Application of deep learning based methodology for the optimisation of monolayer classification and white blood cell localisation in avian blood samples

Erica Andersdotter

Master's Thesis in
Biomedical Engineering

2024

CellaVision, Department of Biomedical Engineering

Supervisors: Kent Stråhlén (CellaVision)
Christian Antfolk (LTH)

Examiner: Martin Stridh

Abstract

Current automated haematology systems lack the functionality of avian blood analysis using 10x magnification, which is an important feature as it allows for faster and more cost-effective blood analysis. The problem originates from the difference in red blood cell morphology in avian blood compared to mammalian blood, as the former are nucleated. The project was divided in two objectives, finding monolayers and localising white blood cells within them. For the first task, a convolutional neural network was optimised with particular focus on monolayer precision and the model's ability to distinguish between classes. This was trained with a multiclass dataset using semi-supervised learning. The proposed model, including loss concentrating on challenging instances, reported an overall accuracy of 95.1% as well as recall and precision values over 87%. Monolayers could be found with 79.9% precision. Future improvements could be expert-based dataset annotations as well as k-fold cross-validation to validate the robustness of the model. The second task deployed an object detection optimisation with varying augmentation settings for white blood cells in 333 samples. The best performing model showed an average precision of 13.7% and a slightly higher value of 48.6% with an overlap threshold of 50%. This signified that the model had difficulties with precise localisations of the white blood cells, which might be explained by the cells having too small relative areas. Future improvements could hence include alternative base models with more suitable architecture for distinguishing small objects in dense images as well as techniques to recreate higher resolution, mimicking images collected with higher magnification.

Keywords: Convolutional neural networks, monolayers, avian, machine learning, object localisation

Acknowledgements

Firstly, I would like to thank Cellavision for giving me the opportunity to carry out my master's thesis project, and for all the support I've received. I would especially like to thank Kent Stråhlén for his work as supervisor. Furthermore, I wish to thank Irme Groothius for her invaluable support during the start of my thesis, as well as Måns Jarlskog, for answering some of my technical questions, and Mats Erikson, for showing me how to collect box coordinates. I'm also grateful to Susanne Chau and the entire BF Applications Team for their encouragement, warm welcome and many fikas.

I would also like to thank my supervisor at LTH, Christian Antfolk, for his guidance.

Finally, my sincerest gratitude to everyone who helped me progress in this project by providing support, knowledge and encouragement.

Contents

Chapter 1
Introduction

Blood analysis has been a common diagnostic tool in both human as well as veterinarian medicine, due to the extent of real-time physiological information that can be conveyed through the blood [1]. It is therefore an essential step in supporting evaluation of a patient's health status [2]. Non-surprisingly, blood samples therefore also have the highest prevalence in laboratory medical facilities compared to other diagnostic tests [1]. Since rapid and accurate diagnoses are significant for a successful treatment of a disease, it is necessary to perform the analysis as quickly as possible [3]. Blood can be analysed on both chemical and physical properties in both qualitative as quantitative approaches, such as the number of red blood cells (RBCs), white blood cells (WBCs), haemoglobin (Hb) concentration, antigens, activity of enzymes, viscosity, acidity and clotting abnormalities, among other things [4]. There are today over one hundred different haematological tests and procedures for a single sample, many of them possible to perform both manually and automatically [2]. Manual blood analysis can be performed through conventional light microscopy by an expert in the subject and has been the standard in modern medicine for decades. But this approach is also time consuming and resource intensive and the result can easily be affected by inter- and intraobserver variability [4, 5]. Automatic and digital analysis methods based on artificial intelligence have therefore become an important step in making the analysis both more accurate and efficient [5, 6].

1.1 Blood smear

A peripheral blood film (PBF) or smear (PBS) is a blood test that examines size, shape, the amount of blood cells and potential foreign bodies in the studied sample and therefore works as an important haematological diagnostic tool. Ideally, the PBS should contain three observable areas: the head of the smear beginning from the application point, a monolayer and a feathered edge [6, 7, 8, 9] , which can be seen in Figure 1.1. The head of the PBS is located at the beginning of the smear on the glass and is usually the thickest part of the test. This means that most cells overlap. The monolayer is a region of the sample which contains a continuous layer with one cell thickness, ideally with no overlap of cells, but often with cells lined up side-by-side. This area can usually be found just adjacent to the feathered edge [8, 10]. The feathered edge is the region on the outer part of the blood smear consisting of sparsely placed cells. These cells are prone to being damaged. Normally, the monolayer is the ideal region to perform RBC and WBC analysis as it enables easier visualisation for the examination, but it can lack information about denser abnormalities, such as erythrocyte aggregation [7, 8, 9, 11].

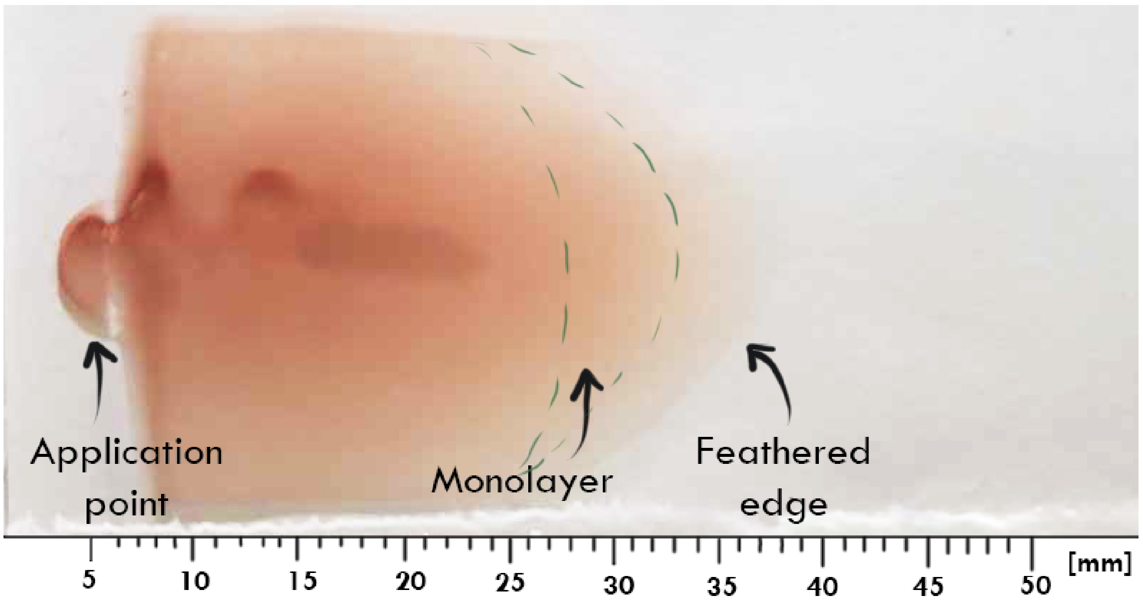


Figure 1.1: Example of a blood smear with its basic components.

1.2 Red blood cell morphology

A red blood cell, or an erythrocyte, is responsible for, among other things, transporting gases such as oxygen and nutrients to and from the tissue [12]. Since blood, and by extension erythrocytes, carry such an essential functionality in the body and is in close relation with tissues, it is very easily affected by external factors. Examining the blood of an organism can therefore relay information regarding potential threats, inflammation and diseases - giving blood tests a good supportive role in diagnostics [11].

The hematopoietic cells of vertebrates are derived from pluripotent stem cells in the bone marrow. These cells are nucleated and can be used to create any cell or tissue the body [13, 14]. In the human species as well as other mammals, erythropoiesis, the development into a mature red blood cell, includes losing organelles and nucleus during maturation; by the time RBCs reach the blood, they are therefore mostly enucleated [13, 15, 16].

1.2.1 Avian red blood cell morphology

In contrast to RBCs of mammals, erythrocytes of other vertebrates, including almost all fish, amphibian, reptile and bird species, mostly remain nucleated throughout their lifespan. Alongside keeping the nucleus, some species also possess organelles such as mitochondria and ribosomes in their mature erythrocytes. This has been demonstrated in fish species and occasionally in amphibians [15]. Avian species' RBCs also contain organelles, even if the investigation regarding functionality of the organelles in RBCs is lacking [15, 17]. This discovery has been somewhat contradicting common beliefs as birds and mammals both are endothermic and presumably face comparable selection pressures [18].

The typical avian erythrocyte is described as ovoid with a centrally placed ovoid nucleus. This appearance is mostly uniform across different avian species, although the size and proportion of blood cell and nucleus can vary [19, 20]. The nucleus consists of clumped chromatin, which increasingly condenses as the erythrocyte ages [21]. There is currently no clear evidence as to why birds keep their nucleus and organelles, and if these are fully functional compared to a normally nucleated cell. However, there have been some studies investigating this. Opposed to mammal erythrocytes, avian RBCs seem to have some level of DNA transcription activity within the nucleus [19]. This is further supported by findings of protein level changes in the cell depending on different environmental or physiological situations [22]. Studies also indicate that avian erythrocytes have direct immunological activity, including antigen presentation and interleukin-like production, but some antiviral functions can to some extent also be seen in enucleated RBCs. Another hypothesis for the existence of nucleated erythrocytes is the lack of Hb-storing in the spleen for birds [23] - suggesting that the functionality of the RBC can help synthesise and repair Hb de novo [17]. This process can for example be seen in fish [24]. Avian erythrocytes have also been shown to contain a higher amount of ATP compared to mammal RBCs, which can indicate that the nucleated RBC takes greater part of energy production in avian species. Avian mitochondria in erythrocytes have also been suggested to act as a reservoir of antioxidants to counteract higher production of reactive oxygen species (ROS), helping to regulate the redox status in the cell, as avian RBCs contain a lot of catalysts for ROS production including molecular O_2 bound to Hb causing autoxidation. Furthermore, considering that flight provides the need to regulate oxygen to adjust for different altitudes as well as demands a lot of energy, it's hypothesised that the nucleated RBCs allows for higher finetuning of Hb-O_2 binding affinity [17].

1.2.2 Extracting avian blood

Peripheral blood from birds is collected through venipuncture. For small birds this is most often collected through the jugular vein while larger birds' blood is collected from the ulnar or wing vein. Alongside proper addition of additives depending on the desired analysis, the blood is prepared on a blood film with the standard two-slide wedge technique also used in human haematology. The blood film is then stained after desired outcome, for avian medicine it is common to use Wright's stain [21].

1.3 Problem statement

Some of the currently most advanced automatic haematology analysers used on the market are today developed by CellaVision. As a company specialising in developing automated digital haematology systems, CellaVision today has a number of different systems in use with different sizes, features and complexities - including DC-1 and DM9600, see Figure 1.2.

Figure 1.2: CellaVision's automatic haematology systems DC-1(left) and DM9600(right) [24, 25].

Similar to traditional microscopy, the automatic haematology systems utilise monolayers when performing blood analysis. However, due to the differences in cell structures between various animal species previously mentioned, the current haematological analysis methods used in the existing systems are unreliable for avian blood samples when using microscopic lenses smaller than 100x, as it is more difficult to find the desired monolayer. Currently, the DC-1 is the only system that includes some form of analysis for avian medicine. However, by being the smallest analyser in the product range, this system has limitations; DC-1 can only analyse one sample at a time, thereby making the process of analysing a large number of samples very time consuming - considering that it approximately can scan up to 10 individual slides an hour. Additionally, it does this analysis at a higher magnification than desirable.

In order to make analysis more time efficient, and thereby more useful in laboratories with high demand, it is essential to use lenses with a lower optical power, preferably at a 10x magnification. Larger systems like DM9600 have the ability to analyse samples in batches and are thus more efficient. They can furthermore analyse and find monolayers within a range of different magnitudes, including 10x, for mammalian blood, but they lack this ability for avian blood analysis.

1.3.1 Aim of the thesis

The aim of this thesis is to demonstrate the application of a deep learning approach, utilising its strong pattern recognition capabilities, to avian blood smear images in order to find their monolayers. In further detail, the aim includes using the features of the DC-1 system to construct a neural network able to determine the existence of a monolayer in images with lower magnification (10x). This is important to be able to efficiently observe and analyse the contents of the blood and thereby perform the desired blood analysis. Furthermore, a secondary aim is to be able to identify and localise the white blood cells in avian samples by using object detection models. Ideally, this will be able to give an automated differentiation from the nucleated erythrocytes and by extension make it possible to perform a WBC analysis in the future.

Chapter 2

Literature review

There have been several attempts to make avian blood cell counting more efficient. Meechart et al. [26] developed a computer vision algorithm based on a threshold selection method while others tried to achieve the same through image cytometry by manually extracting features such as shape and intensity [27]. Although giving promising results, both studies mentioned that further investigations and research were needed to refine the techniques and give more satisfactory results.

Govind et al. [8] tried to relieve the process of manual examination by using whole slide images, as it is common practice to scan and archive high resolution images of a full blood smear. The authors developed a method of finding an optimal area for RBC morphology quantification from these whole slide scans for blood smears across several species including reptiles and one avian species. Similar methods are mentioned to attempt the quantification, such as excluding overlapping cells from the analysis, morphology-based techniques and automated analysis such as CellaVision's Advanced Red Blood Cell Software. However, these techniques are referred to as either decreasing sample size too much, being non-applicable to abnormal morphologies or being too economically and computationally expensive. This further meant that the study was not aimed specifically at finding the monolayer, but argued that inclusion of all erythrocytes within the defined optimal area would lead to higher accuracy compared to that in just the monolayer. The study was performed in a two-stage extraction system - by scanning smears with low resolution and then deciding on decision boundaries with the help of a quadratic determinant analysis classifier. The decided area was then analysed and refined with higher resolution. Subsequently, the cells were segmented based on the species' cell morphology and fed into a convolutional neural network, see Section 3.5, with a SGD optimiser, see Section 3.4.5, for classification. The study's proposed method demonstrated a more sensitive method for cell segmentation performance compared to that of existing literature. However, the lowest magnification used in the study was 40x, which allows for easier separation and hence easier classification of cells compared to 10x as is the aim for this project. Also worth mentioning is that the study did not contain a deep learning approach to finding the optimal area, as deep learning techniques were confined to classifying individually cropped out detected cells from the optimal area, which then were classified to a specific species.

A study conducted by Vogelbacher et al. [28] develops the idea of Govind et al. For the study, whole slide images at x40 and x100 magnification were captured of avian blood samples, which were then annotated by an expert. The whole slide image was then tiled and labelled positive if the cells were evenly distributed, contained no overlapping cells or had high quality. On the other hand, large free spaces, overlapping cells and low quality were considered to be negative. Further annotation included individual cells with segmentation boxes, giving each cell instance a specific mask and label. However, while increasing performance due to less overlap between segmentation masks, it is also mentioned to be very time-consuming. The cells were further labelled according to their cell type. The first neural network, based on EfficientNet, was developed to find an optimal region of blood cell counting, similar to that of the previously mentioned study, while the second neural network, CondInst instance segmentation model, is used for detecting and classifying blood cells by instance segmentation. The study provided great results in both AP, see Section 3.6.2, and interference runtimes, but again, only addressed the problem with a higher magnification.

The effect of magnification of images was addressed in a study conducted by Hoefling, Sing and Moulin [29]. The study compares different deep learning models in a histopathology setting, more specifically VGG-16, ResNet-50 and Inception-v3, all pretrained with ImageNet, at different magnification levels. It further discusses challenges of deep learning in computational pathology particular to whole slide imaging, including large image sizes, artefacts, the multiscale nature of the data as well as difficulty in obtaining annotations within the field. The whole-slide images for the study were generated using 40x magnification and manually manipulated to represent magnification ranging down to 1.25x. The studied objects were however of tissue-size, which means the object size could vary but generally contained larger areas compared to cells. Out of the proposed models, Inception-v3 and ResNet-50 outperformed VGG-16, with Inception-v3 having superior performance. In general, the tissue prediction was increasingly reliable with lower magnitude, which naturally is a consequence of having more structures of the tissue present in the image and hence becomes more distinguishable.

Kittichai et al. takes a deep learning approach in their study [30], deploying a comparison between different convolutional neural networks with the purpose of automatically classifying an avian malaria parasite in blood samples and by extension decrease the inter- and intra examiner variability. Four different CNNs were considered: Darknet, Darknet19, Darknet 19-448 and Densenet201, trained with a dataset of 12761 single-cell images. The study was performed in two stages by combining the object detection model YOLOv3 with the classification models. A hybrid solution of two CNNs was argued to demonstrate increased prediction accuracy. Further, it is also said that a hybrid platform in the object detection model including the YOLO model with a different detector improves the average precision in the proposed detector. The object detection model YOLOv3 was used to detect individual RBCs in microscopic images with 1000x oil immersion magnification. The cropped, single RBCs were then used as inputs for the following classification model. In order to avoid overfitting in the models, augmentation, see Section 3.2.2, such as rotations, brightness, contrast, blurriness and Gaussian noise was applied. For this specific dataset, Darknet yielded the highest accuracy and precision of the proposed models.

Chapter 3
Theory

3.1 Artificial intelligence

The idea of making technology think originated as early as 1950 [31], and the field quickly experienced substantial development into what today is known as artificial intelligence (AI). AI is based on the concept of understanding human intelligence and, ideally, beyond, as well as building entire intelligent entities. Primarily, this is focused on computational ability to solve problems and accomplish set tasks [32, 33]. AI is a general term containing several subfields such as machine learning (ML), deep learning, natural language processing and computer vision.

AI is also becoming increasingly more important to use in medicine and healthcare, even though the field has higher regulatory conditions to fulfil. AI can be useful in informatics approaches such as electronic health records and omics, but also with medical diagnosis, medical statistics and robotic [34]. AI is particularly useful in acting as a complement in diagnosing patients' health statuses, leading to a more secure and efficient process as well as advancing the field further. In pathology, AI often uses pattern recognition methods to incorporate clinical, radiologic and genomic data. Diagnostic pathology mostly utilises microscopic morphology, but this technique has a large error rate in interobserver variability with manual use. Introducing algorithms and AI to diagnostic pathology, especially in terms of segmentation, detection, classification and quantification, to get more consistent and accurate results has hence been proven effective [35].

3.2 Machine learning

With the increasing amount of data circulating in our society, machine learning (ML) has become more and more relevant across multiple fields and industries, such as healthcare, education, manufacturing and marketing; changing the way technology works over just a couple of decades. ML is a subfield of AI that uses algorithms to identify patterns, model data and perform tasks such as classifications and decisions. The goal with ML is to build systems that automatically can improve its own performance through experience [36]. This is based on the idea of showing the models real life data with certain inputs and outputs, which is a far more efficient approach compared to anticipating all possible outcomes manually. Furthermore, it helps with managing large amounts of data, which can be difficult to tackle manually. Typically, ML tasks involve extracting domain-specific features from raw input. This is usually followed by statistical modelling and learning different kinds of models depending on the task at hand. Some common models are tree-based decision models, support vector machines (SVM) and linear regression models [35].

3.2.1 Different types of learning

There are several ways to learn and train a system. The most frequently employed method is supervised learning, especially for prediction [37, 38]. Supervised learning is characterised by utilising annotated training data to instruct the model how to associate the labels with the input data. Thus, the idea is that unseen data will behave similarly to the distribution in the labelled training data and therefore make accurate predictions on new unseen data [39].

Another approach to learning is through unsupervised machine learning methods. Here, the model is trained on non-labeled data to find patterns and relationships. This is for example performed through clustering or dimensionality reduction. The method is especially suitable for description tasks as they lack a variable that can supervise accuracy of for example correct predictions. It can be beneficial to identify underlying or unobserved structures that are difficult to find manually [38].

The two different techniques can also be combined in what is called semi-supervised learning. Semi-supervised learning (SSL) is especially useful in areas where annotation can be difficult, such as medicine and agriculture. It is further practical if the labelling processes are time-consuming, as they in some cases can take too long in order to be beneficial and are often prone to human error. The community has hence tried to develop alternatives to supervised learning, where partial sets of data are annotated while the rest is not. A technique commonly used within the realms of SSL is pseudo-labelling [40]. In pseudo-labeling, a model is firstly trained with the annotated data. The unannotated data is then iteratively classified after predictions of the trained model. When both annotated data and unlabelled data have been divided into categories, the model is further trained while including the new samples [41].

3.2.2 Training ML models

Training ML models can be a challenging task, both in terms of time spent fine-tuning hyperparameters, which form the model configuration settings, to reach desirable performance and generalisation, but it is also computationally expensive due to the number of parameters needing optimisation. Many ML models inherently support the ability of incremental learning, which is a methodology to learn and enhance a designed model by remembering previously learned knowledge [42]. The aim is to learn enough about the data that the model is being trained on, while also keeping its generalisation and thus be able to adapt to new, previously unseen data with adequate performance relative to the training performance. In order to track generalisation, datasets are usually divided into three parts: the training set the network is being trained on, a validation set used to fine-tune hyperparameters, and a test set for a final evaluation of the model's performance [43]. The hyperparameters can be tuned in two different strategies, either using grid search or random search. With grid search, a number of possible values of the hyperparameters will be chosen and then models with different combinations of these will be trained to find the optimal configuration of the given choices. Random search, on the other hand, is, as implied, randomly training models with randomly chosen combinations of hyperparameters. This gives the advantage of covering a larger amount of hyperparameters compared to grid search, but is more difficult to reproduce [44].

When a model achieves good performance on the training set but not on the validation or test set, the model is overfitted. One reason for this could be that the dataset doesn't contain enough samples. A too small set of samples means that the possibility of the training set containing samples similar to those in the test set decreases, thereby increasing the generalisation loss, see Figure 3.1.

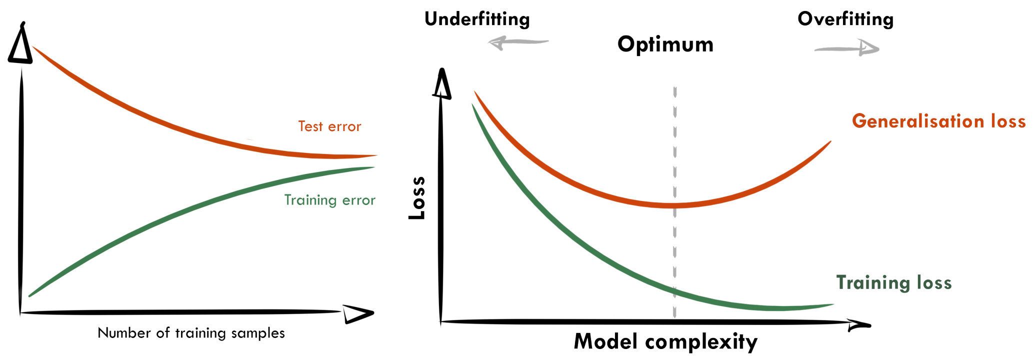


Figure 3.1: The effect on training and generalisation loss by number of samples and model complexity.

Another possible reason is the model becoming too complex. A higher number of parameters are usually able to fit both the true regularities as well as the accidental ones. This means that if the model has too few parameters, it fails to find the true patterns between the samples, thereby underfitting the model. However, if given too much complexity, it will completely learn the training data and fail to generalise, thereby increasing the generalisation loss. It is therefore of importance to find the optimum between under- and overfitting in order to receive better performance [45]. These concepts are illustratively explained in Figure 3.1.

Regularisation

Another common approach to reducing overfitting is using regularisation. Regularisation helps to constrain and control model complexity by adding penalty terms to the loss function. This in turn will regulate the estimated gradients and influence parameter updates, hence leading to a model with better ability to generalise. The most commonly used regularisation techniques are Least Absolute Shrinkage and Selection Operator (LASSO), also called L1 regularisation, Ridge or L2 regularisation, and Elastic Net, also called L1/L2 regularisation. L1 regularisation adds the absolute value of weight magnitude as a penalty term to the loss function while ridge regularisation instead adds the squared magnitude of the coefficient. This means the L1 can set coefficients to exactly 0, making the model ignore features that are of no use to the model learning. This encourages sparsity as it introduces feature selection, hence working well in models with a high number of features. Penalising with the squared magnitude will instead reduce the impact that irrelevant features have on the model without completely removing them, which will stabilise the model while keeping all information [46, 47]. Elastic Net regularisation (L1/L2 regularisation) is a combination of the two regularisation techniques, effectively combining the strengths of both. This has been shown to perform better compared to other linear regression techniques, thereby presenting as robust while also helping with feature selection and parameter shrinkage [47].

Data augmentation

Although machine learning can solve complex problems today, it also comes with the requirement of a large amount of data in order to perform adequately. This is a big bottleneck in fields such as medicine, especially while handling visual data [40]. In this situation, data augmentation can prove itself helpful by artificially introducing random distortions to existing samples. This technique will hence increase the training set with small variations such as geometric and colour space transformations, as well as for example combining images and random erasings. Furthermore, data augmentation will also help models from overfitting, reduce bias, and possibly increase the general model performance. Data augmentation is usually included in the model pipeline [48]. For cell segmentation, data augmentation with random elastic transformations seem to improve performance, especially when having few training samples, as it gives the network a possibility to get used to deformations in the data [49].

3.2.3 Evaluating performance of ML models

While it is custom to track ML models' learning through training and validation accuracy as well as loss over time, there is also a need to perform a proper evaluation to understand the performance and capability of the model. Evaluation of machine learning systems is essential in all applications, especially due to lack of explainability in ML. Quantifying the quality of performance helps the understanding of the system's solution [50]. Both qualitative and quantitative evaluation can be used to determine performance. Qualitative evaluation means for example asking the users if the result is satisfactory while quantitative evaluation focuses on statistical and mechanical methods to validate the performance [51].

Confusion matrix

When it comes to classification tasks, the predicted values can be divided into true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), where the true values signify a correct prediction and vice versa for the negative examples. In an ideal model, FP and FN should be zero, as these indicators signify the wrong classification. Based on situational applications of models, the importance of a non-zero value can vary. In medicine, FN cases can for example indicate that a patient with an illness has been diagnosed as healthy which in the worst case scenario can lead to a fatal outcome. These categories of predicted values are therefore used to calculate metrics with the purpose of measuring model performance with varied significance to outcome [52, 53].

The cross classification these categories explain can be represented by a confusion matrix, also called error matrix, and is commonly used in the area of ML, both for binary and multiclass classification. The layout allows for easier visualisation of performance measurement [54]. Each column illustrates instances of predicted values while each row shows the true instances, as seen in Figure 3.2.

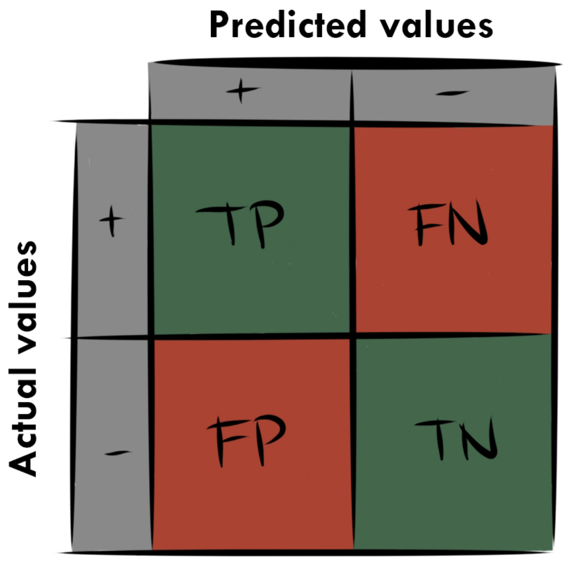


Figure 3.2: Layout of a binary confusion matrix.

Evaluation metrics

Accuracy describes the proportion of true results, that is the number of correct assessments across all samples. This can be calculated as

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{3.1}$$

Accuracy might be the most intuitive metric for the model performance, but most often this can be misleading and not show the true quality of the model, especially in very imbalanced datasets [55]. For a more fair assessment, recall and precision can be used.

Recall, or sensitivity, signifies how well the model can correctly identify the true positives. Recall can be described as

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

Precision consists of the ratio of identified positive class cases of all positive predictions and will hence determine how many of the positive identifications actually were correct. Mathematically, it can be expressed as

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

Precision is most often a trade off with recall, as increasing recall might detect all cases but can also over-classify into that class, thus ending up with potentially affecting unnecessary cases. On the other hand, if the aim is high precision, there is a higher risk of missing cases, which can be harmful in for example diagnostic medicine. When both precision and recall are of importance, the F1 score can combine them using a harmonic mean. This will mean that maximising the F1 score will maximise both precision and recall in relation to each other. F1 score can be explained as

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.4}$$

Which metrics are the most significant depends on the situation of which the model is applied on [56, 57].

Receiver operating characteristic curve

A receiver operating characteristic (ROC) curve is a different way of displaying model performance, where the true positive rate (TPR), also known as recall, is plotted against the false positive rate (FPR), which is the proportion of incorrectly classified negative instances. An illustrative example is shown in Figure 3.3. TPR will measure how well the model correctly identifies the true instances while FPR measure the rate at which the model incorrectly predicts negative cases as positive. Ideally the curve aims to the top left corner, where the FPR is minimised while having a high recall value. This means that the model correctly classifies all true instances while making no false predictions. Ergo, all positive cases will be predicted as positive and all negative cases as negative and therefore presents as the optimal balance between being able to detect all the true instances while avoiding classifying false instances as true. This will maximise the model performance across all thresholds and will by extension maximise the area under the curve (AUC). ROC AUC score quantifies the performance illustrated in the ROC, where a higher value indicates better performance and better ability in distinguishing between classes. Usually, a ROC AUC above 80% is considered good, and over 90% excellent. A value of 50% indicates a random classifier.

