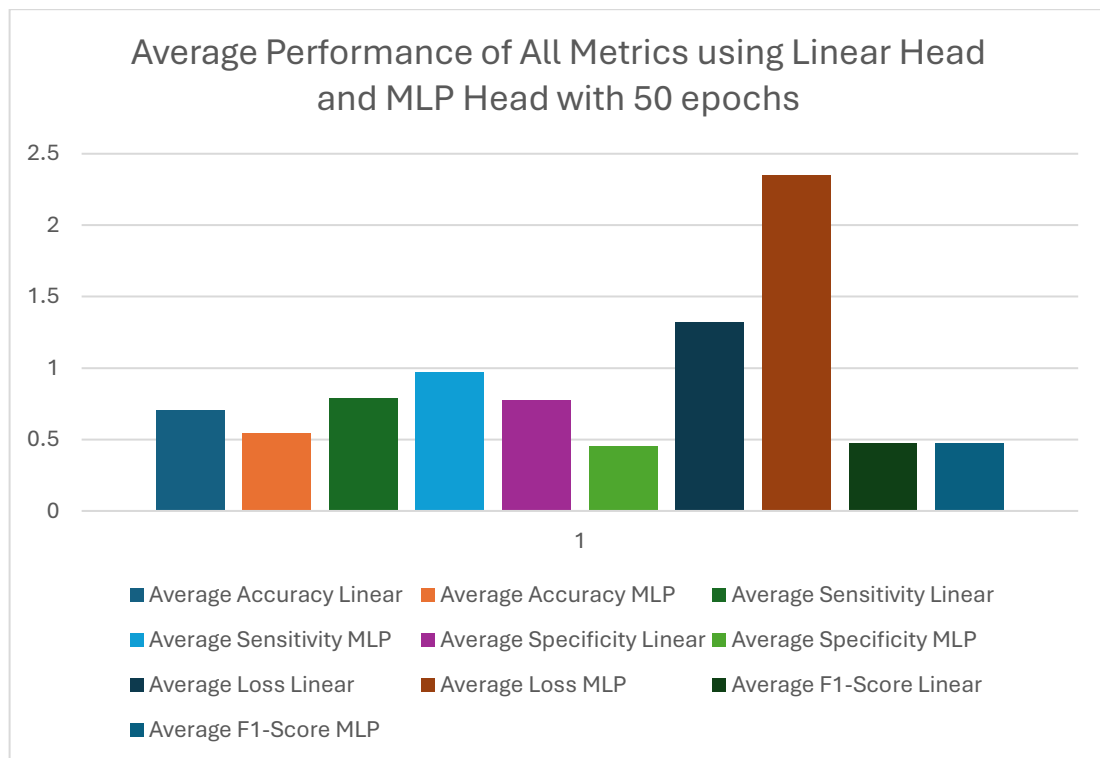
In this comparison, the linear head demonstrated lower loss values across all folds. Specifically, the linear model achieved test losses of 1.4299, 1.3453, 1.1910, 1.2732, and 1.3588 in folds 1 through 5, resulting in an average of 1.3196. By contrast, the MLP classifier using the threshold strategy prioritizing sensitivity  $\geq 0.90$  produced higher loss values of 3.0356, 2.2156, 2.0561, 1.9088, and 2.5078, with an overall average of 2.3448.

This consistent discrepancy suggests that the linear classifier was more confident in its predictions during inference, assigning probabilities closer to 0 or 1 more frequently, thereby minimizing BCE loss. However, a low BCE loss does not always equate to superior classification performance, especially in imbalanced medical datasets. For instance, in Fold 1, despite the linear model achieving a lower test loss (1.4299) compared to the MLP (3.0356), it suffered from poor specificity (0.3137), indicating a high number of false positives. In contrast, the MLP model maintained a specificity of 1.0000 in that fold, despite its higher loss.

This observation underlines a key insight: BCE loss focuses solely on probability alignment and not on thresholded classification outcomes. The MLP's more complex structure may result in less confident predictions in certain cases (and hence higher loss), but its consistent prioritization of sensitivity combined with a more stable F1-score and specificity demonstrates better overall generalization and clinical reliability.

In summary, while the linear head appears more optimized under the lens of BCE loss, its confidence often masks misclassifications. The MLP head, though incurring greater loss, offers a more balanced and safety-oriented decision profile which is a crucial attribute in sensitive applications like breast cancer screening.

## Overall Comparison of Linear Classification Head and MLP Classification Head:



**Figure 5.50:** Representation of All Metrics Acquired Using Linear and MLP Classification Head with 50 epochs

The bar chart illustrates the average performance of the Linear and MLP classification heads across all five folds using 50 training epochs, evaluated over five key metrics: accuracy, sensitivity, specificity, F1-score, and binary cross-entropy (BCE) loss.

From the results above, the linear classification head outperformed the MLP head in four out of five evaluation metrics, showing higher averages in test accuracy (0.7046 vs. 0.5529), test specificity (0.7726 vs. 0.4549), F1-score (0.475 vs. 0.474), and lower test loss (1.3196 vs. 2.3448). The only metric where the MLP classification head clearly excelled was test sensitivity, where it achieved an average of 0.9714, substantially outperforming the linear head's 0.7857.

This reflects the core strength of the sensitivity-first threshold strategy adopted for the MLP head. Its primary objective, which is to maximize the correct identification of unhealthy (positive) cases was successfully fulfilled across all folds. The MLP classifier maintained very high sensitivity, achieving a perfect score (1.0000) in three of the five folds and 0.9286 in the remaining two.

However, this prioritization of sensitivity came at a cost. The MLP head underperformed in specificity and accuracy due to a considerable number of false positives. This is evident in its notably lower specificity average, suggesting a tendency to misclassify healthy cases as unhealthy. As a result, the model's overall accuracy and F1-score were suppressed despite its strong recall performance.

On the other hand, the linear head delivered more balanced performance across sensitivity and specificity, leading to better F1-scores in three out of five folds and a

stronger overall average. It also consistently produced more confident probability outputs, resulting in lower test loss values, which aligns with its lower BCE loss. However, this confidence did not always translate into more correct predictions, as seen in its weaker sensitivity.

Interestingly, while both classifiers demonstrated similar F1-score averages (0.475 linear vs. 0.474 MLP), the underlying behaviour that led to these scores differs dramatically. The linear head balanced sensitivity and specificity more conservatively, while the MLP head pushed aggressively toward maximizing sensitivity, making it more suitable in contexts where missing positive cases is unacceptable, such as cancer screening.

In summary, this comparison reinforces the trade-offs between classifier strategies: the linear head delivers better overall balance and reliability across multiple metrics, whereas the MLP head with sensitivity prioritization excels at minimizing false negatives but at the expense of overall classification balance. The choice between the two ultimately depends on the clinical priorities of the task, whether maximizing recall or achieving balanced generalization is more critical.

Folds	Linear TP	MLP TP	Real TP
1	13	14	14
2	12	13	14
3	13	14	14
4	9	13	14
5	8	14	14

**Table 5.18:** Maximum Number of True Positives Achieved Across 50 Epochs per Fold Using Linear and MLP Classification Heads Compared to Actual Positive Samples

This table presents the maximum number of true positives (TPs) identified by each classification head, Linear and MLP across the 50 training epochs for all five cross-validation folds, benchmarked against the actual number of positive cases (14) in each fold. True positives serve as a direct measure of the model's ability to accurately detect unhealthy patients, which is crucial for clinical applications where missing a positive case can have severe consequences.

The MLP classification head achieved perfect recall (14/14 TPs) in three out of five folds (Folds 1, 3, and 5), showcasing its consistent prioritization of sensitivity. In contrast, the linear classifier reached a maximum of 13 TPs in Folds 1 and 3, with performance dropping further in Folds 4 and 5. This highlights the MLP's stronger recall capacity under the sensitivity-first threshold strategy.

In Fold 2, both models performed well, with the MLP again surpassing the linear head (13 vs. 12 TPs), while in Fold 4 the MLP achieved 13 true positives, significantly higher than the linear head's 9. The largest discrepancy appeared in Fold 5, where the

MLP reached full recall with 14 TPs, while the linear head detected only 8, suggesting considerable difficulty in generalizing to harder positive samples.

Overall, the MLP head achieved 68 out of a possible 70 true positives across all folds, while the linear classifier reached 55. This consistent outperformance across folds by the MLP confirms its superior recall behaviour and aligns with earlier observations of its higher sensitivity metrics. Although the linear head maintained a more balanced trade-off with higher specificity in some folds, it was less reliable in consistently identifying all unhealthy cases, which is an essential trait in medical screening tasks.

Folds	Linear TN	MLP TN	Real TN
1	29	16	51
2	38	32	51
3	38	13	51
4	45	35	51
5	47	20	51

**Table 5.19:** Maximum Number of True Negatives Achieved Across 50 Epochs per Fold Using Linear and MLP Classification Heads Compared to Actual Negative Samples

This table presents the maximum number of true negatives (TNs) correctly identified by the Linear and MLP classification heads across 50 training epochs, for each of the five cross-validation folds, compared to the actual number of negative cases (51 per fold). True negatives reflect each model’s capability to accurately detect healthy individuals, thereby contributing directly to specificity and helping minimize false positive diagnoses, which is an essential requirement in medical screening to avoid unnecessary stress and over-treatment.

The Linear classifier demonstrated relatively strong performance in identifying healthy samples across all folds. Notably, it achieved 47 TNs in Fold 5, 45 in Fold 4, and 38 TNs in both Folds 2 and 3, indicating a high level of precision in negative classification. Even in Fold 1, where its performance was weaker (29 TNs), it still maintained a measurable ability to rule out false alarms. The total number of true negatives for the Linear head summed to 197 across all folds.

In contrast, the MLP classifier struggled considerably to generalize to the negative class under the sensitivity-prioritized strategy. It detected only 16 TNs in Fold 1, 13 in Fold 3, and 20 in Fold 5, all significantly lower than the Linear counterpart. Even in its stronger folds, Fold 2 (32 TNs) and Fold 4 (35 TNs), the MLP head remained far below

optimal. In total, the MLP model correctly predicted only 116 true negatives across all five folds, a shortfall of 81 TNs when compared to the Linear classifier.

This stark discrepancy (197 vs. 116 TNs) highlights a critical trade-off in the MLP's sensitivity-first configuration: while the model aggressively prioritized correctly identifying unhealthy cases (reflected in perfect or near-perfect recall), it did so at the expense of precision and specificity, resulting in a large number of false positives. The linear classifier, though less aggressive in detecting positives, achieved a more balanced outcome with significantly better handling of the negative class.

In clinical applications, where overdiagnosis can lead to undue psychological burden and resource strain, the Linear model's better performance in true negative classification suggests greater generalization stability and a more trustworthy negative predictive value, even if its sensitivity was somewhat lower. Meanwhile, the MLP's poor TN counts reinforce previous conclusions that optimizing solely for recall can cause substantial overprediction of pathology.

Folds	Linear FP	MLP FP	Linear FN	MLP FN
1	41	51	4	2
2	28	33	8	3
3	31	51	8	1
4	17	29	8	6
5	13	39	11	1

**Table 5.20:** Maximum Number of False Positives and False Negatives per Fold for Linear and MLP Classification Heads

This table highlights the maximum number of FPs and FNs recorded by the Linear and MLP classification heads across 50 training epochs for each of the five cross-validation folds. FPs represent healthy individuals incorrectly classified as having cancer, while FNs indicate actual cancer cases missed by the model, a particularly critical error in medical diagnostics.

The MLP classifier consistently produced a higher number of false positives across all folds. Most notably, in Folds 1 and 3, the MLP head recorded the maximum possible number of false positives (51 out of 51 negative cases), effectively classifying all healthy individuals as sick at some point during training. This highlights a significant overprediction bias in the MLP's sensitivity-prioritized strategy, which, while maximizing recall, came at a steep cost in specificity and clinical reliability.

In contrast, the Linear classifier showed a steady decline in FPs from 41 in Fold 1 down to just 13 in Fold 5, reflecting improved precision across training. This progression aligns with earlier observations of better specificity in the Linear head, suggesting a more conservative and calibrated decision boundary for the negative class.

When analysing FNs, the MLP head demonstrated more favourable performance. It recorded only 1 or 2 FNs in most folds, peaking at 6 in Fold 4. Meanwhile, the Linear head ranged from 4 to 11 FNs across folds, with the highest error count observed in Fold 5. This discrepancy reinforces the earlier conclusion that the Linear classifier, although better at limiting FPs, often failed to detect true positive cases, especially in more challenging splits resulting in lower sensitivity.

Overall, these results clearly illustrate the trade-off between the two classifiers: the MLP classification head sacrifices specificity to achieve stronger recall, while the Linear head offers a more balanced, yet imperfect, approach with better control over false alarms. Neither model achieved optimal clinical reliability, as both yielded high critical error counts across multiple folds. These findings support the need for future improvements through ensemble learning, error-aware loss functions, and advanced threshold tuning to mitigate both FP and FN rates simultaneously.

Folds	Linear TP	MLP(Youden's Index)TP	MLP (Sensitivity First) TP	Real TP
1	13	13	14	14
2	12	13	13	14
3	13	12	14	14
4	9	10	13	14
5	8	7	14	14

**Table 5.21:** Maximum Number of True Positives Achieved Across 50 Epochs per Fold Using Linear and both MLP Classification Heads Compared to Actual Negative Samples

Folds	Linear TN	MLP(Youden's Index) TN	MLP (Sensitivity First) TN	Real TN
1	29	51	16	51
2	38	50	32	51
3	38	50	13	51
4	45	50	35	51
5	47	50	20	51

**Table 5.22:** Maximum Number of True Negatives Achieved Across 50 Epochs per Fold Using Linear and both MLP Classification Heads Compared to Actual Negative Samples



Tables 5.21 and 5.22 present the maximum number of True Positives (TPs) and True Negatives (TNs), respectively, achieved across five cross-validation folds for three classification heads: the Linear Head, the MLP with Youden’s Index and Block 11 unfrozen, and the MLP with Sensitivity-First Strategy. Each fold includes 14 actual positive cases and 51 negative cases, serving as a consistent reference for evaluating detection capabilities.

### **True Positives (Table 1):**

The MLP (Sensitivity-First) classifier achieved a perfect recall (14/14 TPs) in all five folds, outperforming both the Linear and the Youden’s-based MLP in every split. This demonstrates the effectiveness of explicitly prioritizing recall during threshold selection, as the sensitivity-constrained strategy ensures no missed cancer cases, which is critical in medical diagnostics where false negatives carry severe clinical consequences.

The Linear Head and Youden’s MLP showed more variable performance. The Linear Head reached 13 TPs in three folds but dropped to 9 and 8 in Folds 4 and 5. The Youden’s MLP matched the Linear Head in Fold 1, outperformed it in Fold 2, but underperformed in Folds 3 and 5. These discrepancies highlight that while Youden’s Index seeks to balance sensitivity and specificity, it can result in compromised recall when negative cases dominate the optimization.

### **Total TP Comparison Across Folds:**

- Linear Head:  $13 + 12 + 13 + 9 + 8 = 55$  TPs
- MLP (Youden’s):  $13 + 13 + 12 + 10 + 7 = 55$  TPs
- MLP (Sensitivity-First):  $14 + 13 + 14 + 13 + 14 = 68$  TPs

Only the Sensitivity-First MLP achieved near-perfect cumulative recall of 97.1%, clearly surpassing the 78.5% of the other two heads.

### **True Negatives (Table 2):**

The opposite trend is observed in Table 2. The Linear Head performed increasingly well across folds, culminating in 47/51 TNs in Fold 5. The Youden’s MLP showed remarkably stable performance, identifying 50 or 51 TNs in all folds, reflecting its optimization toward specificity while maintaining reasonable sensitivity.

In contrast, the Sensitivity-First MLP sacrificed specificity for recall. TN counts were consistently lower, with only 13 correctly predicted in Fold 3 and 16 in Fold 1. This trade-off is expected given the classifier’s design constraint to ensure sensitivity  $\geq 0.90$ , but it illustrates a notable rise in false positives, which includes a clinical concern for over-diagnosis and patient anxiety.

