



University for the Common Good

An Evaluation of Deep Learning Techniques for Chest X-Ray Abnormality Detection and Classification

A dissertation submitted in partial fulfilment of the requirements of
Glasgow Caledonian University for the degree of
Master of Science in Big Data Technologies

Emmanuel Akama (S2227958)

Glasgow Caledonian University (London)

(MSc.) Big-Data Technologies

School of Computing, Engineering and Built Environment

This project report is my own original work and has not been submitted elsewhere in fulfilment of the requirements of this or any other award.

Abstract

Chest X-ray (CXR) is one of the most used diagnostic tools in medical practice, providing crucial insights into the condition of a patient's lungs and thoracic cavity. Despite their widespread use, the interpretation of CXR images remains a challenging task, often requiring significant expertise and experience. Traditional methods of interpreting CXR images by radiologists, while essential, have proven to be time-consuming and prone to human error, particularly under the increased strain and workload brought about by the COVID-19 pandemic era. In response to these challenges, this dissertation investigates the application of deep learning techniques for the development of automated systems to assist radiologists with the detection and classification of abnormalities on CXR images with the aim to enhance diagnostic accuracy and efficiency.

This research provides a comprehensive review of the capacity of popular deep learning architectures, like AlexNet, GoogLeNet, ResNet, and their specialized variants such as R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN, and single shot detectors like SSD and YOLO for the detection, localization and classification of abnormalities on CXR images. The result from experiments conducted using Mask R-CNN model with ResNet50/101 backbone are evaluated based on optimization strategies and performance metrics, including accuracy, precision, recall, and computational efficiency. The study employs the benchmark dataset used for the RSNA Pneumonia Detection Challenge to train and validate region-based proposal network models, specifically the Mask R-CNN model. Various data preprocessing and augmentation techniques are used to ensure robust and generalized model performance. The result will highlight the strength of region-based convolution neural networks, specifically the Mask R-CNN model, providing valuable insights into their practical implementation in medical image analysis.

Acknowledgements

I would like to express my deepest gratitude to my project supervisor Dr. Hakim Mezali, for the unwavering support and guidance throughout the duration of this research project. Thank you for the advice, comments and constructive feedback. Indeed, they have been extremely invaluable. Furthermore, I would also like to thank my lecturer Dr. Ji Qi, for inspiring my passion and understanding of the artificial intelligence/deep learning concepts required for this work. I am sincerely thankful for the knowledge and wisdom you have imparted.

Finally, I would like to express my sincere gratitude to my wife, Mrs. Oghenemega Oghenejevwe Akama the prayers, motivation and unfailing support while I was away. Not forgetting my sister Ms. Mercy Uzezi Akama for been patient and understanding throughout the period my academic journey.

Table of Contents

Abstract	<i>ii</i>
Acknowledgements	<i>iii</i>
Table of Figures	<i>vii</i>
Table of Tables	<i>ix</i>
1 Introduction	1
1.1 Background	1
1.2 Problem Statement	2
1.3 Research Question	2
1.4 Aim and Objectives	3
1.5 Research Methodology	3
1.6 Research Contribution.....	3
1.7 Structure of the Report	3
2 Literature Review	5
2.1 Architecture of Deep Convolutional Neural Networks.....	5
2.1.1 Convolutional Layer	8
2.1.2 Pooling Layer.....	10
2.1.3 Fully Connected Layer.....	16
2.1.4 Activation Functions	16
2.2 Object Detection in Deep Neural Networks	19
2.2.1 Single-Shot Detectors.....	20
2.2.2 Region Proposal-based Detectors	21
2.3 Analysis of Deep Neural Network Architectures	21
2.3.1 LeNet.....	22
2.3.2 AlexNet	22
2.3.3 GoogLeNet	23
2.3.4 R-CNN, Fast R-CNN, Faster R-CNN and Mask R-CNN	23
2.3.5 ResNet.....	24

2.4	<i>Model Training and Optimization Techniques</i>	25
2.4.1	Types of Learning.....	26
2.4.2	Approaches to Learning.....	26
2.4.3	Optimization Techniques	29
3	<i>Methodology</i>	36
3.1	<i>Technological Review</i>	36
3.1.1	Deep Learning Frameworks	36
3.1.2	Development Environments and Tools.....	37
3.2	<i>Benchmark Datasets and Challenges</i>	38
3.2.1	NIH Chest X-Ray 8/14.....	38
3.2.2	CheXpert	39
3.2.3	RSNA Pneumonia Detection Dataset	40
3.3	<i>Research Question</i>	40
3.4	<i>Research Objectives</i>	40
3.5	<i>Interpreting Pneumonia from Chest Radiographs</i>	41
3.6	<i>Model Architecture</i>	42
3.7	<i>Model Training Environment</i>	45
3.8	<i>Model Training Configurations</i>	46
3.8.1	Backbone Network	46
3.8.2	Number of Epochs	46
3.8.3	Weight Decay.....	46
3.8.4	Data Augmentation.....	46
4	<i>Results and Discussions</i>	47
4.1	<i>Model Evaluation Method</i>	47
4.1.1	Precision and Recall	47
4.1.2	Average Precision and Recall	48
4.1.3	Mean Average Precision, Recall and F1 Score	48
4.2	<i>Model Evaluation Process</i>	49
4.3	<i>Model Evaluation Result</i>	50
4.4	<i>Model Evaluation Analysis</i>	Error! Bookmark not defined.

5	<i>Conclusion and Further Work</i>	54
	<i>References</i>	56

Table of Figures

Figure 1 - Structure of a Feed-forward Neural Network	5
Figure 2 - Structure of a Recurrent Neural Network	6
Figure 3 - Computational Graph of a Neuron.	7
Figure 4 - Structure of a Deep Convolutional Neural Network	7
Figure 5 - Convolution Operation with a kernel size of (3,3).....	8
Figure 6 - Max and Average Pooling with filter size (2x2) and stride of one.....	11
Figure 7 - Spatial Pyramid Pooling Layer	12
Figure 8 - PSPNet Semantic Segmentation	13
Figure 9 - Gated-Max-Average Pooling.....	14
Figure 10 - Comparison of Activation Functions.....	17
Figure 11 - Object Detection (with Localization)	20
Figure 12 - YOLO: Single-Shot Detector Model.....	20
Figure 13 - ResNet: Region Proposal-based Detector Model	21
Figure 14 - Inception Module	23
Figure 15 - Feature Pyramid Network Architecture	24
Figure 16 - ResNet Shortcut Connection	25
Figure 17 - Residual (left) and Bottleneck Blocks (right)	25
Figure 18 - Network Optimization with Gradient Descent	26
Figure 19 - Critical Points	27
Figure 20 - Concept of Transfer Learning	28
Figure 21 - Underfitting, Ideal and Overfitting of a Model.....	31
Figure 22 - Dropout (during training)	32
Figure 23 - Model performance	33
Figure 24 - Bias-Variance Trade-Off	34
Figure 25 - High Bias (Overfitting).....	34
Figure 26 - High Variance (Underfitting).....	35
Figure 27 – Eight common thoracic diseases observed in chest X-rays	38
Figure 28 – Proportion of images with multi-labels in each of 8 pathology classes	39
Figure 29 – Sample radiograph from a typical CXR examination	41
Figure 30 – Sample CXR radiograph (with mask location indicating lung opacity)	42

Figure 31 - Image analysis from demo project	43
Figure 32 - Graphical illustration of IoU definition	47
Figure 33 - MS-COCO detection evaluation metrics	48
Figure 34 - Confusion matrix.....	49
Figure 35 - Report from the best performing training run	Error! Bookmark not defined.
Figure 36 - Model performance with validation dataset.....	52
Figure 37 - Model performance with testing dataset.....	53

Table of Tables

Table 1 - Analysis of Convolution and Pooling Layers	16
Table 2 - CheXpert (14 label observations).....	39
Table 3 - Google Colab Pro+ compute resources	45
Table 4 - Results from model training schedules.....	50
Table 5 - Model training logs	51

1 Introduction

1.1 Background

Among the various medical imaging modalities, Chest X-Ray (CXR) imaging holds a critical place due to its widespread use in diagnosing a multitude of thoracic diseases such as pneumonia, tuberculosis, and lung cancer (World Health Organization, 2001). Traditionally, the interpretation of CXR has been the domain of skilled radiologists who meticulously analyse these images to identify and classify abnormalities. However, this process is time-consuming, subjective, and prone to human error. The integration of deep learning techniques offers a promising solution to these challenges by providing automated, accurate, and efficient image analysis tools (Rajpurkar *et al.*, 2017; Wang *et al.*, 2017).

Deep learning, particularly deep convolutional neural networks (DCNN), has demonstrated exceptional performance in object detection and classification tasks. These techniques can learn complex features and patterns from large datasets, making them highly suitable for medical image analysis (Krizhevsky, Sutskever and Hinton, 2017). Technically, CXR abnormality detection involves the partitioning a digital CXR image into meaningful regions, such as identifying abnormalities within the lung boundaries, while classification involves assigning a label to an image, such as diagnosing a specific disease (Rajpurkar *et al.*, 2017; Wang *et al.*, 2017). The application of these region-based techniques to CXR images can potentially enhance diagnostic accuracy, reduce workload for radiologists, and facilitate early detection of diseases.

The focus of this dissertation is on the evaluation of various deep learning techniques for CXR abnormality detection and classification. By systematically evaluating different models and methodologies, this research seeks to identify the most effective approaches for automating the analysis of CXR medical images. The study will explore the use of state-of-the-art DCNN architectures, examine the impact of different preprocessing and augmentation techniques, and assess the models' performance in terms of accuracy, sensitivity, specificity, and computational efficiency (Shreffler and Huecker, 2020). Additionally, the research will investigate the challenges associated with training deep learning models on medical image datasets, such as

the need for large, annotated datasets for model training, and the potential for overfitting from biased datasets. These techniques are fundamental to the development of a state-of-the-art deep learning model that will be able to accurately identify and classify various thoracic diseases, such as pneumonia, tuberculosis, and lung cancer.

1.2 Problem Statement

The advancement of deep learning for computer vision tasks has been marked by several key developments and breakthroughs (Szegedy *et al.*, 2015; Krizhevsky, Sutskever and Hinton, 2017). Current approaches to object recognition and classification make essential use of deep learning methods. These advancements have significantly improved the accuracy, efficiency, and versatility of computer vision systems (He *et al.*, 2016; Huang, Liu, Van Der Maaten and Weinberger, 2017).

On the flip side, the manual interpretation of CXR images is not only time consuming, but also prone to errors due to factors such as reader fatigue and experience levels. Additionally, the increasing volume of imaging data in clinical settings necessitates the development of automated systems to assist radiologists (Rajpurkar *et al.*, 2017; Wang *et al.*, 2017). This dissertation seeks to address these issues by developing a deep learning-based system for the detection and classification of CXR images, with the goal of improving diagnostic accuracy and reducing the workload of healthcare professionals.

1.3 Research Question

Recent advances in deep learning capabilities have shown that modern neural network architectures can achieve human-level performance on several image-processing tasks, including object detection and classification (Russakovsky *et al.*, 2015; Tsung-Yi *et al.*, 2015; He *et al.*, 2016; Huang, Liu, Van Der Maaten and Weinberger, 2017). However, due to the lack of sufficient CXR data, the capacities of these models have not been fully leveraged for detection and classification of medical imaging modalities. With the recent release of public CXR datasets like the NIH Chest X-Ray 8/14, CheXpert and the RSNA Pneumonia Detection datasets, we are faced with the question: *Can we effectively leverage on the capabilities of novel region-based deep learning models to match and possibly surpass human-level performance for abnormality detection and classification of pneumonia from digital CXR images?*

1.4 Aim and Objectives

The primary aim of this research is to evaluate and enhance the application of deep learning techniques for abnormality detection and classification of CXR images.

The study employs the benchmark dataset used for the RSNA Pneumonia Detection Challenge to train and validate state-of-the-art region-based deep learning models (America, 2018; Anouk Stein *et al.*, 2018). To ensure robust and generalized model performance, data preprocessing and augmentation techniques will be used to automatically improve the versatility of the dataset. The results will highlight the strengths region-based deep learning models for medical image analysis, thereby providing valuable insights into their practical implementation in radiology (Litjens *et al.*, 2017).

1.5 Research Methodology

To support our objectives, we will identify and compare the performance of popular deep learning architectures used for object detection and classification by experimenting with different optimization techniques, such as data augmentation, regularization methods (e.g., dropout, weight decay), and learning rate schedules, to enhance model performance. The evaluation of model performance will be based on transfer learning of baseline deep learning models using quantitative and qualitative assessment metrics.

1.6 Research Contribution

Technically, CXR abnormality detection involves the partitioning a digital CXR image into meaningful regions, such as identifying abnormalities within the lung boundaries, while classification involves assigning a label to features in the image, such as diagnosing a specific disease. The application of these techniques to CXR images can potentially enhance diagnostic accuracy, reduce workload for radiologists, and facilitate early detection of diseases (Rajpurkar *et al.*, 2017; Wang *et al.*, 2017). By systematically investigating the structure of various deep learning models, optimization strategies, and practical implementation methods, this study seeks to improve the accuracy, efficiency, and reliability of automated CXR analysis.

1.7 Structure of the Report

This report is organized into five chapters. This chapter presents an introduction into the background, problem statement, and research aim and objectives.

Chapter 2 provides a comprehensive review of the popular deep learning architectures used for object detection and classification. It also includes a critical analysis of advanced optimization techniques, including learning rate schedules, regularization methods, and loss functions that can enhance model performance, particularly in the context of medical imaging processing.

Chapter 3 outlines the methodology employed in this research. It begins with a review of the software, frameworks, and tools used for deep learning, including TensorFlow, Keras, PyTorch, and other relevant libraries. The chapter then describes the CXR benchmark datasets utilized for training and evaluating the deep learning models. It also includes a detailed discussion on the data preprocessing and augmentation techniques, and the specific experimental setups to ensure reproducibility. The methodology section includes a discussion on the criteria for model evaluation and comparison, encompassing both qualitative and quantitative measures.

Chapter 4 provides a critical analysis of experimental results, focusing on the performance of the Mask R-CNN model for CXR abnormality detection and classification. It includes an analysis of various performance metrics such as mean Average Precision (mAP), mean Average Recall (mAR), F1-Score (F1) and Intersection over Union (IoU). The results are compared with baseline models, and the impact of different configurations is discussed.

Chapter 5 summarizes the key findings and contributions of the research. It discusses the limitations of the current study and proposes future research directions. Suggestions for improving the model, extending its applications, and steps for real-world deployment are provided. The chapter concludes with final remarks on the significance of the research and its potential impact for medical image analysis.

2 Literature Review

Recent advances in deep learning capabilities have shown that modern neural network architectures can achieve human-level performance on several image-processing tasks, including object detection and classification (Russakovsky *et al.*, 2015; Tsung-Yi *et al.*, 2015; He *et al.*, 2016; Huang, Liu, Van Der Maaten and Weinberger, 2017). As a natural extension of the successful application of deep learning for many computer vision tasks, there are on-going efforts to leverage the capabilities of deep learning for abnormality detection and classification in a variety of medical imaging modalities (Hou and Gao, 2021; Gancheva, Jongov and Georgiev, 2023; Majkowska *et al.*, 2020; The DeepRadiology Team, 2018).

To broaden our understanding of these capabilities, we will undertake a literature review of the popular deep learning architectures used for object detection and classification with respect to the optimization and performance metrics.

2.1 Architecture of Deep Convolutional Neural Networks

Deep convolutional neural networks (DCNN) are a specialized type of feed-forward and recurrent neural networks composed of interconnecting nodes called neurons (LeCun, Y., Kavukcuoglu and Farabet, 2010). In the feed-forward neural network, computation from preceding layers is not fed back to the current layer (See *Figure 1*). Instead, the information flows in one direction from the input to the output, with computations performed in-between each layer (Goodfellow, Bengio and Courville, 2016, pp.164).

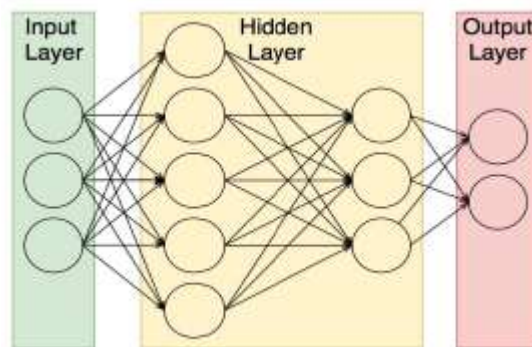


Figure 1 - Structure of a Feed-forward Neural Network

The difference between feed-forward and recurrent neural networks is the later contain connections based on feedback from preceding layers. Hence, information can be fed back

between layers (See *Figure 2*). These networks are currently used to process data presented as a sequence of input features such as time series, text, images, audio and video streams (Le Cun, 1995; Goodfellow, Bengio and Courville, 2016, pp.368).

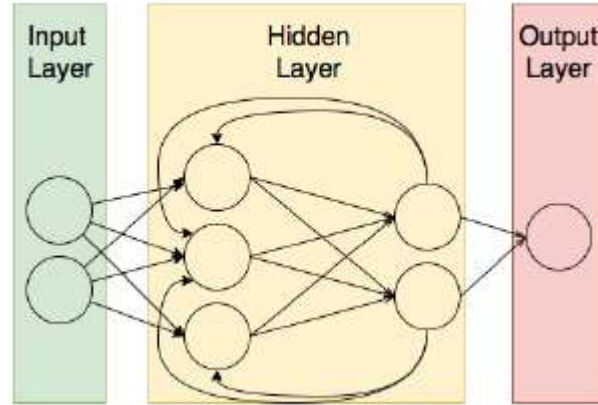


Figure 2 - Structure of a Recurrent Neural Network

A single neuron in the hidden layer of a deep neural network is called a perceptron (Bishop, 2010, pp.192; Goodfellow, Bengio and Courville, 2016, pp.366-367). Each perceptron is connected to a variable number of input weight parameters w , and a single bias parameter b , collectively represented as θ . The input x and an initial set of parameters θ are passed to an activation function $f(x, \theta)$ that determine how much the neuron should be activated based on weight w , and bias b , parameters during the model training phrase (Bishop, 2010, pp.192). Mathematically, the output of the activation function $\hat{y} = f(x, \theta)$ can be expressed as defined in equation (1), where \hat{y} represents the predicted output from the neuron.

$$f(x, \theta) = \sum(x_i w_i + b) \quad (1)$$

Deep neural networks normally consist of multiple layers. This typically include the input layer, one or more convolutional, pooling, and fully connected layers modelled as a directed acyclic computational graph consisting of nodes and directed edges in which a unit of computation is represented as a function of inputs x , along with weights and bias parameters θ , connected to the neuron (Bishop, 2010; Goodfellow, Bengio and Courville, 2016). A graphical representation of this is shown in *Figure 3*.

Deep neural networks models share some similarities with the biological context of brain cells. According to Hubel and Wiesel (1962), the visual cortex of a cat contains simple and complex cells. The simple cells activate when certain shapes are discovered in parts of an image.

Although, complex cells also activate on certain shapes, but with less regard to the location of the shape (Hubel and Wiesel, 1962). This analogy partly reflects the roles of neurons and filters in the convolutional operation of a deep neural network (Le Cun, 1995).

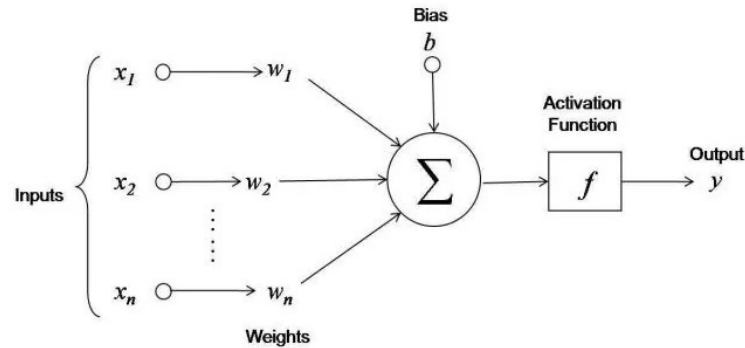


Figure 3 - Computational Graph of a Neuron.

