DETECTION OF MELANOMA (SKIN CANCER) FROM PIGMENTED SKIN LESION IMAGES USING MODIFIED MLP-MIXER MODEL

PRASAD PRAMOD MAHARANA

FINAL THESIS REPORT

Nov 2022

DEDICATION

This work is dedicated to my mother (Gayatri Maharana) and in loving memory of my father (Pramod Maharana) whose constant guidance and encouragement has steered my life. “You may be gone from my sight, but you are never gone from my heart”. I would also like to dedicate this work to all the scientists, researchers and organizations who are working tirelessly to bring in the latest innovations to fight skin cancer.

ACKNOWLEDGEMENT

This research would not have been possible without the assistance, support, and guidance of others. Thankyou Prof. Dr. Ahmed Kaky of Liverpool John Moores University for providing guidance throughout this research project.

Thank you to Dr. Rupal Bhargava and her team for teaching me how to write a good research paper. My heartfelt gratitude and thanks go to Dr. Jyoti Dabass, my thesis supervisor, for reviewing the work, providing valuable feedback, and guiding me through this journey.

Finally, I would like to express my gratitude to my family and friends for their unwavering support and faith in me. Special mention to my work family at General Mills India who have been equally supportive in this research work.

ABSTRACT

Melanoma, one of the most serious forms of skin cancer arises from melanocytes, which produce pigment. According to Cancer Research UK, melanoma has claimed 2,341 lives during 2017-2019 in United Kingdom. However, it is curable and has a survival rate of 87%. Histopathological analysis and biopsy are the main diagnostic methods to detect melanoma. Recent non-invasive cancer detection methods involving imaging pigmented skin lesion images sparked interests in deep learning community. This led to using convolutional neural networks (CNN) and Transformers to accelerate detection of melanoma which showed promising results. The existing CNN and Transformer models rely on convolution and attention modules to detect features and make decisions. These models have few drawbacks like complexity, long training hours and intensive computing resources usage (GPU-graphics processing unit and memory).

To overcome these drawbacks, we have used an approach of multilayer perceptrons (MLP) based mixer architecture. To achieve this, an architecture known as MLP-mixer takes the input images as patches which are converted into a table like form and then fed to a mixer layer. The mixer layer contains 2 MLP blocks, the first one (token mixing) detects features in the image across patches viz. aggregates channels where the feature occurs. The second block looks for features in the patch and associates it with the channel. It also uses Layer Norm along with GELU (Gaussian Error Linear Unit) activations and skip connections. Using experimental settings for parameters like patch size, layers, MLP blocks and dropouts, this proposed model was built and trained on skin lesion images which were pre-processed to ensure uniform input. The best model developed from the experiment obtained binary classification accuracy, precision, recall and f1-score of 87.33%, 87.6%, 87.2% and 87.3% respectively on the HAM10000 dataset. The results were competitive to state-of-the-art (SOTA) CNN and Transformer models on melanoma detection. This concluded in a lightweight model which can be trained faster on economical computation resources and deployed into real-time applications. We hope the results will spark more interests to explore MLP architecture and their development

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Definition**
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DeiT	Data-efficient image Transformer
GELU	Gaussian Error Linear Unit
GPU	Graphics Processing Unit
HAM	Human Against Machine
ICLR	International Conference on Learning Representations
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
ISIC	International Skin Imaging Collaboration
MHAN	Muti-Head Attention Network
MLOps	Machine Learning operations
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
SOTA	State-of-the-art
UK	United Kingdom
ViT	Vision Transformer

CHAPTER 1

INTRODUCTION

1.1 Background

Melanoma, also known as malignant melanoma, is a type of skin cancer that emerges from the pigment-producing cells called melanocytes. Melanoma usually occur in the skin, but may rarely occur in other places like mouth, intestines, or eye (uveal melanoma)(PDQ Adult Treatment Editorial Board, 2002). In UK, there were 16,744 cases on average during the years 2016 – 2018 and 2,341 deaths during the years 2017 – 2019. Melanoma has a survival rate of 87% and a prevention rate of 86% (Cancer Research UK, 2022).

There are diverse methods to diagnose melanoma which include invasive methods like biopsy and histopathological examination. Some of non-invasive techniques include thermography, electrical bio-impedance, tape stripping, dermatoscopy and computer-aided analysis (Narayanamurthy et al., 2018,). These methods sparked interests in utilizing pigmented skin lesion image samples to accelerate melanoma detection which can be followed by in-clinic consultation and treatment. Some of the datasets that pooled such diverse skin lesion images include BCN_20000 dataset (Combalia et al., 2019), HAM10000 dataset (Tschandl et al., 2018)and MSK dataset (Codella et al., 2017).

CNNs have been highly preferred due to their accuracy and reliability for feature detection in classification tasks. In 2012, a 0.23 percent error rate was recorded on the MNIST database. Another article on CNN image classification stated that the learning curve was "surprisingly fast". In the same research work, The MNIST and NORB databases yielded the best published findings of 2011. Later, the 2012 ImageNet Large Scale Visual Recognition Challenge was won by AlexNet with a top-5 error rate of 18.2% (Krizhevsky et al., 2017). CNNs rely on convolution technique to extract features from the input and make decisions. Inspired by the achievements of transformers in natural language processing (NLP) they were extended to computer vision applications. Vision Transformers (ViT) firstly introduced a self-attention-based backbone to computer vision which further led to exploration of strategies to improve training of ViT. This further resulted in ViT achieving comparable performance to CNNs trained solely on ImageNet. It beat the state-of-the-art models on multiple image recognition tasks with

the best model reaching accuracy of 94.55% on CIFAR-100, 90.72% on ImageNet-ReaL and 88.58% on ImageNet (Dosovitskiy et al., 2020).

While CNNs and Transformers have shown remarkable performances, the computational costs on such techniques are usually high. Considering the large computational costs associated with CNNs and attention modules in Transformers, a simple and efficient model that consists of Multilayer Perceptrons (MLP) was proposed in 2021 (Tolstikhin et al., 2021). This new architecture utilised techniques such as token mixing, channel mixing and matrix multiplication instead of convolution.

This new architecture claimed that while convolution and attention give good performance, both aren't necessary which sparked our interest. MLP-mixer achieves competitive results on image classification benchmarks when trained on larger datasets with regularisation techniques, with pre-training and inference costs equivalent to SOTA models. MLP can perform well with modest model sizes, but as the model size increases, it suffers greatly from over-fitting.

In this research proposal some of the drawbacks of CNN and transformers will be addressed by utilizing MLP mixer into image classification task i.e., detection of melanoma. The three datasets have been combined to include diverse images for the model to learn. The focus of this proposed work shall be on creating a model void of intensive computation modules like convolution or attention and performance to the likes of state-of-the-art CNN and Transformer models.

The major contributions are summarized below:

1. A new classification model based on the MLP mixer architecture is proposed to classify the pigmented skin lesion images.
2. A lightweight model that trains faster compared to state-of the art models that requires less computational resources without losing performance.

1.2 Problem Statement and Related Work

CNN models have been very efficient at image classification and image detection tasks. CNNs have been outperforming themselves with new architectures and methods. Although, these models rely on convolution to detect features in images to make decisions. While convolution is good at detecting features, it is not necessary.

As CNN models have evolved, starting from AlexNet, to VGGNet, ResNet and Inception, the complexity and depth has increased as a result of increased number of layers. In recent models (networks), the receptive fields of the input layers cover the entire input image which means that the context used by the features in the resultant output feature map encompasses all the input image pixels(Araujo et al., 2019). Although there has been a rapid advancement of neural network architectures, convolution still remains as the fundamental method in deep neural networks. Drawing inspiration from the long-established image filtering technique, convolution kernels have two exceptional properties that make them so robust, namely spatial-agnostic, and channel-specific (Li et al., 2021). A proposed work in 2021 performed skin cancer detection using CNN which resulted in training accuracy of 83.04%, precision of 0.81 and recall of 0.80 (Subramanian et al., 2021)

Nonetheless, inspired by the many advancements achieved by transformers in Natural Language Processing (NLP), they have been extended into computer vision applications. In particular, Vision Transformer (ViT), a pure Transformer when applied to image recognition, achieves remarkable performance competing with CNNs. Various studies performed by ViT have shown that the state-of-the-art (SOTA) can be accomplished for a broad range of computer vision tasks using self-attention solely without using convolution(Tatsunami and Taki, 2022). The experimentation began with applying standard transformer directly to images with minimum modifications. To achieve this, the image was split into patches and these patches were fed as a sequence of linear embedding as an input to the transformer. Image patches were processed in a similar way as tokens (words) in an NLP application. Then the model was trained as an image classifier in supervised method. When trained on mid-sized datasets like ImageNet without any strong regularization, this model resulted low-key accuracies close to ResNet of comparable size(Dosovitskiy et al., 2020).

This new Transformer structure which made appearance in ICLR in 2021, Vision Transformer, demonstrated that a Transformer implemented directly to a sequence of images (patches) can accomplish a good performance on image classification tasks provided the training dataset is

sufficiently large. Vision Transformer was also tested on skin cancer detection with resulted in AUC (area under ROC curve) of 0.771±0.018 which shows improvement in skin lesion diagnosis tasks (de Lima and Krohling, 2022). Data-efficient image Transformer (DeiT) further indicated that Transformers can be trained on classic scale dataset such as ImageNet-1k along with relevant data augmentation and model regularizations (Zhao et al., 2021). Vision Transformers firstly introduced a self-attention-based backbone to computer vision which further led to exploration of strategies to improve training of ViT. This further resulted in proposal of a knowledge distillation method which helped ViT achieve comparable performance to CNNs trained solely on ImageNet (Wei et al., 2022).

While CNNs and Transformers have shown remarkable performances, the computational costs on such techniques are usually high. Resources like GPU, memory and training time play a major role while implementing models. While these techniques can achieve certain computer vision tasks, it is also important to evaluate the resources required to build and train these models. If a certain model can classify images at an intensive computation cost, it would not yield an efficient solution. Considering the large computational costs associated with convolution modules in CNNs and attention modules in Transformers, a simple and efficient model that consists of Multilayer Perceptrons (MLP) was proposed. This new architecture utilised techniques such as token mixing and channel mixing to capture the relationship between tokens and channels respectively (Guo et al., 2021).

This new architecture was named as MLP-mixer. MLP or Multilayer Perceptron has been around in statistical and machine learning modelling from a very long time but didn’t yield valuable results and hence was pushed back. But with the advent of MLP-mixer, it presented a new direction within computer vision. This new methodology demonstrated that convolution and attention are adequate for good performance, but neither is required. The MLP-mixer is a multilayer perception-specific architecture that combines two different types of layers: one layer where MLPs are applied individually to image patches to "mix" location-specific features, and another layer where MLPs are applied uniformly across image patches to "mix" spatial information. MLP-mixer achieves competitive results on image classification benchmarks when trained on larger datasets and regularisation techniques, with pre-training and inference costs equivalent to SOTA models. The proposed work reached an accuracy of 84.15% (ImNet Top-1 Validation) on ImageNet-21K (Tolstikhin et al., 2021). A projection layer follower by a fully connected layer was proposed in MLP-Mixer, where the images were separated into equal patches and fed through the projection layer. The output from the projection layer was marked

as “Table X”. This table is then put through a channel mixing MLP (analogous to 3x3 convolution) and a token mixing MLP (analogous to 1x1 convolution). A number of these mixer layers will be arranged end to end in the network, together with skip connections, layer normalisation, and GELU- activations. Comparing all the models, MLP-Mixer had the shortest amount of training time. Although the model convergence was sluggish (Kumar and Ramaswamy Karthikeyan, 2021). The MLP-Mixer, developed by Tolstikhin et al., when trained from scratch, achieved a CIFAR10 accuracy of about 80%. Teaming an MLP architecture along with a regularizer to force the model to be close to CNN based networks, Neyshabur achieved 85.19 percent accuracy on CIFAR10 dataset (Lv et al., 2022).

MLP can perform well with modest model sizes, but as the model size increases, it suffers greatly from over-fitting. The author believes that the biggest barrier preventing MLP from obtaining SOTA performance is overfitting (Zhao et al., 2021). MLPs serve as the foundation for the MLP-Mixer architecture. The primary information of the image can be captured by MLP-Mixer by fully combining information from several patches and channels. By disturbing the information mixing mechanism of this architecture, overfitting of source models can be prevented which can further improve transferability of adversarial examples across architectures (Lyu et al., 2022).To improve the new MLP based alternative to Transformers, static parameterization of channel projections and spatial projections was introduced. The research work explored several design options for this architecture and discovered that linear spatial projections along with multiplicative gating perform best. The model was called gMLP since it is composed of basic MLP layers with gating (Liu et al., 2021).

MLP-Mixer have showcased impressive results when trained on the massive dataset JFT-300M, however when trained on a medium-scale dataset like ImageNet-1K, it falls short of its visual Transformer competitors. A gating operation was designed by gMLP (Tang et al., 2022) to improve communications between spatial locations. ResMLP proposed an affine transform layer which assists in stacking a large number of MLP blocks. ResMLP was able to achieve a top-1 accuracy of 98.1% on CIFAR-10 and 87.0% on CIFAR-100(Touvron et al., 2021). Another variant called “Vision Permutator” proposed preserving the original spatial dimensions of the input tokens and the positional information carried by 2D feature representations, in contrast to current MLP based models like MLP_mixer (Tolstikhin et al., 2021) or ResMLP (Touvron et al., 2021), which encode spatial information by flattening the spatial dimensions first and then conducting linear projection along the spatial dimension (Hou et al., 2021).

This research proposal focuses on building a lightweight MLP based architecture that can learn the features from the pigmented skin lesion images. The model shall train on a merged dataset of HAM10000, BCN_20000 and MSK dataset. The results of the classification will be assessed on metrics like accuracy, precision, recall and F-measure.

1.3 Research Questions

The following are some of the research questions that this research proposal attempts to answer.

- How to create a reliable classification model using a relatively new architecture?
- How can we build robust models which do not depend on convolution or attention?
- How well will the model learn features from the input images?
- How stable and fast would the model train without compromising on performance?
- How does the model performance against state-of-the-art CNNs and transformers?

1.4 Aims and Objectives

The main aim of this research is to detect melanoma (skin cancer) from pigmented skin lesion images using modified multilayer perceptron-mixer (MLP) model. The goal is to build an image classification model without leveraging convolution as opposed to conventional CNNs or Transformers while achieving similar performance. Our goal does not take in consideration of outperforming current SOTA models but to demonstrate that an MLP-based model is competitive with convolutional and attention-based models.

The Objectives of the research are listed below:

1. To suggest suitable image pre-processing and transformation techniques to ensure high quality image samples which enhance features and are easier to analyse.
2. To leverage data augmentation techniques to generate plausible samples from existing ones to handle class imbalance in the dataset
3. To determine the model components, parameters and hypermeters along with optimization to train the model.
4. To classify melanoma from the test skin images accurately.
5. To evaluate and analyse the model's performance based on classification metrics like accuracy, precision, recall and F-measure

1.5 Significance of Study

Melanoma has been one of the deadliest skin cancer forms which claims lives each year. With the advent of non-invasive detection techniques like imaging, it is faster to diagnose and treat at early stages. But these imaging techniques involve investing in expensive imaging equipment and equally skilled labor. Deep learning can accelerate detection of melanoma from skin lesion images.

The current state-of the art deep learning models like CNNS and Transformers produce very high accuracy, but it has very high complexity, requires huge dataset, longer and extensive training times and computational cost. CNN models are also slower to train due to operations such as Max Pooling and numbers of layers (Depth). All these drawbacks make it hard to train and deploy a model in a particular application.

This research intends to highlight new and effective MLP-mixer based architectures to perform computer vision tasks such as image classification. Today's image processing networks frequently mix the features between different locations or at a single place. As an illustration, CNNs execute both mixes using convolutions, kernels, and pooling, whereas vision transformers utilize self-attention. Only employing MLPs, MLP-Mixer makes an effort to perform both (mixed features at a single place or mix the features over many locations) in a more "independent" manner. The benefit of employing MLPs, which are essentially just matrix multiplication, is the speed and simplicity of the architecture. Additionally, unlike vision transformers, whose computational complexity is quadratic, the MLP-computational Mixer's complexity is linear as the number of input patches increases. Additionally, the model uses regularization and skip connections. This research intends to provide with a viable alternative to CNNs which provide similar performance with lightweight designs and faster training.

This will benefit the medical diagnostic industry to build robust and cost-effective imaging solutions which can detect melanoma with high precision without requiring expensive imaging devices. This can be considered for primary diagnosis prior to consultation to reduce the treatment initiation time. We also hope this will help spark more interests and further research into MLP for computer vision tasks.

1.6 Scope of Study

The scope of the study of this research work is listed as follows:

- A lightweight image classification model based on MLP-mixer architecture that shall be implemented and trained
- The primary focus is to provide a viable and computationally less expensive alternative to CNNs and Transformers for medical imaging.
- Metrics such as accuracy, precision, recall and F-measure will be used to compare the results with other state-of-the-art models
- The model shall be trained and evaluated on the dataset mentioned in the research methodology section.
- Optimization and modification to outperform state-of-the-art models is not in the scope of this research work

1.7 Structure of Study

The report's format is as follows: Chapter 1 provides a brief overview of the research, a background on the detection of melanoma, and a problem statement. Section 1.3 discusses the aims and objectives, while section 1.4 presents the research questions. Sections 1.5 and 1.6, respectively, discuss the study's significance and scope.

Chapter 2 does a systematic literature review of state-of-the-art CNN and transformer models used in the detection of melanoma. Section 2.1 provides a brief introduction to the topic. Section 2.2 discusses fundamentals of CNN, its working, performance of CNN models on ImageNet Dataset and their drawbacks. Section 2.3 discusses the fundamentals of Transformer models, its working, performance of Transformer models on ImageNet Dataset and their drawbacks. Section 2.4 discusses the related works on melanoma detection using current state of the art CNN and transformer models. Section 2.5 discusses the findings from the papers and section 2.6 provides the summary.

Chapter 3 focusses on the research methodology. Section 3.1 gives an introduction on the methodology followed by question this study aims to answer. The dataset used in this study has been described in section 3.2. The data preprocessing techniques and data augmentation techniques have been described in detail in section 3.3. Section 3.4 illustrates the model architecture and the components of the architecture. fine tuning algorithms and library. Section 3.5 discusses the loss function and optimizers that would be appropriate for this model. Section

3.6 discusses MLP-mixer model's JAX/Flax code implementation from the author's work. Section 3.7 elaborates the experimental approaches and section 3.8 discusses the evaluation of the model. Section 3.9 enlists the required resources and 3.10 summarizes the chapter.

Chapter 4 focusses on the implementation of MLP mixer. Section 4.1 gives a brief introduction to the implementation followed by section 4.2 which discusses the data preparation, pre-processing techniques, and the code implementation. Section 4.3 describes the data splitting method for training and testing followed by code implementation. The MLP mixer model implementation and compilation along with code implementation has been documented in section 4.4 followed by hyper-parameter tuning and its implementation in section 4.5. Section 4.6 discusses the callbacks implementation for the model in keras and section 4.7 formulates the evaluation metrics and their respective code snippets. Section 4.8 briefs the entire chapter.

Chapter 5 discusses the evaluation and performance of the model. Section 5.1 introduces the evaluation followed by section 5.2 that describes the performance of the base model on various metrics. Section 5.3 discusses the performance tuning and section 5.4 shows the best model performance and its parameters. Section 5.5 does a comparison of the model = with state-of-the-art CNN and transformer models and section 5.6 does a discussion on the training, results, and the metrics. Section 5.7 summarizes the chapter.

Chapter 6 discusses the results and the conclusions followed by recommendations.

CHAPTER 2

LITERATURE REVIEW

This section will review the concept of convolutional neural networks and vision transformers, and their use in medical image analysis. We will also review the progress made by CNN (Convolutional Neural Networks) in the detection of skin cancer, the limitations of current research and how we plan to address those limitations through our proposed study.

2.1 Introduction

We assembled various articles for the literature review by searching the Google Scholars database. For our search, we used various keyword combinations like “deep learning”, “skin cancer", "types of skin cancer", "CNN", "transformer", "vision transformer", "skin cancer detection" "skin lesion images", "image enhancements", "medical image analysis" and so forth. To focus on recent work, we chose to collect studies performed between 2018 and 2022. We also went over the other articles that served as the foundation for the recent works to gain a better understanding. After reviewing the abstracts and conclusions of the articles, we decided to exclude a few because they were unrelated to our proposed study. Following the filtering, the remaining articles formed the theoretical foundation of our study. We'll know the answers to the following questions by the end of this section.

