Fully automated hand tracking for Parkinson's Disease Diagnosis

Callum Macpherson
Student ID: 201022895

Supervised by Luisa Cutillo, Samuel Relton, and Hui Fang

Submitted in accordance with the requirements for the
module MATH5872M: Dissertation in Data Science and Analytics
as part of the degree of

Master of Science in Data Science and Analytics

The University of Leeds, School of Mathematics

September 2021

The candidate confirms that the work submitted is his/her own and that appropriate credit has been given where reference has been made to the work of others.

Acknowledgements

I would like to thank my supervisors, Samuel Relton, Luisa Cutillo, and Hui Fang, for their guidance and support over the last three months. Their support and enthusiasm have been excellent. Furthermore, I would like to thank five additional MSc students on this course who assisted with labelling the data discussed in Section 4.4.1: Hanhu Hong, Jiachen Bai, Kunhao Liang, and Siyu An.

I would also like to thank Luisa Cutillo, acting as programme leader and the assessors of this module, for granting formal permission for the report's length to exceed 60 pages. Additional pages were required as this project used image data, which required additional pages to visualise data quality issues. Furthermore, to ensure that the study is well explained and repeatable, additional pages were needed to explain how the multiple data sets were created. Lastly, as a multi-stage machine learning pipeline has been proposed, the various algorithms considered needed to be reviewed and explained in a sufficient level of detail. There are a total of 75 pages in this report, including 4 blank pages between chapters.

Abstract

Background

The global prevalence of Parkinson's is on the rise. There is a global shortage of trained neurologists who can effectively diagnose the disease, requiring an objective tool for diagnosis. Further, the need for remote and objective assessment of the underlying symptoms of Parkinson's disease has been exacerbated since the COVID-19 pandemic. State-of-the-art methods have been demonstrated to diagnose Parkinson's Disease using videos of finger tapping in conjunction with a pose estimation tool named DeepLabCut. However, currently, a user needs to manually label 150 video frames before use which serves as a barrier for clinical deployment.

Methods

An automatic labelling system is proposed in this study. This solution consists of a machine learning pipeline that utilises two existing state-of-the-art algorithms which are combined. Firstly, an object detection model is used that operates on a 1080p image and locates the position of a hand using a bounding box. Secondly, a key point regression model operates on a cropped image using the bounding box coordinates and outputs (x,y) coordinates for the tip of the index finger and thumb. By reviewing existing state-of-the-art algorithms, the selected models for the object detection model were YOLOv4 and YOLOv4-tiny. Similarly, the selected feature extractors selected for the key point regression model were EfficientNet-B0 and EfficientNet-B4.

Results

Both YOLOv4 and YOLOv4-tiny achieve a mean average precision (mAP) of $\sim$ 100% and mean intersection over union (IOU) of $\sim$ 85%. YOLOv4-tiny was the preferred object detection model with a detection speed of 330 FPS which is 6.68$\times$ faster than YOLOv4. The EfficientNet-B0 key point regression model achieved a mean pixel error of 13.41 on validation data with an inference speed of 439 FPS. The EfficientNet-B4 model achieved a mean pixel error of 9.95 on validation data with an inference speed of 19 FPS. The preferred feature extractor for the key point regression model was EfficientNet-B4 as it produced a lower pixel error.

Conclusion

The final automatic labelling system pipeline for this report uses YOLOv4-tiny for object detection and an EfficientNet-B4 backbone pre-trained on ImageNet with a customised top for key point regression to predict coordinates for the tip of the index finger and thumb. The final mean pixel error from the pipeline is 9.95 px which is excellent considering that DeepLabCut achieves 8.39 px on the same data set. Most of the pixel error caused by the pipeline produced in this report is likely due to human variability in labelling and scaling down input images.

Contents

1 Introduction **1**
- 1.1 Existing Literature . . . 3
- 1.2 Aims and Objectives . . . 4

2 Technical Overview **7**
- 2.1 Convolutional Neural Networks . . . 8
  - 2.1.1 Convolution Layers . . . 9
  - 2.1.2 Pooling Layers . . . 11
  - 2.1.3 Fully Connected Layers . . . 12
  - 2.1.4 CNN for multi-class classification example . . . 13
  - 2.1.5 Activation Functions . . . 14
- 2.2 Loss Functions . . . 16
  - 2.2.1 Mean Square Error . . . 17
  - 2.2.2 Intersection over Union (IOU) . . . 17
  - 2.2.3 Mean average Precision (mAP) . . . 18
- 2.3 Optimization . . . 19
  - 2.3.1 Generalization Errors . . . 20
  - 2.3.2 Early stopping . . . 21

3 Computer Vision: Review of methods **23**
- 3.1 Pose Estimation . . . 23
  - 3.1.1 Data sets and Data Augmentation . . . 24
  - 3.1.2 Model Architectures . . . 26
  - 3.1.3 Loss Functions . . . 28
  - 3.1.4 Optimization Algorithms . . . 28
  - 3.1.5 Chosen method for this study . . . 29
- 3.2 A brief history of encoder network architectures . . . 29
  - 3.2.1 LeNet . . . 29
  - 3.2.2 AlexNet . . . 30
  - 3.2.3 VGG-16 . . . 30
  - 3.2.4 ResNet . . . 31
  - 3.2.5 EfficientNet . . . 32
- 3.3 Object Detection Algorithms . . . 34
  - 3.3.1 Two-stage detection models . . . 35
  - 3.3.2 One-stage detection models . . . 36

4 Methods **41**
4.1 Automatic Labelling System: Pipeline . . . 42
4.2 Requirements . . . 44
4.3 Data Collection . . . 45
4.4 Data Preparation . . . 46
4.4.1 Labelling key points for DLC . . . 46
4.4.2 Pitfalls of labelling approach and data quality issues . . . 47
4.4.3 Converting DLC Keypoints to Bounding Box Labels . . . 51
4.4.4 Converting DLC Keypoints to data set for Key Point Regression model 56
4.5 Bounding Box models . . . 59
4.5.1 Environment . . . 60
4.5.2 YOLOv4: Network Architecture . . . 60
4.5.3 YOLOv4-tiny: Network Architecture . . . 61
4.5.4 Training . . . 61
4.5.5 Evaluation . . . 61
4.6 Key point regression models . . . 62
4.6.1 Environment . . . 62
4.6.2 EfficientNet-B0: Network Architecture . . . 63
4.6.3 EfficientNet-B4: Network Architecture . . . 64
4.6.4 Training . . . 65
4.6.5 Evaluation . . . 65

5 Results **67**
5.1 Object Detection models for bounding box coordinates . . . 67
5.2 Key Point Regression Models . . . 69
5.2.1 Discussion . . . 71

6 Conclusion **73**
6.1 Limitations of Study and Suggestions for Future Work . . . 73
6.2 Conclusion . . . 74

A Appendix **77**
A.1 YOLOv4 and YOLOv4-tiny configuration instructions . . . 77
A.2 EfficientNet-B4 Key point regression model predictions . . . 78

List of Figures

2.1 A basic CNN architecture 8
2.2 The operations of a convolutional layer. The computation on the RHS of the convoluted feature details the element-wise multiplication between the input data and the kernel, of which the summation is taken to produce the convoluted feature (Yin 2018). 9
2.3 How filters learn visual features from an input image to store in feature maps (DeepLizard 2021). 10
2.4 Feature maps at different network depths (Dertat 2017) 11
2.5 Max pooling operation example (Dertat 2017) 12
2.6 CNN example for mutli-class classification (Swapna 2021) 13
2.7 The sigmoid function and its derivative 14
2.8 Common activation functions (Kızrak 2020) 16
2.9 Intersection over Union (IOU) 17
2.10 Model Fit: Underfitting, Balanced, Overfitting (Bhande 2018) 20
2.11 Early Stopping Point (PapersWithCode n.d.) 21

3.1 Image Classification on ImageNet, (*PaperswithCode* 2021*a*) 25
3.2 LeNet and AlexNet network architectures 30
3.3 The residual block (He et al. 2016) 32
3.4 EfficientNet performance on ImageNet (Tan & Le 2019). 33
3.5 YOLO Method (Redmon et al. 2016). 37

4.1 Current and proposed DeepLabCut Workflow 41
4.2 Proposed ML Pipeline for the Automatic Labelling System 43
4.3 Workflow of this project 44
4.4 Hand Labelling Protocol 47
4.5 Human variability in labelling (Mathis et al. 2018) 48
4.6 Data quality issues. Left: Soft Tissue Artefact. Right: Self-occlusion. 49
4.7 Additional data quality issues. Left: Blur. Right: Motion Blur. 50
4.8 Frequency distribution of Euclidean distance between index finger and thumb . 51
4.9 Different proportion of original image occupied by hand: Left: Large hand relative to 1080p image space. Right: Small hand relative to image space. 52
4.10 Example bounding box ground truth annotation '.txt' file 53
4.11 Defining bounding box dimensions using existing DLC key point labels 54
4.12 Pandas dataframe named 'yolo_df' containing formatted bounding box values . 55
4.13 Issue caused by 768×768 bounding box: Index finger outside of box 56
4.14 Converting key points coordinates from original image to cropped image 58
4.15 New coordinates relative to origin of cropped image 59

5.1 Object Detection Models: Training loss and validation mAP. Left: YOLOv4-tiny. Right: YOLOv4. 68
5.2 Predictions made by object detection models. Left: YOLOv4-tiny. Right: YOLOv4. 69
5.3 Key Point Regression Models: Training loss and validation loss. Left: EfficientNet-B0. Right: EfficientNet-B4. 70

A.1 EfficientNet-B4 Key point regression model: Predictions on Training Data . . . 78
A.2 EfficientNet-B4 Key point regression model: Predictions on Test Data 79

List of Tables

1.1 Summary of the UPDRS Item 3.4 (Finger Tapping) rating scale (Goetz et al. 2008) 2

3.1 Comparing ResNet and Darknet Variants (Redmon & Farhadi 2018) 39

4.1 Hand labelling protocol. Each key point is visualised in Figure 4.4. 47
4.2 Bounding box ground truth annotation format 53
4.3 Comparison of object detection models used in this study 61
4.4 EfficientNet-B0 Network Architecture with custom top for regression. 64
4.5 EfficientNet-B4 Network Architecture with custom top for regression. 65

Chapter 1

Introduction

Parkinson's Disease (PD) is a progressive, complex neurodegenerative disorder that inhibits movement. Specifically, it is characterized by several primary motor symptoms, including bradykinesia (slowness of movement), muscle rigidity, a resting tremor, postural instability. While the exact cause of the disease is typically unknown, it can be caused by carbon monoxide poisoning, strokes or even drug-induced. The disease is a result of genetic mutations that cause brain cell degeneration, primarily in the midbrain region (Miller 2002). Approximately 10 million patients are affected by PD globally, and that the prevalence of this disease is increasing, resulting in an increased demand for trained neurologists to carry out ongoing motor assessments of patients (Zhao et al. 2020, Shahid & Singh 2020). Additionally, there is a global shortage of trained neurologists, resulting in a decreased supply of clinicians to accurately diagnose PD. Furthermore, since the COVID-19 pandemic, many patients have been unable to carry out routine in-person assessments as PD patients are typically older and considered 'high risk' and have been formally advised to shield (Sibley et al. 2021). Therefore, there is an urgent need to develop new tools to objectively measure PD to enable effective widespread diagnosis, monitoring and future research into the disease (Zhao et al. 2020).

For diagnosing PD, Williams et al. (2020) explains that a principal method used in current clinical practice involves trained neurologists visually observing patients performing a tapping exercise with their index finger and thumb. This method assesses for bradykinesia, which is a cardinal motor feature of the condition that is associated with the slowness of movement, decreased size and speed of movement, or sudden halts as movements are continued (Williams et al. 2020). A patient's ability to perform this tapping exercise is assessed using two validated clinical rating scales: Item 3.4 of the Unified Parkinson's Disease Rating Scale (UPDRS) and the Modified Bradykinesia Rating Scale (MBRS). The UPDRS classifies the severity of PD based on the criteria given in Table 1.1, considering finger tapping speeds, amplitude, and rhythm simultaneously, resulting in a single score from 0 (no PD) to 4 (severe PD). On the other hand, the MBRS is made up of three separate scores for speed, amplitude and rhythm (Williams et al. 2020).

Score	Description
0 - Normal	No problems
1 - Slight	Any of the following: (a) Regular rhythm broken with 1-2 interruptions, (b) slight slowing in tapping speed, (c) amplitude decreases towards end of exercise.
2 - Mild	Any of the following: (a) Regular rhythm broken with 3-5 interruptions, (b) mild slowing in tapping speed, (c) amplitude decreases midway through exercise.
3 - Moderate	Any of the following: (a) Regular rhythm broken with 6+ interruptions, (b) moderate slowing in tapping speed, (c) amplitude decreases from start.
4 - Severe	Cannot perform the exercise due to slowing, interruptions or decrements.

Table 1.1: Summary of the UPDRS Item 3.4 (Finger Tapping) rating scale (Goetz et al. 2008)

Zhao et al. (2020) highlights that this method is inherently subjective as the distance between the index finger and thumb is not measured and is therefore not objective. Further, due to the lack of objective measures in classifying patients severity of PD, there are high levels of inter-variability between clinicians (Goetz et al. 2008, Heldman et al. 2011). Several studies have been published which objectively measure bradykinesia from finger-tapping exercises. However, these include the use of wearable equipment such as gyroscopes (Heldman et al. 2011), electromagnetic sensors (Sano et al. 2016), infrared camera markers (Ržička et al. 2016), or patient interaction with an app (Lee et al. 2016). However, the use of wearable sensors can hinder a patients ability to perform the exercise, serving as a barrier for clinical deployment (Zhao et al. 2020).

Modern advances in a subset of machine learning (ML) called deep learning have facilitated many new technologies within the research area of image analysis, such as image classification, object detection and pose estimation using convolutional neural networks (CNNs). These technologies have been applied to many fields of medical imaging, such as diagnostics in dermatology, radiology, and pathology (Esteva et al. 2019). Existing studies have shown that deep learning tools can be used as a contactless method to quantify measures related to Parkinson's bradykinesia. These will be explored in the following section.

1.1 Existing Literature

Four recent studies have shown that propose contactless methods to measure PD bradykinesia in finger tapping exercises objectively. The study looked at finger tapping for 13 patients with advanced PD and 6 controls that simultaneously tracks both the left and right hand using a pipeline that is dependent on a face detection system (Khan et al. 2014). A drawback of this approach is that the face detection stage in the ML pipeline is not essential to tracking hands, as demonstrated by later papers. Furthermore, tracking both hands simultaneously may prevent patients from monitoring their own hands if the system were to be deployed for private use. If the pipeline only tracked one hand, then a patient could hold their phone with their other hand, which would not be possible with this system.

The second recent study implemented a computer vision approach that uses segmented images to produce an optic flow field that identifies areas of the hand that are moving and can extract key point features that are clinically relevant such as the tip of the index finger and thumb (Williams et al. 2020). This method improves the previous method as it is more practical for self-monitoring as it only detects and tracks key points. Furthermore, the results in the study can be considered more robust as they validate the model on a larger cohort. One potential limitation of this method is that it requires that neurologists have a background in programming to implement. The method suggested in the following paragraph does not require a background in programming to output objectively measured hand tracking.

The third study considered finger tapping, as well as hand clasping and hand pro/supination (twisting at the wrists) (Liu et al. 2019). For finger tapping bradykinesia, the study reported strong correlations between the MDS-UPRDS and video measures of tapping frequency (r=0.91), frequency variation (r=0.82), and amplitude (r=-0.94). On the other hand, the study reported a weak correlation between MDS-UPDRS and video measure for amplitude variation (r=0.39). Limitations of this study are that only two trained clinicians gave assessments that were used as 'ground truth' for comparison of their computer vision methods. Therefore the ground truth comparison is less robust than other more recent studies. Furthermore, 24 FPS videos were used that resulted in motion blur, causing issues in tracking the hand. Also, there were no MBRS ratings used for comparisons, meaning that a cornerstone of PD bradykinesia assessment has not been considered (Zhao et al. 2020).

Lastly, the fourth study to consider finger tapping uses deep learning-based pose estimation technologies such as DeepLabCut as a contactless method to track key points such as fingertips. This study used 137 videos of hands (77 PD and 60 control) with 20 frames of each video had key points labelled such as the tip of the index finger and thumb. Therefore, approximately 2,740 labelled frames were required to train the pose estimation model in DeepLabCut, which achieved a mean absolute error of 8.39 pixels for 1080p videos. A total of 22 trained movement

disorder neurologists made clinical assessments on all participants based on the MBRS and MDS-UPDRS, meaning that more robust 'ground truth' observations existed, and considering both standard assessment scales, which other studies had not achieved. The study reported good Spearman's correlation with clinical ratings -0.74 for tapping speed, 0.66 for amplitude, and -0.65 for rhythm on the MBRS, and -0.69 combined for MDS-UPDRS. A further key benefit of this study compared to others is that it uses DeepLabCut, an open-source pose estimation software that uses a GUI without requiring users to produce sophisticated code. Currently, this makes this method widely available to neurologists that do not have a background in programming. However, a user currently needs to manually label the index finger and thumb in video frames before use, which serves as a bottleneck and a barrier to deployment. This paper suggests that 2,740 frames were labelled of 6 key points, which likely took several hours. However, the developers of DeepLabCut have demonstrated that DeepLabCut only requires less than 150 frames of human labelled data to track 3D hand movements of a mouse (Nath et al. 2019). Nonetheless, the labelling stage of the DeepLabCut workflow serves as a significant barrier to deployment in clinical practice.

A more recent study has measured bradykinesia in facial expressions using Google's pose estimation package called 'MediaPipe'. This study was conducted by Gomez et al. (2021) The facial expressions that patients were asked to perform were: Happy, Wink, Angry and Surprised. Static and dynamic facial expressions were tracked using 468 3D facial landmarks to detect hypomimia, which reduces facial expressiveness caused by bradykinesia. There were a total of 54 participants in the study, both PD and non-PD participants, and the visual landmark data was classified automatically using a support vector machine (SVM) in which a maximum classification accuracy of 77.36% was achieved with the 'Wink' gesture (Gomez et al. 2021).

1.2 Aims and Objectives

This project aims to automate the labelling stage of the DeepLabCut workflow for Parkinson's Disease Diagnosis. An automatic labelling tool will be designed and tested, which takes an image of a hand as input and output labels for the coordinates of the tip of the index finger and thumb.

By producing an automatic labelling system, clinicians will no longer need to manually label 150+ frames before using DeepLabCut for tracking the index finger and thumb for quantifying Parkinson's bradykinesia. Manual labelling serves as a barrier to the widespread clinical deployment of DeepLabCut. Producing an automatic labelling system may result in the deployment of a quantifiable, deep-learning-based system for objectively measuring Parkinson's bradykinesia. Current clinical methods involve visually observing patients performing the exercise, which is therefore not quantified and highly subjective. Different clinicians will often classify the same

patient with a different level of severity of Parkinson's Disease. Thus, producing a quantifiable and objective diagnosis method should result in higher quality diagnosis and treatment plan.

The proposed automatic labelling system will be underpinned by deep-learning-based computer vision technology. Existing state-of-the-art open-source computer vision technologies will be reviewed to assess the feasibility of implementation for use in the proposed automatic labelling system. This will include considering existing pose estimation and object detection algorithms and reviewing modern convolutional neural networks as potential feature extractors for the proposed system. Once appropriate algorithms have been identified and selected, the proposed system will be outlined. The following objectives need to be met to meet the aims of this project:

1. **To give a technical overview of the relevant deep-learning-based computer vision algorithms.** An overview of convolutional neural networks will be provided, including the underpinning core operations, constituent layers, and key terminology.

2. **Review existing algorithms for the proposed use in the automatic labelling system.** A review on markerless motion capture will be produced with a specific focus on how deep learning systems are used in pose estimation. Existing methods to estimate key-point coordinates from image data will be identified and discussed, with a method selected for use in this study. This will be followed by a review on influential convolutional neural networks and design choices that have improved their performance in recent years. Existing state-of-the-art convolutional neural networks will be selected for use in the automatic labelling system for this project. This will be followed by a review of object detection algorithms used as part of the automatic labelling system. A method to estimate key-point coordinates from image data, along with two convolutional neural networks and object detection algorithms, will be selected once this objective is completed.

3. **Outline Proposed System and Working Methods.** Once the preferred algorithms and technologies have been identified; the proposed automatic labelling system will be defined using flow diagrams. Furthermore, a workflow diagram will be produced for data preprocessing, modelling, and evaluation. Any data quality issues will be identified and discussed. Methods used to produce the automatic labelling system will be outlined to ensure that the study is repeatable.

4. **Compare both object detection algorithms and key-point estimation algorithms.** The proposed algorithms that have been used in this study will be compared using defined evaluation metrics. A preferred algorithm will be selected for object detection and key point estimation in the final automatic labelling system.

Chapter 2

Technical Overview

In 1959, the term Machine Learning was coined by Arthur Samuel, and it refers to the field of study that gives computers the ability to learn without being explicitly programmed (Samuel 1959). Interestingly, in many machine learning publications, people often cite this definition as a direct quote of Arthur Samuel despite it never appearing in any of his publications. As summarised by Jordan & Mitchell (2015), Machine Learning is one of the fasting growing fields in technology and is underpinned by both statistical and computing theory. Despite some of the key algorithmic theory being produced as early as 1943, there has been increasing and ongoing breakthroughs in the field of machine learning due to new learning algorithms, an exponential increase in the availability of open-source online data and low-cost computational power (McCulloch & Pitts 1943). Data-driven machine learning methods are being widely implemented in many industries such as science, finance, technology, retail, and health care resulting in evidence-driven decision making (Jordan & Mitchell 2015).

Ayodele (2010) highlights that Machine Learning can be categorised into three core areas; supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is where algorithms learn based on labelled training data to make predictions on test data. The labelled training data is an input with an example labelled output that the model has to learn the underlying relationship. For example, in image classification, a labelled data point would be an image of a dog along with the label 'dog' that a particular algorithm can learn to associate the visual features of the image with the given label to then make future predictions on unseen images. In unsupervised learning, certain algorithms are used to learn patterns in unstructured data. This differs from supervised learning as the data used in these tasks does not have labels. Examples of unsupervised learning are clustering algorithms such as k-means clustering. Reinforcement learning is when an algorithm learns a policy of how to behave based on observations of a given environment. Each possible action the agent can take within the environment will have a reward or punishment incurred by the agent, which guides the algorithm in learning how to interact with that environment. A common analogy for this is that when you are training a dog, you reward it with a treat if it behaves well. If it misbehaves, then you punish it by saying

'no' in a deep tone. This is the same logic that inspired reinforcement learning (Ayodele 2010).

This project aims to build a fully automated hand tracker. This will be achieved by training a given ML model or system on labelled training data. This model will take image frames of a video as input and output the coordinated (x,y) of the index finger and thumb position in image space. Therefore, this is a supervised learning task. An outline of the algorithms used in machine learning-based image analysis will be explained in this chapter.

2.1 Convolutional Neural Networks

A subset of machine learning known as deep learning has continued to outperform previous state-of-the-art machine learning models in several applications, particularly computer vision and natural language processing (Voulodimos et al. 2018). The term deep learning refers to artificial neural networks with multiple layers. Convolutional neural networks (CNN) are among the most popular in deep learning literature in the last decade (Albawi et al. 2017). The surge in developments within this subset of machine learning was a result of extensive, open-source, labelled data sets along with more widely available low-cost Graphics Processing Units (GPUs), allowing for parallel computational methods accelerating the process of training several millions of learnable parameters (Voulodimos et al. 2018).

The modern framework for the CNN was published in 1989, in which a deep learning system named 'LeNet-5' was used to classify handwritten digits (LeCun et al. 1989). Gu et al. (2018) explains that convolutional neural networks are inspired by biological visual perception mechanisms that are used by many creatures. CNN's mimic the behaviour of the visual cortex in regards to detecting visual information in specific receptive fields. A CNN is constructed from three primary neural layer categories: convolutional, pooling, and fully connected layers, as shown in Figure 2.1. The convolution and pooling layers perform mathematical operations on the input image to extract learnable visual features. These learned features are then passed onto fully connected layer(s), which can be used for classification or regression (Gu et al. 2018).

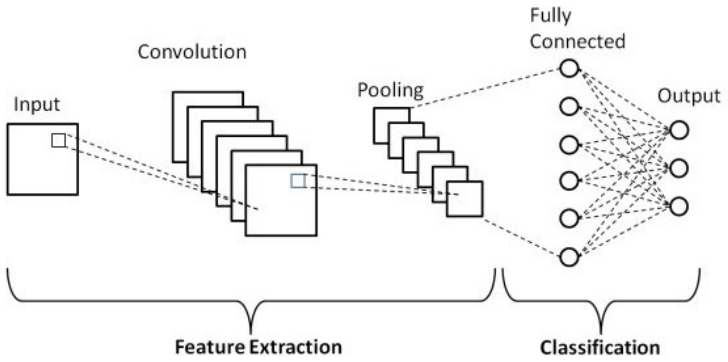

Figure 2.1: A basic CNN architecture

2.1.1 Convolution Layers

The objective of the convolutional layer is explained by Gu et al. (2018) as to learn feature representations of visual features that are present in the input image. This is achieved by passing filters (also commonly referred to as kernels) over the pixels present in the input image using a sliding window operation to perform summation of element-wise multiplication to produce the feature representations, commonly referred to as 'feature maps' or 'convoluted features' as shown in Figure 2.2 (Gu et al. 2018). Convolutional layers increase network efficiency when compared with fully connected layers by weight sharing. By applying the convolution operation, the same weights are used throughout all locations in an image which reduces computational overhead (LeCun et al. 2015).