#### **Total TN Comparison Across Folds:**

- Linear Head:  $29 + 38 + 38 + 45 + 47 = 197$  TNs
- MLP (Youden’s):  $51 + 50 + 50 + 50 + 50 = 251$  TNs
- MLP (Sensitivity-First):  $16 + 32 + 13 + 35 + 20 = 116$  TNs

This sharp contrast reaffirms the balance Youden’s MLP strikes between recall and precision. It vastly outperforms the Sensitivity-First MLP in specificity, while the latter clearly dominates in pure cancer detection.

#### **Conclusion:**

Together, the two tables reveal that the MLP (Sensitivity-First) model is highly effective for recall-centric tasks, critical in scenarios where missing a cancer case is unacceptable. On the other hand, the MLP with Youden’s Index provides the most balanced generalization, with strong specificity and moderate sensitivity. The Linear Head offers a middle ground but is less consistent in both dimensions, particularly struggling in more challenging folds. The choice between these classifiers thus hinges on whether sensitivity or overall balance is prioritized in deployment.

## **5.6 Results and Discussion**

### **5.6.1 Generalization and Model Behaviour**

The comparative evaluation of the linear and MLP classification heads revealed important distinctions in generalization behavior across the five cross-validation folds. All models were trained on the same DINOv2-based frozen feature representations and evaluated under identical training and data split conditions. However, their outcomes particularly regarding sensitivity, specificity, and prediction errors, highlighted the impact of architectural complexity and training strategy.

The MLP head using the Sensitivity-First strategy consistently achieved perfect or near-perfect recall in every fold, identifying nearly all cancer cases (true positives) across all test sets. This indicates that when prioritizing sensitivity in threshold selection (with a hard constraint of  $\geq 0.90$ ), the MLP head becomes highly effective in minimizing false negatives. This strategy is critically important in medical imaging tasks, where failing to identify a diseased patient carries the most severe consequences. However, this improvement came at a substantial cost to specificity, as the model also generated a high number of false positives, particularly evident in its true negative counts, which were significantly lower than the other classifiers in every fold.

In contrast, the MLP head using Youden's Index and block 11 unfrozen offered a more balanced approach. It achieved relatively strong performance across both sensitivity and specificity, maintaining a high number of true negatives (50+ in most folds) while still achieving a competitive number of true positives. This model demonstrated the strongest generalization capacity among all three heads, as it maintained stability across folds without extreme trade-offs. Its F1-scores and overall test losses reflected consistent decision-making and reliable classification of both healthy and unhealthy cases.

The linear classification head, while simpler and faster to train, showed the weakest generalization overall. It demonstrated highly variable behavior across folds, struggling particularly with low specificity and moderate recall in later folds. Although its average test loss was lower, indicating more confident predictions, the model often produced confidently wrong outputs, especially in classifying healthy samples. Its high number of false positives in early folds and a total of only 197 true negatives (compared to 251 by the Youden MLP and 116 by the Sensitivity-First MLP) suggest that its simplified decision surface could not adequately capture the complex boundary between healthy and unhealthy representations.

In summary, the results suggest that:

- The MLP Sensitivity-First head excels in maximizing recall, making it ideal in scenarios where minimizing false negatives is the highest priority.
- The MLP Youden's head provides the most balanced generalization, maintaining both strong specificity and good recall across folds.

- The Linear head, while occasionally competitive in sensitivity, suffered from higher instability, lower specificity, and greater fold-to-fold variability, indicating that its expressiveness was insufficient for the complexity of the task.

These findings reinforce the importance of selecting classifier architectures and evaluation strategies that align with clinical priorities, whether it's maximizing detection (sensitivity), balancing decisions (F1), or limiting false alarms (specificity).

### 5.6.2 Sensitivity vs Specificity Trade Off

In breast cancer detection, the interplay between sensitivity, which is the ability to correctly identify unhealthy cases and specificity, which is the correct classification of healthy individuals is vital. A high sensitivity ensures that cancerous cases are rarely missed, minimizing false negatives, while high specificity prevents false positives that could lead to unnecessary anxiety, tests, and treatments. As such, designing classifiers that balance both metrics is essential for real-world clinical deployment.

Among the three models evaluated, the MLP classifier using the Sensitivity-First strategy was explicitly optimized to prioritize sensitivity. This model achieved perfect recall (1.000) in three out of five folds and 0.9286 in the other two, resulting in the highest average sensitivity (0.9714) across all models. It successfully captured all or nearly all cancer cases across folds, fulfilling its intended goal of minimizing false negatives.

However, this came at a significant cost to specificity. The same model produced very low true negative counts in multiple folds, with specificity dropping as low as 0.2549 in Fold 3 and averaging only 0.4549 overall. This indicates that while the model was extremely cautious in flagging potentially unhealthy patients, it did so at the expense of incorrectly labeling a substantial number of healthy individuals as sick, raising the false positive count dramatically. In clinical practice, this would lead to increased patient stress, redundant diagnostic procedures, and unnecessary medical costs.

In contrast, the MLP model using Youden's Index achieved a more balanced trade-off. Its average sensitivity (0.7857) was still high and its specificity (0.9726) remained very strong, second only to its Sensitivity-First counterpart. This model's behavior demonstrates the value of threshold optimization strategies like Youden's J statistic,

which aim to balance both metrics, providing robustness without leaning too heavily on one side of the trade-off.

The Linear classifier, while exhibiting some competitive results in sensitivity during earlier folds, struggled in the later ones. Its sensitivity fell as low as 0.5714 in Fold 5, and its specificity ranged widely, from 0.5686 to 0.9216. This inconsistency suggests that the model lacked the flexibility to adapt to challenging distributions, sometimes misclassifying healthy patients while failing to detect actual positives. The linear model was unable to reliably manage the sensitivity-specificity trade-off, often favoring one at the cost of the other.

In summary, the MLP with Sensitivity-First strategy maximized recall with unparalleled consistency, but this advantage came with a heavy sacrifice in specificity, making it less practical in situations where false positives must be limited. The MLP with Youden's Index struck a more effective balance, making it the most reliable overall in handling the clinical trade-off. Meanwhile, the Linear model's performance was less stable and often failed to manage the balance appropriately. These findings highlight the need to carefully select training strategies and decision thresholds based on the clinical objectives of the application, whether that be maximum recall, diagnostic balance, or precision-focused screening.

### **5.6.3 Per-Fold Performance Instability**

While average performance metrics provide valuable high-level insights, examining fold-level variability reveals critical differences in stability, robustness, and generalization. In this study, the three evaluated models, Linear, MLP with Youden's Index and MLP with Sensitivity-First Strategy, displayed varying degrees of per-fold instability, particularly in Folds 1, 4, and 5, which emerged as the most challenging splits.

In Fold 1, the Linear classifier achieved 13 out of 14 TPs, resulting in high recall (sensitivity = 0.9286) but only 29 out of 51 TNs, corresponding to a specificity of just 0.5686. This imbalance highlights the model's tendency to over-predict the positive class, generating 41 false positives, considered an unacceptably high number in clinical contexts. By contrast, the MLP with Youden's Index demonstrated flawless specificity, correctly identifying all 51 TNs, while still matching the linear head in true positives

(13/14). Most notably, the MLP with Sensitivity-First strategy achieved the highest recall with 14 out of 14 TPs, but at the extreme cost of specificity, only 16 TNs, yielding 35 false positives. This fold underscores the sensitivity-specificity trade-off: the Sensitivity-First MLP maximized recall but sacrificed diagnostic precision.

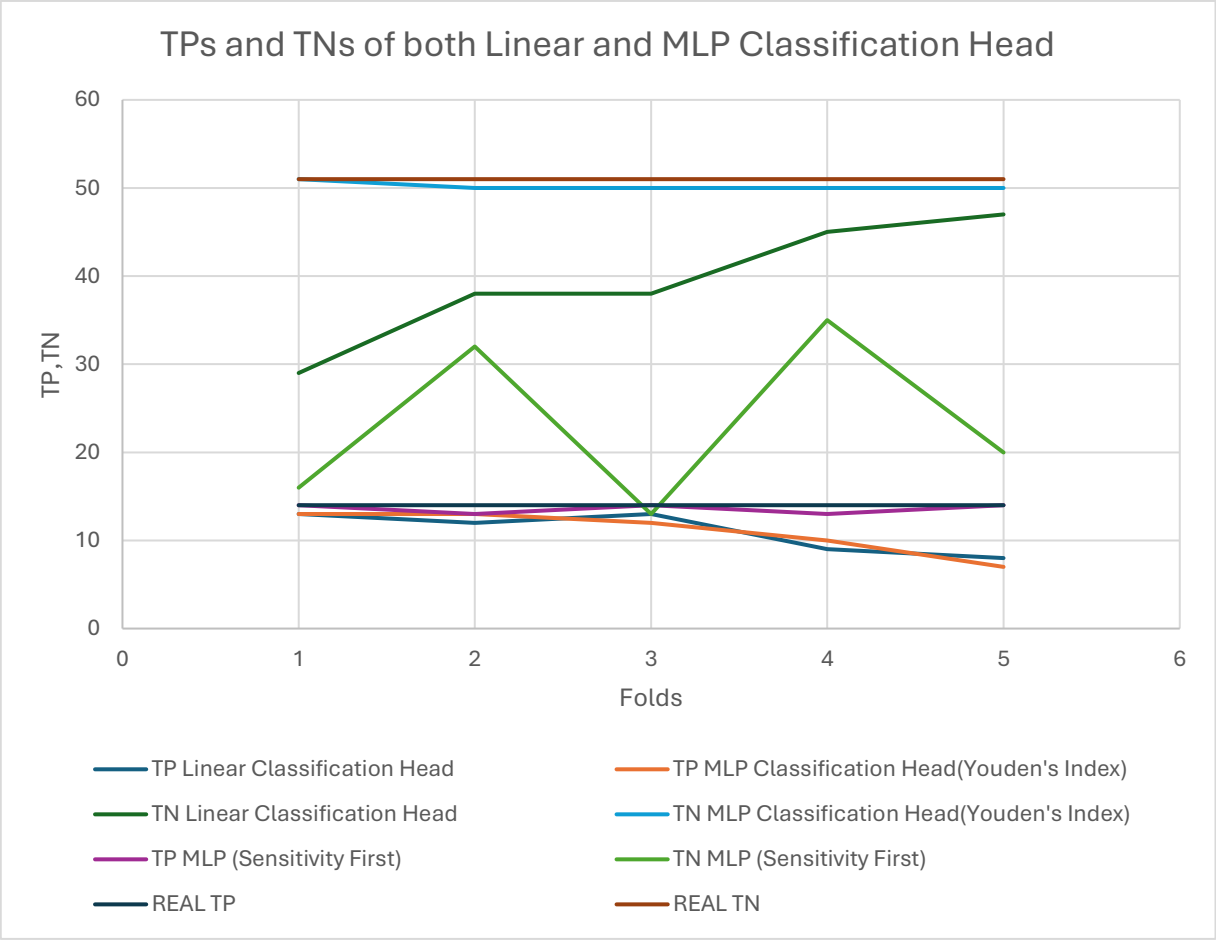
In Fold 5, all models underperformed, but to different extents. The Linear head detected only 8 TPs, and MLP with Youden's Index detected 7, marking their lowest recalls. In contrast, the MLP with Sensitivity-First strategy recovered all 14 TPs, achieving perfect recall, but once again misclassified a large number of healthy cases, only 20 TNs out of 51, reflecting a very low specificity. This pattern exemplifies how extreme sensitivity prioritization can destabilize model precision under fold-specific challenges, and further highlights how the Linear model's sensitivity dropped alongside its specificity, confirming that its performance decays more symmetrically.

Fold 4 further illustrates this contrast. The Linear classifier's sensitivity dropped to 0.6429 (9 TPs), while the MLP with Youden's Index achieved a modest 0.7143 (10 TPs). Meanwhile, the Sensitivity-First MLP recovered 13 TPs, again prioritizing recall. However, specificity dropped significantly to 0.6863 for the Sensitivity-First model, still better than some folds, but notably worse than the 0.9804–1.0000 range seen previously by the MLP with Youden's Index.

Overall, these fold-specific comparisons demonstrate how averages can mask instability, and why clinical deployment requires not just high performance, but consistent performance across data splits. The Linear model frequently exhibited coupled drops in sensitivity and specificity, revealing vulnerability to shifts in data distribution. The MLP with Youden's Index, although not perfect, offered a more stable balance between recall and precision across folds. The MLP with Sensitivity-First strategy, while maximizing recall as designed, introduced extreme variance in specificity, making it less viable in contexts where false positives must be controlled.

In summary, per-fold variability reveals critical trade-offs in model behavior. For high-stakes clinical environments, the choice of model and training strategy must account for these inconsistencies. Among the three, the MLP with Youden's Index emerged as the most stable and balanced option, while the Sensitivity-First MLP achieved best-in-class

recall but with dramatic loss in precision, and the Linear head showed the most unpredictable fluctuations overall.



**Figure 5.51:** Comparison of Maximum True Positives and True Negatives per Fold for Linear and both MLP Classification Heads

#### 5.6.4 Clinical Implications

The comparative evaluation of the Linear, MLP with Youden’s Index, and MLP with Sensitivity-First classification heads brings to light several key clinical insights for the potential deployment of such models in breast cancer screening systems. One of the most critical considerations remains sensitivity, as it governs the model’s ability to detect true cancer cases. Missed detections (FNs) can have severe consequences in diagnostic workflows, delaying treatment or leading to overlooked pathology.

Across folds, the MLP with Sensitivity-First strategy emerged as the most effective model for maximizing recall, achieving perfect sensitivity (14/14 TPs) in multiple folds and outperforming both the Linear and Youden’s-based MLP heads in this regard. From

a clinical safety standpoint, this behaviour aligns with the conservative preference in medicine to prioritize caution, ensuring that no true cases are missed, even at the expense of increased false positives. However, this gain in recall came with a significant trade-off. The model misclassified a large number of healthy patients, leading to very low specificity, with true negatives as low as 13 out of 51 in some folds. This result introduces a new challenge, overdiagnosis, unnecessary emotional stress, and the cost of excessive follow-up procedures.

In contrast, the MLP classifier using Youden's Index offered a more clinically balanced performance. It achieved a relatively strong sensitivity, while maintaining near-perfect specificity across all folds. This translates to fewer false positives and a lower risk of unnecessary medical escalation, making it the most practical option in scenarios where both early detection and precision matter. Its stability in F1-score, specificity, and generalization across folds makes it a clinically reliable candidate for deployment in real-world screening programs.

The Linear classification head, though occasionally competent, particularly in matching the MLPs in recall, demonstrated greater variability across folds and was especially vulnerable to producing confidently incorrect predictions. It struggled with specificity in multiple folds, resulting in high false positive rates. This instability, coupled with **lower** average F1-score, makes the linear model less trustworthy in high-stakes environments, especially given its inability to handle subtle variations in complex thermographic patterns.

From a clinical adoption perspective, these findings reinforce that architectural decisions, even modest ones, can yield meaningful gains in safety and trust. Moving from a simple linear layer to a well-tuned MLP architecture can enable systems to balance sensitivity and specificity more effectively. Moreover, the choice of training objective (e.g., Youden's J vs. sensitivity-maximizing thresholds) has direct implications on clinical utility and must be selected with the application context in mind.

In summary, the MLP with Sensitivity-First strategy may be suitable in screening-first scenarios where missing a case is unacceptable, even if that means higher false alarms.



However, the MLP with Youden's Index provides a more clinically deployable balance, ensuring that both cancer detection and patient well-being are preserved. The Linear head, while efficient, appears insufficient on its own for deployment, though it may still offer value in ensemble or lightweight settings when paired with stronger models.

### **5.7 Comparative Summary to Teachable Machine Model**

In order to establish a baseline and compare the performance of the custom-trained deep learning models built in this thesis, an additional classification model was developed using Google's Teachable Machine platform<sup>[28]</sup>. Teachable Machine is a user-friendly web-based tool that enables rapid creation of machine learning models without requiring in-depth programming knowledge. It leverages transfer learning techniques by adapting pre-trained neural networks to new tasks with a relatively small dataset, making it suitable for quick prototyping and experimentation.