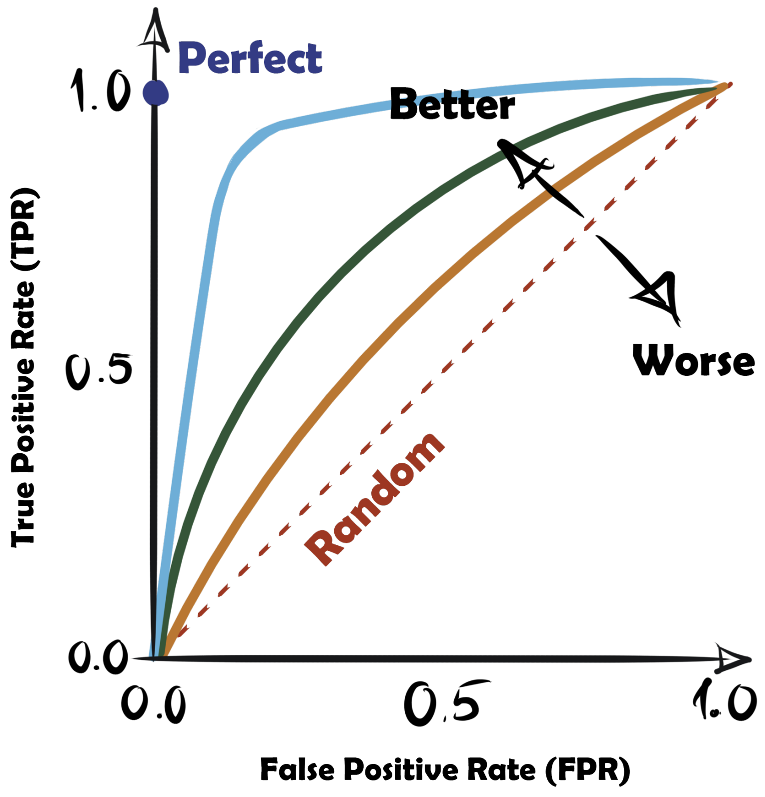


Figure 3.3: An example of a ROC curve.

3.3 Deep learning

Deep learning (DL) is a subfield in ML and is today considered to be a core technology [58]. DL uses networks composed of multiple nested layers to solve tasks by extracting features from inputs. The features are extracted according to hierarchy, where simpler features such as lines and curves form lower levels while more advanced structures form higher levels. The hierarchical nature gives the system the possibility to learn more complex relationships in the data as it builds them out of several simpler ones [59]. This gives the opportunity to perform learning without the feature modelling step usually necessary in common ML models. Additionally, it also differs in its efficiency when dealing with larger amounts of data. DL firstly became popular after its success in visual object recognition [60] and has since been applied in numerous applications such as healthcare, cybersecurity and text analytics [58]. However, due to it being more complex compared to traditional ML models, explainability of DL models is extremely challenging as their interpretability may decrease [58, 60]. Furthermore, it can also be very difficult to optimise a model to ideally fit real-world data, due to its varying nature [58].

3.4 Artificial neural networks

3.4.1 The single neuron

Artificial neural networks (ANNs) are based on the function and structure of the biological natural circuit of neurons that exists in the human brain. Biological neurons are highly interconnected and transmit signals using electric excitation or inhibition in order to relay information. The neuron can be compared to a step function, as it receives and summarises input information and, if the threshold limit is reached, generates a full response independently of the magnitude of the input [61]. Similarly, in ANNs, these electrical signals are represented by positive and negative weight values in the artificial neurons, and thus, neurons form the basic unit of computation in ANNs. As seen in Figure 3.4, each neuron receives input, X_i, from one or several connecting inputs, which for example can contain features of a dataset or images. The inputs are each associated with specific weights, w_i, which are based on the inputs' relative significance to the other sending nodes. Apart from inputs and connected weights, the neuron further needs bias, b, summation function and activation function, f, in order to form an output, Y.

The activation function is necessary to give the model non-linearity and can be compared to the biological function that only signals exceeding a certain threshold will be transmitted. There are several different activation functions, but some of the most common are sigmoid, tanh, Rectified linear Unit and Softmax [62]. The bias is a constant and works to shift, adjusting its position, the activation function, and by extension the product of features and weights. The summation simply sums together the resulting product with added bias and forms the input to the activation function [61, 63].

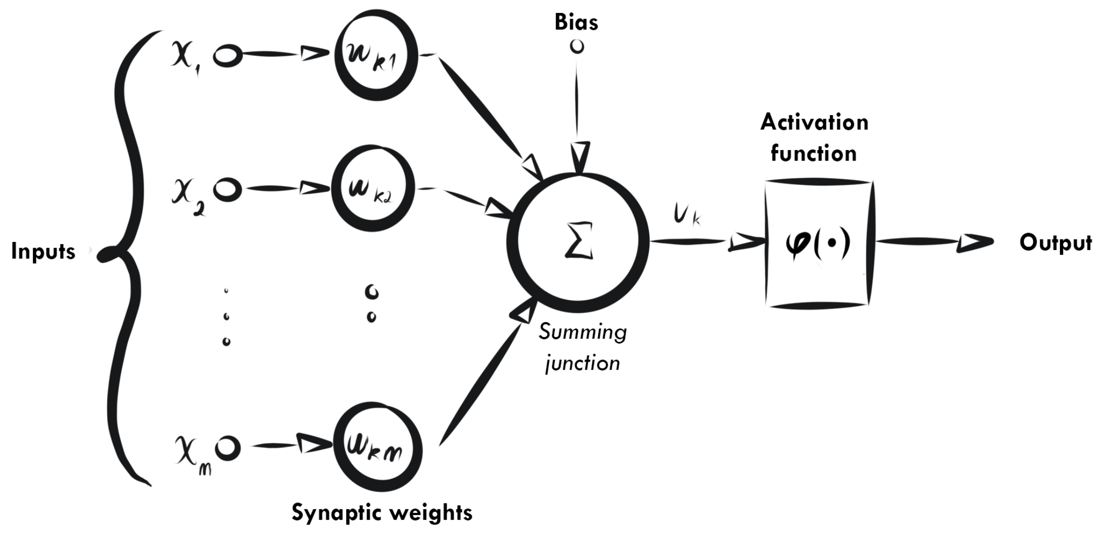


Figure 3.4: Example of a neuron with inputs x, weights w, bias and applied activation function.

3.4.2 Neural networks

Just as ensembles of neurons in the brain can form functional physiological units with specific functions and qualities - the nodes can also be structured in combinations to form computational models that can attempt solving different kinds of problems [61]. Typically, the amount of neurons in a network can range from very few to millions of nodes, arranged in layers. The single computational layer is called a perceptron [63]. By combining these single units interconnectedly and adding non-linearity, the ANN can effectively model complex relationships between input and output data to simulate real-world problems and solutions [63, 62]. The neurons can then efficiently form multi-layer networks where the initial input and output layers are separated by intermediary stages called hidden layers [63, 64], see Figure 3.5. The input is transformed within these hidden layers, which perform their computation through their assigned neurons. The output from each neuron and layer will then be received by the next layer of neurons, where further computation will take place. A network's architecture is determined through the number of hidden layers, often referred to as a network's depth, and the neurons in each of those layers, called the network's width. Generally, the deeper the network, the more complex relationships it can capture and similarly, the wider the network, the more information about the data can be recorded. However, this comes with drawbacks like requiring a larger amount of data and more computational power in order to learn. Furthermore, the more information in the data it will capture, the higher the risk of overfitting to the training data [65].

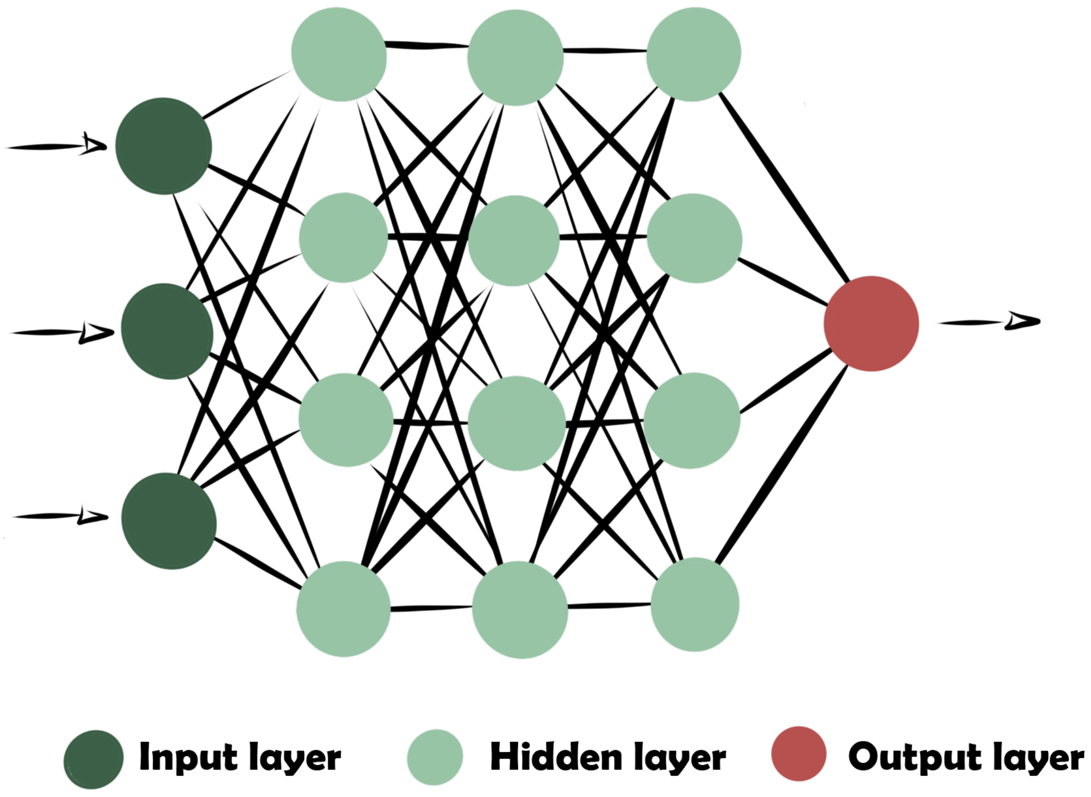


Figure 3.5: Illustrative example of a network.

3.4.3 Forward propagation and backpropagation

To pass information into a network, the most straightforward iteration of a network is called a feedforward network [63]. A feedforward network utilises forward propagation, which means that the network flows the data forward through the nodes. This is performed by the nodes receiving inputs, the weighted sum of inputs in a previous layer (or the first input) and then compute the output based on the activation functions while incorporating weights and biases in the receiving layers [66]. Most neural networks combine forward propagation with backpropagation, both being performed iteratively during the training to increase accuracy of the predictions. Minimising the function loss in a set task is the main goal of the training process of a neural network, as a lower error rate means that the network is learning the data. Backpropagation is a crucial part of the training process for most neural networks, and means backward propagation of errors. The purpose of this is to adjust and finetune the weights of the nodes in the network based on the error rate, also known as loss, computed for each output unit [67]. Despite not being based on a biological function, ANNs trained with backpropagation have been shown to perform better than existing alternatives [68].

3.4.4 Loss functions

How the weights and biases of nodes are adjusted during the backward propagation heavily depends on the loss function. The loss function will quantify the error between the predicted output and the given target label, ergo telling the network how far away from the correct answer it currently is and thereby tracking how far from an accurate prediction the network is. There are currently numerous loss functions available, each with different methods of calculating losses and penalising large errors. This will hence mean that different loss functions affect the model's performance differently [69].

Cross-entropy loss

For classification tasks, binary and categorical cross-entropy loss are commonly used, also called negative log-likelihood, softmax loss or log loss. Cross-entropy will measure how much two distributions differ from each other, in this case the model distribution and the true distribution. Minimising this loss hence means that the two distributions will be closer to each other and thus give a higher amount of correct predictions; a perfect model would have zero in log loss [69]. The loss can be calculated by quantifying the degree of entropy, uncertainty, in the predicted value, and then summed across all variables. For multiclass classification, the loss can be explained as

$$Loss_{Cross-Entropy} = -\sum_{c=1}^{M} log(p_t) \tag{3.5}$$

where M is the number of classes and p_t is defined as

$$p_t = \begin{cases} p & \text{if } y = 1, \\ 1 - p & \text{if } y = 0. \end{cases}$$

with y being the binary indicator if the class label is correct. One of the advantages of cross-entropy loss is that the shape of the function simplifies finding the minimum when using gradients to optimise the network [70]. However, cross-entropy loss works less well when the data contains class imbalance, as the majority class features will dominate the loss function, therefore pushing the adjustment of weights closer to a more confident majority prediction. The minority classes can thereby easily become neglected [71].

Focal loss

Focal loss was originally developed to address foreground-background class imbalance in object detection tasks. Focal loss manipulates standard cross-entropy loss by assigning less importance to examples that are well classified. This leads to predictions of difficult samples improving with training compared to higher confidence in easily classified samples. Reducing the influence of well classified examples is done through down weighting, which adds a modulating factor to the loss. Focal loss can be explained as

$$Loss_{Focal} = -\sum_{i=1}^{M}(i - p_t)^{\gamma} log(p_t) \tag{3.6}$$

where $(1 - p_t)^{\gamma}$ is the modulating factor with γ as focusing factor. When $\gamma = 0$, it equals to cross-entropy loss. Higher γ leads to rescaling of the modulating factor and an increase will lead to more down-weighting of easy examples. Earlier studies have shown that $\gamma = 2$ usually leads to the best performance [72].

α-balanced focal loss

Cross-entropy loss can also address class imbalance by adding a weighting factor, α, to each class - thereby balancing the loss. It is suggested that α could be the inverse class frequency. Balanced cross-entropy can be explained as

$$Loss_{BalancedCross-Entropy} = -\sum_{i=1}^{M} \alpha_i log(p_t) \tag{3.7}$$

However, balanced cross-entropy does not address distinguishing between learning easy and hard examples. Therefore, balanced cross-entropy can be combined with focal loss to create α-balanced focal loss. This loss function utilise both the weighting factor α as well as the focusing parameter γ, thus decreasing the previously mentioned issues[72]. α-balanced focal loss can be described as

$$Loss_{BalancedFocal} = -\sum_{i=1}^{M} \alpha_i (i - p_t)^{\gamma} log(p_t) \tag{3.8}$$

3.4.5 Optimisation

An efficient modern optimisation technique is with the help of gradients, with the goal to adjust the model parameters during backpropagation in order to minimise the set loss function. When minimising a function, its derivative can be useful as it will specify how a change in input will affect the output. The gradient will not only reveal whether the direction should change or not, but also the relative importance of weights, as some connections are more important to the network than others for the specific inputs. An optimisation algorithm called Gradient Descent (GD) utilises this by calculating the gradient at the current position followed by taking a step in the direction of the steepest descent. When repeating this, this will converge the output closer to the local minimum [59].

The learning rate can scale the gradient and thereby control the size of the next step. This is performed as

$$W_{new} = W_{old} - \alpha \frac{\partial J}{\partial W}, \tag{3.9}$$

for the weight update where W represents the weights, alpha the learning rate and J the averaged loss of the entire dataset. Similarly, the bias update can be explained, with b as bias, as

$$b_{new} = b_{old} - \alpha \frac{\partial J}{\partial b}. \tag{3.10}$$

The learning rate is one of the most important settings when configuring a neural network but is also very difficult to balance. If the learning rate is too low, the network might not be able to reach the minimum fast enough. On the other hand, a too large value can push the network to set suboptimal weights too fast and create unstable and diverging training [73], see Figure 3.6.

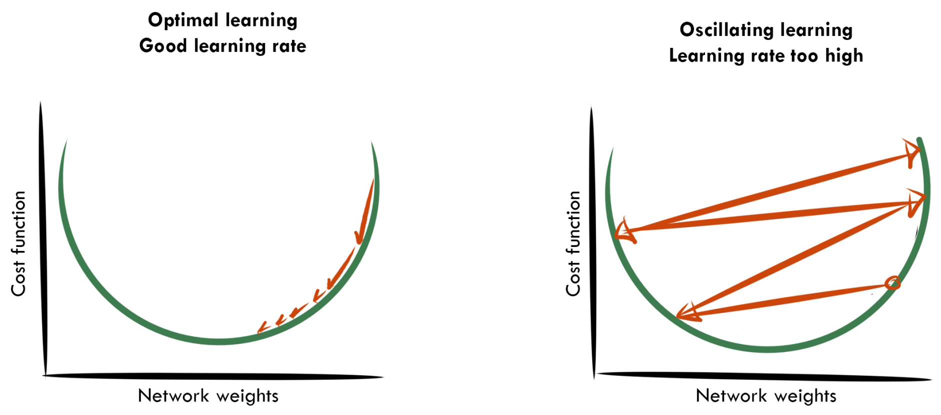


Figure 3.6: Effects of an optimal and a too high learning rate.

The learning rate can be configured with different approaches to find a better fit for the specific network. Apart from having constant learning rates, techniques such as decaying learning rates, which starts with a larger learning rate and then decays it for example exponentially multiple times [74], and cyclical learning rates, which varies the learning rate between a minimum bound and maximum bound throughout the training, have been proven efficient for some applications [75]. There are several optimisers besides GD that utilise the gradients and learning rates for a better optimisation. The Stochastic Gradient Descent (SGD) optimiser is widely used in many approaches and is a variation to the traditional GD algorithm. In contrast to GD, which uses the gradient calculated over the entire dataset, SGD instead computes the gradient using a single data point. Introducing randomness in this way can make the training more stochastic but also make it run faster. The advantage of SGD compared to GD is that it is more memory efficient since it only uses subsets of the data at each calculation [76]. However, due to its stochastic nature, SGD converges less regularly since it keeps oscillating towards convergence. This effect can be decreased with the help of momentum. The purpose of momentum is to accelerate the convergence by smoothing out oscillations in the gradient. The momentum builds up when the gradient consistently stays in the same direction for several iterations, and therefore includes not only the current gradient but also the historical gradients when updating the parameters as a weighted average, thus increasing the speed to convergence [77].

Although SGD has proven to be very efficient for a long time, and is still in use, it has some limitations due to fixed parameters. Therefore, techniques using adaptive learning rate were developed.

Root Mean Square Propagation (RMSProp) is one algorithm that uses adaptive learning rate. This was originally developed to address the vanishing gradient problem. This problem arises when the network is very deep, and as the gradients are back propagated the overall gradient can start to approach zero since the partial gradients can be very small. This results in very slow convergence and difficulty learning hierarchical data. Additionally, there can also be a problem with exploding gradients, which occurs when the gradients become very large in deep networks. This causes the network to diverge and become unstable. RMSProp normalises the gradients, which means that the step for large gradients will be decreased and vice versa for smaller gradients. The normalisation is done using a moving average of squared gradients, where the moving average is calculated as exponentially decaying. This means that the influence of past gradients depends mostly on the closest gradient and less on the earlier. By doing this, it is not necessary to accumulate all the historical gradients, thus decreasing the memory requirement. The gradients are then divided by the square root of these moving averages, and parameters updated accordingly [78]. RMSProp itself has not been officially published, but is still included in most deep learning frameworks and very well known.

Another commonly used optimiser is based on the Adaptive Moment Estimation (ADAM) algorithm. ADAM is very versatile and gives good optimization for a range of different tasks and is therefore one of the most commonly used optimisers [78]. ADAM combines RMSProp and momentum, making it more memory-efficient. ADAM adapts the learning rates based on the first and second momentum, the gradient and the squared gradient, but also smooths the gradient with the help of momentum [79].

The different optimisers perform differently depending on the input data and the tasks, and the most fitting one is usually left for experimentation [59].

3.5 Convolutional neural networks

Convolutional neural networks (CNN) are widely used with image recognition tasks, as it is particularly successful with predictions made in grid-like topological spaces. Hence, it has today become one of the most important types of networks, especially in fields like computer vision and natural language processing, and has enabled achievements such as face recognition and intelligent medical treatments. A CNN is based on the idea of using the convolution operation in one or several layers, instead of using fully connected layers like previously [80, 81, 82]. This makes it possible to automatically extract features from data into feature maps. The convolutional operation is an integral expressing how one function is being modified while shifting another function on top of it; blending the two functions together over a space [83]. For discrete input signals, the operation can be mathematically explained as

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m], \tag{3.11}$$

where f and g are two discrete or continuous functions and m and n is the position of the input and the output signal respectively. The asterisk represents the convolution. In the context of neural networks, a convolutional filter or kernel, which is a matrix of weights, slides across an image with a certain stride value until it traverses the entire image and performs convolution between the kernel and the image pixels [84], see Figure 3.7.

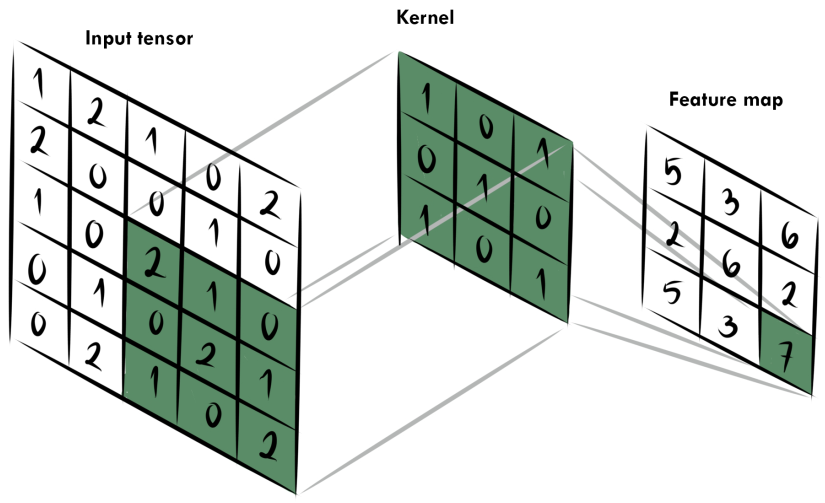


Figure 3.7: Example of a convolutional operation on an input matrix with a kernel size 3x3.

The output is then stored in an output feature map. When dealing with multiple channels, the kernel has the same depth as the input image, which output maps are combined into a one-depth channel output. Depending on the size of the filter as well as its contents, this can affect how the features are extracted and consequently affect the prediction. The filter size is often left for network optimisation while the weights are learnt by the model. To increase performance, the convolutional operation can be used in several filters and layers, thereby capturing multiple high-level features [85]. Hence, convolutional layers effectively combine local information by a smaller number of pixels in order to extract key information that hierarchically distinguishes images, objects, structural patterns, lines etc. from each other [84]. Due to the nature of CNNs, it is more computationally effective compared to fully connected layers. This can be explained by the neurons not having connections with all the neurons in the previous layers, hence reducing the number of parameters. The parameters are further reduced with the help of sharing weights in groups of connected neurons [86].

In a CNN, a convolutional layer is most often followed by a pooling layer. A pooling layer is responsible for reducing the dimensionality of the feature map in preparation for the next convolutional layer, with the aim of reducing the necessary computational expense and decreasing the risk of overfitting [87]. The resulting input to the next convolutional layer is performed on the summarised features after pooling, which means that the model becomes more robust to variations in position. Similar to convolutional layers, a pooling layer also slides a filter across each channel of an image. The two most commonly used types of pooling layers are max pooling and average pooling, see Figure 3.8. In max pooling, the maximum value across the current field of interest is chosen as output. This means that the resulting output contains the most prominent features of the previous feature map while discarding less relevant information. Average pooling instead computes the average of all the elements covered by the filter area. Consequently, this will present the average of the features, thus containing more information than max pooling [88].

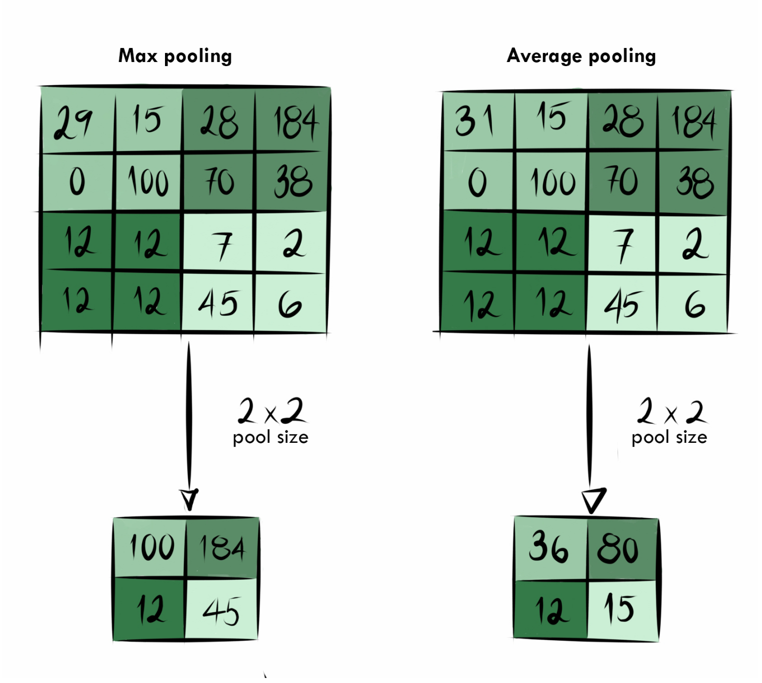


Figure 3.8: Illustrative example of the max and average pooling.

Apart from the two previously mentioned layers, a CNN also contains a fully connected (FC) layer. A FC layer, or a dense layer, is usually present at the end of a CNN and works as a classification layer. A FC layer is used to connect all the neurons between the two different layers with each other. It utilises the flattened output and hence the extracted features from all neurons in the previous layer, usually from a pooling layer, in order to generate a final classification or regression to the output layer. This follows the principle used with other feed-forward ANNs [89]. An example of a simple CNN can be seen in Figure 3.9.

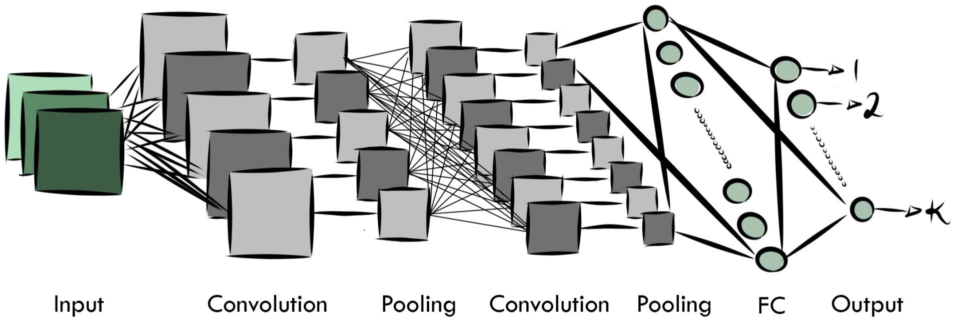


Figure 3.9: Illustrative example of a simple CNN architecture.

Similar to other ANNs, a CNN also contains activation layers in order to add nonlinearity. Additionally, it is also common practice to add batch normalisation and dropout layers. Dropout is added in order to reduce overfitting, as this can be prominent when all the features are connected. Here, a number of neurons, determined by the dropout rate, are dropped, which efficiently reduces the size of the model. In order to prevent vital information loss, the neurons that are disabled are chosen at random during each training iteration [89]. Generally with all deep neural networks in supervised learning, they need to train on a large amount of labelled data to be able to associate features with specific labels. Training is further complicated by the parameters of the previous layers changing during training, thereby changing the distributions of the current layer's inputs, and requires careful finetuning of parameters. Batch normalisation was added as a way to make this process easier as it aims to standardise activations of the input variables, thereby making the assumptions about the previous layers' distributions differ less. As a result, it will speed up the training process and stabilise the network [90].

3.6 Object detection

Computer vision (CV) is another type of AI that focuses on making technological systems interpret the real world visually. The human visual system can easily identify multiple different objects with an incredible speed, and for the sake of automation, among other things, it is desirable to make systems try to mimic the human way of seeing and understanding their environment. It has become a wide-spread application in multiple fields, especially industries based on automation and medical domains[91]. A popular CV technique is object detection, which aims to locate instances of specific classes in images or videos - answering firstly, if there is an object in the image and secondly, where in the image these specific objects are located. It is currently mostly performed with the help of deep learning and CNN, due to their high performance with complex problems and visual data [92, 93]. Object detection is well studied within areas such as detection of faces, pedestrians or vehicles, making it useful for applications such as video surveillance [92].

