

Investigating the CNN training data requirements for classification of COVID-19 from chest X-rays



UNIVERSITY OF
LINCOLN

Teddie-Valentine Botten
25699409

25699409@lincoln.ac.uk

School of Computer Science
College of Science
University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of BSc (Hons) Computer Science

Supervisor Dr. Petra Bosilj

May 2024

Acknowledgements

I would like to show my gratitude to my supervisor Dr. Petra Bosilj for their consistent support and guidance throughout this project. Your expertise and suggestions were invaluable.

Finally, I would like to thank my family and my friends for their constant support and encouragement, providing me with the motivation I needed to progress throughout university.

Abstract

Convolutional Neural Networks (CNNs) have shown to be effective in classifying COVID-19 from X-ray scans but require large amounts of data, requiring an extreme effort to label from medical professionals. This paper attempts to explore the training data required for an effective COVID classifier through training models on different configurations of data after investigating optimal hyperparameters and classifier architecture. We show that although increases in training data improve the model's performance, CNNs can still perform modestly within classification tasks on as low as 250 samples of data.

Table of Contents

1	Introduction	2
2	Project Overview and Management	4
2.1	Aims and Objectives	4
2.1.1	Aims	4
2.1.2	Objectives	5
2.2	Requirements Analysis	6
2.2.1	Functional Requirements	6
2.2.2	Non-Functional Requirements	6
2.3	Project Management	7
2.3.1	Project Plan	9
2.3.2	Risk Analysis	11
3	Literature Review	12
3.1	Early Convolutional Neural Networks	12
3.2	Convolutional Neural Networks Classification and ImageNet	13
3.3	Applications and Challenges of Convolutional Neural Networks	13
3.4	Convolutional Neural Networks for Medical Imaging	15
3.5	Data Scarcity in Medical Imaging	17
4	Methodology & Experimental Design	18
4.1	ResNet	18
4.1.1	Convolution Layers	18
4.1.2	Activation Function	19
4.1.3	Batch Normalisation	21
4.1.4	Pooling Layers	22
4.1.5	Dropout	22
4.1.6	Fully Connected Layers	23
4.1.7	Loss Function	23
4.1.8	Optimiser	24
4.2	Datasets	25

4.3	Experiments	27
4.4	Hyperparameters	29
4.4.1	Evaluation Metrics	29
5	Implementation	31
5.1	Hardware, Toolsets and Environments	31
5.1.1	Programming Language and Libraries	32
5.2	Cleaning and Encoding	33
5.3	Augmentation	34
5.4	Custom Dataset and CNN Model	35
5.5	Training and Validation	36
5.6	Model and Hyperparameter Configuration	39
5.6.1	Batch Size	39
5.6.2	Optimiser and Learning Rate	40
5.6.3	Dropout	41
5.6.4	ReLU	42
6	Results & Discussion	44
6.1	Initial Training Set Splits	44
6.2	Further Training Set Splits	47
7	Conclusion	49
7.1	Reflection	49
7.2	Future Work	50
	References	50

List of Figures

2.1	A KANBAN board used to monitor tasks throughout the project. . .	8
2.2	Project timeline presented as a Gantt chart.	10
4.1	Skip connection (He et al., 2016).	19
4.2	A convolution operation with a 3x3 kernel and stride of 1 with no padding applied (Yamashita et al., 2018).	20
4.3	Graph of the ReLU activation function, demonstrating zero output for negative inputs and a linear response for positive inputs	21
4.4	Example of average pooling and max pooling with a 2x2 matrix and stride of 2.	23
4.5	Different variations of images observed in the dataset.	27
5.1	Structure of dataset folder	35
5.2	Default ResNet Model vs 25 Dropout + ReLU Model	43
6.1	Confusion Matrices of Initial Training Splits	46
6.2	Model performance as the training set size decreases	48

List of Tables

2.1	Risk Analysis and Mitigation Table.	11
4.1	Datasplit for each dataset of COVID-19 Images	26
4.2	Class distributions across dataset split	27
4.3	Initial training set datasplit	28
4.4	Further training set datasplit	28
5.1	Results of different batch size on the test set with Adam Optimiser . .	40
5.2	Results of different batch size on the test set with SGD Optimiser . .	40
5.3	Results of varying learning rates on the test set	41
5.4	Results of the addition of dropout layers	41
5.5	Results of the addition of a ReLU after the linear layer	42
6.1	Results of the different training configurations on the test set	45
6.2	Results of the further training configurations on the test set	47

Listings

5.1	Encoding labels in dataset.py	33
5.2	Data augmentation applied to training set	34
5.3	Custom dataset class definition	35
5.4	Train images in a custom dataset class	36
5.5	CNN Model and Freeze Weights	36
5.6	Training function	36
5.7	Validation loop	38
5.8	Data loader	39
5.9	Dropout layer addition	41
5.10	Dropout layer addition	42

Chapter 1

Introduction

The work presented in this section is an expanded version of the original project proposal (Botten, 2023), serving as a baseline.

Coronavirus (COVID-19) is a respiratory infection caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2), causing a global pandemic in late 2019. There are various forms of detecting COVID-19 such as genome sequencing, molecular testing, and antigen-based testing, with the most common being reverse transcription polymerase chain reaction (RT-PCR). This form of testing requires large centralised labs, and trained personnel, making them unsuitable for a large-scale outbreak. Such factors can possibly lead to test shortages, delays and obstructions, reducing the effectiveness of disease-preventative measures (Lau et al., 2021). Furthermore, the sensitivity (true-positive rate) of these tests is highly unpredictable with findings stating a poor result of 60-70%, with multiple tests needed to produce a positive result (Kanne et al., 2020).

Other modalities such as computer tomography (CT), and X-rays have been used for screening and diagnosing COVID-19, however, manually analysing large volumes of this data is troublesome, and relies on the expertise of the radiologist to produce accurate results. This can be a problem when similarities in diseases, such as pneumonia and COVID-19, prove to be somewhat imperceptible by the human eye (Luz et al., 2022). Clearly, there is still a need for more reliable methods in aiding diagnoses of COVID-19.

Deep learning techniques, specifically convolutional neural networks (CNNs) have a

widespread use in the medical industry (L. Wang et al., 2020) and have performed well in areas such as tumour segmentation, and multiple lung disease classification (Abd-Ellah et al., 2018; Rajpurkar et al., 2017). Although they provide good performance, CNNs require large amounts of varied data to achieve this, and in some domains there is a lack of data directly (Alzubaidi, Fadhel et al., 2020). Techniques such as transfer learning (TL), where weights are obtained from a model trained on a larger dataset and used as a baseline model for training on a smaller dataset, and data augmentation, where random cropping, rotation and erasing of images, can be used to improve the performance of network, and are standard practice when training models for a new task (Alzubaidi, Zhang et al., 2021; Shorten and Khoshgoftaar, 2019). More advanced techniques such as n-shot learning, using n number of examples per class, self-supervised, predicting labels, and weakly supervised, inferring labels from training on smaller datasets, are also used for training on small amounts of annotated data.

These techniques are all within the paradigm of using smaller datasets effectively for intricate computer vision tasks. This project will attempt to explore the relationship between the amount of available training data and CNN model performance by increasing the amount of data a model is being trained on, whilst using techniques such as data augmentation and transfer learning to aid model performance.

Chapter 2

Project Overview and Management

This section serves as the backbone of the project, providing a comprehensive framework that ensures a successful project. The aims and objectives set out the purpose of the project whilst outlining the structured tasks in order to complete the overarching aim. The requirement analysis discusses functional and non-functional requirements critical to the project's success and finally, project management, which discusses the logistical approaches adopted including risk analysis, mitigation and project planning.

2.1 Aims and Objectives

2.1.1 Aims

This project aims to use a pre-trained CNN to deduce the required training data to produce an effective model for classifying COVID-19 from chest X-rays. Training data requirements for deep learning algorithms will be investigated through evaluation metrics, tuning of hyperparameters (learning rate and batch size) and increasing training data. In the original project proposal (Botten, 2023) it was planned to reduce the amount of training data the model is trained on, however, this changed due to the imbalanced dataset that was gathered, which will be discussed in a later section. Thus, a bottom-up approach was taken by increasing the amount of data the model is trained on.

2.1.2 Objectives

The objectives were produced to create a structured approach to fulfil the aims, these objectives should be completed in a chronological order. All objectives are time bound by the due date of the project.

1. Collect and investigate a dataset of chest X-ray images selected from a respected source.
 - Ensure all images are unique and visualise the number of samples per class.
2. Design the evaluation framework
 - Choose appropriate evaluation metrics to assess the effectiveness of the model.
 - Establish train-validation-test split for each experiment
 - Determine the class distributions across training, validation and test sets to maintain proper representation.
3. Research and apply pre-processing techniques and data augmentation to prepare the X-ray scan dataset for use.
4. Train a CNN architecture and evaluate performance
 - Develop and train a CNN model using the prepared dataset.
 - Evaluate the model's performance on a test set to assess its generalisation capability.
5. Tune hyperparameters.
 - At least 10 different settings will be tested to optimise the performance of the model, with the option to explore more settings where found appropriate.

2.2 Requirements Analysis

In this section different types of requirements will be discussed, functional and non-functional. Functional requirements denote the features and functionalities of the artefact to ensure its success, whereas, non-functional requirements are defined as the characteristics of the system and its performance. These requirements shall ensure the objectives set out in [section 2.1](#) are met.

2.2.1 Functional Requirements

A python notebook will be created for exploratory analysis of the dataset where all necessary libraries and functionalities should be imported. This file should investigate the dataset through the number of samples, ensuring images are unique and verify the class-split, this ensures objective 1 is met. Some pre-processing of data may need to be completed such as encoding labels or dropping unnecessary columns so the CNN can take in the images and class labels. In the main training loop an evaluation framework will be established using metrics deemed appropriate in the later sections, and model selection should take place using the chosen metric or loss. Hyperparameter tuning such as change in optimiser, learning rate and dropout, will take place and should be informed by the evaluation metrics.

Above this, the train-test-validation split will be determined and class distributions among the sets should be appropriate to maintain a proper representation of the dataset, again, this will be determined based on the number of samples and class distribution of the dataset. Additionally, data pre-processing and data augmentation techniques should be applied, these include but are not limited to, resizing images, random cropping, random flipping, randomly applied contrast, hue and saturation, these aid model performance (Sarvamangala and Kulkarni, [2022](#)).

2.2.2 Non-Functional Requirements

1. Performance and Efficiency

- The system should be capable of processing large volumes of image data efficiently.
- The CNN model should be able to classify unseen data with a good level of accuracy.

2. Robustness

- The model should maintain high performance regardless of variations in image quality such as different contrasts, brightness and saturations.

3. Reliability

- The classification of unseen samples of data should be reliable and consistent within the CNN model.

2.3 Project Management

To ensure the success of the project, thoughtful planning is essential for managing time and resources efficiently in the project. Although the project is mostly research based, it's crucial to allocate time wisely to maximise both the artefact development and research phases, ensuring they progress concurrently throughout the project timeline. When choosing a methodology the following considerations must be made.

- 1-person team.
- Fixed Deadline.
- No budget.
- Relatively small scale research project.

Due to these constraints, the KANBAN method was used. KANBAN is a form of agile software development where the focus is on visualisation of workflow, flexibility and continuous delivery. KANBAN focuses on the limiting of work-in-progress to prevent overburdening and allows for steady progress without the constraints of fixed iterations such as in SCRUM, as such, is suitable for a 1-person team. Kanban splits tasks into three categories, "To do", "In Progress" and "Complete", these tasks are

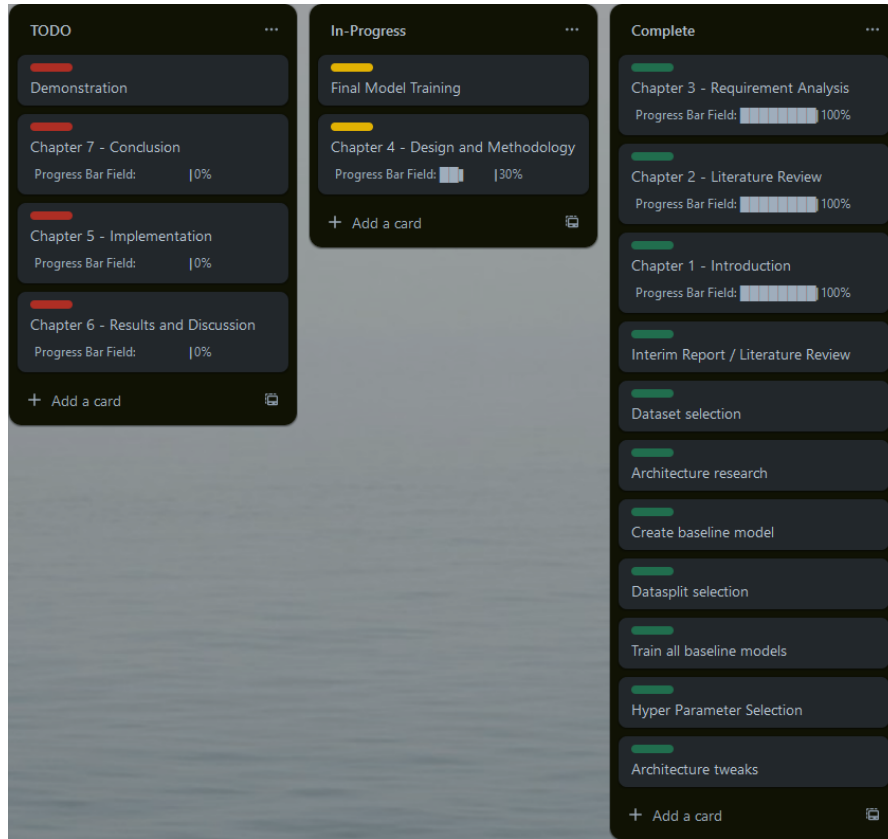


Figure 2.1: A KANBAN board used to monitor tasks throughout the project.

then moved across to board to denote progress or completion. The KANBAN board for the project is shown in [Figure 2.1](#).

