Universitat Oberta de Catalunya

Comparison of different neural network architectures for automatic classification of peripheral blood cell images

Anna Foix Romero

Machine Learning
Máster en Bioinformática y Bioestadística

Advisor: Edwin Santiago Alférez Baquero

27th of May 2022

FINAL WORK CARD

Title:	Comparison of different neural network architectures for automatic classification of peripheral blood cell images
Author:	Anna Foix Romero
Tutor:	Edwin Santiago Alférez Baquero
SRP:	Edwin Santiago Alférez Baquero
Date of delivery:	27th of May 2022
Studies:	Máster en Bioinformática y Bioestadística
Area:	Machine Learning
Language:	English
Number of credits:	15
Keywords:	Deep learning, biomedical images, deep learning, pathology, cell morphology, artificial intelligence

Abstract

The classification of blood cell types in microscope imagery is a key problem in healthcare. Traditional cell microscope morphology is expensive and error-prone, and so automating this is a popular area of research. This thesis performs a comparative analysis of six different deep learning techniques for classifying different blood cell types, to identify the best performing architecture; it also examines the impact of training data size on the quality of each architecture, to study how these architectures can be used while keeping the training dataset small. We find that without pre-training, ResNet is the most successful architecture in terms of accuracy and robustness with small datasets, achieving an accuracy of 95.9% on the largest dataset and an accuracy of 79.1% on a very small dataset; MobileNetV3 achieves even higher performance of 97.1% on the largest dataset but breaks down on the very small dataset. With pre-training, ConvNeXt has the best performance on the very small dataset, achieving an accuracy of 87.8%.

Contents

List of Figures

List of Tables

Chapter 1

Summary

A key problem in the field of healthcare is working with microscopic images of blood cells. From these images, a pathologist can understand the degree to which different types of cell are present in a patient's blood, as well as whether the cells are exhibiting signs of abnormality. However, identifying which types of cells are in an image is an expensive and error-prone process requiring specialist training, and so automating this process is a popular area of research, particularly with the advent of deep learning techniques.

This thesis performs a comparative analysis of different state-of-the-art machine learning techniques for classifying different blood cell types. An appropriate dataset of blood cell images is selected, processed, and then used to train multiple configurations of six different neural network architectures, on varying subset sizes of the dataset. The quality of each resultant model is assessed and contrasted against other models, comparing different architectures trained on several sizes of training dataset.

The results presented show that ResNet and MobileNet V3 are the two most successful architectures overall, with MobileNet V3 achieving classification accuracy of over 97% when trained on a dataset of over 8500 images, and ResNet delivering an accuracy of 79.1% on a tiny dataset of only 200 images. It is also found that ResNet shows the most robust response to decreasing data set size, with a linear but slight decrease in accuracy from 8500 to 2100 training images, and then a slightly accelerated decrease below that. By contrast, ConvNeXt shows the worst response to decreases in training data size, with the smaller configurations significantly dropping in accuracy even with 4200 images in the training data.

In conclusion, when considering models that are trained from scratch, ResNet is the best architecture in terms of both accuracy and ability to provide good results with small datasets. MobileNet V3 is also good, but lacks ResNet's robustness when the dataset is very small. Once pre-training is applied, the best performing architecture on the tiny dataset out of ResNet, MobileNet and ConvNeXt is ConvNeXt.

This thesis contributes to the evaluation and the comparison of each of several architectures when applied to the classification of peripheral blood smear microscopy and to the constraint of small data sets, providing a simple baseline comparison for further research and development. It also provides an implementation of the training process that can be used to recreate or extend the results.

Chapter 2

Introduction

2.1 Context and motivation

In the field of healthcare, the study of blood-based conditions often involves capturing images of blood cells for study and analysis. Visual inspection of the blood cells by the use of peripheral blood smears and optical microscopes can provide insight into conditions such as sickle cell disorders [21], acute myeloid leukemia [15], and many others.

However, blood samples contain multiple different types of cells: red blood cells, white blood cells such as Neutrophils, Basophils, Monocytes, Eosinophils and Lymphocytes, and platelets. Identifying the type of cells is critically important, since most pathological analyses are only interested in particular cells. Besides, understanding the proportion of different types of cell that are present in a sample is also often informative; knowing that a patient has an abnormal proportion of a particular cell type may be crucial to diagnosing a pathology.

While it is possible for a trained pathologist to identify the cell types from images, this is a very slow, tedious, and costly process, and suffers from high subjectivity [13]. Automatic identification of the cell types is therefore an attractive alternative, and has been the focus of much research in recent years, including the use of modern techniques such as deep learning. For example, Convolutional Neural Networks (CNNs) have been used to automatically classify red blood cells [35], and CNNs with Long Short-Term Memory (CNN-LSTMs) have been explored for classification of white blood cells [37]. This work will extend the previous studies by seeking to classify images across a broader range of cell types simultaneously, and by exploring a more diverse set of deep learning approaches.

In addition, the disadvantages of manual identification of cell types also means that obtaining labelled training data sets is a problem, as it requires significant manual labour. While there are some large globally available datasets that can be reused, these result in models which may be less suitable for diagnosis of specific communities or patients. If we can find a way to reduce the amount of data that is needed to train a model and still have high performance, it will make it more possible to train models that are specialised to smaller populations - or even to a single individual - without requiring a large amount of manual inspection and labelling of images to create the training data.

2.2 Objectives of the thesis

The main objective of this masters' thesis is to develop a comparative analysis of different neural network architectures, as applied to the problem of classifying cell images from peripheral blood smears. In addition, this thesis aims to find a model which can produce an accurate classification even with a smaller training data set.

From the above, the following secondary objectives can be planned:

- To review the literature, including elements of the kinds of features that are used to distinguish cells in manual classification, as well as an exploration of prior work on classifying blood cells using deep learning techniques.
- To select an appropriate data set, and implement appropriate preprocessing methods.
- To create a framework for training and evaluating the performance of each neural network architecture, including the selection of appropriate performance metrics.
- To use the framework to evaluate at least 3 different neural network architectures, across at least two different datasets of substantially different size.

2.3 Methodology

For obtaining the datasets, options include searching for existing published datasets, or collaborating with research laboratories to obtain their image sets. In the interests of time, the former option will be used; there are multiple such datasets available publicly, though the dataset from Acevedo et al [2] is the most promising option for this thesis. This labelled dataset consists of 17,092 images across 8 cell types (neutrophils, eosinophils, basophils, lymphocytes, monocytes, immature granulocytes, erythroblasts and platelets). This is a relatively large dataset; to produce a smaller dataset, stratified sampling will be used to select a subset of the images from the dataset, while still maintaining the distribution between cell types.

For developing the framework, a wide body of work that demonstrates possible ways of training and evaluating a neural network architecture exists and can be used for inspiration. Performance metrics will include accuracy metrics suitable for multi-class classification, as well as time needed to train.

There are many deep learning architectures that could be evaluated; the architectures of particular interest include Convolutional Neural Networks (CNN) [51], Vision Transformer (ViT) [38, 46], and MLP-Mixer [43]. If time permits, the thesis will also be extended to include other architectures such as ResNet [49] and ConvNeXt [30]. The PyTorch library [36] has been chosen as the technological foundation to use for implementations of all of these architectures, as it provides a greater level of control and visibility into the training process compared to popular alternatives such as Keras. PyTorch also includes the 'torchvision' library, which offers ready-made implementations of a number of the neural network architectures of interest.

Once each configuration of each architecture has been trained, performance metrics can be gathered for each configuration, and these performance metrics will then be tabulated, compared, and discussed.

2.4 Work planning

Work has been planned based on a capacity of 5 working days per week. This is to better reflect the fact that working 7 days a week is not realistic, nor is it a healthy thing to do for a duration as long as this thesis project.

2.4.1 Tasks

The following table shows the main tasks that have been identified and the estimated duration of work associated with each one.

Thesis proposal	6 days
Work plan	9 days
Literature review and analysis	10 days
Creation of smaller test dataset	3 days
Development of training and evaluation framework	11 days
Implementation and evaluation of CNN	5 days
Implementation and evaluation of Transformer	5 days
Implementation and evaluation of MLP-Mixer	5 days
Documentation	20 days

Table 2.1: Breakdown of tasks and time estimates

2.4.2 Calendar

The start of work is February 16th. A Gantt chart illustrating the expected progression of work over the thesis period is shown below. Each week is from Monday to Friday, so the first week is shortened (as February 16th is a Wednesday).

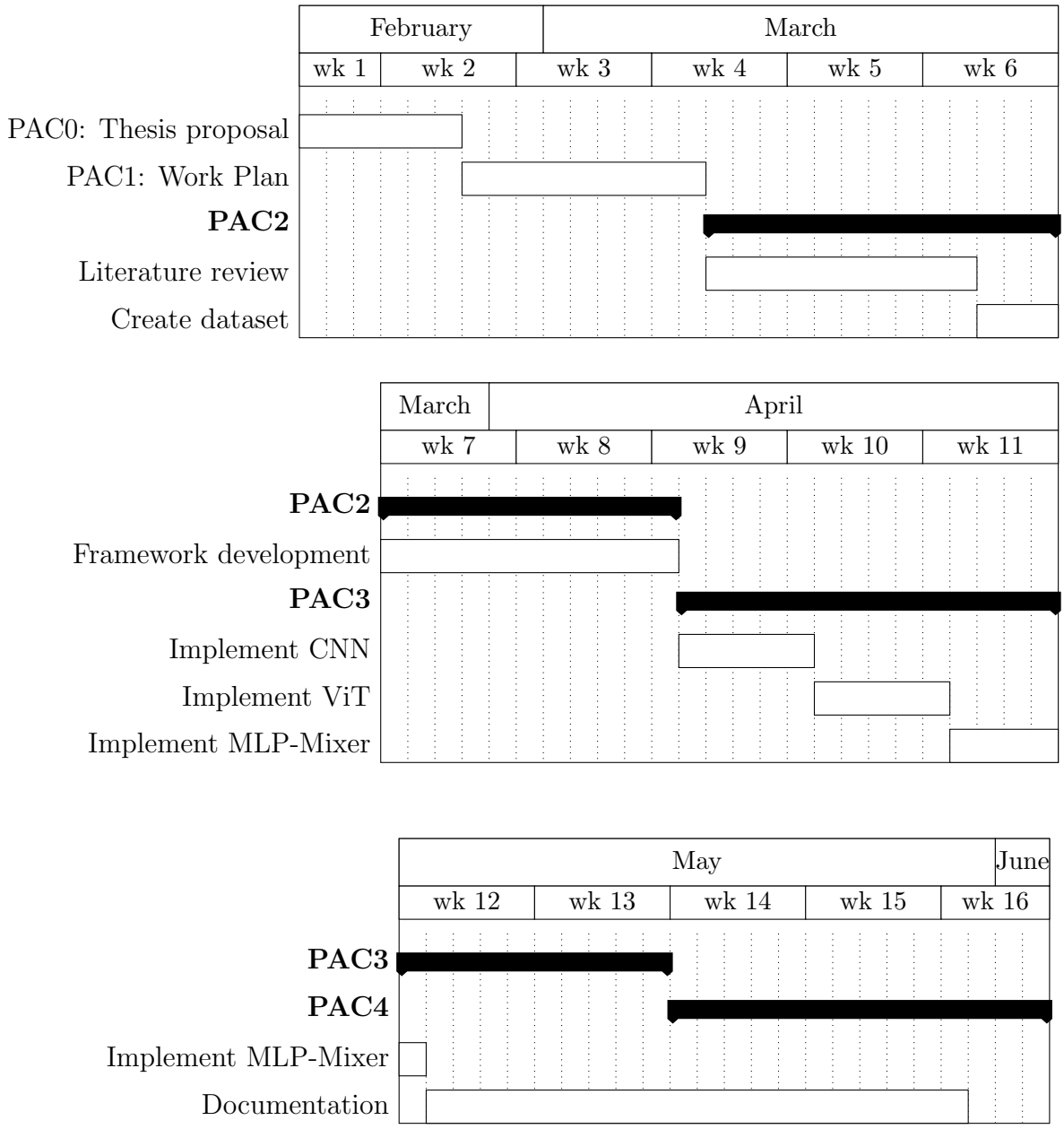

Figure 2.1: Gantt chart of planned project work

2.4.3 Milestones

The key dates at which milestones are expected to be hit are shown in table 2.2.

Thesis proposal submitted	23rd February
Work plan created	8th March
Literature review completed	22nd March
Test datasets prepared	25th March
Framework completed	11th April
CNN evaluated	18th April
ViT evaluated	25nd April
MLP-Mixer evaluated	2nd May
Report completed	30th May

Table 2.2: Milestone dates

2.4.4 Risk Analysis

The risks to this project are limited, as the data is available and the network architectures to implement are well known. At this stage the clearest potential source of problems is the time taken to train each network, particularly when dealing with the larger dataset. To mitigate this, the CUDA acceleration features in PyTorch will be used, which can greatly accelerate the training process; still, it may end up being necessary to exclude some of the largest models if they cannot be fully trained in time.

2.4.5 Resources and equipment

The primary resources this project needs are computational. An estimate of the resources needed is shown in table 2.3. All costs are in British pounds.

Resource	Estimated cost
GeForce 3080 RTX GPU	£1000
Generic x64 computer	£650
Power consumption	£35
Total	£1685

Table 2.3: Required resources and estimated costs

Power consumption is based on an estimate of around 7 full days of 750W consumption, at a unit price of £0.28 per kilowatt-hour. A solid week of processing time should be enough for training and testing all of the different configurations.

2.5 Brief summary of contributions

Chapter 3 discusses the state of the art, including both the use of deep learning for blood cell identification and also an exploration of deep learning techniques more generally.

Chapter 4 describes the methodology used, including the decisions that make up the evaluation framework and the implementation in PyTorch.

Chapter 5 presents the results of the evaluation process across each of the dataset and network configurations tested.

Chapter 6 discusses the results and the insight they provide into the original research questions.

Chapter 7 summarises the conclusions found by the thesis.

Chapter 8 presents a glossary of terms used.

Chapter 3

State of the Art

In this chapter, we will review significant results in the field of deep learning, as well as more specific recent work in the application of deep learning to blood cell classification.

3.1 Deep learning and Convolutional Neural Networks

While the use of neural network-based machine learning techniques dates back to at least the 1960s, for a long time the field was limited by the need to explicitly design and extract features from the data to feed to the network. For example, an image classification problem might be approached by measuring the mean hue and brightness in each quadrant of the image, and then passing those extracted feature values to the network for classification. As a result the performance of a network was heavily dependent on the choices made about the features to extract. Representation learning [6] and deep learning [26] eliminate the need for this manual feature extraction step; they instead provide the raw data to the network, and have the network identify features of interest within the data itself. The "deep" in deep learning refers to the way that the networks typically have many layers, which is what makes the representation learning possible.

One famous early result in this was the use of Convolutional Neural Networks (CNNs) [27, 28]. These networks involve one or more layers of 'convolution' neurons, which apply a

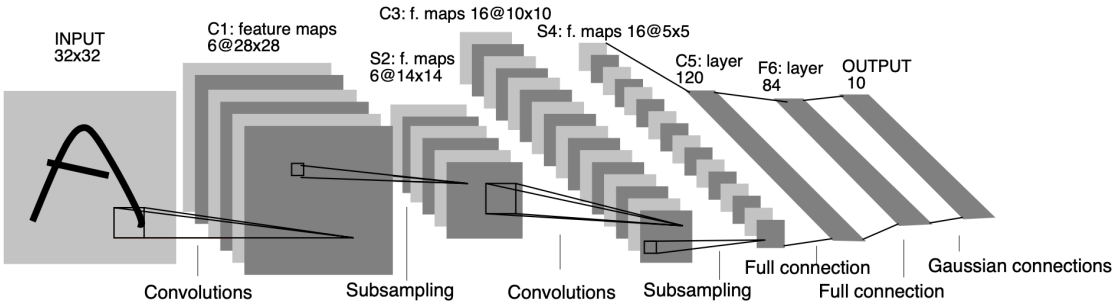

Figure 3.1: Structure of the LeNet-5 architecture. Diagram copied from [28].

shared set of weights to spatially or temporally clustered neighbourhoods of inputs. Because the weights are shared, they can learn a feature and respond to it regardless of where it is located in the input, preventing the network from overfitting on the specific locations of features in the training data. By connecting multiple layers of convolutions together, larger-scale features can be learned. The seminal LeNet-5 architecture presented in [28] used this technique to achieve a 0.95% error rate on classification of images of hand-written characters. Figure 3.1 shows a diagram of this architecture.

Many other network architectures have been developed on top of the convolution concept. The ResNet architecture [18] is a CNN that takes the input to each group of layers and adds it directly to the output from those layers via a 'shortcut connection,' effectively changing their role from learning the full mapping of the input into learning the 'residuals' of that mapping relative to the input. Each block in the structure has a layout like that shown in figure 3.2. The architecture is particularly interesting because the change is purely structural; the shortcut connections do not introduce any new parameters and so the network has the same number of parameters (and therefore the same memory consumption) as a pure CNN. This approach has shown much greater accuracy compared to pure CNNs of equal depth, particularly as the depth of the network increases; when tested on the ImageNet database, the 34-layer configuration achieved an error rate of 7.4% (while a 34-layer plain CNN only achieved 10.02%), and the 152-layer configuration achieved an error rate of 4.49%. ResNet was investigated further in [49], where different hyperparameters were explored such as number of epochs and learning rate; the work showed that the performance of ResNet could be improved further by changing the hyperparameters alone.

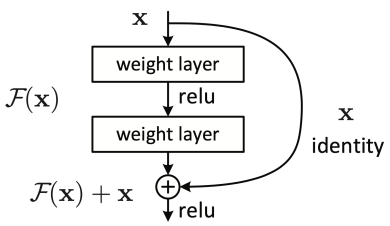

Figure 3.2: Structure of a ResNet block. By copying the input x to the output via the shortcut connection, the weight layers are only required to learn the residual function $\mathcal{F}(x)$. Diagram copied from [18].

One downside to both plain CNNs and ResNet is that it involves many layers being applied sequentially, which means that the critical path for evaluating a network is long; and because each layer acts on the complete output of the layer before, it can be difficult to parallelise or distribute the computation. The Inception architecture [41] uses CNNs but applies groups of layers in parallel, rather than in series; a single Inception block includes CNN layers with kernel sizes (neighbourhood sizes) of 1, 3, and 5, executed in parallel and then combined. These blocks are then stacked into sequential layers, but a 22-layer implementation of the architecture was able to obtain an error rate of 6.67%, comparable to ResNets while using fewer layers and so being easier to train and evaluate in parallel. The Inception architecture was developed further in [42], achieving an error rate of 3.5%. Subsequently, the Xception architecture [11] was presented, replacing each Inception module with a depthwise separable convolution, which the authors argue is an Inception module taken to an "extreme" extent: while InceptionV3 partially decouples cross-channel and spatial correlations, the Xception architecture decouples them completely. They were able to observe a slight improvement over the performance of InceptionV3 when classifying the ImageNet database, and a more significant improvement when training on a larger dataset (350 million images over 17,000 classes).

In addition to advancing the structure and accuracy of CNN architectures themselves, work has also been done to try and understand the intermediate values being transferred within the network. These values are typically very difficult to interpret intuitively, but [51] demonstrated the use of a 'deconvolutional' network - effectively a convolutional network applied in reverse - to convert the feature representation available at intermediary layers of the CNN back into an image representation. These visualisations demonstrate how the features learned by the network are not unintelligible patterns, but recognisable shapes; they also demonstrate how the visualisations can be used to debug and validate the network, e.g. by checking how occluding background parts of the image does or does not affect the significant features.

There are also still opportunities to improve on the basic CNN concept itself. For example, [12] extends the standard convolution kernels by applying learnable offsets to the positions where the kernel is applied, so that instead of filtering a fixed grid of NxN positions, the grid can be perturbed. This allows the network to select more complicated features, and the authors reported a moderate improvement ($\sim 5\%$) in mean average precision when the technique is applied to various CNN architectures.

3.2 Attention-based architectures

In parallel to CNNs, a recent fruitful area of discovery has been the use of attention-based methods. These originate in the field of machine translation, where a recurrent Encoder-Decoder approach [10] was the previous state of the art: each word in the input sentence was successively presented as an input to a recurrent neural network, producing a state vector that represented the sentence as a whole; this state vector was then presented a second recurrent neural network which could then produce the sequence of words in the output sentence. While this approach was somewhat successful, it performed poorly with long sentences, particularly because the state vector acted as a bottleneck on the amount of information that could be passed from the encoder to the decoder.

A new mechanism of 'attention' [5] was presented to solve this. In this model, the intermediate states for each of the encoded words are retained and made available to the decoder, weighted by an 'attention' score which allows the network to use the most relevant information from across the entire sentence without forcing it all to have been encoded into the state vector. Subsequently, the Transformer architecture [46] took this approach even further and actually removed the recurrent aspect of the architecture, instead simply computing the importance of each word relative to the others. By making the architecture not be recurrent, a Transformer network is parallelizable, allowing the model to be trained and evaluated much more quickly than a recurrent model of the same size. The authors trained the network on batches of approximately 25000 words, and achieved BLEU scores that exceeded all previous models, while requiring an order of magnitude less computing power to train.

While the Transformer architecture was originally created for text translation, the Vision Transformer (ViT) [16] architecture adapted it for use in image recognition and classification problems. The structure is shown in figure 3.3. In the ViT architecture, the input image is split into rectangular 'patches' of pixels, and each patch is mapped to an embedding; the

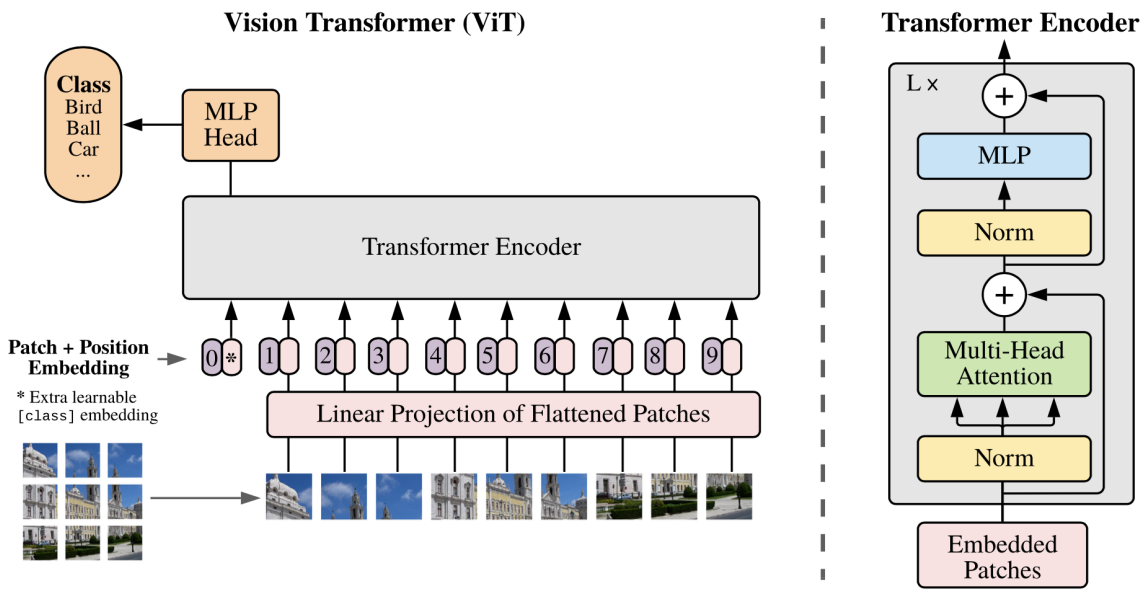

Figure 3.3: Structure of the Vision Transformer architecture. Diagram copied from [16].

image is then presented to the Transformer as a 'sentence' made up of these embeddings, to be 'translated' into an image class. The authors evaluated their implementation, training each model on different datasets ranging from 1.3 million images up to 303 million images. They were able to outperform the previous state of the art networks, but only when training on the largest dataset; however they were also able to complete this training in around 25% of the time. When smaller datasets were used for training, the training process was naturally much faster, but the classification accuracy was not quite as good. It was also shown that the ViT approach is quite robust, handling problems such as image corruption, perturbations, multiple objects in the image, partially masked or occluded objects, and so on [38].