Source: <https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf>

For computer vision and visual recognition tasks, the input of a neural network are static or dynamic images represented as a three-dimensional array of size $H \times W \times D$, where H and W represent the spatial dimensions of the image, and D is the channel dimension (Shelhamer, Long and Darrell, 2017). For example, a 3 channel RGB coloured (2012) image of size 32 x 32 pixels will be represented as a three-dimensional array of size 32 x 32 x 3 (LeCun, Y., Kavukcuoglu and Farabet, 2010; LeCun, Y., 1989).

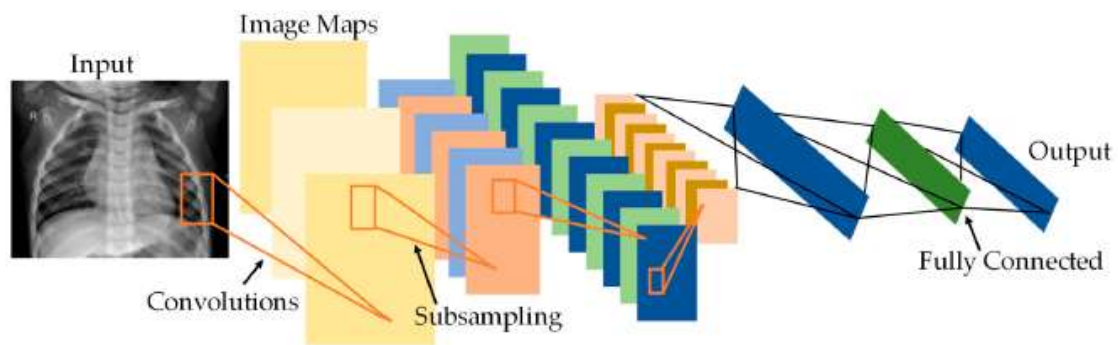


Figure 4 - Structure of a Deep Convolutional Neural Network (Rahman et al., 2020)

The use of deep convolutional neural networks (DCNN) for computer vision and visual recognition tasks offers several advantages over Fully Connected Neural Networks (FCNN). A key advantage of using DCNN is the reduced computation complexity from the parameter sharing (Shelhamer, Long and Darrell, 2017; Goodfellow, Bengio and Courville, 2016). Also, the sparse connectivity leads to reduced memory utilization, since the neurons in each layer are not necessary connected to every neuron in the next layer. Another key advantage of DCNN is

the ability to preserve the spatial hierarchy of the input image using convolution and pooling layers (Shelhamer, Long and Darrell, 2017; Goodfellow, Bengio and Courville, 2016, pp.336).

In the following subsections, we will present a comprehensive review the layers that form fundamental building blocks of a deep convolutional neural network (See *Figure 4*).

2.1.1 Convolutional Layer

The convolution and pooling layers in the DCNN ensure the arrangement of pixels in the input image is maintained throughout the network. This is critical for computer vision and visual recognition tasks like object in image detection and classification (Shelhamer, Long and Darrell, 2017). By using a local receptive field and weight sharing on a small subset of neurons, it is possible to extract features or patterns such as edges, textures, and shapes located next to each other using a filter (also known as kernel) in the convolutional layer (LeCun, Y. *et al.*, 1989). These local patterns are then combined in deeper layers to detect more complex structures (Fukushima, 1980).

Convolution is a mathematical operation that involves the dot wise product of a kernel K and the input I , such that the output $I * K$ represents the transformation of I over K (Goodfellow, Bengio and Courville, 2016, pp.328-330).

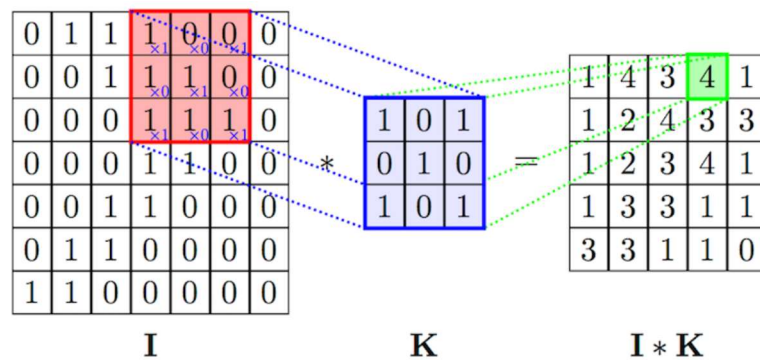


Figure 5 - Convolution Operation with a kernel size of (3,3) and a stride of one
Note: This figure only displays one channel

The input feature map I , has dimensions $H \times W \times D$, where H is the height, W is the width, and D is the number of channels. Similarly, the output feature map $I * K$, sometimes referred as the activation map has dimensions $H' \times W' \times F$, where H' and W' are the spatial dimensions determined by the input size, filter size, stride, and padding, and F is the number of filters. In most instances, the dimension of filter used in the convolution process is such that $H' < H$, and $W' < W$. According to Goodfellow *et.al.* (2016, pp.328), the activation map in DCNN is

more receptive in extracting the spatial hierarchies of the input features than FCNN. Furthermore, due to the operational efficiency of the convolution operation of DCNNs, they are capable of handling high dimensional inputs with reduced computational overhead (Goodfellow, Bengio and Courville, 2016, pp.330).

Most DCNNs use a set of parameters called hyperparameters to adjust the network within the convolution layer (Goodfellow, Bengio and Courville, 2016; Bishop, 2010). The receptive field describe the area of the input I , indicated by I' , that the filter K will perform convolution on when it is located over it. Given an input image of size 32×32 , with 3 channels in RGB format and a receptive field of 3×3 , the filter will have $3 * 3 * 3 = 27$ weight parameters, represented in vectorized form as w , plus one bias parameter b , for each convolution operation of the filter K' over that area of the input I' (Goodfellow, Bengio and Courville, 2016, pp.333). A graphical representation of the convolution and pooling operations of a deep neural network is shown in **Error! Reference source not found..**

Prior to the 2015 ImageNet challenge, most architectures used a single 3×3 kernel. However, unlike the previous networks, Szegedy *et al.* (2015) proved with GoogLeNet that multiple kernels can be used within the same convolutional layers to extract different features in a DCNN (Szegedy *et al.*, 2015).

The number of convolution layers and the number of kernels that can be applied to the input have a direct relationship to the number of output feature maps. As we have seen, the number of kernels used must be carefully chosen based on results from the training dataset and this vary from network to network (Simard, Steinkraus and Platt, 2003; Chu and Krzyżak, 2014).

The convolution operation typically involves the computation of the dot product between the filter weights and the corresponding region of the input feature map (Goodfellow, Bengio and Courville, 2016, pp.328). To ensure that the convolution operation can generate input feature maps of the same size around the edges of the input, it is customary to pad the input image by adding dummy values around the borders. Adding zeros to the borders is called zero padding. Padding is used in conjunction with sliding strategy called stride. The stride defines the step size with which the filter moves across the input feature map to determine how the kernel will convolve over the input during the convolution process (Goodfellow, Bengio and Courville, 2016, pp.332). For example, a stride of 1 means the filter moves one pixel at a time, while a larger stride results in a more significant reduction of spatial dimensions.

2.1.2 Pooling Layer

The objective of the pooling layer is to reduce the spatial dimensions of the input representation of the image using a technique known as translation invariance. Translation invariance minimizes the change observed between the input and output feature maps. This technique allows the pooling layer to summarize regions of the feature maps produced by convolutional layers, making the input representation more manageable and the network more invariant to small translations of the input (Dominik Scherer, Andreas Müller and Sven Behnke, 2010). This improves generalization and reduces overfitting when an input region varies in a minimal way such as being rotated or shifted (Krizhevsky, Sutskever and Hinton, 2017; Goodfellow, Bengio and Courville, 2016, pp.336; Dominik Scherer, Andreas Müller and Sven Behnke, 2010).

There are several types of pooling operations, the most common being max pooling and average pooling.

2.1.2.1 Max Pooling

Max pooling is a pooling strategy used to down-sample feature maps by selecting the maximum value from each non-overlapping sub-region. Applying this technique does not only reduce computational complexity but also provides translation invariance against variations in the input data, which is essential for robust feature extraction. By retaining the most prominent features within each pooling region, max pooling layers help in preserving essential information while discarding less critical details (Boureau, Ponce and Lecun, ; Dominik Scherer, Andreas Müller and Sven Behnke, 2010).

The concept of pooling was first popularized by LeNet-5. Although, it primarily used average pooling, it set the stage for later networks to explore various pooling strategies, including max pooling (LeCun, Yann, Bottou, Bengio and Haffner, 1998). The introduction of max pooling in subsequent networks demonstrated superior performance in tasks requiring high feature discriminability. For example, AlexNet (Russakovsky *et al.*, 2015), the landmark deep learning model that won the ILSVRC challenge in 2012 utilized max pooling after the first and second convolutional layers to reduce the dimensionality of feature maps. It also provided invariance to small translations and distortions in the input images (Krizhevsky, Sutskever and Hinton, 2017). In the VGGNet model, Simonyan and Zisserman (2015) emphasized the importance of depth to capture critical information for subsequent layers by stacking more convolutional layers and employing smaller 3×3 filters (Szegedy *et al.*, 2015; Simonyan and Zisserman, 2015). However, most of the early DCNN models had an inherent degradation problem caused by the vanishing or exploding gradient descent problem (Szegedy *et al.*, 2015; Simonyan and

Zisserman, 2015; He *et al.*, 2016). The ResNet model architecture, introduced by He *et al.* (2016) addressed the degradation problem by utilising max pooling after the initial convolutional layers and within the residual blocks (He *et al.*, 2016).

2.1.2.2 Average Pooling

Average pooling is another widely used pooling strategy that reduces the dimensionality of feature maps by computing the average value within non-overlapping sub-regions. Like max pooling, the technique helps in preserving the spatial information while reducing the computational complexity and mitigating overfitting. However, unlike max pooling, which retains the maximum value in a pooling region, average pooling computes the mean, resulting in a smoother approximation of the input features (Boureau, Ponce and LeCun, 2010). This can be beneficial in tasks where the overall pattern is more important than the presence of prominent features. An example of max and average pooling strategies is shown in *Figure 6*.

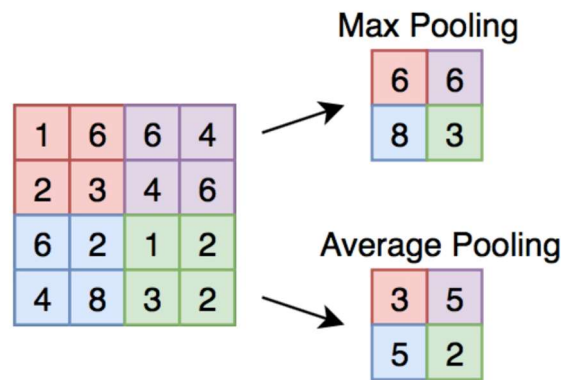


Figure 6 - Max and Average Pooling with filter size (2x2) and stride of one

LeNet-5 primarily used average pooling to reduce the dimensionality of the feature maps while preserving important spatial information while reducing the computational burden, enabling the network to perform well on handwritten digit recognition tasks (LeCun, Y. *et al.*, 1989; LeCun, Yann, Bottou, Bengio and Haffner, 1998). The Inception module in GoogLeNet also included average pooling alongside convolutions and max pooling, leveraging the strengths of each technique (Szegedy *et al.*, 2015). Utilizing average pooling in GoogLeNet was particularly useful in final layers to produce smoother and more generalized feature representations, enhancing the network's performance on complex image classification tasks (Szegedy *et al.*, 2015). Also, in ResNet, average pooling was often used in the global pooling layer before the final fully connected layer, summarizing the entire feature map into a single vector. This helped in reducing the dimensionality without losing the overall context of the image (He *et al.*, 2016).

2.1.2.3 Spatial Pyramid Pooling

Spatial Pyramid Pooling (SPP) is a technique introduced by He *et al.* in (2015) to address the limitations posed by traditional pooling methods such as max and average pooling. The key innovation of SPP is its ability to pool features at multiple spatial scales and aggregate them into a fixed-length representation, regardless of the input image size. The multi-level pooling strategy of SPP makes it more robust to changes in image size and aspect ratio, reducing the need for preprocessing steps like image resizing and cropping (He *et al.*, 2015). Since SPP uses multi-level spatial bins, while the sliding window pooling strategies like max and average pooling uses only a single window size, they are more flexible and resilient in pooling features extracted at variable scales (He *et al.*, 2015).

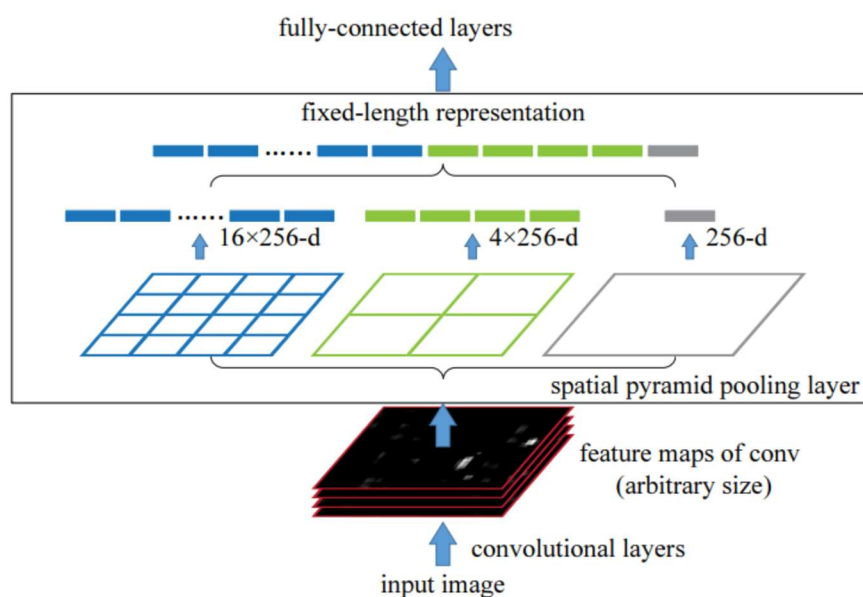


Figure 7 - Spatial Pyramid Pooling Layer (He *et al.*, 2015)

An example SPP is shown in Figure 7, where the bin size is set to 21, therefore splitting the input into 1 + 4 + 16 pooling regions.

PSPNet, introduced by Zhao *et al.* in (2017), employs a pyramid pooling module inspired by SPP to achieve state-of-the-art performance in scene parsing (See Figure 8). The pyramid pooling module in PSPNet pools features at multiple scales and concatenates them to form a rich, multi-scale contextual representation. This approach made it possible to capture global context and fine details, leading to improved segmentation accuracy on datasets such as Cityscapes and ADE20K (Zhao *et al.*, 2017).

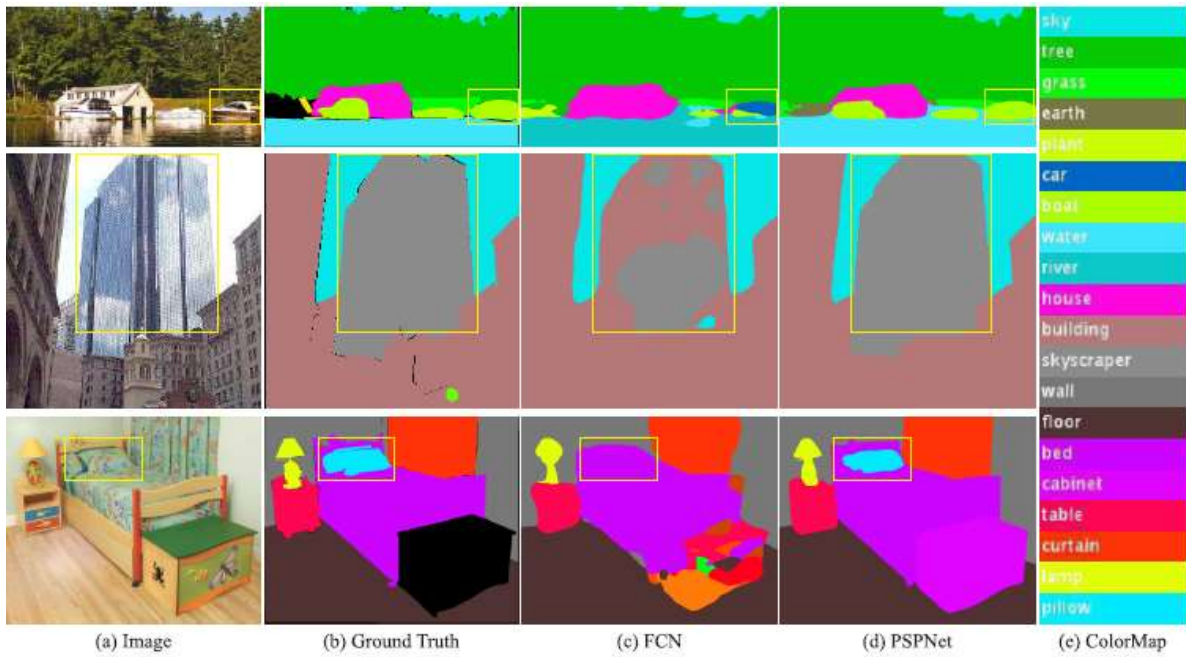


Figure 8 - PSPNet Semantic Segmentation (Zhao et al., 2017)

2.1.2.4 Stochastic Pooling

Stochastic pooling was introduced by Zeiler and Fergus in (2013) as an alternative to deterministic pooling methods by randomly selecting pooling regions based on a probability distribution (Zeiler and Fergus, 2013). Unlike deterministic pooling techniques such as max pooling, which selects the maximum value from each pooling region, or average pooling, which computes the mean, stochastic pooling selects an activation based on a multinomial distribution formed by the activations in the pooling region. This technique introduces randomness by considering different features during the training phrase, potentially leading to a richer representation of the input data, which can help in preventing overfitting and improving the network's ability to generalize (Chen-Yu Lee, Gallagher and Zhuowen Tu, 2018).

There are empirical studies in literature that compared stochastic pooling with max and average pooling to understand its effectiveness. For instance, Jost *et al.* (2015) conducted experiments comparing different pooling strategies and found that stochastic pooling can sometimes outperform max pooling and average pooling, particularly in tasks requiring better generalization (Jost *et al.*, 2015). However, the performance gains are often dataset-dependent, and stochastic pooling may not always be the best choice for every application.

2.1.2.5 Mixed, Gated and Tree Pooling

The desire to introduce more adaptive and robust pooling strategies with learning capabilities prompted Lee, Gallagher and Tu (2018) to experiment with mixed, gated and tree pooling

strategies. They believed that introducing responsiveness into the pooling operation can enhance the performance and adaptability of DCNNs in processing complex visual data (Chen-Yu Lee, Gallagher and Zhuowen Tu, 2018).

Their ideas included using a “mixed” max-average pooling strategy that combines max and average pooling in a learnable manner, allowing the network to decide the optimal mix based on the data. They also introduced a “gated” max-average pooling strategy that used a gating mechanism to dynamically learn the specific pooling function based on the proportion of features present in the pooled regions of the data. The inspiration for their ideas was supported with on findings from Boureau, Ponce and LeCun (Boureau, Ponce and LeCun, 2010), that a combination of both max and average pooling will yield better results than using them separately (See *Figure 9*).