For this study, the Teachable Machine model was trained to classify anterior images into two categories: Healthy *and* Unhealthy. The training dataset consisted of 145 healthy images and 30 unhealthy images. These images were manually labelled and uploaded to the platform to facilitate model training. The first class includes the Healthy Images, and the second class included the Unhealthy Images.

After training, the model's performance was evaluated using the remaining unseen portion of the dataset, which was not used during the training phase. This unseen test set contained 107 healthy images and 41 unhealthy images. The evaluation on this separate dataset allowed an unbiased assessment of the model's generalization capability in classifying new, previously unseen images.

By comparing the results of the Teachable Machine model with those of the custom DINOv2-based model, a comprehensive understanding of the strengths and limitations of each approach was obtained. This comparative analysis highlights differences in accuracy, robustness, and applicability to medical image classification tasks.

To optimize the model's performance, four experiments were conducted by varying key hyperparameters while keeping the batch size fixed at 32. The experiments differed primarily in the number of training epochs and learning rates used:

1. Epochs = 50, batch size = 32, learning rate = 0.0012
2. Epochs = 50, batch size = 32, learning rate = 0.0001
3. Epochs = 50, batch size = 32, learning rate = 0.0005
4. Epochs = 100, batch size = 32, learning rate = 0.0005

These configurations were chosen to investigate how training duration and learning speed affect the model's accuracy and generalization. It should be noted that the unseen test dataset for all experiments consisted of 107 healthy images and 41 unhealthy images, as described previously. The following table summarizes the results obtained from each of these experimental setups.

Configuration	Accuracy	Sensitivity	Specificity	TP	TN	FP	FN
1	0.7230	0.000	1.000	0	107	0	41
2	0.7230	0.000	1.000	0	107	0	41
3	0.7162	0.000	0.9907	0	106	1	41
4	0.7230	0.000	1.000	0	107	0	41

**Table 5.25:** Results obtained from the four experiments conducted using Teachable Machine.

The resulting metrics, summarized in the table, reveal a consistent pattern across all experiments: the sensitivity is 0.000 or near zero, indicating the model fails to correctly identify any of the positive class (Unhealthy) images, while specificity remains very high (close to or at 1.000), showing that the model predominantly predicts the negative class (Healthy) correctly. This is further supported by the confusion matrix counts where true positives (TP) are zero or negligible, and true negatives (TN) equal the full count of healthy samples, with all unhealthy samples being false negatives (FN).

This behaviour strongly suggests the model is heavily biased toward predicting the Healthy class. Several factors may contribute to this imbalance. First, the training dataset is skewed, with 107 Healthy images but only 41 Unhealthy images, which likely causes the model to learn the dominant class more effectively. Additionally, the learning rates in experiments 1 and 2 (0.0012 and 0.0001) might be either too high or too low to effectively adjust weights for the minority class. The third experiment, with a learning rate of 0.0005, shows a slight change with one False Positive noted, but still shows inability to detect positive cases. The fourth experiment doubles the number of

epochs to 100 with the same learning rate (0.0005), but it does not improve sensitivity, suggesting that simply increasing training duration without addressing class imbalance or model capacity does not help.

Overall, these results highlight the need for strategies such as class weighting, data augmentation for the minority class, or using more balanced datasets to improve detection of the Unhealthy class. Additionally, tuning other hyperparameters or exploring alternative model architectures might be necessary to overcome this bias and increase sensitivity without sacrificing specificity.

The evaluation of the three custom classifiers implemented in this thesis, which are Linear Head, MLP with Sensitivity Optimization, and MLP using Youden's Index, demonstrate a clear improvement over the baseline performance of the Teachable Machine model. All classifiers were evaluated against the same unseen dataset comprising 14 truly Unhealthy (positive) cases and 51 truly Healthy (negative) cases.

The Linear Head classifier, which applies a simple linear layer on top of the frozen DINOv2 embeddings, shows relatively balanced performance. Across multiple configurations, it achieved true positive (TP) counts between 8 and 13, and true negative (TN) counts between 29 and 47. Although its sensitivity is slightly lower than optimal in some settings, it demonstrates that even a simple classifier can leverage the rich features from DINOv2 effectively.

The MLP trained with a Sensitivity-First strategy further improved the detection of Unhealthy cases, reaching the maximum TP of 14 (perfect sensitivity) in several configurations. However, this came at the cost of lower TN counts, ranging from 13 to 35, meaning more false positives were introduced. This strategy is particularly valuable in medical or health-related applications where detecting all positives is crucial, and a few false alarms are acceptable.

The most balanced and robust results came from the MLP trained using Youden's Index, a metric that simultaneously considers both sensitivity and specificity. This classifier consistently achieved TP values of 13 or 14, maintaining nearly perfect sensitivity, while also improving TN values. This balance indicates that the model not

only detects Unhealthy cases well but also reduces false alarms, making it suitable for practical applications.

In stark contrast, the Teachable Machine model, although simple and accessible, failed to generalize well to the unseen dataset. As previously discussed, across all its hyperparameter configurations, it predicted all images as Healthy, leading to zero true positives ( $TP = 0$ ) and all 41 Unhealthy cases being misclassified as false negatives. While its specificity was 1.000 (no false positives), its sensitivity was consistently 0.000, rendering it unusable for identifying Unhealthy cases, precisely the task it should prioritize in a diagnostic setting.

This direct comparison highlights the limitations of using general-purpose tools like Teachable Machine for complex, imbalanced classification problems. Despite its convenience, it lacks the capacity for fine-tuning, deeper optimization, and targeted metric improvements.

In conclusion, the use of Vision Transformers, and specifically DINOv2 embeddings, proves significantly more effective for this classification task. The rich feature representations extracted from DINOv2 allow even simple classifiers to outperform Teachable Machine substantially. The fine-tuned MLP with Youden's Index offers the best balance between sensitivity and specificity, making it the most reliable model among those evaluated. These results strongly support the integration of advanced transformer-based models and custom training strategies over black-box AutoML platforms for sensitive and high-stakes applications.

### **5.8 Further Validation Using a Balanced Dataset**

Following the findings in Section 5.6, it was observed that the MLP classification head with partial fine-tuning, specifically the configuration using Youden's Index for threshold selection and unfreezing only the final transformer block (Block 11), demonstrated the most clinically balanced behaviour. This model consistently maintained a strong equilibrium between sensitivity and specificity, making it particularly suitable for real-world diagnostic use where both false negatives and false positives must be minimized.

To further evaluate the robustness and generalization of this architecture under conditions free of class imbalance, an additional experiment was conducted using a balanced subset of the original dataset.

### **Balanced Dataset Construction**

From the full set of 323 thermal breast images, a subset of:

- 71 healthy samples
- 71 unhealthy samples

was selected to form a new dataset with perfectly balanced class distribution (1:1). The images were randomly selected but stratified to ensure diversity and representativeness within both classes. This setting eliminates the bias introduced by the original class imbalance (252 healthy vs. 71 unhealthy), allowing for a more controlled evaluation of the model's learning capacity.

### **Experimental Setup**

The same evaluation strategy was applied as in earlier sections, with the following characteristics:

- Stratified 5-Fold Cross-Validation
- Classifier: MLP head with Block 11 unfrozen (partial fine-tuning)
- Thresholding: Youden's J statistic used to select optimal thresholds in each fold
- Training duration: 50 epochs per fold, selected based on prior empirical validation to ensure stability and sufficient model convergence
- All other hyperparameters remained identical to the original implementation in Section 4.7.2
- Metrics tracked: Accuracy, Sensitivity, Specificity, F1-score, TP, TN, FP, FN, and BCE Loss

This setup was used to verify whether the strong performance observed in the imbalanced setting persists under balanced conditions and whether the MLP architecture with partial fine-tuning remains consistent in its predictive capacity.

## Results

The evaluation metrics obtained across the five folds are summarized in Table 5.26. The table provides detailed insights into how the model performs when class representation is equal, thereby isolating the effect of architecture and training strategy from data imbalance.

Fold	Training Accuracy	Training Sensitivity	Training Specificity	Training Loss	Validation Accuracy	Validation Sensitivity	Validation Specificity	Validation Loss	Test Accuracy	Test Sensitivity	Test Specificity	Test Loss	F1-Score
1	0.8556	0.8667	1	2.2972	0.7391	1	1	0.7279	0.7931	0.9286	1	0.6926	0.8125
2	0.8778	0.9111	1	1.6496	0.6522	0.8333	0.7273	0.705	0.8621	1	0.9333	0.5372	0.8571
3	0.9	0.9565	1	1.3868	0.913	0.9091	1	0.4423	0.8621	1	0.9333	0.4806	0.8571
4	0.8791	0.913	1	1.4177	0.7727	1	0.6364	0.6083	0.8276	1	0.8667	0.5288	0.8276
5	0.9011	0.9565	1	1.2627	0.8636	1	0.7273	0.6461	0.8276	1	0.9333	0.6609	0.8

**Table 5.26:** Performance Metrics of MLP (with partial fine-tuning) results, using a balanced dataset.

As the results of Table 5.26 indicate, the performance of the MLP classifier with partial fine-tuning and Youden's Index on the balanced dataset (71 healthy, 71 unhealthy) demonstrates strong generalization and clinical potential.

Sensitivity remained consistently high across all folds, with three folds achieving perfect recall (1.000) and the lowest value being 0.9286. This shows the model's excellent ability to detect all or nearly all cancer cases, aligning well with clinical goals of minimizing false negatives.

Specificity also remained remarkably stable, with values of 1.000 in most folds and only minor variation (e.g., 0.9333, 0.8667), indicating minimal false positives and strong diagnostic precision. This consistency addresses the key limitation of the sensitivity-first strategy used previously, where specificity suffered greatly.

Test accuracy ranged from 0.7931 to 0.8621, which is acceptable given the clinical emphasis on recall and specificity. Notably, folds with slightly lower accuracy still maintained high sensitivity and specificity, suggesting that errors were not biased toward one class.

F1-scores were consistently high (0.8 to 0.8571), indicating strong harmonic balance between precision and recall. This reflects effective decision boundaries and a reduced trade-off between detecting sick cases and avoiding over-diagnosis.

Test BCE loss values (ranging from 0.48 to 0.69) are relatively low, suggesting good model calibration and confidence in predictions.

### **5.8.1 Comparative Evaluation: Balanced vs. Unbalanced Dataset**

The evaluation of the MLP classification head using partial fine-tuning and Youden's Index reveals a clear improvement in classification stability and performance when trained on a balanced dataset. Compared to its performance on the original unbalanced dataset, the model trained on the balanced version exhibited more consistent behaviour across folds, particularly in terms of test sensitivity and F1-score.

On the unbalanced dataset, the classifier showed high sensitivity and excellent specificity in several folds, but with notable variability. In particular, the sensitivity exhibited significant fluctuations across folds, and the F1-score, which reflects the trade-off between precision and recall, remained modest indicating an imbalance between correctly detected positive cases and the number of false positives.

In contrast, the model trained on the balanced dataset achieved consistently high sensitivity and specificity across all test folds. This stability translated into stronger F1-scores, suggesting more effective and reliable decision-making. The model's performance on the balanced dataset indicates that class balance played a crucial role in enabling the classifier to distinguish between healthy and unhealthy cases without disproportionately favouring one class.

Moreover, the test loss values were generally lower and more consistent in the balanced setup, implying improved model calibration and confidence in predictions. This contrasts with the unbalanced setup, where test loss values were both higher and more variable, likely reflecting the learning challenges imposed by the class distribution skew.

In summary, balancing the dataset not only mitigated the fold-to-fold fluctuations observed in the unbalanced case but also allowed the MLP classifier to better optimize for both sensitivity and specificity simultaneously which is considered a critical requirement for real-world medical screening applications.

In conclusion, the balanced dataset yielded the most reliable and clinically favourable results. When the MLP classifier with partial fine-tuning was trained on this balanced subset, it consistently achieved high sensitivity and perfect or near-perfect specificity

across all test folds, with superior F1-scores and lower test losses compared to the unbalanced setup. These results indicate that the classifier was better able to distinguish between healthy and unhealthy cases without bias, and with reduced variability. In contrast, the same model trained on the unbalanced dataset demonstrated greater performance instability, particularly in F1-score and sensitivity, due to the disproportionate representation of healthy cases. Therefore, for this thesis, the balanced dataset proved to be the most effective foundation for achieving stable and interpretable model performance, reinforcing the importance of dataset composition in medical AI applications.



# Chapter 6

## Conclusion and Future work

---

6.1 Summary of Findings

6.2 Limitations

6.3 Future Work

---

### 6.1 Summary of Findings

This study demonstrated that leveraging frozen features from the self-supervised DINOv2 ViT-S/14 backbone combined with lightweight classifier heads, specifically linear and multi-layer perceptron (MLP), enables effective binary classification of thermal breast images. The MLP head consistently outperformed the linear model across most evaluation metrics, including accuracy, specificity, and F1-score. Both models reached equal average sensitivity, but the MLP achieved more consistent classification, particularly in correctly identifying healthy cases. This confirms the benefits of non-linear architectures for medical classification tasks using rich transformer-based embeddings.

To further investigate performance limitations observed in the unbalanced dataset (with a 252:71 healthy-to-unhealthy ratio), an additional experiment was conducted using a balanced subset of the data (71 healthy and 71 unhealthy images). This follow-up evaluation revealed that the same MLP classifier with partial fine-tuning of the final transformer block and Youden's Index thresholding, achieved significantly improved performance. Across all folds, the model trained on the balanced dataset exhibited higher and more consistent F1-scores, while also preserving strong sensitivity and achieving near-perfect specificity. This suggests that class imbalance in the original dataset likely contributed to performance variability and excessive false positives or false negatives.

While the evaluation metrics indicate relatively strong performance for a lightweight classification system, particularly in terms of sensitivity and specificity, these results should be interpreted with caution. Despite favourable averages, the number of misclassifications across folds in the original unbalanced setting remains too high to justify clinical deployment. In medical diagnosis, especially for breast cancer, even a small number of false negatives or false positives can result in serious consequences, such as delayed treatment or unnecessary interventions. However, the improved results obtained from the balanced dataset highlight the critical role of dataset composition in achieving clinically reliable outcomes.

In conclusion, while the models demonstrate promise for research and development, particularly with balanced training data, further improvements in generalization, class balance handling, and decision transparency are essential before such systems can be considered viable for real-world clinical integration.

## **6.2 Limitations**

Several limitations of this study should be acknowledged:

- **Dataset Size and Class Imbalance:** The dataset consisted of 323 images, with only 72 labelled as unhealthy. This limited the model's exposure to diverse pathology and contributed to class imbalance, which affects recall and precision performance.
- **Frozen Backbone Usage:** The DINOv2 backbone was used in frozen mode to avoid overfitting on the small dataset. While this preserved generality, it also limited the model's ability to adapt to specific patterns in thermal breast imaging.
- **Lack of External Validation:** The same final test set was used across folds, and no external dataset was used for validation. This restricts the generalizability of the models beyond the current dataset.
- **Interpretability Constraints:** No explainability techniques were used to identify what regions of the image contributed to predictions, limiting transparency, which is an essential factor in medical AI.

- **Binary Classification Only:** The problem was framed as binary classification (healthy vs. unhealthy), whereas real-world diagnosis often requires differentiation between benign, malignant, or pre-cancerous findings.

### **6.3 Future Work**

There are several ways this research could be extended and improved in the future. One of the most important next steps would be to seek access to a larger and more diverse dataset, which would allow for better generalization and more reliable evaluation, especially for identifying cancerous cases. In addition, future work could try fine-tuning the DINOv2 model itself, rather than just using its frozen features, to help it learn patterns that are more specific to breast thermal images. It would also be useful to explore ways to explain model predictions, so that we can understand which parts of the image influenced the decision, something that is especially important in medical applications.