3.6.1 Principles of object detection

The problem of locating objects has earlier been solved with, among other things, machine learning strategies, for example analysing the colour histogram and clustering of pixels. These were then used as features into regression models. With a deep learning approach, the general idea behind object detection is creating so-called bounding boxes that surround objects of interest. The bounding boxes' x- and y-coordinates represent where these objects are located in the image. The bounding boxes are then used as input along with the corresponding images to the deep learning model of choice, which in turn will be trained and finally result in refined bounding boxes as outcome [93]. When classifying a single class, it can be considered an object localisation problem, or a single class object detection, but it can also be extended into a multi-class problem, where it is necessary to both localise and classify different objects. This will transform the problem into an object detection problem; object detection is a combination of both image classification and object localisation [91].

In order to provide a relatively fast solution to a given task with adequate accuracy, most networks use anchor boxes for detection. Anchor boxes are predefined bounding boxes with specific measurements in height and width, see Figure 3.10.

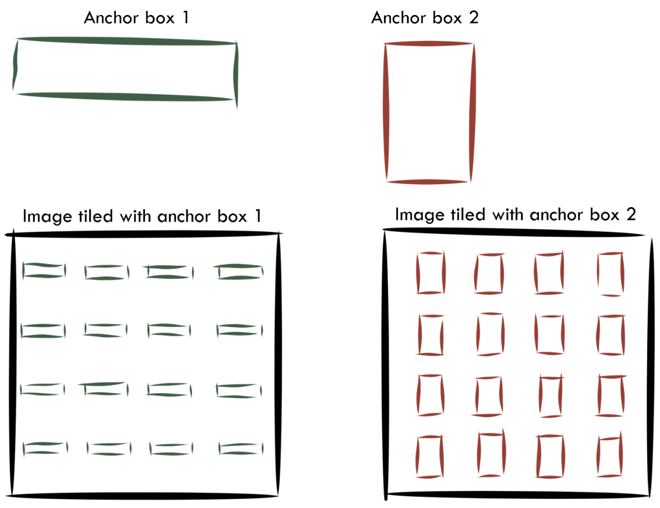


Figure 3.10: Anchor boxes tiled over image while being mapped by CNN output.

The anchor boxes are tiled over the image while the network calculates predictions based on content in that specific anchor box - such as background, probability of intersection over the ground truth bounding boxes and their offset, the adjustments of coordinates to align the anchor boxes more accurately, see Figure 3.11. With further training, the anchor boxes get refined to better match the bounding boxes with the help of the predictions [94]. The use of anchor boxes allows for real-time detection, as it can detect multiple objects across the whole image at once with the help of the spatial awareness of CNNs, as opposed to for example detectors with sliding windows [95]. After optimising the predictions for anchor boxes, the anchor boxes predicted to belong to the background are removed while the remaining boxes are filtered depending on their prediction's confidence level. The technique

Non-maximum suppression (NMS) is then used to eliminate potential duplicates. This is performed by choosing the proposed boxes with the highest confidence score and calculating the intersection over union (IoU), displaying a ratio of overlap, against every other proposed box. If the IoU is higher than the set threshold, it will be used as output. This process is repeated until all proposed predictions are analysed, leaving the final output [94].

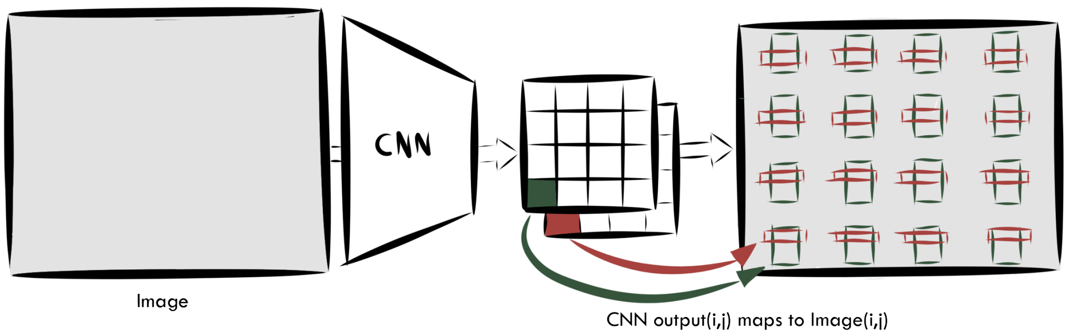


Figure 3.11: Two different anchor boxes and their tiling over an image.

Object detection was at first proposed with a two stage approach, where the detection process is divided into firstly recognising regions of interest, usually with the help of the mentioned anchor boxes, and secondly classifying and refining the regions. This approach includes famous models such as R-CNN, Faster R-CNN and Cascade R-CNN [96]. As opposed to these two-staged detectors, one-stage detectors are becoming increasingly more common. They include a detector using a single pass through the network without using previous computation. Popular one stage detectors using anchors are Single Shot MultiBox Detector (SSD), RetinaNet and early versions of You Only Look Once (YOLO) [97]. While anchor boxes can be helpful in guiding the model into the size range of interest of the objects, more state-of-the-art models provide an anchor-free detection, allowing for more flexibility in detecting objects with various sizes and ratios as well as decrease the complexity of the model. This will in turn allow for a faster computation. This can be seen in models such as Fully Convolutional One-Stage Detector (FCOS) [96]. This is performed by directly predicting bounding boxes from extracted features, pixel-wise instead of anchor-wise. The models can be divided into three parts: the backbone, the neck and the head. The backbone is the main feature extractor, most often consisting of some sort of CNN architecture, providing a hierarchical representation of the input. The choice of backbone can therefore significantly influence the performance of the model. The neck will follow by including more contextual information [98]. Many anchor-free detection models, as well as some models using anchors, rely heavily on feature pyramids in their neck part. Feature pyramids are feature extractors that stack the same image several times on top of each other in different scalings, using a bottom-up and top-down pathway, which can be seen in Figure 3.12. The bottom-up is commonly applied in CNNs' feature extraction, where the spatial resolution decreases on higher levels, increasing the abstraction. This means that features with semantic value can be extracted, but they may be distorted due to manipulations of the image. Thus, layers are reconstructed to give lateral connections to the semantic layers and thereby increasing the accuracy of the object locations [99]. Lastly, the head is responsible for the model's predictions, including bounding boxes and confidence levels [98]. However, despite one-stage detectors normally being faster compared to two-stage detectors, it usually comes with the cost of worse performance [97]. In order to improve performance of a detector, similar strategies as previously mentioned can be deployed, including hyperparameter finetuning, data augmentation and optimisation algorithms.

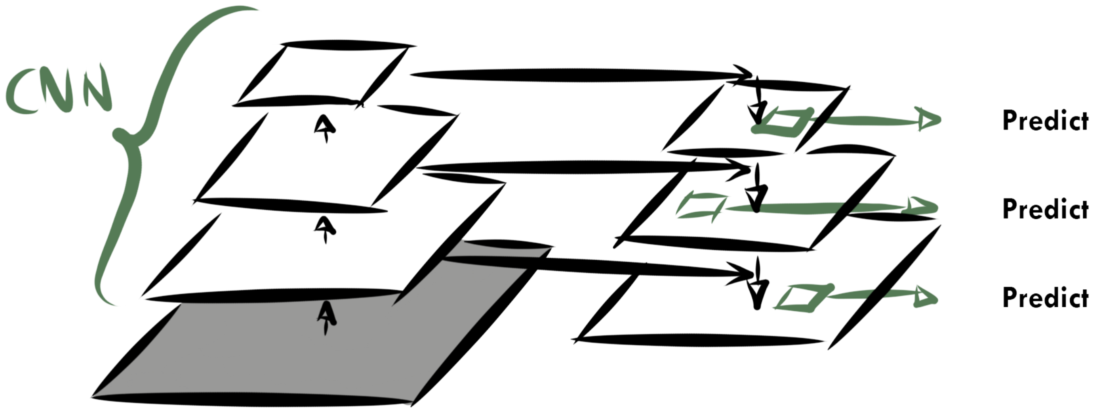


Figure 3.12: Feature pyramid network with its bottom-up and top-down pathways.

3.6.2 Evaluating performance

Evaluation metrics in object detection include Intersection over Union, Average Precision (AP) and Mean Average Precision (mAP). IoU compares the bounding boxes of the ground truth to the prediction and can be seen as an accuracy of the model for detection tasks. IoU can be calculated by dividing the area of intersection of the two bounding boxes with their union area, see Figure 3.13, giving a ratio [100]. Normally, an IoU score of >0.5 is considered a good prediction, as it is unlikely that an exact match will be predicted. Furthermore, in most cases, coordinates with a close match will provide equal information when working with bounding boxes [101].

Similarly to classification, true positives, false positives and false negatives are still highly relevant in object detection as this can be seen as the model's accuracy. In object detection terms, TP will mean that the predicted bounding box of a model exists at a certain position and that it is correct for the set IoU threshold. Following, a bounding box is FP if the bounding box exists in a certain position but is incorrect. Lastly, FN is used when the model did not predict an expected bounding box at that particular position when there should have been one. TP, FP, FN can then act as foundation for calculating precision and recall. Ideally these should both be high, but is often a tradeoff between the two metrics [102, 103]. Generally, this can be adjusted with the model's probability confidence threshold, which will determine the confidence level of the model's predictions. A higher confidence level will mean that the model is more confident in its predictions, which will increase the precision but decrease the recall.

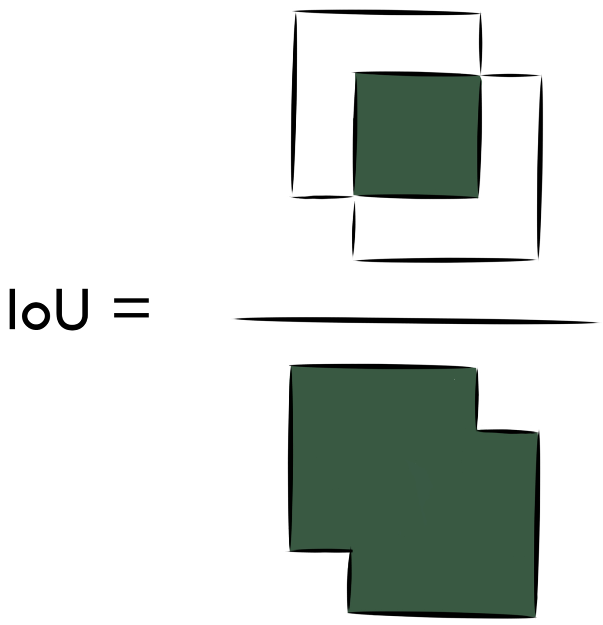


Figure 3.13: Intersection over Union forms a ratio describing the amount of intersection between two boxes.

On the other hand, a lower confidence threshold will predict more boxes and thus increase the chance of not missing a ground truth box and therefore increase the recall but lower the precision as it provides more FP. This can visually be represented in precision-recall curves, which plots recall over precision at different confidence thresholds. Another approach to precision-recall (PR) curves can be plotting at different IoU thresholds, as this will affect the sensitivity of what classifies as TP, affecting both precision and recall [59, 104, 105]. This approach is for example used to evaluate the large Common Objects in Context (COCO) dataset [106].

The PR curve can be extended into the metric AP, as it is represented by the area under the PR curve. This will hence quantify the balance between FP and FN, meaning that a higher value indicates a better model performance , and also decrease the arbitrariness of considering different thresholds. Each class has its own AP, but can be summarised in mAP [107]. It can thereby be argued that this metric gives a more realistic evaluation for real-life applications [108]. The metric mAP can hence be explained as

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i, \tag{3.12}$$

where $\boldsymbol{N}$ is the total number of classes and $\boldsymbol{AP_k}$ the $\boldsymbol{AP}$ of class $\boldsymbol{k}$ [107]. Which metric to use under what circumstances depends on application. For the COCO dataset evaluation, mAP is calculated in a range of IoU thresholds between 0.5 and 0.95, with a 0.05 step interval. The PASCAL VOC metric is instead the AP at the IoU threshold 0.5 [106].

3.7 Transfer learning

Although DL algorithms now show superior performance for image analysis tasks, it also comes with some limitations. Most DL algorithms are developed under the pretence of a large amount of data, as it is based on learning common features between data samples. This is especially true for convolutional neural networks, which today is the top performing DL technique when it comes to medical image analysis. However, in the medical community, data scarcity and limited expert-annotated data is a major bottleneck for these approaches, due to inaccessibility and high expenses. In order to solve this, many practices now involve transfer learning (TL) and domain adaptation techniques [109]. Traditionally, machine learning assumes the same data distribution and feature space in both the training and testing data. TL instead mimics the human cognition and experience of learning, where knowledge in similar contexts can be applied in new settings - thereby giving better performance compared to that of no previous skills or information. The purpose of TF is hence to improve learning by transferring knowledge from one domain to a different but similar domain [110], thus reducing training time for acceptable performances and potentially increasing evaluation metrics [109].

Models pre-trained with the dataset ImageNet [111], containing millions of labelled natural images, are commonly used with TL. These models are usually CNNs of different depth and have led to significant development of performance in natural image analysis tasks. Furthermore, this has also been seen improving tasks in the biological field [112], even though the distance from natural images to medical imaging dataset is highly influential to the network's performance. However, as with all ML solutions, setting the hyperparameters is still of utmost importance in order to achieve satisfactory results. Furthermore, in order to adapt the pre-trained models to the data of interest, it can be necessary to fine-tune the network - freezing and retraining some, or all, of the model's latest layers. Some studies in the subject of histopathology image classification have however shown that fine-tuning top layers, the final layers of the model, does not provide significant improvement to justify further training [113]. The amount of layers to freeze or retrain is left for optimisation, and based on the level of abstraction decreasing from the lowest level and up [114].

3.7.1 InceptionV3

The Inception network was originally developed by Szgedy et al. [115] as part of GoogleNet for the ImageNet Recognition Challenge in 2014 . The motivation behind Inception was to create a network with an increased number of network levels and number of units at each layer in order to be able to deal with difficult challenges. However, deeper and denser networks usually entail increased computational expense. This issue was approached by designing a CNN with sparse architecture instead of with fully connected layers like previous networks. Thus, the network was designed to consist of repeated components called inception modules. These layers have several parallel convolutional layers with different sizes and are concatenated into a single output vector. The dimensionality issue was further addressed by including 1x1 convolutional layers - effectively reducing dimensionality - as well as parallel max pooling layers in the inception layers. In order to improve performance, further versions were created with an increased number of network levels and number of units at each layer - with the motivation of a larger network being better at handling difficult challenges. While deeper networks usually are more computationally expensive, this was solved by decreasing the convolution size from 5x5 in InceptionV1 to 3x3 in InceptionV3. InceptionV3 also adds, among other things, an RMSprop optimiser and more batch normalisation [116, 117].

The general architecture of the InceptionV3 can be seen in Figure 3.14.

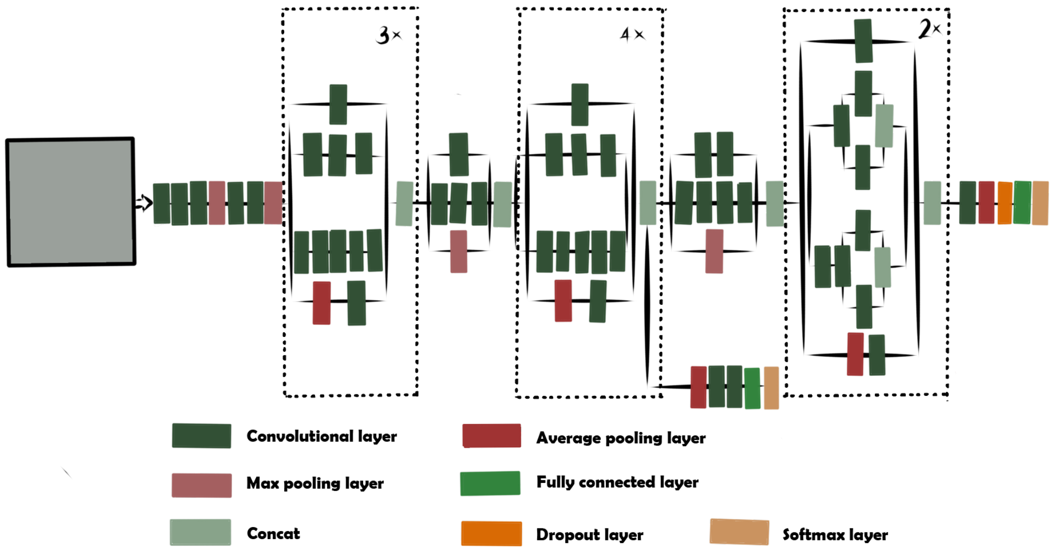


Figure 3.14: Architecture of InceptionV3.

3.7.2 YOLO

You Only Look Once (YOLO) is a series of open-source computer vision models that currently are state-of-the-art with superior performance. The given name stands for the ability of predicting every present object within one forward pass, therefore "only looking once". The YOLO models were created as a regression task instead of classification for predicting box coordinates. The latest iteration of the model series is YOLOv8 [118], released in 2023. Instead of using anchor-based detection, YOLOv8 predicts object centres which simplifies the model and decreases the computational complexity. YOLOv8 uses the concept of grid cells, dividing the image into cells depending on the scale and resolution, where each feature map or grid cell predicts an object whose centre exists in that cell. The predictions are then stacked together to create final predictions with the help of feature pyramids. As of today, YOLOv8 is not published in any official paper, but an architecture, recognised by the developing team, can be seen in Appendix A.1.

Chapter 4
Method

4.1 System overview

To perform the project, a stationary computer with the graphic card NVIDIA GeForce RTX 3060 Ti with 8 GB of memory was used. CUDA toolkit 12.2 and NVIDIA CUDA Deep Neural Network (cuDNN) 8.9.6 were utilised in order to execute the network training efficiently with the mentioned GPU. The code was developed in Python 3.9.18 with Keras and TensorFlow 2.8 for the monolayer task. The localisation task was performed in Python 3.10.12 with Keras and Tensorflow 2.15 using a WSL2 Ubuntu virtual environment.

4.2 Finding monolayers

The following section refer to the first part of the project that focuses on finding monolayers in avian blood samples.

4.2.1 Data collection

Using the DC-1 system, a number of avian blood samples accessible at CellaVision were analysed using 100x magnification. The performed analysis included scanning the blood smear and thereafter providing a monolayer coordinate range, if the monolayer region existed in the specific sample. The coordinates for the WBC monolayers in relevant samples could then be extracted from the log file of the system.

Images from the same samples were then collected from the DM9600 system in 10x magnification. As both systems had individual limitations, not all image data of the relevant samples from the DC-1 system could be collected. Therefore, the data that existed across both the DC-1 and DM9600 were matched up and the rest of the data was discarded. In total, 13020 images could be collected for 110 samples. The collected images were originally captured with magnification power 10x in 640 x 480 pixels. The sample as a whole can be represented by a reference system, as seen in Figure 4.1, with the range [5000, 21000] μm in x-axis and [5000, 50000] μm in y-axis. Each image can be related to the reference system and the blood smear through its midpoint coordinates.

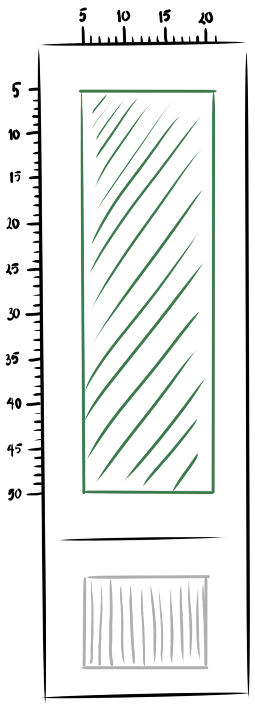


Figure 4.1: Reference system for blood slide sample.

Initially, the data was labelled in a binary manner, where images within the coordinate range of monolayer were classified as 'monolayers' whereas the rest were labelled as 'not monolayer'. The distribution of samples can be seen in Figure 4.2. This configuration of the dataset will henceforth be mentioned as the binary dataset.

Furthermore, the data was also labelled into a second multiclass dataset, with the classes: 'too sparse', 'monolayer' and 'too thick'. A threshold for the monolayer label was set to be 49.5%, ergo an image needed to at least contain 49.5% monolayer in its range to be classified as a monolayer. This further meant that 35 samples contained no monolayers as it fell below the threshold, resulting in 75 samples containing at least one image classified as a monolayer. The other labels were set on a visual basis; the images that were less dense compared to the monolayers were classified as 'too sparse' and the images that were more dense compared to monolayers were classified as 'too thick'.

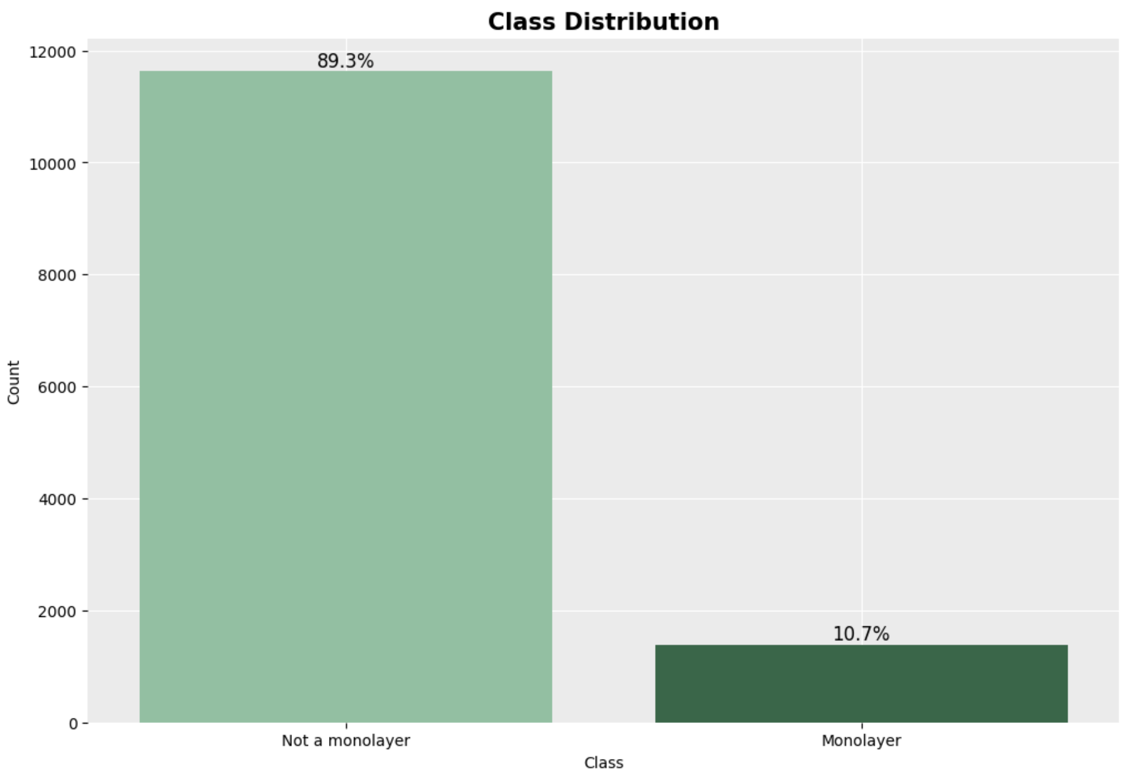


Figure 4.2: Class distribution of the binary dataset.

Moreover, there was also a need for manual adjustments of some images, as they, after visual inspection, were distinctly different to a monolayer even though they were in the monolayer coordinate range, most likely due to errors of the analysers. Some examples of these types of misclassifications can be seen in Figure 4.3. No 'monolayers' labels were added to any images, even if they visually could be estimated as one.

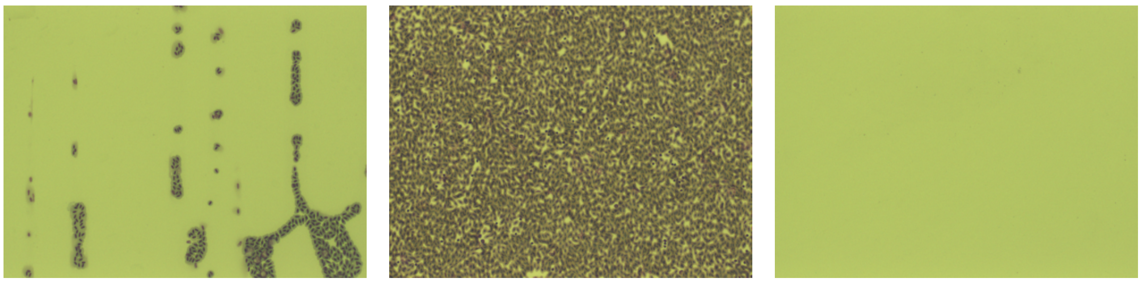


Figure 4.3: Examples of images in need of manual adjustments, from label 'monolayer' to 'too thick' or 'too sparse'.

Arbitrary examples of the final dataset with their corresponding labels can be seen in Figure 4.4 and the final distribution can be seen in Figure 4.5.

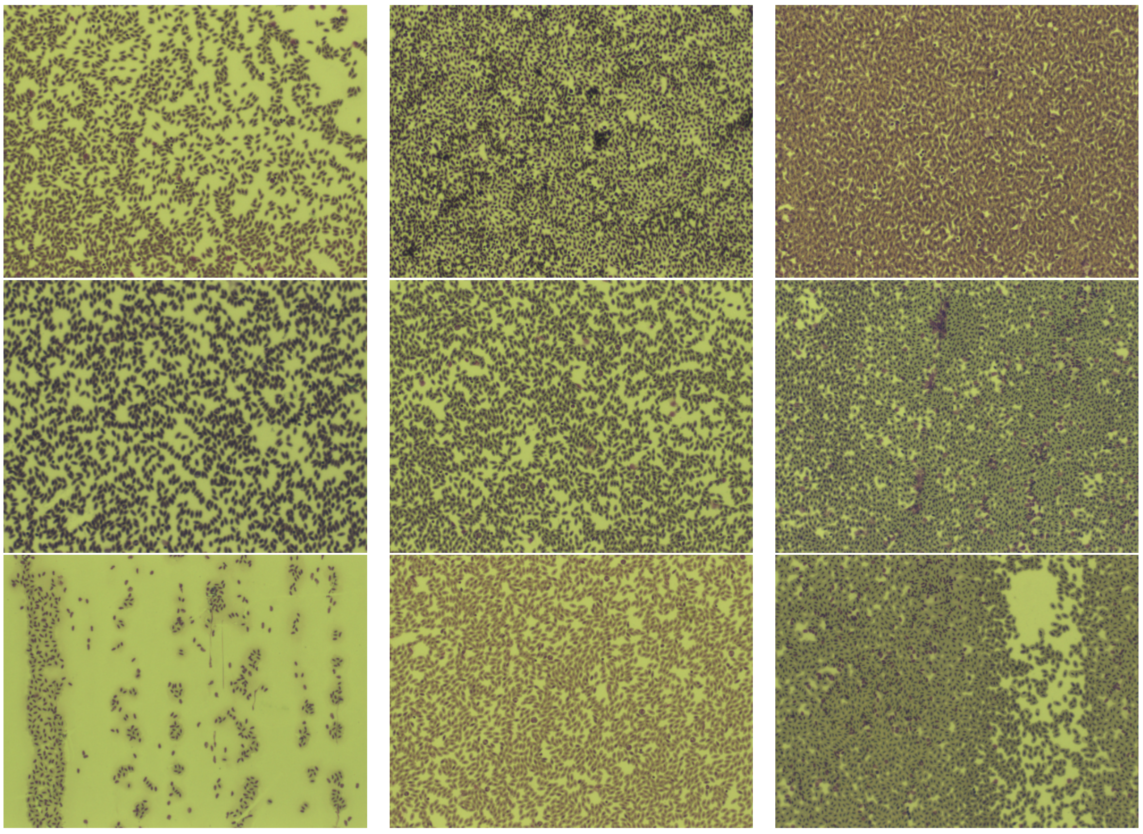


Figure 4.4: Examples of images being classified as too sparse, monolayer or too thick.

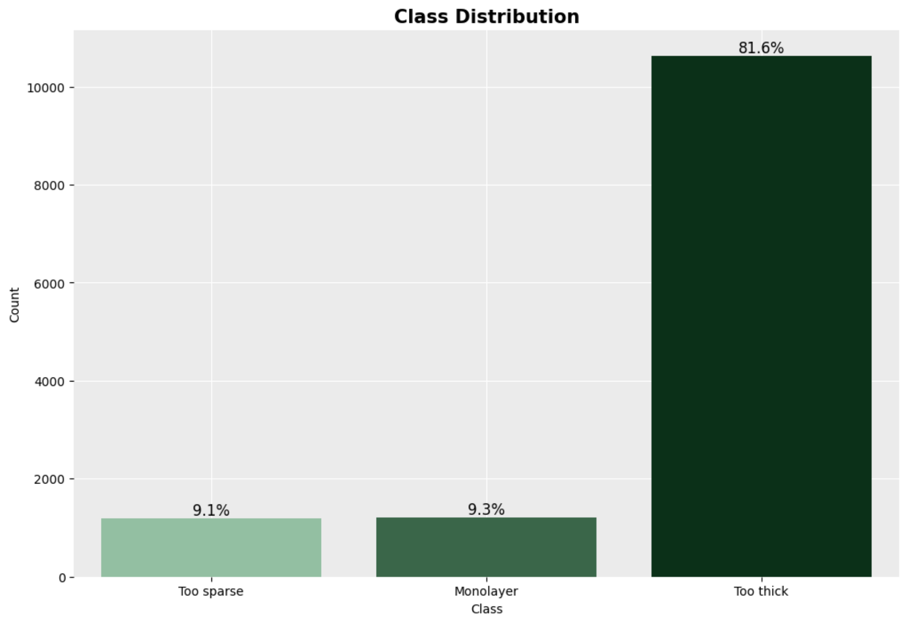


Figure 4.5: Class distribution of entire multiclass dataset.