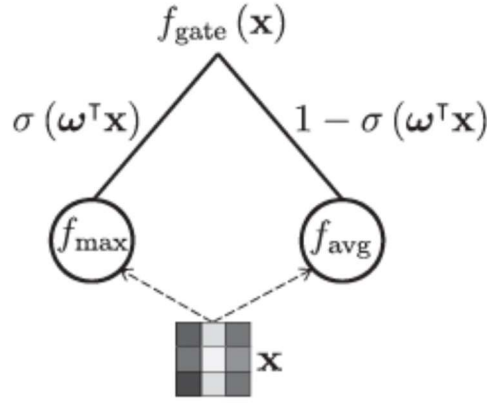


Figure 9 - Gated-Max-Average Pooling

The output of their gated max-average pooling operation is given in equation (2).

$$f_{gate} = \sigma(w^T x) f_{max} + (1 - \sigma(w^T x)) f_{avg} \quad (2)$$

The gated max-average pooling operation in equation (2) is “responsive” based on the learned parameters σ in the gating function f_{gate} , based on the combination of max and average pooling functions indicated as f_{max} and f_{avg} . They also introduced the idea of “tree” pooling strategy that implements a hierarchical pooling structure that can capture multi-level feature interactions using the idea of binary trees. An extension of equation (2) based on two learned parameters, v_1^T and v_2^T , that represents the leaf of a binary tree is presented in (3).

$$f_{tree} = \sigma(w^T x) v_1^T + (1 - \sigma(w^T x)) v_2^T \quad (3)$$

2.1.2.6 RoI Pooling and RoI Align

Region of Interest (RoI) pooling was introduced as part of the Fast R-CNN architecture in 2015 (Girshick, 2015). The core idea behind RoI pooling is to enable the network to handle multiple regions of interest (RoIs) of varying sizes, by transforming these RoIs into fixed-size feature maps that can be fed into fully connected layers. RoI pooling takes the feature map produced by the convolutional layers and applies spatial binning to each RoI. This results in a fixed-size output for each RoI, which can be processed by the subsequent fully connected layers (Ren, He, Girshick and Sun, 2015). However, the process of spatial binning and pooling introduced quantization errors (Gray and Neuhoff, 1998), as it involved discretizing the continuous coordinates of the RoIs into fixed-size bins. This can lead to misalignment between the original RoIs and the pooled feature maps, potentially affecting detection accuracy (Girshick, 2015, pp.1442).

To address the quantization issue, He *et al.* (2017) introduced RoI Align in Mask R-CNN (He, Gkioxari, Dollar and Girshick, 2020). The technique involved the use of bilinear interpolation to compute the exact values of the input features at four regularly sampled locations in each RoI bin.

2.1.2.7 Analysis

Pooling is a fundamental operation that play a crucial role in down-sampling feature maps, reducing dimensionality, and providing translation invariance. Various pooling strategies have been developed over the years, each offering distinct advantages and disadvantages.

Both Max and Average Pooling are deterministic, meaning they produce consistent outputs for the same input, which makes them stable and predictable (Krizhevsky, Sutskever and Hinton, 2017). However, their simplicity can also be a limitation, as they may not capture complex spatial hierarchies or effectively account for variations in the input (Boureau, Ponce and LeCun, 2010). Stochastic and advanced pooling methods offer significant advantages in terms of flexibility, adaptability, and the ability to capture richer spatial information (Zeiler and Fergus, 2013). These methods are particularly beneficial in scenarios where capturing multi-scale features, preventing overfitting, or dealing with variable input sizes is crucial (He *et al.*, 2015). However, they often require more computational resources, careful tuning, and may introduce variability in model performance (Girshick, 2015; He *et al.*, 2017; Chen-Yu Lee, Gallagher and Zhuowen Tu, 2018).

In *Table 1*, we present a summary analysis of the convolution and pooling layers and the features they introduce in a DCNN.

Feature	Convolutional Layer	Pooling Layer
Purpose	Feature extraction	Dimensionality reduction, translation invariance
Operation	Applies learnable filters to input	Summarizes regions of the feature maps
Learnable Parameters	Filter size, stride, padding	Pool size, stride
Output	Feature maps with detected patterns	Reduced feature maps
Role in DCNN	Detects local patterns and features	Reduces spatial dimensions, enhances invariance
Impact on Computation	Adds complexity due to learnable parameters	Reduces complexity by decreasing feature map size
Translation Invariance	Limited (relies on local receptive fields)	High (invariance to small translations)

Table 1 - Analysis of Convolution and Pooling Layers

2.1.3 Fully Connected Layer

Most of the state-of-the-art DCNNs typically include several fully connected layers following a combination of convolution and pooling. Fully connected (FC) layers are typically positioned towards the end of the network (Krizhevsky, Sutskever and Hinton, 2017a; Szegedy *et al.*, 2015; Simonyan and Zisserman, 2015). Unlike convolutional layers that preserve spatial hierarchies, FC layers treat their input as a single vector, with each neuron in each FC layer connected to every neuron in the preceding layer. The dense interconnectivity of neurons in this network structure enhances the combination the extracted features in sophisticated ways that facilitate the final prediction of object classes, usually with the softmax classifier, defined in equation (7), that outputs the probability distribution from the logits received from the last fully connected layer. Each element in the output vector represents the model's confidence in the input belonging to a particular class (Krizhevsky, Sutskever and Hinton, 2017; Szegedy *et al.*, 2015; Simonyan and Zisserman, 2015). Depending on the tasks however, the fully connected layer might not necessarily represent the last layer in a deep neural network (Shelhamer, Long and Darrell, 2017; He *et al.*, 2017).

2.1.4 Activation Functions

Activation functions introduce non-linearity into the network, enabling it to learn and represent complex patterns. They are applied to the output of each neuron in both convolutional and fully connected layers, allowing the network to approximate any arbitrary function (Goodfellow,

Bengio and Courville, 2016). The choice of activation function can significantly impact both the performance and training times of DCNNs (Krizhevsky, Sutskever and Hinton, 2017). In *Figure 10*, we present a graphical comparison activation functions reviewed in this chapter.

2.1.4.1 Sigmoid Function

The sigmoid function, defined in equation (3), maps input values to the range (0, 1).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

It was widely used useful for binary classification tasks in early neural networks due to its smooth gradient and clear probabilistic interpretation. However, because it is prone to vanishing gradient problems which can slow down or halt training in deep networks when the input is far from zero, causing slow convergence (Glorot and Bengio, 2010).

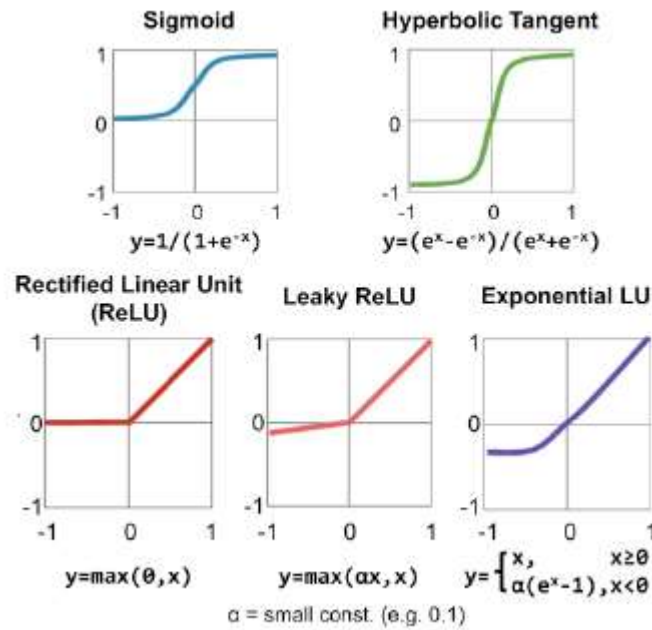


Figure 10 - Comparison of Activation Functions

2.1.4.2 Hyperbolic Tangent Function

The hyperbolic tangent function, also known as *tanh* function in equation (4), maps the input values to the range (-1, 1).

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4)$$

It is essentially a rescaled version of the sigmoid function with zero-centred outputs to centre the data and stronger gradient outputs to make model training faster. Like the sigmoid function,

tanh function is also susceptible to the vanishing or exploding gradient problem that can slow down or halt convergence during training (LeCun, Yann *et al.*, 2002; Xu, Huang and Li, 2016).

2.1.4.3 Rectified Linear Unit (ReLU)

The rectified linear unit (ReLU) function, also known as ReLU in equation (5), has become the de-facto activation function for many types of DCNNs in recent times due to the efficient computation resulting from its simple thresholding operation.

$$ReLU(x) = \max(0, x) \quad (5)$$

It alleviates the vanishing gradient problem, since it does not produce saturated outputs for positive inputs. However, it is susceptible to the dying ReLU problem, where neurons can become inactive if they output zero for all inputs, and the outputs are non-zero centred (Maas, Hannun and Ng, 2013).

2.1.4.4 Leaky and Parametric ReLU

Leaky ReLU addresses the dying ReLU problem by allowing a small, non-zero gradient when the input is negative by introducing the hyperparameter a . This is shown with the function defined in equation (6).

$$Leaky ReLU(x) = \max(ax, x) \quad (6)$$

According to Maas *et al.* (2013) and Xu *et al.* (2016), an implementation of the Leaky ReLU activation function can achieved a higher performance when compared with ReLU (Maas, Hannun and Ng, 2013; Xu, Huang and Li, 2016). Parametric ReLU (PReLU) generalizes this by making the slope of the negative part a learnable parameter.

2.1.4.5 Softmax

The softmax function defined in equation (7), is a stochastic function typically used to provide probabilistic interpretation of the outputs in the output layer of classification networks.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (7)$$

with x_i being the element on position i of the input vector which has the size j .

The softmax activation function has a greater computational overhead than the sigmoid and hyperbolic tangent functions due to the exponential and normalization operations (Goodfellow,

Bengio and Courville, 2016; Bishop, 2010). Furthermore, it tends to be sensitive to large input values, leading to numerical instability.

2.1.4.6 Swish

Swish is an activation function introduced by Ramachandran *et al.* in (2017). Albeit slightly more computationally expensive, it demonstrates better performance, in many deep learning tasks compared to ReLU. It is defined as $Swish(x) = x * \sigma(x)$, where $\sigma(x)$ is the sigmoid function. The swish activation function benefits from a smooth and non-monotonic function that improves gradient flow during training (Ramachandran, Zoph and Le, 2017).

2.1.4.7 Analysis

The choice of activation function is critical to the performance and efficiency of deep learning models. While traditional activation functions like Sigmoid and Tanh laid the foundation for early neural networks, they are generally less effective for training modern deep learning models due to their propensity for causing the vanishing gradient problem (Maas, Hannun and Ng, 2013; Xu, Huang and Li, 2016). Albeit, these functions can still be useful in specific scenarios, such as binary classification or tasks where smooth, bounded outputs are needed (Glorot and Bengio, 2010; LeCun, Yann, Bottou, Bengio and Haffner, 1998). In recent years, these methods have largely been supplanted by ReLU and its variants, which offer better performance in deeper networks (Nair and Hinton, 2010). Newer functions like Swish may even offer marginal improvements in certain scenarios, particularly in very deep networks, but with an associated increase in computational cost (Ramachandran, Zoph and Le, 2017). Hence, the selection of an activation function should be informed by the specific requirements of the task, the depth of the network, and the computational resources available (LeCun, Yann *et al.*, 2002).

2.2 Object Detection in Deep Neural Networks

Object detection in DCNN is a two-fold task. The first task involves figuring out “what” object classes is contained in the image. This process is called object recognition. This is achieved using an algorithm that outputs a probability distribution across a set of pre-defined class labels in the image, such as dog, cat etc. The second task is determining “where” the object classes are located. This process is called object localization. This typically involves predicting the position of a bounding box that determines the location of the object class. In the example

shown in *Figure 11*, object localization involves predicting the blue bounding box associated with the dog.

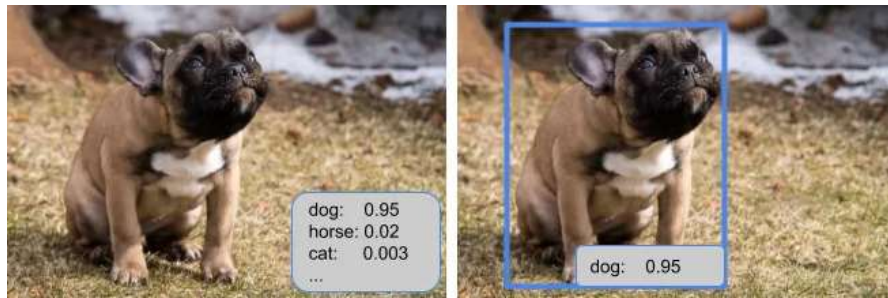


Figure 11 – DCNN-based object detection (with Localization)

There are two major approaches to deep neural network-based object detection. This includes the single-shot and region-based detector approaches.

2.2.1 Single-Shot Detectors

Single-shot detectors are a family of object detection algorithms perform object detection as well as classification in a single forward pass of the network. It uses feature maps from different layers to predict bounding boxes and class scores, enabling detection at various scales. This enhances their speed and accuracy, making them more suitable for real-time applications.

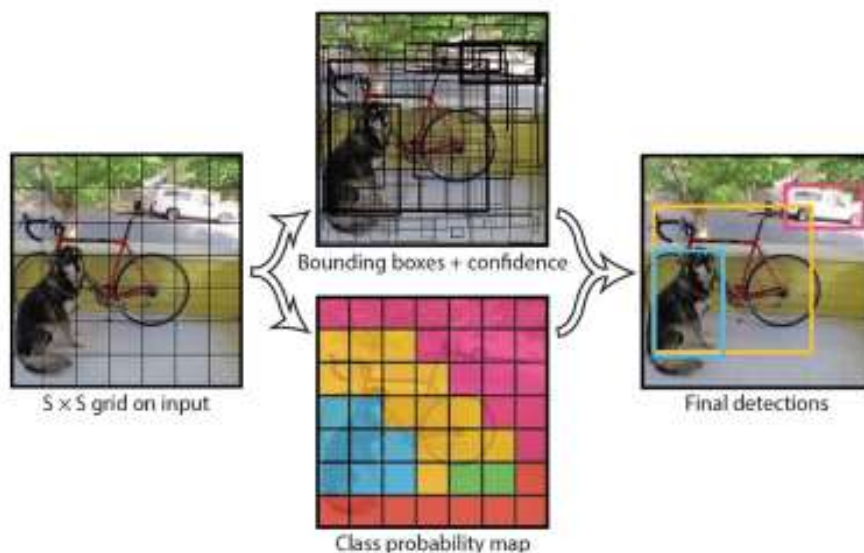


Figure 12 - YOLO: Single-Shot Detector Model (Redmon, Divvala, Girshick and Farhadi, 2016)

Examples of single-shot detectors include the Single Shot MultiBox Detector (SSD), You Only Look Once (YOLO), and Feature Pyramid Network (FPN) (Redmon, Divvala, Girshick and Farhadi, 2016; Boureau, Ponce and LeCun, 2010).

In the YOLO single-shot detector example shown in *Figure 12*, the input image is divided into an $S \times S$ grid, B bounding boxes are predicted (regression) and a class is predicted among C classes (classification) over the most confident ones (Redmon, Divvala, Girshick and Farhadi, 2016)

2.2.2 Region Proposal-based Detectors

Region proposal-based detectors, on the other hand, use a multi-stage approach to extract features from each region proposal for classification and simultaneously train a regression model to predict the correct bounding boxes from learned offsets. However, it is more computationally expensive because of its multi-stage approach without shared computation. Notable models that implement this approach include Fast R-CNN, Faster R-CNN, Mask R-CNN, and their subsequent improvements (Girshick, 2015a; Ren, He, Girshick and Sun, 2015; He *et al.*, 2017).

In the region proposal-based detector model shown in *Figure 13*, the input image feeds a ResNet model to produce feature maps. The RPN detects the Region of Interests and a score is computed for each region to determine the most likely object if there is one (Dai, Li, He and Sun, 2023).

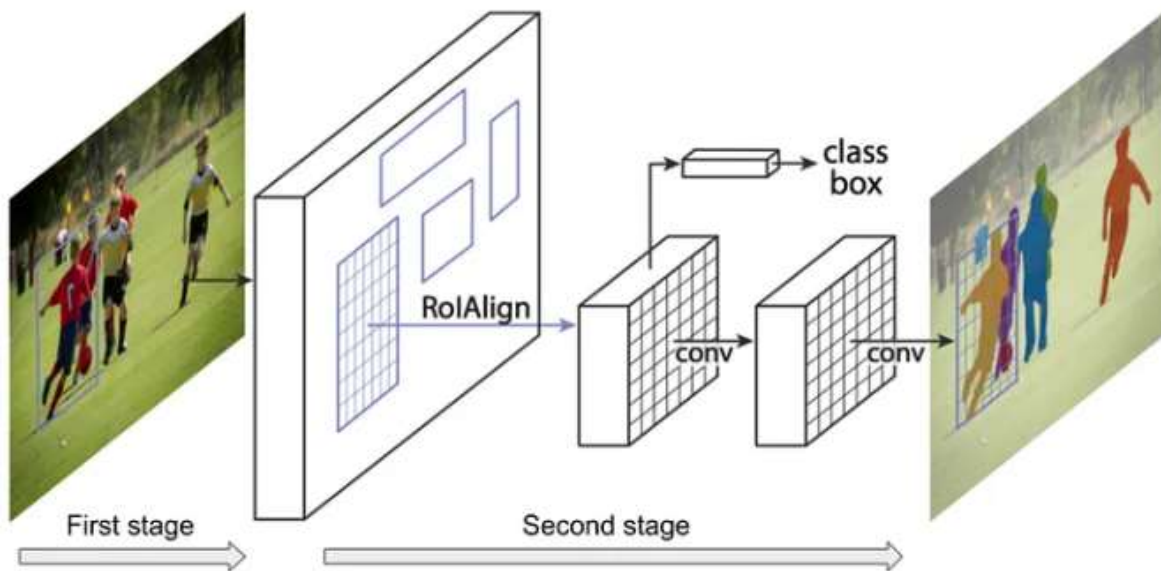


Figure 13 - ResNet: Region Proposal-based Detector Model (He et al., 2016)

2.3 Analysis of Deep Neural Network Architectures

In this section, we present an analysis of the notable deep neural networks and their accomplishments in handling complex visual recognition tasks.

2.3.1 LeNet

LeNet was a groundbreaking neural network architecture that laid the foundation for modern convolutional neural networks. It was designed primarily for handwritten digit recognition, specifically to classify digits from the MNIST dataset. LeNet-5, the most well-known version of LeNet had seven layers (excluding the input layer) that follow a pattern of alternating convolutional and pooling layers that culminated into a set of fully connected layers (LeCun, Yann, Bottou, Bengio and Haffner, 1998).

The LeNet architecture incorporated a couple of novel training strategies including the use of a backpropagation algorithm to optimize the hyperparameters through gradient descent (LeCun, Yann, Bottou, Bengio and Haffner, 1998; LeCun, Y. *et al.*, 1989). Although efficient for its time, LeNet's training process was limited by the computational resources available in the late 1990s.

2.3.2 AlexNet

AlexNet was a landmark deep neural network architecture that significantly advanced the field of computer vision. It won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a substantial margin, reducing the top-5 error rate from 26% to 15.3% (Russakovsky *et al.*, 2015b). This achievement highlighted the potential of deep learning in image classification tasks and spurred a wave of research and development in the field of computer vision (Krizhevsky, Sutskever and Hinton, 2017).