Another direction would be to move beyond binary classification (healthy vs. unhealthy) and instead predict more detailed labels, such as whether a case is benign or malignant, which would make the model more helpful in real diagnostic settings. Improving the training process, for example by trying different types of image transformations or combining the outputs of multiple models, could help make the results more stable and reduce errors. Testing the models on completely unseen data from other sources would also be important to check if they work well in real-world situations. Finally, future research could explore better ways of deciding when to classify an image as positive or negative, to make the trade-off between false positives and false negatives more acceptable in clinical practice.

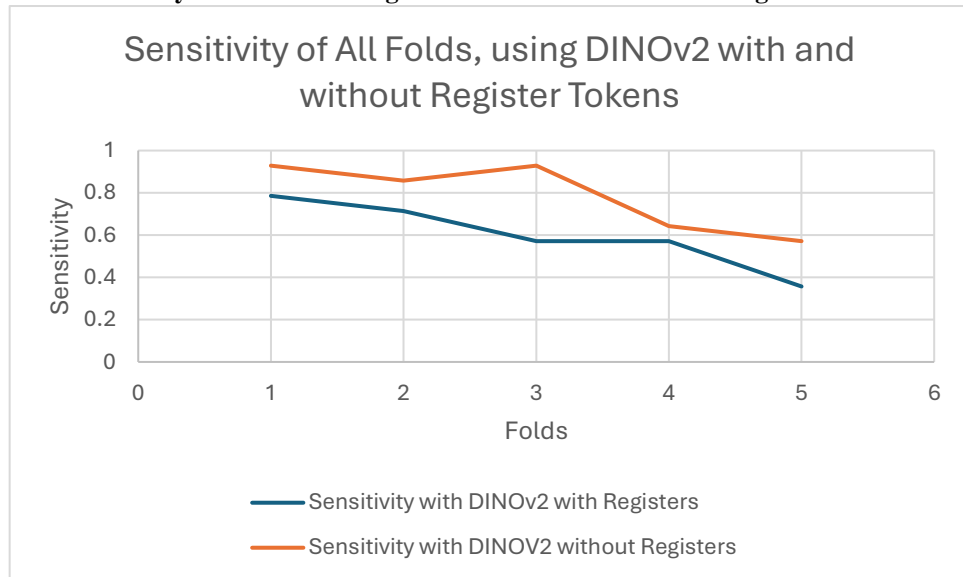
Lastly, an additional potential direction would be to explore alternative self-supervised learning approaches beyond DINOv2. While DINOv2 provided strong pretrained features in this study, future research could compare its performance with other self-supervised frameworks, which may capture different types of visual information. Additionally, investigating other Vision Transformer architectures, could reveal whether different transformer designs offer advantages in the context of breast thermal image classification.

## 7 Appendix

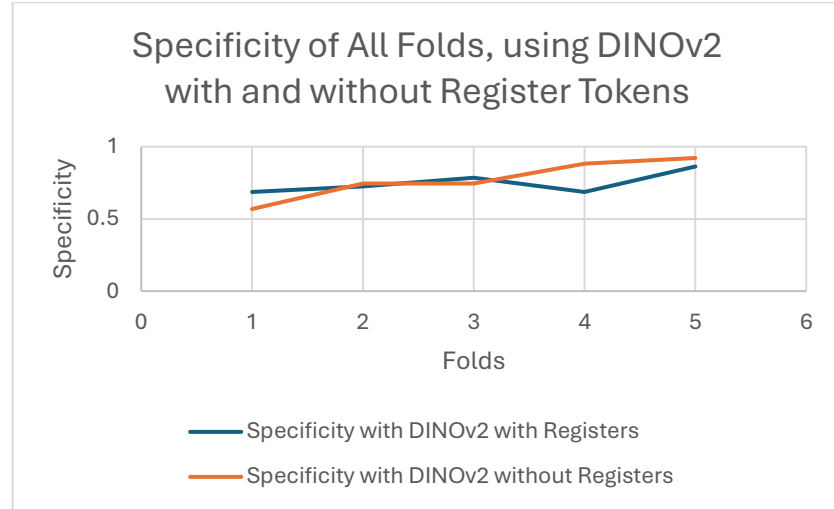
### i. Model Architecture – DINOv2 ViT-s/14 (Printed with print(model))

```
DinoVisionTransformer(
  (patch_embed): PatchEmbed(
    (proj): Conv2d(3, 384, kernel_size=(14, 14), stride=(14, 14))
    (norm): Identity()
  )
  (blocks): ModuleList(
    (0-11): 12 x NestedTensorBlock(
      (norm1): LayerNorm((384,)), eps=1e-06, elementwise_affine=True)
      (attn): MemEffAttention(
        (qkv): Linear(in_features=384, out_features=1152, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=384, out_features=384, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (ls1): LayerScale()
      (drop_path1): Identity()
      (norm2): LayerNorm((384,)), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=384, out_features=1536, bias=True)
        (act): GELU(approximate='none')
        (fc2): Linear(in_features=1536, out_features=384, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
      (ls2): LayerScale()
      (drop_path2): Identity()
    )
  )
  (norm): LayerNorm((384,)), eps=1e-06, elementwise_affine=True)
  (head): Identity()
)
```

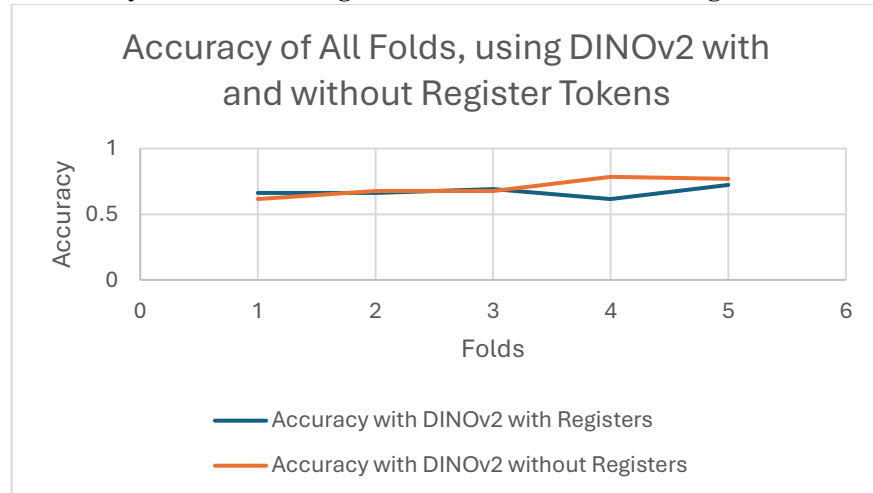
### ii. Sensitivity of All Folds Using DINOv2 With and Without Register Tokens



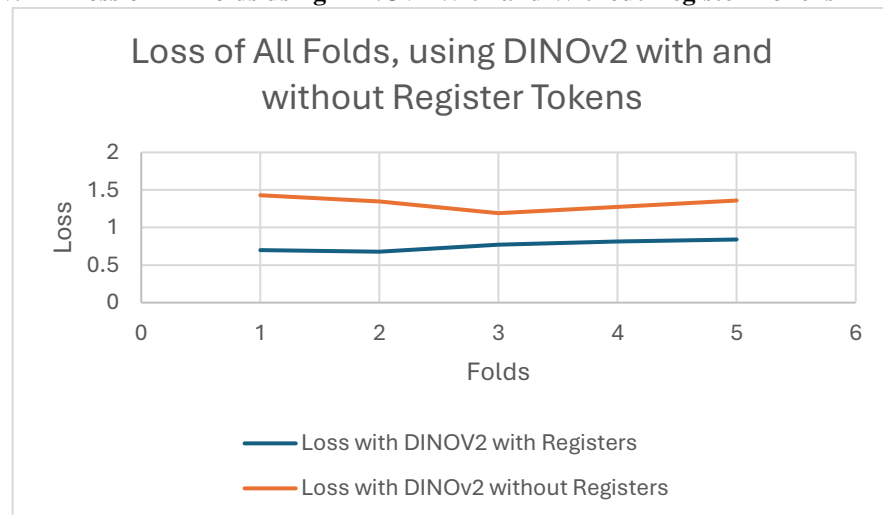
iii. **Specificity of All Folds Using DINOv2 With and Without Register Tokens**



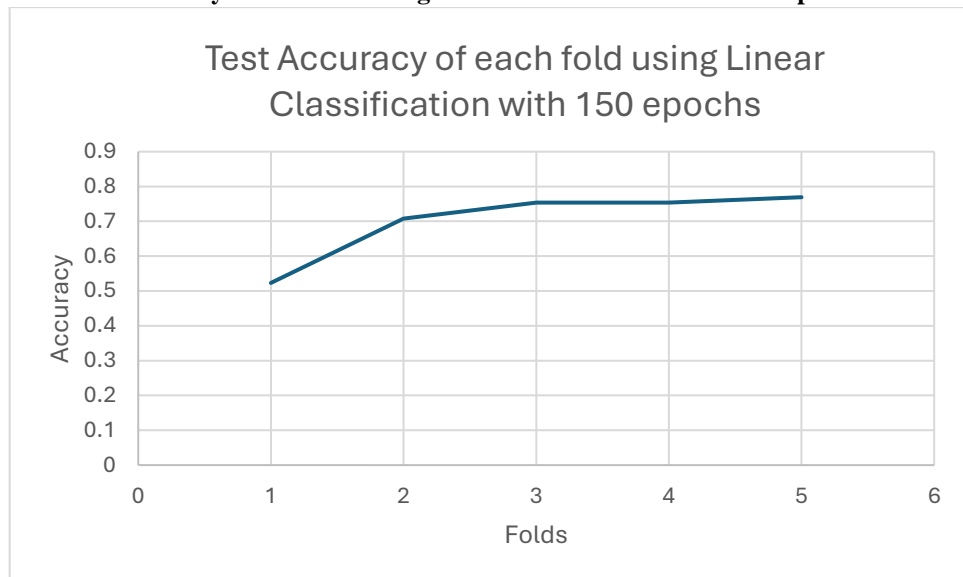
iv. **Accuracy of All Folds Using DINOv2 With and Without Register Tokens**



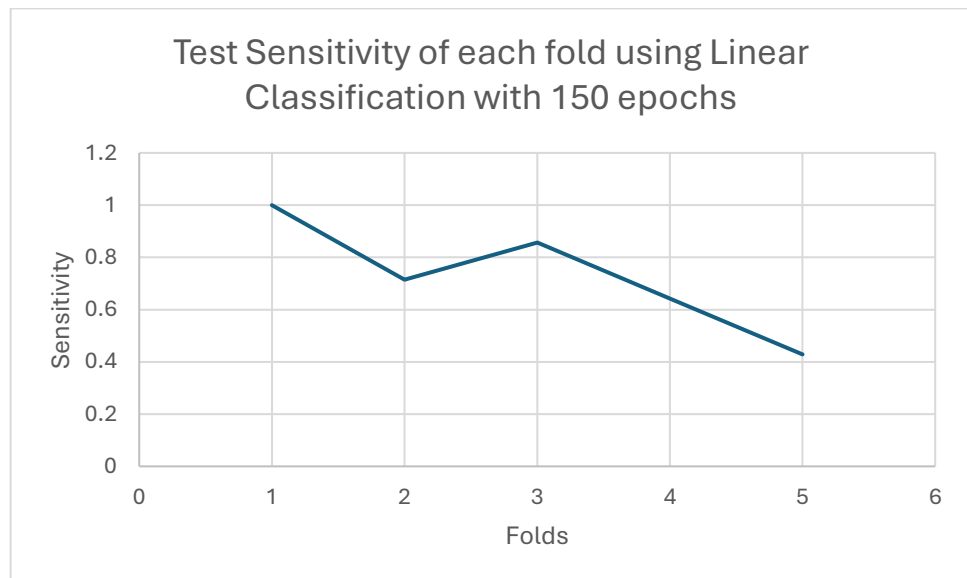
v. **Loss of All Folds using DINOv2 With and Without Register Tokens**



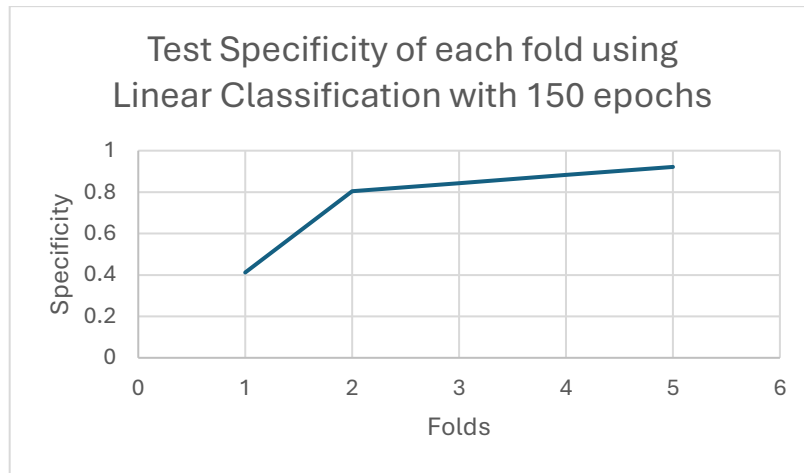
**vi. Accuracy of each fold using Linear Classification with 150 epochs**



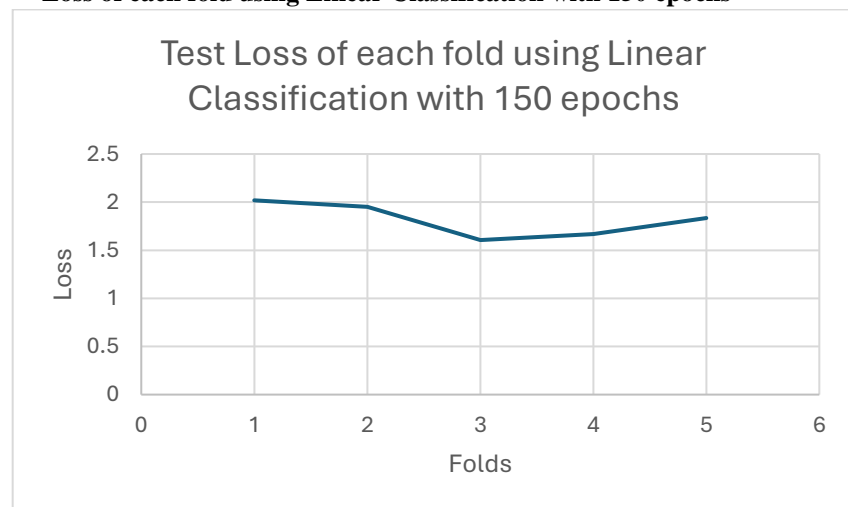
**vii. Sensitivity of each fold using Linear Classification with 150 epochs**



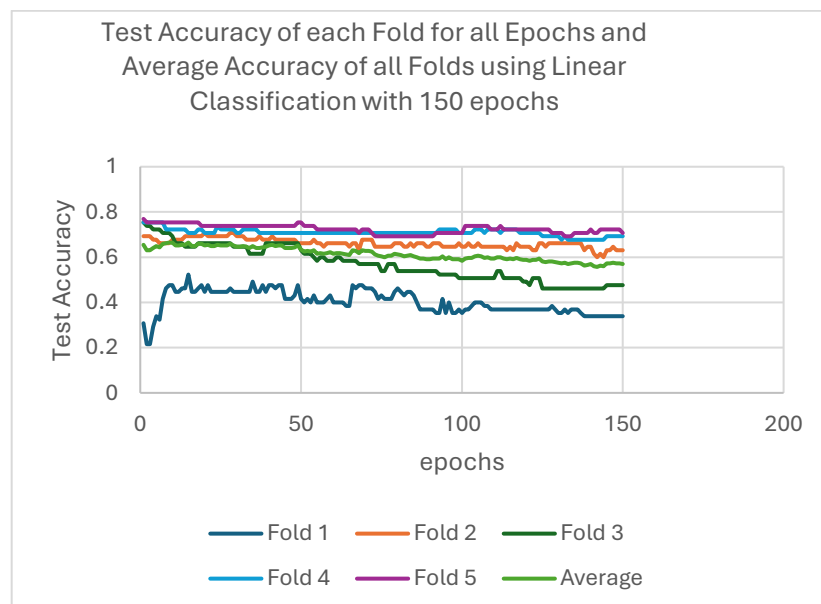
**viii. Specificity of each fold using Linear Classification with 150 epochs**