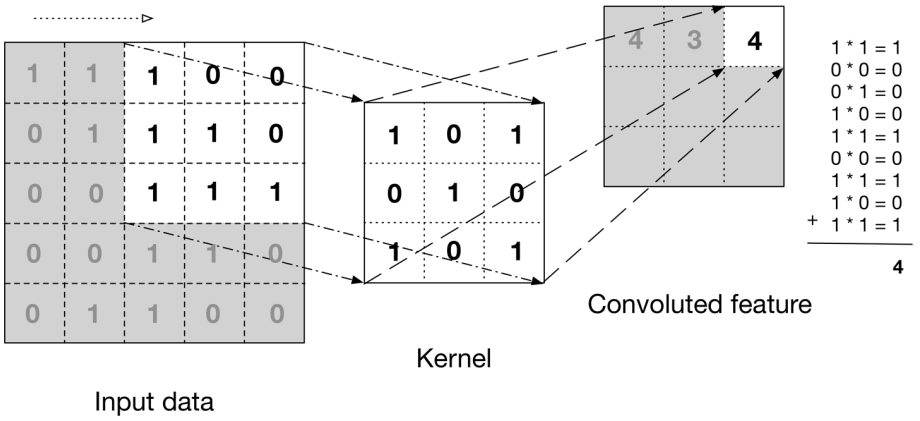

Figure 2.2: The operations of a convolutional layer. The computation on the RHS of the convoluted feature details the element-wise multiplication between the input data and the kernel, of which the summation is taken to produce the convoluted feature (Yin 2018).

Each layer contains several filters, and these contain the learnable parameters in convolutional layers, which are the weights of each filter. The early layers in the network learn simple patterns in the input image, such as edges, corners, and curves. The weights of the filters are learned through a process known as back-propagation, in which the weights of the filter are iteratively updated to minimise the difference between predicted outputs and ground truth outputs. In hidden layers, the filters are convolved over feature maps from the previous layer, and new feature maps are produced that are more abstract and complex visual representations of the original image (Voulodimos et al. 2018). In comparison to the earlier layers, hidden layers learn visual features that are more complex and often abstract when manually observing feature maps from deep hidden layers. The 'receptive field' is a window of pixels in the original input image that affect a particular pixel in a feature map (Gu et al. 2018).

The size of the filter used in each layer is a crucial hyperparameter in convolutional layers. The filter size determines the receptive field of the original image that the filter can observe. A

larger filter such as 7×7 will be passed over a greater region of the original input image and extract visual features that are typically larger and more general. A smaller filter will result in a smaller receptive field for a given feature map, learning local and more detailed visual features. Typically, smaller filters are used, such as a 3×3 filter, as most of the useful visual features in an image are small; therefore, it makes sense to observe small subsections of the image in detail. Many useful visual features, such as vertical or horizontal edges, will be present in an image several times. Therefore, a single horizontal edge detecting filter can be passed over an entire image extracting the location of all horizontal edges in the image. The presence, location and proximity of all learned visual features are then used in making final predictions of the considered output for the task at hand. Other core hyperparameters related to the convolutional layers include the number of filters applied in a given layer. Adding additional filters is known as making the network 'wider'. Additionally, another hyperparameter is the number of layers in the network, where adding more layers is referred to as making the network 'deeper'. Making a network wider or deeper increases the learning capacity of the network but can also make it prone to overfitting.

A brief example is shown in Figure 2.3 to provide further intuition to convolutional layers. This example is elementary and assumes that the filters' numerical weights have been learned without manual hand-tuning. See the first row of Figure 2.3, which shows a grey-scale hand-written digit which is from the MNIST data set (LeCun & Cortes 2010). The input has been repeated four times as four filters will be used on this data. Each input image has dimensions 28x28 for the height and width of the image. The second row denoted by 'Filter (numerical)' contains four 3x3 filters with weights that a CNN has learned. The numerical values have been represented visually in the third row in which 1 corresponds to white, 0 corresponds to grey, and -1 corresponds to black on a continuous grey scale.

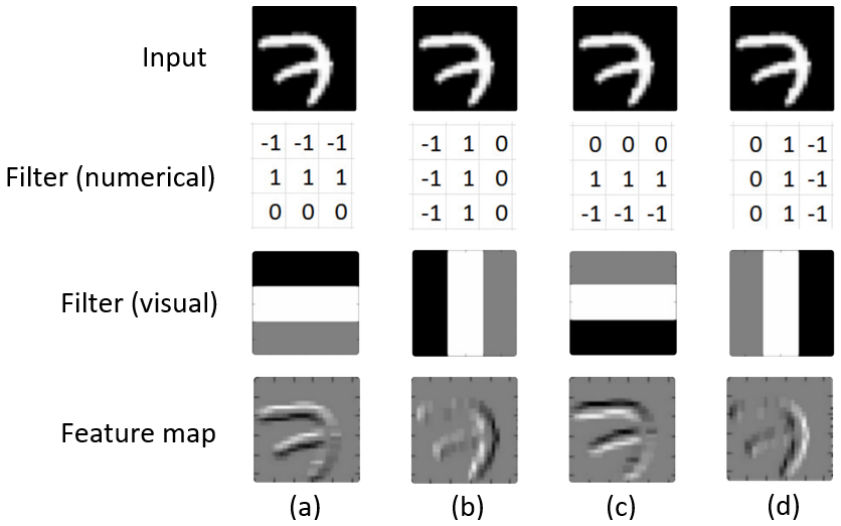

Figure 2.3: How filters learn visual features from an input image to store in feature maps (DeepLizard 2021).

The bottom row in Figure 2.3 shows the feature maps that are outputted as a result of passing each of the 3x3 filters over the 28x28 input image. Each of the feature maps clearly show the number 7 and are some form of a visual representation of the original input image. However, there are clear differences between each of the four feature maps due to being produced by different filters. We can see in the feature maps that all of the filters are detecting edges. In the feature maps, the brightest pixels can be considered the visual feature filtered and learned of the original input image. In feature map (a), we can see that the brightest pixels are in the top horizontal edges of the 7 in the input image, indicating that the filter associated with this feature map is a top horizontal edge detector. Feature map (b) shows the brightest pixels on the left vertical edges, and therefore the filter associated with this feature map is a left vertical edge detector. Similarly, the filter associated with feature map (c) detects horizontal bottom edges, and (d) detects right vertical edges. The filters given in this example are elementary and would likely be present in the early layers of a CNN. In reality, many different types of filters become increasingly more sophisticated and abstract in deeper hidden layers of a given CNN.

For example, see Figure 2.4, in which five feature maps are displayed produced from a CNN, analysing an image of a cat. As explained by Dertat (2017) the images are from different layers at increasing depth through the network. The left-hand side image is from the shallowest layer, whereas the far right image is a feature map from the deepest layer. It is clear that the deeper in the network that the feature map is extracted from, the less the feature map looks like the original image. For example, in the feature maps associated with block4 and block5, there is arguably no cat present in the image. Deeper feature maps encode higher-level visual features such as 'cat eyes' and 'cat nose' that are more sophisticated than general shape detectors which are present in the early layers of the network. Therefore, a helpful way to think about deeper feature maps is that they contain abstract representations of visual features associated with the class of the image rather than easily interpreted visual features. The information stored in the deeper feature maps is still highly useful in making predictions, but it is harder to visually interpret for humans (Dertat 2017).

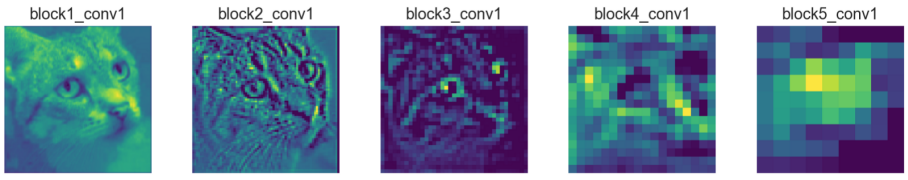

Figure 2.4: Feature maps at different network depths (Dertat 2017)

2.1.2 Pooling Layers

Typically, directly after convolutional layers, there are Pooling Layers. The objective of the pooling layers is explained by Voulodimos et al. (2018) is to downsample (or subsample) the input dimensions for the next convolutional layer. The depth of each feature map is not affected;

however, the height and width are reduced in dimension leading to an inherent loss of information. Interestingly, this loss of information is often advantageous as it prevents the network from overfitting the training data, increases the networks ability to generalise to unseen data, and reduces the computational and memory costs in later stages of the network. There are many common pooling layer variants. However, the most common are max and average pooling. It has been demonstrated in the literature that max-pooling can lead to faster convergence than average pooling and is also more beneficial in preventing overfitting in a CNN (Scherer et al. 2010). There are also additional variations of pooling layers that can be applied to different applications, such as stochastic pooling and spatial pyramid pooling, which is used in object detection (He et al. 2015).

See Figure 2.5 for an example of max-pooling in which the LHS 4×4 grid is a feature map outputted from the previous convolutional layer in a hypothetical network. The way that max-pooling works is simple, as this example uses a 2×2 window and a stride of 2, the feature map is split into a total of four 2×2 sub-grids indicated by the red, green, yellow, and blue boxes on the LHS of Figure 2.5. The maximum value of each sub-grid is taken and stored in the output of the pooling layer. For example, in the red sub-grid in the LHS of Figure 2.5 the values are 1, 1, 5, 6, of which the maximum value of this set is 6. Therefore the number 6 is displayed in red in the output matrix on the RHS of Figure 2.5.

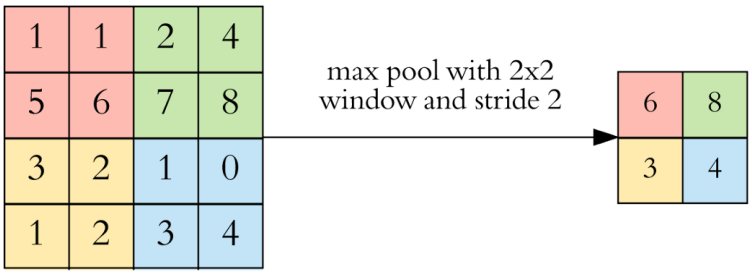

Figure 2.5: Max pooling operation example (Dertat 2017)

2.1.3 Fully Connected Layers

There are often several blocks of convolution and pooling layers; however, the amount of these blocks can vary between networks. Some networks have fully connected layers as the final layer for both regression and classification tasks. Voulodimos et al. (2018) highlights that fully connected networks provide high-level reasoning based on the learned visual features in the input image to make the final outputted prediction. Firstly, the multiple 2D feature maps are flattened into 1D feature vectors, which are then passed as inputs to the fully connected layer. Neurons in the fully connected layer are connected to every pixel (or neuron) in the feature map, which is why they are called fully connected layers. The inputted 1D feature vector is multiplied by the weights of the fully connected network, and a bias term is added. The purpose

of the bias term is to offset the activation function by translating the activation function to the left or right, which increases the learning capacity of the network. The weights and bias are the learnable parameters in these layers, which are updates iteratively through back-propagation. The output of a fully connected layer can be passed as the input for deeper fully connected layers, or it can be the final output of the model. If a fully connected layer is the final output layer of a given model, the number of neurons in the final layer equals the number of predictions the network will make. This makes the number of neurons in the final layer a crucial parameter when designing a network architecture (Voulodimos et al. 2018).

2.1.4 CNN for multi-class classification example

A brief explanation has been given on convolutional, pooling, and fully connected layers. A CNN used for multi-class classification is shown in Figure 2.6. An image of a zebra is inputted into the network, and the visual features of the image are extracted as it is passed through the three convolution and pooling layers. The final feature maps outputted by the last pooling layer is flattened to a 1D feature vector which is then passed into 4 fully connected layers. As this is a multi-class classification problem, the final fully connected layer is passed through a softmax activation function which outputs a final probabilistic distribution predicting which animal was in the input image. The final fully connected layer has 3 neurons indicated by the 3 dark blue dots in FIGURE, as this represents the number of pre-defined classes. As shown in the peach coloured box in Figure 2.6, the class 'zebra' was predicted with a probability of 0.7, 'horse' has an associated probability of 0.2, and dog has an associated probability of 0.1. As zebra has the highest probability, this is the final prediction outputted from the network. It is interesting to note at this stage that 'horse' has a slightly higher probability than the class dog, which is likely due to horses and zebras typically having more similar visual features than zebras and dogs. The visual features have been extracted in the feature extraction stage of the network. This network architecture is specifically for the multi-class classification of the three mentioned animals, and the architecture will differ depending on the application of the network.

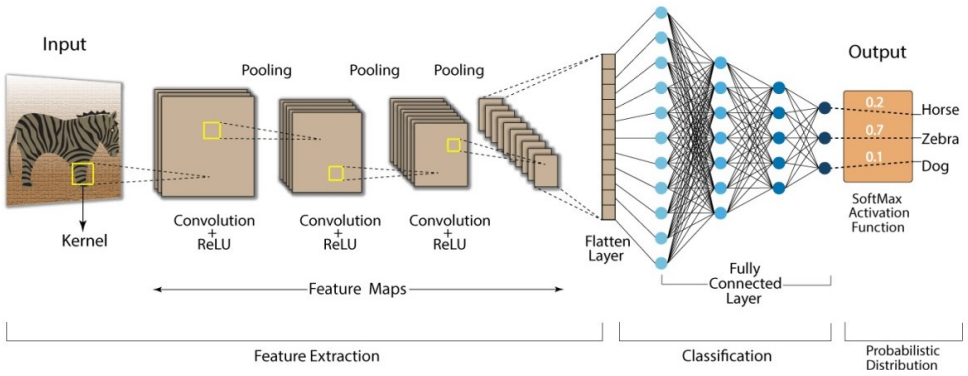

Figure 2.6: CNN example for mutli-class classification (Swapna 2021)

2.1.5 Activation Functions

Artificial neural networks are inspired by biological neural networks, which comprise a large number of interconnected neurons that may or may not fire in response to a stimulus. In artificial neural networks, activation functions are the mathematical tool that is used to mimic the threshold activation in biological neural networks that determine if the input to a given neuron is sufficient to cause the neuron to 'fire'. In this section, a brief summary will be given to outline the most commonly used activation functions. Activation functions perform a mathematical operation on the weighted sum of inputs to a given neuron (plus a bias term). The operation often transforms the weighted sum to a value that is between some upper and lower limits. Such transformations are most often non-linear as this allows ANNs to learn functions that are also non-linear. In other words, activation functions enable ANNs to become universal function approximators. By applying non-linear activation functions to ANNs, in theory, they can learn any function. ANNs being universal function approximators is the inherent quality of deep learning methods that allows them to be so versatile and used in many applications such as machine translation for language, object detection in computer vision, and simple linear regression.

The Sigmoid Function

The sigmoid function is an 'S' shaped function that maps any input value in the interval from $[-\infty, +\infty]$ to the range $[0, 1]$. This is a non-linear function with a smooth derivative often used in the final layer of a neural network designed for binary classification. The sigmoid function is given displayed in Figure 2.7, and the equation of the sigmoid function is

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

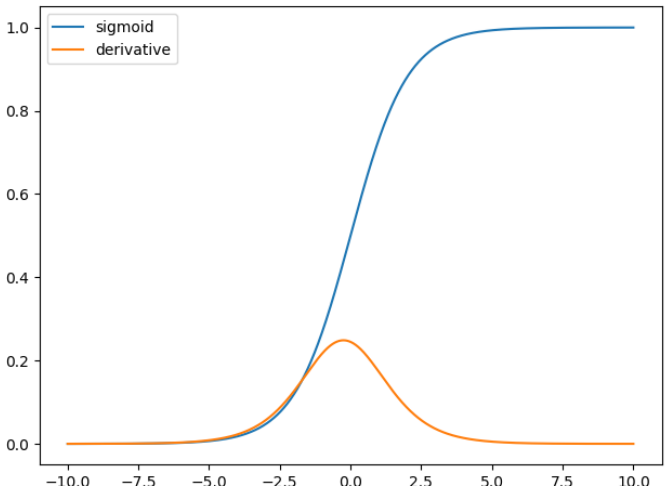

Figure 2.7: The sigmoid function and its derivative

where x is the weighted sum of inputs to a given neuron plus a bias term. As shown in

Figure 2.7, the sigmoid function will take most negative inputs and return a value that is close to zero. Furthermore, for most positive inputs, the sigmoid function will return a value that is close to one. This is the case for all inputs that lie outside of the range of $[-5, 5]$, which causes the derivative indicated by the orange curve in Figure 2.7 to become very small and approximately equal to zero. This causes an issue known as the vanishing gradient problem and results in a network not being able to reach a global optimum value, and this occurs when the sigmoid function is used in hidden layers of an ANN. The vanishing gradient problem arises because the weights associated with each layer of the network are iteratively updated based on their gradient. As the gradient of the sigmoid function becomes very small for large negative or positive values of x, the weights are updated by tiny amounts each iteration. For randomly initialised weights that happen to have a large absolute magnitude, they are likely to remain close to their initial value after thousands of epochs.

The ReLU Function, and it's variants

The vanishing gradient problem in hidden layers is resolved by using the Rectified Linear Unit (ReLU) function. The ReLU function, and its variants, are the most widely used among machine learning researchers and practitioners (Rasamoelina et al. 2020). The ReLU function is defined as

$$f(x) = \max\{0, x\} \tag{2.2}$$

This function outputs zero for all negative values and therefore is non-differentiable at zero. This causes the gradients to be equal to zero for a negative input which means that no further learning occurs, which is a problem referred to as the dying ReLU problem (Lu et al. 2019). This problem is rectified by a variant known as the Leaky ReLU activation function. The Leaky ReLU activation function is defined as

$$f(x) = \max\{0.01x, x\} \tag{2.3}$$

which ensures that the function remains differentiable for negative values of x. This activation function is used in the YOLOv4-tiny object detection algorithm used in this report. There are also many other common activation functions such as Mish, which is a state-of-the-art non-monotonic activation function that is defined as

$$f(x) = tanh \times ln(1 + e^x). \tag{2.4}$$

The Mish activation function is used in the YOLOv4 object detection algorithm used in this study. Providing a full explanation of the details of this activation function falls outside the scope of this study, and full details can be found in the original paper by Misra (2019).

Another variant of the ReLU function is known as the SiLU function or Swish, which uses the sigmoid function and is defined as

$$f(x) = x \times \sigma(x) \tag{2.5}$$

in which σ is the sigmoid equation defined in Equation 2.1. The Swish activation function is used in EfficientNet and is shown in Figure 2.8, also showing other common activation functions. Further details on the Mish activation function can be found in a paper by Ramachandran et al. (2017).

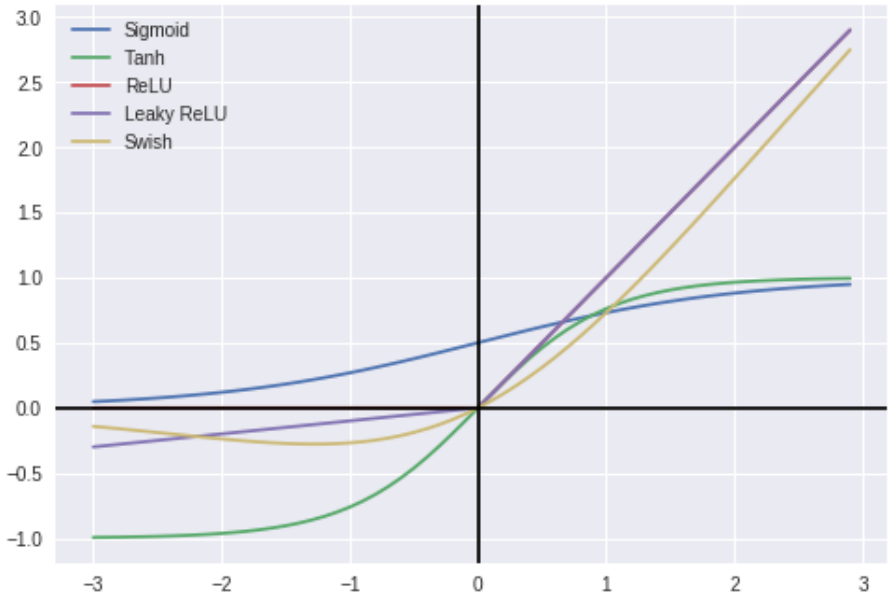

Figure 2.8: Common activation functions (Kızrak 2020)

2.2 Loss Functions

Loss, error, or cost functions are used to calculate the loss associated with a given prediction by the model and the ground truth label for the same given training sample. The loss associated with the prediction is then backpropagated through the network by taking the derivative of the loss with respect to each layer of weights in the networks hierarchy. These gradients are then used to update the weights to minimize an optimization algorithm allowing the network to learn to make better predictions in the future. The loss function that is chosen will depend on the application of the neural network. For example, regression-based problems often use the Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE). In contrast, classification problems may use Cross-Entropy Loss or Hinge Loss. It is important to note that loss functions used in regressions would not be applicable to compute for a classification problem and vice versa. Furthermore, there are task-specific loss functions such as Intersection over Union, which measure the loss associated with bounding box predictions. The exact loss function selected will depend on the application, the network and the data that is being used. The loss functions that are used in this

report will be outlined and justified in this section.

2.2.1 Mean Square Error

The Mean Square Error (MSE) is one of the most common loss functions used for regression-based problems. Specifically, it considers the error value of the Euclidean distance between the predicted value and the ground truth value. The error is squared, summed over all samples (Neill & Hashemi 2018). The MSE expressed as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{2.6}$$

where y_i is the ground truth value for a given sample, $\hat{y_i}$ is the predicted value for a given sample, and n is the total number of samples (in a batch or data set). As the Euclidean distance is squared between the ground truth and predicted values, large errors are penalised heavily. This makes this loss function particularly useful for applications in which accurate predictions are essential, such as key point estimation. The final validation loss for the key point regression model in this report will represent the MSE for the validation set. The square root of the MSE can be taken, giving the Root Mean Square Error (RMSE), which gives a value for the mean error associated with the model, which will give the mean pixel error associated with the final model.

2.2.2 Intersection over Union (IOU)

To evaluate the quality of object localization for object detection tasks, the most frequently used metric is Intersection over Union (IOU), also known as the Jaccard Index. The IOU is defined in Figure 2.9. The two boxes in the numerator and denominator of the formula have been included to illustrate the terms 'intersection' and 'union', and one box can be thought of as the predicted bounding box and the other as the ground truth annotated bounding box for a given object in an image. The term 'intersection' refers to the overlap between the two boxes. Specifically, the intersection is the area of pixels of overlap between the predicted bounding box and the ground-truth bounding box label. The term 'union' refers to the total area of the predicted bounding box and the ground truth bounding box.

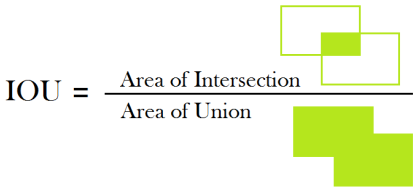

Figure 2.9: Intersection over Union (IOU)

Typically an IOU threshold is also set, which determines how tightly the predicted bounding

box encapsulates the specified object or not. A 'loose' bounding box will contain a greater number of background pixels than a 'tight' bounding box. Researchers most frequently set this threshold to be equal to 0.5. This means that if the IOU is above 0.5, it does produce a final predicted bounding box. If it is less than 0.5, it does not produce a final bounding box.

2.2.3 Mean average Precision (mAP)

The mean average precision is a metric that is commonly used to compare object detection algorithms. The precision of a model is the ratio of true positive predictions over the total number of positive predictions and is given by

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} , \tag{2.7}$$

in which TP is True Positive, and FP is False Positive. So, for example, if we have a data set with 120 images of hands, with only 1 hand in each image. This data set has been inputted to an object detection model that has predicted 100 hands and associated bounding box coordinate values and confidence values. Of the 100 total predicted bounding boxes, 20 of them have a confidence value below the threshold of 0.5, meaning that they must be considered False Positives. This leaves a total of $100 - 20 = 80$ True Positives. Therefore the precision of the model in predicting the class 'hand' would be Precision $= 80/(80 + 20) = 0.8$.

When considering precision alone, it does not consider the total number of objects present in the data. Continuing on from the previous example, there were 120 images, each containing one hand. Consider if a model predicts 80 correct bounding boxes and 20 incorrect bounding boxes for 10,000 images of hands, the precision of the model would still be 0.8. Therefore, it is important to consider the relationship between the number of correct predicted bounding boxes and the total number of hands present in the data set. This is achieved by a metric called Recall which is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} , \tag{2.8}$$

where TP is True Positives and FN is False Negatives. For the example considered with 120 images of single hands, the recall would be given by Recall $= 80/(80 + 40) = 0.667$.

Object detection models are usually evaluated by considering different IOU thresholds. Different IOU thresholds will often result in different TP predictions. A way to compare models among all considered IOU thresholds is to take the Mean Average Precision (mAP) of each model. The mAP of a single model will also give an indication of how well it performs across different IOU thresholds for all classes considered for a given model. It is worth highlighting that the mAP for a single class object detection model is the same value as the model's Average Precision (AP). To calculate the average precision, the average value of 11 points on the

precision-recall curve for each possible IOU threshold for the same class (Bochkovskiy et al. 2020). As the model in this report considers only one class for object detection (hand), the mAP of the model is equal to the AP of the only class 'hand'.

2.3 Optimization

Optimization algorithms are the protocols used to change the parameters of a neural network, such as the weights and bias terms, to enable the network to make better predictions in the future. Optimization algorithms update the weights and bias terms with the objective of minimizing the loss or error associated with the network. This occurs by passing input into the network to obtain a predicted output, known as a forward pass. A loss function is used to quantify the error between the predicted output and the ground truth label associated with that training sample. The error is then backpropagated through the network by taking the derivative of the error with respect to the weights at each stage of the network from the output layer to the input layer. The calculated gradients are then used in conjunction with a learning rate to update each weight and bias term for each iteration. An iteration refers to the number of batches that the model has seen. Often a single batch will contain 32 or 64 training samples. This process requires a high number of iterations to reach a global minimum of the optimization algorithm. Furthermore, this process may require multiple epochs to converge to the global minimum. An epoch refers to the number of times the model has seen the entire data set (Doshi 2020).

The learning rate is an important parameter to select to ensure that the optimization algorithm can converge to a local minimum. If the learning rate selected is too small, the weights will be updated in small steps, and learning can occur too slowly. This causes issues as it increases the time it takes for the model to converge, and therefore the learning rate needs to be sufficiently large to meet the time constraints of the task at hand. On the other hand, if the learning rate is set too large, the steps taken can be too large, and the algorithm may miss the global minimum. Furthermore, if the learning rate is sufficiently large, it can cause a phenomenon known as divergence in which the loss will increase at each step rather than decrease. Therefore, an appropriate learning rate needs to be selected to ensure the steps are sufficiently large so that training does not take too long and that the steps are sufficiently small to ensure the model reaches a global minimum and does not diverge.

