Automated Localisation and Classification of Trauma Implants in Leg X-rays through Deep Learning

by
Matthew Clive Swanevelder

Research assignment presented in partial fulfilment of the requirements for the degree of MEng (Structured) (Industrial Engineering) at the Faculty of Engineering at Stellenbosch University

Supervisor: Prof Andries P. Engelbrecht
Co-supervisor: Dr Franz F. Birkholtz

March 2024

Declaration

By submitting this mini-dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2024

Abstract

Revision surgery often requires orthopedic surgeons to pre-operatively identify failed implants in order to reduce the complexity and cost of the surgery. Surgeons typically examine the X-rays of a patient for preoperative implant identification, even though this method is time-consuming and occasionally unsuccessful. This study investigates the use of deep learning to automate the identification of trauma implants in leg X-rays. The investigation assesses the performance of various object detection and classification models on a dataset of trauma implants, aiming to identify the optimal deep learning solution. Challenges related to research include limited data, imbalanced class distributions, and the presence of multiple implants in the X-ray images.

The results of the investigation indicate that the optimal deep learning solution is a two-model pipeline that employs a you only look once (YOLO) object detection model and a densely connected convolutional neural network (DenseNet) classification model. The DenseNet classification model classifies the trauma implants localised by the YOLO object detection model. The proposed pipeline achieves a mean average precision (intersection of union threshold of 0.5) of 0.967 for implant localisation and an accuracy of 73.7% for implant classification. The results of the study provide proof that deep learning models are capable of identifying trauma implants. Additionally, the study offers a deep learning solution that can be utilised in future research related to identifying trauma implants.

Opsomming

Daar word dikwels, in die beplanningsfase van hersieningschirurgie, van ortopediese chirurge vereis om die mislukte implantate te identifiseer en dus die kompleksiteit en koste van die operasie te minimaliseer. Die proses behels in die algemeen die deeglike ondersoek van X-strale om voor chirurgie die implantate te identifiseer, alhoewel hierdie metode tydrowend en soms onsuksesvol is. Hierdie studie ondersoek die gebruik van diep-leer modelle om die identifikasie van trauma-implantate in X-strale te outomatiseer. Te same met dit, word verskeie voorwerp-opsporing en klassifikasie modelle evalueer met data van trauma-implantate, om 'n optimale diep-leer oplossing te identifiseer. Uitdagings gebonde aan hierdie projek sluit, onder andere, beperkte data, ongebalanseerde klas distribusies, en die teenwoordigheid van verskeie implantate in enkele X-strale in.

Die resultate van hierdie ondersoek toon aan dat die optimale diep-leer oplossing 'n twee-model pyplyn is wat 'n *you only look once* (YOLO) voorwerp-opsporing model en 'n *densely connected convolutional neural network* (DenseNet) klassifikasie model gebruik. Die DenseNet klassifikasie model klassifiseer die trauma-implantate wat deur die YOLO voorwerpdeteksie model gelokaliseer is. Die voorgestelde diep-leer pyplyn het 'n *mean average precision* (*intersection of union threshold* van 0.5) van 0.967 vir die lokalisering van implantate bereik en 'n akkuraatheid van 73.7% vir die klassifikasie van die implantate. Die resultate van die studie bied bewyse dat diep-leer modelle wel gebruik kan word om trauma-implante te identifiseer. Die studie bied ook 'n diep-leer oplossing wat in toekomstige navorsing gebruik kan word.

Acknowledgements

I would like to acknowledge the support of my supervisor, Prof Andries Engelbrecht, and my co-supervisor, Dr Franz Birkholtz, who provided valuable feedback and guidance throughout this project. In addition, I extend my gratitude to Dr Festus Iiyambula for his assistance in labeling the trauma implant data.

I would also like to thank my family for their unconditional love, patience, and understanding during my studies. Their constant encouragement kept me motivated. Lastly, I wish to express my gratitude to my partner, Elizabeth, whose unwavering moral support and assistance was invaluable throughout this project.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ANN	Artificial Neural Network
AP	Average Precision
API	Application Programming Interface
AUC	Area Under the Receiver Operating Characteristic Curve
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DenseNet	Densely Connected Convolutional Neural Network
FNN	Feed-Forward Neural Network
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IoU	Intersection of Union
JPEG	Joint Photographic Experts Group
mAP	Mean Average Precision
R-CNN	Regions with Convolutional Neural Networks
ReLU	Rectified Linear Unit

ResNet	Residual Network
RoI	Region of Interest
RPN	Region Proposal Network
YOLO	You Only Look Once

Chapter 1

Introduction

A survey conducted in 2012 on members of the American Association of Hip and Knee Surgeons (AAHKS) examined the methods used to pre-operatively identify implant types for revision arthroplasty [55]. The paper found that out of 605 responses, 96.3% of the members indicated that examination of a patient's X-ray was their most used method for pre-operatively identifying implants with other common methods involving the use of hospital operative records (78.1%) or orthopedic office/clinic records (71.3%). The survey also indicated that the estimated time spent on pre-operative identification of implants was 20 minutes per case for surgeons and 30 minutes per case for medical staff. Furthermore, the surgeons also reported that for approximately 10% of cases the failed implant components could not be identified pre-operatively. The background of the problem is examined on this chapter, and the research statement, objectives, project scope, and research methodology of this study are introduced. In addition, this chapter discusses how this study contributes to the field of data science.

1.1 Problem Background

Orthopedic revision surgery involves the removal or replacement of an orthopedic implant due to a fracture, infection, or implant failure. When performing revision surgery, it is essential for orthopedic surgeons to identify the implanted hardware pre-operatively in order to acquire the appropriate implant-specific tool kits and to reduce the complexity of the surgery. Surgeons often rely on the surgical notes and documentation to identify implants. However, there are many cases where the patient's surgical history

is not available and the surgeons have to use alternative approaches that are time-consuming and less reliable, such as visual comparisons of implant X-rays. Failure to pre-operatively identify implants can result in increased operating room procedure time, higher complexity surgery, and greater healthcare costs [55].

This problem presents an opportunity for the design of a deep learning product that can assist orthopedic surgeons in accurate identification of implants from X-rays and to reduce the time required to do so. The purpose of this research assignment is to complete the first step in designing this deep learning product which involves conducting a critical review of deep neural networks and proposing a deep learning solution. This review has to consider the following important aspects that are related to the problem, namely the shortage of training data, potential skew class distributions in the data, and that each input image could possibly contain multiple objects (implants) that need to be classified.

1.2 Problem statement

Pre-operative identification of failed orthopedic implants using X-rays is a common challenge faced by many orthopedic surgeons. Research suggests [55] that identification of failed implants using X-rays is time consuming and often futile. Failure to pre-operatively identify failed implants can increase surgery complexity and costs. Automation of orthopedic implant identification using deep learning can expedite the process and increase the number implants correctly identified. This project conducts a critical review of deep neural network models that are suitable for the automation of X-ray analysis. The models considered must be able to localise and classify different types of orthopedic trauma implants, such as plates, nails/rods, screws, wires or pins, in leg X-rays. Empirical analysis is used to identify the most accurate deep learning model.

The following research questions are derived from the problem statement to help guide the scope and objectives of the project:

1. What is the most effective deep learning model for trauma implant localisation in X-rays?

2. Which deep learning architecture is best for trauma implant classification?
3. How can impediments such as limited data samples and skew class distributions be addressed?
4. What are the main challenges related to automating trauma implant detection and classification?

1.3 Objectives

In order to answer the presented research questions, the following research objectives must be achieved:

1. Conduct a literature review of important topics related to the research problem.
2. Analyse the provided X-ray data and perform any necessary pre-processing. Annotate each image in the dataset by assigning a label and location coordinates (bounding boxes) to each implant. Collaborate with orthopedic surgeon, Dr Franz Birkholtz, to perform detailed implant labelling.
3. Train different object detection models to localise trauma implants. Train different classification models to perform detailed classification on the localised implants. Evaluate the effectiveness of each model using appropriate metrics.
4. Propose a final deep learning solution that combines the most effective object detection and classification model.

1.4 Scope

The ultimate objective is to create a solution that can classify implants, not just by their global type, but also identify the specific model and manufacturer of the implant. However, for pragmatic reasons, this study focuses on global implant classification and acts as a proof of concept for future research.

The deep learning models employed in this study are trained and tested on the dataset provided by Dr Franz Birkholtz. Due to time and data constraints the study focuses on detection and classification of trauma implants located in the leg and foot regions. To

help reduce the complexity of the problem only anterior-posterior and lateral X-rays are included in the dataset. Given the time constraint, the number of models evaluated in the study is restricted to three object detection models and four classification models.

1.5 Research methodology

The research methodology describes the approaches that are utilised to accomplish each of the project objectives.

The first objective involves conducting a literature review, which is crucial to the study, because the knowledge gained from the review helps to guide decisions related to the other objectives. In order to understand the provided data, the literature review first investigates different trauma implants as well as the medical imaging techniques used to examine implants. Literature covering neural networks, the training of neural networks, and computer vision concepts is then discussed to provide insight for the modelling and evaluation steps of the project. Lastly, different deep learning architectures and their healthcare applications are explored to direct the overall strategy used for the study and to assist in model selection.

For the second objective, all the images in the dataset are scaled to the same size, and image annotation is done using the *Labelme* software. Annotation involves outlining each implant in an image with a square bounding box and assigning the basic implant type (i.e. plate, screw, pin) to the bounding box. A separate classification dataset is generated by cropping each implant using the bounding box coordinates and then scaling all the cropped images to the same size. Detailed implant labelling is performed on the classification dataset with the help of an orthopedic surgeon. Image augmentation is used to increase the number of training samples and sample diversity of each dataset. After annotation and labelling of the data, analysis is done to determine whether the data has a skew class distribution. Further image processing is implemented to address skew class distributions or to improve model performance.

For objective three, four classification and three object detection models are trained

and evaluated to determine which models are most effective in trauma implant localisation and classification. The group of models selected for evaluation is based on models used in the medical applications discussed in the literature review. Object detection models are used to localise implants and classification models are used to classify the localised implants. Limited project data and time constraints mean that the classification models are trained and evaluated on data representing the basic implant type (i.e. plate, pin, screw) with the most balanced class distribution.

The final objective involves the combination of the best performing object detection and classification model to form a deep learning pipeline for implant detection and classification. The deep learning pipeline uses the object detection model to crop and segment the implants in the input image. The cropped regions of interest are then fed to the classification model to generate detailed label predictions.

1.6 Contributions to data science

This project contributes to data science by providing a proof of concept that can act as a stepping stone to the development of a deep learning product used for trauma implant detection and classification. The study also identifies core challenges related to trauma implant detection and classification that can provide insight for future studies. Additionally, this project contributes a labelled and annotated trauma implant dataset that can be used in future research.

1.7 Mini-dissertation outline

This document is organised as follows:

Chapter 2 provides a general overview of the different types of trauma implants and also reviews medical imaging techniques used to examine fractures and failed implants.

Chapter 3 is an introduction to deep learning that focuses on the architecture of convolutional neural networks (CNN) as well as concepts related to the training and testing of neural networks.

Chapter 4 examines the architectures of different object detection and deep learning neural networks that have been utilised for medical image analysis.

Chapter 5 reviews recent studies that investigate the use of deep learning models to detect and classify implants.

Chapter 6 covers the data analysis, data preparation, and overall procedure used to train and to evaluate the selected object detection and classification models.

Chapter 7 presents the results of the study, while chapter 8 delves into the discussion, offering a comprehensive analysis of the results.

Chapter 9 concludes the dissertation by summarising the research problem and the key findings of the project.

Chapter 2

Orthopedic Implants

The types of orthopedic implants used by surgeons can be divided into two main categories, namely joint replacement implants and temporary fracture fixation implants. This project focuses on fracture fixation implants, also referred to as trauma implants, which are used to temporarily join fractured bones in order to assist bone healing [22]. Chapter 2 provides a general overview of the different trauma implants and explains the imaging techniques used to assess skeletal trauma.

2.1 Trauma Implants

The main types of trauma implants are plates, screws, external fixators, intramedullary nails, cables, pins, and wires [10, 53]. Factors that have to be considered when selecting a trauma implant include the severity of the injury, injury location, patient acuity, mechanical demands of fractured bone, expected rate of healing, and proximity to skin, blood vessels, and nerves.

Bone screws are the most commonly used trauma implants. The different variations of bone screw are defined by the head, shaft, thread, and tip of the screw. These screw variations can be grouped into two primary groups, namely cancellous and cortical screws. Cancellous screws have larger threads with an increased pitch and are designed for the softer cancellous bone; whereas cortical screws are used in the harder cortical bone and have smaller threads and a lower pitch [53].

Fixation plates are secured using bone screws and act as internal splints that fixate broken bone fragments. The names used for different plates can be complex and are usually based on either the plate shape, the plate width, screw hole shape, surface contact characteristics or the intended site of application. Traditional classification utilises five general groups to describe plates, based on their function, namely protection, compression, buttress, bridge, and condylar plates [53]. Figure 2.1 illustrates the five general plates. The classification of modern plates also includes anatomical pre-contoured locking plates and straight locking plates. Locking plates have threaded holes that allow the head of a fixation screw to be locked in place [6].

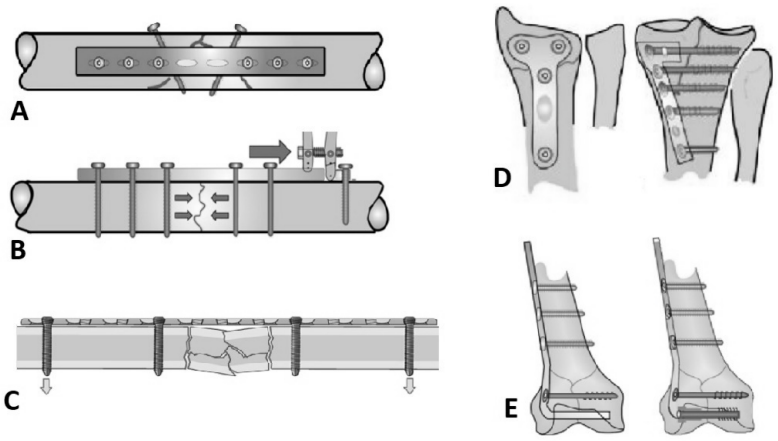


Figure 2.1: Plates: Protection (A), Compression (B), Bridge (C), Buttress (D), Condylar (E) [55]

External fixation devices are normally used for open or infected fractures and can be described as multiple percutaneous pins that are fixated into the affected bone and attached to an external scaffolding [10]. Figure 2.2 provides examples of pin fixators and ring fixators, which are the two main types of external fixators. Pin fixators are commonly used to efficiently stabilise diaphyseal fractures and are ideal for routine trauma surgery, whereas ring fixators are more suited for ongoing deformity correction, limb lengthening, and non-union management [53].

Intramedullary nails function as internal bone splints and extend from the one end of a long bone to the other via the medullary canal. The nail can be described as a

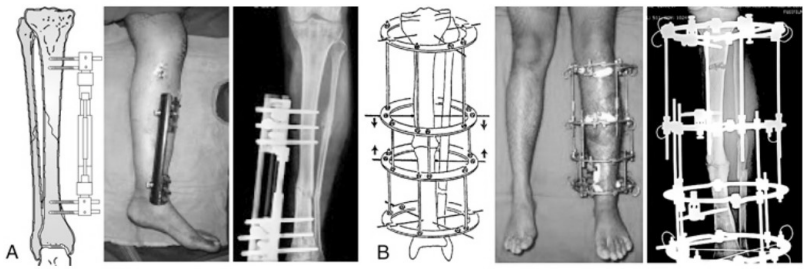


Figure 2.2: Pin fixator (A) and Ring fixator (B) [53]

load sharing device that allows forces to be transferred between the polar ends of the bone, while preventing angulation, translation, and rotation of the bone fragments. Nail designs can vary according to their extreme ends, cross sectional shape, diameter, curvature, length, composition material and need for supplementary fixation devices [53]. Intramedullary fixation can be done using a single locking nail or multiple flexible nails, with the latter approach commonly used for children and adolescents. The single nail systems can be divided into two main types, namely standard locking nails and cephalomedullary locking nails [4]. Figure 2.3 illustrates the different intramedullary nails.

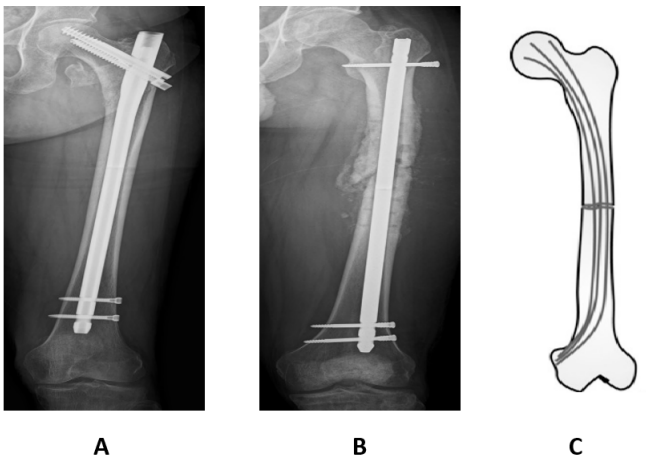


Figure 2.3: Cephalomedullary locking nail (A), Standard locking nail (B), Multiple flexible nails (C)

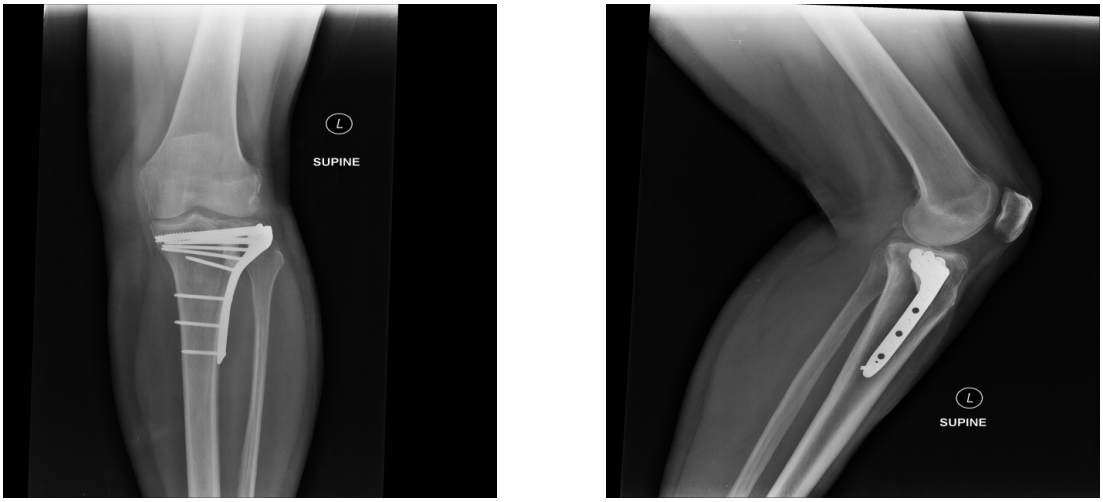
Wires and cables are regularly used for internal fixation and are employed in a cir-

clage or tension band manner. Circlage wiring is used to either treat certain long bone fractures or to reinforce fixation implants, while tension band wiring is typically used for fracture fixation in small spaces (e.g. patella fractures) and makes use of muscle activity to place compression on the fracture. Surgical pins, also known as Kirschner wires or K-wires, are straight wires that are resistant to bending and can be used for provisional or definitive fracture fixation. Pins are popular due to their versatility and small diameters which limit the trauma inflicted on soft tissue and bone.

2.2 Medical Imaging

Radiographs (plain X-rays) account for roughly 75% of all medical examinations and is the primary imaging modality used for fracture and fracture fixation assessments [14, 31]. Radiographs are produced by emitting an X-ray beam that passes through a patient and strikes a detector. The X-ray beam is absorbed in different amounts by the various tissue densities (i.e. air, water, fat, bone) in the body and the radiograph image is generated by the unabsorbed radiation striking the detector [31].

Examinations are generally done using a combination of different projections and the most general views used in radiography are the anterior-posterior, lateral and oblique views. For fractures along the shaft/diaphyseal segment of long bone a minimum of two views are required for a safe assessment, whereas for joint dislocations and fractures at the end segments of a bone, a minimum of three views are required (i.e. anterior-posterior, lateral, and oblique view) [40]. There are also many other specialized X-ray views for specific fracture patterns, which are beyond the scope of this review. Figure 2.4 illustrates the difference between the anterior-posterior and lateral view of a knee.