- What is convolutional neural network in deep learning? How does it function? What are the most promising deep learning algorithms that have been developed recently?
- What is Vision transformer? How does it reduce dependency on CNN models?
- What drawbacks do CNN and transformer models have? How does MLP mixer address those limitations
- Can MLP-mixer aid in the detection and diagnosis of skin cancer? What datasets are publicly available for skin cancer detection tasks? Compare the most recent skin cancer detection research.

2.2 Convolutional Neural Networks (CNN)

CNNs (Convolutional Neural Networks) are multi-layer neural networks that are primarily designed to recognize patterns in images (O'Shea and Nash, 2015). The CNN models gained prominence through the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for the classification of object categories from millions of images. They were first proposed by (Fukushima, 1988) and advanced through the introduction of backpropagation by (LeCun et al., 1989) for handwritten zip code recognition. It has resulted in the development of numerous powerful CNN models over the last decade (Russakovsky et al., 2014). In recent years, CNN models such as VGGNet, GoogLeNet, and ResNet have improved image classification accuracy, with Top-5 error rates of 7.3%, 6.7%, and 3.6%, respectively (Pak and Kim, 2017)

2.2.1 Architecture CNN models

The image pixel values are stored in the input layers, and the image prediction value is stored in the output layers. In addition to the input and output layers, CNN has three types of layers: convolution layers, pooling layers, and fully connected layers.

- CNN's core building block is the **Convolution Layer**, which performs the majority of the network's computation work. It is made up of a feature detector, which is also known as a kernel/filter. The kernels are a two-dimensional array that represents a matrix. The dimension of the matrix is known as the neuron's receptive field. It generates a "feature map" by only covering a specific region of the input image. The kernel generates feature maps by striding after each operation to cover the entire image. Each of these feature maps is processed by an activation function to generate output for the next layer. The Rectilinear Unit (ReLU) is a popular activation function for the convolution layer (O'Shea & Nash, 2015)

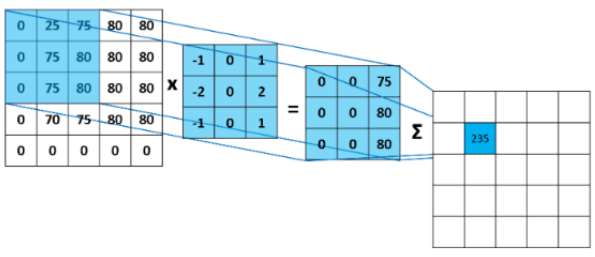


Figure 2.1 Convolution Operation Illustration - Basics (MLNotebook, 2017)

The dimension of the convolution layer's output can be controlled using three hyperparameters. The quantity of kernels/filters: Because each kernel generates a feature map, increasing the number of kernels increases the number of feature maps and, as a result, the depth of the output.

1. Strides are the distances or pixels moved by the kernel after each convolution operation. Low strides cause receptive field overlapping and the generation of more feature maps. It will result in a higher output dimension(O'Shea and Nash, 2015)
2. To fit the kernel into the input image, zero-padding adds zeroes at the border pixel. It enables the inclusion of elements outside the image in the dimensions of feature maps.

- The **Pooling Layer** is similar to a convolution layer in that it uses a max function (known as the Max-pooling layer) or an average function (known as the Average-pooling layer) to obtain the output of a receptive field instead of a convolution operation. This aids in reducing the activation maps and thus the output dimension. The pooling layer reduces network computation and prevents overfitting by reducing the dimensionality of input to the succeeding layer (O'Shea and Nash, 2015).
- Each neuron in the **Fully Connected Layers** is linked to all the neurons in the next layer. This allows the network to learn from all 22 combinations of the previous layer. This layer contributes to the network's feature extraction and classification operations.

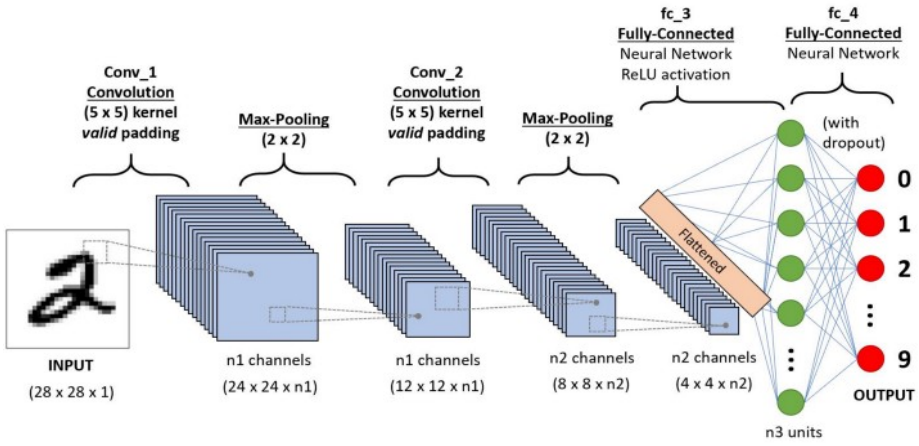


Figure 2.2 An Illustration of Convolutional Neural networks (Sumit Saha, 2018)

2.2.2 Working of CNN models

A CNN model for a specific classification task is created through a training process in which the network learns to recognize and classify image features. The model can also be trained to recognize labels associated with different classes. To create a CNN model, Image Data ingestion, image pre-processing, network training, and classification are all done in a sequence. (Hasan et al., 2019)

1. Image Ingestion: Because CNN networks require a large amount of data for training, a dataset with a large volume of high-quality images is required to create a generalizable model.
2. Image Preprocessing: High-quality images help the network learn features more effectively. Preprocessing images is required to improve image quality, reduce noise, and enhance hidden features. Image filters can be used to enhance various aspects of an image. Data augmentation can be used to increase the image dataset volume to allow the model to learn patterns.
3. Network Training: The gradient descent algorithm is used to train a network through repeated iterations of forward and backpropagation to update the weights and optimize the output error. Transfer learning is a promising technique for lowering network training costs, improving accuracy, and reducing overfitting. Ensemble learning can also aid in network generalization.
4. The network generates a probability score for each output class in the network's final phase. The probability score indicates how likely an image is to belong to a specific class. (Hosny et al., 2018)

2.2.3 Performance of CNN models on ImageNet Challenge

Table 2.1 Performance of CNN models on the ImageNet (ILSVRC) challenge

Model	Depth (Layers)	Parameters	Size (MB)	Top-1 Accuracy	Top-5 Accuracy
Xception	81	22.9M	88	79.0%	94.5%
VGG16	16	138.4M	528	71.3%	90.1%
VGG19	19	143.7M	549	71.3%	90.0%
ResNet50	107	25.6M	98	74.9%	92.1%
ResNet50V2	103	25.6M	98	76.0%	93.0%
ResNet101	209	44.7M	171	76.4%	92.8%
ResNet101V2	205	44.7M	171	77.2%	93.8%
ResNet152	311	60.4M	232	76.6%	93.1%
ResNet152V2	307	60.4M	232	78.0%	94.2%
InceptionV3	189	23.9M	92	77.9%	93.7%
InceptionResNetV2	449	55.9M	215	80.3%	95.3%
MobileNet	55	4.3M	16	70.4%	89.5%
MobileNetV2	105	3.5M	14	71.3%	90.1%
DenseNet121	242	8.1M	33	75.0%	92.3%
DenseNet169	338	14.3M	57	76.2%	93.2%
DenseNet201	402	20.2M	80	77.3%	93.6%
NASNetMobile	389	5.3M	23	74.4%	91.9%
NASNetLarge	533	88.9M	343	82.5%	96.0%
EfficientNetB0	132	5.3M	29	77.1%	93.3%
EfficientNetB1	186	7.9M	31	79.1%	94.4%
EfficientNetB2	186	9.2M	36	80.1%	94.9%
EfficientNetB3	210	12.3M	48	81.6%	95.7%
EfficientNetB4	258	19.5M	75	82.9%	96.4%
EfficientNetB5	312	30.6M	118	83.6%	96.7%
EfficientNetB6	360	43.3M	166	84.0%	96.8%
EfficientNetB7	438	66.7M	256	84.3%	97.0%

2.2.4 Challenges of CNN

Despite CNNs being powerful predictors, they too suffer from drawbacks which limits their potential. The limitations of CNN models are described as follows:

1. Translational Invariance

   They fail to encode object position and orientation. They have difficulty classifying images in different positions. This limitation is attributed to the use of stride, which disregards the sampling theorem, and fully connected layers, which lack spatial reasoning. (Mouton et al., 2021)

2. Generalization-Translation Invariance tradeoff

   In the case of pooling kernel size, there is a trade-off between generalization and translation invariance, with larger kernel sizes resulting in better invariance but poorer generalization. (Mouton et al., 2021)

3. Longer Training Time

   Since CNNs have high computational operations, they tend to get slower. Factors like increase in depth/number of layers increase the training time and the model converges slower. When working with a large dataset through a pipeline that includes aggressive image augmentation, even performing adequate transfer learning on a pre-trained model such as VGG16 or ResNet can take over an hour per epoch. (Logan Joe, 2018)

4. Expensive Hardware

   CNNs require intensive computation resources like GPU – graphics processing Unit, high-capacity memory to perform faster along with high-speed CPU. The above listed CNN models were trained on state-of-the-art computing systems with data center GPUs like Nvidia A100 and V100 which are extremely expensive.

2.3 Vision Transformer Model

The Vision Transformer model came into picture due to the popularity of Transformers models in Natural Language Processing (NLP) (Vaswani et al., 2017). While Transformers became de facto standard in NLP, there were limited applications to computer vision tasks. Attention is used in vision either in conjunction with convolutional networks or to replace specific components of convolutional networks while maintaining their overall structure. The Vision Transformer model showed that a pure transformer model applied directly to sequence of patches of image can perform well on classification tasks. (Dosovitskiy et al., 2020)

2.3.1 Architecture of Vision Transformer

Because it was designed for NLP, the standard Transformer model received a one-dimensional sequence of word embeddings as input. When applied to the task of image classification in computer vision, the Transformer model receives input data in the form of two-dimensional images. Transformer models are based on the concept of self-attention.

- **Linear Projection of Flattened Patches**

  The input image is cut up into smaller two-dimensional patches to structure the input image data in a way that resembles how the input is structured in the NLP domain (in the sense of having a sequence of individual words). These yields $\frac{HW}{P^2}$ patches also known as visual tokens, with each patch having a resolution of (P, P) pixels. (Dosovitskiy et al., 2020)

- **Transformer Encoder**

  The encoder is made up of several blocks, each of which contains three major processing elements: Layer Norm, Multi-head Attention Network (MHAN), and Multi-Layer Perceptrons (MLP). Layer Norm keeps the training process on track while allowing the model to adapt to variations in the training images. MHAN is a network that generates attention maps based on embedded visual tokens. These attention maps assist the network in focusing on the most important regions of the image, such as object (s). Attention maps are similar to the concept found in traditional computer vision literature (e.g., saliency maps and alpha matting).

- **MLP Classifier Head**

  The MLP is a two-layer classification network that ends with a GELU (Gaussian Error Linear Unit). The final MLP block, also known as the MLP head, serves as the transformer's output. A SoftMax application on this output can provide classification labels (if the task is Image Classification

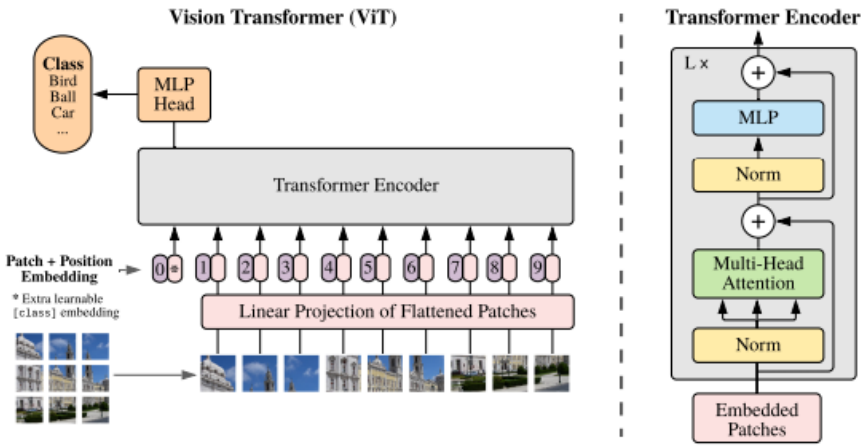


Figure 2.3 Illustration of Vision Transformer Architecture (Dosovitskiy et al., 2020)

2.3.2 Working of Visual Transformer

The Vision Transformer's first layer linearly projects the flattened patches into a lower-dimensional space. The top principal components of the learned embedding filters are depicted in the Figure 2.3. The components have the appearance of plausible basis functions for a low-dimensional representation of the fine structure within each patch.

Following the projection, the patch representations are enhanced with a learned position embedding. Figure 2.3 (center) demonstrates how the model learns to encode distance within the image in the similarity of position embeddings, with closer patches having more similar position embeddings. The row-column structure is also visible; patches in the same row/column have similar embeddings. Finally, for larger grids, a sinusoidal structure is sometimes visible. Because position embeddings learn to represent 2D image topology, hand-crafted 2D-aware embedding variants do not improve.

The Transformer Encoder outputs are then fed into a Multilayer Perceptron for image classification. The input features accurately capture the essence of the image, making the MLP

head's classification task much easier. The Transformer has several outputs. Only the one associated with the special (class) embedding is fed into the classification head; the others are ignored. As expected, the MLP produces a probability distribution of the classes to which the image could belong.

Figure 2.4 Examples of attention on input images.(Dosovitskiy et al., 2020)

2.3.3 Performance of Transformer models in ImageNet Challenge

Table 2.2 Performance of Transformer models on the ImageNet (ILSVRC) challenge

Model Name	Model	Params (M)	Top-1 Accuracy
Data efficient image Transformer	DeiT-Ti	5	72.2%
Data efficient image Transformer	DeiT-S	22	79.8%
Data efficient image Transformer	DeiT-B	86	81.8%
Tokens-To-Token Vision Transformer	T2T-ViT-14	21.5	81.5%
Tokens-To-Token Vision Transformer	T2T-ViT-19	39.2	81.9%
Tokens-To-Token Vision Transformer	T2T-ViT-24	64.1	82.3%
Pyramid Vision Transformer	PVT-Small	24.5	79.8%
Pyramid Vision Transformer	PVT-Medium	44.2	81.2%
Pyramid Vision Transformer	PVT-Large	61.4	81.7%
Turbulence Neural Transformer	TNT-S	23.8	81.5%
Turbulence Neural Transformer	TNT-B	65.6	82.9%
Conditional Positional Encoding Vision Transformers	CPVT-S	23	80.5%
Conditional Positional Encoding Vision Transformers	CPVT-B	88	82.3%
Model Name	Model	Params (M)	Top-1 Accuracy
Swin Transformer	Swin-T	29	81.3%

Swin Transformer	Swin-S	50	83.0%
Swin Transformer	Swin-B	88	83.3%

2.3.4 Challenges of Image Transformer Models

Transformers rely on the concept of self-attention unlike CNNs which rely on convolution operations. Despite being state-of-the-art models, they suffer from limitations. These limitations are described as follows:

1. Pretraining

   Transformers are generally pre-trained utilizing tasks on large-scale picture datasets since they require minimal existing information about the structure of the issue. (Khan et al., 2021)

2. Data Hungry

   Transformers are models that require a lot of data. The fundamental explanation stems from the concept of inductive bias. However, if Transformers can be fed a large quantity of data, the global method will produce greater outcomes.

3. High Computation Cost

   An empirical study on the scalability of Vision Transformers demonstrated that scaling up on compute, model, and size of training samples improves performance. Only big models (with more parameters) gain from more training data; smaller models plateau rapidly and cannot profit from new data. Hence scaling up transformers is computationally expensive. (Zhai et al., 2021)

4. Interpretation

   Despite several pioneering research, visualizing and understanding Transformers remains an unresolved challenge, and ways to produce spatially accurate activation-specific representations are required. (Khan et al., 2021)

2.4 Related Works

We investigated recent research trends in Melanoma (skin cancer) detection in order to comprehend the methodology employed and assess the significance of these methods. Such research works are summarized below.

Table 2.3 Performance of CNN models on skin lesion classification task.

Study	Methodology Used	Type of Classification Problem	Dataset	Performance
(Esteva et al., 2017)	Inception V3 with pretrained weights (ImageNet)	Binary (Melanoma / Non-Melanoma)	Combination of the ISIC, Dermofit Library22 and data from the Stanford Hospital	Accuracy: 72.1% AUC: 0.91
(Haenssle et al., 2018)	Google's Inception v4 CNN architecture with pretrained weights (ImageNet)	Binary (Melanoma / Melanocytic nevi)	ISIC2017	Sensitivity: 95%, specificity: 80%, and ROC AUC: 0.95
(Marchetti et al., 2018)	Ensemble of CNN models	Binary (Melanoma / Melanocytic nevi)	ISIC2017	Sensitivity: 58%, specificity: 88%,
(Bi et al., 2017)	ResNet with pretrained weights (ImageNet)	Binary (Melanoma / Non-Melanoma)	ISIC 2017	AUC: 0.915 (Melanoma)
(Romero-Lopez et al., 2017)	VGGNet with pretrained weights (ImageNet)	Binary (Benign/ Malignant)	ISIC 2017	Sensitivity:78.66%

(Pham et al., 2020)	InceptionV3 network with customized fully connected layers	Binary (Melanoma / Melanocytic nevi)	ISIC 2019	AUC : 0.943
(Ali et al., 2021)	Custom CNN 4 conv layers and one fully connected layer with data augmentation	Binary (Benign/ Malignant)	HAM10000	Accuracy: 90.16%, Precision: 94.63, Recall:93.91, F1 score: 92.69
(Ech-Cherif et al., 2019)	MobileNetV2 CNN Model with transfer learning	Binary (Benign/ Malignant)	DermNet, ISIC and Dermofit	Accuracy: 91.33
(ABAYOMI-ALLI et al., 2021)	SqueezeNet model with data augmentation	Binary (Benign/ Malignant)	P H2	accuracy (92.18%), sensitivity (80.77%), specificity (95.1%), and F1-score (80.84%).

Table 2.4 Performance of Transformer models on skin lesion classification task

Study	Methodology Used	Type of Classification Problem	Dataset	Performance
(Xin et al., 2022)	Vision Transformer model with Data Augmentation	Multiclass	HAM 10000 dataset	Accuracy: 94.1%
(He et al., 2022)	Fully Transformer Network with spatial pyramid transformer	Multiclass	ISIC 2018	Accuracy: 89.6% Sensitivity: 85.7
(Chen et al., 2021)	Vision Transformer model along with convolution layers to segment and classify images	Binary (Melanoma/seborrheic keratosis)	ISIC 2017	Accuracy: 89.2% Sensitivity: 68.0

(Pedro and Oliveira, 2022)	Vision Transformer [Best Model] pretrained on ImageNet	Multiclass	HAM10000 dataset	Accuracy: 73.7%
(Zhou and Luo, 2021)	Mutual Attention Transformer [Novel method]	Multiclass	HAM10000	Accuracy: 92.5%
(Xie et al., 2021)	Swin Transformer Pretrained on ImageNet-1K and the parameter-free attention module SimAM along with data augmentation	Binary (Melanoma/Non-Melanoma)	ISIC 2017	Precision: 74.0% and AUC:0.900

2.5 Discussions

From the Literature review, we learned that Convolutional Neural Networks have been the de facto standard for skin cancer classification. Although modern self-attention-based models like Vision Transformers have made significant progress in the skin cancer detection space.

Most studies and novel models have been done on the Human Against Machine (HAM) 10000 dataset and the ISIC dataset which is a reliable public data source for skin lesions and their accurate labels (diagnostic) (Tschandl et al., 2018). Another noticeable pattern is the usage of high computation resources in the model training. A lot of models in the above studies have been trained GPUs like V100, A100 while some on GTX Titan. These GPUs are expensive computation cost to train these models. Also, the training time plays a crucial role which depends on the hardware and data pre-processing in the generalization of the model.

Though CNNs and transformers show remarkable results on the classification tasks, the cost associated with training and inference is very high. Most of the studies have missed out on creating faster and less resource hungry models without losing performance which can compete with state-of-the-art models. Hence, we need a model which requires less computational cost, lesser training time without compromising much on the performance. In our study, we look forward to implementing modern image classification models which are computationally less expensive and faster to train which results in faster deployment.

2.6 Summary

In this section, we have discussed the various CNN and transformer models which have state-of the art performance along with their challenges. We have also discussed their performance on melanoma detection and their related discussion. This review has given us a background on the current techniques being used.

