DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2020

An Efficient Deep Learning Solution to 3D Hand Pose Estimation for XR Applications using Monochrome Cameras

YASMIN BABA

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

An Efficient Deep Learning Solution to 3D Hand Pose Estimation for XR Applications using Monochrome Cameras

YASMIN BABA

Master in Computer Science
Date: June 22, 2020
Supervisor: Haibo Li
Examiner: Danica Kragic Jensfelt
School of Electrical Engineering and Computer Science
Host company: ManoMotion AB
Swedish title: En Energibesparande Djupinlärningslösning på Handpositionsuppskattning i 3D för XR-applikationer med hjälp av Monokroma Kameror

Abstract

Recent advancements in the field of deep learning are providing new ways to naturalise user interactions in extended reality (XR) applications. An effective approach is through the use of 3D hand tracking, enabling users to interact with virtual content using their hands, rather than hand-held controllers. 3D hand tracking is a computer vision task most commonly referred to as 3D hand pose estimation, which aims to estimate the location of human hand joints from images and videos. This thesis proposes a novel and lightweight deep learning solution to 3D hand pose estimation, providing a model that is suitable for real-time use on embedded devices, such as XR headsets. The proposed solution is split into two deep neural networks: a convolutional neural network that performs 2D hand joint estimation and a feed-forward neural network that lifts 2D hand joint coordinates to 3D by inferring the final depth coordinates. The design of the proposed 2D joint prediction sub-network incorporates parts of the MobileNetV2 architecture, in addition to a final differentiable spatial to numerical transform layer that allows for accurate numerical coordinate regression. The depth estimation sub-network utilises a simple multi-stage feed-forward architecture to provide iterative refinement of the predicted depth coordinates. With a current lack of research and solutions surrounding the use of monochrome cameras in hand pose estimation, this work provides a first look into hand pose estimation using monochrome data in addition to an analysis regarding how the loss of colour information within training data impacts the performance of a hand pose prediction model.

Sammanfattning

Ny teknologisk utveckling inom fältet djupinlärning har lett till nya sätt att naturliggöra en användares interaktioner inom XR-applikationer. Ett effektivt sätt att närma sig detta är användandet av handspårning i 3D, vilket låter användare interagera med virtuellt innehåll med sina händer, snarare än med hjälp av kontroller. Handspårning i 3D är en uppgift för datorseende som oftast refereras till som 3D-handpositionsuppskattning, och som söker lokalisera handens leder utifrån bilder och videor. Den här uppsatsen föreslår en ny och lättviktig djupinlärningslösning på handpositionsuppskattning i 3D, och tillhandahåller en modell som passar för realtidsanvändning på inbäddade enheter, såsom XR-headset. Den föreslagna lösningen är delad i två djupa neuronnät: ett faltningsneuronnätverk som utför uppskattning av handens leder i 2D, och ett feed forward-neuronnärverk som lyfter 2D-koordinaterna till 3D genom att inferera djup. Utformningen av det föreslagna subnätverket som förutsäger leder i 2D inkorporerar delar av MobileNetV2-arkitekturen, utöver ett slutligt, deriverbart lager som omvandlar spatiala data till numerisk data, och tillåter träffsäker numerisk koordinatregression. Sub-nätverket för djupuppskattning använder en enkel feed-forward arkitektur i flera steg för att tillhandahålla repeterad förfining av de förutspådda djupkoordinaterna. I den nuvarande bristen på forskning och lösningar rörande användandet av monokroma kameror i handpositionsuppskattning tillhandahåller det här arbetet en första titt på handpositionsuppskattning med hjälp av monokroma data, utöver en analys rörande hur förlusten av färginformation inom träningsdata påverkar prestationen hos en modell för förutsägande av handposition.

Contents

1 Introduction

1.1 Background

The field of deep learning is paving the way for the future of extended reality (XR) applications [1], where XR is an umbrella term that refers to any computer-altered realities, namely virtual reality (VR), augmented reality (AR) and mixed reality (MR). The implementation of eye tracking, natural language processing and body pose estimation via deep learning are a few popular approaches that are providing users with a more immersive experience; by naturalising user interactions within XR environments, they are expanding the possibilities and demand for future XR applications [2]. Another effective way to naturalise user interactions is by utilising 3D hand tracking, rendering hardware such as hand-held controllers obsolete. The use of hands delivers a stronger sense of user presence, allowing for more intuitive interactions with virtual objects [3]. Due to recent advancements in deep learning solutions, hand tracking is now robust enough to be introduced into the XR market [1]. For instance, Oculus Quest is the first VR headset on the market to support 3D hand tracking features; made by Facebook, its recent release has sparked a growing amount of attention within the VR development community [4].

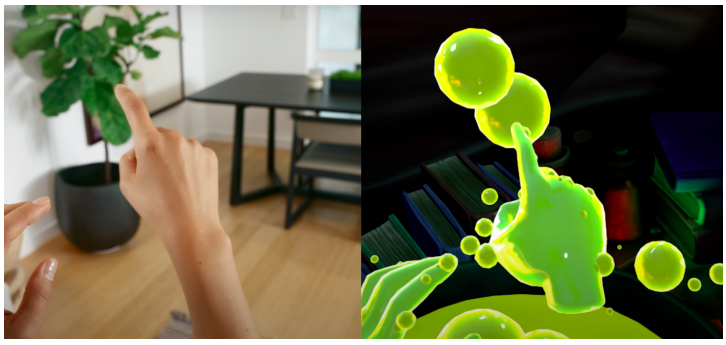


Figure 1.1: Demo of Oculus Quest's Hand Tracking Technology [4]

Hand tracking aims to automatically detect sets of 2D or 3D hand joint locations from images and videos, a problem most commonly referred to as hand pose estimation. Although hand pose estimation has been referenced to as early as the nineties, activity in the field was fairly recently ignited by the appearance of RGB-D cameras [5]. RGB-D cameras apprehend high-quality depth maps of an image scene via depth sensors, providing invariance to background changes in addition to robustness against lighting conditions [6]. However, RGB-D cameras tend to fail in outdoor environments and have relatively high power consumption, often rendering them unsuitable for use in embedded devices [6]. More recently, the advent of deep learning has enabled researchers to explore the use of 2D RGB cameras, as convolutional neural networks (CNNs) have introduced new levels of accuracy and robustness in addition to faster execution times [7]. These cameras are cheaper than RGB-D cameras and are more commonly found in consumer devices such as smart-phones and computers. Nevertheless, 3D hand joint estimation from monocular RGB images is a more difficult task due to their intrinsic scale and depth ambiguity [8]. Training an end-to-end CNN to infer 3D joint coordinates from images is challenging and requires a large amount of annotated 3D training data, which is not easily obtainable. It is therefore more common to utilise CNNs to only estimate 2D joint locations, which can then be lifted to 3D via a variety of techniques. Lifting hand joint coordinates is a very complex task as the space of possible 3D poses consistent with the 2D locations is infinite [9]. Standard approaches often require post-processing in order to comply with skeletal constraints and bone-lengths, which are often too computationally complex to incorporate into an embedded device [3]. Deep learning approaches have recently provided alternative solutions which have produced some promising results that this work aims to extend upon [10].

The majority of XR headsets currently favour the use of monochrome cameras over RGB cameras, which is suspected to be due to their lower cost and greater sensitivity to low-light [11]. Although primarily used for positional tracking, these attributes suggest that monochrome cameras may also be more suited for hand tracking applications. However, no published research to date has been found to perform hand pose estimation using monochrome cameras. Due to growing prospects surrounding 3D hand tracking in XR applications, this thesis aims to bridge the gap in research by proposing a lightweight deep learning solution for use on XR headsets using monochrome cameras in addition to providing new insights into how the use of monochrome data over coloured data impacts the performance of a 3D hand pose prediction model.

1.2 Challenges

Hand pose estimation is a challenging problem as it is highly dependent on the viewpoint of the hand being observed. The human hand has 27 degrees of freedom [12], is chromatically uniform in appearance and suffers from severe self-occlusions [13]; these factors cause difficulty in inferring accurate hand joint locations in unconstrained environments. Although recent deep learning solutions have overcome these challenges, a large amount of diverse data is required to achieve state-of-the-art results. Obtaining such quantities of data is a challenge in itself and annotated data sets tend to contain a significant amount of noise, which hinders the solution's performance [14].

1.3 Problem Formulation

This thesis was proposed by ManoMotion AB, a Swedish start-up that builds state-of-the-art hand tracking technologies and is seeking to integrate its algorithms into XR headsets using monochrome cameras. The motivation behind the research conducted in this thesis stems from a lack of published work surrounding the use of monochrome cameras in 3D hand pose estimation. To bridge the gap in research, the contribution of this thesis is twofold.

Firstly, this work aims to present a novel deep learning solution to the 3D hand pose estimation problem, specifically for use on embedded devices using monochrome cameras such as XR headsets. The solution must output accurate, real-time 3D hand joint predictions from an egocentric viewpoint and will provide a first look into 3D hand pose estimation using monochrome cameras. Therefore, the first research question explored is:

How can deep learning be used to build an accurate 3D hand pose prediction model for real-time use in XR headsets using monochrome cameras?

Secondly, an additional aim of this work is to provide new insights regarding the effects of training a hand pose prediction model on monochrome versus coloured data. This will clarify if one type of training data is advantageous over the other and whether XR headsets equipped with monochrome cameras are at a disadvantage in the context of 3D hand tracking. The second research question is therefore defined as:

How does the loss of colour information within training data impact the performance of a hand pose prediction model?

1.4 Social and Ethical Considerations

It is important to consider how accurate 3D hand tracking technologies will impact society once they become widely accessible in the XR market. Although speculative, it is believed that the biggest impact within the near future will stem from connecting head-mounted displays (HMDs) to mobile phones. Qualcomm, a corporation that manufactures most of the chips found in current XR headsets, has recently released a new platform supporting 5G networks and up to seven cameras, sparking speculation that the next generation of HMDs will indeed be driven by mobile phones [15]. Smart glasses may have the biggest impact on society, such as the future release of Apple Glass which is rumoured to utilise hand gestures as inputs and connect to users' iPhones [16]. If such a device is indeed released, it could provide users with an easier way to communicate, scroll and play games right in front of their eyes through the use of hand gestures, which may transform the way mobile phones are utilised all together [17]. Accurate 3D hand tracking is also likely to transform the use cases for other kinds of XR headsets; for instance, the use of hands can provide those who struggle to use hardware and hand-held controllers access to the XR world, which would be particularly revolutionary for those who are restricted by disabilities. Moreover, XR environments could be utilised to connect individuals across the world by projecting their avatar into the same virtual room as the user [18]. This is especially relevant at the time of writing this thesis, where travel bans and social distancing rules are being enforced on a global scale as a result of the Covid-19 outbreak. The implementation of 3D hand tracking will also likely shape the way people learn; for example, it would enable surgeons to practice life-saving procedures without the risk of inflicting harm on real patients [19]. Overall, the accessibility to robust 3D hand tracking technologies will likely impact society on a large scale, specifically regarding how we communicate, work and game on a day-to-day basis.

On the other hand, the ethical implications of employing 3D hand tracking in XR applications must also be considered. In order to perform accurate 3D hand pose estimation, a large amount of data must be recorded to train the prediction model [20]. This raises issues of privacy, as images of people in the background will be gathered most likely without consent. Due to current facial recognition technologies and the creation of Deepfakes, fake images and videos are now indistinguishable from authentic ones, sparking controversy and fear surrounding non-consensual filming [21]. Furthermore,

3D hand tracking may be the subject of discrimination disputes if it fails to work on all types of hands [22]. To avoid this, the training data must include a significant number of hands of different skin colours, ages and shapes; collecting large amounts of data is already challenging and thus needing to collect such a diverse data set introduces further difficulties. Finally, issues of user health must also be reviewed. In terms of physical health, users may experience eye-strain headaches from excessively looking at screens or shoulder discomfort from arm extensions [23]. More threatening, however, are the dangers of users becoming unaware of their real surroundings. For instance, when immersing oneself into virtual reality, users become more susceptible to accidents if they are not in a safe environment [24]. Further to this, there are legitimate concerns regarding the consequences these types of technologies may impose on mental health, as very realistic experiences or addictive use may lead to dissatisfaction with the real world or isolation in a virtual world [25].

1.5 Thesis Outline

Chapter 2 presents a literature review that summarises prior research in the field of 3D hand pose estimation. Chapter 3 provides an overview of the theory behind the techniques used in this work, primarily deep learning methods and how they can be used to achieve 3D hand pose estimation. Chapter 4 describes the methodology behind the proposed model, in addition to the limitations encountered surrounding data availability. Chapter 5 presents the empirical findings from this work and the final model design. Chapter 6 provides a detailed discussion surrounding the acquired results, including how well the model performed, how colour loss impacted its performance and lastly, the scope for future work. Chapter 7 provides a summary of the final conclusions.

2 Literature Review

3D hand pose estimation is a computer vision task that aims to locate 3D joint coordinates of human hands from images and videos. The subject has gained an increasing amount of attention from researchers in the last decade due to technological advances that have enabled drastic improvements in predictive performance [1]. This chapter provides a summary of the related work associated with the topic of this thesis. Technical details will be explored later in Chapter 3.

The field of 3D hand pose estimation has moved away from classical approaches that employ wearable hardware, such as data gloves [26] or body cameras [27], to vision-based techniques that no longer require user instrumentation. The most common vision-based techniques can be categorised into two approaches, namely generative and discriminative methods. Generative methods, also known as model-based or top-down approaches, treat 3D hand pose estimation as an optimization problem and generally estimate pose parameters by fitting a hand template to features observed in the input data [28][29] [30]. These methods rely heavily on model initialisation and temporal information, such as the anatomical size of the hand and hand motion constraints. Due to this, these methods suffer to recover from errors and often need to be re-initialised during running time [31]. Discriminative methods, on the other hand, directly infer joint locations from images without the need for explicit size or motion constraints. Popular data-driven algorithms such as nearest neighbour [32], decision forests [33] [34] and convolutional neural networks [5] [35] are capable of automatically extracting the direct mapping between input features and hand pose parameters, as all necessary information is encoded in the data they learn from. Although less accurate, discriminative methods are generally preferred for real-time hand pose estimation as they are much faster and do not require initialisation [36]. Some works have implemented a combination of generative and discriminative methods in order to extract their corresponding benefits [37] [38] [7] [39]. These approaches are

known as hybrid methods and most often employ a discriminative method to arrive quickly at an intermediate solution, which is then refined by a generative component. For example, Facebook's Oculus Quest achieves robust 3D hand tracking through the use of a hybrid method, employing a deep learning approach for 2D joint estimation and model fitting for 3D pose recovery [4].

A discriminative method that has achieved the most success in articulated pose estimation is the convolutional neural network (CNN), a deep learning solution that has introduced new levels of accuracy and efficiency, particularly from monocular RGB images [40] [41] [42]. Classical architectures use regression techniques to learn the direct mapping between an input image and the corresponding output joint coordinates [43] [44][45]. However, this direct mapping, whether it is to 2D or 3D, is highly non-linear and is difficult for a CNN to learn when trained via direct numerical coordinate regression [35]. CNN training was later revolutionised through the use of heatmaps following the work of Tompson et al. [39], who used CNNs to infer 2D hand joint locations and inverse kinematics to infer 3D hand pose; heatmaps are 2D feature maps generated by CNNs that represent the probability of a joint occurring in a spatial location. Training via heatmaps increases a network's ability to learn; however, this approach only provides 2D information. Therefore, in order to infer 3D joint coordinates, post-processing must be implemented to retrieve the final depth coordinates [35]. Ge et al. [35] overcame this problem through the use of multiple cameras, fusing multi-view heatmaps together to extract predicted 3D hand joint locations. More recent works have proposed end-to-end 3D joint prediction networks, utilising multiple cameras and heatmaps to directly infer 3D hand joint locations, such as those proposed by Qiu et al. [46], Li et al. [8] and Iskakov et al. [47]. Moreover, Moon et al. [48] proposed an end-to-end network that directly infers 3D joint locations from a single depth camera. Overall, these solutions are not attractive in the context of this thesis as they use quite heavy and complex 3D CNN configurations and require a large amount of 3D labelled data, which is difficult to obtain. As many hand tracking devices do not use multiple cameras or depth sensors, it was more common to find solutions that firstly infer 2D joint locations via CNNS and then estimate final depth coordinates through 2D-to-3D lifting techniques. Previous works have achieved lifting through model-based approaches [7], computer vision techniques [47] [3] or deep learning methods [10] [41]. Unlike the former, deep learning methods do not require post-processing in order to comply with skeletal constraints or bone-lengths, thus making them, from a computational standpoint, more suitable for use on embedded devices [3].

CNNs are specifically used in image analysis, as their efficient intrinsic design allows for fast training and prediction times [3]. Efficiency primarily stems from the use of convolution computations throughout the network. In order to use CNNs in real-time on embedded devices, a large amount of research has been dedicated to designing CNN architectures that achieve greater speed by improving the way convolution operations are performed. One approach to speeding up a CNN is to reduce its number of parameters, which in turn reduces the number of convolution operations. SqueezeNet [49] achieves this through the introduction of a Fire module, which uses squeeze and expand layers to reduce the number of channels involved in convolution operations. Other works have focused on improving the convolution operation itself, such as MobileNet [50] which introduced the depth-wise separable convolution, an operation that eliminates redundant calculations found within standard convolution. EffNet [51] provides a similar solution, using spatial separable convolutions instead. ShuffleNet [52] also achieves computational efficiency via channel shuffle operations and group convolutions, where convolution is performed independently for a group of channels. CondenseNet [53], on the other hand, inherently learns group features by splitting convolutional filters into groups and gradually removing neural connections to less important features per group. Finally, MobileNetV2 [54] expands upon the original MobileNet [50] through the introduction of a resource-efficient block made of inverted residual connections and linear bottlenecks. MobileNetV2 has achieved a significant amount of attention due to its very low number of parameters and online accessibility. In the field of hand pose estimation, Gouidis et al. [55] utilised a MobileNetV2-like architecture to perform accurate 2D hand pose estimation on a mobile phone.

The majority of work within the field of 3D hand pose estimation has achieved high-quality hand tracking through the use of RGB-D cameras, which possess depth sensors [30] [56] [57] [58]. Historically, RGB-D cameras have been preferred over RGB due to their inherent access to high-quality depth maps; however, RGB-D cameras possess constraints regarding natural light as well as limitations on the effective range of observations [7]. Furthermore, due to their relatively high power consumption and cost, they are not commonly found in embedded devices. Following the advent of CNNs, approaches using monocular RGB cameras have managed to largely outperform traditional methods and have moved the field away from the use of RGB-D cameras [35]. Inferring 3D joints from RGB information alone is more desirable, especially

as the majority of the world's camera systems do not record depth information [36]. Whilst conducting this literature review, no published research regarding the use of monochrome cameras within the field of hand pose estimation was found. Monochrome cameras do not only cost less than RGB cameras, but possess higher resolution and greater sensitivity to low-light, rendering them more desirable for mass production and with the ability to produce high-quality images in both indoor and outdoor environments. These attributes stem from the differences in how RGB and monochrome cameras absorb light [59]. RGB cameras consist of individual pixels that can only capture either red, blue or green colours of light, meaning that each pixel effectively captures a third of incoming light. Monochrome cameras, on the other hand, consist of pixels that capture all incoming light, regardless of colour and thus are much more light efficient [60]. The resultant difference in quality between RGB and monochrome cameras is illustrated below in Figure 2.1.