(a) Anterior-posterior view (b) Lateral view

Figure 2.4: Examples of different X-ray projections of a knee.

2.3 Summary

An important preliminary step in every data science project is to gain an understanding of the problem domain. Chapter 2 provided a short overview of the different types of common trauma implants, namely plates, screws, external fixators, intramedullary nails, cables, pins, and wires. The medical imaging techniques and projections used for fracture fixation assessments were also summarised.

Chapter 3

Deep Learning

Deep learning is a type of machine learning that utilises artificial neural networks with multiple processing layers to extract and learn complex patterns and relationships within data. Computer vision is a sub-field of deep learning that focuses on extraction of useful information from digital images and videos. Popular computer vision applications include image classification, object detection, image segmentation, and face recognition [26]. The applications discussed in this project utilise image classification and object detection models to locate and classify orthopaedic implants.

Image classification is the process of classifying an entire image into a specific class category, whereas object detection involves identification of the categories and locations of objects in an image [26]. Object detection models are divided into one-stage and two-stage detectors. The two-stage detectors first generate object region proposals, which are then passed to an object classification model. On the other hand, one-stage detectors predict the category probabilities and locations of objects directly from the input image.

The advancements in computer vision over the last decade, specifically in pattern recognition applications, has led to increased interest in medical image analysis. The extensive review papers by Litjens *et al.* [29] and Sahiner *et al.* [41] summarise recent papers related to deep-learning-based medical image analysis and reveal that the most popular type of neural network used, is the convolutional neural network and its derivatives. Chapter 3 provides an overview of convolutional neural networks and

discusses important concepts related to network training and optimisation. The topics discussed include machine learning fundamentals, convolutional neural networks, data preparation, transfer learning, and evaluation metrics.

3.1 Machine Learning Fundamentals

The main challenge faced in machine learning is to design a model that not only performs well on the data used to train the model, but also demonstrates strong performance on new unseen data. The generalisation of a model refers to how well the model reacts to new unseen data. Goodfellow *et al.* [15] use two factors to describe the effectiveness of a machine learning model, namely the model's ability to produce a small training error and the model's ability to achieve a small difference between the training and testing error. In this context, the training error refers to how well the model performs on the data used to train the model, while the testing error measures the performance of the model on unseen testing data.

Underfitting and overfitting are two common problems faced in machine learning that have an impact on the effectiveness of a model [15]. Underfitting occurs when a machine learning model is unable to achieve a satisfactory training error, while overfitting occurs when there is a significant difference between training error and testing error of a model. The likelihood of a model overfitting or underfitting is related to the *capacity* of the model [15]. A model with insufficient capacity is unable the learn the structure of the training data, thereby leading to the model underfitting. On the other hand, a model with too much capacity will memorise/overfit the training data resulting in poor performance on the unseen testing data. The capacity of a neural network is controlled by the number of layers and nodes in the network [15]. Thus, neural networks that are larger and more complex will possess greater capacities. It is important to note that the amount of model capacity utilised also relies on the effectiveness of the training process.

Figure 3.1 illustrates how the training and testing error relates to the model capacity. The red line indicates the optimal model capacity, which is the point where the model achieves a satisfactory training error and the difference between the training and testing error is marginal.

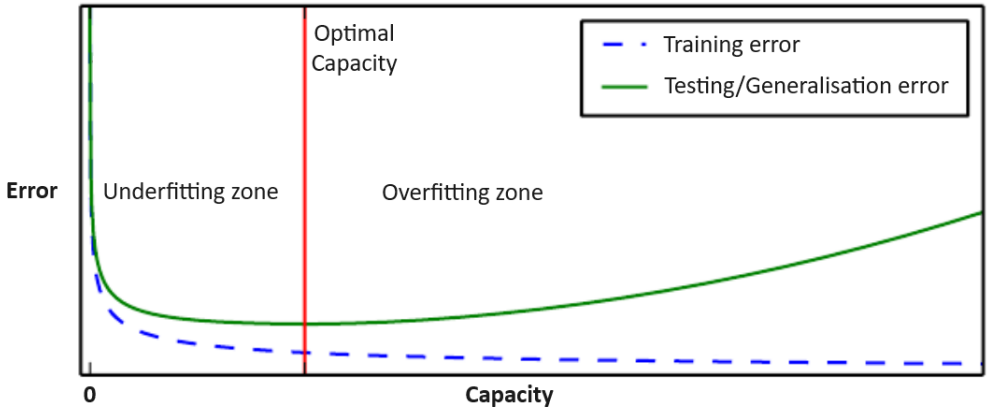


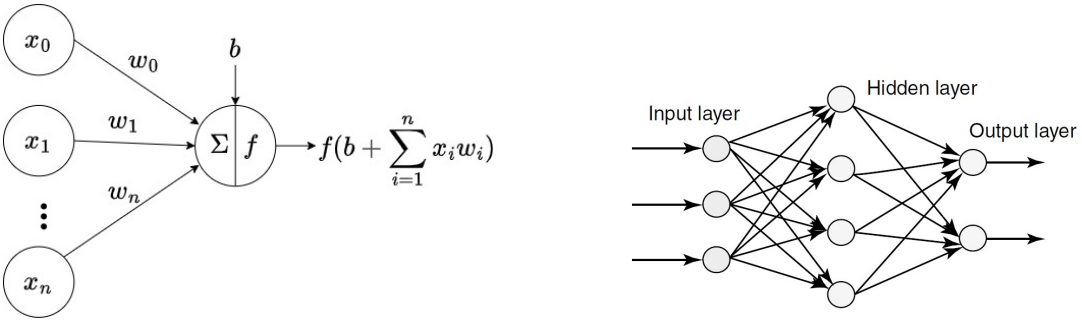
Figure 3.1: The relationship between model capacity and error [15].

3.2 Convolutional Neural Networks

The artificial neural network (ANN) [33] is a computational model inspired by the structures and functions of the biological nervous system. It is a collection of interconnected processing units, called artificial neurons or nodes, that process and transfer data to each other. Each node receives inputs from other nodes and computes the weighted sum of the inputs. The weighted sum is then passed to an activation function which determines whether the node should be activated or not by mapping the resulting values to a range of 0 to 1 or -1 to 1 [24]. To guarantee that some nodes in the network are activated even when the input signals are small, bias values are employed [34]. Biases are scalar values that are added to the weighted sum which is then passed to the activation function. Figure 3.2(a) illustrates a typical artificial node, where x_n refers to the n-th input signal, w_n the corresponding weight, b the node bias, and f the activation function. ANNs can be trained iteratively by adjusting the weights and biases of each artificial neuron in the network, using a chosen learning algorithm, until a desired input-output relationship is achieved.

All ANN architectures are comprised of the same basic set of neuron layers, which include an input layer, one or more hidden layers and an output layer. The main distinguishing factor between the different types of ANN architectures is how the data is passed from the input to the output layer. One of the most common types of ANN

architectures, called the feed-forward neural network (FNN), is illustrated in figure 3.2(b). FNNs transfer information from the input to the output layers in a strictly forward only direction [1].



(a) Artificial neuron

(b) FNN architecture

Figure 3.2: Artificial neural networks

The convolutional neural network (CNN) [15, 33] is a type of FNN that is typically used for computer vision tasks such as image classification, object detection, and segmentation. With large input arrays, such as images, traditional ANNs tend to suffer from computational complexity issues, whereas the unique architecture of the CNN allows it to solve the same problems with significantly fewer model parameters [26]. This contributes to one of the main advantages that CNN models have above traditional ANNs which is the ability to extract key features directly from input images without the need for manual feature extraction.

The typical CNN architecture consists of three types of hidden layers, namely a convolutional, pooling, and fully-connected layer. It is usually composed of multiple blocks of convolutional and pooling layers, followed by one or more fully-connected layers as seen in figure 3.3. The convolutional layer is the core building block of a CNN and is responsible for extraction of the relevant features from the input which also helps to reduce the complexity of the model. Extraction is performed by applying a series of filters/kernels to the input, as illustrated in figure 3.4. Each filter slides across the input volume and performs a convolution operation, which involves computing the dot product between the filter and the input values of a local region at any given position. Results are then consolidated in a feature map, which is a lower dimensional summary

of the input array [48].

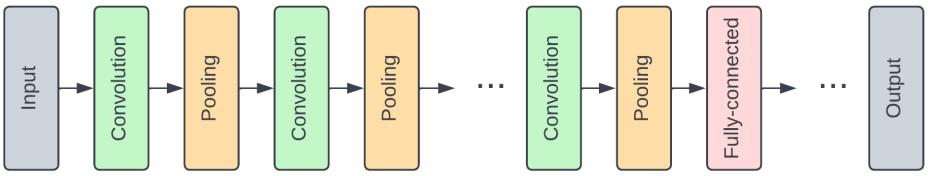


Figure 3.3: CNN architecture

The complexity of a convolutional layer and the size of the feature map is determined by three hyper-parameters, namely the depth, the stride and the setting of the zero-padding. The depth refers to the number of filters used in the convolution operation, while the stride indicates the number of pixels with which the filter is moved for each convolution operation. Larger depth values will increase the number of features extracted from the input at the cost of computational complexity. Larger stride values will reduce the size of the feature map as well as the computational complexity. Zero-padding involves padding the border of the input array to ensure that the relevant features on the edges of the input array are also included in the feature map [30].

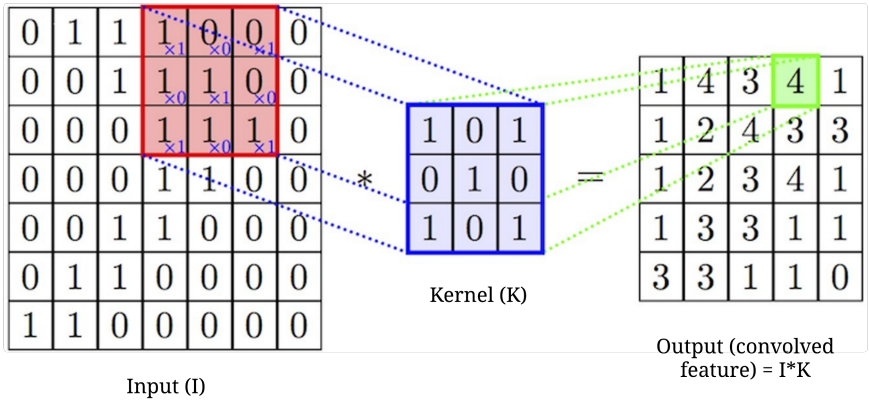


Figure 3.4: Convolutional layer [48]

In order to reduce the non-linearity in CNN models, element-wise activation functions are applied to the generated feature map. The rectified linear unit (ReLU) activation function is typically used for the hidden layers of CNN models, although the hyperbolic tangent and sigmoid functions can also be used [30]. Section 3.2.2 discusses the

different activation functions.

The purpose of the pooling layer is to reduce the dimensionality of the input feature map while preserving the most important information. Pooling involves dividing the feature map into non-overlapping regions and then performing a mathematical operation, such as *max* or *mean*, on each region to produce a single result. The results are then consolidated to form a down-sampled version of the input feature map [48]. Figure 3.5 is an example of a pooling layer that uses a *max* operation.

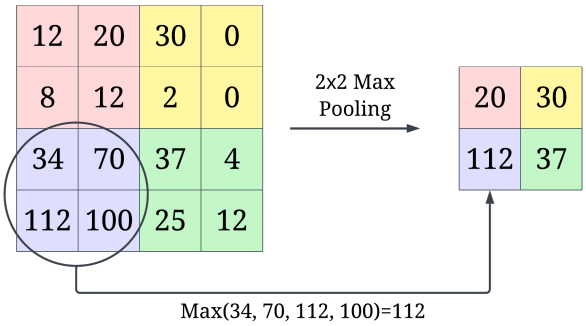


Figure 3.5: Pooling layer

The main benefits of using pooling layers are that they [26, 30]

- reduce the number of model parameters and computations, which improves the computational efficiency of the model and helps prevent model overfitting;
- improve the robustness of the network to small variations in the input data; and
- help to produce an almost scale invariant version of the input image, which assists object detection where the position of the object could vary in the input image.

The last segment of the CNN architecture consists of fully-connected layers which are responsible for mapping the feature vectors generated by the convolutional and pooling layers to the final output classes. Each neuron in a fully-connected layer is connected to every neuron in the previous and next layer, which allows the CNN to learn the non-linear combinations of the extracted features [30].

3.2.1 Training Process

Training of a CNN model can be described as the iterative adjustment of the parameters of the model in order to minimise a specific loss function. Gradient descent is an optimisation algorithm commonly used for training machine learning and deep learning models. Implementation of gradient descent learning with CNN models requires a cyclic process of performing forward-propagation, loss computing, and back-propagation [34, 57].

Forward-propagation involves computation and storing of the intermediate and output variables of a neural network for a given input image k [57]. A loss function is then used to calculate the loss value C_k that quantifies the dissimilarity between the predicted output values of the model and the ground truth values [34]. Back-propagation determines what measure of responsibility should be assigned to each of the model parameters (weights and bias) with regards to the loss value C_k [34]. This is also referred to as calculating the gradient of loss function with respect to network parameters [57]. To summarise the calculus in the forward and back-propagation steps, a small example network is provided in the figure 3.6. This summary is based on the work of Nielsen [33]. The figure represents the last two layers of a network with three nodes denoted by $n_j^{(L-1)}$ that feed into two output nodes denoted by $n_i^{(L)}$, where L refers to the network layer.

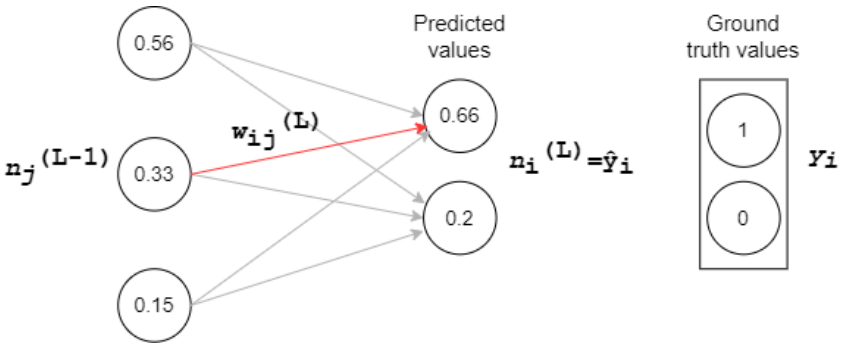


Figure 3.6: Example network

The output node values, $s_i^{(L)}$, are calculated using the following equations:

$$s_i^{(L)} = b_i^{(L)} + \sum_{j=0}^{N^{(L-1)}-1} w_{ij}^{(L)} n_j^{(L-1)} \tag{3.1}$$

$$n_i^{(L)} = \sigma(s_i^{(L)}) \tag{3.2}$$

where N is the number of nodes in a layer, w the weight parameter, b the bias parameter and σ the activation function. The loss for the input sample k is then calculated as follows:

$$C_k = \sum_{i=0}^{N^{(L)}-1} \alpha(n_i^{(L)}, y_i) \tag{3.3}$$

where α is the chosen loss function and y_i the ground truth values. After the loss is computed, back-propagation is used to determine the relationship between the function C_k and the model weights using the *chain rule* expression given below:

$$\frac{\partial C_k}{\partial w_{ij}^{(L)}} = \frac{\partial s_i^{(L)}}{\partial w_{ij}^{(L)}} \frac{\partial n_i^{(L)}}{\partial s_i^{(L)}} \frac{\partial C_k}{\partial n_i^{(L)}} \tag{3.4}$$

To calculate the gradient of loss function with respect to the bias, the *chain rule* expression is updated as follows:

$$\frac{\partial C_k}{\partial b_i^{(L)}} = \frac{\partial s_i^{(L)}}{\partial b_i^{(L)}} \frac{\partial n_i^{(L)}}{\partial s_i^{(L)}} \frac{\partial C_k}{\partial n_i^{(L)}} \tag{3.5}$$

Each fraction in equation (3.4) and (3.5) can be described as a ratio that indicates the sensitivity that the numerator variable has towards small changes in the denominator variable. For example, the $\frac{\partial C_k}{\partial n_i^{(L)}}$ fraction is a ratio of how responsive C_k is towards small changes in the node $n_i^{(L)}$. In this case, $\frac{\partial C_k}{\partial n_i^{(L)}}$ is simply the partial derivative of the loss function with respect to node $n_i^{(L)}$. For model parameters in the preceding layers, back propagation is performed using equations (3.4) and (3.5) with the respective layer variables, but there is a change in how the $\frac{\partial C_k}{\partial n}$ ratio is determined. For example, $\frac{\partial C_k}{\partial n_j^{(L-1)}}$ is calculated as follows:

$$\frac{\partial C_k}{\partial n_j^{(L-1)}} = \sum_{i=0}^{N^{(L)}-1} \frac{\partial s_i^{(L)}}{\partial n_j^{(L-1)}} \frac{\partial n_i^{(L)}}{\partial s_i^{(L)}} \frac{\partial C_k}{\partial n_i^{(L)}} \tag{3.6}$$

Therefore, further back-propagation requires more ratios in order to create a *path* between the selected weight/bias parameter and the calculated output loss.

Lastly, the loss gradient values ($\frac{\partial C}{\partial w}$, $\frac{\partial C}{\partial b}$) computed using back-propagation are consolidated to create a loss gradient vector $\nabla \mathbf{C}(\mathbf{p})$, where $\mathbf{p}$ is a vector containing the weights and biases of the model. The loss gradient vector $\nabla \mathbf{C}(\mathbf{p})$ is used to perform gradient descent learning, which is the process of determining a local minimum point of the surface that represents the output loss as a function of the weights and biases [34]. This is done by computing the gradient of each point in the surface, i.e. $\nabla \mathbf{C}(\mathbf{p})$, and then decreasing parameters that have positive loss gradients and increasing parameters that have negative loss gradients. These parameter adjustments are proportional to the scale of the loss gradient values and a selected learning rate [34], as indicated in the equation below:

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \eta \nabla \mathbf{C}(\mathbf{p}_t) \tag{3.7}$$

where vector $\mathbf{p}_{t+1}$ is the updated model parameters, vector $\mathbf{p}_t$ is the old model parameters, and η is the learning rate.

It is important to note that there are three different types of gradient descent learning algorithms, namely batch gradient descent (BGD), stochastic gradient descent (SGD) and mini-batch gradient descent (MBGD) [26, 34]. For BGD the overall loss is calculated across the entire training set before calculating the loss gradients and adjusting the model parameters. BGD usually converges to a more accurate solution compared to SGD and MBGD given that the loss gradients represent the entire training set. However, BGD can have slow convergence and be memory-intensive with large datasets, because the algorithm usually requires the entire training set to be loaded into memory [26]. SGD calculates the loss gradients and updates the model parameters for every training sample, which allows for faster convergence [26, 34]. The drawback of the SGD approach is the increased variance in the model parameter adjustments, which leads to oscillations in the loss function output. MBGD is a trade-off between BGD and SGD. It reduces the oscillation in the loss by calculating the loss over a small batch of training samples before computing the loss gradients and updating the model parameters.

To address the issue of loss oscillations encountered in MBGD and SGD, a momentum term is usually added to the parameter update function. The root mean squared propagation (RMSprop) [32] and adaptive moment estimation (Adam) [23] training algorithms are also typically used in combination with SGD or MBGD to reduce the oscillation in the loss and to accelerate model training [26].

The momentum term optimises SGD by accelerating the gradient descent in the direction of a minimum point and reducing oscillations. This is done by adding a fraction of the previous update vector to the current update vector using [15]

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{v}_t \tag{3.8}$$

with

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} - \eta \nabla \mathbf{C}(\mathbf{p}_t) \tag{3.9}$$

where β is the momentum coefficient and $\mathbf{v}_t$ the parameter update vector. The $\beta \mathbf{v}_{t-1}$ momentum term increases when consecutive loss gradients are in the same direction and decreases when the loss gradients are in opposite directions. Similar to a ball rolling down a slope and gaining momentum, the $\beta \mathbf{v}_{t-1}$ term allows SGD learning to gain momentum.