Another methodology, waterfall, was considered, this is a sequential approach where each phase must be completed to progress to the next phase. Requirements are gathered and documented, the system is designed and planned, and development starts, with testing and deployment only completed in the later phases. This approach requires comprehensive planning and vast documentation (Despa, 2014) and the rigidity of the phases makes it difficult to accommodate for changing requirements, showing the methodology is ill-fitting for a solo developer. Therefore, the waterfall method was not chosen.

2.3.1 Project Plan

A project timeline was created and is shown in [Figure 2.2](#) to help keep the project on track and avoid scope creep. The objectives set in the project timeline help fulfil the objectives set out in [section 2.1](#).

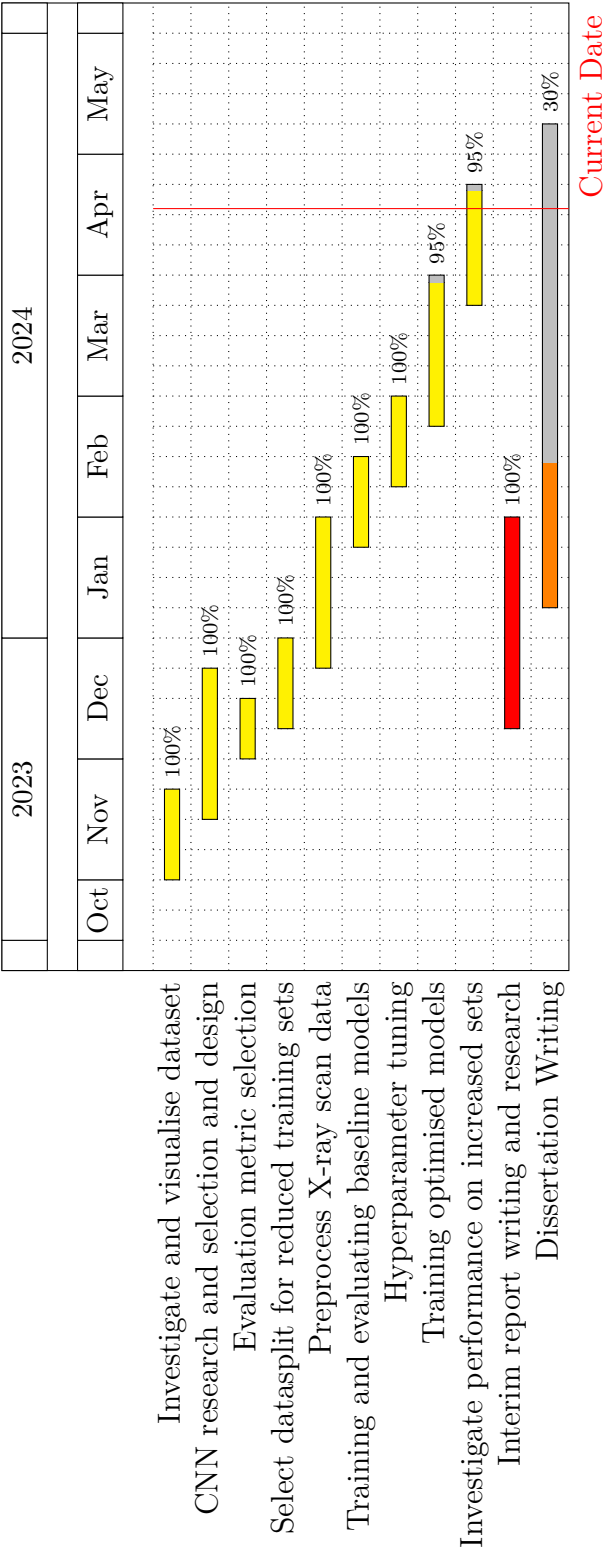


Figure 2.2: Project timeline presented as a Gantt chart.

2.3.2 Risk Analysis

At the start of the development of the project a risk analysis was produced (Botten, 2023), in order to ensure the project progressed smoothly with no unexpected interruptions, it outlines risks and potential mitigation strategies. The risk analysis in Table 2.1 builds upon those risks and makes minimal changes to mitigation strategies due to the progress of the project. To mitigate these risks further, bi-weekly meetings were held with the project supervisor to discuss problems, progression, ideas and delays.

Table 2.1: Risk Analysis and Mitigation Table.

Risk	Description	Probability	Impact	Mitigation
Limited availability of training data	Due to the nature of the project, a large amount of data is required for an effective model	Medium	High	Dataset has already been selected with a realistic number of training images. Other techniques such as data augmentation can be applied to artificially increase data diversity on the dataset.
Large amount of computational power required	CNN's require a large amount of hardware resources depending on the number of trainable parameters of the architecture. The amount of data also effects the amount of resources required.	Medium	Medium	Research and select architectures with a realistic number of trainable parameters for the scope of the project
Baseline model performance is unsatisfactory	The baseline performance of the network could be inadequate. This would make comparisons on increased amounts of data futile if the performance is already ineffective.	Medium	Medium	A literature has been performed with references to performances of other works for classification of COVID-19 from X-ray scans.
Data quality and variability.	Publicly available datasets may contain noise or biases that may impact model performance. There may be variability in the quality and characteristics of the chest X-ray across different image sources.	Medium	Medium	Apply data preprocessing techniques in an attempt to diminish these biases and variations in data.

Chapter 3

Literature Review

This chapter presents a comprehensive literature review that builds upon the interim report (Botten, 2024). It aims to explore early convolutional neural networks such as LeNet, to more modern architectures such as AlexNet and GoogLeNet. Additionally, the wide applicability of CNNs in fields beyond image classification, such as segmentation and object detection and the applications, vulnerabilities and mitigation strategies of CNNs have been examined.

3.1 Early Convolutional Neural Networks

Convolutional neural networks have revolutionised the space of computer vision, removing the need for the traditional two-stage approach to a complex classification problem. One of the first applications of CNNs, LeNet was introduced in the work by LeCun et al., 1998, for recognition of hand-written digits. Initially, computer vision techniques used the two-stage approach of first manual feature extraction (finding descriptive items or regions within images such as SIFT or SURF (Lowe, 2004; Bay et al., 2006)) using the raw data, which needed to be completed for each new task as it was domain-specific, then a training classifier, which is a generalised and trainable module, to generate scores (LeCun et al., 1998). These methods are now commonly used for simpler computer vision problems that can be deployed on power-efficient machines such as micro-controllers (Mahony et al., 2019). Instead of manual feature extraction, CNNs provide automatic feature extraction within its convolutional layers in combination with the fully connected layers (classifier), so that less expert analysis and fine-tuning are needed compared to traditional computer vision methods. CNNs

provide a higher accuracy at the cost of increased computational power and the requirement of larger amounts of annotated data (Mahony et al., 2019).

3.2 Convolutional Neural Networks Classification and ImageNet

In 2012, the AlexNet architecture proved to outperform traditional machine learning methods by 10.9% in top-5 error rate (Krizhevsky et al., 2012) during the ImageNet large-scale visual recognition challenge, a dataset containing 1.3 million images with over 100 classes (Krizhevsky et al., 2012), providing a remarkable breakthrough in the field of computer vision tasks. The success demonstrated by AlexNet inspired the development of complex and deeper architectures such as GoogLeNet in 2014, with 22 layers compared to AlexNet's 14. GoogLeNet introduced parallel convolutional layers for dimensionality reduction with feature concatenation, thus reducing the computational load whilst preserving important information, only at a minor cost of performance (C. Szegedy, Liu et al., 2015) but still improving upon its predecessor with a 6.67% top-5 error rate. Other architectures such as ResNet increased the number of layers to the hundreds by introducing residual blocks which contain skip connections (He et al., 2016). These skip connections bypass layers by adding the input of a layer directly to the function applied by the residual block, providing an alternate route (He et al., 2016). These residual blocks help alleviate the vanishing gradient problem which plagued networks with a larger number of layers (He et al., 2016), where the deeper the network, the harder and longer time it took to train. The ResNet network improved upon GoogLeNet with a top-5 error rate on ImageNet of 4.49% in its 152-layer network.

3.3 Applications and Challenges of Convolutional Neural Networks

So far the only applications of CNNs discussed were for the recognition of handwritten digits and ImageNet classification. Still, CNNs have been used to provide

efficient and accurate models in many domains. For example, in audio classification in Hershey et al., 2017, where they evaluate several CNN architectures' performance on classifying 70 million labelled video soundtracks whilst varying sizes of training sets and label vocabulary. The authors concluded that increasing the size of the training set above 700,000 videos improved performance minimally and a larger label set modestly improved performance but exponentially increased the time to train the network if a bottleneck layer is not present. In Hsieh and Kiang, 2020, they investigated the use of CNNs for land type classification using hyperspectral imagery data, proving they could perform well on 2 relatively small datasets, achieving 99.8% accuracy, concluding that CNNs perform worse on higher-dimensional data and superior when augmentation is applied. Other examples include weed segmentation and classification using SegNet, in Sa et al., 2018; various models were trained using a multitude of data splits and input channels in order to classify crop-weed sample images, deducing that there were negligible variations in performance among the options. Furthermore, they demonstrated that using pre-trained models on colour images, whilst fine-tuning the end layers of the model had a negligible effect on performance output; with the author suggesting that their images are too dissimilar to those found on the ImageNet dataset to provide an increase in performance.

Moreover, CNNs are commonly used as backbones in models for feature extraction, such as in work by Valappil and Memon, 2021. The approach consisted of using a CNN for feature extraction of an image and then passing those features to an SVM model for binary classification of vehicle or background objects; however, a small dataset of only 200 images was used. The author determined the performance of the model improved upon existing methods such as k-means clustering and KLT tracker by achieving an accuracy of 98% but concluded that their model needs to be trained on an extended dataset such as in work by Ke et al., 2018, where a model is trained on 20,000 images obtaining an accuracy of 95.8%, to show it's genuine performance.

Despite their success in many applications across computer vision tasks (Hsieh and Kiang, 2020; Hershey et al., 2017; Sa et al., 2018), CNNs face several challenges. CNNs are supervised learning models and one of the most significant challenges is the training data, which large amounts are required to achieve an effective and robust

model (Alzubaidi, Zhang et al., 2021); without this training data, models have a lack of information leading to underfitting (Sarvamangala and Kulkarni, 2022). Although adequate performance is shown in Sa et al., 2018, the author manually planted, used varying amounts of herbicides in a designated 40x40m area, and manually labelled samples explicitly for use in the project, obtaining approximately 450 samples, stating that there is an insufficient amount of training data for the model to achieve an optimal performance. Other challenges include imbalanced datasets, which can create a bias toward the majority class therefore negatively affecting the performance of the model on the minority class, and hardware resources, which are essential for training the CNNs, are immense and expensive (Sarvamangala and Kulkarni, 2022). Further challenges such as adversarial attacks, where small structured imputations in the input image cause CNNs to incorrectly classify images with the modified image imperceptible to the human eye (Szegedy et al., 2014). These imputations or noise can drastically reduce the performance of networks as shown in work by Geirhos et al., 2018, where VGG-16 dropped from 89% accuracy to 44% accuracy whilst humans outperformed networks with a 75% accuracy on an object classification task. These limitations can be mitigated by simple techniques such as transfer learning and data augmentation (Sarvamangala and Kulkarni, 2022), so they will be used in the project, however, adversarial attacks are harder to mitigate against and defence against these attacks creates models with slightly worse performance than conventional models (Madry et al., 2019).

3.4 Convolutional Neural Networks for Medical Imaging

CNNs have widespread use throughout the medical industry in domains such as disease diagnosis (Rajpurkar et al., 2017), segmentation (Abd-Ellah et al., 2018), and registration (De Vos et al., 2017). They have comparable results to expert clinicians and in some cases, like disease diagnosis, even outperform them (Shen et al., 2019).

Rajpurkar et al., 2017 used an improved DenseNet (Huang et al., 2018) architec-

ture with 121 convolutional layers called CheXNet for classifying 112,120 X-ray images containing fourteen types of diseases from the ChestX-ray14 (X. Wang et al., 2017) dataset. Medical datasets are frequently imbalanced with the negative/ordinary class being overrepresented (Sarvamangala and Kulkarni, 2022), as is with the ChestX-ray14 dataset, with one of the classes, 'hernia', only accounting for 0.2% and 'no finding' comprising 53% of images. Although the dataset was unbalanced, the model performed well in minority classes, achieving an impressive 0.8887 AUC score for pneumothorax and outperforming four radiologists on average across all classes, however, these metrics were gathered from a test set of only 420 images of which the class splits were not shown.

L. Wang et al., 2020 uses residual convolutional networks (He et al., 2016) as a baseline model and then uses generative synthesis to find optimal macro, and micro architectures, to classify COVID-19 and pneumonia from chest X-ray scans with their model, COVID-NET. The author comprised the largest dataset of images (at the time), COVIDx, by combining five different available datasets resulting in 13,000 images available, however only 358 of them COVID positive. The model was trained with data augmentation techniques and performed well on minority classes compared to contemporary architectures achieving a 98.9% precision value on the COVID-19 class. High precision values are essential for medical models as it ensures fewer false positives, thus less of a burden is placed upon healthcare facilities so patients are not admitted if they do not have a falsely diagnosed illness (L. Wang et al., 2020).