Following the presentation of the Transformer architecture, the MLP-Mixer architecture [43] was presented, shown in figure 3.4. The authors theorised that it may be possible to achieve good results with a much simpler approach, not using convolution or attention, but only very simple multi-layer perceptrons. The same approach in ViT of breaking the image down into 'patches' is used, but then these are put through a series of alternating 'channel-mixing' layers, where the channels in each patch are connected but each patch is processed independently, and 'token-mixing' layers, where the data from each patch is connected together but the channels are kept separate. Each of these layers is a simple fully connected linear layer, combined with a gaussian error linear unit activation function, so the overall structure of the architecture is very simple and quite easy to parallelise. The authors pretrained the model on two datasets, one containing 14 million images over 21,000 classes and one containing 300 million images over 18,000 classes, and then 'fine-tuned' it to a number of smaller datasets to evaluate performance for each of those datasets. Compared to ViT, MLP-Mixer performance was generally slightly worse, but very close, while being typically much faster to train. Subsequently, the gMLP architecture [29] introduced a limited attention mechanism to MLPs in the form of a 'spatial

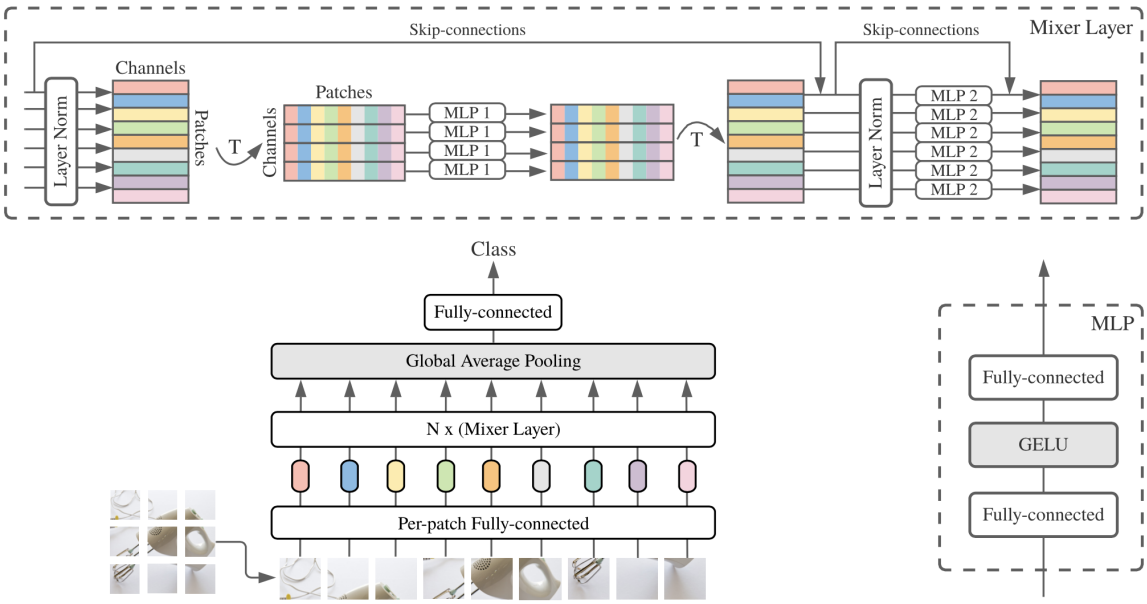

Figure 3.4: Structure of the MLP-Mixer architecture. Diagram copied from [43].

gating' mechanism, achieving a 3% improvement in accuracy while reducing the number of parameters by 66%.

While ViT has been very successful as an architecture for image classification problems, there are still many questions about why it performs as well as it does. One theory is that the high performance of the architecture is not because of the Transformer architecture underlying it, but simply because of the decision to split the input image up into rectangular patches, and that applying that concept back to traditional CNNs would also be successful. The ConvMixer architecture [44] implements this idea, with a design that is similar to the MLP-Mixer architecture but uses convolutions instead of fully-connected layers. When testing this architecture on the ImageNet-1K dataset, and using the hyperparameters from [49] they were able to demonstrate accuracy significantly higher than ResNet, ViT and MLP-Mixer. Separately, the ConvNeXt architecture [30] takes some of the innovations developed in the advances of the Transformer architecture and use them to 'modernise' a ResNet, achieving error rates equal to or even better than Transformer, while also achieving a higher training and evaluation speed.

3.3 Identification of blood cells

Manual identification of blood cells from peripheral blood smear imagery is based on a number of factors, including the presence or absence of a nuclear and cytoplasmic granules, cell size, nuclear size and shape, chromatin appearance, and cytoplasmic staining [39]. For example, erythrocytes (red blood cells) typically have a uniform size and shape, and do not contain organelles or granules, while neutrophils can be identified by a nucleus which is segmented into three to five lobes and the presence of multiple sizes of granules. If the cells are unhealthy then

the typical characteristics may be altered, for example in images from a patient with sickle-cell disease.

Even if the cells are healthy, it can be often be difficult to identify cells due to problems with the images: overlapping cells, blurry images, poor contrast, etc. CNNs have been applied to identification of abnormal red blood cells in this environment [35]; following image preprocessing techniques such as edge detection, a 13-layer CNN with 3×3 kernels was able to obtain accuracy of 98.5% after only 5 epochs of training.

Classification of white blood cells is a more difficult problem, as the cells are more diverse and have more distinguishing features that must be taken into account. Previous work has primarily focused on variations of CNN architectures. In 2019, Acevedo et al [1] studied the use of the Vgg-16 and InceptionV3 CNN architectures in classifying images across 8 cell types, and achieved global classification accuracy values of 96% and 95% on the two architectures respectively.

Many attempts focus on enriching or preprocessing the data that is fed to the CNN. In one study, canonical correlation analysis was used [50] to combine pairs of images of the same cell, synthesising a more 'three-dimensional' understanding of the cell; this combined view of the cell is then fed to a CNN, and subsequently to a Long Short-Term Memory (LSTM). The authors used 9957 images to train the network, evaluating different architectures for the CNN component; they found their best results with the Xception architecture [11], achieving precision and recall scores of 96.2% and 97%.

In another study presenting the Faster R-CNN technique [22], the Mask RCNN [17] approach was leveraged and extended, and applied to notably smaller dataset of only 145 cells across 87 images, further reduced by a 70%/30% train/test split. They evaluated multiple different backbone architectures, observing that ResNet50 gave them the best results, though they only achieved a maximum precision of 95.7%. This work was then extended further [9], employing a multi-level approach: an initial Faster R-CNN network was used to separate mononuclear from polymorphonuclear cells, and then a different MobileNet CNN [20] was used to classify the specific cell type within each group. This approach was able to obtain a mean accuracy of 98.36% across 4 cell classes.

Deformable CNNs [12] have also been applied to the classification of white blood cells [50], combined with a transfer learning approach [7]. Initially a 16-layer CNN was trained on a dataset of 11865 high-quality images; transfer learning was then used to train a second network containing 5 deformable CNN layers and 13 regular CNN layers. This approach yielded mean precision and recall values of 94.5% and 95.7% respectively.

Chapter 4

Methodology

4.1 Dataset selection

There are multiple blood cell image datasets publicly available for use in machine learning studies, but not all are suitable for the specific problem this thesis is addressing. Some datasets are more targeted at problems of image segmentation than classification, and contain many cells, often overlapping, with limited specificity in the classes used for the labels - sometimes no more detailed than 'red blood cell', 'white blood cell' and 'platelet' [4, 8]. The Raabin-WBC dataset [24] is more suitable: it consists of images that focus on individual cells, and cells are labelled up to 5 different types of cell, with the dataset also including labels for other interesting things in the images such as burst cells or artifacts. However, the dataset only contains images of white blood cells, and many of the images have not been labelled at all. Similarly, Zheng [52] offers two datasets of cell images across 5 classes, but again the datasets are limited to white blood cells only, and because the datasets were created more for the purposes of segmentation, the authors warn that the cell type labelling has not been reviewed by experts and may be inaccurate.

The paper by Acevedo et al [2] introduces the dataset that has ultimately been selected for this work (the same dataset that was previously used in [1]). The dataset contains 17,092 images acquired using a CellaVision DM96 analyser, distributed unequally across 8 classes, with each class representing a different type of cell. Figure 4.1 shows the distribution of images across those classes. While it might seem like a good idea to limit the dataset such that each class is represented equally, in practice the unbalanced proportions are more realistic; any dataset collected from peripheral blood smears in the field will not contain all cell types in equal proportion, because the blood itself does not contain all cell types in the same proportion. As such, a network architecture which is able to perform well after being trained on an unbalanced dataset like this is likely to be more useful when trained on future datasets.

The images range in size from 360×360 to 366×369 pixels. Figure 4.2 shows an instance of an image from each class. There are batch effects present in the data; most noticeably in this figure, immature granulocytes (the "ig" class) have a different exposure to the others, though it can be seen in the full dataset that similar differences in exposure are present in other classes as well.

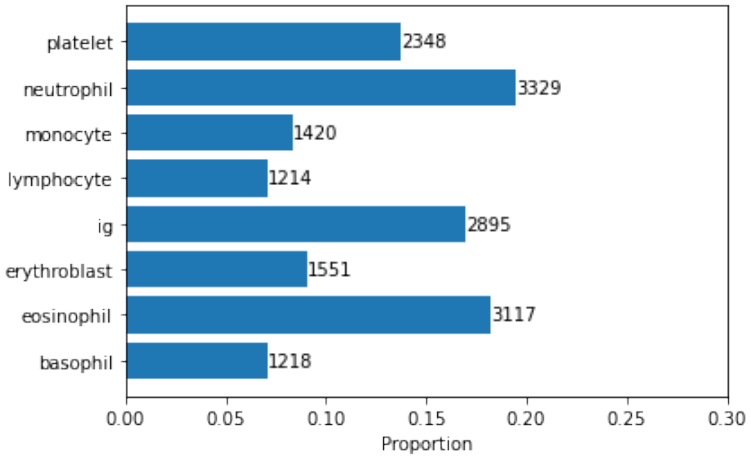

Figure 4.1: Distribution of classes in full dataset.

In order to evaluate the network architectures on smaller datasets, it is necessary to create one or more subsets of the full dataset. A stratified sampling technique is used to ensure that each smaller dataset contains the same proportion of images from each class as the full dataset, to make comparisons between the large and small evaluation results more meaningful.

4.2 Data preprocessing

While many previous studies have performed quite extensive preprocessing on the data, a decision was made to perform minimal preprocessing for a few reasons. The intention of deep learning techniques is to learn the features from the raw data, rather than relying on explicit preprocessing to generate features; as such, extensive preprocessing should not be necessary. As such the approach to preprocessing is less about extracting features and more about suppressing features that we do not wish the networks to learn.

As can be seen in figure 4.2, the images are fundamentally monochromatic. The intensity of the stain on each cell is significant, but the purple colour of the stain itself is not; it also seems a likely source of batch effects (unintended biases affecting groups of images), where different batches of stain fluid are presented with different colours. As such, the first step in the preprocessing pipeline is to convert the images to their luminance values (i.e. to greyscale). This is done using a weighted average of the three channels ($R = 0.2989$, $G = 0.587$, $B = 0.114$), as per the ITU-R recommendation [45].

The luminance values are then copied back to all three channels, as the majority of the architectures being evaluated expect a three-channel input. This may create a different result compared to altering each model to only expect 1 input channel, as in principle, different random weights are assigned to each channel and so the training process may result in each channel being processed slightly differently, forming a kind of partially 3-ensemble classifier; it also means that part of the model has 3 times the number of parameters, impacting memory usage and training time. Studying this effect is outside of the scope of the work but could be

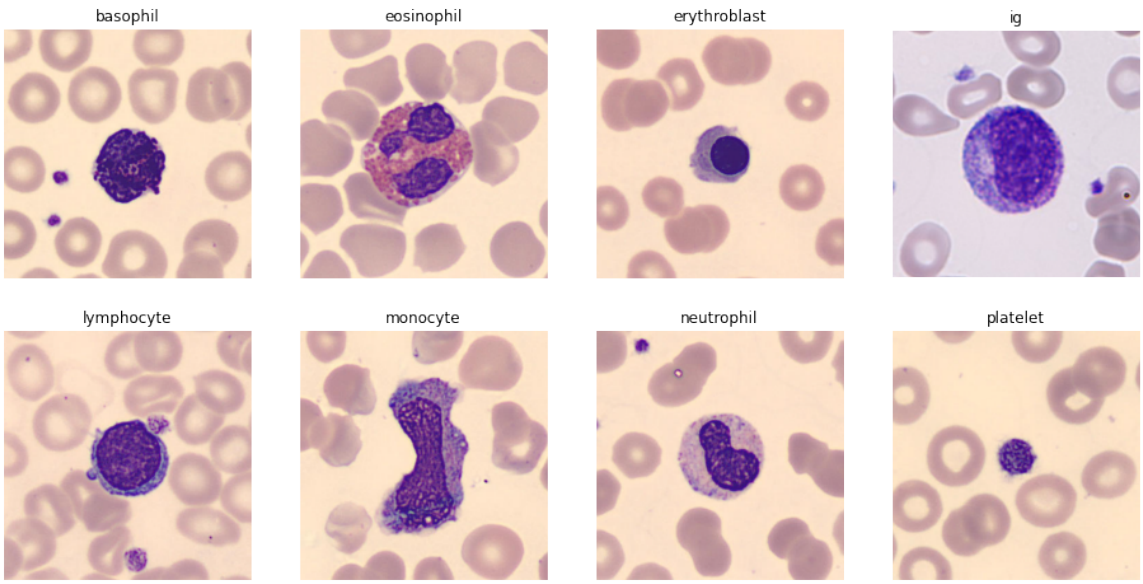

Figure 4.2: Unprocessed images from the Acevedo dataset.

an interesting future question to answer.

To provide the networks with a consistent input size, the images are then centre-cropped to 352x352 pixels. This value has been chosen so that the architectures which break the image down into patches can use patch sizes that are powers of two, the same as in the papers that presented them (22 patches along each axis for 16x16 patches, and 11 patches along each axis for 32x32 patches). The cell of interest in each image is close to the centre, so centre-cropping is appropriate.

Lastly, the values are normalised with means of 0.5 and standard deviations of 0.225 in each channel, to reduce the effect of differing exposure and contrast levels. These values were chosen as simplified, channel-uniform estimates of the ImageNet mean and standard deviation and seem to work well, though a more rigorous approach would be to measure the mean and standard deviation from the dataset and use that.

Examples of the resulting images can be seen in figure 4.3. Unfortunately, even after pre-processing, there is clearly still a difference in exposure between the images.

To counter this effect, data augmentation techniques could be used, generating additional instances of the images in each class, adjusted to varying levels of luminance. This would help to ensure that the model does not learn the average or modal luminance of the image as a predictive factor. Data augmentation was not explored in this thesis but could be explored as future work.

There is also the possibility that there may be batch effects in the form of the number of other cells being present in each image, causing the networks to learn the presence of other cells as a predictor for the class. Mitigating this problem - or even conclusively determining whether the problem exists - could be done by using image segmentation techniques to isolate the cell of interest in each image, and then discarding the background pixels. This kind of

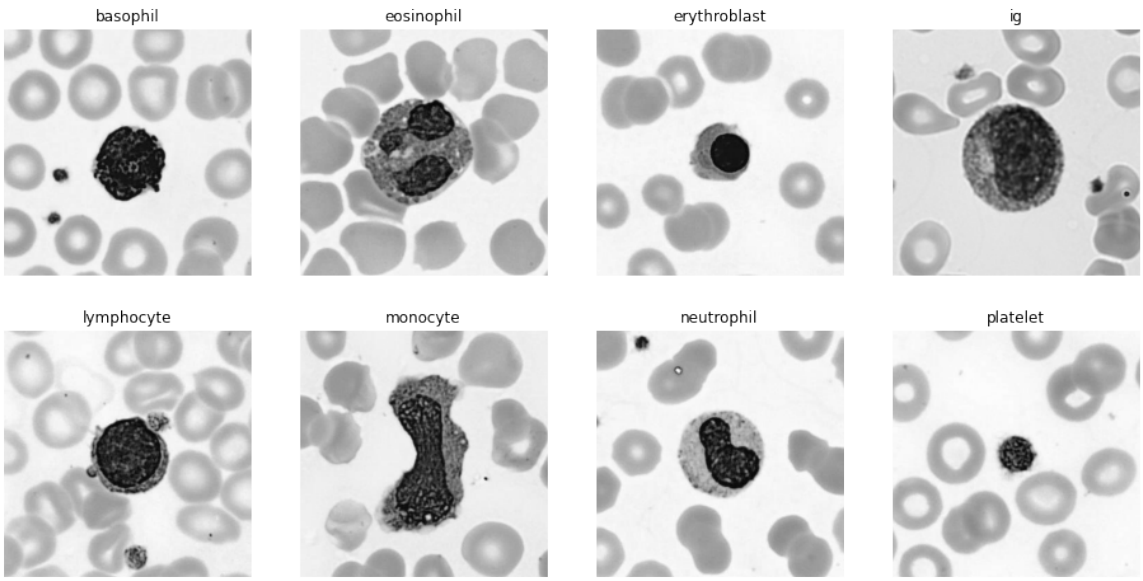

Figure 4.3: Images after preprocessing.

image segmentation problem is a field of study in itself, and outside of the scope of this work.

In addition to the image processing, the label for each image is transformed from a class index into a vector of length 8, representing the label in a one-hot encoding (a vector where the component corresponding to the correct class is set to one and the others are all set to zero).

4.3 Training and evaluation process

The dataset is split into training, validation, and test parts, in a 5:2:3 ratio; this ratio is used because a 70%/30% split between data used in the training process and test process is common in the literature, and the 70% is further broken down into 50% and 20% because this again is close to a 70%/30% split. The validation dataset is used to measure the performance of the model at the end of each training epoch, and the test dataset is used to measure the performance of the model at the end of the training process. Distinct validation and test datasets are used, rather than reusing the same dataset for both, because the the validation dataset will affect the training process by controlling the learning rate and early stopping; for an objective measurement of the quality of the model, the test dataset must not have been used to influence the training of the model in any way.

Each network is expected to output a vector of length 8, expressing for each class the probability that the image is a member of that class. Then, all the architectures share the same combined softmax and cross-entropy loss function, L_{CE}:

$$S(y)_i = \frac{e^{y_i}}{\sum_{c=1}^{8} e^{y_c}}$$

$$L_{CE}(y_{pred}, y_{true}) = -\sum_{c=1}^{8} y_{true,c} \times log(S(y_{pred})_c)$$

Intuitively, this means that the predictions made by the network are first normalised by the softmax function, and then the cross-entropy function is applied. As the one-hot encoding of the true class is zero for all components apart from the correct class, this means that the softmax value for the correct class is selected, and then the negative log is applied. The effect of this is that the loss function is minimized when output of the softmax function for the correct class is as close to 1 as possible, meaning that the network must not only produce a high value for the correct class, but also produce minimal values for other classes.

All of the different network architectures are trained using the same standard backpropagation training algorithm, implemented using the PyTorch framework [36]. Stochastic gradient descent is used to update the parameters. The core of the algorithm is shown in figure 4.4 and will be explained briefly now.

Each training epoch is broken into two phases, a training phase and a validation phase. In the training phase, the model is first placed into 'training mode' by calling `model.train()`. Then, we loop through the training data in batches. For each batch, we pass the batch of input images X to the model, computing predictions for each image. Then, these predictions are passed to the loss function, along with the batch of correct classifications, to compute the overall loss for this batch. Then, the gradient values for each parameter are reset to 0, and then the loss is back-propagated through the model, calculating new gradients for each parameter. Finally, the optimizer - a stochastic gradient descent optimizer, using the PyTorch `SGD` class with no momentum or dampening - is instructed to update all parameters by one step, the size of which is determined by the learning rate.

Once training has completed for all batches in an epoch, the validation phase begins. The model is placed into 'evaluation' mode, which disables the calculation of gradients to reduce memory usage and speed up evaluation, and also disables dropout and batch normalization. For each batch of data in the validation set, we pass the batch of images to the model to evaluate them and produce predictions, and then pass these predictions to the loss function along with the correct classifications, just as in the training phase. The resulting loss value for each batch is accumulated. The class with highest probability is calculated for each prediction and compared to the correct class values, to calculate the number of images that were predicted in the correct classes. The total number of images in the batch is also accumulated. After validation is completed, the mean accuracy and loss values are calculated by dividing the total number of correct classifications and total loss by the number of instances validated.

Finally, once the training process is completed, the model is evaluated on the data in the test dataset. The implementation of this is almost identical to the implementation of model validation at the end of each epoch. The predictions made by the model are then stored in the evaluation report, for later calculation of performance metrics.

To accelerate all training and evaluation processes, CUDA technology [34] is used, allowing the computation to be performed by the GPU far more quickly than it would be on the CPU. The primary drawback of this approach is that the model and training process are then limited by GPU memory, which is typically less than CPU memory; for this thesis a Geforce RTX 3080

```
for epoch in range(epochs):

    model.train()

    for (X, y) in train_loader:
        pred = model(X)
        loss = loss_fn(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    model.eval()

    test_loss, validated, correct = 0, 0, 0
    with torch.no_grad():
        for X, y in validation_loader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()

            validated += y.shape[0]
            correct += ((pred.argmax(1) == y.argmax(1))
                        .type(torch.float).sum().item())

    accuracy = correct / validated
    average_loss = test_loss / len(validation_loader)
```

Figure 4.4: Core training loop

GPU with 10GB of memory was used. Most significantly, this affected the usable batch size for the larger models: the batch size for each network was initially set to 64, but as this resulted in the training process running out of memory for some of the larger networks, this value was reduced for those networks by a factor of 2 until training was able to complete successfully.

4.4 Regularization techniques

In order to regularize the models and reduce the training time, the mean validation loss is used to affect the training process by two mechanisms: learning rate adjustment and early stopping.

To configure the learning rate, it was decided to adopt an adaptive approach, decreasing the learning rate in response to the mean validation loss. The initial learning rate is set to

0.1 for all networks. Then, if the loss of the model evaluated on the validation set has not decreased by more than 1×10^{-4} over the previous 10 epochs, then the learning rate is reduced by a factor of 10. This helps to reduce the problem of weights oscillating when the network is converging around a minima. The Pytorch `ReduceLROnPlateau` class is used to implement this mechanism.

In combination with the decrease in learning rate, an early stopping mechanism is also implemented. Three implementations were evaluated:

- Firstly, a function based on the range of mean validation loss over the preceding 20 epochs was used. If the total range of mean validation loss values was below 5×10^{-3} then training was halted. However, this proved to be ineffective at stopping training for some of the networks with noisier loss values, particularly when training with the smaller datasets.

- Secondly, an approach based on linear regression was tested, taking the mean validation loss over the preceding 20 epochs, eliminating the values in the upper and lower quartiles, and then fitting a linear model to the remaining data and measuring the slope. While this approach fit the intuition that we want to stop training once the loss has 'levelled off' and is no longer improving, in practice it proved too unreliable, failing to stop the training process in situations where it should have done.

- Lastly, a more standard approach was tried, where the mean validation loss at the end of each epoch is compared to the previous best mean validation loss. If the mean validation loss of the current epoch is at least 0.1 lower than the previous best, then a checkpoint is created from the current values of the model parameters and the mean validation loss of the current epoch is adopted as the new best mean validation loss. Once 20 epochs have passed with no further improvements to the best mean validation loss, training is halted. The mechanism is also disabled for the first 10 epochs, to ensure that at least 10 epochs of training are always performed. The implementation of this mechanism is shown in figure 4.5.

The third implementation was judged to be the most reliable, and so that is the method which is ultimately used in this thesis. Each network is trained for a maximum of 100 epochs, with the early stopping mechanism typically ending training before that point. Once training is completed, the model parameters are reloaded from the most recent checkpoint before the model is evaluated on the test dataset.

4.5 Implementation of models

In this work, several architectures are implemented, with multiple configurations for each architecture, generally matching the configurations described in the paper that presented each architecture:

- First, the famous LeNet-5 [28] architecture was implemented. The original implementation expects 32×32 greyscale images, so the implementation was modified to accept

```
if epoch >= 10:

    if (best_average_loss is None
        or average_loss < best_average_loss - 0.1):

        torch.save({
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict()
        }, checkpoint_path)

        best_average_loss = average_loss
        epochs_no_improvement = 0

    else:
        epochs_no_improvement = epochs_no_improvement + 1

    if epochs_no_improvement >= 20:
        break  # Exit the training loop
```

Figure 4.5: Implementation of early stopping

images of size 352×352. It is also the only network to be given a single-channel input, instead of a 3-channel input. While this architecture is relatively old and has been surpassed, it can serve as something of a baseline, as well as being a fast network to train when iterating on the evaluation framework.