The RMSprop and Adam training algorithms make use of adaptive learning rates to enhance gradient descent [15]. The intuition behind the use of adaptive learning rates is that the loss gradients of a multi-layer network can differ significantly for different layers in the network. Thus, a single learning rate cannot adequately update all the weights in a multi-layer network. The RMSprop [15, 32] algorithm was created for MBGD and works by multiplying the global learning rate with a local gain that is computed for each weight as follows

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \nabla \mathbf{C}(\mathbf{p}_t) \tag{3.10}$$

with

$$\mathbf{s}_t = \rho \mathbf{s}_{t-1} + (1 - \rho)(\nabla \mathbf{C}(\mathbf{p}_t) \odot \nabla \mathbf{C}(\mathbf{p}_t)) \tag{3.11}$$

where $\mathbf{s}_t$ is a vector containing the moving average of the squared loss gradient for each weight, ρ is the decay rate coefficient, and ϵ is a constant added for stability. Note that

$\odot$ symbolises the Hadamard product (element-wise multiplication). The $\frac{1}{\sqrt{\mathbf{s}_t+\epsilon}}$ term represents the local gains of each weight, which is multiplied with the global learning rate, η.

Adam [15, 23] is an optimisation algorithm that can be described as a combination of both momentum and RMSProp. The method utilises the moving average of both the past gradients and the past squared gradients as follows

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon} \tag{3.12}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - (\rho_1)^t} \tag{3.13}$$

$$\hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - (\rho_2)^t} \tag{3.14}$$

$$\mathbf{v}_t = \rho_1 \mathbf{v}_{t-1} + (1 - \rho_1)\nabla \mathbf{C}(\mathbf{p}_t) \tag{3.15}$$

$$\mathbf{s}_t = \rho_2 \mathbf{s}_{t-1} + (1 - \rho_2)(\nabla \mathbf{C}(\mathbf{p}_t) \odot \nabla \mathbf{C}(\mathbf{p}_t)) \tag{3.16}$$

where $\mathbf{v}_t$ and $\mathbf{s}_t$ are the first and second moment of gradients, respectively. The ρ_1 and ρ_2 coefficients determine the decay rate of the moving averages. Kingma and Ba [23] noted that the method tends to be biased towards zero for initial time steps. To counter the bias, equations (3.13) and (3.14) were added to the optimisation. The model parameters are then updated using equation (3.12) and the bias-corrected averages, $\hat{\mathbf{v}}_t$ and $\hat{\mathbf{s}}_t$. Note that operations are applied element-wise in equation (3.12).

3.2.2 Activation Functions

The activation functions used in the hidden layers of a network help to attain non-linear mappings between the input and output of the model [36]; whereas the activation functions in the output layer determine the type of predictions the model can make.

Activation functions that are frequently used in CNN models include the sigmoid, hyperbolic tangent (tanh), softmax, ReLU and leaky ReLU functions [26]. The sigmoid, hyperbolic tangent, and softmax functions are logistic class transforms with S-shaped curves. Illustrated in figure 3.7, the sigmoid function maps any real value x to the range of (0, 1), while the hyperbolic tangent function maps input values to the range of (-1, 1). The softmax function utilises a collection of sigmoid functions, which allows it to

take a vector of real values as input and output a vector of probabilities [36]. The use of logistic class transforms for hidden layer nodes is not recommended due to vanishing gradients [36] that can slow down the model learning process. Vanishing gradients occur when the gradients of the loss function, with respect to the weights and biases of the network, become extremely small as they are back-propagated through the network layers during training. This is due to the derivative of the activation function being small. Therefore, the hyperbolic tangent and sigmoid functions are typically used in the output layer of binary classification models and the softmax function is generally used in the output layer of multi-classification models [26].

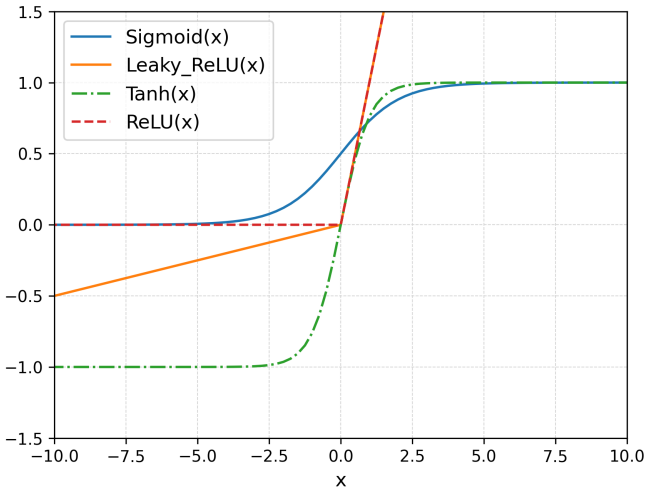


Figure 3.7: Activation Functions

The ReLU and leaky ReLU functions were created to counter vanishing gradients and are therefore commonly used for the hidden layer nodes of deep neural networks [26]. ReLU is a non-linear function that outputs zero when the input x is negative and $f(x) = x$ for inputs that are positive. Aside from preventing vanishing gradients, the ReLU function is also more computationally efficient compared to the sigmoid and hyperbolic tangent functions [26]. The main downfall to the ReLU function is that some of the model weight updates could force certain nodes to become permanently fixed at zero [2]. This phenomenon is referred to as *dying ReLU* and to counter it, the leaky ReLU function is proposed [2]. Leaky ReLU is a variation of the ReLU function that

introduces a small slope for inputs below zero, rather than limiting negative inputs to zero. This slope is defined as a very small linear component of x, as illustrated in figure 3.7.

3.2.3 Training Algorithm Hyper-parameters

The hyper-parameters of a training algorithm are parameters that are set prior to model training and impact the behavior and performance of the training algorithm. When selecting hyper-parameters, the objective is to enable the model to efficiently capture the structure of the data without overfitting or underfitting the data [34].

The loss gradients computed in back-propagation are multiplied with the learning rate before updates are performed on the weights and biases of the model. Thus, the learning rate is a hyper-parameter that affects the step-size of the model updates. Models with small learning rates suffer from slow training speeds and are also likely to become stuck in local minima instead of the loss minimum point. Models with high learning rates are quicker to train, but stand the risk of overshooting the loss minimum point [34]. A popular approach involves starting with a high learning rate and gradually reducing the rate while training the model [26]. This helps to accelerate training and to mitigate the risk of overshooting the optimal solution.

The number of epochs is a hyper-parameter that defines the number of times the entire training data-set is passed to the model [26]. The model performance on the training and validation data can help to determine whether the number of epochs should be increased or decreased [26]. Poor performance on training data can be due to the number of epochs being too low. Conversely, a possible indication of too many epochs is when the training loss continues to decrease and the validation loss starts to increase. A possible strategy for determining the optimal number of epochs is to use a large number of epochs and then stop model training when a specific condition is met, such as stopping when the validation loss starts to increase [26].

For MBGD, the mini-batch size hyper-parameter refers to the number of samples used for each model update. Small batch sizes are less representative of the full data-set and therefore lead to increased variance in the loss. On the other hand, larger batch

sizes require more memory resources and can lead to slower convergence [26]. Optimal batch sizes are directly proportional to the learning rate and training set size [16] and the recommended guideline for selecting batch sizes is to choose sizes that are powers of 2 (e.g. 8, 16, 32 and 64) for efficient CPU/GPU utilisation [26].

Dropout [46] is a computationally inexpensive, but effective technique used to mitigate overfitting in a neural network. The technique involves temporarily deactivating random neurons in the input or hidden layers of the neural network, ensuring that these deactivated neurons do not participate in the forward and back-propagation during a particular training iteration. Dropout helps to prevent the formation of complex co-adaptations between nodes and instead forces each node to learn to identify features that contribute to the network producing accurate outputs. The dropout technique is controlled using the dropout probability hyper-parameter, which is specified for each layer of the network and denotes the probability of a neuron being deactivated within that layer.

3.2.4 Loss Functions

Loss functions quantify the model performance and, in combination with all the parameter configurations, helps define the search space of the problem. There are a multitude of different loss functions that can be selected when designing a CNN model, with some being better suited for regression problems and others complementing classification models. The most popular used loss functions include the mean squared error (MSE), mean absolute error (MAE) and cross entropy loss functions [26].

The MSE and MAE loss functions are used for regression problems, which are problems that require a model to predict a continuous numerical value. An example of a regression model is a model that predicts the coordinates for an object bounding box. The equations used to compute MSE and MAE loss are as follows [34]:

$$C_{MSE} = \frac{1}{2M} \sum_{i=1}^{M} \sum_{j=1}^{N} (\hat{y}_{ij} - y_{ij})^2 \tag{3.17}$$

$$C_{MAE} = \frac{1}{2M} \sum_{i=1}^{M} \sum_{j=1}^{N} |\hat{y}_{ij} - y_{ij}| \tag{3.18}$$

where M is the number of samples in the batch, N the number of output nodes, and y_{ij} the ground truth values. Only the output nodes are considered when calculating the loss. Therefore, the equations are simplified by dropping the L notation and using $\hat{y}_{ij}$ for the predicted output values. For data-sets that contain outliers the MAE loss function is more robust compared to the MSE function. This is because the MSE function squares the outlier error, which results in the model assigning greater importance to outlier errors. However, the MAE loss function is a non-differentiable function making it more difficult to calculate loss gradients [26].

There are two types of cross-entropy loss functions, namely binary and categorical cross-entropy loss. These loss functions are typically used for classification problems, which are problems that require a model to predict a discrete or categorical value. According to Nielsen [33], cross-entropy loss functions are well-liked, because they are less susceptible to the slow learning, unlike quadratic loss functions. Slow learning can be attributed to the derivative of an activation function being small for certain input values. For example, the derivative of the sigmoid function tends to zero as the input values become extremely large or small. When the derivative of the activation function is small, the $\frac{\partial n_i^{(L)}}{\partial s_i^{(L)}}$ ratio from equation (3.4) is also small, which results in smaller loss gradients and slow model learning. To counter this, the partial derivative of the cross-entropy loss function cancels out the $\frac{\partial n_i^{(L)}}{\partial s_i^{(L)}}$ ratio in the *chainrule* equation (3.4). The cross entropy loss function is described by the following equation:

$$C_{CE} = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N}[y_{ij}\ln(\hat{y}_{ij}) + (1-y_{ij})\ln(1-\hat{y}_{ij})] \tag{3.19}$$

The difference between binary and categorical cross entropy loss functions is the activation functions used to compute the $\hat{y}_{ij}$ output probabilities. Binary cross entropy loss functions are used in combination with the sigmoid activation function, whereas, categorical cross entropy loss functions are used with the softmax activation function [**?**].

3.3 Dataset Preparation

Prior to the training and testing of a machine learning model, raw input data needs to be transformed to meet the input requirements of the model. Furthermore, appropriate

data preparation steps can assist a model in learning the underlying patterns in the data and can lead to a more robust model. Typical data preparation steps for computer vision models include image normalisation, image augmentation, and data partitioning.

Image normalisation is the process of adjusting the range of the pixel intensity values within an image. Normalisation reduces the difference in magnitude between the different features, thereby mitigating the risk of features with large magnitudes dominating the learning process [45]. The min-max normalisation (MMN) is a popular normalisation technique where each pixel is normalised as follows [45]

$$x_i' = \frac{x_i - min(x)}{max(x) - min(x)}(max_{new} - min_{new}) + min_{new} \tag{3.20}$$

where x_i' is the normalised pixel value, x_i is the original pixel value, and $min(x)$ and $max(x)$ is the minimum and maximum pixel intensities for image x. The max_{new} and min_{new} scalars are used as upper and lower bounds for the normalised output. The input data for a neural network is typically scaled to the range of [0,1] to correspond to the ranges of the activation functions. This is because when input values with large magnitudes are fed to an activation function, like the Sigmoid or ReLU function, the output will either be zero or very large, which can result in loss gradients vanishing or exploding [11].

Training of deep neural networks requires a large quantity of labeled data to achieve satisfactory results and to avoid overfitting. However, it is often difficult to attain sufficient data and thus image augmentation is utilised. Image augmentation is implemented after image preprocessing and can be described as generating new training samples from existing samples [43]. The new samples are created by performing image transformations, such as flipping, scaling, rotating and color adjusting, on existing samples. Image augmentation can either be offline or online [43]. Offline augmentation is done prior to model training and the generated samples are stored, whereas online augmentation is done dynamically during training and does not require generated samples to be stored. Offline augmentation is more computationally efficient compared to online augmentation, but does require more storage space.

In order to evaluate how well a model generalises to unseen data it is important to

apply data partitioning. The hold-out method is the standard method used to evaluate the generalisability of a model and involves splitting the dataset into non-overlapping subsets, namely a training set and a test set [15]. The training set typically represents a large portion (e.g. 80%) of the data and the remaining portion is used for the test set. In addition, it is recommended to utilise a validation set when performing hyper-parameter tuning [15]. A validation set is a small subset that is extracted from the training set to guide hyper-parameter adjustments. By utilising a validation set, it is possible to assess how well the model generalises to unseen test set data after hyper-parameter optimisation.

Other popular methods used to assess the generalisability of a model include cross-validation and bootstrapping [15, 34]. Cross-validation repeats model training and testing for different randomly chosen subsets. The most common version of cross-validation is k-fold cross-validation, which involves splitting the dataset into k non-overlapping equally-sized subsets, called *folds*. Model training and testing is then repeated k times, where each iteration uses a different fold as the test set and the remaining $k-1$ folds as the training set. The model is then evaluated on average of the k performance scores. Cross-validation is a good alternative to the hold-out method in cases were data is limited and partitioning of data will result in a very small test set.

Bootstrapping is a statistical resampling technique that involves generating J samples in order the obtain J model performance estimates [47]. Each sample is collected using sampling with replacement from the original dataset, with every bootstrap sample containing the same number of data points as the original dataset. The estimates calculated for each sample can be used to produce a performance distribution of the model. Thus, bootstrapping is an effective method for evaluating the variability and robustness of a model, particularly when data is limited.

3.4 Transfer Learning and Fine Tuning

Transfer learning [5, 43] is a popular technique that is used to efficiently train deep learning models with less data. It initiates model training by taking the knowledge gained from solving one problem and applying it to a new, similar problem. This

process involves using the weights of a network that has been pre-trained on a large dataset to initialise a new network. The final layers of the initialised network are then trained on a new custom dataset, with the other layers frozen to avoid information loss. Networks used in computer vision applications are typically pretrained on large visual datasets such as ImageNet [7] or Common Objects in Context (COCO) [27].

Fine tuning is similar to transfer learning, with the difference being that it involves retraining the entire or large portions of the initialised network, rather than just the final layers [5]. Transfer learning and fine tuning are often used in conjunction to further refine the network and enhance performance.

3.5 Model Evaluation metrics

Evaluation metrics are utilised to quantify the quality of the trained model with respect to certain criteria. The majority of the evaluation metrics are computed using the true positive (TP), true negative (TN), false positive (FP) and false negative (FN) measures. Negative and positive samples that are correctly classified are indicated by the TN and TP measures, while the misclassified negative and positive samples are specified by the FN and FP measures [2]. Some of the commonly used evaluation metrics are listed below [2, 34, 35]:

- Accuracy is the most used metric and shows the fraction of correctly classified samples out of the total number of samples evaluated, calculated as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.21}$$

- The sensitivity or recall metric measures the fraction of positive samples that are correctly classified and can also be described as how well the model avoids predicting false negatives, calculated as

$$Sensitivity/Recall = \frac{TP}{TP + FN} \tag{3.22}$$

- Specificity measures the fraction of negative samples that are correctly classified. Thus, it is also an indication of how well the model avoids predicting false positives. Specificity is calculated as

$$Specificity = \frac{TN}{TN + FP} \tag{3.23}$$

- Precision measures the fraction of positive samples correctly classified out of all the positive predictions. The difference between the precision and sensitivity is that precision measures how many positive predictions are correct, while sensitivity measures the coverage of the true positive samples. Precision is calculated as

$$Precision = \frac{TP}{TP + FP} \tag{3.24}$$

- The F1-score is the harmonic mean between the sensitivity and precision scores and is meant to give an overall performance score, calculated as

$$F1score = 2 \times \frac{\text{Sensitivity} \times \text{Precision}}{\text{Sensitivity} + \text{Precision}} \tag{3.25}$$

- The area under the receiver operating characteristic curve (AUC) [21] is a metric that is commonly used for binary classification models and measures the overall performance of a model. The AUC metric represents the area under the ROC curve, where a score of 1 indicates optimal classification performance. The receiver operating characteristic curve (ROC) plots the sensitivity versus false positive rate (1-specificity) of a model for different classification thresholds applied to the output. If the AUC score is 0.5 or less, then the model has a classification performance that is no better than random guessing.

Multi-class classification models are more difficult to evaluate using the metrics above compared to binary classification models. This is because the TP, TN, FP and FN measures are suited to binary problems. To address this, multi-class problems are divided into a series of binary problems. For each binary problem, a single class represents the positive samples, while the remaining classes represent the negative samples. The metric calculated for each class is then averaged to determine the overall performance [19]. For imbalanced mutli-class datasets, the score of each class can be weighted according to class support. This allows the metric average to be more representative of the entire dataset and not biased towards the more dominant class. Note that accuracy can also be calculated for multi-class problems by dividing the total number of correctly classified samples by the total number of samples evaluated.

For the evaluation of object detection models, the intersection of union (IoU) is the defacto evaluation metric [39]. IoU measures the similarity of two arbitrary shapes by

dividing the overlapping area of the shapes with the area of union, as illustrated in figure 3.8. By incorporating a threshold value, the IoU can also be used to determine the TP and FP measures of object detection results [39]. For example, if the $IoU > 0.5$ the result is regarded a TP, otherwise it is a FP. This thresholding technique allows other performance metrics, such as precision, to be used when evaluating object detection models.

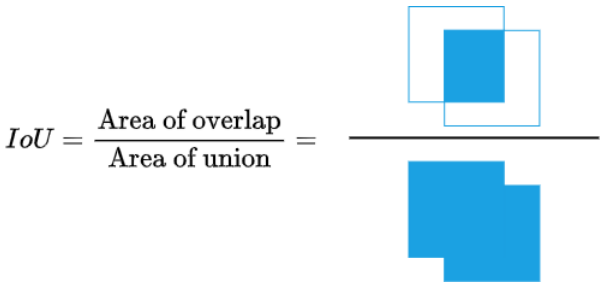


Figure 3.8: Intersection of union

In addition to IoU, the average precision (AP) and mean average precision (mAP) are metrics that are typically used to evaluate object detection models [18]. The AP metric is a single value that summarises the precision-recall (PR) curve. Similar to the ROC curve, the PR curve plots the precision versus sensitivity/recall for different thresholds. However, in the context of object detection, IoU thresholds are applied. Using the values of the PR curve the AP metric is calculated as follows [18]:

$$AP = \sum_{k=0}^{M-1} (Recall_k - Recall_{k+1}) * Precision_k \tag{3.26}$$

where M is the number of thresholds, $Recall_k$ the recall score at threshold k, and $Precision_k$ the precision score at threshold k. Thus, AP is the summation of the precision scores at each threshold, with each precision score weighted by the corresponding increase in the recall score.

Calculation of the mAP involves computation of the AP score for each class in the dataset and then calculation of the mean of the AP scores [18]. The equation for mAP is as follows:

$$mAP = \frac{1}{R} \sum_{j=0}^{R-1} AP_j \tag{3.27}$$

where R is the number of classes in the dataset and AP_j the AP score for class j.

3.6 Summary

Chapter 3 describes the fundamentals of machine learning and provides a comprehensive overview of the CNN architecture as well as the concepts related to training neural networks. In addition, the chapter explores the different data preparation steps utilised for training computer vision models and techniques used to efficiently train models with less data, namely transfer learning and fine tuning. Lastly, the different evaluation metrics used for classification and object detection are reviewed.

Chapter 4

Deep Convolutional Neural Network Architectures

This chapter provides an overview of the different deep learning architectures that are discussed in chapter 5. The chapter comprises of two main sections, namely image classification architectures and object detection architectures.

4.1 Image classification architectures

Image classification models are used to predict whether a specific object is present in an image. This section discusses common image classification architectures. The architectures reviewed include the Residual network, Inception-v3 network, densely connected CNN, and EfficientNet network.

4.1.1 Residual Network

The Residual network (ResNet), introduced by He *et al.* [17] in 2015, is a feed-forward neural network that uses skip connections to mitigate the vanishing gradient problem experienced when training very deep neural networks. Skip connections, also referred to as residual connections, add alternative paths for data to reach the latter layers of the network by bypassing some of the network layers. The ResNet architecture has a modular design that is built by stacking multiple *residual building blocks*. Each residual block typically consists of two 3×3 convolutional layers, ReLU activations, and a skip connection, as illustrated in figure 4.1.