CHAPTER 3

RESEARCH METHODOLOGY

In this section, we will go over the various measures we'll take to create a more efficient and generalizable TB detection model. The following are the questions that we want to answer in this area.

- What public dataset would be used to train and test the model? How do we treat issues with data like image quality, class balance and scoring?
- How will we pre-process images and build a data pipeline for the model training and evaluation? Do we need data Augmentation techniques.
- What models will be used for the skin lesion classification task?
- What architecture and the components will the model employ?

3.1 Introduction

Previous studies have focused on building CNN and transformer models for classification of skin lesion images. These models have proved their prediction power and have shown remarkable results. These models also come with caveats like high computational cost, heavy data augmentations and high consumption of data. Hence, we intend to propose a lightweight model that trains faster and uses less computational resources without compromising much on performance and prediction power. This model could help reduce the over training time and deployment while achieving good results.

While exploring options for choosing the right model, we set our criteria as follows:

1. A lightweight design and architecture that does not rely on convolution or attention mechanism
2. Relatively faster to train which takes less training time and generalizes well.
3. Uses less computational resources like Memory and GPU
4. Performance like CNN and transformer models but not expecting to outperform them.

In our selection of the model, we decided to choose MLP-mixer model, an architecture that is entirely based on multi-layer perceptrons (MLPs). MLP-Mixer has two types of layers: one with MLPs applied to image patches separately (i.e., "mixing" per-location characteristics) and one

with MLPs applied across patches (i.e., "mixing" spatial information). MLP-Mixer achieves competitive performance on image classification benchmarks when trained on big datasets or with contemporary regularization algorithms, with pre-training and inference costs like state-of-the-art models (Tolstikhin et al., 2021).

Since MLP mixer was a relatively new architecture, the native study employed a custom model written in JAX/Flax and pre-trained on two public datasets: ILSVRC2021 ImageNet, and ImageNet-21k, a superset of ILSVRC2012 that has 21k classes and 14M images (Deng et al., 2009). To assess performance at larger scale, the model was trained on JFT-300M, a proprietary dataset which has 300M examples and 18k classes (Sun et al., 2017). We had limited options to port the original model and weights written in JAX/Flax to TensorFlow compatible code, hence we decided to implement the architecture in TensorFlow and train the model from scratch. Since the model has features like faster training and convergence, training from scratch was easier and hassle free. This also helps us understand the model's effectiveness on learning a task from scratch without any pre-training.

3.2 Dataset Description

For our study, we have chosen HAM10000 ("Human Against Machine with 10000 training images") dataset (Tschandl et al., 2018). The dataset contains 10015 dermatoscopic pictures with a resolution above 512 x 512 that may be used as a training set for academic machine learning. Cases include a representative collection of all important diagnostic categories within realm of pigmented lesions:

- Actinic keratoses and intraepithelial carcinoma / Bowen's disease (**akiec**),
- basal cell carcinoma (**bcc**),
- benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, **bkl**),
- dermatofibroma (**df**),
- melanoma (**mel**),
- melanocytic nevi (**nv**)
- and vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, **vasc**).

More than 50% of lesions are verified by histopathology (histo); the balance of the cases has been confirmed by follow-up examination (follow up), expert consensus (consensus), or in-vivo

confocal microscopy (confocal). The dataset contains lesions with many pictures, which may be tracked using the HAM10000 metadata file's ‘lesion id’ field. The dataset has been modified to convert the multiclass dataset into a binary classification task.

This dataset has been utilized widely on various CNN models to showcase the model’s predictive powers and high accuracy. To ensure a fair comparison with CNN and transformer models, we decided to train our proposed model’s performance on the same dataset, hence have been resized to 100 x 100 for our study. The dataset is split into 70:20:10 for training, test and validation.

Table 3.1 Distribution of images across the classes in the dataset

Dataset Name	Skin Lesion Type short name	Skin Lesion Type Name	Count of Images
HAM10000	akiec	Actinic keratoses and intraepithelial carcinoma / Bowen's disease (**akiec**),	327
	bcc	basal cell carcinoma (**bcc**),	514
	bkl	benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, **bkl**),	1099
	df	dermatofibroma (**df**),	115
	mel	melanoma (**mel**),	1113
	nv	melanocytic nevi (**nv**)	6705
	vasc	and vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, **vasc**).	142
	Total Images		10015

3.3 Data Pre-processing

Data pre-processing is required to improve the image characteristics, generate more images for training and regularization and increase model learning's focus on the targeted region.

3.3.1 Grouping Data

The original dataset has 10015 images belonging to 7 classes. We regrouped the data to have two classes of data (Binary classification) viz Melanoma and Non-Melanoma. The group "Melanoma" contains images from the class Melanoma (*mel*). Binning the other groups into non-melanoma had a different approach. We considered Melanocytic nevi (*nv*), benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, *bkl*), dermatofibroma (*df*) and vascular lesions as non-melanoma since they are non-cancerous in nature(Luba et al., 2003). The rest two classes namely (basal cell carcinoma (bcc) and Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec) are either cancerous or precancer in nature and have not been considered (National Health Service for England, 2020)

Table 3.2 New Image distribution after data regroup

Class	Count of Images
Melanoma – Class 1	1113
Non-Melanoma – Class 0	8061

3.3.2 Image Normalization

Normalization is a technique used in image processing to modify the range of pixel intensity values. Examples of applications would be photos with weak contrast from glare. Normalization is often referred to as histogram stretching or contrast stretching. Normalization helps in smoothing out intensities and enables the model to learn features from the model in a better way. We have applied a normalization logic to convert intensities of the original image to the range [0,255]. The norm used is "Min max normalization" which works by performs a linear transformation on the original data. This technique gets the scaled data in the desired range. Mathematically It can be expressed as:

$$New\ (I_n) = \left(\frac{I - I_{min}}{I_{max} - I_{min}}\right) * I_{New\_max} \quad (3.1)$$

Where I_n = Intensity of nth pixel of Image

I_{min} = Minimum value of intensity of the entire Image

I_{max} = Max value of intensity of the entire Image

I_{New_max} = Max value of Intensity of the desired range

$New\ (I_n)$ = New Intensity of nth pixel of Image

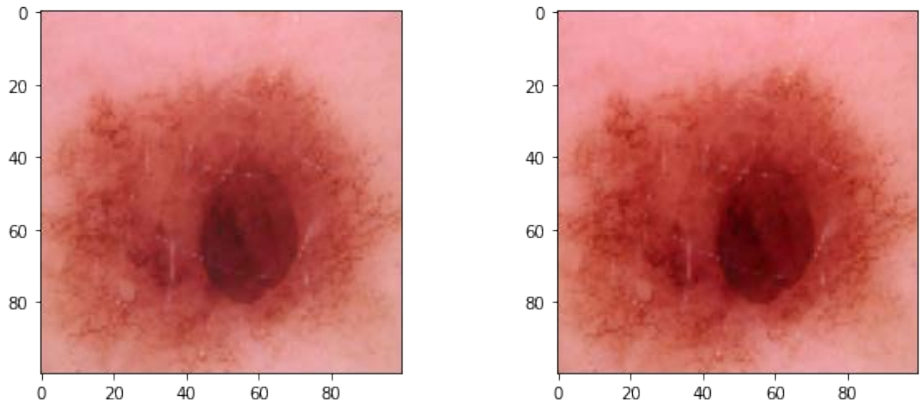


Figure 3.1 Skin image before Normalization (left) and after normalization (right)

3.3.3 Class balance and Data Augmentation

A situation known as class imbalance occurs when there are significantly fewer observations in one class than in the other classes. This issue is prevalent in situations where anomaly detection is essential, such as electricity theft, bank fraud, the detection of rare diseases, etc. The prediction model created using traditional machine learning methods in this case may be unreliable and biased. Hence, we need to rebalance class to ensure our model is bias free. In our case, the ratio for Melanoma to Non-melanoma is roughly 1:7 (1113 / 8061 = 7.24). So, we need to sample 7 images for each image that represents “melanoma”, this is also known as “Over-sampling (Ando and Huang, 2017). By creating additional data points from existing data, a group of techniques known as "data augmentation" can be used to artificially enhance the amount of data. This includes making minor adjustments to the data or creating new data points using Image processing libraries or deep learning models. We have used augmentations that alter the image intensity, color, shift the pixels and hue-saturation change. We also added gaussian noise, equalize, sharpen and random rotation (Limit to 70°) to the images.

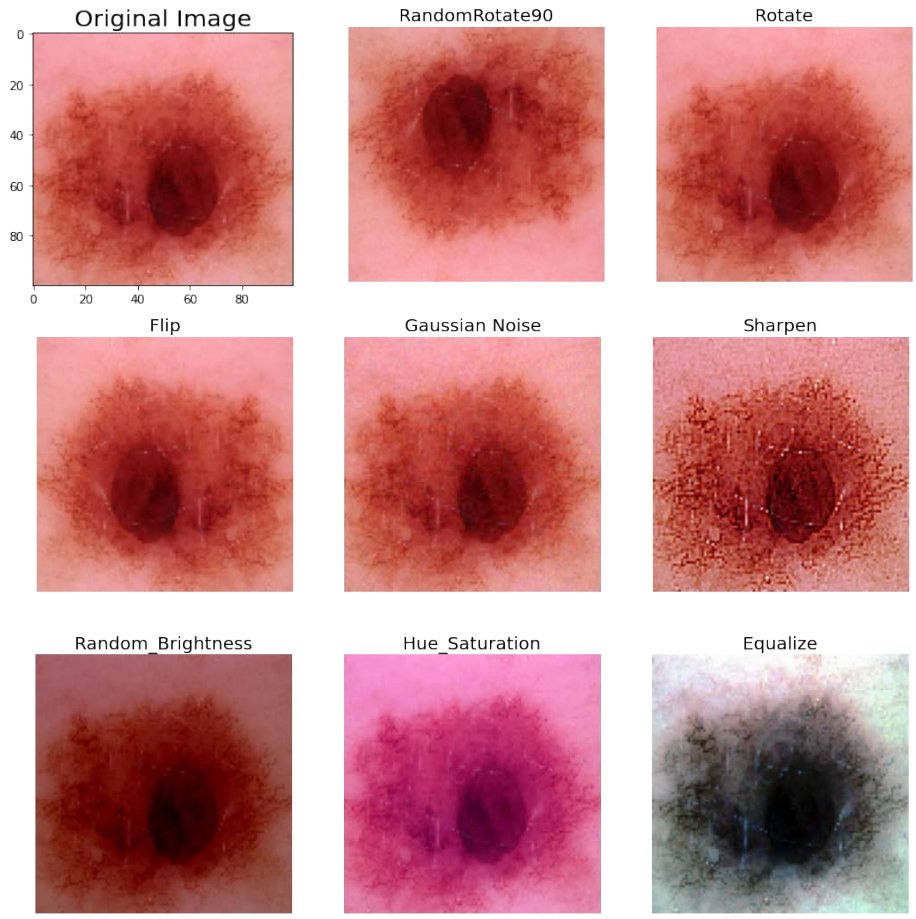


Figure 3.2 Different Image Augmentation applied to training image.

3.4 Model Architecture

As the model is relatively new with implementation in JAX/Flax, we have built the model in TensorFlow from available Keras documentation. Trained the model from scratch and then infer the results. We have also fine-tuned hyperparameters to get the best performance out of the model. The proposed architecture takes input in form of patches which is fed to a Mixer layer, the fully connected layers with GELU activations and a linear classification head. The architecture also uses skip-connections and regularization like dropout and LayerNorm. The MLP-Mixer architecture consists of three main components:

1. Per-patch linear embedding.
2. Mixer layers.
3. Classification head.

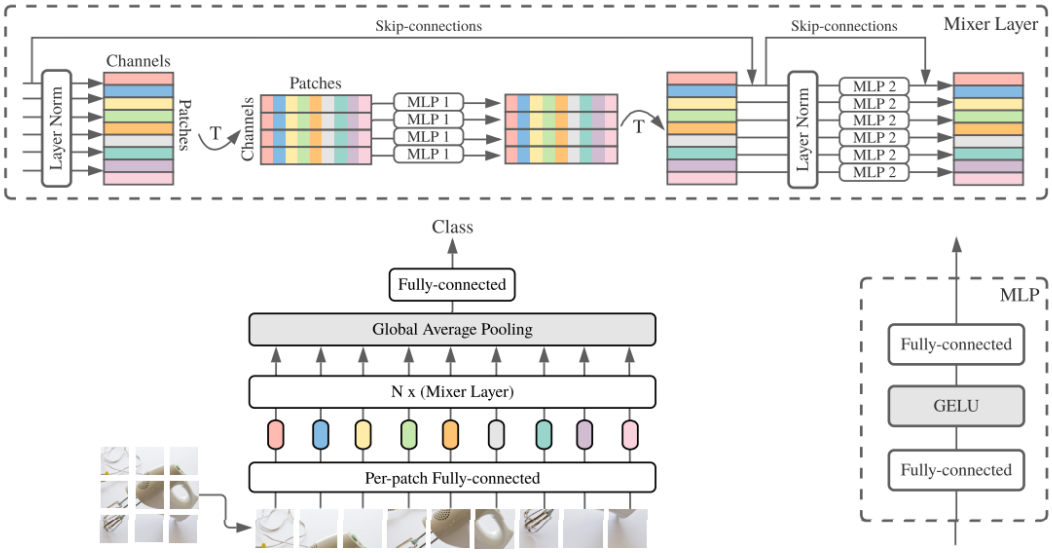


Figure 3.3 MLP-mixer architecture

3.4.1 Per-patch linear embedding

The input images with the dimensions (H, W) are split into a series of S separate, non-overlapping image patches in this layer. Every patch has dimensions (P, P). Thus, it divides the input image's dimension by the size of each patch to determine the number of patches needed for each image such that $S = \frac{HW}{P2}$. Through a projection layer, these patches are projected linearly. Simple fully linked layer with output dimension C describes this projection layer. A 2D input table containing picture patches (tokens) labelled X on the architecture is the outcome which is used on the mixer layers. One point to notice here is that the H value is independent of the number of patches or patch sizes that enables the network to grow linearly rather than quadratically. This results in reduced process parameters and a higher throughput of 120 images/sec/core which is roughly 3 times than ViT's 32 images/sec/core as per the author.

3.4.2 Mixer Layers

Two different MLP layer types are present in the mixer layer. It's vital to understand that CNN inspired the concept for these two layers.

1. Channel-mixing

   This MLP enables communication across various channels. It uses individual rows of the table as inputs and performs independent operations on each place. It only processes the channels after processing a single patch of the image.

2. Token Mixing

   This MLP helps communication across several spatial positions (or tokens) within an image. An important point to note is that the picture patches are the tokens. It operates on each channel individually and takes independent columns of the table as inputs.

As we see at the bottom of the architecture, the input to the network is patches of images. These patches are projected linearly into an H-dimension latent representation (which is hidden) and passed on to the Mixer layer. Referring to the top part of the architecture which shows the Mixer layer, the patches are converted to into a table form, let's call that X, when projected to the mixer layer.

The mixer layers can be mathematically defined as:

$$U_{*,i} = X_{*,i} + W_2\sigma\left(W_1 LayerNorm(X)_{*,i}\right), \quad for\ i = 1 \dots C \tag{3.2}$$

$$Y_{*,i} = \text{Uj,}* + W_4\sigma\left(W_3 LayerNorm(U)_{j,*}\right), \quad for\ j = 1 \dots S \tag{3.3}$$

Where X = columns of linear projection table

S= input sequence of non-overlapping image patches

W_n= weights of layer n

C= hidden dimension

σ = element wise GELU activation

Also, the architecture uses Layer Norm which is often used in Transformer architectures. Below image shows Layer Norm vs Batch Norm. There are skip connections included with the GELU activation for non-linearity and dropout for regularization.

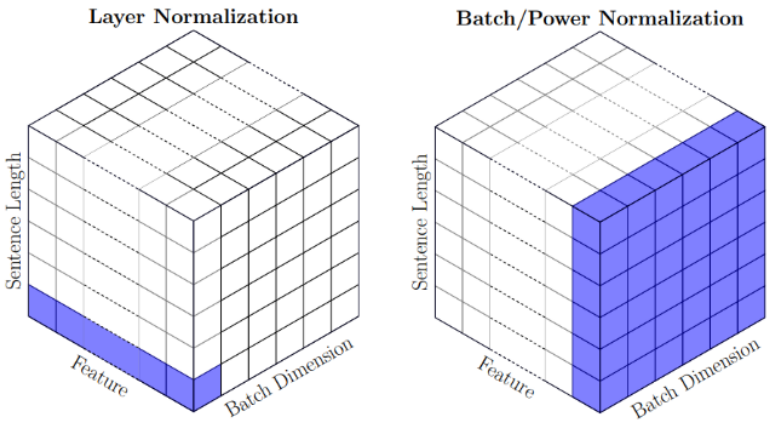


Figure 3.4 Layer Normalization (left) and Batch/Power Normalization (right) (Shen et al., 2020)

3.4.3 Classification head

This is the last layer in the MLP-Mixer architecture. It consists of a standard classification head, a global average pooling layer, and a linear classifier. The output can be a dense layer with a sigmoid layer (Binary classification) or a dense layer with SoftMax output (Multiclass Classification). The head also has a dropout layer to tackle overfitting

3.4.4 Loss Function

A loss function analyses how effectively the neural network represents the training data by comparing the target and anticipated output values. We try to reduce this difference in output between the expected and the goal during training. There are different types of loss function depending on the type of task. Since we are performing a binary classification, we will focus on Binary Cross Entropy

Binary Cross Entropy

In order to determine which of the predetermined categories the provided input will fall under, classification neural networks first output a vector of probabilities, and then they choose the category with the highest probability as the final output. There are only two real values of y that may be classified as valid in binary classification: 0 or 1. It is necessary to compare the actual value (0 or 1) with the likelihood that the input falls into that category

- p(i) = probability that the category is 1
- 1-p(i) = probability that the category is 0
  in order to appropriately calculate the loss between the actual and anticipated values.

Binary Cross Entropy can be defined as:

$$CE = \frac{1}{n}\sum_{i=1}^{N}(y_i * \log(p_i)) + ((1 - y_i) * \log(1 - p_i))$$

3.4.5 Optimizer

We must adjust the weights for each epoch during deep learning model training and reduce the loss function. An optimizer is a procedure or method that alters neural network properties like weights and learning rates. As a result, it aids in decreasing total loss and raising precision. A

deep learning model often has millions of parameters, making the process of selecting the proper weights for the model more challenging.

AdamW Optimizer

AdamW is an enhanced variant of Adam in which weight decay is only implemented after adjusting the parameter-wise step size. The weight decay or regularisation term is only inversely proportionate to the weight itself because it does not enter the moving averages. AdamW has demonstrated better training loss and that models trained with it generalise far better than models trained with Adam, enabling the new version to compete with stochastic gradient descent with momentum (Loshchilov and Hutter, 2017)

Algorithm 2 Adam with L_2 regularization and Adam with weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{x}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{x}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{x}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{x}_{t-1}) + w\boldsymbol{x}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + w\boldsymbol{x}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{x}_t$

Figure 3.5 AdamW optimizer algorithm steps (Loshchilov and Hutter, 2017)

3.5 Implementing the MLP-mixer model

Before implementing the model in TensorFlow, we studied the author's MLP mixer code written in JAX/Flax. This formed our base to look back and validate our implementation.

There are three sections to the model design.