Dataset for semi-supervised learning

With the purpose of trying to lessen the subjectivity of classifying the images, a dataset for pseudo labelling was also prepared. 164 images were left out of labelling, based on the level of difficulty to determine which class it belonged to, corresponding to 1.3% of the entire dataset. Examples can be seen in Figure 4.6.

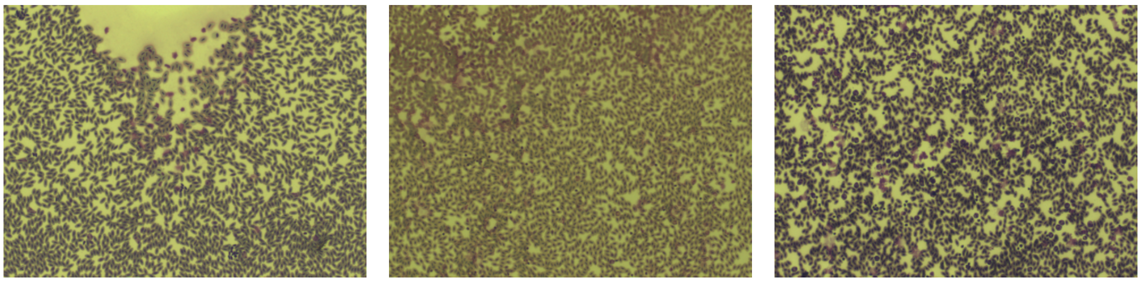


Figure 4.6: Examples of images with no label.

The dataset distribution used with pseudo labelling can be seen in Figure 4.7.

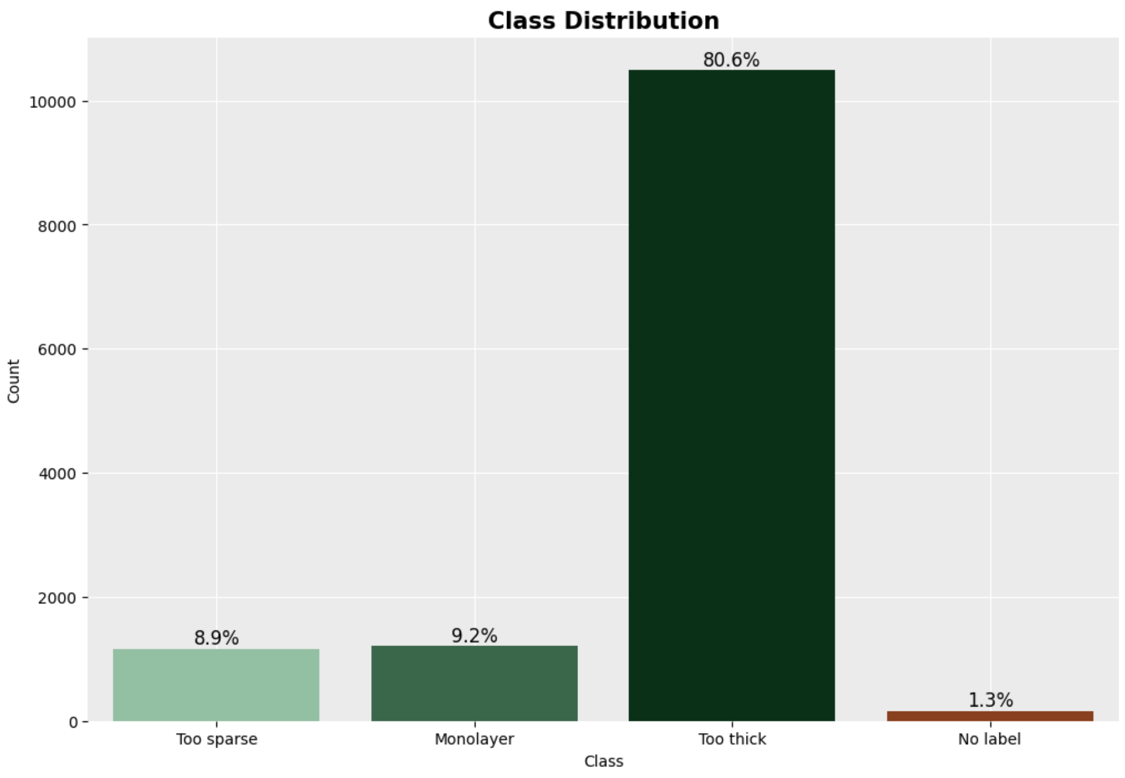


Figure 4.7: Class distribution of the entire multiclass pseudo dataset.

4.2.2 Pre-processing

Pre-processing involved downsampling images to the correct dimensions corresponding to the specific networks, that is 224x224x3 for EfficientNetB0 and ResNet50, 240x240x3 for EfficientNetB1, and 299x299x3 for InceptionResnetV2 and InceptionV3 [119]. Furthermore, pre-processing also included normalisation to a pixel value in the range of 0 to 1. Images were loaded into memory in batches in order to accommodate the memory limitation, and split into a training, validation and test set in a (0.6, 0.2, 0.2) ratio. Further, different configurations of data augmentation were applied to the images. Both normalisation and augmentation were built as layers of the model architecture. This limited the possible augmentation techniques to those available in the Keras Layers API. As a final step of preparing the dataset for training, the labels were one-hot encoded.

4.2.3 General test design

The general model design included using a pre-trained transfer model as a base model initialised on the ImageNet dataset with average pooling. As previously mentioned, the model design also included preprocessing layers in a sequential manner, such as resizing and normalisation of images, as well as the desired augmentation. Additional layers were then added to make the model better fit the created dataset. The added layers initially consisted of a flattening layer and a dense layer, with a value equal to the number of classes, giving the output with a softmax activation function. The network was trained for a predefined number of epochs while monitoring training as well as validation loss. Throughout the process, the model with the lowest validation loss was consistently saved, in order to ensure preservation of the best performing model. After training the network for the set number of epochs, the probabilities associated with each class of the images were computed by the trained model. The classes were then assigned to the images by selecting the classes with the highest probability in the predicted outcomes. Finally, evaluation metrics such as accuracy, precision, recall and F1-score were calculated. In most tests, performance was also calculated in terms of class precision, specifically the class representing monolayers.

The models in the following tests were created, trained and tuned experimentally, following results of previous models and not in a purely sequential manner as presented below. The presented tests are organised as follows: I relates to the binary dataset, II to the multiclass dataset and III refers to semi-supervised learning of the multiclass dataset. Additional tests can be seen in Appendix B.

4.2.4 Initial tests with binary dataset

Test Ia: Selection of baseline architecture

Initially, a smaller exploration of different pre-trained transfer learning models with the binary dataset was conducted in order to evaluate which transfer learning model architecture could be the most suitable for the created dataset. The considered transfer learning models were based on accessibility in Keras Applications, which is part of the library housing pre-trained deep learning models, as well as successful models identified in the performed literature review. Thus, the selected models were EfficientNetB0, EfficientNetB1, Inception-ResNetV2, InceptionV3 and ResNet50. These models incorporate diverse architectures with varying numbers of parameters - ensuring a spectrum of complexities. The test was performed with identical parameter configurations, which included a constant learning rate of 10^{-5} with a batch size of 32 trained for 50 epochs. The training was optimised with the optimiser ADAM set to minimise the cross-entropy loss. They all had mild augmentation applied, presented in Table 4.1. Due to the class imbalance in the binary dataset, the models were mostly evaluated based on class precision and class recall for the monolayers, on top of the general evaluation metrics. The best performing model was chosen as the base model for the following tests.

Table 4.1: Augmentation for testing of transfer learning models with the binary dataset.

Augmentation	**Configuration**
Random Flip	Horizontal & Vertical
Random Rotation	0.4, filled with reflected pixels
Random Translation	Height and width factor 0.2, filled with reflected pixels
Random Zoom	0.3
Random Contrast	0.4

Test Ib: Adding generalisation

As a continuation of the previous test, some hyperparameters were changed in order to investigate generalisation and decrease overfitting of the transfer learning model InceptionV3. This included adjusting the learning rate as well as adding dropout and an additional dense layer.

Test Ic: Dropout and batch size with fine-tuned TL models

Generalisation was further tested in combination with fine-tuning of the pre-trained transfer learning models. Fine-tuning was necessary to test in order to find out to which extent the model needed to be fine-tuned, if at all. The fine-tuning was performed by firstly fixing the lower layers of the pre-trained network, thereby preserving the previously learnt knowledge of feature representation in ImageNet. The top layers of the model, believed to be more task-specific, could then be trained with a learning rate of 10^{-3} for a set number of epochs. The fine-tuning of top layers was then followed by training of lower layers. The learning rate was lowered to enable easier adjustment of the learnt weights to the domain of the custom dataset. For this step, two different learning rates were tested: 10^{-6} and 10^{-7}, thereby creating two different configurations of the models with fine-tuning. These two configurations were further tested with an added dropout layer with two different values, 0.5 and 0.8, to increase generalisation. Lastly, the batch sizes were varied between three values: 16, 32, 64. Larger batch sizes could not be considered due to memory limitations in the GPU.

4.2.5 Multiclass testing

Test IIa: Batch size and effect of fine-tuning in multiclass dataset

Test IIa was performed closely related to test Ic but set in the multiclass environment, in order to see how the multiclass dataset was affected by the similar parameters and training. The multiclass testing started based on knowledge learnt from previous tests, and the tests were thereby based on the best performing models and hyperparameters. As opinions of how significant it is to fine-tune the pre-trained transfer learning models seemed to be contradicting, as previously mentioned, it was important to test whether this would affect performance with the given data. Like Test Ic, the top layers were trained with the learning rate 10^{-3}. The learning rate of the lower layers was set to 10^{-6}. The performances of these models were then compared to performances of models without fine-tuning, trained for the same amount of epochs and with the learning rate that the majority of the layers were trained on in the fine-tuning models, that is 10^{-6}. The models without fine-tuning were still initialised on ImageNet but set to train all layers at once. This testing was combined with two different batch sizes.

Test IIb: Optimiser

The model was initially set up with the ADAM optimiser, but it might not be the ideal optimiser for all datasets. In order to test the effect of the optimiser, the ADAM algorithm was compared to the RMSProp algorithm. Furthermore, ADAM contains the epsilon parameter, which is a constant value added for numerical stability, to avoid dividing by zero when updating variables at times where the gradient is close to zero. The epsilon value defaults to 10^{-7}. It is generally desirable to have a lower epsilon as a larger epsilon value will mean smaller weight updates which in turn makes the training progress slower. However, this might not be the best value for larger networks such as InceptionV3 initialised on the ImageNet dataset. For a more efficient convergence less sensitive to minor changes of the denominator for this type of networks, a value of 0.1 is suggested instead [120]. Therefore, the data is also trained on the ADAM optimiser with default value as well as with the suggested value to investigate if this improves model learning. Other parameters were identical between the tested models in the training phase and based on previous results, including a constant learning rate of 10^{-6}.

Test IIc: Learning rate adjustments

As learning rate is a crucial hyperparameter in network training, it was imperative to evaluate the impact of different adjustments of the learning rate. Learning rate is to some extent adjusted by the optimiser, as the RMSProp used in the model design is adaptive, but can be set to have a specific shape. The effect of learning rates had to some extent been investigated in earlier tests, but then mainly focusing on constant learning rates. This test instead aims at figuring out the impact of different learning rate adjusters lowering learning rates in a controlled manner, as this theoretically should allow the network to learn more details as it trains in higher levels of the feature hierarchy. More specifically, the test focuses on learning rate schedulers implementing either step wise lowering of the learning rate or implementing exponential learning rate decay. Initial values and final values of the schedulers were varied, in order to optimise and find the most beneficial configuration for model performance. The range of initial values varied between 10-3 and 10-4 and between 10^{-7} and 10^{-9} for the final values. For the exponential learning rate decay scheduler, the decay steps with a decay factor computed as

$$LrDecayFactor = (\frac{lr_{final}}{lr_{initial}})^{\frac{1}{epochs}} \tag{4.1}$$

and the steps per epoch were calculated as

$$Steps_{epoch} = \frac{\text{size of training set}}{\text{batch size}} \tag{4.2}$$

The general shapes of these learning rate schedulers as well as schedulers in later tests can be seen in Appendix A.2.

Test IId: Loss functions

Optimisation during training of deep neural networks is based on the specified loss function, as this guides the model towards better parameter values for the dataset. This makes the loss function an essential parameter for model performance, and what loss function to use can vary between different objectives and approaches. Different loss functions have varying properties, as some for example can be sensitive to noise and specific types of data. Depending on the nature of the data, it can be necessary to adjust the loss function to account for data qualities such as class imbalance by for example choosing a loss function that focuses on samples that are hard to classify. For a clear comparison, three models with different loss functions were firstly trained with identical configurations. The investigated losses included cross-entropy loss, α-balanced focal cross-entropy loss and hinge loss. The models were trained with default values of loss hyperparameters. For α-balanced focal cross-entropy loss, this meant values of 0.25 and 2.0 for α and γ respectively.

α-balanced focal cross-entropy loss was further investigated with varying values of the hyperparameters α and γ. The two hyperparameters used their default values as starting points and were then tuned experimentally. α was varied with a single value between 0.005 and 0.4. The suggested inverse frequency of classes was also tested. On the other hand, γ was varied between 2.0 and 10.0. As γ controls the focusing effect of the focal loss function, an increased value will enhance the effect of loss for misclassified samples, in particular hard-to-classify samples. A value of 0 will mean that the α-balanced focal cross-entropy loss is equal to cross-entropy loss. The test therefore focused on increasing the value instead of lowering it, with the goal to increase precision for monolayers considered hard to classify. Most of the models were trained for 50 epochs, but in some cases they were also trained for 200 epochs to give the network longer time to learn.

Test IIe: Expansion of the model architecture

The architecture of a neural network plays a significant part when it comes to its ability to extract relevant information from the data it is applied on. Therefore, it was important to investigate whether the model performance could benefit from increased model capacity. This was carried out by expanding the model from its general architecture by adding several dropout and dense layers to the existing model, which was the transfer learning base model as well as a flattening layer and a classifying dense layer consisting of three neurons (equal to three classes). These two types of layers were added in combination, as dense layers work to add model complexity and number of parameters while dropout layers add regularisation and therefore decreases the risk of overfitting the model. The layers were added after the flattening layer, starting with an added dropout layer and then alternating between the two different types of layers. Parameters that were tested included varying the number of added layers, the number of neurons in the dense layers as well as the rate of dropout. The classes were predicted by an identical dense layer with three neurons as used in previous architectures.

4.2.6 Tests with pseudo-labelled multiclass dataset

Training the multiclass dataset with supervised learning was performed in two steps. Firstly, the labelled data was trained for a determined number of epochs with a specified model just as with the other two datasets. The best performing model from this training was then loaded and used for prediction of the unlabelled data. The images were then sorted according to their predicted classes and incorporated with the rest of the data. The labelled as well as the previously unlabelled data was then trained together using the same model and similar evaluation as previously were performed.

Test IIIa: Learning rates with semi-supervised learning

Learning rate adjustments done in Test IIc, were also tested with the semi-supervised learning to assess the consistency across the different datasets. A constant learning rate of 10^{-6} was used as a baseline for comparison. Two different step wise schedulers were considered starting from two different values. The first one started with a learning rate of 10^{-4}, trained for 10 epochs and then lowered while the second one started with a higher value of 10^{-3}, lowered to 10^{-4} after 5 epochs and then lowered again after training for 10 epochs total. A third step wise scheduler was created, starting with a linear warm-up of the learning rate. This is believed to help add stability during early training and avoid erratic model behaviour of early training. The linear warm-up was carried out during the first 5 epochs, and then behaved equal to the second step wise learning rate scheduler. The general shapes can be seen in Appendix A.2.

Test IIIb: Multiple learning rate schedulers

Once training with semi-supervised learning involves continuing learning after the unlabelled samples have been classified, this also means that the majority of the network has already been trained for a while when including the new samples. Therefore, it was of interest to see whether a scheduler starting from a lower learning rate would improve the performance of the model - assuming that the network has already learnt the lower levels of the dataset also present in the new samples and then focus on learning details.

The data was firstly trained with the same exponential learning rate decay scheduler starting from 10^{-4} and ending on 10^{-6}. A second scheduler was created with varied initial and final learning rates, ranging from 10^{-5} to 10^{-7}. This was compared to the performance of two identical schedulers. The models were all trained with an expanded model architecture trained to optimise focal loss.

The concept was also applied to a step wise learning rate scheduler. Here, different initial and final values for both the first and the second scheduler were tested with the aim to find the optimal combination. The values ranged between 10^{-3} and 10^{-6}. These models were trained to optimise performance on categorical cross-entropy loss using a dropout layer with 0.8 as drop out rate, a flattening layer and a classifying dense layer. The models were then compared to a model trained using the same scheduler for both training phases.

Test IIIc: Image aspect ratio

According to the documentation, pretrained transfer learning models suggest a set image size with equal height and width. As images were not collected in this ratio, they needed to be preprocessed to fit the requirements. Two techniques were considered, either cropping the images to maintain image ratio or distorting them by pushing them to have the same height and width. Two models deploying the techniques of maintaining image ratio and distorting image aspect ratio respectively, using an expanded architecture, were trained with an exponential learning rate decay scheduler between the values 10^{-4} to 10^{-6} and 10^{-5} to 10^{-7}. Performance was evaluated and compared to each other in order to find out which image aspect ratio benefitted the aim.

Test IIId: Augmentation & Imbalanced data

Augmentation is a crucial step in network training, both in terms of added regularisation and generalisation to the model, by for example introducing noise and variability, as well as expansion of the dataset. It was therefore of interest to investigate how different augmentation techniques and their hyperparameters affected the capability of the model. As previously mentioned, the augmentation was designed to be included as layers within the Keras model design, which limited the possible augmentation techniques. The test originated with the augmentation settings set in Table 4.1, including flipping, rotating, translating, zooming and adding contrast. Hyperparameters within these were then adjusted and the performance was then compared to the initial augmentation. The model was trained using a step wise learning rate scheduler earlier introduced with top layers consisting of a dropout layer with dropout rate 0.8, a flattening layer, and a classifying dense layer.

Some of the augmentation combinations were also tested with an upsampled dataset. An imbalance dataset can often lead to bias towards the majority class, and thereby performing poorly on minority classes. Upsampling can help with creating more samples of the under-represented classes which then can be used to present these more often to the network. This can furthermore help with allowing the network to learn classes more efficiently, with less bias. Another strategy to handle imbalanced data is by downsampling the majority class, but this can lead to information loss and is less desirable with sparse datasets [121]. Although oversampling for class imbalance learning is a frequently used technique, some argue that it is unreliable and ineffective [122]. The ‘too sparse’ class and the ‘monolayer’ class were up-sampled to the same number of samples as the ‘too thick’ class. This was performed using the library Albumentations [123], by creating small variations to the original class samples with mild augmentation. Two models were then trained with different augmentation settings similar to previously and compared as they were trained with the same model configurations besides the mentioned changes.

Test IIIe: L1/L2-regularisation with expanded model architecture

Test IIIe incorporated Elastic Net (L1/L2)-regularisation on the expanded model architecture. Elastic Net-regularisation was used to add generalisation to the architecture in order to reduce overfitting and provide a more stable training. The strength of regularisation was optimised by varying values of both the L1 and the L2 kernel. L1 was varied between 10^{-6} and 10^{-5} while L2 was varied between 10^{-4} and $5 * 10^{-5}$. All were trained using α-balanced focal loss and compared to a model trained with identical configuration apart from regularisation.

4.2.7 Evaluation of proposed model

After concluding the series of tests, a final model was chosen based on the observed results. The selection was particularly made with attention to precision of the class 'monolayer', considering the aim of the project. To ensure model stability and generalisation, the model was then evaluated on the previously unseen test dataset. Besides performance metrics including accuracy, precision, recall and f1-score as well as more specific metrics such as class precision, the confusion matrix was calculated and ROC-AUC curve was plotted.

4.3 Locating WBCs

Data for the WBC localisation was extracted from monolayers collected in the previous task. Box coordinates from these monolayers were calculated by manually circling possible WBC's in the images, using internal CellaVision software. The circles were transformed to boxes using the minimum and maximum x- and y-values. These were then saved in an XML file for that specific image. The box coordinates could then be extracted with its corresponding file and saved in a dataframe. In total, 18666 boxes were collected representing WBCs in 333 images and 29 samples. Each image had an area of 480 x 640 = 307200 pixels^2 with an average box area of 216 pixels^2.

In order to fit restrictions set by the transfer learning models, the height and width of the image needed to be equal. As aspect image ratio is important for localisation, considering the spatial awareness, the images were padded with zero padding to the correct dimensions of 640x640 pixels. The data could then be divided into a training dataset and a validation dataset. Similar to the previous task, augmentation was included as layers in the defined model. Initial augmentation to the training dataset contained random horizontal flipping, random contrast, jittered resize and random value of hue.

The model consisted of a pre-trained YOLOV8 Detector with a backbone. A range of different backbones were tested by varying different sizes, depths of feature pyramid depths and architectures. The backbones were further tested by freezing pre-trained weights and only training top layers, as well as training all layers at once. The best performing backbone was then chosen based on the evaluated results and used in the following tests. The models were then trained on the training data for 200 epochs, initially using an exponential learning rate decay scheduler with initial value 10^{-3}, calculating the decay steps and learning rate decay factor as previously. The training was set to optimise the loss function Complete IoU (CIoU) using ADAM. Finally, COCO metrics were computed through a callback function during training. This had to be done to efficiently be able to perform necessary calculations while minimising storage of previous predictions and thereby consider limitations in system memory. The model was saved based on the best validation mean average precision and used for evaluation. Finally, tracked metrics could be extracted and precision and recall could be calculated at determined thresholds for a more extensive evaluation.

Hyperparameter optimisation was performed in several steps. After the backbone was decided, image and batch size was tested and determined. These hyperparameters had to be tested in combination, as a larger image size put constraints on the maximum batch size. The tests were initialised on a batch size of 4, but a batch size of 2 was also considered in the test. The need to increase image size was to account for the small magnification, and increasing image size would also increase the size of the WBC boxes. Hence, increasing image size was assumed to help the network find small objects. Therefore, apart from the original 640x640 pixels, 800x800 pixels and 960x960 pixels were tested while the batch size was decreased accordingly. 960x960 was the largest image size possible to test considering the memory limitations.

Moreover, the loss function was tested by training three models with similar model configurations on three different types of losses. Only the box loss was considered, as there was no classification problem to solve. The tested losses included Complete IoU (CIoU), mean squared error and Smooth L1 loss. The learning rate was tested in a similar manner; four different models with the same configurations were trained using different learning rate schedulers and varied values of initial learning rate.

The most extensive test in this task included adjusting augmentation settings, which was deemed important as the dataset was relatively sparse. This included testing more commonly used parameters such as hue, brightness, colour channel shift, saturation, and flip. Elastic transformations suggested to improve performance in cell environments were tested in the form of shear deformation. More advanced techniques such as grid masks were also tested. Grid mask was considered as a way of introducing diversity and robustness to the training data and thus increase generalisation. Grid mask divides the image into a grid in which certain cells are masked out. As the network then can not learn from the blocked cells, it forces the model to learn from other features. Thus, the bias of the model might decrease as it might stop relying on specific patterns, which leads to better performance.

Chapter 5
Results

5.1 Finding monolayers

5.1.1 Initial tests with binary dataset

The following section explains the results following the tests performed in Section 4.2.4

Test Ia: Selection of baseline architecture

Evaluation metrics of the performance of five different transfer learning models trained with the same hyperparameter configuration following Test Ia can be seen in Table 5.1. Accuracy and loss over epochs for the training and validation dataset trained with the InceptionV3 network as a base model as well as its related ROC curve can be seen in Appendix A.3.

Table 5.1: Initial exploration of pre-trained transfer architectures presented with their respective number of parameters.

Model	Number of parameters	Accuracy	Class precision (mono/no mono)	ROC AUC
EfficientNetB0	5.3 M	0.88676	0.00000/0.89258	0.569636
EfficientNetB1	7.8 M	0.89712	0.55000/0.91098	0.58925
InceptionResNetV2	55.8 M	**0.92253**	0.67014/**0.96331**	**0.82671**
InceptionV3	23.8 M	0.91206	**0.67949**/0.94981	0.82386
ResNet0	25.6 M	0.90015	0.60938/0.93367	0.76986

Test Ib: Adding generalisation

Shown in Table 5.2 are the performances of Test Ib; models using InceptionV3 as base model with varying added layers on top, including a dropout layer, one or two dense layers and a flattening layer and two different learning rates. The training history for the first model can be seen in Appendix A.4.

Table 5.2: Evaluation metrics of models with changed constant learning rates and added generalisation in the top layers.

Learning rate	Top layers	Accuracy	ROC AUC
10^{-6}	Flatten, Dropout (0.5), Dense(2)	0.91129	0.83110
	Flatten, Dense(1024, relu), Dropout(0.5), Dense (2)	0.90668	0.73310
10^{-7}	Flatten, Dropout (0.5), Dense(2)	0.89977	0.55273

Test Ic: Dropout and batch size with fine-tuned TL models

The results of testing with the transfer learning approach of fine-tuning top layers in the InceptionV3 network followed by training of lower layers, as defined in Test Ic, can be seen in Table 5.3

Table 5.3: Parameters and metrics for models with fine-tuned transfer learning approach.

Epochs	Dropout	Learning rate	Batch size	Accuracy	ROC AUC
50∗2	0.5	10^{-3}+10^{-6}	16	0.92166	0.84167
			32	0.91936	0.84834
	0.8			**0.92628**	**0.86888**
100∗2			16	0.91934	0.85074
			64	0.92128	0.84941
		10^{-3}+10^{-7}	16	0.91283	0.77312
			64	0.91359	0.80854

5.1.2 Multiclass testing

Following section refer to tests performed on the multiclass-labelled dataset, explained in Section 4.2.5

Test IIa: Batch size and effect of fine-tuning in multiclass dataset

Table 5.4 shows how model performance is affected by varying batch sizes between 16 and 32 as well as the effect of fine-tuning top layers of the determined model architecture.

Table 5.4: Effects of fine-tuning with different batch sizes in multiclass dataset.

Epochs	Learning rate	Batch size	Accuracy	Precision	Recall	F1-score	ROC AUC
50	10^{-6}	16	0.88642	0.75004	0.75978	0.74539	0.82011
		32	**0.90867**	0.78974	0.81858	0.79871	0.84738
50∗2	10^{-3}+10^{-6}	16	0.88411	0.75713	0.75743	0.74266	0.82215
		32	0.89985	**0.79707**	**0.82821**	**0.81095**	**0.84930**

Test IIb: Optimiser

The resulting table for testing two different optimisers and adjusting the epsilon parameter according to Test IIb can be seen in Table 5.5.

Table 5.5: Evaluation metrics for comparison between ADAM and RMSProp optimisers.

Optimiser	Epochs	Epsilon	Accuracy	Precision	Recall	F1-score	ROC AUC
ADAM	50	10^{-7}	0.90867	0.78974	0.81858	0.79871	0.84930
		0.1	0.84996	0.59586	0.46545	0.49215	0.76322
	200	0.1	0.89946	0.80049	0.65033	0.66740	0.86782
RMSProp	50	10^{-7}	**0.92671**	**0.81189**	**0.86814**	**0.836098**	**0.87775**

Test IIc: Learning rate adjustments

Table 5.6 presents the evaluation metrics obtained from the conducted Test IIc of specific learning rate adjustments; learning rate schedulers with step wise lowering of the learning rate and exponential learning rate decay.

Table 5.6: Performance of models trained with learning rate schedulers. Ep stands for epochs while s, m and t stands for sparse, monolayer and thick respectively.

Type of LR controller	Learning rate	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
Scheduler	Ep<=10: 10^{-2} Ep>10: lr$*e^{-1}$	0.91903	0.78881	**0.87866**	0.82699	0.85068	0.80435/ 0.57055/ **0.99152**
	Ep<=10: 10^{-4} Ep>10: lr$*e^{-1}$	**0.93553**	0.81283	0.86198	0.83545	0.87600	0.76259/ **0.68605**/ 0.98986
Exponential learning rate decay	$10^{-3} \rightarrow 10^{-9}$	0.92056	0.79295	0.82290	0.80720	0.85925	0.81226/ 0.59289/ 0.97371
	$10^{-4} \rightarrow 10^{-6}$	0.93476	**0.82207**	0.85941	**0.83921**	**0.87973**	0.80000/ 0.67443/ 0.98179

Test IId: Loss functions

Table 5.7 shows the performance of a model trained while minimising three different types of losses - cross-entropy, α-balanced focal cross-entropy and hinge loss. The attempt to further adjust parameters within α-balanced focal cross-entropy is presented in B.2.