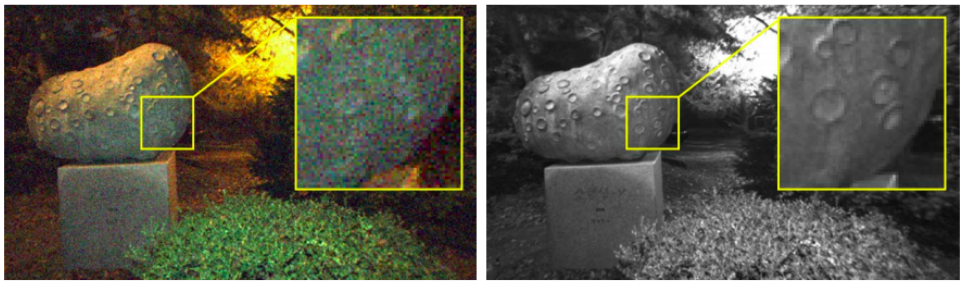


Figure 2.1: Resolution of RGB versus Monochrome Cameras [60]

The lack of published research and usage of monochrome cameras in 3D hand tracking is believed to be a result of a lack of relevant applications. Firstly, consumer devices such as mobile phones and computers already utilise RGB cameras for a wide variety of applications, rendering the use or addition of monochrome cameras needless. Furthermore, viewing hands in grayscale is aesthetically less attractive than viewing them in colour; this intuitively would result in lower demand for monochrome cameras, especially in augmented reality scenarios. These statements, however, are speculative on my behalf as no published research has directly referred to the benefits or drawbacks of using monochrome cameras for 3D hand tracking in XR applications. Instead, it was found that monochrome cameras are commonly used for positional tracking, although published research is predominately associated with vehicle use [61] [62]. Nevertheless, through market research, popular VR headsets developed by Vive, Varjo and Oculus were found to currently utilise monochrome cam-

eras to track users' positions in virtual environments, proving many headsets are indeed equipped with monochrome cameras. Oculus Quest, however, is the first and only VR headset on the market to take advantage of these cameras for 3D hand tracking. Using four monochrome cameras, the headset is capable of recording a wide and high-quality view of the image scene, which is then used to achieve accurate and robust 3D hand tracking [4]. As VR headsets portray simulated environments, the users are never exposed to the real monochrome images being recorded; therefore, there seems to be no disadvantages associated with colour loss in terms of user experience. Behind-the-scenes hand tracking may also be applicable in certain AR/MR applications; for instance, smart glasses project virtual information onto glass, meaning users observe the real world around them but with an additional layer of virtual content. In such a case, hand tracking could be achieved using monochrome cameras, although this is again speculative on my behalf. With the benefits provided by monochrome imagery and the release of Oculus Quest's hand interaction feature, it is expected that other XR headset developers will follow suit in the use of cheap, low-power, high-quality monochrome cameras for 3D hand pose estimation.

In conclusion, extensive research in the field of 3D hand pose estimation has mainly been centered around RGB-D cameras, model-based approaches and more recently, convolutional neural networks. In the context of this thesis, the most attractive solutions in the field utilise convolutional neural networks to infer 2D joint locations and then employ 2D-to-3D lifting techniques. Deep learning has also been proven to provide a lightweight solution to lifting without the need for any post-processing, proving most suitable for use in XR headsets. The aim of this work is to follow the trend in deep learning solutions by designing two separate neural networks: one that infers 2D hand joint coordinates from monocular images and one that infers the corresponding depth coordinates. Furthermore, it is noteworthy that whilst conducting this literature review, no published research was found regarding the use of monochrome imagery or the impacts of colour loss in hand pose estimation. To bridge this gap in research, the proposed solution will be trained on images captured from a monochrome camera and an analysis regarding the effects of colour loss in hand pose estimation will additionally be provided.

3 Theoretical Background

The following chapter summarises the key theoretical concepts required to understand the proposed deep learning solution to the 3D hand pose estimation problem. Sections 3.1 and 3.2 review the topics of machine learning and deep learning respectively. Section 3.3 describes convolutional neural networks and why they are commonly used in image analysis. Section 3.4 presents MobileNetV2, a highly efficient convolutional neural network architecture. Section 3.5 explores the training process of neural networks as well as techniques to improve their performance. Finally, Section 3.6 highlights how deep learning methods can be used to solve pose estimation.

3.1 Machine Learning

Machine learning is a subfield within artificial intelligence used to automatically extract patterns from a set of data in order to build highly accurate prediction models. The most common type of machine learning technique is supervised learning, in which the expected output variable, or label, is provided for each sample in the training data set. Given a set of labelled data, a machine learning model leverages algorithms to 'learn' a function that approximates the relationship between a set of descriptive features (input data) and a target function (output label). Once the model has been exposed to a sufficient amount of examples during training, it can then be used to make accurate predictions on unseen, yet similar data. Supervised learning problems can be grouped into two main types: classification or regression problems. A classification problem is one which outputs a discrete category whereas a regression problem outputs a vector of numerical quantities [63]. This thesis treats 3D hand pose estimation as a supervised regression problem, by which the input data, $\mathbf{x}$, are monochrome images of hands and the labels, $\mathbf{y}$, are the corresponding 3D pixel coordinates of the hands' joints in the image space, as illustrated in Figure 3.1.

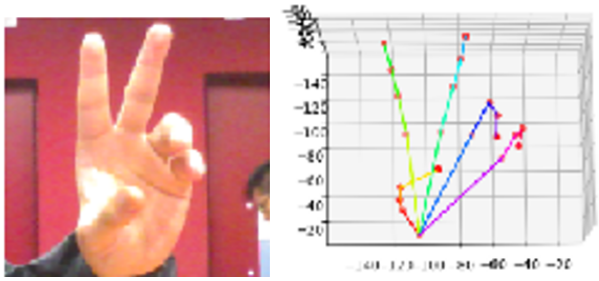


Figure 3.1: Example of an Input Image (left) and Output 3D Coordinate Labels (right) [64]

The main goal of supervised machine learning is to build a prediction model that performs well on unseen data. If this is achieved, the model is said to generalise well and has managed to learn the true underlying relationship between the input and output. Prediction models tend to generalise badly if the model is either too complex or too simple for the provided data set. If the model is too complex, it will learn the patterns in the training data too well, so much so that it learns random noise fluctuations as legitimate patterns. This is known as overfitting and is visualised below in Figure 3.2. A model that overfits the training data will therefore struggle to predict on unseen data as it has failed to approximate the true underlying function. On the other hand, if the model is too simplistic, it will struggle to learn important patterns in the training data and thus will perform badly on both the training data and on unseen data. This phenomenon is referred to as underfitting. A good machine learning model that learns the true underlying input-output function therefore must find a balance between overfitting and underfitting [63]. Techniques to improve a model's generalisation error will be discussed later in Section 3.5.2.

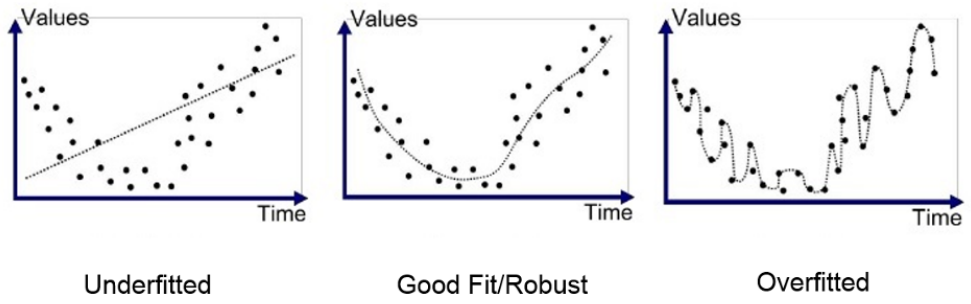


Figure 3.2: Possible Network Performances [65]

3.2 Deep Learning

3.2.1 Artificial Neural Networks

Deep learning is a popular branch of machine learning capable of extracting highly complex patterns in data. Deep learning models extend upon the Perceptron, a simple machine learning algorithm used for binary classification [66]. The simplest type of model is the feed-forward neural network (FNN), also known as the Multi-Layer Perceptron, where information solely flows in the direction of input to output. A FNN consists of an input layer and output layer that are often connected through multiple hidden layers through a series of nodes (see Figure 3.3). In this type of network architecture, each node is connected to all nodes in the previous and next layers. The connections between nodes are parameterised by weight values, which are continuously updated throughout training time to better the prediction model (details on how a network is trained will be discussed in Section 3.5). Each individual node is a perceptron, or more commonly known as an artificial neuron [67].

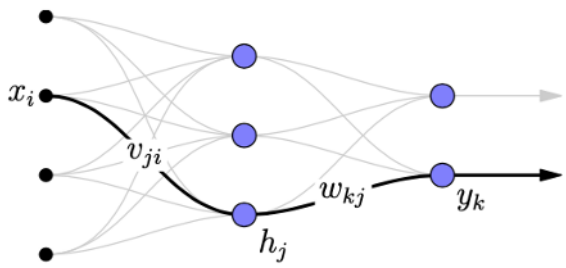


Figure 3.3: An Example of a Simple Feed-Forward Neural Network [68]

Deep learning methods have successfully been used as prediction models due to their ability to extract very complex features in an end-to-end fashion. They achieve this through very deep neural networks, consisting of many hidden layers stacked between the input and output layers. The deeper a neural network, the more complex features it can learn; however, care must be taken to not go too deep as overfitting may occur if the model becomes too complex. Conversely, if too shallow, the network is likely to underfit the training data.

3.2.2 Activation Functions

As highlighted in Figure 3.4, the design of an artificial neuron was inspired by neurons in the brain. Biological neurons receive electrical signals through dendrites and send them to other neurons along axons, depending on whether the signal activity is prominent enough. Artificial neural networks loosely mimic this process through the use of weighted arcs and nodes that contain activation functions. An activation function is analogous to the cell body and is responsible for allowing information to pass from one artificial neuron to the next. It receives a weighted sum of inputs from the previous layer's nodes which is then transformed through a non-linear function [69]. The main purpose of an activation function is to introduce non-linearity into the network, enabling it to learn complex mappings between the input and output. Depending on the activation function's threshold, the information passed onto the next neuron will be amplified or dampened accordingly [67].

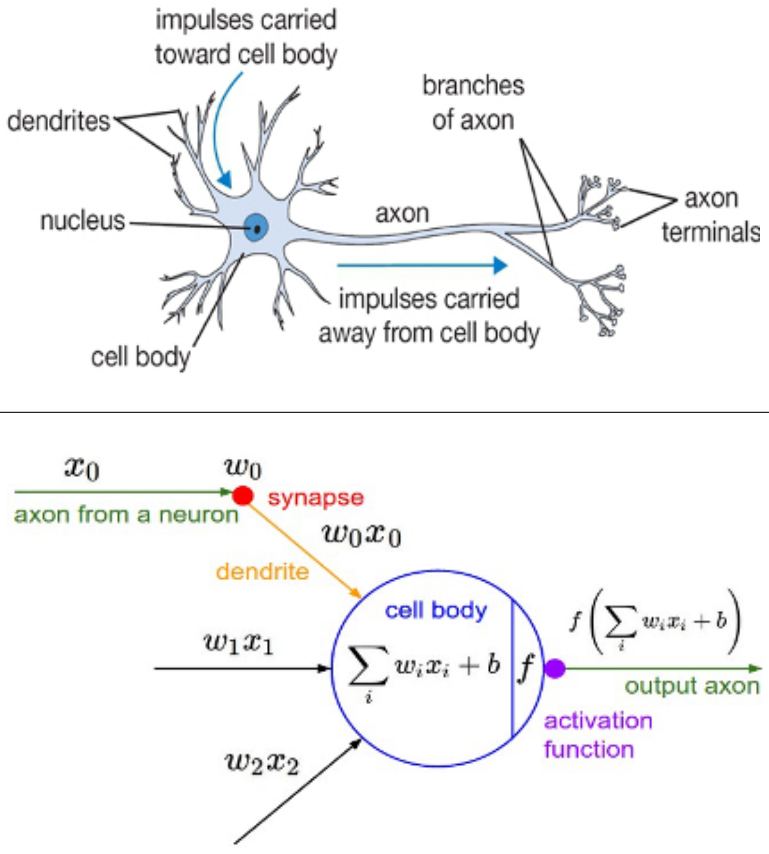


Figure 3.4: Comparison of a Biological Neuron (top) and an Artificial Neuron (bottom) [69]

Historically, the most popular activation functions are the sigmoid and hyperbolic tangent (tanh) functions. As illustrated in Figure 3.5, these functions force large values to 1.0 and small values to either 0.0 or -1.0 and thus are said to saturate extreme values. Once saturated, it becomes difficult for the corresponding weight parameters to be updated later on during training. Saturation is therefore a major drawback in network training as useful information hidden within 'extreme' weight values may be lost [70].

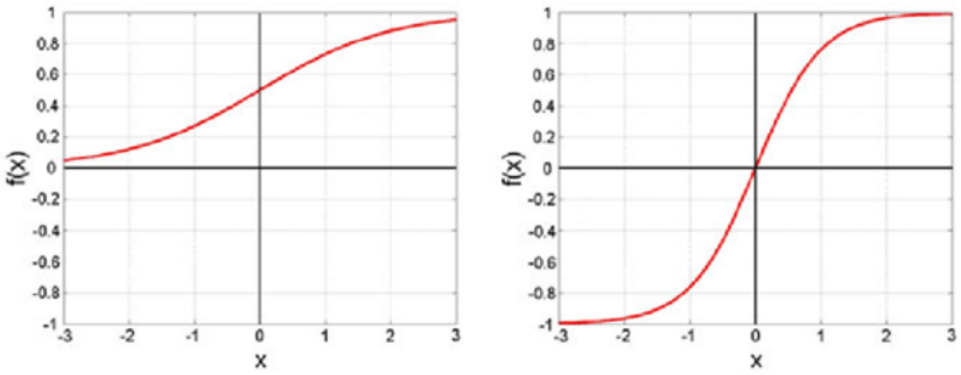


Figure 3.5: Sigmoid (left) and Tanh (right) Activation Functions [70]

Activation functions such as the sigmoid and tanh functions also suffer from an unstable behaviour referred to as vanishing gradients. Neural networks update their parameters by calculating gradients in order to minimise errors (this will be further explained in Section 3.5). Gradients are calculated via the chain rule and therefore if one of the derivative terms is very small, the total product will be equal to or very close to zero. This inhibits weights from being able to update any further during training time [66]. One approach to minimise the vanishing gradient problem is to use a non-linear activation function that seems linear. Linear models have desirable properties: they generalize well, are easy to optimise (no vanishing gradients) and unlike the sigmoid and tanh functions, do not saturate and lose information. The ReLu activation function has become a popular solution that is able to preserve the benefits of a linear function whilst additionally introducing non-linearity into the network. ReLu is a piece-wise linear function, also known as a hinge function, that allows positive values to pass through to the next layer and sets negative values to zero, as illustrated in Figure 3.6 [67].

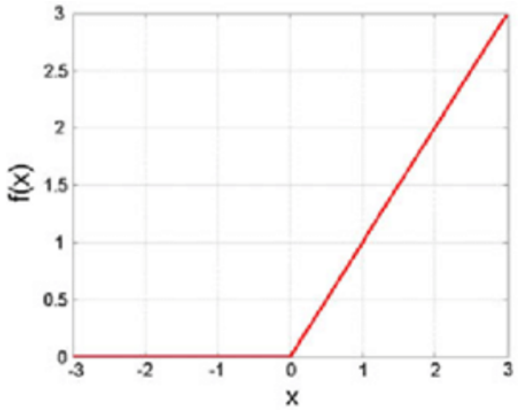


Figure 3.6: The ReLu Activation Function [70]

Models that use ReLu are often easier to train and achieve better performances than those using sigmoid or tanh functions. Firstly, ReLu does not require the use of an exponential calculation, reducing the network's computational complexity. Secondly, ReLu is able to set values to a true zero, whereas the others functions merely approximate it. This introduces sparsity into the network which speeds up learning and simplifies the model. Finally, ReLu is a non-saturating function with a constant gradient value which reduces the likelihood of vanishing gradients occurring.

3.3 Convolutional Neural Networks

Although FNNs are capable of building powerful prediction models, their performance dramatically worsens when analysing images. Images are fed into a neural network as an array of values representing pixel intensities ranging between 0 to 255. As images contain a large number of pixels and often consist of three channels, the number of parameters in the network becomes substantially high. As all neurons in each layer are connected to all neurons in the previous layer, the number of parameters in a FNN rapidly increases as the dimension of the input increases, rendering the network computationally expensive and often intractable [71]. Convolutional neural networks (CNNs) have been designed to overcome this problem through the implementation of weight sharing within the network's layers. This significantly lowers the number of parameters in the network, allowing for a more efficient approach to feature extraction from images [67]. Weight sharing has also been proven to allow CNNs to learn high-level features within an image and achieve spatial generalisation. Spatial generalisation enables the network to perform feature

extraction independent of its location within an image. For instance, if a hand detection model only observes a hand in the bottom left of each image during training time, a CNN will be able to detect a hand in any location within an image during inference time [72]. This is a highly desirable trait for a prediction model and is the primary reason why many refer to CNNs as analogous to the visual cortex in the human brain. CNNs are built in a relatively similar manner to FNNs but achieve greater efficiency through a series of convolutional layers and pooling layers [71].

3.3.1 Convolutional Layers

Convolutional layers consist of filters, or kernels, which are small matrices used to perform convolution operations on pixel values across the image volume. Referring to the example illustrated in Figure 3.7, a 3x3 filter will scan the entire image volume, performing element-wise multiplications on each 3x3 area it passes (referred to as the receptive field). The result is a new feature map, containing single values that represent each receptive field viewed by the filter. Such operations can be implemented on an image without much information loss as neighbouring pixels tend to be highly correlated [73].

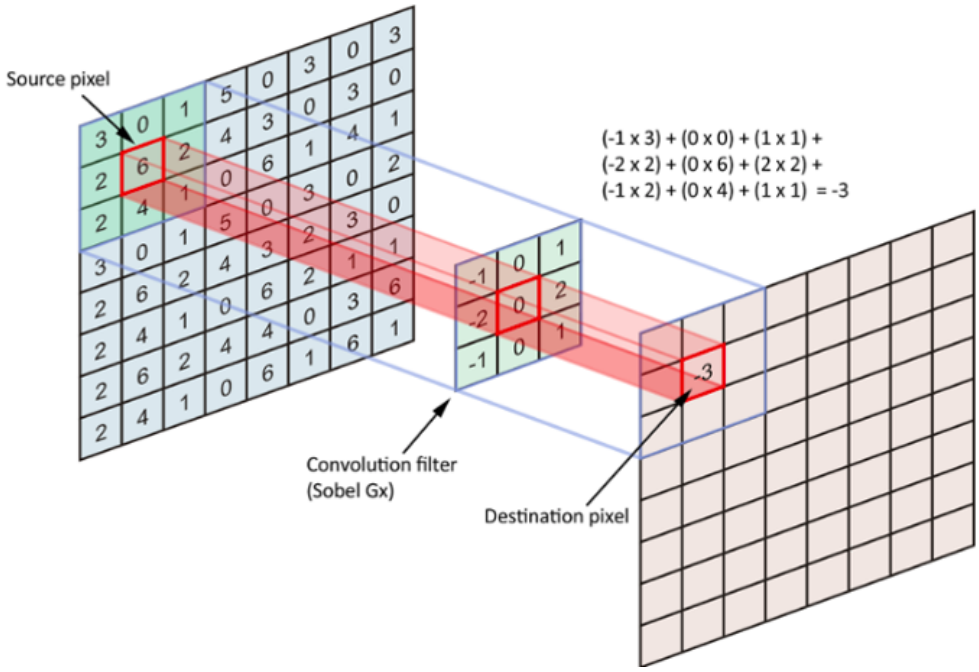


Figure 3.7: Example of a Convolutional Layer [73]

The resulting feature map following a convolutional layer is often smaller than the input feature map. Down-sampling of feature maps is beneficial in terms of efficiency, as the number of parameters from layer to layer reduces which in turn increases the speed of the network [71]. The size of the output feature map is determined by two parameters: stride and padding. Stride controls the horizontal and vertical steps the filter takes across the image. If stride is set to 1, the filter will move one pixel at a time for each horizontal and vertical movement; whereas if set to 2, the filter will move two pixels and so forth. Stride can therefore be used to down-sample the image. However, it may be possible that feature maps throughout the network become too small such that key information is lost. To prevent this, padding can be used to maintain or increase the dimension of a feature map. As shown below, padding adds a border of pixel values (often of value zero) around the feature map in order to prevent dimension reduction [73].