In Horry et al., 2020, the VGG architecture (Simonyan and Zisserman, 2015) with transfer learning was selected for diagnosing COVID-19 using multiple image modalities, X-ray, CT and ultrasound. The author carefully curated the dataset by removing images containing intrusive medical devices, which images commonly have in the medical industry. This reduced their dataset to 140 COVID-positive images for X-ray scans, 349 for CT scans and 399 for ultrasound scans. Data augmentation techniques were used to artificially increase the dataset to 2,920 COVID-positive X-ray images, 6,000 CT images and 2,720 ultrasound images, with the dataset being relatively balanced. Although the author lacked data originally, the network performed well, with ultrasound performing the best with 100% sensitivity compared

to 86% X-ray and 83% CT. An interesting finding was that the pre-trained model adjusted well toward the ultrasound scans, which appeared noisy and difficult to interpret to the human eye. The author stated the X-ray and CT modalities were challenging due to the lack of available data and high variability between images.

3.5 Data Scarcity in Medical Imaging

Both Rajpurkar et al., 2017, and L. Wang et al., 2020, used relatively small datasets whilst incorporating data augmentation techniques to improve the performance of their models, there seems to be a sincere lack of labelled data for specified domains (Sa et al., 2018; L. Wang et al., 2020; Rajpurkar et al., 2017). Conversely, general image classification has large publicly available datasets.

A multitude of factors contribute to data scarcity: annotation requires extensive time and effort from one or multiple experts to reduce human error (this is not limited to the medical domain (Sa et al., 2018)), sufficient cases are not available (Alzubaidi, Fadhel et al., 2020), patient variability in disease (Sarvamangala and Kulkarni, 2022), privacy concerns with identifiable information in patient images and lack of standardised data (Razzak et al., 2018), meaning datasets cannot be combined into larger ones. In essence, CNNs can perform modestly on small medical datasets by using approaches such as data augmentation and pre-training (Sarvamangala and Kulkarni, 2022) but are still not optimal. Other methods such as transfer learning can also improve performance, however, Alzubaidi, Fadhel et al., 2020, suggests that the data used for transfer learning should be from a similar domain in cases such as medicine in order to gain the most optimal network possible. This is due to the differences in lighting, shapes, resolution and colours of the images used in general image datasets such as ImageNet (Alzubaidi, Al-Amidie et al., 2021).

There is a lack of training data for optimal deep-learning applications with the time-consuming cost, and lack of experts to annotate images being major factors.

Chapter 4

Methodology & Experimental Design

This section will delve into the fundamentals of the ResNet architecture, covering the functionalities of each layer. Experimentation will be described where parameters such as learning rate, batch-size and optimiser will be evaluated, aiming to identify the most effective configurations for achieving optimal result for the project.

4.1 ResNet

The ResNet architecture was introduced in works by, He et al., 2016. The primary objective of the network was to create a deep architecture that was free of the vanishing gradient problem that plagued larger networks by introducing skip connections. These skip connections bypass one or more convolution layers allowing it to learn residual functions (difference between desired mapping of input data and its current representation). An identity function x is used to preserve the gradient which does not increase complexity within the network, thus no decrease is seen in computational performance (He et al., 2016; Adaloglou, 2020). An example of a skip connection can be seen in Figure 4.1.

The baseline architecture used in the project is ResNet-50 which is comprised of 49 convolution layers with a single fully connected layer.

4.1.1 Convolution Layers

The convolution layer is the most significant layer of the CNN architecture and is used for feature extraction. A small array called a kernel is applied across the input

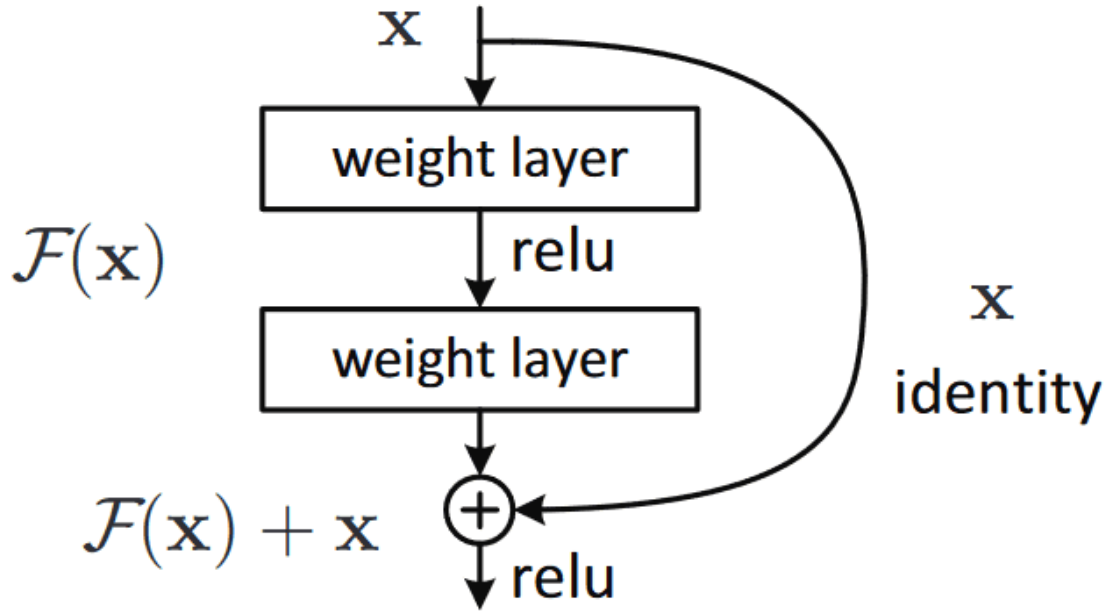


Figure 4.1: Skip connection (He et al., 2016).

tensor (an array of numbers) where the element-wise product is calculated between the kernel and the input, this is summed to obtain an output value for the output matrix, called a feature map (Yamashita et al., 2018). This process is repeated by sliding the kernel across the input tensor according to a stride (the number of pixels it moves across the input tensor) until the kernel cannot slide across any longer. The size of the kernel can be varied to sizes of 5x5 or 7x7 but is typically 3x3. Moreover, padding can be applied to the kernel, where values, generally zeros, are added to the edges of the input so as not to lose information around the boundary of the input. This ensures the feature map retains the same shape as the original input. An example of a convolution operation can be seen in Figure 4.2.

4.1.2 Activation Function

An activation function plays a crucial role in CNNs by converting the input signal into an output signal, which is subsequently passed to the next layer of the network (Sharma et al., 2017). Non-linear activation functions allow the CNN to learn non-linear relationships between the input and the output, therefore the network can capture more complex patterns and relationships. These functions are differentiable to allow the implementation of back-propagation in calculating errors or losses

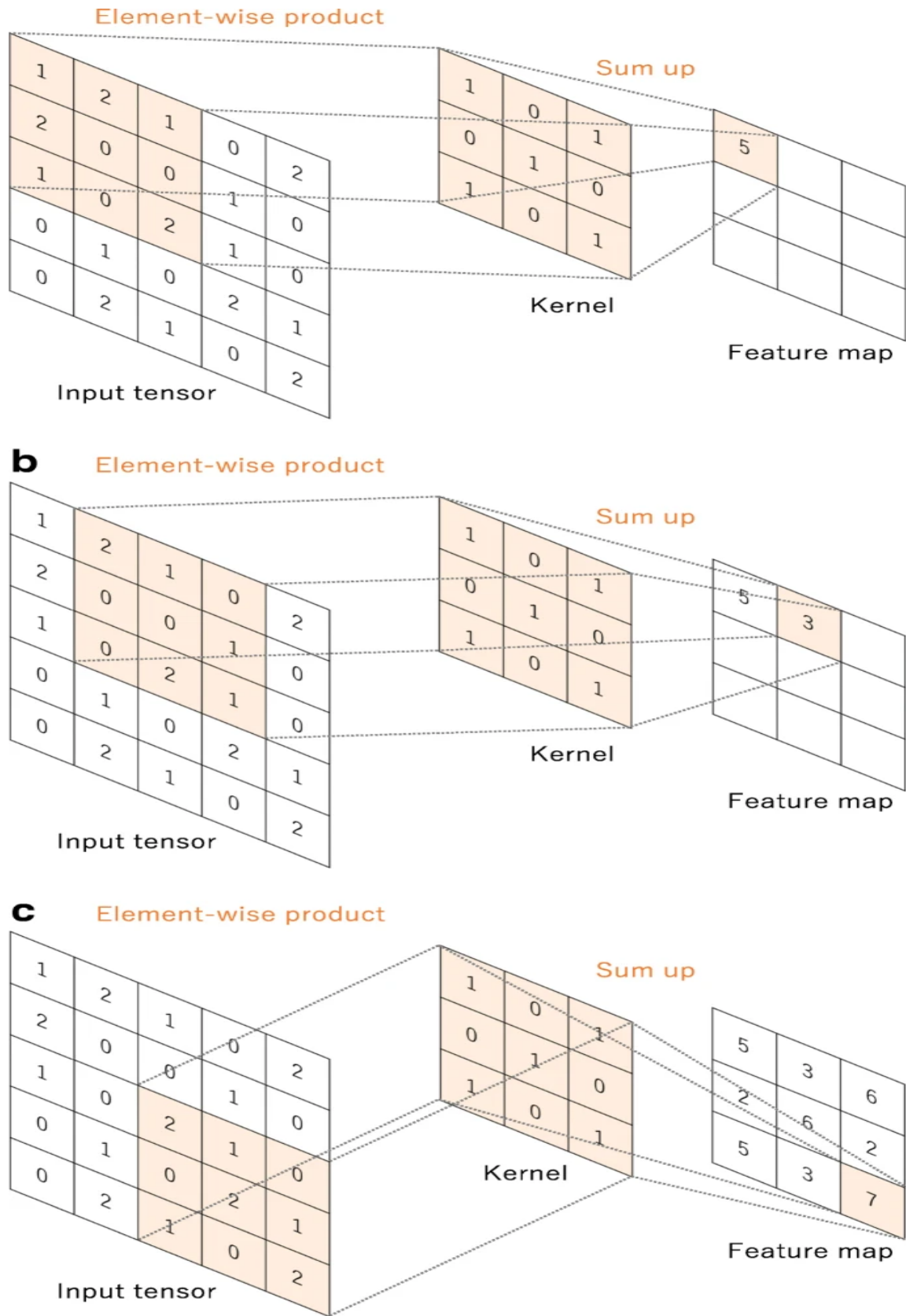


Figure 4.2: A convolution operation with a 3x3 kernel and stride of 1 with no padding applied (Yamashita et al., 2018).

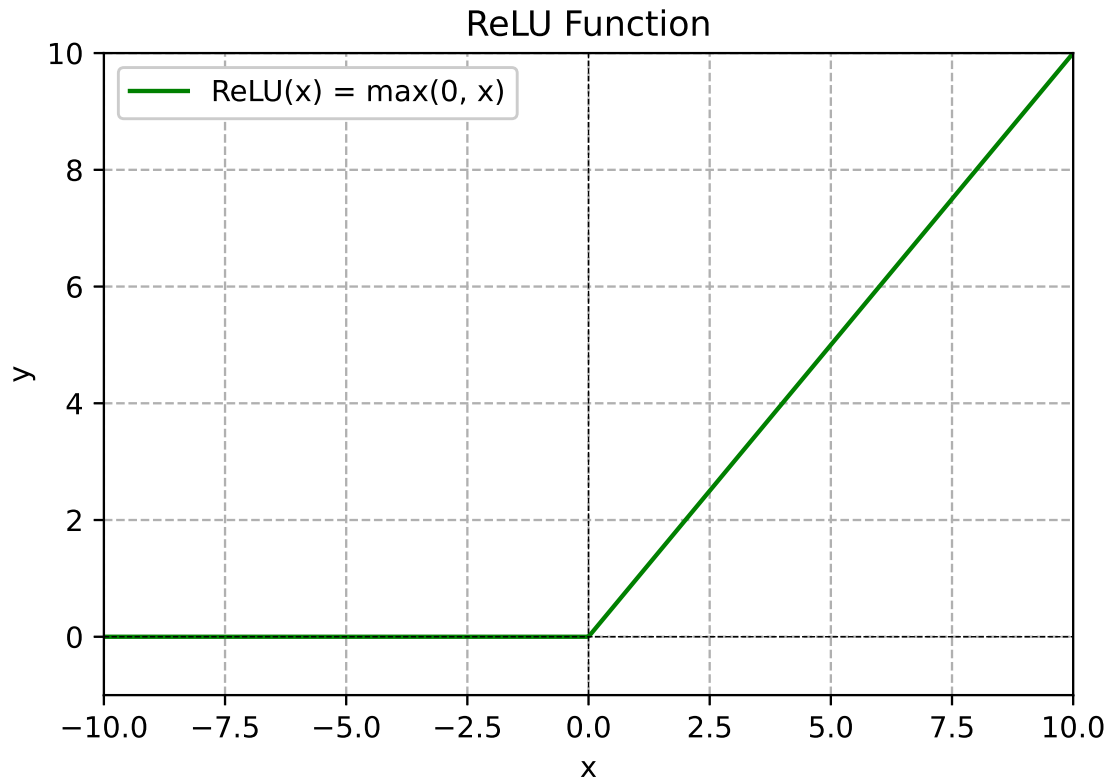


Figure 4.3: Graph of the ReLU activation function, demonstrating zero output for negative inputs and a linear response for positive inputs

relative to the weights, then adjusting these weights according to an optimisation algorithm such as gradient descent (Sharma et al., 2017).

In the ResNet network, the rectified linear unit (ReLU) is used and is defined as $f(x)_{\text{ReLU}} = \max(x, 0)$, shown in Figure 4.3. The ReLU is preferred over other activation functions such as sigmoid or tanh as it helps alleviate the vanishing gradient problem and provides faster computations. However, it has a significant limitation of sometimes 'dying' where weights will not be updated for that in neuron future learning (Nwankpa et al., 2018).