1. MLP block

   The MLP block contains 2 dense layers with a GELU activation between the layers.

```
class MlpBlock(nn.Module):
  mlp_dim: int
  @nn.compact
  def __call__(self, x):
    y = nn.Dense(self.mlp_dim)(x)
    y = nn.gelu(y)
    return nn.Dense(x.shape[-1])(y)
```

Figure 3.6 JAX/Flax implementation of MLP block

2. Mixer Block

   The Mixer block handles mixing the information received from each MLP and returns them to classifier head. The mixer layers consist of 2 MLP blocks namely, token mixing and channel mixing.

```
class MixerBlock(nn.Module):
  tokens_mlp_dim: int
  channels_mlp_dim: int
  @nn.compact
  def __call__(self, x):
    y = nn.LayerNorm()(x)
    y = jnp.swapaxes(y, 1, 2)
    y = MlpBlock(self.tokens_mlp_dim, name='token_mixing')(y)
    y = jnp.swapaxes(y, 1, 2)
    x = x+y
    y = nn.LayerNorm()(x)
    return x+MlpBlock(self.channels_mlp_dim, name='channel_mixing')(y)
```

Figure 3.7 JAX/Flax implementation of Mixer block

3. MLP-Mixer model compilation along with classifier head

   This is the final block where the MLP blocks, and the mixer are integrated following a classifier head which is customized as per the classification task (Binary/Multiclass)

```
class MlpMixer(nn.Module):
  num_classes: int
  num_blocks: int
  patch_size: int
  hidden_dim: int
  tokens_mlp_dim: int
  channels_mlp_dim: int
  @nn.compact
  def __call__(self, x):
    s = self.patch_size
    x = nn.Conv(self.hidden_dim, (s,s), strides=(s,s), name='stem')(x)
    x = einops.rearrange(x, 'n h w c -> n (h w) c')
    for _ in range(self.num_blocks):
      x = MixerBlock(self.tokens_mlp_dim, self.channels_mlp_dim)(x)
    x = nn.LayerNorm(name='pre_head_layer_norm')(x)
    x = jnp.mean(x, axis=1)
    return nn.Dense(self.num_classes, name='head',
                    kernel_init=nn.initializers.zeros)(x)
```

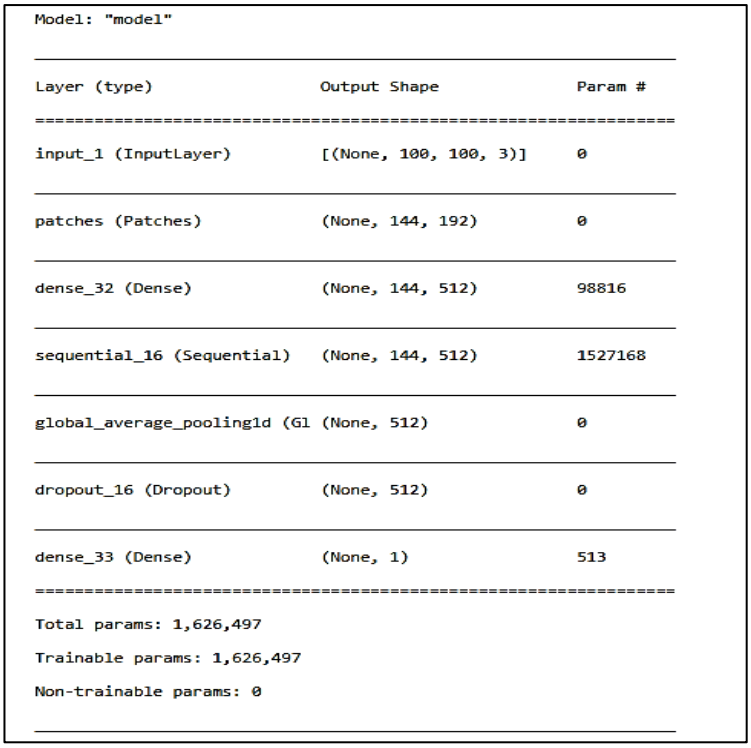
Figure 3.8 JAX/Flax implementation of MLP-Mixer module with classifier head layer

3.5.1 Fine Tuning the MLP-mixer model

Following Steps were performed to fine tune the model

1. Build the Patch extraction layer to convert the images to patches
2. Build the MLP mixer module with the 2 MLP blocks
3. Add a GlobalAveragePooling to flatten the output into 1-Dimension from MLP mixer layer in step 2
4. Add a Dropout Layer for regularization of the network
5. Add a Dense neuron with Sigmoid Layer activation for our Binary classification task
6. Train the layers from scratch along with callbacks (Early Stopping and Learning Rate Scheduler) along with a low learning rate.

Figure 3.9 Steps to fine-tune and train the MLP-mixer model



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100, 100, 3)]     0
_________________________________________________________________
patches (Patches)            (None, 144, 192)          0
_________________________________________________________________
dense_32 (Dense)             (None, 144, 512)          98816
_________________________________________________________________
sequential_16 (Sequential)   (None, 144, 512)          1527168
_________________________________________________________________
global_average_pooling1d (Gl (None, 512)               0
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_33 (Dense)             (None, 1)                 513
=================================================================
Total params: 1,626,497
Trainable params: 1,626,497
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.10 Parameters of implemented MLP mixer model

The MLP mixer model implemented has around 1.6M parameters which is lesser than CNN models and Transformer models.

3.6 Experimental Approaches

For achieving best results and performance from the model, we have experimented with various parameters of the model. The parameters include changing the patch size, number of blocks, hidden units/embedding dimension for the MLPs, dropout rate and the learning rate. We have also fine-tuned the parameters with Keras Tuner – Hyperband Algorithm to try combinations of hyperparameters and select the optimum values. The Hyperband approach allocates a preset resource, such as iterations, data samples, or features, to randomly chosen configurations in a non-stochastic, infinite-armed bandit problem (Li et al., 2016)

3.6.1 Keras Tuner

The challenges of hyperparameter search are addressed by Keras Tuner, an intuitive and scalable framework for hyperparameter optimization. Utilize one of the search algorithms offered to quickly create search space and find the ideal hyperparameter values for the models. In addition

to having the algorithms for Bayesian Optimization, Hyperband, and Random Search built-in, Keras Tuner is also made to be simple for researchers to enhance to test out novel search strategies (O'Malley et al., 2019).

3.7 Evaluation

A confusion matrix summarizes how well the model performed on unobserved data. This matrix can be used to create a number of attributes that can be used to compare the effectiveness of any model for a given classification task.

Table 3.3 Confusion Matrix

	Actual Negative (0)	Actual Positive (1)
Predicted Negative (0)	True Negative (TN)	False Negative (FN)
Predicted Positive (1)	False Positive (FP)	True Positive (TP)

The following performance metrics have been derived from the Confusion Matrix

1. Accuracy

   It counts the number of correctly identified observations, both positive and negative.

   $$Accuracy = \frac{T_P+T_N}{T_P+T_N+F_P+F_N} \quad (3.4)$$

2. Precision

   It counts the number of observations that match as positive predictions.

   $$Precision = \frac{T_P}{T_P+F_P} \quad (3.5)$$

3. Recall or Sensitivity

   It measures the proportion of observations that we have categorized as positive out of all observations that are positive.

   $$Recall\ or\ Sensitivity = \frac{T_P}{T_P+F_N} \quad (3.6)$$

4. F1- Score

   It is the harmonic mean between recall and precision.

$$F1\ score = \ 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.7}$$

5. ROC AUC score

   We can calculate the Area Under the ROC Curve, or ROC AUC score, to get a single number that tells us how good our curve is. The more top-left the curve, higher the area and thus the higher the ROC AUC score.

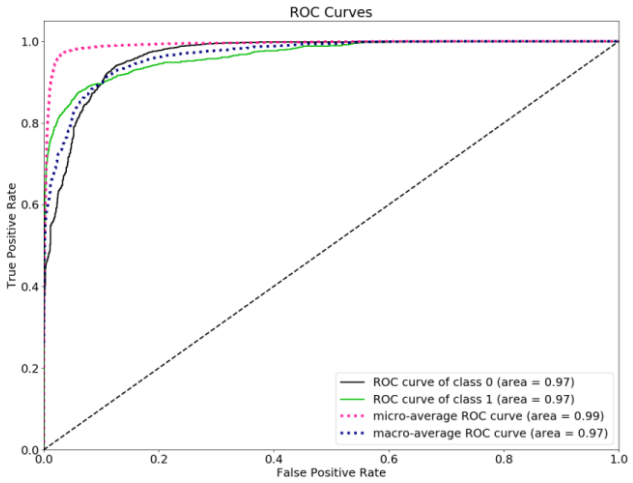


Figure 3.11 Representation of ROC AUC Curve

3.8 Resources Requirements

Our Experiment was performed using the following resources.

Table 3.4 System resources used for the study

Device Name	Kaggle Notebook
CPU	Intel(R) Xeon(R) CPU @ 2.00GHz
RAM	13GB
GPU	Nvidia Tesla T4 / 16GB
Storage	30GB
Libraries	OpenCV (cv2), Pandas, Numpy, Albumentations, TensorFlow, Keras and TensorFlow Addons, sklearn and Keras tuner

3.9 Summary

This section describes the architecture of the MLP mixer model along with the data processing. It discusses Data augmentation and class balance as well. The model components and the JAX/Flax implementation by author have been studied. Training the model requires the right selection of loss function and optimizer and this section helped in picking the right ones. Apart from that, the experimental approaches and evaluation metrics have been outlined. The required resources have been listed to ensure the study has no roadblocks.

CHAPTER 4

IMPLEMENTATION

4.1 Introduction

To speed up the process of detecting melanoma from skin lesion images, an early, reliable, and cost-effective method of detection has become necessary. To enable such a process, we intend to automate melanoma detection using deep learning models that can be trained using low-cost computation resources. On our dataset, we followed the process of data preparation, image processing, data augmentation, and model fine-tuning to create the model proposed in our research. To create the proposed work, we used the libraries OpenCV, Keras, Albumentations, and TensorFlow.

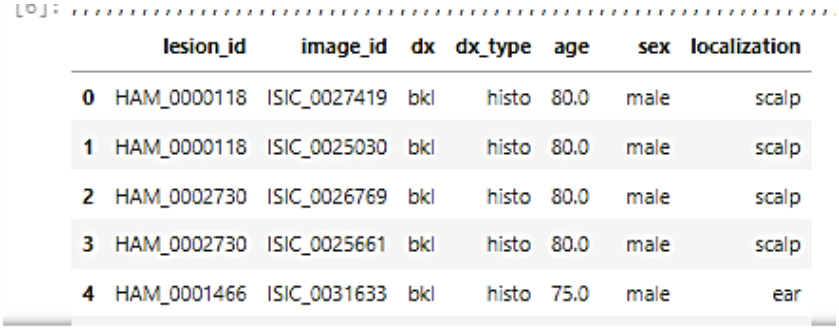
The following section explains the data preparation process, image augmentation via Albumentations, model building, and fine-tuning of the MLP-mixer model. Following that, we explain how to evaluate models on test sets and evaluate metrics such as accuracy, precision, recall, and F1-score.

4.2 Data Preparation

The HAM10000 dataset forms our foundation to Train, test and validate the accuracy of the model. The HAM10000 dataset contains 10015 dermatoscopic pictures with a resolution above 512 x 512 belonging to 7 types or class of skin lesion types that may be used as a training set for academic machine learning. We have performed grouping on data that reduces the dataset with 7 classes (multiclass) to 2 classes (Binary) viz. Melanoma/Non-Melanoma. The class “Melanoma” contains images from the class Melanoma (*mel* while the class “Non-Melanoma” contains images from Melanocytic nevi (*nv*), benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, *bkl*), dermatofibroma (*df*) and vascular lesions. The rest two classes namely (basal cell carcinoma (bcc) and Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec) are either cancerous or precancer in nature and have not been considered.

In the original dataset, the images were packed into two folders with suffixes part_1 and part_2. The image names were mapped into a csv file “HAM10000_metadata.csv” which contains

information about the lesion name (label), captured part of body and patient info (age,sex) without any personally identifiable information (PII)



	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

Figure 4.1 Snapshot of the records in the HAM10000 metadata file.

4.2.1 Data loading and pre-processing

The metadata in tabular form was loaded in a Pandas data frame and used as reference to load the images and their respective labels correctly. The short lesion names were mapped with full names and regrouped into two classes as mentioned. They were assigned labels "0" – non-melanoma and '1" – Melanoma as a part of the data pre-processing.

Table 4.1 Dataset description of HAM10000 dataset

	Skin Lesion Type Name	Original Labels	New Labels
HAM10000	Actinic keratoses and intraepithelial carcinoma / Bowen's disease (**akiec**),	akiec	Not Used
	basal cell carcinoma (**bcc**),	bcc	Not Used
	benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, **bkl**),	bkl	Non-Melanoma
	dermatofibroma (**df**),	df	Non-Melanoma
	melanoma (**mel**),	mel	Melanoma
	melanocytic nevi (**nv**)	nv	Non-Melanoma
	and vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, **vasc**).	vasc	Non-Melanoma

To load the images, we have employed OpenCV to load, resize to 100 x 100 and then normalize to range [0,255] before loading them in a list. Another thing to note here is that OpenCV reads an image in BGR format (so color channels of the image have the following order: Blue, Green, Red) unlike the popular RBG color format. So, when using OpenCV, we need to explicitly convert the images to RGB. Finally, the loaded images and labels in the list are converted into a NumPy array for image processing.

```
x = []          # Hold resized images.
y = []          # Hold image lesion ID from the data set.
img_width=100
img_height=100
# Listing all files in the part_1, part_2 dirs
lista1 = os.listdir('../input/skin-cancer-mnist-ham10000/HAM10000_images_part_1/')
lista2 = os.listdir('../input/skin-cancer-mnist-ham10000/HAM10000_images_part_2/')
img_list=df_skin['image_id'].tolist()
# [+] Handling images from part 1 directory
for i in range(len(lista1)):

    # [+] Features: reading and resize the photo.
    fname_image = lista1[i]
    fname_ID = fname_image.replace('.jpg', '')
    if fname_ID in img_list:
        file_to_read ='../input/skin-cancer-mnist-ham10000/HAM10000_images_part_1/' + str(fname_image)  # resolve image name
        img = cv2.cvtColor(cv2.imread(file_to_read), cv2.COLOR_BGR2RGB)
        img2 = cv2.resize(img,(img_height, img_width))
        cv2.normalize(img2, img2, 0, 255, cv2.NORM_MINMAX)# append the new image to the list x.
        x.append(img2)
        # Targets: Finding the image lesion ID and append it to the y list.
        output = np.array(df_skin[df_skin['image_id'] == fname_ID].lesion_ID)
        y.append(output[0])

    # inform the user with the number of loaded images each 100 img.
        if i % 100 == 0:
            print(i, 'images loaded')
```

Figure 4.2 Code Snippet to load the images and the label

4.2.2 Label Encoding

Label encoding is the process of transforming labels into a numeric form so that they may be read by machines. The operation of such labels can then be better determined by machine learning techniques. It is a significant supervised learning pre-processing step for the structured dataset. For encoding our labels, we used a LabelEncoder() function from sklearn preprocessing module. We created a dictionary with the definitions for each label i.e., 0- non melanoma and 1- melanoma.

```
# Lesion/dis names are given in the description of the data set.
lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl':'Benign keratosis-like lesion',
    'vasc': 'Vascular Lesions',
    'df':'Dermatofibroma'
}

lesion_ID_dict = {
    'nv': '0',
    'mel': '1',
    'bkl':'0',
    'vasc':'0',
    'df':'0',
}
```

Figure 4.3 Mapping the lesion names as per data regrouping

```
encoder=LabelEncoder()
y= encoder.fit_transform(y)
y=np.asarray(y)
```

Figure 4.4 Converting the lesion_id into encoded variable y

4.2.3 Class Imbalance

An imbalanced classification problem is one in which the distribution of examples across the known classes is biased or skewed. The distribution can range from a minor bias to a severe imbalance, with one example in the minority class for hundreds, thousands, or millions in the majority class or classes. As a result, models show poor predictive performance, particularly for minorities. This is a problem because the minority class is typically more important, and thus the problem is more sensitive to classification errors for the minority class than for the majority class.

Table 4.2 Class-wise distribution of Images

Class	Count of Images
Melanoma – Class 1	1113
Non-Melanoma – Class 0	8061

4.2.4 Data Augmentations

Image augmentation is a technique for modifying existing data to generate more data for the model training process. To put it another way, it is the process of artificially increasing the available dataset for training a deep learning model. Albumentations is a library that simplifies the data augmentation process with a simple interface and supports various geometric, transforms, blur, and dropout-based methods.

Steps for creating the Augmentation pipeline using Albumentations

1. Load the Images and labels from disk
2. Import Albumentations and define augmentation pipeline
3. Pass the images to the augmentation pipeline which returns altered images

```
import albumentations as A

transform = A.Compose([
        A.RandomRotate90(),
        A.Rotate(limit=70),
        A.Flip(),
        A.GaussNoise(),
        A.OneOf([
            A.Sharpen(),
            A.RandomBrightnessContrast(),
        ], p=0.3),
        A.HueSaturationValue(p=0.3),
        A.Equalize (mode='cv', p=0.4)
        ],p=0.5),


#Function to return augmented image after processing
def produce_new_img(image):
    augmented_image = transform(image=image)['image']
    return augmented_image
```

Figure 4.5 Code snippet for creating augmentation pipeline using Albumentations

To handle the class imbalance, we have oversampled the minority class samples (here – melanoma images) using simple data augmentations like Rotate, flip, adding gaussian noise, sharpen and random brightness. The additional images and their respective labels have been appended to the original dataset in NumPy Array form. The original dataset had 1 melanoma record for every ≈ 8 non-melanoma record, so we augmented 8 different images for each melanoma image which resulted into an image count of 8904 which is a bit higher than the non-melanoma image count which is 8061.

```
# Sampling of Minority samples Set
    # Targets: Finding the image lesion ID and append it to the y list.

def augmentor(x,y,k):
    print("Augmentation Has started")
    x_aug=[]
    y_aug=[]
    for i in range (len(x)):
        img = x[i]
        label=y[i]

        # [+] Add more images for class (k)
        if label==k:
            for j in range(7):    # Add 8 augmented images for each image of class (k)=1
                x_aug.append(produce_new_img(img))
                y_aug.append(label)

    print("Added {} images to Label {}".format(len(x_aug),k))
    x=np.concatenate((x, x_ag), axis=0)
    y=np.concatenate((y, y_ag), axis=0)
    print("New Shape of X: ",x.shape)
    print("New Shape of y: ",y.shape)
    print("Augmentation Completed")

    return x,y
```

Figure 4.6 Code Snippet for augmenting and appending the minority class to the dataset

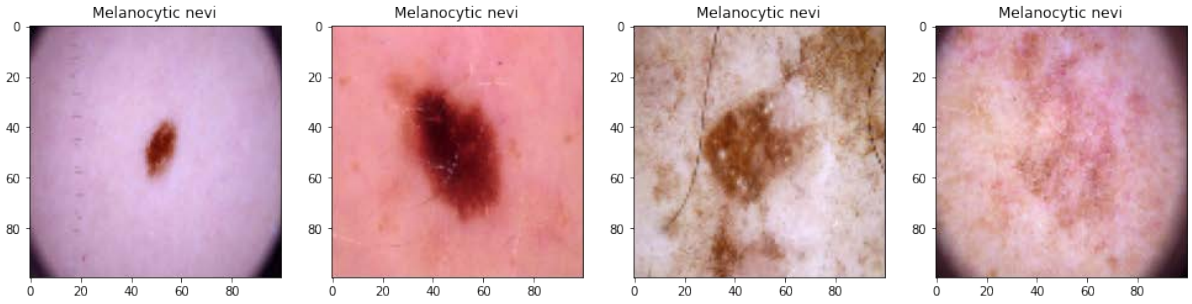


Figure 4.7 Images from the dataset after augmentation with label annotation

Table 4.3 Image Distribution after Data augmentation

Class	Count of Images
Melanoma – Class 1	8904
Non-Melanoma – Class 0	8061

4.3 Data Split for Training and Testing.

The dataset has then been split into train, validation, and test in the ratio 70:10:20 using sklearn's train test split utility. Since our dataset was small, data augmentation suited better than using other over sampling strategies like SMOTE (synthetic minority oversampling technique).

```
# split in 80% training and 20% test data
X_train, X_test, y_train, y_test = train_test_split(x,                  # Images array.
                                                    y,            # The training set.
                                                    test_size = 0.20,   # Split data set into 80/20
                                                    random_state = 50,  # Shuffling number to random the set.
                                                    stratify = y,
                                                    shuffle=True)       # Mix training and test sets.
```

```
# split in 80% training and 20% test data
X_train, X_val, y_train, y_val = train_test_split(X_train,                  # Images array.
                                                  y_train,             # The training set.
                                                  test_size = 0.10,    # Split data set into 90/10.
                                                  random_state = 50,   # Shuffling number to random the set.
                                                  stratify = y_train,
                                                  shuffle=True)        # Mix training and validation sets.
```

Figure 4.8 Code Snippet to split data into Train, test and validation set

4.4 MLP mixer model implementation

The following steps were followed to implement the MLP mixer model.

Step 1: Implement the Patch extraction layer

The patch extraction layer converts the input images into patches and feeds the patches as linear embeddings to the model similar to NLP models.

```
class Patches(layers.Layer):
    def __init__(self, patch_size, num_patches,**kwargs):
        super(Patches, self).__init__()
        self.patch_size = patch_size
        self.num_patches = num_patches

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, self.num_patches, patch_dims])
        return patches
```

```
    def get_config(self):

        config =super(Patches, self).get_config()
        config.update({
            'patch_size': self.patch_size,
            'num_patches': self.num_patches
        })
        return config

    @classmethod
    def from_config(cls, config):
        return cls(**config)
```

Figure 4.9 Code Implementation of Patch extraction layer in Keras.

