3D HAND TRACKING FROM AN RGB CAMERA

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2024

Ryan Karibwami

Supervised by Aphrodite Galata

Department of Computer Science

Contents

Word Count: 10207

List of Tables

List of Figures

Abstract

In this project, I explore methods to recreate a 3D hand-detection pipeline from scratch which only requires images taken from an RGB camera without depth being necessary. This pipeline includes three distinct parts, hand detection, keypoint detection, and pose estimation. Each of these parts utilises machine learning to create models to achieve their varying purposes. The final aim of this project is to create a custom complete pipeline working to a high quality and allow a more accessible route into the area of study for future use.

The final sections are completed to a high quality. The hand detection step achieved a high accuracy and the keypoint detection and pose estimation steps both achieved low errors. When combined into a single pipeline, these sections interact efficiently. The pipeline is capable of running in real-time without a cost to its performance, giving high-quality outputs.

Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank my supervisor Aphrodite for being such a great help during the process of this project (and for the free luxury hot chocolates too). My family for being so understanding when I've been so busy and forgot to call as much. My friends from home for being awesome and fine that I haven't been able to run our D&D game for so long. And finally, my friends I've met this year who've kept me (relatively) sane, especially Rosie and Paulina who started this project alongside me and have become great friends since.

Chapter 1

Introduction

Tracking motion is a difficult task even for humans, becoming even harder when tracking not just the 2-dimensional movement but also the depth for accurate pose detection. Although some have resolved this using a device with a depth sensor, it has been shown possible to detect the depth using machine-learning models. This means that a simple RGB camera, which many people have access to, may be used.

Many individual parts surrounding the task have been created, but a complete pipeline is far rarer. For the pipelines that do exist, many have become outdated, causing them to become difficult to run without massive changes to the code. This has been a core reason for choosing the direction of this project, stemming from the frustration from the inaccessibility of many current complete pipelines. This report details methods used to achieve the pipeline, with hopes of assisting future work in the field.

1.1 Aims

The main aim of this project has been to create a custom pipeline for 3D hand tracking, designed so only a standard camera is required. By removing the need for depth sensing the pipeline can be used on any image or sequence, therefore specialised hardware is not needed.

The key end goals of the pipeline are to:

- Create a system for detecting if a hand is in a frame, and if there is, whether it is left or right.
- Create a system for labelling the keypoints of a hand.
- Create a system for posing a 3D model.

- Join the previously mentioned steps into a single system with efficient interaction.

Each of the above goals should be achieved with a high performance relevant to the step. At a minimum, the pipeline should be able to take a single image, predict the correct hand label, predict keypoints that somewhat resemble the input hand, and finally pose a 3D model that resembles the original hand. The steps should also be reproducible with minimal strain by anyone who chooses to do so.

Chapter 2

Background

3D hand-tracking is an area that has had a lot of research and development. The following section highlights much of the history behind this field. It also describes the different methods used in this project. As well as this, additional information that is needed to fully understand this project is included.

2.1 Previous Work

Before beginning this project, I explored many papers that I felt could be relevant and may help in the development of the pipeline. Here I highlight key papers that were the most relevant to the final design.

Zhang et al. [19] created a powerful system for predicting 2.5D hand poses. This became a large inspiration for my final design, both due to the effective power of the system and having a lot of its materials readily available.

Zimmermann et al. [22] created a large dataset of real-world hands. The dataset contains images of hands from multiple views, with various backgrounds using a green screen. The dataset also has annotations for both hand keypoints and 3D pose. This dataset has been used in each stage of training my dataset due to it having all the required data, making sure the stages can become consistent with each other.

Mueller et al. [12] approached tracking real-time hand pose via an RGB-D camera such that they get a colour image and depth reading. The method was produced with an aim, which they claim to have achieved, of being robust to both self-occlusions (occlusions made by the same hand) and occlusions by objects.

Spurr et al. [16] created a system for optimising 2D keypoint annotations. Their method applies real-world constraints on hands as a loss function (see 2.3.4). Using this system they can limit abnormalities in predicted hands that are not biologically possible

by the standard hand. The proposed constraints are called biomechanical constraints or BMC.

2.2 Pipeline Overview

Before delving further into the background of areas of this project it is important to get a brief overview of the proposed pipeline to be created such that it becomes clearer why each of its steps is important and what they do.

The final pipeline consists of three distinct parts - hand detection, keypoint detection, and pose estimation. Each of the steps utilises machine learning to create models that achieve their specific task. The flow of this pipeline is shown in Figure 2.1.