Various popular optimisation algorithms are used in a convolutional neural network, such as Gradient Descent, Stochastic Gradient Descent, and Adam. The optimisation algorithm used in the key point regression model for this report is Adam as it has been shown to outperform traditional gradient descent algorithms (Kingma & Ba 2014). Further details can be found in a paper by Kingma & Ba (2014), but most importantly, it is an extension of the gradient descent algorithm explained in this section but is more computationally efficient and deals better with

sparse gradients experienced in computer vision projects (Kingma & Ba 2014, Brownlee 2021).

2.3.1 Generalization Errors

Generalization error refers to the measure of how accurately an algorithm can predict values for previously unseen data. This error can be caused by a poorly fit model, sampling error, and noise in the data. Three classifications can summarise the fit of a model to a given data set: underfitting, balanced, overfitting. Underfitting occurs when the model is not able to make accurate predictions on the training data. This occurs because the model is not sophisticated enough to capture the relationship between the inputs and outputs. A model that underfits the data has high bias and low variance. On the other hand, overfitting occurs when the model performs well on training data but makes poor predictions on validation or testing data. This occurs because the model is learning the trend in the data and the noise present in the data. A model that overfits the data has low bias and high variance. A balanced model will make good predictions on training and unseen data. This is when the model has been able to learn the relationship between the inputs and outputs without learning the noise that was present in the training data. A balanced model will have low bias and low variance and ideally should be able to perform the prediction task with the same accuracy as a human or better.

An example of the three model fits discussed is shown in the context of a regression problem in Figure 2.10. Consider that the data points to be a low order polynomial function such as a quadratic. A balanced model is shown in the middle plot, in which the plotted line representing the model's predictions accurately captures the trend of the data. The trend line does not pass through every data point. However, this is due to noise in the data, which can be present for a variety of reasons. The left plot shows an underfitting model, which makes predictions based on the equation of a straight line or a first-order polynomial. This model will make poor predictions on training data and unseen data. The right plot shows a model that is overfitting the training data as it has learned the noise. This is likely to make poor predictions on data that is out of the sample, such as at a later time than is plotted on the axis in Figure 2.10 right.

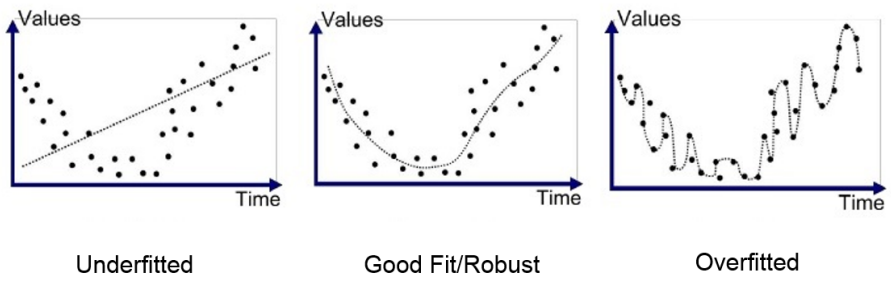

Figure 2.10: Model Fit: Underfitting, Balanced, Overfitting (Bhande 2018)

2.3.2 Early stopping

An example of the three model fits discussed is shown in the context of a regression problem in Figure 2.10. Consider that the data points to be a low order polynomial function such as a quadratic. A balanced model is shown in the middle plot, in which the plotted line representing the model's predictions accurately captures the trend of the data. However, the trend line does not pass through every data point. Noise in the data can be present for various reasons, but it is not desirable for a model to learn. The left plot shows an underfitting model, which makes predictions based on the equation of a straight line or a first-order polynomial. This model will make poor predictions on training data and unseen data. The right plot shows a model that is overfitting the training data as it has learned the noise. This is likely to make poor predictions on data out of the sample, such as at a later time than is plotted on the axis in Figure 2.10 right.

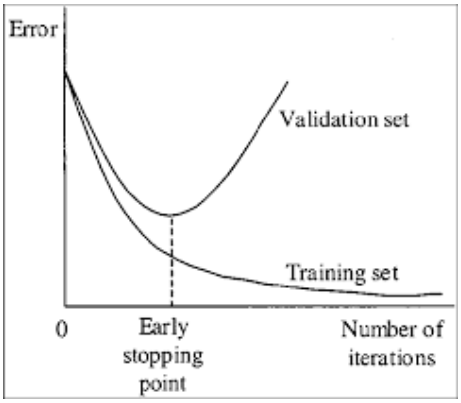

Figure 2.11: Early Stopping Point (PapersWithCode n.d.)

Chapter 3

Computer Vision: Review of methods

Seeking more precise and resilient movement measurement methods has a multitude of applications in a variety of industries. These methods include measurement devices such as GPS trackers, video camera footage, microphones using ultrasound technologies, and various bespoke or tailored sensors. Using video camera footage is the most widely available technique as the ownership of smartphones able to capture high-quality videos is so high (e.g. 82% in the Uk and 77% in the USA), allowing for high resolutions observations of behaviour (Newzoo 2021, Zhao et al. 2020). Furthermore, video camera footage allows for high-resolution observations of behaviour. Motion can be captured in video footage with high levels of accuracy using specialist equipment such as 'Mo-cap' suits (Riecickỳ et al. 2018). However, recent breakthroughs in deep learning technologies have allowed for advances in measuring movement in video footage without markers (Wu et al. 2020, Mathis et al. 2020).

The key benefit of markerless key-point based pose estimation with deep learning is that it reduces the need for specialist equipment, such as 'Mo-cap' suits or electronic sensors. Marker-based motion capture is achieved through physically creating contrast on points of interest. This can be achieved through the use of colours, LEDs, or reflective markers, which allows for existing computer vision technologies to track these key points with ease (Mathis et al. 2020). On the other hand, markerless motion capture requires no specialist equipment to be worn when recording the video. However, it does require some kind of fit-for-purpose deep learning network that has been trained based on annotated example images (Mathis et al. 2020). Furthermore, a key benefit of deep learning based algorithms is that they can be tailored for different tasks allowing different types of animals or body parts to be tracked (Mathis et al. 2020).

3.1 Pose Estimation

In this section, current pose estimation methods will be outlined and reviewed. Based on this, a preferred method will be selected for use in the automatic labelling system for DeepLabCut.

As highlighted by Mathis et al. (2020), numerous pose estimation algorithms have been developed, which are comprehensively reviewed by Moeslund et al. (2006), Poppe (2007). However, algorithms based on neural networks (or deep learning) offer the greatest accuracy when evaluated on data for human pose estimation (Insafutdinov et al. 2016, Xiao et al. 2018). Generally speaking, deep learning based pose estimation algorithms typically consist of two stages, an encoder and decoder network (also often referred to as backbone and output heads, respectively). The role of the encoder is to learn and extract visual features from each image in a video. These learned features are used by a decoder network in predicting each relevant body part along with the location in the frame (and usually returns image coordinates of key points). Pose estimation algorithms essentially fall under the umbrella of object detection, which is a field of research that is advancing rapidly (Mathis et al. 2020). The ongoing development of new algorithms and techniques improves both both the accuracy and efficiency of deep learning algorithms to perform tasks in object detection and pose estimation (Wu et al. 2020).

Deep learning systems consist of four core components, including a data set (including ground truth annotations), a model architecture, a loss function, and an optimisation algorithm (Goodfellow et al. 2016). Mathis et al. (2020) highlights that the data set defines the relationship between the inputs and outputs that the model architecture must learn. For pose estimation, the model must learn to take an image as the input and predict a given pose as the output. Learning occurs by updating the model's weights by the optimisation algorithm that minimises the loss function over multiple iterations. In each iteration, the quality of the predicted pose is evaluated by comparing them with the ground truth annotated pose. The weights are adjusted over numerous iterations until the predicted pose is as close to the ground truth pose as possible. There are multiple options or variations to consider for each of the four key components of a deep-learning-based pose estimation system. The selected combination of components will affect the performance of the model. These will be discussed in detail in the following sections.

3.1.1 Data sets and Data Augmentation

Various data sets are to pre-train computer vision models used in deep learning based pose estimation systems through a process known as 'transfer learning'. Two types of data sets are typically relevant for training pose estimation systems. The first type of data set are those used for image classification tasks, for example, ImageNet (Deng et al. 2009) or the CIFAR data sets (Krizhevsky et al. 2009). For example, the ImageNet data set contains 14.2 million images of 21 thousand classes. Therefore, there are many images with a multitude of learnable image features present in this data. Such open-source data sets allow research teams to build state of the art models and allow for easy comparison between models. The accuracy of models has drastically improved in the last 12 years, as shown in Figure 3.1. Training networks on data

sets such as ImageNet can take several days. For example, it took 17 days to train the widely used DNN 'ResNet' on ImageNet (James 2017). Using datasets such as ImageNet for transfer learning allows researchers to save days of training by downloading pre-trained weights for robust filters that have learned to identify many image features. Therefore, networks pre-trained on these data sets are extremely useful in pose detection for saving on training time and obtaining trained and robust convolution filters (Mathis et al. 2020). Comparison of models trained on ImageNet can be achieved easily through 'Top-1' Accuracy, which refers to the number of correctly predicted instances by a given model where the single class with the highest confidence value is the ground truth class. Further, 'Top-5' Accuracy refers to the number of correctly predicted instances by a given model where any one of the top 5 highest confidence predictions match the ground truth class (Krizhevsky et al. 2012).

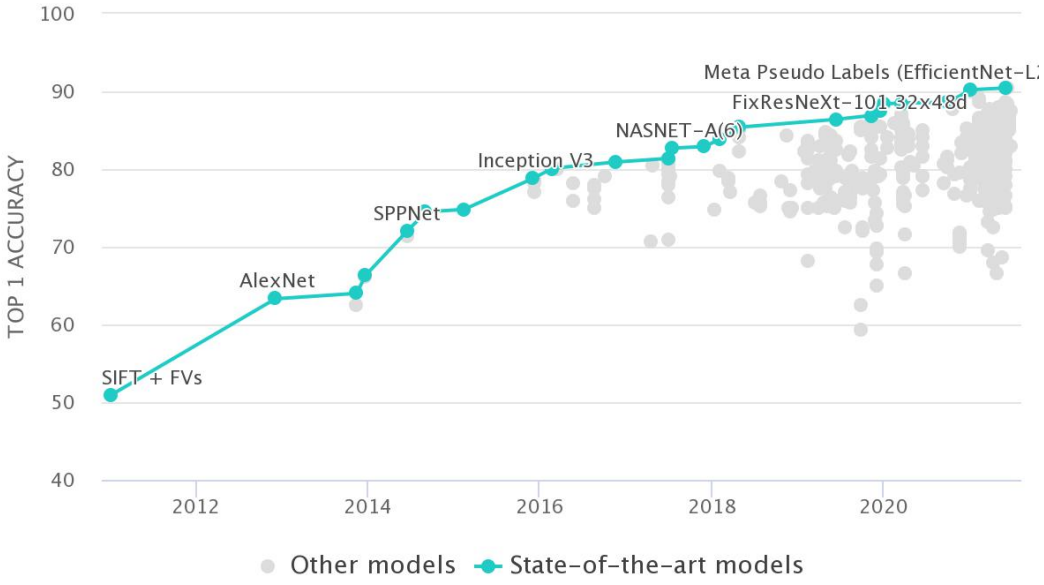

*Figure 3.1: Image Classification on ImageNet, (*PaperswithCode *2021*a*)*

The second type of data sets that are relevant to pose estimation are those that have been made for purpose in pose estimation. These can be general-purpose pose estimation data sets such as MS COCO (Lin et al. 2014) or MPII pose (Andriluka et al. 2014) which are used for benchmarking pose estimation models. Alternatively, there are pose estimation data sets specific to particular applications, including NYU hands which is specifically for benchmarking hand pose estimation models (Tompson et al. 2014).

A common issue when building pose estimation models in laboratory conditions is having insufficient data to train the model. This issue can be addressed by implementing 'Data Augmentation' techniques which artificially increase the amount of training data by creating modified copies of previously existing data. These techniques involve applying various transfor-

mations, including geometric transformations (such as flip/crop/rotate/translate), colour transformations (altering the RGB colour channels, greyscale, intensifying a given colour), kernel filters (sharpen/blur an image). Data Augmentation techniques increase the model's ability to deal with noise in the data, making it more robust and less likely to overfit the data that the model is trained on. When designing a deep learning pose estimation system, the adopted data augmentation techniques must not negatively affect the semantic information of the image. Different techniques can be applied depending on what the intended purpose of the model is. Furthermore, certain features within the dataset may encourage the use of a specific data augmentation technique.

3.1.2 Model Architectures

Deep learning based markerless pose estimation systems typically follow variations of the same key network architecture. As highlighted by Mathis et al. (2020), DL pose estimation systems are typically composed of an encoder (backbone) network and one or multiple decoder (heads) networks. The encoder network architecture is typically based on generic CNN models used in object detection or classification. Specifically, which encoder architecture is specified is often the most influential decision that impacts various properties of the overall system. These factors include computational requirements, inference speeds, and the amount of training data required. Typical encoder network architectures used in DL pose estimation systems include variations of stacked hourglass networks (Newell et al. 2016), ResNets (He et al. 2016), MobileNetV2s (Sandler et al. 2018), or EfficientNets (Tan & Le 2019). A key benefit of using these architectures for pose estimation systems is that pre-trained weights are often open-source on one or more large scale benchmark datasets such as ImageNet (Ridnik et al. 2021). There is evidence in the current literature to suggest that pre-trained encoder networks have proven to be a benefit to pose estimation for small-scaled experiments as they are more robust, have shorter training times, better performance, and less training data required (He et al. 2019, Mathis et al. 2018, 2020, 2021).

The performance of encoder network architectures is a highly active area of research (Mathis et al. 2020). There have been multiple breakthroughs in the last few years in not only the accuracy of network architectures in object classification benchmark data sets (e.g. ImageNet), but also the speed of computation (Kornblith et al. 2019, Goodfellow et al. 2016). Such breakthroughs are largely dictated by factors such as the number of trainable parameters. Due to the importance of the ImageNet benchmark data set, the amount of active research in this area, and the current rate of improved performance in image classification, there will likely be frequent performance improvements for the foreseeable future. This continued progress necessitates that developers stay up to date with the latest breakthroughs in object detection technologies, which is made easy through sources such as PapersWithCode (2021*b*) which compare the performance of state-of-the-art network architectures in all computer vision and natural language processing

areas, and also provide links to relevant research articles and source code.

There are also a variety of decoder network architectures that are used in pose estimation. In typical convolutional encoders, input images are passed through a series of convolution filters and pooling layers. Encoders gradually downsample the original high-resolution input image and produce feature maps with learned image representations of the original visual features in the input images. In regression-based systems, these downsampled feature maps can be directly to make predictions for key points of interest by passing the downsampled representation through a flatten layer, followed by fully connected layers (Mathis et al. 2020). However, when the downsampled feature maps can be upsampled to increase the resolution again using deconvolutional layers (Insafutdinov et al. 2016, Xiao et al. 2018). This upsampling process is referred to as the decoder (head) network and typically involves taking feature maps from multiple stages in the encoder network and upsampling them to a given resolution. There are also various other state-of-the-art approaches to pose estimation network architectures. These include stacked hourglass networks, which do not implement transfer learning but instead use skip connections to transfer visual information from the encoder network to the decoder network at the same relative position in the convolution hierarchy, which supports upsampling feature maps (Newell et al. 2016).

DeepLabCut is the pose estimation tool that has been demonstrated to be effectively used in quantifying Parkinson's bradykinesia (Zhao et al. 2020). Currently, DeepLabCut requires an automatic labelling system to increase usability for widespread deployment. It is essential to explain the network architecture of this pose estimation tool as it is a primary motivation behind this paper. DeepLabCut was create using DeeperCut, a state-of-the-art pose estimation algorithm and was developed by Insafutdinov et al. (2016) and adopts a ResNet-50 encoder architecture (He et al. 2016). DeeperCut was then adapted to produce DeepLab by Chen et al. (2017) which the final convolutional layer in the encoder network was removed and replaced by dilated deconvolutional layers with a stride of 16. Dilated layers were used to larger receptive fields with higher spatial information whilst using less computational resources and therefore taking less time to complete these operations (Mathis et al. 2020). A final single deconvolutional layer (with a stride of 8) was then used for upsampling to produce output heat maps that predicted the location of the key points. The DeepLab network architecture has inspired many other pose estimation algorithms, and various backbone architectures can be used to give a differing performance in speed and accuracy (Xiao et al. 2018, Mathis et al. 2020). These networks were adapted to produce DeepLabCut, a (multi) animal pose estimation tool that is open-source and includes a GUI. This tool easily allows non-programmers to extract frames from videos, label key points in the frames, create training data sets, initialise training, and evaluate the network (Nath et al. 2019).

3.1.3 Loss Functions

Any key point of interest in pose estimation can be represented by (x,y) coordinates on an image (Mathis et al. 2020). There are two underlying approaches in making predictions on these key points, which result in two core loss functions that can be adopted as part of the ML system. Firstly, regression-based loss functions can be implemented where the coordinates are targets that must be predicted the model. On the other hand, classification-based loss functions can be implemented, which is more widely applied in practice. When treating pose estimation as a classification problem, the coordinates of a key point are mapped onto a grid which is the same dimensions as the original input image, and the model produces a heatmap of pixel-based probabilities for each key point considered. The critical difference between regression and classification is that a regression loss function is defined by sets of (x,y) coordinates and classification problems are defined by images (of ground truth pixels and pixel heatmap predictions). The key benefit of classification is that it is a fully convolutional process and can make several predictions for the same body part in the same image. For example, when making predictions for the tip of multiple index fingers from different hands in a single video frame. The data set used in this study only contains a single hand in each frame, and therefore this benefit of classification-based approaches is redundant. Furthermore, the performance of classification-based approaches has been demonstrated to be lower than regression-based approaches when using low-resolution images (Li et al. 2021).

The mean squared error (MSE) is a standard loss function used to assess the quality of predictions in both regression and classification based pose estimation algorithms. The MSE measures the euclidean distance between predicted and ground truth pixels. Alternatively, the percentage of correct key points (PCK) can be used to refer to the proportion of predicted key points that fall within a distance of the ground truth. An example user-defined distance could be the human labelling variability within the training data if this value is known.

3.1.4 Optimization Algorithms

Pose estimation tools typically adopt a multi-stage training approach in which a pre-trained encoder is pre-trained for an unrelated task such as image classification on data sets such as ImageNet or CIFAR. Encoder networks are typically pre-trained using stochastic gradient descent (SGD) or other popular variants of this core method, such as Adam (Kingma & Ba 2014). The encoders weights are downloaded and frozen so that they can not be trained in later stages (Mathis et al. 2021). The decoder, top, or head of the network is then trained using SGD variants such as the Adam optimizer to calibrate the network for a given pose estimation application (Mathis et al. 2020). Using a pre-trained encoder and a fine-tuned decoder can increase the model's ability to generalize to unseen images as the pre-trained weights are more robust (Mathis et al. 2021).

3.1.5 Chosen method for this study

Transfer learning will be utilised in the methods for this study as it has been demonstrated that pre-trained encoder networks are more robust, offer better performance, and require less training data and time than training from scratch (Mathis et al. 2021). The selected encoder will be trained on ImageNet as this is the primary data set used to pre-train models, and it contains a high amount of classes and training data. A regression-based approach has been selected as only two key points are being predicted: the tip of the index finger and thumb for a single hand. Therefore, the core benefit of classification-based approaches, which is their ability to predict the same body part for multiple animals in the same frame, would not be utilised. Furthermore, regression-based approaches have been demonstrated to exhibit better performance when making predictions on low-resolution images (Li et al. 2021). Many pre-trained networks such as ResNet and EfficientNet take low-resolution images as inputs such as 224×224, further justifying a regression-based approach (He et al. 2016, Tan & Le 2019). Lastly, data preprocessing is more simple for regression-based approaches as there is no need to convert key point coordinates to pixel-based heatmaps. Therefore implementing a regression-based method will take less time. The regression-based approach will involve extracting downsampled feature maps to directly predict key points using a flatten layer followed by fully connected layers. Data augmentation will not be implemented in this study as the original images are already labelled, and therefore applying augmentation techniques such as stretching, rotating or flipping the images may negatively impact the quality of the semantic information between the images and the labels as the labelled coordinates would be on the scale of the original image. The chosen pre-trained encoder used as the feature extractor will be discussed in the next section.

3.2 A brief history of encoder network architectures

In this section, a brief history of influential convolutional neural networks encoders will be given. Based on this review, pre-trained encoder networks will be selected as feature extractors in the key point estimation model for use in the automatic labelling system.

3.2.1 LeNet

Arguably the most influential CNN is the first one that used backpropagation, or gradient-based learning, to learn filter weights in convolutional layers as this is the key underpinning concept that serves as the foundation for all state-of-the-art networks today. This network was designed by LeCun et al. (1998), and named 'LeNet'. It was released in 1998 designed for optical character recognition (OCR), which took in 28 × 28 grey-scale images of hand-written digits classified from 0-9 using the famous MNIST data set. The significance of LeNet was it highlighted that automatically learned visual feature extractors could outperform handcrafted feature extractors (LeCun et al. 1998). See Figure 3.2 highlighting the network architecture for LeNet

with two convolution and pooling layers followed by multiple dense layers, also known as fully connected layers.

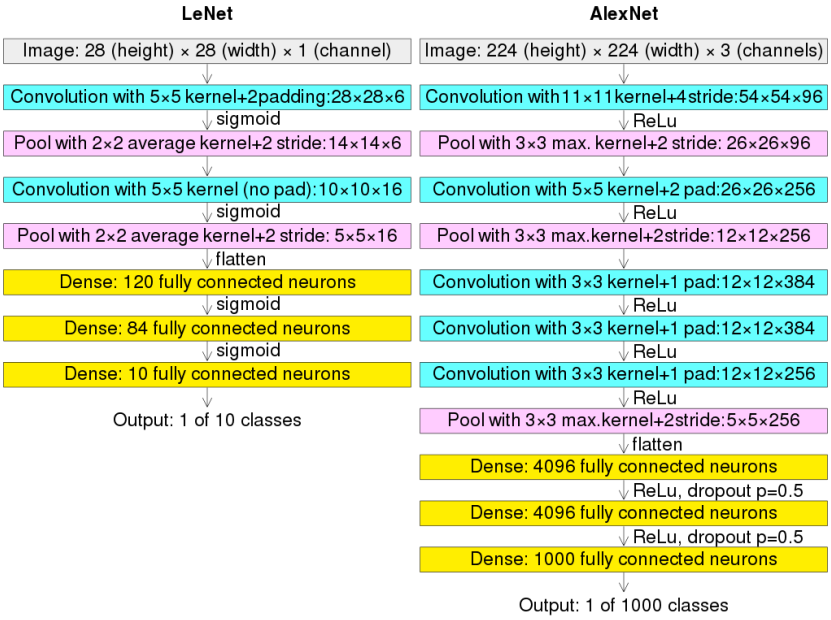

Figure 3.2: LeNet and AlexNet network architectures

3.2.2 AlexNet

A more recent influential CNN is similarly named after the designer 'AlexNet' and was released by Krizhevsky et al. (2012). AlexNet took nearly six days to train on two NVIDIA GTX 580 3GB GPUs, and the author did highlight that training would not be feasible for large scale object classification would not be possible without training the networks on GPUs. AlexNet would not be feasible to train on a CPU because of the number of learnable parameters in the network. For example, LeNet had 802,816 weights whereas AlexNet had 62,378,344 weights. That means over 77× the number of learnable parameters that need to be iteratively updated for each backward pass through the network. Although AlexNet was not the first network to utilise a GPU for training, it is considered one of the most influential papers in computer vision as it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 and, as a result, inspired many future designers to train CNNs for large scale object detection using GPUs. Furthermore, the network architecture itself was well designed and therefore achieved a top-5 error of 15.3%, which outperformed the runner up with by more than 10.8% (Krizhevsky et al. 2012, Russakovsky et al. 2015).

3.2.3 VGG-16

Shortly following AlexNet was the release of VGG-16, which was published by Simonyan & Zisserman (2014) from the Visual Geometry Group (VGG) at the University of Oxford. VGG-

16 was primarily inspired by AlexNet and was entered into the ImagNet challenge in 2014, in which it achieved a top-5 error rate of 7.32%, which is less than half that achieved by AlexNet only two years prior. VGG-16 was able to outperform AlexNet by reducing the kernel size from 11×11 and 5×5 in the first few layers of AlexNet to multiple 3×3 filters consecutively shown in VGG models. The additional Conv layers resulted in 16 to 19 trainable layers, which took 2-3 weeks to train the network on NVIDIA Titan Black GPUs, as a result of 138 million parameters (Simonyan & Zisserman 2014). VGG-16 indicated that increasing the depth of a network by having consecutive convolutional layers could result in better accuracy in large scale image classification Simonyan & Zisserman (2014). However, the paper also indicated that beyond 20 layers, accuracy could decrease due to optimisation issues with SGD, with deeper networks being outperformed by shallower CNNs. This issue is caused by vanishing/exploding gradients He et al. (2016).

3.2.4 ResNet

ResNet was proposed by He et al. (2016), and they identified an issue with training deeper networks known as the vanishing/exploding gradient issue and a method to overcome it. As neural networks are universal function approximators, in theory, by adding more layers to a DNN, they should learn progressively more complex features and offer better performance. He et al. (2016) highlights that most networks that are considered 'deep' contain roughly 16 to 30 layers and demonstrates how much deeper networks such as one with 56 layers can exhibit higher training and test error than a shallower counterpart. Further, if this problem was caused by overfitting, then regularization techniques should combat this issue, but they do not. Therefore, this issue is an optimization issue which was later called the vanishing/exploding gradient problem. When a network becomes too deep, the gradients used by the loss function in optimization tend to zero after several iterations. This leads to weights no longer updating, and therefore no further learning occurs, causing demonstrated training and test loss not to reach the global theoretical loss minimum (He et al. 2016).

He et al. (2016) addresses the vanishing gradient problem by proposing the 'residual block', which is demonstrated in Figure 3.3. The key characteristic of the residual block is the 'skip connection' identity mapping which adds the feature map outputted from a previous layer to a later layer as demonstrated in FIGURE. The previous feature map has to be multiplied by a linear projection to ensure that the dimensions of both feature maps are the same. These skip connections allow information to be passed between layers and prevent the vanishing gradient problem from occurring. Due to residual blocks, a network with 152 layers was produced with a top-5 error of 3.57%, and it won the ILSVRC 2015. Although this network is much deeper than previous networks such as VGG-16, it has less than half the parameters due to having far less fully connected layers (He et al. 2016).