AlexNet's neural network architecture consisted of eight layers: five convolutional layers responsible for feature extraction followed by three fully connected layers interspersed between the first, second, and fifth convolutional layers to perform down-sampling, reducing the spatial dimensions of the feature maps and introducing translation invariance. This is followed by use of the ReLU activation function and local response normalization (LRN) instead of traditional hyperbolic or sigmoid functions to mitigate the vanishing gradient problem by introducing non-linearity to accelerates convergence, enabling the network to learn complex patterns (Maas, Hannun and Ng, 2013; Xu, Huang and Li, 2016). The final three layers are fully connected, culminating in a 1000-way softmax layer that produces a distribution over the 1000 class labels of the ImageNet dataset (Krizhevsky, Sutskever and Hinton, 2017b). While subsequent architectures have built upon and surpassed AlexNet's achievements, its contributions to the resurgence of neural networks and the establishment of deep learning as a dominant force in artificial intelligence remain foundational (Silver *et al.*, 2017; Wu *et al.*, 2016).

2.3.3 GoogLeNet

GoogLeNet (2015) [8], also known as Inception V1, won the 2014 ImageNet Object Detection and Classification challenge with a top-5 error rate of 6.67% (Russakovsky *et al.*, 2015; Szegedy *et al.*, 2015). They introduced the Inception module (as seen *Figure 14*).

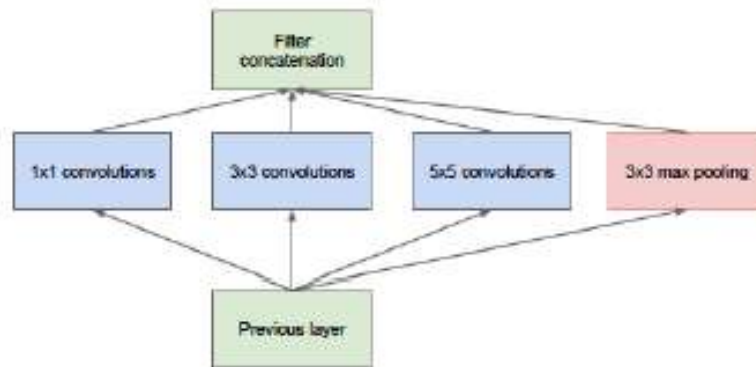


Figure 14 - Inception Module (Szegedy et al., 2015)

This novel architecture allowed the network to capture multi-scale features by applying multiple convolutional and pooling operations in parallel and concatenating their outputs along the channel dimension, enabling the network to capture information at various scales, whilst reducing the risk of losing spatial information. Instead of fully connected layers at the end, it used global average pooling to average each feature map to a single value (Lin, M., Chen and Yan, 2013). This was done to reduce the total number of parameters to mitigate the risk of overfitting.

2.3.4 R-CNN, Fast R-CNN, Faster R-CNN and Mask R-CNN

Region-based Convolutional Neural Networks (R-CNN), Fast R-CNN, Faster R-CNN and Mask R-CNN are a series of models developed to advance the field of object detection. R-CNN was introduced by Girshick *et al.* in 2014. They used the selective search algorithm, introduced by Uijlings *et al.* in (2013) to propose RoIs where objects may be located and then classifies these regions using a CNN (Uijlings, van de Sande, Gevers and Smeulders, 2013). R-CNN significantly improved the accuracy of object detection over traditional methods. However, training the model required a multi-stage training process that resulted in very slow detection times. Consequently, Fast R-CNN was introduced to reduce the computational overhead required to train R-CNN by integrating region proposal generation and CNN processing into a single network. Although, the end-to-end training pipeline simplified the process of object detection, it still relied on external region proposal methods like selective search, which could be memory-intensive due to the need to store feature maps for all RoIs (Girshick, 2015). Hence,

another modification called Faster R-CNN was introduced by Ren *et al.* in (2015). It built upon Fast R-CNN by incorporating a Region Proposal Network (RPN) that generate region proposals directly within the network. The integration of RPN allowed for nearly real-time object detection by eliminating the need for external region proposals (Ren, He, Girshick and Sun, 2015).

Mask R-CNN builds on the strengths of Faster R-CNN by adding a branch for predicting segmentation masks for each RoI, enabling precise object segmentation (He *et al.*, 2017).

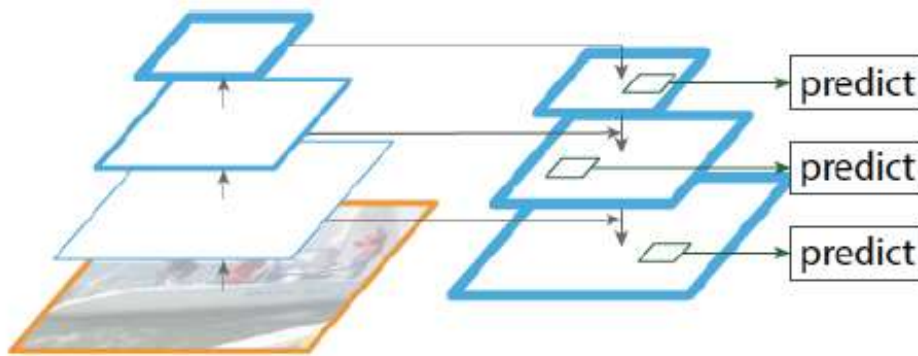


Figure 15 - Feature Pyramid Network Architecture (Lin, T. et al., 2017)

This enhanced its versatility and adaptation to various tasks, but also increased the complexity and computational demands.

The use of R-CNN models has significantly improved the accuracy of object detection over traditional methods reaching mAP scores of 70.0% for the 2007 PASCAL VOC test dataset, 68.8% for the 2010 PASCAL VOC test dataset and 68.4% for the 2012 PASCAL VOC test dataset (Girshick *et al.*, 2014).

2.3.5 ResNet

To address the degradation problem that lead to higher training error due to vanishing or exploding gradient in deep neural networks, He *et al.* (2016) introduced the concept of residual learning based on residual blocks which include a shortcut connection that bypasses one or more layers. The output of these layers is added to the input to form a residual block given by: $y = f(x) + x$. The central idea of the residual block is to learn residual function $f(x)$ with reference to the layer inputs x , instead of learning unreferenced functions directly. With the use of residual blocks, He *et al.* (2016) was able to train deep networks with up to 152 layers with remarkable improvements in accuracy on several benchmark datasets. See Figure 16.

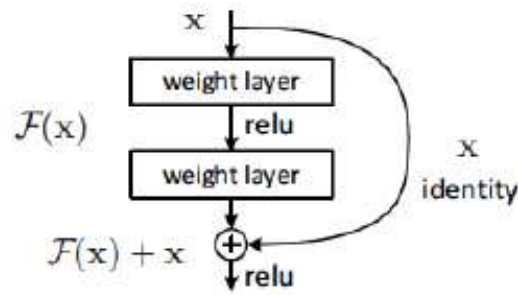


Figure 16 - ResNet Shortcut Connection (He et al., 2016)

In the ImageNet 2015 Detection and Localization challenge (Krizhevsky, Sutskever and Hinton, 2017), ResNet-152 outperformed all previous networks, including VGGNet and GoogLeNet, achieving a top-5 error rate of 3.57% (He et al., 2016).

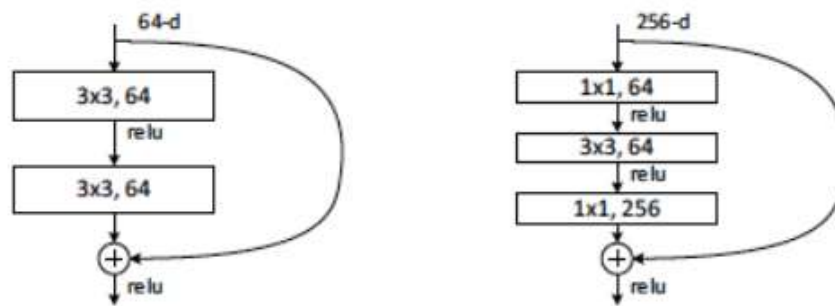


Figure 17 - Residual (left) and Bottleneck Blocks (right) (He et al., 2016)

To further optimize the network architecture, they modified the residual block to a bottleneck block (shown in Figure 17). This improved the training time and prevented it from being too large (He et al., 2016).

2.4 Model Training and Optimization Techniques

The success of deep neural networks for computer vision and visual recognition tasks has been marked by several key developments and breakthroughs. The performance of these networks hinges significantly on the training and optimization techniques employed to optimize model parameters (weights and biases) to minimize a loss function (Bottou, 2010). The choice of training method used often depend on the specific application, data availability, and computational resources (Glorot and Bengio, 2010).

In this section, we present a review of the key training and optimization strategies used in training and tuning deep neural networks, including gradient descent variants, regularization techniques, and advanced methods like transfer learning and curriculum learning.

2.4.1 Types of Learning

There are different types of learning. Supervised learning is the most common form of machine learning. It typically involves learning an objective function $\hat{y} = f(x, \theta)$ that predicts the outcome of output variable \hat{y} , based on a set input features x , and model parameters θ , using a labelled dataset. The goal of the objective function is to predict \hat{y} as close to the ground truth y as possible through an adaptation of model parameters θ (Goodfellow, Bengio and Courville, 2016, pp.171). Unsupervised learning involves training a model to learn the underlying structure or distribution of an unlabelled dataset. In semi-supervised learning, a small amount of labelled data is combined with a large amount of unlabelled data to improve learning performance. Reinforcement learning (RL) involves training an agent to make decisions by interacting with an environment, optimizing actions to maximize cumulative rewards.

2.4.2 Approaches to Learning

There are different approaches to learning. This includes batch training, stochastic and mini-batch gradient descent, adaptive learning rate, transfer, self-supervised, and reinforcement learning.

2.4.2.1 Gradient Descent

Gradient descent is a mathematical optimization technique used to find the minimum (or maximum) value of a function (Bottou, 2012).

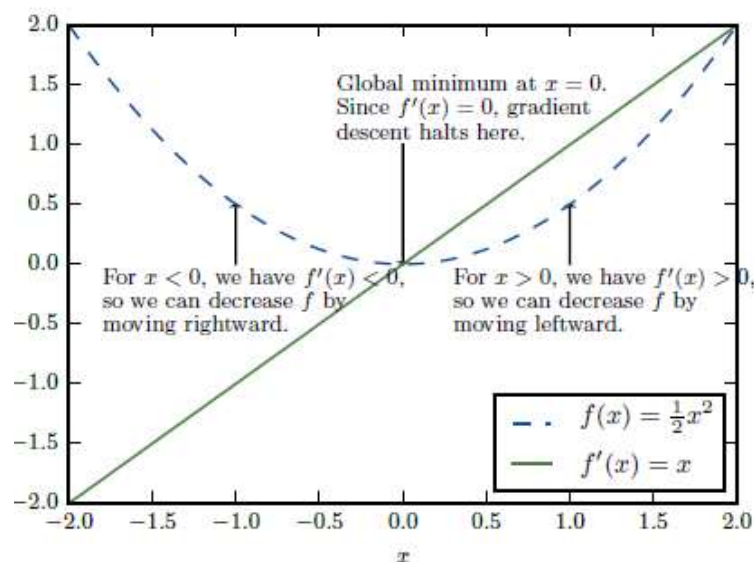


Figure 18 - Network Optimization with Gradient Descent (Goodfellow, Bengio and Courville, 2016, pp.80)

In many machine learning tasks, gradient descent is used as a cost function $f(L, \theta)$ to minimize loss L , using a set of model parameters θ . The loss L indicate the difference between the

predicted outputs of a model \hat{x}_i and the true outputs x_i (Bottou, 2012; Goodfellow, Bengio and Courville, 2016, pp.79).

$$L_i = |x_i - \hat{x}_i| \quad (8)$$

The goal gradient descent is to improve the model's performance by finding the local minimums of the objective function $f(L, \theta)$, using the derivative of the objective function $f(L, \theta)$, as the cost function $f'(L, \theta)$, by adjusting the model parameters θ , so that $f'(L, \theta)$ is at the minimum value, where $L = 0$. This is shown in *Figure 18*.

There are three critical points where the gradient can be zero. These are the local minima, local maxima and saddle points (Bottou, 2012; Goodfellow, Bengio and Courville, 2016, pp.84) shown in *Figure 19*.

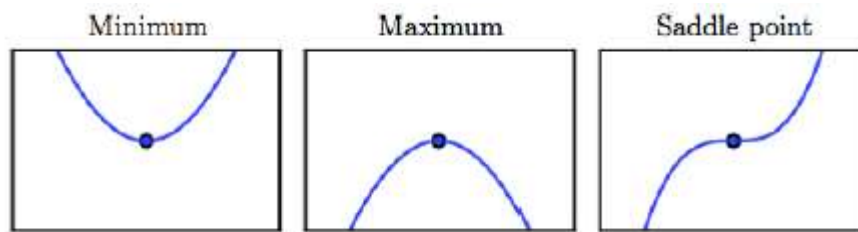


Figure 19 - Critical Points (Goodfellow, Bengio and Courville, 2016, pp.81)

2.4.2.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) uses a model training approach that performs iterative updates to the model parameters using one training example at a time (Bottou, 2012). This method is highly sequential and iterative, as each step involves processing a single sample in the dataset, leading to rapid but noisy updates.

2.4.2.3 Batch Training

Batch training processes the entire dataset in a sequential manner, updating the model parameters iteratively based on the cumulative gradients from all samples in the dataset. The use of the entire dataset guarantees the precise calculation of gradient estimates (Bottou, 2010). However, it can be computationally intensive and slow to converge due to the need to process the entire dataset before each update.

2.4.2.4 Mini-Batch Gradient Descent

This method uses a combined approach that involves the training of small subsets of the dataset in each iteration. It strikes a balance between batch and stochastic methods leveraging on parallel processing capabilities of modern hardware like GPUs and TPUs to ensure faster and more stable convergence (Masters and Luschi, 2018).

2.4.2.5 Adaptive Learning Rate Methods

This approach minimizes the need for extensive manual tuning of the learning rate by iteratively updating the model parameters, incorporating sequential adjustments based on past performance. Different adaptive learning rate strategies can be applied to optimize the learning process of a neural network (Bottou, 2010, 2012; Sutskever *et al.*, 2013). For example, Adaptive Gradient Algorithm (AdaGrad), adapts the learning rate of a model based on the frequency of each parameter updates. Root Mean Square Propagation (RMSprop) adjust the learning rate for each parameter based on a moving average of the squared gradients. Adaptive Moment Estimation (Adam) combines the ideas of AdaGrad and RMSprop by maintaining the moving averages of both the gradients and their squares .

2.4.2.6 Transfer Learning

Transfer learning involves a sequential process of pre-training a model on a large source dataset followed by iterative fine-tuning on a target task-specific dataset (See *Figure 20*). This approach leverages previously learned features to enhance performance on new tasks (Rahman *et al.*, 2020).

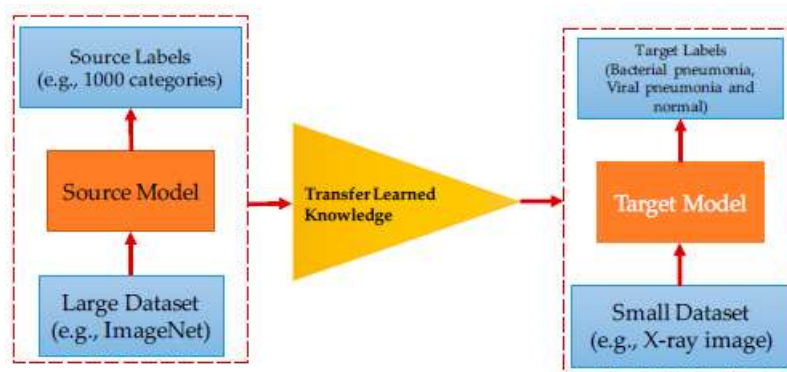


Figure 20 - Concept of Transfer Learning (Rahman et al., 2020)

2.4.2.7 Self-Supervised Learning

Self-supervised learning involves generating auxiliary tasks from the data itself, creating labels for training in a sequential manner. This iterative approach uses the inherent structure of the data to learn useful representations that generalize well across various tasks.

2.4.2.8 Reinforcement Learning

Reinforcement learning (RL) is a sequential and iterative learning process that optimizes the actions taken by an agent as it interacts with an environment over time, while enhancing decisions made to maximize cumulative rewards. Although using this approach requires

substantial computational resources and training time, it is capable of learning complex sequences of actions for long-term goals (Silver *et al.*, 2017).

2.4.3 Optimization Techniques

Training a deep neural network involves three crucial steps: the forward pass, backward propagation, and the computation of losses (LeCun, Yann *et al.*, 2002; Bishop, 2010; Goodfellow, Bengio and Courville, 2016). The set of optimization techniques used during each step play a vital role in the learning process.

Popular network models utilize optimization techniques such as initiation, regularization, backpropagation and losses to help the model capture the underlying patterns in the data and generalizes well to unseen examples. Some of these techniques have already been highlighted in the discussion on popular network models in section 2.3.

2.4.3.1 Initialization

Due to the high number of parameters (weights and biases) required to train a neural network, it is crucial to have an effective strategy to properly initialize them. This improves performance during training by avoiding exploding and vanishing gradient problems. Common initialization strategies include zero initialization that initiates all weights to zero, and random initialization where weights are initialized randomly from a uniform or normal distribution (Sutskever *et al.*, 2013).

The winning entry of the 2012 ImageNet competition, AlexNet, initialized weights based on a Gaussian distribution with a mean of zero and a standard derivation of 0.01 (Krizhevsky, Sutskever and Hinton, 2017). Since this strategy is not feasible if the networks have a very high number of layers (Mishkin and Matas, 2015; Simonyan and Zisserman, 2015), more efficient weight initialisation methods have been proposed. For example, Glorot and Bengio (Glorot and Bengio, 2010) proposed the Xavier Initialization to initialize weights based on the number of inputs and outputs of the neurons, n_{in} and n_{out} with a variance of $\frac{2}{n_{in} + n_{out}}$ (Glorot and Bengio, 2010).

The technique promoted stable learning by maintaining the variance of activations and gradients throughout the network. However, He *et al.*, (2015) claimed that the technique only achieves good results on linear activations and not on ReLU and PReLU (He *et al.*, 2015). Therefore, they proposed the He Initialization to initialise weights from a Gaussian distribution with a mean of zero and variance of $2 / n_l$, where n is the number of connections into the layer

and l is the specific layer, with a bias of zero. They claim that the technique provided faster convergence for DCNNs that used ReLU and its variants.

2.4.3.2 Forward Pass, Backpropagation and Losses

Training a deep neural network typically involves three crucial steps: the forward pass, backward propagation (or backpropagation), and the computation of losses. Each step plays a vital role in the learning process, ensuring that the model captures the underlying patterns in the data and generalizes well to unseen data (Bottou, 2012, 2010; LeCun, Yann *et al.*, 2002).

In the forward pass, each layer of the network applies a set of operations to transform the input data progressively from raw features to the final output. This is based on an objective function $\hat{y} = f(x, \theta)$ that aims to predict the outcome of output variable \hat{y} , based on a set input features x , and model parameters θ . During the backward propagation, the network computes the gradient of the loss function $f(L, \theta)$, with respect to weight of each model parameter θ . This information is then used to update the weights, minimizing the loss function iteratively. The loss function measures the discrepancy between the predicted output \hat{x}_i and the actual target x_i . Hence, it serves as the objective function that the optimization algorithm seeks to minimize (Bottou, 2012; LeCun, Yann *et al.*, 2002). Using the chain rule of calculus, the gradient of the loss function $f'(L, \theta)$ with respect to weight of the model parameter θ can be calculated. This involves propagating the error backward through the network (LeCun, Yann *et al.*, 2002). The gradients are used to update the weights in the network, typically through gradient descent or its variants (e.g., SGD, Adam). The learning rate controls the size of the update step. Hence, the choice of learning rate and optimization algorithm can be used to significantly mitigate slow convergence or getting stuck in local minima (Bottou, 2012).