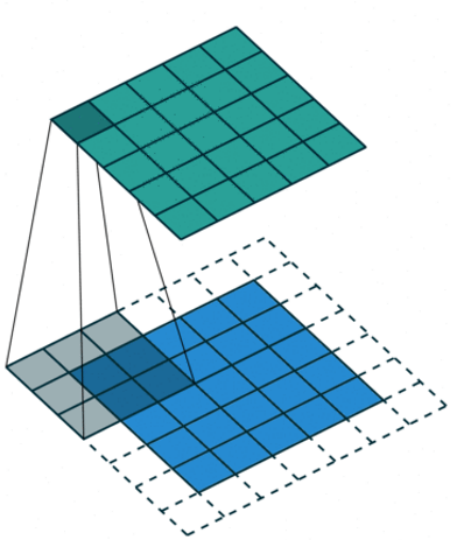


Figure 3.8: Padding in a Convolutional Layer [73]

The weight parameters of a CNN are found in the convolutional filters, which are updated throughout training whilst the network learns the input-to-output mapping. Each filter in and across different layers can be viewed as individual feature detectors. In the first layer, a filter is only capable of detecting low-level features such as edges and colour. The resulting feature map in the second layer will therefore illustrate the most likely locations this particular feature will exist in the image volume. An increasing number of filters will result in a deeper feature map and will provide the network with

more information regarding the features present within the input image. As the number of layers is increased, lower level feature maps merge together into ones that represent higher-level features such as shapes, patterns and objects [74]. An example of patterns recognised by filters throughout layers of a CNN can be visualised, as shown in Figure 3.9.

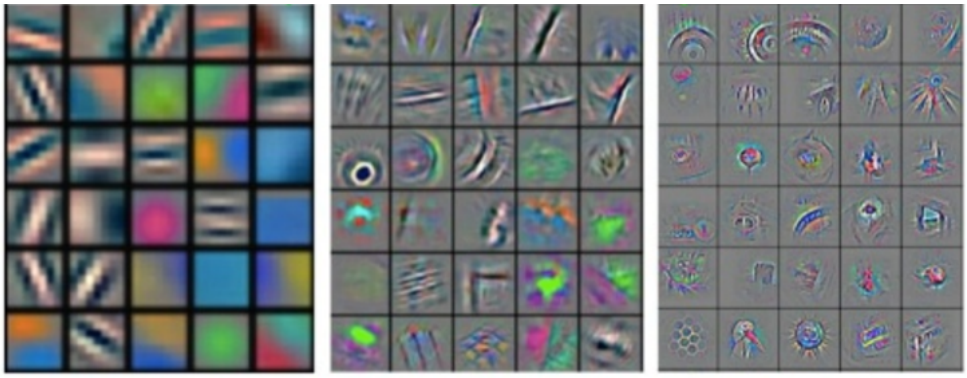


Figure 3.9: Features Detected by Filters in the First Layer (left) to Deeper Layers (right) in a CNN [75]

3.3.2 Pooling Layers

A key problem with feature maps is that they may be sensitive to the location of features within an input. To provide translation invariance, pooling layers are added throughout the network to down-sample feature maps and provide robustness against feature location [76]. A common type of pooling is max pooling, which assigns the maximum value in a selected window to the resulting output feature map [71].

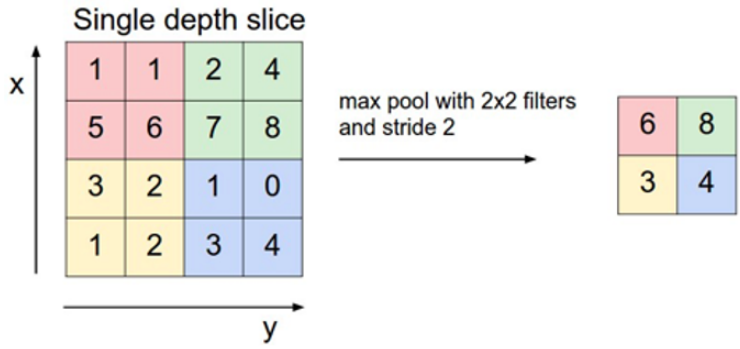


Figure 3.10: Example of Max Pooling [71]

3.3.3 Activation Layers

As discussed in Section 3.2.2, neural networks utilise activation functions in order to introduce non-linearity into the network. Without non-linearity, neural networks would only be capable of learning linear mappings and thus would not be able to extract complex relationships between input features and output labels. In CNNs, non-linearity is introduced via the use of activation layers. The summed results of each convolutional layer are often fed into an activation layer, where they are transformed by an activation function. The most commonly used activation function in CNNs is ReLu due to reasons described in Section 3.2.2 [67].

3.3.4 Fully-Connected Layers

Finally, in order to output a vector of numerical quantities, be it class probabilities or regression values, the last few layers of a CNN often consist of fully-connected layers [71]. These layers are identical to those found in feed-forward neural networks, as described in Section 3.2.1. However, the use of a fully-connected layer as an output layer in hand pose estimation has some significant drawbacks. Fully-connected layers are highly prone to overfitting and tend to worsen the spatial generalisation abilities of a CNN [72]. Details on how numerical hand joint coordinates are extracted from CNNs will be discussed later in Section 3.6.

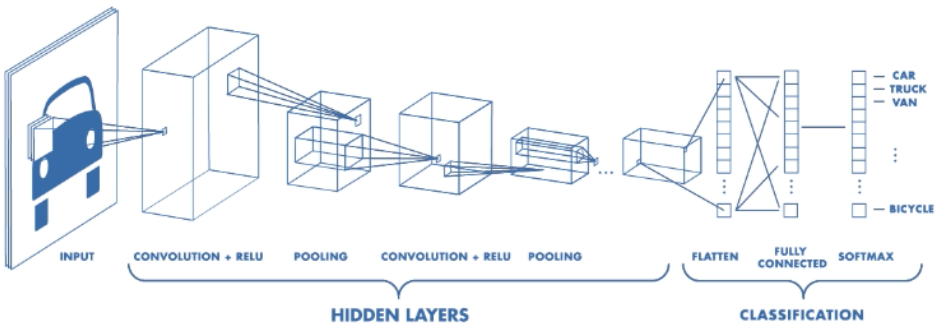


Figure 3.11: Example of an End-To-End Convolutional Neural Network [73]

3.4 MobileNetV2

MobileNetV2 [54] is a popular lightweight convolutional neural network that forms part of the proposed solution. The network is an extended version of MobileNet [50], which introduced a new type of efficient convolution known as the depth-wise separable convolution. To understand the benefits of depth-wise separable convolutions, it is worth reviewing how standard convolutional layers in CNNs perform convolution computations. Taking the example shown in Figure 3.12, a $3\times3\times3$ convolutional filter traverses the input volume, performing element-wise multiplications on each field it passes. Each convolution operation results in a single value in the output feature map, rendering it unit depth. In other words, the inner product is taken across each channel of the feature map with the corresponding channel in the filter, and the results are summed into a single value.

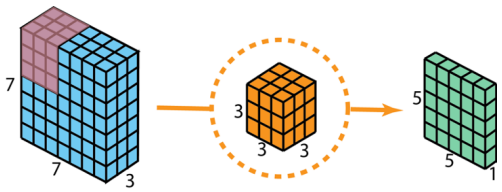


Figure 3.12: A Multi-Channel Convolution Operation [77]

As described in Section 3.3.1, filters are used to detect different features in the image space and thus there tends to be many filters within each convolutional layer. As an example, if 128 filters exist within a layer, the corresponding output feature map will have a depth of 128, as each unit matrix resulting from each filter convolution will be stacked together, as shown in Figure 3.13. Overall, every channel of the input volume will be filtered 128 times, and this is where an inefficiency lies.

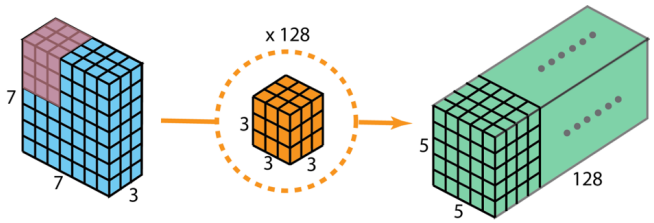


Figure 3.13: A Convolutional Layer with Multiple Filters [77]

Depth-wise separable convolutions achieve the same spatial filtering, but at a much faster rate by splitting the operation into the following two layers: a depth-wise convolutional layer and a point-wise convolutional layer.

1. Depth-wise convolutional layers achieve spatial filtering by applying a single convolutional filter per channel of the input feature map. The resulting output feature maps per channel are then stacked together as shown in Figure 3.14.

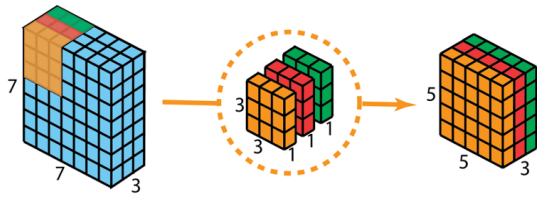


Figure 3.14: Depth-Wise Convolutional Layer [77]

2. Point-wise convolutional layers pass a 1×1 convolutional filter through the stacked feature maps resulting from the depth-wise convolutional layer. This produces linear combinations of the input channels to construct new features.

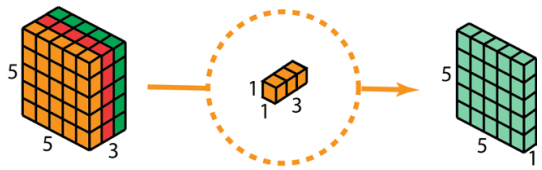


Figure 3.15: Point-Wise Convolutional Layer [77]

In order to construct multiple feature detectors, more filters can be added in this layer. For example, to achieve an output of depth 128, 128 (1×1) convolutional filters must be present in a point-wise convolutional layer.

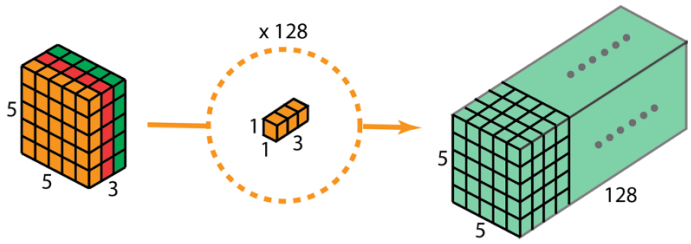


Figure 3.16: Point-Wise Convolutional Layer with Multiple Filters [77]

Performing convolution operations in this manner radically increases the speed of the network [50]. To understand the scale of MobileNet's efficiency, consider the example illustrated in the previous figures. If computed using standard convolution, there would be 128 ($3\times3\times3$) filters that would move 7×7 times, resulting in $128\times3\times3\times3\times7\times7$ = 169,344 multiplications. Using depth-wise separable convolutions, there are 3 ($3\times3\times1$) filters that move 7×7 times, resulting in $3\times3\times3\times7\times7$ = 1,323 multiplications in the first layer. Then there are $128\times1\times1\times3\times5\times5$ = 9,600 multiplications in the second layer, giving a total of 10,923 multiplications overall. Depth-wise separable convolutions therefore resulted in $15\times$ fewer parameters than standard convolutions in this example, providing a scale of efficiency that is highly desirable for real-time use in embedded devices. Another way to interpret the effect of using depth-wise separable convolutions is to observe that the input feature map will be transformed 128 times in normal convolutional layers, whereas it is only transformed once and then elongated into 128 channels using depth-wise separable convolutional layers. Saving the input feature map from being transformed multiple times thus reduces the computational cost of the network [78].

MobileNetV2 [54] introduces a neural network architecture that extends upon the depth-wise separable convolutions introduced in the original MobileNet paper [50]. It achieves greater performance through the use of inverted residual and linear bottleneck layers. Each of these layers receives a low dimensional feature map and perform separable convolutions in three steps [54]:

1. The feature map is firstly expanded into a higher dimension via a point-wise convolution
2. Spatial filtering is then performed via depth-wise convolutions
3. Finally, the feature map is projected to a lower dimension through another set of point-wise convolutions

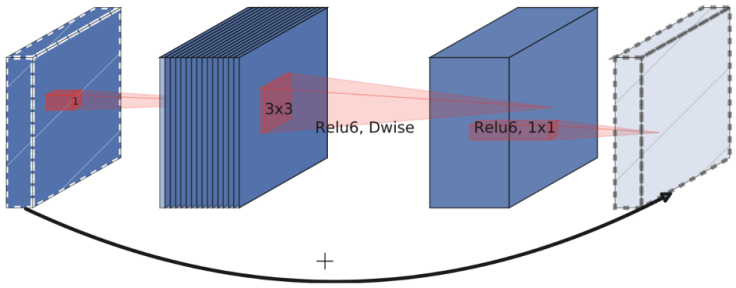


Figure 3.17: Inverted Residual and Linear Bottleneck Layer [79]

As shown in Figure 3.17, a residual connection flows from the input to the output, which serves to improve the ability of gradient propagation across the layers. Classical residual connections connect layers that have a high number of channels whereas an inverted residual connection connects bottlenecks, i.e. the low dimensional feature maps. The argument behind this design choice stems from the notion that bottlenecks contain all necessary information whilst expansion layers solely exist to accompany non-linear transformations. Therefore, the inverted design proposed in MobileNetV2 is much more memory efficient, which is vital for embedded applications [54].

The main idea behind the use of linear bottlenecks is to reduce the size of the operating space, which in turn reduces the number of computations, making the network more efficient. The MobileNetV2 paper [54] highlights that activation layers form a 'manifold of interest' that is assumed to be of lower dimension. In other words, it explains that individual pixels spanning the depth of a feature map are believed to lie in a manifold, where the information encoded can be embedded into a lower-dimensional subspace. The paper also highlights that activation functions such as ReLu support reductions in dimensionality with little information loss. ReLu collapses if the result of the transformation is a non-positive number; however, this collapse occurs in individual channels. If the structure within the activation manifold holds, information may still be preserved through other channels. Therefore, the paper concludes that ReLu is capable of preserving information, given that the input manifold lies in a lower-dimensional subspace of that of the input space [54]. The architecture of MobileNetV2 presented in the original paper [54] is illustrated below in Figure 3.18. As shown, the network is predominately made up of bottleneck blocks.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 3.18: The MobileNetV2 Architecture [54]

3.5 Network Training

Before a deep neural network can be used as a prediction model, it must be trained on a large amount of training data. During training, an input sample is propagated forwards through the network which will then output a specific prediction depending on the network's weight parameters [63]. As data is labelled in supervised learning, the loss between the prediction and the true label can be computed through a loss function. For regression problems, the most common loss used is the mean squared error loss, or MSE, otherwise known as the Euclidean loss. To calculate the MSE, an average is taken across the squared differences between the prediction and the true label. This way, bigger differences result in a larger error value than smaller ones [74].

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.1}$$

where N is the number of training examples, y_i is the ground truth label and $\hat{y}_i$ is the prediction outputted by the network.

To improve the network's performance, the loss is propagated backwards through the network, adjusting the weight parameters accordingly in order to model the input-output relationship better; this process is called backpropagation [74]. Backpropagation aims to minimize the loss function, a task classified as an NP-hard problem [80]. Due to the high levels of non-linearity present in a deep neural network, the landscape of the loss function is most often high-dimensional and non-convex, making it extremely difficult to find the global minimum [81].

3.5.1 Gradient Descent

The minimum of the loss function is found using gradient descent, an iterative optimization algorithm that moves along the direction that brings the steepest decline in the loss function [81]. Gradient descent updates the weights via the following update rule:

$$\theta^{+} = \theta - l\nabla_{\theta}\mathcal{J}(\theta) \tag{3.2}$$

where θ represents the network's weight parameters, l is the learning rate and $\mathcal{J}(\theta)$ is the loss function.

The learning rate l is a hyper-parameter that controls the amount the weights are adjusted with respect to the gradient of the loss function [82]. If high, the network will learn quickly, but at the risk of overshoot and missing potential optima. If low, the algorithm will move towards the optimum more steadily, as the gradient is calculated more frequently. This may allow the model to learn more optimal weights; however, convergence will be much slower [74]. The loss function $\mathcal{J}(\theta)$ is the mean loss function, calculated on all training samples:

$$\mathcal{J}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{J}(\theta, x_i, y_i) \tag{3.3}$$

where N is the number of training examples, θ represents the network's weight parameters, x_i is an input and y_i is the corresponding label.

This type of gradient descent is known as batch gradient descent, whereby every sample in the training data set is analysed before the network parameters are updated. This algorithm, however, becomes intractable when the training data set is very large as it requires loading all data samples into memory and calculates a mean loss function over all training samples before each update [82]. The main flaw of batch gradient descent is that it easily gets stuck in local minima and at saddle points, such as those illustrated in Figure 3.19. Convergence to the global optimum is therefore not guaranteed for non-convex error surfaces [81].

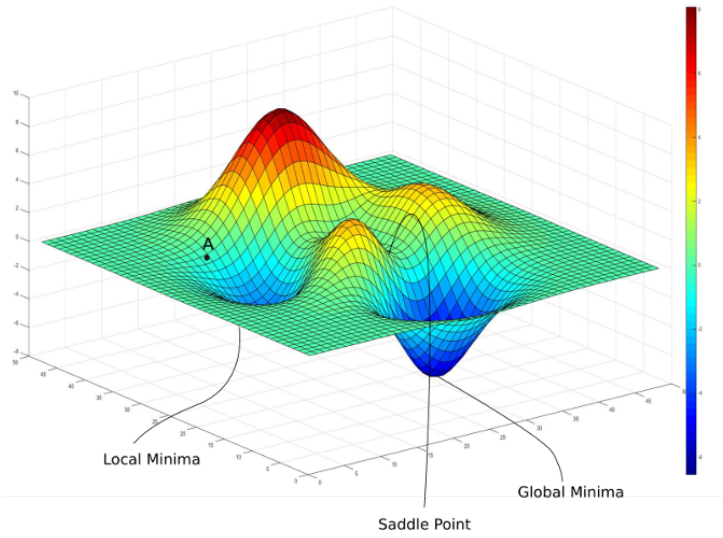


Figure 3.19: Example of a Loss Function's Landscape [81]

To overcome these drawbacks, the stochastic gradient descent algorithm is often utilised instead [81]. Rather than computing the gradient using the whole data set, stochastic gradient descent updates the weights incrementally after each training sample, which are selected at random [82]:

$$\theta^{+} = \theta - l\nabla_{\theta}\mathcal{J}(\theta, x_i, y_i) \tag{3.4}$$

where θ represents the network's weight parameters, l is the learning rate and $\mathcal{J}(\theta, x_i, y_i)$ is the loss function over a single training example.

The path to convergence using stochastic gradient descent fluctuates heavily due to the frequency and randomness of the weight updates. This can cause the algorithm to overshoot more easily, often complicating convergence. However, this fluctuation is also advantageous as it allows the algorithm to jump out of shallow local minima [82]. Furthermore, stochastic gradient descent has been proven to be much faster than batch gradient descent by performing one update at a time and avoiding redundancy. Batch gradient descent computes gradients for all training examples and thus performs many redundant computations when similar examples exist within the data set [83].

To compromise between batch gradient descent and stochastic gradient descent, the mini-batch gradient descent algorithm can be used [82]. It performs updates on mini-batches of training examples as follows:

$$\theta^{+} = \theta - l\nabla_{\theta}\mathcal{J}(\theta, x_{i:i+n}, y_{i:i+n}) \tag{3.5}$$

where n is the number of training examples in the batch, θ represents the network's weight parameters, l is the learning rate and $\mathcal{J}(\theta, x_{i:i+n}, y_{i:i+n})$ is the mean loss function across a batch of training examples.

Each iteration of training, an epoch, consists of viewing each data sample in the training set once. When using mini-batch gradient descent, the training set is split randomly into batches of a predefined size for every epoch. The number of epochs is set as a hyper-parameter and if too low, may prevent the model from fully converging. As the update frequency is now less, the variance is reduced, leading to a more stable convergence. Like batch gradient descent, the error is accumulated across mini-batches of training examples; however, the randomness of batch selection still enables the ability to escape from local minima [82].

One problem with updating weights via backpropagation is the assumption that weights in other layers are fixed whilst the weights in one layer are updated. However, this is not the case here as weights are updated simultaneously. This slows down training as the initialisation of parameters must be chosen carefully and the learning rate must be set to a low value to enable stable updates. To speed up training, batch normalisation layers were introduced by Ioffe and Szegedy [84] in order to help coordinate the update of parameters across different layers. Batch normalisation normalises activations within the network by subtracting the output by the batch mean and then dividing it by the batch standard deviations, as shown in Figure 3.20. Applying normalisation ensures the gradients are more predictive as the optimisation function becomes more smooth. As a result, a high learning rate can be selected which drastically increases the speed of training time [84].