4.1.3 Batch Normalisation

The batch normalisation layer attempts to normalise the inputs at each layer within a network to a normal distribution. This layer's function is to address the problem of "internal covariate shift", where the distribution of each layer's inputs change during training, increasing training time (Ioffe and C. Szegedy, 2015).

The mini-batch mean and variance of the inputs are calculated. The input is then normalised by subtracting by the mean and dividing by the square root of the variance plus an ϵ constant to prevent division by 0 errors. This is then scaled by learnable parameters β and γ .

Normalising the inputs to layers helps keep the gradients in a more manageable range during backpropagation, leading to faster convergence. Due to this improved stability, higher learning rates can be used, reducing the time to train a network. This is largely because the normalisation helps prevent small changes in parameters from amplifying into larger sub optimal changes in gradients and parameters.

4.1.4 Pooling Layers

The task of the pooling layer is to sub-sample the feature maps, this creates smaller feature maps while retaining the important information (features) from the original maps produced by the convolution operation (Alzubaidi, Zhang et al., 2021). Shrinking the feature map reduces the computational power required to process the data (Bhatt et al., 2021). Similar to the convolution layer, a kernel size and stride are initialised from which the output value is calculated. A multitude of pooling methods are available, but only two will be discussed as they are used within the ResNet architecture.

One of the pooling methods used in the ResNet architecture is max pooling, which extracts the maximum value from the input feature map and drops the other values. The other pooling method used is average pooling, which extracts the average value from the input feature map and drops the remaining values. An example of which is shown in [Figure 4.4](#)

4.1.5 Dropout

The task of the dropout layer is to aid the generalisation of the model by zeroing random values during each epoch of training, meaning these connections are ignored during forward and backwards passes through the network. A dropout rate p is applied which represents the probability that a neuron will be dropped. This prevents

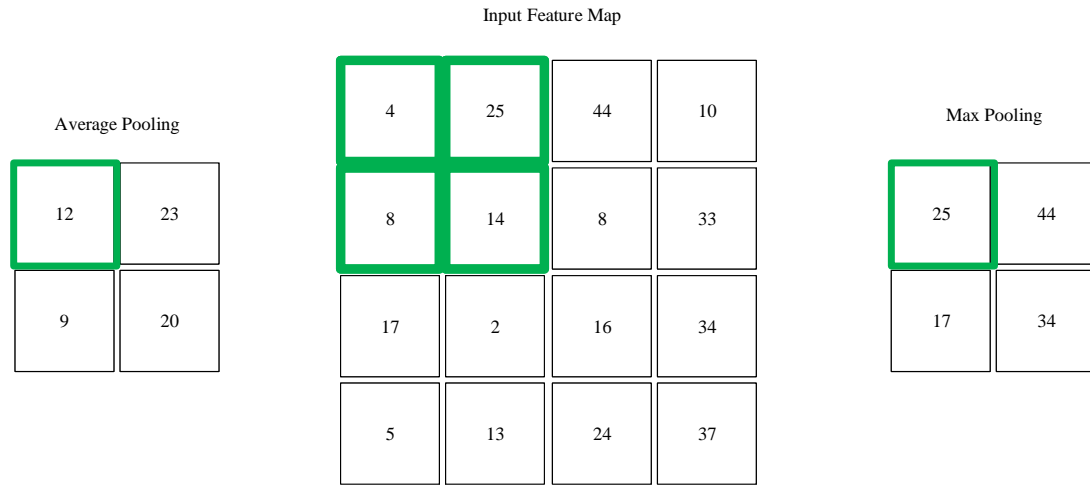


Figure 4.4: Example of average pooling and max pooling with a 2x2 matrix and stride of 2.

the model from learning specific features and relying on the presence of other neurons within the network, helping prevent overfitting (Hinton et al., 2012).

4.1.6 Fully Connected Layers

The fully connected layer is utilised as the classifier of the network, it is connected to the final convolution or pooling layer, which extracts the features of the images, these features are flattened to a one-dimensional array of numbers to be classified. Each input of this array is connected to each output by a weight which is learnable. The final fully connected layer generally has the same number of output nodes as the number of classes the user is attempting to classify.

4.1.7 Loss Function

The task of the loss function is to measure how accurately the model's predictions match the actual ground truth labels associated with the input data. This serves as a feedback mechanism for the model during training, with its main goal to minimise this loss by adjusting parameters to improve the accuracy of the model.

The loss function used in the project was the cross-entropy loss in which the output

of the function is a probability between 0 and 1, it is considered the standard metric for training classification models across deep learning applications (Lucena, 2022).

4.1.8 Optimiser

An optimiser algorithm aims to find the most optimal model weights by adjusting them in order to minimise a loss function - this helps the model produce more accurate predictions (Fauziah and Gunawan, 2023). The two optimisers experimented with in this project were Stochastic Gradient Descent (SGD) with momentum and Adaptive Moment Estimation (Adam) (Kingma and Ba, 2017).

SGD takes a random sample from the input data and updates the gradient based on the samples. It aims to find the global minimum by moving in the opposite direction of the gradient of the loss function $J(\theta)$, scaled by the learning rate, η with respect to the current weights θ_i (Fauziah and Gunawan, 2023; Haji and Abdulazeez, 2021).

$$\theta_{i+1} = \theta_i - \eta \cdot \Delta J(\theta_i)$$

While SGD is a simple and effective solution it can struggle with saddle points and noisy gradients, hence a parameter, momentum is introduced in order to accelerate the gradient descent towards reducing the loss function through a velocity vector along various iterations (Dogo et al., 2022; Sutskever et al., 2013). The parameters are then updated accordingly.

$$\nu_{i+1} = \mu\nu_i - \eta \cdot \Delta J(\theta_i)$$

$$\theta_{i+1} = \theta_i + \nu_{i+1}$$

Where μ is the momentum coefficient between 0 and 1 and $\Delta J(\theta_i)$ is the gradient at θ_i .

The other optimiser, Adam, calculates adaptive learning rates for each parameter within a network, so each parameter has a different learning rate. Instead of the

learning rate being a hyper-parameter, it varies over time during training. This method stores the exponentially decaying average of past squared gradients in order to adjust the learning rate, whilst also storing the exponentially decaying average of past gradients, similar to momentum. The update rules for Adam are as follows (Kingma and Ba, 2017; Kandel and Castelli, 2020; Ruder, 2017):

1. Calculate gradient of loss function with respect to parameters.

$$g_i = \Delta J(\theta_i)$$

2. Update biased first moment estimate.

$$m_{i+1} = \beta_1 m_i + (1 - \beta_1) g_i$$

3. Update biased second moment estimate

$$v_{i+1} = \beta_2 v_i + (1 - \beta_2) g_i^2$$

4. Correct bias in first moment.

$$\widehat{m}_{i+1} = \frac{m_{i+1}}{1 - \beta_1^i}$$

5. Correct bias in second moment.

$$\widehat{v}_{i+1} = \frac{v_{i+1}}{1 - \beta_2^i}$$

6. Update the parameters

$$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{\widehat{v}_{i+1} + \epsilon}} \widehat{m}_{i+1}$$

β_1 and β_2 are exponential decay rates and ϵ is a small constant to avoid dividing by 0 errors. The advantages of Adam include memory and computational efficiency whilst also performing better on high dimensional problem spaces and those with sparse gradients such as computer vision problems (Kingma and Ba, 2017).

4.2 Datasets

Kaggle hosts publicly available datasets some for use in computer vision competitions. Some of these datasets are reliable and selected by researchers within medical

domains. There were only two datasets available for use within the project that had a sufficient amount of data.

- COVID-19 Radiography Database (Chowdhury et al., 2020; Rahman et al., 2021)
- COVID-CXR4 (L. Wang et al., 2020)

The COVID-19 Radiography Database (CRD) features purely frontal view chest X-ray images collected from multiple sources. These images have all been standardised to a resolution of 256×256 pixels, with the data split being slightly more balanced in comparison to the COVID-CXR4 dataset as shown in Table 4.1. The images are in '.png' format and are separated into directories based on class, COVID-19 positive and normal, so a separate label file must be manually generated with filename and class. Whilst the images are standardised and there is little variance between them, it is the smaller of the two datasets and may not have enough COVID-positive samples without using methods such as data augmentation or resampling.

Table 4.1: Datasplit for each dataset of COVID-19 Images

Dataset	COVID	Normal	Total
Chowdhury et al., 2020; Rahman et al., 2021	3,616	10,192	13,808
L. Wang et al., 2020	65,681	19,137	84,818

The COVID-CXR4 database contains approximately six times as many images as the previous dataset and is comprised of 9 different datasets, including the images from the previous dataset. Due to the multiple other sources of the dataset, images vary in size, contrast, view and invasive medical devices such as wires are present, thus the visibility is extremely different among the dataset as shown in Figure 4.5. The image sizes vary drastically in this dataset with the lowest being 157×156 pixels whereas the largest and most occurring image size is 1024×1024 pixels. The images are in '.jpg' image format, with the labels file containing the columns 'patient id', 'filename', 'class' and 'data source'. The labels are in a readable format with little data cleaning needing to be completed.

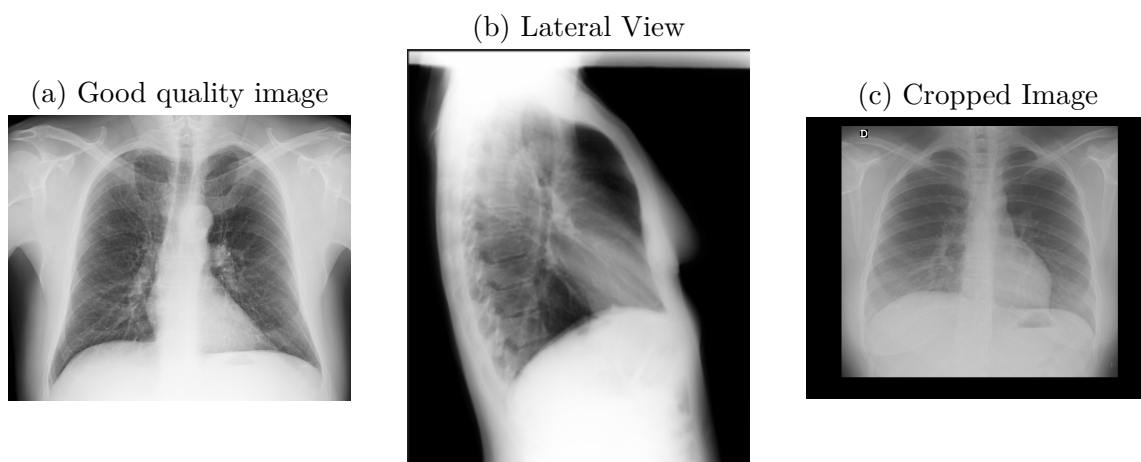


Figure 4.5: Different variations of images observed in the dataset.

Table 4.2: Class distributions across dataset split

Type	Positive	Negative	Total
Train	57199	10664	67863
Validation	4241	4232	8473
Test	4241	4241	8482

Ultimately, this dataset was chosen. Although the dataset does contain a wider variety in image sizes, contrast and view, it is more representative of real-world medical scenarios the network would be used in - where not all institutions have a standardisation of an x-ray scan procedure, thus images are varied.

4.3 Experiments

Once downloaded, the dataset is pre-split into train, test and validation folders; of which the class distributions are shown in [Table 4.2](#). As observed in the provided dataset, the training set is heavily skewed towards the positive samples. This imbalance can lead the model to develop a bias toward the majority class, potentially resulting in poor predictive performance on the minority class, thus the proportion of positive and negative samples will be varied among configurations. On all of these training splits, experimentation will be performed on the same validation and test set.

The validation and testing set both have equal class distributions so should allow for

a fair assessment of the model’s ability to generalise across both classes, even if the training set is in a 5:1 ratio of positive to negative. Moreover, the data is split in an 80:10:10 ratio providing a sufficient allocation of data for each subset. Therefore, this data split will be kept for the remaining testing in the project.

Additionally, the training set should be split into different configurations in order to meet the main aim of the project, as shown in [Table 4.3](#). This approach aims to understand how changes in the training dataset size affect the model’s ability to classify samples of data effectively. These will be referred to as the initial training set splits.

Table 4.3: Initial training set datasplit

Training Set	Positive	Negative	Total
Split 1	5000	10664	15664
Split 2	10000	10664	20664
Split 3	20000	10664	30664
Split 4	40000	10664	50664
Split 5	57199	10664	67863

Furthermore, due to the performance results of the data splits in [Table 4.3](#), further smaller configurations were tested for an extended analysis which can be seen in [Table 4.4](#). These will be referred to as further training set splits.

Table 4.4: Further training set datasplit

Training Set	Positive	Negative	Total
Split 6	8000	8000	16000
Split 7	4000	4000	8000
Split 8	2000	2000	4000
Split 9	1000	1000	2000
Split 10	500	500	1000
Split 11	250	250	500
Split 12	125	125	250

4.4 Hyperparameters

In order to obtain the most optimal model possible, hyperparameter tuning and model architecture experiments were conducted. This testing explored the choice of the optimiser, batch size, learning rate and addition of dropout layers in the model architecture and how these impact the model's performance. The following parameters will be tested.

- Batch Sizes: 16, 32, 64, 128.
- Optimisers: SGD with momentum and Adam.
- Learning Rates: 0.001, 0.0001, 0.00001.
- Dropout: 0.25, 0.5, 0.75.

All models should be pre-trained and instantiated with ImageNet weights, making use of transfer learning. The weights within the network will be frozen other than the FC layer which will be trainable, significantly reducing the time to train the network whilst providing good performance.

4.4.1 Evaluation Metrics

Evaluation metrics are used to assess the performance of a model during training and the generalisation ability of a model on unseen data. Additionally, these metrics are used for model selection during training such that the most optimal classifier is selected, among multiple trained classifiers, on a validation set (unseen data).