- ResNet [18] was implemented with five configurations, with 18, 34, 50, 101, and 152 layers respectively.
- MobileNet V3 [19] was implemented across two configurations, corresponding to the 'MobileNetV3-Small' and 'MobileNetV3-Large' configurations defined in the paper.
- ConvNeXt [30] was implemented across 4 configurations, corresponding to the 4 configurations in the paper excluding ConvNeXt-XL. The primary difference between each configuration is the number of channels, though the 'convnext-tiny' configuration also uses fewer blocks, making it a more shallow network.
- Vision Transform (ViT) [16] was implemented across 4 configurations, based on the 'ViT-Base' and 'ViT-Large' configurations from the paper: each is implemented in two configurations, one with a patch size of 16×16 and one with a patch size of 32×32.
- MLP-Mixer [43] was implemented across 6 configurations, corresponding to the 'S/32', 'S/16', 'B/32', 'B/16', 'L/32', and 'L/16' configurations from the paper. Like the ViT con-

figurations, these configurations represent two different patch sizes across three different network sizes.

Each of these architectures are implemented using the Torchvision library [33], with the exception of MLP-Mixer, which is implemented using an open-source package for PyTorch [47], and LeNet-5, which is implemented within the code accompanying this thesis. The code for the LeNet-5 implementation is shown in figure 4.6.

4.6 Pre-training

For some of the architectures, the methodology in their presenting papers includes some form of pre-training. For the bulk of this thesis, no pre-training is performed, in order to evaluate a 'baseline' form of each architecture. This allows us to more directly compare the network architectures.

However, an additional analysis on the tiny dataset is performed for ResNet, MobileNet and ConvNet, using pre-trained models available in the torchvision library. The published weights for each model are based on training using the ImageNet dataset, for the ImageNet-1K classification problem.

For each model, the published weights are downloaded and installed. Because these parameter values are only suitable for a network configuration with 1000 output classes, the final layer of the network is then removed and replaced with a new layer that has 8 output classes. When training the model, only the parameters in that final layer are adjusted; the remaining parameters in the model are kept constant. The batch sizes, learning rates, and early stopping parameters are otherwise the same as the non-pre-trained models.

In principle, it would also be possible to evaluate the performance of ViT and MLP-Mixer after pre-training, but it is not so straightforward to adapt existing pre-trained models to the necessary configuration. The ViT models require an input image size of 224×224 pixels, and adjusting this to our larger image size is technically difficult without invalidating a large part of the network, as it alters the sequence length. The MLP-Mixer implementation used in this thesis does not offer any pre-trained models for use. Possible solutions to these problems are discussed in the section on future work at the end of this thesis.

4.7 Running of experiments

For each configuration of each implemented architecture, a model was trained on datasets at four scales, as shown in table 4.1.

Jupyter notebooks are used for running the training process for each model. The notebook makes it possible to launch the training process for each individual model, and to monitor the training progress in a dynamic graph of the model's validation loss, accuracy, and learning rate. These notebooks are then run to train each of the models and record the performance and results of testing each model.

	Training size	Validation size	Testing size	Total size	Proportion
large	8544	3415	5125	17084	100.0%
medium	4250	1700	2550	8500	49.7%
small	2100	840	1260	4200	24.6%
tiny	200	80	120	400	2.3%

Table 4.1: Sizes of datasets used for training models

At the end of each training and testing process, a report file is saved to disk using the Python 'pickle' library, containing information about the training parameters, information about each epoch of the training process such as the validation loss and accuracy at that epoch, and information about the final test of the trained model. The parameters of the trained model itself are also saved to disk for future use if needed. Creating these files on disk makes it easy to then load and calculate quality metrics in other Python scripts and notebooks. The report object itself offers a few methods to help with examining the quality of the model, such as methods to calculate accuracy or to generate confusion matrices.

```
class LeNet5(nn.Module):
    def __init__(self, num_classes):
        super(LeNet5, self).__init__()

        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(6),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2)
        )
        self.fc1 = nn.Linear(115600, 120)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(120, 84)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(84, num_classes)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)

        x = torch.flatten(x, start_dim=1)

        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x
```

Figure 4.6: LeNet5 implementation

Chapter 5

Results

5.1 Performance by dataset size

5.1.1 Large dataset

The training details and accuracy for each network trained on the 'large' dataset configuration (8544 training images) are shown in Table 5.1. All of the configurations were trained from scratch.

The configuration with the best accuracy is `mobilenet_large`, with an accuracy of 97.1%. Overall, MobileNet and ResNet have the best accuracy scores. LeNet5 has the lowest accuracy of only 70%. With regards to training time, the ConvNeXt configurations were clearly the slowest to train; this was not only due to their configurations training for a higher number of epochs than other architectures, before early stopping was applied, but also due to the higher seconds-per-epoch values compared to the other architectures. This is despite the number of parameters being comparable to other configurations; for example, `convnext_tiny` has 27 million parameters while `resnet_101` has 42 million, and yet each epoch of `convnext_tiny` training took more than three times as long to perform on average.

The previous behaviour could be explained by means of two possible reasons. One, the smaller batch sizes used - necessary to be able to fit the model and data into available GPU memory - may have slowed down the training, because the parameters are updated after each batch is completed and so a smaller batch size means that parameters are updated more often within each epoch. However, as pointed out in the previous paragraph, `convnext_tiny` took more than three times as long as `resnet_101` per epoch, despite having a larger batch size, which means that in practice it will have performed half as many parameter updates and yet still took more than 3 times as long. The other possibility is that the structural properties of ConvNeXt make it slower to train than the other architectures. The higher numbers of epochs performed prior to early stopping - and in fact for `convnext_tiny` early stopping did not apply before the limit of 100 epochs was reached - also suggests that the network is slower to converge to a stable position compared to the other architectures.

The training loss for the best configuration in each architecture is shown in figure 5.1. One immediately notable aspect is that the two best-performing architectures - `resnet_34`

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy
convnext_tiny	27826280	32	100*	39372.405	393.724	0.803
convnext_small	49460840	16	79	22896.489	289.829	0.842
convnext_base	87574664	8	68	20278.432	298.212	0.850
convnext_large	196242632	8	63	46613.753	739.901	0.840
lenet5	13885580	64	32	432.554	13.517	0.700
mlpmixer_s32	4640048	64	31	473.770	15.283	0.789
mlpmixer_b32	10909124	64	31	784.265	25.299	0.776
mlpmixer_s16	17530280	64	31	1329.452	42.886	0.818
mlpmixer_b16	30244472	32	31	2811.958	90.708	0.832
mlpmixer_l32	31283584	32	32	1527.609	47.738	0.829
mlpmixer_l16	71133928	8	31	7564.939	244.030	0.825
mobilenet_small	1526056	64	39	640.624	16.426	0.945
mobilenet_large	4212280	64	51	1508.823	29.585	0.971
resnet_18	11180616	64	40	1119.375	27.984	0.953
resnet_34	21288776	64	46	1921.404	41.770	0.959
resnet_50	23524424	32	40	3069.293	76.732	0.940
resnet_101	42516552	16	32	3923.816	122.619	0.937
resnet_152	58160200	16	31	5319.037	171.582	0.917
vit_b_16	86025992	16	39	6663.277	170.853	0.867
vit_b_32	87516680	4	31	2772.768	89.444	0.787
vit_l_16	303604744	4	36	19202.255	533.396	0.833
vit_l_32	305592328	4	43	8485.654	197.341	0.767

Table 5.1: Large dataset evaluation results. Epoch values marked with * indicate configurations where the early stopping mechanism did not end the training early.

and `mobilenet_large` - are also the least stable in early training, as they show the greatest variability in loss before converging very quickly on stable (and low) values. The ConvNeXt configuration shows a stable downward trend for most of the training process, and perhaps would have continued to improve further if the early stopping mechanism had not halted it. MLP-Mixer appears to reach a stable value quite early, where successive training only made it slightly worse; ViT also appears to have been oscillating a bit, with loss values in a consistent range for around 20 epochs.

Confusion matrices for the best configuration of each architecture are shown in 5.2. It can be seen that the most common kind of misclassification involves the `ig` class: MLP-Mixer, ViT and ConvNeXt all misclassify many `monocyte` images as `ig` images, and many `ig` images as `monocyte` or `neutrophil`. It is worth remembering that `ig` was the class where a difference in overall exposure was observed during preparation of the dataset.

Mean average precision (mAP) scores [40] were also calculated for each of these best con-

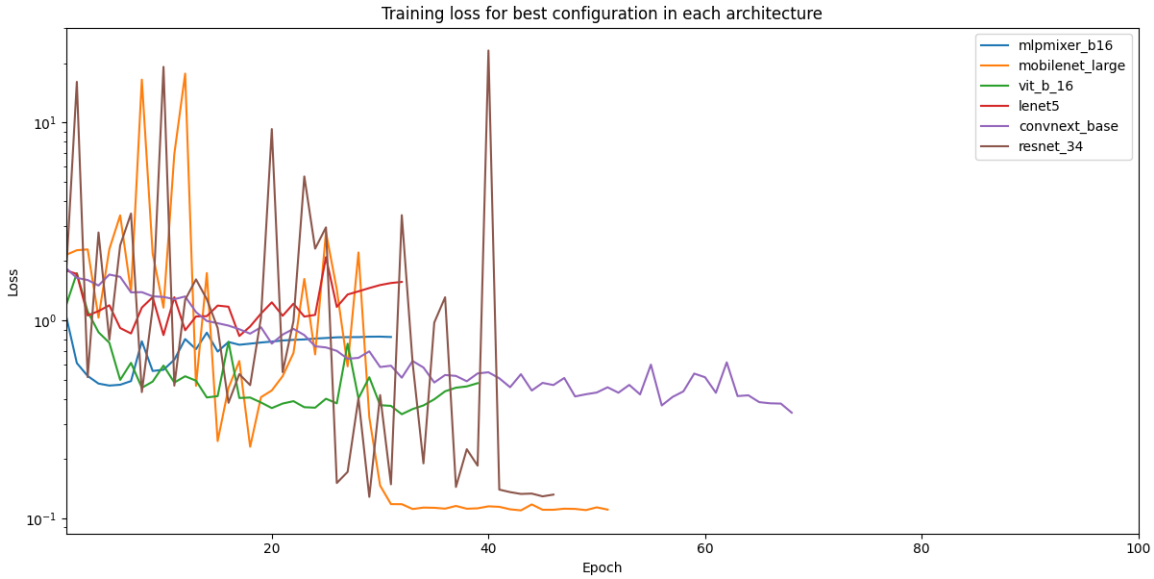

Figure 5.1: Validation loss during training on large dataset, for the best-performing configuration of each architecture.

figurations. Average precision, and mean average precision, are measurements from the field of document retrieval, and are useful when considering situations where only part of the predicted data is needed, such as a clinician executing a query such as "give me N images of class `platelet`". In this situation, the precision of the classifier in those first N images is more important than the precision of the classifier across the whole class. This is depicted in Figure 5.3.

In this case, the results that the classifier predicted for each class were sorted in descending order using the raw output value for that class, and the first N were taken and used to compute a precision score, by counting the number of those N predictions that were true positives and dividing the result by N. These precision scores were then averaged across all 8 classes to obtain the mean average precision value for the given value of N. The results of this calculation for $N = 5$ and $N = 100$ are shown in Table 5.2.

	mAP @ 5	mAP @ 100
mobilenet_large	1.000	0.99875
resnet_34	1.000	0.99625
vit_b_16	1.000	0.96875
mlpmixer_b16	1.000	0.96250
convnext_base	0.925	0.93875
lenet5	0.925	0.86750

Table 5.2: Mean average precision (mAP) scores for best classifier configurations on the large dataset configuration.

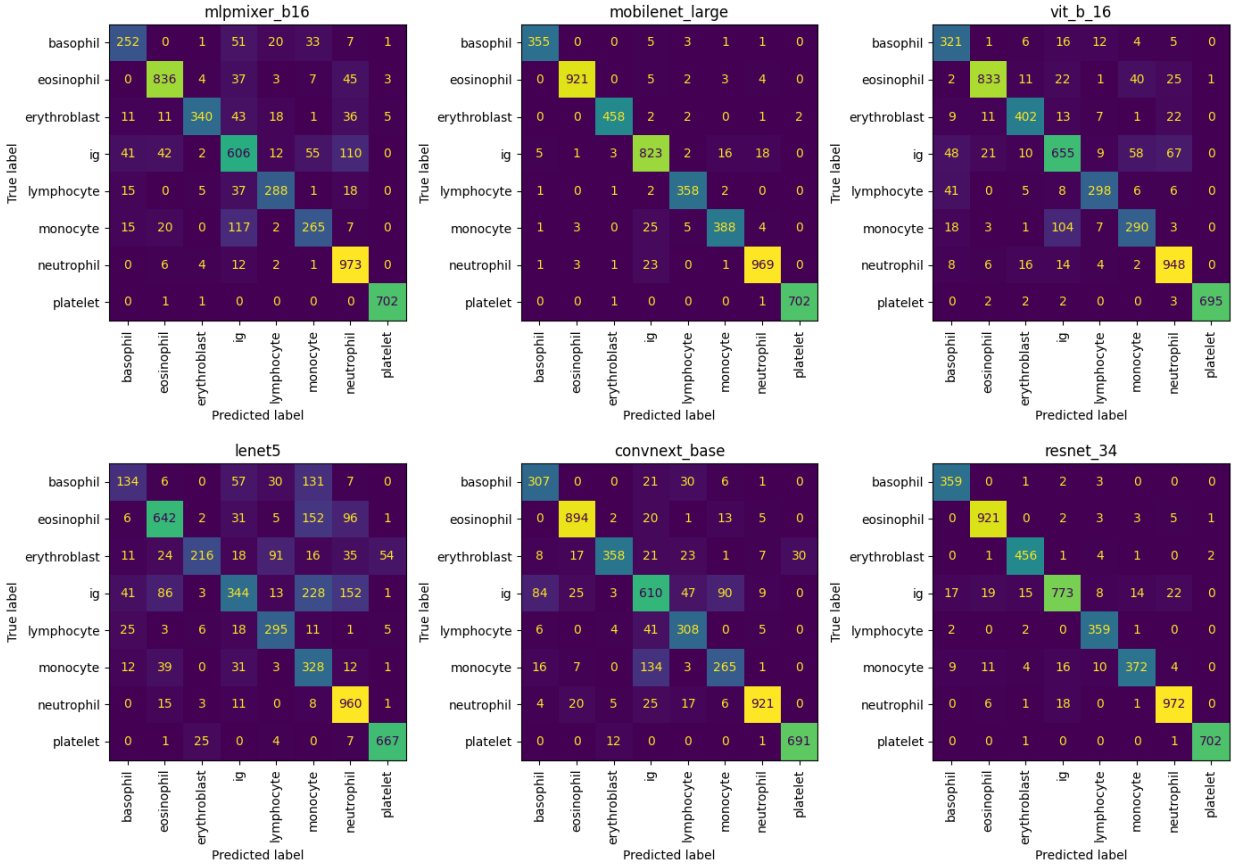

Figure 5.2: Confusion matrices when testing the large dataset configuration, for the best-performing configuration of each architecture.

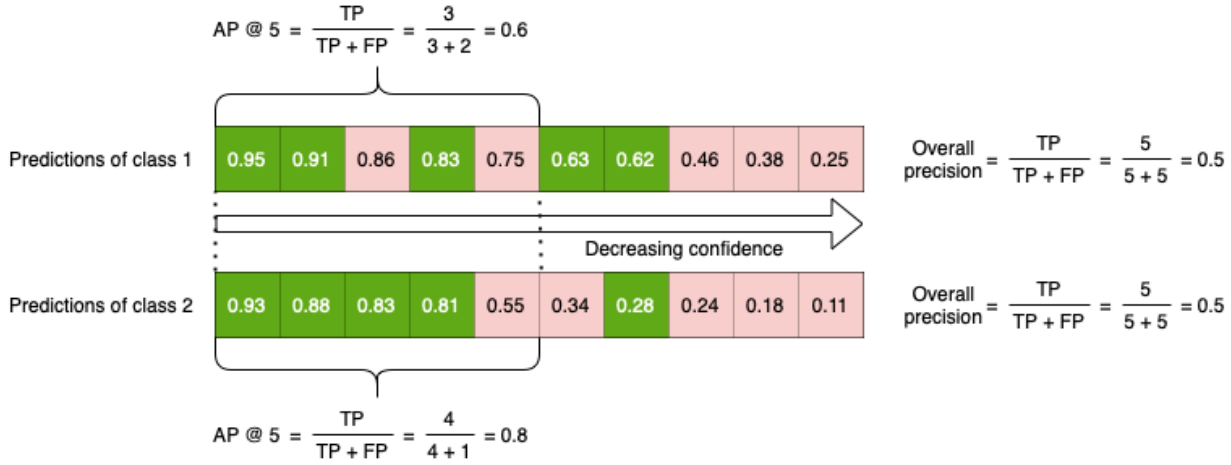

Figure 5.3: Diagram showing mean average precision compared to overall precision. Each row of coloured boxes shows the instances that a classifier predicted for two classes. Green boxes are correct predictions (true positives, TP) and red boxes are incorrect predictions (false positives, FP), and the boxes are ordered in decreasing confidence from left to right. Both classes have an equal number of true and false positives, so in each class the overall precision is 0.5. However, the average precision at rank 5 (AP @ 5) score calculates precision only within the first 5 boxes, giving 0.6 for class 1 and 0.8 for class 2. The mean average precision at rank 5 (mAP @ 5) is the mean of the AP @ 5 scores, so 0.7.

5.1.2 Medium dataset

The training details and accuracy for each network trained on the 'medium' dataset configuration (4250 training images) are shown in Table 5.3. All of the configurations were trained from scratch.

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy
convnext_tiny	27826280	32	51	12885.321	252.653	0.430
convnext_small	49460840	16	82	15342.594	187.105	0.743
convnext_base	87574664	8	100*	14861.772	148.618	0.853
convnext_large	196242632	8	100*	33614.001	336.140	0.839
lenet5	13885580	64	32	221.014	6.907	0.604
mlpmixer_s32	4640048	64	32	241.749	7.555	0.723
mlpmixer_b32	10909124	64	31	386.460	12.466	0.692
mlpmixer_s16	17530280	64	33	696.778	21.114	0.793
mlpmixer_b16	30244472	32	31	1402.966	45.257	0.769
mlpmixer_l32	31283584	32	36	823.259	22.868	0.784
mlpmixer_l16	71133928	8	31	3730.173	120.328	0.739
mobilenet_small	1526056	64	56	491.767	8.782	0.914
mobilenet_large	4212280	64	54	846.217	15.671	0.924
resnet_18	11180616	64	53	754.564	14.237	0.929
resnet_34	21288776	64	49	1002.958	20.469	0.928
resnet_50	23524424	32	34	1219.694	35.873	0.896
resnet_101	42516552	16	48	2913.578	60.700	0.914
resnet_152	58160200	16	33	2808.913	85.119	0.905
vit_b_16	86025992	16	52	4418.889	84.979	0.862
vit_b_32	87516680	4	31	1390.486	44.854	0.755
vit_l_16	303604744	4	59	15906.408	269.600	0.867
vit_l_32	305592328	4	43	4033.520	93.803	0.789

Table 5.3: Medium dataset evaluation results. Epoch values marked with * indicate configurations where the early stopping mechanism did not end the training early.

The configuration with best accuracy was `resnet_18` with an accuracy of 92.9%. As with the large dataset configuration, ResNet and MobileNet are the two highest-performing architectures. Now the best configuration is a ResNet rather than a MobileNet, though the accuracy values are very similar. The lowest performing configuration is `convnext_tiny`, with 43.0% accuracy. With regards to training time, `convnext_large` was again the slowest configuration to train, with an average seconds per epoch of 336.14. This is 45% of the seconds per epoch in the large configuration, while this dataset is 49.7% of the size of the large one, so the reduction in time per epoch is close to proportional if not slightly better. However, the total training

time for the configuration is 72% of the time taken to train with the large, because while the seconds per epoch was reduced the configuration was not halted by the early stopping mechanism and so executed for the full 100 epochs, while with the large dataset it was halted after 63 epochs. ConvNeXt also continues to exhibit higher training times than ViT, despite ViT's larger configurations having a greater number of parameters.

It is very interesting that of all the ResNet configurations, the best performer is actually the smallest. The decrease in performance as the network grows is not as pronounced as with the large dataset, but it is still present. While all the ResNet configurations were halted by the early stopping mechanism, the most likely explanation is that the larger models were overfitted, and that ideally the training should have been stopped earlier. This will be explored in more detail in section 5.2.2.

The validation loss over the training period for the best configuration of each architecture is shown in figure 5.4. As before, we see substantial variance in the MobileNet and ResNet configurations, though for ResNet it is easier to see the troughs gradually improving until they reach what is eventually selected as the minimum loss. ConvNeXt also demonstrates the same gradual improvement as in the large dataset, albeit with a less steep slope; it also appears to have stabilised at the end of the training process, suggesting that had it not hit the 100-epoch limit then the early stopping mechanism would have halted training soon after this point anyway.

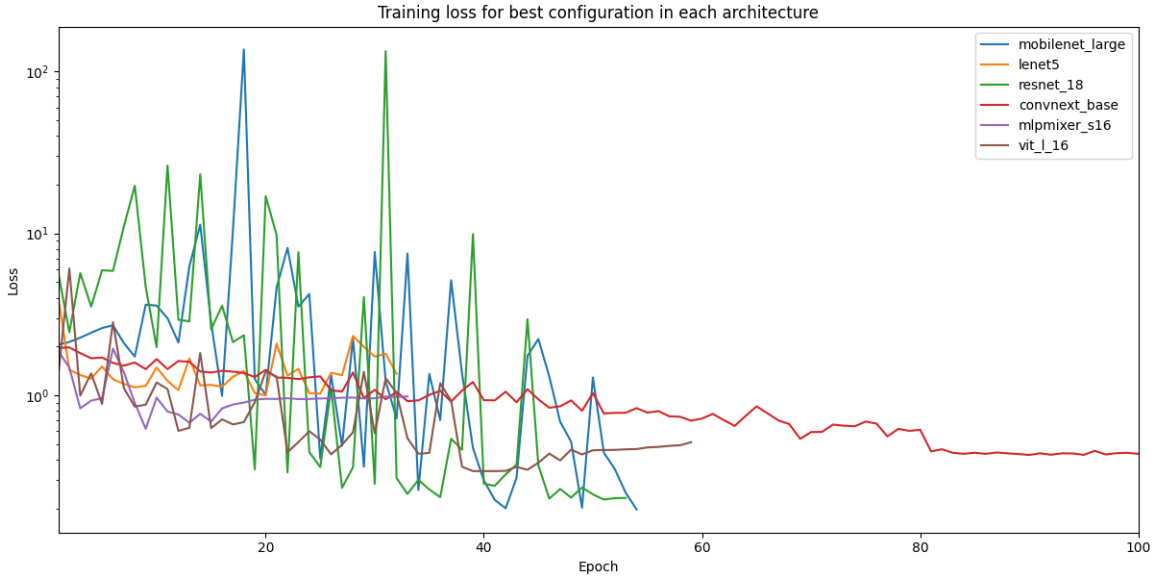

Figure 5.4: Validation loss during training on medium dataset, for the best-performing configuration of each architecture.

Confusion matrices for the best performing configuration in each architecture are shown in 5.5. As with the large dataset, we can see that the class most frequently involved in misclassifications is `ig`, typically being confused with `neutrophil` and `monocyte`. As noted with the large dataset, this is likely related to the different exposure level in class `ig` that the preprocessing attempted to account for, but did not completely eliminate.