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 3.20: The Batch Normalisation Algorithm, taken from the Original Paper [84]

3.5.2 Generalisation Error Reduction

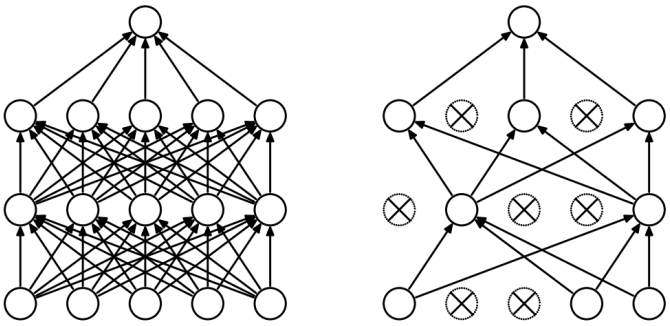
In order to assess how well a model generalises, the data set is often split into three subsets: training, validation and test. The network is trained on the training set, viewing all training samples once per epoch and updating the parameters accordingly via backpropagation. After each training epoch, the model is run with the current parameter values on the validation set, a subset of data that was not seen during training. This provides an unbiased evaluation of the model, which is commonly used to tune the model's hyper-parameters.

The validation set is also used to observe how well the model is generalising on unseen data. For instance, a clear sign that the model is overfitting the training set is given when the validation loss is much higher than the training loss. Finally, once the model has converged after a certain number of epochs, the test set is used to evaluate its final performance, often against competing models [66]. Although some use the validation set as the test set this is not good practice; the test set is often selected to include a greater amount of diverse examples in order to gather a better insight into the generalisation ability of the trained model [85].

The main goal of deep learning is to build a highly accurate prediction model [67]. To achieve this, a very large data set is required to expose the network to as much diverse data as possible during training [20]. Given too few examples, the network is likely to quickly overfit the training data; a common occurrence due to the difficulty of obtaining extremely large labelled data sets [63]. Moreover, as briefly discussed in Section 3.5.1, the model's hyper-parameters contribute significantly to how well the network will learn and thus generalise. Common hyper-parameters include the number of layers, size of feature maps, learning rate, batch size and number of epochs. Given a limited amount of data, the following techniques and hyper-parameter tunings can be implemented to improve the generalisation error of a deep neural network.

Dropout Regularisation

One approach to reducing overfitting is to train different model architectures on the same data set and average the predictions from each model. Yet this is rarely done as it is very computationally expensive and time-consuming to train multiple types of deep neural networks. Instead, a popular regularisation technique called dropout is often adopted to achieve similar results. Dropout was introduced by Srivastava et al. [86] and involves randomly disregarding, or 'dropping', neurons across layers during training time, as shown in Figure 3.21. Doing so allows the network to approximate a large number of different neural network architectures, as the configuration of each layer changes each time the weights are updated. Using dropout thus reduces overfitting as it forces the network to learn more robust features as co-dependency between neurons is reduced [86].



(a) Standard Neural Net (b) After applying dropout.

Figure 3.21: Example of Random Dropout in a Feed-Forward Neural Network [86]

Data Augmentation

A network's learning ability is highly dependent on the number of examples it is exposed to throughout training time [20]. If the training set is either small or consists of many similar examples, the network is likely to quickly overfit the training data. A common technique to reduce overfitting is to implement data augmentation, which artificially increases the amount of diversity within the training data set. The increased data diversity improves the network's ability to learn the underlying features in the data, rather than random patterns found in the training data set [20]. Figure 3.22 illustrates some possibilities of image augmentation, which most often include rotations, cropping and colour alterations.

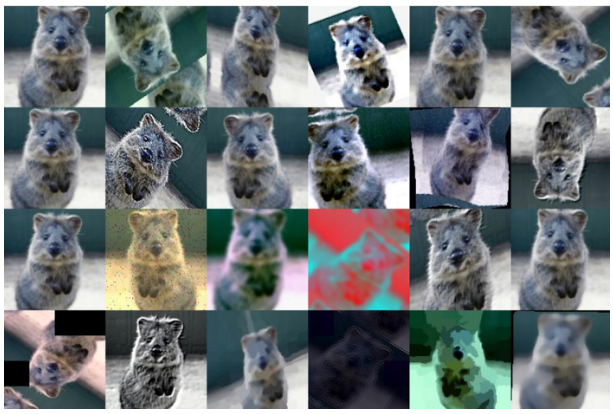


Figure 3.22: Examples of Possible Image Augmentations [87]

Learning Rates and Optimisers

As described in Section 3.5.1, the learning rate is a key parameter that affects the performance of gradient descent. A low learning rate results in smaller weight updates and will provide a slow, yet more stable convergence towards the optimum. Larger learning rates, on the other hand, result in a faster convergence in addition to increased noise on the gradients, which in turn can have beneficial regularisation effects and reduce overfitting [88]. Instead of manually selecting a predefined learning rate, it is possible to utilise an optimiser that applies an adaptive learning rate. The optimisers used throughout this work are the Nadam and RMSProp optimisers, both of which are capable of converging faster than standard gradient descent algorithms discussed in Section 3.5.1. Unlike the latter, they set a high initial learning rate which is gradually reduced throughout training. The learning rate is automatically adjusted in response to the performance of the model. Adaptive learning rates allow the network to learn quickly at first and then take smaller steps towards the local minimum, resulting in a faster and more stable convergence. The main difference between Nadam and RMSProp is that Nadam utilises momentum [88]. Momentum adds a time element to the parameter updates and stores the gradient calculations for use in the next update. These calculations are used to control how much the weights are updated in the next iteration. Overall, the use of such optimisers can improve the performance of gradient descent and thus improve the network's ability to learn [89].

Batch Size

The selected batch size is an important hyper-parameter when training the network using mini batch gradient descent. It controls the accuracy of the estimated error gradient which in turn controls the weight updates; therefore, if batch size is chosen wisely, the performance of the network will improve [90]. A batch size of 32, for example, means 32 samples within the training data will be used at one time to estimate the error of the gradient. Small batch sizes can be beneficial as they offer a regularisation effect through the introduction of noise in the learning process [91]. On the other hand, small batches must often be paired with a lower learning rate in order to maintain stability due to a high variance in the gradient estimation [67]. Larger batches therefore allow for faster and more stable convergences. Batch size can be found via trial and error by evaluating the model on the validation data set.

The Number of Epochs

A key hyper-parameter that affects the performance of the prediction model is the number of epochs. The number of epochs determines how long the network is trained for and therefore must be set high enough to ensure full convergence. However, setting the number of epochs too high may lead to overfitting of the training data. To avoid this, early stopping can be implemented in order to find the optimal number of epochs via the use of callbacks. A callback monitors the validation performance of the model per epoch and saves the best performing model as the final model. This way, even if the number of epochs is set too high, early stopping will only save the results from the epoch that resulted in the model with the lowest generalisation error [92].

3.6 Hand Pose Estimation

Hand pose estimation aims to model the pose of a human hand by estimating the locations of hand joints in images or videos. Although there is no global agreement on a set number of joints, the 21 joints illustrated in Figure 3.23 is the most popular model and is thus used throughout this work [1].

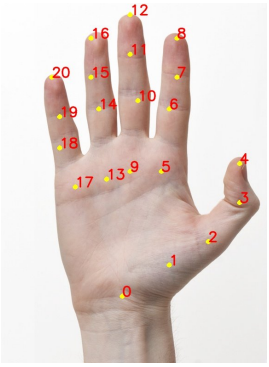


Figure 3.23: 21 Joint Locations of the Hand [93]

The most successful hand pose estimation solutions utilise convolutional neural networks, which have introduced new levels of accuracy and speed [7]. CNNs provide a desirable solution due to their ability of achieving both spatial generalisation and end-to-end differentiability. The former attribute allows the network to perform feature extraction independent of their location within an image, as described earlier in Section 3.3, whereas the latter allows for smooth end-to-end training via backpropagation. Popular CNN hand pose solutions, however, often sacrifice one of these properties for the other [72].

3.6.1 Traditional Solutions

The first type of solution involves adding a fully-connected layer to the end of a CNN architecture in order to perform direct regression between input images and output numerical joint coordinates. Although this method results in an end-to-end trainable network, the weights of the final layer will be highly dependent on the spatial distribution of the inputs during training. To understand this further, imagine the training data set consists entirely of joint coordinates located within the left-hand half of the image. During training, many of the weight parameters in the final fully-connected layer, which represent the right-hand half of the image, will not be trained properly. This leads to a model that will not be able to perform hand pose estimation well when a hand appears in the right-hand half of the image, even though the convolutional part of the network remains spatially invariant [72].

The second type of solution overcomes this problem by performing regression via heatmap matching [72]. A heatmap is a 2D feature map that is naturally outputted by a CNN and provides a loose representation of the location of a joint within the image space, as shown in Figure 3.24. They can be viewed as a 2D Gaussian distribution whereby the probability of a feature occurring in a particular spatial location is represented via pixel intensity [39].

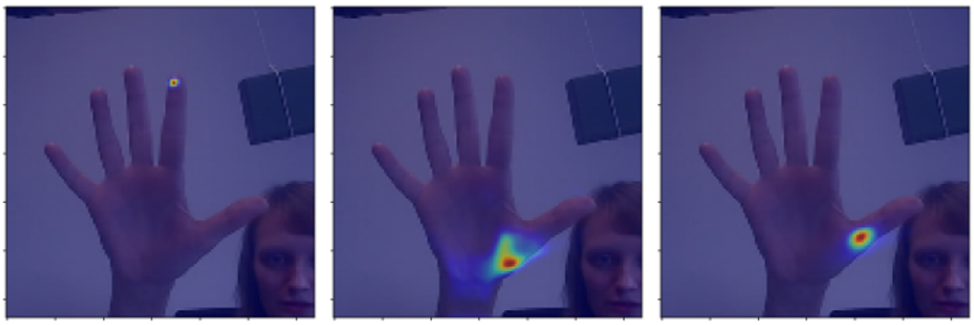


Figure 3.24: Heatmaps Representing the Likelihood of Joint Locations [94]

Training a CNN via heatmap matching is achieved by calculating the loss between predicted and ground truth heatmaps, which are generated from numerical coordinates. This training approach reduces the learning capacity of the network, and as translation invariance is preserved, improves the generalisation error. However, heatmap matching disconnects the loss function from the true desirable metric: the distance between the predicted and the ground truth coordinates. To retrieve numerical joint coordinates from pre-

dicted heatmaps, it is common to compute the argmax of the heatmap's pixel values, which extracts the location of the brightest pixel. As the argmax is a non-differentiable operation, it cannot be used in training an end-to-end network and thus is often only employed during inference. However, this operation is not entirely desirable as it often leads to quantisation errors. This is due to the fact that the coordinate precision is tied to the heatmap's resolution. In other words, all pixel values, not just the brightest, contribute to the loss and choosing the brightest pixel as the final joint location will often be inaccurate [72].

3.6.2 The DSNT Layer

Due to the drawbacks of using fully-connected layers or heatmap matching to perform regression in pose estimation, Nibali et al. [72] introduced an alternative method. They proposed the use of a differentiable spatial to numerical transform (DSNT) layer, which can be used as the final layer of a pose prediction model. Not only does this solution preserves both spatial generalisation and end-to-end differentiability, but it also introduces no additional parameters and outperforms the accuracy of the other two traditional approaches. By preserving end-to-end differentiability, the heatmaps in a CNN are learned implicitly and learn to evolve to produce more accurate coordinate predictions [72].

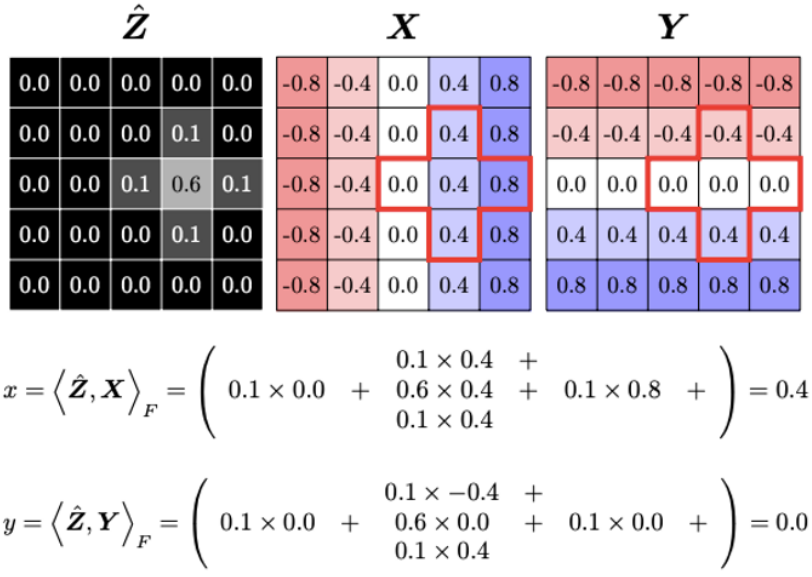


Figure 3.25: Example of Computations in a DSNT Layer [72]

A DSNT layer receives a normalised heatmap as an input, referred to as $\hat{\boldsymbol{Z}}$ in Figure 3.25. The heatmap is represented as a $m \times n$ matrix and is converted into a coordinate map via the following transformations [72]:

$$X_{i,j} = \frac{2j - (n+1)}{n} \qquad Y_{i,j} = \frac{2j - (m+1)}{m} \tag{3.6}$$

where each entry of the X and Y maps is an individual coordinate of x and y respectively and each coordinate is scaled such that the corners of the map lie at $(-1, -1), (-1, 1), (1, -1)$ and $(1, 1)$.

The key difference between the DSNT layer and heatmap matching is that predicted coordinates are analogous to the mean of the coordinates, rather than the mode. Unlike the mode, the mean is differentiable and thus allows for end-to-end training. The DSNT layer calculates the mean via the Frobenius inner product, as demonstrated in Figure 3.25. This forces the network to learn heatmaps that are symmetrical around the prediction location as symmetrical off-centre values cancel one another out. Furth to this, during training, the network will automatically learn activation thresholds in order to prevent outliers being present in the heatmap, as it punishes heatmaps with low confidence outliers [72].

Finally, the loss utilised in the paper [72] is the Euclidean distance between the ground truth and the predicted coordinates:

$$\mathcal{L}_{euc}(\mu, p) = ||p - \mu||_2 \tag{3.7}$$

where p is represents the ground truth coordinates and μ represents the predicted coordinates.

4 Methodology

This chapter presents the methodology behind the proposed solution to 3D hand pose estimation as well as the data sets and evaluation metrics used to assess its performance. Section 4.1 provides an overview of the proposed 3D hand pose estimation solution and the reasons behind its overall design. Section 4.2 describes the data sets utilised including the preprocessing and augmentations applied to them before and during training. Section 4.3 discusses the techniques followed in order to perform model selection. Finally, Section 4.4 presents the loss and evaluation metrics used throughout training, model selection and testing.

4.1 Overview of Proposed Method

As presented in Section 1.3, the motivation behind this thesis is twofold. Firstly, this work aims to propose a novel deep learning solution to the 3D hand pose estimation problem, specifically for use in XR applications using monochrome cameras. To accomplish real-time use on an XR headset, a focus was placed on designing an efficient and lightweight solution. The second focus of this work stems from the lack of published research surrounding the use of monochrome cameras in hand pose estimation, which are commonly found in XR headsets. Therefore, in addition to providing an efficient solution, an investigation into how colour loss may impact the performance of a hand pose prediction model was conducted.

Following findings realised in the literature review, it is believed that an end-to-end 3D hand pose prediction network would require a highly complicated and heavy configuration in order to achieve good generalisation. This stems from the fact that the direct mapping between image features and 3D joint coordinates is highly non-linear, particularly from monocular images. To learn this mapping, solutions often utilise a multi-camera setup [8] [46] [47]

or depth-sensing technology [48], which is not relevant for this work. Furthermore, training such a configuration would require a large amount of 3D labelled data, which is difficult to obtain and as highlighted later in Section 4.2, was not provided for this work. Otherwise, it was found that deep learning solutions have been widely successful in both 2D pose estimation and in 2D-to-3D coordinate lifting (see Chapter 2). Therefore, in order to achieve an accurate and lightweight solution to 3D hand pose estimation, a decision was made to split the overall model into two separate neural networks. This work proposes a novel 2D joint prediction convolutional neural network, that infers 2D joint coordinates from images, in addition to a depth estimation feed-forward neural network, that infers depth coordinates from 2D joint coordinates. Although these two sub-networks must be trained separately, they can be concatenated together during inference time in order to predict real-time 3D hand joint coordinates directly from an image, as shown in Figure 4.1. Not only does this design choice result in an overall lightweight model, but it also enables modular use, which may be beneficial given certain application uses or restrictions surrounding the type of available labelled training data.

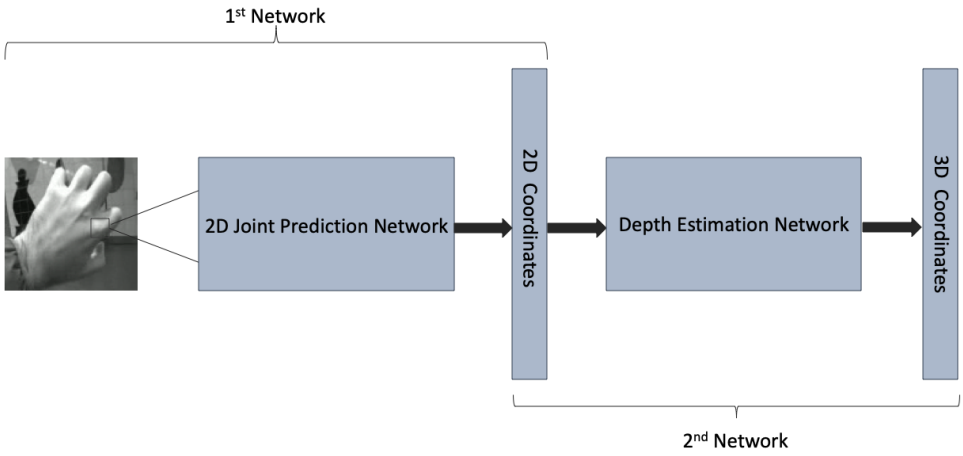


Figure 4.1: Overview of the Proposed Solution for 3D Hand Joint Estimation

Finally, in order to conduct an investigation surrounding how colour loss may impact the performance of a hand pose prediction model, the 2D joint prediction network was trained on both monochrome and RGB images. This provides a direct comparison and new insights regarding how the use or lack of colour may benefit or hinder the performance of the proposed network.

4.2 Data

4.2.1 Provided Data Sets

This thesis was proposed by ManoMotion AB and all data sets featured throughout this report were provided by them. No suitable single data set was provided to train both proposed sub-networks; one that contains images, 2D labels and 3D labels. Instead, three different data sets were provided to serve separate purposes:

- The Vive data set: to train and evaluate the 2D joint prediction model and provide a first insight into 2D hand pose estimation using monochrome cameras typically found in XR headsets
- The RGB data set: to investigate how colour loss impacts the performance of a hand pose prediction model
- The 3D data set: to train and evaluated the depth estimation network

Further details surrounding each data set will be presented in this section. Each data set consists of hands captured in three different poses: grab, half grab and open, as illustrated in the Figures 4.2 and 4.3.

The Vive Data Set

The Vive data set contains 40,792 images of hands recorded from a Vive VR headset and does not contain any 2D or 3D ground truth joint coordinates. The headset is equipped with two monochrome fish-eye cameras, rendering all images distorted and colourless, as shown in Figure 4.2. Monochrome cameras achieve greater resolution and sensitivity to low-light compared to RGB cameras (see Chapter 2) and thus the recorded images are of very high-quality.

Figure 4.2: Examples of Images in the Vive Data Set

The initial objective for this work was to train both of the proposed sub-networks on the Vive data set, in order to build a 3D hand joint prediction model for the Vive headset. However, this was not possible due to the absence of 2D and 3D ground truth labels. An attempt to label the data set was therefore pursued. Manually labelling the data set would have been extremely time consuming and could not be considered for the time-scale of this project. Instead, the software package DeepLabCut [95] was utilised to employ automatic labelling. Given a small number of manually labelled frames, this tool can label the remaining frames in a data set via the use of a powerful deep neural network. The initial idea was to use this tool to label 2D joint locations, which could then be lifted to 3D through triangulation computations; this could be achieved as the Vive data set contains stereo images. However, the performance of the tool is highly sensitive to changes in hand appearances and backgrounds. As the Vive data set contains many different hands and backgrounds, the tool failed to produce highly accurate 2D labels which in turn led to unsatisfactory triangulation results. Retrieving 3D labels in this manner was therefore abandoned. Overall, the Vive data set and the generated DeepLabCut [95] 2D coordinates labels were used to train and evaluate the proposed 2D joint prediction network.