The metric used for model selection and testing throughout the project was accuracy, this is defined as the fraction of correctly classified predictions from all predictions the model has made.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

Where TP is True Positives, TN is True Negatives, FP is False Positives and FN is

False Negatives. Accuracy can be misleading in the case of imbalanced datasets for example: if 90% of the data is from one class, the model may only predict the majority class for all inputs and still achieve a high accuracy despite not learning to classify the minority class correctly. Therefore when selecting the testing and validation split careful consideration must be made to ensure a balanced distribution of classes. This was taken into account when choosing to keep the pre-split dataset, however, the pre-split data already had properly representative testing and validation test sets, so accuracy was kept.

Chapter 5

Implementation

In this chapter, the practical aspects of deploying a CNN model will be discussed, including the hardware used and software configurations used, to allow for reproducible experiments. Other aspects such as data handling, cleaning and encoding, which are critical in preparing the dataset for use within the CNN model, will be discussed. Furthermore, the custom dataset configurations, implementation of model adjustments and hyperparameter tuning, will be detailed.

5.1 Hardware, Toolsets and Environments

The models used in the project were all trained on one of two machines - the difference in hardware components should not make a difference to the results achieved by the model. The hardware configuration should only affect the time taken for the model to train. The specifications for each machine are shown below.

1. RTX 3070 Machine

- CPU: Ryzen 5 5600X, 6-core/12 thread @3.70GHz
- GPU: NVIDIA RTX 3070
 - Dedicated GPU Memory: 8GB
 - Shared GPU Memory: 16GB
 - GPU Memory: 24 GB
- RAM: 32GB @ 3200 MHz

2. GTX 1650 Machine

- CPU: i5-9300H, 4-core/8 thread @ 2.40GHz
- GPU: NVIDIA GTX 1650 Max-Q
 - Dedicated GPU Memory: 4GB
 - Shared GPU Memory: 8GB
 - GPU Memory: 12 GB
- RAM: 16GB @ 2667 MHz

5.1.1 Programming Language and Libraries

The programming language selected was Python, it provides the machine learning capabilities needed for the project in multiple different frameworks such as PyTorch, TensorFlow and Keras. Python also provides data visualisation and analysis libraries used within the project such as Matplotlib, Tensorboard, and Pandas.

The chosen framework to implement the CNN was PyTorch. PyTorch is an easy to learn and accessible machine learning library, providing extensive support for CUDA, NVIDIA's parallel computing platform, for GPU acceleration allowing significant speedup of the training process of CNNs. This feature makes it highly suitable for handling the computationally intensive tasks typical in deep learning. The ease of use was extremely appealing as this was the author's first project using a deep-learning framework.

All code was completed within Visual Studio Code (VS Code), and training was completed using Jupyter Notebooks. These notebooks offer a versatile environment suitable for iterative testing such as in this project, where cells (code blocks) are executed individually and allow for extensive data visualisation with the aforementioned libraries.

To ensure version matching between machines, an environment and package manager, Anaconda, was used in which all libraries were installed.

1. Python 3.11.5

- An environment is created, with this Python version being compatible with the latest version (at the time) of PyTorch.

```
1 conda create -n py3115 python=3.11.5
```

2. PyTorch

- PyTorch and other torch libraries are installed, CUDA version 12.1 is also installed so GPU acceleration can be enabled for use.

```
1 conda install pytorch=2.1.0 torchvision=0.16.0 torchaudio
  =2.0.0 pytorch-cuda=12.1 -c pytorch -c nvidia
```

3. Tensorboard and JupyterLab

- Tensorboard allows for live data visualisations used to track metrics for experiments, JupyterLab is needed for the notebook functionality.

```
1 conda install -c conda-forge jupyterlab tensorboard
```

4. Scikit-learn

- Scikit-learn is used to generate the accuracy metric for model selection and testing.

```
1 conda install -c conda-forge scikit-learn
```

5.2 Cleaning and Encoding

Although the dataset is mostly clean, some columns such as 'patient id' and 'data source' should be removed as they are unnecessary to the model. The labels of each image, 'positive' and 'negative' should also be encoded so the CNN model can correctly handle the dataset (see [Listing 5.1](#)).

```
1 def encode_labels(input_file_path, output_file_path):
2
3     with open(input_file_path, 'r') as input_file:
4         lines = input_file.readlines()
5
6     encoded_lines = []
```



```

7     for line in lines:
8         image_name, label = line.strip().split()
9         # Encode 'positive' as 1 and 'negative' as 0
10        encoded_label = '1' if label.lower() == 'positive' else '0'
11        encoded_lines.append(f'{image_name} {encoded_label}\n')
12
13    with open(output_file_path, 'w') as output_file:
14        output_file.writelines(encoded_lines)

```

Listing 5.1: Encoding labels in dataset.py

5.3 Augmentation

Data augmentation techniques were applied to the dataset with the use of the transforms module from torchvision. This helps fulfil the objectives set out in [section 2.1](#), and prevents the model from overfitting to the dataset. The images are initially resized to ensure that training does not take an extreme length of time, random rotation, flipping and colour jitter are applied. The testing and validation set had no data augmentation techniques applied to them, only being resized to 232×232 and converted to a tensor.

```

1 from torchvision import transforms
2 transform_train = transforms.Compose([
3     transforms.Resize((232, 232)),
4     transforms.CenterCrop(224),
5     transforms.RandomRotation(10),
6     transforms.RandomHorizontalFlip(0.2),
7     transforms.RandomApply([transforms.ColorJitter(brightness
8     =0.25, contrast=0.25, saturation=0.25, hue=0.25)], p=0.5),
9     transforms.ToTensor()])

```

Listing 5.2: Data augmentation applied to training set

5.4 Custom Dataset and CNN Model

In order for the dataset to be loaded into memory, it must initially be converted into a PyTorch dataset using the 'Dataset' class. The folder structure of the dataset can be seen in [Figure 5.1](#).







 test	20/02/2024 01:41	File folder	
 train	20/02/2024 02:01	File folder	
 val	20/02/2024 02:03	File folder	
 encoded_test.txt	11/02/2024 00:44	Text Document	510 KB
 encoded_train.txt	11/02/2024 00:45	Text Document	4,081 KB
 encoded_val.txt	11/02/2024 00:42	Text Document	452 KB

Figure 5.1: Structure of dataset folder

A custom dataset class is created called 'COVIDDataset' and each PyTorch dataset class must implement three methods, the initialisation of the dataset (folders, labels), the length of the dataset, and fetching items of data from the dataset, see [Listing 5.3](#).

```
1 class COVIDDataset(Dataset):
2     def __init__(self, txt_file, root_dir, transform=None):
3         self.annotations = pd.read_csv(txt_file, header=None, sep =
4         ', ')
5         self.root_dir = root_dir
6         self.transform = transform
7
8     def __len__(self):
9         return len(self.annotations)
10
11     def __getitem__(self, idx):
12         img_path = os.path.join(self.root_dir, self.annotations.iloc
13         [idx, 0])
14         image = Image.open(img_path).convert("RGB")
15         label = int(self.annotations.iloc[idx, 1])
16
17         if self.transform:
18             image = self.transform(image)
19         return image, label
```

Listing 5.3: Custom dataset class definition

The root folder path, text file path and transforms must be defined in the main Python/Jupyter Notebook file and passed to the dataset class when created. Transforms are also applied to the images too, the code is implemented in [Listing 5.4](#).

```
1 from dataset import COVIDDataset
2
3 train_imgs = COVIDDataset(txt_file='dataset/encoded_train.txt',
    root_dir='dataset/train', transform=transform_train)
```

Listing 5.4: Train images in a custom dataset class

Next, the CNN model should be instantiated with the ImageNet pretrained weights from PyTorch, which improves upon the model from the original paper, and these weights should be frozen to ensure only the classifier is being trained. Furthermore, this is where the model architecture experimentation will be completed, where the model's fully connected layer can be varied, this is shown in [Listing 5.5](#).

```
1 model = models.resnet50(weights=ResNet50_Weights.DEFAULT)
2 for param in model.parameters():
3     param.requires_grad = False
4
5 num_ftrs = model.fc.in_features
6
7 model.fc = nn.Linear(num_ftrs, 2)
```

Listing 5.5: CNN Model and Freeze Weights

5.5 Training and Validation

The model's training is defined in a function 'train_one_epoch' which takes in the current epoch and a Tensorboard writer instance for use in live data logging (see [Listing 5.6](#)). The training function processes batches of data from the training data loader where each batch consists of inputs and their corresponding labels. The gradients from the previous batch are cleared to prevent accumulation as to prevent incorrect parameter updates. Loss is calculated and the parameters are updated, then the predictions are generated and stored for accuracy calculations.

```

1 def train_one_epoch(epoch_idx, tb_writer):
2     running_loss = 0.0
3     all_labels = []
4     all_preds = []
5
6     for i, data in enumerate(train_loader):
7         inputs, labels = data
8         inputs, labels = inputs.to(device), labels.to(device)
9         optimizer.zero_grad()
10        outputs = model(inputs)
11        loss = criterion(outputs, labels)
12        loss.backward()
13        optimizer.step()
14        preds = torch.argmax(outputs, 1)
15        all_labels.extend(labels.cpu().numpy())
16        all_preds.extend(preds.cpu().numpy())
17        running_loss += loss
18
19        #Log loss every 10 batches
20        if i % 10 == 9:
21            last_loss = running_loss / 10
22            print(f'Batch {i+1} Loss: {last_loss}')
23            tb_x = epoch_idx * len(train_loader) + i + 1
24            tb_writer.add_scalar('Loss/train per batch', last_loss,
25                                tb_x)
26
27            running_loss = 0.0
28
29        print(f'Training accuracy: {accuracy_score(all_labels, all_preds
30            )}')
31
32        tb_writer.add_scalar('Accuracy/train', accuracy_score(all_labels
33            , all_preds), epoch_idx+1)
34
35    return last_loss

```

Listing 5.6: Training function

Once the model is finished training, it is set to evaluation mode and as such, deactivates layers like dropout during inference. The validation loop follows the same

pattern by iterating over a dataloader but with gradient calculations disabled in order to save memory. The model is conditionally saved if the current validation loss is the lowest observed loss or if the current accuracy is the highest, ensuring that the best performing models are preserved. The implementation is shown in [Listing 5.7](#)

```
1 for epoch in range(EPOCHS):
2     print(f'Epoch {epoch+1}/{EPOCHS}')
3     model.train()
4     epoch_loss = train_one_epoch(epoch, tb_writer)
5     running_vloss = 0.0
6     all_labels = []
7     all_preds = []
8     with torch.no_grad():
9         for i, vdata in enumerate(val_loader):
10             vinputs, vlabels = vdata
11             vinputs, vlabels = vinputs.to(device), vlabels.to(device)
12             voutputs = model(vinputs)
13             vloss = criterion(voutputs, vlabels)
14             running_vloss += vloss
15             preds = torch.argmax(voutputs, 1)
16             all_labels.extend(vlabels.cpu().numpy())
17             all_preds.extend(preds.cpu().numpy())
18
19     avg_vloss = running_vloss / (i+1)
20     acc = accuracy_score(all_labels, all_preds)
21     if avg_vloss < best_vloss:
22         best_vloss = avg_vloss
23         print('Saving at epoch ', epoch_number+1, 'for loss')
24         model_path = (f'{epoch_number+1}_INSERT_MODEL_NAME.pt')
25         torch.save(model.state_dict(), model_path)
26     if acc > best_acc:
27         best_acc = acc
28         print('Saving at epoch', epoch_number+1, 'for accuracy')
29         model_path = (f'{epoch_number+1}_INSERT_MODEL_NAME.pt')
30         torch.save(model.state_dict(), model_path)
```

Listing 5.7: Validation loop

5.6 Model and Hyperparameter Configuration

Hyperparameters are extremely important in the development of CNN models as these directly influence their performance and efficiency. Proper tuning of these parameters can significantly enhance the model’s accuracy, prevent overfitting and ensure convergence during training. This is completed by testing a range of hyperparameters to find the most effective combinations that can be used to optimise the model’s performance for a given task.

All testing for model architecture selection and hyperparameter selection was completed using the balanced split of data, 10k COVID positive images and 10k COVID negative images. The testing starts by using the default ResNet50 model with the final FC layer adjusted to two outputs to match the number of classes unless otherwise stated. The Adam optimiser is used with β ’s of 0.9 and 0.999 and SGD is always paired with a μ (momentum) value of 0.9.

5.6.1 Batch Size

Batch size denotes the number of training samples that will be passed through the network during training to make a single update to the parameters (Radiuk, 2017). Larger batch sizes provide a more accurate estimate of the gradient but with less frequent updates whilst being more computationally demanding in memory, whereas smaller batch sizes update more frequently but also introduce more noise into the gradient (Lin, 2022). Optimisers such as Adam and SGD use mini-batch sizes, usually powers of 2, to make these parameter updates. Other optimisers such as gradient descent, calculate the loss and gradient using the entire dataset, making updates extremely computationally expensive, especially with larger datasets.

The batch size is changed in the dataloader of PyTorch. The ‘shuffle’ parameter ensures the data is sufficiently shuffled after each epoch to prevent overfitting, and the ‘num_workers’ parameter denotes how many sub-processes to use for data loading, speeding up the process, see [Listing 5.8](#).