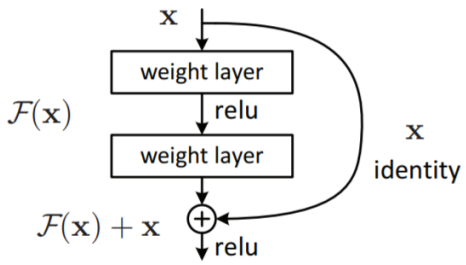

Figure 3.3: The residual block (He et al. 2016)

3.2.5 EfficientNet

In the last few paragraphs in this section, various CNNs have been compared that all aim to improve accuracy on the ImageNet data set by increasing the network's size whilst also trying to improve the efficiency of the network. A key issue in designing CNNs is how to effectively balance the depth of the network (the number of layers), with the width of the network (the number of filters or kernels), and the resolution of the network (the size of the input image). Increasing any of these dimensions will result in a greater computational overhead, which can be easily calculated and is measured in FLOPS, whilst also demonstrating further increases in measured accuracy on data sets such as ImageNet. Continuously increasing the depth of networks does not infinitely improve the accuracy. It was demonstrated that ResNet-1000 has similar accuracy as ResNet-101 despite having approximately 10$\times$ the amount of layers (Tan & Le 2019). Researchers at Google, Tan & Le (2019), present a new state-of-the-art mobile sized baseline architecture accompanied by a compound scaling method to improve the accuracy of the baseline model whilst optimizing the efficiency of the up-scaled model. Compound scaling refers to simultaneously increasing at least two of the following: depth, width and resolution network (Tan & Le 2019).

The first significant contribution of the paper by Tan & Le (2019) is that they propose EfficientNet-B0. The authors highlight that having a good baseline architecture is a critical design decision for compound scaling, as making a given network larger does not change the underlying operators. Therefore they use a Neural Architecture Search, a technique for automating the design of DNNs, that optimises by accuracy and FLOPS, resulting in EfficientNet-B0. This network has only 5.3 million parameters and 0.39 billion FLOPS and achieves a top-5 accuracy of 93.3% on ImageNet (ResNet-152 achieved a top-5 accuracy of 93.8% with 60 million parameters and 11 billion FLOPS.

EfficientNet-B0 is underpinned by a type of short-cut connection, inspired by ResNet variants, known as mobile inverted bottleneck connections (MBConv). MBCov uses the depthwise-separable convolution rather than the conventional convolution operation that is more widely used. This concept essentially splits the convolutional operation into firstly a depthwise convo-

lution, a spatial convolution performed on each input channel in isolation. This is followed by a pointwise convolution which is essentially a 1×1 convolution, which projects the outputted channels from the depthwise convolution into a new channel space chollet2017xception. This operation results in a significant reduction in parameters and makes convolutional layers far more computationally efficient. However, it reduces the network's learning capacity, which can be an issue in very small networks. MBConv blocks also use inverted residual blocks, which are based on 'skip connections introduced for ResNet by He et al. (2016). In the original residual blocks, skip connections are used to connect layers with many channels, known as wide layers. The connected layers bypassed multiple narrow layers with a comparatively smaller number of channels. However, for inverted residual blocks, the skip connections are between narrow layers, which bypass wider layers (Sandler et al. 2018). The last key method implemented in MBConv blocks is referred to as 'Squeeze and Excitation' blocks. These essentially assign weights to each output channel rather than weighting them all equally. Which allows the network to learn how to prioritize different channels rather than treating all extracted visual features with equal importance (Hu et al. 2018).

Tan & Le (2019) also released a method for compound scaling of convolutional neural networks. If an input image has a larger resolution, the network requires additional layers to increase the receptive field size and additional channels to capture fine-grained visual information on the larger image. Figure 3.4 shows the seven variants of the EfficientNet models, where B0 represents the baseline model, which takes an input image sized 224×224 and B7 represents the largest and highest ImageNet top-1 accuracy model, which takes an input image sized 600×600. The position of the red curve is to the top left of every other plotted point on FIGURE. This highlights the importance of the compound scaling method and an optimized baseline architecture, as no other model with the same number of parameters can offer similar performance to the EfficientNet variants (Tan & Le 2019).

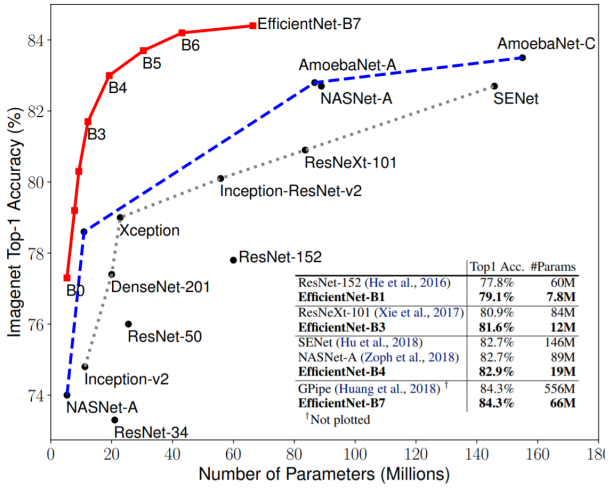

Figure 3.4: EfficientNet performance on ImageNet (Tan & Le 2019).

EfficientNet-B0 will be selected as the first pre-trained encoder used for the key point regression model in this report as it offers a higher top-1 accuracy on ImageNet than ResNet-50 but contains significantly fewer parameters. Additionally, EfficientNet-B4 will be used for an additional key point regression model, as this model offers the best trade-off between the number of parameters and ImageNet top-1 accuracy, as shown in Figure 3.4. This will allow for remarks to be made about the compound scaling method in the context of pose estimation.

3.3 Object Detection Algorithms

In this section, state-of-the-art object detection models will be reviewed. An object detection model will be used to produce bounding box coordinates that enclose the hands. A cropped image based on the bounding boxes will be used for key point estimation. This approach has been selected as it reduces the number of background pixels that are being inputted into the model, and therefore, the model will likely be able to make better predictions. Feasible methods will be selected for comparison for use in the automatic labelling study proposed in this study.

Humans can observe images and the world around them instantly. As eloquently expressed by Redmon et al. (2016), humans can identify what objects are around them, where these objects are located, and how to interact with objects and their surrounding environment. The human biological visual processing system allows them to process visual stimuli fast and accurately. This enables the brain to process a response to visual stimuli and perform complex tasks with little conscious input, such as driving a car. Algorithms that can detect and localize objects with the same inference speed and accuracy as humans would allow tasks such as driving a car to be undertaken by computers. Further, accurate algorithms that can identify and locate visual stimuli allow for a multitude of new technologies that could benefit humankind through computer-based or robotic systems (Redmon et al. 2016).

One of the fundamental visual recognition problems in computer vision is referred to as 'object detection'. Object detection predicts not only object categories but also the location of each object in the image. Formally, this would be referred to as combining image classification and object localization within the field of computer vision (Wu et al. 2020). The state-of-the-art object detection algorithms are currently underpinned by deep learning algorithms, divided into two key structures. Firstly, there are two-stage object detectors such as Region-Based CNN (R-CNN) and developments of this approach such as fast and faster R-CNN (Girshick et al. 2014, Girshick 2015, Ren et al. 2015). Secondly, there are one-stage object detectors such as YOLO and similar variants such as revisions of YOLO and single-shot multibox detectors (Redmon et al. 2016, Bochkovskiy et al. 2020, Liu et al. 2016). Multi-stage detectors typically offer better accuracy and are considered state-of-the-art on open-source benchmark data sets, whereas single-stage detectors require less memory and can run object detection in real-time and there-

fore have far greater inference speeds (Wu et al. 2020).

3.3.1 Two-stage detection models

As explained by Wu et al. (2020) two-stage detection models split object detection into two distinct stages. A sparse set of proposal regions is generated in the first stage, and feature vectors are generated for object class predictions in the second stage using a CNN for each proposal region. The proposal regions are generated such that all objects within a given image belong to at least one proposal region. Each proposed region is then passed into a deep learning algorithm that classifies each proposal as a predefined class label or background in the image. In some models, there is also additional refinement of the original proposal region (Wu et al. 2020).

The most famous multi-stage detector is known as R-CNN, which has been improved since 2014 producing newer algorithms such as Fast R-CNN, Faster R-CNN and SPP-net (Wu et al. 2020). Explained by Girshick et al. (2014) in R-CNN's release paper in 2014, this new method offered state-of-the-art performance on a public benchmark data set (the Pascal VOC2010), offering a mAP of 53.7% which beat the predecessor by 13.3%. The R-CNN pipeline has four distinct components. Firstly, for each image, approximately 2000 sparse proposal regions are generated using 'Selective Search', which is an algorithm designed to reject regions of the image which can easily be identified as background areas (Uijlings et al. 2013). Secondly, each proposal region is cropped to a fixed size and passed into a CNN that outputs a feature vector. Thirdly, each feature vector is passed into a separate support vector machine (SVM) classifier to classify the learned visual features of each proposed region into one of the pre-defined classes. Lastly, the original proposal regions were refined to tightly formed bounding boxes around the classified objects using feature maps extracted from the CNN. R-CNN implemented the idea of transfer learning by using an encoder network that was pre-trained on the ImageNet data set but re-training final fully connected layers for the detection application. Utilising transfer learning offered performance gains, faster training times, and more robust learned visual features for classification by the SVM (Girshick et al. 2014).

R-CNN was a novel approach that achieved state-of-the-art performance by utilising learned visual features extracted using a CNN encoder, which was more robust and informative than visual features learned by hand-crafted feature extractors, as used in previous methods as detailed by Wu et al. (2020). Despite offering state-of-the-art performance, there were several flaws associated with this two-stage object detection pipeline. Firstly, there were issues associated with repeated computation as the proposal regions often overlapped, and the visual features for each proposal region was extracted by the CNN encoder *individually*. This meant that many pixels from the original input image were passed through the CNN multiple times, which massively hinders inference speeds. Furthermore, the Selective Search algorithm operated using low-level visual cues, which meant that it struggled to produce proposals for images with complex back-

grounds and was not able to benefit from parallel computation benefits offered by GPUs. Due to each stage of the network being fully independent of one another, the entire pipeline could not be optimised end-to-end, making it far more challenging to reach a global optimum for the two-stage system (Wu et al. 2020). Despite this algorithm inspiring many other object detection models with different stages, the inability to optimise two-stage models in an end-to-end manner remains an inherent limitation of models structured this way therefore, they will not be considered for hand detection in the automatic labelling system in this study.

3.3.2 One-stage detection models

One-stage detectors do not have separate proposal generation and region classification, and they perform both of these tasks using one network. This is achieved by considering all regions in the image as potential objects. All regions are then classified as a pre-defined class or as background (Wu et al. 2020).

YOLO

You Only Look Once (YOLO) was proposed by Redmon et al. (2016), and was unique to previous object detection algorithms in the sense that it attempted to solve object detection using a regression-based approach. This single regression problem took the pixels of an image as input and directly outputted bounding box coordinates and class probabilities, making it particularly simple when compared with two-stage detectors. YOLO utilises a single CNN that passes each input image through the network in a single forward pass to predict object classification and localisation, eliminating the repetition of computation often present in two-stage detectors. Furthermore, multiple bounding boxes for multiple objects are predicted in an image simultaneously, making it suitable for real-time two-object detection. The YOLO framework allows end-to-end training through the use of a single CNN and supports real-time detection whilst retaining high accuracy and precision. YOLO uses a network architecture that was inspired by GoogLeNet for image classification (Szegedy et al. 2015). The network used in YOLO has 24 convolutional layers followed by two fully connected layers. The network is implemented on 'Darknet', an open source neural network framework written in C and CUDA. The network was trained on ImageNet-1000 for approximately 1 week and the model achieved a top-5 accuracy of 88% on the ImageNet 2012 validation set. Due to having a lightweight network architecture and being implemented on the Darknet framework, YOLO can make predictions at 45 FPS with a yet faster 'Fast YOLO' being able to make predictions at 155 FPS as it had fewer convolutional layers (9 Conv layers instead of 24) (Redmon et al. 2016).

As explained in by Redmon et al. (2016), YOLO places an S $\times$ S grid onto the input image (this was a 7$\times$7 grid in the first version of YOLO as shown in Figure 3.5). If the centre of an object falls within a given grid cell, that cell is responsible for detecting that object. Each grid cell makes predictions for B bounding boxes and confidence scores based on how accurate the

predicted box is based on the IOU between the predicted box, and the ground truth labelled box. Each cell produces a vector containing a numerical representation of the confidence that the cell contained an object, the coordinates of the centre of the bounding box, the height and width of the bounding box, and the class of the object. YOLO uses a multi-part loss function which optimizes the sum of squared error (SSE) for localization loss, confidence loss, and classification loss simultaneously (Redmon et al. 2016).

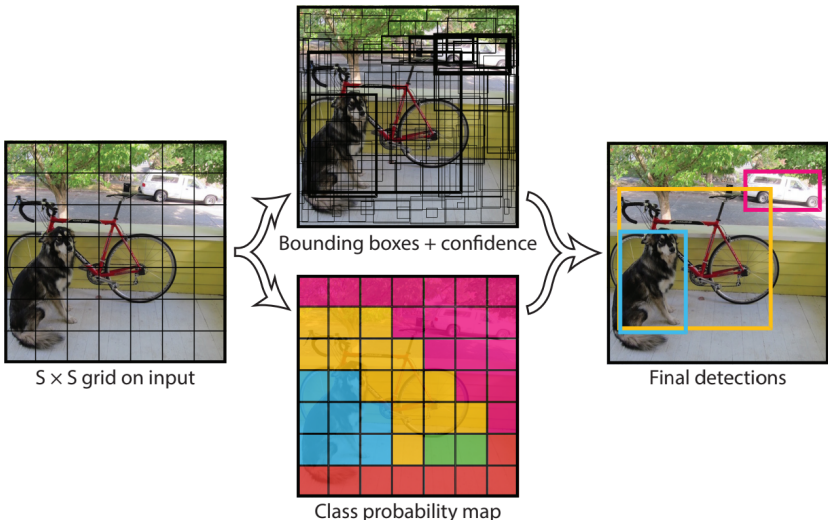

Figure 3.5: YOLO Method (Redmon et al. 2016).

The design of YOLO poses high spatial constraints on predicted bounding boxes. As the image is split into an S×S grid and each cell can only predict up to two objects from only one class, the model struggles to predict multiple small objects in close proximity to one another in image space. As highlighted by the author, the model struggles to detect an entire flock of birds. Another issue is that only the last feature map is used for making predictions. This means that only relatively course features are used for predicting bounding box coordinates, and therefore struggles to make predictions on objects at different scales or aspect ratios that are not present in the training data. A further issue is that the main source of error is localization. The author speculates that this is caused by treating errors the same in small and large bounding boxes, as a small error in a large bounding box has a much smaller impact on IOU than a small error in a small bounding box (Redmon et al. 2016).

YOLOv2

Redmon & Farhadi (2017) later proposed an improved version of YOLO named YOLOv2, which addressed the limitations of its predecessor. This model maintained real-time inference speeds but achieved significantly improved accuracy. YOLOv2 exhibited state-of-the-art performance on public benchmark data sets, which was a tremendous contribution to object detection

and computer vision. This was achieved mainly by three core design changes from the previous version of YOLO.

As highlighted by Redmon & Farhadi (2017), a regularisation technique called Batch Normalization (BatchNorm) was used in YOLOv2 on all convolutional layers. BatchNorm essentially takes all the outputted weights and bias terms from a given convolutional layer and normalises the values so that they all lie on the same scale. This helps prevent overfitting and the exploding gradient problem too. Furthermore, dropout layers can be removed without the model overfitting as BatchNorm is a regularisation technique. By using BatchNorm, the mAP of YOLO improved by 2% (Redmon & Farhadi 2017).

Another key improvement made in YOLOv2 was by training the backbone network architecture on higher resolution images. Redmon & Farhadi (2017) states that all previous state-of-the-art object detection methods utilise a backbone that is trained on ImageNet for object classification. After the release of AlexNet, most of these networks take an image input size of 256×256. For YOLOv2, the network is initially trained on input images that are 224×224 and then the classification network is fine-tuned for a further 10 epochs, which take input images at a resolution of 448×448. This allows the network to learn more fine-grained, higher resolution visual information and results in the mAP of YOLO improving by nearly 4%.

Lastly, as outlined by Redmon & Farhadi (2017) the first version of YOLO extracts visual features from an image using a convolutional network and then makes coordinate predictions for the bounding box using fully connected layers after the feature extractor. YOLOv2 adopts a technique that was used in Faster R-CNN in which rather than predicting coordinated directly, they predict bounding boxes by using hand-picked priors (Ren et al. 2015). Instead of fully connected layers, the network can use convolutional layers to predict offsets and confidence values for hand-picked priors (Redmon & Farhadi 2017). Since the network now contains no fully connected layers, only convolutional layers, and the network only has to predict prior anchor box offsets rather than full bounding box coordinates, the problem is simplified, and learning is easier for the network (Redmon & Farhadi 2017). Extending this, Redmon & Farhadi (2017) proposed a method for automatically finding more appropriate anchor box priors than hand-picked priors by using k-means clustering on the training set to find more appropriate priors, helping both optimization and localization (Redmon & Farhadi 2017, Wu et al. 2020).

Another significant contribution of YOLOv2 was the backbone feature extractor that is used. Redmon & Farhadi (2017) puts particular emphasis on the applications for detection, such as robotics and driver-less cars, relying on low-latency predictions. This requires that the backbone architecture has to be accurate but also must have rapid inference speeds. Many object detection frameworks rely on VGG-16 as the backbone architecture as a base feature extractor as it is

highly accurate and performs a top-5 accuracy score of 90%. However, it is excessively complex and requires 30.69 billion floating-point operations for a 224×224 image. Therefore YOLOv2 proposes the Darknet-19, which contains 19 convolutional layers and 5 max-pooling layers. This was inspired by GoogLeNet and is an improved backbone to the network used in the first version of YOLO. It only requires 5.58 billion floating-point operations per image and achieves a top-5 accuracy of 91.2%.

YOLOv3

YOLOv3 was released by Redmon & Farhadi (2018) which included a few small design changes and a new larger network that made system more accurate whilst still maintaining real-time inference speeds. The backbone used in YOLOv3 takes the previous backbone used in YOLOv2 and implements shortcut connections proposed in ResNet by He et al. (2016). In YOLOv2, Darknet-19 was used which consisted of 19 convolutional layers with 5 max pooling layers, the new network used in YOLOv3 incorporates successive 3×3 and 1×1 convolutional layers which allow shortcut connections and a much deeper backbone to be used. The new backbone has a total of 53 convolutional layers and is therefore named Darknet-53. Table 3.1 compares Darknet variants with Resnet variants.

Backbone	Top-1 Accuracy	Top-5 Accuracy	Billion Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 3.1: Comparing ResNet and Darknet Variants (Redmon & Farhadi 2018)

Table 3 was originally produced by Redmon & Farhadi (2018) and each network was trained with the same parameters and settings, taking input images that were 256×256 on a Titan X GPU. This shows that by implementing shortcut connections, Darknet-53 obtains accuracy on par with ResNet-152, which was the state-of-the-art performance in classification, with Darknet-53 having inference speeds that are 2× faster as indicated by the 'FPS' column in Table 3. Furthermore, Darknet-53 achieved the highest measured billion floating-point operations per second at 1457 BFLOP/s, meaning that the structure of the network is more optimal for utilising a GPU for training when compared to state-of-the-art ResNet variants.

Another critical change in YOLOv3 is that it makes predictions across scales, rather than just based on the final output of the feature extractor, and this method was inspired by feature pyramid networks (Lin et al. 2017). This involves extracting feature maps at different stages in the network hierarchy and up-sampling them to the same dimension as the final feature map that is outputted from the network. The three feature maps are concatenated to produce a 3-

D tensor in which anchor box offsets are predicted for the feature map at each hierarchical layer in the network that is considered. The final bounding box prediction benefits from the prior contribution made on predicting bounding boxes for previous feature maps. Furthermore, this method allows finer-grained visual information to be retained from earlier layers and more meaningful semantic information to be up-sampled for making final predictions (Redmon & Farhadi 2018).

YOLOv4

At the end of the YOLOv3 paper, the researcher and developer of the three versions of YOLO Redmon & Farhadi (2018) stated that they would no longer be undertaking computer vision research due to ethical concerns of privacy and military applications. This left a vital question in the computer vision community - would there be new and improved versions of YOLO in the future? Bochkovskiy et al. (2020) published YOLOv4, which once again improved the accuracy to achieve state-of-the-art performance in real-time. This paper offers many contributions with several design changes that were evaluated in an ablation study, such as a new backbone, new regularization techniques, new activation functions, and new data augmentation techniques. Full detail of these techniques are available in the original paper, and considering that all aforementioned design choices included in the three previous versions of YOLO are still included in YOLOv4 (Bochkovskiy et al. 2020), detailing all the new design choices are not possible in this review due to project constraints. All of the core design decisions for the YOLO algorithm have already been outlined in this section which are still essential building blocks of YOLOv4. However, some key benefits and techniques will be explained in a high-level overview.

Bochkovskiy et al. (2020) explains that the current issue with CNN-based object detection systems is that the most accurate systems do not operate in real-time (30 FPS or more), and they require multiple GPUs for training with large mini-batch sizes. This key issue is addressed in YOLOv4, in which they take YOLOv3 along with various other design decisions to produce a system with state-of-the-art accuracy that can be trained on a single conventional GPU such as an NVIDIA 1080 Ti or 2080 Ti. YOLOv4 was the first representative one-stage object detection model released with state-of-the-art performance since the release of YOLOv3, in which significant gains in detection accuracy had occurred. On the MS COCO data set with a 416 $\times$ 416 input image size, YOLOv3 achieved an AP of 31% at 35 FPS whilst YOLOv4 achieved an AP of 43% at 31 FPS. This is a significant increase in performance whilst still maintaining an inference speed of at least 30 FPS meaning and is therefore still considered a real-time detection system Bochkovskiy et al. (2020). This method achieves state-of-the-art accuracy and achieves real-time inference speeds and is therefore, will be used to detect the presence of a hand in the ML pipeline.

Chapter 4

Methods

This chapter explains the methods used for the analysis of this project. Firstly, it is important to highlight at which stage of the existing workflow the labelling phase exists for DLC and where precisely the contributions of this project will be in the DLC work pipeline. See Figure 4.1, which displays a high-level overview of the DeepLabCut workflow in the first flowchart. This involves using the DLC GUI to label video frames which are then inputted into a deep neural network for training. The core contribution of this project is to fully automate the labelling phase, meaning that you do not have to manually label the frames using the GUI whatsoever. Therefore, see the second flowchart that highlights a new proposed DLC workflow. This involves creating a DLC project and extracting the frames as usual but passing these into an 'Automatic Labelling System' (stage 2.) to label key points. Each frame of video will be labelled automatically and then passed back into the DLC workflow as usual.

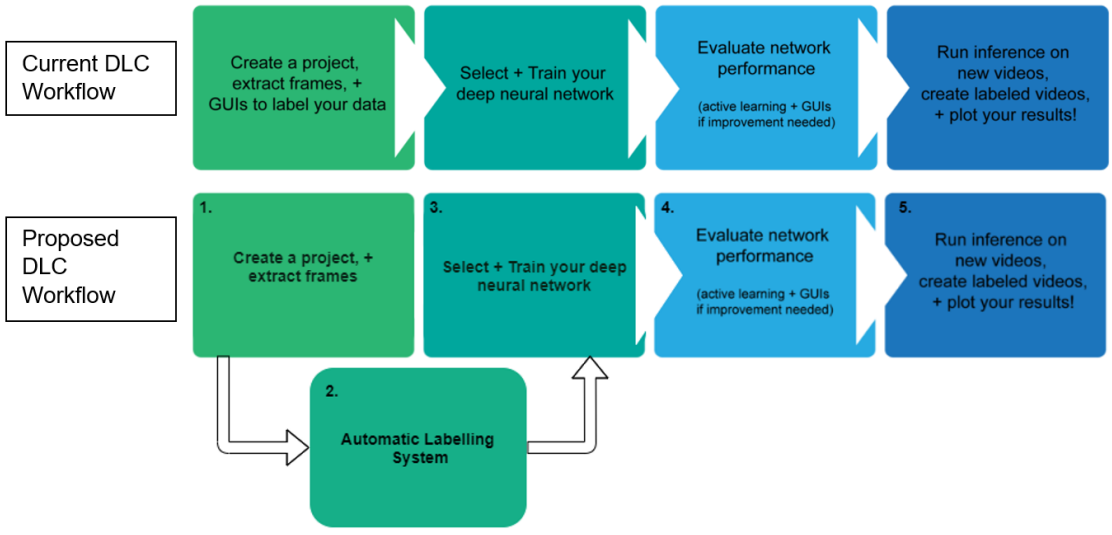

Figure 4.1: Current and proposed DeepLabCut Workflow

4.1 Automatic Labelling System: Pipeline

In this section, the design of the automatic labelling system pipeline will be detailed. The pipeline illustrated in Figure 4.2 shows example inputs and outputs to the right-hand side of the flow chart, demonstrating data examples at each stage. This pipeline was inspired by Google's solution MediaPipe Hands which was released by Zhang et al. (2020) which similarly uses a palm detector called 'BlazePalm' to detect the presence of a hand. BlazePalm outputs bounding box coordinates, which is used to produce a cropped image of the hand. This cropped image is then passed into a hand landmark model, which predicts 21 coordinates of the main joints in the hand (Zhang et al. 2020). Unfortunately, this solution is not suitable for the data set used in this project as the palms are not visible in any of the frames, and therefore, the presence of the hand would not easily be detected by the model.

The first stage of the system pipeline is the 'Object Detection Model' to detect the presence of a hand. The Object Detection Model takes a video frame as input as demonstrated by the 1920×1080 resolution image to the right-hand side. The hand detection model will output the object class, which will always be 0 corresponding to 'hand', the coordinates of the centre of the bounding box and the boxes height and width for this object. The green dot in Figure 4.2 demonstrates the absolute coordinates for the centre of the box. The green dot coordinates are visualised in the second item in the example column in Figure 4.2. See the red square in Figure 4.2 indicating the bounding box. Two object detection models will be used for this task and compared to identify a preferred model, which are YOLOv4 and YOLOv4-tiny.