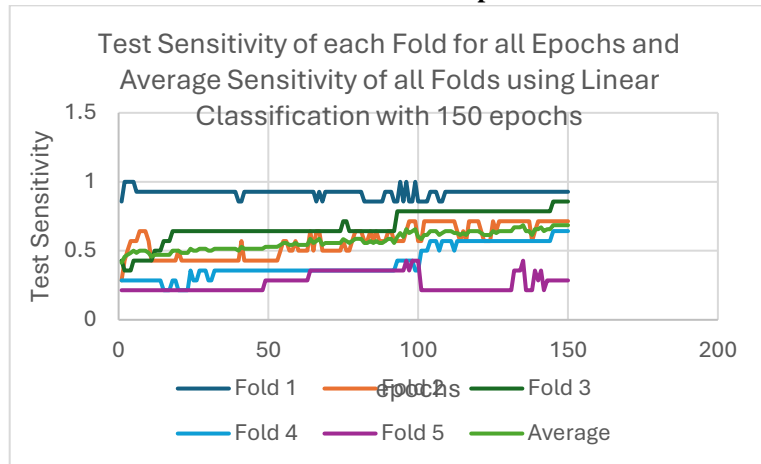
**ix. Loss of each fold using Linear Classification with 150 epochs**



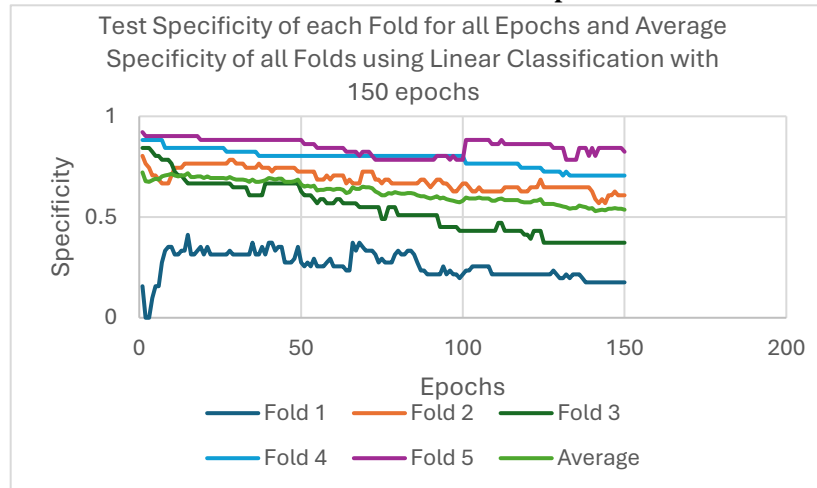
**x. Test Accuracy per epoch and Average Accuracy, across all folds using Linear Classification Head with 150 epochs.**



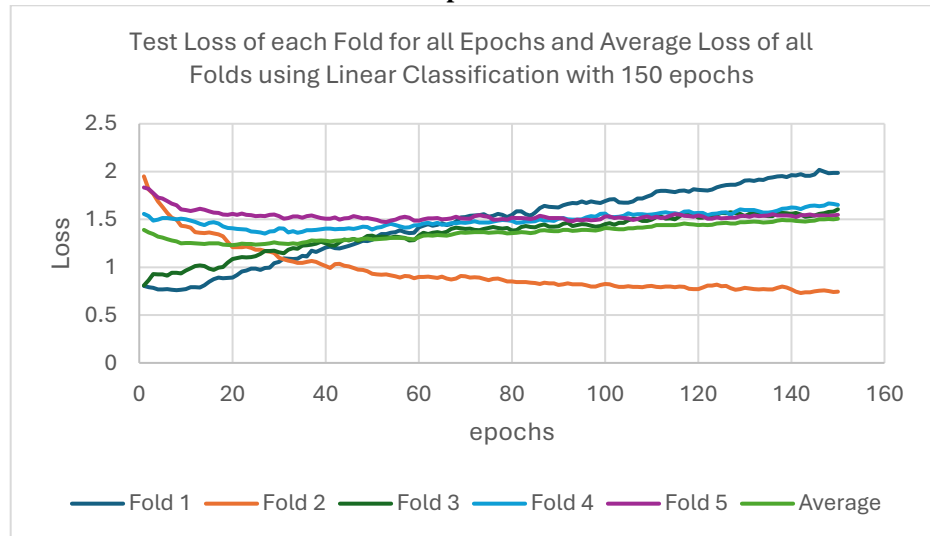
- xi. **Test Sensitivity per epoch and Average Sensitivity, across all folds using Linear Classification Head with 150 epochs.**



- xii. **Test Specificity per epoch and Average Specificity, across all folds using Linear Classification Head with 150 epochs.**

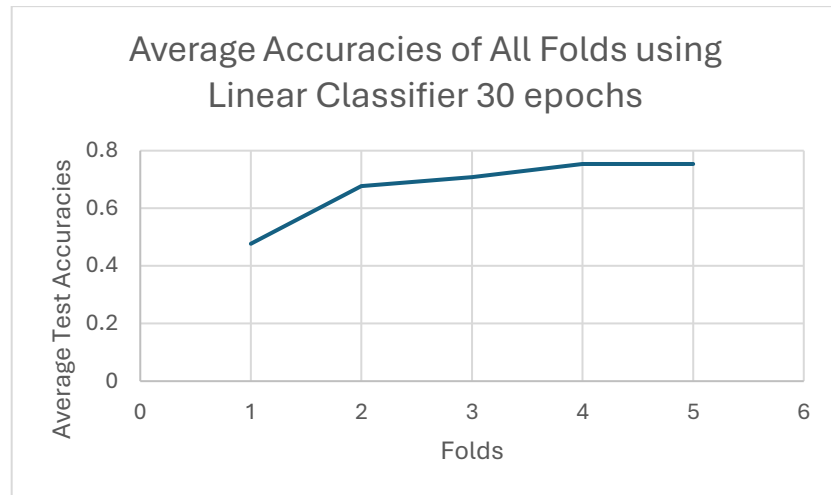


- xiii. **Test Loss per epoch and Average Loss, across all folds using Linear Classification Head with 150 epochs.**

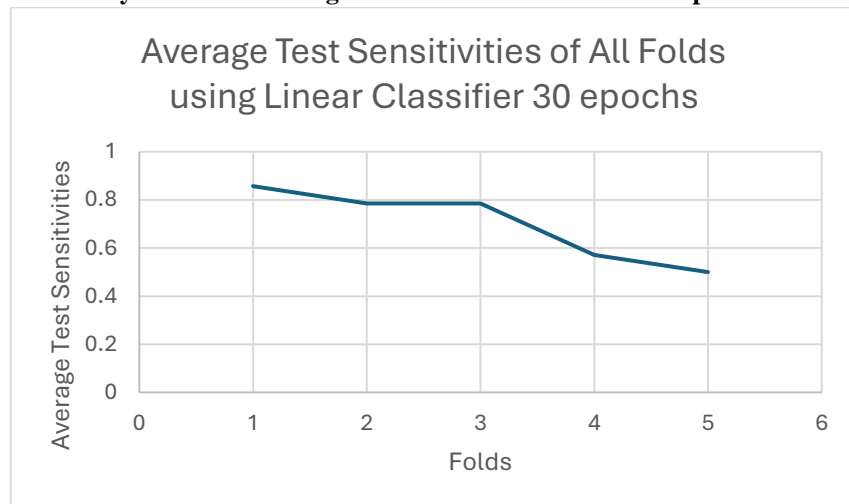


- xiv. **Accuracy of each fold using Linear Classification with 30 epochs**

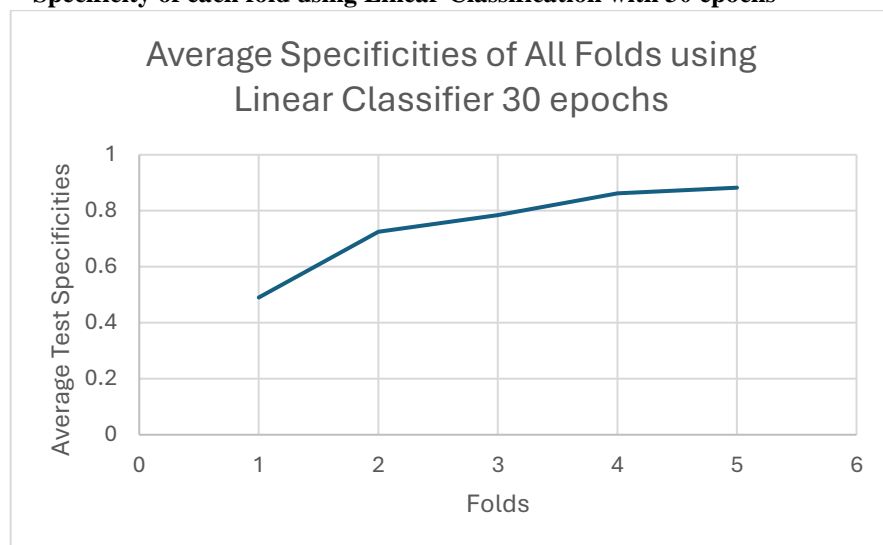




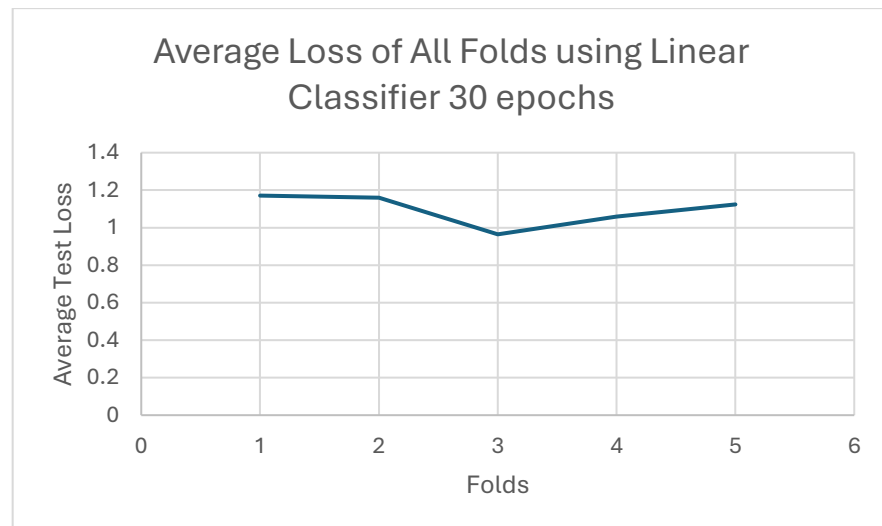
xv. **Sensitivity of each fold using Linear Classification with 30 epochs**



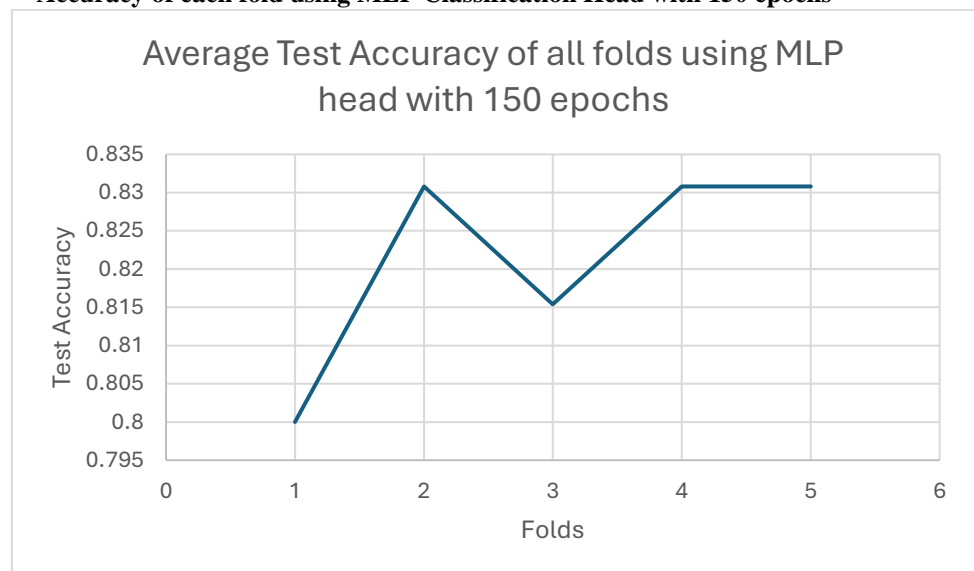
xvi. **Specificity of each fold using Linear Classification with 30 epochs**



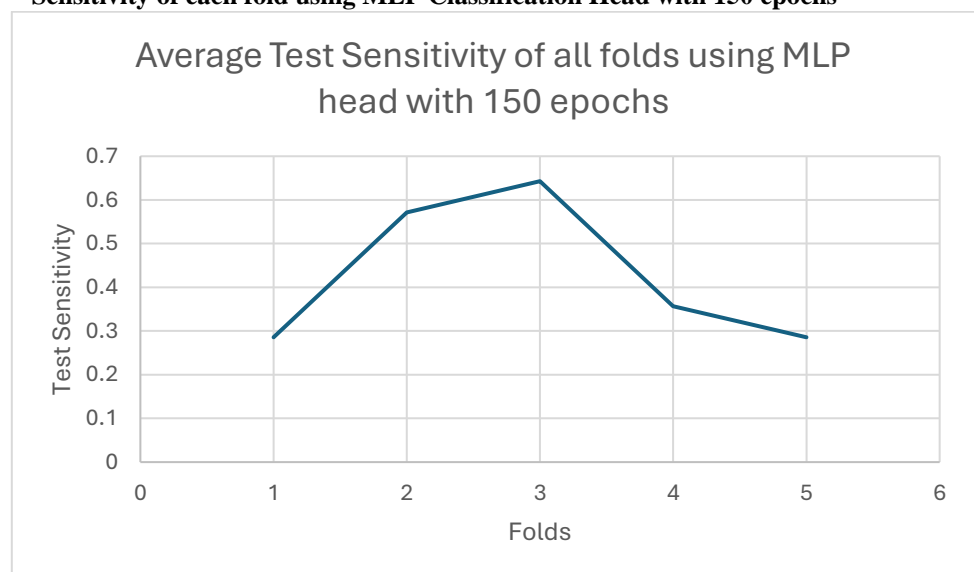
xvii. **Loss of each fold using Linear Classification with 30 epochs**



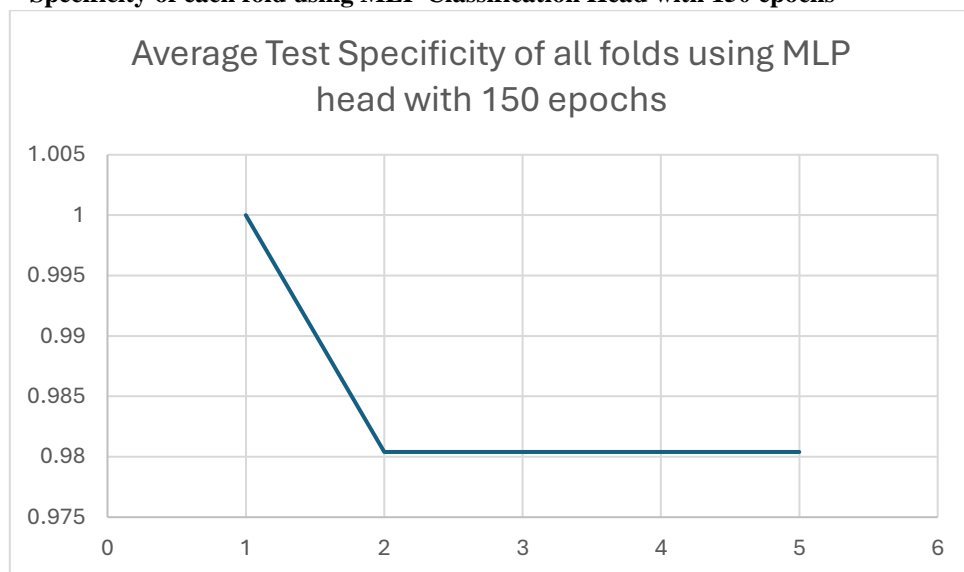
**xviii. Accuracy of each fold using MLP Classification Head with 150 epochs**