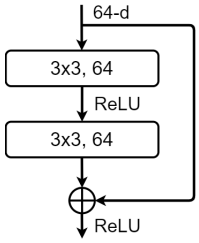


Figure 4.1: Residual building block [17]

Element-wise addition is used to merge the skip connection with the layer output, as shown in figure 4.1. In order to perform element-wise addition, the skip connection and layer output need to have the same dimensions. Residual blocks with matching input and output dimensions utilise identity skip connections, which perform identity transformations and do not introduce additional parameters to the model. When the input dimensions of a residual building block are smaller than the output dimensions, the following two skip connection options are considered: Either use identity skip connections with zero padding to increase the dimensions or use projection skip connections that utilise 1×1 convolution to match the dimensions. It is important to note that projection shortcuts introduce extra parameters and computational complexity to the model.

The experiments done in the original paper [17] compared the ResNet architecture to the simpler visual geometry group (VGG) architecture [44], and the results showed that the ResNet architecture is easier to optimise and can achieve higher accuracy with deeper networks. ResNet variants differ according to network depth, with the 18-layer (ResNet18) and 50-layer (ResNet50) variants being utilised in the applications discussed in chapter 5.

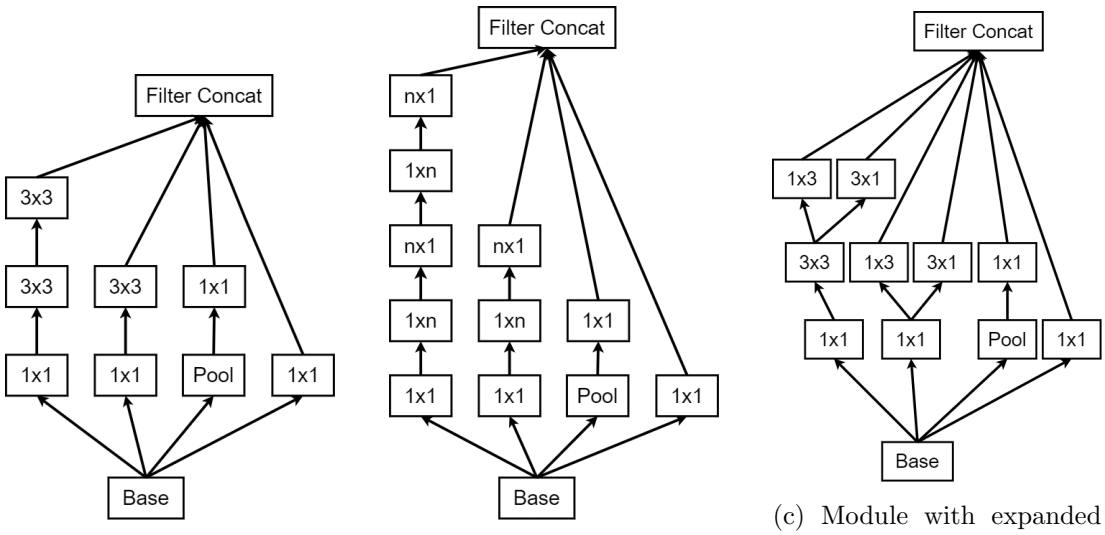
4.1.2 Inception-v3

Inception networks utilise parallel convolution filters with different sizes to enable the network to capture information at multiple scales. The Inception-v3 was presented by Szegedy *et al.* [50] in 2016 as an improved version of the GoogLeNet (Inception-v1) [49] architecture. The paper proposes the use of factorised convolutions to further reduce

the computational complexity of the inception architecture and to allow the network to be efficiently scaled-up.

Convolution factorisation involves replacing convolutions that have large spatial filters with two or more layers of smaller filter convolutions. Additional optimisations presented in the paper include the use of auxiliary classifiers and efficient grid-size reductions. Auxiliary classifiers are output classifiers that are added to certain intermediate layers of the network to help push valuable gradients directly to lower layers of the network and to enhance the convergence of deep networks.

The core of the Inception-v3 architecture utilises three types of inception modules, which are illustrated in figure 4.2. The module layout in the network can be described as three conventional modules (illustrated in fig 4.2(a)) that feed into five factorised modules (illustrated in fig 4.2(b)), which are then followed by two modules with expanded filter bank outputs (illustrated in fig 4.2(c)).



(a) Conventional module (b) Factorised module (c) Module with expanded filter bank output

Figure 4.2: Inception modules [50]

The computational cost of the Inception-v3 network, which has 42 layers, is only 2.5 times higher than the GoogLeNet network, which has 22 layers. The increase in layers allows the Inception-v3 network to achieve significantly better performance scores

compared to the VGG network and earlier inception architectures.

4.1.3 Densely Connected Convolutional Neural Network

Huang *et al.* [20] present the idea of a densely connected convolutional network (DenseNet), which is a network where each layer is connected to every other layer in a feed-forward manner. Similar to the ResNet architecture, DenseNet utilises skip connections to pass the feature maps from each layer to their subsequent layers. The feature maps from the different layers are then combined using concatenation, unlike ResNet that implements element-wise summation.

The core of the DenseNet architecture is constructed by stacking multiple *dense blocks* with *transition layers* in between. Each dense block consists of multiple layers with each layer being a composite function of three consecutive operations, namely batch normalisation, ReLU activation, and 3×3 convolution. The input of each layer in a dense block is comprised of the concatenated feature maps of the previous layers in the block. In order to perform concatenation, the feature maps must be of the same size and therefore down-sampling occurs within the transition layers. Transition layers typically consist of a batch normalisation layer, a 1×1 convolutional layer, and a 2×2 average pooling layer with a stride of 2.

The DenseNet architecture is parameter efficient and also enhances the flow of information and gradients through the network, which makes the model easier to train. Additionally, the dense connections of the architecture aid regularisation, which reduces the risk of the model overfitting. Experiments from the study [20] show that the DenseNets are capable of matching the performance of ResNets, while requiring significantly less parameters.

4.1.4 EfficientNet

The EfficientNet models, proposed by Tan and Le [51] in 2019, were designed to achieve an optimal trade-off between accuracy and model complexity by utilising a compound model scaling method. The scaling method is used to adjust model depth, width and resolution parameters using a single compound coefficient. With this approach the network can be scaled in a balanced manner to fit the specific requirements of the problem.

To demonstrate the efficacy of the scaling method, a baseline network was designed using a multi-objective neural network search that optimises both accuracy and computational cost. Table 4.1 defines the different blocks of the resulting baseline architecture with $i = 1$ describing the first section of the architecture and $i = 9$ the final section. The input resolution for each block is denoted with r_i, while c_i refers to the number of output channels/feature maps. Furthermore, each block consists of l_i layers with each layer performing the operation describe by ϕ_i. The *MBConv* operation in table 4.1 refers to the mobile inverted bottleneck convolution proposed by Sandler *et al.* [42].

Table 4.1: EfficientNet baseline model B0 [51]

Block i	**Operation** ϕ_i	**Resolution** r_i	**#Channels** c_i	**#Layers** l_i
1	Conv 3x3	224x224	32	1
2	MBConv1, k3x3	112x112	16	1
3	MBConv6, k3x3	112x112	24	2
4	MBConv6, k5x5	56x56	40	2
5	MBConv6, k3x3	28x28	80	3
6	MBConv6, k5x5	28x28	112	3
7	MBConv6, k5x5	14x14	192	4
8	MBConv6, k3x3	7x7	320	1
9	Conv 1x1 & Pooling & FC	7x7	1280	1

The baseline network, EfficientNet-B0, is then scaled up using the compound coefficient to obtain the EfficientNet B1 to B7 models. Scaled up models utilise the same block operations ϕ_i described in table 4.1, but differ according to resolution, number of channels and number of layers. Results from the study showed that the EfficientNet models achieved state-of-the-art accuracy on six commonly used transfer data-sets, while requiring an order of magnitude less parameters.

4.2 Object detection architectures

Object detection is an extension of image classification that not only focuses on predicting if an object is present in an image, but also determining the location of the object in the image. This section reviews object detection architectures that have been

utilised for implant detection, namely the faster region-based convolutional neural network (faster R-CNN), the you only look once network (YOLO), and the Efficientdet network.

4.2.1 Faster Region-Based Convolutional Neural Network

The faster R-CNN detection model was proposed by Ren *et al.* [38] and is an optimised version of the older, R-CNN [13] and fast R-CNN [12], models. Region-based convolutional neural network (R-CNN) models use a two-stage approach for object detection with the first stage generating region proposals for object localisation and the second stage predicting the object type for each proposed region. The faster R-CNN model consists of two main modules, namely a region proposal network (RPN) and a fast R-CNN network.

The first step of the faster R-CNN model typically involves conversion of an input image into a single scale feature map using a backbone network. Backbone networks are feature extraction networks that use the convolutional layers of pretrained CNNs. The extracted feature map is then passed to the RPN, which is a convolutional network that outputs rectangular region proposals. Each region proposal is also assigned an objectness score, which indicates the probability that the region contains an object. The RPN architecture is typically made up of a convolutional layer with a set of fixed-sized filters called *anchors*, followed by two sibling 1×1 convolutional layers. *Anchors* serve as predefined reference boxes used to generate region proposals.

As illustrated in figure 4.3, for each sliding window location, multiple different anchor boxes are applied to the feature map and for each anchor a lower dimensional (256-d) feature is generated. The variable k defines the number of regions generated per window location and also refers to the number of predefined anchor boxes used. The anchor boxes are centred around the sliding window location and vary in scale and aspect ratio to accommodate objects of different sizes and shapes.

Features generated are then passed to the two sibling layers, namely the box-regression layer (*reg*) and the box-classification layer (*cls*). The *reg* layer outputs $4k$ coordinates representing the k box proposals, while the *cls* layer produces $2k$ scores indicating the

probability of a proposal containing an object or not. The proposals generated by the RPN are then mapped back to the input feature maps, as shown in figure 4.4.

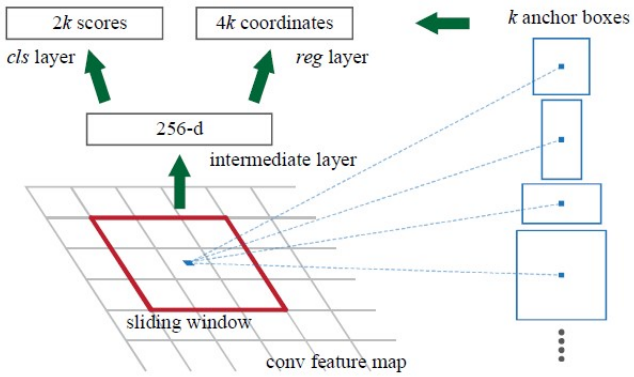


Figure 4.3: Region proposal network [38]

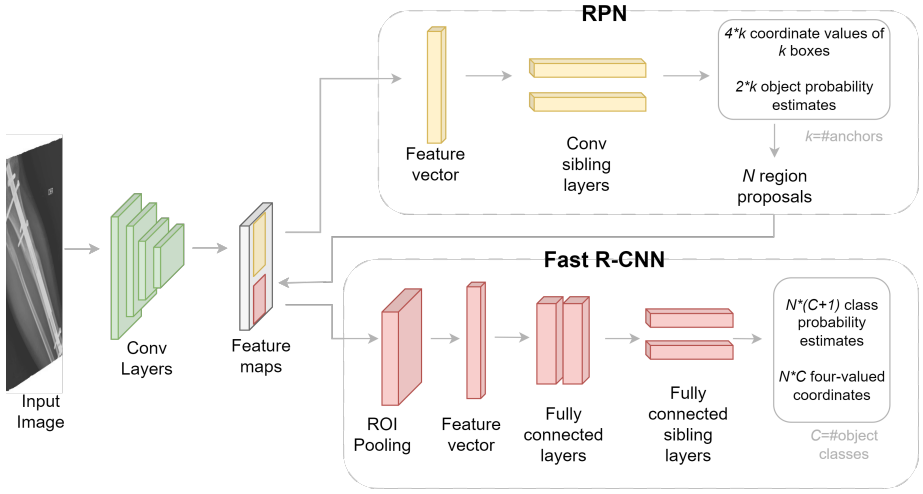


Figure 4.4: Faster region-based convolutional neural network

Stage two of the faster R-CNN uses a modified fast R-CNN network that excludes its original region proposal section and instead takes feature maps with RPN proposals as input. The fast R-CNN architecture used consists of a region of interest (RoI) pooling layer, fully connected layers, and two output sibling layers. As described in [12], the RoI pooling layer uses max pooling to extract feature vectors from the feature map for each of the RPN region proposals. These feature vectors are then passed through

a sequence of fully connected layers that eventually branch into two fully connected sibling layers, as shown in figure 4.4. The one sibling layer is responsible for estimating the softmax probabilities for each of the object classes including a background class, while the other sibling layer produces four real value coordinates for every one of the object classes.

To assist in the detection of objects of varying scale, the faster R-CNN model discussed in chapter 5 also utilises a feature pyramid network (FPN) [28]. An FPN is a feature extraction network that is used to create multi-scale feature representations that are then fed to the RPN. FPN has a top-down architecture with lateral connections, which can be used to generate multiple semantic rich feature maps of varying scale. New feature maps are generated using merging operations on feature maps extracted from the backbone network.

The purpose of the FPN is to fuse the important semantic detail from the low resolution feature maps with the higher resolution feature maps. Figure 4.5 illustrates these merging operations, which involves the conversion of the feature maps to the same sizes and then performing a series of element-wise summations from the top feature map to the bottom feature map. Scaling of the feature map sizes is done using 1×1 convolutional layers and upsampling methods. Each element-wise summation produces a new merged feature map that is then passed to the prediction network, which in this case is the RPN.

4.2.2 You Only Look Once Model

The you only look once (YOLO) model illustrated in figure 4.6 is an one-stage object detection model that was initially presented by Redmon *et al.* [37]. Unlike two-stage detection models (e.g. R-CNN), the YOLO model uses a single network consisting of 24 convolutional layers and two fully connected layers to predict the bounding boxes and class probabilities directly from the input images. The network implements a combination of 1×1 and 3×3 convolutional layers with the 1×1 filters being used to reduce the dimensionality of preceding feature maps.

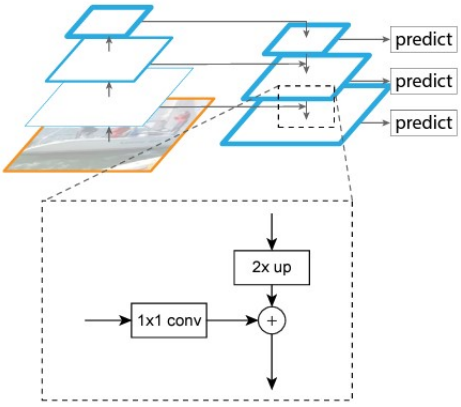


Figure 4.5: Feature pyramid network [28]

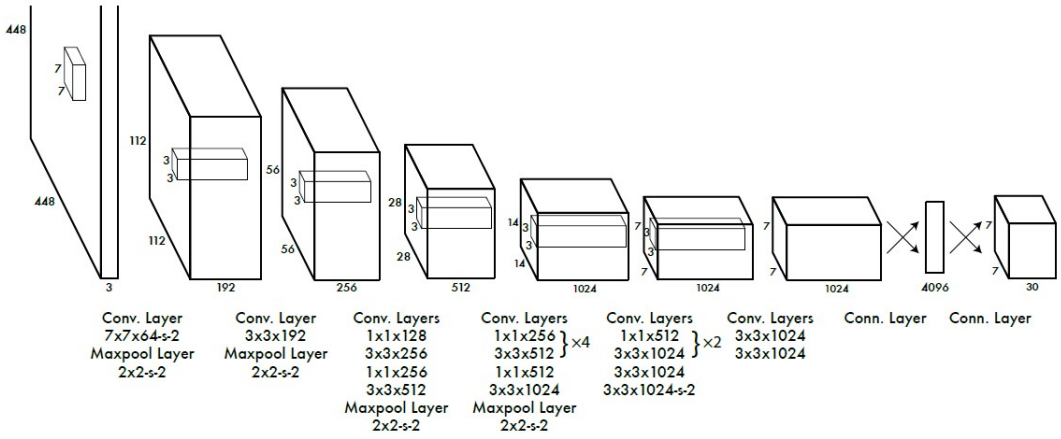


Figure 4.6: You only look once model architecture [37]

The detection process divides the input image into a $S \times S$ grid and then predicts B bounding boxes for each of the grid cells. The grid cell that contains the center of an object is responsible for detecting that object. All bounding boxes are defined by five values. Four of these values represent the (x, y, w, h) box coordinates, with the fifth value being a confidence value that indicates the probability of the box containing an object. The x and y variables mark the centre point of the bounding box, while the w and h variables represent the width and height of the bounding box. The confidence score is determined by calculating the intersection of union (IoU) between the predicted box and the ground truth. Additionally, a single set of C conditional class probabilities (conditioned on the cell containing an object) is also estimated for each

grid cell. These class probabilities are linked specifically to the cell. To determine the class specific confidence scores for each of the bounding boxes, the C conditional class probabilities are multiplied with each of the B box confidence scores. The output shape of the YOLO model can therefore be described as $S \times S \times (B * 5 + C)$.

Since the initial YOLO model was proposed, there has been many new model variations, for example the YOLOv5[1] and YOLOv8[2], that allow improved accuracy and faster calculation speeds.

4.2.3 EfficientDet

The EfficientDet model was proposed by Tan *et al.* [52] and was designed to strike a balance between accuracy and computational efficiency. EfficientDet is a one-stage object detection model that features two main optimisations used in combination with an EfficientNet backbone network, namely bi-directional feature pyramid networks (BiFPN) and compound scaling. The complete model architecture is provided in figure 4.7.

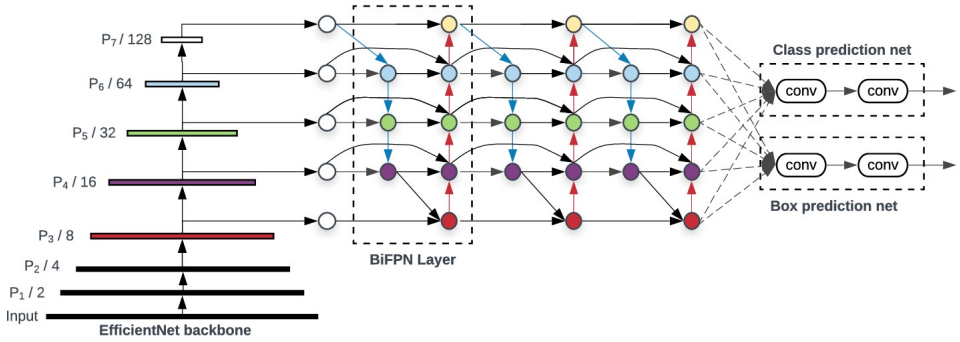


Figure 4.7: EfficientDet architecture (P_i refers to the extracted features for layer i of the backbone network) [52]

Similar to the FPN used for the faster R-CNN model, the BiFPN acts as a feature extraction network that uses feature maps from the EfficientNet backbone network as input and performs feature fusion. The difference between BiFPN and FPN is that BiFPN uses bi-directional connections to perform top-down and bottom-up feature

[1] https://github.com/ultralytics/yolov5
[2] https://github.com/ultralytics/ultralytics

merging, while FPN only performs top-down feature merging. This allows the BiFPN to capture the fine-grained details of the higher resolution feature maps as well as the semantic information of the lower resolution feature maps. As indicated in figure 4.7, EfficientDet repeats the BiFPN merging process multiple times before feeding the fused features to a box and class prediction network.

Compound scaling as noted above is the procedure of simultaneously scaling multiple model dimensions to optimally fit the requirements of the object detection problem. The EfficientDet model uses a compound coefficient Φ to jointly scale the dimensions of the input resolution, EfficientNet backbone network, BiFPN network, and class/box network. Table 4.2 shows the dimensions of eight baseline model configurations and their corresponding scaling coefficient values.