The RGB Data Set

The RGB data set contains 143,927 images of hands taken by a RGB camera on a mobile phone. In addition to images, the data set also includes the corresponding 2D ground truth joint coordinates. This data set is much larger than the Vive data set and includes a greater variety of hands and backgrounds as a result; however, similar to the Vive data set, the 2D ground truth labels are not highly accurate. As the images were captured by a standard RGB camera on a mobile phone, they are coloured, of lower quality and do not contain fish-eye distortion, as illustrated in Figure 4.3.

Figure 4.3: Examples of Images in the RGB Data Set

Part of this work aims to analyse the effects of using monochrome cameras over RGB cameras in hand pose estimation. Although monochrome cameras achieve greater image quality, the impact of a loss of colour information remains unclear. To conduct a proper investigation, two data sets consisting of identical images would be required: one captured from a monochrome camera, and the other from a RGB camera. The performance of the 2D joint prediction network could then be evaluated on both data sets, resulting in a direct comparison. However, obtaining a large amount of identical data recorded from two different cameras is not easy and was not provided in this work. Instead, it was decided that an analysis be performed solely on the effects of colour loss, rather than comparing cameras. This was achieved through the use of the RGB data set, as coloured images can easily be converted to monochrome images. The 2D joint prediction network was therefore separately trained and evaluated on both coloured and monochrome versions of the RGB data set.

The 3D Data Set

Finally, the 3D data set contains 29,561 vectors of 3D hand joint coordinates recorded using RGB cameras and depth sensors. The depth of the joints are not relative to the camera but to the corresponding wrist coordinate, which is not included in the data set but always lies at the origin (before standardisation, see Section 4.2.2). The labelling of the joints in the figures below follows the numbering shown in Figure 3.23.

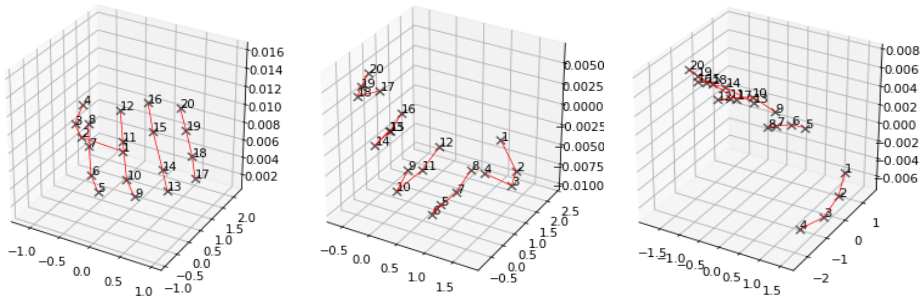


Figure 4.4: Examples of Coordinates in the 3D Data Set

The 3D data set was provided to train the proposed depth estimation network, which infers depth coordinates, $\boldsymbol{z}$, from 2D joint coordinates, $\boldsymbol{x}$ and $\boldsymbol{y}$. The data set does not contain any images and thus could not be used to train the 2D joint prediction network.

4.2.2 Data Preprocessing

Although not explored in this work, it is assumed that a hand detector is used in conjunction with the 2D joint prediction network. Hand detectors are commonly used to crop the image around a detected hand, enabling the removal of irrelevant background which in turn improves the network's learning ability and thus its predictive performance [7]. Therefore, before feeding the 2D data sets (the Vive and RGB data sets) into the proposed 2D joint prediction network, the images and labels were cropped around the hand to mimic the use of a hand detector. Cropping does not apply to the 3D data set, as it does not contain any images. Instead, the 3D data set underwent standardisation before being used to train the depth estimation network, in order to eliminate the effects of scaling and translation of the 3D hands. The standardisation method performed stems from the work of Zhao et al. [10] and is calculated for each coordinate as:

$$\hat{x}_{i,j} = \frac{x_{i,j} - \bar{\boldsymbol{x}}_{\boldsymbol{i}}}{(\sigma(\boldsymbol{x}_i) + \sigma(\boldsymbol{y}_{\boldsymbol{i}}))/2} \tag{4.1}$$

$$\hat{y}_{i,j} = \frac{y_{i,j} - \bar{\boldsymbol{y}}_{\boldsymbol{i}}}{(\sigma(\boldsymbol{x}_i) + \sigma(\boldsymbol{y}_{\boldsymbol{i}}))/2} \tag{4.2}$$

$$\hat{z}_{i,j} = \frac{z_{i,j} - \bar{\boldsymbol{z}}_{\boldsymbol{i}}}{(\sigma(\boldsymbol{x}_i) + \sigma(\boldsymbol{y}_{\boldsymbol{i}}))/2} \tag{4.3}$$

where $\bar{\boldsymbol{x}}_{\boldsymbol{i}}$, $\bar{\boldsymbol{y}}_{\boldsymbol{i}}$ and $\bar{\boldsymbol{z}}_{\boldsymbol{i}}$, are the mean values and $\sigma(\boldsymbol{x}_i)$, $\sigma(\boldsymbol{y}_i)$ and $\sigma(\boldsymbol{z}_i)$ are the standard deviation of the elements in $\hat{\boldsymbol{x}}_i$, $\hat{\boldsymbol{y}}_i$ and $\hat{\boldsymbol{z}}_i$ respectively. The same equations are applied to the x and y coordinates during inference time.

Prior to training, each data set was split into a training set, validation set and test set in the ratio of 80:10:10 in order to provide a way to evaluate the performance of the networks. As discussed in Section 3.5.2, data augmentation reduces a model's generalisation error by introducing greater diversity into the data set. In the 2D prediction model, data augmentation was performed via rotations, whereby images and their respective labels were randomly rotated between 0^o and 90^o. Random rotation was applied constantly during training in order to diversify each training batch. The validation set, on the other hand, was rotated prior to training in order to ensure that every version of the model was evaluated on the same validation data. No rotation augmentation was applied to the 3D data set as the hands were already recorded from a variety of angles.

4.3 Model Selection

The proposed sub-networks were designed with efficiency in mind, which can be measured via the number of parameters present within the networks. Extensive experiments were conducted in order to select the best performing architectures; those that achieve great efficiency without sacrificing predictive performance. These experiments focused on the tuning of different hyper-parameters which as discussed in Section 3.5.2, can heavily influence the generalisation ability of a deep neural network. Firstly, the number of layers were altered in order to find the optimal trade-off between the network's learning ability and the number of parameters. Secondly, the resolution of heatmaps found within the 2D joint prediction network was experimented with in order to improve the accuracy of the conversion to output numerical coordinates. Finally, key hyper-parameters such as the batch size, learning rate and the number of epochs were selected to ensure that both networks converged fully and in a quick and stable manner. Experiments surrounding the learning rate were performed by implementing different optimisers, such as Nadam and RMSProp (see Section 3.5.2). To select the optimal number of epochs, early stopping was implemented such that the model that achieved the best validation performance was the model saved at the end of training (see Section 3.5.2). Other than the number of epochs, all hyper-parameter tuning was performed via grid search and the final model selections were obtained based on their performances on the relevant validation data sets.

4.4 Loss and Evaluation Metrics

The loss used to train both the 2D joint prediction model and the depth estimation model was the mean squared error (MSE) loss. As discussed in Section 3.5, the MSE loss is the most common loss used in regression problems and is calculated as follows:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2 \tag{4.4}$$

where N is the number of training examples, $\boldsymbol{y}_i$ represents a vector of either 2D or depth ground truth coordinates and $\hat{\boldsymbol{y}}_i$ is a vector of the predicted 2D or depth coordinates outputted by the network.

The 2D joint prediction model was assessed quantitatively on the MSE loss whereas the depth estimation model was assessed on the average Procrustes distance (following the work of Zhao et al. [10]). The average Procrustes distance is calculated as:

$$AverageProcrustes = \frac{1}{N}\sqrt{(z_1 - \hat{z}_1)^2 + ... + (z_N - \hat{z}_N)^2} \tag{4.5}$$

where N is the number of joints, z is the ground truth depth coordinate and $\hat{z}$ is the predicted depth coordinate. The main advantage of this metric is that it makes the error measure invariant to scale.

5 Results

The following chapter presents the findings acquired throughout this piece of research. Section 5.1 presents the design process behind the proposed 2D joint prediction network. Section 5.2 reviews the quantitative and qualitative performance of the proposed 2D joint prediction network on the Vive and RGB data sets. Section 5.3 presents the design choices and the final architecture of the proposed depth estimation network. Finally, Section 5.4 reviews the depth estimation network's performance on the 3D data set.

5.1 Proposed 2D Joint Prediction Network

The proposed 2D joint prediction network was designed such that it can be used on an embedded device, such as an XR headset. A focus was thus placed on its efficiency, which was quantified by the number of parameters within the network. Following the success of convolutional neural networks in image feature extraction, a number of efficient CNN architectures were considered throughout the design process. As described in Section 3.3, CNNs achieve high levels of efficiency in addition to spatial generalisation through intrinsic weight sharing and feature map down-sampling. However, since the advent of CNNs, researchers have found traditional convolution computations to contain redundancies and have proposed alternative architectures that perform convolution more efficiently [49] [50] [51] [52] [53]. These architectures were primarily designed for use on embedded devices and have achieved state-of-the-art performances with much fewer parameters compared to traditional CNNs (please refer to Chapter 2 for more details). Yet none of these architectures have outperformed the more recent MobileNetV2 solution proposed by Howard et al. [54]. MobileNetV2 extends upon the depth-wise separable convolutions introduced in the original paper [50] through the use of inverted residuals and linear bottleneck layers (see Section 3.4). Due to MobileNetV2's unparalleled success, its architecture was chosen to form the basis of the 2D

joint prediction network. Moreover, as discussed in Section 3.6, in order for a CNN to learn the direct mapping between an input image and output numerical joint coordinates, the best approach is to use a DSNT layer [72] as the final layer of the network. Compared to classical methods, such as regression via fully-connected layers or heatmap matching, the use of a DSNT layer enables end-to-end training whilst preserving spatial generalisation. Furthermore, in addition to outperforming classical methods, the DSNT layer introduces no additional parameters and thus does not hinder the efficiency of the design. Due to these advantages, the DSNT layer was assigned as the last layer of the proposed network architecture.

The complete MobileNetV2 network can be loaded in Python using the `keras.applications.MobileNetV2` package. The package provides two alternatives regarding the initialisation of the network's weight parameters. The initial weights can either be randomly generated or pre-trained weights (trained on the ImageNet data set) can be loaded. Using pre-trained weights as the initial weights of an untrained network is known as transfer learning. Transfer learning is beneficial as the data set used to train the pre-trained weights is likely to share similar low-level features with the new data set, and thus similar weight parameters in the low-levels of the network [96]. This therefore allows the network to train more easily, which in turn improves predictive performance; it was found that the use of the pre-trained weights improved the validation loss of the final proposed network by 28% compared to the randomly initialised weights. Furthermore, the predefined MobileNetV2 network only accepts inputs consisting of three channels. As monochrome images consist of one channel, they must be transformed into three channels before being fed into the network. To accomplish this, the `cv2.imread()` function provided by OpenCV was utilised, which reads in any type of image in a three-channel format. The DSNT layer does not exist within any pre-made Python packages and was therefore custom-built following the design presented in the paper [72]. The aim of the DSNT layer is to convert the CNN's output heatmaps into a vector of numerical values representing the x and y coordinates of each predicted joint. As this work aims to infer 21 joints, 21 heatmaps must be fed into the DSNT layer. To convert MobileNetV2's output heatmaps into 21 heatmaps, a standard convolutional layer was used. Once this novel design was trained and was found to be able to learn, experiments surrounding a variety of design choices were conducted in order to find the optimal trade-off between accuracy and speed. As discussed in Section 4.3, experiments were approached in a grid search manner.

It was discovered that the heatmaps outputted by MobileNetV2 were of low resolution (in the final design, the heatmaps were 8×8 in size); if too low, the accuracy of the conversion from heatmaps to numerical coordinates in the DSNT layer will be reduced. It was eventually found that up-sampling the heatmaps to a resolution of 32×32 produced the best results on the validation set. Up-sampling was achieved through the use of transposed convolutions, a layer that can be viewed as the inverse of a standard convolutional layer. Unlike other up-sampling techniques, which require a predefined interpolation method, transposed convolutions have learnable parameters that enable the network to learn to up-sample optimally [97]. Otherwise, as discussed in Section 3.5.1, batch normalisation is often used to simplify and speed up training as it smooths out the optimisation function and improves the ability to perform backpropagation. Batch normalisation layers were therefore added after each convolutional and transposed convolutional layer. Once these main design choices were made, techniques to improve the network's performance and reduce overfitting were implemented. As described in Section 3.1, overfitting occurs when the network is too complex for the provided data set.

In order to improve the performance of the network, a number of hyper-parameters were tuned. Firstly, to reduce the complexity of the network, and thus reduce overfitting, a number of layers in MobileNetV2 were removed. MobileNetV2 is made up of blocks of bottleneck stages (see Section 3.4) and therefore the simplest way to prune it is by removing blocks from the bottom of the architecture. Pruning the last 3 blocks (39 layers) resulted in the greatest accuracy versus speed trade-off. Secondly, the learning rate and batch size hyper-parameters were carefully selected as they play an important role in the performance of gradient descent and thus the network's ability to converge well. An adaptive learning rate was implemented, via the use of the Nadam optimiser (see Section 3.5.2). The optimal values for the learning rate and batch size were found to be 0.0001 and 64 respectively, where the selected learning rate is the initial value used by the Nadam optimiser. Finally, dropout regularisation was implemented throughout the network's up-sampling layers as it reduced the network's generalisation error even further. As described in Section 3.5.2, dropout improves a neural network's ability to learn the true underlying patterns between the input and output by reducing the co-dependency between the neurons in each layer. The optimal dropout rate was eventually found to be 0.3. A comparison between different architectures and hyper-parameter choices will be provided later in Section 5.2.

The architecture of the final 2D joint prediction network is presented below in Table 5.1. The key motivation behind the creation of this network was to build a fast and lightweight novel design, without hindering accuracy. Speed was achieved as the final network design contains 606,865 parameters (accuracy will be quantified in the following section). For comparison, the original MobileNetV2 contains around 2.23 million parameters and the MobileNetV2-like solution presented by Gouidis et al. [55] contains 7.89 million, highlighting just how efficient the proposed 2D joint estimation network is.

Layer	Output Size	Filter Size	Stride	Zero Padding	Activation
Input Image	128x128x3	-	-	-	-
MobileNetV2 (-39)	8x8x96	-	-	-	-
Conv2DTranspose	16x16x72	2x2	2	same	ReLu
BatchNormalization	16x16x72	-	-	-	-
Dropout (0.3)	16x16x72	-	-	-	-
Conv2DTranspose	32x32x64	2x2	2	same	ReLu
BatchNormalization	32x32x64	-	-	-	-
Dropout (0.3)	32x32x64	-	-	-	-
Conv2D	32x32x21	1x1	1	same	ReLu
BatchNormalization	32x32x21	-	-	-	-
DSNT	1x42	-	-	-	-

Table 5.1: The Proposed 2D Joint Prediction Network Architecture

5.2 2D Joint Inference Results

In order to evaluate the performance of the proposed 2D joint prediction network, three data sets were utilised. Firstly, the network was trained and tested on the Vive data (see Section 4.2.1) in order to provide a first insight into 2D hand pose estimation using monochrome cameras typically found in XR headsets. The network was then trained separately on two versions of the RGB data set (see Section 4.2.1): a coloured version and a monochrome version. This was done to provide a direct comparison regarding how colour loss may impact the performance of a hand pose prediction model. The following section presents the quantitative and qualitative results retrieved from training the proposed networks on these three data sets. A discussion regarding the results presented in this section will be provided later in Chapter 6.

5.2.1 Quantitative Results

The 2D joint estimation network was designed with the intention for use on XR headsets, which utilise high-quality fish-eye monochrome cameras. It is noteworthy that no public data set was found to contain images of hands captured by such cameras. Additionally, no published research to date performs 2D hand pose estimation on such data. Therefore, the proposed 2D joint estimation network could not be benchmarked against other existing works. Instead, the network was benchmarked against variants of its design to provide an insight into how it achieves a trade-off between accuracy and speed. Table 5.2 illustrates a small number of configurations that were compared during the design process, where the speed of each design is represented by the architecture's number of parameters and the accuracy by the validation loss. Note that the configurations illustrated are part of the same network design and only differ on the basis of hyper-parameter selection, hence why they are compared on the validation loss and not the test loss. In order to find the best performing hand pose prediction model for use in XR headsets, hyper-parameter tuning was performed through a grid search and the models were trained and evaluated on the Vive data set. The Vive data set is the only data set provided in this work that contains images recorded from a VR headset, and thus a high-quality fish-eye monochrome camera.

Model	# of Pruned Blocks	Learning Rate	Batch Size	Dropout Rate	# of Parameters	Validation Loss
0	0	0.0001	64	-	2,654,865	0.00577
1	2	0.0001	64	-	1,115,025	0.00597
2	3	0.0001	64	-	606,865	0.00615
3	4	0.0001	64	-	486,097	0.00630
4	0	0.0001	64	0.3	2,654,865	0.00538
5	2	0.0001	64	0.3	1,115,025	0.00580
6	3	0.0001	64	0.3	606,865	0.00474
7	4	0.0001	64	0.3	486,097	0.00593
8	3	0.0001	128	0.3	606,865	0.00512
9	3	0.0001	32	0.3	606,865	0.00483
10	3	0.001	128	0.3	606,865	0.00488
11	3	0.0001	64	0.5	606,865	0.00564
12	3	0.0001	64	0.1	606,865	0.00615

Table 5.2: Model Comparisons

As highlighted in Table 5.2, the sixth model configuration was found to achieve the best results in terms of both speed and accuracy. This model was therefore chosen to serve as the final proposed 2D joint estimation network, as presented previously in Section 5.1.

The Vive Data Set

Figure 5.1 illustrates the loss graph obtained after the proposed network was trained on the Vive data set. As shown, the network managed to converge in a quick and stable manner. The loss graph is a key indicator of whether a neural network is overfitting or underfitting. As discussed in Section 3.5.2, if the validation loss is much higher than the training loss, the network is overfitting the training data. Conversely, the network is likely to be underfitting if the validation loss is much lower than the training loss. As the validation and training losses remained very similar throughout training, there are no signs that the network overfit or underfit the Vive data set.

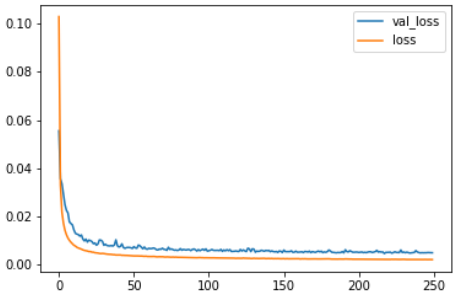


Figure 5.1: MSE Losses per Training Epoch

Furthermore, the loss on the test data set was very similar to the validation loss (and thus the training loss), as shown in Table 5.3. This suggests that the network managed to generalise very well indeed.

Validation Loss	Test Loss
0.00474	0.00458

Table 5.3: Validation Loss versus Test Loss

The RGB Data Set

In order to provide a direct analysis regarding the impact of colour loss in hand pose estimation, the network was trained on both monochrome and coloured version of the RGB data set. Figure 5.2 illustrates the corresponding loss graphs in order to provide a direct comparison. As shown, the monochrome data produced a slightly higher validation loss, indicating a higher generalisation error.