Table 5.7: Effects of different types of losses trained with default values.

Loss	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
Cross-entropy	0.92517	0.80559	0.83212	**0.81828**	0.86769	**0.81853**/ 0.62400/ 0.97425
α-balanced focal cross-entropy	**0.92594**	**0.80904**	0.80600	0.80602	**0.87610**	0.77193/ **0.69006**/ 0.96512
Hinge	0.91289	0.77117	**0.83700**	0.80073	0.84184	0.77580/ 0.55634/ **0.98138**

Test IIe: Expansion of the model architecture

Performance following Test IIe can be seen in Table 5.8. This presents effects of expanded model architecture, referred to as model configuration.

Table 5.8: Performance with expanded model architectures. Dr indicates a dropout layer while D stands for a dense layer. F stands for flatten. The values within the parentheses represent the value of dropout respectively the number of neurons.

Model configuration	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
F, Dr(0.8), D(3)	0.93476	0.82207	0.85941	0.83921	0.87973	0.80000/ 0.67443/ 0.98179
F, Dr(0.2), D(128), Dr(0.2), D(64), Dr(0.2), D(32) ,D(3)	0.93707	0.83997	0.86768	0.85330	0.88870	0.87649/ 0.66539/ 0.97804
F, Dr(0.4), D(256), Dr(0.4), D(128), Dr(0.4), D(64), Dr(0.4), D(3)	0.92594	0.79594	0.84487	0.81808	0.86234	0.78169/ 0.62400/ 0.98214
F, Dr(0.8), D(256), Dr(0.8), D(128), Dr(0.8), D(64), Dr(0.8), D(3)	0.81581	0.27194	0.33333	0.29952	-	-/-/ 0.81581
F, Dr(0.5), D(256), Dr(0.5), D(128), Dr(0.5), D(64), Dr(0.5), D(3)	0.93208	0.83586	**0.87639**	0.85290	0.88173	**0.89744**/ 0.62951/ 0.98065
F, Dr(0.4), D(1028), Dr(0.4), D(256), Dr(0.2), D(128), Dr(0.2), D(64), Dr(0.2), D(32), D(3)	0.93208	0.80721	0.86822	0.83480	0.86962	0.79021/ 0.64259/ **0.98882**
F,Dr(0.2), D(1028), Dr(0.2), D(256), Dr(0.2), D(128), Dr(0.2), D(64), Dr(0.2), D(32), D(3)	**0.93937**	**0.84838**	0.87219	**0.85977**	**0.89395**	0.88525/ **0.68182/** 0.97807

5.1.3 Tests with pseudo-labelled multiclass dataset

The following section refer to tests performed with the pseudo-labelled multiclass dataset, see Section 4.2.6 based on semi-supervised learning.

Test IIIa: Learning rates with semi-supervised learning

Model performance from training with four different types of learning rate shapes over epochs, following Test IIIa, can be seen in Table 5.9.

Table 5.9: Evaluation metrics from models trained with three different types of learning rates.

Type of scheduler	Values	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
Constant	10^{-6}	0.91524	0.78032	0.80679	0.79293	0.85115	0.77559/ 0.59438/ 0.97100
Scheduler	Ep<=10: 10^{-4} Ep>10: lr$*e^{-1}$	0.93896	0.83079	0.85768	0.84322	0.88789	0.81226/ 0.69748/ 0.98263
	Ep<=5: 10^{-3} Ep>5 & <=10: 10^{-4} Ep>10: lr$*e^{-1}$	**0.94984**	**0.86026**	**0.87783**	**0.86875**	**0.90827**	**0.84211/ 0.75309/ 0.98559**
Scheduler with linear warm up	Ep<=5: Linear warm up, Ep>5& <=10: 10^{-3}, Ep>10 & <=15: 10^{-4}, Ep>15:lr$*e^{-1}$	0.91330	0.77333	0.83975	0.80305	0.84299	0.78067/ 0.55670/ 0.98260

Test IIIb: Multiple learning rate schedulers

Table 5.10 performance metrics of three different models following Test IIIb. The first model has identical schedulers for both training with and without the pseudo-labelled samples while the second and third models have distinctly different schedulers when training including the pseudo-labelled samples. These employ decreased initial and final values for the learning rates schedulers.

Table 5.10: Evaluation of training with two separate exponential learning rate decay schedulers with different initial and final values. Trained with α-balanced focal loss.

First scheduler	Second scheduler	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
$10^{-4} \rightarrow 10^{-6}$	$10^{-4} \rightarrow 10^{-6}$	0.93935	0.82153	0.86695	0.84122	0.88264	0.78169/ 0.69362/ **0.98928**
	$10^{-5} \rightarrow 10^{-6}$	0.94090	0.84404	0.84858	0.84468	0.89748	0.81081/ 0.74419/ 0.97712
	$10^{-5} \rightarrow 10^{-7}$	**0.95101**	**0.87460**	**0.88252**	**0.87787**	**0.91529**	**0.86853/ 0.77533/ 0.97994**

The same concept applied to a step wise learning rate scheduler can be seen in Appendix B.3.

Test IIIc: Image aspect ratio

Following Test IIIc, the effect following two different pre-processing techniques in a model with expanded architecture can be seen in Table 5.11.

Table 5.11: Performance of expanded model, trained with exponential learning rate decay scheduler with two different image preprocessing techniques. Trained with learning rates $10^{-4} \rightarrow 10^{-6}$ and $10^{-5} \rightarrow 10^{-7}$.

Aspect ratio	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
Distorted	**0.94751**	0.85607	**0.88773**	**0.87081**	0.90251	**0.86864**/ 0.71168/ **0.98788**
Preserved	0.93974	**0.85713**	0.81882	0.82848	**0.90893**	0.79105/ **0.81212**/ 0.96821

Test IIId: Augmentation & Imbalanced data

The effects of changing augmentation hyperparameters can be seen in Table 5.12.

Table 5.12: Evaluation metrics of models with different augmentation settings. F, R, T, Z, and C stands for Random Flip, Random Rotation, Random Translation and Random Contrast respectively. H & V means horizontal and vertical flipping while h+w stands for height and width. Hyperparameters changed from the initial augmentation settings are marked in bold.

Model configuration	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
F:H&V R:0.4, reflect T: h+w factor=0.2 Z:0.3 C:0.4	0.94984	0.86026	0.87783	0.86875	0.90827	0.84211/ 0.75309/ 0.98559
F:H&V R:0.4, reflect T:h+w factor=0.2 Z:0.3 C:**1.0**	0.95568	0.86969	0.89411	0.88112	0.91532	0.84047/ 0.77824/ **0.99037**
F:H&V R:**0.8**, reflect T:h+w factor=0.2 Z:0.3 C:**1.0**	0.95373	0.87132	**0.90221**	**0.88570**	0.91282	0.85058/ 0.85058/ 0.98744
F:H&V R:**0.8**, reflect T:h+w factor=0.2 Z:**0.6** C:**1.0**	**0.95723**	**0.87774**	0.89224	0.88396	**0.92098**	0.83333/ **0.81140**/ 0.98849
F:H&V R:**0.8**, reflect T:h+w factor=**0.5** Z:**0.6** C:**1.0**	0.94946	0.86720	0.87789	0.87226	0.91115	0.86179/ 0.75848/ 0.98134
F:H&V R:**1.0, wrap** T:h+w factor=0.2 Z:**0.6** C:**1.0**	0.91641	0.80229	0.81737	0.80564	0.86154	**0.86957**/ 0.56311/ 0.97422

Table 5.13 presents performance metrics after training models with an upsampled dataset, following Test IIId.

Table 5.13: Performance of model trained with upsampled dataset using different augmentation settings.

Model configuration	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
F:H&V R:**0.8**, reflect T: h+w factor=0.2 Z:**0.6** C:**1.0**	**0.92885**	**0.82678**	0.84179	**0.83334**	**0.87946**	**0.88000**/ **0.62637**/ 0.97396
F:H&V R:**1.0, wrap** T: h+w factor=0.2 Z:**0.6** C:**1.0**	0.90241	0.76912	**0.85756**	0.80474	0.83271	0.82308/ 0.49858/ **0.98571**

Test IIIe: L1/L2-regularisation with expanded model architecture

Optimisation of L1/L2 kernel values for elastic net regularisation and its impact of model performance, related to Test IIIe can be seen in Table 5.14.

Table 5.14: Performance of models with added L1 and L2 regularisation. First model represents model with no regularisation.

L1	L2	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
-	-	**0.94518**	**0.86552**	0.86464	**0.86508**	**0.90861**	**0.87500**/ **0.74583**/ 0.97571
10^{-6}	10^{-5}	0.92846	0.80117	0.83434	0.81224	0.86859	0.74748/ 0.67633/ 0.97969
$5*10^{-6}$	$5*10^{-5}$	0.93818	0.83131	**0.87478**	0.85176	0.88481	0.84109/ 0.66794/ **0.98489**
$5*10^{-5}$	$5*10^{-5}$	0.93468	0.82150	0.82234	0.82094	0.88573	0.80478/ 0.68349/ 0.97622
10^{-5}	10^{-4}	0.93157	0.82521	0.83359	0.82890	0.88231	0.83133/ 0.67249/ 0.97182

5.1.4 Proposed model

The performance for the proposed model can be seen in Table 5.15 with the chosen hyperparameters and techniques being displayed in Table 5.16. Additionally, the confusion matrix is plotted in Figure 5.1 with its corresponding ROC curve in Figure 5.2, using predicted classes, and in Figure A.9, Appendix A.8, using predicted probabilities.

Table 5.15: Performance metrics of proposed model evaluated on the test dataset.

Accuracy	**Precision**	**Recall**	**F1-score**	**ROC AUC**	**Class precision (s/m/t)**
0.95140	0.87256	0.87890	0.87431	0.87430	0.83721/0.79909/0.98138

Table 5.16: Hyperparameters of proposed model design.

Hyperparameter	**Decision**	**Values**
Baseline architecture	InceptionV3	
Fine-tuning top layers of TL model	No	
Batch size		32
Loss function	α–balanced focal cross-entropy loss	$\alpha = 0.25$, $\gamma = 0.2$
Learning rate	Multiple exponential learning decay schedulers	1:$10^{-4} \rightarrow 10^{-6}$, 2:$10^{-5} \rightarrow 10^{-7}$
Optimiser	RMSProp	Default
Dropout layers	Yes, 5 repetitive layers	0.2 $*5$
Model configuration	F, Dr(0.2), D(1028), Dr(0.2), D(256), Dr(0.2), D(128) , Dr(0.2), D(64), Dr(0.2), D(32), D(3)	
Image aspect ratio	Preserved	
Augmentation	Flip (F), rotation(R), translation(T), zoom((Z), contrast(C)	F:H&V R:0.8, reflect T:h+w factor=0.2 Z:0.6 C:1.0
Oversampling minority classes	No	
L1/L2-regularisation	No	

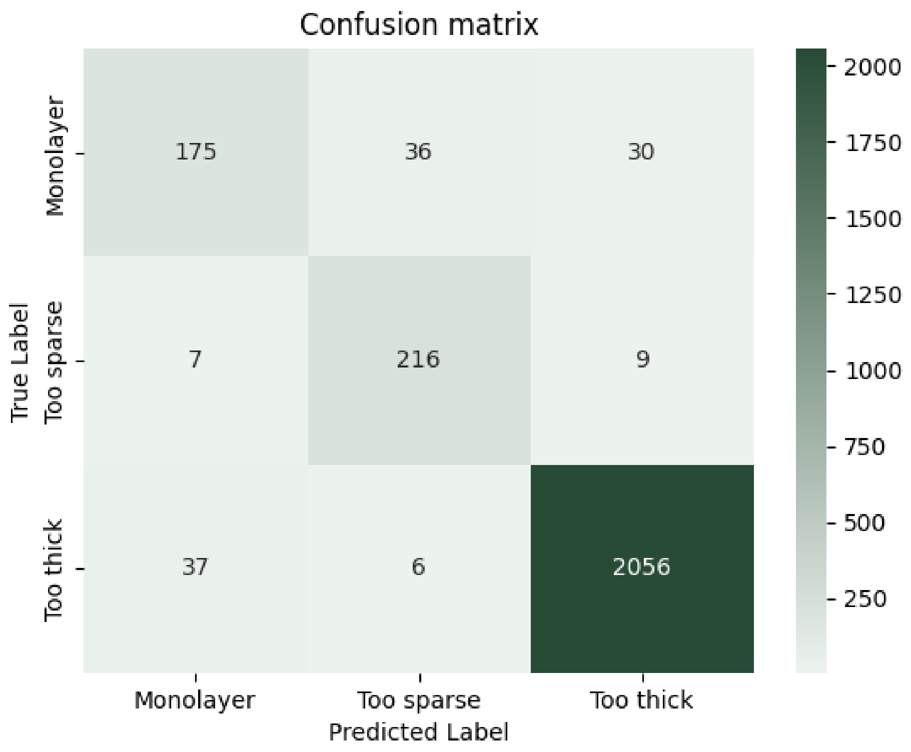


Figure 5.1: Confusion matrix of proposed model.

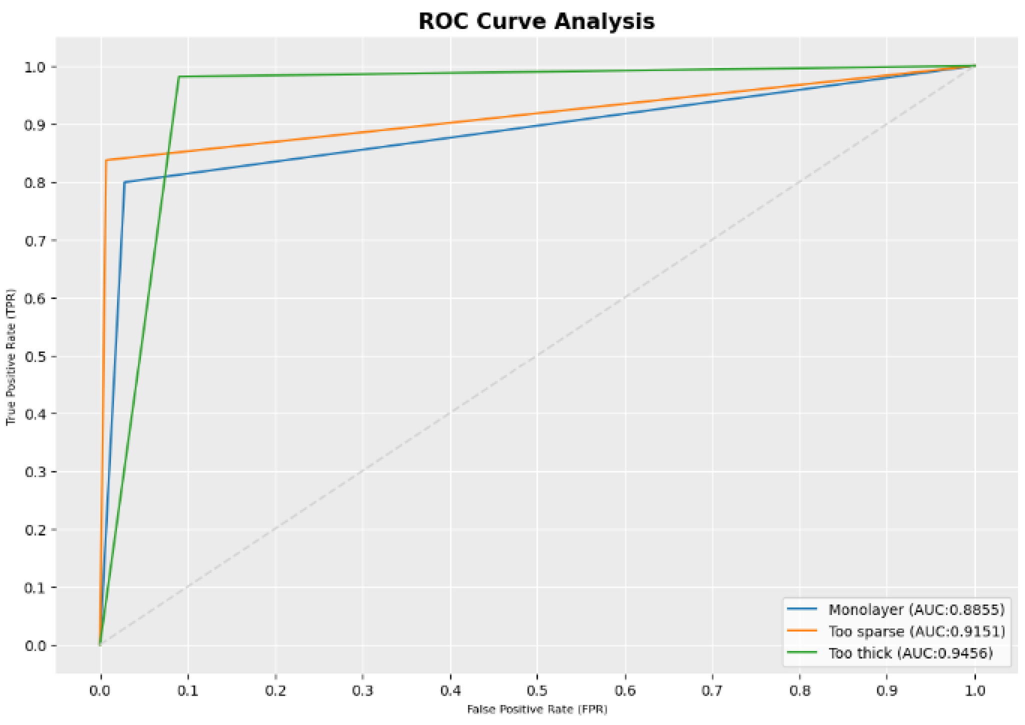


Figure 5.2: ROC curve using predicted classes.

5.2 Locating WBCs

The following section refer to tests performed for attempting to solve the second task of locating white blood cells in avian monolayers.

Selection of model backbone

Evaluation metrics representing models trained with different backbones and freezing bottom layers can be seen in Table 5.17. Similarly, the models trained on all layers at once can be seen in Table 5.18. Table 5.19 on the other hand illustrates the model performance using pre-loaded weights or not.

Table 5.17: Average precision between IoU=0.5 and 0.95 and average precision at IoU=0.5 for YOLOv8 models trained with pre-trained backbones with various depths of feature pyramid networks(fpn) and frozen bottom layers.

Backbone	**Number of parameters**	**Initialised weights**	**Depth of fpn**	**mAP@ [0.5:0.95]**	**mAP@[0.5]**
CSPDarkNet, L	27.11 M	ImageNet	3	0.08821	0.34635
EfficientNetV2,B2	8.77 M		2	0.09037	0.34389
EfficientNetV2,S	20.33 M		2	0.09196	0.34618
ResNet50	23.56 M		2	0.10724	**0.49726**
YOLOv8, XS	1.28 M	COCO	2	0.08008	0.29578
YOLOv8, S	5.09 M		2	0.08552	0.33550
YOLOv8, M	11.87 M		2	**0.11055**	0.41179
YOLOv8, L	19.83 M		3	0.10868	0.41620
YOLOv8, XL	30.97 M		3	0.09627	0.36000

Table 5.18: Average precision between IoU=0.5 and 0.95 and average precision at IoU=0.5 for YOLOV8 models trained with pre-trained backbones with various depths of feature pyramid networks(fpn).

Backbone	Number of parameters	Initialised weights	Depth of fpn	mAP@ [0.5:0.95]	mAP@[0.5]
CSPDarkNet, L	27.11 M	ImageNet	3	0.11078	0.41896
EfficientNetV2,B2	8.77 M		2	0.11562	0.41787
EfficientNetV2,S	20.33 M		2	0.12701	**0.47704**
ResNet50	23.56 M		2	0.12417	0.45901
YOLOv8, M	11.87 M	COCO	2	**0.13117**	0.46912
YOLOv8, L	19.83 M		3	0.12303	0.45397
YOLOv8, XL	30.97 M		3	0.11873	0.43411

Table 5.19: Evaluation metrics of models with the same configurations apart from loading pre-trained weights.

Load pre-trained weights	mAP@[0.5:0.95]	mAP@[0.5]
True	**0.13293**	**0.46977**
False	0.11957	0.43377

Image and batch size

Performance of models trained on specific image and batch size can be seen in Table 5.20.

Table 5.20: Evaluation metrics of models with the varying image and batch size.

Image size	Batch size	mAP@[0.5:0.95]	mAP@[0.5]
640x640	2	0.12809	**0.46644**
640x640	4	**0.13031**	0.45133
800x800	2	0.11411	0.41908
960x960	2	0.10497	0.38484

Loss functions

Table 5.21 shows the performance of a model trained while minimising three different types of losses - CIoU loss, Smooth L1 loss and mean squared error loss.

Table 5.21: Evaluation of three different types of losses.

Loss	mAP@[0.5:0.95]	mAP@[0.5]
CIoU	**0.11949**	**0.43998**
Mean Squared Error	0.11686	0.42199
Smooth L1	0.10558	0.42633

Learning rate

Evaluation metrics of models trained with two different types of learning rate schedulers with varying values can be seen in Table 5.22.

Table 5.22: Values and types of learning rates with their corresponding performance.

Type of learning rate	Values	mAP@ [0.5:0.95]	mAP@[0.5]
Exponential learning rate decay	$10^{-4} \rightarrow 10^{-7}$	0.11045	0.40294
	$10^{-3} \rightarrow 10^{-6}$	0.12809	**0.46644**
Reduced on plateau	Initial: 10^{-3}	**0.13031**	0.45133
	Initial: 10^{-1}	0.00008	0.00029

Augmentation

The effect of applying different augmentation techniques with varying rates can be seen in Table 5.23.

5.2.1 Best performing model

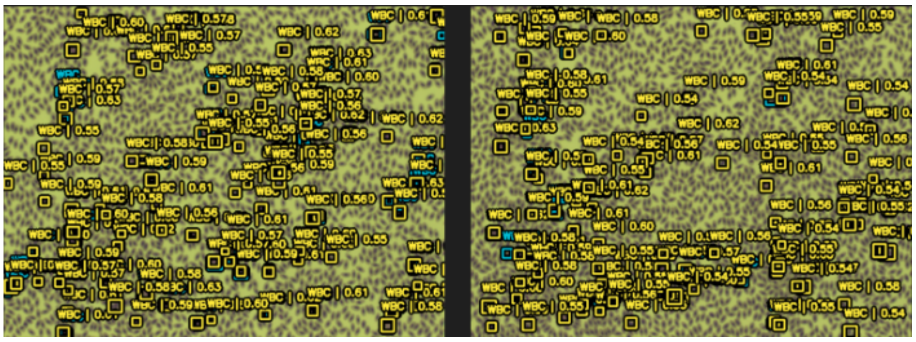
Table 5.24 shows evaluation metrics of the best performing model from previous tests, which performed with general mAP of 0.13407 and mAP@[0.5:0.95] 0.48597 with 50% IoU. The predictions are further visualised along with their true boxes at different IoU and confidence threshold values in Figure 5.3.

Table 5.23: Evaluation metrics of models with different types of augmentation.B stands for brightness, C for contrast, CS for channel shift, F for flip, G for gridmask, H for hue, JR for jittered resize, Sat for saturation and Sharp for sharpness.

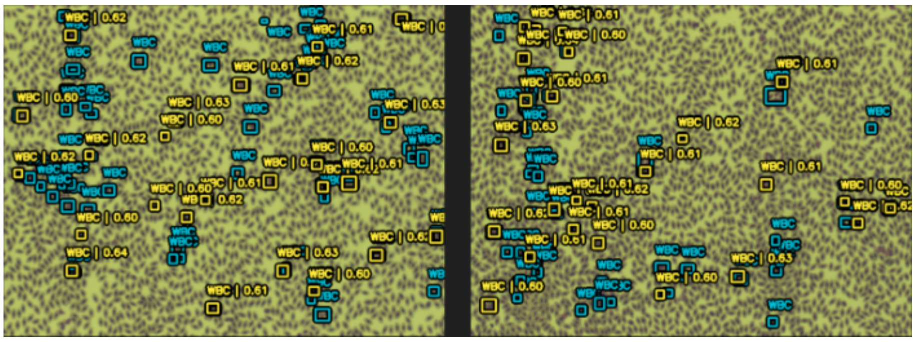
Augmentation	**mAP@[0.5:0.95]**	**mAP@[0.5]**
B:(-0.3, 0.3) **C**:(-0.2, 0.2) **F**:Horizontal **G**:ratio=(0, 0.2), rot=(0.15) **H**:(0, 0.1) **JR**:(1, 1) **Sat**:(0.3, 0.6) **Sharp**:(0.1, 0.5) **Shear**:(0.3, 0.3)	0.00161	0.01239
B:(-0.3, 0.3) **C**:(-0.2, 0.2) **F**:Horizontal **H**:(0, 0.1) **JR**:(1, 1) **Sat**:(0.3, 0.6) **Sharp**:(0.1, 0.5) **Shear**:(0.3, 0.3)	0.01359	0.05318
C:(-0.2, 0.2) **F**:Horizontal **JR**:(1, 1) **Sat**:(0.3, 0.6) **Sharp**:(0.1, 0.5)	0.07648	0.31821
B:(-0.3, 0.3) **C**:(-0.2, 0.2) **F**:Horizontal **H**:(0, 0.1) **JR**:(1, 1) **Sat**:(0.3, 0.6) **Sharp**:(0.1, 0.5)	0.08759	0.33682
B:(-0.3, 0.3) **CS**:(0, 0.2) **C**:(-0.2, 0.2) **F**:Horizontal **H**:(0, 0.1) **JR**:(1, 1) **Sat**:(0.3, 0.6) **Sharp**:(0.1, 0.5)	0.10462	0.35320
B:(-0.3, 0.3) **CS**:(0, 0.2) **C**:(-0.2, 0.2) **F**:Horizontal **H**:(0, 0.1) **JR**:(0.8, 1) **Sat**:(0.3, 0.6) **Sharp**:(0.1, 0.5)	0.12616	0.45384
B:(-0.3, 0.3) **CS**:(0, 0.2) **C**:(-0.2, 0.2) **F**:Horizontal **H**:(0, 0.1) **JR**:(0.8, 1) **Sat**:(0.3, 0.6)	**0.13407**	**0.48597**

Table 5.24: Values and types of learning rates with their corresponding performance.

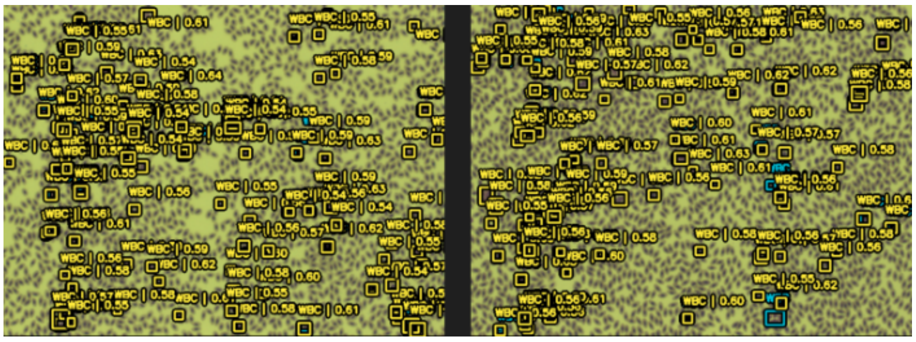
IoU	**Confidence**	**Precision**	**Recall**
50%	30%	0.32689	0.83713
	60%	0.66317	0.38530
30%	30%	0.37406	0.95913
	60%	0.74462	0.43424



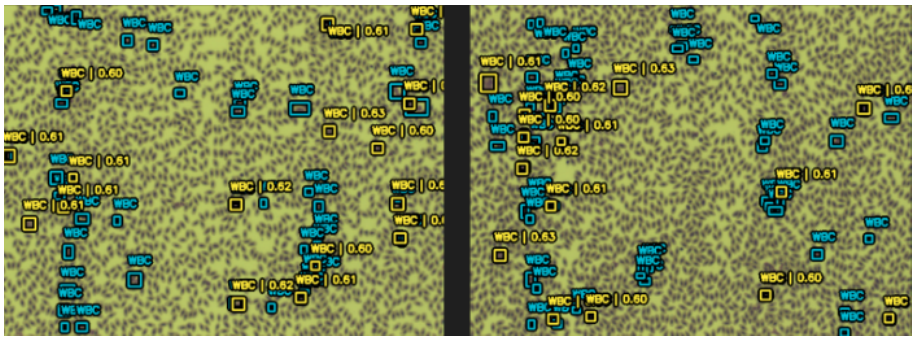
a) IoU = 0.3, c = 0.2



b) IoU = 0.3, c = 0.6



c) IoU = 0.5, c = 0.2



d) IoU = 0.5, c = 0.6

Figure 5.3: Localisation of predicted and true WBC boxes with different confidence (c) and IoU thresholds.

Chapter 6
Discussion

6.1 Finding monolayers

6.1.1 Baseline architecture

Generally, all models from Test Ia, presented in Table 5.1, have high values of accuracy. For most models, the ROC AUC score is also considerably high. However, considering that the distribution is 89.3% of non-monolayers to 10.7% of monolayers, the dataset is imbalanced. This means that the model can for example consistently predict the dominating class and still receive high accuracy without finding any of the monolayers. This is the case for EfficientNetB0, which received a class precision of 0.0% for the monolayer class while still having an accuracy of 88.7%. This architecture also reports the lowest ROC AUC score, just slightly higher than 50%, which means that its ability to discriminate between the two classes is weak. Similar findings are discovered for the EfficientNetB1. In contrast, the rest of the models have higher performance across the evaluation metrics. Considering the enhancement of model performance in larger networks, it signifies that a more complex architecture is necessary and that an increased number of parameters is beneficial.

This however is not supported while comparing the performance of ResNet50 to InceptionV3. ResNet50 contains more parameters compared to InceptionV3, 25.6 millions to 23.8 millions, but still reports lower performance. Furthermore, InceptionV3 has almost similar performance as InceptionResNetV2 while having less than half of the number of parameters. This might be explained by the architecture of InceptionV3. As previously explained, the network architecture is designed to contain inception modules. These inception modules utilise multiple filter sizes and allow features in different scales to be found in parallel with each other. By using parallel branches and multiple filter sizes, the dimensionality can be reduced. Meanwhile, ResNet50 contains residual connections, which addresses the vanishing gradient problem by creating skips in the layers and thereby creating a shortcut path for the data during backpropagation. This allows for deeper training, making it possible to efficiently create and learn more complex architectures without losing the gradient. As ResNet50 is slightly deeper, with slightly more parameters, it is also more prone to overfit, which might explain the loss in performance. The results hence suggest that InceptionV3 might be a better fit to the specific dataset, but as they were all trained without much hyperparameter tuning, it is possible that they eventually could perform equally well with continued optimisation.

InceptionResNetV2 seems to have an overall better performance for the dataset. This is probably due to the architecture, which is designed to incorporate both the efficient inception modules from InceptionV3 and the residual connections from ResNet50, allowing for deeper networks. This however presents itself by a substantial increase in the number of parameters, which in turn affects the computational expense. Considering that the performance was only marginally enhanced using this architecture, it was therefore deemed more useful to lessen the demand on memory. Thus, InceptionV3 was chosen to be used for further optimisation of the task.