These bounding box coordinates then need to be passed into a 'Cropping Function'. In which the image is cropped so that it only contains pixels within the bounding box. The cropped image contains less visual noise as fewer background pixels are present. The image's resolution in the example in Figure 4.2 has been reduced from 1920×1080 to 768×768. Therefore, more than 72% of the redundant pixels have been removed for passing the image of the participant's hand into the key point regression model.

The cropped image of the hand is inputted into the 'Key Point Regression Model'. This convolutional neural network will output a 4-tuple of the (x,y) coordinates for the thumb and the index finger. Two backbone architectures will be used for this task and compared to identify a preferred model, which are EfficientNet-B0 and EfficientNet-B4.

The key point coordinates have been predicted on the axis of the cropped image. To be inputted into DeepLabCut, they must be converted back to coordinates for the full resolution image. This will be achieved by passing the key point coordinates on the axis of the cropped image into a 'Coordinate Conversion Function', which will use the bounding box coordinates for each image to convert the key point coordinates relative to the axis of the 1080p image.

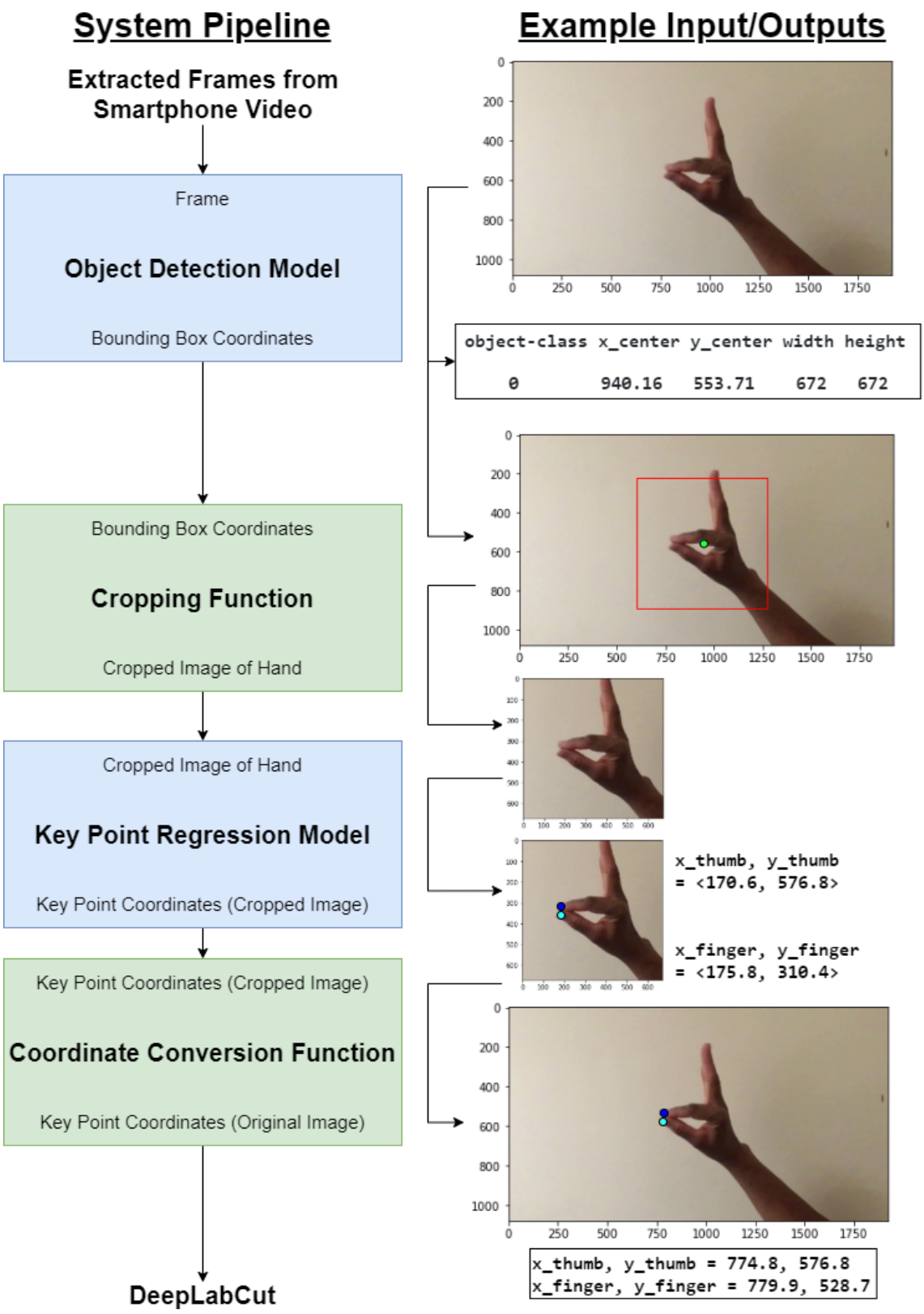

Figure 4.2: Proposed ML Pipeline for the Automatic Labelling System

4.2 Requirements

The automatic labelling system requires three core components in order to meet the aims and objectives of this project, as illustrated in Figure 4.3:

1. There will be a 'Data pre-processing phase' to ensure it is in the correct format specified for the system that has been designed or selected.

2. There will be a 'Model Training phase' as the pre-processed data will need to be passed into the labelling system to build and train the models. All relevant figures, diagrams and outputs will be collected.

3. There will be an 'Evaluation phase' in which all figures, diagrams and outputs produced by the model will need to be discussed and analysed.

Multiple models will be compared for the object detection and key point estimation models, with a preferred solution identified for each model. 5-fold cross-validation will be undertaken to evaluate the robustness of the key point regression model results. In which five different training and test splits will be taken. For each split, both models will be trained, and the MSE of the models will be compared, and an error analysis will be undertaken.

The Object Detection model will be acceptable if it can crop all the images to contain the two key points of interest, the tip of the thumb and the index finger. The core focus of this project will be identifying a key point estimation model that is both lightweight and accurate enough to produce key point coordinates for DeepLabCut automatically.

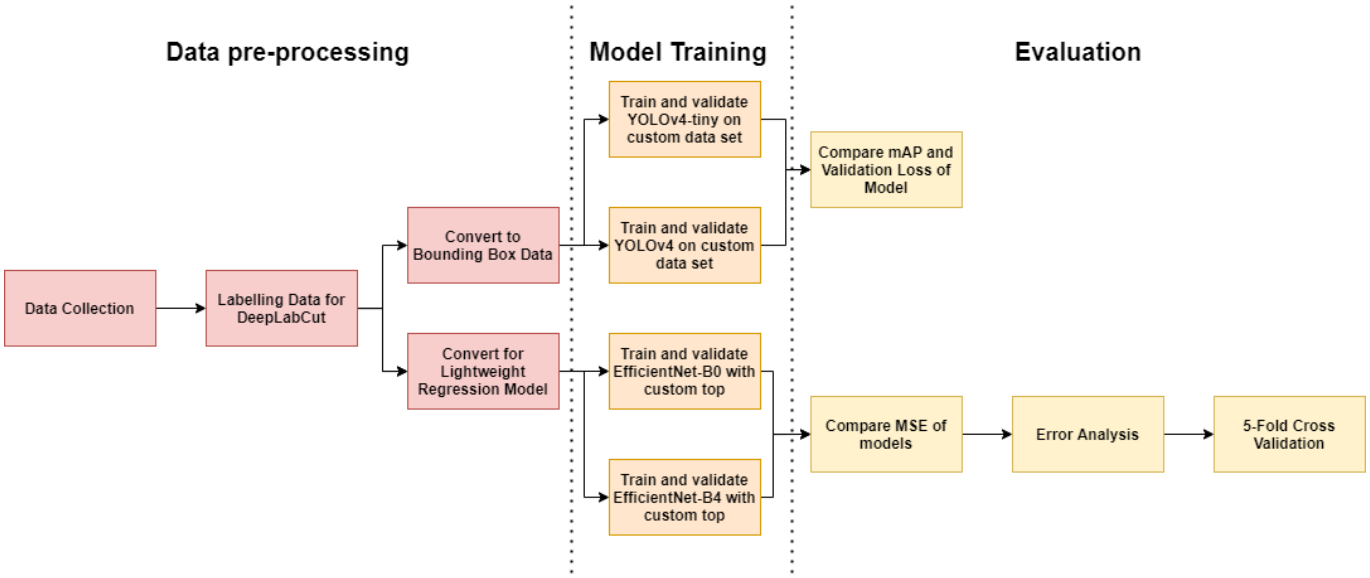

Figure 4.3: Workflow of this project

4.3 Data Collection

The data used in this project is a subset of a data set that was previously used in two previous publications with authors from Leeds Institute of Health Sciences in partnership with researchers from other universities (Zhao et al. 2020, Williams et al. 2020).The data that has been used for this project contains a total of 75 video recordings of human hands (right hands only). The videos were captured of 40 Parkinson's patients, 20 control patients (typically the spouse of a given Parkinson's patient), and 15 additional control patients (either hospital staff or children). Each of the participants had given written consent for their data to be used in the study. Two videos were rejected early in the project for all students using this data, as they were filmed at 30 FPS which would likely cause issues if they were to be used in training a DeepLabCut model.

Each of the 40 Parkinson's Patients had previously been diagnosed by a consultant neurologist specialising in movement disorders at Leeds Teaching Hospitals NHS Trust. The Parkinson's Patients were invited to attend a research clinic appointment specifically to gather this data set. Each of the diagnosed patients was subjectively and objectively in an 'on' motor state, which means that they met the following criteria:

1. The patients reported that they felt 'on'. This is a widely accepted and understood term that refers to patients ability to have an awareness that their medications are working effectively and that the symptoms of their PD have been reduced by these medications.

2. The consultant neurologist reported that the patient looked 'on'. This term is clinically accepted that a trained specialist can observe a response in symptoms as a result of a medication taken by a PD patient.

3. The PD patient was currently taking their medication as prescribed and no medication had been missed prior to recording.

Control participants were invited that were companions or family members of the PD patients, and these were typically the spouse of the patient. There were also additional control participants that were staff members at Leeds General Infirmary. All control patients had no previous neurological diagnosis and were not on any medication that could cause PD like symptoms such as tremor, rigidity, bradykinesia or any other movement impairment.

The remaining 73 videos used for this project were filmed using an integrated smartphone camera on an iPhone SE. The integrated video camera was configured to record at 60 FPS at 1920×1080 pixels (1080p). The smartphone was placed on a tripod, and no additional lighting was used to enhance the recording, only ambient lighting in the hospital room. Each of the participants placed their elbow on the arm of a chair and lifted their forearms to a 45° angle from the ground. The patient's hands and arms were not fixed or held in any other way and were completely free to move as advised by 'Item 3.4 of the Movement Disorder Society revision of

the Unified Parkinson's Disease Rating Scale' (MDS-UPDRS) (Goetz et al. 2008). The only visible body parts in the video frame were the hand and forearm. The distance between the smartphone camera and the patient's arm was approximately 1m. However, this was not strictly defined, and the distance does vary within the sample.

Each of the participants in the study was instructed to tap their index finger and thumb "as quickly and as big as possible" for at least 10 seconds. Digits 1 and 2, the thumb and index finger, respectively, were closest to the camera to try and prevent self occlusions of the key points of interest. No explicit instructions were given to the patients about the required position of digits 3 to 5. Although the researcher gave a brief demonstration to the patients in which digits 3 to 5 were fully extended, some patients performed the tapping exercise with digits 3 to 5 fully retracted (as if digits 3 to 5 were forming part of a closed fist). Each video was manually edited to make the total length of each video 11 seconds, 1 second of the patients hands stationary prior to initiating the tapping exercise, followed by 10 seconds of the patient completing the tapping exercise.

4.4 Data Preparation

The data preparation process used to create the data sets for the models used in this study will be outlined in this section.

4.4.1 Labelling key points for DLC

Of the remaining 73 videos in the data set for this project, 63 (86%) videos were allocated for training and 10 (14%) were allocated for testing. The training data set needed key points annotated, such as tips of the fingers and thumb, which will be explained later in this section. As there were five students in the cohort completing independent research projects on the same data set, the 64 training videos were split between the five students for labelling, as this meant the labelling phase of the project for each student could be accelerated.

Each video is approximately 11 seconds long and was filmed at 60 fps for 660 frames per video. To reduce the labelling burden whilst retaining sufficient spatial semantics of the finger-tapping exercise, every 4^{th} frame was labelled. The frame rate of the videos was reduced from 60 fps to 15 fps to reduce the total number of frames for labelling from 44,000 to 11,000 frames approximately. Labelling was completed by each of the 5 students on their own personal devices in a local environment using the DeepLabCut Graphical User Interface (GUI). The process took approximately 30 person-hours to complete due to the repetitive nature of the task. The aims and objectives between each of the student's research projects varied, and not all students would use each of the 6 points labelled on the training data. However, all students labelled all 6 points to ensure every individual had sufficient points labelled for their task at hand. The position of

the points labelled are summarised in Figure 4.4.

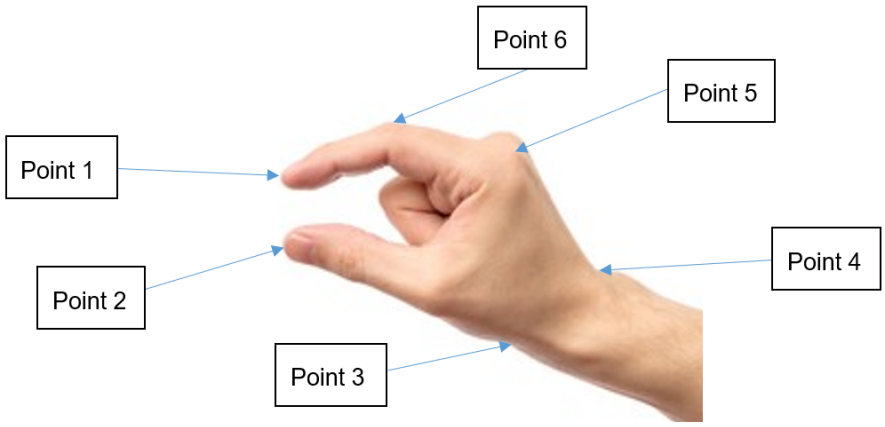

Figure 4.4: Hand Labelling Protocol

Written instructions were given detailing the exact position of each of the six key points, as explained in Table 4.1. The two points of paramount importance are Points 1 and 2, as the distance between these two points produces the time series data used to classify each participant on the MDS-UPDRS.

Point	Description
1	Tip of the index finger (the point where finger and background meet)
2	Tip of the thumb (the point where the thumb and background meet)
3	Wrist joint at the bottom of the thumb muscle (the point where joint and background meet)
4	Wrist joint on top side
5	First knuckle of the index finger (the top of the peak)
6	Second knuckle of the index finger (the top of the peak)

Table 4.1: Hand labelling protocol. Each key point is visualised in Figure 4.4.

4.4.2 Pitfalls of labelling approach and data quality issues

A common issue with deep learning based key point estimation can occur due to issues with the labelled training data. For example, when labelling pixel-based coordinates for key points on frames of video that are 1920 $\times$ 1080 (1080p) resolution, there are 2,073,600 pixels in each of the frames. Therefore, there is always likely to be a level of human variability as often, despite having a detailed labelling protocol, there are geometric limitations to the body parts that are being labelled. Further, a point described as 'the tip of the index finger (the point where the finger and background meet)' as described in TABLE will be inherently ambiguous as there is no clear vertex for an index finger as the fingertips is an irregular curve. The exact 'tip' of the finger may change based on the angle from which it is observed as the finger moves in

relation to the perspective of the smartphone video camera. Therefore, this makes labelling the tip ambiguous. Similar statements apply to the other six key points labelled in this data set.

The implications of human labelling variability were summarised in detail by Mathis et al. (2018) in which the author explains how this variability serves as a lower limit to the accuracy of the model in predicting key points for unseen test data. This is shown in Figure 4.5 in which the RMSE (pixels) sits below the human variability within the training data, showing that the model is able to track the labelled data with extremely high accuracy. However, the RMSE (pixels) for the test data converges to the human variability within the training data. It is worth noting that the model can still predict the test data extremely accurately as an approximate 2.5 pixel RMSE is for an image that contains over 2 million pixels.

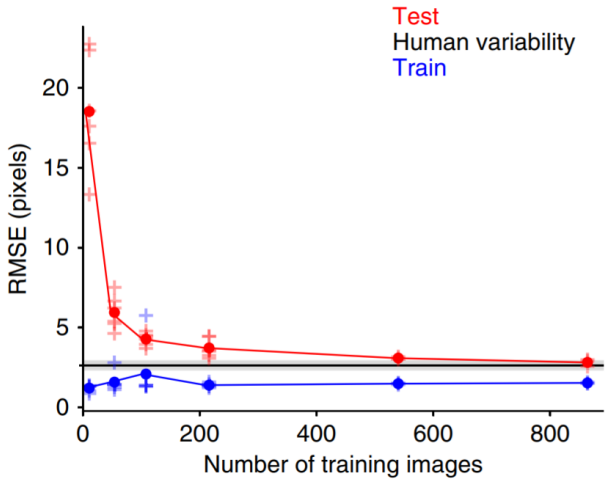

Figure 4.5: Human variability in labelling (Mathis et al. 2018)

The extent of human variability is likely to depend on many factors. Including the animal or body part that is being labelled, the proportion that the body part occupies in the video camera field of view, and how careful the individual(s) are in the labelling process. The tracking accuracy using DeepLabCut on this data set has been demonstrated to be mean absolute error of 8.39 pixels when trained over 1 million iterations on approximately twice the amount of videos present in comparison to the data used in this study (Zhao et al. 2020). However, the data was relabelled by five students independently in this study, meaning there is likely to be further human variability between each individual. Therefore, it is speculated that a human variability of 8 or 9 pixels can be considered as a likely minimum variability in the training data used for this project. The human variability will likely be greater than this, and this will be considered when evaluating the performance of the final model.

A further issue with labelling key points for animal pose estimation is referred to as 'soft-tissue artefact', which is a commonly used term amongst movement scientists and refers to the deformation of tissues lying beneath the skin such as muscle or fat, which serves as a common

obstacle to tracking accurate skeletal kinematics (Camomilla et al. 2017). In the data used for this project, for example, it may be inherently difficult to label the wrist joint on the top side as often the joint may not be clearly visible if a participant in the study does not have pronounced bones in this area due to the bones being covered by a sufficient amount of fat or muscle. See Figure 4.6 below, which highlights the ambiguity in the position of the wrist joint on the top side due to soft tissue artefact as indicated by the annotated yellow box. Soft tissue artefact was something that the author of this paper observed to be a potential issue for key points labelled 3,4,5,6 in Figure 4.4 and Table 4.1 in certain patients and is therefore speculated to increase the human labelling variability present in the training data. These points are the upper and lower wrist and the two labelled knuckles on the index finger. See the yellow box in Figure 4.6 to visualise how the exact pixel location is likely to vary for these key points.

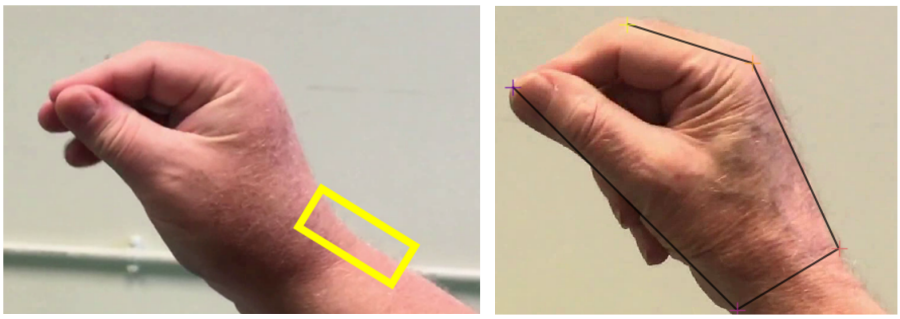

Figure 4.6: Data quality issues. Left: Soft Tissue Artefact. Right: Self-occlusion.

Another frequent data quality issue present in the data set were self-occluded key points, as shown in Figure 4.6, in which the thumb occludes the tip of the index finger. As per the guidance of the DeepLabCut developer, these labels were skipped and not labelled (Nath et al. 2019). This was only an issue for the tip of the index finger in the data which means that these frames should not be used in training the finger detector in this report. Therefore there are 156 frames that were omitted from the data set. Only the frames with self-occlusions were omitted, not the entire video allowing any frame with complete labels to be utilised for training.

Some data quality issues arose from the video camera quality itself. As previously stated, the video was captured using an integrated smartphone video camera (iPhone SE) which was configured to record video at 60 fps at 1080p. The videos were taken in ambient lighting conditions with no additional specialist lighting used to capture the video recordings. Figure 4.7 shows that the position of key point 6 (the second knuckle of the *index finger*) is hard to distinguish against the position of the second knuckle of the *middle finger* as it is difficult to make out the finger boundaries due to general blur in the image. This blur results in the camera not being focused on the hand, which is likely due to the poor lighting in this participants video recording. In Figure 4.7, it is clear that the lighting incident on the hand in the left image is of a much lower luminosity than the light rays incident on the hand in the right image. General blur caused by

the integrated camera not being properly focused on the hand is likely to further increase human variability within the training data.

Furthermore, see Figure 4.7 right, indicating a frame example in the data set in which the index finger is more blurry than the rest of the hand in the image. This phenomenon is referred to as 'motion blur' and is a common issue when capturing fast-moving objects in video footage (Dai & Wu 2008). This is a prevalent data quality issue in the data set, particularly with control participants rather than PD participants. This is because the control participants can perform the finger-tapping exercise at a much faster speed/frequency than the PD patients, resulting in more frames with motion blur occurring. As this issue occurs in multiple frames, it may not be a massive issue as the key point estimation model produced in this project may have the learning capacity to identify motion blur with the tip of the index finger. However, in cases of extreme motion blur where the tip is not visible in the frame at all, the model is likely to struggle to label the index finger. Motion blur is likely to increase the human variability of labelling this key point in the training data as the exact position of the index finger is not clear in the frame, as shown in Figure 4.7 right.

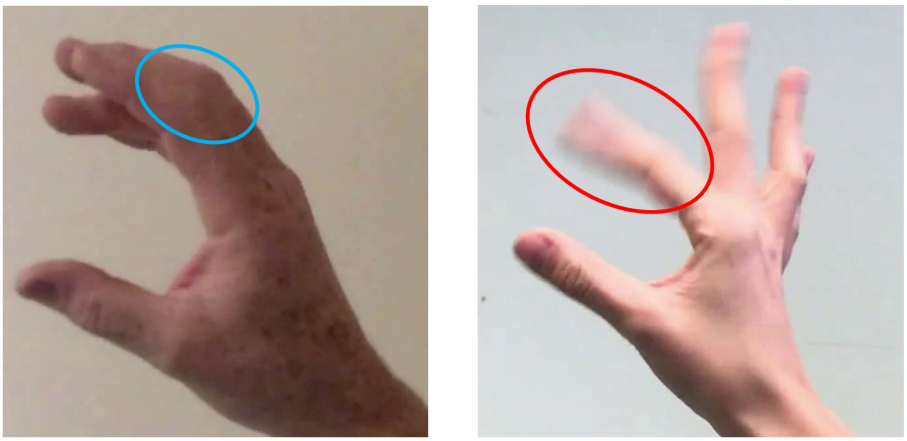

Figure 4.7: Additional data quality issues. Left: Blur. Right: Motion Blur.

Figure 4.8 shows a frequency distribution of the Euclidean distance between the labelled coordinates of the tip of the index finger and thumb in the data used in this project. The mean Euclidean distance between the index finger and thumb tip is 191.67, and the standard deviation is 152.05. The distribution peak is around 50 px, with over 1000 video frames having a Euclidean distance of 50 px. PD patients cannot reach the same peak amplitude as non PD patients, which contributes to the largest peak being below 150 px. Videos of all patients contain frames in the 'closed position', where the index finger and thumb are touching. Therefore, the 'closed position' is oversampled. Furthermore, the key point estimation model's accuracy will likely be overestimated because of this. The majority of samples in the data set are in the closed position, and the ground truth location of the tip of the index finger and thumb occupy a

very small amount of pixels. Therefore, it is likely that the model will easily predict key point locations for the closed position, but the majority of the error associated with predictions will be in the 'open position' at maximum amplitude.

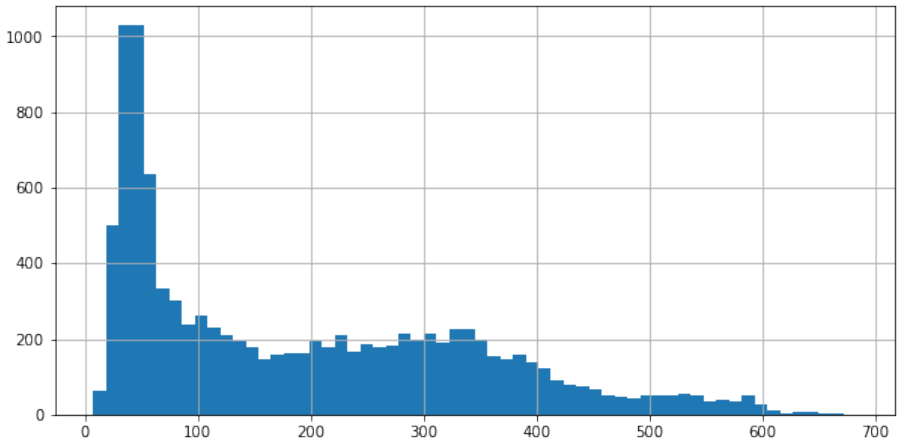

Figure 4.8: Frequency distribution of Euclidean distance between index finger and thumb

4.4.3 Converting DLC Keypoints to Bounding Box Labels

There are approximately 10,000 images of hands labelled with key point coordinates that were labelled by five students for DeepLabCut as outlined in Section 4.4.1. In order to make a bounding box model for object detection, ground truth bounding box labels were created for the training data. There are several methods to manually label bounding boxes onto images using GUIs as programmes such as 'v7labs' recommended by the developer of YOLOv4 (Bochkovskiy et al. 2020). However, this would be very time consuming, especially as the labelling phase for the approximate 10k images took five people a total of approximately 29 person-hours (of constant labelling at 100% efficiency of a highly repetitive task). Due to the time constraints of this project and also concerns of labelling accuracy if the task were to be rushed, a decision was made to use the existing key point labels and convert these into bounding box coordinates for the object detection model.