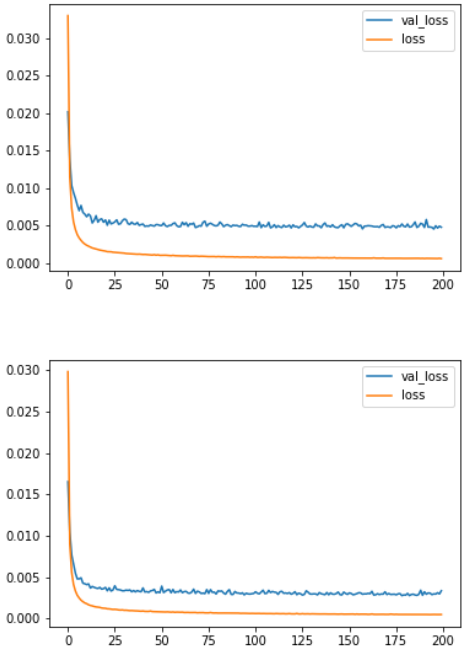


Figure 5.2: Monochrome (top) and Coloured (bottom) MSE Losses per Training Epoch

The network trained on monochrome data additionally resulted in a higher test loss compared to the network trained on coloured data, as shown in Table 5.4.

Validation Loss	Test Loss
0.00451	0.0049

Validation Loss	Test Loss
0.00274	0.0031

Table 5.4: Monochrome (left) and Coloured (right) Validation Loss versus Test Loss

5.2.2 Qualitative Results

In order to visualise how well the network performed, the ground truth (red and black) and predicted joints (blue and green) were compared on a variety of test images across the data sets. Note how the ground truth joint coordinates in all data sets are themselves not very accurate.

The Vive Data Set

Considering the imperfect ground truth labels, the model achieved good visual generalisation overall. The following images in Figure 5.3 provide examples of cases where the network performed well across the three gestures.

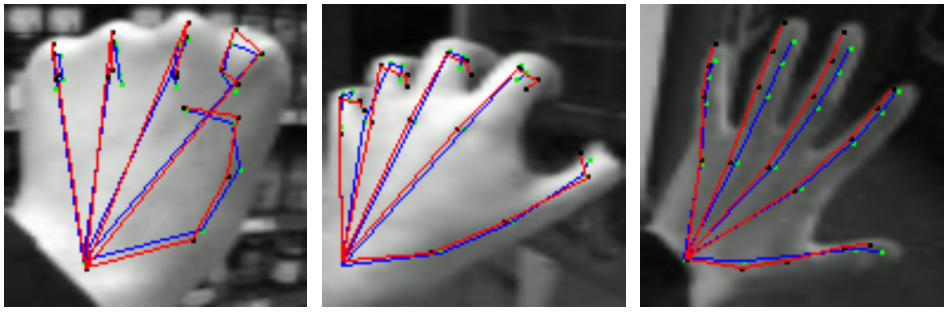


Figure 5.3: Examples of Good Predictions

However, a common failure was noticeably the lack of ability to detect edges of the knuckles and fingertips, as shown below in Figure 5.4.

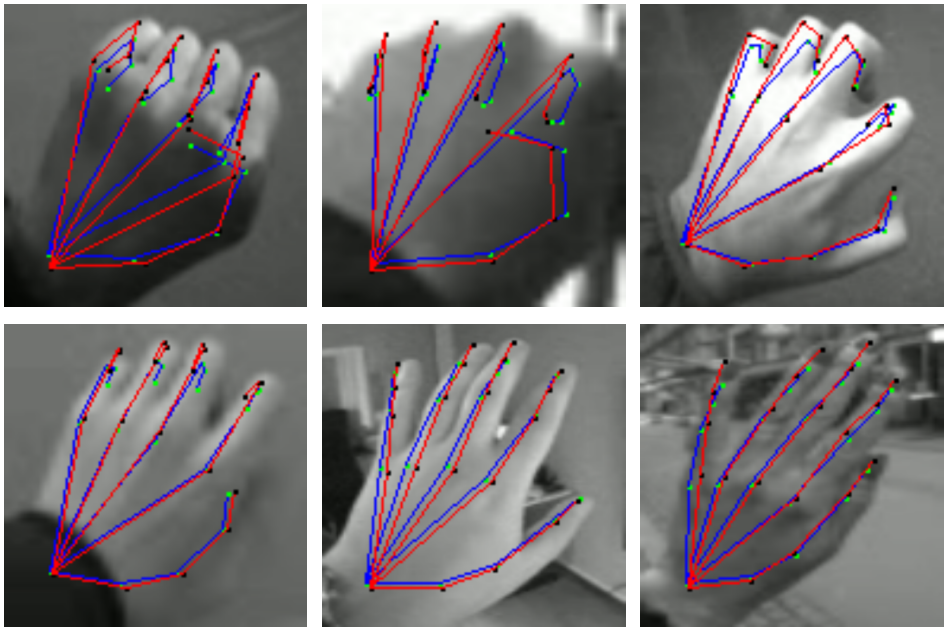
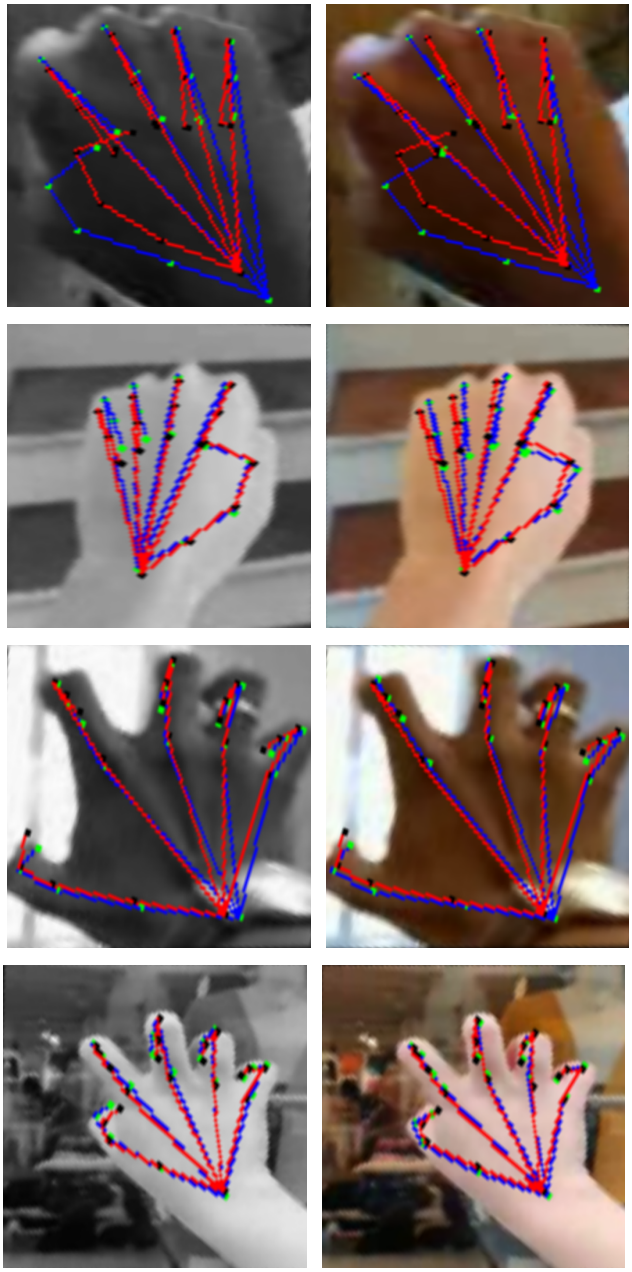


Figure 5.4: Examples of Inaccurate Predictions

The RGB Data Set

In order to compare the performance of the network when trained on coloured versus monochrome data, images within the RGB test set were directly compared. Overall, the vast majority of predictions were similar across all gestures, as illustrated in the following figures.



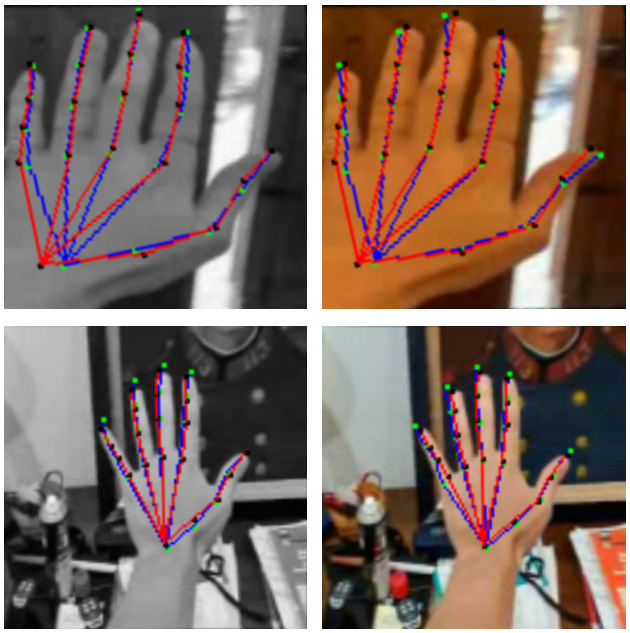


Figure 5.5: Examples of Similar Predictions

One noticeable difference, however, was how the network trained on coloured images produced more volatile predictions on the fingertips, as shown in Figure 5.6.

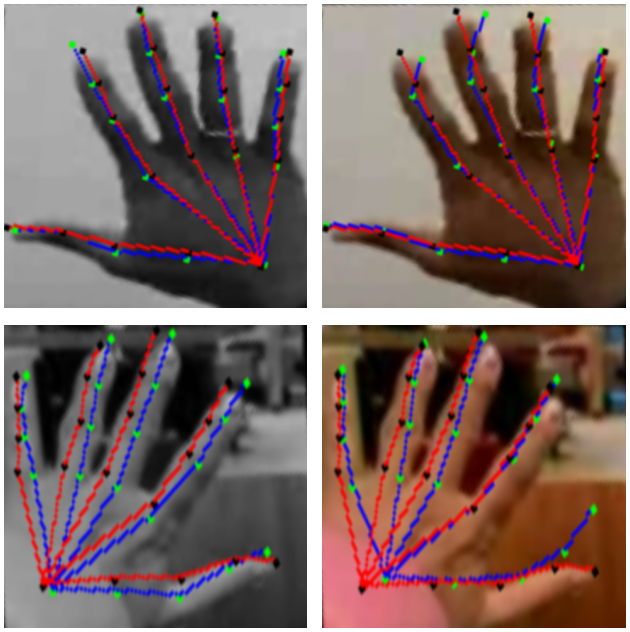


Figure 5.6: Examples of Volatile Fingertip Predictions on Coloured Data

Conversely, the network trained on coloured images was found to outperform the network trained on monochrome images in certain examples. However, these cases were infrequent and no obvious patterns were recognised regarding why they occurred. Examples of such scenarios are shown below in Figure 5.7.

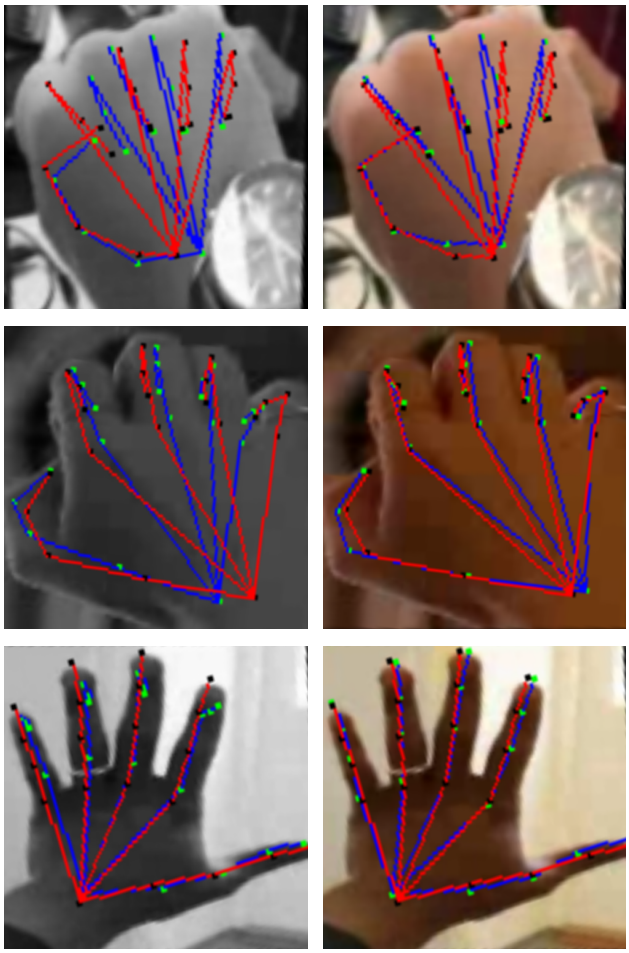


Figure 5.7: Examples of Failed Monochrome Predictions

5.3 Proposed Depth Estimation Network

To provide a full solution to 3D hand pose estimation, this thesis also presents a method to lift 2D hand joint coordinates to 3D. Following the research conducted in the literature review, it was found that 2D-to-3D lifting can be accomplished using model-based, computer vision or deep learning approaches. Unlike the former, deep learning approaches do not rely on any initialisation or temporal information, such as skeletal constraints or bone lengths; all relevant information is automatically encoded in the data they learn from [35]. A deep learning approach was therefore deemed more suitable for use on an XR headset, given the need for real-time inference and a lightweight model. Whilst conducting further research, the solution presented by Zhao et al. [10] particularly stood out due to its extremely simple design; a feed-forward neural network that consists of 6 layers, as shown below in Listing 5.1 where n is the number of 2D landmarks (in this case landmarks refer to hand joints). The network contains a mere 9,020 parameters and achieves 'extremely low' reconstruction errors, as claimed in the research paper [10]. Due to its very lightweight and accurate design, the network is ideal for use in XR headsets and was chosen to form the basis of the proposed depth estimation network. The network uses the RMSProp optimiser, which as described in Section 3.5.2, utilises adaptive learning rates and momentum to optimise the convergence of the network.

```
from keras.models import Model
from keras.layers import Dense, Input

inputs = Input(shape=(2n,))
x = Dense(2n, activation='tanh')(inputs)
x = Dense(2n, activation='tanh')(x)
x = Dense(2n, activation='tanh')(x)
x = Dense(2n, activation='tanh')(x)
x = Dense(2n, activation='tanh')(x)
output = Dense(n, activation='tanh')(x)
```

Listing 5.1: Zhao et al.'s Network Design Implemented in Python

Given a training set with m 3D landmark points, the network proposed by Zhao et al. aims to learn the following function:

$$\hat{\boldsymbol{z}}_i = f(\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{y}}_i) \tag{5.1}$$

where each coordinate is standardised as described in Section 4.2.2.

Although Zhao et al.'s network achieves both speed and accuracy, attempts to extend upon its design were made in order to improve its performance even further. Some initial attempts consisted of changing the number of layers, in addition to implementing regularisation techniques such as dropout. However, neither alterations had a significant effect on the network's performance. It was found through further research that the use of a multi-stage configuration may enhance the network's performance [98]. Multi-stage architectures are commonly found in pose estimation methods, where each stage consists of a simple network that contains up-sampling and down-sampling paths [99]. Some of the most famous multi-stage architectures include the convolutional pose machine [100] and the hourglass network [101]. The solution proposed by Tome et al. [9] was particularly thought-provoking; they utilised a multi-stage CNN to employ iterative refinement of predicted heatmaps. They achieved iterative refinement by feeding both the previous stage's heatmaps as well as the input image into the input of each stage. Inspired by these works, Zhao et al.'s network was transformed into a multi-stage network, whereby each stage was connected to the previous stage's output and to the input through a residual connection. Experiments surrounding the number of stages and layers were conducted as well as the position/rate of dropout regularisation. In the end, the architecture that proved most favourable was a multi-stage configuration presented by Ludlow [98]. Before this configuration is revealed, Table 5.5 provides some examples of the experiments conducted, benchmarked against Zhao et al.'s 6 layer design.

Architecture	**Batch Size**	**Dropout Rate**	**Procrustes Distance (mm)**
Zhao et al.	64	-	0.00187
Zhao et al.	64	0.3	0.00199
7 Layers	32	0.2	0.00172
4 Stages	32	Mix of 0.2 and 0.1	0.00164
5 Stages	32	Mix of 0.2 and 0.1	0.00169
5 Stages	64	Mix of 0.2 and 0.1	0.00152
6 Stages	64	Mix of 0.2 and 0.1	0.00161

Table 5.5: Architecture Comparisons

The final architecture of the proposed depth estimation network is illustrated below in Table 5.6. The fundamental idea behind this design is to use Zhao et al.'s network in the first stage to provide an initial approximation of the depth coordinates, which the following stages then iteratively refine. The network consists of five stages and contains 52,480 parameters.

Stage Num	Layer Num	Layer Name	Output Size	Connection	Activation	Dropout
1	-	Input	40	-	-	-
	1	Dense	40	-	tanh	0.2
	2	Dense	40	-	tanh	-
	3	Dense	40	-	tanh	-
	4	Dense	40	-	tanh	0.2
	5	Dense	40	-	tanh	-
	6	Dense	20	-	tanh	-
2	-	Concatenate	-	1 & 6	-	-
	7	Dense	60	-	tanh	-
	8	Dense	60	-	tanh	0.1
	9	Dense	40	-	tanh	-
	10	Dense	40	-	tanh	-
	11	Dense	20	-	tanh	-
	12	Dense	20	-	tanh	-
3	-	Concatenate	-	1 & 12	-	-
	13	Dense	60	-	tanh	-
	14	Dense	60	-	tanh	0.1
	15	Dense	40	-	tanh	-
	16	Dense	40	-	tanh	-
	17	Dense	20	-	tanh	-
	18	Dense	20	-	tanh	-
4	-	Concatenate	-	1 & 18	-	-
	19	Dense	60	-	tanh	-
	20	Dense	60	-	tanh	0.1
	21	Dense	40	-	tanh	-
	22	Dense	40	-	tanh	-
	23	Dense	20	-	tanh	-
	24	Dense	20	-	tanh	-
5	-	Concatenate	-	1 & 24	-	-
	25	Dense	60	-	tanh	-
	26	Dense	60	-	tanh	0.1
	27	Dense	40	-	tanh	-
	28	Dense	40	-	tanh	-
	29	Dense	20	-	tanh	-
	30	Dense	20	-	tanh	-

Table 5.6: The Proposed Depth Estimation Network Architecture

5.4 Depth Inference Results

The proposed model was trained on the 3D data set (see Section 4.2.1). This section presents the quantitative and qualitative results on the test set.

5.4.1 Quantitative Results

The proposed network was benchmarked against Zhao et al.'s network on the test set. Table 5.7 presents the results of their respective average Procrustes metrics.

Model	Average Procrustes Distance (mm)
Zhao et al. [102]	0.00187
Proposed Depth Estimation Model	0.00152

Table 5.7: Model Comparison

5.4.2 Qualitative Results

To visualise how well the network performed, the ground truth and predicted coordinates were compared. The network generalised very well across the three gestures, as shown in the following figures.