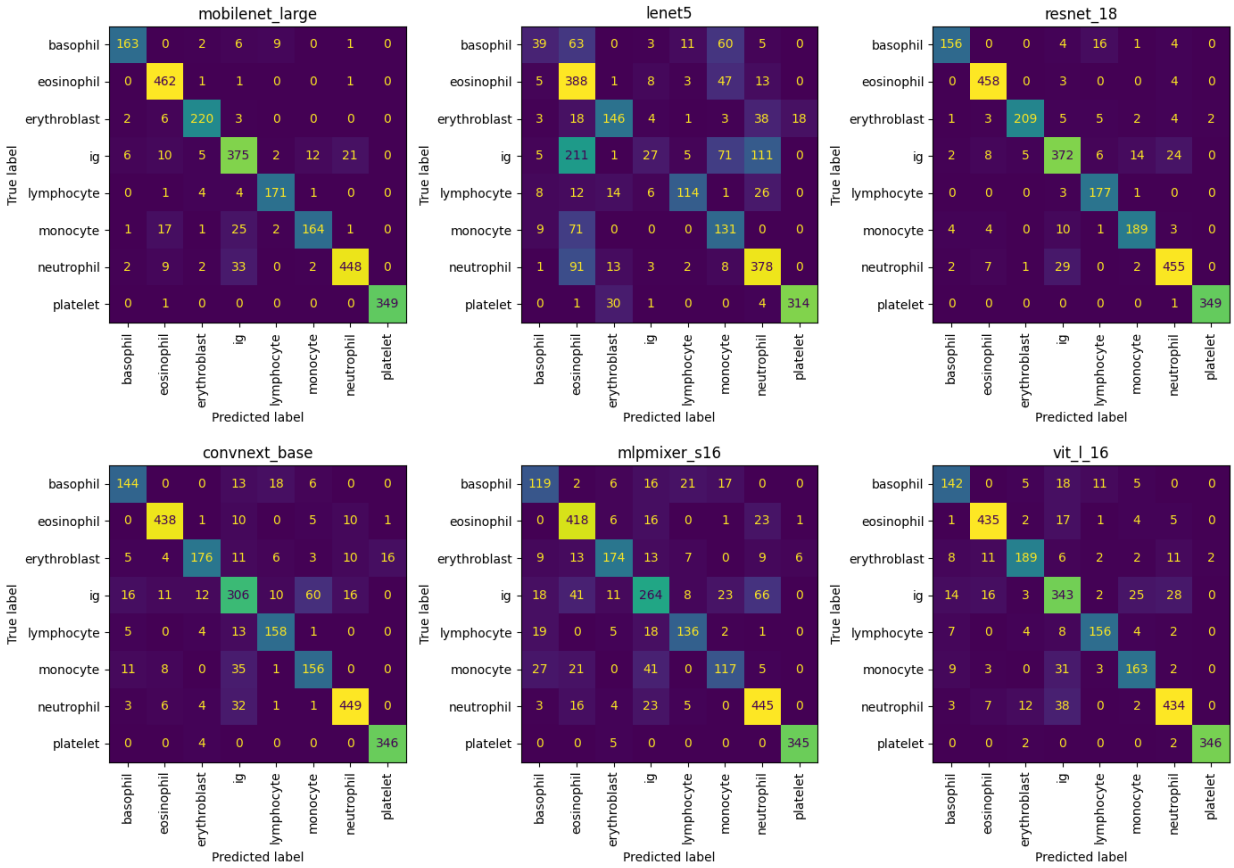

Figure 5.5: Confusion matrices when testing the medium dataset configuration, for the best-performing configuration of each architecture.

Lastly, the mean average precision scores for the best performing configurations are shown in table 5.4. These were calculated in the same way as for the large dataset configuration, as described in the previous section. A particularly interesting result here is that `mobilenet_large` has a slightly better mAP@100 score than `resnet_18`, despite having slightly lower average accuracy overall. This means that the difference must be in the lower-confidence predictions: the test data set included 2550 images, and the mAP@100 score is only derived from the 800 of them with the highest confidence values, so for `resnet_18` to have higher overall accuracy but lower mAP@100, it must have classified the remaining 1750 images slightly better than MobileNet.

	mAP @ 5	mAP @ 100
mobilenet_large	1.000	0.99375
resnet_18	1.000	0.98875
vit_l_16	0.975	0.95000
convnext_base	0.975	0.93125
mlpmixer_s16	0.950	0.90750
lenet5	0.800	0.71375

Table 5.4: Mean average precision (mAP) scores for best classifier configurations on the medium dataset configuration.

5.1.3 Small dataset

The training details and accuracy for each network trained on the 'small' dataset configuration (2100 training images) are shown in Table 5.5. All of the configurations were trained from scratch.

The configuration with the best performance for this size of dataset was `resnet_34`, with 91.4%. ResNet and MobileNet are again the two best performing architectures. The worst performance is from `lenet5` with 42.5%, though ConvNeXt comes close. With regards to training time, `convnext_large` was again the slowest configuration to train, though unlike the medium configuration the time is comparable to the slowest ViT configuration, `vit_l_16`.

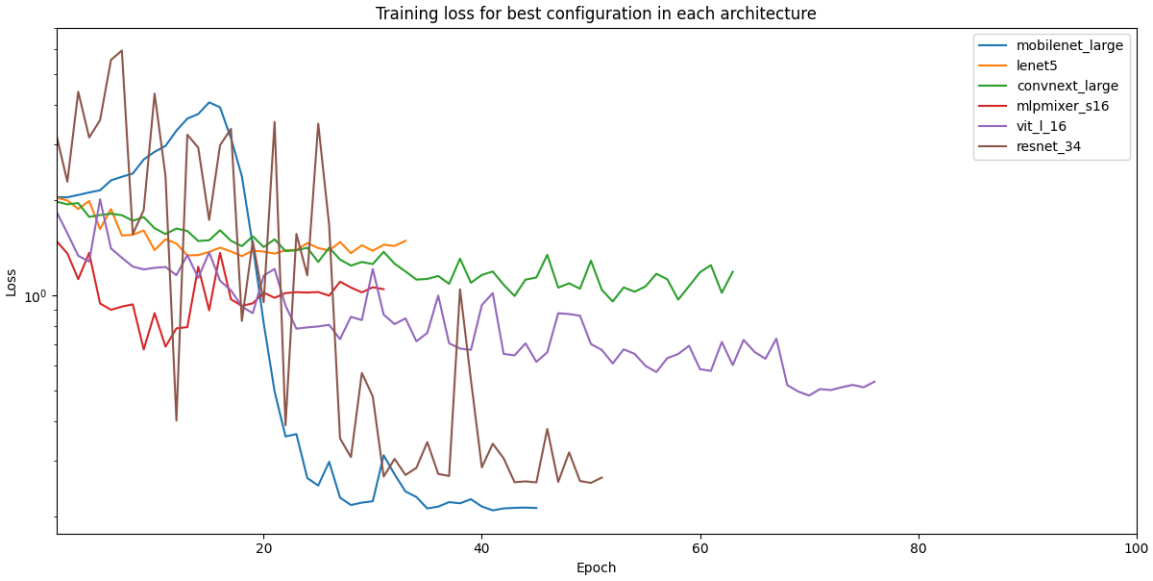

Figure 5.6: Validation loss during training on small dataset, for the best-performing configuration of each architecture.

Validation loss during training is shown in figure 5.6. ResNet continues to show very high variance in loss over the course of training, though this time it seems to be the only configuration

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy
convnext_tiny	27826280	32	63	7868.949	124.904	0.466
convnext_small	49460840	16	54	4616.715	85.495	0.482
convnext_base	87574664	8	34	2562.891	75.379	0.441
convnext_large	196242632	8	63	10597.445	168.213	0.619
lenet5	13885580	64	33	163.799	4.964	0.425
mlpmixer_s32	4640048	64	33	133.229	4.037	0.705
mlpmixer_b32	10909124	64	31	195.001	6.290	0.701
mlpmixer_s16	17530280	64	31	329.826	10.640	0.755
mlpmixer_b16	30244472	32	32	701.543	21.923	0.736
mlpmixer_l32	31283584	32	31	356.401	11.497	0.737
mlpmixer_l16	71133928	8	31	1917.389	61.851	0.744
mobilenet_small	1526056	64	61	283.035	4.640	0.896
mobilenet_large	4212280	64	45	367.448	8.166	0.900
resnet_18	11180616	64	40	286.596	7.165	0.905
resnet_34	21288776	64	51	531.132	10.414	0.914
resnet_50	23524424	32	45	812.103	18.047	0.912
resnet_101	42516552	16	40	1268.082	31.702	0.907
resnet_152	58160200	16	31	1330.057	42.905	0.861
vit_b_16	86025992	16	45	1920.397	42.675	0.779
vit_b_32	87516680	4	40	911.545	22.789	0.745
vit_l_16	303604744	4	76	10249.075	134.856	0.799
vit_l_32	305592328	4	42	1966.547	46.823	0.778

Table 5.5: Small dataset evaluation results.

doing so; the other configurations do have varying loss, but over a much smaller range. It is also notable that the configuration which achieves the lowest loss is actually `mobilenet_large`, not `resnet_34`, but this does not translate into the best accuracy as shown in figure 5.7; this means that while the predictions made by `mobilenet_large` are slightly less often correct than `resnet_34`, when they are correct they must have slightly better contrast between the predicted class and the other classes.

With respect to the other configurations, ViT shows a downward trend across its training. Despite being the last configuration to be halted, it may be the case that a reduced threshold for improvement in the early stopping mechanism would have allowed it to continue and improve further; however, given the rate of improvement, we could predict that it would need to continue training for quite some time before reaching loss values similar to MobileNet or ResNet. ConvNeXt also showed a downward trend over the first 40 epochs, but by the time the early stopping mechanism halted the training for it, the trend looks much more flat, so further training is less likely to have improved performance further.

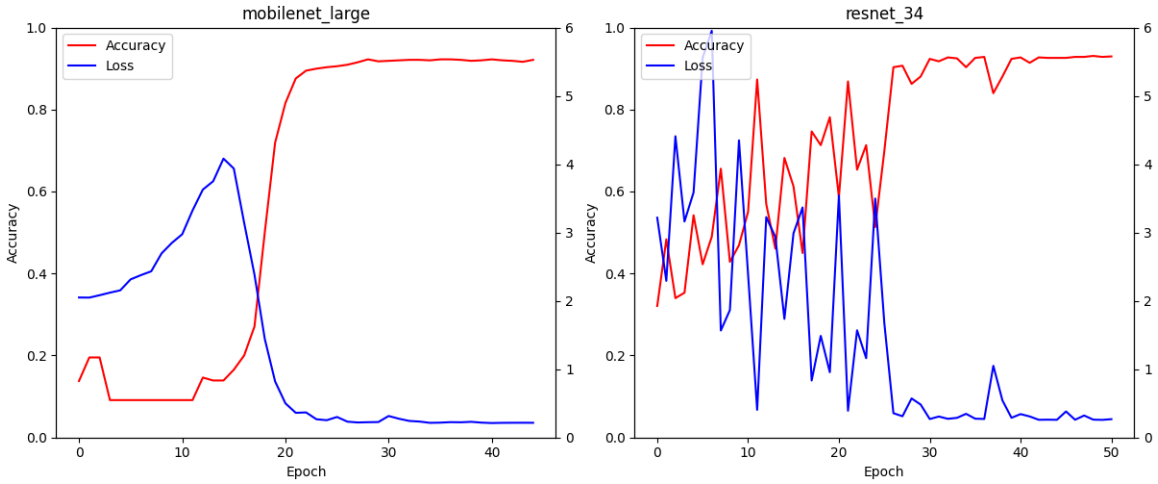

Figure 5.7: Accuracy and validation loss during training on small dataset, for MobileNet and ResNet.

Confusion matrices for the best performing configuration in each architecture are shown in figure 5.8. As with the larger datasets, we again see that there is a slightly higher density of errors involving the `ig` class, with the a number of instances of `monocyte` and `neutrophil` being classified as `ig`, and in multiple cases also a number of `ig` images being classified as `neutrophil`.

	mAP @ 5	mAP @ 100
resnet_34	1.000	0.92000
mobilenet_large	1.000	0.90500
vit_l_16	0.850	0.79375
mlpmixer_s16	0.900	0.75625
convnext_large	0.725	0.43625
lenet5	0.425	0.35250

Table 5.6: Mean average precision (mAP) scores for best classifier configurations on the small dataset configuration.

Lastly, the mean average precision figures for the best configuration of each architecture are shown in table 5.6. We can see that the top 5 predictions of both `resnet_34` and `mobilenet_large` are completely correct, and the top 5 for `mlpmixer_s16` and `convnext_large` are also much higher quality than the average prediction accuracies for those configurations, meaning that their highest-confidence predictions are more accurate than their predictions in general.

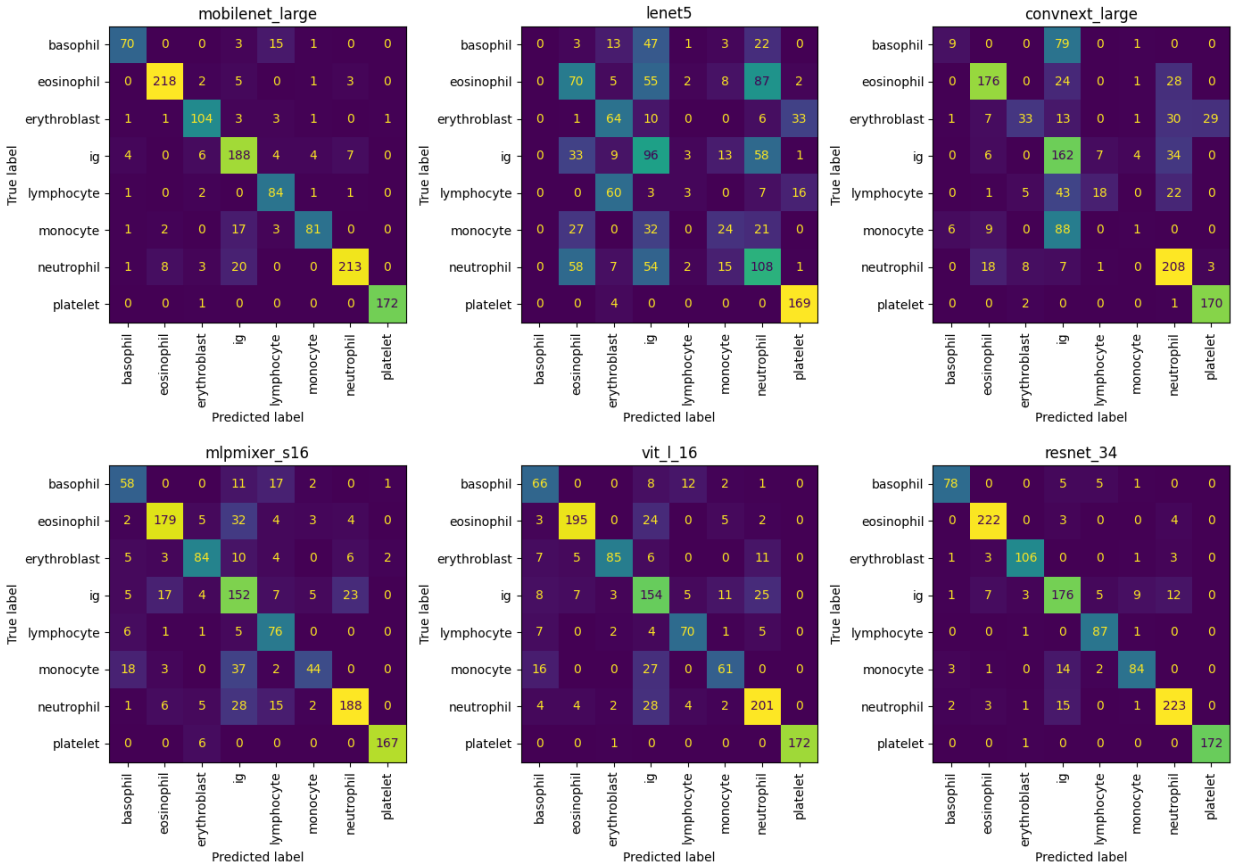

Figure 5.8: Confusion matrices when testing the small dataset configuration, for the best-performing configuration of each architecture.

5.1.4 Tiny dataset

The training details and accuracy for each network trained on the 'tiny' dataset configuration (200 training images) are shown in Table 5.7. All of the configurations were trained from scratch.

The configuration with the best performance for this dataset size is `resnet_34`, at 79.1% accuracy; the configurations with the worst performance are `convnext_base` and both MobileNet configurations, all at 17.4% accuracy. It is remarkable that we see multiple configurations with identical accuracy values; there are in fact 3 instances of configurations producing identical results: `convnext_base` and the two MobileNet configurations with 17.4% accuracy, `convnext_tiny` and `convnext_large` with 20.0% accuracy, and three of the ViT configurations with 59.1% accuracy.

Figure 5.9 shows the confusion matrices for each group. It is clear that for the first two groups, the identical accuracy is because all configurations have simply learned to predict the same class for every input. The third group, containing the three vision transformer configurations, has produced a more useful result; for this row, the confusion matrices are clearly

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy
convnext_tiny	27826280	32	47	535.680	11.397	0.200
convnext_small	49460840	16	31	319.004	10.290	0.183
convnext_base	87574664	8	31	252.687	8.151	0.174
convnext_large	196242632	8	31	493.900	15.932	0.200
lenet5	13885580	64	66	32.936	0.499	0.313
mlpmixer_s32	4640048	64	65	38.343	0.590	0.357
mlpmixer_b32	10909124	64	68	56.832	0.836	0.513
mlpmixer_s16	17530280	64	75	97.247	1.297	0.548
mlpmixer_b16	30244472	32	54	126.355	2.340	0.487
mlpmixer_l32	31283584	32	59	80.992	1.373	0.539
mlpmixer_l16	71133928	8	31	190.420	6.143	0.504
mobilenet_small	1526056	64	31	18.974	0.612	0.174
mobilenet_large	4212280	64	31	28.229	0.911	0.174
resnet_18	11180616	64	44	45.630	1.037	0.713
resnet_34	21288776	64	70	81.073	1.158	0.791
resnet_50	23524424	32	71	134.500	1.894	0.757
resnet_101	42516552	16	59	190.597	3.230	0.765
resnet_152	58160200	16	48	208.561	4.345	0.774
vit_b_16	86025992	16	41	188.609	4.600	0.391
vit_b_32	87516680	4	45	110.871	2.464	0.591
vit_l_16	303604744	4	66	915.314	13.868	0.591
vit_l_32	305592328	4	47	236.630	5.035	0.591

Table 5.7: Tiny dataset evaluation results.

different, but each has the same sum along its diagonal. So for this third group, it looks likely that it is more of a matter of chance that the accuracy values are identical, made more likely by the fact that the test set is comparatively small.

The validation loss during training for both of the MobileNet configurations is shown in Figure 5.10. Both configurations start with similar loss values at their first epoch, improve slightly, and then get worse as training continues. Both configurations also halted training at 31 epochs, indicating that the model selected by early stopping is the model at epoch 11. Despite the fact that subsequent epochs did have a lower loss value, they were not lower than the loss value at epoch 11 by a margin of at least 0.1, and so they were not recognised by the early stopping mechanism as being improvements.

Figure 5.11 shows the validation loss of the ConvNeXt models on the tiny dataset. These loss curves look fairly typical, with a rapid decrease from the starting value followed by stabilisation; the larger configurations decrease more quickly, and stabilise more quickly, than the smaller

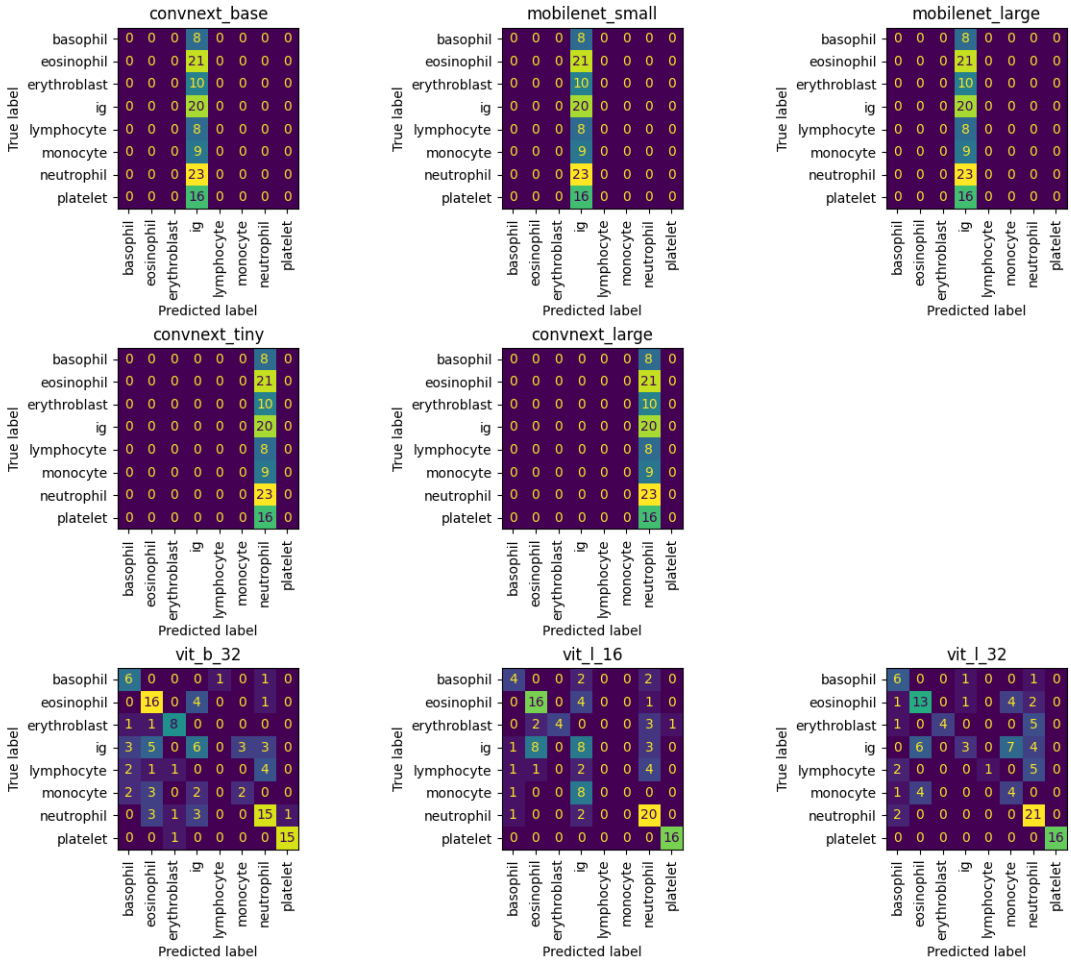

Figure 5.9: Confusion matrices for the configurations that produced non-unique accuracy scores when trained on the tiny dataset. Each row of matrices indicates models with identical accuracy.

configurations. However, all configurations are stabilising around the same loss value, which suggests that they have all found the same local minima in the loss function and are fitting to it. We saw in Figure 5.9 that three of the four configurations classify all inputs into one class; the fourth configuration, `convnext_small`, is depicted in Figure 5.12, and we can see that while it does not have the same accuracy as the other models, it has also learned the behaviour of predicting a the same class for all inputs. So this behaviour is present for all four models; the only reason that all four do not have identical accuracy is because they have not all learned the same class.

As noted, the most accurate models are the ResNets. Their validation loss is shown in figure 5.13. Considering that the vertical axis is on a log scale, the initial loss values, especially for `resnet_50`, are extremely high. They drop quickly, but show high variance for some time until eventually stabilising. Unlike with the large dataset, we do not see the trend of accuracy decreasing as the network gets larger; the second smallest configuration, `resnet_34`, happens to have highest accuracy in this case, but the largest configuration `resnet_152` then has the

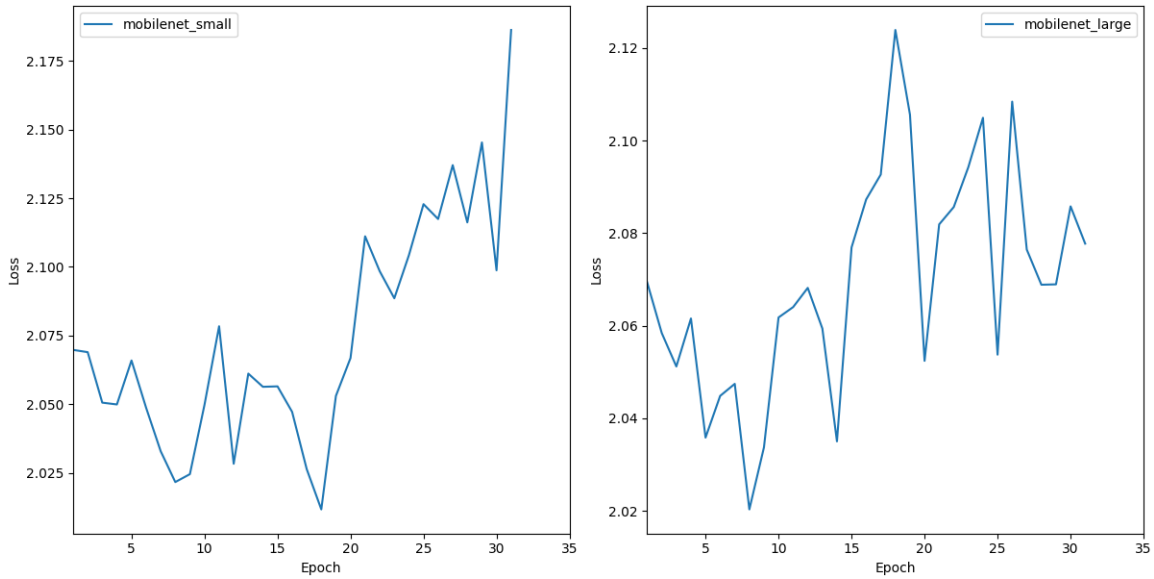

Figure 5.10: Validation loss during training of the MobileNet configurations on the tiny dataset.

second-highest accuracy. So, while the larger networks may have been overfitting in the case of the large dataset, with the tiny dataset they do not appear to show the same problem.