Table 5.1: Results of different batch size on the test set with Adam Optimiser

Batch Size	Learning Rate	Optimiser	Accuracy on Test Set
128	1×10^{-3}	Adam	79.500%
64	1×10^{-3}	Adam	78.650%
32	1×10^{-3}	Adam	78.825%
16	1×10^{-3}	Adam	78.550%

Table 5.2: Results of different batch size on the test set with SGD Optimiser

Batch Size	Learning Rate	Optimiser	Accuracy on Test Set
128	1×10^{-3}	SGD	79.865%
64	1×10^{-3}	SGD	78.725%
32	1×10^{-3}	SGD	79.100%
16	1×10^{-3}	SGD	78.625%

```
1 train_loader = DataLoader(train_imgs, batch_size=128, shuffle=True,
    num_workers=6)
```

Listing 5.8: Data loader

The results of changing the batch size are shown in [Table 5.2](#) and [Table 5.1](#). Generally, as the batch size decreases so does the accuracy of the training set, however, the differences between them are extremely minimal. The batch size of 128 on both Adam and SGD scored the highest on the test set with 79.500% and 79.865% respectively, suggesting SGD may be marginally more effective as an optimiser at this learning rate.

5.6.2 Optimiser and Learning Rate

The learning rate details the size of updates made to the model's weights at each iteration while aiming to minimise the loss function. A properly set learning rate enables efficient convergence to this minimum whereas a high learning rate may overshoot the optimal values. Conversely, a learning rate that is too low can result in a significant increase in time to train the model before reaching optimal performance. Hence, attempts will be made to find the optimal learning rate.

Due to the results found when varying batch sizes, the best performing batch size, 128, was selected and used for further testing. Next, the learning rate was varied

Table 5.3: Results of varying learning rates on the test set

Batch Size	Learning Rate	Optimiser	Accuracy on Test Set
128	1×10^{-4}	Adam	80.325%
128	1×10^{-4}	SGD	76.675%
128	1×10^{-5}	Adam	77.550%
128	1×10^{-5}	SGD	70.250%

Table 5.4: Results of the addition of dropout layers

Batch Size	Learning Rate	Optimiser	Dropout	Accuracy on Test Set
128	1×10^{-4}	Adam	0	80.325%
128	1×10^{-4}	Adam	25	80.500%
128	1×10^{-4}	Adam	50	80.125%
128	1×10^{-4}	Adam	75	80.275%

among models, the results of which can be seen in [Table 5.3](#). The performance of the model increases when the learning rate is decreased to 1×10^{-4} using the Adam optimiser, however, a subsequent decrease to 1×10^{-5} negatively impacts the performance of the model. On the other hand, SGD shows a significant drop in performance as the learning rate decreases, suggesting that SGD could require a higher learning rate to converge effectively.

5.6.3 Dropout

The best configuration was taken from the previous tests, Adam with learning rate 1×10^{-4} , for further testing. Next, the addition of dropout within the FC layer was explored. The implementation for the addition of dropout layers can be seen in [Listing 5.9](#).

```

1 model.fc = nn.Sequential(
2     nn.Linear(num_ftrs, 512),
3     nn.Dropout(0.25),
4     nn.Linear(512, 2))

```

Listing 5.9: Dropout layer addition

In [Table 5.4](#) the results show the effect of the addition of dropout layers. These seemed to only marginally increase the model's accuracy on the test set, with higher

Table 5.5: Results of the addition of a ReLU after the linear layer

Batch Size	Learning Rate	Optimiser	Dropout	ReLU in FC Layer	Accuracy on Test Set
128	1×10^{-4}	Adam	0	Yes	82.500%
128	1×10^{-4}	Adam	25	Yes	84.500%
128	1×10^{-4}	Adam	50	Yes	83.925%
128	1×10^{-4}	Adam	75	Yes	83.875%

values of dropout providing worse results than with 0 dropout. The most effective model with 25% dropout achieved 80.500% accuracy on the test set, only minimally increasing over the network without dropout.

5.6.4 ReLU

In the previous subsections the best configuration was taken from the previous testing and used for further testing, however, due to the results being so marginal in the addition of dropout, these configurations were tested further with the addition of a ReLU activation function before the dropout layer (see [Listing 5.10](#)).

```

1 model.fc = nn.Sequential(
2     nn.Linear(num_ftrs, 512),
3     nn.ReLU(),
4     nn.Dropout(0.25),
5     nn.Linear(512, 2))

```

Listing 5.10: Dropout layer addition

The results in [Table 5.5](#) show that incorporating a ReLU consistently improves the model across all configurations compared to the previous results. The best result, an accuracy of 84.500% was achieved with the 25% dropout configuration. Increasing the amount of dropout did show a slight drop off in performance but this was extremely marginal; 0.575% difference compared with 50% dropout and 0.625% compared with 75% dropout, following similar results to the experiment with the addition of dropout.

Shown in [Figure 5.2](#), is the training and validation loss for the default ResNet model with batch size 128 and Adam optimiser with 1×10^{-4} learning rate, and one with

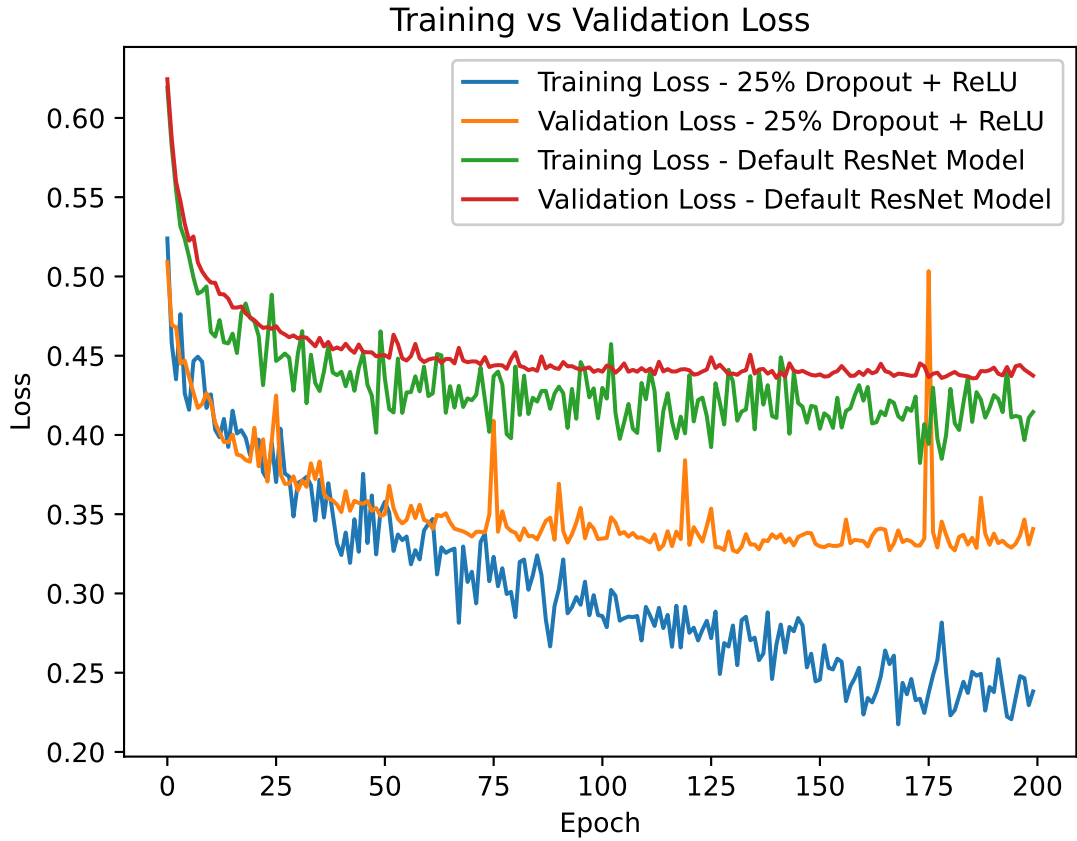


Figure 5.2: Default ResNet Model vs 25 Dropout + ReLU Model

25% dropout with a ReLU in the FC Layer - both with the same aforementioned parameters. Both the training and validation loss for the modified architecture are significantly lower than that of the default model. Moreover, the training and validation loss for the default model starts moving sideways around epoch 85 suggesting further training does not significantly improve the performance of the model. In contrast, for the modified model, the training loss constantly decreases whilst the validation loss starts to move sideways at around epoch 125. There is no obvious increase in validation loss for both models, suggesting that overfitting has not occurred, but the models do seem to stagnate in learning new features, possibly due to the current configuration of hyperparameters. Therefore, the number of epochs each model in the training split configurations will be trained on will be 200, as this leaves ample room for the model to improve and converge.

Chapter 6

Results & Discussion

This section will analyse the outcomes of the experiments proposed in [section 4.3](#). Given the class imbalance, this chapter aims to explore how the configuration of data splits influences the model’s ability to generalise and create an effective model. All models in this section use the same validation and test set described in [Table 4.2](#) unless otherwise stated. All of these models were trained on the RTX 3070 machine.

6.1 Initial Training Set Splits

Due to the previously completed testing, the best performing model architecture and hyperparameters were taken and used for the main experiments of this project.

- Batch Size: 128
- Optimiser: Adam
- Dropout: 25%
- ReLU in FC Layer: Yes
- Number of Epochs: 200

The training data is split as discussed in [Table 4.3](#).

[Table 6.1](#) shows the results of the initial training set experiments. Split 1, which had the lowest number of positive samples, resulted in the lowest accuracy of 82.575% and Split 3, performed the best with 84.500%. Interestingly, Split 2, the balanced split did not perform the best. Subsequent increases in class imbalance do affect the

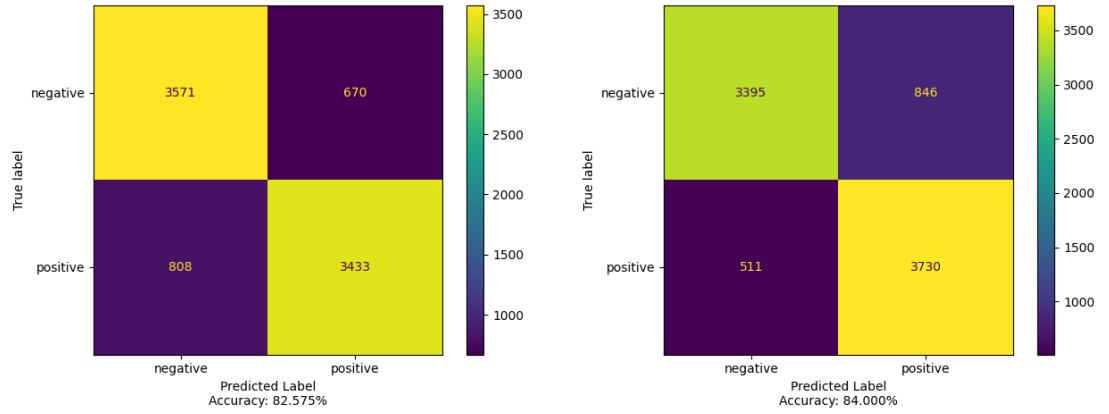
Table 6.1: Results of the different training configurations on the test set

Training Set	Accuracy on Test Set	Training Time (s)
Split 1	82.575%	15707
Split 2	84.000%	17622
Split 3	84.500%	22130
Split 4	84.025%	31493
Split 5	83.775%	38880

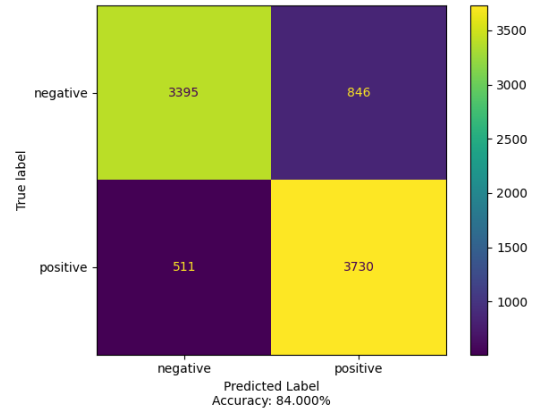
model’s accuracy on the test set negatively, but not as significantly as expected. The model still performs as a good classifier even on imbalanced data.

The result of increasing the number of images the model trained on significantly increased the time to train on the hardware setup. Split 2, trained on 20,000 images took only 17622 seconds to train (approximately 5 hours) achieving an 84.000% accuracy whereas, Split 3, trained on 30,000 images, took 22130 seconds (approximately 6 hours) to train, achieving a .500% increase in accuracy. It seems like there are diminishing returns between splits, where the increase in images requires longer training time but only results in marginally better accuracy until a point where the accuracy starts to decrease again but the models take significantly longer to train.

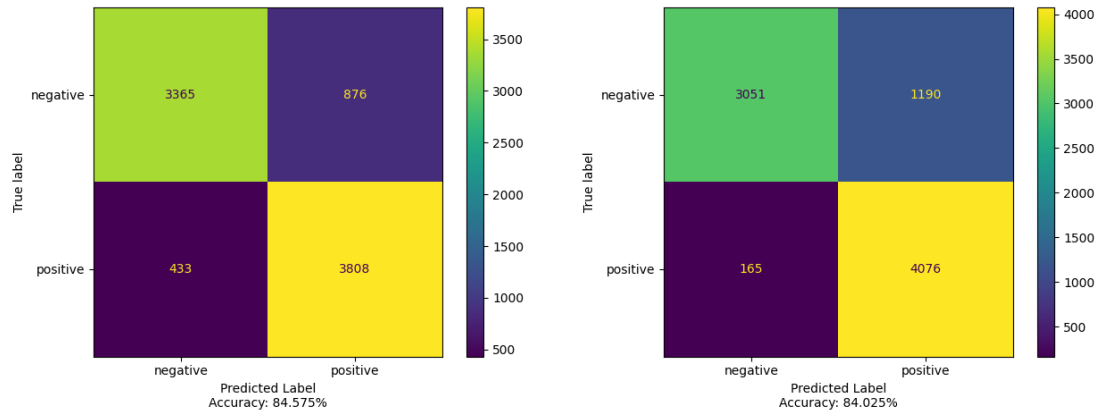
Furthermore, the result of increasing the class imbalance in favour of the positive class meant that this class was predicted a significant amount more than the negative class. In [Figure 6.1](#) the predictions of the model are shown. As the class imbalance increases, the number of false negatives (FN) decreases and false positives (FP) increases, due to the model being more inclined to predict a positive class as it is exposed to more positive than negative data. An emphasis is given on reducing FN in certain medical scenarios like disease diagnosis. Missing a diagnosis can lead to the spread of a disease such as in this case, COVID-19, showing the importance of having enough positive samples of data in a medical scenario. On the other hand, an increase in false positives can put a burden on medical staff as more people are admitted to hospitals and require treatment.