The get_config() method shown in Figure 4.9 appears below the Patches block of code is functionally not required but it allows TensorFlow to identify the components while saving and loading these models in tf format which can loaded on other system for evaluation.

Step 2: Implement the MLP block and Mixer layer

The MLP mixer layer contains the two MLP blocks along with the mixing logic.

```
class MLPMixerLayer(layers.Layer):
    def __init__(self, num_patches, hidden_units, dropout_rate, *args, **kwargs):
        super(MLPMixerLayer, self).__init__(*args, **kwargs)
        self.num_patches = num_patches
        self.hidden_units = hidden_units
        self.dropout_rate = dropout_rate

        self.mlp1 = keras.Sequential(
            [
                layers.Dense(units=num_patches),
                tfa.layers.GELU(),
                layers.Dense(units=num_patches),
                layers.Dropout(rate=dropout_rate),
            ]
        )
        self.mlp2 = keras.Sequential(
            [
                layers.Dense(units=num_patches),
                tfa.layers.GELU(),
                layers.Dense(units=hidden_units),
                layers.Dropout(rate=dropout_rate),
            ]
        )
        self.normalize = layers.LayerNormalization(epsilon=1e-6)

    def call(self, inputs):
        # Apply layer normalization.
        x = self.normalize(inputs)
        # Transpose inputs from [num_batches, num_patches, hidden_units] to [num_batches, hidden_units, num_patches].
        x_channels = tf.linalg.matrix_transpose(x)
        # Apply mlp1 on each channel independently.
        mlp1_outputs = self.mlp1(x_channels)
        # Transpose mlp1_outputs from [num_batches, hidden_dim, num_patches] to [num_batches, num_patches, hidden_units].
        mlp1_outputs = tf.linalg.matrix_transpose(mlp1_outputs)
        # Add skip connection.
        x = mlp1_outputs + inputs
        # Apply layer normalization.
        x_patches = self.normalize(x)
        # Apply mlp2 on each patch independtenly.
        mlp2_outputs = self.mlp2(x_patches)
        # Add skip connection.
        x = x + mlp2_outputs
        return x
```

Figure 4.10 Code Implementation of MLP mixer block in Keras.

Step 3: Build the classifier head for the model

The classifier head puts the model components together. The input layer is connected to the patches layer which is forwarded to the MLP mixer module. The Output of the module is flattened using Global Average Pooling 1D and then connected to a dropout layer which get connected to a final layer of single neuron with a ‘sigmoid’ activation.

```python
def build_classifier(blocks, positional_encoding=False):
    inputs = layers.Input(shape=input_shape)
    # Create patches.
    patches = Patches(patch_size, num_patches)(inputs)
    # Encode patches to generate a [batch_size, num_patches, embedding_dim] tensor.
    x = layers.Dense(units=embedding_dim)(patches)
    if positional_encoding:
        positions = tf.range(start=0, limit=num_patches, delta=1)
        position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=embedding_dim
        )(positions)
        x = x + position_embedding
    # Process x using the module blocks.
    x = blocks(x)
    # Apply global average pooling to generate a [batch_size, embedding_dim] representation tensor.
    representation = layers.GlobalAveragePooling1D()(x)
    #apply dropout
    representation = layers.Dropout(rate=dropout_rate)(representation)
    # Compute logits outputs.
    output = layers.Dense(1, activation="sigmoid")(representation)
    # Create the Keras model.
    return keras.Model(inputs=inputs, outputs=output)
```

Figure 4.11 Code Implementation of classifier in Keras

Step 4: Define an Experiment to train the model

We write an experiment that defines the hyper parameters and the model fit parameters which will be written in a function which will be called in later steps. The experiment compiles the following parameters.

```python
weight_decay = 0.0001
num_epochs = 100
dropout_rate = 0.1
image_size = 100  # We'll resize input images to this size.
patch_size = 8  # Size of the patches to be extracted from the input images.
num_patches = (image_size // patch_size) ** 2  # Size of the data array.
embedding_dim = 256  # Number of hidden units.
num_blocks = 4   # Number of blocks.
learning_rate=0.001

print(f"Image size: {image_size} X {image_size} = {image_size ** 2}")
print(f"Patch size: {patch_size} X {patch_size} = {patch_size ** 2} ")
print(f"Patches per image: {num_patches}")
print(f"Elements per patch (3 channels): {(patch_size ** 2) * 3}")
```

Figure 4.12 Hyper-parameters for the base model

```
def run_experiment(model):
    # Create Adam optimizer with weight decay.

    optimizer=tfa.optimizers.AdamW(
        learning_rate=learning_rate,weight_decay=weight_decay)
    # Compile the model.
    model.compile(
        optimizer=optimizer,
        loss=keras.losses.BinaryCrossentropy(),
        metrics=[
            keras.metrics.BinaryAccuracy(name="accuracy")
                ]
    )


    reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', factor=0.6,
                              patience=3, min_lr=0.00000001,verbose=1)

    # Create an early stopping callback.
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor="val_accuracy", patience=20, restore_best_weights=True,mode="max"
    )

    #create a model checkpoint
    filepath='/kaggle/working/mlp-mixer.h5'
    checkpoint=tf.keras.callbacks.ModelCheckpoint(filepath,
        monitor="val_accuracy",verbose=1, save_weights_only=False,mode="max",save_best_only = True )

    # Fit the model.
    history = model.fit(x=X_train,y=y_train,
        batch_size=batch_size,
        shuffle=True,
        epochs=num_epochs,
        validation_data=(X_val,y_val),
        callbacks=[early_stopping,checkpoint,reduce_lr]
    )


    # Return history to plot learning curves.
    return history
```

Figure 4.13 Implementation of the experiment for model training and hyperparameters

Step 5: Compile the model and the defined hyperparameters.

This step compiles all the components along with the hyper parameters

```
#**** Compiling the model ****
mlpmixer_blocks = keras.Sequential([MLPMixerLayer(num_patches, embedding_dim, dropout_rate) for _ in range(num_blocks)])
mlpmixer_classifier = build_classifier(mlpmixer_blocks)
```

Figure 4.14 Compiling the model and the hyper-parameters

Step 6: Training the model on dataset

Running the training block compiled in experiment function of the model.

```
#**** Training the model ****
history = run_experiment(mlpmixer_classifier)
```

Figure 4.15 Running the training block of code

4.5 Model Hyper parameter Tuning

Hyperparameters are the fixed settings that determine its performance. These parameters include the following: input size, output size, patch size, embedding dim, number of layers (num blocks), training epochs, batch size, optimizer selection, regularization method, activation function, learning rate, and loss function. A proper selection of hyperparameters can help us improve the performance of our chosen model. We ran tuning experiments with all of the available hyperparameters to find the best patch size, embedding dim, number of layers, and drop out.

The learning rate has been set to 0.001 with a learning rate scheduler that monitors accuracy and reduces the learning rate with an exponential curve of factor 0.6.

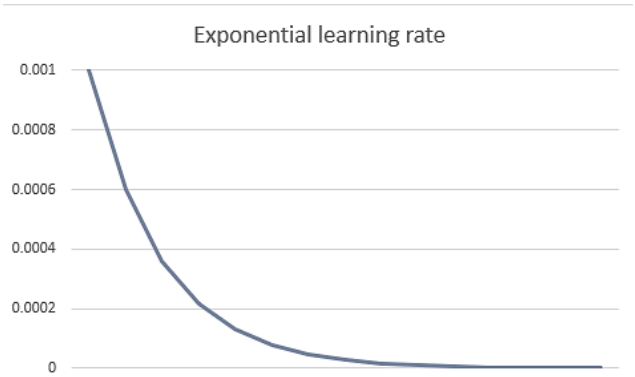


Figure 4.16 Exponential learning rate scheduler graph

Models can have a large number of hyperparameters and determining the best combination of parameters can be thought of as a search problem. The following are the two most effective Hyperparameter tuning strategies:

- GridSearchCV

  The machine learning model is evaluated for a variety of hyperparameter values in the GridSearchCV approach. This approach seeks the best set of hyperparameters from a grid of hyperparameter values.

- RandomizedSearchCV

  Because it only goes through a limited number of hyperparameter settings, RandomizedSearchCV overcomes the shortcomings of GridSearchCV. It moves randomly within the grid to find the best set of hyperparameters. This method eliminates unnecessary computation.

4.5.1 Keras Tuner

Keras Tuner is a scalable, easy-to-use hyperparameter optimization framework that alleviates the pain points associated with hyperparameter search. With a define-by-run syntax, we can easily configure a search space, then use one of the available search algorithms to find the best hyperparameter values our model. Keras Tuner includes built-in Bayesian Optimization, Hyperband, and Random Search algorithms and is designed to be easily extended by researchers to experiment with new search algorithms

Setting up Keras Tuner for our model

The Keras Tuner has a 3-step approach to run the hyperparameter tuning.

Step 1: Set the Tuner search space and define the model within the tuner.

```
import keras_tuner as kt

def build_model(hp):
    hp_weight_decay = 0.0001
    hp_dropout_rate = hp.Choice('dropout_rate', values=[0.1,0.2,0.3])
    hp_image_size = 100  # We'll resize input images to this size.
    hp_patch_size =  hp.Choice('patch_size', values=[8,16,32]) # Size of the patches to be extracted from the input images.
    hp_num_patches = (hp_image_size // hp_patch_size) ** 2  # Size of the data array.
    hp_embedding_dim = hp.Choice('embedding_dim', values=[256,384,512])  # Number of hidden units.
    hp_num_blocks = hp.Choice('num_blocks', values=[4,6,8,10])   # Number of blocks.
    hp_learning_rate=0.0001
    positional_encoding=False
    blocks=keras.Sequential([MLPMixerLayer(hp_num_patches, hp_embedding_dim, hp_dropout_rate) for _ in range(hp_num_blocks)])
    inputs = layers.Input(shape=input_shape)
    # Create patches.
    hp_patches = Patches(hp_patch_size, hp_num_patches)(inputs)
    # Encode patches to generate a [batch_size, num_patches, embedding_dim] tensor.
    x = layers.Dense(units=hp_embedding_dim)(hp_patches)
    if positional_encoding:
        positions = tf.range(start=0, limit=hp_num_patches, delta=1)
        position_embedding = layers.Embedding(
            input_dim=hp_num_patches, output_dim=hp_embedding_dim
        )(positions)
        x = x + position_embedding
    # Process x using the module blocks.
    x = blocks(x)
    # Apply global average pooling to generate a [batch_size, embedding_dim] representation tensor.
    representation = layers.GlobalAveragePooling1D()(x)
    #apply dropout
    representation = layers.Dropout(rate=dropout_rate)(representation)
    # Compute sigmoid outputs.
    output = layers.Dense(1, activation="sigmoid")(representation)
    # Create the Keras model.
    classifier= keras.Model(inputs=inputs, outputs=output)
    classifier.compile(optimizer=tfa.optimizers.AdamW(
                               learning_rate=hp_learning_rate,
                               weight_decay=hp_weight_decay),
                       loss=keras.losses.BinaryCrossentropy(),
                       metrics=[keras.metrics.BinaryAccuracy(name="accuracy")])

    return classifier
```

Figure 4.17 Defining the search space and the model in Keras Tuner

The tuner search space for each of parameters is defined below:

Table 4.4 Search space for the hyper parameters

Parameters	Search space
dropout rate	0.1,0.2,0.3
patch size	8,16,32,64
embedding dim	256, 384, 512
num blocks	4,6,8,10

Step 2: Define the tuner parameters

The tuner parameters include the hyper parameter tuning method which defines the search algorithm. Objective defines the metric and the direction to move. If accuracy is the objective, then the direction ideally would be max or maximizing the metric.

The epochs will limit how many times the model trains on a specific set of parameters which will set the training precedence. Directory would set the location to save the files and project name would define the folder.

```
tuner = kt.Hyperband(
    build_model,
    objective=kt.Objective("val_accuracy", direction="max"),
    max_epochs=30,
    directory='/kaggle/working/keras-tuner',
    project_name='keras_tuner_mlp_1')
```

Figure 4.18 Defining the Keras Tuner parameters

Step 3: **Running the search space and getting the results**

Once the tuner parameters are set. We run the tuner search and it runs a few epochs for each of the search space combinations and decides the best parameter. It usually takes around 2 hrs since the combinations are around 144.

```
tuner.search(X_train, y_train, epochs=30, validation_data=(X_val,y_val),callbacks=[tf.keras.callbacks.EarlyStopping(patience=5)])
```

Figure 4.19 Running the Keras Tuner search

Once the tuner search is completed, we get the best parameters by passing the results_summary() to the tuner object.

```
tuner.results_summary(1) #Get the hyperparameters for best score . 1
```

Figure 4.20 Fetching the best parameters from tuner object.

Based on the results of the Keras Tuner we have chosen the following hyperparameter settings that will be used during the model training phase.

Table 4.5 Best Hyper-parameters for the model.

Parameter Name	Parameters	Value
Input Shape	input_size	100 x 100, 3 Channels
Patch Size	patch_size	8
Hidden Units	embedding_dim	512
Layers	num_blocks	8
Learning Rate	Learning_rate	0.001
Optimizer	optimizer	AdamW
Weight Decay	weight_decay	0.0001
Loss	loss	BinaryCrossentropy
Epochs	epochs	100
Metric	metric	accuracy
Batch size	Batch_size	32

4.6 Callbacks for Models

Model callbacks are a set of actions that must be performed when the model reaches a predefined point in its training. These callbacks include saving the weights (ModelCheckpoint), lowering the learning rate (ReduceLROnPlateau), and terminating training early when the model is no longer able to learn (EarlyStopping). Note that Callbacks have been written as a part of the experiment function. The condition template for such callbacks is defined below.

```
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                              factor=0.6,
                                              patience=3,
                                              min_lr=0.0000001,
                                              verbose=1)

# Create an early stopping callback.
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_accuracy",
                                                  patience=20,
                                                  restore_best_weights=True,
                                                  mode="max"
)

#create a model checkpoint
filepath='/kaggle/working/mlp-mixer.h5'
checkpoint=tf.keras.callbacks.ModelCheckpoint(filepath,
                                              monitor="val_accuracy",
                                              verbose=1,
                                              save_weights_only=False,mode="max",
                                              save_best_only = True )
```

Figure 4.21 Callbacks for the model training.

4.7 Model Evaluation

To evaluate the model, we use the Test set which does not have any heavy augmentations. We use this test data to predict batch data with the model. We used sklearn's classification report and confusion matrix function to evaluate results of the model.