As the networks were trained without any hyperparameter tuning and somewhat arbitrary augmentation, InceptionV3, see Appendix A.3, was overfitted seemingly quickly. This highlights the need for optimisation. Furthermore, although InceptionV3 received a relatively high ROC AUC score, the plotted ROC curve, visualising the trade-off between sensitivity and 1-specificity, shows that the model still struggles with correctly classifying the samples. This is particularly applicable when it comes to the false positive rate.

6.1.2 Fine-tuning transfer learning models

Following the results in Table 5.4, fine-tuning the transfer learning models show a small improvement in model performance when using a batch size of 32, but this enhancement is very slight. The same findings can be found when comparing the first model of Table 5.2 to the fine-tuned version in Table 5.3. One reason as to why fine-tuning seems to be less effective is that the domain of images differ significantly. The pre-trained learning models are initialised on ImageNet, which mostly contains larger objects such as animals, plants and vehicles. This implies that the pretrained weights are less relevant to learning blood samples. To ensure if this is true, it could be of interest to test the same model configurations but without preloaded weights - if they perform equally well, the pre-trained weights are not particularly applicable when learning the dataset, which in turn makes fine-tuning only top layers less useful. When testing fine-tuning using a batch size of 16, the two models seem to perform equally well. However, this could be a consequence of the batch size being too small and the learning being more unstable, leading to inconsistent results. Furthermore, it could mean that the set learning rates were less appropriate for the batch size. Ultimately, as the fine-tuning only gave a moderate improvement while making the training time much longer, the fine-tuning was henceforth excluded. Nevertheless, as the testing of fine-tuning the top layers to the extent of different learning rates with different batch sizes, it is possible that for example freezing a different amount of layers could be more beneficial to model improvement.

6.1.3 Batch sizes

The matter of batch size was tested in both Test Ic and Test IIa - testing across three different values of 16, 32 and 64. When using a smaller dropout rate, there is a slight increase in ROC AUC score when training with the batch size 32 compared to training a model with batch size 16. Generally, apart from the smaller dropout rate, the batch size of 16 shows lower performance compared to the larger batch size. The same trend can also be observed in Test IIa. For the two presented configurations, a batch size of 32 yields higher accuracy values compared to a batch size of 16 - 88.6% opposed to 90.9% and 88.4% to 90.0% respectively. Similar differences can also be seen in the other measured metrics. Among those, recall seems to improve especially with the larger batch size, with 76.0% compared to 81.9% for the first model and 75.7% to 82.8% in the second. The fact that the accuracy is higher for a smaller dropout rate could possibly be explained to models usually training in a more erratic behaviour compared to a larger batch size. Smaller batch sizes can potentially introduce variance in the gradient estimates. This is due to the model making estimates based on a fewer number of samples, which makes it more vulnerable to random fluctuations in the data. By extension, the updates of the model parameters can hence become noisy. Too small batch size can therefore more easily overfit to noise rather than fit to the actual pattern in the data. Contrary to this, larger batch sizes average over a larger amount of samples, which makes fluctuations affect optimisation less. This usually provides more reliable and smooth gradient estimates and faster convergence. A larger batch size allows therefore for a more representative estimation of the true gradient and better model generalisation. However, this seems to be true only to a certain extent, as the batch size of 64 performs worse. This might be due to larger batches having a risk of getting stuck in local minimas or in saddle points, providing overly confident models with poor accuracy. As they contain more variance and less frequent model updates, larger batch sizes tend to smooth out variations in the training data which might contain important details. Hence, performance will worsen. Thus, 32 seemed to be the most optimal batch size out of the tested alternatives for the data.

6.1.4 Loss functions

The choice of loss functions is vital as the optimisation aims to minimise it and is therefore directly related to model performance. The investigated losses have differences in how they calculate deviations in the predictions to the true labels as well as in how they penalise faulty predictions. This also results in varying model performance but as seen in Table 5.7, they all perform relatively well. However, they favour different metrics. In terms of accuracy and precision, both cross-entropy loss and α-balanced focal cross-entropy loss outperform hinge loss. The former have metrics above 92% and 80% respectively while the latter is slightly behind with 91% and 77%. This might be explained by the differences in their computation of penalty. Hinge loss enforces a strict margin between classes with a hyperplane forming a decision boundary. Incorrect predictions will be placed on the wrong side of this decision boundary and thereby penalised based on the margin between the predicted class and the true class, increasing linearly with the margin. That way, it encourages the maximal margin to the decision boundary and thereby separate positive and negative instances. Contrary to this, cross-entropy loss has a soft margin, allowing for a more flexible penalisation which is useful when instances are hard to separate from each other and in need of a non-linear decision boundary. Worth noticing is that hinge loss is primarily based on the class predictions as opposed to cross-entropy loss, and by extension also focal cross-entropy loss, which is designed to optimise the predicted probabilities. Since some instances are seemingly hard to classify, especially between the monolayer class and the 'too thick' class, it can mean that the probabilities for the two classes are similar to each other. A penalty using probabilities might therefore be more representative as small differences still can result in meaningful losses, effectively allowing the model to learn more details and recognise smaller variations.

Despite having lower precision, hinge loss has the best performance of the three when it comes to recall. As previously mentioned, recall and precision are often a trade-off, and a harsher focus for precision will usually mean a lower value of recall. As hinge loss might have difficulty in separating the classes well, the margin might be wider. This will allow more instances to be correctly classified. Moreover, cross-entropy loss has, although lower, similar recall value while focal loss is presenting an even lower value. Focal loss puts emphasis on the hard-to-classify samples, and thereby has its strength in classifying challenges. This can for example be seen in the class precision for the monolayers, where it has a distinctly higher value compared to the other loss functions. As focal loss focuses on difficult samples by down-weighting well-classified samples, it seems to have a trade-off with recall even here. In addition, the challenging instances are potentially not distributed across all samples, but focused in the monolayer class. The focus of those instances might then lead to a less-generalised model across the rest of the classes. This can be supported by the class precision for the thick samples, which is the majority class, being the lowest while using this type of loss. Hence, focal loss seems to be well suited for minority classes containing a lot of difficult samples, for example monolayers, if finding all the instances is less important.

In the exploration that followed, focal loss was attempted to be optimised with the purpose of trying to increase general recall while keeping the same class precision. This was further necessary since the default values are based on experimental studies which might not be suitable for this dataset. The down-weighting of well-classified samples is controlled by the modulation factor α, and a too high value can lead to an extreme focus of precision of challenging instances.

Values represented in Table B.3 show that the default values still provide the highest class precision for monolayers and give the highest ROC AUC score among the tested models. This shows that the default values for α and γ provide the most reliable performance when focusing on correctly classifying monolayers. Using the suggestion of inverse frequency as α did also not provide any improvements. In terms of accuracy and general precision, α-value of 0.05 seemed to perform the best with the default γ-value, as seen in Figure B.1. This is to be expected as the well-classified samples are less down-weighted and the model can therefore increase the overall confidence in its predictions. This does however decrease precision for the monolayer class. On the other hand, recall is also increased from 79.6% to 84.3%, which could mean that it is still more beneficial to the model performance as it may also include more instances from the monolayer class. Generally, precision seems to decrease with increased γ, which can be explained by the down-weighting of easy samples being more aggressive. However, this trend doesn't seem to be consistent in terms of monolayer precision, but as the models are also differing in α it is difficult to say if the difference is due to the gamma-value or simply the combination of hyperparameters. When comparing gamma values for identical α, when α is 0.005 and γ is either 2 or 5, the value 5 gives a better overall performance in all metrics. This suggests that a lower α and a higher γ provides an enhanced balance by reducing the emphasis but still allowing for a fast decrease in loss for easy instances. Nonetheless, further explorations are needed in order to conclude whether there are more suitable configurations than the default values for the focal loss.

6.1.5 Learning rates

Learning rate was tested at several stages during the process of optimisation. Firstly, in Test Ia, two constant learning rates, see Table 5.2, were applied while training the binary test. Comparing the models of otherwise identical model configurations, in this case while having the same top layers, the higher rate of 10^{-6} performs better compared to 10^{-7}. The same findings are reported in Table 5.3 in fine-tuned models. As low learning rate means that the parameters are updated slowly and taking smaller steps towards the minima during data training, it can mean that the rate of 10^{-7} simply has not converged during the 50 epochs that it was trained for. Most often, low learning rates are good to use in the later stages of training, as it helps to work against oscillations often found when using higher learning rates, by learning in a more stable way. On the other hand, the lower performance might also be due to the optimisation getting stuck in a local minima. For this specific dataset, a higher initial learning rate is beneficial for training.

As the constant values seemed to converge training at an early stage, more advanced learning shapes were introduced. Firstly, the step wise scheduler decreased the constant initial rate at a specific epoch. As seen in Table 5.6, the first two models started at a learning rate of 10^{-2}, lowered at different epochs, while the third was initialised with 10^{-4}. The latter increased performance across all metrics, and resulted in the highest accuracy. A particularly noteworthy improvement was a class precision for monolayers of 68% while reporting a high recall value, which for the former two models was just above 57%. This improvement was also reflected in the superior ROC AUC score of the three. Higher initial learning rate hence seems to only be beneficial to a certain extent, and that an initial learning rate still needs to be relatively small in order to not miss important patterns in the data.

Secondly, a scheduler using exponential learning rate decay was implemented and tested across a range of initial and final values. Among the models using exponential learning rate decay, the best performing model had a slower descending curve across a smaller range, which points to smaller changes providing more stable training and by extension smoother convergence. A smaller range further allows for the optimal learning rate to work during a longer time frame compared to a larger range, which will improve the model performance. The need for a lower initial learning rate is further proven with the model starting at 10^{-3} performing better than the model initialised on 10^{-2}.

Testing on the pseudo-labelled dataset, reported in Table 5.9, the stepwise scheduler was extended with a linear learning rate warm-up. This was meant to add stability and increase learning during early training as previous models had contained drastic jumps in losses during early epochs. However, apart from lowered performance, likely due to not being able to learn lower hierarchical features because of the low initial learning rate, it also increased the loss by tenfold. The linear-warm up was hence deemed as not successful.

Generally, the schedulers using exponential learning rate decay seem to report more consistent results compared to the stepwise scheduler, which could be explained by the stepwise scheduler needing more hyperparameter tuning. This can further be seen in Table 5.9, where the learning rates are specified in additional steps and resulting in better performance. Given the flexibility of a stepwise scheduler, it is therefore likely that it eventually could outperform an exponential decay learning rate scheduler, if optimised correctly. It could therefore be of interest to for example implement a decreasing learning rate when reaching a plateau, for a more natural learning controlled by the data instead of being predetermined. Moreover, the more consistent results could also be explained by the learning rate curve over epochs being more smooth, thereby manipulating the data in a more continuous manner. The exponential learning rate decay scheduler also gives a generally better model performance, despite the scheduler having the best monolayer precision, as they perform overall well over all classes. This is supported by a superior ROC AUC score. Nonetheless, compared to a constant learning rate, scheduling the learning rate and decreasing it over a limited range of learning rates allows the network to learn better by making it possible to learn higher-level features at a slower speed - thereby optimising the speed to reach the minima of the loss function.

Specific to training using semi-supervised learning was the possibility of using multiple schedulers, presented in Table 5.10 and Table B.4. As mentioned, the idea was that the network had already learned lower level features and that focus could be put on details, thereby requiring lower learning rates. This hypothesis could be supported by the results seen in Table 5.10providing clear improvements on all fronts. Besides reporting enhancement by adding slower rates, the slowest tested rate performed best. This signified that enough learning of the lower features could be achieved during the first training session to be able to improve learning using lower rates in a second scheduler. Models presented in Table B.4 supported the enhancement using a second scheduler, but seemed more sensitive to the values set in the hyperparameters, consistent with previous discussion.

6.1.6 Optimiser

The optimiser to use for the dataset was evaluated in Test IIb. Firstly, the ADAM algorithm was evaluated based on the suggestion of increasing its epsilon hyperparameter value from the default 10^{-7} to 0.1. When training for 50 epochs, the default value performs better across all evaluation metrics, as seen in Table 5.5. This is particularly noticeable on the recall value where the model using the increased value has a recall of 46.5% while the default value has 81.8%. However, as previously stated, a larger epsilon value will lead to smaller weight updates, which thereby makes the model converge slower. This can further be confirmed by looking at the training history, see Appendix A.5. When training for 200 epochs, the recall value dramatically improves to 65.0%. Improvements can also be seen for the other metrics.

However, it would most likely require even longer training to be able to reach similar performance as to that of the default value. Using a larger epsilon for the ADAM optimiser could therefore not be justified. It's possible that a larger epsilon value than the default could improve the optimisation overall if the increase was smaller than 0.1. As it interacts with other hyperparameters, it is also possible that another model configuration would interact differently then the ones tested and hence yield different results. This was however not tested and will be left for future studies.

Comparing the RMSProp and ADAM algorithm initialised on their default values of 10^{-7}, the RMSProp performed noticeably better on all metrics, reaching an F1-score of 83.6% compared to 79.9% in the model optimised with ADAM. This difference may be due to the difference in their adaptive learning rate mechanism. RMSProp adapts the learning rate solely based on the magnitude of gradients as opposed to ADAM, which bases the mechanism on both the first and second moments of gradients. This makes the adaptive learning rate more complex for the ADAM algorithm, and the learning rates will be adjusted more aggressively, especially considering that it also includes bias correction. As a consequence, ADAM's additional complexity can cause instabilities in the optimisation when in presence of variations such as noisy or sparse gradients. Noise and sparsity is expected in the created datasets, as images contain variations in for example contrast and density. On the contrary, RMSProp, due to its simplicity, is usually more robust to the variations present in the dataset. Furthermore, as ADAM is sensitive to the learning rate settings determined in the model design, there is a possibility that this type of optimisation could perform better with specific hyperparameter tuning compared to the RMSProp. However, as this dataset presents itself as sensitive to the learning rate, stable learning rate updates seem vital for better model performance.

6.1.7 Dropout and extended model

Regularisation like dropout layers are usually added to prevent overfitting of the networks. As the initial InceptionV3 model quickly failed to generalise, as seen in Appendix A.3, it seemed likely that the model was in need of higher regularisation. This would make the model generalise better and hence make the model more robust to various representations of the data. By dropping out random neurons during training, the model is encouraged to learn the different instances independently of each other as well as learn the important features as opposed to noise in the training dataset. Adding a dropout layer, as in the model presented in Table 5.2, had little effect on accuracy, but increases the ROC AUC score and hence indicates an improvement of the model. Introducing some form of regularisation thereby implies that the model can be more prevented from learning inaccurate patterns for example from noise. Furthermore, looking at the training history in Appendix A.4, the curves are more smooth and it seems that the overfit of the model is less extreme. Additionally, Table 5.2 also presents an interesting finding in the placement of the dropout layer. Comparing the first and second model, it can be seen that the second model contains an added dense layer before the dropout layer. This addition results in lower performance, both in terms of accuracy but also in ROC AUC score relating to the models specificity and sensitivity. Apart from implying that an added dense layer increases the risk of overfitting as a consequence to an increased number of parameters, it also seems to illustrate that the placement of dropout matters. Early dropout seems to be more effective.

As the dropout rate determines the proportion of neurons to randomly drop during training, it is essential to balance this rate well. When comparing a higher rate of dropout in fine-tuned models presented in Table 5.3, the models using a batch size 32 can be seen to have an improvement with a higher value of dropout, especially when it comes to the ROC AUC score. This implies that the model learns to better distinguish between the two classes with a higher dropout rate. For the model with batch size 16, the accuracy doesn't increase when increasing the dropout rate. However, the ROC AUC score still increases with higher dropout rate, indicating an improvement of the model. On the other hand, this does not necessarily mean that a higher rate is always better. If the dropout rate is set too high, the model performance can decrease. This can be seen for a model trained with the extended model architecture in Test IIe, seen in Table 5.8. The model repeatedly drops 80% of its neurons, albeit randomly chosen, which is a behaviour that results in a ten percentage point reduction in accuracy. Even more importantly, the other evaluation metrics perform terribly and the class precisions for class minorities are non-existent. The model performance decreases as the dropout rate is too high. This will cause the model to underfit and reduce its effectiveness and capacity. Most likely, the repeated high dropout rate caused too little information in the data. As the rate is repeated four times at a rate of 80%, the retained information will be $0.2^4 = 0.0016$, which means that only 0.16% of the original information is left, which understandably is too little to learn any meaningful patterns. Instead, repeated dropout layers seem to work better with a rate of 0.2. This would mean that 80% of the connections are kept between each layer, resulting in $(1 - 0.2)^4 = 40.96\%$ of retained connections during each run, which is a drastically improved amount of data compared to the previous rate. Thus, increasing the dropout rate will lead to more generalisation and in turn better performance, but the rate needs to be set in relation to having an adequate amount of connections between neurons remaining.

There are benefits to repeated dropout layers, as this acts as a further form of regularisation as it repeatedly drops random neurons during multiple points in the network. This means that repeating dropout layers will increase generalisation. This can be seen in Appendix A.6, where the losses are shown in a model with 0.8 dropout layer compared to a model with a repeated dropout layer with dropout rate 0.2. Even though the total dropout is lower with the repeating model compared to the single dropout model, the loss is more controlled. This implies that the model using repeated dropout blocks responds better to generalisation. However, this might also be due to, or in combination with, the expanded model architecture. Having added dense layers increases the model capacity, which allows for more complex patterns to be captured. By using repeated dropout layers on top, the regularisation can be designed to be more flexible, learning more advanced data while limiting the overfit. This is further supported by, if the dropout rate is reasonable, the larger architectures working better - for example, having six dense layers seem to perform slightly better compared to four dense layers.

6.1.8 Image aspect ratio

The choice of preserving or distorting the image aspect ratio of the samples were evaluated in Test IIIc. It can be seen from Table 5.11 that the two techniques had different strengths in their performance. A distorted aspect ratio gave a higher accuracy of the validation dataset, 94.75% compared to 93.98% for the preserved aspect ratio. The same trend was observed in the recall value, presenting at 88.77% as opposed to 81.88%. However, the preserved image ratio seemed to perform better in terms of precision. Although the general precision for all classes were relatively equal for both pre-processing techniques, the precision for the 'monolayer' class was significantly improved from 71.17% for the distorted image aspect ratio to 81.21% for the preserved aspect ratio. This suggests that maintaining the image aspect ratio could be important for enhancing the network's ability to correctly identify cell monolayers. The overall better performance when using a distorted image aspect ratio could be due to the amount of information the technique keeps. Contrary to distorting the aspect ratio, the preserved aspect ratio loses information as they are cropped to fit the desired image dimensions. In the carried out scenario, 25% of the pixels were removed, which also means an information loss of around 25%. As this is done on every image in the dataset, this will lead to a distinct reduction of data, which can affect the features it is capable of extracting. In addition, deep learning networks rely heavily on having an adequate amount of data in order to properly learn the features specific to the dataset. Reducing data, especially in an already relatively small dataset, can decrease the network's ability to generalise well. On the other hand, distorting the image aspect ratio can in itself also result in loss of information. Altering proportions of an image may lead to distortion and deformation of features within the sample, for example cell boundaries and cell shapes, which in extension can make important details less distinguishable. This can make regions where the spatial relationship is important harder to recognise and potentially explain why images containing monolayers had a higher amount of misclassifications and lower class precision; the distorted image ratio affected crucial features in monolayers, thereby making them harder to identify. As the preserved aspect ratio kept the integrity of the features and their spatial relationships, the precision for monolayers could increase with the cost of overall performance.

6.1.9 Augmentation

The number of available augmentation techniques were somewhat limited due to the choice of performing it through sequential layers in Keras. Yet, the importance of augmentation could be seen in Table 5.12. Compared to the initial, somewhat mild, values presented in Table 4.1, increasing the values of the available augmentation yielded higher metrics and enhanced generalisation. Increased contrast in particular seemed to improve the model. This is probably due to variations in the images as they vary in both density and lighting; some instances naturally have higher contrast. Increasing the possibility of presenting this type of contrast will hence give the model an opportunity to learn and distinguish those features. Moreover, a higher value of random zoom increased the class precision for monolayers to 81.1%. This suggests that zooming on images might present the model with images containing more distinguishable features specific to monolayers, allowing the network to learn and differentiate to other classes. Although harsher augmentation allows for generalisation, an increased translation decreases the evaluation metrics. This can also be seen in a too high value of rotation. This might be due to loss of information that rotation and translation brings. The methods fill missing pixels after transformation, but in some cases these can contain important features. Missing these features will hence worsen model performance. Generally, the network improves with increased data diversity.

6.1.10 Class imbalance

Comparing model performances with identical configurations between the normal dataset in Table 5.12 to the oversampled dataset in Table 5.13 , it can be seen that oversampling does not seem to be helpful for improving model learning. In fact, both models performed worse across the majority of the metrics when using an upsampled dataset. This suggests that the models might become overfitted and fail to understand meaningful patterns in the data. The used oversampling method does not add any information, as it simply increases the occurrence of minority class samples by presenting them more often along with smaller variations. However, merely presenting instances of minority classes more frequently might not be enough to understand the underlying pattern in the class imbalance, especially if the data is too complex. The network might instead be in need of alternative techniques to learn the details of the minority classes. Additionally, only one method of handling imbalanced data was investigated. In future studies it could be of interest to explore different strategies, such as Synthetic Minority Oversampling Technique (SMOTE). SMOTE might be a better alternative as it creates synthetic diversity by interpolating between existing samples using the nearest neighbours in the feature space of the minority class. On the other hand, SMOTE imposes a risk of introducing noise that does not originally exist in the samples, which in turn will decrease generalisation. This risk is less in the performed technique, as it retains original data to a higher extent. It is also possible that handling imbalance data in this way is simply ineffective, as suggested by previously mentioned studies.

6.1.11 L1/L2-regularisation

An addition of regularisation was tested in Test IIe shown in Table 5.14, which is ranged in order of increased regularisation strength. As seen in Appendix A.7, increasing regularisation strength provides a more smooth loss curve. This suggests that the learning is more consistent and converging efficiently. It further implies that the regularisation properly prevents overfitting without affecting the performance too much. Looking at the values presented in Table 5.14, the model without regularisation has the highest performance. This is however to be expected as it penalises the model, creating a less complex model and suppresses the information to some extent. The last three models are however not differing much relative to their regularisation strength, apart from in the recall value. All the other metrics are seemingly similar. This can indicate that harsher regularisation could be necessary, but as the loss function is seemingly smooth it might not need additional regularisation. The model with the smaller regularisation strength, has slightly lower performance, which is most noticeable in recall as well as precision. This could be due to L1 and L2 having different effects on the model, and that the suggested combination is less efficient. For example L1 regularisation favours feature selection and features important for classification and a higher value could possibly increase performance for minority classes. This is supported by a higher monolayer precision with an increased L1-value. Nevertheless, as the differences are small, no conclusions can be drawn of specific regularisation values.

6.1.12 Binary, multiclass and pseudo-labelled multiclass dataset

While trained with a constant learning rate of 10^{-6} , the difference between the binary dataset and the multiclass dataset are reported slightly higher in accuracy for the binary dataset but higher ROC AUC value in the multiclass dataset. The loss in accuracy, albeit small as it stands 91.1% in the binary dataset, shown in Table 5.2, to 90.9% in Table 5.4, could be due to a multiclass problem being more complex to solve. Having multiple classes means that the network needs to learn how to distinguish additional classes, and in turn learn more defined patterns in the data. The model seems, however, more suitable for a multiclass problem, as the ROC AUC score increases from 83.1% to 84.7%. This might be a benefit of the distinct patterns of each class, as the model might generalise less well when containing a larger range of samples in one class; more definition as to what makes up a class seems to increase the model's ability to find and discriminate the correct patterns of each class. Furthermore, the same model configuration applied on the semi-supervised dataset, see Table 5.9, increases the ROC AUC score even further, to 85.1%. The improvement from the multiclass dataset to the pseudo-labelled multiclass dataset can not be expected to be dramatic, as the amount of pseudo-labelled instances are relatively few at 1.3% of the entire dataset. It can however be seen in later tests that models with superior performance are trained on the pseudo-labelled dataset. Using semi-supervised learning, there is a risk that all the previously unlabelled instances are labelled as the majority class according to the present bias, and that the model performs better due to the new data already fitting into what it's previously learnt. The labels were however manually inspected after the labelling and deemed fairly justified. The better performance might be possible because of errors in the annotation of the fully labelled multiclass dataset.

6.1.13 Proposed model

The proposed model was based on techniques and values of hyperparameters with superior performance in performed tests, especially regarding class precision of monolayers and ROC AUC score. These evaluation metrics were chosen as they are highly responsive to changes in model performance and thus, sensitive to the choice of hyperparameter values. Class precision was deemed important as found monolayers are meant to be used for further analysis. A faulty instance increases the risk that an analysis would fail, thereby wasting end-users time. On the contrary, if recall was considered to be more crucial in the model performance, it would most likely mean that a higher amount of images would be classified as monolayer, as recall often comes with trade-off with precision. The risk of failed analyses increases with a larger number of incorrectly classified monolayers, which in turn likely results in even more ineffective time. Hence, the run time for a completed and valid analysis of the entire blood sample would increase. This would make the system less effective for the intended analysis and cause delays in the workflow. Furthermore, considering that blood analysis can relay significant findings for diagnosis of medical conditions and illnesses, this could potentially have an effect on treatment and thereby impact patients' healths. However, it is of course necessary to have an adequate number of correctly classified monolayers to be able to perform the analysis, for example to find the desired amount of WBCs in the blood sample.

The results for the final evaluation, found in Table 5.15, show similar performance as to expectations based on tests on the same hyperparameters on the validation dataset. The test dataset and the validation dataset might contain differences in their dataset as they are not identical, but they seem to have enough similarities to still yield an adequate performance. The model hence seems to generalise well. The generalisation could however be improved using L1/L2-regularisation, as it reported to achieve smoother training. This was however not included in the final model as it also came with a decrease in performance. As seen in Table 5.15, the precision is still the highest for denser images, which was anticipated considering that the data is imbalanced and the model thus has a bias towards that specific class. On the other hand, the bias does not seem to be especially noticeable in its predictions of the misclassified monolayers, as they predict 'too sparse' and 'too thick' almost equally, as seen in the confusion matrix presented in Figure 5.1. However, with the predicted monolayers, it seems that the model has a harder time distinguishing between monolayers and denser images compared to monolayers and sparse images. This could possibly be improved using even harsher and alternative augmentation. Furthermore, even the 'too sparse' have a slightly better class precision than the 'monolayer' class, which probably is due to the more distinguishable patterns in the class; 'too sparse' is also easier for humans to detect compared to finding monolayers. The annotation is based upon the monolayer coordinate range provided by the DC-1 system, and given that this is not yet optimised for avian blood it is possible that it has incorrectly labelled some monolayers as non-monolayers. Hence, the proposed model might classify actual monolayers as monolayers but as the label might be wrong, this will be considered a faulty classification and assumed as decreased performance. The true monolayer classification might therefore be higher than presented, and needs to be confirmed with an expert in the area. The ROC curve seen in Figure 5.2 reports similar findings - 'too thick' has an almost perfect true positive rate while the other two classes are located slightly below them, meaning that the dense class is easier to distinguish. However, compared to the initial ROC curve using the baseline architecture, as seen in Appendix A.3, the performance has improved significantly. All in all, considering the high reportings of evaluation metrics and the presented confusion matrix and ROC curve, the model seems to, at least to some extent, successfully be able to find monolayers within a range of collected images.

6.2 Locating WBCs

For the second task, it can generally be seen that the data adjusts to the task relatively badly, supported by results in Table 5.17-5.24. Among the initial backbones, YOLOv8 with the size m reports the highest overall average precision at 11.01% while ResNet50 has the highest average precision at intersection of 50%. Overall, the average precision at 50% IoU is significantly higher compared to the AP calculated between thresholds 0.5 to 0.95. This signifies that the models have difficulty in finding precise, or at least adequate, coordinates of the objects. This of course poses a greater challenge compared to just locating it with any overlap, probably due to the objects in question having such a small area compared to the entire image. However, higher overlap would mean better precision of the model and is thus more desirable. Similar results can be found in Table 5.18, where the entire model is being trained at once, as YOLOv8 with size m again has the highest average precision. However, this configuration presents EfficientNetV2-S as the best performing at a 50% IoU threshold. It can also be seen that the data seems to be less modelled with a backbone with fewer number of parameters. At the same time, it also does not require the largest amount of parameters that usually indicate more complex architectures. This might be due to a too simple model not being able to efficiently capture any patterns in the data, which will result in the lower performance that can be seen for example of YOLOv8-XS and S. On the other hand, too complex models can generalise less well, as it seems to be the case with YOLOv8-XL and CSPDarkNet. The decrease in performance could naturally also be caused by how the backbone handles data, as they have slightly different approaches to this. Furthermore, Table 5.19 shows that pre-trained weights are beneficial both in overall average precision but also at a lower intersection threshold. This indicates that the COCO dataset contains some features similar to the ones found in the data, and thus provides some knowledge.