Table 4.2: Compound Scaling [52]

Model	**Input Size**	**Backbone Network**	**BiFPN #channels**	**BiFPN #layers**	**Box/Class #layers**
D0($\Phi = 0$)	512	B0	64	3	3
D1($\Phi = 1$)	640	B1	88	4	3
D2($\Phi = 2$)	768	B2	112	5	3
D3($\Phi = 3$)	896	B3	160	6	4
D4($\Phi = 4$)	1024	B4	224	7	4
D5($\Phi = 5$)	1280	B5	288	7	4
D6($\Phi = 6$)	1280	B6	384	8	5
D6($\Phi = 7$)	1536	B6	384	8	5

4.3 Summary

Chapter 4 presented the different image classification and object detection architectures utilised in the medical applications discussed in chapter 5. The image classification architectures reviewed include the ResNet, DenseNet, InceptionV3, and EfficientNet networks. A core concept used in the classification architectures involves combining data from different layers of the architecture to allow for deeper learning without vanishing gradients.

The object detection models discussed in chapter 4 include the faster R-CNN, YOLO and EfficientDet model. The object detection models reviewed can be divided into two main types, namely one-stage and two-stage. Two-stage models first generate object region proposals and then perform object classification on each region proposal, whereas one-stage models perform object classification and bounding-box regression directly without the use of pre-generated region proposals.

Chapter 5

Deep Learning Systems in Healthcare

The use of deep learning models to classify orthopedic trauma implants is a relatively unexplored topic, and to guide the deep learning model selection for this study, a review of the latest articles related to automated implant identification was done. These articles include studies done by Dutt *et al.* [9] and White *et al.* [54], both of which propose two-step approaches implementing object detection and classification of orthopaedic implants. Research done by Lee *et al.* [25] and Belete *et al.* [3] is also discussed, focusing solely on implant detection and implant classification respectively.

5.1 Localisation and classification of cervical spine hardware

Dutt *et al.* [9] presented a weakly supervised deep learning pipeline which can be used to automatically localise and classify 10 brands of cervical spine hardware from radiographs. Training, validation, and testing of the pipeline was done with a dataset containing 10589 radiographic images representing 984 patients. The images included in the dataset were restricted to one brand of hardware per image and all images were labelled according to a brand.

The object detection stage of the pipeline used the EfficientDet-D0 model proposed by Tan *et al.* [52], which takes a 512x512 radiographic image as input and outputs a

dictionary containing the general-class of each device detected, the confidence scores for each detection, and the bounding box coordinates for each of the devices. The bounding boxes were manually added to approximately 10% of the images in the dataset and labelled according to two general-classes, namely anterior and posterior devices. The EfficientDet network was pretrained using the COCO dataset and additional training was done by applying the hold-out method to the manually annotated images. Online augmentation by means of image flipping and color adjustments was also implemented to help improve the generalisability of the model. The trained object detection model was then used to annotate the remaining 9545 images in the dataset, hence the model being described as weakly supervised.

For the classification stage of the pipeline, the detected devices are classified using class-specific classification models. This means that two classification models are trained; one for anterior implant classification and one for posterior implant classification. As indicated in figure 5.1, each classification model consists of a pretrained (ImageNet) DenseNet121 backbone network followed by two fully-connected layers (i.e. FC2, FC3) and an output layer. The backbone network converts each object detected in the first stage into a 1000 logits (raw outputs of a model) feature vector. The radiographic view of the original input X-ray was also encoded and concatenated with the feature vector before feeding into the two fully connected layers. A softmax function was used in the output layer to predict the probabilities of the implant representing each of the hardware brands.

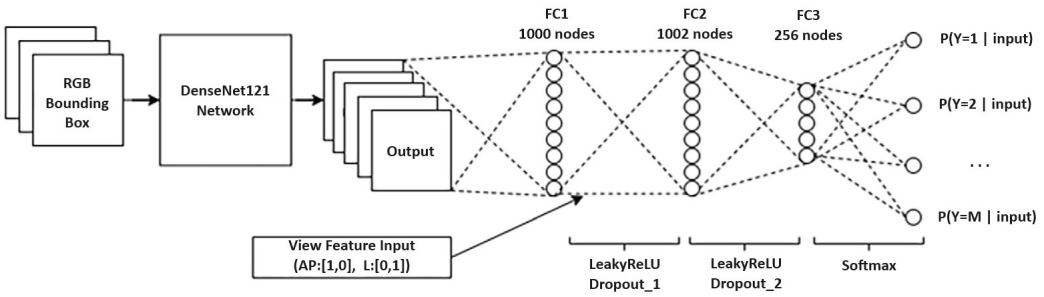


Figure 5.1: Classification stage of pipeline [9]

Training and testing of classification models was done using the hold-out approach.

Before splitting the dataset into training, validation and test sets for brand classification, the dataset was first compartmentalised according to patient and examination. This was done to reduce the risk of the model overfitting by avoiding any overlap between the training, validation, and test sets. For training, different learning rates were set for the DenseNet121 network and the two fully-connected layers. This allowed the model to train faster without making significant changes to the pretrained weights. Image augmentations, such as horizontal flipping, color adjustments, image rotation, and shearing, were also applied to the training data used for the classification models to reduce the risk of overfitting. To further optimise the model performance, ten-fold cross-validation was used to perform hyper-parameter tuning. Evaluation of the classification models was done using 1000 iterations of bootstrapping on the hold-out test set. For each iteration a randomly sized subset was sampled for the test set with sample replacement allowed. The average F1-score, sensitivity, and specificity scores were then calculated across the 1000 iterations.

The results of the study show that the hardware localisation stage of the pipeline was able to achieve an IoU of 86.8% and an F1-score (at IoU threshold of 50%) of 94.9%. For anterior implant classification, the model achieved an F1-score, sensitivity, and specificity of 98.7%, 98.7% and 99.2%, respectively. Slightly lower performance scores were noted for the classification of the posterior implants with the respective scores being 93.5%, 92.6% and 96.1%. The study also indicated the effectiveness of the implant localisation stage of the pipeline by comparing the classification model performance when given a whole X-ray image and an implant localised image. The results indicated that implant localisation improves the classification accuracy with a higher F1-score for both anterior (98.7% vs 92.8%) and posterior (93.5% vs 83.7%) hardware predictions.

5.2 Detection and classification of implanted electronic devices

White *et al.* [54] described a methodology for the detection and classification of lead-less implanted electronic devices (LLIED) on chest X-rays. The dataset used in the study contains 4871 chest X-ray images representing 1100 patients. All images were manually

annotated and labelled to represent nine different types of devices. The dataset was then split into a training/validation set consisting of 80% of the data and a testing set made up of the remaining 20%. Precaution was taken to avoid splitting multiple images of the same patient pathology into different groups. This helped to avoid the risk of patient-memory when testing the model.

For device detection, the faster R-CNN ResNet-50 [38] model was used due to its superior accuracy and adequate speed. The detection model was pretrained using the COCO data set and fined tuned using the hold-out method on data from the training/-validation sub-population. For the training of the detection model, the labels of the dataset were simplified to allow all LLIEDs to be represented by one class. To ensure 100% detection, a low probability threshold of 0.00002 was set for the faster R-CNN, which resulted in the model predicting an excessive number of boundary boxes with a high number of false positives. Therefore, a filter was also added to limit boundary boxes to a specific size and non-maximum suppression was applied to overlapping boundary boxes.

The classification of the detected devices was done using a multi-class CNN based on the Inception-v3 architecture. In addition to classifying the localised devices, the classification model also assisted in filtering out any remaining false positive boundary boxes. The weights of the classification model were initialised using transfer learning, which involved the replacement of the final layers of a pretrained (ImageNet) Inception-v3 model with a 1024-node fully connected layer followed by a sigmoid output layer. Offline augmentations, such as image flipping, scaling, and translation were applied to the RoIs of the training/validation sub-population to help reduce class imbalances. Finally, the model was fine-tuned using the hold-out approach on the ground truth RoIs.

The results indicated that the model is able to achieve 100% detection sensitivity, 46% detection precision and a 98.9% brand classification accuracy. The study identified the low probability threshold as the main limitation of the proposed model. The low probability threshold of the faster R-CNN model resulted in a large number of false positive predictions which reduced the detection precision. To counter this, the study suggested that the probability threshold parameter be adjusted according to the

X-ray image quality. For example, if the signal-to-noise ratio of the X-ray is poor a low threshold should be used, while for better signal-to-noise ratios a higher default threshold should be used.

5.3 Detection of implants on knee radiographs

Lee *et al.* [25] focused on developing a deep learning model that can detect 17 different types of implants on knee radiographs. The data used in the study was restricted to only anterior-posterior knee radiographs and consisted of two datasets, namely an internal and an external dataset. The internal dataset contained 5206 X-ray images representing 4435 patients from a single tertiary institution, while the external dataset had 238 X-ray images that were retrieved from a separate tertiary institution. Preprocessing of the data involved the annotation of all the images to indicate the type and location of each object, as well as setting the JPG quality of the internal data to 70 and the external data to 95.

The YOLOv5 deep learning algorithm was chosen for the implant detection, because of its fast and light architecture. One of the lightest and simplest versions of the YOLOv5 model, the YOLOv5s, was selected and pretrained using the COCO dataset. The model was then fine-tuned using a hold out approach on the internal dataset and model overfitting was mitigated by applying online image and color space augmentations to the training data set.

To measure the performance of the model, an IoU threshold of 0.5 was used to distinguish between false and true positives. The model achieved an accuracy, sensitivity and specificity score of 95.3%, 81% and 99% respectively, for the internal test set. Whereas the results for the external test set were 95.6%, 49.3% and 97.5%. The limitations of the study included large class imbalances in the internal dataset and low model sensitivity towards certain implants such as staples. The model was also trained on default hyperparameters and therefore model performance could likely be increased with hyperparameter tuning.

5.4 Classification of knee replacement implants

Belete *et al.* [3] developed a CNN model that can classify seven different types of total knee replacement (TKR) implants on radiographs. A dataset containing 558 anterior-posterior knee radiographs was used to train, validate, and test the model. The dataset also consisted of radiograph images with no implants to see whether the model can predict if the radiograph did not contain an implant. Preprocessing of the data was done by manual cropping of all the radiograph images to only include the implants, which helped to reduce the computational complexity of the model. Additionally, images were converted to grayscale and resized to match the size of input layer for the model.

The CNN model used in the study was based on the ResNet-18 architecture and was pretrained on the ImageNet dataset. The default 1000 feature output layer of the ResNet-18 model was replaced with an eight feature output layer, representing the seven implant classes and the one no-implant class. The hold-out approach was used to fine tune the model and a combination of offline and online data augmentations were utilised to limit overfitting. The study tested the use of both the Adam and SGD optimiser to reduce the cross-entropy loss and found that the SGD optimiser outperformed the Adam optimiser.

Results of the study indicated that the model was able classify all classes with a 100% performance score for both the accuracy and the F1-score metrics. The need for manual segmentation of input images was the main limitation to this model and the study suggested that future work attempt to automate the image segmentation using detection models like YOLO.

5.5 Summary

Table 5.1 summarises the dataset sizes, number of classes, and performance scores of the medical deep learning systems reviewed. The challenges that have to be considered before designing a trauma implant classification model include insufficient data samples, skew class distributions, and multi-class classification. The studies discussed in this chapter do not necessarily suffer from insufficient data, but do employ methods

that can be used to address data shortages. These include the use of data augmentation to artificially increase data as well as the utilisation of pretrained models to reduce the amount of data required for training. Three out of the four studies used online augmentation with the most common techniques being horizontal image flipping and image rotation. With regards to pretrained models, all object detection models were pretrained on the COCO dataset, while all classification models were pretrained on the ImageNet dataset.

The issue of imbalanced class distributions is observed in studies by Dutt *et al.* [9] and White *et al.* [54], which implement methods involving label weight adjustments and upsampling via augmentation, respectively. Weight adjustment of the training labels encourages the model to focus on underrepresented classes during training, while upsampling uses augmentation to artificially increase the number of samples in the minority classes.

A challenging aspect of trauma implant classification is that a single X-ray image can contain multiple implants that need to be classified. To address this, three of the applications [9, 25, 54] utilise object detection models to enable the classification and localisation of multiple implants in a single X-ray image. The article by Belete *et al.* [3] uses manual segmentation to localise and crop implants from the X-ray images, but suggests the use of an object detection model to replace manual segmentation.

Another challenge of performing detailed trauma implant classification is that there is a multitude different implant types and for every implant type there are numerous different brands and variations. The two-step model presented by Dutt *et al.* [9] uses a divide-and-conquer approach that allows it to perform more detailed implant classification. The process entails the utilisation of an object detection model to first classify the general type of implant, which is then followed by employing image classification models that are tailored to each general type of implant.

Table 5.1: Overview of the different medical deep learning systems reviewed.

Application	Model	#Images	#Classes	Detection Scores (IOU_{thresh} =0.5)	Classification Scores
Localisation and classification of cervical spine hardware [9]	EfficeintDet + DenseNet121	10589	10	F1-score =94.9%	F1-score =96.1%; Sensitivity =95.7%; Specificity =97.7%
Detection and classification of implanted electronic devices [54]	Faster R-CNN + InceptionV3	4871	9	F1-score =63%; Sensitivity =100%; Precision =46%	Accuracy =98.9%
Detection of implants on plain knee radiographs [25]	YOLOv5s	5206	17	Accuracy =95.3%; Sensitvity =81%; Specificity =99%	-
Classification of knee replacement implants on plain radiographs [3]	ResNet-18	558	8	-	F1-score =100%; Accuracy =100%

Chapter 6

Materials and Methods

This chapter discusses the data and methods utilised to train and test the chosen implant detection and implant classification models. The methodology described in this chapter is used to identify an optimal object detection model as well as an optimal classification model. The optimal models are then combined to form a final deep learning solution for trauma implant detection and classifcation.

For the task of object detection, the performance of the EfficientDet, faster R-CNN, and YOLO models are assessed. While for implant classification, the feature extraction capabilities of the DenseNet, Inception-v3, and ResNet architectures are evaluated. Chapter 6 is divided into four main sections, namely dataset prepration, implant detection modeling, implant classification modeling, and the final model pipeline.

6.1 Dataset Preparation

This section presents the data and the data preparation steps employed for both the classification and object detection models. The raw dataset used in the study was provided by Dr Franz Birkholtz under ethical clearance granted by the Health Research Ethics Committee of Stellenbosch University. The data was collected from a single institution and consisted of 204 X-ray images representing 98 patients. All data was anonymised and X-ray images were stored in JPEG format.

6.1.1 Implant detection dataset

Given the limited data, the implant detection dataset was only split into two subsets: a training set comprising of 80% of the data and a testing set consisting of the remaining 20%. To prevent X-rays from the same patient appearing in both the training and testing set, the images were split by patient. Preparation steps taken for the object detection data included image resizing, image annotation, and image augmentation. The object detection models chosen for the study all had the same input shape of $640 \times 640 \times 3$. Therefore the images in both the data subsets were kept in red-green-blue (RGB) format (three color channels) and resized to 640×640 to match the input shape of the models.

Image annotation was performed on the resized images using the *Labelme*[1] annotation tool. Preliminary analysis revealed that the data represented four categories of implants, namely plates, screws, intramedullary nails, and pins. These categories were used as labels for the implant annotation. Figure 6.1 illustrates examples of ground truth bounding boxes for the different implant categories.

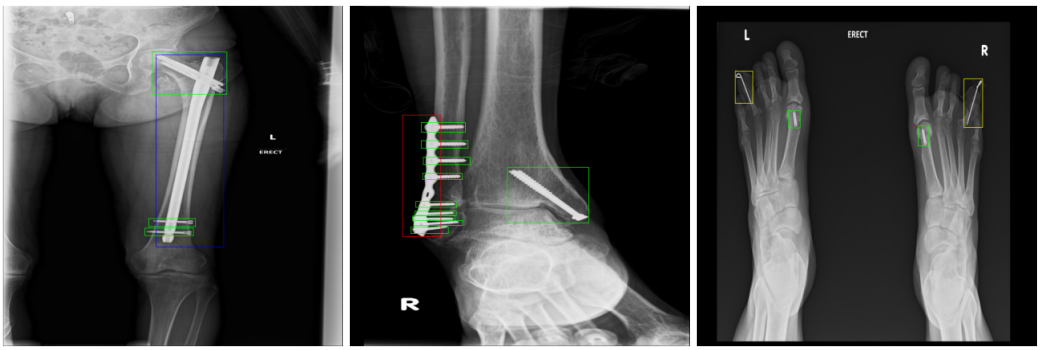


Figure 6.1: Ground truth bounding boxes for different implant categories (Intramedullary Nail=Blue, Screw=Green, Plate=Red, Pin=Yellow)

Further data analysis was performed following the image annotation to gain insights into the distribution of the implant categories. Table 6.1 summarises the distribution for the training and testing set. Examination of table 6.1 reveals that 156 of the 164

[1] https://github.com/wkentaro/labelme

unique images in the training set contain screw implants, while pin implants are only present in 6 unique images. The number of screw bounding boxes is also significantly larger compared to the other implants. This is mostly due to screws also being used to secure implants, such as plates and intramedullary nails.

Table 6.1: Distribution of the implant categories in object detection dataset. *The total unique patients and unique images does not equate to the sum of implant class values due to some patients having multiple different implants.

Implant Class	Training set			Test set		
	Unique Patients	Unique Images	Boxes	Unique Patients	Unique Images	Boxes
Screw	78	156	827	18	38	220
Plate	37	70	90	10	20	26
Intramedullary Nail	34	76	84	4	10	10
Pin	3	6	7	1	2	2
Total	78*	164*	1008	20*	40*	258

Image augmentation was utilised to artificially expand the training set and perform upsampling on the pin implant category. Offline augmentation was chosen to ensure that all models were trained on exactly the same data. The following augmentation techniques were implemented using the *Albumentations*[1] computer vision tool:

- Random horizontal flipping: Flip image around the y-axis.
- Random rotation: Rotate image by a random angle selected from the range (-5°,5°).
- Gaussian blur: Apply blur to image using a Gaussian filter with a random kernel size. The Gaussian kernel size range was set to (3,7).
- Random brightness and contrast: Randomly adjust the brightness and contrast of the image. The factor range for changing the brightness and contrast was set at (-0.2, 0.2).

The application probability for the flipping, rotation and blur transforms was set at $p = 0.5$, while the brightness and contrast transform had a application probability of

[1] https://albumentations.ai/docs/

$p = 0.2$. Figure 6.2 presents the results obtained when the augmentation techniques specified above were applied to a given sample.

Figure 6.2: Image augmentation results.
Transformations used for new image 1: *[GaussianBlur, RandomBrightnessContrast]*
Transformations used for new image 2: *[Rotate(angle: -3.228), HorizontalFlip, GaussianBlur, RandomBrightnessContrast]*

Using image augmentation, the number of images in the training set was increased by generating four additional images for every original image containing a pin implant and two additional images for every other original image. This process helped to increase the training set from 164 images to 504 images. Furthermore, the percentage of unique images in the training set that contained pin implants was increased from 3.66% to 5.95%. Even though the upsampling only marginally increased the pin implant images, more aggressive upsampling was not considered due to the risk of too many similar images in the dataset.

6.1.2 Implant classification dataset

The evaluation of the implant classification models was limited to one implant category due to the data and time constraints of the project. The images of intramedullary nails displayed the least variation between the anterior-posterior and lateral projections, and had the most balanced class distribution. Therefore, all classification models were evaluated on classifying the different types of intramedullary nails.

Out of the initial dataset of 206 images, 85 images contained intramedullary nails and

were used for the classification dataset. Classification labels were assigned to each of the 85 images with the help of an orthopedic surgeon. The images utilized for the classification models were generated by cropping each intramedullary nail using the ground truth bounding boxes. Table 6.2 outlines the class distribution of the intramedullary nails in terms of the number of patients and the number of cropped images.

Table 6.2: Distribution of classes in intramedullary nail classification dataset

Intramedullary nail class	Unique Patients	Unique cropped images	Unique cropped images after upsampling
Cephalomedullary nail (long)	10	26	26
Tibial nail	10	28	28
Femur trauma nail	6	12	24
Cephalomedullary nail (short)	4	8	24
Hind foot nail	4	10	30
Retrograde femur nail	2	-	-
Retrograde lengthening femur nail	1	-	-
Lengthening tibial nail	1	-	-

To help reduce the noise in the data, the three classes with the lowest number of unique pasients, namely retrograde femur nail, retrograde lengthening femur nail, and lengthening tibial nail, were excluded from the training and testing of the classification models. Figure 6.3 illustrates the different nail implants that were used for the evaluation of the classification models. In order to mitigate the effect of the imbalanced class distribution, image augmentation was used to perform upsampling. The image augmentation was done prior to the image cropping and the same augmentation techniques were used as those employed for the object detection dataset. Table 6.2 indicates the number of cropped images for each class after upsampling was applied.