Several possible approaches can convert the existing key point labels from Section 4.4.1 to ground-truth bounding box labels. The objective of the bounding box model is to crop the hand contained in the image to reduce the input dimensions for the regression model. A lower resolution image will make the regression model more lightweight (and therefore suitable for mobile applications) while retaining as much spatial visual information about the hand and fingers. The key point regression model will take an input image with a fixed size and a square aspect ratio (i.e. $224 \times 224 \times 3$). Therefore, a decision was made to define the bounding box labelled coordinates to the same dimensions for all images. The benefit in doing this is that when the cropped image is outputted from the bounding box model, they are all the same dimension.

Therefore, when all cropped images that have been outputted from the bounding box model are resized from 768×768 to 224×224 for the key point regression model, they are all resized by the same scaling factor. This means that the pixel error of the model can be analysed and discussed with greater confidence that any given error is directly caused by the model, rather than being caused by different scale factors for images of different image resolutions. For example, if some cropped images were 448×448×3, and others were 672×672×3, the images would be reduced by a scale factor of 2 and 3, respectively. Furthermore, due to the different scaling factors, the human labelling error in the original key point labels would also be resized by different scale factors. If the bounding box model were designed to output bounding boxes that cropped the images of hands to several different sizes, then it would be more challenging to make accurate conclusions about the pixel error of the key point regression model. A further benefit to defining all the bounding boxes to a fixed size is that the proportion of the field of view occupied by the hand and fingers will vary between images. The distance between the camera and the participant's hand varied when the data was collected, despite being instructed to sit 1m from the camera. Therefore, the hands occupy a different proportion of the cropped images, as shown in Figure 4.9, two 1080p images from the original data. If two boxes of the same dimension were drawn around each hand, the hand in the left figure would occupy a large proportion of the cropped image, whereas the hand in the right figure will occupy a much smaller proportion of the cropped image.

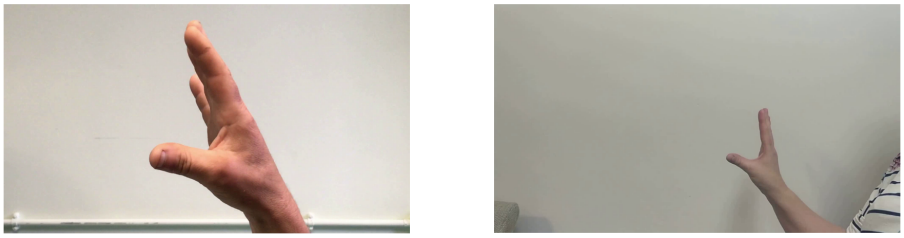

Figure 4.9: Different proportion of original image occupied by hand: Left: Large hand relative to 1080p image space. Right: Small hand relative to image space.

On the other hand, if a dynamic bounding box were used, it would perfectly crop all hands with no redundant background pixels. When this was resized to the original input image, the hand would always occupy the vast majority of the cropped region, with a similar proportion of the field of view occupied in image space for all examples in the training data. This could make the key point regression model less robust to labelling key points at different scales. Furthermore, the variability in the proportion of the cropped image occupied by the hand may enhance the model's ability to associate the spatial information of the hands with the labelled coordinates due to more variability in the training data.

YOLOv4 label format

A specific bounding box label format is required for the YOLOv4 object detection algorithm. The developer states that for each image in the training data, a '.txt' file must be created containing ground truth labelled annotations with the same filename prefixing the '.txt' and '.png', label and image pair@(Bochkovskiy et al. 2020). The '.txt' file must contain each of the values in Table 4.2 for each object in the image, with a new line for each object in the image.

Value	Description
object class	integer object number from 0 to (classes-1)
x_centre	x coordinate of the centre of the bounding box, normalised to the width of the image
y_centre	y coordinate of the centre of the bounding box, normalised to the height of the image
width	the width of the bounding box, normalised to the width of the image
height	the height of the bounding box, normalised to the height of the image

Table 4.2: Bounding box ground truth annotation format

All normalised values were calculated by dividing the absolute value by the total image width in each plane. For example, the normalised x coordinate of the centre of the bounding box is given by; ⟨x_centre⟩ = ⟨x_absolute⟩/⟨image_width⟩. So in summary, for a hypothetical training image named 'img1.png' there would be a corresponding file called 'img1.txt' containing the values shown in Figure 4.10. For absolute clarity the columns represent the ⟨**object-class**⟩⟨**x_centre**⟩ ⟨**y_centre**⟩⟨**width**⟩⟨**height**⟩ but these labels must **not** be in the file. This example has one line for one bounding boxes. For the data used in this project, there is only one class which is 'hand', and there is only one object per image. Therefore, there is only 1 line per '.txt' file.

File Edit Format View Help

```
0 0.3550332301062562 0.6240209325865981 0.4 0.7111111111111111
```

Figure 4.10: Example bounding box ground truth annotation '.txt' file

In order to convert the existing six key points that were labelled for use in DeepLabCut to an appropriately sized bounding box that is annotated in the format required for YOLOv4, the following steps were undertaken.

1. **Gather DLC key point coordinates into a single data frame** - Each of the 63 videos had an `h5` file containing approximately 165 rows of coordinates for the 6 key points, one row for each frame of video. All of the `h5` files were read in using the Python library `pathlib` into a `pandas` data frame. A final data frame with 10,371 entries was assigned to a variable named `master_df`.

2. **Image inspection to determine bounding box dimensions** - Using the Python `PIL Image` library, video frames were inspected for the largest hand to determine an appropriate bounding box size and position for bounding boxes. A decision was made that the centre point of each box would be defined as the mid-point between the thumb coordinate and the coordinate of the first knuckle of the index finger. After visualising different box dimensions such as 672 $\times$ 672 and so on, it was decided that for the YOLOv4 model, the most appropriate dimension was 768 $\times$ 768 as this dimension contained all key points of image whilst not going outside of the dimensions of the original image in the majority of photos. An example of the bounding box coordinates has been displayed in Figure 4.11, in which the green point is the box centre based off the midpoint between the tip of the thumb and the first knuckle of the index finger. The red box is a 768$\times$768 bounding box with the green point at its centre. Also, the red bounding box clearly contains the two key points of interest for this project which are the tip of the index finger and the tip of the thumb.

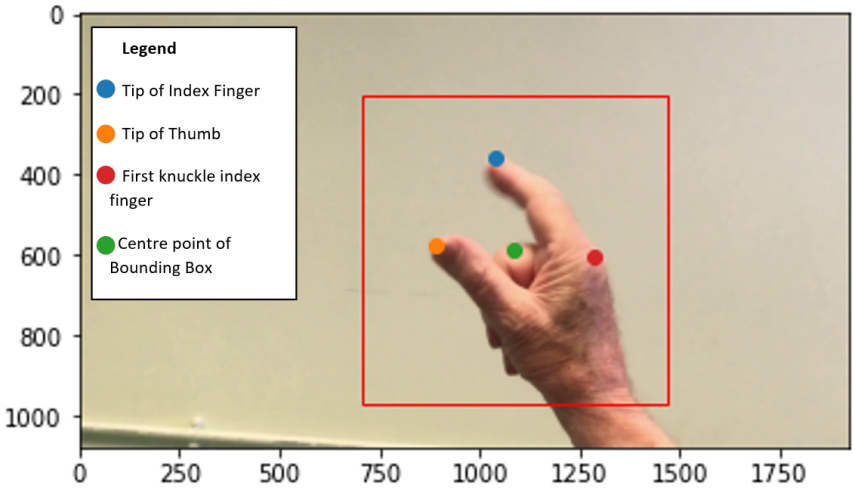

Figure 4.11: Defining bounding box dimensions using existing DLC key point labels

3. **Renaming all image files to have unique values** - For each patient, there were approximately 165 images that all were named in a sequence from 'img000.png' to 'img165.png' or similar. Therefore, as there were a total of 63 participants, there were also 63 images named 'img000.png' which would not be appropriate for YOLOv4 as each image needed to be renamed to have a unique value using the `pathlib` library in Python. Therefore, each image was renamed to contain the Participant ID prefixing the image frame number. For example, this would be 'P52_R_10s_lowfps_img068.png' in which this is the image for the PD patient 52, video frame number 068.

4. **Normalising the bounding box values and adding a class value for each image** - Each of the coordinates and the height and width of the bounding box dimensions were defined

on an absolute scale of the original input image. These needed to be normalised to be the correct format for YOLOv4. Therefore the height value was normalised by computing $768/1080 = 0.711$ and the width value was normalised by $768/1920 = 0.4$ and the (x,y) coordinates for the centre of the box were computed similarly taking 1080 for x coordinates and 1920 for y coordinates. New columns were added to the `master_df` data frame using the `pandas` library in Python for the normalised values. Furthermore, a class value was appended to `master_df` with a value of '0' for all images, as there is only one class for this data (hand) and only one hand per image. The class value must be a integer value. If this is a float value (i.e. 0.0), then YOLOv4 will interpret the second 0 as the x dimension for the centre of the bounding box and the model will systematically produce bounding boxes on the far left side of the image.

5. **Creating new data frame containing only relevant values** - A new data frame was created using the `pandas` library in Python and was named `yolo_df` this contained the bounding box annotations only. The object class, (x,y) coordinates of the centre of the box and height and width of the bounding box were merged into one column called 'txt', which contained the relevant values in the format that YOLOv4 requires. All the individual columns were dropped as they were no longer relevant. The final `yolo_df` is shown in Figure 4.12. The `drop.na()` method was used to remove all rows with empty values, reducing the data frame from 10,371 to 10,367. These were due to self-occluded index fingers.

	patient ID	img no	txt
0	OC01_R_10s_lowfps	img000	0 0.3550332301062562 0.6240209325865981 0.4 0....
1	OC01_R_10s_lowfps	img001	0 0.3565287015447635 0.6188124911855016 0.4 0....
2	OC01_R_10s_lowfps	img002	0 0.3555566538800887 0.6212641526875038 0.4 0....
3	OC01_R_10s_lowfps	img003	0 0.3543937992744019 0.622035317581063 0.4 0.7...
4	OC01_R_10s_lowfps	img004	0 0.35556541973734 0.6196958633162866 0.4 0.71...
...	...	...	...

Figure 4.12: Pandas dataframe named 'yolo_df' containing formatted bounding box values

6. **Creating '.txt' files** - Finally, a '.txt' file was created for each frame using a Python filehandle, which took the 'patient ID' and 'img_no' columns displayed in Figure 4.12 for the file name and the 'txt' column for the contents of the annotation file.

7. **Creating training and validation data** - All of the images and annotation files were then added to a single folder and split into training and validation data. This was done using the `pathlib` and `shutil` libraries in Python. The data was split so that there was 80% of the data used for training and 20% of the data used for validation. These folders were

compressed into '.zip' files to be uploaded to a Google Drive for use in Google Colab. Resulting in a total of 8268 training samples and 2102 validation samples.

4.4.4 Converting DLC Keypoints to data set for Key Point Regression model

Once the DLC labelled data was converted into a data set with labelled bounding box coordinates to build the object detection model, an additional data set was required to build the key point regression model. As the bounding box model has been designed to output cropped images that are fixed in dimension, these could be cropped prior to building the object detection model using Python to enable both the object detection model and the key point regression model to be built simultaneously. Therefore, the ground truth bounding box coordinates were used to crop all the original images.

Cropping the images

An issue was discovered with the data set for the object detection model, shown in Figure 4.13. The 768×768 bounding box ground truth annotations did not contain the tip of the index finger in all frames. Therefore, to produce the data set for the key point regression model, the ground truth bounding box dimensions were increased to 1120×1120 to ensure that the cropped images contained both the tip of the index finger and thumb for all video frames.

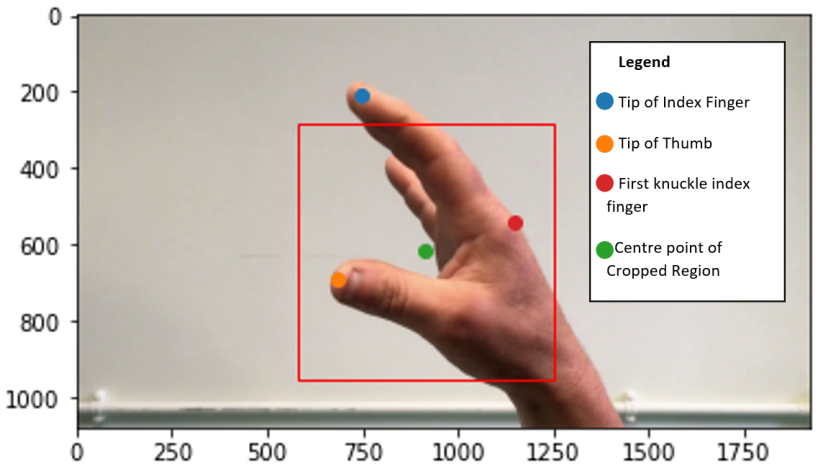

Figure 4.13: Issue caused by 768×768 bounding box: Index finger outside of box

The images were cropped in Python using the following steps:

1. **Copy original dataset and create dataframe of all coordinates** - All original images from the DLC data set were copied to a new directory to be cropped using the `shutil.copy()` method. Also, similarly to the first step taken in when creating the YOLOv4 data set the `h5` files were all read into a data frame assigned to the variable `master_df` using the `pathlib` and `pandas` libraries in Python.

2. **Determining appropriate box size for cropping** - The `PIL.Image.crop()` method was used for cropping the images from their full resolution to a reduced resolution for the lightweight key point regression model. When attempting to crop the images using the ground truth bounding box annotations, it was discovered that the (768×768) bounding boxes would not contain the coordinate of the index finger. This is an issue as this is an essential point for the regression model. Therefore, a final bounding box size was 1120×1120 as this resulted in the tip of the index finger and thumb fully enclosed by the bounding box in all frames.

3. **Cropping the images** - The images are being cropped based on ground truth bounding box annotations. This produced a data set for the key point regression model without requiring the YOLOv4 model to be finalised for this project. The `PIL.Image.crop()` method was used which takes a 4-tuple of the left, right, top, and bottom boundary lines of the desired crop region relative to the image space of the original input image. This can be explained when visualising FIGURE in which the red box represents the desired cropped region. The top boundary line for the cropped region lies at approximately y=300, the right boundary line lies at x = 1250. These were calculated using the following computations for all images:

   (a) Left = Centre x - (Box width / 2)

   (b) Right = Centre x + Box width / 2)

   (c) Top = Centre y - (Box height / 2)

   (d) Bottom = Centre y + (Box height / 2)

   This was converted into a 4-tuple which was then used to crop the images with the aforementioned `PIL.Image.crop()` method.

Converting the key point coordinates

Once all the images had been cropped, the coordinates of the key points also needed to be updated. The original labelled coordinates are based on the origin of the 1080p image, represented by the black dot in Figure 4.14. As the relative position of each hand varied between participants and also different frames for the same participant, the coordinates needed to be updated relative to the new origin of the cropped image. See Figure 4.14 in which the key points of the hand have been defined and the dimensions of the bounding box used to crop the image.

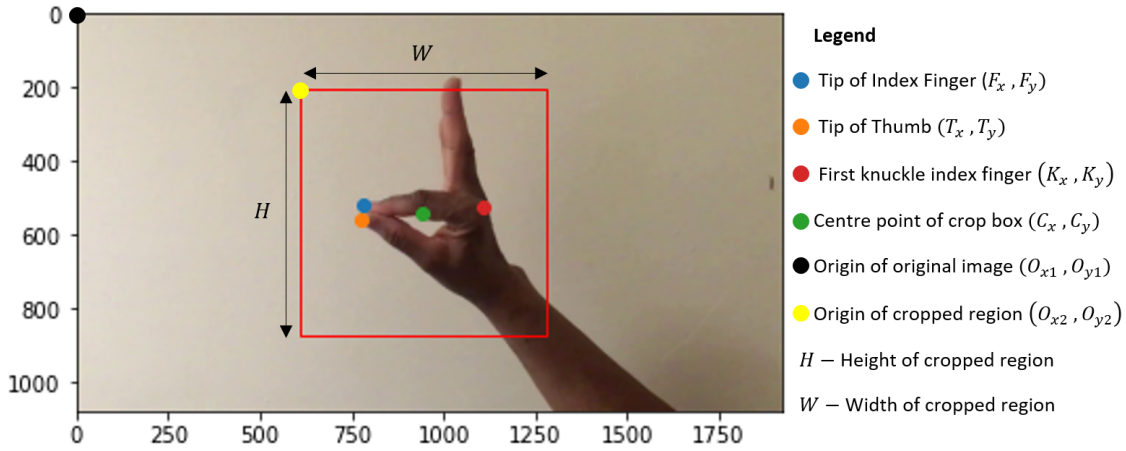

Figure 4.14: Converting key points coordinates from original image to cropped image

The following steps were undertaken to convert the coordinates of the tip of the index finger and thumb from their position on the original 1920×1080 image to their relative position on the cropped image:

1. **Finding the origin of the cropped region** - Firstly, the coordinates of the cropped region need to be calculated. The origin of the cropped region is represented by the yellow point in Figure 4.14, which have the coordinates of (O_{x2}, O_{y2}) on the axis of the original input image. These values can be calculated using the coordinates of the centre of the cropped region as defined in FIGURE, and the dimensions of the cropped region as follows:

   (a) $O_{x2} = C_x - (W/2)$

   (b) $O_{y2} = C_y - (H/2)$

2. **Updated coordinates for the tip of the index finger** - Now that the origin of the new cropped region has been calculated on the coordinate axis of the input image, the updated coordinates can be calculated for the tip of the index finger relative to the origin of the new cropped region. This is given by:

   (a) $F_{x2} = F_{x1} - O_{x2}$

   (b) $F_{y2} = F_{y1} - O_{y2}$

   Where F_{x1} is the original x-coordinate of the tip of the index finger on the image axis of the original 1920×1080 image and F_{x2} is the updated x-coordinate for the tip of the index finger on the image axis of the cropped region, based on the new origin of the cropped region.

3. **Updated coordinates for the tip of the thumb** - The updated coordinates of the tip of the thumb are calculated similarly to the tip of the index finger:

(a) $T_{x2} = T_{x1} - O_{x2}$

(b) $T_{y2} = T_{y1} - O_{y2}$

Where T_{y1} is the original y-coordinate of the tip of the thumb on the image axis of the original 1920×1080 image and T_{y2} is the updated x-coordinate for the tip of the thumb on the image axis of the cropped region, based on the new origin of the cropped region.

See Figure 4.15, which shows the plotted coordinates for the tip of the index finger and thumb and on the image axis of the cropped image. This is the cropped image that was shown in Figure 4.14, and when comparing the two images you can observe that the axis for the original image and the cropped image has changed. It is clear that the tip of the middle finger has been cropped out of image space by using this method. The model may struggle to learn visual features associated with this class if in some training instances certain digits are contained within the ground truth bounding box label and in other instances they are not contained within the ground truth bounding box label. However, it was decided that as long as long as the bounding box contained the two key points required for the regression model in this project (the tip of the thumb and the tip of the index finger), that this was not an issue for this study.

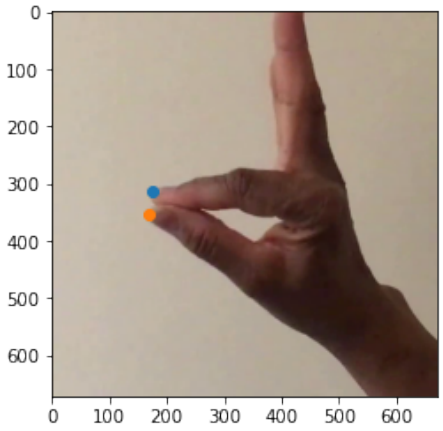

Figure 4.15: New coordinates relative to origin of cropped image

4.5 Bounding Box models

This section of the report outlines the network architectures for models proposed for the 'Object Detection Model' in the ML pipeline to produce bounding box coordinates as a crop region for the image of a hand. The two models will be trained to detect a single class (hand), and the two models will be compared with a preferred model identified. The two proposed models are YOLOv4 and YOLOv4-tiny, which share similar structures and methods; however, the tiny version contains significantly fewer parameters. Both networks methods were released by Bochkovskiy et al. (2020) and YOLOv4 achieves state-of-the-art results on the MS COCO

data set 43.5% AP (65.7% AP50) at 65 FPS on a Tesla V100. Also on the MS COCO data set, YOLOv4-tiny achieves 22.0% AP (42.0% AP50) at a speed of 443 FPS on an RTX 2080 Ti (Bochkovskiy et al. 2020). Clearly, YOLOv4 achieves an accuracy that is far higher than YOLOv4-tiny, however, it has an inference speed that is nearly 7 times slower, which requires a high-end GPU. Furthermore, YOLOv4-tiny has been demonstrated to achieve 25 FPS inference speeds using CPUs, making it feasible for deployment on mobile and embedded devices (Jiang et al. 2020). Despite YOLOv4-tiny showing lower AP metrics on the MS COCO data set, is it accurate enough to produce bounding box coordinates for a single-class model?

4.5.1 Environment

YOLOv4 is underpinned by Darknet, which was published by the developer of YOLOv1-3, Redmon (2013–2016). Darknet is an open-source neural network framework written in C and CUDA. It is fast and easy to install using the Linux command line and supports CPU and GPU computation. It has only two dependencies, OpenCV for image processing and CUDA for GPU support.

A virtual environment called 'Google Colab Pro' was used for the object detection models used in this project. The data set and YOLOv4 configuration files were uploaded to a Google Drive, which was mounted to a Colab virtual machine (VM). Subsequently, the Darknet git repository was cloned and built to the same VM. Weights to the relevant model were downloaded that were pre-trained on the MS-COCO data set. A key benefit of using these pre-trained weights is that the MS-COCO is already trained on the class 'person' and therefore, the model is likely to generalize quickly as the visual features associated with the class 'hand' will be a subset of the class 'person' of which the weights can already classify. Google Colab Pro gives shared allocation to a Tesla P100-PCIE-16GB GPU and 26GB of RAM, allowing for much faster training times than if undertaken on a laptop.

4.5.2 YOLOv4: Network Architecture

YOLOv4 consists of three core stages; a backbone, neck, and head. The backbone is CSPDarknet-53 which is pre-trained on ImageNet and can extract visual features for object detection. The Darknet variant used in this model has 137 pre-trained convolutional layers with 'Mish' activation functions. The necks used in this system are Spatial Pyraming Pooling (SPP) and Path Aggregation Network (PAN), which extract feature maps at various stages of the backbone and allow for the use of fine and coarse visual feature maps be used for the final predictions in object detection. The SPP block is added since it significantly increases the receptive field size of the network, which separates the most important spatial features whilst having minimal impact on inference speeds. Lastly, the head in this system is the anchor box method used in YOLOv3, which is used to predict the class, confidence scores, and final bounding boxes for objects.

4.5.3 YOLOv4-tiny: Network Architecture

YOLOv4-tiny is the compressed version of the YOLOv4 architecture designed by Bochkovskiy et al. (2020). The network has been simplified and the number of parameters have been reduced to make it viable for mobile and embedded devices (Jiang et al. 2020). The Darknet variant used in this model only has 29 pre-trained convolutional layers compared with 137 in the full YOLOv4 model. Furthermore, the 'Mish' activation function was replaced with 'Leaky ReLU' activation functions to reduce computational overhead further. YOLOv4 tiny still uses PANet for the neck to extract feature maps at different scales and anchor-based YOLOv3 for the head of the network for making final predictions. However, it does not use SPP, and therefore YOLOv4-tiny has a smaller receptive field than YOLOv4. One further key difference is that YOLOv4-tiny uses only two heads, whereas YOLO-v4 uses a total of three heads. Therefore, YOLOv4-tiny will make fewer anchor box predictions per frame of video, which speeds up the network and contributes to a lower AP on MS COCO. A high-level overview of the network is given in Table 4.3, and for further details of the network architectures, please refer to papers the following cited papers, as explaining all design decisions in detail is not possible due to the constraints of this project (Bochkovskiy et al. 2020, Jiang et al. 2020).

Component	YOLOv4	YOLOv4-tiny
Backbone	CSP-Darknet 137 convolutional layers Mish activation functions	CSP-Darknet 29 convolutional layers Leaky ReLU activation functions
Neck(s)	SPP and PANet	PANet only
Head	3× YOLOv3 (Anchor Based)	2× YOLOv3 (Anchor Based)

Table 4.3: Comparison of object detection models used in this study

4.5.4 Training

Configuration (cfg) files containing the network architectures and the training files were downloaded from the GitHub repository published by Bochkovskiy et al. (2020). These cfg files for each network were downloaded for training object detection for a custom data set and adapted using specific instructions listed on the GitHub repository. These changes are summarised in Appendix A.1.

4.5.5 Evaluation

Bochkovskiy et al. (2020) instructs users that you usually 2000 iterations is sufficient per class. However, you should ideally train for at least as many iterations as you have training images and not less than 6000 iterations in total. Furthermore, the model can stop being trained when the average training loss no longer decreases for many iterations. Additionally, a target training loss value can be from 0.05 (for a small model with a simple an 'easy' data set) to 3.0 (for a

big model with a 'difficult' data set) (Bochkovskiy et al. 2020). As the data set in this class only contains one class, and as YOLOv4-tiny is a smaller model, if it has sufficient learning capacity for this task it should achieve an average training loss of 0.05 as a target. For the larger YOLOv4, the target should be less than 3.0 as this study's data set contains only one class and could be considered more simple than the MS COCO data set, which contains 80 classes. Furthermore, once training has finished, the mAP should be checked for weights that have been saved at different checkpoints throughout training. This means that the mAP will be compared every 1000 iterations, and there will be six checkpoints per model. The checkpoint with the highest mAP can have a higher training loss than other checkpoints as the model can overfit the training data. By selecting the weights with the highest mAP, the 'early stopping point' can be identified: the point with the highest mAP but not necessarily the lowest average training loss. Therefore, the final model will be less likely to overfit the training data.