2.4.3.3 Regularization

Goodfellow (2016, pp.117) defined regularization as “*any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error*”. This normally involve the use of learning techniques that help to smoothen the learning process, by adding penalty terms using techniques like dropout, early stopping, and data augmentation that aid in capturing essential patterns in the data while avoiding overfitting (Goodfellow, Bengio and Courville, 2016). Overfitting occurs when a model learns not only the underlying patterns in the training data but also the noise and outliers, leading to poor generalization when applied on new or unseen data (See Figure 21).

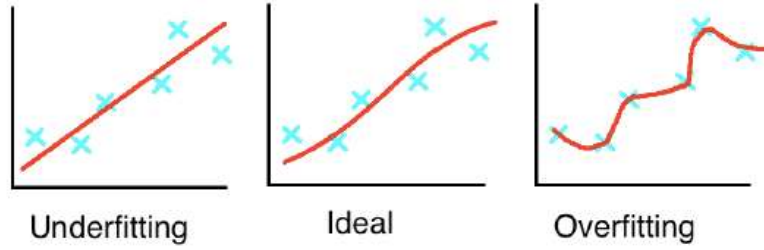


Figure 21 - Underfitting, Ideal and Overfitting of a Model (Goodfellow, Bengio and Courville, 2016)

Regularization techniques that use penalty terms often introduces a hyperparameter λ , that is larger than zero and smaller than one, to penalize the cost function C_0 , resulting in the new cost function C , with n being the number of classes and w being the weight parameters (Goodfellow, Bengio and Courville, 2016, pp.119).

There are three common type of regularization approaches that use the hyperparameter λ . The L1 Regularization (Lasso) by adding the absolute value of the coefficients as a penalty term λ to the loss function C_0 . L2 Regularization (Ridge) adds the squared value of the coefficients as a penalty term λ to the loss function C_0 . This approach is more sensitive to the noise introduced by outliers. Elastic Net combines both L1 and L2 regularization to strike a good balance between the sparsity of L1 and the smoothness of L2. The regularization terms λ_i penalizes large weights leading to a smoother model that avoids overfitting. The formula for L1, L2 and Elastic Net regularization is defined in equations (9), (10) and (11) below.

$$C = C_0 + \lambda \sum_{i=1}^n |w_i| \quad (9)$$

$$C = C_0 + \lambda \sum_{i=1}^n w^2 \quad (10)$$

$$C = C_0 + \lambda_1 \sum_{i=1}^n |w_i| + \lambda_2 \sum_{i=1}^n w^2 \quad (11)$$

Other approaches to regularization include the use of dropout that temporarily drops out (sets to zero) a random fraction of neurons during training, early stopping that terminates training when the performance on the validation set starts to degrade, and data augmentation that apply transformations to artificially increasing the size and diversity of the training set.

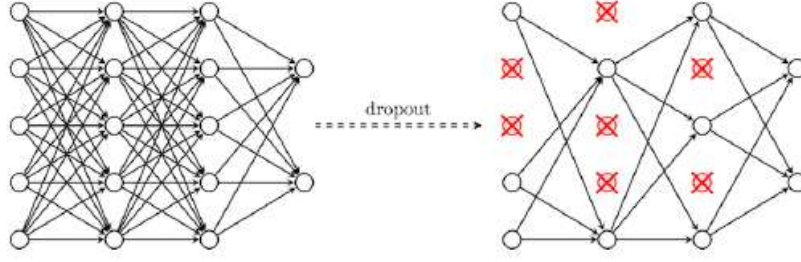


Figure 22 - Dropout (during training) (Goodfellow, Bengio and Courville, 2016)

A model that generalizes well will have similar performance on both the training data and unseen test data. The most common way to evaluate generalization is through an appraisal of model's performance metrics (such as, accuracy, precision, recall, F1-score) on validation and test sets. Using cross validation techniques like k-fold cross-validation can provide robust estimates of model performance by training and testing on different subsets of the data (Goodfellow, Bengio and Courville, 2016, pp.119).

2.4.3.4 Normalization

Normalization is a technique used to range of the data features to ensure that the different features in the data contribute equally to the model training process. A common normalization technique called the z-score (standard) normalization defined in equation (12), involves transforming the data so that it has a mean of zero and a standard deviation of one or scaling the data to fall within a specific range, such as [0, 1] (Patro and Sahu, 2015).

$$x' = \frac{x - \mu}{\sigma} \quad (12)$$

Robust scaling is useful when the dataset contains outliers. It uses methods that scale the data using statistical measures like median and interquartile range (IQR), where IQR is the difference between the 75th and 25th percentiles. This is defined in equation (13).

$$x' = \frac{x - \text{median}}{\text{IQR}} \quad (13)$$

Similarly, logarithmic scaling performs transformation to compress the range of the data. This approach is effective in normalizing skewed distributions where a few data points are much larger than the rest. This is defined in equation (14).

$$x' = \log(x + 1) \quad (14)$$

Ioffe and Szegedy (2015) introduced batch normalization to reduce internal covariate shift that can occur in a neural network. They describe internal covariate shift as “*the change in the distribution of network activations due to the change in network parameters during training*” (Ioffe and Szegedy, 2015). This was based on the justification that rescaling the input distribution in one layer using standard normalization techniques can get amplified further down into deeper layers whose input distribution will then change, leading to longer training times. To reduce internal covariate shift, they applied standard normalization given in equation (15) to the output of each mini-batch by adjusting the mean μ_B and variance σ_B^2 of a mini-batch B , consisting of the inputs $B = \{x_1 \cdots x_m\}$, before applying the activation, so that the mean activation is 0 with a standard deviation of 1. The hyperparameter ϵ , is a small added constant to avoid dividing by zero.

$$\hat{x} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (15)$$

Scaling the data features to a similar range can help to adjust the sensitivity the model to initial weights. This improves the model performance by ensuring that gradients do not become too small or too large. Popular models that used normalization techniques include ResNet (2015) and Inception v1 (GoogLeNet, 2014). Both ResNet and GoogLeNet used batch normalization extensively to manage the complex architectures to ensure efficient training (Ioffe and Szegedy, 2015).

2.4.3.5 Bias-Variance Trade-Off

Bias-variance trade-off is a crucial concept that help explain a model generalization ability.

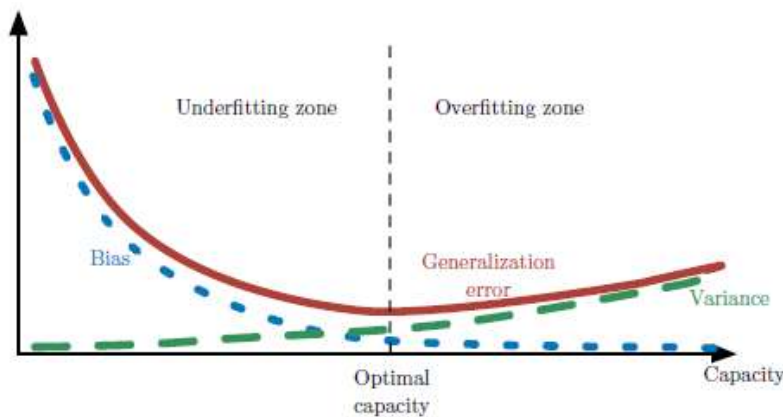


Figure 23 - Model performance (Goodfellow, Bengio and Courville, 2016)

Bias refers to the error introduced by approximating model hyperparameters during model training (See *Figure 23*). In the context of DCNNs, high bias is typically an indication of a model with too few layers or neurons to capture the underlying patterns in the data. Models with low bias can represent the training data more accurately. However, this might lead to overfitting on the training data, especially if too many parameters are used relative to the amount of training data (See *Figure 23*).

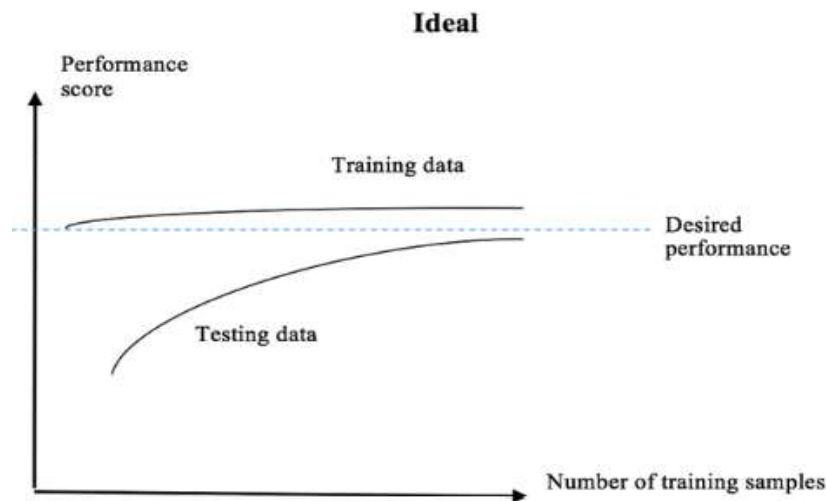


Figure 24 - Bias-Variance Trade-Off

Variance refers to the sensitivity of a model to fluctuations in the training data. It represents the amount by which the model's predictions would change if it were trained on a different training set.

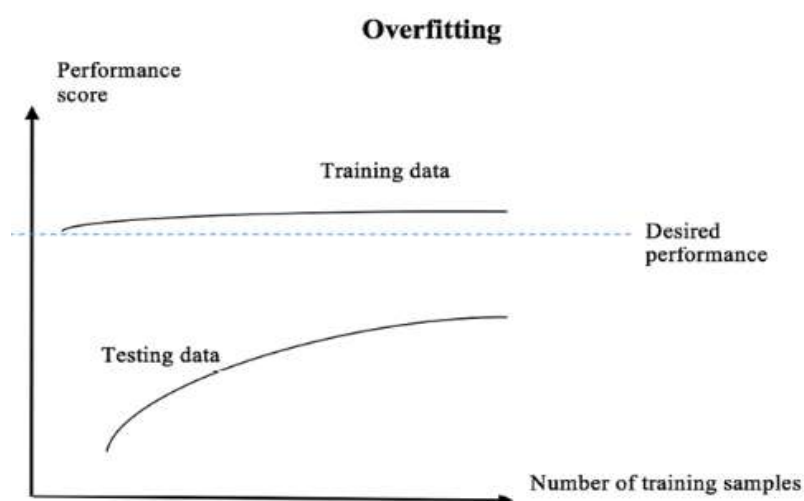


Figure 25 - High Bias (Overfitting)

A model with high variance will overfit the training data, leading to poor generalization on the testing data (See *Figure 23*). Hence, adjusting the structure of the neural network (number of layers, number of neurons per layer) can help in finding a balance between bias and variance. Regularization techniques like L1 and L2 regularization that add a penalty to the loss function for large coefficients, and cross-validation that help in estimating the performance of the model on unseen data, can set the right balance that ensures the model is neither too simple to capture the patterns in the training data nor too complex to generalize beyond the training data (See *Figure 23*).

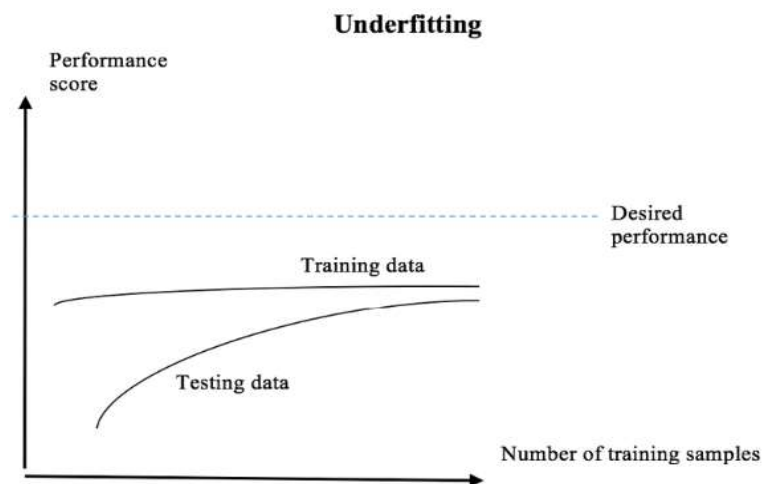


Figure 26 - High Variance (Underfitting)

3 Methodology

3.1 Technological Review

This section contains a review of the popular deep learning frameworks and tools used in the development, training, and deployment deep learning models.

3.1.1 Deep Learning Frameworks

Deep learning frameworks are essential tools used in the development, training, and deployment deep learning models. They provide the necessary abstractions and utilities to manage complex neural network architectures, handle large datasets, and optimize computational performance.

TensorFlow¹ was developed by the Google Brain Team². It uses Tensors as the fundamental data structures for computations. Operations in TensorFlow are executed as directed acyclic graph (DAG) where the nodes represent operations (e.g., addition, multiplication) and edges represent tensors flowing between operations. This graph-based approach enhances computational efficiency, especially in parallel and distributed environments (Abadi *et al.*, 2016).

PyTorch³ was developed by Facebook's AI Research lab (FAIR)⁴. It has gained popularity for its dynamic computation graph and simplicity in model building and debugging (Lambeta *et al.*, 2021; Paszke *et al.*, 2019). Unlike TensorFlow's static graphs, PyTorch provides dynamic computation graphs, allowing for more flexibility and ease of use during model development. It can also work with TorchScript to enable the transition from eager execution to a more optimized and production-friendly execution mode (Abadi *et al.*, 2016; Paszke *et al.*, 2019).

Keras⁵ is a high-level neural network API written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Its highly modular structure abstracts the complexities of backend computations. This flexibility facilitates the combination of different neural network

¹ <https://www.tensorflow.org/>

² <https://research.google.com/teams/brain>

³ <https://pytorch.org/>

⁴ <https://ai.meta.com/research/fair-paris/>

⁵ <https://keras.io/>

components such as layers, optimizers, activation functions in a straightforward manner (Chollet, 2015). As part of the TensorFlow ecosystem, it also benefits from the robustness and scalability of the TensorFlow framework. Keras is completely integrated into TensorFlow from version 2.0+.

Apache MXNet⁶ is an open-source deep learning framework designed for efficiency and flexibility, supporting both symbolic and imperative programming (Chen *et al.*, 2015). Similar to Keras, it offers a high-level, easy-to-use API (Gluon) for building and training neural networks. It also has a strong integration with Amazon Web Services (AWS), making it the preferred choice for cloud-based deep learning tasks on Amazon SageMaker.

Caffe⁷ is a deep learning framework developed by the Berkeley Vision and Learning Center (BVLC). Like Keras, it provides a rich collection of pre-trained models that can be easily fine-tuned for specific tasks (Jia *et al.*, 2014).

3.1.2 Development Environments and Tools

Due to the high number of concurrent operations performed when training a model, deep neural network frameworks like TensorFlow and PyTorch often take advantage of accelerated processing units like Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) with thousands of smaller cores to handle large volumes of data simultaneously. GPUs and TPUs are more power-efficient for deep learning tasks than CPUs, making them cost-effective for large-scale training.

Apart from development tools used in offline environments, there are cloud-based resources like Google Colab⁸, Kaggle⁹, and Amazon SageMaker¹⁰ that provide powerful computational resources and collaborative environments for deep learning projects. While Google Colab and Kaggle are excellent for prototyping and learning, Amazon SageMaker on the other hand, is well-suited for large-scale and enterprise-level machine learning solutions.

⁶ <https://mxnet.apache.org/>

⁷ <https://caffe.berkeleyvision.org/>

⁸ <https://colab.research.google.com/>

⁹ <https://www.kaggle.com/>

¹⁰ <https://aws.amazon.com/sagemaker/>

3.2 Benchmark Datasets and Challenges

Datasets are fundamental requirements for the development of machine learning models (Tsung-Yi *et al.*, 2015; Krizhevsky, Sutskever and Hinton, 2017). Since, CXR are critical for advancing diagnosis of various lung conditions through medical imaging research, a large catalogue of professionally annotated CXR datasets can serve as the fundamental instruments for advancing medical image analysis through machine learning (Majkowska *et al.*, 2020; Mery, 2015; Jaeger *et al.*, 2014).

In this section, we provide a review of two notable public CXR benchmark datasets used for medical imaging research.

3.2.1 NIH Chest X-Ray 8/14

Chest X-Ray 14 is one of the largest publicly available CXR datasets from the US National Institute of Health (NIH). It contains 112,120 frontal-view CXR images of 30,805 unique patients with 14 common thorax disease categories: Atelectasis, Consolidation, Infiltration, Pneumothorax, Edema, Emphysema, Fibrosis, Effusion, Pneumonia, Pleural_Thickening, Cardiomegaly, Nodule, Mass and Hernia (Wang *et al.*, 2017).

Chest X-Ray 8 is an extension of the Chest X-Ray 14 dataset. It comprises of 108,948 frontal view CXR images of 32,717 unique patients (collected from 1992 to 2015) with the text-mined 8 common disease labels, mined from the text radiological reports via NLP techniques (Wang *et al.*, 2017). The images are presented in PNG format of 1024*1024 resolution size. Approximately 1000 CXR images were annotated with bounding boxes location of disease label categories (see *Figure 27*).

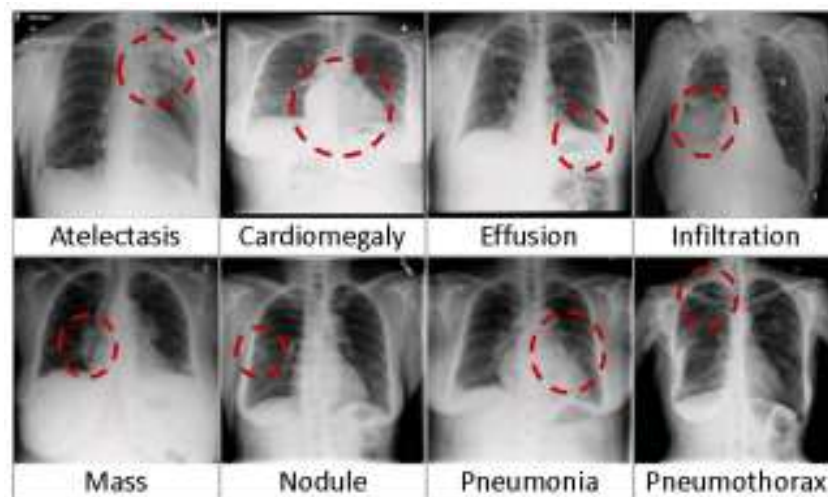


Figure 27 – Eight common thoracic diseases observed in chest X-rays (Wang *et al.*, 2017)

However, the authors acknowledged that some label classes were underrepresented (see *Figure 28*). This can impact the training of some machine learning models (Litjens *et al.*, 2017; Shreffler and Huecker, 2020; Glorot and Bengio, 2010).

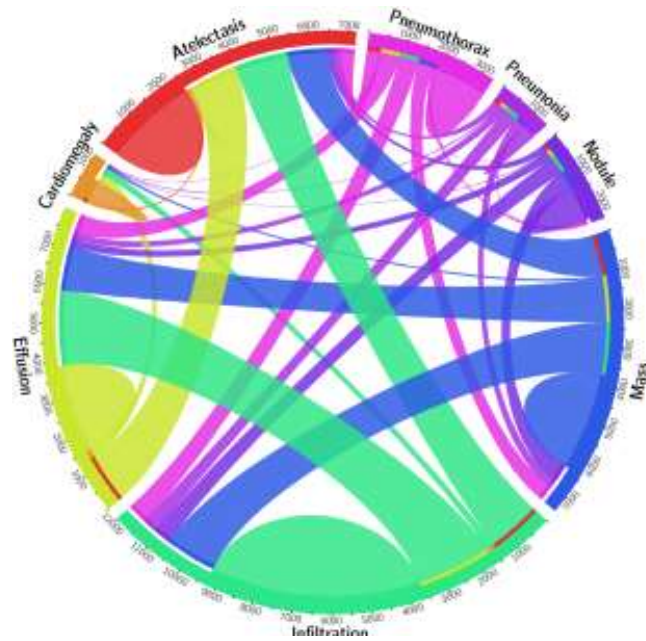


Figure 28 – Proportion of images with multi-labels in each of 8 pathology classes

3.2.2 CheXpert

This CXR dataset consists of 224,316 retrospectively collected chest radiographs of 65,240 patients with 14 common chest radiographic observations from Stanford Hospital (between October 2002 and July 2017), along with their associated radiology reports (Rajpurkar *et al.*, 2017). They used a labelling tool to extract observations as positive, negative, or uncertain from the free-text radiology reports. This is shown in *Table 2* below.