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

#Confusion Matrix and Classification Report
y_pred = mlpmixer_classifier.predict(X_test,verbose=1,)
y_pred_= [1 if (x>=0.5)  else 0 for x in y_pred ]
labels=np.unique(df_skin['lesion_ID'])
print('Confusion Matrix')
cm=confusion_matrix(y_test, y_pred_)
plt.figure(figsize=(10,7))
sns.heatmap(cm,cmap='Blues',annot=True,fmt="g",annot_kws={'size':14})
plt.show()
print("*"*50)
print('Classification Report')
print(classification_report(y_test, y_pred_))
print("*"*50)
```

Figure 4.22 Code snippet for model evaluation

4.8 Summary

In this section, we have implemented MLP mixer model and its components in TensorFlow. We have also implemented the data ingestion, data augmentation and training. The model hyper parameters have been tuned using Keras Tuner and we have set callbacks to ensure the model training is smooth. Finally, we have set the model evaluation metrics like accuracy, precision, recall and F1 score.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Introduction

In this section we will discuss the results obtained after the implementation of our proposed model, fine tuning and show some predictions on the test set. We have also discussed the significance of the results and compared them with the existing works.

5.2 MLP Mixer Model Performance on Testing set

We have built the MLP mixer using existing documentation on Keras and TensorFlow. The testing set had 3323 images in a batch size 32. In the below table we have listed the performance of our base model.

Table 5.1 Performance of the MLP mixer base model

Model	Parameters				Results	weighted avg	weighted avg	
Model Name	patch size	embedding dim	num blocks	dropout rate	Test Accuracy	Precision	Recall	F1-score
MLP mixer (base)	8	512	4	0.1	84.04%	0.84	0.84	0.84

5.3 Results from Fine Tuning the model

5.3.1 Performance

The base model yielded a good result on the dataset. Hence, we decided to fine tune the model to improve the overall performance. Keras tuner performed a Hyperband algorithm to find the best performing model parameters.

Table 5.2 Model evaluation for hyperparameter selection

Model	Optimizer	dropout rate	patch size	embedding dim	num blocks	Test Accuracy
MLP mixer	AdamW	0.1	8	256	4	84.0%
		0.3	8	256	4	81.2%
		0.2	16	384	16	77.2%
		0.3	32	512	32	65.4%
		0.1	16	384	16	74.8%
		0.1	32	512	8	73.2%
		0.1	**8**	**512**	**8**	**87.3%**

Findings:

From the above evaluation table, we observed that as the patch size and the embedding dim increase, the performance tends to drop. On investigating this, we realized that the number of patches factor depends on the image size and the patch size such that $= \frac{Image\ size}{Patch\ size^2}$. Since the image size is fixed at 100 x 100, increasing the patch size beyond a limit makes the number of patches very small which degrades the performance. The original design (Tolstikhin et al., 2021) has also described that the model has been employed on images with resolution 224 x 224. Since our dataset had around 16K images after performing data augmentation and class imbalance, resizing to 224 x 224 impacted our memory usage and was hitting the memory limit. Hence, we decided to cap the patch size at 8.

The intention of this study is to showcase that MLP mixer can competitively well like state-of-the-art models on economical computation resources. We intend to focus on light weight models which are faster to train while using lesser computation resources. If the model is trained on a larger dataset with state-of-the-art computation resources, it can compete with state-of-the-art models and even outperform them.

5.3.2 Additional Performance Tweaks

To improve the over performance of the model, we selected the best hyperparameters from the tuner search. Apart from those hyperparameters, we changed the weight decay of the optimizer AdamW from 0.0001 to 0.00001. The learning rate scheduler was configured to 0.6 which resulted in an exponential learning rate.

5.3.3 Light Weight

Apart from the performance, the model is equally compact and lightweight. The trained model along with weights takes **25MB** of space on disk. The following is the comparison with size of CNN models trained on ImageNet.

Table 5.3 Size comparison of MLP mixer with CNN models

Model	Size (MB)	Size of MLP mixer	Change
Xception	88	25	-63.0
VGG16	528	25	-503.0
VGG19	549	25	-524.0
ResNet50	98	25	-73.0
ResNet50V2	98	25	-73.0

5.3.4 Training Time

Since the model does not depend much on convolution or attention mechanisms, it trains faster on low-cost computation resources. Our Kaggle notebook had a Nvidia Tesla T4 16GB GPU attached and the training time was very less. The entire model was able to train under 60mins.

Table 5.4 Training Time - MLP mixer

Training Time	Time (s)	Images/epoch	ms/step
Per epoch	45	362	124.3093923
Total Time*	2250s or 42 mins	18100	124.3093923

*Total time is calculated as the Time taken per epoch x total number of epochs

5.4 Best Model

After analyzing the results from model fine tuning and evaluation, we found the best model with patch size 8, embedding dim 512 and number of layers 8 (num_block) and dropout rate of 0.1. The accuracy of the model was **87.3%** on the test set. The proposed MLP mixer generalized well on the dataset without any pretraining.

Table 5.5 Parameters of MLP mixer model (best performing)

Parameter Name	Parameters	Value
Input Shape	input_size	100 x 100 , 3 Channels
Patch Size	patch_size	8
Hidden Units	embedding_dim	512
Layers	num_blocks	8
Learning Rate	learning rate	0.001
Optimizer	optimizer	AdamW
Weight Decay	weight_decay	0.0001
Loss	loss	BinaryCrossentropy
Epochs	epochs	100
Metric	metric	accuracy

Table 5.6 Performance of Fine-tuned MLP mixer model

Model	Parameters				Results	weighted avg	weighted avg	
Model Name	patch size	embedding dim	num blocks	dropout rate	Test Accuracy	Precision	Recall	F1
MLP mixer (base)	8	512	4	0.1	84.04%	0.84	0.84	0.84
MLP mixer (fine-tuned)	**8**	**512**	**8**	**0.1**	**87.33%**	**0.876**	**0.872**	**0.873**

5.4.1 Confusion Matrix

A confusion matrix is a table that is frequently used to describe the performance of a classification model (or "classifier") on a set of test data with known true labels. The confusion matrix itself is simple to grasp, but the associated terminology can be perplexing.

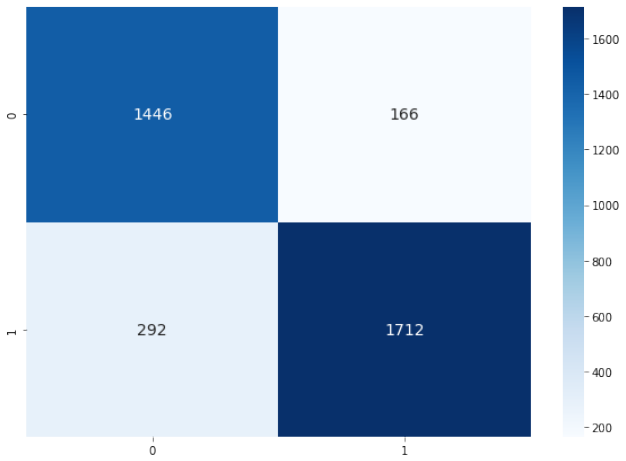


Figure 5.1 Confusion matrix of MLP mixer model on test set

From the confusion matrix, we can deduce that the classification of the model is quite accurate. It is also worth noting that the misclassification of 0 as 1. Ie False Positives is 166 and the misclassification of 1 as 0 .i.e false negatives is 292. The confusion matrix based metrics are listed below.

Table 5.7 Metric evaluation based on confusion matrix

Metrics	Value (Class Melanoma)	Value (Class Non-Melanoma)
Accuracy	87.3%	87.3%
Misclassification Rate	12.7%	12.7%
Precision	91.2%	89.7%
Recall	85.4%	83.2%
F1 Score	88.2%	86.3%
AUC score	0.87	0.87

5.4.2 Area Under The - Receiver Operating Characteristics (AUC-ROC) curve

The AUC - ROC curve is a performance metric for classification problems at various threshold levels. AUC represents the degree or measure of separability, while ROC is a probability curve. It indicates how well the model can distinguish between classes. The greater the AUC, the better the model predicts 0 classes as 0 and 1 classes as 1. Similarly, the higher the AUC, the better the model distinguishes between patients with and without the disease. An excellent model has an AUC close to one, indicating that it has a high level of separability. A poor model has an AUC close to zero, indicating that it has the worst measure of separability. In fact, it means that the result is being reciprocated. It predicts 0s to be 1s and 1s to be 0s. When the AUC is 0.5, the model has no class separation capacity at all. Random classifier has AUC of 0.5. AUC is expected to be greater than 0.5 to be accepted as a good classifier.

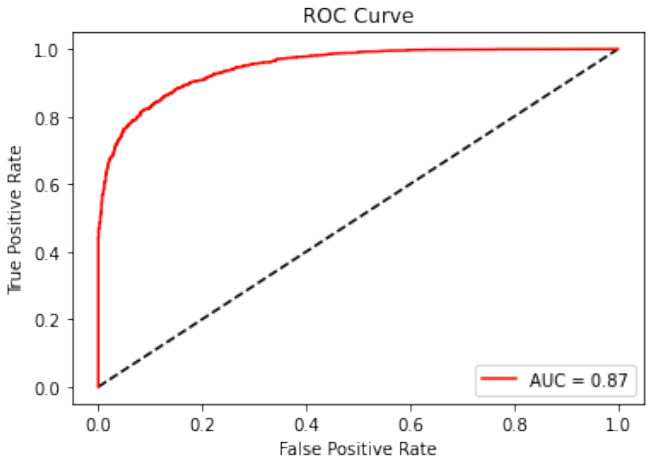


Figure 5.2 Receiver Operating Characteristics (ROC) Curve of MLP-mixer model

5.4.3 Model Training History

The model training history containing metrics accuracy and loss were captured to assess the model's training. The Figure 5.3 illustrates the training of the model.

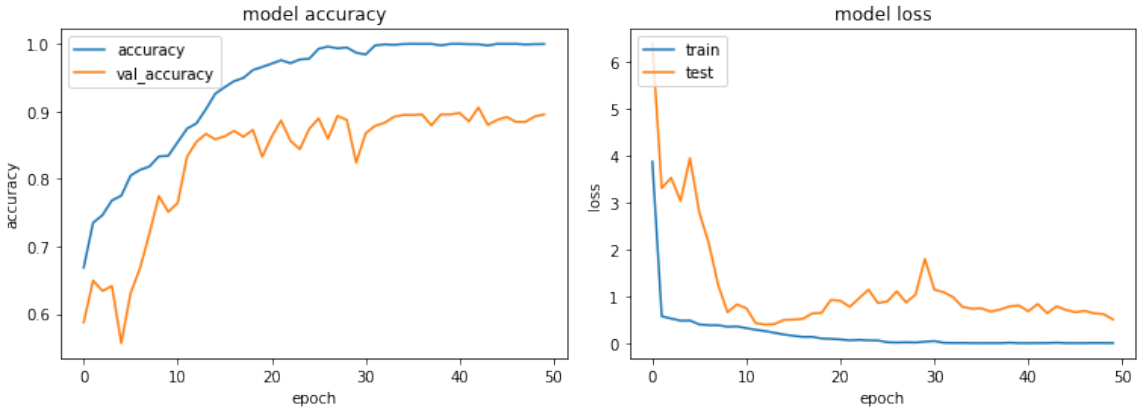


Figure 5.3 Plot of Training and Test Accuracy (left) and Loss (right) over training epochs

5.4.4 Prediction on unseen samples

To demonstrate the model's classification performance, we passed 5 unseen samples of skin lesion images to the model and plotted the image and actual vs predicted label and the results looks good. We were able to correctly identify all 5 samples.

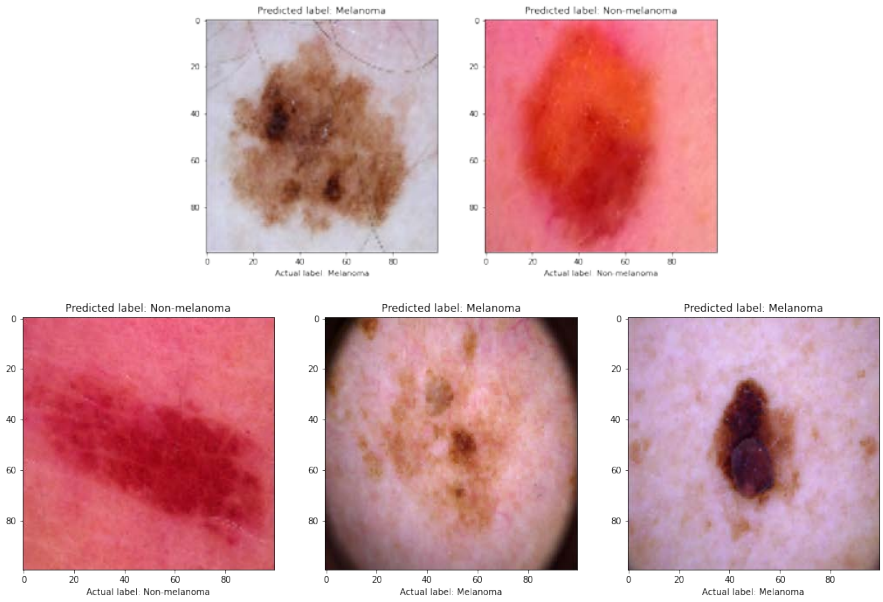


Figure 5.4 Predictions on five unseen samples of images.

5.5 Comparison with studies on CNN and Transformer models

HAM10000 dataset is popular among the machine learning community for skin cancer detection. These works focus on detection of melanoma either on multiclass or binary classification. In this classification task, our model has performed competitively with the other models. The summary of the comparison is mentioned below.

Table 5.8 Performance of MLP mixer model with other models.

Authored Work	Method	Accuracy	Precision	Recall	F1 score	AUC
(Esteva et al., 2017)	Inception V3	72.1	-	-	-	0.91
(Haenssle et al., 2018)	Inception V4	-	-	95	-	0.95
(Marchetti et al., 2018)	CNN ensemble	-	-	58	-	-
(Bi et al., 2017)	ResNet	-	-	-	-	0.915
(Romero-Lopez et al., 2017)	VGGNet	-	-	78.66	-	-
(Pham et al., 2020)	Inception V3	-	-	-	-	0.943
(Ali et al., 2021)	CNN (4 Conv Layer, 1 FC layer)	90.16	94.63	93.91	92.69	-
(Ech-Cherif et al., 2019)	MobileNetV2	91.33	-	-	-	-
(ABAYOMI-ALLI et al., 2021)	SqueezeNet	92.18	-	80.77	80.84	-
(Chen et al., 2021)	Vision Transformer	89.2	-	68	-	-
(Xie et al., 2021)	Swin Transformer	-	74	-	-	0.9
Our Proposed Work	**MLP-mixer model**	**87.33%**	**87.6**	**87.2**	**87.3**	**0.87**

5.6 Discussion

We have completed our research objectives outlined in Section 1.4 model (Aims and Objectives) that inspires to construct a lightweight, faster-to-train, and well-generalizing model. Our goal of creating a lightweight model that can compete with convolutional or attention-based models was met with success. With Dice Coefficient (F1 score) and binary classification accuracy of 87.0% and 87.3%, respectively, the classification model performed well.

We have supplemented skin lesion photos for the goal of class balance (Figure 4.7 Samples for image enhancements). For picture augmentation, we employed Albumentations library. The MLP-mixer model was created using the Keras documentation. To flatten the output of the MLP-mixer for the classification head, we added a Global Pooling Average layer, followed by a Dropout layer, and completed the model with a single neuron with sigmoid function. We have carried out preliminary research for the hyperparameter selection (as shown in Table 5.2) Initial model evaluation for hyperparameter selection). To improve our model, we employed the hyperparameters that were doing the best. Selected hyperparameters and enhanced images were used in the training of the suggested MLP-mixer model. The model's performance was evaluated on the test set.

We selected the best model from the hyper parameter tuning which maximized accuracy on the dataset. The best model parameters have been shown in Table 5.5 and the best performance of the model in Table 5.7. The confusion matrix plotted in Figure 5.1 shows that the model is able to classify well. Looking at the diagonal elements of the confusion matrix, we can deduce that classification rate is very good. To further elaborate on the performance, Table 5.7 highlights the metrics based on confusion matrix.

The area under curve - receiver operating characteristics (AUC-ROC) plot helps to understand the classifier’s ability to classify the samples accurately. An Ideal classifier should have the curve leaning on top left corner and having area=1 under the curve while a random classifier has a linear plot with area=0.5. A good classifier should have AUC better than 0.5. Our model has an AUC of 0.87 with a curve that leans towards the y-axis as shown in Figure 5.2.

From Figure 5.3 It can be observed that the validation accuracy is fluctuating, and the model is overfitting to an extent. We tried increasing the dropout factor but that degraded the performance. Hence, we kept the fine-tuned parameters as suggested by Keras Tuner. Another thing to note is that the training is a bit unstable compared to CNNs since the losses fluctuates a lot.

We passed 5 random images of skin lesions to the trained model to predict the lesion type and the model was able to identify them well as shown in Figure 5.4

From the comparison study on melanoma classification, the CNN and transformer models have performed extremely well. Our proposed model sits well between the CNN and transformer model with an accuracy of 87.3%, precision of 87.6% and recall of 87.2% and AUC of 0.87. The comparison of the models is shown in Table 5.8

With the implementation of our proposed MLP-mixer, we have achieved our aim to create a lightweight model that trains faster while using economical computation cost.

5.7 Summary

In this section, we have showcased the results from our MLP mixer model's melanoma detection. The model has performed well, and we have achieved the objectives set for this study.

CHAPTER 6

CONCLUSION AND DISCUSSION

6.1 Introduction

We have emphasised our accomplishments, their relevance, and their contribution to the skin cancer detection methods after carefully examining the data from the preceding section. We have also offered suggestions for more research in this area.

6.2 Conclusion

The goal of this research work was to develop a lightweight, dependable model that could more reliably categorise skin lesion photos into melanoma and non-melanoma. After using our methods, we were able to create a model that performed well on the HAM10000 dataset with accuracy, precision, and recall of 87.33%, 87.6%, and 87.2%, respectively. In this sense, our model has performed competitively with other models that has been created on this dataset. The findings demonstrate that our experimental model is a viable and trustworthy method for skin cancer identification (melanoma). This portable model can help medical professionals detect melanoma from skin lesions more accurately. Our model can train without GPUs as well, in contrast to most existing CNN and transformer models, which need expensive processing resources to do so. In addition to being precise, our model is lightweight and trains faster than state-of-the-art models.

6.3 Contribution to Knowledge

According to our study, the melanoma detection method we've suggested is among the earliest to make use of a contemporary image classification model like MLP-mixer. Our study shown that, in contrast to CNN and transformer models, the MLP-mixer model will enhance the classification process while consuming less time and computing resources. The results of the fine-tuning demonstrate the model's overall performance and dependability. With a dice coefficient (F1-score) and binary classification accuracy of 87.3% and 87.33%, respectively, we have built a model that generalises effectively. Although the model couldn't beat CNN models, the outcomes are on par with state-of-the-art Transformer and CNN models.

6.4 Future Recommendations

Our research was restricted to using skin lesion photos to identify melanoma in a binary classification problem. This can be further extended to multiclass classification of skin lesion. The model can generalise well and beat CNN models on large dataset and extensive augmentations like CutMix, RandAugment, and Mixup. Regularization techniques like gradient clipping and early stopping can prevent overfitting in the model. Considering the model was trained on a small dataset, training on a large dataset of different image classification tasks can be experimented.

REFERENCES

ABAYOMI-ALLI, O.O., DAMAŠEVIČIUS, R., MISRA, S., MASKELIŪNAS, R. and ABAYOMI-ALLI, A., (2021) Malignant skin melanoma detection using image augmentation by oversampling in nonlinear lower-dimensional embedding manifold. *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, 29SI-1, pp.2600–2614.

Ali, M.S., Miah, M.S., Haque, J., Rahman, M.M. and Islam, M.K., (2021) An enhanced technique of skin cancer classification using deep convolutional neural network with transfer learning models. *Machine Learning with Applications*, 5, p.100036.

Ando, S. and Huang, C.Y., (2017) Deep Over-sampling Framework for Classifying Imbalanced Data. pp.770–785.

Araujo, A., Norris, W. and Sim, J., (2019) Computing Receptive Fields of Convolutional Neural Networks. *Distill*, [online] 411. Available at: https://distill.pub/2019/computing-receptive-fields [Accessed 25 Jul. 2022].

Bi, L., Kim, J., Ahn, E. and Feng, D., (2017) Automatic Skin Lesion Analysis using Large-scale Dermoscopy Images and Deep Residual Networks.

Cancer Research UK, (2022) *Melanoma skin cancer statistics*. [online] Available at: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/melanoma-skin-cancer#heading-Four [Accessed 25 Jul. 2022].

Chen, J., Chen, J., Zhou, Z., Li, B., Yuille, A. and Lu, Y., (2021) MT-TransUNet: Mediating Multi-Task Tokens in Transformers for Skin Lesion Segmentation and Classification.

Codella, N.C.F., Gutman, D., Celebi, M.E., Helba, B., Marchetti, M.A., Dusza, S.W., Kalloo, A., Liopyris, K., Mishra, N., Kittler, H. and Halpern, A., (2017) Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC). [online] Available at: http://arxiv.org/abs/1710.05006.

Combalia, M., Codella, N.C.F., Rotemberg, V., Helba, B., Vilaplana, V., Reiter, O., Carrera, C., Barreiro, A., Halpern, A.C., Puig, S. and Malvehy, J., (2019) BCN20000: Dermoscopic Lesions in the Wild. [online] Available at: http://arxiv.org/abs/1908.02288.

Deng, J., Dong, W., Socher, R., Li, L.-J., Kai Li and Li Fei-Fei, (2009) ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp.248–255.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N., (2020) An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. [online] Available at: http://arxiv.org/abs/2010.11929.

Ech-Cherif, A., Misbhauddin, M. and Ech-Cherif, M., (2019) Deep Neural Network Based Mobile Dermoscopy Application for Triaging Skin Cancer Detection. In: *2019 2nd*

International Conference on Computer Applications & Information Security (ICCAIS). IEEE, pp.1–6.

Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M. and Thrun, S., (2017) Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, [online] 5427639, pp.115–118. Available at: http://www.nature.com/articles/nature21056.

Fukushima, K., (1988) Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 12, pp.119–130.

Guo, J., Tang, Y., Han, K., Chen, X., Wu, H., Xu, C., Xu, C. and Wang, Y., (2021) Hire-MLP: Vision MLP via Hierarchical Rearrangement. [online] Available at: http://arxiv.org/abs/2108.13341.

Haenssle, H.A., Fink, C., Schneiderbauer, R., Toberer, F., Buhl, T., Blum, A., Kalloo, A., Hassen, A.B.H., Thomas, L., Enk, A., Uhlmann, L., Alt, C., Arenbergerova, M., Bakos, R., Baltzer, A., Bertlich, I., Blum, A., Bokor-Billmann, T., Bowling, J., Braghiroli, N., Braun, R., Buder-Bakhaya, K., Buhl, T., Cabo, H., Cabrijan, L., Cevic, N., Classen, A., Deltgen, D., Fink, C., Georgieva, I., Hakim-Meibodi, L.-E., Hanner, S., Hartmann, F., Hartmann, J., Haus, G., Hoxha, E., Karls, R., Koga, H., Kreusch, J., Lallas, A., Majenka, P., Marghoob, A., Massone, C., Mekokishvili, L., Mestel, D., Meyer, V., Neuberger, A., Nielsen, K., Oliviero, M., Pampena, R., Paoli, J., Pawlik, E., Rao, B., Rendon, A., Russo, T., Sadek, A., Samhaber, K., Schneiderbauer, R., Schweizer, A., Toberer, F., Trennheuser, L., Vlahova, L., Wald, A., Winkler, J., Wölbing, P. and Zalaudek, I., (2018) Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. *Annals of Oncology*, 298, pp.1836–1842.

Hasan, M., Barman, S. das, Islam, S. and Reza, A.W., (2019) Skin Cancer Detection Using Convolutional Neural Network. In: *Proceedings of the 2019 5th International Conference on Computing and Artificial Intelligence - ICCAI '19*. New York, New York, USA: ACM Press, pp.254–258.

He, X., Tan, E.-L., Bi, H., Zhang, X., Zhao, S. and Lei, B., (2022) Fully transformer network for skin lesion analysis. *Medical Image Analysis*, 77, p.102357.

Hosny, K.M., Kassem, M.A. and Foaud, M.M., (2018) Skin Cancer Classification using Deep Learning and Transfer Learning. In: *2018 9th Cairo International Biomedical Engineering Conference (CIBEC)*. IEEE, pp.90–93.

Hou, Q., Jiang, Z., Yuan, L., Cheng, M.-M., Yan, S. and Feng, J., (2021) Vision Permutator: A Permutable MLP-Like Architecture for Visual Recognition. [online] Available at: http://arxiv.org/abs/2106.12368.