After upsampling and cropping the images, each image in the dataset was resized and normalised. Two out of the three feature extraction architectures evaluated for the implant classification had default input shapes of $224 \times 224 \times 3$. Thus, all the cropped images were resized to $224 \times 224 \times 3$. All images where normalised using min-max normalisation, which scaled the pixel intensity values of to images from [0,255] to [0,1].

Figure 6.4 illustrates examples of the intramedullary nail images after all the preprocessing steps.

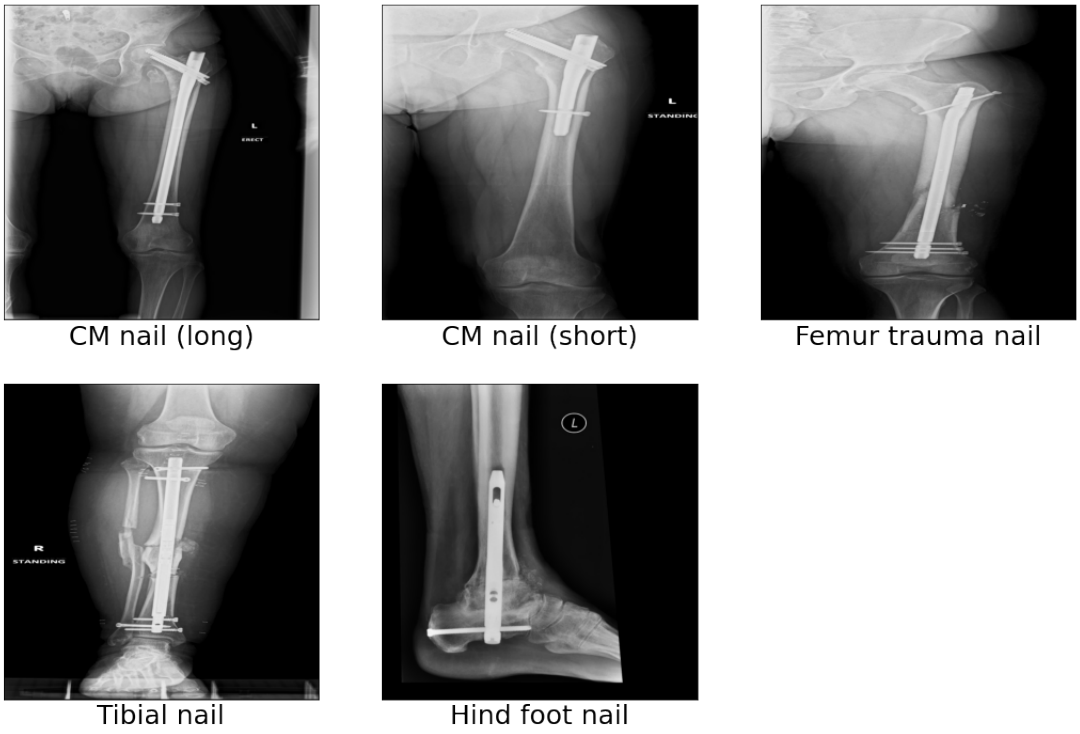


Figure 6.3: The different intramedullary nail types used for classification.
(CM = Cephalomedullary)

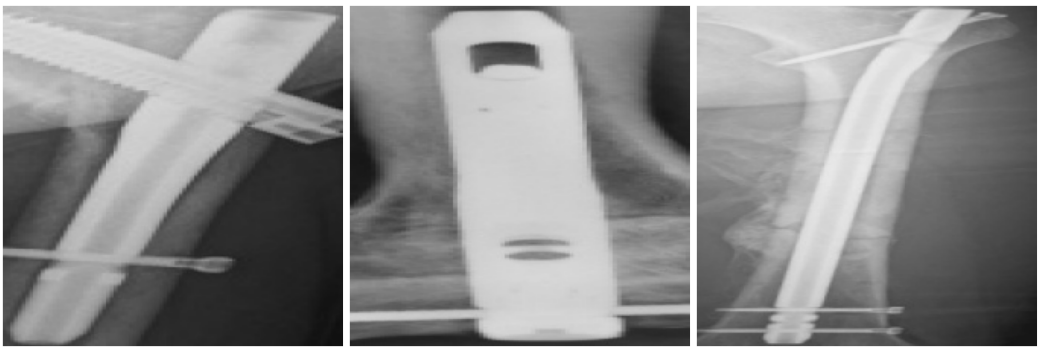


Figure 6.4: Examples of the pre-processed intramedullary nail images

Both the cross-validation and hold-out methods where utilised for the classification modeling. Therefore, additional data preparation steps taken for the hold-out ap-

proach involved splitting the data into a training, validation and testing set. To ensure that there was no overlap between the training/validation and testing set, the testing set was constructed by extracting the data of five distinct patients form classification dataset. Each test set patient corresponded to one of the intramedullary nail classes. The remaining data in the classification dataset was then split into a training set (90%) and a validation set (10%). For a more robust analysis, the data partitioning was repeated three times and the classification models were evaluated on each of the three different training, validation, and test sets. It is important to note that upsampling techniques were not applied to the test sets.

6.2 Implant detection modeling

The selection of object detection models evaluated in this study was guided by the applications reviewed in chapter 5 and included the YOLOv5s, Faster-RCNN (ResNet-50), and EfficientDet-D1 model. For the EfficientDet model, the D1 variant was chosen instead of the D0 variant in order to ensure that all the models had an uniform input size of $640 \times 640 \times 3$. A single framework suitable for the implementation of all three object detection models could not be identified. Thus, the Tensorflow Object Detection API[1] was utilised for the Faster-RCNN (ResNet-50) and EfficientDet-D1 models, while the Ultralytics API [2] was used for the implementation of the YOLOv5s model.

All models were pre-trained on the COCO dataset and then fine-tuned. Fine-tuning involved retraining the entire model on the trauma implant training set discussed in section 6.1.1. The number of classes to be detected by the models was set to four, representing the basic implant categories. Each model underwent training using initial learning rates of 0.001, 0.01, and 0.1, with the aim of identifying the optimal learning rate for each model. Training was performed over 12600 steps (100 epochs) using the SGD optimiser with a momentum of 0.9 and a cosine decay learning rate schedule. The cosine decay was set to decay the learning rate over 12600 steps without warm-up. Due to the computational constraints, a small batch size of four was selected for training. Other configuration steps taken included removing the default online augmentation

[1] https://github.com/tensorflow/models/blob/master/research/object_detection
[2] https://github.com/ultralytics/yolov5

steps, to ensure that all models were trained on the same data, and standardising the non max suppression applied to each of the models.

The object detection models were evaluated during training at regular intervals on the test set described in section 6.1.1. Unlike the Ultralytics[1] framework, the TensorFlow Object Detection[2] API treats evaluation/validation as an independent process that should be launched in parallel with the training job. The simultaneous execution of the training process and evaluation process in single Google Colab[3] notebook was a troublesome task, and therefore the Faster-RCNN (ResNet-50) and EfficientDet-D1 models were evaluated post-training using model checkpoints saved at every 600 steps ($\approx$ 5 epochs) during training.

Initial analysis was performed on the object detection models by plotting the mean average precision at different IoU thresholds for every training checkpoint. To summarise the model performance at each checkpoint, a fitness score was calculated using

$$fitness = (0.9 \times \text{mAP}_{0.5}) + (0.1 \times \text{mAP}_{0.5:0.95}) \tag{6.1}$$

where $\text{mAP}_{0.5}$ refers to the mean average precision for an IoU threshold of 0.5 and $\text{mAP}_{0.5:0.95}$ refers to the mean average precision for multiple IoU thresholds ranging from 0.5 to 0.95. Equation (6.1) was based on the default fitness function used in the Ultralytics framework. The optimal model weights for each object detection architecture was identified by selecting the weights corresponding to the model training checkpoint that achieved the highest fitness score on the test set.

Multi-class confusion matrices were utilised to visualise the prediction performance per implant category for each of the object detection architectures. The confusion matrix analysis focused on predictions with confidence scores larger than 0.25. The following definitions were used to create the multi-class confusion matrix and to differentiate between true positive (TP), false positive (FP), and false negative (FN) predictions:

[1] https://github.com/ultralytics/yolov5
[2] https://github.com/tensorflow/models/blob/master/research/object_detection
[3] https://colab.google/

- Correct prediction (TP): The predicted bounding box has an $IOU \geq 0.5$ with a ground truth box and the correct implant category is predicted.
- Mispredicted (FP): The predicted bounding box has an $IOU \geq 0.5$ with a ground truth box, but the incorrect implant category is predicted.
- Ghost Prediction (FP): The predicted bounding box does not have an $IOU \geq 0.5$ with a ground truth box.
- Undetected (FN): A groundtruth box that does not have an $IOU \geq 0.5$ with any of predicted bounding boxes.

In object detection a true negative (TN) refers to an image that does not contain an object. Therefore, the TN measure was excluded from the object detection evaluation, because all training and testing images contained objects. Figure 6.5 illustrates which cells in the confusion matrix relate to the definitions above. The *ghost prediction* row and *undetected* column was added to the confusion matrix to offer better insight into the various types of prediction errors that occurred.

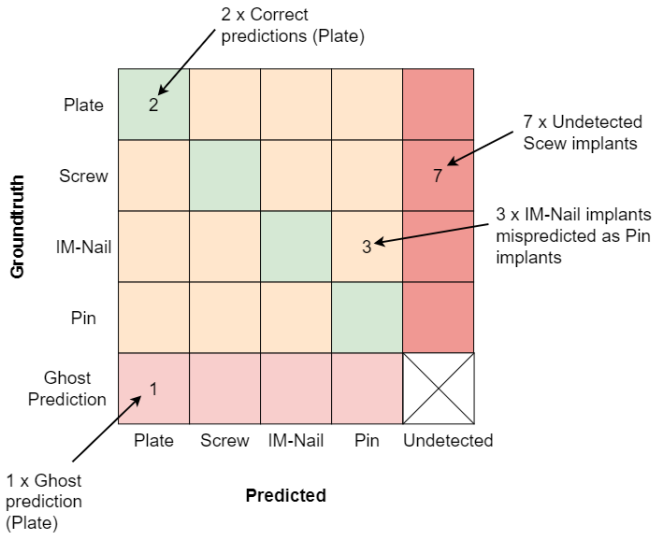


Figure 6.5: Custom object detection confusion matrix

In addition to the confusion matrix analysis, the precision, recall, and F1-score metrics were calculated using the true positive (TP), false positive (FP), and false negative

(FN) definitions above. The metrics were calculated per implant category for both the training and testing set to determine how well each model architecture fits the data. When calculating the precision scores, the sum of false positives (FP) per implant category was determined by adding the number of *ghost predictions* with the total mispredictions related to the specific implant. For mispredictions per implant, only mispredictions with groundtruth annotations associated with the specific implant were considered.

6.3 Implant classification modeling

The classification models evaluated in this project were based on the classification models used in the applications discussed in chapter 5. All models were trained using a transfer learning approach, which involved using a network pretrained on the ImageNet dataset to generate a feature array and adding fully connected layers at the end of the architecture to perform classification on the extracted features. All classification models were implemented using the Keras[1] deep learning API.

Each model evaluated comprised of a feature extraction network that fed into three fully connected hidden layers and an output layer. For the hidden layers, two 1024-node layers and a 256-node layer were used. All hidden layers employed ReLU activation functions and in addition dropout layers (dropout=0.5) were incorporated between the hidden layers to mitigate possible overfitting. To perform multi-class classification, an output layer with five nodes utilising softmax functions was selected. Each node in the output layer represented the probability of one of the implant classes corresponding to the input image. Therefore, the implant class predicted by the model is the class that relates to the output node with the largest value/probability. It is important to note that all the data labels had to be one-hot encoded to train and test the multi-class classification model. Figure 6.6 illustrates the general architecture of the classification model.

The feature extraction architectures evaluated were the DenseNet121, Inception-v3,

[1] https://keras.io/api/applications/

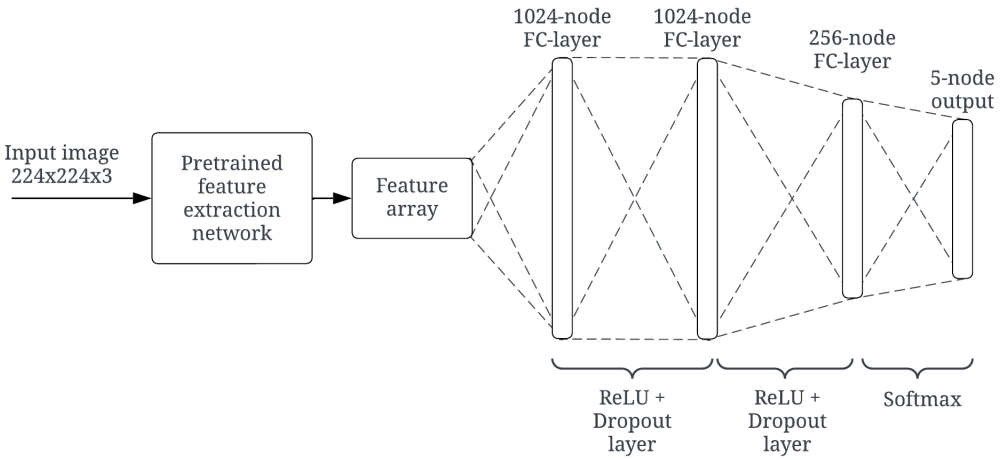


Figure 6.6: Classification model architecture

ResNet18, and ResNet101 architectures. In order to obtain robust performance estimates of the feature extraction architectures with the limited data, a form of leave-one-out cross validation (LOOCV) was utilised. For each fold of LOOCV, the data of a single patient was temporarily removed from the dataset and used for model testing, while the remaining data was used for model training. This process was repeated for each unique patient in the dataset and the model predictions on the test data were stored for each fold. Model performance was measured by calculating the accuracy, weighted F1-score, weighted precision and weighted recall metrics on the combined model predictions of all the folds. The area under the receiver operating characteristic curve (AUC) metric was also calculated using the one versus rest method.

The purpose of the LOOCV was to determine which pretrained feature extraction network is the most effective for the given implant data. Therefore, the weights of feature extraction network were kept unchanged and only the weights of the added fully connected layers were trained. For each cross-validation fold the models were trained for 20 epochs using cross-entropy loss and a batch size of eight. The model weights were optimised using the Adam optimiser with a learning rate of 1×10^{-3}. Because of the limited number of unique patients per implant class there was a risk of introducing class imbalance when removing the data of a single patient during cross-validation. To counter this, class weights were calculated for each fold and used during model train-

ing. The Scikit-learn[1] *compute_class_weight* function was utilised to compute the class weights as follows:

$$W_i = \frac{T}{K * n_i} \tag{6.2}$$

where W_i is the weight of class i, T is the total number of samples, K is the number of classes, and n_i is the number of samples for class i.

The final model consisting of the most efficient extraction network, identified using the LOOCV, was then trained and evaluated using the hold-out method. To maximise the training efficiency and to optimise model performance given the limited training data, model training was performed over two stages. For the first stage the weights of the feature extraction network were kept unchanged and only the weights of the added fully connected layers were trained. Fine-tuning was then performed for the second stage, which involved training the entire network at a very low learning rate.

The Adam optimiser was utilised for both stages, and the epoch and batch size hyper-parameters were kept at 20 and eight, respectively. The learning rate for the first stage was kept at 1×10^{-3}, while the learning rate for second stage was set to 1×10^{-5}. In addition a callback function was introduced for both stages, which would stop the model training of the given stage if the validation loss stopped decreasing for three consecutive epochs. The final classification model was evaluated on three different training, validation, and testing sets to help perform a robust performance analysis.

The accuracy, weighted F1-score, weighted precision, weighted recall, and AUC metrics were then calculated for each of the three different hold-out test sets. In addition, the confusion matrices were also plotted to visualise the model predictions for the three test sets.

6.4 Final model pipeline

The optimal trauma implant identification solution combined the optimal object detection and classification model into a single pipeline. To train and test the pipeline, the same three training and testing patient populations were used as in the classification

[1] https://scikit-learn.org/stable/modules/classes.html

modeling. Thus, both the implant detection model and implant classification model were trained using the data from the same patient populations. For testing, the test set data was fed into the implant detection model, and its output served as the input for the implant classification model. The performance of the pipeline on the three test sets was measured using the accuracy, weighted F1-score, weighted precision, weighted recall, and AUC metrics. In addition, the output of the pipeline was also summarised using confusion matrices.

6.5 Summary

Chapter 6 demonstrated the material and methods used to investigate which object detection and classification models are the most efficient in trauma implant detection and classification. The chapter started with an analysis of the data and presented the data preparation steps taken before training and testing the implant detection and classification models. In addition, the steps used in modeling the object detection and classification models were discussed, along with the methods and metrics employed to evaluate model performance. Lastly, the process used to test the final solution, which combines the optimal object detection model and classification model into a single pipeline, was discussed.

Chapter 7

Results

This chapter presents the results of the model evaluation methods described in chapter 6. The results chapter is divided into three sections, with the first two sections presenting the implant detection and implant classification results. The final section assesses the optimal deep learning pipeline, which combines the two optimal models identified in the initial sections.

7.1 Implant Detection

Figure 7.1 illustrates the performance progression during training for each of the object detection architectures. Each graph displays the respective model's $mAP_{0.5}$ and $mAP_{0.5:0.95}$ progression for three different initial learning rates. Additionally, the mAP results that relate to the model checkpoint with highest fitness score is also marked for each model architecture.

Analysis of figure 7.1 reveals that the faster R-CNN (ResNet50) and YOLOv5s models had distinctly higher mAP results compared to the EfficientDet-D1 model, for all three learning rates. The optimal model weights for each architecture was based on the model checkpoint with the highest fitness score. The optimal model checkpoint for the faster R-CNN (ResNet50) architecture was at 3600 training steps, whereas the optimal model checkpoints for the EfficientDet-D1 and YOLOv5s models were at 8400 and 8316 steps respectively. All three architectures achieved their optimal performance with the initial learning rate set to 0.01. The $mAP_{0.5}$ results for optimal faster R-CNN (ResNet50),

EfficientDet-D1 and YOLOv5s models were 0.903, 0.673 and 0.967 respectively, while the $mAP_{0.5:0.95}$ results were 0.520, 0.406, and 0.564 respectively. Additional analysis was performed on the optimal model weights for each architecture using the F1-score, precision, and recall metrics as well as confusion matrices.

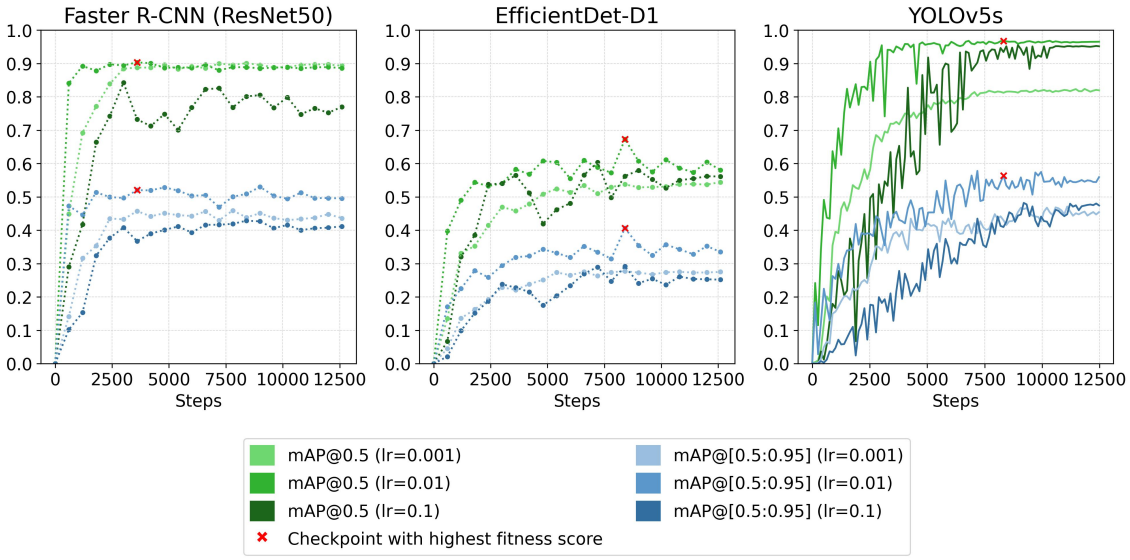


Figure 7.1: The mean average precision (mAP) scores of the object detection architectures, calculated on the test set for each of the training checkpoints (lr=learning rate).