Lastly, the mean average precision scores for the classifiers are shown in table 5.8. This includes multiple configurations for some of the architectures, as they have equally high accuracy. However, we notice that their mean average precision scores are not equal, either at $N = 5$ or at $N = 100$. The three vision transformer architectures have equal total accuracy, and equal mean average precision at $N = 100$, but exhibit differences at the $N = 5$ level; this suggests that `vit_b_32` may be the best of the three configurations because while it classifies things equally well as the other two configurations on average, its highest-confidence predictions are slightly more correct than the other two.

The two MobileNet configurations have different mean average precision scores, despite it being shown in figure 5.9 that they are in fact classifying all inputs the same way (into class `ig`). This is possible because the mean average precision is based only on the first N results for each class. As neither configuration makes any predictions in other classes, the precision scores for those classes are 0, while the precision score for class `ig` depends on how many of the N predictions with the highest values for `ig` are truly of that class. So we can conclude that for `mobilenet_small`, while it does predict `ig` for every input, its 5 strongest predictions are all wrong; while for `mobilenet_large`, 1 of its 5 strongest predictions is correct (giving a precision of 0.2 for the class, and a mean average precision of 0.025 across the 8 classes). Conversely, across the 100 strongest predictions, `mobilenet_small` includes a greater number of correct `ig` predictions compared to `mobilenet_large`.

If we were to measure mean average precision at $N = 120$, the size of the entire test dataset, then we would expect both networks to have an equal score. In fact, we can measure this for any value of N in the range $1 \leq N \leq 120$; this is depicted in figure 5.14, and from the plot we can see that while `mobilenet_large` is more accurate within the first 20 results, after that

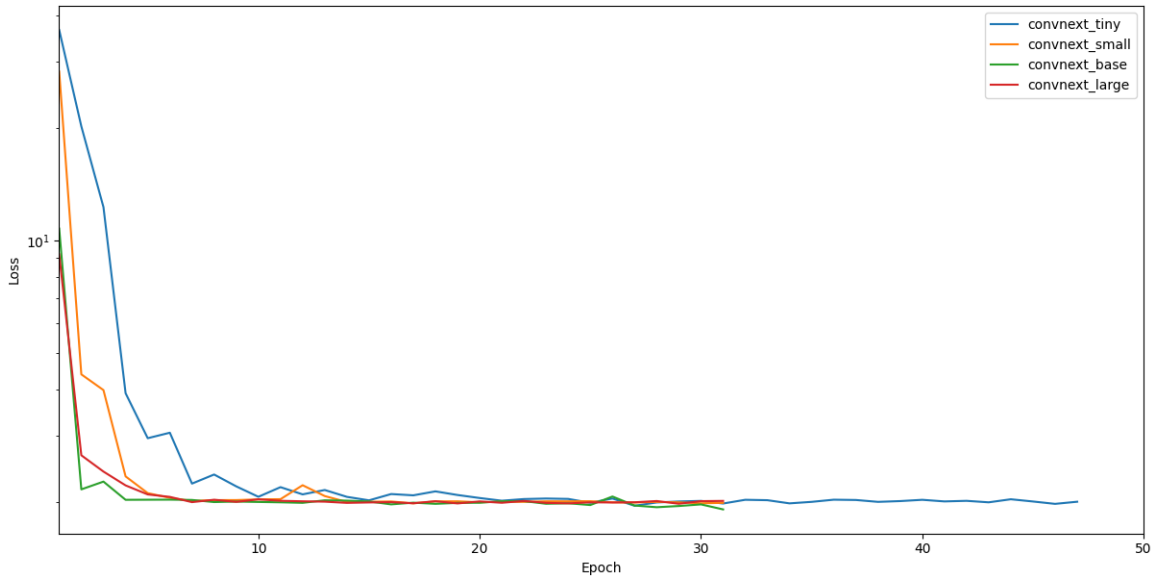

Figure 5.11: Validation loss during training of the ConvNeXt configurations on the tiny dataset.

point `mobilenet_small` has a greater density of correct classifications until almost the entire testing set is retrieved.

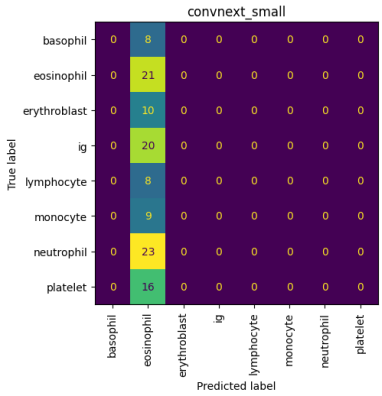

Figure 5.12: Confusion matrix of convnext_small on tiny dataset.

	mAP @ 5	mAP @ 100
resnet_34	0.750	0.11375
vit_b_32	0.625	0.08500
vit_l_16	0.400	0.08500
vit_l_32	0.600	0.08500
mlpmixer_s16	0.575	0.07875
lenet5	0.150	0.04500
convnext_large	0.050	0.02875
convnext_tiny	0.000	0.02750
mobilenet_small	0.000	0.02500
mobilenet_large	0.025	0.01625

Table 5.8: Mean average precision (mAP) scores for best (or best-equal) classifier configurations on the tiny dataset configuration.

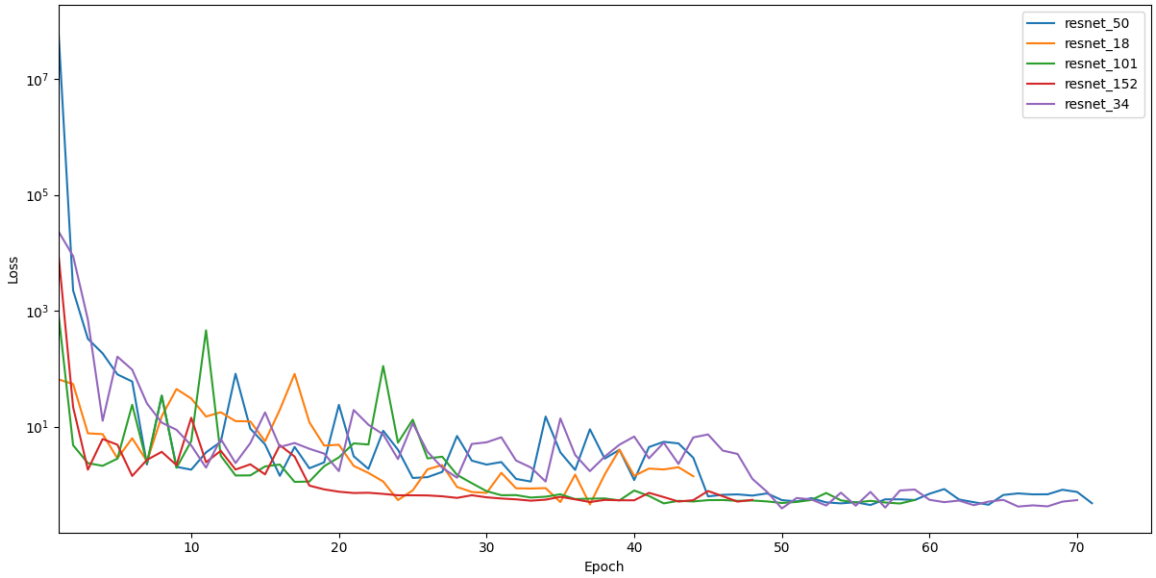

Figure 5.13: Validation loss during training for all ResNet configurations on tiny dataset.

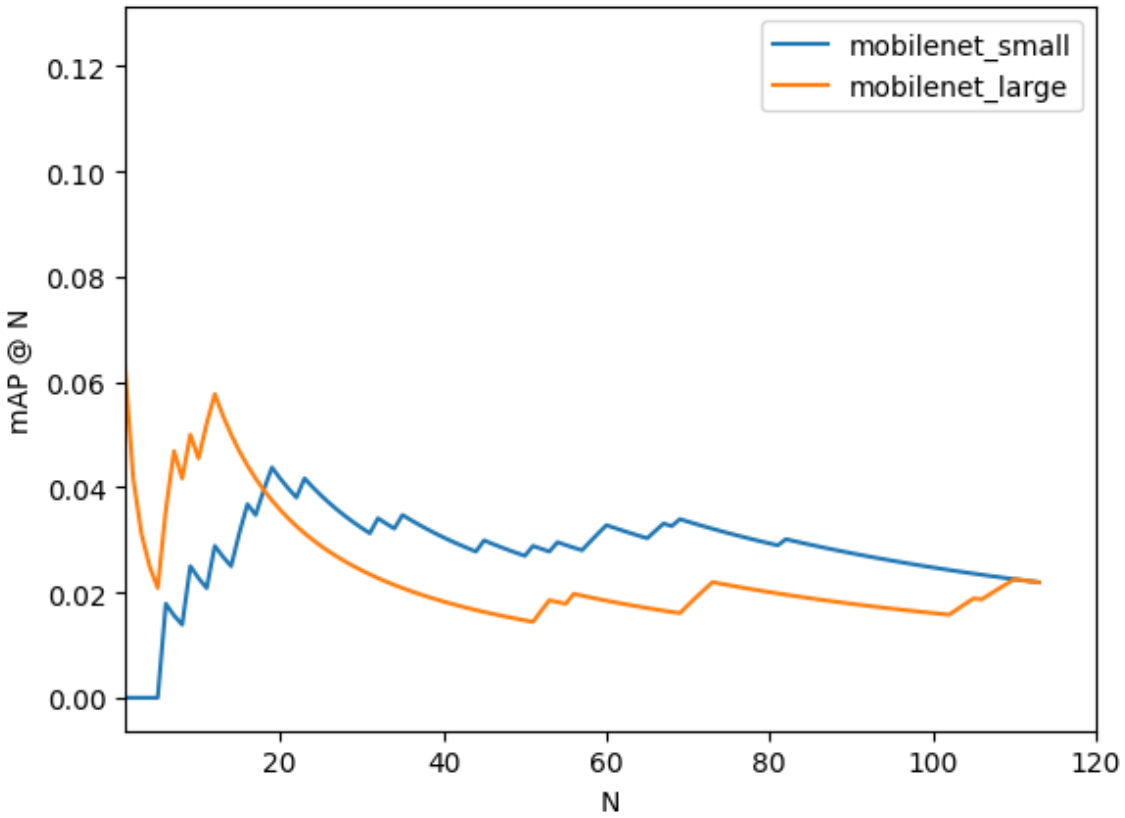

Figure 5.14: Mean average precision of both MobileNet configurations on the tiny dataset.

5.2 Performance by network architecture

5.2.1 LeNet-5

The results of evaluating LeNet-5 across each of the different dataset sizes are collated in table 5.9. These are the same results as presented in section 5.1, but collated across all four dataset configurations for easier comparison.

Dataset	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy
large	64	32	432.554	13.517	0.700
medium	64	32	221.014	6.907	0.604
small	64	33	163.799	4.964	0.425
tiny	64	66	32.936	0.499	0.313

Table 5.9: Evaluation results for LeNet-5 architecture across all dataset sizes.

The overall accuracy of LeNet-5 is not very high even with the largest dataset, and gets substantially worse as the size of the dataset decreases. This is perhaps to be expected from the architecture which is, relatively speaking, the oldest one being evaluated; it should also be noted that the original paper developed LeNet-5 for classification of 32x32 images of letters and numbers, so adapting it to much larger images of blood cells is quite a departure from its original purpose. Still, it is an interesting baseline to have, and it is quite significant that it outperforms some of the much newer architectures, such as `convnext_tiny` on the medium dataset.

The characteristics of the training process are also quite interesting. Figure 5.15 shows how the validation loss and accuracy of the model progressed when training on the large dataset. This training process was halted at 32 epochs, meaning that the epoch with lowest loss was epoch 12; while the variance hides it a bit, it can be seen from the graph that the loss followed a worsening trend from that point. However, the accuracy did not; the accuracy when the training was halted was similar to the accuracy at epoch 12. What this implies is that the predictions made by the network still had the highest values for the correct classes roughly the same proportion of the time, but that the values for the other classes were getting higher, such that the maximum class in each prediction was becoming a smaller proportion of the result.

Figure 5.16 shows the validation loss during training across each of the datasets. Each network demonstrates a rapid improvement in loss over the first few epochs, followed by a more gradual improvement over the subsequent 10 epochs. The models training on the large and medium configurations show much higher variance in loss compared to the small and tiny configurations, and they also begin to get worse after around the 20th epoch; the small and tiny configurations look more stable, with perhaps a slight decrease in performance towards the end of the training period.

It is not clear why the large and medium configurations have much higher variance. One possibility is that it is related to the number of batches; as the batch size is the same for each

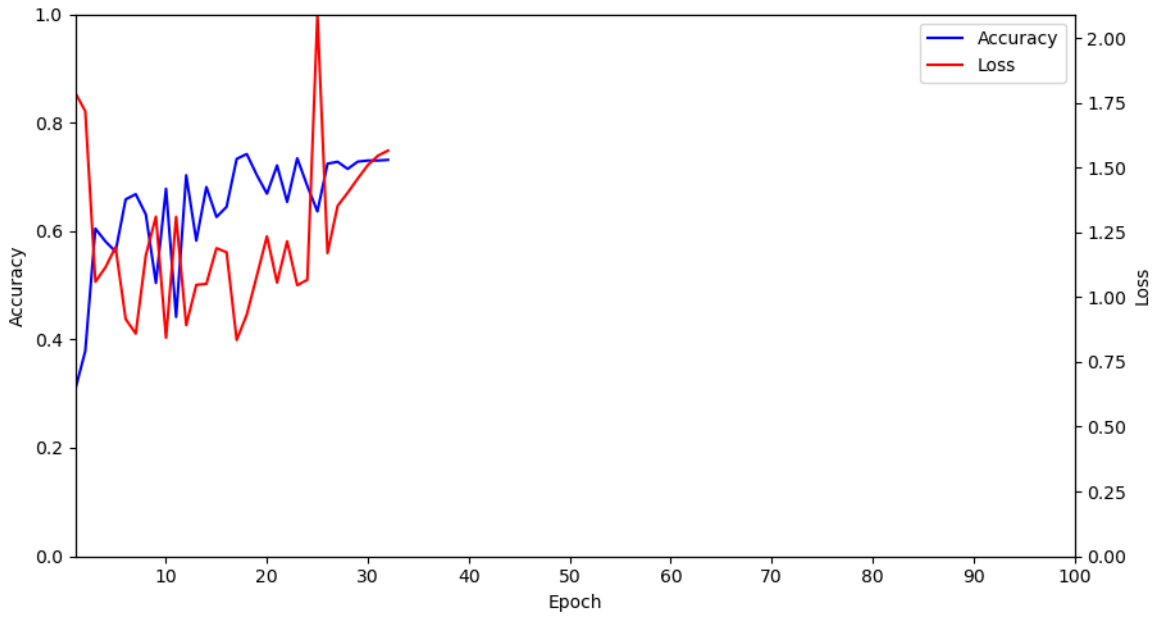

Figure 5.15: Validation loss and accuracy during LeNet5 training, large dataset.

dataset size, but the dataset size is different, then the larger datasets will have a greater total number of batches per epoch. This means that for the larger datasets, the model parameters are updated a greater number of times within each epoch. However, the gradients are recomputed for each individual batch within the epoch, so it is not immediately clear how a greater number of updates within an epoch would result in a higher overall change for the epoch.

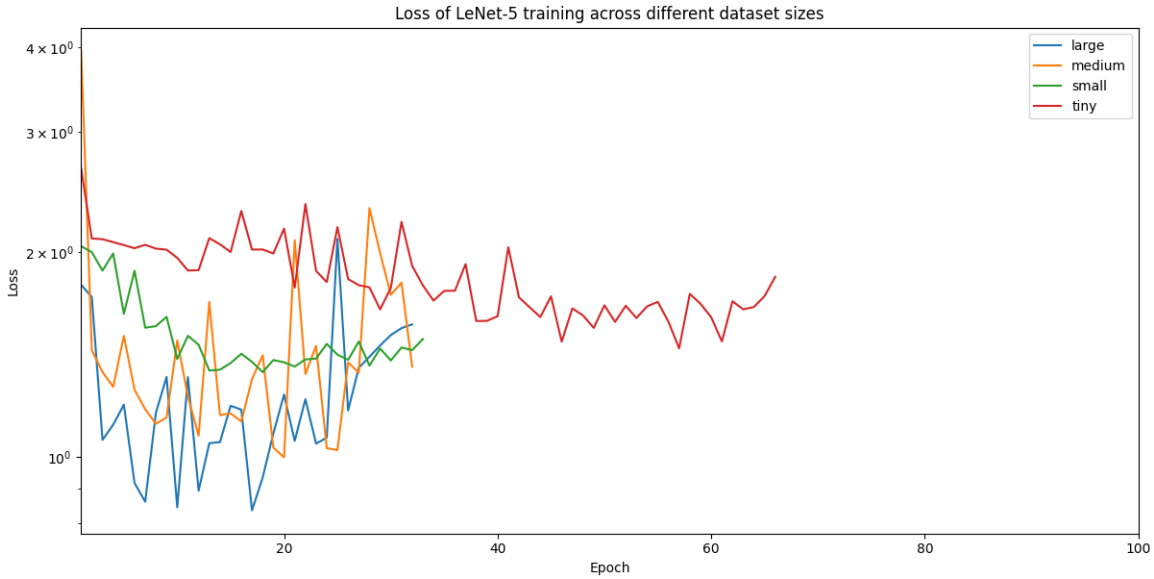

Figure 5.16: Validation loss for LeNet5 across all dataset sizes.

5.2.2 ResNet

The results of evaluating the ResNet configurations across each of the different dataset sizes are collated in table 5.10. These are the same results as presented in section 5.1, but collated across all four dataset configurations for easier comparison.

ResNet tends to rank amongst the best architectures for each size of dataset. It can be seen that for each size of network, the overall decrease in performance between the large, medium, and small datasets is similar; there is no more than a 5% decrease in performance between the large and small datasets. However, the decrease in accuracy between small and tiny datasets is more pronounced, of 10%-20%. This is clearer to see in figure 5.17.

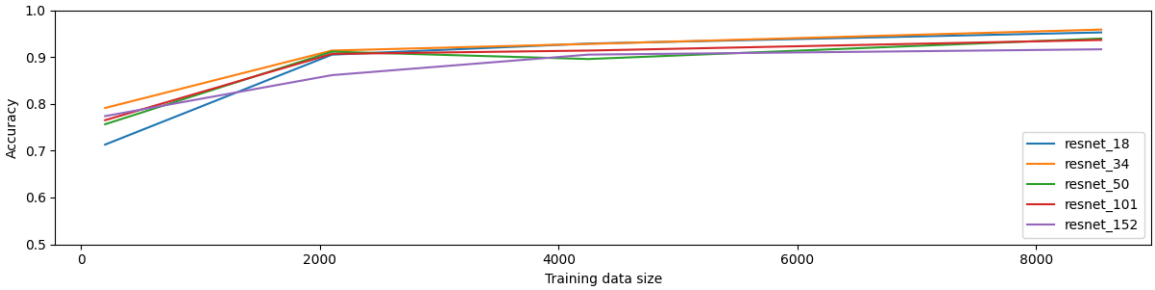

Figure 5.17: Accuracy of each ResNet configuration as a function of training data size.

Figure 5.18 shows the loss during the training process for each ResNet configuration on both the large and small datasets. We can see from the first plot that all configurations follow a similar pattern: a general reduction in loss over the first 10 epochs, followed by a more shallow trend downwards. There is also considerable variance in loss; while all configurations exhibit

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy	Dataset
resnet_18	11180616	64	40	1119.375	27.984	0.953	large
resnet_18	11180616	64	53	754.564	14.237	0.929	medium
resnet_18	11180616	64	40	286.596	7.165	0.905	small
resnet_18	11180616	64	44	45.630	1.037	0.713	tiny
resnet_34	21288776	64	46	1921.404	41.770	0.959	large
resnet_34	21288776	64	49	1002.958	20.469	0.928	medium
resnet_34	21288776	64	51	531.132	10.414	0.914	small
resnet_34	21288776	64	70	81.073	1.158	0.791	tiny
resnet_50	23524424	32	40	3069.293	76.732	0.940	large
resnet_50	23524424	32	34	1219.694	35.873	0.896	medium
resnet_50	23524424	32	45	812.103	18.047	0.912	small
resnet_50	23524424	32	71	134.500	1.894	0.757	tiny
resnet_101	42516552	16	32	3923.816	122.619	0.937	large
resnet_101	42516552	16	48	2913.578	60.700	0.914	medium
resnet_101	42516552	16	40	1268.082	31.702	0.907	small
resnet_101	42516552	16	59	190.597	3.230	0.765	tiny
resnet_152	58160200	16	31	5319.037	171.582	0.917	large
resnet_152	58160200	16	33	2808.913	85.119	0.905	medium
resnet_152	58160200	16	31	1330.057	42.905	0.861	small
resnet_152	58160200	16	48	208.561	4.345	0.774	tiny

Table 5.10: Evaluation results for ResNet architecture across all dataset sizes.

'spikes', the effect is significantly more pronounced in the smaller configurations, `resnet_18` and `resnet_34`. On the smaller dataset, shown in the second plot, all five configurations show greater variance over the first 20 to 25 epochs, before suddenly dropping and stabilising.

It is notable that the highest average accuracy scores are found from the smaller configurations. The average of the accuracy scores across the large, medium, and small datasets for `resnet_18` is 92.9% and for `resnet_34` is 93.4%, while `resnet_50` is 91.6%, `resnet_101` is 91.9% and `resnet_152` is only 89.4%. Increasing the size of the network seems to increase the training time while only worsening the accuracy. This is less true for the tiny configuration, where `resnet_18` has the lowest accuracy score of 71.3%, but `resnet_34` has the highest score of 79.1%.

This may be a result of the way the early stopping mechanism was configured and applied. Figure 5.19 shows how the validation loss and accuracy progressed over the training period for the `resnet_18` configuration on the large dataset, while 5.20 shows the corresponding plot for the `resnet_152` configuration. The former model was trained for 40 epochs and the latter for 31, before early stopping terminated each one, and this means that the optimum model selected were the ones from the checkpoints at epochs 20 and 11 respectively. The plots of the

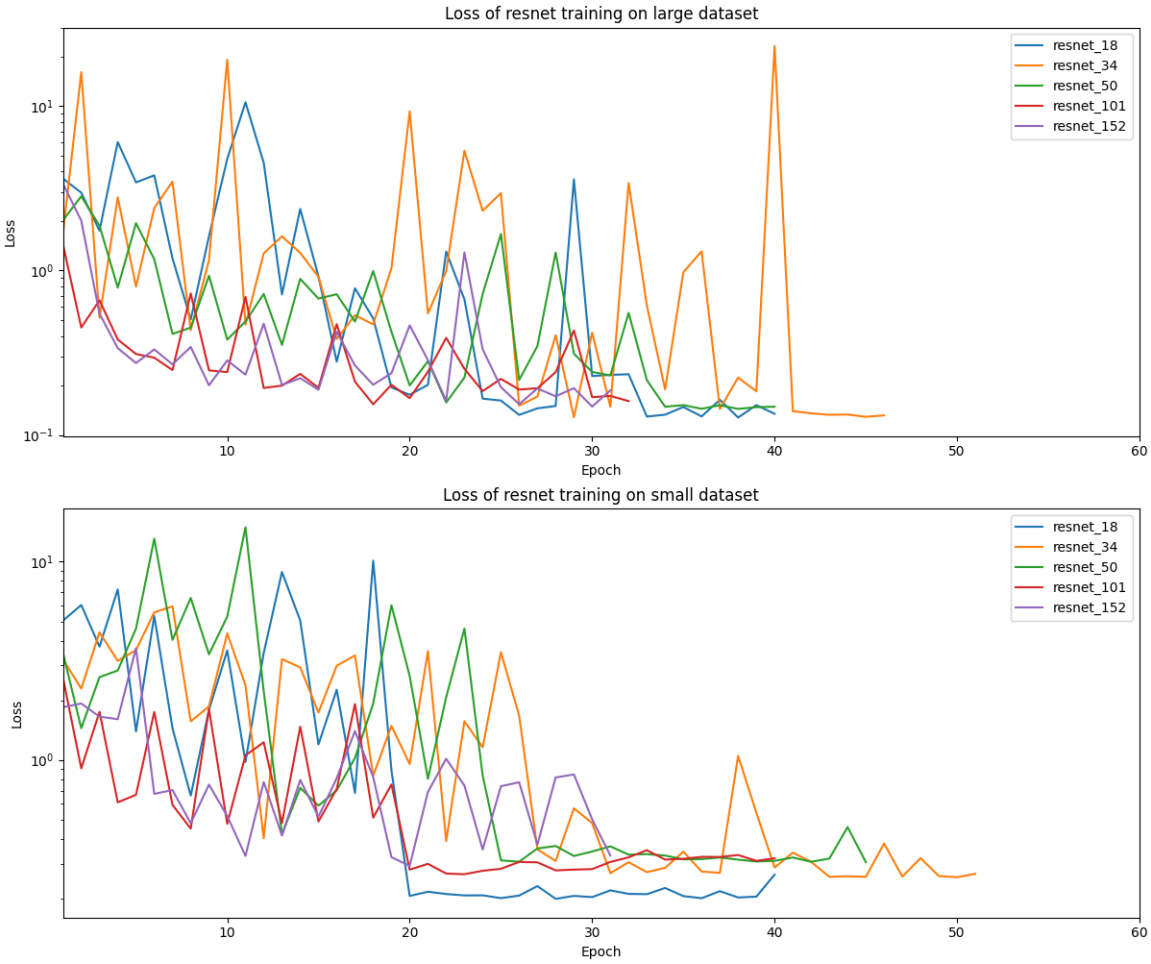

Figure 5.18: Training loss for each ResNet architecture on both large and small datasets.

training process demonstrate how the larger architecture made much greater progress in the initial epochs, reaching a loss value around 0.3 within the first 5 epochs before stabilising, while the smaller architecture shows a much less regular progression over the initial epochs before reaching a similar loss value around epoch 20. As the threshold for determining whether the model is still improving is that an epoch must have a loss of at least 0.1 below the previous best, when the model has already achieved a loss of 0.3 then a further reduction in loss as large as 0.1 is perhaps unrealistic.