The hypothesis of a larger image size being able to provide easier to detect WBC boxes can not be supported by the results in Table 5.20. Comparing models trained with the same batch size, the smallest size is still the most optimal of the tested sizes. This might be explainable by the architecture of the YOLOv8 detector being designed for that specific image size, and hence is optimised to utilise this size the best. In addition to this, it might point to the resizing technique having difficulty in recreating features found in the original images.
The loss function with results shown in Table 5.21 suggests that the model is better optimised using CIoU loss compared to Mean squared error and L1 smooth loss. CIoU takes the aspect ratio of the bounding boxes into account when calculating the error, which means that the spatial relationship is highly valued. This will of course yield more accurate results for localising WBCs. In contrast, the latter two losses penalise based solely on distance and not on box overlap, indicating that this functionality is important for better performance.
The results of varying the learning rate schedulers are seen in Table 5.22. Similar to the previous task, the model performance seems sensitive to learning rate hyperparameters. For example, decreasing the learning rate when reaching a plateau seems to work well if it is correctly initialised. This is supported by the fact that the model trained with this type of scheduler with initial value of 10-3 yielding the highest result for overall AP. As the learning rate decrease is greatly controlled by the training of the data itself, it can easier get out of local minima, for example, resulting in better training. On the other hand, a higher initial value makes the model perform terribly. Initalised on the same well performing value of 10^{-3}, the exponential learning rate decay scheduler seems to work with similar performance. Smaller initial values than 10^{-3} seem to be less advantageous.

As mentioned earlier, elastic transformations can in some cases be useful for cell segmentation, as it mimics cell deformations that can be present in blood samples. This was tested with the added random shear transformation. However, as seen in Table 5.23, the elastic transformations seemed to have the opposite effect and instead worsen the model performance. This might be due to the value range being less optimal, thus providing exaggerated deformations. It could further be caused by the elastic deformations not working with the current resolution, and that a higher magnification would be necessary for decent results. Another augmentation technique that seemed to worsen the performance was the sharpness layer. The sharpness layer was assumed to be helpful as it introduces variations that are similar to images with lower quality, or if a sharp image for some reason could not be collected. The sharpness layer hence introduces blurry edges as variations. However, the sharpness layer might work in a less effective way, for example by blurring in a non-realistic way or blurring too much, making it more difficult to distinguish individual cells. The grid masks also proved to not be applicable in this problem. More commonly used augmentation techniques such as brightness, contrast and colour distortions seemed to be beneficial.

Considering the performed tests, the task presented itself with many limitations and challenges, and no ideal solution could be found. Looking at Table 5.24, it can be seen that it is possible to find more boxes, as the recall is as high as 95%. This is however presented at a low IoU threshold and a low confidence level. Considering that its respective precision is so low, the model hence creates a great number of boxes, and some of them happen to overlap true boxes. This can be seen in Figure 5.3. With an increased IoU threshold and confidence level, the precision increases but also misses a high amount of boxes. As seen in Figure 5.3d, the predicted boxes seem to be located close to the true boxes, but have difficulties with finding more exact coordinates with a greater confidence. Furthermore, a high amount of incorrect boxes will lead to longer runtime when performing analysis, which means that it might still be more efficient to scan using a larger magnification if the overclassification can not be remedied. The task hence needs further research to find a more optimal solution.

6.3 Limitations and future research

One major limitation of the project is the annotation of the dataset. The labels are mainly based on the monolayer range of each sample received when scanning the blood samples using the DC-1. Although this provided a solid foundation to annotating the dataset, there was a need to manually adjust some instances, which signifies that the analysis this entire project is based on in itself is not fully accurate. While manually adjusting the dataset, the decision was made to not label any images as monolayers even in cases where it seemed warranted. This was done in order to minimise inaccuracies due to lack of expertise in the subject. However, this could also mean that monolayers were inaccurately classified as other classes and thereby confusing the network learning, most likely resulting in worse model performance. Thus, with the purpose of improving performance, expert annotation of the dataset could be both advantageous and necessary. Furthermore, as CNNs in particular thrives on higher amounts of data, and the dataset collected is quite limited in terms of number of monolayers, it could be beneficial to collect a more extensive dataset in the future. It might also be helpful to divide the dataset into more specific classes, which could differentiate between distinctly different images, for example those that were too dense, to images located closely to monolayers - slightly too dense.

Another attempt at improving model performance could be the method of how the monolayers are classified. As of now, this project used the images as they were collected, apart from the mentioned pre-processing. This meant classifying images as monolayers if 49.5% of the image contained a monolayer region. However, it might improve model performance if the images were for example cropped to only contain regions of monolayers and avoid mixing different regions. This could make the patterns in monolayers more distinct and thus easier for the network to learn and distinguish from other classes. In extension, this would also provide a larger amount of monolayers, as quite a few had to be ignored with the set threshold limit. Another limitation was the timeframe and workforce of the project. This meant that not all configurations could be tested and that some tests had to be very limited, which further meant that the most optimal solution might not have been found. Additionally, the system setup provided some memory limitations which meant that certain parameters and configurations had to be abandoned as the system could not allocate enough memory in the GPU to run the training. Additionally, apart from the already mentioned possible improvements, future research should include k-fold cross-validation of the model. Cross-validation was not considered due to time constraints and computational runtime, but would increase model robustness as well as reliability and validity.

For the task of locating WBCs, an improvement could probably be found in greater computational power as this yet again proved to be a limitation. This would allow larger batch sizes in addition to faster training, which in turn would lead to more hyperparameter optimisation being able to be performed. This would also allow alternative architectures such as RetinaNet or single-staged object detection models to be considered. YOLOv8 is a relatively fast and efficient object detection model, but this also comes at a trade-off with loss in performance. It could therefore be possible that the data, with the dense images it contains, is in need of that performance increase and a model with greater ability to distinguish objects in denser images. Furthermore, instance segmentation as suggested in the Literature Review, could provide better results as it is more precise to the boundaries in the objects. This would however require a different kind of annotation than what was available to achieve in this project. Another suggestion for further research would be dividing images into smaller segments and increasing the resolution of each segment with interpolation - thereby creating images that look like images collected with higher optical power. This could further be extended into a combination of the object detection model with for example a neural network implementing super resolution, which allows for upscaling and sharpening without losing any important features. The time frame also set a limitation while creating the dataset, as it only localises and not classifies. For future research it is both beneficial and necessary to annotate the data with more specific classes, as this is a crucial functionality and could provide the model with more assistance in its optimisation. It could also prove advantageous to annotate both WBCs and RBCs, thereby helping the model to distinguish between the two cell types. Worth mentioning is also that considering that it is the secondary objective, it could not be investigated to the same extent as the first aim, especially in combination with the increased run time of the second model.

Chapter 7
Conclusion

While focusing on the model's ability to distinguish between classes and the precision of monolayer classifications, hyperparameters and architecture could be optimised - resulting in a final proposed model aimed to find monolayers in blood samples with an accuracy of 95.1% and overall precision of 87.2% as well as a recall value of 87.9%. This corresponded to a monolayer precision of 79.9%. This was achieved by allowing slower convergence and emphasising correct classification of challenging blood samples. The greatest limitation to this task was correct annotation, which was partly remedied with semi-supervised learning - allowing the network to label part of the data according to learnt knowledge of data-specific features. However, future research includes basing annotation on expert knowledge. Further improvements might be possible to investigate more hyperparameter configurations as well as redefining what is included in a monolayer label, for example by cropping images to only contain regions of monolayer. Overall, the proposed model was successfully able to correctly find and classify monolayers to a certain degree. The same can however not be said about localising the WBCs within the monolayers. While using a YOLOv8 detector combined with a YOLOv8 backbone of moderate size, it was only able to achieve an average precision in the range of 50% to 95% IoU of 13.7%. The initial model could be somewhat improved with the help of optimising hyperparameters and adding harsher augmentation. The average precision reported a higher value of 48.6% with a lower intersection threshold, signifying that the model has difficulties with precise localisation. This was further supported by visualisations of the predicted boxes. Recall could be improved by lowering the confidence level and the IoU threshold, but came with a trade-off with precision. To improve this, it could be beneficial to expand the amount of data and annotate it further with RBCs and WBCs as well as specific types of WBCs. This could potentially help the model to learn better. Future research could also include alternative models that have higher focus on performance, for example single-staged object detectors, and ability to distinguish objects in dense images, for example using RetinaNet. This however demands higher computational power.

References

[1] Lima-Oliveira G, Lippi G, Salvagno G, Pichetch G, Guidi G. Laboratory Diagnostics and Quality of Blood Collection. J Med Biochem. 2015;34(3):288-94.

[2] Hedge R, Prasad K, Hebbar H, Sandhya I. Peripheral blood smear analysis using image processing approach for diagnostic purposes: A review. Biocybernetics and Biomedical Engineering. 2018;38(3):467-80.

[3] Gunčar G, Kukar M, Notar M, Brvar M, Černelč P, Notar M, et al. An application of machine learning to haematological diagnosis. Scientific Reports. 2018;8:411.

[4] Gulati G, Song J, Florea A, Gong J. Purpose and Criteria for Blood Smear Scan, Blood Smear Examination, and Blood Smear Review. Ann Lab Med. 2013;33:1-7.

[5] Ohsaka A. Artificial intelligence (AI) and hematological diseases: establishment of a peripheral blood convolutional neural network (CNN)-based digital morphology analysis system. The Japanese Journal of Clinical Hematology. 2020;61(5):564-9.

[6] Joubert J, Weyers R, Raubenheimer J. Reducing unnecessary blood smear examinations: can Sysmex blood cell analysers help? Medical Technology SA. 2014;28.

[7] Adewoyin AS, Nwogoh B. Peripheral Blood Film - A Review. Ann Ib Postgrad Med. 2014;12(2):71-9.

[8] Govind D, Lutnick B, Tomaszewski JE, Sarder P. Automated erythrocyte detection and classification from whole slide images. J Med Imaging. 2018;5(2):027501.

[9] Mohammed EA, Mohamed MMA, Far BH, Naugler C. Peripheral blood smear image analysis: A comprehensive review. Journal of Pathology Informatics. 2014;5(1):9.

[10] Verma A, Verma M, Singh A. Animal tissue culture principles and applications. Animal Biotechnology. 2020:269-93.

[11] Joubert J, Weyers R, Raubenheimer J. Reducing unnecessary blood smear examinations: can Sysmex blood cell analysers help? Medical Technology SA. 2014;28(1).

[12] Kuhn V, Diederich L, Stevenson Keller T, Kramer C, Lückstädt W, Panknin C, et al. Red Blood Cell Function and Dysfunction: Redox Regulation, Nitric Oxide Metabolism, Anemis. Antioxid Redox Signal. 2017;26(13):718-42.

[13] Beug H, Bauer A, Dolznig H, vLindern M, Lobmayer L, Mellitzer G, et al. Avian erythropoiesis and erythroleukemia: towards understanding the role of the biomolecules involved. BBA - Reviews on Cancer. 1996;1288(3):M35-47.

[14] Moras M, Lefevre SD, Ostuni MA. From Erythroblasts to Mature Red Blood Cells: Organelle Clearance in Mammals. Front Physiol. 2017;8:1076.

[15] Stier A, Bize P, Schull Q, Zoll J, Singh F, Geny B, et al. Avian eryhtrocytes have functional mitochondria, opening novel perspectives for birds as animal models in the study of ageing. Frontiers in Zoology. 2013;10(33).

[16] Pikora K, Kretowska-Grundwals A, Krawczuk-Rybak M, Sawicka-Zukowska M. Diagnostic Value and Prognostic Significance of Nucleated Red Blood Cells (NRBCs) in Selected Medical Conditions. Cells. 2023;12(14):1817.

[17] Yap KN, Zhang Y. Revisiting the question of nucleated versus enucleated erythrocytes in birds and mammals. Am J Physiol Regul Integr Comp Physiol. 2021;321(4):R547-57.

[18] Grigg GC, Beard LA, Augee ML. The Evolution of Endothermy and Its Diversity in Mammals and Birds. Physiological and Biochemical Zoology: Ecological and Evolutionary Approaches. 2004;77(6):982-97.

[19] Glomski CA, Pica A. The Avian Erythrocyte: Its Phylogenetic Odyssey. 1st ed. Boca Raton: CRC Press; 2011.

[20] Clark P. Assessment of avian erythrocytes that exhibit variant nuclear morphology. Comp Clin Pathol. 2015;24:486-90.

[21] Ritchie BW, Harrison GJ, Harrison LR. Avian Medicine: Principles and Application. Lake Worth: Wingers Publishing; 1994.

[22] Nombela I, Lopez-Lorigados M, Salvador-Mira ME, Puente-Marin S, Chico V, Ciordia S, et al. Integrated transcriptomic and proteomic analysis of red blood cells from rainbow trout challenged with VHSV point towards novel immunomodulant targets. Vaccines (Basel). 2019;7:63.

[23] John JL. The avian spleen: a neglected organ. Q Rev Biol. 1994;69(3):327-51.

[24] CellaVision. CellaVision DC-1; 2022. [Internet]; [cited 2024]. `https://www.cellavision.com/products/analyzers/cellavisionr-dc-1`.

[25] CellaVision. CellaVision DM9600; 2022. [Internet]; [cited 2024]. `https://www.cellavision.com/products/analyzers/cellavisionr-dm9600`.

[26] Meechart K, Auethavekiat S, Sa-ing V. An Automatic Detection for Avian Blood Cell based on Adaptive Thresholding Algorithm. In: BMEiCON; 2019. p. 1-4.

[27] Beaufrere H, Ammersbach M, Tully Jr T. Complete Blood Cell Count in Psittaciformes by Using High-Throughput Image Cytometry: A Pilot Study. J of Avian Medicine and Surgery. 2013;27(3):211-7.

[28] Vogelbacher M, Strehmann F, Bellafkir H, Mühling M, Korfhage N, Schneider D, et al. Identifying and Counting Avian Blood Cells in Whole Slide Images via Deep Learning. Birds. 2024;5(1):48-66.

[29] Hoefling H, Sing T, Hossain I, Boisclair J, Doelemeyer A, Flandre T, et al. HistoNet: A Deep Learning-Based Model of Normal Histology. SageJournals. 2021;49(4):49-4.

[30] Kittichai V, Kaewthamasorn M, Thanee S, Jomtarak R, Klanboot K, Naing K, et al. Classification for avian malaria parasite Plamodium gallinaceum blood stages by using deep convolutional neural networks. Scientific Reports. 2021;11:16919.

[31] Turing AM. Computing Machinery and Intelligence. Mind. 1950;49:433-60.

[32] McCarthy J. What is artificial intelligence?; 2007. Available from: `https://www-formal.stanford.edu/jmc/whatisai.pdf`. Stanford University.

[33] Russel S, Norvig P. Artificial Intelligence A Modern Approach. 3rd ed. Essex: Pearson Education; 2016.

[34] Hamet P, Tremblay J. Artificial intelligence in medicine. Metabolism. 2017;69:S36-40.

[35] Chang HY, Jung CK, Woo JI, Lee S, Cho J, Kim SW, et al. Artificial Intelligence in Pathology. J Pathol Transl Med. 2019;53(1):1-12.

[36] Jordan MI, Mitchell TM. Machine learning: Trends, perspectives, and prospects. Sciencemag. 2015;349(6245):255-60.

[37] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer; 2011.

[38] Jiang T, Gradus JL, Rosellini AJ. Supervised Machine Learning: A Brief Primer. Behavior Therapy. 2020;51(5):675-87.

[39] Cunningham P, Cord M, Delany SJ. Supervised Learning, Machine Learning Techniques for Multimedia, Cognitive Technologies. Berlin: Springer; 2008.

[40] Ferreira REP, Lee YJ, Dórea JRR. Using pseudo-labeling to improve performance of deep neural networks for animal identification. Scientific Reports. 2023;13:13875.

[41] Lee DH. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In: ICML 2013 Workshop: Challenges in Representation Learning. vol. 3; 2013. p. 896-901.

[42] Clearwater SH, Cheng TP, Hirsh H, Buchanan BG. Incremental Batch Learning. In: Proceedings of the Sixth International Workshop on Machine Learning; 1989. p. 366-70.

[43] Xu Y, Goodacre R. On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning. J Anal Test. 2018;2(3):249-62.

[44] Liashchynskyi P, Liashchynskyi P. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS; 2019. Available at: `https://www.researchgate.net/publication/337916821_Grid_Search_Random_Search_Genetic_Algorithm_A_Big_Comparison_for_NAS`. Internet.

[45] Ying X. An Overview of Overfitting and its Solutions. J Phys: Conf Ser. 2019;1168(2).

[46] Demir-Kavuk O, Kamada M, Akutsu T, Knapp EW. Prediction using step-wise L1, L2 regularization and feature selection for small data sets with large number of features. BMC Bioinformatics. 2011;12:412.

[47] Zou H, Hastie T. Regularization and Variable Selection Via the Elastic Net. Journal of the Royal Statistical Society Series B: Statistical Methodology. 2005;67(2):301-20.

[48] Shorten C, Khoshgoftaar TM. A Survey on Image Data Augmentation for Deep Learning. Journal of Big Data. 2019;6:60.

[49] Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: MICCAI; 2015. p. 234-41.

[50] Zhou J, Gandomi AH, Chen F, Holzinger A. Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics. Electronics. 2021;10(5):593.

[51] Dalianis H. Evaluation Metrics and Evaluation. In: Clinical Text Mining. Springer; 2018. .

[52] Baratloo A, Hosseini M, Negida A, El Ashal G. Part 1: Simple Definition and Calculation of Accuracy, Sensitivity and Specificity. Emerg(Tehran). 2015;3(2):48-9.

[53] Monaghan TF, Rahman SN, Agudelo CW, Wein AJ, Lazar JM, Everaert K, et al. Foundational Statistical Principles in Medical Research: Sensitivity, Specificity, Positive Predictive Value, and Negative Predictive Value. Medicina (Kaunas). 2021;57(5):503.

[54] Düntsch I, Gediga G. Confusion Matrices and Rough Set Data Analysis. J Phys: Conf Ser. 2019;1229:012055.

[55] Wegier W, Ksieniewicz P. Application of Imbalanced Data Classification Quality Metrics as Weighting Methods of the Ensemble Data Stream Classification Algorithms. Entropy (Basel). 2020;22(8):849.

[56] von Stralen KJ, Stel VS, Reitsma JB, Dekker FW, Zoccali C, Jager KJ. Diagnostic methods I: sensitivity, specificity, and other measures of accuracy. Kidney International. 2009;75(12):1257-63.

[57] Zhu W, Zeng N, Wang N. Sensitivity, Specificity, Accuracy, Associated Confidence Interval and ROC Analysis with Practical SAS Implementations. NESUG. 2010.

[58] Sarker IH. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN Computer Science. 2021;2:420.

[59] Goodfellow I, Bengio Y, Courville A. Deep Learning. Cambridge: The MIT Press; 2016.

[60] Chang HY, Jung CK, Woo JI, Lee S, Cho J, Kim SW, et al. Artificial Intelligence in Pathology. J Pathol Transl Med. 2019;53(1):1-12.

[61] Hinton GE. How Neural Networks Learn from Experience. Scientific American. 1992;267(3):144–151.

[62] Sharma S, Athaiya A. Activation Functions in Neural Networks. IJEAST. 2020;4(12):310-6.

[63] Aggarwal CC. Neural Networks and Deep Learning. 1st ed. Springer; 2019.

[64] Islam M, Chen G, Jin S. An Overview of Neural Network. American Journal of Neural Networks and Applications. 2019;5(1):7-11.

[65] Nguyen T, Raghu M, Kornblith S. Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth. In: ICLR; 2021. .

[66] Meyer-Baese A, Schmid B. Specialized Neural Networks Relevant to Bioimaging, Pattern Recognition and Signal Analysis in Medical Imaging. 2nd ed.; 2014.

[67] Singh A, Kushwaha S, Alarfaj M, Singh M. Comprehensive Overview of Backpropagation Algorithm for Digital Image Denoising. Electronics. 2022;11(10):1590.

[68] Song Y, Lukasieqicz T, Xu Z, Bogacz R. Can the Brain Do Backpropagation? - Exact Implementation of Backpropagation in Predictive Coding Networks. In: Adv Neural Inf Process Syst. vol. 33; 2020. p. 22566-79.

[69] Kerkhof M, Wu L, Perin G, Picek S. No (good) loss no gain: systematic evaluation of loss functions in deep learning-based side-channel analysis. Journal of Cryptographic Engineering. 2023;13:311-24.

[70] Wu MT. Confusion matrix and minimum cross-entropy metrics based motion recognition system in the classroom. Scientific Report. 2022;12:3095.

[71] Rezaei-Dastjerdehei M, Mijani A, Fatemizadeh E. Addressing Imbalance in Multi-Label Classification Using Weighted Cross Entropy Loss Function. In: ICBME; 2020. p. 333-8.

[72] Lin TY, Goyal P, Girshick R, He K, Dollár P. Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020;42(2):318-27.

[73] Krittanawong C, Johnson KW, Rosenson RS, Wang Z, Aydar M, Baber U, et al. Deep learning for cardiovascular medicine: a practical primer. European Heart Journal. 2019;00:1-15.

[74] You K, Long M, Wang J, Jordan MI. How Does Learning Rate Decay Help Modern Neural Networks? 2019. [Internet] Available from: `https://arxiv.org/abs/1908.01878`.

[75] Smith LN. Cyclical Learning Rates for Training Neural Networks. In: IEEE WACV; 2017. p. 464-72.

[76] Tian Y, Zhang Y, Zhang H. Recent Advances in Stochastic Gradient Descent in Deep Learning. Mathematics. 2023;11(3):682.

[77] Fu J, Wang B, Zhang H, Zhang Z, Chen W, Zheng N. When and Why Momentum Accelerates SGD: An Empirical Study. 2023. [Internet] Available from: `arXiv:2306.09000`.

[78] Elshamy R, Abu-Elnasr O, Elhoseny M, Elmougy S. Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning. Scientific Report. 2023;13:8814.

[79] Kingma DP, Lei Ba J. Adam: A Method for Stochastic Optimization. In: ICLR; 2015. .

[80] Ketkar N, Moolayil J. Convolutional Neural Networks. In: Deep Learning with Python. Berkeley: Apress; 2021. p. 194-242.

[81] Zhang X, Zhang X, Wang W. Convolutional Neural Network. In: Intelligent Information Processing with Matlab. Singapore: Springer; 2023. p. 39-71.

[82] Li Z, Liu F, Yang W, Peng S, Zhou J. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. IEEE Transactions on Neural Networks and Learning Systems. 2022;33(12):6999-7019.

[83] Weisstein EW. Convolution;. From MathWorld–A Wolfram Web Resource. Available from: `https://mathworld.wolfram.com/Convolution.html`.

[84] Krichen M. Convolutional Neural Networks: A Survey. Computers. 2023;12(8):151.

[85] Yamashita R, Nishio M, Kinh Gian Do R, Togashi K. Convolutional neural networks: an overview and application in radiology. Insights into Imaging. 2018;9:611-29.

[86] Li Z, Liu F, Yang W, Peng S, Zhou J. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. IEEE Transactions on Neural Networks and Learning Systems. 2022;33(12):6999-7019.

[87] Musa N, Gital AY, Aljojo N, Chiroma H, Adewole KS, Mojeed HA, et al. A systematic review and Meta-data analysis on the application of Deep Learning in Electrocardiogram. J Ambient Intell Human Comput. 2023;14:9677-750.

[88] Yani M, Irawan B, Setiningsih C. Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. In: J. Phys.: Conf. Ser.. vol. 1201; 2019. .

[89] Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data. 2021;8(53).

[90] Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: ICML. vol. 37; 2015. p. 448-56.

[91] Pathak AR, Pandey M, Rautaray S. Application of Deep Learning for Object Detection. Procedia Computer Science. 2018;132:1706-17.

[92] Zou Z, Chen K, Shi Z, Guo Y, Ye J. Object Detection in 20 Years: A Survey. Proceedings of the IEEE. 2023;111(3):257-76.

[93] Amit Y, Felzenszwalb P, Girshick R. Object Detection. In: Ikeuchi K, editor. Computer Vision. Cham: Springer; 2021. .

[94] Terven J, Cordova-Esparza DM, Gonzalez J. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. Machine Learning and Knowledge Extraction. 2023;5(4):1680-716.

[95] Zou Z, Chen K, Shi Z, Guo Y, Ye J. Object Detection in 20 Years: A Survey. In: Proceedings of the IEEE. vol. 111; 2023. p. 257-76.

[96] Carranza-Garcia M, Torres-Mateo J, Lara-Benitez P, Garcia-Gutierrez J. On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data. Remote Sens. 2021;13(1):89.

[97] Soviany P, Ionescu R. Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction. In: SYNASC; 2018. p. 2029-214.

[98] Kateb F, Monowar M, Hamid A, Ohi A, Mridha M. FruitDet: Attentive Feature Aggregation for Real-Time Fruit Detection in Orchards. Agronomy. 2021;11:2440.

[99] Lin TY, Dollár P, Girshick R, He K, Hariharan B, Belongie S. Feature Pyramid Networks for Object Detection. In: CVPR; 2017. p. 936-44.

[100] Rezatofighi H, Tsoi N, JunYoung G, Sadeghian A, Reid I, Savarese S. Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression. In: CVPR; 2019. p. 658-66.

[101] Padilla R, Netto S, dSilva E. A Survey on Performance Metrics for Object-Detection Algorithms. IWSSIP. 2020.

[102] Jha S, Seo C, Yang E, Joshi G. Real-time Object Detection and Tracking System for Video Surveillance System. Multimedia Tools and Applications. 2020;80:3981-96.

[103] Padilla R, Passos W, Dias T, Netto S, dSilva E. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. Electronics. 2021;10(3):279.

[104] Liu C, Tao Y, Liang J, Li K, Chen Y. Object Detection Based on YOLO Network. In: ITOEC; 2018. p. 799-803.

[105] Park I, Kim S. Performance Indicator Survey for Object Detection. In: ICCAS; 2020. p. 284-8.

[106] Tsung-Yi L, Maire M, Belongie S, Bourdev L, Girshick R, Hays J, et al. Microsoft COCO: Common Objects in Context. CoRR. 2014. [Internet]. Available from: http://arxiv.org/abs/1405.0312.

[107] Zhao L, Li S. Object Detection Algorithm Based on Improved YOLOv3. Electronics. 2020;9(3):537.

[108] Padilla R, Netto S, dSilva E. A Survey on Performance Metrics for Object-Detection Algorithms. IWSSIP. 2020.

[109] Kim H, Cosa-Linan A, Santhanam N, Jannesari M, Maros M, Ganslandt T. Transfer learning for medical image classification: a literature review. BMC Medical Imaging. 2022;22:69.

[110] Weiss K, Khoshgoftaar T, Wang D. A survey of transfer learning. Journal of Big Data. 2016;3:9.

[111] Russakovsky O, Deng J, Su H, et al. Imagenet Large Scale Visual Recognition Challenge. Int J Comput Vis. 2015;115:211-52.

[112] Zhang W, Li R, Zeng T, Sun Q, Kumar S, Ye J. Deep Model Based Transfer and Multi-Task Learning for Biological Image Analysis. IEEE Trans Big Data. 2020;6(2):322-33.

[113] Kieffer B, Babaie M, Kalra S, Tizhoosh H. Convolutional neural networks for histopathology image classification: Training vs. Using pre-trained networks. In: IPTA; 2017. p. 1-6.

[114] Vrancic G, Podgorelec V. Transfer Learning With Adaptive Fine-Tuning. IEEE Access. 2020;8:196197-211.

[115] Szegedy C, Liu W, Jia Y, Sermanet P. Going deeper with convolutions. In: CVPR; 2015. p. 1-9.

[116] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the Inception Architecture for Computer Vision. In: CVPR; 2016. p. 2818-26.

[117] Ahn J, Kim S, Ahm K, Cho S, Lee K. A deep learning model for the detection of both advanced and early glaucoma using fundus photography. Plos one. 2019;14(1).

[118] Ultralytics. YOLOv8; 2024. `https://github.com/ultralytics/ultralytics`.

[119] Chollet F, et al. Keras; 2024. Internet. Available from: `https://keras.io/api/applications/`.

[120] tf.keras.optimizers.Adam; 2024. Internet. Available from: `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam`.

[121] Kotsiantis S, Kanellopoulos D, Pintelas P. Handling imbalanced datasets: A review. GESTS Transactions on Computer Science and Engineering. 2006;30.

[122] Hassanat AB, Tarawneh AS, Altarawneh GA. Stop Oversampling for Class Imbalance Learning: A Critical Review. 2022. Available from: arXiv:2202.03579.

[123] Buslaev A, Parinov A, Khvedchenya E, Iglovikov VI, Kalinin AA. Albumentations: fast and flexible image augmentations; 2018. Available from: arXiv:1809.06839. https://arxiv.org/abs/1809.06839.

[124] Ultralytics, Brief summary of YOLOv8 model; 2024-01-10. https://github.com/ultralytics/ultralytics/issues/189.