The confusion matrices in figure 7.2 provide a summary of the prediction errors made by the different object detection models. The most critical failure scenario for an implant detection model would be its failure to detect an implant. Figure 7.2 indicates that the YOLOv5s model had failed to detect 25 implants, whereas the number of undetected implants for the faster R-CNN and EfficientDet-D1 models were 72 and 70 respectively. Furthermore, the EfficientDet-D1 model had the largest variety of implants not detected, namely plates, screws, and intramedullary nails. The majority of the undetected implants, for all three models, were screw implants.

Limiting the number of implant misclassifications during object detection is also critical for performing accurate implant classification. The total number of misclassifications for both the faster R-CNN (ResNet50) and YOLOv5s models was two, whereas

the EfficientDet-D1 model misclassified a total of nine implants. An overview of the misclassifications for all the models indicates that the plate implants were generally confused for screws or IM-nails, while some IM-nails were misclassified as pin or plate implants. The EfficientDet-D1 model also misclassified all the pin implants as either plates or IM-nails. All three models accurately predicted all the detected screw implants, which is most likely due to the higher distribution of screw implants in the training set.

The total number of ghost predictions per model provides a good indication of how accurate each model was in predicting bounding box coordinates. The faster R-CNN (ResNet50) model had the most ghost predictions with a total of 60, while the EfficientDet-D1 and YOLOv5s models had a total of 41 and 31 respectively. The overall majority of the ghost predictions were screw predictions.

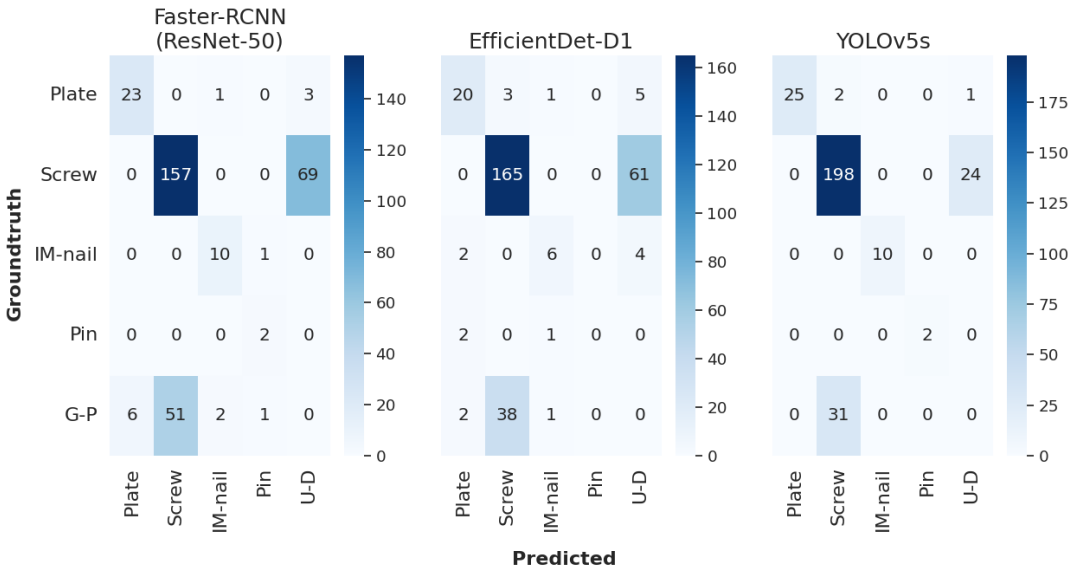


Figure 7.2: Confusion matrices for test set predictions. (G-P = Ghost predictions; U-D = Undetected)

Tables 7.1, 7.2, and 7.3 specify the F1-score, precision, and recall values of each object detection model. The metrics were calculated for both the training and testing set to provide insight into how well each model fitted the data. Table 7.1 indicates that the

faster R-CNN (ResNet50) model achieved a precision and recall average of 97.5% and 99.4%, respectively, for the training set, while the averages were 73.9%, and 89.5%, respectively, for the testing set.

Table 7.2 shows that the precision and recall averages for the EfficientDet-D1 model were 83.0% and 99.2%, respectively, for the training set. Compared to the faster R-CNN (ResNet50) and YOLOv5s model, the EfficientDet-D1 model had a significantly lower precision score for the training set. Furthermore, the EfficientDet-D1 model achieved a precision and recall average of 56.2% and 53.2%, respectively, on the testing set. Thereby, making the model the poorest overall performer on the test set. Table 7.3 shows that the YOLOv5s model achieved a precision and recall average of 95.4% and 99.2%, respectively, for the training set, while the metric averages were 94.8% and 96.3%, respectively, for the test set. The fact that the YOLOv5s model achieved high performance scores for both the training and test set indicates that the model fitted the data optimally and was able to generalise to unseen testing data.

Table 7.1: Performance metrics - faster R-CNN (ResNet50)

	Train set			Test set		
	F1-score (%)	Precision (%)	Recall (%)	F1-score (%)	Precision (%)	Recall (%)
Plate	99.5	98.9	100	82.1	76.7	88.5
Screw	97.1	96.7	97.5	72.3	75.5	69.5
IM-nail	100	100	100	86.9	76.9	100
Pin	97.2	94.6	100	79.9	66.7	100
Average	98.4	97.5	99.4	80.3	73.9	89.5

To gain a deeper understanding of the differences in the performance scores presented in tables 7.1, 7.2 and 7.3, figure 7.3 illustrates the training and testing classification loss for each of the object detection models. The plots show that the difference between the training and testing loss starts to increase early on for both the faster R-CNN (ResNet50) and EfficientDet-D1 models. This clearly indicates that the faster R-CNN (ResNet50) and EfficientDet-D1 models experienced overfitting, resulting in poor performance on the test set data. Conversely, the difference between the training and

testing loss for the YOLOv5s model was much smaller and decreased over the 12600 training steps.

Table 7.2: Performance metrics - EfficientDet-D1

	Train set			Test set		
	F1-score (%)	Precision (%)	Recall (%)	F1-score (%)	Precision (%)	Recall (%)
Plate	95.5	91.5	100	78.4	76.9	80.0
Screw	97.5	97.2	97.7	76.9	81.3	73.0
IM-nail	78.7	65.4	98.9	63.1	66.7	60.0
Pin	87.5	77.8	100	0.0	0.0	0.0
Average	89.8	83.0	99.2	54.6	56.2	53.2

Table 7.3: Performance metrics - YOLOv5s

	Train set			Test set		
	F1-score (%)	Precision (%)	Recall (%)	F1-score (%)	Precision (%)	Recall (%)
Plate	96.3	93.0	100	94.3	92.6	96.2
Screw	94.9	93.1	96.7	87.8	86.5	89.2
IM-nail	99.2	98.4	100	100	100	100
Pin	98.6	97.2	100	100	100	100
Average	97.3	95.4	99.2	95.5	94.8	96.3

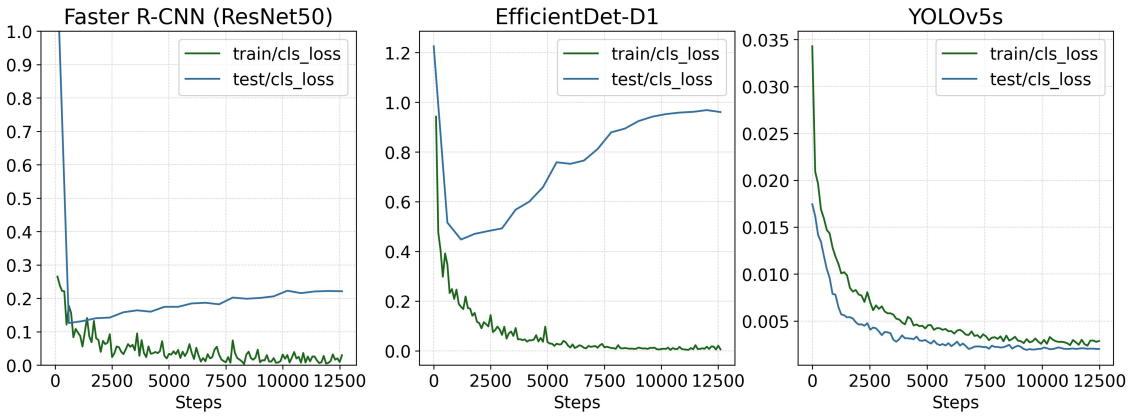


Figure 7.3: The classification losses of each object detection model (lr=0.01).

7.2 Implant Classification

The LOOCV results in figure 7.4 and table 7.4 highlight the efficiency of each of the feature extraction networks assessed. Figure 7.4 depicts the cross-entropy loss during training for each of the models. The graph illustrates the training loss across the 20 epochs for every cross-validation fold as well as the average training loss of each model over 20 epochs. The loss plot indicates that the loss for both the ResNet models converged much quicker than the loss of the DenseNet121 and Inception-v3 models. The DenseNet121 and Inception-v3 models also achieved a significantly lower average loss (at epoch=20) of 0.194 and 0.234 respectively. On the other hand, the ResNet18 and ResNet101 had an average loss (at epoch=20) of 1.617 and 1.607 respectively.

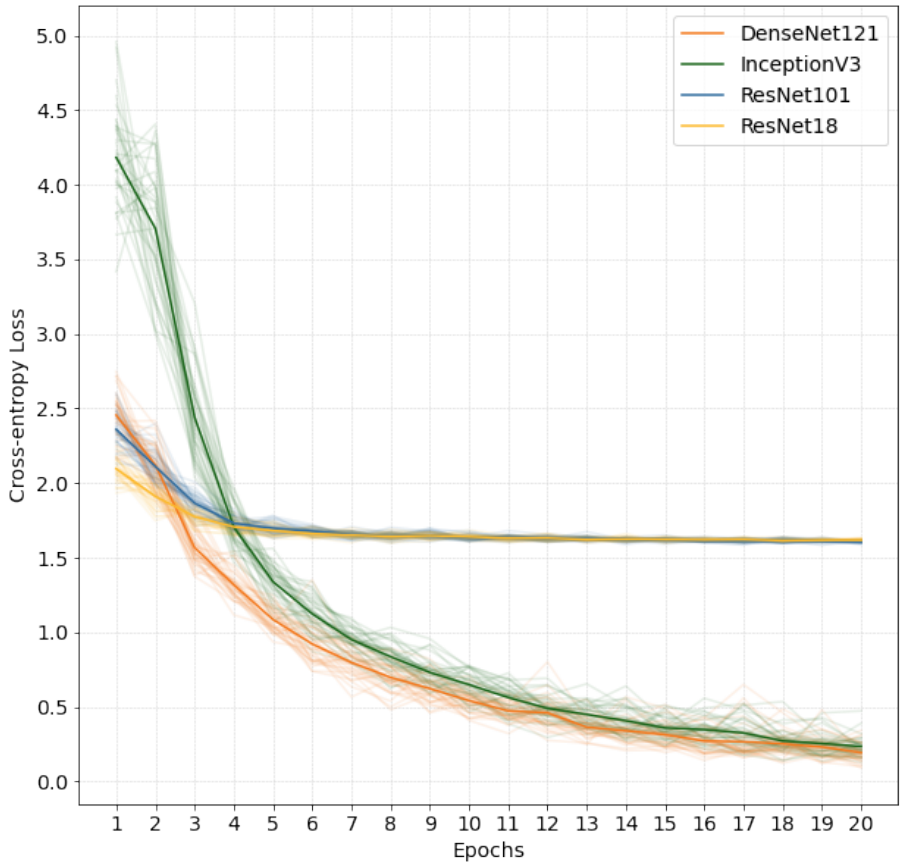


Figure 7.4: The training cross-entropy loss per model for each leave-one-out cross-validation fold. The average cross-entropy loss is also plotted for each model.

Table 7.4 presents the accuracy, F1-score, precision, recall and AUC metrics calculated during LOOCV for each model. The table indicates a noticeable disparity in the performance of the ResNet models and the performance of the DenseNet121 and Inception-v3 models. The DenseNet121 model achieved the highest performance scores with an accuracy, precision, and recall of 72%, 72.3% and 72%, respectively. Whereas, the Inception-v3 model had an accuracy, precision, and recall of 66.7%, 69.2% and 66.7%, respectively. The AUC values close to 0.5, indicated that the predictive abilities of both ResNet18 and ResNet101 classification models were no better than random guessing.

Table 7.4: Leave-one-out cross-validation results

Model	Accuracy (%)	F1-score (%)	Precision (%)	Recall (%)	AUC
DenseNet121	72.0	71.7	72.3	72.0	0.921
Inception-v3	66.7	66.6	69.2	66.7	0.885
ResNet18	23.5	21.8	21.8	23.5	0.455
ResNet101	24.2	24.0	25.1	24.2	0.534

Therefore, the DenseNet121 network was selected as the optimal feature extraction network for the trauma implant classification model. Table 7.5 indicates the performance scores of the DenseNet121 classification model for three distinct hold-out test sets. The average accuracy, precision and recall scores for the hold-out test sets were 79.6%, 84.9% and 79.6%, respectively. The confusion matrices in figure 7.5 summarise the predictions made by the DenseNet121 model for each of the three test sets. The misclassifications for the three test sets were distributed across all the classes, with the tibial nail class having the highest number of misclassifications, totalling three. Out of the seven misclassifications the model wrongfully predicted five of the implant images as cephalomedullary nail (long) implants.

Table 7.5: Performance scores of the DenseNet121 classification model.

Hold-out test set	Accuracy (%)	F1-score (%)	Precision (%)	Recall (%)	AUC
Test set 1	81.8	81.8	90.9	81.8	0.978
Test set 2	80.0	79.3	83.3	80.0	0.925
Test set 3	76.9	75.9	80.5	76.9	0.970
Average	79.6	79.0	84.9	79.6	0.958

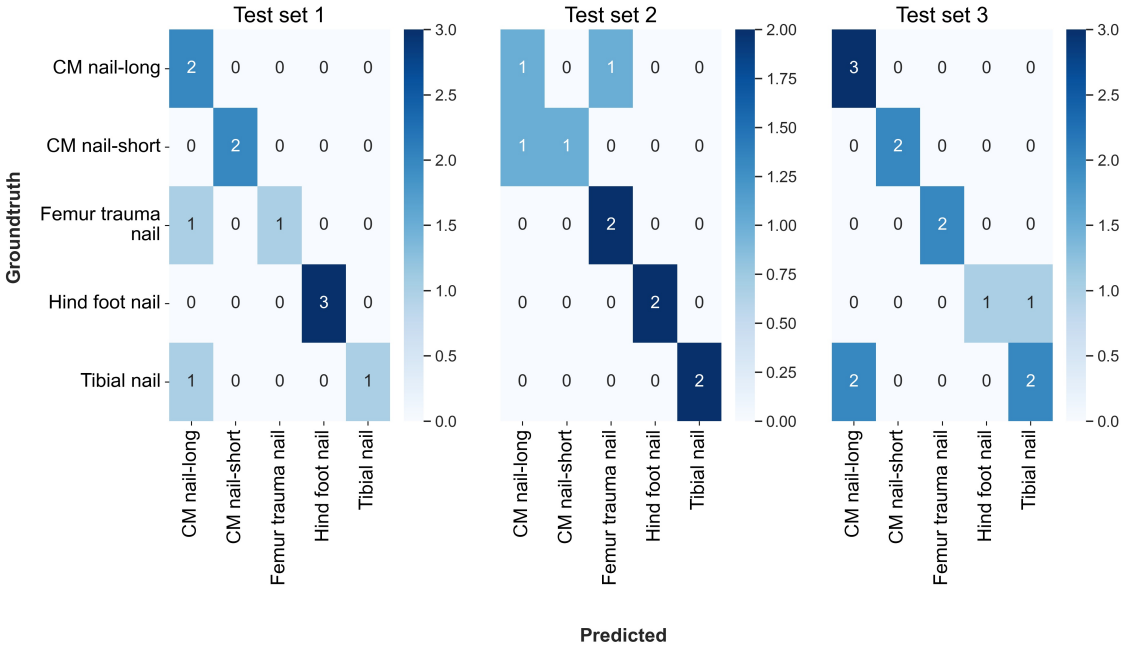


Figure 7.5: Confusion matrices of the DenseNet121 classification model. (CM = Cephalomedullary)

7.3 Optimal pipeline

The optimal solution, based on the results in sections 7.1 and 7.2, was a pipeline that combined the YOLOv5s object detection model with the DenseNet121 classification model as illustrated in figure 7.6. Table 7.6 indicates that the proposed pipeline achieved an average accuracy, precision and recall score of 73.7%, 81.9% and 73.7%, respectively, for the three test sets. A comparison of the performance scores in table 7.5 and table 7.6 revealed that there was a marginal decrease in model performance when the predicted bounding boxes were utilised for image cropping instead of the ground

truth bounding boxes.

The confusion matrices in figure 7.7 provided an overview of the predictions made by the proposed pipeline, for the three test sets. A combined total of nine misclassifications were observed over the three test sets with the femur trauma nail and tibial nail classes accounting for three misclassifications each. Out of the nine misclassifications, the model misclassified six of the implants as cephalomedullary nails (long). There was also a clear similarity between the confusion matrices in figure 7.5 and figure 7.7, indicating that the utilisation of predicted bounding boxes had a minimal effect on the classification results.

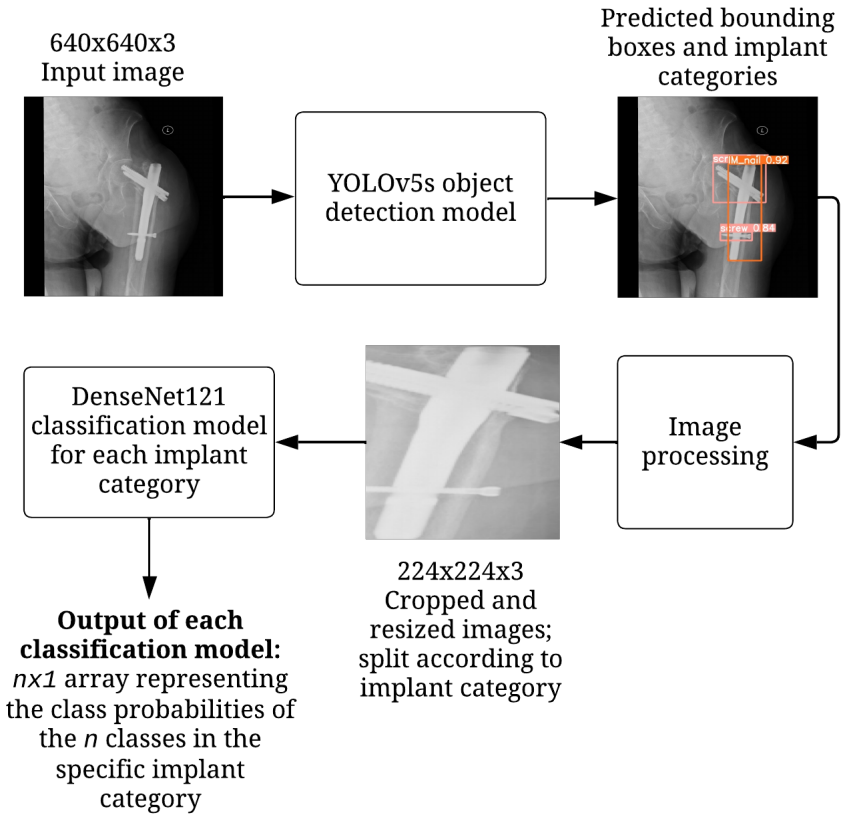


Figure 7.6: Optimal trauma implant detection and classification pipeline

Table 7.6: Performance scores of the proposed pipeline.