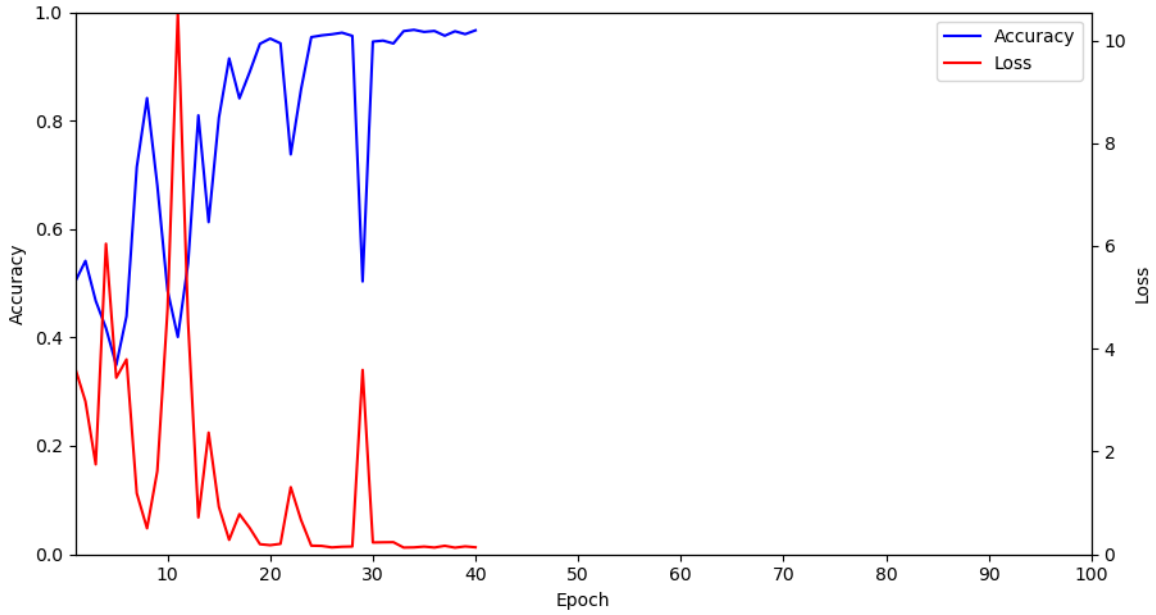

Figure 5.19: Validation loss and accuracy during ResNet18 training, large dataset.

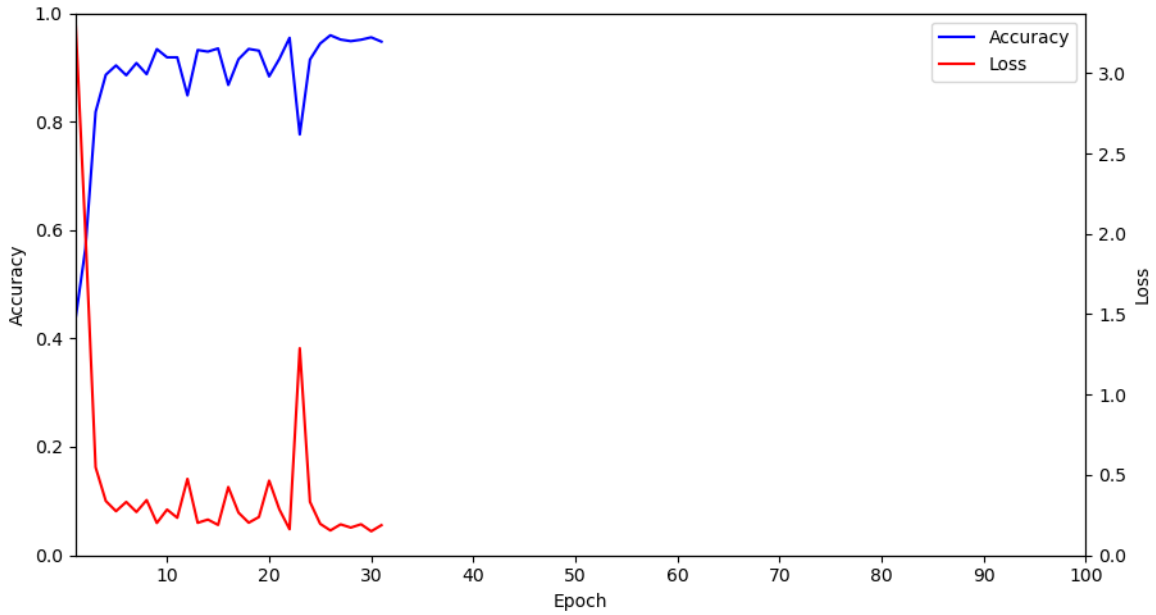

Figure 5.20: Validation loss and accuracy during ResNet152 training, large dataset.

5.2.3 MobileNet V3

The results of evaluating the MobileNet V3 configurations across each of the different dataset sizes are collated in table 5.11. These are the same results as presented in the section 5.1, but collated across all four dataset configurations for easier comparison.

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy	Dataset
mobilenet_small	1526056	64	39	640.624	16.426	0.945	large
mobilenet_small	1526056	64	56	491.767	8.782	0.914	medium
mobilenet_small	1526056	64	61	283.035	4.640	0.896	small
mobilenet_small	1526056	64	31	18.974	0.612	0.174	tiny
mobilenet_large	4212280	64	51	1508.823	29.585	0.971	large
mobilenet_large	4212280	64	54	846.217	15.671	0.924	medium
mobilenet_large	4212280	64	45	367.448	8.166	0.900	small
mobilenet_large	4212280	64	31	28.229	0.911	0.174	tiny

Table 5.11: Evaluation results for MobileNet configurations across all dataset sizes.

MobileNet V3 is a high-performing architecture at larger data sizes, being the highest performer in the large dataset and coming close to the highest in the medium and small datasets. Notably, the accuracy figures seen here are significantly higher than the accuracies given in the MobileNet V3 paper [19], where the two configurations were tested on the ImageNet database. In the paper, top-1 accuracies of around 75% were reported, but here we see accuracies in the 90%-97% range.

However, it completely breaks down in the tiny dataset, with both large and small configurations ultimately learning to predict all outputs in the same class, as discussed in section 5.1.4. An interesting possible extension of this thesis would be to experiment with additional dataset sizes between tiny (200 training images) and small (2100 training images), to get a better idea of the size at which this breakdown begins to occur.

Excluding the tiny dataset, MobileNet V3 seems quite robust to the size of the dataset. Halving the number of training images when going from the large to the medium dataset results in a drop in accuracy below 5%, and halving it again (from medium to small) produces a further drop of only 2.5%. This is visualised in figure 5.21, where we can see that the decrease in performance has a gentle slope.

Figure 5.22 shows the validation loss over the training process for both configurations of MobileNet. It can be seen from the figure that `mobilenet_small` is subject to much more variance in the training process; this is particularly noticeable when comparing the configurations on the small dataset, where `mobilenet_large` has a much smoother curve of the loss changing compared to `mobilenet_small`. However, interestingly, the loss curves for the medium dataset are quite similar, especially once the difference in scale is taken into account. It is curious that both the large and small datasets ultimately result in convergence to a stable value, while

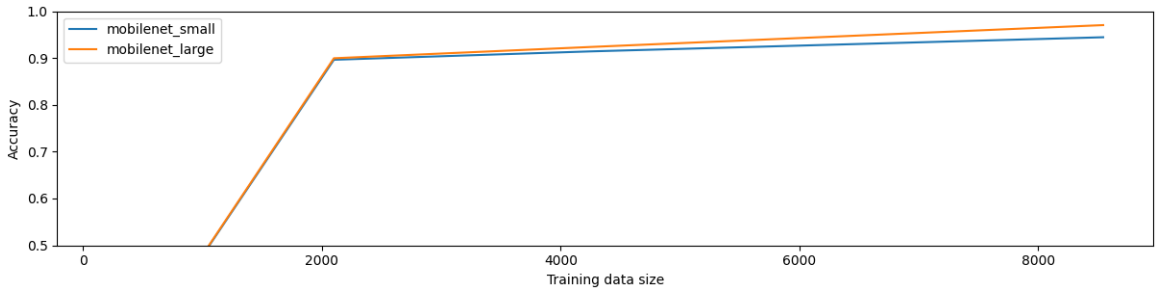

Figure 5.21: Accuracy of each MobileNet V3 configuration as a function of training data size.

medium does not; this may be because of the early stopping mechanism, and if the training process continued on the medium dataset then perhaps it would also reach a point of more stable behaviour.

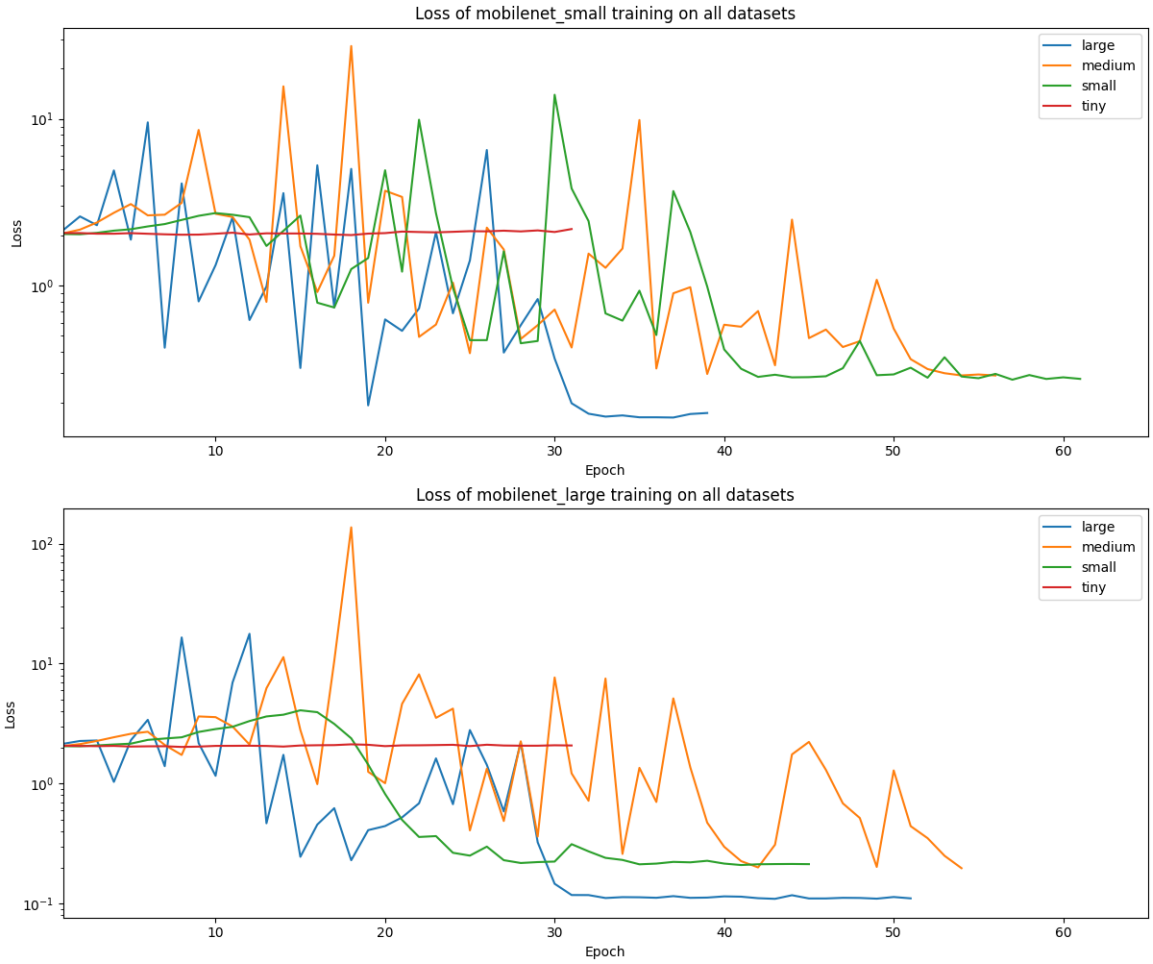

Figure 5.22: Validation loss and accuracy during MobileNet V3 training.

5.2.4 Vision Transformer (ViT)

The results of evaluating the Vision Transformer configurations across each of the different dataset sizes are collated in table 5.12. These are the same results as presented in section 5.1, but collated across all four dataset configurations for easier comparison.

Overall, the performance of ViT is disappointing, typically in the 75%-85% range. This is significantly worse than the results in the paper [16], where a configuration such as `vit_l_16` achieved a mean accuracy of 99.74% on the Oxford Flowers benchmark and 99.42% on the CIFAR-10 benchmark; here, the same configuration does not even exceed 90%. However, it is important to note that in the paper, the networks were pre-trained on the JFT-300M dataset before being fine-tuned for each benchmark, while here we have not performed any pre-training.

The configurations that break the input image down into a larger number of smaller patches (`vit_b_16` and `vit_l_16`) have consistently higher performance than the configurations that use a smaller number of larger patches. This is despite the latter configurations having a higher number of parameters. Intuitively it makes some sense that breaking the image down into

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy	Dataset
vit_b_16	86025992	16	39	6663.277	170.853	0.867	large
vit_b_16	86025992	16	52	4418.889	84.979	0.862	medium
vit_b_16	86025992	16	45	1920.397	42.675	0.779	small
vit_b_16	86025992	16	41	188.609	4.600	0.391	tiny
vit_b_32	87516680	4	31	2772.768	89.444	0.787	large
vit_b_32	87516680	4	31	1390.486	44.854	0.755	medium
vit_b_32	87516680	4	40	911.545	22.789	0.745	small
vit_b_32	87516680	4	45	110.871	2.464	0.591	tiny
vit_l_16	303604744	4	36	19202.255	533.396	0.833	large
vit_l_16	303604744	4	59	15906.408	269.600	0.867	medium
vit_l_16	303604744	4	76	10249.075	134.856	0.799	small
vit_l_16	303604744	4	66	915.314	13.868	0.591	tiny
vit_l_32	305592328	4	43	8485.654	197.341	0.767	large
vit_l_32	305592328	4	43	4033.520	93.803	0.789	medium
vit_l_32	305592328	4	42	1966.547	46.823	0.778	small
vit_l_32	305592328	4	47	236.630	5.035	0.591	tiny

Table 5.12: Evaluation results for ViT configurations across all dataset sizes.

smaller patches will allow the network to learn more fine-grained policies, though taken to an extreme (of 1x1 patches) this would very likely break down, so there must be some optimal patch size that could be searched for as a hyperparameter.

Compared to some of the other architectures, we also see from figure 5.23 that ViTs are more sensitive to smaller datasets, where the performance gets worse at a nonlinear rate as the size of the training data decreases, especially for the smaller patch sizes.

It is also interesting to examine the errors made by ViT in general. Figure 5.24 shows the average distribution of classifications across each size of dataset, for each configuration. We see some noticeable patterns in the errors across all configurations: classification of `ig` as `monocyte` or `neutrophil`, but also misclassification of `erythroblast`, `ig` and `lymphocyte` as `neutrophil`. While we have previously noted that some instances in the `ig` class have a different exposure and that may have affected classifications involving that class, that does not explain all of these misclassifications.

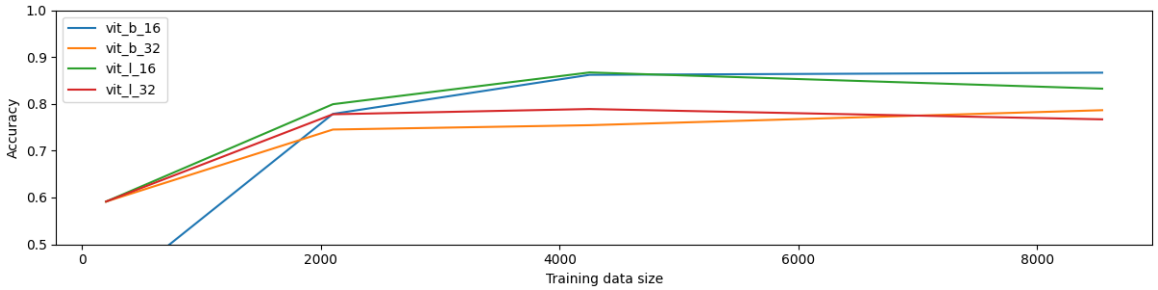

Figure 5.23: Accuracy of each ViT configuration as a function of training data size.

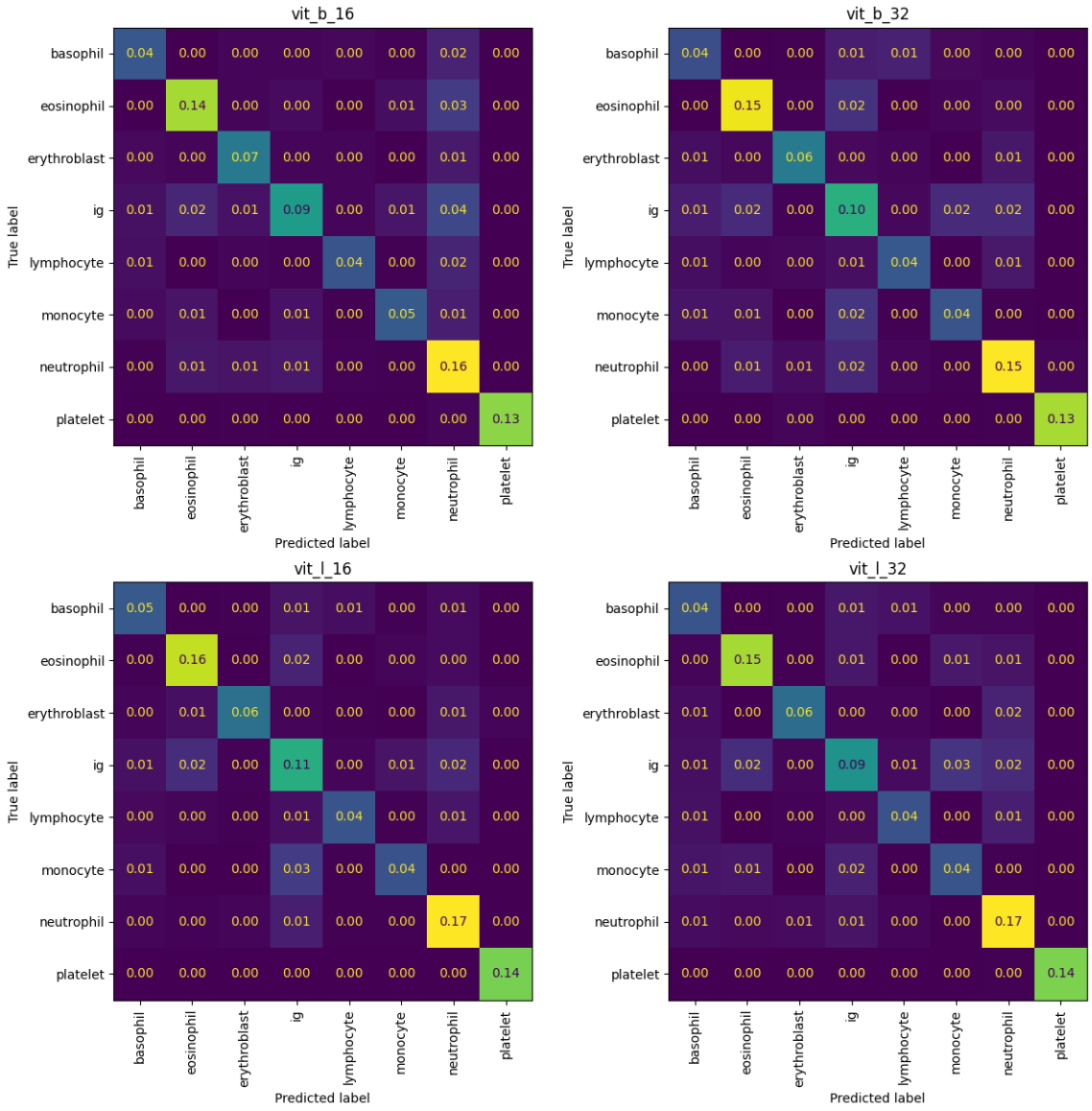

Figure 5.24: Average normalized confusion matrix for each ViT configuration across all dataset sizes.

5.2.5 MLP-Mixer

The results of evaluating the MLP-Mixer configurations across each of the different dataset sizes are collated in table 5.13. These are the same results as presented in section 5.1, but collated across all four dataset configurations for easier comparison.

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy	Dataset
mlpmixer_s32	4640048	64	31	473.770	15.283	0.789	large
mlpmixer_s32	4640048	64	32	241.749	7.555	0.723	medium
mlpmixer_s32	4640048	64	33	133.229	4.037	0.705	small
mlpmixer_s32	4640048	64	65	38.343	0.590	0.357	tiny
mlpmixer_b32	10909124	64	31	784.265	25.299	0.776	large
mlpmixer_b32	10909124	64	31	386.460	12.466	0.692	medium
mlpmixer_b32	10909124	64	31	195.001	6.290	0.701	small
mlpmixer_b32	10909124	64	68	56.832	0.836	0.513	tiny
mlpmixer_s16	17530280	64	31	1329.452	42.886	0.818	large
mlpmixer_s16	17530280	64	33	696.778	21.114	0.793	medium
mlpmixer_s16	17530280	64	31	329.826	10.640	0.755	small
mlpmixer_s16	17530280	64	75	97.247	1.297	0.548	tiny
mlpmixer_b16	30244472	32	31	2811.958	90.708	0.832	large
mlpmixer_b16	30244472	32	31	1402.966	45.257	0.769	medium
mlpmixer_b16	30244472	32	32	701.543	21.923	0.736	small
mlpmixer_b16	30244472	32	54	126.355	2.340	0.487	tiny
mlpmixer_l32	31283584	32	32	1527.609	47.738	0.829	large
mlpmixer_l32	31283584	32	36	823.259	22.868	0.784	medium
mlpmixer_l32	31283584	32	31	356.401	11.497	0.737	small
mlpmixer_l32	31283584	32	59	80.992	1.373	0.539	tiny
mlpmixer_l16	71133928	8	31	7564.939	244.030	0.825	large
mlpmixer_l16	71133928	8	31	3730.173	120.328	0.739	medium
mlpmixer_l16	71133928	8	31	1917.389	61.851	0.744	small
mlpmixer_l16	71133928	8	31	190.420	6.143	0.504	tiny

Table 5.13: Evaluation results for MLP-Mixer configurations across all dataset sizes.

As can be seen in figure 5.25, performance of MLP-Mixer seems to be more or less linear in the size of the dataset for sizes above 2100 (i.e. for all dataset configurations except tiny), but it then sharply decreases below that point. The configurations with 16x16 patch sizes consistently have slightly higher accuracy than the configurations with 32x32 patch sizes.