4.6 Key point regression models

This section of the report outlines the network architectures for models proposed for the 'Key Point Regression Model' in the ML pipeline to produce key point coordinates of the tip of the index finger and thumb for the cropped images. The two models will be trained to predict a 4-tuple containing the coordinates $\langle thumb_x, thumb_y, finger_x, finger_y \rangle$ for each frame of video inputted to the model. The MSE of the two models will be compared, and a preferred model will be identified, with example predictions shown. The two proposed models are EfficientNet-B0 and EfficientNet-B4. These networks belong to the same family of network architectures that have been reviewed in SECTION. EfficientNet-B4 is simply an upscaled version of EfficientNet-B0. It is deeper, wider, and takes a higher resolution input image. EfficientNet-B4 achieves a higher top-5 accuracy on ImageNet than EfficientNet-B0, but both are considered mobile network architectures. EfficientNet-B0 is the most lightweight version of the EfficientNet family, whereas EfficientNet-B4 seems to offer the best tradeoff between the number of parameters and accuracy. Both networks used in this project will be pre-trained on ImageNet, and the only trainable parameters will be in three fully connected layers that use the learned visual features to predict the coordinates of the key points. In other words, all weights associated with the feature extraction part of the network will be frozen. Backbone network architectures that have been pre-trained on ImageNet have been demonstrated to be robust and to generalize well. By comparing EfficientNet-B0 to EfficientNet-B4, two similar state-of-the-art backbones can be compared, and the impact of compound scaling can be identified in the context of regression-based key point and pose estimation models.

4.6.1 Environment

The source code for EfficientNet-B0 was written in TensorFlow 2, an open-source machine learning platform written in a combination of C++ and CUDA, making it extremely fast and of-

fers GPU support. Furthermore, it is very easy to access and operate TensorFlow 2 using Python libraries. Since EfficientNet was published, there has been an implementation of EfficientNet-B0 released on a high-level TensorFlow 2 API for deep learning called Keras. Keras is written in Python using TensorFlow 2 and allows for models such as VGG-16, ResNet-152, and EfficientNet-B0 in various applications. Therefore, TensorFlow 2 will be used to build both the custom model and EfficientNet-B0. The models were built using Jupyter notebooks on a local machine and saved in a local directory with the data set.

A virtual environment called 'Vast.ai' was used for the key point regression models in this project. The Vast.ai instance gave dedicated access to a NVIDIA RTX 3090 24.3GB GPU and 32 GB of RAM, allowing for faster training times than on Google Colab. Furthermore, image processing is easier using vast.ai than Google Colab pro as you can open Jupyter notebooks as if they were on your local machine without having to interact with your Google Drive and Virtual Machine, which was discovered after building the object detection models in this report. Furthermore, vast.ai offers a 'pay as you go' system at approximately 0.50 cents per hour with dedicated resource allocation, which made it both cheaper and faster than using Colab Pro. The data set and Jupyter notebook files of the models were uploaded to a GitHub repository, which were then cloned to a Vast.ai instance. This was the fastest way of accessing the files on multiples instances as the download speed of the instance was approximately 85$\times$ faster than the upload speed of the local internet connection.

4.6.2 EfficientNet-B0: Network Architecture

EfficientNet-B0 is the baseline architecture designed by a multi-objective architecture search to optimize both FLOPS and accuracy (Tan & Le 2019). The model used in this study was pre-trained on ImageNet for the 1000 class classification challenge. In this study, the final layers, or 'top', was removed and replaced with three fully connected layers, with the final layer having four neurons to produce a 4-tuple of the key point coordinates previously defined. The network is designed to take input images that have a resolution of 224$\times$224 with 3 input channels. The 3 input channels are for RGB channels, which is common for coloured images to be represented. EfficientNet-B0 has a total of 8 million parameters of which 4 million are trainable. This is because the weights have been frozen for stage 1-9 as these are the pre-trained feature extraction layers that have been pretrained on ImageNet. The 4 million trainable parameters are in layers 10-13 which are being trained to fine-tune the network for key point estimation. The network has a total of 241 layers, the majority of which come from mobile inverted blocks which use squeeze and excitation blocks and a Swish activation function which are explained in Section 3.2.5. These blocks are repeated multiple times as indicated by Table 4.4, in which a high-level overview of the network architecture has been given. MBConv1 represents the first type of mobile inverted block, further details can be found in articles by Agarwal (2020) and Borad (2021).

Stage	Operator	Input Resolution (height, width)	Output Channels	No. Blocks
1	Conv2D block, k×3	224, 224	32	1
2	MBConv1, k3×3	112, 112	16	1
3	MBConv6, k3×3	112, 112	24	2
4	MBConv6, k5×5	56, 56	40	2
5	MBConv6, k3×3	28, 28	80	3
6	MBConv6, k5×5	14, 14	112	3
7	MBConv6, k5×5	14, 14	192	4
8	MBConv6, k3×3	7, 7	320	1
9	Conv2D block, k1×1	7, 7	1280	1
10	Flatten	-	62,720	1
11	Fully Connected (Dense)	-	64	1
12	Fully Connected (Dense)	-	64	1
13	Fully Connected (Dense)	-	4	1

Table 4.4: EfficientNet-B0 Network Architecture with custom top for regression.

4.6.3 EfficientNet-B4: Network Architecture

EfficientNet-B4 follows a similar architecture to B0. It is the same baseline architecture that has been upscaled for resolution, width and depth. The input resolution for EfficientNet-B4 is 380×380, whereas B0 is 224×224. Furthermore, it is demonstrated in the output channels columns in Tables 4.4 and 4.5 that at each stage through the feature extraction stages in the network (prior to the 'Flatten' operator) that there are significantly more output channels in B4 compared to B0, meaning that B4 is wider than B0. Furthermore, there are a total of 477 layers in B4, which comprise the multiple blocks summarised in Table 4.4. There are a total of 34.2 million parameters in this network, of which 16.5 million are trainable. There are approximately 4× more trainable parameters in B4 compared to B0 due to the network dealing with a greater resolution and more output channels being outputted from the feature extractor. This is shown in Stage 10 in Table 4.5, in which the 'Flatten' stage has over 4× the number of channels.

Stage	Operator	Input Resolution (height, width)	Output Channels	No. Blocks
1	Conv2D block, k3×3	380, 380	48	1
2	MBConv1, k3×3	190, 190	24	2
3	MBConv6, k3×3	95, 95	32	4
4	MBConv6, k5×5	48, 48	56	4
5	MBConv6, k3×3	24, 24	112	6
6	MBConv6, k5×5	24, 24	160	6
7	MBConv6, k5×5	12, 12	272	8
8	MBConv6, k3×3	12, 12	448	2
9	Conv2D block, k1×1	12, 12	1792	1
10	Flatten	-	258,048	1
11	Fully Connected (Dense)	-	64	1
12	Fully Connected (Dense)	-	64	1
13	Fully Connected (Dense)	-	4	1

Table 4.5: EfficientNet-B4 Network Architecture with custom top for regression.

4.6.4 Training

The data set of 10,239 cropped images were split into 90% training images (9213 instances) and 10% validation images (1026 instances). To prevent the models from overfitting the training data, the *keras.callbacks.EarlyStopping()* method was used to stop the model from training if the model did not decrease within the last 10 epochs. This is a regularization technique called early stopping which is used to prevent the validation loss from increasing as in explained in Section 2.3.2. The number of neurons where hand tuned in layers 11 and 12 for both networks, however the validation performance was given for the final configurations demonstrated in Tables 4.4 and 4.5. The learning rate was also varied but the final learning rate used was 2×10^{-5}.

4.6.5 Evaluation

The mean square error was calculated at each epoch of training for both the training and validation data to compare the models. This can be used to calculate the root mean squared error, which represents the mean pixel error for each model. Furthermore, the inference speed was measured for each model by measuring the time taken to make predictions on the entire validation batch and dividing by the number of images in this data set. There is also a time-dependent objective of carrying out k-fold cross-validation on these models to increase the chance that the model has been trained on every data point. Also, to ensure that any conclusions made are not due to a sampling generalization error caused by the training and validation split in the data.

Chapter 5

Results

This chapter will present the results from the object detection model comparison, and the key point regression model comparison. A preferred model will be identified for each model, along with optimal weights and a justification as to why it is the preferred model for this application.

5.1 Object Detection models for bounding box coordinates

Figure 5.1 shows the training loss and validation mAP for the two object detection models built for this research project, the left-hand figure is for YOLOv4-tiny, and the right-hand figure is for the full YOLOv4. The overall trend for the training loss for both models follows very similar behaviour. The training loss decreases rapidly for the first 200-300 iterations; after that, the training loss decreases gradually. The training loss for YOLOv4-tiny seems to follow a much smoother curve, and therefore the model seems to be indicating that the model is learning the features of the training set with a sufficiently small learning rate throughout training. Furthermore, the final training loss is 0.0343, which is lower than the target value of 0.05 set by the developer of YOLOv4-tiny, which suggests the model has successfully learned the features within the training set (Bochkovskiy et al. 2020). For YOLOv4, on the other hand, the training loss curve decreases less smoothly, and there is some fluctuation indicated by the zig-zag appearance of the blue curve. The final training loss for YOLOv4 is 0.4277, which is significantly lower than the target of 3.0, which was discussed in Section 4.5.5 (Bochkovskiy et al. 2020).

More importantly, the red line in Figure 5.1 represents the validation mAP which shows how well the model can make predictions on unseen data. Both YOLOv4 and YOLOv4-tiny achieve an mAP of 100% indicated in Figure 5.1. The red curve starts at 1000 iterations, as this is when the first mAP calculation is performed throughout the training schedule. YOLOv4-tiny appears to reach a higher mAP at this point at 99%, whereas YOLOv4 only achieves 66%. This is unsurprising as YOLOv4 has significantly more training parameters and therefore requires more iterations of training to reach the highest mAP for this data set. Fortunately, the best weights are saved, which maximise the mAP allowing the early stopping point to be identified.

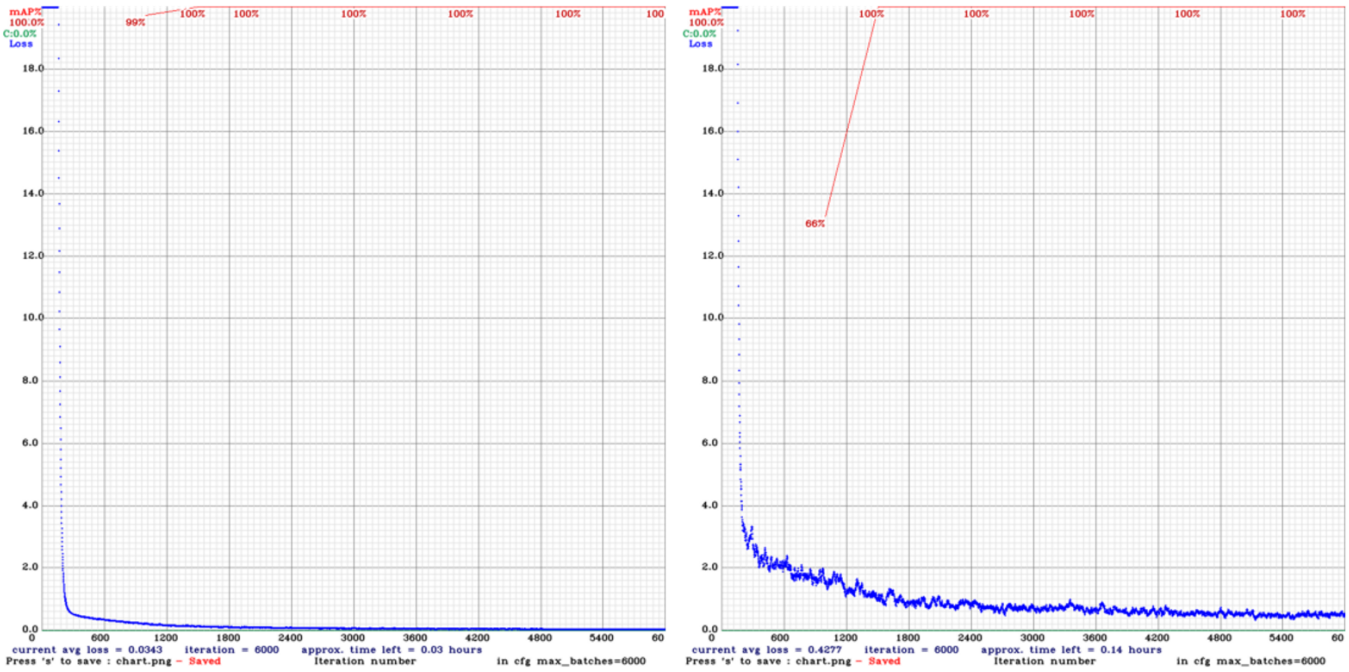

Figure 5.1: Object Detection Models: Training loss and validation mAP. Left: YOLOv4-tiny. Right: YOLOv4.

Interestingly, both models achieve the exact same maximum mAP of 99.9979%, which means they identify ∼100 % of the hands in images. However, the maximum mAP and, therefore, the early stopping point occurs at a different number of iterations. YOLOv4-tiny reaches maximum mAP at 4093 iterations, whereas YOLOv4 reaches the maximum mAP at 1515 iterations. This is interesting as mAP increased initially for YOLOv4 slower initially than YOLOv4-tiny, which was understandable because it had more parameters and required more iterations to learn the features of the training data. Therefore, it could be expected that YOLOv4-tiny would reach the maximum mAP faster than YOLOv4. However, this could be caused by a given training image not being seen until later on in training by YOLOv4-tiny due to random shuffling prior to training. Although the early stopping point has been identified for each model through the iteration with the highest mAP, the mAP remains to be really high and rounded to 100% for both models after this point. Therefore, it could be argued that early stopping is not essential for this data. It would be really useful if the developers of YOLOv4 plotted validation loss as in their in-built functions, as this could allow further discussion about overfitting. However, it is unlikely the model is overfitting as the mAP remains stable. YOLOv4 achieves a slightly higher average IOU of 85.06%, whereas YOLOv4-tiny achieves an average IOU of 84.07%, which is a really small difference and is unlikely to impact detection performance largely. The difference in bounding box size for a given image is unlikely to be largely different, and both models should be able to predict bounding box coordinates for hands exceptionally well. The IOU is a metric for measuring the overlap between ground truth and predicted bounding boxes,

whereas the AP considers the number of TP and FP as discussed in Section 2.2.

Both models achieve very similar performance in maximum mAP and average IOU at the early stopping point as discussed, which means they should both make excellent predictions. Therefore, to decide the preferred model for the final pipeline, considerations need to be made of the models' inference speeds, memory requirements, and example predictions. YOLOv4 makes predictions with a mean inference speed of 20.25 milliseconds per image which means it can make predictions at 49.4 FPS on a Tesla P-100 16GB GPU. YOLOv4-tiny is much faster, with a mean inference speed of 3.03 milliseconds per image which means it can make predictions at 330 FPS on the same GPU. Both models achieve real-time performance on a GPU, but YOLOv4-tiny is significantly faster whilst maintaining the same mAP and similar IOU. Based on this, YOLOv4-tiny is the preferred model. Predictions are shown in Figure 5.2, in which the bounding boxes produced by both models seem very similar. Figure 5.2 left shows the prediction made by YOLOv4-tiny, which appears to be a tighter fit than YOLOv4 on the right. Furthermore, YOLOv4-tiny predicted this example with a confidence of 0.99, whereas YOLOv4 has a slightly lower confidence of 0.97.

In summary, both models perform a maximum mAP of 100% on validation data with a similar average IOU of approximately 85%. However, YOLOv4-tiny has inference speeds approximately 7 times faster than YOLOv4. Therefore, the preferred model is YOLOv4-tiny.

Figure 5.2: Predictions made by object detection models. Left: YOLOv4-tiny. Right: YOLOv4.

5.2 Key Point Regression Models

Figure 5.3 shows the training loss and validation loss for the two key point regression models built for this research project, the left-hand plot is from training EfficientNet-B0 and the right-hand plot is from training EfficientNet-B4. EfficientNet-B0 takes a total of 73 epochs to converge, which took approximately 45 minutes to train. Both the training and validation loss curves decrease rapidly initially until about 25 epochs and then more gradually after this point. The model exhibits some variance indicated by the orange line following the same trend as the blue line in the left plot. The final training loss is 127.24 (MSE) which gives a pixel

error of 11.28 by taking the square root of the MSE. Similarly, the final validation loss was 179.76, which equates to a pixel error of 13.41. It was speculated that the human labelling variability was approximately 8 or 9 pixels and was justified in Section 4.4.2. Therefore, given that the model is able to make predictions on unseen validation data with a mean pixel error of 13.41, the model is performing well. Furthermore, the resolution of the image was resized from 1120×1120 down to 224×224, which means that the original image has been downsized by a scale factor of 5. The labels on the other hand, were left on the original scale of 1120×1120 as this allows them to easily be converted back to the axis of the full 1080p image by the Co-ordinate Conversion Function. However, this means that every 1 pixel inputted into the image represents a 5×5 pixel grid on the original image axis. Therefore, a labelled coordinate on the downsized image can represent 1 of 25 coordinates on the full-sized image. This is likely to have contributed to the training and validation error present in Figure 5.3, as the model cannot learn associate visual features in a lower resolution with coordinates on a higher resolution that it has not been trained on.

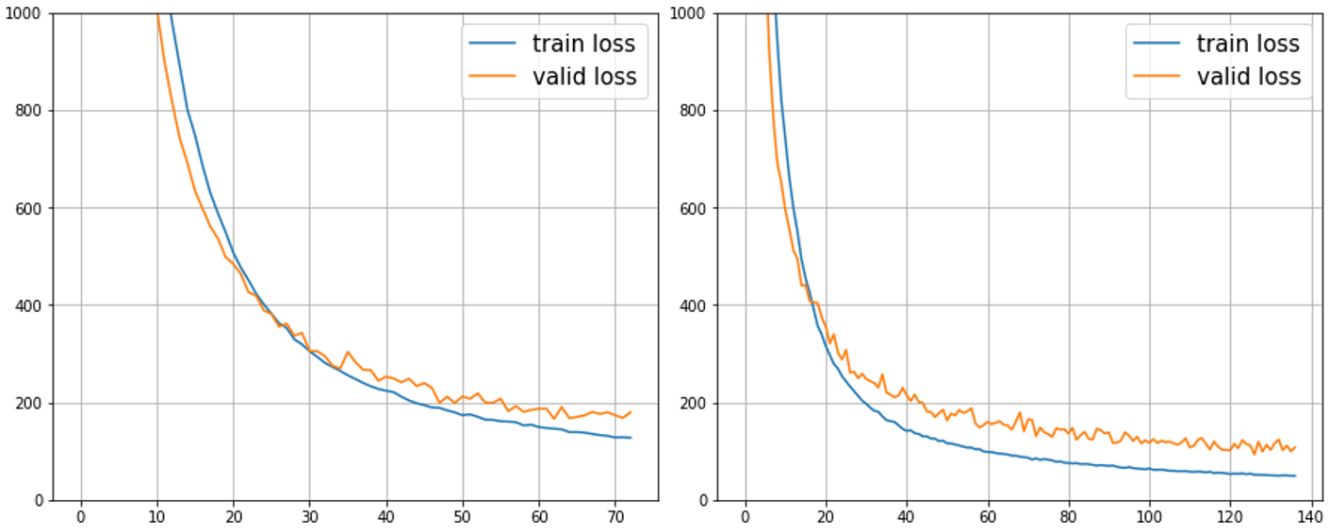

Figure 5.3: Key Point Regression Models: Training loss and validation loss. Left: EfficientNet-B0. Right: EfficientNet-B4.

Similarly to B0, B4 begins to converge at approximately 25 epochs. However, for B4, both loss curves continue to decrease until the early stopping point at 137 epochs. The final training loss is 49.30, which gives a mean pixel error of 7.02 px and the final validation loss is 99.00, which gives a mean pixel error of 9.95 px. As the training loss is 7.02 px, it is unlikely that the human variability in the data is approximately 8 or 9 pixels, as previously speculated. The true human variability is likely to be a value $\leq$ 7.02 px as some additional loss is likely to be caused by the model and by rescaling the input image dimensions. The resolution of the image was resized from 1120×1120 to 380×380 as this is the input size defined by the developers for EfficientNet-B4, which means the original image has been downsized by a factor of $\sim$2.95.

Therefore each pixel inputted to the model represents an approximate 3×3 grid on the axis of the original image and a labelled coordinate on the downsized image can represent 1 of 9 on the full-size image.

The mean inference speed of B0 was 2.27 milliseconds per image, which is over 439 FPS on an NVIDIA RTX 3090 24.3GB GPU. For B4, on the other hand, the mean inference speed was 51.77 milliseconds per image, which is over 19 FPS on the same GPU. Therefore, B0 can be considered real-time, but B4 cannot be considered real-time on this GPU. The requirement for a model to be considered 'real-time' is having an FPS greater than 30 FPS. Furthermore, the FPS is likely to be much slower for both models on a CPU, which may be an issue for mobile devices deployment.

5.2.1 Discussion

It has been previously demonstrated that the mean pixel error using DeepLabCut on the same data set was 8.39 pixels using 1920×1080 resolution images (Zhao et al. 2020). EfficientNet-B0 and B4 achieve a mean pixel error of 13.41 px and 9.95 px, respectively is quite remarkable considering that the models are making predictions using resized images. In summary, 1920×1080 resolution images of hands have been cropped to 1120×1120 images to remove excess background pixels. The cropped image was downsized by a scale factor of 5 for EffcientNet-B0 and ∼2.95 for Efficient-B4. Therefore, the quality of the data being inputted to the EfficientNet models in this report contains less semantic visual details as multiple pixels have been condensed into single pixels. Further, given that the pixel error on validation data of the EfficientNet models is not significantly worse than using DeepLabCut due to making predictions on lower quality images suggests that EfficientNet pre-trained on ImageNet is an excellent backbone for human hand pose estimation tasks.

EfficientNet-B0 achieves real-time inference speeds (439 FPS) on a GPU, whereas EfficientNet-B4 does not (19 FPS). EfficientNet-B4 being nearly 23× slower is due to a large number of weights in the fully-connected layers which are much slower to compute than fully convolutional operations. The more accurate B4 model is slower, and the less accurate B0 model is faster. Labelling accuracy is likely to be the more important factor than inference speed at this stage, as being able to produce an accurate time-series plot for the euclidean distance between tip of the index finger and thumb is essential for classifying patients on the MDS-UPDRS. If the key points are not tracked accurately, then measurable tapping metrics such as rhythm, amplitude and speed are likely to be negatively impacted by labelling error caused by the automatic labelling system. Therefore, the preferred key regression model for the final ML pipeline is EfficientNet-B4 as it achieves a lower mean pixel error on the validation set of 9.95 px. Predictions on training and unseen data have been given in Appendix A.2 to allow for multiple predictions to be visualised. On the vast majority of predictions, the predictions look very good. Some predictions are slightly poorer when a hand is not in the 'closed' position.

Chapter 6

Conclusion

6.1 Limitations of Study and Suggestions for Future Work

YOLOv4-tiny achieves a mean IOU of 84.07%, which is extremely good and certainly good enough for localising hands in this application with a sufficient level of accuracy. However, this could likely be improved by using higher quality data. The bounding box coordinate labels were produced by adapting existing labelled key points using methods discussed in Section 4.4.3. As a result, often the bounding box labels did not fully enclose all digits of the hand for certain training samples. For other training samples, there were excess background pixels, i.e. the bounding box fit the hand 'loosely'. Therefore, the performance of this model could be improved by manually labelling each frame of video to ensure that each bounding box encloses the entire hand and to ensure that each box does not contain excessive background pixels. Poor bounding box labels likely served as a limiting factor for this project as they were not consistent between frames.

Furthermore, YOLOv4 instructs users to train the model for at least as many iterations as training images (Bochkovskiy et al. 2020). The training data set contained 8268 images but was only trained for 6000 iterations, which does not follow the instructions of Bochkovskiy et al. (2020). However, given the performance of a model, it is unlikely that further iterations would have improved the model's performance. Lastly, the bounding box training labels were 768×768 in dimension, which was later discovered was too small to fully enclose all key points. Therefore, the model should be retrained for bounding boxes with a larger dimension. Alternatively, a 200-pixel buffer could be added to the height and the width of the bounding box dimensions before passing the images into the cropping function.

The key point regression model using EfficientNet-B4 achieved a good mean pixel error of 9.95 px however, this could be improved by potentially using a model that takes an input image of higher resolution. EfficientNet-B4 downsized 1120×1120 images down to 380×380; therefore, each pixel in the input image represented an approximate 3×3 grid on the original

image. This serves as a physical limit to how accurately the model can predict pixel coordinates on the original image as much of the visual semantic information is lost by downsizing the image. Backbones that take a larger input, such as EfficientNet-B7 (which takes an input of 600×600), should be considered to investigate the proportion of the pixel error that is due to an error caused by downsizing the image. However, this was demonstrated to some extent by comparing EfficientNet-B0 and EfficientNet-B4. Furthermore, if EfficientNet-B7 did not result in a significantly lower pixel error, then further comments could be made about the contribution of error due to the variability of human labelling.

Furthermore, the key point estimation model that used EfficientNet-B4 achieved an inference speed of 19 FPS which did not achieve real-time performance on an Nvidia RTX 3090. Therefore would not be suitable for real-time deployment on mobile or embedded devices. The inference speed of the network was limited by a bottleneck in the 'top' of network that is responsible for regression. As the convolutional layers were flatted to a feature vector of 258,048 neurons which was passed on to the fully connected 'top' of the network, there were over 16 million operations performed to produce the 4-tuple of coordinates. Alternatively, the same pre-trained backbone could be used, but the problem could be treated as a classification problem rather than a regression problem. This would involve removing the fully connected layers that cause the bottleneck in inference speed and replacing it with deconvolutional layers compared with ground truth heat maps. This would reduce the number of operations required in a forward pass and likely reduce the inference speed.

Additionally, due to the constraints of this project, k-fold cross-validation was not carried out. Therefore, a time-dependent objective was not met. K-fold cross-validation should be carried out in future studies to ensure the results in this study are robust and repeatable.

Lastly, it is possible that the 9.95 mean pixel error associated with the final model is likely to be an overestimate of how accurate the model can predict key points. This is due to the 'closed' position, where the index finger and thumb are touching, being oversampled. Further, as the two key points being predicted are touching, they occupy a very small area of pixels in the image space. Due to the proximity of the two key points, the mean squared error is likely to be lower for the 'closed' position than when the index finger are not touching or at maximum amplitude. It is likely that the mean pixel error is higher for any open position compared to the closed position which was explored in Section 4.4.2.