[125] Mavaie P, Holder L, Skinner MK. Hybrid deep learning approach to improve classification of low-volume high-dimensional data. BMC Bioinformatics. 2023;24:419.

Appendices

Appendix A

Complementary information

A.1 YOLOv8 Architecture

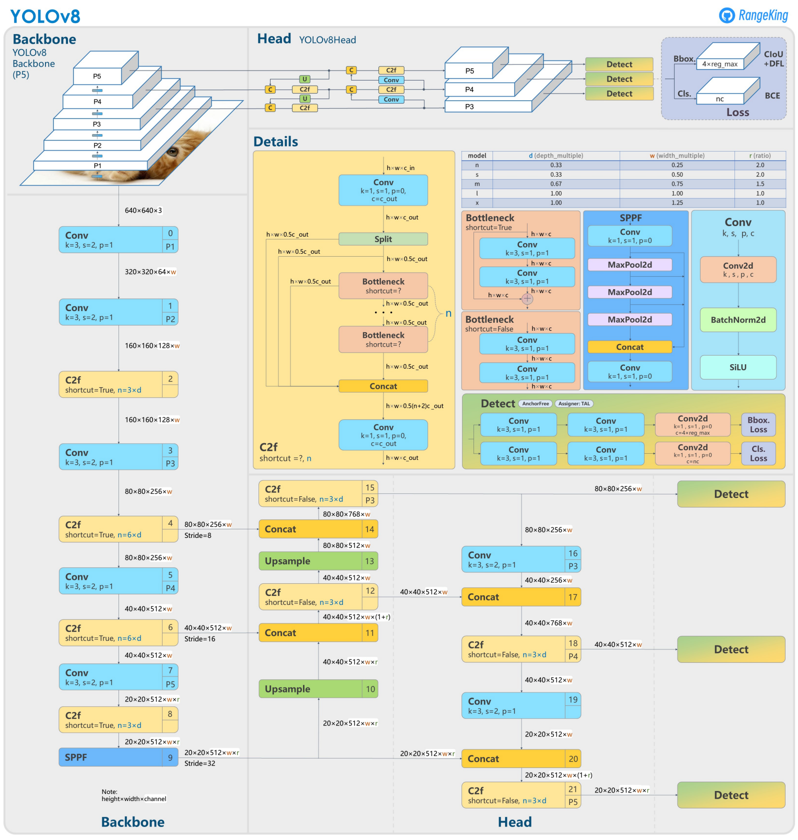


Figure A.1: YOLOv8 architecture [124].

A.2 Learning Rate Shapes

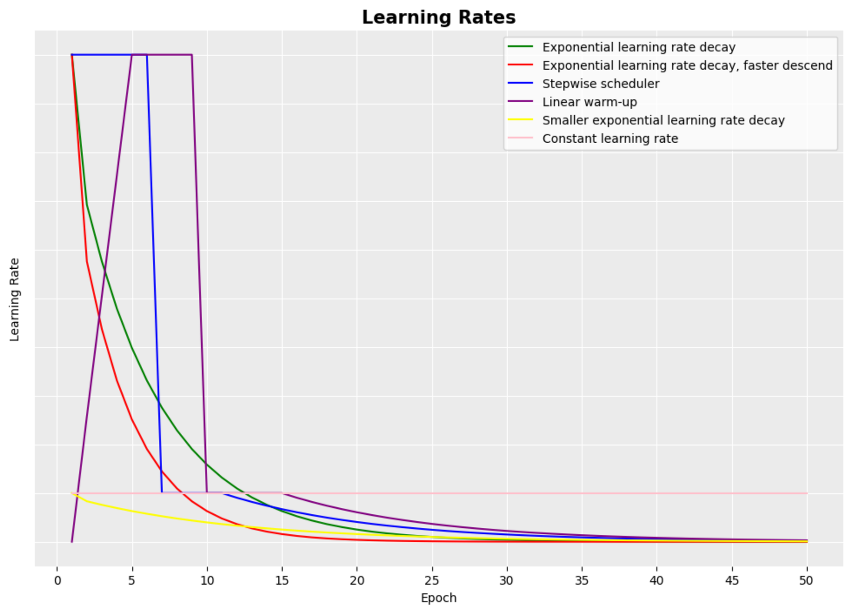


Figure A.2: Shape of different learning rate schedulers including step wise learning rate schedulers, schedulers with exponential learning rate decay, a scheduler including linear warm-up and a constant learning rate.

A.3 Initial training

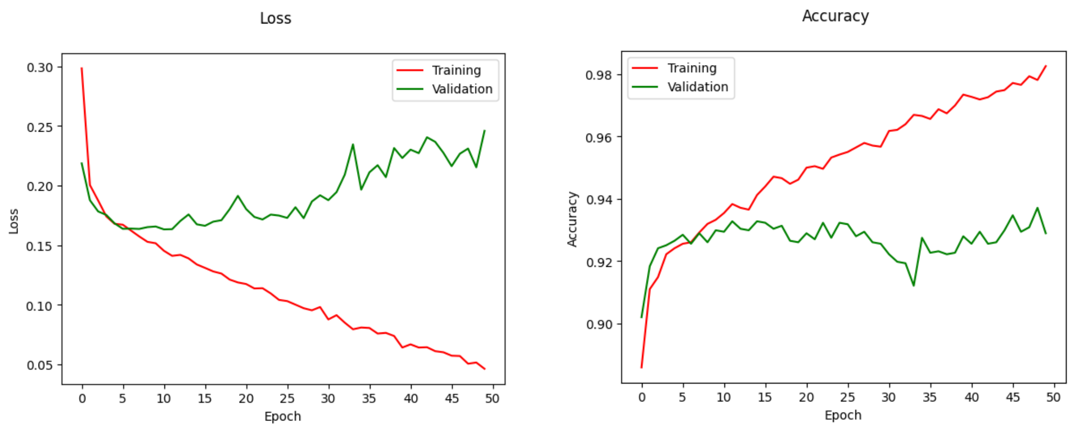


Figure A.3: Loss (left) and accuracy (right) of training (red) and validation (green) dataset with the InceptionV3 transfer learning model as base model.

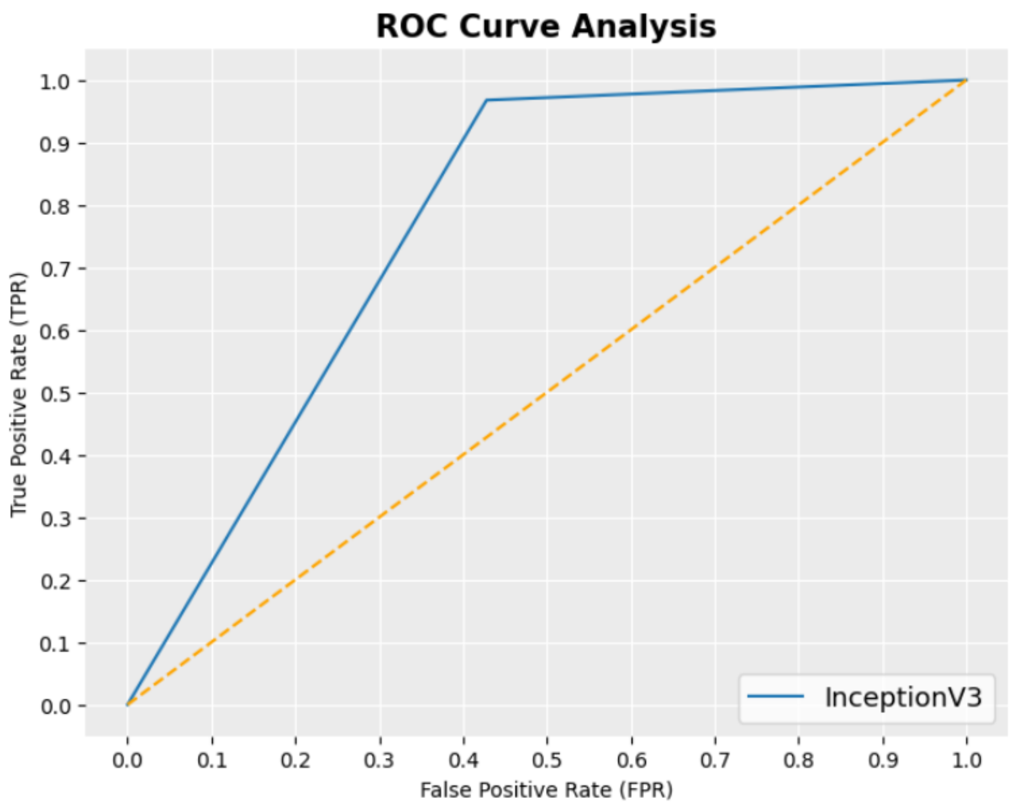


Figure A.4: ROC curve for initial model using InceptionV3 as base model. Trained on the binary dataset.

A.4 Initial training with dropout

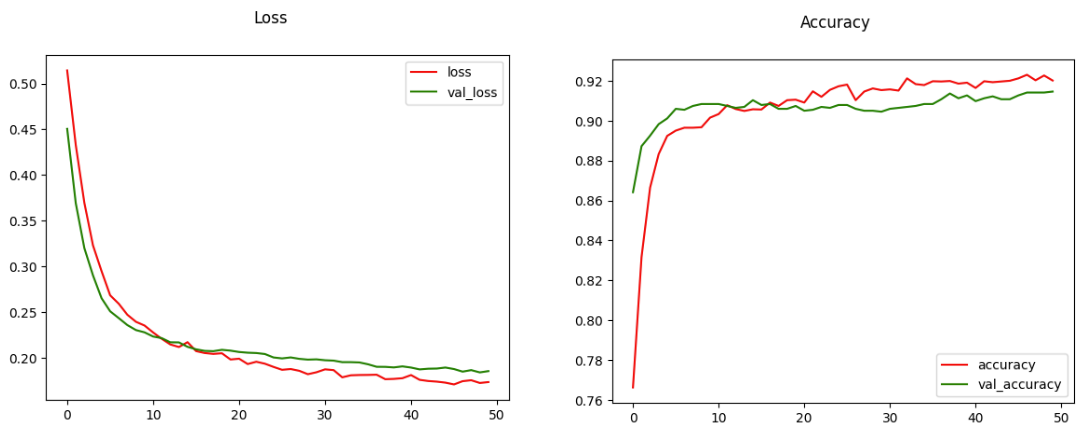


Figure A.5: Loss (left) and accuracy (right) of training (red) and validation (green) dataset with the InceptionV3 transfer learning model as base model with added dropout.

A.5 Testing optimisers

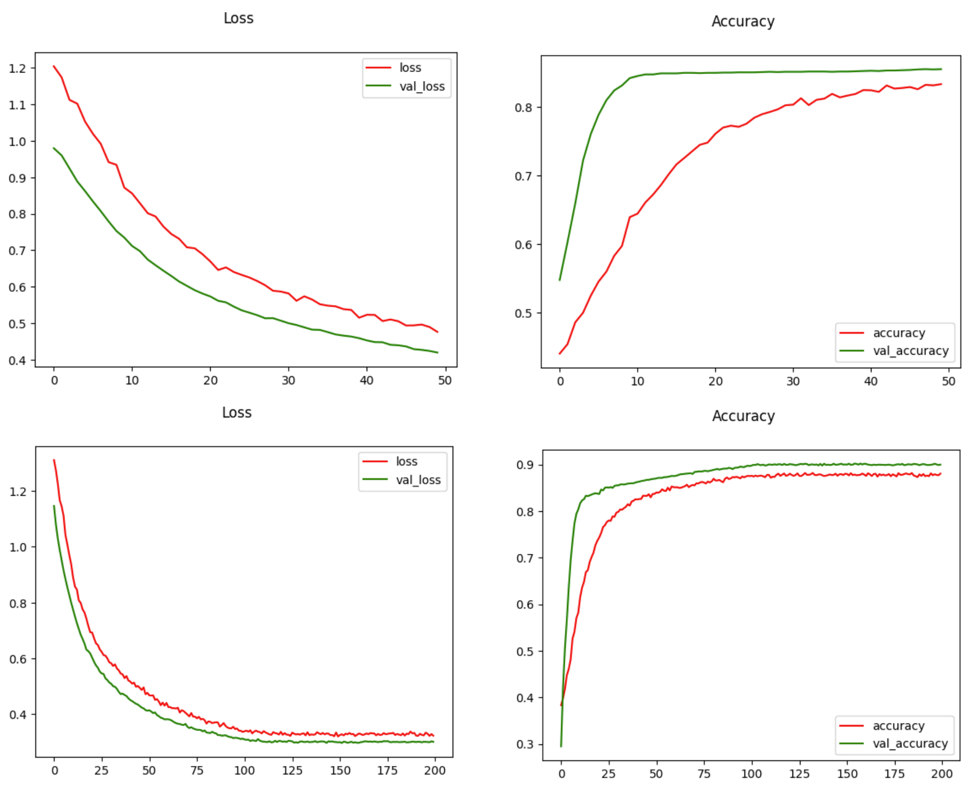


Figure A.6: Loss (left) and accuracy (right) of training (red) and validation (green) dataset trained for 50 (top) and 200 (bottom) epochs with the ADAM optimiser initialised with 0.1 epsilon value.

A.6 Training with extended dropout design

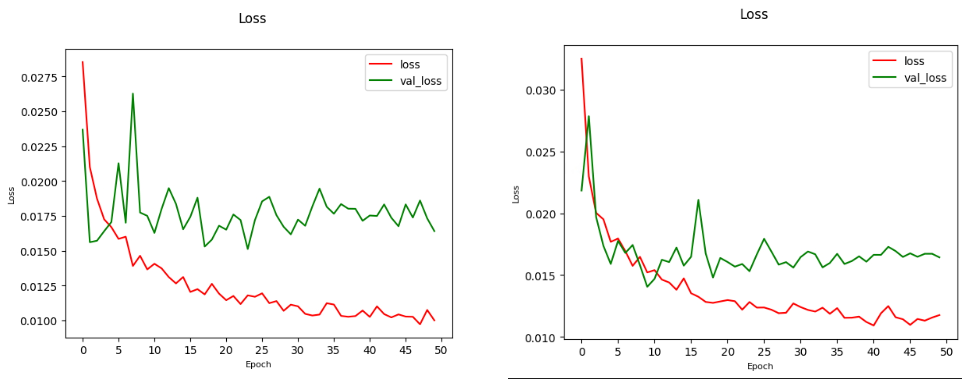
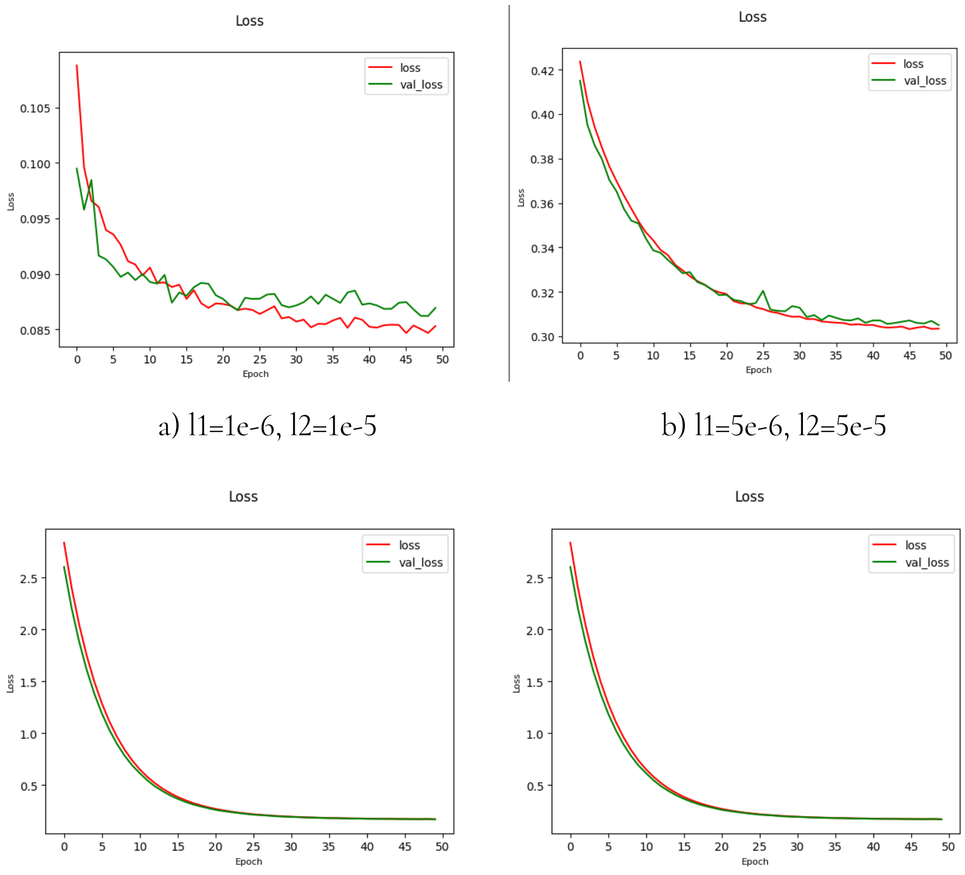


Figure A.7: Loss functions of model trained with dropout rate of 0.8 (left) and with four repeated dropout layers with dropout rate 0.2.

A.7 Training with L1/L2-regularisation



a) l1=1e-6, l2=1e-5

b) l1=5e-6, l2=5e-5

c) l1=5e-5, l2=5e-5

d) l1=1e-5, l2=1e-4

Figure A.8: Loss of training (red) and validation (green) dataset trained with focal loss with L1/L2-regularisation.

A.8 Final ROC

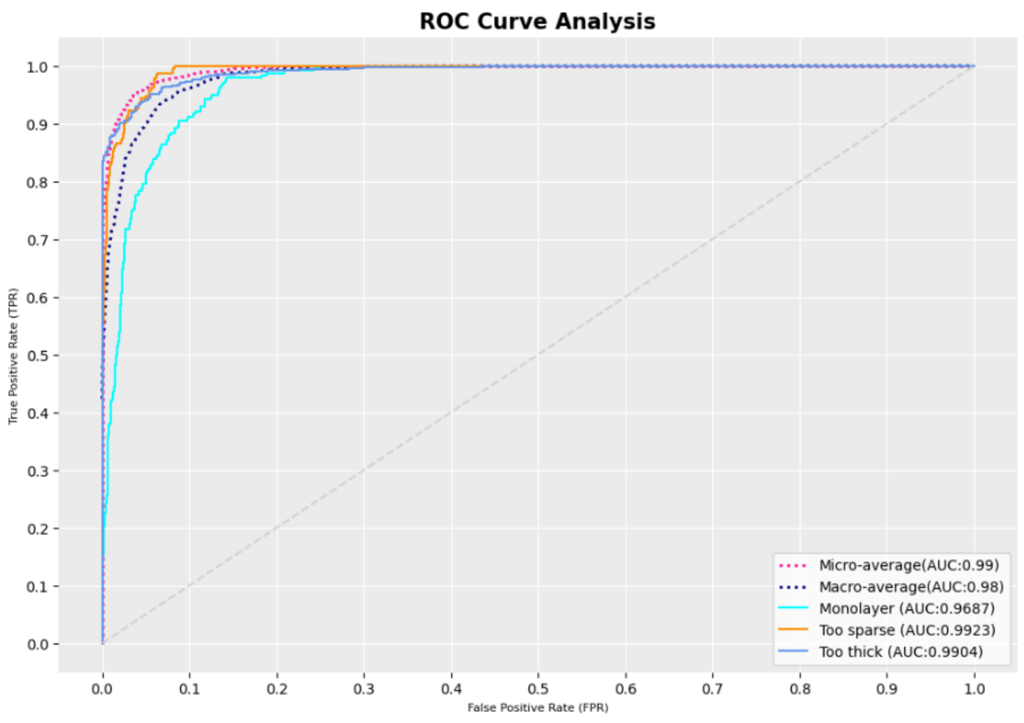


Figure A.9: ROC curve using predicted probabilities.

Appendix B

Additional tests

B.1 Combined neural network and support vector machine classifier

Both deep learning in neural networks and non-deep learning in machine learning methods have their benefits and drawbacks. As previously mentioned, deep learning has its strength in being able to model complex relationships in the data features, but also requires a large amount of data. It is also necessary to optimise the hyperparameters well in order to receive adequate performance. Some studies have demonstrated that a hybrid solution of a deep neural network in combination with a ML method can outperform the two models individually [125]. This was explored in a smaller test in which a neural network was integrated with the machine learning method SVM, hoping to be able to combine complex features with the efficiency of handling smaller datasets and generalisation inherited in the SVM algorithm. This was performed by using an InceptionV3 model with expanded architecture and added elastic net-regularisation. In order to extract features to be used as input to the SVM, top layers of the InceptionV3 network were removed and the features could be extracted by being predicted in the network. This was tested with both solely using the ImageNet weights as well as with weights set after training the model on the dataset. The SVM was initialised with a linear kernel, using default values with the hyperparameter C set to the value 1.0. The features were scaled prior to being fed into the SVM model, trained and evaluated. Hyperparameters such as kernel and C were varied between different models.

The performance of a neural network and SVM combination can be seen in Table B.1 with a InceptionV3 trained on the data, and in Table B.2, initialised solely with ImageNet weights.

Table B.1: Performance following training with a combined neural network trained on the data with a SVM classifier. The first model represents the trained model with a normal network classifier.

Kernel	C	Accuracy	Class precision (s/m/t)
-	-	0.95	0.88/0.76/0.98
Sigmoid	0.5	0.78	0.34/0.27/0.83
	1	0.81	0.15/0.5/0.83

Table B.2: Performance following training with a combined neural network initialised on ImageNet weights with a SVM classifier.

Kernel	C	Accuracy	Class precision (s/m/t)
Linear	0.5	0.72	0.24/0.15/0.94
	1	0.81	0.18/0.16/0.84
	5	0.81	0.47/0.11/0.82
	10	0.52	0.01/0.19/0.99
Sigmoid	0.5	0.81	0.14/0.19/.082
	1	0.8	0.5/0.18/0.82
	10	0.72	0.03/0.10/0.81
Radial basis function	1	0.82	0.0/0.0/0.82
Poly	1	0.82	0.64/0.0/0.82

Looking at the results from Table B.2 and Table B.1, the combination of a neural network, previously trained or not, was not able to achieve similar results as to the standard neural network. Some improvements can be seen while attempting to optimise the network, which indicates that it needs further adjustments to be able to improve to adequate performance. Furthermore, optimisation could include Principal Component Analysis to help with the dimensionality as well as find the optimal number of top layers to remove. There is a possibility that this combination simply is not fit for the task and data. The hypothesis of increased performance following a combined ML approach can hence not be supported with these results, but future research could include further optimisation in order to achieve benefits from this combination.

B.2 Optimising focal loss

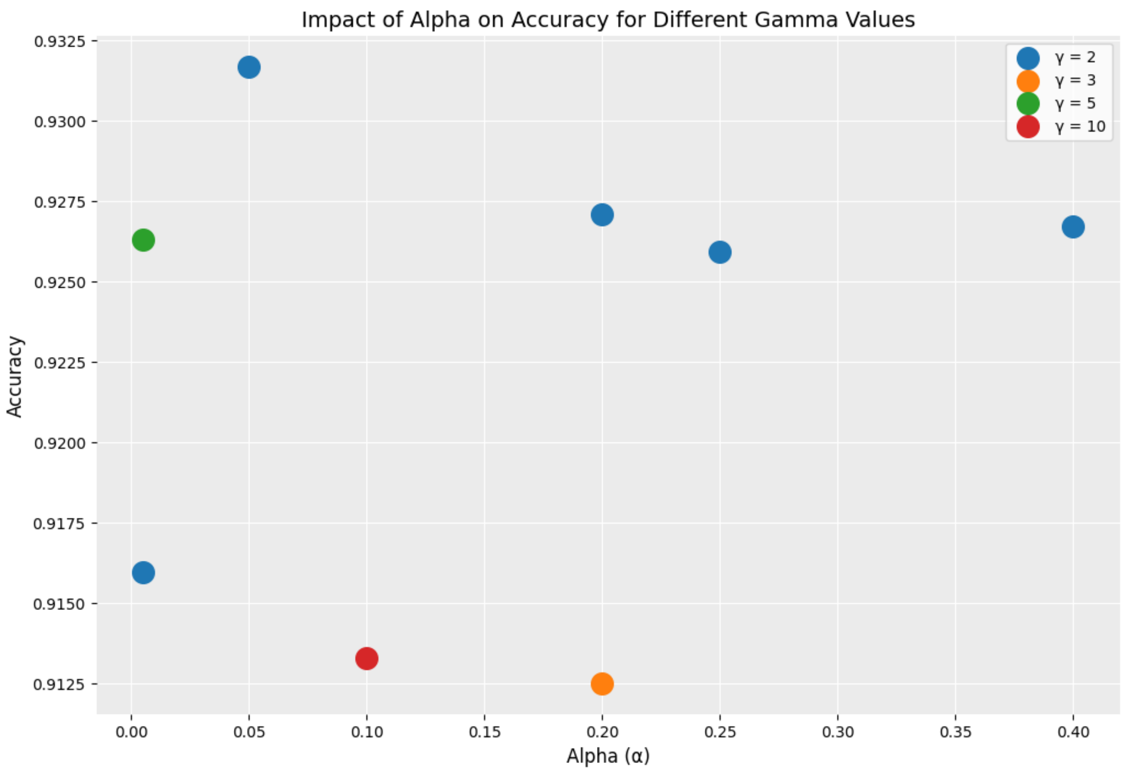


Figure B.1: Impact on accuracy of varying α and γ values for α-balanced focal cross-entropy loss. The colours represent different values of γ.

Table B.3: Effects of different types of losses trained with default values.

Epochs	γ	α	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
	2.0	0.005	0.91596	0.76631	0.79084	0.77164	0.84761	0.74098/ 0.58730/ 0.97064
	2.0	0.05	**0.93170**	**0.82082**	0.84356	0.83097	0.87569	0.82528/ 0.66234/ 0.97483
	2.0	0.2	0.92709	0.81612	**0.84759**	**0.83111**	0.87199	**0.84921/** 0.62406/ 0.97510
50	2.0	0.25	0.92594	0.80904	0.79600	0.79442	**0.87610**	0.77193/ **0.69006/** 0.96512
	2.0	0.4	0.92671	0.80498	0.82070	0.81112	0.87047	0.80812/ 0.63470/ 0.97212
	2.0	1/0.2, 1/0.2, 1/0.6	0.91251	0.77768	0.84281	0.80570	0.84447	0.81853/ 0.52978/ **0.98471**
	3.0	0.2	0.92095	0.78753	0.82694	0.80561	0.85657	0.80073/ 0.58537/ 0.97649
	5.0	0.005	0.92632	0.80334	0.83401	0.81736	0.86737	0.81250/ 0.62185/ 0.97567
	10.0	0.1	0.91328	0.77455	0.79188	0.78239	0.84862	0.80989/ 0.54783/ 0.96593
200	5.0	0.005	0.91711	0.77103	0.78894	0.76700	0.85152	0.70245/ 0.63855/ 0.97209

B.3 Multiple learning rate schedulers

Performance after training with two separate step wise learning rate schedulers can be seen in Table B.4. Here, different initial and final values for both the first and the second scheduler were tested with the aim to find the optimal combination. The first model represents the model trained with the same scheduler for both phases.

Table B.4: Performance of models trained with two distinct step wise schedulers trained with cross-entropy loss.

First scheduler	Second scheduler	Accuracy	Precision	Recall	F1-score	ROC AUC	Class precision (s/m/t)
	Ep<=5: 10^{-3} Ep>5 & <=10: 10^{-4} Ep>10:lr$*e^{-1}$	0.91019	0.76951	0.79945	0.78338	0.84336	0.79167/ 0.54317/ 0.97371
Ep<=5: 10^{-3} Ep>5 & <=10: 10^{-4} Ep>10:lr$*e^{-1}$	Ep<=5: 10^{-4} Ep>5 & <=10: 10^{-5} Ep>10:lr$*e^{-1}$	**0.92535**	**0.81346**	0.83121	**0.82187**	0.87146	**0.83665**/ 0.63445/ 0.96928
	Ep<=5: 10^{-5} Ep>5 & <=10: 10^{-6} Ep>10:lr$*e^{-1}$	0.92691	0.80699	0.80698	0.80599	**0.87410**	0.81200/ **0.63889**/ 0.97009
Ep<=5: 10^{-3} Ep>5 & <=10: 10^{-5} Ep>10:lr$*e^{-1}$	Ep<=5: 10^{-4} Ep>5 & <=10: 10^{-5} Ep>10:lr$*e^{-1}$	0.90941	0.77625	**0.87042**	0.81450	0.83972	0.82264/ 0.51429/ **0.99182**
Ep<=5: 10^{-4} Ep>5 & <=10: 10^{-5} Ep>10:lr$*e^{-1}$	Ep<=5: 10^{-5} Ep>5 & <=10: 10^{-6} Ep>10:lr$*e^{-1}$	0.91757	0.77657	0.84198	0.80589	0.84739	0.77936/ 0.56667/ 0.98367