These results are in fact comparable to the ImageNet top-1 accuracy values reported in the paper [29]; the authors of that paper reported accuracy values ranging between 71.8% and 81.6%, close to the range of 69.2% to 83.2% reported across the large, medium, and small

datasets. No pre-training was used to obtain the figures in the paper so the comparison is fair.

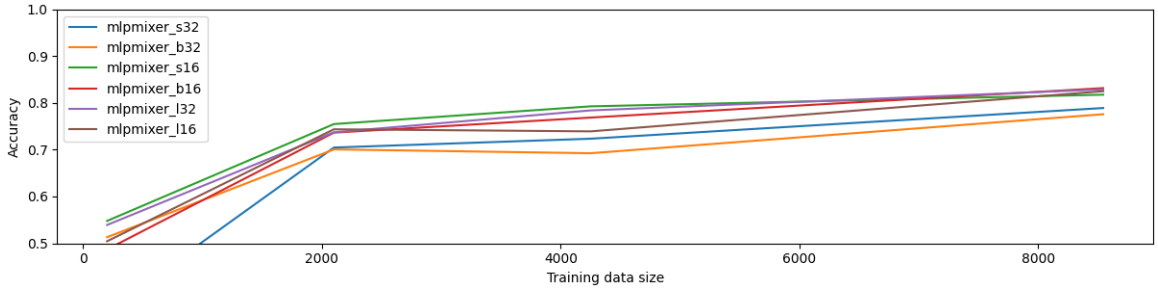

Figure 5.25: Accuracy of each MLP-Mixer configuration as a function of training data size.

Training behaviour is also very consistent across the different configurations. In all dataset configurations except for tiny, the early stopping mechanism halted training at or soon after the 31st epoch, which is the earliest point at which it is permitted to; this implies that in all of these cases the first 10 epochs were generally enough to bring the model to a minimum loss value, while training on the tiny dataset required 2-2.5x more epochs to reach a point of no further improvement.

5.2.6 ConvNeXt

The results of evaluating the ConvNeXt configurations across each of the different dataset sizes are collated in table 5.14. These are the same results as presented in section 5.1, but collated across all four dataset configurations for easier comparison.

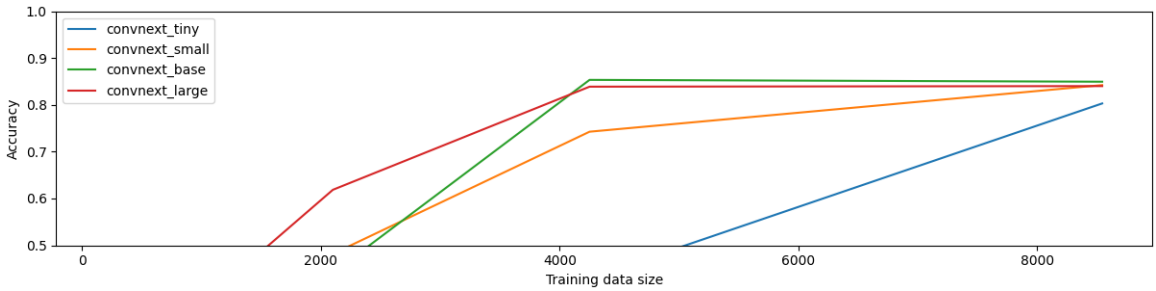

Figure 5.26: Accuracy of each ConvNeXt configuration as a function of training data size.

ConvNeXt was the slowest network to train, and competed with LeNet-5 for worst classification performance. While the numbers seen on the large dataset - 80.3% to 85.0% - are similar to the numbers reported in the paper [30], the numbers for the smaller datasets are much worse. Figure 5.26 visualises this, and it is clear that the performance drops off much more quickly than the other models: the larger configurations `convnext_large` and `convnext_base` do not see much decrease in accuracy when going from 8544 to 4250 training images, but drop off quickly if the dataset size is lowered further, while the smaller configuration `convnext_small` has a more noticable decrease, and `convnext_tiny` drops hugely between the large and medium dataset

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy	Dataset
convnext_tiny	27826280	32	100*	39372.405	393.724	0.803	large
convnext_tiny	27826280	32	51	12885.321	252.653	0.430	medium
convnext_tiny	27826280	32	63	7868.949	124.904	0.466	small
convnext_tiny	27826280	32	47	535.680	11.397	0.200	tiny
convnext_small	49460840	16	79	22896.489	289.829	0.842	large
convnext_small	49460840	16	82	15342.594	187.105	0.743	medium
convnext_small	49460840	16	54	4616.715	85.495	0.482	small
convnext_small	49460840	16	31	319.004	10.290	0.183	tiny
convnext_base	87574664	8	68	20278.432	298.212	0.850	large
convnext_base	87574664	8	100*	14861.772	148.618	0.853	medium
convnext_base	87574664	8	34	2562.891	75.379	0.441	small
convnext_base	87574664	8	31	252.687	8.151	0.174	tiny
convnext_large	196242632	8	63	46613.753	739.901	0.840	large
convnext_large	196242632	8	100*	33614.001	336.140	0.839	medium
convnext_large	196242632	8	63	10597.445	168.213	0.619	small
convnext_large	196242632	8	31	493.900	15.932	0.200	tiny

Table 5.14: Evaluation results for ConvNeXt configurations across all dataset sizes. Epoch values marked with * indicate configurations where the early stopping mechanism did not end the training early.

sizes. The figures in the paper include measurements from ConvNeXts that went through a pre-training process with the ImageNet-22k dataset, while the networks here have not been pre-trained, so it is perhaps expected that we would see worse performance; but it is interesting that even without pre-training we do see comparable performance for the large dataset.

ConvNeXt was also the architecture which most frequently was not stopped by the early stopping mechanism, meaning that it continued to show reductions in loss throughout the training period. An example of this is shown in figure 5.27, which is the training of `convnext_tiny` on the large dataset; there is a very clear linear decrease in loss and corresponding linear increase in accuracy, and so training for a greater number of epochs would likely have seen the accuracy increase further. However, the high seconds per epoch values - in this case, 393.72 seconds, or 6.5 minutes, per epoch - makes increasing the total number of epochs an expensive proposition. Figure 5.28 puts this in context with the other model configurations; in fact it seems that all configurations follow this generally linear improvement in loss, but slope of the improvement for the tiny configuration is much worse than for the larger configurations.

When reviewing the results of the tiny dataset, it was observed that all four ConvNeXt configurations had learned the behaviour of predicting the same class for all inputs. Figure 5.29 shows the confusion heatmaps for all configurations across all datasets, making it easy to see the progression towards this state as the dataset size decreases. We can see that even

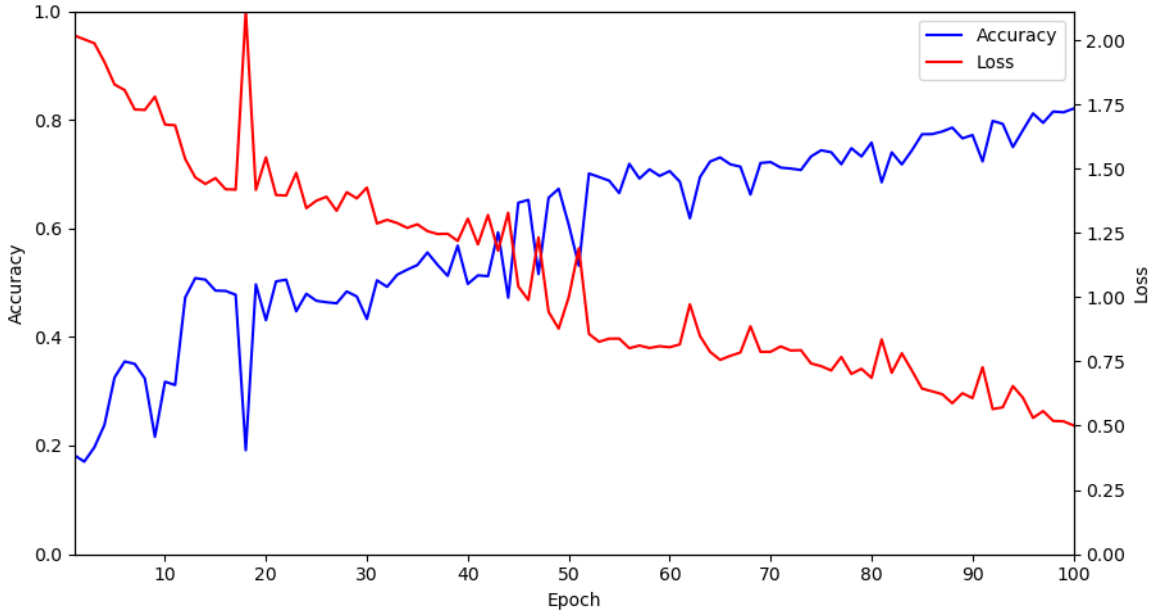

Figure 5.27: Validation loss and accuracy during `convnext_tiny` training, large dataset.

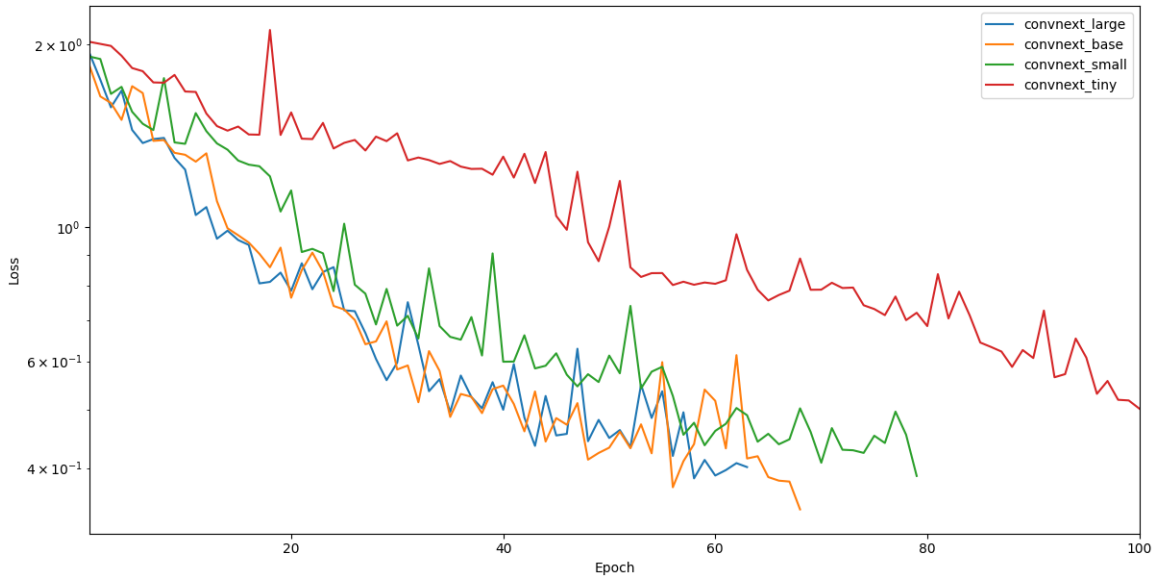

Figure 5.28: Validation loss for all configs training on the large dataset.

with the medium dataset the `convnext_tiny` model has begun to output classifications in only a limited subset of the available classes, and by the time we reach the small dataset, all four configurations are showing a strong bias towards only one or two classes. Notably, `platelet` seems to be exempt from this effect, in that the majority of instances in that class are still predicted correctly even when the rest of the model has broken down.

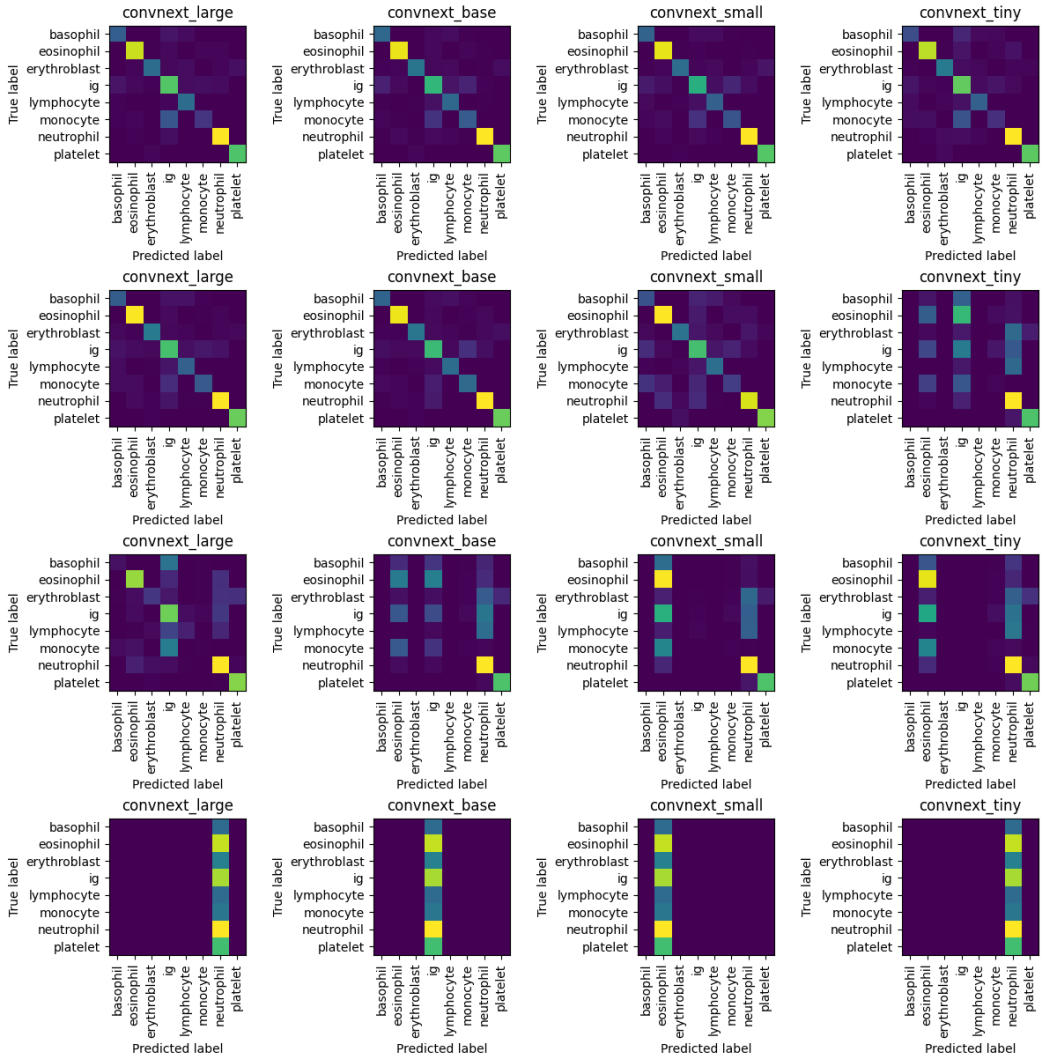

Figure 5.29: Confusion heatmaps for all configs on all datasets. Each row of maps was produced from models training on the same dataset, with the largest dataset at the top and the smallest at the bottom.

5.3 Performance of pre-trained models

In addition to evaluating models trained from scratch, three architectures - ResNet, MobileNetV3, an ConvNeXt - were also evaluated on the tiny dataset using pre-trained models

available in the torchvision library [3]. Each model was pre-trained on the ImageNet database, optimizing for the ImageNet-1K classification task; the output layer of each model was then discarded and replaced with a new output layer configured for 8 classes, instead of the 1000 classes present in ImageNet. The resulting models were then trained on the tiny dataset. Only the parameters in the new output layer were trained; the remaining parameters within the network were excluded from modification. Batch sizes were the same as for the non-pre-trained models, as listed in section 5.1.4.

The evaluation results of each pre-trained model are shown in table 5.15.

	Parameters	Batch size	Epochs	Total training time (secs)	Training time per epoch (secs)	Accuracy
convnext_tiny	7688	32	32	30.907	0.966	0.800
convnext_small	7688	32	45	61.835	1.374	0.870
convnext_base	10248	32	52	95.751	1.841	0.852
convnext_large	15368	32	42	126.303	3.007	0.878
mobilenet_small	599048	64	70	33.167	0.474	0.461
mobilenet_large	1240328	64	60	33.369	0.556	0.513
resnet_18	4104	64	56	32.573	0.582	0.783
resnet_34	4104	64	57	41.400	0.726	0.678
resnet_50	16392	64	67	65.944	0.984	0.687
resnet_101	16392	64	53	74.024	1.397	0.730
resnet_152	16392	64	66	119.799	1.815	0.730

Table 5.15: Evaluation results for pre-trained model configurations on the tiny dataset.

From these configurations, the highest performer is `convnext_large` with 87.8% accuracy, while the lowest performer is `mobilenet_small` with 46.1%. This surpasses the best performance on the tiny dataset with the non-pre-trained models (`resnet_34`, with 79.1% accuracy), and also surpasses the performance of non-pre-trained ConvNeXt even on the large dataset.

Compared to the results for the non-pre-trained models on the tiny dataset shown in table 5.7, the differences are stark. Table 5.16 shows the ratios between the results of the non-pre-trained models and the pre-trained models. We can see that all of the ConvNeXt configurations are at least 4x as accurate, and the MobileNet configurations are 2.5-3x as accurate. However, with the exception of `resnet_18`, the ResNet configurations are slightly less accurate as a result of pre-training. We do also see significantly reduced training times and seconds per epoch values for all configurations, which is to be expected when the number of parameters actually being optimised has been reduced so much - in the case of `convnext_large`, only 0.0078% as many parameters need to be trained.

Validation loss for the best configuration of each model is shown in figure 5.30. One immediately notable aspect is that while the ResNet and ConvNeXt configurations show a clear downward trend, the MobileNetV3 configuration has not decreased much at all across the training period. While the pre-trained configuration is clearly much better than the non-pre-trained

	Accuracy	Epochs	Parameters	Training time	Time per epoch
convnext_tiny	4.000000	0.680851	0.000276	0.057696	0.084742
convnext_small	4.761905	1.451613	0.000155	0.193839	0.133534
convnext_base	4.900000	1.677419	0.000117	0.378933	0.225902
convnext_large	4.391304	1.354839	0.000078	0.255725	0.188749
mobilenet_small	2.650000	2.258065	0.392547	1.748004	0.774116
mobilenet_large	2.950000	1.935484	0.294455	1.182076	0.610739
resnet_18	1.097561	1.272727	0.000367	0.713846	0.560879
resnet_34	0.857143	0.814286	0.000193	0.510643	0.627105
resnet_50	0.908046	0.943662	0.000697	0.490285	0.519555
resnet_101	0.954545	0.898305	0.000386	0.388378	0.432346
resnet_152	0.943820	1.375	0.000282	0.574409	0.417752

Table 5.16: Proportion change from non-pre-trained to pre-trained configurations.

one, which ultimately classified all images in a single class, it also looks like additional epochs will not significantly improve its performance further. For ResNet, we see the same kind of high variance as we have seen with non-pre-trained models, while the ConvNeXt exhibits it to begin with, but then stops at a point consistent with the reduction in learning rate shown in figure 5.31. The reduction in learning rate does not seem to have the same stabilising effect on the ResNet.

The confusion matrices for the best pre-trained configuration of each architecture are shown in figure 5.32. We can see that for `resnet_18` and `convnext_large`, we do not see the same kind of pattern as we have seen in previous confusion matrices; in particular, there is no confusion between classes `ig`, `monocyte` and `neutrophil`. For MobileNet V3, we see that the errors are not distributed across the matrix, but are primarily focused around misclassifying instances as `lymphocyte`.

	mAP @ 5	mAP @ 100
convnext_large	0.950	0.12625
resnet_18	0.850	0.11250
mobilenet_large	0.425	0.07375

Table 5.17: Mean average precision results for pre-trained configurations.

Lastly, the mean average precision scores for the best pretrained models are shown in figure 5.17. Both `resnet_18` and `convnext_large` show an mAP@5 accuracy which is greater than their average accuracy (78.3% and 87.8% respectively), while for `mobilenet_large` the mAP@5 accuracy is slightly lower than the average accuracy (51.3%). The mAP@100 accuracy for all models is low, but this is because the tiny test dataset does not contain enough instances in each class.

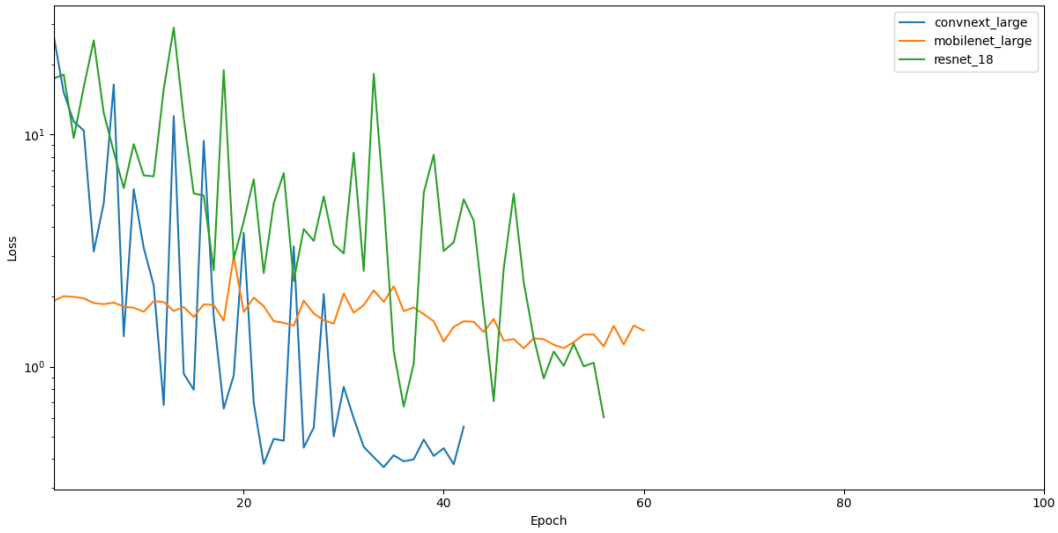

Figure 5.30: Validation loss for pre-trained models.

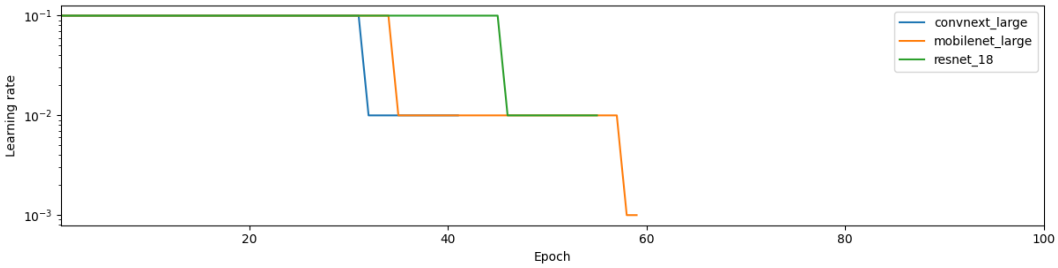

Figure 5.31: Learning rate of pretrained models during training process

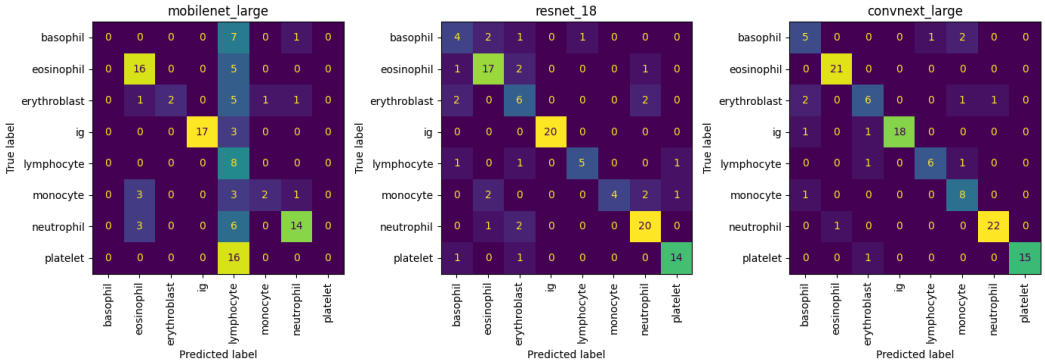

Figure 5.32: Confusion matrices for pre-trained models.

Chapter 6

Discussion

As per the original proposal, we are interested in answering two questions: firstly, which architecture is the best choice for classification of peripheral blood smear imagery? Secondly, to what extent is it possible to adapt these architectures to smaller datasets?