Hold-out test set	Accuracy (%)	F1-score (%)	Precision (%)	Recall (%)	AUC
Test set 1	81.8	81.8	90.9	81.8	0.967
Test set 2	70.0	71.3	76.7	70.0	0.925
Test set 3	69.2	68.9	78.2	69.2	0.951
Average	73.7	74.0	81.9	73.7	0.948

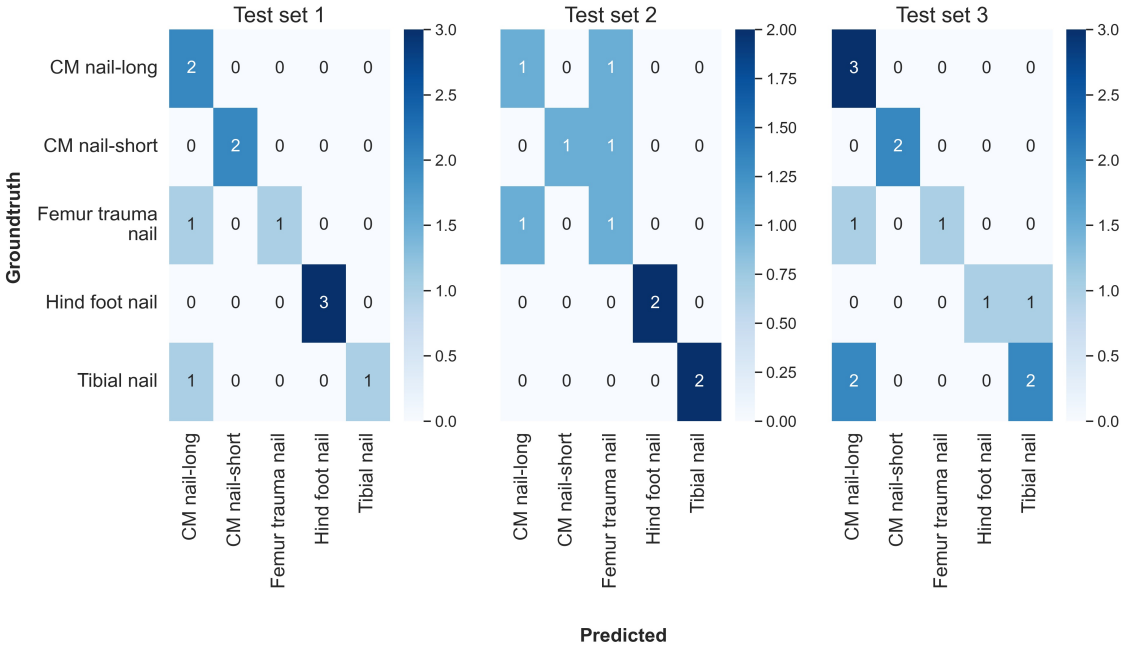


Figure 7.7: Confusion matrices of the proposed pipeline (CM=Cephalomedullary).

7.4 Summary

Chapter 7 provided the performance results of the object detection and classification models evaluated. Out of three object detection models evaluated, the YOLOv5s model achieved the best performance on the test data with an F1-score of 95.5%. The leave-one-out cross-validation results revealed that the pretrained DenseNet121 architecture was the most effective in feature extraction, thus it was chosen for the optimal classification model. The DenseNet121 classification model achieved an average accuracy and F1-score of 79.6% and 79.0%, respectively, across three distinct hold-out test sets. Lastly, the optimal pipeline combined the YOLOv5s object detection model and the

DenseNet121 classification model. Across three distinct hold-out test sets, the pipeline achieved an average accuracy and F1-score of 73.7% and 74.0%, respectively.

Chapter 8

Discussion

This project investigated the concept of automating trauma implant localisation and classification for leg X-rays using deep learning. Chapter 8 outlines the main discoveries of the investigation and offers interpretations based on the findings. In addition, the implications and limitations related to the investigation are discussed and suggestions are made for future research.

For trauma implant localisation the faster R-CNN (ResNet50), EfficientDet-D1, and YOLOv5s models were evaluated on their ability to detect four categories of trauma implants, namely plates, screws, pins, and intramedullary nails. The results revealed that the object detection model with the highest performance on the trauma implant dataset was the YOLOv5s model. The YOLOv5s model achieved the highest $mAP_{0.5}$, $mAP_{0.5:0.95}$, precision, and recall scores on the test set. Furthermore, the small difference in the training and testing performance of the YOLOv5s model indicated that the model had an optimal capacity. In contrast, the results also revealed that both the faster R-CNN (ResNet50) model and the EfficientDet-D1 model suffered from overfitting, which led to poor test set performance for both models. The overfitting was likely a result of excessively high model capacity, as well as a very limited training set.

To obtain a rough estimate of the capacity of each object detection model, the number of parameters per model was considered. The faster R-CNN (ResNet50) model used the most parameters, totaling 42×10^6, while the number of parameters used in the YOLOv5s and EfficientDet-D1 models were 7.2×10^6 and 6.6×10^6, respectively. Thus,

the faster R-CNN (ResNet50) had a significantly higher number of parameters, which likely contributed to the model overfitting. On the other hand, the EfficientDet-D1 model also experienced overfitting, but had the lowest number of parameters. This could be due to the training set being too small, and as a consequence the EfficientDet-D1 model possibly memorised specific examples of implants rather than learning the underlying patterns.

To identify areas of difficulty for the object detection models, the confusion matrices in figure 7.2 were examined. The matrices indicated that the models struggled the most to detect screw implants. This was likely due to occlusion caused by many of the screw implants overlapping in the training and testing images, as illustrated in figure 8.1. Furthermore, the majority of the ghost predictions for each model were associated with the screw implant. This suggests that the overlapping screw implants likely introduced noise into the training set, leading to a decrease in bounding box accuracy. It is also important to note that the training set had a significantly larger distribution of screw implants, due to the screw implants also being used in the fixation of plate and intramedullary nail implants. Thus, it is probable that the models were biased towards the screw implants, contributing to the high number of screw implant ghost predictions.

Figure 8.1: Overlapping implants

For trauma implant classification, the DenseNet121, Inception-v3, ResNet18 and ResNet101 classification models were assessed for their ability to classify five types of intramedullary nails. The findings indicated that, among the four models assessed, the DenseNet121

classification model attained the highest accuracy, precision, recall, and AUC scores, closely followed by the performance scores of the Inception-v3 model. On the contrary, the pretrained ResNet networks were unable to extract valuable features from the trauma implant dataset, resulting in the predictive abilities of the ResNet models being no better than random guessing. This was likely due to pretrained ResNet networks extracting features that were more tailored to the ImageNet dataset and failed to adequately represent the trauma implant dataset.

Additional examination of the DenseNet121 classification model included an assessment of the performance of the model on three different training, validation, and testing sets. The results indicated that the DenseNet121 classification model was able to achieve consistent and satisfactory results across the three test sets. The confusion matrices for the three test sets indicated that the misclassifications made by the model were distributed across all the nail classes and that the majority of misclassified implants were classified as long cephalomedullary nails. The misclassifications were most likely due to the limited training data, while the high number of implants misclassified as cephalomedullary nail (long) suggests that the model was potentially biased towards the cephalomedullary nail (long) class.

Based on the performances of the models that were evaluated, the proposed deep learning model for automating trauma implant localisation and classification is a combination of the YOLOv5s object detection model with the DenseNet121 classification model into a single pipeline. It was noted that the classification performance of the pipeline was only marginally lower than that of the DenseNet121 classification model when provided with input that was segmented using the ground truth bounding boxes. This indicated that the pipeline was successful in localising the implants and provided accurate input to the classification model.

Table 8.1 compares the proposed deep learning model with previous studies employing similar approaches. The table illustrates that the performance of the proposed deep learning model was notably lower when compared to the results of the previous studies. However, the performance of the proposed model can be deemed respectable, especially considering the substantial difference in the number of images utilised per study. The

satisfactory performance of the model, given the very limited data, highlighted the effectiveness of techniques employed to address limited data and skewed class distributions, namely image augmentation, class weight adjustment, and transfer learning.

Table 8.1: Comparison with previous studies.

Application	Model	#Images	#Classes	Classification Scores
Localisation and classification of cervical spine hardware [9]	EfficeintDet + DenseNet121	10589	10	F1-score=96.1%; Recall=95.7%
Detection and classification of implanted electronic devices [54]	faster R-CNN + Inception-v3	4871	9	Accuracy=98.9%
Localisation and classification of trauma implants in leg x-rays (This study)	YOLOv5s + DenseNet121	204	5 (IM_nail classes)	Accuracy =73.7%; F1-score=74.0%; Recall=73.7%

Image augmentation was utilised to artificially increase the training set size as well as to balance the class distribution by upsampling. In addition, class weights were utilised in the training of the classification model to account for any class imbalance. The use of transfer learning allowed for more efficient model training by relying on knowledge gained from other datasets, such as ImageNet [7] and COCO [27].

Pre-operative identification of failed trauma implants is a time consuming and often challenging task for many surgeons, therefore highlighting the necessity for a product capable of efficiently and accurately classifying trauma implants with a high level of detail. This study verifies the concept of automating trauma implant localisation and classification using deep learning and in addition provides a deep learning approach that can act as a stepping stone for future research.

This study had several limitations, with the main limitation being the limited size of the dataset utilised. The limited dataset did not represent the full trauma implant population and contributed to some of the models overfitting. The shortage of data also meant that the implant detection had to be limited to four categories of implants,

while the implant classification was limited to five classes of a single implant category. Another limitation of the study was that the classification labeling was conducted by a single orthopedic surgeon. Thus, the accuracy of the classification model was limited to the expertise of the orthopedic surgeon and could not surpass it. Due to time constraints, the study could also only evaluate three different object detection models and four different classification models. Lastly, many of the default model hyper-parameters were selected during the evaluation of the models. Therefore, the performance of each model could likely be improved by implementing hyper-parameter tuning.

Recommendations for future research include using a larger dataset consisting of data collected from multiple institutions. This will mitigate sampling bias and allow for improved model generalisation. Furthermore, it is recommended that data labelling also be based on surgical notes and documentation. Due to the close proximity and varying orientations of the trauma implants, future research is also recommended to employ oriented object detection [8, 56]. Oriented object detection is not restricted to axis-aligned bounding boxes, thereby allowing for enhanced bounding box accuracy in datasets that contain objects with varying orientations. It was also noted that the intramedullary nail classes refer to the anatomical position of the implant. Thus, future research should consider utilising the object detection model to also localise the anatomical regions of the implants, which can then be used to enhance the classification accuracy.

8.1 Summary

Chapter 8 summarised the findings of the investigation and delved into various factors that might have impacted the results. The results of the study were then compared to the results of previous studies that employed similar approaches. Lastly, the limitations related to the investigation as well as recommendations for future studies were discussed.

Chapter 9

Conclusions

In conclusion, this study explored the application of various deep learning models for identifying trauma implants in leg X-rays, with the aim of finding an optimal deep learning solution. The main challenges associated with the research problem included constrained data availability, imbalanced class distributions, and the potential presence of multiple implants within images. The outcome of the investigation was a deep learning pipeline capable of localising four categories of trauma implants (plates, screws, intramedullary nails, and pins) with a mean average precision (intersection of union threshold of 0.5) of 0.967 and an average F1-score (intersection of union threshold of 0.5) of 95.5%. In addition to the implant localisation, the pipeline demonstrated the ability to classify the localised intramedullary nails into five classes, achieving an average accuracy of 73.7% and an F1-score of 74.0%. The proposed multi-model pipeline employed a YOLOv5s model to localise implants, which then fed into a DenseNet121 implant classification model.

Despite severe data limitations, the deep learning pipeline achieved a satisfactory performance, affirming the feasibility of employing deep learning models for automating trauma implant identification in leg X-rays. Furthermore, the study presented a deep learning approach that can be employed in future research. The next phase in creating a deep learning product for automating trauma implant identification will involve training and testing the proposed pipeline on a significantly larger dataset. Employing a larger dataset will facilitate more detailed classifications of trauma implants and will enhance the pipeline's ability to generalise to unseen data.

References

[1] Abraham, A. (2005). Artificial neural networks. In *Handbook of Measuring System Design*, John Wiley & Sons. 15

[2] Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaria, J., Fadhel, M.A., Al-Amidie, M. & Farhan, L. (2021). Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, **8(53)**, pp. 1–74. 23, 29

[3] Belete, S.C., Batta, V. & Kunz, H. (2021). Automated classification of total knee replacement prosthesis on plain film radiograph using a deep convolutional neural network. *Informatics in Medicine Unlocked*, **25**, pp. 100669. 45, 50, 51, 52

[4] Browner, B., Jupiter, J., Krettek, C. & Anderson, P. (2014). *Skeletal Trauma*. Elsevier Health Sciences. 9

[5] Chollet, F. (2021). *Deep Learning with Python, Second Edition*. Manning. 28, 29

[6] Cronier, P., Pietu, G., Dujardin, C., Bigorre, N., Ducellier, F. & Gerard, R. (2010). The concept of locking plates. *Orthopaedics & Traumatology: Surgery & Research*, **96(4)**, pp. S17–S36. 8

[7] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, IEEE. 29, 80

[8] Ding, J., Xue, N., Long, Y., Xia, G.S. & Lu, Q. (2019). Learning RoI transformer for oriented object detection in aerial images. In *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition, pp. 2849–2858, IEEE. 81

[9] Dutt, R., Mendonca, D., Phen, H.M., Broida, S., Ghassemi, M., Gichoya, J., Banerjee, I., Yoon, T. & Trivedi, H. (2022). Automatic localization and brand detection of cervical spine hardware on radiographs using weakly supervised machine learning. *Radiology: Artificial Intelligence*, **4(2)**, pp. e210099. viii, 45, 46, 51, 52, 80

[10] Friis, E. (2017). *Mechanical testing of orthopaedic implants*. Woodhead Publishing. 7, 8

[11] Gavali, P. & Banu, J.S. (2019). Chapter 6 - deep convolutional neural network for image classification on cuda platform. In *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, pp. 99–122, Academic Press. 27

[12] Girshick, R. (2015). Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1440–1448. 38, 39

[13] Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587. 38

[14] Glaudemans, A.W.J.M., Dierckx, R.A.J.O., Gielen, J.L.M.A., Zwerver, J.H. & Zwerver, J. (2015). *Nuclear Medicine and Radiologic Imaging in Sports Injuries*. Springer Berlin / Heidelberg, Berlin, Heidelberg. 10

[15] Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. MIT Press. viii, 13, 14, 15, 21, 22, 28

[16] He, F., Liu, T. & Tao, D. (2019). Control batch size and learning rate to generalize well: theoretical and empirical evidence. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 1143–1152. 25

[17] He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, IEEE. viii, 33, 34

[18] He, K., Lu, Y. & Sclaroff, S. (2018). Local descriptors optimized for average precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 596–605, IEEE. 31

[19] Hossin, M. & Sulaiman, M.N. (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, **5(2)**, pp. 1. 30

[20] Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K.Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, IEEE. 36

[21] Huang, J. & Ling, C.X. (2005). Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, **17(3)**, pp. 299–310. 30

[22] Jin, W. & Chu, P.K. (2019). Orthopedic implants. In *Encyclopedia of Biomedical Engineering*, pp. 425–439, Elsevier, Oxford. 7

[23] Kingma, D. & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*. 21, 22

[24] Krogh, A. (2008). What are artificial neural networks? *Nature Biotechnology*, **26(2)**, pp. 195–197. 14

[25] Lee, D.W., Lee, S., Ko, S., Jo, C., Park, J., Choi, B.S., Krych, A.J., Pareek, A., Han, H.S., Ro, D.H. *et al.* (2022). Automated detection of surgical implants on plain knee radiographs using a deep learning algorithm. *Medicina*, **58(11)**, pp. 1677. 45, 49, 51, 52

[26] Li, Z., Liu, F., Yang, W., Peng, S. & Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, **33(12)**, pp. 6999–7019. 12, 15, 17, 20, 21, 22, 23, 24, 25, 26

[27] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. & Zitnick, C.L. (2014). Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision*, pp. 740–755, Springer. 29, 80

[28] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B. & Belongie, S. (2017). Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2117–2125. viii, 40, 41

[29] Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., van der Laak, J.A., van Ginneken, B. & Sánchez, C.I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, **42**, pp. 60–88. 12

[30] Loussaief, S. & Abdelkrim, A. (2018). Convolutional neural network hyperparameters optimization based on genetic algorithms. *International Journal of Advanced Computer Science and Applications*, **9(10)**. 16, 17

[31] Mettler, F.A. & Mettler, J., Fred A. (2013). *Essentials of Radiology*. Elsevier - Health Sciences Division, 3rd edition. 10

[32] Mustapha, A., Mohamed, L. & Ali, K. (2021). Comparative study of optimization techniques in deep learning: Application in the ophthalmology field. *Journal of Physics: Conference Series*, **1743(1)**, pp. 012002. 21

[33] Nielsen, M.A. (2015). *Neural networks and deep learning*. Determination Press San Francisco, CA, USA. 14, 15, 18, 26

[34] Patterson, J. (2017). *Deep learning : a practitioner's approach*. O'Reilly Media, Inc., Sebastopol, CA. 14, 18, 20, 24, 25, 28, 29

[35] Powers, D.M. (2011). Evaluation: From precision, recall and f-factor to ROC, informedness, markedness correlation. *Journal of Machine Learning Technologies*, **2(1)**, pp. 37–63. 29

[36] Rasamoelina, A.D., Adjailia, F. & Sinčák, P. (2020). A review of activation function for artificial neural network. In *Proceedings of the IEEE 18th World Symposium on Applied Machine Intelligence and Informatics*, pp. 281–286. 22, 23

[37] Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, IEEE. viii, 40, 41

[38] Ren, S., He, K., Girshick, R. & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **39(6)**, pp. 1137–1149. viii, 38, 39, 48

[39] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I. & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 658–666. 30, 31

[40] Rogers, L.F. (2014). *Imaging skeletal trauma, 4th edition*. Elsevier Health Sciences, London. 10

[41] Sahiner, B., Pezeshk, A., Hadjiiski, L.M., Wang, X., Drukker, K., Cha, K.H., Summers, R.M. & Giger, M.L. (2019). Deep learning in medical imaging and radiation therapy. *Medical Physics*, **46(1)**, pp. e1–e36. 12

[42] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. & Chen, L.C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520. 37

[43] Shorten, C. & Khoshgoftaar, T.M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, **6(1)**, pp. 1–48. 27, 28

[44] Simonyan, K. & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations*, pp. 1–14, Computational and Biological Learning Society. 34

[45] Singh, B.K., Verma, K. & Thoke, A. (2015). Investigations on impact of feature normalization techniques on classifier's performance in breast tumor classification. *International Journal of Computer Applications*, **116(19)**, pp. 11–15. 27

[46] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, **15(1)**, pp. 1929–1958. 25

[47] Streukens, S. & Leroi-Werelds, S. (2016). Bootstrapping and pls-sem: A step-by-step guide to get more out of your bootstrap results. *European Management Journal*, **34(6)**, pp. 618–632. 28

[48] Sultana, F., Sufian, A. & Dutta, P. (2018). Advancements in image classification using convolutional neural network. In *Proceedings of the Fourth International Conference on Research in Computational Intelligence and Communication Networks*, pp. 122–129, IEEE. viii, 16, 17

[49] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9. 34

[50] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826. viii, 34, 35

[51] Tan, M. & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*, pp. 6105–6114. x, 36, 37

[52] Tan, M., Pang, R. & Le, Q.V. (2020). Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10781–10790. viii, x, 42, 43, 45

[53] Thakur, A. (2015). *Elements of Fracture Fixation*. Elsevier India. viii, 7, 8, 9

[54] White, R.D., Demirer, M., Gupta, V., Sebro, R.A., Kusumoto, F.M. & Erdal, B.S. (2022). Pre-deployment assessment of an AI model to assist radiologists in chest x-ray detection and identification of lead-less implanted electronic devices for pre-MRI safety screening: realized implementation needs and proposed operational solutions. *Journal of Medical Imaging*, **9(5)**, pp. 054504. 45, 47, 51, 52, 80

[55] Wilson, N.A., Jehn, M., York, S. & Davis, C.M. (2014). Revision total hip and knee arthroplasty implant identification: Implications for use of unique device identification 2012 AAHKS member survey results. *The Journal of Arthroplasty*, **29(2)**, pp. 251–255. viii, 1, 2, 8

[56] Xia, G.S., Bai, X., Ding, J., Zhu, Z., Belongie, S., Luo, J., Datcu, M., Pelillo, M. & Zhang, L. (2018). DOTA: A large-scale dataset for object detection in aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3974–3983. 81

[57] Zhang, A., Lipton, Z.C., Li, M. & Smola, A.J. (2023). *Dive into deep learning*. Cambridge University Press. 18