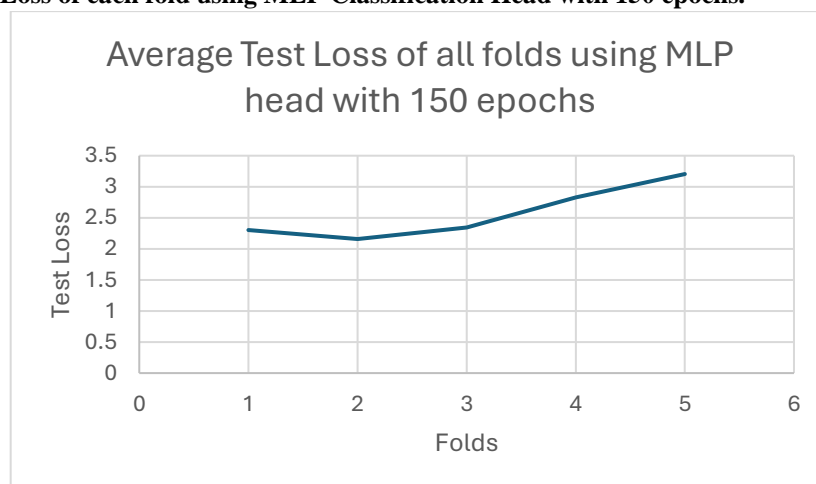
**xix. Sensitivity of each fold using MLP Classification Head with 150 epochs**



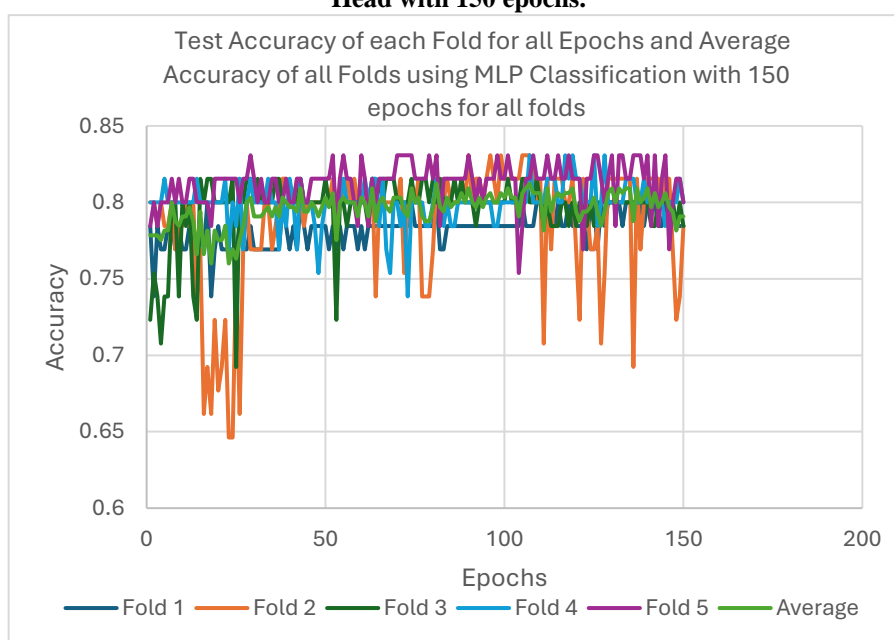
**xx. Specificity of each fold using MLP Classification Head with 150 epochs**



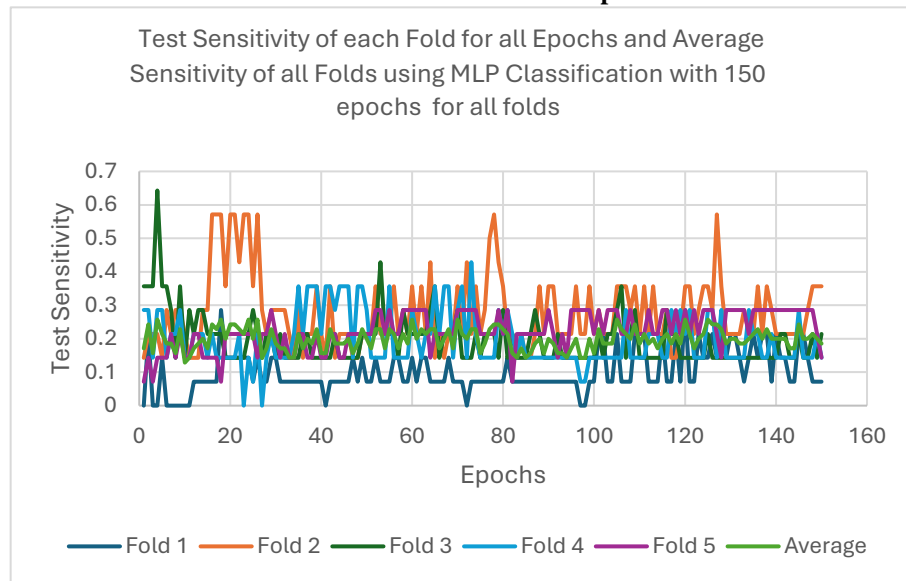
**xxi. Loss of each fold using MLP Classification Head with 150 epochs.**



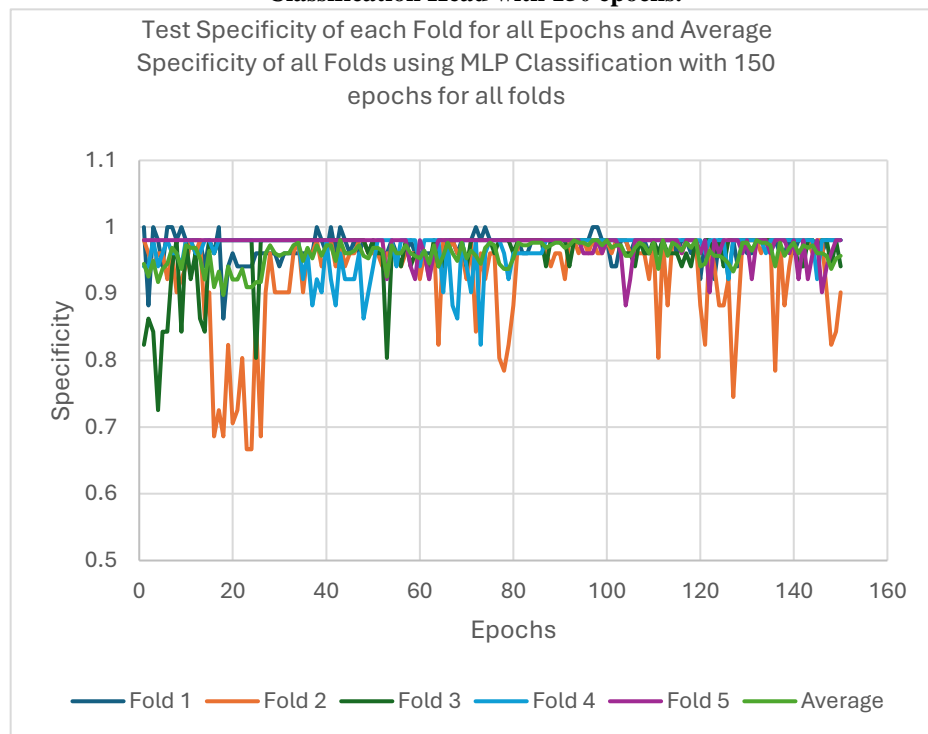
**xxii. Test Accuracy per epoch and Average Accuracy, across all folds using MLP Classification Head with 150 epochs.**



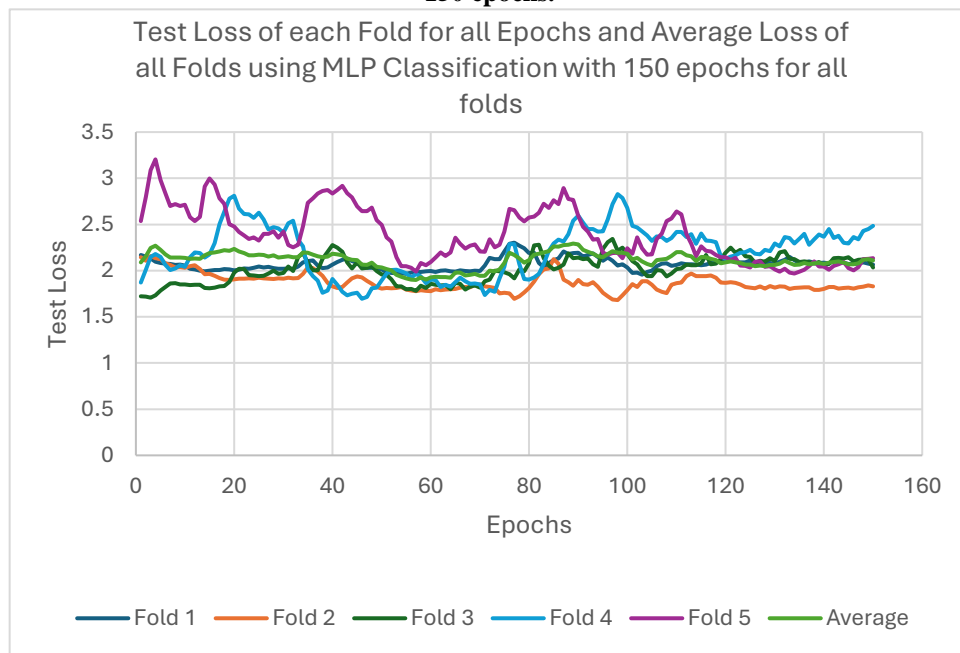
**xxiii. Test Sensitivity per epoch and Average Sensitivity, across all folds using MLP Classification Head with 150 epochs.**



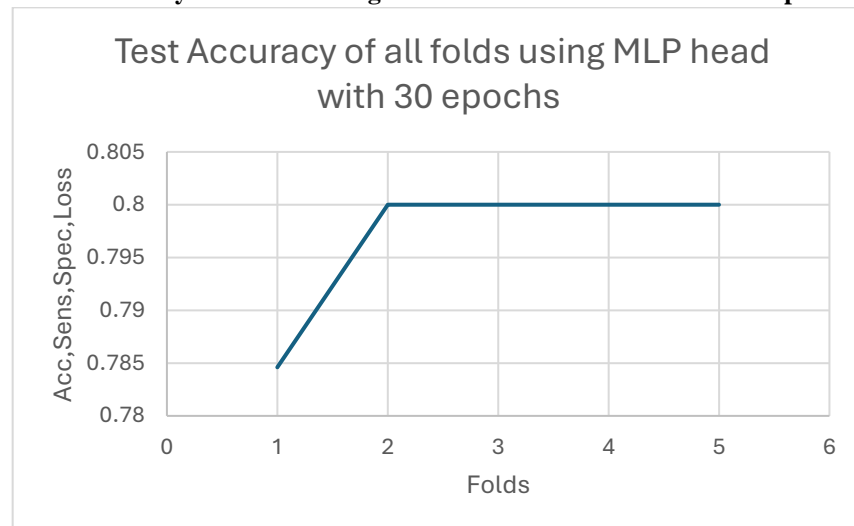
**xxiv. Test Specificity per epoch and Average Specificity, across all folds using MLP Classification Head with 150 epochs.**



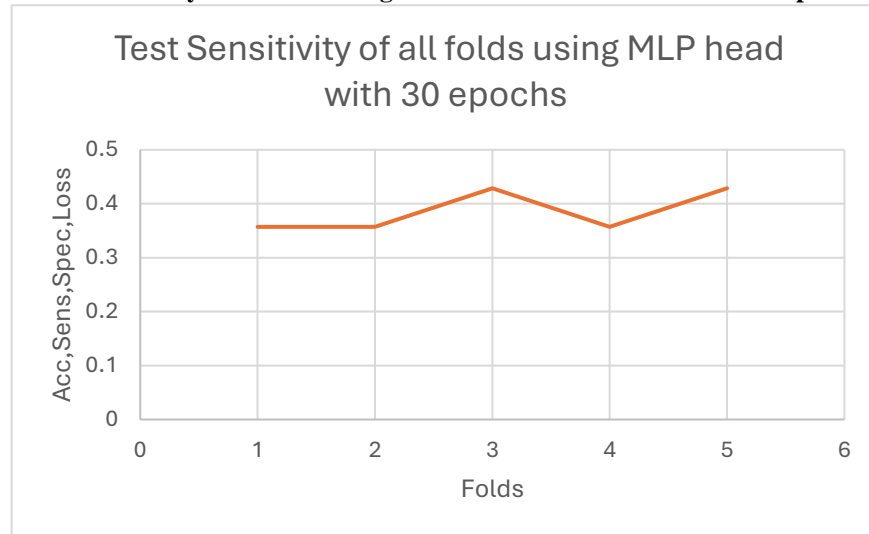
**xxv. Test Loss per epoch and Average Loss, across all folds using MLP Classification Head with 150 epochs.**



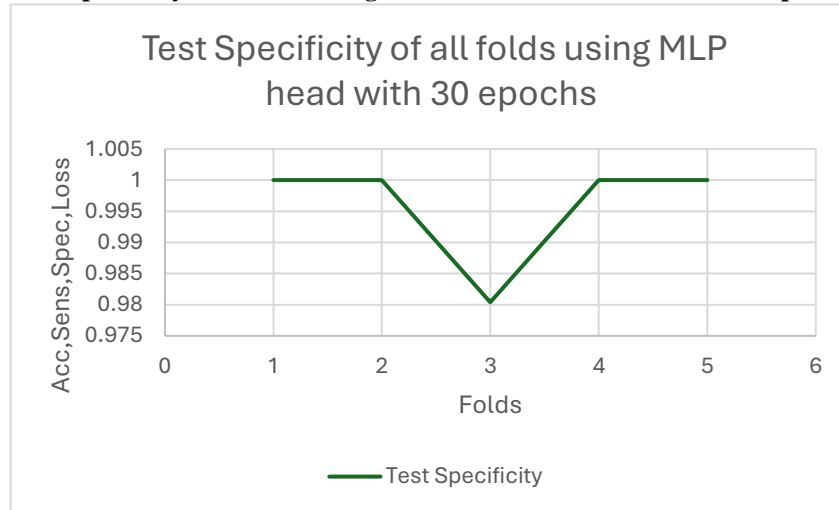
**xxvi. Test Accuracy of all folds using MLP Classification Head with 30 epochs.**