Pathology	Positive (%)	Uncertain (%)	Negative (%)
No Finding	16627 (8.86)	0 (0.0)	171014 (91.14)
Enlarged Cardiom.	9020 (4.81)	10148 (5.41)	168473 (89.78)
Cardiomegaly	23002 (12.26)	6597 (3.52)	158042 (84.23)
Lung Lesion	6856 (3.65)	1071 (0.57)	179714 (95.78)
Lung Opacity	92669 (49.39)	4341 (2.31)	90631 (48.3)
Edema	48905 (26.06)	11571 (6.17)	127165 (67.77)
Consolidation	12730 (6.78)	23976 (12.78)	150935 (80.44)
Pneumonia	4576 (2.44)	15658 (8.34)	167407 (89.22)
Atelectasis	29333 (15.63)	29377 (15.66)	128931 (68.71)
Pneumothorax	17313 (9.23)	2663 (1.42)	167665 (89.35)
Pleural Effusion	75696 (40.34)	9419 (5.02)	102526 (54.64)
Pleural Other	2441 (1.3)	1771 (0.94)	183429 (97.76)
Fracture	7270 (3.87)	484 (0.26)	179887 (95.87)
Support Devices	105831 (56.4)	898 (0.48)	80912 (43.12)

Table 2 - CheXpert (14 label observations) (Rajpurkar *et al.*, 2017)

3.2.3 RSNA Pneumonia Detection Dataset

The RSNA Pneumonia dataset was created on Kaggle¹¹ as part of the RSNA Pneumonia Detection Challenge¹² by the Radiological Society of North America (RSNA) in 2018 to facilitate the development of machine learning models for detection of pneumonia from digital CXR images (America, 2018; Anouk Stein *et al.*, 2018). About 26,684 CXR images of 26,684 unique patients were sourced from the NIH Chest X-ray14 dataset (Wang *et al.*, 2017). After a rigorous selection process, expert radiologists were tasked to manually annotate the images with bounding boxes around areas of lung opacities that are indicative of pneumonia, ensuring that the dataset includes high-quality labels for training and validation purposes. Although the team did not directly detail demographic information about the patients, such as age, gender, or comorbidities, which could be useful for more comprehensive analyses, they believe the dataset can contribute to the development of clinical decision support tools that can assist radiologists in diagnosing pneumonia more efficiently and accurately (America, 2018; Kermany *et al.*, 2018).

3.3 Research Question

Recent advancements in deep learning have demonstrated that contemporary neural network architectures can attain human-level performance in various image-processing tasks, such as object detection and classification (Russakovsky *et al.*, 2015; Tsung-Yi *et al.*, 2015; He *et al.*, 2016; Huang, Liu, Van Der Maaten and Weinberger, 2017). However, the potential of these models has not been fully realized for detecting and classifying medical imaging modalities due to the lack of sufficient CXR data. With the recent availability of public CXR datasets such as NIH Chest X-Ray 8/14, CheXpert, and RSNA Pneumonia Detection, the question arises: *Can we harness the capabilities of advanced deep learning models like Mask-RCNN and ResNet to achieve, and possibly exceed, human-level performance in detecting and classifying pneumonia characterized as lung opacities in digital CXR images?*

3.4 Research Objectives

Pneumonia is a life-threatening disease condition that leads to the inflammation of the lungs. It is transferred to humans by various pathogens, including bacteria, viruses, fungi, and parasites. According to the World Health Organization (WHO), pneumonia is the single largest

¹¹ <https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/overview>

¹² <https://www.rsna.org/rsnai/ai-image-challenge/rsna-pneumonia-detection-challenge-2018>

infectious cause of death in children worldwide, accounting for approximately 15% of all deaths of children under 5 years old (Rudan *et al.*, 2008). In low- and middle-income countries, the burden of pneumonia is even more severe due to factors such as limited access to healthcare, malnutrition, and a higher prevalence of underlying health conditions (Rajaratnam *et al.*, 2010). Although pneumonia is a common condition, accurately diagnosing it is a challenging process that requires the evaluation of a chest radiographs by highly trained specialists, along with confirmation through clinical history, vital signs, and other laboratory examinations (World Health Organization, 2001; Cherian *et al.*, 2005). The integration of deep learning techniques offers a promising solution to these challenges by providing automated, accurate, and efficient image analysis tools (Visuña, Yang, Garcia-Blas and Carretero, 2022; Rajpurkar *et al.*, 2017).

The primary aim of this research is to evaluate and enhance the application of deep learning techniques for abnormality detection and classification of CXR images with the digital CXR image and annotations presented for the 2018 RSNA Pneumonia Detection Challenge (Anouk Stein *et al.*, 2018). To support this objective, we will conduct training experiments using different configuration on a Mask-RCNN model with a ResNet50/101 backbone by applying optimization techniques, such as data augmentation, regularization methods (e.g., dropout, weight decay), and learning rate schedules, to enhance model performance using quantitative and qualitative assessment metrics.

3.5 Interpreting Pneumonia from Chest Radiographs

Traditional methods of interpreting pneumonia from CXR radiographs usually involves several steps to accurately diagnose and evaluate the condition.



Figure 29 – Sample radiograph from a typical CXR examination

The process typically begins with a CXR examination capturing either the anteroposterior (AP), posteroanterior (PA), or lateral view of the patient's lungs. The CXR image is then processed to enhance and highlight the visibility of anatomical structures (Cherian *et al.*, 2005). This is followed by a visual examination of the CXR image by a trained radiologist or medical professional to assess the lungs for areas of increased opacity, which may indicate fluid, pus, or other materials in the alveoli that is characteristic of a patient with pneumonia (World Health Organization, 2001; Cherian *et al.*, 2005). An example of a typical CXR radiograph is presented in *Figure 29*.

With recent advances in artificial intelligence and deep learning, automated systems can be developed to assist in interpreting digital CXR radiographs. The system can help in preprocessing the images to enhance the picture quality. Furthermore, the system can automatically identify areas of opacity, air bronchograms, and other signs indicative of pneumonia (Visuña, Yang, Garcia-Blas and Carretero, 2022; Rajpurkar *et al.*, 2017). This can improve diagnostic accuracy by providing a second opinion, highlight areas of concern. For example, the CXR image shown in *Figure 30*. The highlighted mask maps the area of the lung infected with pneumonia.

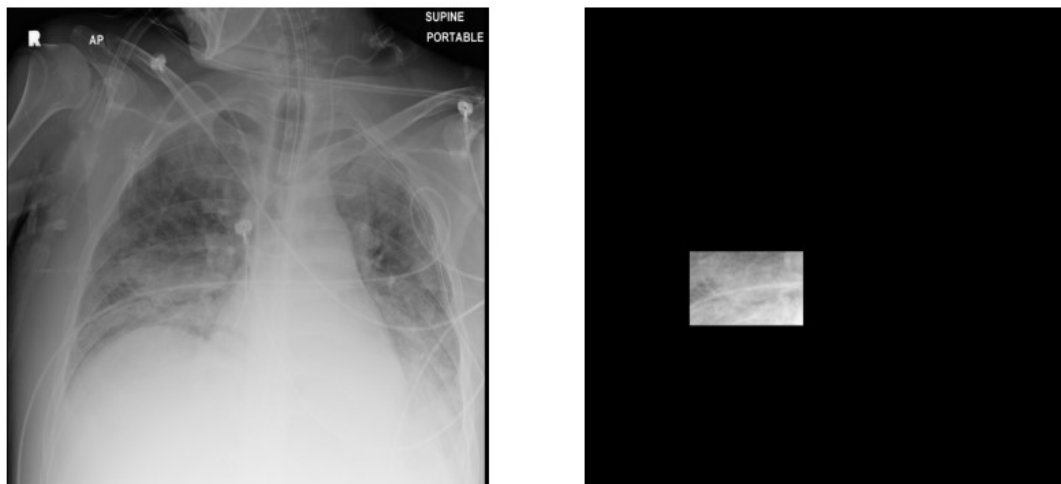


Figure 30 – Sample CXR radiograph (with mask location indicating lung opacity)

3.6 Model Architecture

Region-based Convolutional Neural Networks (R-CNN), Fast R-CNN, Faster R-CNN and Mask R-CNN are a series of models developed to advance the field of object detection. They generally use a selective search algorithm based on RPN to identify RoIs to apply feature maps

instead of taking the entire image as input (Uijlings, van de Sande, Gevers and Smeulders, 2013).

On this progression, Mask R-CNN builds on the strengths of Faster R-CNN by adding a branch for predicting segmentation masks for each RoI, enabling precise object segmentation (He *et al.*, 2017). An example of object classification (“person” and “ball” classes), detection (bounding box), and segmentation (coloured masks) from a trained Mask R-CNN model was shown in *Figure 13*. The mask prediction head of the Masked R-CNN model archived mean Average Precision (mAP) of 35.7 on MS-COCO at 5 frames/second (He, Gkioxari, Dollar and Girshick, 2020).

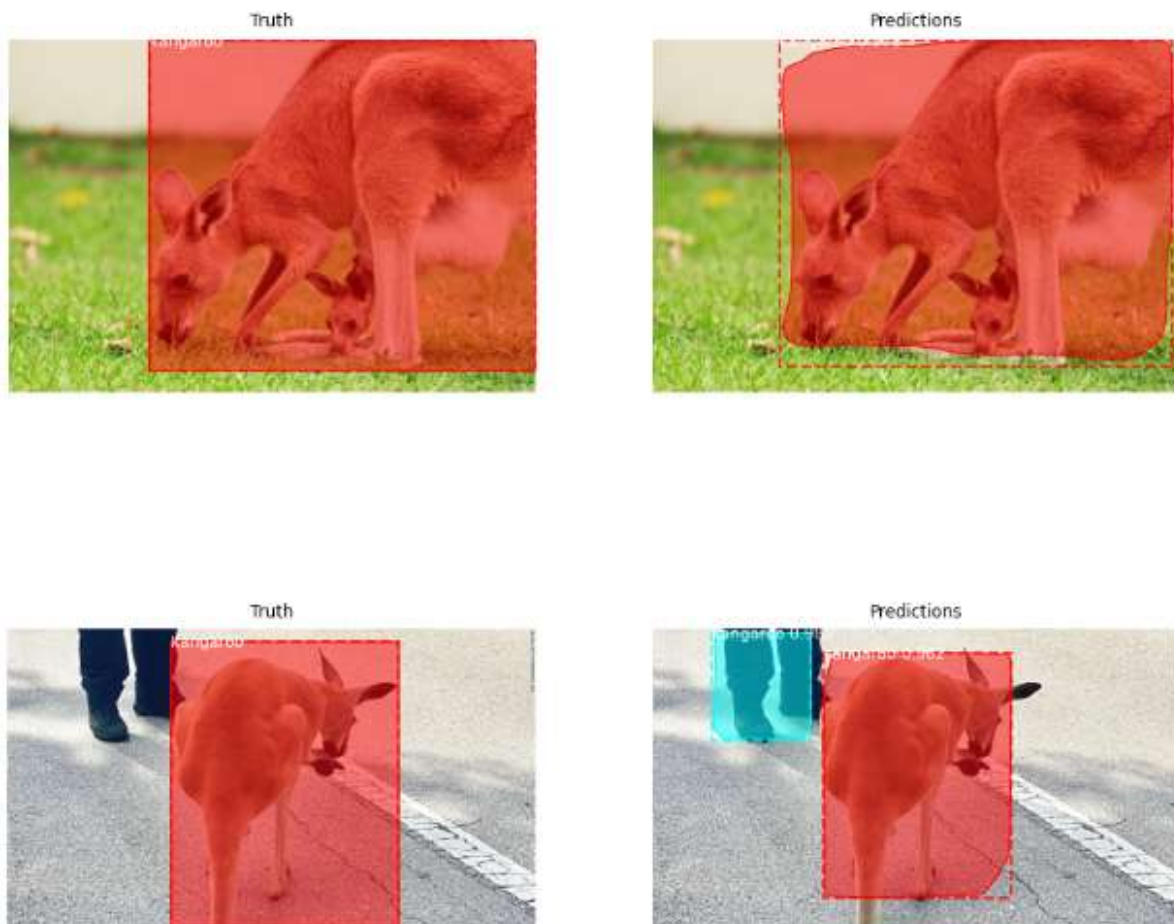


Figure 31 - Image analysis from demo project

This project is based on an open-source implementation of the Mask R-CNN model by *Waleed Abdulla* (Waleed Abdulla, 2017). This implementation is published under the MIT license which allows the modification and distribution rights without restrictions. It consists of the

following five main modules written in Python 3.6, TensorFlow 1.3 and Keras 2.0.8 in the folder *mrcnn* which can be found in *Waleed Abdulla's* GitHub repository¹³.

model.py – This module contains the main code implementation of the Mask R-CNN model architecture using the Keras API in TensorFlow. It includes classes and methods to facilitate the following:

- The construction of the backbone network (typically ResNet50 and 101 variants).
- Region Proposal Network (RPN) for generating region proposals.
- Region of Interest (RoI) pooling and alignment.
- Prediction heads for bounding box regression, class prediction, and mask prediction.

This module also contains functions for compiling the model, training, and inference.

parallel_model.py – This module initialises the model and adds multi-GPU support for models that need to be trained on more than one GPU.

config.py – This module contains the configuration settings for the Mask R-CNN model. It includes parameters that control the training process, such as learning rates, batch sizes, image dimensions, and the number of steps per epoch.

utils.py – This utility module includes various helper functions and classes used throughout the implementation. It contains functions for:

- Image manipulation and preprocessing (e.g., resizing, padding).
- Dataset handling and loading.
- Calculating metrics and managing bounding boxes.
- Visualizing predictions and ground truth data.

visualize.py – This module provides tools for visualizing the results of the Mask R-CNN model. It includes functions to:

- Display images with overlaid masks, bounding boxes, and class labels.
- Show training progress and evaluation metrics.

¹³ https://github.com/matterport/Mask_RCNN

- Generate and save visualizations of model predictions for analysis and debugging.

The original Mask R-CNN implementation from *Abdulla* has been independently upgraded to TensorFlow 2.x by various members of the open-source community. In this project, an upgraded copy of *Abdulla*'s original Mask R-CNN implementation was cloned from *Zunayed Mahmud*'s GitHub repository¹⁴ since it was complaint with Python 3.10 and TensorFlow 2.15+ with support for GPU with CUDA 12.2 and CuNN 8.6 on Google Colab¹⁵. It is imperative to use the implementation with support for TensorFlow 2.x as version 1.x was no longer permitted for use on Google Colab¹⁶. The current runtime environment on Google Colab comes preinstalled with TensorFlow 2.17, CUDA 12.2 and CuNN 8.6.

3.7 Model Training Environment

For this project, the implementation of the Mask R-CNN model was setup to run on Google Colab Pro+ with extended compute runtime. At development time (August 2024), the default Linux kernel runtime environment had TensorFlow 2.17, Keras 3.4.1, CUDA 12.2 and CuNN 8.6 preinstalled. The author of the upgraded Mask R-CNN model for TensorFlow 2.x confirmed it was tested on TensorFlow 2.14.1. However, after a review of the hardware acceleration issues on Google Colab for TensorFlow 2.10.x to 2.14.x, we discovered GPU acceleration was disabled in Google Colab due to version incompatibility of CUDA and CuNN¹⁷. To enable GPU hardware acceleration to facilitate speedy training of the model without impacting on other dependencies, we downgraded the versions of TensorFlow and Keras on the runtime environment to 2.15.0 respectively. We ran mock tests to confirm the setup was compatible with TensorFlow 2.14.1. The analysis of a sample image with the accompanying demo project is shown in *Figure 31*.

The compute resource used to model training is summarized in *Table 3* - Google Colab Pro+ compute resources below.

Runtime OS	<i>Linux kernel</i>
System RAM	<i>12 GB</i>
GPU RAM	<i>16 GB (T4)</i>
Hard Disk	<i>200 GB</i>

¹⁴ https://github.com/z-mahmud22/Mask-RCNN_TF2.14.0

¹⁵ <https://colab.research.google.com/notebooks/relnotes.ipynb>

¹⁶ <https://github.com/googlecolab/colabtools/issues/3266>

¹⁷ <https://github.com/googlecolab/colabtools/issues/4235>

3.8 Model Training Configurations

In this section, we detail the configuration settings used in the training run of our model. While largely adhering to the default settings, we specifically adjusted the number of epochs, the weight decay, and the backbone network to optimize our model's performance for the given task.

Two sets of model training schedules will be performed based ResNet-50 and 101 variants as the backbone network. The first schedule train only the network head with pre-trained MS-COCO weights. The second schedule will train the complete network with data augmentation.

3.8.1 Backbone Network

The backbone network is a critical component of the Mask R-CNN architecture that serve as the feature extractor for the subsequent stages of the model. In our experiments, we used both the ResNet-50 and 101 variants as the backbone network. This was done to explore the effects of computational complexity on model performance, both in terms of accuracy and speed.

3.8.2 Number of Epochs

The number of epochs represents the number of times the entire training dataset is passed through the network during training. We set the number of epochs to 25 at 200 steps per epoch to ensure the model has sufficient iterations to learn complex patterns from the CXR images. This choice is motivated by the need to achieve a balance between underfitting and overfitting.

3.8.3 Weight Decay

Weight decay is a regularization technique used to prevent overfitting by penalizing large weights in the model. In our training runs, we set the weight decay parameter to 0.001 for transfer learning with the MS-COCO weights, and also for the complete network. This value was selected to maintain the balance between model complexity and generalization, ensuring that the model does not overfit the training data.

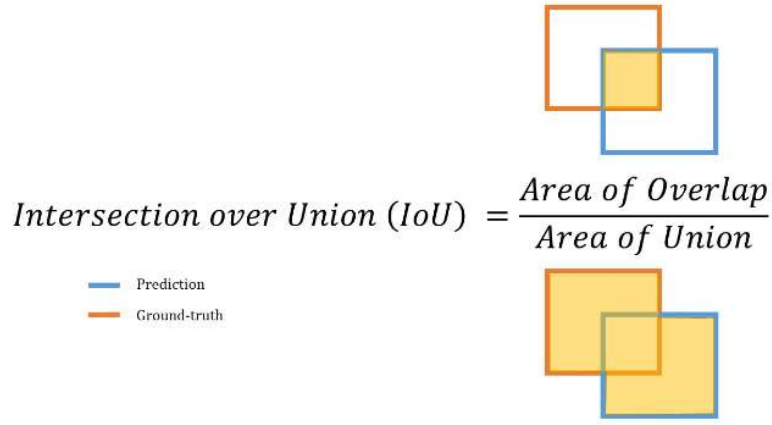
3.8.4 Data Augmentation

Data augmentation techniques such as horizontal flipping, vertical flipping, and slight rotations are applied to the training images during complete training runs. We used this technique to artificially increase the diversity of the training dataset during complete training runs to improve the robustness of the model and reduce the risk of overfitting.