The two architectures that consistently show the highest accuracy are ResNet and MobileNet, with the exception of the tiny dataset configuration where the MobileNet configurations break down. So, if we are looking for a single architecture which balances high accuracy against robustness with small data sets, ResNet seems to be the clear winner. Conveniently, it is also one of the fastest to train, with even the largest configuration (152 layers) taking under an hour and half to train on the largest dataset.

One of the advantages of having used the same dataset as [1] is that we are able to directly compare our results. In that paper, fine-tuning the Vgg-16 architecture was able to classify the images with 96% accuracy. We have not examined the Vgg-16 architecture in this thesis, but we note that the MobileNet V3 model was able to surpass that performance and achieve 97.1% accuracy, despite using a substantially smaller training data set size, no data augmentation, and training from scratch. MobileNet V3's degradation in performance as the dataset got smaller perhaps makes it less suitable for some applications, but if a large dataset is available then it appears to be the best choice.

It is interesting that the best ResNet configuration for each size of dataset is typically not the largest one. In fact, the smallest two configurations (18 and 34 layers) have better accuracy than the largest one across all sizes of dataset. Despite the larger model being capable of learning more complex functions than the smaller model, it has not done so in this case. This is not likely to be a consequence of the architecture itself, but instead a consequence of the hyperparameters used; it has been previously observed [49] that changes to the training process itself can significantly improve ResNet performance, and while in this thesis it was helpful to keep the training parameters consistent to make the results more comparable, it is possible that the larger configurations would have benefitted from more flexibility and exploration around learning rate, batch size, and early stopping.

It is also interesting to see MobileNet perform so well, considering that the primary motivation behind its design is to optimise for the ability to run on mobile and embedded systems. We might have expected that these hardware constraints would have resulted in a trade-off,

where accuracy is worsened in order to make the network smaller and more efficient to evaluate, but it seems that this is not the case: despite being much smaller, it is more than capable of keeping up with the performance of ResNet, at least for larger training data set sizes.

ViT, MLP-Mixer, and ConvNeXt are newer and more advanced architectures, but they did not demonstrate good performance, both compared to the other models in this thesis and also compared to their performances as presented in their papers. The primary explanation here is that none of these models were subject to a pre-training process. This suggests that the use of pre-training and transfer learning is a significant factor in the success of these architectures; it is not the structure of the network alone that makes them successful, but the state that the network is in after the pre-training process is completed.

This raises a lot of questions about that pre-training process: what are the properties of a good dataset to use for pre-training? What are the features that the model has actually learned from the pre-training? Would the model perform just as well if pre-trained on different datasets? Given that these architectures are developed primarily with recognition of full-color photographs of real-world objects in mind, it makes sense to use a dataset such as ImageNet [14] for pre-training them, as the dataset is built on that kind of image; but when applying the model to a much more specialised domain such as peripheral blood smear microscopy, is the same kind of pre-training really appropriate? If the features that the model has learned are oriented around features such as colour, then this will not help when dealing with monochrome microscope images, but features that are more related to shape or curvature may well be useful.

We have seen a number of instances where the training process may not have been sufficiently regularised. The high variability in loss for some networks, as well as the sometimes surprising behaviour of the early stopping mechanism, suggest that further iteration on both the learning rate adjustment and the early stopping mechanism may have been beneficial. In particular, once it became necessary to alter the batch size in order to fit into available GPU memory, it would probably have been appropriate to adjust the learning rate as well; one suggestion [25] is that when multiplying the batch size by k, the learning rate should also be multiplied by $\sqrt{k}$, though other authors suggest a simple linear scale. This may have produced better accuracy from some of the models, at the expense of having an additional difference between the configurations to reason about.

When considering the best-performing models in the large dataset, figure 5.2 shows that both ResNet and MobileNet tended to have the same confusion: mistaking monocytes and neutrophils for immature granulocytes, and vice versa. Note that neither network mistook a significant number of monocytes for neutrophils or neutrophils for monocytes. It would be interesting to carry out a study of the classification mistakes made by specialist clinicians, to see whether this is a common confusion amongst specialists as well, or whether it is an area where the deep learning model is falling behind the specialist.

The last point to make with regards to the first question is about a limitation of the dataset. While the dataset used includes 8 different types of cell, 6 of them are types of white blood cell. Of the remaining two, one is the platelet, and the other is the erythroblast - an immature red blood cell with a nucleus. The dataset does not include any erythrocytes (mature red blood cells). These cells may be relatively easy to identify because they are the ones lacking a nucleus, but given that the original goal of the thesis was to be able to identify all of the

common and clinically interesting cells in blood smear microscopy, it would surely have been good to include the erythrocytes, the most common cell type in the blood, within the dataset. In fact, almost all of the images in the dataset do include red blood cells in the background - their clear 'donut' shape is visible in the example images in figure 4.2 - but they are not isolated or labelled, and the staining method used to highlight the other cell types would not work with them, so preparing equivalent images may be difficult. It is possible that this aspect of the problem is better solved simply by image segmentation techniques, where all cells present in an image are distinguished, and then classification could be applied on the basis that all cells which are not the central cell in the image should be classified as erythrocytes.

Regarding the question of how well these techniques can be adapted to smaller datasets: we have seen that the ResNet architecture still retains an accuracy of 79.1% even when training with only 200 labelled images, but the other architectures all demonstrated a significant loss in quality. However, once pre-training was applied, ConvNeXt then became the most successful architecture, achieving an accuracy of 87.8%.

Pre-training evidently makes a significant difference, at least for ConvNeXt and MobileNet V3. If the networks are expected to learn even the most basic features of images - such as the the idea that adjacency is important, something that is structurally encoded in CNNs such as ResNets but is not present in an architecture such as ViT - then it is not actually necessary to learn these things from the small dataset of blood cell images, but can be learned from other image datasets first, and then transfer learning can be used to begin training on the blood cell images with a model that already has many of the interesting relationships captured in its parameters.

We should also consider the choice of 200 as the size for the tiny data set. This was chosen somewhat arbitrarily, and we have seen that the quality of each architecture appears to decrease in a non-linear way, so small changes to this number may produce a disproportionate effect on prediction quality. If clinicians are prepared to label an additional 50 or 100 images - or if they are in practice not prepared to label as many as 200 - then it may have significant impact on the performance. Another way to increase this number, without requiring any extra work by clinicians, would be to use data augmentation techniques to generate additional synthetic data from the existing labelled images. The risk with this technique is that it fails to capture the true diversity of how blood cells may present in microscope images, as the transformations we would perform in data augmentation would likely be limited to things that change the entire image - adjusting the exposure, the contrast, rotating or flipping, etc - but it would still be worth exploring.

From the confusion matrices in figure 5.9, we saw that both ConvNeXt and MobileNet broke down to the point of predicting all inputs as a single class, though ViT was a bit more successful. One thing to note about the ViT confusion matrices, as well as the ConvNeXt matrices on the small dataset in figure 5.29, is that accuracy of predictions for platelets remained high even when the rest of the model was breaking down. It is interesting that platelets would be exempt in this way. One possible explanation is that platelets tend to be smaller than the other types of cell, and so the network was simply classifying based on cell size, but more investigation would be needed to confirm this.

Chapter 7

Conclusions

7.1 Conclusions

In this thesis, we have explored the use of six different neural network architectures to the problem of classifying blood cell images, with particular attention paid to the effect of dataset size on classifier performance. We conclude that the best choice of architecture for classifying such images, when no models are pre-trained, is the ResNet architecture. We also conclude that the ResNet architecture is the most robust choice when training on very small datasets, that it is possible to train a suitable ResNet model with only 200 images and still obtain a classification accuracy of 79.1%. When model pre-training and transfer learning is used, we conclude that ConvNeXt surpasses ResNet, with a classification accuracy of 87.8%. Our observation of MobileNet V3 achieving classification accuracy of 97.1% also exceeds the previous best known performance of any classifier on this specific dataset.

Beyond these answers to the initial research questions, there are also a number of other things we have learned. We have seen that a number of other architectures - including ViT, MLP-Mixer, and ConvNeXt - do not perform nearly as well as in their literature across most configurations, at least when not pre-trained. We have also seen that their performance degrades with small datasets, particularly with ConvNeXt, and that this degradation in performance is both non-linear and also dependent on network size in most cases. However, we have also seen that applying pre-training results in significant performance improvements for ConvNeXt and MobileNet V3 at least, when classifying images in a small dataset.

With reference to the original research objectives, we believe they have all been met to a reasonable standard, and in some cases exceeded: the original objective was to develop a comparative analysis of 3 architectures across 2 sizes of dataset, but in fact we have developed an analysis of 6 architectures across 4 sizes of dataset.

7.2 Future work lines

There are a number of interesting potential lines of future work. The most obvious one is to extend the analysis to include pre-training of the remaining architectures, either by finding and

adapting existing pre-trained models, or by performing pre-training of each model ourselves. It would then make sense to extend the experiments with the pre-trained models to include all four sizes of dataset, rather than only the tiny dataset size. Given that blood cell imagery is quite specialised and different from what will be found in most large image databases (such as ImageNet), performing our own pre-training would also make it possible to explore whether using different pre-training datasets have different impacts.

It would then be natural to extend the comparison to additional deep learning architectures, such as Xception [11] or ConvMixer [44], or to other newer state-of-the-art networks that may be published after this thesis.

Given the quite pronounced difference between 'small' and 'tiny' dataset sizes in many of the results, another simple extension of this thesis would be to conduct further experiments with additional dataset sizes between the two. This could provide more insight into the point at which MobileNet breaks down, as well as better refining the nonlinear decrease in performance experienced by ResNet (see figure 5.17), amongst other things.

As with many machine learning problems, further experimentation could be performed with the learning hyperparameters. In particular, the approach taken to adjusting the learning could be changed, or at least the initial learning rate and thresholds for adjusting it could be altered; this may help to reduce the high variance seen in some of the models and configurations, such as the smaller ResNet models. Secondly, it may be appropriate to experiment with the batch size. In this thesis the batch size was generally set for practical reasons, using a value that was as high as possible while still fitting inside available GPU memory; if a GPU with more video memory could be obtained, or if the implementation could be altered to use some kind of distributed training and evaluation across multiple GPUs, then it would be possible to explore larger batch sizes without running out of GPU memory. Alternatively, smaller batch sizes could be explored for the smaller models, increasing training time but allowing more direct comparison between the results from different model configurations. Thirdly, the early stopping mechanism could be further developed; while two strategies were explored and ruled out in this thesis, the mechanism that was used may still not be optimal, particularly with regards to the threshold of 0.1 required for an epoch to be considered an improvement over previous epochs. Expressing the threshold as a percentage of the loss value, rather than an absolute value, may help to prevent the early stopping mechanism from being applied prematurely when the model has reached low loss values but is still improving. Another aspect of the learning process that could be modified is the optimizer used; adopting the Adam [23] or AdamW [31] optimisers may improve accuracy of multiple configurations.

There are multiple opportunities regarding data augmentation. Firstly, it would be appropriate to explore data augmentation as a way of further mitigating batch effects in the data. It was seen in the confusion matrices for each of the datasets that most of the misclassifications involved the `ig` class, which we observed at the dataset preparation stage was exhibiting a batch effect with a difference in exposure compared to the other classes. While we attempted to correct for this by normalizing each of the images, an alternative approach would be to instead deliberately introduce additional images in each class which have intentionally altered exposure, to try and prevent the models from learning that the difference in exposure is significant. It still might be a challenge to do this in a way that results in an equal distribution of

exposure levels for each cell type, but it seems worth exploring.

Secondly, it would be worthwhile to explore the extent to which data augmentation can compensate for datasets of limited size. As the goal of using a small dataset is to limited the number of images that must be manually labelled, data augmentation techniques offer a way to obtain a larger dataset without increasing the amount of manual work that must be performed. We saw that the performance of most models dropped sharply between the small (2100 training images) and tiny (200 training images) dataset configurations; it may be possible to take the 200 manually labelled images represented by the tiny set but increase the dataset size to 2100 by generating copies of these images with altered exposure, that are rotated, translated, have noise added, etc. If this is able to recreate the accuracy seen on the small dataset then it would be a notable improvement.

Another interesting direction of exploration would be to examine whether image segmentation techniques, such as StarDist [48] or SplineDist [32], can assist the classification process. In this thesis, we have provided raw images to each model and asked for a single classification of each image, but in a clinical setting it may be impractical to assume that every microscopic image from a peripheral blood smear only contains a single cell of interest, and that we may in fact want to identify multiple different cells at different locations in the image. Furthermore, segmentation techniques may allow us to provide the model with an extra input channel containing a mask that indicates which pixels are actually part of the cell of interest. If the segmentation technique used is able to generate a separate mask for each cell in the image, then this may not only allow us to increase the accuracy of the classifier for the individual cell by preventing it from using information that is not part of the cell itself, but it would also allow us to feed the same input image to the classifier with a different mask for each cell, producing a separate classification result for each cell in the image.

For the models themselves, it could be interesting to explore whether modifying the implementations to accept 1-channel grayscale inputs instead of 3-channel has any impact on performance, either from the point of view of accuracy or from the point of view of training behaviour and speed. It was not possible to explore this during this thesis due to the fact that PyTorch does not expose direct control over the number of input channels for its implementations of the models, but given more time to develop custom implementations of each architecture similar to the LeNet-5 implementation, adapting the architectures to 1-channel inputs should be achievable. Conversely, it may also be interesting to remove the greyscale preprocessing step and provide the networks with full colour images; in principle discarding the colour information is forcing the networks to learn features based on shape and value rather than colour, but this could be explored experimentally.

7.3 Planning retrospective

Overall, the work has proceeded according to the plan and methodology laid out at the beginning of the thesis.

The biggest deviation was the decision to perform additional experiments with pre-trained models. It was not realised until the initial experiments had been completed that the models

without pre-training would have such poor performance on the tiny dataset, and it was not until this late point that it was discovered that the torchvision library's pre-trained models could be used for transfer learning. So, the additional experiments with pre-trained models were a late addition to the thesis.

With regards to the sequence of tasks carried out, the set of architectures to be evaluated was expanded from 3 to 6 prior to performing analysis on the small dataset. In hindsight, this was not the best choice; it would have been better to complete the analysis on the first three models entirely, before expanding the scope to include 3 additional models. This would have reduced risk by ensuring that the initial objectives were fully met before expanding the scope, and would have helped to inspire investigation of pre-trained models earlier in the project.

The original estimate of 5 days for the implementation of each architecture turned out to be a significant overestimate, due to the discovery of the model library in torchvision that made it extremely quick to create and train instances of each architecture of interest.

With regards to the methodology, it was necessary to iterate somewhat on the implementation of the training process, particularly with regards to the early stopping mechanism. The original work plan suggested spending 11 days to develop this implementation of the training process before carrying out the training for each architecture of interest, but in practice this was not reflective of the right way to develop such an implementation: the implementation needed to be developed at the same time as implementing at least two of the architectures, in order to fully ensure that the implementation was fit for purpose. The initial 10 days were useful for becoming familiar with the PyTorch framework and setting up the basic training algorithm, as well as supporting facilities such as tools for visualisation and evaluation, but it became clear as architectures were subsequently implemented that further changes were needed to the core algorithm and so these were made simultaneously with implementing the architectures.

The initial plan also laid out 20 days at the end of the project for writing this thesis report, but in hindsight it would have been a better plan to break the report up into smaller milestones and ensure that it was being written throughout the entire course of the project. Writing the analysis of the early results as soon as they were available, rather than waiting until all architectures had been fully evaluated for each dataset before beginning to write the analysis of that dataset, would have revealed some issues earlier and perhaps presented more of an opportunity to improve the methodology and increase the contribution made by this thesis.

Chapter 8

Glossary

backpropagation training An algorithm for training a neural network in which the gradients and necessarily modifications to network parameters are computed one layer at a time, starting at the final layer and working backwards, as opposed to calculating the gradients across the entire network simultaneously. 25

batch effect An observable and unintended bias in sections of the input, resulting from aspects of the way that 'batch' of data was collected which differs from other data. For example: bias introduced by using a different microscope; using a staining fluid which was not prepared to the exact same specifications as for other samples; a different background light level when taking photographs; differences in how clinicians carried out the data collection process; etc. 21, 22

channel One component of information that is present for each entry in the data. For example, colour images typically have 3 channels for each pixel in the image: one for red, one for green, and one for blue. 22

checkpoint A record of the current state of the system, including all information required to restore the system to this state at a future point in time. 27

Convolutional Neural Network A neural network which features one or more layers that apply a shared set of weights to multiple different neighbourhoods in the input data. 11, 16

early stopping The practice of halting the training process for a model once it appears that further training will not produce any improvement. 26

epoch One iteration of the training process, including a complete pass through the training data and any subsequent validation and adjustment to the learning parameters. 25

GPU Graphics Processing Unit, a specialist piece of hardware in a computer. While GPUs were originally created for graphics rendering tasks, they have been found in recent years

to be also very well suited to large-scale matrix operations, such as those involved in evaluating neural networks. 26

gradient descent An algorithm for finding the minimum value of a function. The function is differentiated with respect to its inputs, in order to find the gradient. This gradient is used to modify each input by a small amount, and then the function is evaluated again. The process is repeated until the gradients are sufficiently small. 25

learning rate When performing gradient descent, the amount by which parameters are modified with each step of the algorithm. A small learning rate means that parameters are modified by small amounts, while a larger learning rate means that they are modified by larger amounts. 26

loss function A function that describes the level of error in predictions made by a model. The loss function must always be differentiable with respect to the parameters of the model, because this allows us to derive a measure of the extent to which changes in the parameters will change the loss function, so that we can iteratively modify the parameters and reduce the level of error. 24

one-hot encoding A method of representing a numeric variable which is categorical (i.e. the ID number of a category) as a vector of numbers, where each component of the vector corresponds to one possible category. The vector is zero in all components apart from the component representing the desired ID number; that component is set to one. 23, 24

peripheral blood smear A thin layer of blood which is smeared on a microscope slide or film and then stained to make it possible to examine the cells under a microscope. 11, 12, 19

pre-training Training a model on a separate dataset before using transfer learning techniques to adapt and train it for the problem at hand, so that the initial weights of the network have already been fitted to a (typically related) problem. 29, 56, 59, 60, 66

preprocessing Modifications made to data before analysing it or using it to fit a model. This may include modifications such as cropping, rotating, reducing noise, increasing or decreasing brightness and contrast, or more complex modifications such as edge detection. 22

Bibliography

[1] Acevedo, A., Alférez, S., Merino, A., Puigví, L., and Rodellar, J. Recognition of peripheral blood cell images using convolutional neural networks. *Computer Methods and Programs in Biomedicine 180* (2019), 105020.

[2] Acevedo, A., Merino, A., Alférez, S., Molina, Á., Boldú, L., and Rodellar, J. A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. *Data in brief 30* (04 2020), 105474–105474.

[3] Albardi, F., Kabir, H. M. D., Bhuiyan, M. M. I., Kebria, P. M., Khosravi, A., and Nahavandi, S. A comprehensive study on torchvision pre-trained models for fine-grained inter-species classification. *2021 IEEE International Conference on Systems, Man, and Cybernetics* (2021).

[4] Aslan, A. Blood cell detection dataset from peripheral blood smear, 2400+ labelled cell images, April 2020. https://aslan.md/blood-cell-detection-dataset-from-peripheral-blood-smear/.

[5] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate, 2014.

[6] Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives, 2012.

[7] Bozinovski, S. Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica 44* (2020), 291–302.

[8] Cheng, S. Bccd dataset. Github repository, February 2018. https://github.com/Shenggan/BCCD_Dataset.

[9] Cheuque, C., Querales, M., León, R., Salas, R., and Torres, R. An efficient multi-level convolutional neural network approach for white blood cells classification. *Diagnostics 12*, 2 (2022).

[10] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[11] Chollet, F. Xception: Deep learning with depthwise separable convolutions, 2016.

[12] Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., and Wei, Y. Deformable convolutional networks, 2017.

[13] De Vet, H. C. W., Koudstaal, J., Kwee, W.-S., Willebrand, D., and Arends, J. W. Efforts to improve interobserver agreement in histopathological grading. *Journal of Clinical Epidemiology 48*, 7 (2022/03/03 1995), 869–873.

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

[15] Döhner, H., Estey, E. H., Amadori, S., Appelbaum, F. R., Büchner, T., Burnett, A. K., Dombret, H., Fenaux, P., Grimwade, D., Larson, R. A., Lo-Coco, F., Naoe, T., Niederwieser, D., Ossenkoppele, G. J., Sanz, M. A., Sierra, J., Tallman, M. S., Löwenberg, B., and Bloomfield, C. D. Diagnosis and management of acute myeloid leukemia in adults: recommendations from an international expert panel, on behalf of the european leukemianet. *Blood 115*, 3 (2010), 453–474.

[16] 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. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[17] He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2980–2988.

[18] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[19] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3, 2019.

[20] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[21] Jung, J., Matemba, L. E., Lee, K., Kazyoba, P. E., Yoon, J., Massaga, J. J., Kim, K., Kim, D.-J., and Park, Y. Optical characterization of red blood cells from individuals with sickle cell trait and disease in tanzania using quantitative phase imaging. *Scientific Reports 6*, 1 (2016), 31698.

[22] Khouani, A., El Habib Daho, M., Mahmoudi, S. A., Chikh, M. A., and Benzineb, B. Automated recognition of white blood cells using deep learning. *Biomedical Engineering Letters 10*, 3 (2020), 359–367.

[23] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization, 2014.

[24] Kouzehkanan, Z. M., Saghari, S., Tavakoli, S., Rostami, P., Abaszadeh, M., Mirzadeh, F., Satlsar, E. S., Gheidishahran, M., Gorgi, F., Mohammadi, S., and Hosseini, R. A large dataset of white blood cells containing cell locations and types, along with segmented nuclei and cytoplasm. *Scientific Reports 12*, 1 (2022), 1123.

[25] Krizhevsky, A. One weird trick for parallelizing convolutional neural networks, 2014.

[26] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[27] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation 1*, 4 (12 1989), 541–551.

[28] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[29] Liu, H., Dai, Z., So, D., and Le, Q. Pay attention to mlps. *Advances in Neural Information Processing Systems 34* (2021).

[30] Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s, 2022.

[31] Loshchilov, I., and Hutter, F. Decoupled weight decay regularization, 2017.

[32] Mandal, S., and Uhlmann, V. Splinedist: Automated cell segmentation with spline curves. In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)* (2021), pp. 1082–1086.

[33] Marcel, S., and Rodriguez, Y. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia* (New York, NY, USA, 2010), MM '10, Association for Computing Machinery, pp. 1485–1488.

[34] Nickolls, J., Buck, I., Garland, M., and Skadron, K. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue 6*, 2 (mar 2008), 40–53.

[35] Parab, M. A., and Mehendale, N. D. Red blood cell classification using image processing and cnn. *SN Computer Science 2*, 2 (2021), 70.

[36] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance

deep learning library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[37] Patil, A., Patil, M., and Birajdar, G. White blood cells image classification using deep learning with canonical correlation analysis. *IRBM 42*, 5 (2021), 378–389.

[38] Paul, S., and Chen, P.-Y. Vision transformers are robust learners. *arXiv preprint arXiv:2105.07581* (2021).

[39] Peripheral blood lab. Systems Cell Biology @ Yale. http://medcell.med.yale.edu/systems_cell_biology/blood_lab.php.

[40] Robertson, S. A new interpretation of average precision. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2008), SIGIR '08, Association for Computing Machinery, pp. 689–690.

[41] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *CoRR abs/1409.4842* (2014).

[42] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision, 2015.

[43] Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems 34* (2021).

[44] Trockman, A., and Kolter, J. Z. Patches are all you need? *arXiv preprint arXiv:2201.09792* (2022).

[45] Union, I. T. Recommendation itu-r bt.601-7: Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios, 2017.

[46] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

[47] Wang, P. An all-mlp solution for vision, from google ai. Github repository. https://github.com/lucidrains/mlp-mixer-pytorch.

[48] Weigert, M., Schmidt, U., Haase, R., Sugawara, K., and Myers, G. Star-convex polyhedra for 3d object detection and segmentation in microscopy. In *The IEEE Winter Conference on Applications of Computer Vision (WACV)* (March 2020).

[49] Wightman, R., Touvron, H., and Jégou, H. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476* (2021).

[50] Yao, X., Sun, K., Bu, X., Zhao, C., and Jin, Y. Classification of white blood cells using weighted optimized deformable convolutional neural networks. *Artificial Cells, Nanomedicine, and Biotechnology 49*, 1 (2021), 147–155. PMID: 33533656.

[51] Zeiler, M. D., and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision* (2014), Springer, pp. 818–833.

[52] Zheng, X., Wang, Y., Wang, G., and Liu, J. Fast and robust segmentation of white blood cell images by self-supervised learning. *Micron 107* (2018), 55–71.