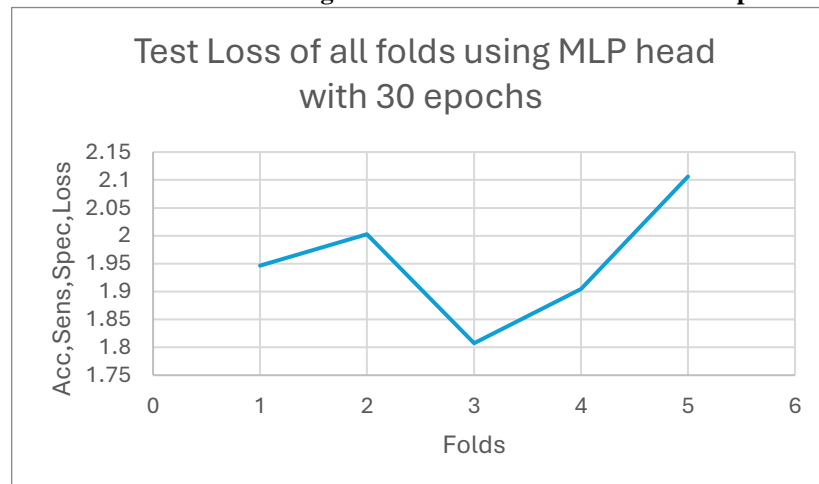
xxvii. **Test Sensitivity of all folds using MLP Classification Head with 30 epochs.**



xxviii. **Test Specificity of all folds using MLP Classification Head with 30 epochs.**



xxix. **Test Loss of all folds using MLP Classification Head with 30 epochs.**



xxx. DINOv2 compatible transformations on thermal breast images (colorization, normalization and resizing of images).

```
from PIL import Image
#define the output directory for transformed images
output_dir = "/content/drive/MyDrive/Dataset/DINOv2_images"
os.makedirs(output_dir, exist_ok=True) # Create folder if it doesn't exist

#define DINOv2-compatible transformations
dinov2_transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to 224x224
    transforms.ToTensor(), # Convert image to tensor (C, H, W)
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406], # Normalize using ImageNet values
        std=[0.229, 0.224, 0.225]
    )
])

#initialize tensor to store transformed images for DINOv2
imgs_tensor = torch.zeros(len(img_paths), 3, 224, 224) # (N, C, H, W)

#function to process image (Colorize + DINOv2 Transform) and save it as ONE file
def process_image(image_path, index):
    # Open and convert to grayscale
    img = Image.open(image_path).convert("L")
    img_np = np.array(img)

    #apply JET colormap
    img_color = cv2.applyColorMap(img_np, cv2.COLORMAP_JET)
    colored_img = Image.fromarray(img_color) # Convert to PIL format

    #apply DINOv2 transformations
    img_tensor = dinov2_transform(colored_img)
    imgs_tensor[index] = img_tensor # Store the transformed tensor

    #convert the tensor back to a displayable image
    img_numpy = img_tensor.permute(1, 2, 0).mul(torch.tensor([0.229, 0.224, 0.225])).add(torch.tensor([0.485, 0.456, 0.406])) # Undo normalization
    img_numpy = torch.clamp(img_numpy, 0, 1).cpu().numpy() # Ensure values are in range [0,1]
    img_resized = Image.fromarray((img_numpy * 255).astype(np.uint8)) # Convert to 8-bit image

    #generate new filename and save the transformed image in the DINOv2 folder
    base_name = os.path.basename(image_path).split('.')[0]
    dinov2_path = os.path.join(output_dir, f"{base_name}_DINOv2.jpg")
    img_resized.save(dinov2_path)

    return dinov2_path # Return the saved image path

#process all images
for i, img_path in enumerate(img_paths):
    print(f"Processing {img_path}...")
    dinov2_image_path = process_image(img_path, i)

print("All images have been **colorized, transformed, and saved** in 'DINOv2_images'.")
print("Final dataset tensor shape:", imgs_tensor.shape) # Should be (N, 3, 224, 224)

#extra step to check ** (optional)
for path in os.listdir(output_dir):
    full_path = os.path.join(output_dir, path)
    img = Image.open(full_path)
    if img.size != (224, 224):
        print(f"Not resized: {full_path} | Size: {img.size}")
```

**xxxi. Stratified-K(5)-Folds for the creation of training, validation and testing set.**

```
#Splitting the Dataset (into training, validation and testing set) using StratifiedKFolds
#imports and configuration
import os
import torch
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
import shutil

#number of k folds to use
n_splits = 5
#folder paths for saving split data and transformed images
output_folder = "/content/drive/MyDrive/Dataset/Split_Folds"
dinov2_images_dir = "/content/drive/MyDrive/Dataset/DINOv2_images"
os.makedirs(output_folder, exist_ok=True)

#check if the folds already exist
folds = []
found_all_folds = True

#try loading all 5 pre-saved fold files
for fold in range(1, n_splits + 1):
    fold_path = os.path.join(output_folder, f"dinov2_fold_{fold}.pt")
    if os.path.exists(fold_path):
        print(f"Fold {fold} found. Loading from file.")
        fold_data = torch.load(fold_path, weights_only=False)
        folds.append(fold_data)
    else:
        print(f"Fold {fold} not found. Will re-run data splitting.")
        found_all_folds = False
        break

#print the class distribution of the existing folds
if found_all_folds:
    print("\nLoaded Fold Stats:")

    for i, fold in enumerate(folds):
        print(f"\nFold {i + 1} - Dataset Breakdown:")

        for split_name, labels in {
            "Train": fold["y_train"],
            "Validate-Test": fold["y_test"],
            "Final Test": fold["y_final"]
        }.items():
            total = len(labels)
            healthy = sum(1 for label in labels if label == "Healthy")
            unhealthy = sum(1 for label in labels if label == "Unhealthy")
            print(f"    {split_name}: {total} images ({healthy} Healthy, {unhealthy} Unhealthy)")

#if any of the folds isn't foundm run stratified k-fold splitting
if not found_all_folds:
    print("\nRe-running StratifiedKFold splitting and saving folds...")

    labels = np.array(img_labels) # labels as NumPy array
    all_indices = np.arange(len(imgs_tensor)) #original indices for tracking
```



```

#split into 80% train/test and 20% final test and track the original indices
X_train_test, X_final, y_train_test, y_final, train_test_indices, final_indices = train_test_split(
    imgs_tensor, labels, all_indices, test_size=0.20, stratify=labels, shuffle=True, random_state=42
)

print(f"\nAfter split: {X_train_test.shape[0]} train/test, {X_final.shape[0]} final test")

skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

for fold, (train_index, test_index) in enumerate(skf.split(X_train_test, y_train_test)):
    X_train, X_test = X_train_test[train_index], X_train_test[test_index]
    y_train, y_test = y_train_test[train_index], y_train_test[test_index]

    # create the fold dictionary
    fold_data = {
        "X_train": X_train, "y_train": y_train,
        "X_test": X_test, "y_test": y_test,
        "X_final": X_final, "y_final": y_final, #final set shared across all folds
    }
    folds.append(fold_data)

    #save the newly created fold
    fold_path = os.path.join(output_folder, f"dinov2_fold_{fold + 1}.pt")
    torch.save(fold_data, fold_path)
    print(f"Saved fold {fold + 1} -> {fold_path}")

    split_images_root = os.path.join(output_folder, "SPLITS", f"fold_{fold + 1}")
    os.makedirs(split_images_root, exist_ok=True)

#map split names to indices and labels
split_mapping = {
    "Train": (train_index, y_train),
    "Validate-Test": (test_index, y_test),
    "Final Test": (final_indices, y_final)
}

print(f"\n Fold {fold + 1} - Dataset Breakdown:")

for split_name, (indices, y_labels) in split_mapping.items():
    split_folder = os.path.join(split_images_root, split_name)
    os.makedirs(split_folder, exist_ok=True)

    #print split summary
    total = len(indices)
    healthy = sum(1 for label in y_labels if label == "Healthy")
    unhealthy = sum(1 for label in y_labels if label == "Unhealthy")
    print(f" {split_name}: {total} images ({healthy} Healthy, {unhealthy} Unhealthy)")

    #copy image files into the appropriate split folder
    for idx in indices:
        original_index = idx if split_name == "Final Test" else train_test_indices[idx]

        orig_img_path = img_paths[original_index]
        base_name = os.path.basename(orig_img_path).split('.')[0]
        transformed_name = f"{base_name}_DINOv2.jpg"
        transformed_path = os.path.join(dinov2_images_dir, transformed_name)

    if os.path.exists(transformed_path):
        shutil.copy(transformed_path, os.path.join(split_folder, transformed_name))
    else:
        print(f"Missing transformed image: {transformed_path}")

```

xxxii. **Final Implementation of Linear Classification Head : Training, Evaluation and Testing.**

```
# #NEW Pure Linear HEAD https://github.com/NielsRogge/Transformers-Tutorials/blob/
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, f1_score
import numpy as np
import os
from sklearn.preprocessing import StandardScaler
import glob

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#labeling the healthy and unhealthy class to numerical values
label_mapping = {"Healthy": 0, "Unhealthy": 1}
folds_folder = "/content/drive/MyDrive/Dataset/Split_Folds"

#freeze the entire model
for param in model.parameters():
    param.requires_grad = False

#define and attach linear head
num_features = model.norm.weight.shape[0]
model.fc = nn.Linear(num_features, 1).to(device)

#loop through each fold
for fold in range(1, 6):
    print(f"\n Fold {fold}")

    fold_path = os.path.join(folds_folder, f"dinov2_fold_{fold}.pt")
    data = torch.load(fold_path, weights_only=False)
    final_test_folder = os.path.join("/content/drive/MyDrive/Dataset/Split_Folds/SPLITS", f"fold_{fold}", "Final Test")
    final_filenames = sorted([os.path.basename(f) for f in glob.glob(os.path.join(final_test_folder, "*"))])

    #prepare list to collect misclassified samples
    misclassified_final_samples = []

    def encode_labels(y):
        return torch.tensor([label_mapping[label] for label in y], dtype=torch.float32)

    X_train = data["X_train"].to(device)
    y_train = encode_labels(data["y_train"]).to(device)
    X_test = data["X_test"].to(device)
    y_test = encode_labels(data["y_test"]).to(device)
    X_final = data["X_final"].to(device)
    y_final = encode_labels(data["y_final"]).to(device)

    #feature extraction with debugging
    try:
        print("Extracting DINOv2 features...")
        with torch.no_grad():
            scaler = StandardScaler()

            f_train = model.forward_features(X_train)["x_norm_clstoken"]
            f_test = model.forward_features(X_test)["x_norm_clstoken"]
            f_final = model.forward_features(X_final)["x_norm_clstoken"]
```

```

        f_train = scaler.fit_transform(f_train.cpu())
        f_test = scaler.transform(f_test.cpu())
        f_final = scaler.transform(f_final.cpu())

        f_train = torch.tensor(f_train, device=device).float()
        f_test = torch.tensor(f_test, device=device).float()
        f_final = torch.tensor(f_final, device=device).float()

    print("Feature extraction successful.")
except Exception as e:
    print(f"Feature extraction failed: {e}")
    continue

#prepare data loaders of each set
train_loader = DataLoader(TensorDataset(f_train, y_train), batch_size=32, shuffle=True)
test_loader = DataLoader(TensorDataset(f_test, y_test), batch_size=32)
final_loader = DataLoader(TensorDataset(f_final, y_final), batch_size=32)

#loss & optimizer
pos_weight = torch.tensor([1.35], device=device)
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
optimizer = optim.AdamW(model.fc.parameters(), lr=1e-3)

#training loop
n_epochs = 50
for epoch in range(n_epochs):

    #train
    model.fc.train()
    train_loss = 0.0

train_preds_raw, train_labels_list = [], []

    for features, labels in train_loader:
        optimizer.zero_grad()
        outputs = model.fc(features).squeeze(1)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        train_preds_raw.extend(outputs.detach().cpu().numpy())
        train_labels_list.extend(labels.cpu().numpy())

    #validate
    model.fc.eval()
    val_preds_raw, val_labels_list = [], []

    with torch.no_grad():
        for features, labels in test_loader:
            outputs = model.fc(features).squeeze(1)
            val_preds_raw.extend(outputs.cpu().numpy())
            val_labels_list.extend(labels.cpu().numpy())

    fpr, tpr, thresholds = roc_curve(val_labels_list, val_preds_raw)
    high_sens_mask = tpr >= 0.90
    candidate_thresholds = thresholds[high_sens_mask]

    if candidate_thresholds.size > 0:
        f1_scores = []
        for i, thresh in enumerate(candidate_thresholds):

```

```

        for i, thresh in enumerate(candidate_thresholds):
            preds = (np.array(val_preds_raw) > thresh).astype(int)
            if (tpr[high_sens_mask][i] >= 0.90):
                f1_scores.append(f1_score(val_labels_list, preds))
            else:
                f1_scores.append(0)
        best_thresh = candidate_thresholds[np.argmax(f1_scores)]
    else:
        print("No threshold satisfies sensitivity  $\geq 0.90$  – using 0.5 fallback.")
        best_thresh = 0.5

#metrics: Training
train_preds = (np.array(train_preds_raw) > best_thresh).astype(int)
train_acc = accuracy_score(train_labels_list, train_preds)
tn, fp, fn, tp = confusion_matrix(train_labels_list, train_preds).ravel()
train_sens = tp / (tp + fn) if (tp + fn) > 0 else 0
train_spec = tn / (tn + fp) if (tn + fp) > 0 else 0
train_f1 = f1_score(train_labels_list, train_preds)

#metrics: Validation
val_preds = (np.array(val_preds_raw) > best_thresh).astype(int)
val_acc = accuracy_score(val_labels_list, val_preds)
tn, fp, fn, tp = confusion_matrix(val_labels_list, val_preds).ravel()
val_sens = tp / (tp + fn) if (tp + fn) > 0 else 0
val_spec = tn / (tn + fp) if (tn + fp) > 0 else 0
val_f1 = f1_score(val_labels_list, val_preds)

#validation loss
validation_loss = 0.0
with torch.no_grad():
    for features, labels in test_loader:

        #validation loss
        validation_loss = 0.0
        with torch.no_grad():
            for features, labels in test_loader:
                outputs = model.fc(features).squeeze(1)
                loss = criterion(outputs, labels)
                validation_loss += loss.item()
        validation_loss /= len(test_loader)

#metrics: Final Test
final_preds_raw, final_labels = [], []
test_loss = 0.0
with torch.no_grad():
    for features, labels in final_loader:
        outputs = model.fc(features).squeeze(1)
        final_preds_raw.extend(outputs.cpu().numpy())
        final_labels.extend(labels.cpu().numpy())
        loss = criterion(outputs, labels)
        test_loss += loss.item()
    test_loss /= len(final_loader)

final_preds = (np.array(final_preds_raw) > best_thresh).astype(int)
final_acc = accuracy_score(final_labels, final_preds)
tn, fp, fn, tp = confusion_matrix(final_labels, final_preds).ravel()
final_sens = tp / (tp + fn) if (tp + fn) > 0 else 0
final_spec = tn / (tn + fp) if (tn + fp) > 0 else 0
final_f1 = f1_score(final_labels, final_preds)

```

```

#after all epochs of the fold finished ==
#print(f"\nMisclassified images for Fold {fold}:")
for fold_id, epoch_id, filename, true_label, pred_label in misclassified_final_samples:
    if epoch_id == n_epochs: # only print misclassifications from final epoch
        print(f"[Fold {fold_id}] {filename} | True: {true_label} → Predicted: {pred_label}")

#save misclassified Final Test images for this fold ==
output_path = f"/content/drive/MyDrive/Dataset/misclassified_fold_{fold}.txt"
with open(output_path, "w") as f:
    for fold_id, epoch_id, filename, true_label, pred_label in misclassified_final_samples:
        f.write(f"Fold {fold_id}, Epoch {epoch_id}, File: {filename}, True Label: {true_label}, Predicted: {pred_label}\n")

```

### xxxiii. Final Implementation of MLP Classification Head: Training, Evaluation and Testing.

```

#Classification using MLP head
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from torchvision import transforms
from torch.utils.data import DataLoader, Dataset
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, f1_score
import os
import glob

#custom dataset class with optional augmentation
class AugmentedTensorDataset(Dataset):
    def __init__(self, X, y, transform=None):
        self.X = X #input features
        self.y = y #labels 1 or 0
        self.transform = transform #optional data augm

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        x = self.X[idx]
        y = self.y[idx]
        if self.transform:
            x = self.transform(x)
        return x, y

#ensure model is loaded
assert 'model' in globals(), "DINOv2 model not loaded!"
#set the device for training
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# Define final_filenames for tracking misclassified images
final_test_folder = os.path.join("/content/drive/MyDrive/Dataset/Split_Folds/SPLITS", f"fold_{fold}", "Final Test")
final_filenames = sorted([os.path.basename(f) for f in glob.glob(os.path.join(final_test_folder, "*"))])

#freeze all model parameters except for final transformer block and classification head
for param in model.parameters():
    param.requires_grad = False
for name, param in model.named_parameters():
    if "block.11" in name or "fc" in name:
        param.requires_grad = True

#MLP HEAD
num_features = model.norm.weight.shape[0]
model.fc = nn.Sequential(
    nn.Linear(num_features, 128),
    nn.BatchNorm1d(128),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(128, 1)
).to(device)