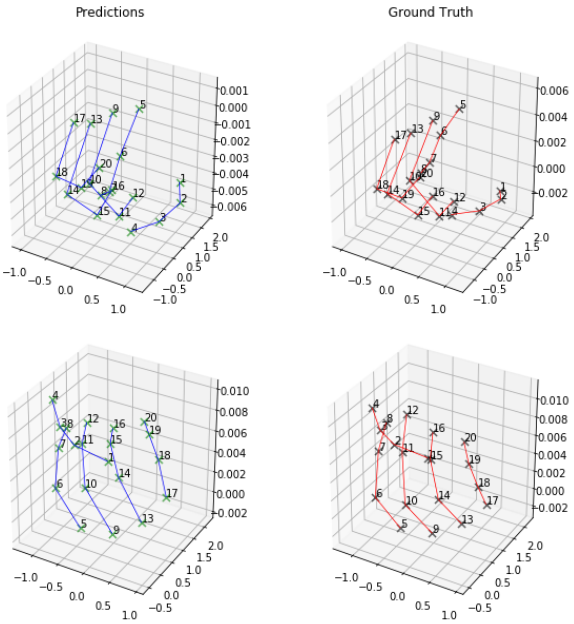


Figure 5.8: 3D Grab Predictions versus Ground Truth

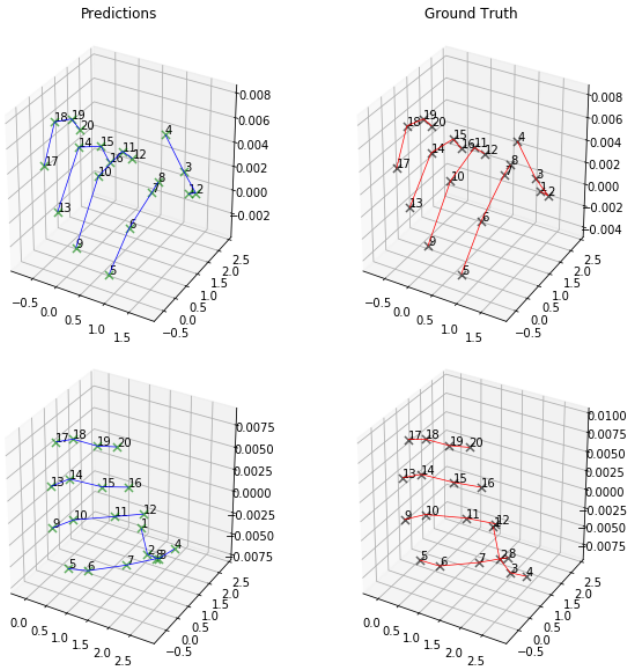


Figure 5.9: 3D Half Grab Predictions versus Ground Truth

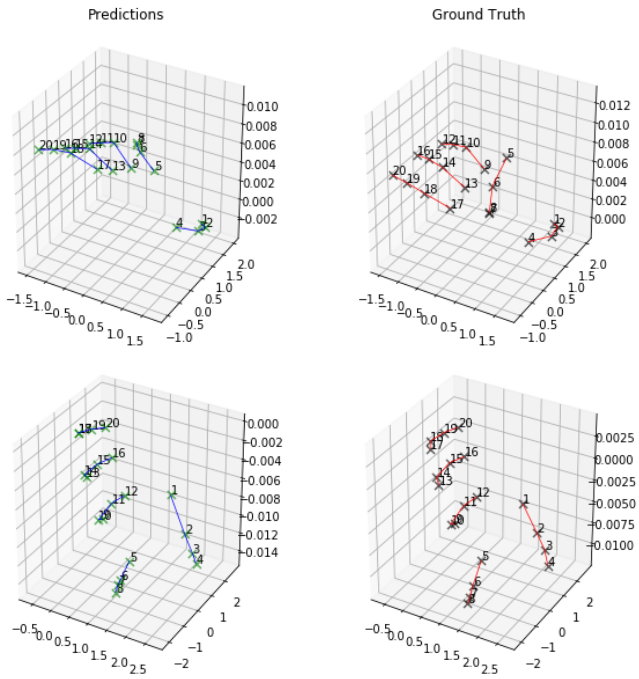


Figure 5.10: 3D Open Predictions versus Ground Truth

The numbering of joints matches those shown in Figure 3.23. Moreover, no wrist joints are present within the 3D data set (see Section 4.2.1). To gain a clearer insight into the quality of inference, the predictions and ground truth coordinates were plotted in 2D. The following figures illustrate selected plots that are representative of the overall results on the test set. As shown, the error range was tightly bounded.

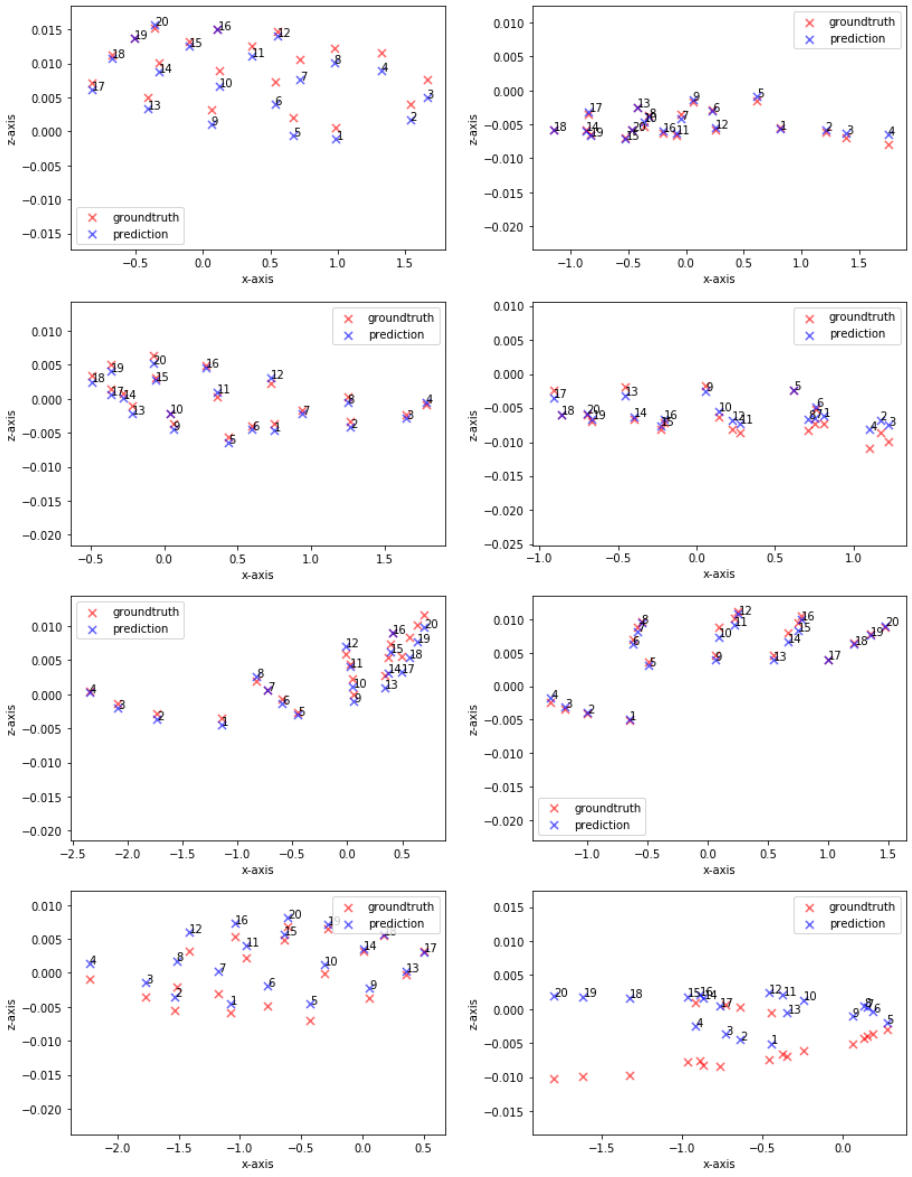


Figure 5.11: Examples of Test Results Plotted in 2D

The 3D data set was also found to contain some noisy ground truth labels; for instance, the top-right plot in Figure 5.10 is an example of a noisy open hand gesture, where the index knuckle is much lower than the other knuckles. The excellent generalisation capability of this network is portrayed in this example, as the prediction shown on the left produced a perfect open hand shape. Other examples where the network recovered acceptable shapes compared to the noisy ground truth labels are illustrated below in Figure 5.11.

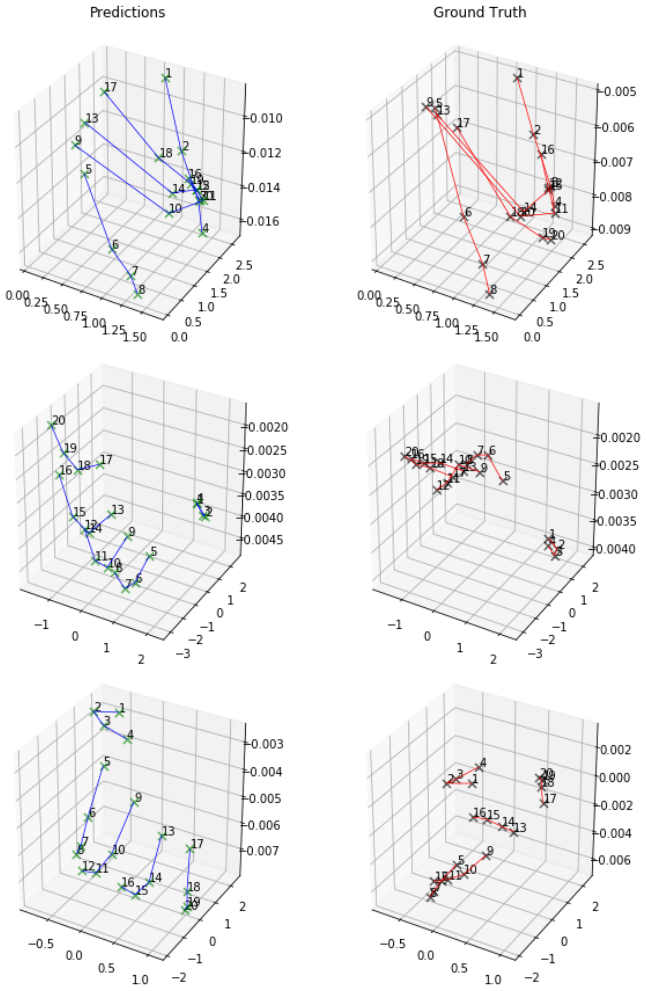


Figure 5.12: Examples of Noisy Ground Truth Labels

6 Discussion

This chapter provides a detailed analysis of the results presented in Chapter 5. Sections 6.1 and 6.2 examine the performance of the proposed 2D joint prediction network and depth estimation network respectively. Section 6.3 provides an analysis surrounding the impacts of colour loss in hand pose estimation in order to provide insights specifically for XR applications seeking to utilise hand tracking through monochrome cameras. Finally, Section 6.4 proposes future work that may follow this piece of research.

6.1 Proposed 2D Joint Prediction Network

A key contribution of this thesis is the proposal of a novel and efficient deep learning solution to 2D hand pose estimation. The proposed design takes advantage of MobileNetV2 [54], a powerful convolutional neural network architecture that achieves state-of-the-art efficiency. Furthermore, through the use of a final DSNT layer [72], the proposed network was trained in a manner that outperforms conventional methods, such as those that perform regression via fully-connected layers or heatmap matching. The use of a DSNT layer is highly desirable in pose estimation as it preserves both spatial generalisation and end-to-end network training without the introduction of additional parameters. The proposed design is believed to be the first to join MobileNetV2 with a DSNT layer in order to solve hand pose estimation. Through the use of efficient convolutional computations and direct coordinate regression, the proposed solution was able to achieve accurate predictions whilst containing a low number of parameters, proving suitable for real-time use in embedded devices such as XR headsets. To analyse how well the network performed, it was trained on three data sets: the Vive data set, the RGB data set and a monochrome version of the RGB data set. The Vive data set was utilised to analyse the network's performance on images recorded from cameras typically found in XR headsets. Given that no known published research has provided such findings in

the past, the predictions on the Vive test data provide a first look into 2D hand pose estimation using high-quality, fish-eye distorted, monochrome images. Another aim of this work was to provide insights into how colour loss may impact the performance of a hand pose estimation model. As the Vive data set solely consists of monochrome images, the effects of colour cannot be directly observed. Instead, the RGB data set was provided, enabling the network to be trained on coloured and monochrome versions of the data set for a direct comparison. An analysis regarding the impact of colour loss will be presented later in Section 6.3.

Following the quantitative and qualitative results presented in Sections 5.2.1 and 5.2.2 respectively, the proposed 2D joint estimation network was found to generalise very well. Although the loss metrics are not very interpretive, the fact that the losses across the training, validation and test data sets were alike and low indicates that the model succeeded in extracting the underlying features between the input images and 2D joint coordinates. Upon visual inspection, the majority of predictions looked very similar to the ground truth coordinates, providing further confidence in the network's ability to learn. However, it was found that the network frequently failed to infer the knuckle and fingertip joints when trained on the Vive data set, placing the predictions slightly below their true locations. This is a clear sign that the network failed to detect joint edges well. It is worth noting that such failures were not observed when the network was trained on the RGB data set (both coloured and monochrome versions). Therefore, to understand why they may have occurred, it is important to analyse the limitations of the Vive data set itself. The Vive data set consists of high-quality images recorded from the Vive headset, which is equipped with stereo fish-eye monochrome cameras. Firstly, the data set contains a total of 40,792 images which is a very small amount of training data, especially for tackling a complex task such as 2D hand pose estimation. On top of this, as the data set was recorded using stereo cameras, half of the images were recorded from the left camera and the others from the right. This introduces a very large amount of redundancy in the training data set, meaning the number of diverse images within the data set is around half its actual size. It is well known in the field of deep learning that the performance of a neural network is not only highly dependent on the amount of training data it observes, but on the realism and diversity of the examples within the data set [20]. The quality of the data is so vital that it is labelled as important as the choice of network architecture [32]. Using a small data set, especially one containing redundancy, is very likely to prevent the network from extracting

the full mapping between the input and output, as the network was not exposed to enough examples during training time. When trained on the RGB data set (both versions), the network did not experience the same phenomenon regarding the failure to detect knuckle and fingertip edges well. As the RGB data set is a much larger and a more diverse data set, the size and lack of diversity of the Vive data set is therefore believed to be the prime reason behind the failures observed.

Another factor believed to have hindered the network's ability to learn was the use of noisy ground truth labels during training. As discussed in Section 4.2.1, the ground truth labels of the Vive data set were labelled using the DeepLabCut software package [95] that failed to produce highly accurate labels. The RGB data set was also found to contain inaccurate ground truth labels, most notably on the fingertips where the "ground truth" label commonly laid above the true location. Overall, it is clear to see throughout Section 5.2.2 that the ground truth joints were imperfect across both data sets. There always lies a balance between the amount of noise in a neural network, whether it is introduced via labels or regularisation techniques. On one hand, too much noise may inhibit the learning of certain features, such as the knuckle edges in the Vive data set, or may cause the learning of undesirable features, such as extended fingertips in the RGB data set. On the other hand, noise can be beneficial in a regularisation sense as it reduces overfitting and in some aspects, adds more diversity into the data. Although not measurable, the noisy labels are believed to have had a negative effect on the learning process of the network, most prominently on the learning of finger joint locations.

Overall, it is difficult to measure just how well the 2D joint estimation network learnt. A key sign that it generalised well was the similarity in losses across the training, validation and test data sets of both the Vive and RGB data sets. This indicates the generalisation error is low and the network was able to learn the underlying features in the data well. In addition to this, it was found through qualitative analysis that the prediction joints were very similar to the ground truth, supporting the claim of good generalisation. However, the network was found to struggle more on the Vive data set than either versions of the RGB data set and is believed to be due to its small size and large amount of redundant data. Furthermore, both data sets contain noisy labels which are believed to have inhibited learning of true hand joint locations. Conclusively, the proposed 2D joint estimation network generalised well in the face of data limitations whilst containing very few parameters.

6.2 Proposed Depth Estimation Network

This work aims to present a solution for 3D hand pose estimation and thus, in addition to proposing a 2D joint prediction model, presents a deep learning solution that performs accurate 2D-to-3D coordinate lifting. Whilst conducting research into the current deep learning solutions, Zhao et al.'s [10] work particularly stood out due to its extremely trivial architecture in addition to its claims of high accuracy. The lightweight architecture is ideal for use in embedded devices due to its low number of parameters. In an attempt to extend upon the network, further research was conducted surrounding techniques to improve its performance. A multi-stage configuration [98] was eventually adopted to perform iterative refinement of the depth coordinate predictions. Trained on the 3D data set, the quantitative and qualitative results were presented in Sections 5.4.1 and 5.4.2 respectively.

The performance of the proposed depth estimation network was quantitatively benchmarked against Zhao et al.'s [10] work. Although the proposed multi-stage design outperformed Zhao et al.'s [10] network, the improvement was rather small. It can therefore be concluded that Zhao et al.'s [10] trivial network performed very well on the 3D data set and implementing a multi-stage configuration did not improve its performance significantly. Nevertheless, it is important to note that the 3D data set is a relatively simple data set, consisting of only three gestures. Therefore, given a more complex data set, multi-stage networks may be worth implementing if Zhao et al.'s [10] network is found to struggle; however, this is speculative and is left for future work. From qualitative findings, the proposed depth estimation network was found to perform very well. No major prediction failures were observed on the test set and the error range was tightly bounded. Although the 3D data set was found to contain some noisy examples, the amount was small and did not have any noticeable effects on the learning process. Overall, the results from the proposed depth estimation network are conclusively of excellent standard.

6.3 Colour Loss in Hand Pose Estimation

Prior research in the field of hand pose estimation to date has focused on the use of either RGB or RGB-D cameras and have thus neglected the use of monochrome cameras, which is believed to be due to a lack of use cases. Nonetheless, monochrome cameras may be worth utilising as they are not only cheaper but achieve higher resolution and greater sensitivity to low light [59]. These attributes suggest that they are capable of producing higher-quality images in both indoor and outdoor environments and are more suitable for mass production in embedded devices compared to RGB and RGB-D cameras. The advent of XR has introduced new uses for monochrome cameras, with many XR headsets on the market currently utilising them for positional tracking. Oculus Quest is the first headset to use them for real-time 3D hand tracking [4]; following the recent release of this feature, it is likely that the demand for hand tracking technologies using monochrome cameras will rise. This work therefore aims to bridge the gap in research by performing hand pose estimation using monochrome data in addition to providing new insights into how colour loss may affect the performance of a hand pose prediction model. In order to investigate the effects of colour loss, the proposed 2D joint prediction model was trained on the RGB data set in addition to a monochrome version of the RGB data set to provide a direct comparison. The two versions of the data set are completely identical, except for colour and random data augmentation that occurs during training. Direct comparisons of the quantitative and qualitative results were presented in Sections 5.2.1 and 5.2.2 respectively.

The performances of the networks were firstly compared quantitatively through their respective loss graphs, in addition to their performances on the test data. The network trained on monochrome images was found to have a higher test loss, in addition to a larger gap between the validation and training losses throughout training time (see Figure 5.2). This is an indication that the use of monochrome imagery led to a larger generalisation error, suggesting the network became more susceptible to overfitting. Overfitting occurs when a network is too complex for a given data set and causes the network to learn patterns in the training data too well, so much so that it learns patterns of random noise rather than the true underlying function. A key sign that a network is overfitting is when the validation loss is much higher than the training loss, as the network fails to generalise well on unseen data. These findings therefore suggest that monochrome data is easier to learn from than coloured data for a hand pose prediction model. On the other hand, as the quantitative differ-

ences between the networks are quite small, it is difficult to state if this theory is true as it is possible that the random data augmentations may have affected the learning processes of networks. Therefore, the true impact of colour loss, regarding the quantitative results, is inconclusive.

The qualitative results, on the other hand, provided a clearer insight into how colour loss affected the learning process of the hand joint prediction model. A key realisation was made regarding how the network trained on monochrome images may have outperformed the network trained on coloured images. As illustrated in Section 5.2.2, the coloured network produced more volatile predictions on the fingertips, causing them to bend in physically impossible ways. This phenomenon was never observed on the monochrome data and is therefore believed to be a result of colour interference. This finding suggests that training on monochrome data resulted in a more robust prediction model, especially against changing backgrounds and lighting conditions. Moreover, the network trained on monochrome images was found to always respect the motion constraints of the human hand, unlike the coloured network which produced random fluctuations during inference time. This suggests that networks trained on monochrome data learn hand pose parameters primarily on shape. However, it is noteworthy that whilst visually comparing the test results of both networks, the vast majority of predictions were found to be highly alike indicating that colour loss did not have a very significant impact on the model's overall performance. Furthermore, some cases were found where the coloured network outperformed the monochrome network; however, this did not occur very often and no obvious indications why they occurred were realised. In conclusion, based on the qualitative findings, colour loss was not found to have a negative effect on the overall performance of the hand pose prediction model. Instead, the use of monochrome images produced a model that proved to not only be more robust to background and lighting interference but managed to learn the intrinsic motion constraints of the human hand better than the network trained on coloured images.

Overall, the use of colour in hand pose estimation adds an extra layer of information for the network to learn from. Although this may prove useful in some scenarios, it intuitively makes sense that colour may hinder the network's performance during inference time due to changing light conditions, shading on parts of the hand or interference from colours in the background. As hands tend to be chromatically uniform in colour, such interferences may be especially harmful to a network that has not been trained on a very large and

diverse data set. From the research conducted in this work, it can be concluded that there is no obvious benefit of using coloured imagery over monochrome imagery in 2D hand pose estimation. The network trained on monochrome images was found to perform just as well, if not more robustly, and thus is deemed more suited for hand pose estimation due to the unconstrained environment it often occurs in. Finally, it is worth noting that a true comparison between monochrome cameras and RGB cameras was not conducted in this work as the monochrome images used in this analysis were converted from images captured by a RGB camera. Therefore, the true impact of using monochrome cameras over RGB cameras in hand pose estimation remains uncertain. However, following the research conducted in this thesis, it is believed that the use of monochrome cameras will not hinder the performance of a hand pose prediction model; due to their superior light efficiency compared to RGB cameras (see Chapter 2), they are more likely to result in better predictive performance.