4 Result and Discussion

4.1 Model Evaluation Method

In this section, we outline the evaluation method for assessing the performance of our Mask R-CNN model, with a focus on the mean Average Precision (mAP) and Average Recall (mAR). The mAP and mAR are widely recognized evaluation metrics in object detection and instance segmentation tasks, providing a comprehensive measure of a model's accuracy across different classes and intersection over union (IoU) thresholds (Russakovsky *et al.*, 2015). A graphical illustration of the definition of IoU is presented in Figure 1Figure 32.


$$\text{Intersection over Union (IoU)} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

— Prediction
— Ground-truth

Figure 32 - Graphical illustration of IoU definition (Anouk Stein *et al.*, 2018)

4.1.1 Precision and Recall

Precision is a model performance metric that measures the proportion of true positive detections among all positive detections, while recall measures the proportion of true positive detections among all ground truth instances. Both of these metrics depend on the number of true positive (TP), false positive (FP) and false negative (FN) predictions.

- **True Positives (TP):** Correctly detected objects where the predicted bounding box overlaps sufficiently with the ground truth bounding box. This is defined by the IoU threshold.
- **False Positives (FP):** Predicted bounding boxes that do not overlap sufficiently with the ground truth bounding box.
- **False Negatives (FN):** Ground truth objects that were not detected by the model.

Mathematically, precision and recall are defined in equation (16) and (17) respectively:

$$Precision = \frac{TP}{TP + FP} \quad (16)$$

$$Recall = \frac{TP}{TP + FN} \quad (17)$$

4.1.2 Average Precision and Recall

Average Precision (AP) computes the average precision value for recall value for each class over a range from 0 to 1. Mathematically, AP is defined by the integration of the area under the precision-recall curve. It provides a single scalar value that summarizes the model's performance. This is given in equation (18).

$$AP = \sum_n (R_n - R_{n-1}) P_n \quad (18)$$

where R_n and P_n are the recall and precision at the n th threshold.

4.1.3 Mean Average Precision, Recall and F1 Score

The mean Average Precision (mAP) and mean Average Recall (mAR) is the mean of the AP and AR values for all classes in the dataset. In our case, just one positive class (Lung Opacity). For region-based models, mAP and mAR is typically evaluated at multiple IoU thresholds to capture the model's performance comprehensively.

Average Precision (AP):	
AP	% AP at IoU=.50:.05:.95 (primary challenge metric)
AP _{IoU=.50}	% AP at IoU=.50 (PASCAL VOC metric)
AP _{IoU=.75}	% AP at IoU=.75 (strict metric)
AP Across Scales:	
AP _{small}	% AP for small objects: area < 32 ²
AP _{medium}	% AP for medium objects: 32 ² < area < 96 ²
AP _{large}	% AP for large objects: area > 96 ²
Average Recall (AR):	
AR _{max=1}	% AR given 1 detection per image
AR _{max=10}	% AR given 10 detections per image
AR _{max=100}	% AR given 100 detections per image
AR Across Scales:	
AR _{small}	% AR for small objects: area < 32 ²
AR _{medium}	% AR for medium objects: 32 ² < area < 96 ²
AR _{large}	% AR for large objects: area > 96 ²

Figure 33 - MS-COCO detection evaluation metrics

The most common IoU thresholds used are 0.5 and a range of thresholds from 0.5 to 0.95 in steps of 0.05. See Figure 33 (Tsong-Yi *et al.*, 2015). However, in our use-case, we will perform model evaluation with IoU thresholds set at 0.25, 0.50, 0.75 and 0.95. representing 25%, 50%, 75% and 95% respectively.

Mathematically, the mAP and mAR are computed as shown in equation (19) and (20).

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (19)$$

$$mAR = \frac{1}{N} \sum_{i=1}^N AR_i \quad (20)$$

where N is the number of classes, and AP_i and AR_i is the average precision and recall for class i . The F1 score is the harmonic mean of the precision and recall as shown in equation (21).

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (21)$$

The confusion matrix that summarizes the validation and testing instances by their predicted and actual values is usually presented as a contingency table as shown in *Figure 34*.

		Predicted	
		Negative	Positive
Actual	Negative	TN	FP
	Positive	FN	TP

TN = True Negative
 FP = False Positive
 FN = False Negative
 TP = True Positive

Figure 34 - Confusion matrix

4.2 Model Evaluation Process

The evaluation process of the model involves the following steps:

1. **Inference:** Running the model on a set of validation and test images to generate predicted bounding boxes, masks, and class labels.
2. **Match predictions to Ground Truth:** Using the IoU metric to match each predicted bounding box to the corresponding ground truth bounding box.
3. **Calculate AP and AR values:** Computing the AP and AR values for the positive class (Lung Opacity) at multiple IoU thresholds.
4. **Aggregate AP and AR values:** Computing the mean AP and AR values for the positive class to obtain the mAP and mAR.

By employing the mAP and mAR metrics, we ensure a comprehensive evaluation of the model performance, considering the precision and recall values across various detection thresholds

and classes. In our case, this is a single *Lung Opacity* class indicating the presence of pneumonia. The evaluation process provides a reliable explanation of the result generated, which is a key requirement in medical image analysis.

4.3 Model Performance Analysis

The comparative performance of the pre-trained (head with MS-COCO weights) and complete training schedule of our Mask R-CNN model on the validation dataset containing 1,500 samples is presented in *Table 4* – Evaluation result from model training schedules. The evaluation process focused on the mAP, mAR and F1 scores over the following IoU thresholds: 0.25, 0.50, 0.75 and 0.95 representing 25%, 50%, 75% and 95% (not included) of the area of overlap of the predicted bounding box over the ground truth bounding box instances as defined in sections 4.1 and 4.2 for the two (2) variants of the Mask R-CNN backbone: ResNet 50 and 101.

Network Backbone	Trained Layers	Is the Data Augmented?	IoU	mAP	mAR	F1
ResNet50	Head	No	0.25	0.9308	0.9530	0.9418
			0.50	0.6058	0.6655	0.6342
			0.75	0.1285	0.1825	0.1508
	Complete	Yes	0.25	0.8346	0.8402	0.8374
			0.50	0.4988	0.4794	0.4889
			0.75	0.0750	0.1032	0.0869
ResNet101	Head	No	0.25	0.8777	0.9062	0.8917
			0.50	0.5769	0.5868	0.5818
			0.75	0.0982	0.1557	0.1204
	Complete	Yes	0.25	0.8521	0.8762	0.8640
			0.50	0.5769	0.5868	0.5818
			0.75	0.0897	0.1482	0.1118

Table 4 – Evaluation result from model training schedules

The comparative performance analysis showed varying mAP, mAR, and F1 scores across different IoU thresholds on the ResNet 50 and 101 variants, with results from ResNet101 lagging behind across all thresholds. Although both ResNet50 and ResNet101 are designed to mitigate the vanishing/exploding gradient problem through their residual connections, the increased depth of ResNet101 can still make it more susceptible to this issue, particularly in situations where the network parameters are not carefully tuned (He *et al.*, 2016). This was essentially the problem in our case, since both the ResNet 50 and 101 variants were trained

using the same set of configuration parameters. As expected, the models trained with pre-trained MS-COCO weights out-performed the one with complete training on all layers (even with data augmentation). This is possible due to (1) number of training samples in the MS-COCO dataset (Tsung-Yi *et al.*, 2015), and (2) training is optimized for the network head rather than the complete network (Rahman *et al.*, 2020). See the highlighted portion of the report presented in *Table 4*. The graphical chart and trace logs comparing the training and validation losses of the best performing training run (highlighted in report presented in *Table 4*) is presented in **Error! Reference source not found.** and *Table 5* respectively.

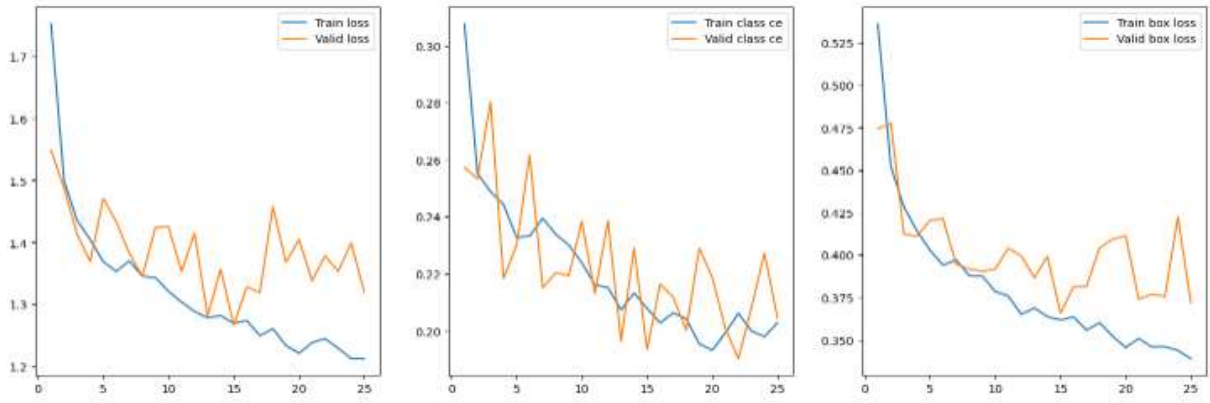


Figure 35 - Report from the best performing training run

	loss	rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss
1	1.752046	0.051905	0.394385	0.307649	0.538041	0.481997	1.548407	0.041897	0.373562	0.257273	0.474339	0.401337
2	1.499180	0.035085	0.376938	0.255174	0.452193	0.378272	1.487394	0.032923	0.333333	0.253434	0.477501	0.390204
3	1.435185	0.034618	0.350251	0.248873	0.428331	0.373093	1.413174	0.032898	0.338140	0.280311	0.412483	0.351373
4	1.404551	0.034442	0.343892	0.244274	0.414132	0.387811	1.389338	0.031222	0.354739	0.218423	0.410930	0.354022
5	1.388931	0.030091	0.339338	0.232780	0.402790	0.383932	1.470895	0.035543	0.398791	0.230059	0.420478	0.387754
6	1.353147	0.031074	0.337140	0.233241	0.393907	0.357784	1.432980	0.034527	0.344220	0.261659	0.421194	0.371380
7	1.370178	0.032099	0.335583	0.239538	0.397134	0.384846	1.382237	0.028788	0.375932	0.215125	0.394590	0.387804
8	1.345339	0.030699	0.337897	0.233887	0.387855	0.350020	1.345230	0.032341	0.348250	0.220398	0.391839	0.354805
9	1.343207	0.031182	0.337289	0.230090	0.387870	0.357016	1.424035	0.035588	0.412895	0.219352	0.390379	0.365821
10	1.321230	0.031009	0.332408	0.223882	0.378485	0.355486	1.428254	0.033984	0.394431	0.238557	0.391788	0.368535
11	1.303879	0.029788	0.328347	0.218180	0.375793	0.355772	1.353318	0.032059	0.345888	0.213038	0.404015	0.358320
12	1.288545	0.029047	0.329714	0.215189	0.384929	0.349688	1.414863	0.031273	0.373095	0.238714	0.399374	0.372236
13	1.278715	0.029049	0.322372	0.207335	0.389737	0.351222	1.281588	0.024484	0.308248	0.198438	0.385489	0.367972
14	1.281981	0.030021	0.325341	0.213321	0.383540	0.349659	1.358736	0.033515	0.339780	0.220150	0.399198	0.355114
15	1.269582	0.028384	0.319938	0.207881	0.381820	0.351780	1.268983	0.028280	0.328820	0.193548	0.385829	0.352887
16	1.273908	0.027885	0.327288	0.202803	0.383537	0.352414	1.328413	0.031048	0.344185	0.218344	0.381355	0.355483
17	1.249820	0.027348	0.311872	0.208289	0.355711	0.348820	1.319051	0.028207	0.339750	0.211848	0.381443	0.357803
18	1.260805	0.027808	0.315988	0.204265	0.380093	0.352670	1.457797	0.037477	0.454832	0.200188	0.404221	0.381079
19	1.233849	0.028400	0.310779	0.195441	0.352077	0.347151	1.387868	0.031371	0.340359	0.229002	0.409225	0.357911
20	1.221313	0.027281	0.308630	0.193258	0.345488	0.348679	1.404548	0.033717	0.384518	0.218748	0.411337	0.358232
21	1.238533	0.028103	0.314053	0.199331	0.350787	0.348278	1.338446	0.034533	0.375518	0.200917	0.373878	0.359802
22	1.244794	0.027984	0.318248	0.208202	0.345812	0.348550	1.378357	0.027901	0.418433	0.190242	0.378823	0.364959
23	1.229454	0.027293	0.306745	0.199939	0.346985	0.349512	1.353222	0.028791	0.399988	0.205295	0.375533	0.342645
24	1.212876	0.026873	0.300018	0.198001	0.343715	0.344488	1.398846	0.031688	0.340003	0.227259	0.422302	0.368594
25	1.212538	0.026480	0.301283	0.202895	0.338884	0.343053	1.320041	0.028909	0.384599	0.204480	0.371702	0.352341

Table 5 - Model training log comparing training and validation losses

Note: Zoom for better viewing experience

```
[ ] # print the best epoch
best_epoch = np.argmin(history["val_loss"])
print("Best Epoch:", best_epoch + 1, history["val_loss"][best_epoch])
```

Best Epoch: 15 1.2669630765914917

Figure 36 - Best epoch of training run

From the graph presented in *Error! Reference source not found.*, there are indications that the model could have converged again if allowed to run for more than 25 epochs. However, given the limited time available for this project and the time it takes to complete a single training schedule, it was difficult to extend the training schedule more than the 24-hour session time-out period set on Google Colab Pro+.

In *Figure 37* and *Figure 38*, we present graphical illustrations of the model's performance with samples from the validation and training datasets respectively. The images presented in the first column represent the ground truth of abnormality locations in the validation and testing datasets. The second and third columns represent the prediction of corresponding class and mask locations.

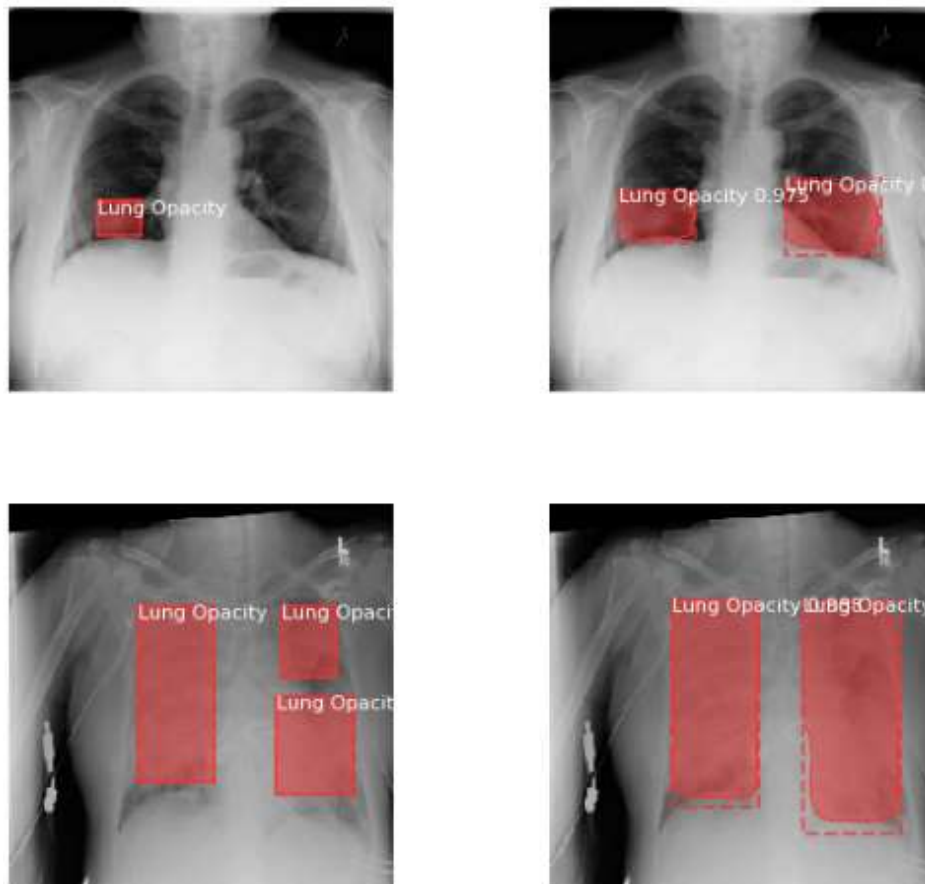


Figure 37 - Model performance with validation dataset

Furthermore, it is important to state that we were not able to achieve 95% IoU during any of the initial training runs as the results for all metrics was 0.0 for both the ResNet 50 and 101 variants. Though it was an ambitious try, we could tweak the model to attain some result in this category in future training schedules.

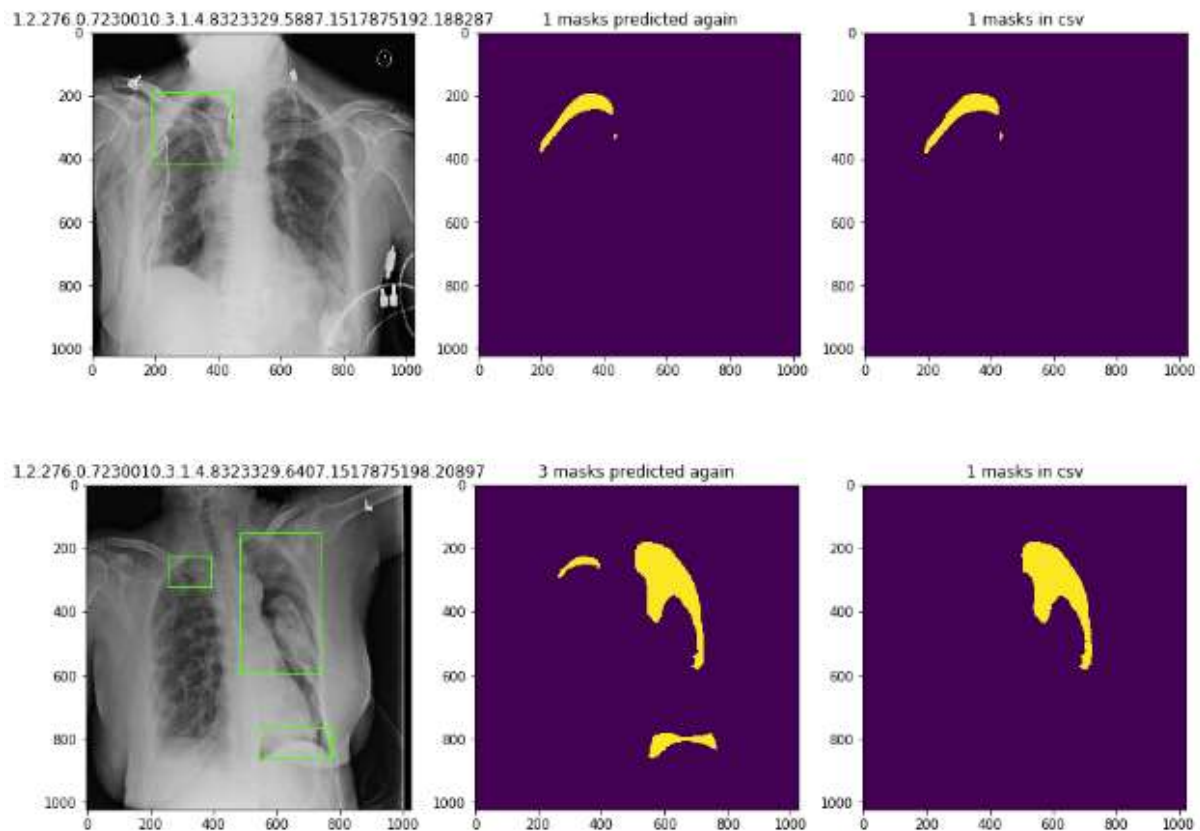


Figure 38 - Model performance with testing dataset

5 Conclusion and Further Work

Chest X-Ray (CXR) imaging holds a critical place due to its widespread use in diagnosing a multitude of thoracic diseases such as pneumonia, tuberculosis, and lung cancer (World Health Organization, 2001). Traditionally, the interpretation of CXR has been the domain of skilled radiologists who meticulously analyse these images to identify and classify abnormalities. However, this process is time-consuming, subjective, and prone to human error. Additionally, the increasing volume of imaging data in clinical settings necessitates the development of automated systems to assist radiologists (Rajpurkar *et al.*, 2017; Wang *et al.*, 2017).