6.2 Conclusion

This project aimed to automate the labelling stage of the DeepLabCut workflow. Which for this application, takes an image of a hand as input and output labels for the coordinates of the

tip of the index finger and thumb. The automatic labelling system was designed by reviewing state-of-the-art deep-learning-based computer vision algorithms which were selected for the final pipeline, to try and optimise accuracy and efficiency. YOLOv4 and YOLOv4-tiny were the selected object detection algorithms used to detect and localise a hand in a 1080p frame. Both YOLOv4 and YOLOv4-tiny achieved an mAP of 99.9979%. However, YOLOv4-tiny was selected as the preferred model as the inference speed was nearly $7\times$ faster. YOLOv4-tiny had a mean IOU of 84.07%, which was less than 1% lower than YOLOv4. However, this was not an issue as the model was able to produce bounding boxes that fully enclosed the hands containing all key points of interest for tracking.

Once the hand had been localized in the original image space using YOLOv4-tiny, a cropped image was passed into a key point estimation model. A regression-based approach was adopted that predicted a 4-tuple containing the (x,y) coordinates for the index finger and thumb tip. EfficientNet-B0 and EfficientNet-B4 were compared as feature extractors, followed by two fully connected layers containing 64 neurons each and a final fully connected layer with 4 neurons to produce the 4-tuple of predicted coordinates. On validation data, the mean pixel error was 13.41 px and 9.95 px for EfficientNet-B0 and EfficientNet-B4, respectively. A large proportion of this error is likely due to variability in human labelling in the training data and downsizing the input image. Downsizing the image reduces the visual semantic information present in the data, which serves a physical constraint that even a perfect model could not overcome. As accuracy is of primary importance for the key point estimation model, EfficientNet-B4 was the preferred model and had an inference speed of 19 FPS which could not be considered 'real-time'; however, it was only 11 FPS too slow. EfficientNet-B0 achieved 439 FPS which is excellent; however, the error of the model was greater, and therefore the EfficientNet-B4 was selected for the final system.

Code has not been explicitly given for the final 'Cropping Function' and 'Coordinate Conversion Function' for the final ML pipeline. However, a similar approach can be adopted that was used to create the data sets as explained in Section 4.4. The primary focus of this paper was to build the models for an automatic labelling system to label to coordinates of the tip of the index finger and thumb. The final automatic labelling system pipeline for this report uses YOLOv4-tiny for object detection and an EfficientNet-B4 backbone pre-trained on ImageNet with a customized top for key point regression to predict coordinates for the tip of the index finger and thumb. The final mean pixel error from the pipeline is 9.95 px which is excellent considering it has been demonstrated that DeepLabCut achieves 8.39 px on the same data set (Zhao et al. 2020).

Appendix A

Appendix

A.1 YOLOv4 and YOLOv4-tiny configuration instructions

The following changes need to be made to the cfg files named 'yolov4-custom.cfg' and 'yolov4-tiny.cfg' that were available to download from the GitHub repository: (https://github.com/AlexeyAB/darknet/tree/master/cfg) (Bochkovskiy et al. 2020).

Three colour channels were used inputted to both models, and both had a learning rate decay of 0.0005 by default. There were some default differences in the cfg files that were not altered as there were no instructions to do so, such as learning rate was 0.001 for YOLOv4 and 0.00261 for YOLOv4-tiny. Furthermore, momentum was 0.949 for YOLOv4 and 0.9 for YOLOv4-tiny. The following instructions applied to both models however:

- batch=64,
- subdivisions=16,
- max_batches=([number of]classes× 2000) but not less than 6000 or the number of training images. f.e. max_batches=6000,
- change line steps to 80% and 90% of max_batches, f.e. steps=4800, 5400
- set network size width =416, height =416
- change number of classes, f.e. classes=1
- change filters=255 to filters=(classes+5)× in the [convolutional] layer before each [yolo] head. f.e. for 1 class, filters = 18.

A.2 EfficientNet-B4 Key point regression model predictions

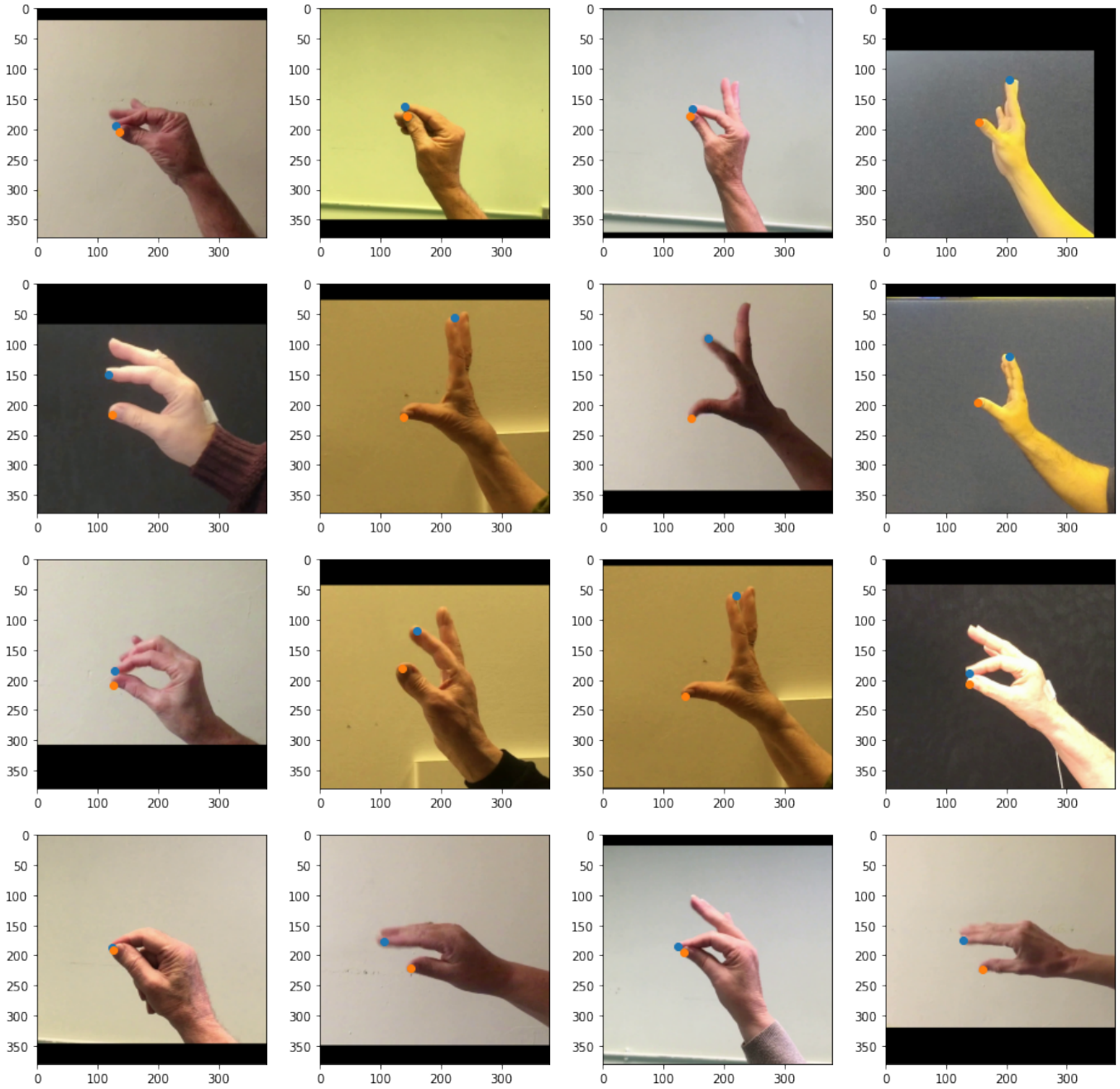

Figure A.1: EfficientNet-B4 Key point regression model: Predictions on Training Data

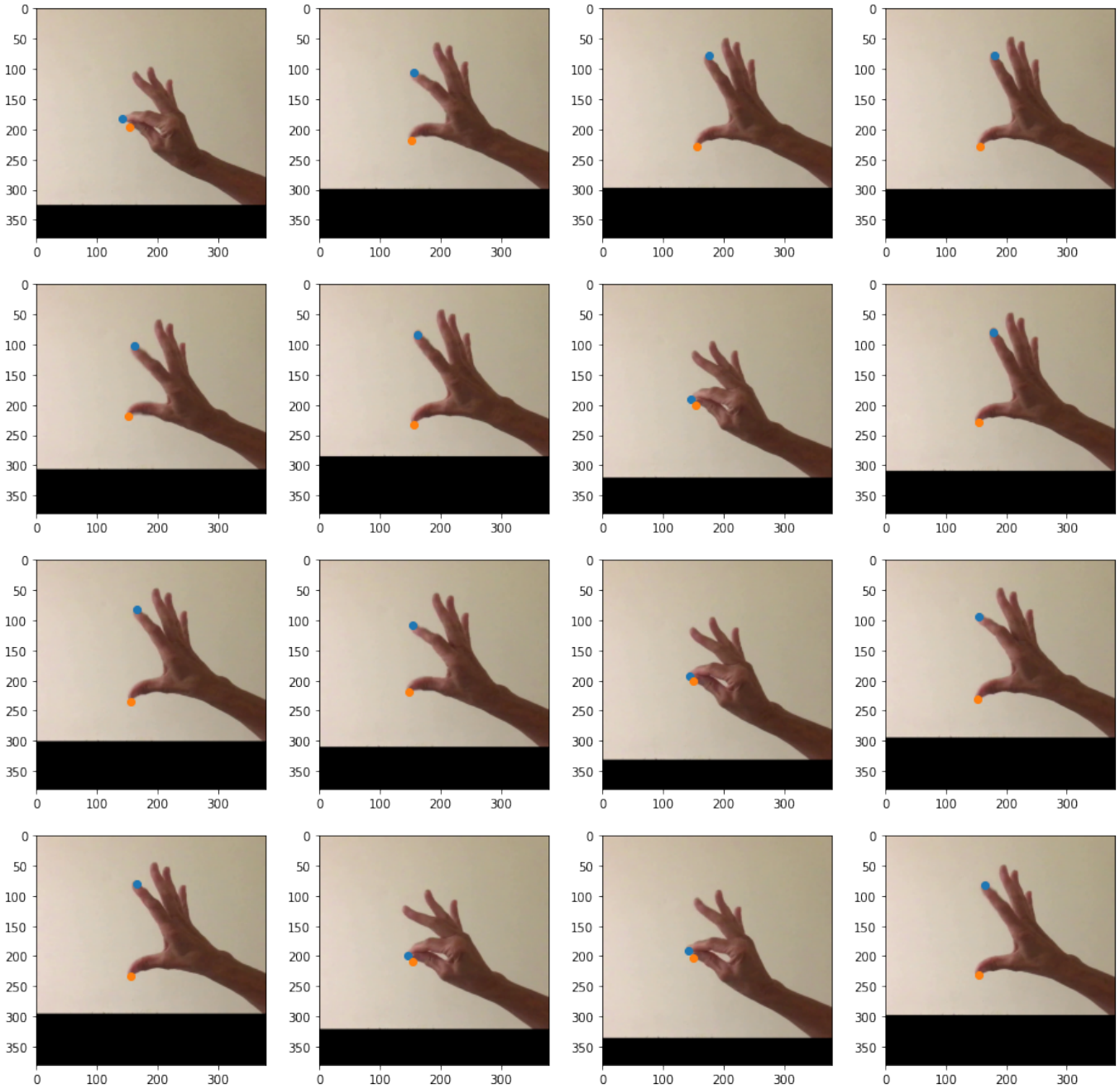

Figure A.2: EfficientNet-B4 Key point regression model: Predictions on Test Data

Bibliography

Agarwal, V. (2020), 'Complete architectural details of all efficientnet models'.
URL: *https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142*

Albawi, S., Mohammed, T. A. & Al-Zawi, S. (2017), Understanding of a convolutional neural network, *in* '2017 International Conference on Engineering and Technology (ICET)', Ieee, pp. 1–6.

Andriluka, M., Pishchulin, L., Gehler, P. & Schiele, B. (2014), 2d human pose estimation: New benchmark and state of the art analysis, *in* 'Proceedings of the IEEE Conference on computer Vision and Pattern Recognition', pp. 3686–3693.

Ayodele, T. O. (2010), 'Types of machine learning algorithms', *New advances in machine learning* **3**, 19–48.

Bhande, A. (2018), 'What is underfitting and overfitting in machine learning and how to deal with it.'.
URL: *https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76*

Bochkovskiy, A., Wang, C.-Y. & Liao, H.-Y. M. (2020), 'Yolov4: Optimal speed and accuracy of object detection', *arXiv preprint arXiv:2004.10934* .

Borad, A. (2021), 'Image classification with efficientnet: Better performance with computational efficiency'.
URL: *https://datamonje.medium.com/image-classification-with-efficientnet-better-performance-with-computational-efficiency-f480fdb00ac6*

Brownlee, J. (2021), 'Gentle introduction to the adam optimization algorithm for deep learning'.
URL: *https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/*

Camomilla, V., Dumas, R. & Cappozzo, A. (2017), 'Human movement analysis: The soft tissue artefact issue', *Journal of biomechanics* **62**, pp–1.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K. & Yuille, A. L. (2017), 'Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs', *IEEE transactions on pattern analysis and machine intelligence* **40**(4), 834–848.

Dai, S. & Wu, Y. (2008), Motion from blur, *in* '2008 IEEE Conference on Computer Vision and Pattern Recognition', IEEE, pp. 1–8.

DeepLizard (2021), 'Convolutional neural networks (cnns) explained'.
URL: *https://deeplizard.com/learn/video/YRhxdVk$_s$Is*

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009), Imagenet: A large-scale hierarchical image database, *in* '2009 IEEE conference on computer vision and pattern recognition', Ieee, pp. 248–255.

Dertat, A. (2017), 'Applied deep learning - part 4: Convolutional neural networks'.
URL: *https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2*

Doshi, S. (2020), 'Various optimization algorithms for training neural network'.
URL: *https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6*

Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S. & Dean, J. (2019), 'A guide to deep learning in healthcare', *Nature medicine* **25**(1), 24–29.

Girshick, R. (2015), 'Fast r-cnn. arxiv 2015', *arXiv preprint arXiv:1504.08083* .

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.

Goetz, C. G., Tilley, B. C., Shaftman, S. R., Stebbins, G. T., Fahn, S., Martinez-Martin, P., Poewe, W., Sampaio, C., Stern, M. B., Dodel, R. et al. (2008), 'Movement disorder society-sponsored revision of the unified parkinson's disease rating scale (mds-updrs): scale presentation and clinimetric testing results', *Movement disorders: official journal of the Movement Disorder Society* **23**(15), 2129–2170.

Gomez, L. F., Morales, A., Orozco-Arroyave, J. R., Daza, R. & Fierrez, J. (2021), Improving parkinson detection using dynamic features from evoked expressions in video, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 1562–1570.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep learning*, MIT press.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J. et al. (2018), 'Recent advances in convolutional neural networks', *Pattern Recognition* **77**, 354–377.

He, K., Girshick, R. & Dollár, P. (2019), Rethinking imagenet pre-training, *in* 'Proceedings of the IEEE/CVF International Conference on Computer Vision', pp. 4918–4927.

He, K., Zhang, X., Ren, S. & Sun, J. (2015), 'Spatial pyramid pooling in deep convolutional networks for visual recognition', *IEEE transactions on pattern analysis and machine intelligence* **37**(9), 1904–1916.

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.

Heldman, D. A., Giuffrida, J. P., Chen, R., Payne, M., Mazzella, F., Duker, A. P., Sahay, A., Kim, S. J., Revilla, F. J. & Espay, A. J. (2011), 'The modified bradykinesia rating scale for parkinson's disease: reliability and comparison with kinematic measures', *Movement Disorders* **26**(10), 1859–1863.

Hu, J., Shen, L. & Sun, G. (2018), Squeeze-and-excitation networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 7132–7141.

Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M. & Schiele, B. (2016), Deepercut: A deeper, stronger, and faster multi-person pose estimation model, *in* 'European Conference on Computer Vision', Springer, pp. 34–50.

James, M. (2017).
URL: *https://www.i-programmer.info/news/202-number-crunching/11140-imagenet-training-record-24-minutes.html*

Jiang, Z., Zhao, L., Li, S. & Jia, Y. (2020), 'Real-time object detection method based on improved yolov4-tiny', *arXiv preprint arXiv:2011.04244* .

Jordan, M. I. & Mitchell, T. M. (2015), 'Machine learning: Trends, perspectives, and prospects', *Science* **349**(6245), 255–260.

Khan, T., Nyholm, D., Westin, J. & Dougherty, M. (2014), 'A computer vision framework for finger-tapping evaluation in parkinson's disease', *Artificial intelligence in medicine* **60**(1), 27–40.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kornblith, S., Shlens, J. & Le, Q. V. (2019), Do better imagenet models transfer better?, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 2661–2671.

Krizhevsky, A., Hinton, G. et al. (2009), 'Learning multiple layers of features from tiny images'.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), 'Imagenet classification with deep convolutional neural networks', *Advances in neural information processing systems* **25**, 1097–1105.

Kızrak, A. (2020), 'Comparison of activation functions for deep neural networks'.
URL: *https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a*

LeCun, Y., Bengio, Y. & Hinton, G. (2015), 'Deep learning', *nature* **521**(7553), 436–444.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W. & Jackel, L. (1989), 'Handwritten digit recognition with a back-propagation network', *Advances in neural information processing systems* **2**.

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

LeCun, Y. & Cortes, C. (2010), 'MNIST handwritten digit database'.
URL: *http://yann.lecun.com/exdb/mnist/*

Lee, C. Y., Kang, S. J., Hong, S.-K., Ma, H.-I., Lee, U. & Kim, Y. J. (2016), 'A validation study of a smartphone-based finger tapping application for quantitative assessment of bradykinesia in parkinson's disease', *PloS one* **11**(7), e0158852.

Li, Y., Yang, S., Zhang, S., Wang, Z., Yang, W., Xia, S.-T. & Zhou, E. (2021), 'Is 2d heatmap representation even necessary for human pose estimation?', *arXiv preprint arXiv:2107.03332* .

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.

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* 'European conference on computer vision', Springer, pp. 740–755.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. & Berg, A. C. (2016), Ssd: Single shot multibox detector, *in* 'European conference on computer vision', Springer, pp. 21–37.

Liu, Y., Chen, J., Hu, C., Ma, Y., Ge, D., Miao, S., Xue, Y. & Li, L. (2019), 'Vision-based method for automatic quantification of parkinsonian bradykinesia', *IEEE Transactions on Neural Systems and Rehabilitation Engineering* **27**(10), 1952–1961.

Lu, L., Shin, Y., Su, Y. & Karniadakis, G. E. (2019), 'Dying relu and initialization: Theory and numerical examples', *arXiv preprint arXiv:1903.06733* .

Mathis, A., Biasi, T., Schneider, S., Yuksekgonul, M., Rogers, B., Bethge, M. & Mathis, M. W. (2021), Pretraining boosts out-of-domain robustness for pose estimation, *in* 'Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision', pp. 1859–1868.

Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W. & Bethge, M. (2018), 'Deeplabcut: markerless pose estimation of user-defined body parts with deep learning', *Nature neuroscience* **21**(9), 1281–1289.

Mathis, A., Schneider, S., Lauer, J. & Mathis, M. W. (2020), 'A primer on motion capture with deep learning: principles, pitfalls, and perspectives', *Neuron* **108**(1), 44–65.

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.

Miller, J. L. (2002), 'Parkinson's disease primer', *Geriatric nursing* **23**(2), 69–75.

Misra, D. (2019), 'Mish: A self regularized non-monotonic neural activation function', *arXiv preprint arXiv:1908.08681* **4**, 2.

Moeslund, T. B., Hilton, A. & Krüger, V. (2006), 'A survey of advances in vision-based human motion capture and analysis', *Computer vision and image understanding* **104**(2-3), 90–126.

Nath, T., Mathis, A., Chen, A. C., Patel, A., Bethge, M. & Mathis, M. W. (2019), 'Using deeplabcut for 3d markerless pose estimation across species and behaviors', *Nature protocols* **14**(7), 2152–2176.

Neill, S. P. & Hashemi, M. R. (2018), 'Ocean modelling for resource characterization', *Fundamentals of ocean renewable energy* pp. 193–235.

Newell, A., Yang, K. & Deng, J. (2016), Stacked hourglass networks for human pose estimation, *in* 'European conference on computer vision', Springer, pp. 483–499.

Newzoo (2021), 'Top countries by smartphone users'.
URL: *https://newzoo.com/insights/rankings/top-countries-by-smartphone-penetration-and-users/*

PaperswithCode (2021*a*).
URL: *https://paperswithcode.com/sota/image-classification-on-imagenet*

PapersWithCode (2021*b*), 'Papers with code - browse the state-of-the-art in machine learning'. **URL:** *https://paperswithcode.com/sota*

PapersWithCode (n.d.), 'Papers with code - early stopping explained'. **URL:** *https://paperswithcode.com/method/early-stopping*

Poppe, R. (2007), 'Vision-based human motion analysis: An overview', *Computer vision and image understanding* **108**(1-2), 4–18.

Ramachandran, P., Zoph, B. & Le, Q. V. (2017), 'Searching for activation functions', *arXiv preprint arXiv:1710.05941* .

Rasamoelina, A. D., Adjailia, F. & Sinčák, P. (2020), A review of activation function for artificial neural network, *in* '2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)', IEEE, pp. 281–286.

Redmon, J. (2013–2016), 'Darknet: Open source neural networks in c', http://pjreddie.com/darknet/.

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.

Redmon, J. & Farhadi, A. (2017), Yolo9000: better, faster, stronger, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 7263–7271.

Redmon, J. & Farhadi, A. (2018), 'Yolov3: An incremental improvement', *arXiv preprint arXiv:1804.02767* .

Ren, S., He, K., Girshick, R. & Sun, J. (2015), 'Faster r-cnn: Towards real-time object detection with region proposal networks', *Advances in neural information processing systems* **28**, 91–99.

Ridnik, T., Ben-Baruch, E., Noy, A. & Zelnik-Manor, L. (2021), 'Imagenet-21k pretraining for the masses', *arXiv preprint arXiv:2104.10972* .

Riecickỳ, A., Madaras, M., Piovarci, M. & Durikovic, R. (2018), Optical-inertial synchronization of mocap suit with single camera setup for reliable position tracking., *in* 'VISIGRAPP (1: GRAPP)', pp. 40–47.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. (2015), 'ImageNet Large Scale Visual Recognition Challenge', *International Journal of Computer Vision (IJCV)* **115**(3), 211–252.

Ržička, E., Krupička, R., Zárubová, K., Rusz, J., Jech, R. & Szabó, Z. (2016), 'Tests of manual dexterity and speed in parkinson's disease: Not all measure the same', *Parkinsonism & related disorders* **28**, 118–123.

Samuel, A. L. (1959), 'Some studies in machine learning using the game of checkers', *IBM Journal of research and development* **3**(3), 210–229.

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.

Sano, Y., Kandori, A., Shima, K., Yamaguchi, Y., Tsuji, T., Noda, M., Higashikawa, F., Yokoe, M. & Sakoda, S. (2016), 'Quantifying parkinson's disease finger-tapping severity by extracting and synthesizing finger motion properties', *Medical & biological engineering & computing* **54**(6), 953–965.

Scherer, D., Müller, A. & Behnke, S. (2010), Evaluation of pooling operations in convolutional architectures for object recognition, *in* 'International conference on artificial neural networks', Springer, pp. 92–101.

Shahid, A. H. & Singh, M. P. (2020), 'A deep learning approach for prediction of parkinson's disease progression', *Biomedical Engineering Letters* **10**(2), 227–239.

Sibley, K. G., Girges, C., Hoque, E. & Foltynie, T. (2021), 'Video-based analyses of parkinson's disease severity: A brief review', *Journal of Parkinson's Disease* (Preprint), 1–11.

Simonyan, K. & Zisserman, A. (2014), 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556* .

Swapna (2021), 'Convolutional neural network: Deep learning'.
URL: *https://developersbreach.com/convolution-neural-network-deep-learning/*

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.

Tan, M. & Le, Q. (2019), Efficientnet: Rethinking model scaling for convolutional neural networks, *in* 'International Conference on Machine Learning', PMLR, pp. 6105–6114.

Tompson, J. J., Jain, A., LeCun, Y. & Bregler, C. (2014), 'Joint training of a convolutional network and a graphical model for human pose estimation', *Advances in neural information processing systems* **27**, 1799–1807.

Uijlings, J. R., Van De Sande, K. E., Gevers, T. & Smeulders, A. W. (2013), 'Selective search for object recognition', *International journal of computer vision* **104**(2), 154–171.

Voulodimos, A., Doulamis, N., Doulamis, A. & Protopapadakis, E. (2018), 'Deep learning for computer vision: A brief review', *Computational intelligence and neuroscience* **2018**.

Williams, S., Relton, S., Fang, H., Alty, J., Qahwaji, R., Graham, C. D. & Wong, D. C. (2020), 'Supervised classification of bradykinesia in parkinson's disease from smartphone videos', *Artificial Intelligence in Medicine* **110**, 101966.

Wu, X., Sahoo, D. & Hoi, S. C. (2020), 'Recent advances in deep learning for object detection', *Neurocomputing* **396**, 39–64.

Xiao, B., Wu, H. & Wei, Y. (2018), Simple baselines for human pose estimation and tracking, *in* 'Proceedings of the European conference on computer vision (ECCV)', pp. 466–481.

Yin (2018), 'A summary of neural network layers'.
URL: *https://medium.com/machine-learning-for-li/different-convolutional-layers-43dc146f4d0e*

Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L. & Grundmann, M. (2020), 'Mediapipe hands: On-device real-time hand tracking', *arXiv preprint arXiv:2006.10214* .

Zhao, Z., Williams, S., Hafeez, A., Wong, D., Relton, S., Fang, H. & Alty, J. (2020), 'The discerning eye of computer vision: Can it measure parkinson's finger tap bradykinesia?', *Journal of the Neurological Sciences* **416**, 117003.