6.4 Future Work

The following section provides some possibilities for future work that may follow this piece of research. As no known prior published research has presented results on hand pose estimation using monochrome cameras, or an analysis regarding the effects of colour loss, there are many possible extensions that may be pursued.

Data

One key extension of this work involves the obtainment of better training data. Firstly, the limitations surrounding the Vive data set were found to restrict the 2D joint prediction network's learning due to its small amount of diverse training examples and noisy 2D labels. Therefore, possible extensions may involve recording more diverse data or providing a more accurate labelling procedure, both of which will improve the performance of the network greatly. Secondly, 3D ground truth labels for the Vive data set were not obtained. Collecting 3D ground truth labels is most often achieved via the use of depth sensors and/or a multi-camera setup. These methods and technologies were not utilised when the Vive data set was recorded and thus inhibited the creation of a complete 3D hand joint prediction model for the Vive headset. Therefore, another possible improvement would be to record both 2D and 3D ground truth labels from the Vive headset in order to observe real-time 3D pose estimation on an XR headset.

Experiments

Part of this work focused on the effects of colour loss in hand pose estimation, as XR headsets are beginning to utilise monochrome cameras to perform hand tracking. As discussed in Section 6.3, a true comparison between the use of monochrome versus RGB cameras was not performed. Therefore, one possible extension of this work would be to record identical data from both types of cameras to provide a true analysis regarding their impacts on the performance of a hand pose prediction model. Furthermore, cameras found in XR headsets are fish-eye cameras, which result in distortion of the hand, especially as it moves closer to the camera. It may therefore also be worth conducting experiments on the effects of fish-eye distortion in hand pose estimation by training the prediction model on an undistorted version of the Vive data set.

Multi-Camera Model

This work provides a solution to 3D hand pose estimation from a monocular monochrome camera. However, XR headsets are commonly equipped with multiple cameras; for instance, Oculus Quest performs 3D hand tracking through the use of four monochrome cameras, making use of the different viewpoints to achieve accurate 3D hand tracking [4]. Although the Vive data set was recorded using stereo cameras, a multi-view solution was not explored in order to provide a more modular solution; furthermore, the RGB and 3D data sets are monocular data sets and thus a multi-view solution would not have been appropriate. It would therefore be interesting to explore how the proposed model could be used in a multi-camera set-up and whether 3D hand joint predictions from multiple views can be fused together to provide more accurate and robust results. This may prove beneficial especially regarding the inference of occluded joints.

7 Conclusion

Many XR headsets currently rely on monochrome cameras to perform positional tracking, primarily due to their lower cost, higher resolution and greater sensitivity to low-light compared to RGB cameras. These attributes render monochrome cameras more suitable for mass production and with the ability to produce high-quality images in both indoor and outdoor environments, suggesting they may also be more suitable for use in hand tracking applications. Due to a lack of published research and solutions surrounding 3D hand pose estimation using monochrome cameras, the work presented throughout this thesis bridges the gap in research by exploring two research questions.

The first research question examined how deep learning methods could be used to build an accurate 3D hand pose prediction model, suited for real-time use in XR headsets. This work proposed a novel solution consisting of two efficient deep neural networks: the 2D joint prediction network and the depth estimation network. The 2D joint prediction network makes use of MobileNetV2 [54], an efficient convolutional neural network, in addition to a final DSNT layer [72] to perform accurate numerical coordinate regression. The network achieves accurate 2D hand pose estimation whilst containing a very small number of parameters. Trained on monochrome data, this network was used to provide a first insight into hand pose estimation using images taken from monochrome cameras typically found in XR headsets. Its performance, however, was limited by the size and lack of diverse training examples within the utilised data set (the Vive data set). As a result, the network frequently produced inaccurate predictions on knuckles and fingertips. As such inaccuracies were not observed when the network was trained on a larger and more diverse data set (the RGB data set, both coloured and monochrome versions), the limitations surrounding the Vive data set are believed to be the prime reasons behind the network's failures. Furthermore, both the Vive and RGB data sets contain a large amount of noise, regarding their 2D ground truth labels.

This factor is also believed to have resulted in imperfect predictions, particularly on the fingers. Otherwise, the 2D joint prediction network trained in a quick and stable manner and achieved a very low generalisation error, indicating that it managed to learn the input-output mapping very well in the face of data limitations. On the other hand, the depth estimation network was inspired by the work of Zhao et al. [10], who presented an extremely simple and lightweight network that achieves accurate 2D-to-3D coordinate lifting. The trivial network was extended into a multi-stage configuration [98] which provides iterative prediction refinement to further improve the network's accuracy. The depth estimation network was found to achieve excellent predictions on the 3D data set, whilst also containing very few parameters. It was concluded that the multi-stage configuration did not improve the performance of Zhao et al.'s [10] simple network by a significant amount; multi-stage configurations are nonetheless believed to be beneficial when training on a more complex data set, although this is left for future work.

The second research question examined how the loss of colour information within training data affects the performance of a hand pose prediction model. To conduct an investigation, the proposed 2D joint prediction model was trained on both coloured and monochrome versions of the RGB data set in order to gain new insights that have never been presented within the field of hand pose estimation. The quantitative findings acquired indicated that monochrome images are easier to learn hand pose parameters from whereas the qualitative findings indicated that training on monochrome images resulted in a hand pose prediction model that learnt primarily on shape. The network trained on coloured images produced volatile predictions, especially on the fingertips; this suggests that its performance was more sensitive to changing lighting and background conditions. As such volatility was not observed when trained on monochrome data, it was concluded that training the proposed hand pose prediction model on monochrome data resulted in a more robust prediction model overall, one that always respected the motion constraints of the human hand and was not sensitive to colour and background interference. This in turn concludes that XR headsets equipped with monochrome cameras possess an advantage in the context of 3D hand tracking and should utilise them over RGB cameras.

Bibliography

Books

[63] John D. Kelleher, Brian Mac Namee, and Aoife D'Arcy. *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. The MIT Press, 2015.

[66] Michael Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[67] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Papers

[1] Bardia Doosti. *Hand Pose Estimation: A Survey*. June 2019.

[2] Kevin VanHorn, Meyer Zinn, and Murat Cobanoglu. *Deep Learning Development Environment in Virtual Reality*. June 2019.

[3] Onur Guleryuz and Christine Kaeser-Chen. *Fast Lifting for 3D Hand Pose Estimation in AR/VR Applications*. Oct. 2018.

[5] Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. *Hands Deep in Deep Learning for Hand Pose Estimation*. Feb. 2015.

[6] Benjamin Luff et al. *Efficient and precise interactive hand tracking through joint, continuous optimization of pose and correspondences*. July 2016.

[7] Paschalis Panteleris, Iason Oikonomidis, and Antonis A. Argyros. *Using a single RGB frame for real time 3D hand pose estimation in the wild*. Dec. 2017.

[8] Yuncheng Li et al. *End-to-End 3D Hand Pose Estimation from Stereo Cameras*. 2019.

[9] Denis Tome, Chris Russell, and Lourdes Agapito. *Lifting from the Deep: Convolutional 3D Pose Estimation from a Single Image*. July 2017.

[10] Ruiqi Zhao, Yan Wang, and Aleix Martinez. *A Simple, Fast and Highly-Accurate Algorithm to Recover 3D Shape from 2D Landmarks on a Single Image*. Sept. 2016.

[11] Yikang Li et al. *Disentangling Pose from Appearance in Monochrome Hand Images*. Apr. 2019.

[12] Markus Schlattmann and Reinhard Klein. *Simultaneous 4 gestures 6 DOF real-time two-hand tracking without any markers*. Jan. 2007.

[13] Jiawei Zhang et al. *3D Hand Pose Tracking and Estimation Using Stereo Matching*. Oct. 2016.

[14] Markus Oberweger et al. *Efficiently Creating 3D Training Data for Fine Hand Pose Estimation*. June 2016.

[19] R McCloy and Robert Stone. *Science, medicine, and the future: Virtual reality in surgery*. Nov. 2001.

[20] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. Dec. 2017.

[21] Thanh Nguyen et al. *Deep Learning for Deepfakes Creation and Detection*. Sept. 2019.

[22] World Economic Forum Global Future Council on Human Rights. *How to Prevent Discriminatory Outcomes in Machine Learning*. Mar. 2018.

[25] Frederick Aardema, Sophie Cote, and Kieron O'Connor. *Effects of virtual reality on presence and dissociative experience*. Dec. 2006.

[26] Laura Dipietro, A.M. Sabatini, and Paolo Dario. *A Survey of Glove-Based Systems and Their Applications*. Aug. 2008.

[27] David Kim et al. *Digits: Freehand 3D interactions anywhere using a wrist-worn gloveless sensor*. Oct. 2012.

[28] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis Argyros. *Efficient model-based 3D tracking of hand articulations using Kinect*. Jan. 2011.

[29] Alexandros Makris, Nikolaos Kyriazis, and Antonis Argyros. *Hierarchical Particle Filtering for 3D Hand Tracking*. June 2015.

[30] Andrea Tagliasacchi et al. *Robust Articulated-ICP for Real-Time Hand Tracking*. Aug. 2015.

[31] Fuyang Huang et al. *Structure-Aware 3D Hourglass Network for Hand Pose Estimation from Single Depth Image*. Dec. 2018.

[32] James Supancic et al. *Depth-based hand pose estimation: methods, data, and challenges*. Apr. 2015.

[33] Danhang Tang, Tsz-Ho Yu, and Tae-Kyun Kim. *Real-Time Articulated Hand Pose Estimation Using Semi-supervised Transductive Regression Forests*. Dec. 2013.

[34] Xiao Sun et al. *Cascaded hand pose regression*. June 2015.

[35] Liuhao Ge et al. *Robust 3D Hand Pose Estimation in Single Depth Images: from Single-View CNN to Multi-View CNNs*. June 2016.

[36] Pashalis Panteleris and Antonis Argyros. *Back to RGB: 3D Tracking of Hands and Hand-Object Interactions Based on Short-Baseline Stereo*. Oct. 2017.

[37] Chen Qian et al. *Realtime and Robust Hand Tracking from Depth*. June 2014.

[38] Toby Sharp et al. *Accurate, Robust, and Flexible Real-time Hand Tracking*. Apr. 2015.

[39] Jonathan Tompson et al. *Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks*. Aug. 2014.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Jan. 2012.

[41] Christian Zimmermann and Thomas Brox. *Learning to Estimate 3D Hand Pose from Single RGB Images*. May 2017.

[42] Franziska Mueller et al. *GANerated Hands for Real-Time 3D Hand Tracking from Monocular RGB*. June 2018.

[43] Xinghao Chen et al. *Pose Guided Structured Region Ensemble Network for Cascaded Hand Pose Estimation*. Aug. 2017.

[44] Liuhao Ge et al. *3D Convolutional Neural Networks for Efficient and Robust Hand Pose Estimation from Single Depth Images*. July 2017.

[45] Hengkai Guo et al. *Region ensemble network: Improving convolutional network for hand pose estimation*. Sept. 2017.

[46] Haibo Qiu et al. *Cross View Fusion for 3D Human Pose Estimation*. Oct. 2019.

[47] Karim Iskakov et al. *Learnable Triangulation of Human Pose*. Oct. 2019.

[48] Gyeongsik Moon, Ju Chang, and Kyoung Lee. *V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation from a Single Depth Map*. June 2018.

[49] Forrest Iandola et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and textless1MB model size*. Feb. 2016.

[50] Andrew Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Apr. 2017.

[51] Ido Freeman, Lutz Roese-Koerner, and Anton Kummert. *EffNet: An Efficient Structure for Convolutional Neural Networks*. Jan. 2018.

[52] Xiangyu Zhang et al. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. June 2018.

[53] Gao Huang et al. *CondenseNet: An Efficient DenseNet Using Learned Group Convolutions*. June 2018.

[54] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. June 2018.

[55] Filippos Gouidis et al. *Accurate Hand Keypoint Localization on Mobile Devices*. Dec. 2018.

[56] Henning Hamer, Esther Koller-Meier, and Luc Van Gool. *Tracking a Hand Manipulating an Object*. Sept. 2009.

[57] Hui Liang, Junsong Yuan, and Daniel Thalmann. *Resolving Ambiguous Hand Pose Predictions by Exploiting Part Correlations*. July 2015.

[58] Franziska Mueller et al. *Real-Time Hand Tracking under Occlusion from an Egocentric RGB-D Sensor*. Oct. 2017.

[60] Hae-Gon Jeon et al. *Stereo Matching with Color and Monochrome Cameras in Low-Light Conditions*. June 2016.

[61] S. Strygulec et al. *Road Boundary Detection and Tracking using monochrome camera images*. Jan. 2013.

[62] Samuel Scheidegger et al. *Mono-Camera 3D Multi-Object Tracking Using Deep Learning Detections and PMBM Filtering*. June 2018.

[64] Wei Yang et al. *3D Human Pose Estimation in the Wild by Adversarial Learning*. Mar. 2018.

[70] Jin-Cheng Li et al. *Bi-firing deep neural networks*. Feb. 2014.

[72] Aiden Nibali et al. *Numerical Coordinate Regression with Convolutional Neural Networks*. Jan. 2018.

[75] Matthew Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Neural Networks*. Nov. 2013.

[80] Hao Li et al. *Visualizing the Loss Landscape of Neural Nets*. Dec. 2017.

[84] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Feb. 2015.

[86] Nitish Srivastava et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. June 2014.

[88] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. Sept. 2016.

[91] D. Wilson and Tony Martinez. *The general inefficiency of batch training for gradient descent learning*. Jan. 2004.

[94] Umar Iqbal et al. *Hand Pose Estimation via Latent 2.5D Heatmap Regression*. Apr. 2018.

[99] Wenbo Li et al. *Rethinking on Multi-Stage Networks for Human Pose Estimation*. Jan. 2019.

[100] Shih-En Wei et al. *Convolutional Pose Machines*. Jan. 2016.

[101] Alejandro Newell, Kaiyu Yang, and Jia Deng. *Stacked Hourglass Networks for Human Pose Estimation*. Oct. 2016.

[102] Ruiqi Zhao, Yan Wang, and Aleix Martinez. *A Simple, Fast and Highly-Accurate Algorithm to Recover 3D Shape from 2D Landmarks on a Single Image*. Sept. 2016.

Online Resources

[4] Shangchen Han et al. “Using deep neural networks for accurate hand-tracking on Oculus Quest”. In: *Facebook AI Blog* (Sept. 2019). URL: https://ai.facebook.com/blog/hand-tracking-deep-neural-networks/.

[15] Scott Stein. “New details on Qualcomm’s 5G VR prototypes: Cloud rendering, eye tracking, high-res displays”. In: *CNET* (Feb. 2020). URL: https://www.cnet.com/news/qualcomms-new-5g-vrar-headset-tech-looks-like-a-superpowered-oculus-quest/.

[16] “Apple Glasses: VR and AR Are Coming”. In: *MacRumors* (May 2020). URL: https://www.macrumors.com/roundup/apple-glasses/.

[17] Samantha KellerDirector, EnhancedTECHSamantha Keller, and Sam. “Can Smart Glasses Replace the Smartphones?” In: *Enhanced Tech* (May 2020). URL: https://www.enhancedtech.com/blog/can-smart-glasses-replace-the-smartphones/.

[18] Yang LI et al. “Gesture interaction in virtual reality”. In: *Virtual Reality Intelligent Hardware* (Aug. 2019). URL: https://www.sciencedirect.com/science/article/pii/S2096579619300075.

[23] “Virtual Reality, Real Injuries: OSU Study Shows how to Reduce Physical Risk in VR”. In: (Jan. 2020). URL: https://today.oregonstate.edu/news/virtual-reality-real-injuries-osu-study-shows-how-reduce-physical-risk-vr.

[24] Sandee LaMotte. “The very real health dangers of virtual reality”. In: *CNN* (Dec. 2017). URL: https://edition.cnn.com/2017/12/13/health/virtual-reality-vr-dangers-safety/index.html.

[59] “Color vs. Monochrome Sensors”. In: *Red Digital Cinema* (). URL: https://www.red.com/red-101/color-monochrome-camera-sensors.

[65] Anup Bhande. “What is underfitting and overfitting in machine learning and how to deal with it”. In: *Medium* (Mar. 2018). URL: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76.

[68] KTH Royal Institute of Technology. “Introduction to Machine Learning”. In: (2019). University Course.

[69] Stanford University. In: *CS231n Convolutional Neural Networks for Visual Recognition* (). University Course. URL: https://cs231n.github.io/neural-networks-1/.

[71] Stanford University. In: *CS231n Convolutional Neural Networks for Visual Recognition* (). University Course. URL: https://cs231n.github.io/convolutional-networks/.

[73] freeCodeCamp.org. "An intuitive guide to Convolutional Neural Networks". In: *freeCodeCamp.org* (Feb. 2018). Accessed: 24-04-2020. URL: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/.

[74] Adit Deshpande. "A Beginner's Guide To Understanding Convolutional Neural Networks". In: (). URL: https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/.

[76] Jason Brownlee. "A Gentle Introduction to Pooling Layers for Convolutional Neural Networks". In: *Machine Learning Mastery* (July 2019). URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.

[77] Kunlun Bai. "A Comprehensive Introduction to Different Types of Convolutions in Deep Learning". In: *Medium* (Feb. 2019). URL: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215.

[78] Chi-Feng Wang. "A Basic Introduction to Separable Convolutions". In: *Medium* (Aug. 2018). URL: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728.

[79] Luis Gonzales. "A Look at MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *Medium* (Nov. 2019). URL: https://medium.com/@luis_gonzales/a-look-at-mobilenetv2-inverted-residuals-and-linear-bottlenecks-d49f85c12423.

[81] Ayoosh Kathuria. "Intro to optimization in deep learning: Gradient Descent". In: *Paperspace Blog* (Dec. 2019). URL: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/.

[82] "Stochastic Gradient Descent - Mini-batch and more". In: *Adventures in Machine Learning* (Mar. 2017). URL: https://adventuresinmachinelearning.com/stochastic-gradient-descent/.

[83] Imad Dabbura. “Gradient Descent Algorithm and Its Variants”. In: *Medium* (Sept. 2019). URL: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3.

[85] Tarang Shah. “About Train, Validation and Test Sets in Machine Learning”. In: *Medium* (Dec. 2017). URL: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[87] Algorithmia. “Introduction to Dataset Augmentation and Expansion”. In: *Algorithmia Blog* (Feb. 2019). URL: https://algorithmia.com/blog/introduction-to-dataset-augmentation-and-expansion.

[89] Casper Hansen. “Optimizers Explained - Adam, Momentum and Stochastic Gradient Descent”. In: *Machine Learning From Scratch* (Dec. 2019). URL: https://mlfromscratch.com/optimizers-explained/#momentum.

[90] Jason Brownlee. “How to Control the Stability of Training Neural Networks With the Batch Size”. In: *Machine Learning Mastery* (Jan. 2020). URL: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/.

[92] Jason Brownlee. “Use Early Stopping to Halt the Training of Neural Networks At the Right Time”. In: *Machine Learning Mastery* (Oct. 2019). URL: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.

[93] Vikas Gupta. “Hand Keypoint Detection using Deep Learning and OpenCV”. In: *Learn OpenCV* (Oct. 2018). URL: https://www.learnopencv.com/hand-keypoint-detection-using-deep-learning-and-opencv/.

[95] “DeepLabCut: a Software Package for Animal Pose Estimation”. In: *GitHub* (Mar. 2020). URL: https://github.com/DeepLabCut/DeepLabCut.

[96] Sarkar. "A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning". In: *Medium* (Nov. 2018). URL: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

[97] Naoki Shibuya. "Up-sampling with Transposed Convolution". In: *Medium* (Mar. 2019). URL: https://medium.com/activating-robotic-minds/up-sampling-with-transposed-convolution-9ae4f2df52d0.

[98] Richard Ludlow. "3d Human Pose Estimation from 2d Keypoints". In: (June 2018). URL: https://github.com/rludlow/3d-pose-2d-keypoints/blob/present/Ludlow_3d_pose_2d_keypoints.pdf.