```

```

folds_folder = "/content/drive/MyDrive/Dataset/Split_Folds"
label_mapping = {"Healthy": 0, "Unhealthy": 1} #map classes
n_epochs = 30
batch_size = 32
learning_rate_fc = 2e-4
learning_rate_vit = 5e-7
weight_decay = 1e-4

#loop over each fold
for fold in range(1, 6):
    print(f"\nTraining on Fold {fold}...")
    #load fold data
    fold_path = os.path.join(folds_folder, f"dinov2_fold_{fold}.pt")
    data = torch.load(fold_path, map_location=device, weights_only=False)
    #extract features and labels and move to device
    X_train = data["X_train"].float().to(device)
    y_train = torch.tensor([label_mapping[l] for l in data["y_train"]], dtype=torch.float32, device=device)
    X_test = data["X_test"].float().to(device)
    y_test = torch.tensor([label_mapping[l] for l in data["y_test"]], dtype=torch.float32, device=device)
    X_final = data["X_final"].float().to(device)
    y_final = torch.tensor([label_mapping[l] for l in data["y_final"]], dtype=torch.float32, device=device)
    #handle class imbalance by using a positive class weight
    pos_weight = torch.tensor([len(y_train) / (2 * y_train.sum())], dtype=torch.float32, device=device)
    criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)

    #optimizer for mlp and last block
    optimizer = optim.AdamW([
        {'params': model.fc.parameters(), 'lr': learning_rate_fc},
        {'params': model.blocks[11:].parameters(), 'lr': learning_rate_vit},
    ], weight_decay=weight_decay)

    #learning rate scheduler
    scheduler = optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=10, T_mult=2)

    #compute mean and std for normalization
    mean = X_train.mean(dim=(0, 2, 3), keepdim=True)
    std = X_train.std(dim=(0, 2, 3), keepdim=True) + 1e-6

    #define data augmentation
    train_transform = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(45),
        transforms.ColorJitter(brightness=0.6, contrast=0.6),
        transforms.RandomErasing(p=0.7, scale=(0.02, 0.25)),
        transforms.Normalize(mean=mean.squeeze(), std=std.squeeze()),
    ])
    test_transform = transforms.Compose([transforms.Normalize(mean=mean.squeeze(), std=std.squeeze())])

    #create datasets and data loaders
    train_dataset = AugmentedTensorDataset(X_train, y_train, train_transform)
    test_dataset = AugmentedTensorDataset(X_test, y_test, test_transform)
    final_dataset = AugmentedTensorDataset(X_final, y_final, test_transform)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

```

```

test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
final_loader = DataLoader(final_dataset, batch_size=batch_size, shuffle=False)
#create a list to track misclassified samples
misclassified_final_samples = []

#training loop
for epoch in range(n_epochs):
    model.train()
    train_loss = 0
    train_preds_raw, train_labels = [], []

    #TRAINING
    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        #forward pass through vit and mlp
        outputs = model.fc(model.forward_features(batch_X)["x_norm_clstoken"]).squeeze(1)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        train_preds_raw.extend(outputs.detach().cpu().numpy())
        train_labels.extend(batch_y.cpu().numpy())

    #optimal threshold using roc curve
    fpr, tpr, thresholds = roc_curve(train_labels, train_preds_raw)
    best_threshold = thresholds[np.argmax(tpr - fpr)]
    train_preds = (np.array(train_preds_raw) > best_threshold).astype(int)
    train_f1 = f1_score(train_labels, train_preds)
    train_acc = accuracy_score(train_labels, train_preds)

    tn, fp, fn, tp = confusion_matrix(train_labels, train_preds).ravel()
    train_sens = tp / (tp + fn) if (tp + fn) > 0 else 0
    train_spec = tn / (tn + fp) if (tn + fp) > 0 else 0

    #VALIDATION
    model.eval()
    val_preds_raw, val_labels = [], []
    val_loss = 0
    with torch.no_grad():
        for batch_X, batch_y in test_loader:
            outputs = model.fc(model.forward_features(batch_X)["x_norm_clstoken"]).squeeze(1)
            loss = criterion(outputs, batch_y)
            val_loss += loss.item()
            val_preds_raw.extend(outputs.cpu().numpy())
            val_labels.extend(batch_y.cpu().numpy())

    val_preds = (np.array(val_preds_raw) > best_threshold).astype(int)
    val_f1 = f1_score(val_labels, val_preds)
    val_acc = accuracy_score(val_labels, val_preds)
    tn, fp, fn, tp = confusion_matrix(val_labels, val_preds).ravel()
    val_sens = tp / (tp + fn) if (tp + fn) > 0 else 0
    val_spec = tn / (tn + fp) if (tn + fp) > 0 else 0

    #FINAL TESTING
    final_preds_raw, final_labels = [], []
    final_loss = 0
    with torch.no_grad():
        for batch_X, batch_y in final_loader:
            outputs = model.fc(model.forward_features(batch_X)["x_norm_clstoken"]).squeeze(1)
            loss = criterion(outputs, batch_y)
            final_loss += loss.item()
            final_preds_raw.extend(outputs.cpu().numpy())
            final_labels.extend(batch_y.cpu().numpy())

    final_preds = (np.array(final_preds_raw) > best_threshold).astype(int)
    final_f1 = f1_score(final_labels, final_preds)
    final_acc = accuracy_score(final_labels, final_preds)
    tn_f, fp_f, fn_f, tp_f = confusion_matrix(final_labels, final_preds).ravel()
    final_sens = tp_f / (tp_f + fn_f) if (tp_f + fn_f) > 0 else 0
    final_spec = tn_f / (tn_f + fp_f) if (tn_f + fp_f) > 0 else 0

```

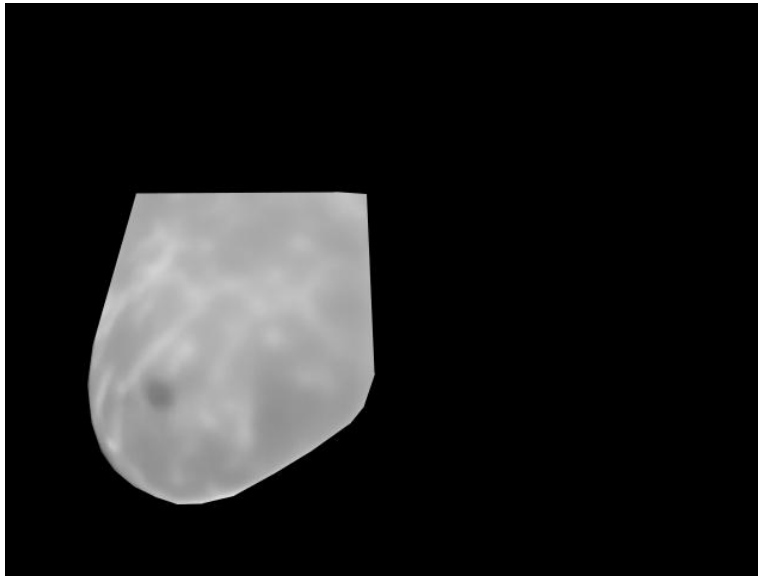
```

print(f"Epoch {epoch+1}/{n_epochs} | Train | Loss: {train_loss:.4f} | Acc: {train_acc:.4f} | Sens: {train_sens:.4f} | Spec: {train_spec:.4f} | F1: {train_f1:.4f} | "
      f"Validate | Loss : {val_loss:.4f} | Acc: {val_acc:.4f} | Sens: {val_sens:.4f} | Spec: {val_spec:.4f} | F1: {val_f1:.4f} || "
      f"Final Test | Loss : {final_loss:.4f} | Acc: {final_acc:.4f} | Sens: {final_sens:.4f} | Spec: {final_spec:.4f} | F1: {final_f1:.4f} | "
      f"TP: {tp_f} | FP: {fp_f} | FN: {fn_f} | TN: {tn_f}")

#track misclassified images in final test
if epoch == n_epochs - 1: # only track misclassified images after the final epoch
    for global_idx, (pred, label) in enumerate(zip(final_preds, final_labels)):
        if pred != label:
            filename = final_filenames[global_idx]
            misclassified_final_samples.append((fold, epoch+1, filename, int(label), int(pred)))
    scheduler.step()
#after all epochs for this fold, print and save the misclassified images
print(f"\nMisclassified images for Fold {fold}:")
for fold_id, epoch_id, filename, true_label, pred_label in misclassified_final_samples:
    if epoch_id == n_epochs: # Only print misclassifications from final epoch
        print(f"[Fold {fold_id}] {filename} | True: {true_label} → Predicted: {pred_label}")

```

xxxiv. **Example of Cropped Image which was excluded from the dataset: Missing Front View of Both Breasts** <sup>[27]</sup>





## References :

- [1]. G. Ayana, K. Dese, N. Husen, Y. Dereje, Y. Kebede, H. Barki, F. Mulugeta, D. Amdissa, B. Habtamu, and S.-W. Choe, "Vision-Transformer-Based Transfer Learning for Mammogram Classification." *Diagnostics*, 13(2), 178, 2023.
- [2]. Europa Donna Cyprus, "Breast Cancer Facts," Europa Donna Cyprus, 2024. [Online]. Available: <https://europadonna.com.cy/en/breast-and-gynaecological-cancers/breast-cancer/breast-cancer-facts/>. Accessed: Mar. 18, 2025.
- [3]. Breastcancer.org, "Breast Cancer Facts and Statistics," Breastcancer.org, 2024. [Online]. Available: <https://www.breastcancer.org/facts-statistics>. Accessed: Mar. 18, 2025.
- [4]. Ezra, "Why Is Early Detection of Breast Cancer Important?," Ezra, 2024. [Online]. Available: <https://ezra.com/blog/why-is-early-detection-of-breast-cancer-important>. Accessed: Mar. 18, 2025.
- [5]. L. S. Garia and M. Hariharan, "Vision Transformers for Breast Cancer Classification from Thermal Images," in *Robotics, Control and Computer Vision: Select Proceedings of ICRCCV 2022*, H. Muthusamy, J. Botzheim, and R. Nayak, Eds., Lecture Notes in Electrical Engineering, vol. 1009, Singapore: Springer, 2023, pp. 177–186. [Online]. Available: <https://dokumen.pub/qdownload/robotics-control-and-computer-vision-select-proceedings-of-icrccv-2022-9819902355-9789819902354.html>
- [6]. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale," arXiv:2010.11929, 2020.
- [7]. A. Abunasser, M. L. L. Toledo, F. J. P. Lopes, and R. S. Oliveira, "BCCNN: A Custom CNN Architecture for Breast Cancer Detection and Classification in Histopathological Images," *Diagnostics*, vol. 13, no. 5, 2023. [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC10162639/>.
- [8]. Yan, J. (2023). Study for Performance of MobileNetV1 and MobileNetV2 based on Breast Cancer. *Highlights in Science, Engineering and Technology*, 39, 10-14.
- [9]. M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "DINOv2: Learning Robust Visual Features without Supervision," arXiv:2304.07193, 2024.
- [10]. X. Lu, H. Wang, and D. Fischer, "DINO-Mix: Enhancing Visual Place Recognition with Foundational Vision Model and Feature Mixing," *ResearchGate*, 2023. [Online]. Available: [https://www.researchgate.net/publication/377064388\\_DINO-Mix\\_Enhancing\\_Visual\\_Place\\_Recognition\\_with\\_Foundational\\_Vision\\_Model\\_and\\_Feature\\_Mixing](https://www.researchgate.net/publication/377064388_DINO-Mix_Enhancing_Visual_Place_Recognition_with_Foundational_Vision_Model_and_Feature_Mixing). Accessed: May 5, 2025.
- [11]. T. Darcet, M. Oquab, J. Mairal, and P. Bojanowski, "Vision Transformers Need Registers," arXiv:2309.16588, 2023.
- [12]. Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345–1359.
- [13]. Facebook Research, "DINOv2: Learning Visual Features without Supervision," GitHub, 2024. [Online]. Available: <https://github.com/facebookresearch/dinov2?tab=readme-ov-file>. Accessed: Sept. 09, 2024.
- [14]. D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," 2016. [Online]. Available: <https://arxiv.org/pdf/1606.08415>. Accessed: May 5, 2025.
- [15]. J. Ba, J. Kiros, and G. Hinton, "Layer Normalization," 2016. [Online]. Available: <https://arxiv.org/pdf/1607.06450>. Accessed: May 5, 2025.
- [16]. V. Pariza, "NeCo: A New SSL Post-Training Approach for Improving DINOv2's Spatial Representations in 19 GPU Hours with Patch Neighbor Consistency," GitHub, 2024. [Online]. Available: <https://github.com/vpariza/NeCo>. Accessed: Jan. 20, 2025.
- [17]. IBM Corp., "Area under the Curve," *IBM Documentation*, 2024. [Online]. Available: <https://www.ibm.com/docs/en/spss-statistics/30.0.0?topic=schemes-area-under-curve>. Accessed: Mar. 4, 2025.
- [18]. N. Rogge, "Train a Linear Classifier on Top of DINOv2 for Semantic Segmentation," GitHub, 2024. [Online]. Available: [https://github.com/NielsRogge/Transformers-Tutorials/blob/master/DINOv2/Train\\_a\\_linear\\_classifier\\_on\\_top\\_of\\_DINOv2\\_for\\_semantic\\_segmentation.ipynb](https://github.com/NielsRogge/Transformers-Tutorials/blob/master/DINOv2/Train_a_linear_classifier_on_top_of_DINOv2_for_semantic_segmentation.ipynb). Accessed: Feb. 20, 2025.

- [19]. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <https://www.deeplearningbook.org/>. Accessed: Mar. 07, 2025.
- [20]. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv*, Feb. 13, 2020. [Online]. Available: <https://arxiv.org/abs/2002.05709>. Accessed: Mar. 09, 2025.
- [21]. "Database for Mastology Research," Visual Lab, Universidade Federal Fluminense, 2024. [Online]. Available: <https://visual.ic.uff.br/en/proeng/thiagoelias/>. Accessed: Nov. 01, 2024.
- [22]. S. Rodriguez-Guerrero, H. Loaiza Correa, A.-D. Restrepo-Girón, L. A. Reyes, L. A. Olave, S. Diaz, and R. Pacheco, "Breast Thermography," Mendeley Data, Version 3, Feb. 5, 2024. [Online]. Available: <https://data.mendeley.com/datasets/mhrt4svjxc/3>. Accessed: Jan. 01, 2025.
- [23]. "Scikit-Learn Cross-Validation Module," Scikit-learn.org, 2024. [Online]. Available: [https://scikit-learn.org/1.5/modules/cross\\_validation.html](https://scikit-learn.org/1.5/modules/cross_validation.html). Accessed: Nov. 11, 2024.
- [24]. A. Luque, A. Carrasco, A. Martín, and A. de las Heras, "The impact of class imbalance in classification performance metrics based on the binary confusion matrix," *Pattern Recognition*, vol. 91, pp. 216–231, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0031320319300950>.
- [25]. D. M. Berrar, "ROC curves – Practical Stats in Medical Research," [Online]. Available: <https://practical-stats-med-r.netlify.app/roc>. Accessed: Feb. 27, 2025.
- [26]. F. Pedregosa, G. Varoquaux, A. Gramfort, et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc\\_curve.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html)
- [27]. "A New Database for Breast Research with Infrared Image," ResearchGate, 2015. [Online]. Available: [https://www.researchgate.net/publication/272274878\\_A\\_New\\_Database\\_for\\_Breast\\_Research\\_with\\_Infrared\\_Image](https://www.researchgate.net/publication/272274878_A_New_Database_for_Breast_Research_with_Infrared_Image). Accessed: Nov. 01, 2024
- [28]. Google, "Teachable Machine," Google, [Online]. Available: <https://teachablemachine.withgoogle.com/>.