Recent advances in deep learning capabilities have shown that modern neural network architectures can achieve human-level performance on several image-processing tasks, including object detection and classification (Russakovsky *et al.*, 2015; Tsung-Yi *et al.*, 2015; He *et al.*, 2016; Huang, Liu, Van Der Maaten and Weinberger, 2017). However, due to the lack of sufficient CXR data, the capacities of these models have not been fully leveraged for detection and classification of medical imaging modalities. This dissertation presented an evaluation of deep learning-based systems for the detection and classification of CXR images, with the goal of improving diagnostic accuracy and reducing the workload of healthcare professionals. To support our objective, we employed the benchmark dataset used for the RSNA Pneumonia Detection Challenge to train a Mask R-CNN model with ResNet50/101 backbone. To ensure robust and generalized model performance, data preprocessing and augmentation techniques was used to automatically improve the versatility of the dataset. The best performing model (ResNet50 pre-trained with MS-COCO weights) achieved an F1 score of 0.9418, 0.6342 and 0.1508 when the IoU threshold was set to 25%, 50% and 75% respectively. It is believed that this possibility was due to (1) number of training samples in the MS-COCO dataset (Tsung-Yi *et al.*, 2015), and (2) training was optimized for the network head rather than the complete network (Rahman *et al.*, 2020).

Further work is required to improve the performance of the model in the 50 - 75%, and possibly, up to the 95% IoU threshold bands in future training schedules. This might include tweaking the model configuration parameters. For example, training for more epochs with reduced learning rate and weight decay values. It will also be beneficial to extend the implementation of the model to federated environments with self or reinforcement learning methodologies to

manage the bias and variance that can result from data drift in live production environments (Lu *et al.*, 2018; Gama *et al.*, 2014). Finally, it would help to perform the experiments on these improvements with increased and well annotated data samples since this factor alone can have a severe impact on the performance of the model (Rajpurkar *et al.*, 2017; Wang *et al.*, Jul 2017; Giełczyk, Marciniak, Tarczewska and Lutowski, 2022).

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. and Zheng, X. (2016) 'TensorFlow: A system for large-scale machine learning', *arXiv.org*, Available at: 10.48550/arxiv.1605.08695 .
2. America, R.N. (2018) 'No title', *RSNA Pneumonia Detection Challenge: Can You Build an Algorithm That Automatically Detects Potential Pneumonia Cases*, .
3. Anouk Stein, M., Wu, C., Carr, C., Shih, G. and Dulkowski, J. (2018) 'RSNA pneumonia detection challenge', *kalpathy, Leon Chen and Luciano Prevedello, Marc Kohli MD, Mark McDonald, Peter, Phil Culliton, Safwan Halabi MD, and Tian Xia, "RSNA pneumonia detection challenge*, .
4. Bishop, C.M. (2010) *Pattern recognition and machine learning*. 10. (corr. printing) edn. New York [u.a.]: Springer.
5. Bottou, L. (2012) 'Stochastic gradient descent tricks' *Neural Networks: Tricks of the Trade: Second Edition* Springer, pp. 421–436.
6. Bottou, L. (2010) 'Large-scale machine learning with stochastic gradient descent', *Proceedings of COMPSTAT'2010: 19th International Conference on Computational Statistics Paris France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pp. 177–186.
7. Boureau, Y., Ponce, J. and LeCun, Y. (2010) 'A theoretical analysis of feature pooling in visual recognition', *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 111–118.
8. Boureau, Y., Ponce, J. and Lecun, Y. *A Theoretical Analysis of Feature Pooling in Visual Recognition*.
9. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C. and Zhang, Z. (2015) 'MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems', *arXiv.org*, Available at: 10.48550/arxiv.1512.01274 .

10. Chen-Yu Lee, Gallagher, P. and Zhuowen Tu (2018) 'Generalizing Pooling Functions in CNNs: Mixed, Gated, and Tree', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4), pp. 863–875. Available at: 10.1109/TPAMI.2017.2703082
Available at: <https://ieeexplore.ieee.org/document/7927440>.
11. Cherian, T., Mulholland, E.K., Carlin, J.B., Ostensen, H., Amin, R., Campo, M.d., Greenberg, D., Lagos, R., Lucero, M. and Madhi, S.A. (2005) 'Standardized interpretation of paediatric chest radiographs for the diagnosis of pneumonia in epidemiological studies', *Bulletin of the World Health Organization*, 83, pp. 353–359.
12. Chollet, F. (2015) *Keras (GitHub)*. Available at: <https://github.com/keras-team/keras>.
13. Chu, J.L. and Krzyżak, A. (2014) 'Analysis of Feature Maps Selection in Supervised Learning Using Convolutional Neural Networks', *Advances in Artificial Intelligence*, , pp. 59–70. Available at: 10.1007/978-3-319-06483-3_6 Available at: http://link.springer.com/10.1007/978-3-319-06483-3_6.
14. Dai, J., Li, Y., He, K. and Sun, J. (2023) 'R-FCN: Object Detection via Region-based Fully Convolutional Networks', *arXiv.org*, Available at: 10.48550/arxiv.1605.06409 .
15. Dominik Scherer, Andreas Müller and Sven Behnke (2010) 'Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition', *Artificial Neural Networks – ICANN 2010*, 6354, pp. 92–101. Available at: 10.1007/978-3-642-15825-4_10 Available at: http://link.springer.com/10.1007/978-3-642-15825-4_10.
16. Fukushima, K. (1980) 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position', *Biological Cybernetics*, 36(4), pp. 193–202. Available at: 10.1007/bf00344251 Available at: <https://www.ncbi.nlm.nih.gov/pubmed/7370364>.
17. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. and Bouchachia, A. (2014) 'A survey on concept drift adaptation', *ACM computing surveys (CSUR)*, 46(4), pp. 1–37.
18. Gancheva, V., Jongov, T. and Georgiev, I. (2023) *Medical X-ray Image Classification Method Based on Convolutional Neural Networks*. Springer Nature Switzerland, pp. 225 .
19. Giełczyk, A., Marciniak, A., Tarczewska, M. and Lutowski, Z. (2022) *Pre-processing methods in chest X-ray image classification*. Public Library of Science (PLOS).
20. Girshick, R. (2015) 'Fast R-CNN', *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

21. Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014) 'Rich feature hierarchies for accurate object detection and semantic segmentation', *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.
22. Glorot, X. and Bengio, Y. (2010) 'Understanding the difficulty of training deep feedforward neural networks', *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256.
23. Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep learning*. 1st edn. London, England: The MIT Press.
24. Gray, R.M. and Neuhoff, D.L. (1998) 'Quantization', *IEEE Transactions on Information Theory*, 44(6), pp. 2325–2383.
25. He, K., Gkioxari, G., Dollar, P. and Girshick, R. (2020) 'Mask R-CNN', *IEEE transactions on pattern analysis and machine intelligence*, 42(2), pp. 386–397. Available at: 10.1109/TPAMI.2018.2844175 Available at: <https://ieeexplore.ieee.org/document/8372616>.
26. He, K., Gkioxari, G., Dollár, P. and Girshick, R. (2017) 'Mask R-CNN', *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.
27. He, K., Zhang, X., Ren, S. and Sun, J. (2016) 'Deep residual learning for image recognition', pp. 770–778. Available at: 10.1109/CVPR.2016.90<https://ieeexplore.ieee.org/document/7780459>.
28. He, K., Zhang, X., Ren, S. and Sun, J. (2015) 'Spatial Pyramid Pooling in Deep Convolutional Networks for Visual recognition' *Computer Vision – ECCV 2014* Cham: Springer International Publishing, pp. 346–361.
29. Hou, J. and Gao, T. (2021) *Explainable DCNN based chest X-ray image analysis and classification for COVID-19 pneumonia detection*. Springer Science and Business Media LLC.
30. Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K.Q. (2017) *Densely Connected Convolutional Networks*. IEEE.
31. Hubel, D.H. and Wiesel, T.N. (1962) 'Receptive fields, binocular interaction and functional architecture in the cat's visual cortex', *The Journal of physiology*, 160(1), pp. 106–154. Available at: 10.1113/jphysiol.1962.sp006837 .
32. Ibraheem, N.A., Hasan, M.M., Khan, R.Z. and Mishra, P.K. (2012) *Understanding Color Models: A Review*.

33. Ioffe, S. and Szegedy, C. (2015) 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', *International conference on machine learning*, pp. 448–456.
34. Jaeger, S., Candemir, S., Antani, S., Wáng, Y.J., Lu, P. and Thoma, G. (2014) 'Two public chest x-ray datasets for computer-aided screening of pulmonary diseases', *Quantitative imaging in medicine and surgery*, 4(6), pp. 475–477. Available at: 10.3978/j.issn.2223-4292.2014.11.20 Available at: <https://www.ncbi.nlm.nih.gov/pubmed/25525580>.
35. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T. (2014) 'Caffe: Convolutional Architecture for Fast Feature Embedding', *arXiv.org*, Available at: 10.48550/arxiv.1408.5093 .
36. Jost, T.S., Dosovitskiy, A., Brox, T. and Riedmiller, M. (2015) *Striving for simplicity: The all convolutional net*. Ithaca: Cornell University Library, arXiv.org. Available at: <https://search.proquest.com/docview/2081662541>
37. Kermany, D.S., Goldbaum, M., Cai, W., Valentim, C.C.S., Liang, H., Baxter, S.L., McKeown, A., Yang, G., Wu, X., Yan, F., Dong, J., Prasadha, M.K., Pei, J., Ting, M.Y.L., Zhu, J., Li, C., Hewett, S., Dong, J., Ziyar, I., Shi, A., Zhang, R., Zheng, L., Hou, R., Shi, W., Fu, X., Duan, Y., Huu, V.A.N., Wen, C., Zhang, E.D., Zhang, C.L., Li, O., Wang, X., Singer, M.A., Sun, X., Xu, J., Tafreshi, A., Lewis, M.A., Xia, H. and Zhang, K. (2018) 'Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning', *Cell*, 172(5), pp. 1122–1131.e9. Available at: 10.1016/j.cell.2018.02.010 Available at: <https://dx.doi.org/10.1016/j.cell.2018.02.010>.
38. Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2017) 'ImageNet classification with deep convolutional neural networks', *Communications of the ACM*, 60(6), pp. 84–90. Available at: 10.1145/3065386 .
39. Lambeta, M., Xu, H., Xu, J., Po-Wei Chou, Wang, S., Darrell, T. and Calandra, R. (2021) *PyTouch: A Machine Learning Library for Touch Processing*. Ithaca: Cornell University Library, arXiv.org. Available at: <https://search.proquest.com/docview/2533577591>
40. Le Cun, Y. (1995) 'Convolutional networks for images, speech, and time series', *The handbook of brain theory and neural networks*, , pp. 255–258.
41. LeCun, Y. (1989) 'Generalization and network design strategies', *Technical Report*, .

42. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. and Jackel, L.D. (1989) 'Backpropagation Applied to Handwritten Zip Code Recognition', *Neural Computation*, 1(4), pp. 541–551. Available at: 10.1162/neco.1989.1.4.541
Available at: <https://direct.mit.edu/neco/article/doi/10.1162/neco.1989.1.4.541>.
43. LeCun, Y., Kavukcuoglu, K. and Farabet, C. (2010) 'Convolutional networks and applications in vision', pp. 253–256. Available at:
10.1109/ISCAS.2010.5537907<https://ieeexplore.ieee.org/document/5537907>.
44. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, 86(11), pp. 2278–2324.
45. LeCun, Y., Bottou, L., Orr, G.B. and Müller, K. (2002) 'Efficient backprop' *Neural networks: Tricks of the trade* Springer, pp. 9–50.
46. Lin, M., Chen, Q. and Yan, S. (2013) 'Network In Network', Available at:
10.48550/arxiv.1312.4400 .
47. Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B. and Belongie, S. (2017) 'Feature pyramid networks for object detection', *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125.
48. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., Van Der Laak, J.A., Van Ginneken, B. and Sánchez, C.I. (2017) 'A survey on deep learning in medical image analysis', *Medical image analysis*, 42, pp. 60–88.
49. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J. and Zhang, G. (2018) 'Learning under concept drift: A review', *IEEE Transactions on Knowledge and Data Engineering*, 31(12), pp. 2346–2363.
50. Maas, A.L., Hannun, A.Y. and Ng, A.Y. (2013) 'Rectifier nonlinearities improve neural network acoustic models', *Proc. icml*, pp. 3.
51. Majkowska, A., Mittal, S., Steiner, D.F., Reicher, J.J., McKinney, S.M., Duggan, G.E., Eswaran, K., Cameron Chen, P., Liu, Y., Kalidindi, S.R., Ding, A., Corrado, G.S., Tse, D. and Shetty, S. (2020) 'Chest Radiograph Interpretation with Deep Learning Models: Assessment with Radiologist-adjudicated Reference Standards and Population-adjusted Evaluation', *Radiology*, 294(2), pp. 421–431. Available at:
10.1148/radiol.2019191293 Available at:
<https://www.ncbi.nlm.nih.gov/pubmed/31793848>.

52. Masters, D. and Luschi, C. (2018) 'Revisiting small batch training for deep neural networks', *arXiv preprint arXiv:1804.07612*, .
53. Mery, D. (2015) *Computer Vision for X-Ray Testing : Imaging, Systems, Image Databases, and Algorithms*. 1st edn. Cham: Springer International Publishing.
54. Mishkin, D. and Matas, J. (2015) 'All you need is a good init', *arXiv preprint arXiv:1511.06422*, .
55. Nair, V. and Hinton, G.E. (2010) 'Rectified linear units improve restricted boltzmann machines', *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.
56. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Lu, F., Bai, J. and Chintala, S. (2019) 'PyTorch: An Imperative Style, High-Performance Deep Learning Library', *arXiv.org*, Available at: 10.48550/arxiv.1912.01703 .
57. Patro, S. and Sahu, K.K. (2015) 'Normalization: A preprocessing stage', *arXiv preprint arXiv:1503.06462*, .
58. Rahman, T., Chowdhury, M.E., Khandakar, A., Islam, K.R., Islam, K.F., Mahbub, Z.B., Kadir, M.A. and Kashem, S. (2020) 'Transfer learning with deep convolutional neural network (CNN) for pneumonia detection using chest X-ray', *Applied Sciences*, 10(9), pp. 3233.
59. Rajaratnam, J.K., Marcus, J.R., Flaxman, A.D., Wang, H., Levin-Rector, A., Dwyer, L., Costa, M., Lopez, A.D. and Murray, C.J. (2010) 'Neonatal, postneonatal, childhood, and under-5 mortality for 187 countries, 1970–2010: a systematic analysis of progress towards Millennium Development Goal 4', *The Lancet*, 375(9730), pp. 1988–2008.
60. Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Langlotz, C., Shpanskaya, K., Lungren, M.P. and Ng, A.Y. (2017) 'CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning', *arXiv.org*, Available at: 10.48550/arxiv.1711.05225 .
61. Ramachandran, P., Zoph, B. and Le, Q.V. (2017) 'Searching for activation functions', *arXiv preprint arXiv:1710.05941*, .

62. Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016) 'You Only Look Once: Unified, Real-Time Object Detection', *arXiv.org*, Available at: 10.48550/arxiv.1506.02640 .
63. Ren, S., He, K., Girshick, R. and Sun, J. (2015) 'Faster R-CNN: Towards real-time object detection with region proposal networks', *Advances in neural information processing systems*, 28.
64. Rudan, I., Boschi-Pinto, C., Biloglav, Z., Mulholland, K. and Campbell, H. (2008) 'Epidemiology and etiology of childhood pneumonia', *Bulletin of the World Health Organization*, 86, pp. 408–416B.
65. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L. (2015) *ImageNet Large Scale Visual Recognition Challenge*. Springer Science and Business Media LLC, pp. 211 .
66. Shelhamer, E., Long, J. and Darrell, T. (2017) 'Fully Convolutional Networks for Semantic Segmentation', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4), pp. 640–651. Available at: 10.1109/TPAMI.2016.2572683 Available at: <https://ieeexplore.ieee.org/document/7478072>.
67. Shreffler, J. and Huecker, M.R. (2020) 'Diagnostic testing accuracy: Sensitivity, specificity, predictive values and likelihood ratios', .
68. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. and Hassabis, D. (2017) 'Mastering the game of Go without human knowledge', *Nature*, 550(7676), pp. 354–359. Available at: 10.1038/nature24270 Available at: <https://www.ncbi.nlm.nih.gov/pubmed/29052630>.
69. Simard, P.Y., Steinkraus, D. and Platt, J.C. (2003) 'Best practices for convolutional neural networks applied to visual document analysis', pp. 958–963. Available at: 10.1109/ICDAR.2003.1227801 <https://ieeexplore.ieee.org/document/1227801>.
70. Simonyan, K., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
71. Sutskever, I., Martens, J., Dahl, G. and Hinton, G. (2013) 'On the importance of initialization and momentum in deep learning', *International conference on machine learning*, pp. 1139–1147.

72. Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) 'Going deeper with convolutions', pp. 1–9. Available at:
10.1109/CVPR.2015.7298594<https://ieeexplore.ieee.org/document/7298594>.
73. The DeepRadiology Team (2018) 'Pneumonia Detection in Chest Radiographs', *arXiv (Cornell University)*, Available at: 10.48550/arxiv.1811.08939 Available at:
<https://arxiv.org/abs/1811.08939>.
74. Tsung-Yi, L., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. and Dollár, P. (2015) *Microsoft COCO: Common objects in context*. Ithaca: Cornell University Library, arXiv.org. Available at:
<https://search.proquest.com/docview/2081545902>
75. Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T. and Smeulders, A.W.M. (2013) 'Selective Search for Object Recognition', *International journal of computer vision*, 104(2), pp. 154–171. Available at: 10.1007/s11263-013-0620-5 .
76. Visuña, L., Yang, D., Garcia-Blas, J. and Carretero, J. (2022) *Computer-aided diagnostic for classifying chest X-ray images using deep ensemble learning*. Springer Science and Business Media LLC.
77. Waleed Abdulla (2017) 'Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow', *GitHub repository*, Available at:
https://github.com/matterport/Mask_RCNN.
78. Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M. and Summers, R.M. (2017) 'ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases', *arXiv.org*, Available at: 10.48550/arxiv.1705.02315 .
79. World Health Organization (2001), *Standardization of interpretation of chest radiographs for the diagnosis of pneumonia in children*, .
80. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. and Dean, J. (2016) *Google's neural machine translation system: bridging the gap between human*

and machine translation. Ithaca: Cornell University Library, arXiv.org. Available at:

<https://search.proquest.com/docview/2080906303>

81. Xu, B., Huang, R. and Li, M. (2016) 'Revise saturated activation functions', *arXiv preprint arXiv:1602.05980*.
82. Zeiler, M.D. and Fergus, R. (2013) *Stochastic Pooling for Regularization of Deep Convolutional Neural Networks*. Ithaca: Cornell University Library, arXiv.org. Available at: <https://search.proquest.com/docview/2084992017>
83. Zhao, H., Shi, J., Qi, X., Wang, X. and Jia, J. (2017) 'Pyramid scene parsing network', *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2881–2890.