Khan, S., Naseer, M., Hayat, M., Zamir, S.W., Khan, F.S. and Shah, M., (2021) Transformers in Vision: A Survey.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., (2017) ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, [online] 606, pp.84–90. Available at: https://dl.acm.org/doi/10.1145/3065386.

Kumar, N.S. and Ramaswamy Karthikeyan, B., (2021) Diabetic Retinopathy Detection using CNN, Transformer and MLP based Architectures. In: *ISPACS 2021 - International Symposium on Intelligent Signal Processing and Communication Systems: 5G Dream to Reality, Proceeding*. Institute of Electrical and Electronics Engineers Inc.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. and Jackel, L.D., (1989) Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 14, pp.541–551.

Li, D., Hu, J., Wang, C., Li, X., She, Q., Zhu, L., Zhang, T. and Chen, Q., (2021) Involution: Inverting the Inherence of Convolution for Visual Recognition. [online] Available at: http://arxiv.org/abs/2103.06255.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. and Talwalkar, A., (2016) Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization.
de Lima, L.M. and Krohling, R.A., (2022) Exploring Advances in Transformers and CNN for Skin Lesion Diagnosis on Small Datasets. [online] Available at: http://arxiv.org/abs/2205.15442.

Liu, H., Dai, Z., So, D.R. and Le, Q. v., (2021) Pay Attention to MLPs. [online] Available at: http://arxiv.org/abs/2105.08050.

Logan Joe, (2018) *Improving CNN Training Times In Keras*. [online] Medium. Available at: https://joelognn.medium.com/improving-cnn-training-times-in-keras-7405baa50e09 [Accessed 20 Nov. 2022].

Loshchilov, I. and Hutter, F., (2017) Decoupled Weight Decay Regularization.
Luba, M.C., Bangs, S.A., Mohler, A.M. and Stulberg, D.L., (2003) Common benign skin tumors. *American family physician*, 674, pp.729–38.

Lv, T., Bai, C. and Wang, C., (2022) MDMLP: Image Classification from Scratch on Small Datasets with MLP. [online] Available at: http://arxiv.org/abs/2205.14477.

Lyu, H., Wang, Y., Tan, Y., Zhou, H., Zhao, Y. and Zhang, Q., (2022) Boosting Adversarial Transferability of MLP-Mixer. [online] Available at: http://arxiv.org/abs/2204.12204.

Marchetti, M.A., Codella, N.C.F., Dusza, S.W., Gutman, D.A., Helba, B., Kalloo, A., Mishra, N., Carrera, C., Celebi, M.E., DeFazio, J.L., Jaimes, N., Marghoob, A.A., Quigley, E., Scope, A., Yélamos, O. and Halpern, A.C., (2018) Results of the 2016 International Skin Imaging Collaboration International Symposium on Biomedical Imaging challenge: Comparison of the accuracy of computer algorithms to dermatologists for the diagnosis of melanoma from dermoscopic images. *Journal of the American Academy of Dermatology*, 782, pp.270-277.e1.

MLNotebook, (2017) *Convolutional Neural Networks - Basics*. [online] Available at: https://mlnotebook.github.io/post/CNN1/ [Accessed 18 Nov. 2022].

Mouton, C., Myburgh, J.C. and Davel, M.H., (2021) Stride and Translation Invariance in CNNs.

Narayanamurthy, V., Padmapriya, P., Noorasafrin, A., Pooja, B., Hema, K., Firus Khan, A.Y., Nithyakalyani, K. and Samsuri, F., (2018) Skin cancer detection using non-invasive techniques. *RSC Advances*, [online] 849, pp.28095–28130. Available at: http://dx.doi.org/10.1039/C8RA04164D.

National Health Service for England, (2020) *Skin cancer (non-melanoma)*. [online] Available at: https://www.nhs.uk/conditions/non-melanoma-skin-cancer/ [Accessed 23 Nov. 2022].
O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L. and others, (2019) *KerasTuner*. Available at: https://github.com/keras-team/keras-tuner [Accessed 25 Nov. 2022].

O'Shea, K. and Nash, R., (2015) An Introduction to Convolutional Neural Networks.
Pak, M. and Kim, S., (2017) A review of deep learning in image recognition. In: *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*. IEEE, pp.1–3.
PDQ Adult Treatment Editorial Board, (2002) *PDQ Melanoma Treatment*. [online] Bethesda, MD: National Cancer Institute. Available at: https://www.cancer.gov/types/skin/hp/melanoma-treatment-pdq [Accessed 25 Jul. 2022].

Pedro, R. and Oliveira, A.L., (2022) Assessing the Impact of Attention and Self-Attention Mechanisms on the Classification of Skin Lesions. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp.1–8.

Pham, T.C., Hoang, V.D., Tran, C.T., Luu, M.S.K., Mai, D.A., Doucet, A. and Luong, C.M., (2020) Improving binary skin cancer classification based on best model selection method combined with optimizing full connected layers of Deep CNN. In: *2020 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*. IEEE, pp.1–6.

Romero-Lopez, A., Giro-i-Nieto, X., Burdick, J. and Marques, O., (2017) Skin Lesion Classification from Dermoscopic Images Using Deep Learning Techniques. In: *Biomedical Engineering*. Calgary,AB,Canada: ACTAPRESS.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L., (2014) ImageNet Large Scale Visual Recognition Challenge.
Subramanian, R.R., Achuth, D., Kumar, P.S., Naveen kumar Reddy, K., Amara, S. and Chowdary, A.S., (2021) Skin cancer classification using Convolutional neural networks. In: *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. [online] IEEE, pp.13–19. Available at: https://ieeexplore.ieee.org/document/9377155/.

Sumit Saha, (2018) *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. [online] Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [Accessed 18 Nov. 2022].

Sun, C., Shrivastava, A., Singh, S. and Gupta, A., (2017) Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, pp.843–852.

Tang, C., Zhao, Y., Wang, G., Luo, C., Xie, W. and Zeng, W., (2022) *Sparse MLP for Image Recognition: Is Self-Attention Really Necessary?* [online] Available at: www.aaai.org.

Tatsunami, Y. and Taki, M., (2022) Sequencer: Deep LSTM for Image Classification. [online] Available at: http://arxiv.org/abs/2205.01972.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M. and Dosovitskiy, A., (2021) MLP-Mixer: An all-MLP Architecture for Vision. [online] Available at: http://arxiv.org/abs/2105.01601.

Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Izacard, G., Joulin, A., Synnaeve, G., Verbeek, J. and Jégou, H., (2021) ResMLP: Feedforward networks for image classification with data-efficient training. [online] Available at: http://arxiv.org/abs/2105.03404.

Tschandl, P., Rosendahl, C. and Kittler, H., (2018) The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, [online] 51, p.180161. Available at: http://www.nature.com/articles/sdata2018161.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., (2017) Attention Is All You Need.

Wei, G., Zhang, Z., Lan, C., Lu, Y. and Chen, Z., (2022) ActiveMLP: An MLP-like Architecture with Active Token Mixer. [online] Available at: http://arxiv.org/abs/2203.06108.

Xie, J., Wu, Z., Zhu, R. and Zhu, H., (2021) Melanoma Detection based on Swin Transformer and SimAM. In: *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*. IEEE, pp.1517–1521.

Xin, C., Liu, Z., Zhao, K., Miao, L., Ma, Y., Zhu, X., Zhou, Q., Wang, S., Li, L., Yang, F., Xu, S. and Chen, H., (2022) An improved transformer network for skin cancer classification. *Computers in Biology and Medicine*, 149, p.105939.

Zhai, X., Kolesnikov, A., Houlsby, N. and Beyer, L., (2021) Scaling Vision Transformers.
Zhao, Y., Wang, G., Tang, C., Luo, C., Zeng, W. and Zha, Z.-J., (2021) A Battle of Network Structures: An Empirical Study of CNN, Transformer, and MLP. [online] Available at: http://arxiv.org/abs/2108.13002.

Zhou, L. and Luo, Y., (2021) Deep Features Fusion with Mutual Attention Transformer for Skin Lesion Diagnosis. In: *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, pp.3797–3801.

APPENDIX A: RESEARCH PROPOSAL

DETECTION OF MELANOMA (SKIN CANCER) FROM PIGMENTED SKIN LESION IMAGES USING MODIFIED MLP-MIXER MODEL

PRASAD PRAMOD MAHARANA

RESEARCH PROPOSAL

JULY 2022

Abstract

Melanoma, one of the most serious forms of skin cancer arises from melanocytes, which produce pigment. According to Cancer Research UK, melanoma has claimed 2,34 lives during 2017-2019 in United Kingdom. Although, it is curable and has a survival rate of 87%. Histopathological analysis and biopsy are the main diagnostic methods to detect melanoma. Recent non-invasive cancer detection methods involving imaging techniques pigmented skin lesion images sparked interests in deep learning community. This led to using convolutional neural networks (CNN) and Transformers to accelerate detection of melanoma which showed promising results. The existing CNN and Transformer models rely on convolution and attention modules to detect features and make decisions. These models have few drawbacks like complexity, long training hours and intensive computing resources usage (GPU-graphics processing unit and memory).

In this proposed work, a new approach of using multilayer perceptrons (MLP) based architecture is proposed to overcome these drawbacks without compromising performance. To achieve this, an MLP architecture known as MLP-mixer will take the input images as patches which are converted into a table like form and then fed to a mixer layer. The mixer layer contains 2 MLP blocks, the first one (token mixing) detects features in the image across patches viz. aggregates channels where the feature occurs. The second block looks for features in the patch and associates it with the channel. It also uses Layer Norm along with GELU (Gaussian Error Linear Unit) activations and skip connections. This model will be trained on skin lesion images which will be pre-processed to ensure uniform input. The results will be compared quantitively to state-of-the-art (SOTA) CNN and Transformer models. This will conclude in a lightweight model which can be trained faster on economical computation resources and deployed into real-time applications. We hope the results will further initiate more interests to explore MLP architecture.

Table of Contents

List of Figures

List of Abbreviations

Abbreviation	Definition
UK	United Kingdom
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
MLP	Multi-Layer Perceptron
GELU	Gaussian Error Linear Unit
ICLR	International Conference on Learning Representations
ViT	Vision Transformer
DeiT	Data-efficient image Transformer
MLOps	Machine Learning operations

1. Background

Melanoma, also known as malignant melanoma, is a type of skin cancer that emerges from the pigment-producing cells called melanocytes. Melanoma usually occur in the skin, but may rarely occur in other places like mouth, intestines, or eye (uveal melanoma)(PDQ Adult Treatment Editorial Board, 2002). In UK, there were 16,744 cases on average during the years 2016 – 2018 and 2,341 deaths during the years 2017 – 2019. Melanoma has a survival rate of 87% and a prevention rate of 86% (Cancer Research UK, 2022).

There are diverse methods to diagnose melanoma which include invasive methods like biopsy and histopathological examination. Some of non-invasive techniques include thermography, electrical bio-impedance, tape stripping, dermatoscopy and computer-aided analysis (Narayanamurthy et al., 2018,). These methods sparked interests in utilizing pigmented skin lesion image samples to accelerate melanoma detection which can be followed by in-clinic consultation and treatment. Some of the datasets that pooled such diverse skin lesion images include BCN_20000 dataset (Combalia et al., 2019), HAM10000 dataset (Tschandl et al., 2018)and MSK dataset (Codella et al., 2017).

CNNs have been highly preferred due to their accuracy and reliability for feature detection in classification tasks. In 2012, a 0.23 percent error rate was recorded on the MNIST database. Another article on CNN image classification stated that the learning curve was "surprisingly fast". In the same research work, The MNIST and NORB databases yielded the best published findings of 2011. Later, the 2012 ImageNet Large Scale Visual Recognition Challenge was won by AlexNet with a top-5 error rate of 18.2% (Krizhevsky et al., 2017). CNNs rely on convolution technique to extract features from the input and make decisions. Inspired by the achievements of transformers in natural language processing (NLP) they were extended to computer vision applications. Vision Transformers (ViT) firstly introduced a self-attention-based backbone to computer vision which further led to exploration of strategies to improve training of ViT. This further resulted in ViT achieving comparable performance to CNNs trained solely on ImageNet. It beat the state-of-the-art models on multiple image recognition tasks with the best model reaching accuracy of 94.55% on CIFAR-100, 90.72% on ImageNet-ReaL and 88.58% on ImageNet (Dosovitskiy et al., 2020).

While CNNs and Transformers have shown remarkable performances, the computational costs on such techniques are usually high. Considering the large computational costs associated with CNNs and attention modules in Transformers, a simple and efficient models that consists of Multilayer Perceptrons (MLP) was proposed in 2021 (Tolstikhin et al., 2021). This new architecture utilised techniques such as token mixing, channel mixing and matrix multiplication instead of convolution.

This new architecture claimed that while convolution and attention give good performance, both aren’t necessary which sparked our interest. MLP-mixer achieves competitive results on image classification benchmarks when trained on larger datasets with regularisation techniques, with pre-training and inference costs equivalent to SOTA models. MLP can perform well with modest model sizes, but as the model size increases, it suffers greatly from over-fitting.

In this research proposal some of the drawbacks of CNN and transformers will be addressed by utilizing MLP mixer into image classification task i.e., detection of melanoma. The three datasets have been combined to include diverse images for the model to learn. The focus of this proposed work shall be on creating a model void of intensive computation modules like convolution or attention and performance to the likes of state-of-the-art CNN and Transformer models.

The major contributions are summarized below:

1. A new classification model based on the MLP mixer architecture is proposed to classify the pigmented skin lesion images.
2. A lightweight model that trains faster compared to state-of the art models that requires less computational resources without losing performance.

2. Problem Statement and Related Work

CNN models have been very efficient at image classification and image detection tasks. CNNs have been outperforming themselves with new architectures and methods. Although, these models rely on convolution to detect features in images to make decisions. While convolution is good at detecting features, it is not necessary.

As CNN models have evolved, starting from AlexNet, to VGGNet, ResNet and Inception, the complexity and depth has increased as a result of increased number of layers. In recent models (networks), the receptive fields of the input layers cover the entire input image which means that the context used by the features in the resultant output feature map encompasses all the input image pixels(Araujo et al., 2019). Although there has been a rapid advancement of neural network architectures, convolution still remains as the fundamental method in deep neural networks. Drawing inspiration from the long-established image filtering technique, convolution kernels have two exceptional properties that make them so robust, namely spatial-agnostic, and channel-specific (Li et al., 2021). A proposed work in 2021 performed skin cancer detection using CNN which resulted in training accuracy of 83.04%, precision of 0.81 and recall of 0.80 (Subramanian et al., 2021)

Nonetheless, inspired by the many advancements achieved by transformers in Natural Language Processing (NLP), they have been extended into computer vision applications. In particular, Vision Transformer (ViT), a pure Transformer when applied to image recognition, achieves remarkable performance competing with CNNs. Various studies performed by ViT have shown that the state-of-the-art (SOTA) can be accomplished for a broad range of computer vision tasks using self-attention solely without using convolution(Tatsunami and Taki, 2022). The experimentation began with applying standard transformer directly to images with minimum modifications. To achieve this, the image was split into patches and these patches were fed as a sequence of linear embedding as an input to the transformer. Image patches were processed in a similar way as tokens (words) in an NLP application. Then the model was trained as an image classifier in supervised method. When trained on mid-sized datasets like ImageNet without any strong regularization, this model resulted low-key accuracies close to ResNet of comparable size(Dosovitskiy et al., 2020).

This new Transformer structure which made appearance in ICLR in 2021, Vision Transformer, demonstrated that a Transformer implemented directly to a sequence of images (patches) can accomplish a good performance on image classification tasks provided the training dataset is sufficiently large. Vision Transformer was also tested on skin cancer detection with resulted in AUC (area under ROC curve) of 0.771±0.018 which shows improvement in skin lesion diagnosis tasks (de Lima and Krohling, 2022). Data-efficient image Transformer (DeiT) further indicated that Transformers can be trained on classic scale dataset such as ImageNet-1k along with relevant data augmentation and model regularizations (Zhao et al., 2021). Vision Transformers firstly introduced a self-attention-based backbone to computer vision which further led to exploration of strategies to improve training of ViT. This further resulted in

proposal of a knowledge distillation method which helped ViT achieve comparable performance to CNNs trained solely on ImageNet (Wei et al., 2022).

While CNNs and Transformers have shown remarkable performances, the computational costs on such techniques are usually high. Resources like GPU, memory and training time play a major role while implementing models. While these techniques can achieve certain computer vision tasks, it is also important to evaluate the resources required to build and train these models. If a certain model can classify images at an intensive computation cost, it would not yield an efficient solution. Considering the large computational costs associated with convolution modules in CNNs and attention modules in Transformers, a simple and efficient models that consists of Multilayer Perceptrons (MLP) was proposed. This new architecture utilised techniques such as token mixing and channel mixing to capture the relationship between tokens and channels respectively (Guo et al., 2021).

This new architecture was named as MLP-mixer. MLP or Multilayer Perceptron has been around in statistical and machine learning modelling from a very long time but didn't yield valuable results and hence was pushed back. But with the advent of MLP-mixer, it presented a new direction within computer vision. This new methodology demonstrated that convolution and attention are adequate for good performance, but neither is required. The MLP-mixer is a multilayer perception-specific architecture that combines two different types of layers: one layer where MLPs are applied individually to image patches to "mix" location-specific features, and another layer where MLPs are applied uniformly across image patches to "mix" spatial information. MLP-mixer achieves competitive results on image classification benchmarks when trained on larger datasets and regularisation techniques, with pre-training and inference costs equivalent to SOTA models. The proposed work reached an accuracy of 84.15% (ImNet Top-1 Validation) on ImageNet-21K (Tolstikhin et al., 2021). A projection layer follower by a fully connected layer was proposed in MLP-Mixer, where the images were separated into equal patches and fed through the projection layer. The output from the projection layer was marked as "Table X". This table is then put through a channel mixing MLP (analogous to 3x3 convolution) and a token mixing MLP (analogous to 1x1 convolution). A number of these mixer layers will be arranged end to end in the network, together with skip connections, layer normalisation, and GeLU- activations. Comparing all the models, MLP-Mixer had the shortest amount of training time. Although the model convergence was sluggish (Kumar and Ramaswamy Karthikeyan, 2021). The MLP-Mixer, developed by Tolstikhin et al., when trained from scratch, achieved a CIFAR10 accuracy of about 80%. Teaming an MLP architecture along with a regularizer to force the model to be close to CNN based networks, Neyshabur achieved 85.19 percent accuracy on CIFAR10 dataset (Lv et al., 2022).