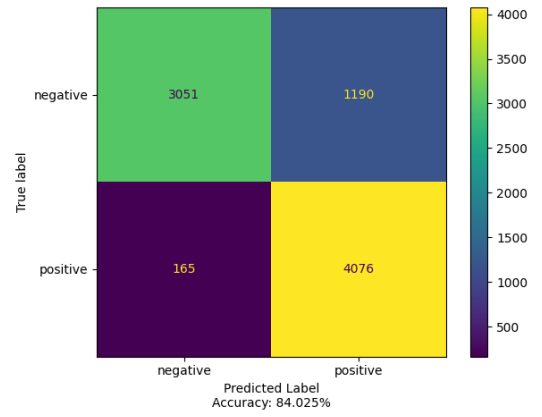
(a) Split 1



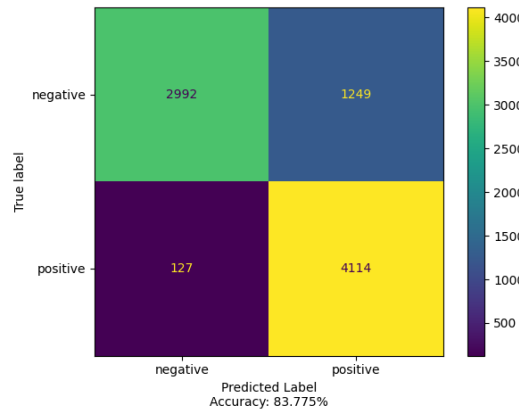
(b) Split 2



(c) Split 3



(d) Split 4



(e) Split 5

Figure 6.1: Confusion Matrices of Initial Training Splits

Table 6.2: Results of the further training configurations on the test set

Training Set	Accuracy on Test Set	Training Time (s)
Split 6	83.000%	15966
Split 7	82.450%	12143
Split 8	81.150%	10570
Split 9	79.025%	8388
Split 10	76.325%	7924
Split 11	75.325%	7751
Split 12	72.250%	7528

6.2 Further Training Set Splits

Based on the outcomes of the previous tests, where the model maintained good performance as a COVID classifier despite the significantly imbalanced dataset, additional testing was conducted to identify when the model's accuracy began to decline. These tests were performed with scaled training and validation sets such that if 4000 images were used in total for training, then 400 images would be used for validation. It would be unreasonable to use the same 8000 validation images for extremely small splits of data as this would be unrealistic.

These training data compositions of the further splits are outlined in [Table 4.4](#)

The evaluation results of the further training split experiments are summarised in [Table 6.2](#). Obviously, there is a general trend of decrease in accuracy as the number of images in the training set decreases, shown in [Figure 6.2](#). However, this negative effect is not as large as expected with the lowest accuracy, Split 12, only being trained on 125 positive and 125 negative samples still achieving an impressive 72.250% accuracy on the test set. These results show that although CNNs perform more effectively on larger amounts of labelled data - smaller amounts can still be used to classify COVID from X-ray scans at a modest accuracy which is critical in medical scenarios where data may be limited.

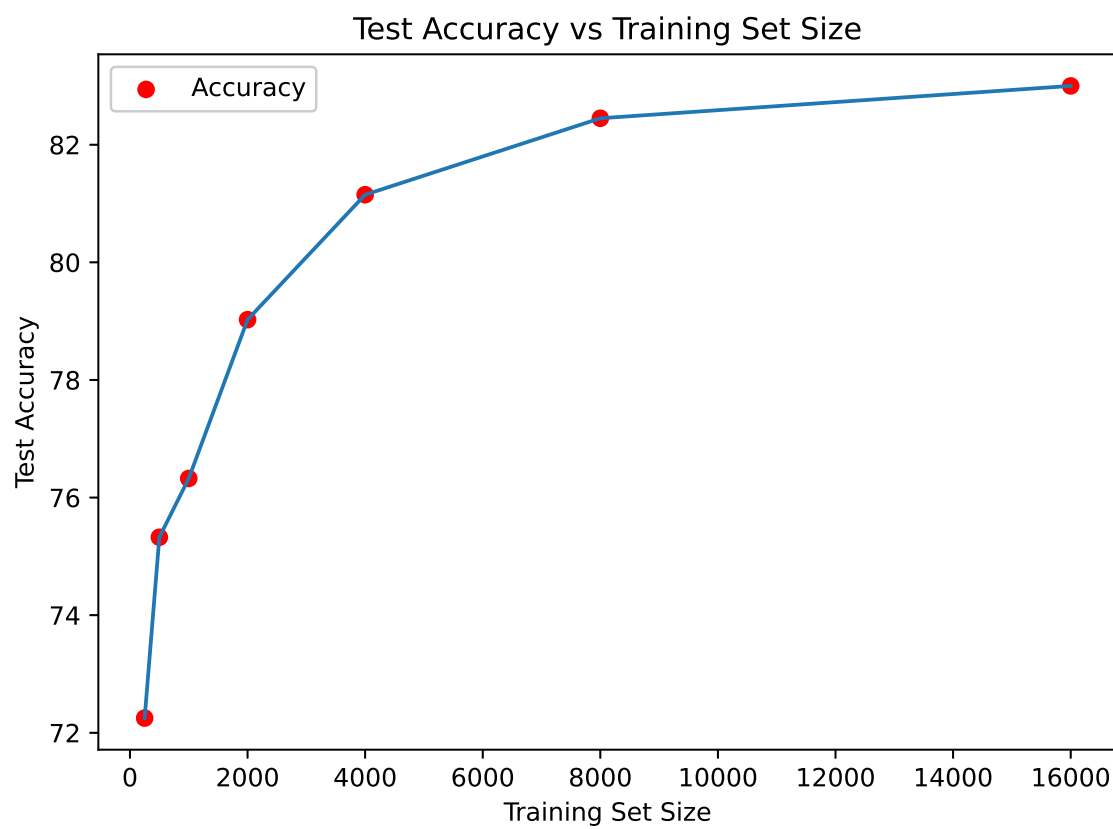


Figure 6.2: Model performance as the training set size decreases

Chapter 7

Conclusion

7.1 Reflection

All objectives outlined in [section 2.1](#) were met following the requirements analysis. A dataset, COVID-CXR4 was obtained from Kaggle from reputable researchers, and the class distribution of the dataset was investigated leading to the subsequent training set experimentation.

The evaluation framework was designed to use accuracy and loss as the appropriate metric to assess the effectiveness of the model, these were used for model selection. The train-validation-test split was established as the dataset was already pre-split in an 80:10:10 ratio, with proper class representation among the validation and testing sets. The training set size was experimented such that multiple training sets were used to train the model.

Data pre-processing and augmentation techniques were applied to the dataset to prepare the X-ray scan data for use within the model. Resizing images, centre cropping, random rotation, random flipping and random colour jitter were added to images.

Multiple CNN models were trained and evaluated on a representative test set. The results from this evaluation meant the hyperparameters of the model could be tuned in order to optimise the model, resulting in higher accuracy than the baseline ResNet architecture.

This optimised model was successfully tested on various training configurations from

a total of approximately 66,000 images to 250 images. The results of these training configurations show that CNNs perform better with larger amounts of data until a certain point where the class imbalance, specific to our dataset, affects the accuracy and performance starts to slightly decline. Furthermore, we show that within binary classification CNNs can still perform moderately with small amounts of labelled training data.

7.2 Future Work

Hyperparameters such as batch size could have been tuned separately on the further training splits as using 128 batch sizes for 250 samples of data is not practical. The effect of techniques such as cross-validation could be investigated on the smaller splits of training data, which have proven to positively impact accuracy (Badža and Barjaktarović, 2020). Further regularisation techniques like label smoothing which introduces noise into the training process have also proven to improve the accuracy of models for classification tasks (C. Szegedy, Vanhoucke et al., 2015) and should be examined.

References

- Abd-Ellah, Mahmoud Khaled, Ali Ismail Awad, Ashraf A. M. Khalaf and Hesham F. A. Hamed (Sept. 2018). ‘Two-phase multi-model automatic brain tumour diagnosis system from magnetic resonance images using convolutional neural networks’. In: *EURASIP Journal on Image and Video Processing* 2018.1, p. 97. ISSN: 1687-5281. DOI: [10.1186/s13640-018-0332-4](https://doi.org/10.1186/s13640-018-0332-4). URL: <https://doi.org/10.1186/s13640-018-0332-4> (cit. on pp. 3, 15).
- Adaloglou, Nikolas (2020). ‘Intuitive Explanation of Skip Connections in Deep Learning’. In: <https://theaisummer.com/>. URL: <https://theaisummer.com/skip-connections/> (cit. on p. 18).
- Alzubaidi, Laith, Muthana Al-Amidie, Ahmed Al-Asadi, Amjad J. Humaidi, Omran Al-Shamma, Mohammed A. Fadhel, Jinglan Zhang, J. Santamaría and Ye Duan (2021). ‘Novel Transfer Learning Approach for Medical Imaging with Limited Labeled Data’. In: *Cancers* 13.7. ISSN: 2072-6694. DOI: [10.3390/cancers13071590](https://doi.org/10.3390/cancers13071590). URL: <https://www.mdpi.com/2072-6694/13/7/1590> (cit. on p. 17).
- Alzubaidi, Laith, Mohammed A. Fadhel, Omran Al-Shamma, Jinglan Zhang, J. Santamaría, Ye Duan and Sameer R. Oleiwi (2020). ‘Towards a Better Understanding of Transfer Learning for Medical Imaging: A Case Study’. In: *Applied Sciences* 10.13. ISSN: 2076-3417. DOI: [10.3390/app10134523](https://doi.org/10.3390/app10134523). URL: <https://www.mdpi.com/2076-3417/10/13/4523> (cit. on pp. 3, 17).
- Alzubaidi, Laith, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaria, Mohammed A. Fadhel, Muthana Al-Amidie and Laith Farhan (Mar. 2021). ‘Review of deep learning: concepts, CNN architectures, challenges, applications, future directions’. In: *Journal of Big Data* 8.1, p. 53. ISSN: 2196-1115. DOI: [10.1186/s40537-021-00444-8](https://doi.org/10.1186/s40537-021-00444-8) (cit. on pp. 3, 15, 22).
- Badža, Milica M and Marko Č Barjaktarović (2020). ‘Classification of brain tumors from MRI images using a convolutional neural network’. In: *Applied Sciences* 10.6, p. 1999 (cit. on p. 50).
- Bay, Herbert, Tinne Tuytelaars and Luc Van Gool (2006). ‘SURF: Speeded Up Robust Features’. In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 404–417. ISBN: 978-3-540-33833-8 (cit. on p. 12).
- Bhatt, Dulari, Chirag Patel, Hardik Talsania, Jigar Patel, Rasmika Vaghela, Sharnil Pandya, Kirit Modi and Hemant Ghayvat (2021). ‘CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope’. In: *Elec-*

- tronics 10.20. ISSN: 2079-9292. DOI: [10.3390/electronics10202470](https://doi.org/10.3390/electronics10202470). URL: <https://www.mdpi.com/2079-9292/10/20/2470> (cit. on p. 22).
- Botten, Teddie-Valentine (Nov. 2023). *Investigating the CNN training data requirements for classification of COVID-19 chest X-rays* (cit. on pp. 2, 4, 11).
- Botten, Teddie-Valentine (Feb. 2024). *Literature Review and Progress Update* (cit. on p. 12).
- Chowdhury, Muhammad E. H., Tawsifur Rahman, Amith Khandakar, Rashid Mazhar, Muhammad Abdul Kadir, Zaid Bin Mahbub, Khandakar Reajul Islam, Muhammad Salman Khan, Atif Iqbal, Nasser Al Emadi, Mamun Bin Ibne Reaz and Mohammad Tariqul Islam (2020). ‘Can AI Help in Screening Viral and COVID-19 Pneumonia?’ In: *IEEE Access* 8, pp. 132665–132676. DOI: [10.1109/ACCESS.2020.3010287](https://doi.org/10.1109/ACCESS.2020.3010287) (cit. on p. 26).
- De Vos, Bob D, Floris F Berendsen, Max A Viergever, Marius Staring and Ivana Išgum (2017). ‘End-to-end unsupervised deformable image registration with a convolutional neural network’. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*. Springer, pp. 204–212 (cit. on p. 15).
- Despa, Mihai Liviu (2014). ‘Comparative study on software development methodologies.’ In: *Database Systems Journal* 5.3 (cit. on p. 8).
- Dogo, Eustace M., Oluwatobi J. Afolabi and Bhakisipho Twala (2022). ‘On the Relative Impact of Optimizers on Convolutional Neural Networks with Varying Depth and Width for Image Classification’. In: *Applied Sciences* 12.23. ISSN: 2076-3417. DOI: [10.3390/app122311976](https://doi.org/10.3390/app122311976). URL: <https://www.mdpi.com/2076-3417/12/23/11976> (cit. on p. 24).
- Fauziah, Syifa and Putu Harry Gunawan (Aug. 2023). ‘Analysis of CNN Optimizer for Classifying Letter Images’. In: pp. 13–17. DOI: [10.1109/ICoDSA58501.2023.10277347](https://doi.org/10.1109/ICoDSA58501.2023.10277347) (cit. on p. 24).
- Geirhos, Robert, David H. J. Janssen, Heiko H. Schütt, Jonas Rauber, Matthias Bethge and Felix A. Wichmann (2018). *Comparing deep neural networks against humans: object recognition when the signal gets weaker*. arXiv: [1706.06969](https://arxiv.org/abs/1706.06969) [cs.CV] (cit. on p. 15).
- Haji, Saad Hikmat and Adnan Mohsin Abdulazeez (2021). ‘Comparison Of Optimization Techniques Based On Gradient Descent Algorithm: A Review’. In: *PalArch's Journal of Archaeology of Egypt / Egyptology* 18.4, pp. 2715–2743. URL: <https://archives.palarch.nl/index.php/jae/article/view/6705> (cit. on p. 24).
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren and Jian Sun (2016). ‘Deep residual learning for image recognition’. In: *Proceedings of the IEEE conference on com-*