MLP can perform well with modest model sizes, but as the model size increases, it suffers greatly from over-fitting. The author believes that the biggest barrier preventing MLP from obtaining SOTA performance is overfitting (Zhao et al., 2021). MLPs serve as the foundation for the MLP-Mixer architecture. The primary information of the image can be captured by MLP-Mixer by fully combining information from several patches and channels. By disturbing

the information mixing mechanism of this architecture, overfitting of source models can be prevented which can further improve transferability of adversarial examples across architectures (Lyu et al., 2022).To improve the new MLP based alternative to Transformers, static parameterization of channel projections and spatial projections was introduced. The research work explored several design options for this architecture and discovered that linear spatial projections along with multiplicative gating perform best. The model was called gMLP since it is composed of basic MLP layers with gating (Liu et al., 2021).

MLP-Mixer have showcased impressive results when trained on the massive dataset JFT-300M, however when trained on a medium-scale dataset like ImageNet-1K, it falls short of its visual Transformer competitors. A gating operation was designed by gMLP (Tang et al., 2022) to improve communications between spatial locations. ResMLP proposed an affine transform layer which assists in stacking a large number of MLP blocks. ResMLP was able to achieve a top-1 accuracy of 98.1% on CIFAR-10 and 87.0% on CIFAR-100(Touvron et al., 2021). Another variant called "Vision Permutator" proposed preserving the original spatial dimensions of the input tokens and the positional information carried by 2D feature representations, in contrast to current MLP based models like MLP_mixer (Tolstikhin et al., 2021) or ResMLP (Touvron et al., 2021), which encode spatial information by flattening the spatial dimensions first and then conducting linear projection along the spatial dimension (Hou et al., 2021).

This research proposal focuses on building a lightweight MLP based architecture that can learn the features from the pigmented skin lesion images. The model shall train on dataset of HAM10000. The results of the classification will be assessed on metrics like accuracy, precision, recall and F-measure.

3. Research Questions

The following are some of the research questions that this research proposal attempts to answer.

- How to create a reliable binary classification model using a relatively new architecture?
- How can we build robust models which do not depend on convolution or attention?
- How well will the model learn features from the input images?
- How stable and fast would the model train without compromising on performance?
- How does the model performance against state-of-the-art CNNs and transformers?

4. Aims and Objectives

The main aim of this research is to detect melanoma (skin cancer) from pigmented skin lesion images using modified multilayer perceptron-mixer (MLP) model. The goal is to build an image classification model without leveraging convolution as opposed to conventional CNNs or Transformers while achieving similar performance. Our goal does not take in consideration of outperforming current SOTA models but to demonstrate that an MLP-based model is competitive with convolutional and attention-based models.

The Objectives of the research are listed below:

1. To suggest suitable image pre-processing and transformation techniques to ensure high quality image samples which enhance features and are easier to analyse.
2. To leverage data augmentation techniques to generate plausible samples from existing ones to handle class imbalance in the dataset
3. To determine the parameters and hypermeters along with optimization to train the model faster and ensure it is an optimum solution that generalizes well.
4. To detect melanoma from the test skin images accurately.
5. To evaluate and analyse the model's performance based on classification metrics like accuracy, precision, recall and F-measure

5. Significance of Study

Melanoma has been one of the deadliest skin cancer forms which claims lives each year. With the advent of non-invasive detection techniques like imaging, it is faster to diagnose and treat at early stages. But these imaging techniques involve investing in expensive imaging equipment and equally skilled labor. Deep learning can accelerate detection of melanoma from skin lesion images.

The current state-of the art deep learning models like CNNS and Transformers produce very high accuracy, but it has very high complexity, requires huge dataset, longer and extensive training times and computational cost. CNN models are also slower to train due to operations such as Max Pooling and numbers of layers (Depth). All these drawbacks make it hard to train and deploy a model in a particular application.

This research intends to highlight new and effective MLP-mixer based architectures to perform computer vision tasks such as image classification. Today's image processing networks frequently mix the features between different locations or at a single place. As an illustration, CNNs execute both mixes using convolutions, kernels, and pooling, whereas vision transformers do so using self-attention. Only employing MLPs, MLP-Mixer makes an effort to perform both (mixed features at a single place or mix the features over many locations) in a more "independent" manner. The benefit of employing MLPs, which are essentially just matrix multiplication, is the speed and simplicity of the architecture. Additionally, unlike vision transformers, whose computational complexity is quadratic, the MLP-computational Mixer's complexity is linear as the number of input patches increases. Additionally, the model uses regularization and skip connections. This research intends to provide with a viable alternative to CNNs which provide similar performance with lightweight designs and faster training.

This will benefit the medical diagnostic industry to build robust and cost-effective imaging solutions which can detect melanoma with high precision without requiring expensive imaging devices. This can be considered for primary diagnosis prior to consultation to reduce the treatment initiation time. We also hope this will help spark more interests and further research into MLP for computer vision tasks.

6. Scope of Study

The scope of the study of this research work is listed as follows:

- A lightweight image classification model based on MLP-mixer architecture that shall be implemented and trained
- The primary focus is to provide a viable and computationally less expensive alternative to CNNs and Transformers for medical imaging.
- Metrics such as accuracy, precision, recall and F-measure will be used to compare the results with other state-of-the-art models
- The model shall be trained and evaluated on the dataset mentioned in the research methodology section.
- Optimization and modification to outperform state-of-the-art models is not in the scope of this research work

7. Research Methodology

In this section, the details of the proposed MLP-mixer based model is described.

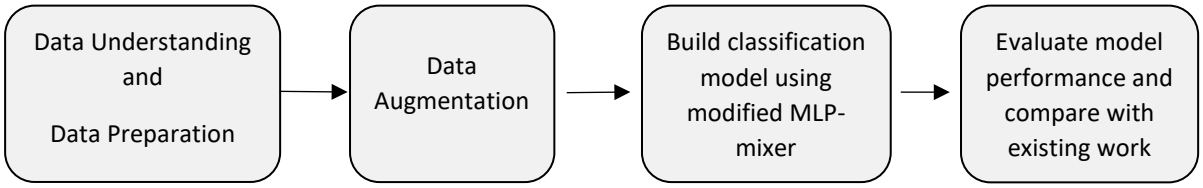


Figure 1 Experiment Flow chart

As illustrated in the flow diagram, we will understand the dataset and pre-process the images for ingestion. Data augmentation will be done to treat class imbalance in the dataset and then build the MLP-mixer and train it on the dataset. On completion of training, we will evaluate the model on the test set with metrics like precision, recall, accuracy and F-measure. To compare the model's performance, we will study recent works which will be a benchmark for our model.

7.1 Data Understanding and Preparation

The dataset that we will be using for this research is the HAM10000 dataset which is a large collection of multi-source dermatoscopic images of common pigmented skin lesions. The HAM10000 dataset contains 10015 dermatoscopic pictures with a resolution above 512 x 512 belonging to 7 types or class of skin lesion types that may be used as a training set for academic machine learning. The dataset contains a complete listing of all significant diagnostic subcategories for pigmented lesions, including: Basal cell carcinoma (bcc), benign keratosis-like lesions (solar lentigines/seborrheic keratoses and lichen-planus like keratoses (bkl), dermatofibroma (df), melanoma (mel), melanocytic nevi (nv), and vascular lesions are all examples of actinic keratoses and intraepithelial carcinoma/ (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage (vasc) (Codella et al., 2017; Tschandl et al., 2018; Combalia et al., 2019).

We will perform grouping on data that reduces the dataset with 7 classes (multiclass) to 2 classes (Binary) viz. Melanoma/Non-Melanoma. The class "Melanoma" will contain images from the class Melanoma (*mel* while the class "non-Melanoma" will contain images from Melanocytic nevi (*nv*), benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, *bkl*), dermatofibroma (*df*) and vascular lesions. The rest two classes namely (basal cell carcinoma (bcc) and Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec) are either cancerous or precancer in nature and will not been considered.

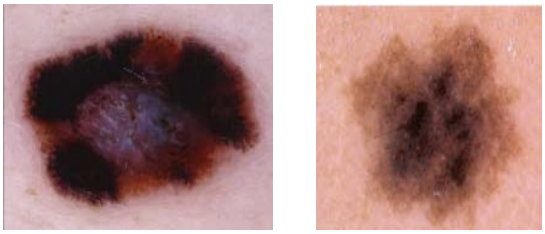


Figure 2 Sample images from the dataset

The dataset will be split into 80:20 ratio where 13,572 images will form the train set while the rest 3,393 will be used for evaluation/test set. The images will be resized and cropped to 100 x 100 resolution.

7.2 Data Augmentation

We have relatively medium dataset (16K images) but it has less images of target class: melanoma so we will treat class imbalance in the dataset. To build a robust model, we need high quality diverse target data. So, it can learn from the variance in the dataset and capture the patterns. Lesser data means the model cannot generalize well. So, to increase the volume of target class data, we will perform some image transformation on the input images. Changes such as translations, rotations (0-90 degree), flipping horizontal or vertical, color of image and hue shall be applied. Additionally, we will also perform operations like cropping, adding gaussian blur or combinations of both methods. Adding diversity in input images will allow the model to learn the patterns since the quality of the model depends on the quality of input.

7.3 Architecture

The proposed architecture takes input in form of patches which is fed to a Mixer layer, the fully connected layers with GELU activations and a linear classification head. The architecture also uses skip-connections and regularization like dropout and LayerNorm.

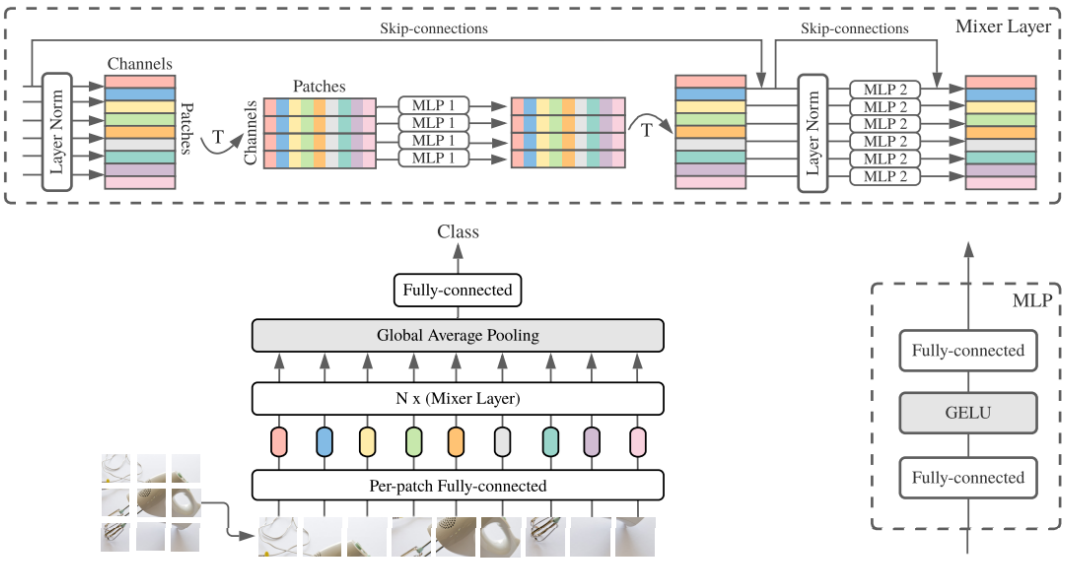


Figure 3 MLP-mixer architecture

As we see at the bottom of the architecture, the input to the network is patches of images. These patches are projected linearly into an H-dimension latent representation (which is hidden) and passed on to the Mixer layer. One point to notice here is that the H value is independent of the number of patches or patch sizes that enables the network to grow linearly rather than quadratically. This resultes in reduced process parameters and a higher throughput of 120 images/sec/core which is roughly 3 times than ViT's 32 images/sec/core.

Referring to the top part of the architecture which shows the Mixer layer, the patches are converted to into a table form, let's call that X, when projected to the mixer layer. The mixer layers consists of 2 MLP blocks namely, token mixing and channel mixing. Token mixing block acts on transpose of X i.e. columns of linear projection table. Every row has the same channel information for all the patches. This is further fed to a block of 2 Fully Connected Layers. This block holds the same type of information i.e. features in the image across patches. The weights are shared in the MLP1 layer shown in figure. The second layer called the Channel mixing block acts on another transpose of X. Each patch is taken and applied computation across all the channels of the patch. This is looking for features in that patch only but associating it with the channel.

The mixer layers can be mathematically defined as:

$$U_{*.i} = X_{*,i} + W_2\sigma\big(W_1 LayerNorm(X)_{*,i}\big), \quad for\ i = 1 \dots C$$

$$\mathrm{Y} *.\mathrm{i} = \mathrm{Uj},* + W_4\sigma\big(W_3 LayerNorm(U)_{j,*}\big), \quad for\ j = 1 \dots S$$

Where X = columns of linear projection table
S= input sequence of non-overlapping image patches
W_n= weights of layer n
C= hidden dimension
σ = element wise GELU activation

Also, the architecture uses Layer Norm which is often used in Transformer architectures. Below image shows Layer Norm vs BatchNorm. There are skip connections included with the GELU activation for non-linearity and dropout for regularization.

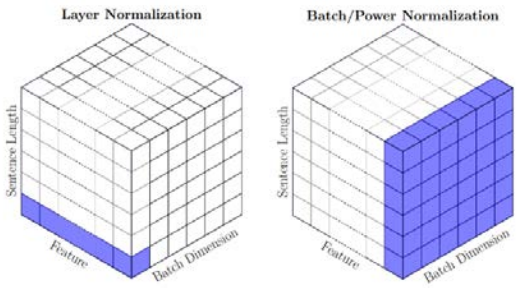


Figure 4 Layer Normalization (left) and Batch/Power Normalization (right) (Shen et al., 2020)

At the end, the mixer layer uses a classification head with global pooling layer and followed by a linear classifier.

7.4 Evaluation of model

After finetuning and optimizing the models, they will be evaluated on basis of the test set to check their prediction capability. The MLP-mixer model and the CNN models will be evaluated on the following metrics: Accuracy, precision, recall and AUC/ROC.

The evaluation is based on the following concept:

True Negative (TN) represents the number of negative samples predicted as negative.
True Positive (TP) represents the number of positive samples predicted as negative.
False Negative (FN) represents the number of positive samples predicted as negative.
False Positive (FP) shows the number of negative samples predicted as positive.

Accuracy can be defined as the proportion of correctly classified samples to the total number of samples. Accuracy can be misleading if the dataset is imbalanced and therefore, we need to consider other metrics based on confusion matrix. Precision can be defined as the ratio of correctly classified positives (TP) to the total predicted positives (TP+FP). On the other hand, Recall can be defined as ratio of correctly classified positives (TP) to the total number of samples that are positive. Precision and recall are good metrics to evaluate a classifier but there is a trade-off. Hence to find the equilibrium between Precision and recall, we use a metric called F-measure. To fully examine the effectiveness of the model, we need to consider both precision and recall. Additionally, the AUC/ROC curve helps visualize the performance of a classification model at all classification thresholds.

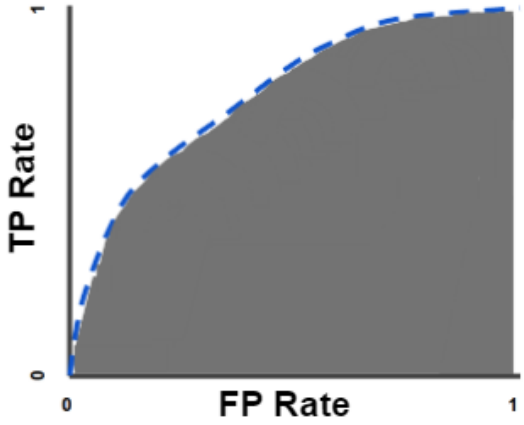


Figure 5 AUC (Area under ROC curve)

7.4 Comparing the classification model with existing works

We will additionally study recent works performed on classification of melanoma. We will also optimize and fine tune the models on their parameters to improve the results. This additional study on recent works will help us to create a good benchmark to compare against our model.

8. Requirements Resources

Hardware resources

To perform the research, a laptop with the following specifications will be used.

- Processor: AMD Ryzen 5 3500
- RAM: 16 GB
- 15-20 GB of storage, for the dataset, libraries and storing models
- GPU GTX 1660 which would be sufficient for pre-processing, analysis, and model training.

Software resources

- For Development, we will use Jupyter Notebook and keep Google Colab for contingency.
- Major Packages include Pandas, Numpy PyTorch and Tensorflow (contingency)

9. Research Plan

Below chart highlights plan for this research work

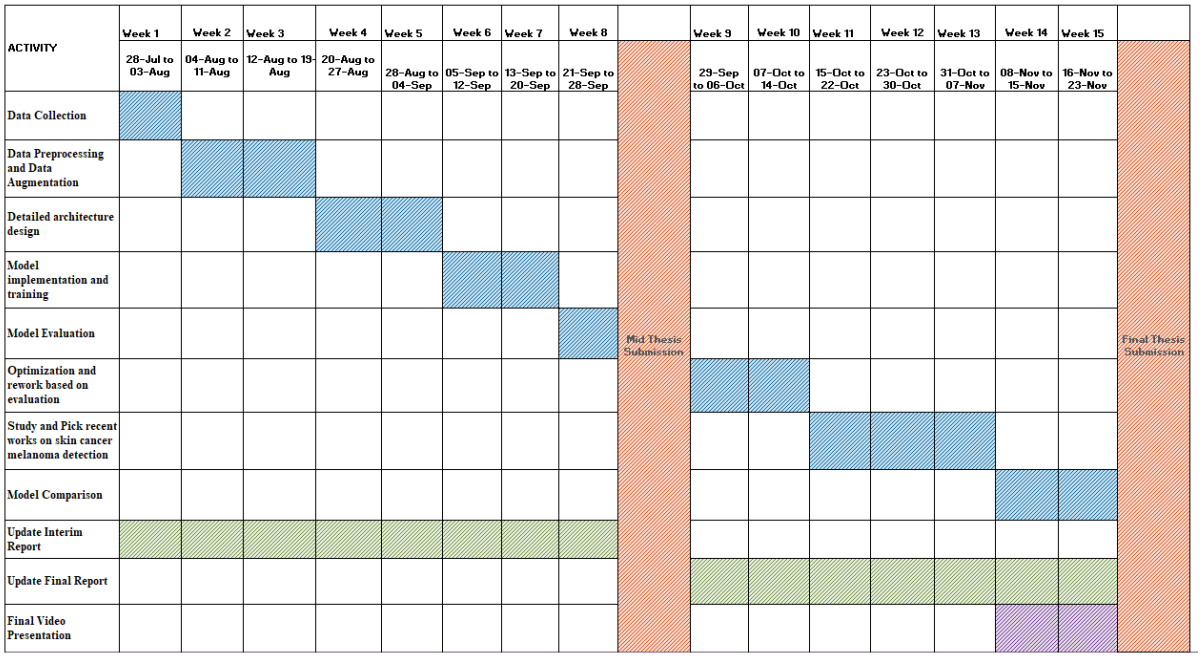


Figure 6 Research Plan Schedule

References

Araujo, A., Norris, W. and Sim, J., (2019) Computing Receptive Fields of Convolutional Neural Networks. *Distill*, [online] 411. Available at: https://distill.pub/2019/computing-receptive-fields [Accessed 25 Jul. 2022].

Cancer Research UK, (2022) *Melanoma skin cancer statistics*. [online] Available at: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/melanoma-skin-cancer#heading-Four [Accessed 25 Jul. 2022].

Codella, N.C.F., Gutman, D., Celebi, M.E., Helba, B., Marchetti, M.A., Dusza, S.W., Kalloo, A., Liopyris, K., Mishra, N., Kittler, H. and Halpern, A., (2017) Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC). [online] Available at: http://arxiv.org/abs/1710.05006.

Combalia, M., Codella, N.C.F., Rotemberg, V., Helba, B., Vilaplana, V., Reiter, O., Carrera, C., Barreiro, A., Halpern, A.C., Puig, S. and Malvehy, J., (2019) BCN20000: Dermoscopic Lesions in the Wild. [online] Available at: http://arxiv.org/abs/1908.02288.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N., (2020) An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. [online] Available at: http://arxiv.org/abs/2010.11929.

Guo, J., Tang, Y., Han, K., Chen, X., Wu, H., Xu, C., Xu, C. and Wang, Y., (2021) Hire-MLP: Vision MLP via Hierarchical Rearrangement. [online] Available at: http://arxiv.org/abs/2108.13341.

Hou, Q., Jiang, Z., Yuan, L., Cheng, M.-M., Yan, S. and Feng, J., (2021) Vision Permutator: A Permutable MLP-Like Architecture for Visual Recognition. [online] Available at: http://arxiv.org/abs/2106.12368.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., (2017) ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, [online] 606, pp.84–90. Available at: https://dl.acm.org/doi/10.1145/3065386.

Kumar, N.S. and Ramaswamy Karthikeyan, B., (2021) Diabetic Retinopathy Detection using CNN, Transformer and MLP based Architectures. In: *ISPACS 2021 - International Symposium on Intelligent Signal Processing and Communication Systems: 5G Dream to Reality, Proceeding*. Institute of Electrical and Electronics Engineers Inc.

Li, D., Hu, J., Wang, C., Li, X., She, Q., Zhu, L., Zhang, T. and Chen, Q., (2021) Involution: Inverting the Inherence of Convolution for Visual Recognition. [online] Available at: http://arxiv.org/abs/2103.06255.

de Lima, L.M. and Krohling, R.A., (2022) Exploring Advances in Transformers and CNN for Skin Lesion Diagnosis on Small Datasets. [online] Available at: http://arxiv.org/abs/2205.15442.

Liu, H., Dai, Z., So, D.R. and Le, Q. v., (2021) Pay Attention to MLPs. [online] Available at: http://arxiv.org/abs/2105.08050.

Lv, T., Bai, C. and Wang, C., (2022) MDMLP: Image Classification from Scratch on Small Datasets with MLP. [online] Available at: http://arxiv.org/abs/2205.14477.

Lyu, H., Wang, Y., Tan, Y., Zhou, H., Zhao, Y. and Zhang, Q., (2022) Boosting Adversarial Transferability of MLP-Mixer. [online] Available at: http://arxiv.org/abs/2204.12204.

Narayanamurthy, V., Padmapriya, P., Noorasafrin, A., Pooja, B., Hema, K., Firus Khan, A.Y., Nithyakalyani, K. and Samsuri, F., (2018) Skin cancer detection using non-invasive techniques. *RSC Advances*, [online] 849, pp.28095–28130. Available at: http://dx.doi.org/10.1039/C8RA04164D.

PDQ Adult Treatment Editorial Board, (2002) *PDQ Melanoma Treatment*. [online] Bethesda, MD: National Cancer Institute. Available at: https://www.cancer.gov/types/skin/hp/melanoma-treatment-pdq [Accessed 25 Jul. 2022].

Shen, S., Yao, Z., Gholami, A., Mahoney, M.W. and Keutzer, K., (2020) PowerNorm: Rethinking Batch Normalization in Transformers. [online] Available at: http://arxiv.org/abs/2003.07845.

Subramanian, R.R., Achuth, D., Kumar, P.S., Naveen kumar Reddy, K., Amara, S. and Chowdary, A.S., (2021) Skin cancer classification using Convolutional neural networks. In: *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. [online] IEEE, pp.13–19. Available at: https://ieeexplore.ieee.org/document/9377155/.

Tang, C., Zhao, Y., Wang, G., Luo, C., Xie, W. and Zeng, W., (2022) *Sparse MLP for Image Recognition: Is Self-Attention Really Necessary?* [online] Available at: www.aaai.org.

Tatsunami, Y. and Taki, M., (2022) Sequencer: Deep LSTM for Image Classification. [online] Available at: http://arxiv.org/abs/2205.01972.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M. and Dosovitskiy, A., (2021) MLP-Mixer: An all-MLP Architecture for Vision. [online] Available at: http://arxiv.org/abs/2105.01601.

Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Izacard, G., Joulin, A., Synnaeve, G., Verbeek, J. and Jégou, H., (2021) ResMLP: Feedforward networks for image classification with data-efficient training. [online] Available at: http://arxiv.org/abs/2105.03404.

Tschandl, P., Rosendahl, C. and Kittler, H., (2018) The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, [online] 51, p.180161. Available at: http://www.nature.com/articles/sdata2018161.

Wei, G., Zhang, Z., Lan, C., Lu, Y. and Chen, Z., (2022) ActiveMLP: An MLP-like Architecture with Active Token Mixer. [online] Available at: http://arxiv.org/abs/2203.06108.

Zhao, Y., Wang, G., Tang, C., Luo, C., Zeng, W. and Zha, Z.-J., (2021) A Battle of Network Structures: An Empirical Study of CNN, Transformer, and MLP. [online] Available at: http://arxiv.org/abs/2108.13002.