- puter vision and pattern recognition, pp. 770–778. DOI: <https://doi.org/10.48550/arXiv.1409.4842> (cit. on pp. v, 13, 16, 18, 19).
- Hershey, Shawn, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron Weiss and Kevin Wilson (2017). ‘CNN Architectures for Large-Scale Audio Classification’. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. URL: <https://arxiv.org/abs/1609.09430> (cit. on p. 14).
- Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever and Ruslan R. Salakhutdinov (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv: [1207.0580](https://arxiv.org/abs/1207.0580) [cs.NE] (cit. on p. 23).
- Horry, Michael J., Subrata Chakraborty, Manoranjan Paul, Anwaar Ulhaq, Biswajeet Pradhan, Manas Saha and Nagesh Shukla (2020). ‘COVID-19 Detection Through Transfer Learning Using Multimodal Imaging Data’. In: *IEEE Access* 8, pp. 149808–149824. DOI: [10.1109/ACCESS.2020.3016780](https://doi.org/10.1109/ACCESS.2020.3016780) (cit. on p. 16).
- Hsieh, Tien-Heng and Jean-Fu Kiang (2020). ‘Comparison of CNN Algorithms on Hyperspectral Image Classification in Agricultural Lands’. In: *Sensors* 20.6. DOI: [10.3390/s20061734](https://doi.org/10.3390/s20061734) (cit. on p. 14).
- Huang, Gao, Zhuang Liu, Laurens van der Maaten and Kilian Q. Weinberger (2018). *Densely Connected Convolutional Networks*. arXiv: [1608.06993](https://arxiv.org/abs/1608.06993) [cs.CV] (cit. on p. 15).
- Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: [1502.03167](https://arxiv.org/abs/1502.03167) [cs.LG] (cit. on p. 21).
- Kandel, Ibrahim and Mauro Castelli (2020). ‘The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset’. In: *ICT Express* 6.4, pp. 312–315. ISSN: 2405-9595. DOI: <https://doi.org/10.1016/j.icte.2020.04.010>. URL: <https://www.sciencedirect.com/science/article/pii/S2405959519303455> (cit. on p. 25).
- Kanne, Jeffrey P., Brent P. Little, Jonathan H. Chung, Brett M. Elicker and Loren H. Ketai (Aug. 2020). *Essentials for Radiologists on COVID-19: An Update—Radiology Scientific Expert Panel*. en. DOI: [10.1148/radiol.2020200527](https://doi.org/10.1148/radiol.2020200527). URL: <http://dx.doi.org/10.1148/radiol.2020200527> (cit. on p. 2).
- Ke, Ruimin, Zhibin Li, Jinjun Tang, Zewen Pan and Yinhai Wang (2018). ‘Real-time traffic flow parameter estimation from UAV video based on ensemble classifier and optical flow’. In: *IEEE Transactions on Intelligent Transportation Systems* 20.1, pp. 54–64 (cit. on p. 14).
- Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: [1412.6980](https://arxiv.org/abs/1412.6980) [cs.LG] (cit. on pp. 24, 25).

- Krizhevsky, Alex, Ilya Sutskever and Geoffrey E Hinton (2012). ‘ImageNet Classification with Deep Convolutional Neural Networks’. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou and K.Q. Weinberger. Vol. 25. Curran Associates, Inc. (cit. on p. 13).
- Lau, Yee Ling, Ilyiana Binti Ismail, Nur Izati Binti Mustapa, Meng Yee Lai, Tuan Suhaila Tuan Soh, Afifah Haji Hassan, Kalaiarasu M Peariasamy, Yee Leng Lee, Maria Kahar Bador Abdul Kahar, Jennifer Chong and Pik Pin Goh (Mar. 2021). ‘Correction: Development of a reverse transcription recombinase polymerase amplification assay for rapid and direct visual detection of Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2)’. en. In: *PLoS One* 16.3, e0249100 (cit. on p. 2).
- LeCun, Y., L. Bottou, Y. Bengio and P. Haffner (1998). ‘Gradient-based learning applied to document recognition’. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: <https://doi.org/10.1109/5.726791> (cit. on p. 12).
- Lin, Runze (2022). ‘Analysis on the selection of the appropriate batch size in CNN neural network’. In: *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*. IEEE, pp. 106–109 (cit. on p. 39).
- Lowe, David G. (Nov. 2004). ‘Distinctive Image Features from Scale-Invariant Keypoints’. In: *International Journal of Computer Vision* 60.2, pp. 91–110. ISSN: 1573-1405. DOI: [10.1023/B:VISI.0000029664.99615.94](https://doi.org/10.1023/B:VISI.0000029664.99615.94) (cit. on p. 12).
- Lucena, Brian (2022). *Loss Functions for Classification using Structured Entropy*. arXiv: [2206.07122](https://arxiv.org/abs/2206.07122) [stat.ML] (cit. on p. 24).
- Luz, Eduardo, Pedro Silva, Rodrigo Silva, Ludmila Silva, João Guimarães, Gustavo Miozzo, Gladston Moreira and David Menotti (Mar. 2022). ‘Towards an effective and efficient deep learning model for COVID-19 patterns detection in X-ray images’. In: *Research on Biomedical Engineering* 38.1, pp. 149–162. ISSN: 2446-4740. DOI: [10.1007/s42600-021-00151-6](https://doi.org/10.1007/s42600-021-00151-6). URL: <https://doi.org/10.1007/s42600-021-00151-6> (cit. on p. 2).
- Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras and Adrian Vladu (2019). *Towards Deep Learning Models Resistant to Adversarial Attacks*. arXiv: [1706.06083](https://arxiv.org/abs/1706.06083) (cit. on p. 15).
- Mahony, Niall O’, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Adolfo Velasco-Hernández, Lenka Krpalkova, Daniel Riordan and Joseph Walsh (2019). ‘Deep Learning vs. Traditional Computer Vision’. In: *CoRR* abs/1910.13796. DOI: <https://doi.org/10.48550/arXiv.1910.13796> (cit. on pp. 12, 13).
- Nwankpa, Chigozie, Winifred Ijomah, Anthony Gachagan and Stephen Marshall (2018). *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. arXiv: [1811.03378](https://arxiv.org/abs/1811.03378) [cs.LG] (cit. on p. 21).
- Radiuk, Pavlo (Dec. 2017). ‘Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets’. In: *Information Technology*

- and *Management Science* 20, pp. 20–24. DOI: [10.1515/itms-2017-0003](https://doi.org/10.1515/itms-2017-0003) (cit. on p. 39).
- Rahman, Tawsifur, Amith Khandakar, Yazan Qiblawey, Anas Tahir, Serkan Kiranyaz, Saad Bin Abul Kashem, Mohammad Tariqul Islam, Somaya Al Maadeed, Susu M. Zughaier, Muhammad Salman Khan and Muhammad E.H. Chowdhury (2021). ‘Exploring the effect of image enhancement techniques on COVID-19 detection using chest X-ray images’. In: *Computers in Biology and Medicine* 132, p. 104319. ISSN: 0010-4825. DOI: <https://doi.org/10.1016/j.compbiomed.2021.104319>. URL: <https://www.sciencedirect.com/science/article/pii/S001048252100113X> (cit. on p. 26).
- Rajpurkar, Pranav, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Yi Ding, Aarti Bagul, Curtis P. Langlotz, Katie S. Shpanskaya, Matthew P. Lungren and Andrew Y. Ng (2017). ‘CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning’. In: *CoRR* abs/1711.05225. arXiv: [1711.05225](https://arxiv.org/abs/1711.05225). URL: <http://arxiv.org/abs/1711.05225> (cit. on pp. 3, 15, 17).
- Razzak, Muhammad Imran, Saeeda Naz and Ahmad Zaib (2018). ‘Deep Learning for Medical Image Processing: Overview, Challenges and the Future’. In: *Classification in BioApps: Automation of Decision Making*. Ed. by Nilanjan Dey, Amira S. Ashour and Surekha Borra. Cham: Springer International Publishing, pp. 323–350. ISBN: 978-3-319-65981-7. DOI: [10.1007/978-3-319-65981-7_12](https://doi.org/10.1007/978-3-319-65981-7_12). URL: https://doi.org/10.1007/978-3-319-65981-7_12 (cit. on p. 17).
- Ruder, Sebastian (2017). *An overview of gradient descent optimization algorithms*. arXiv: [1609.04747](https://arxiv.org/abs/1609.04747) [cs.LG] (cit. on p. 25).
- Sa, Inkyu, Zetao Chen, Marija Popović, Raghav Khanna, Frank Liebisch, Juan Nieto and Roland Siegwart (2018). ‘weedNet: Dense Semantic Weed Classification Using Multispectral Images and MAV for Smart Farming’. In: *IEEE Robotics and Automation Letters* 3.1, pp. 588–595. DOI: [10.1109/LRA.2017.2774979](https://doi.org/10.1109/LRA.2017.2774979) (cit. on pp. 14, 15, 17).
- Sarvamangala, D. R. and Raghavendra V. Kulkarni (Mar. 2022). ‘Convolutional neural networks in medical image understanding: a survey’. In: *Evolutionary Intelligence* 15.1, pp. 1–22. ISSN: 1864-5917. DOI: [10.1007/s12065-020-00540-3](https://doi.org/10.1007/s12065-020-00540-3). URL: <https://doi.org/10.1007/s12065-020-00540-3> (cit. on pp. 6, 15–17).
- Sharma, Sagar, Simone Sharma and Anidhya Athaiya (2017). ‘Activation functions in neural networks’. In: *Towards Data Sci* 6.12, pp. 310–316 (cit. on pp. 19, 21).
- Shen, Jiayi, Casper J P Zhang, Bangsheng Jiang, Jiebin Chen, Jian Song, Zherui Liu, Zonglin He, Sum Yi Wong, Po-Han Fang and Wai-Kit Ming (Aug. 2019). ‘Artificial Intelligence Versus Clinicians in Disease Diagnosis: Systematic Review’. In: *JMIR Med Inform* 7.3, e10010 (cit. on p. 15).

- Shorten, Connor and Taghi M. Khoshgoftaar (July 2019). ‘A survey on Image Data Augmentation for Deep Learning’. In: *Journal of Big Data* 6.1. ISSN: 2196-1115. DOI: [10.1186/s40537-019-0197-0](https://doi.org/10.1186/s40537-019-0197-0). URL: <http://dx.doi.org/10.1186/s40537-019-0197-0> (cit. on p. 3).
- Simonyan, Karen and Andrew Zisserman (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv: [1409.1556](https://arxiv.org/abs/1409.1556) [cs.CV] (cit. on p. 16).
- Sutskever, Ilya, James Martens, George Dahl and Geoffrey Hinton (2013). ‘On the importance of initialization and momentum in deep learning’. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, pp. 1139–1147 (cit. on p. 24).
- Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich (2015). ‘Going deeper with convolutions’. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9. DOI: <https://doi.org/10.48550/arXiv.1409.4842> (cit. on p. 13).
- Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens and Zbigniew Wojna (2015). *Rethinking the Inception Architecture for Computer Vision*. arXiv: [1512.00567](https://arxiv.org/abs/1512.00567) [cs.CV] (cit. on p. 50).
- Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow and Rob Fergus (2014). *Intriguing properties of neural networks*. arXiv: [1312.6199](https://arxiv.org/abs/1312.6199) [cs.CV] (cit. on p. 15).
- Valappil, Najiya K and Qurban A Memon (2021). ‘CNN-SVM based vehicle detection for UAV platform’. In: *International Journal of Hybrid Intelligent Systems* 17.1-2, pp. 59–70 (cit. on p. 14).
- Wang, Linda, Zhong Qiu Lin and Alexander Wong (Nov. 2020). ‘COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images’. In: *Scientific Reports* 10.1, p. 19549. ISSN: 2045-2322. DOI: [10.1038/s41598-020-76550-z](https://doi.org/10.1038/s41598-020-76550-z). URL: <https://doi.org/10.1038/s41598-020-76550-z> (cit. on pp. 3, 16, 17, 26).
- Wang, Xiaosong, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri and Ronald M. Summers (2017). ‘ChestX-Ray8: Hospital-Scale Chest X-Ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases’. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. DOI: [10.1109/cvpr.2017.369](https://doi.org/10.1109/cvpr.2017.369). URL: <http://dx.doi.org/10.1109/CVPR.2017.369> (cit. on p. 16).
- Yamashita, Rikiya, Mizuho Nishio, Richard Kinh Gian Do and Kaori Togashi (2018). ‘Convolutional neural networks: an overview and application in radiology’. In: *Insights into Imaging* 9.4, pp. 611–629. ISSN: 1869-4101. DOI: [10.1007/s13244-](https://doi.org/10.1007/s13244-018-0163-1)

018-0639-9. URL: <https://doi.org/10.1007/s13244-018-0639-9> (cit. on pp. v, 19, 20).