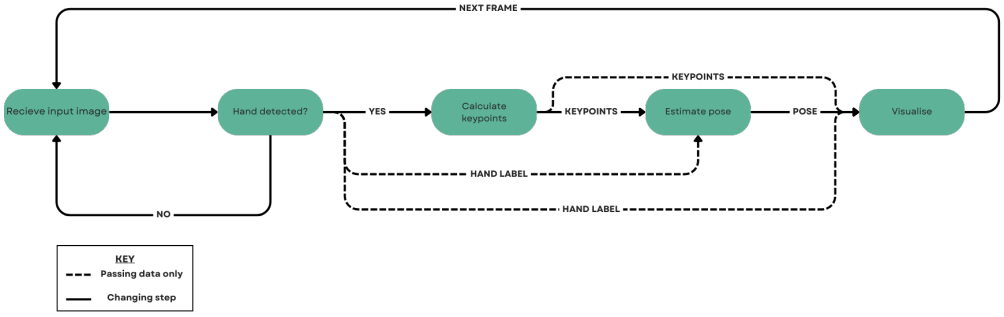


Figure 2.1: A flowchart displaying the design of the pipeline

This first begins with an image that is then fed into the hand detection model. In the case that a hand is detected, the same image is passed onto the keypoint detection model. The results of this step are fed into a pose estimation model, dependent on the label of the original detected hand, such that the pose is mapped to the corresponding left or right-hand model.

The following sections give a brief overview of each part, an explanation of their main purpose, and key problems to overcome.

2.2.1 Hand Detection

This stage was included for two main purposes - to ensure a hand is in a frame and to distinguish left and right hands.

The former is necessary to ensure that tracking is not conducted on frames where a hand is undetected - which in this project are referred to as “null frames”. By skipping null frames, unnecessary processing is decreased, increasing the efficiency, this also prevents the pipeline from giving erroneous keypoint and pose predictions based on some non-existent hand.

The latter is necessary for the final section of the pipeline. For the keypoint labelling section, handedness does not need to be known, as a single model can be trained to reliably predict keypoints. However, for pose estimation, a separate model is needed for both right and left hands, this is due to them using different 3D models. Therefore, to be able to use the pipeline on both right and left hands, the handedness should be predicted for any given frame.

The classification needs to be capable of differentiating right and left hands to a high degree. This can prove to be a difficult task due to these not being distinct objects, but instead in the most basic form being mirror reflections of each other.

The step should be able to detect hands in a range of contexts. This means that it should be capable of detecting hands in a diverse number of situations such as different environments, positions in the frame, being partly occluded etc.

2.2.2 Keypoint Labelling

This step is the major backbone of the system, predicting the landmarks of the hand.

21 landmarks are used, as described in Figure 2.2. The use of 21 landmarks replicates 15 of the natural joints in a human hand, as well as the wrist and each of the 5 fingertips. This allows for complex articulation of any given hand.

This step takes a given image and matches the 21 keypoints to points in the image corresponding to x and y coordinates. These values give 21 2D coordinates which can be joined together to make a hand representation. These coordinates can also be used to display the representation onto the input image.

Additionally, the z coordinate is calculated which gives a predicted depth value. This allows for a 3D representation of the hand to be created. This can then therefore be used in the next step of pose estimation.

This step needs to overcome the problem of occlusions. Any given hand may be occluded by an object or by the hand itself. The step should therefore be capable of

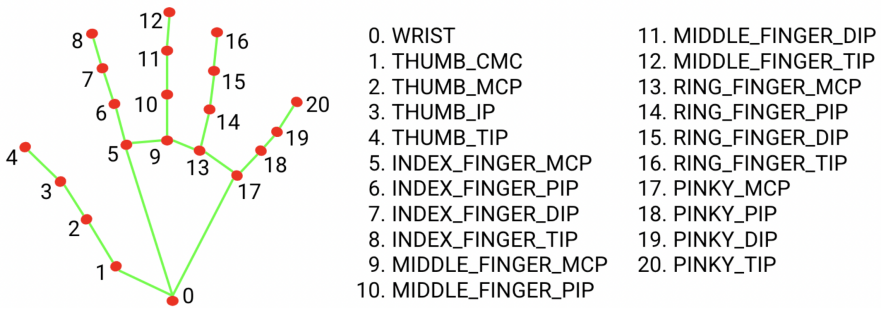


Figure 2.2: Described hand landmarks [1]

overcoming this and accurately predict the correct keypoints in occluded areas of the hand to achieve a realistic output.

As with the hand detection step, it should also adapt to labelling keypoints on both right and left hands. However, this should not be as hard a problem since in this stage they are being treated equally.

Also like the hand detection step, this step should adapt to a wide range of situations to become more generalised.

2.2.3 Pose Estimation

This is the final step of the pipeline, which uses the keypoints predicted in the previous step to predict the pose of a 3D model.

The MANO hand model [14] was chosen as the 3D model. This is a popular hand model that has been used in a wide variety of projects.

This step predicts both the global rotation of the hand as well as the rotations at 15 predefined joints in the MANO model. These can then be applied to the MANO model to create a posed hand.

Each of the rotations is defined as three angles across each axis, as defined in Figure 2.3, locally to the point of rotation. In the case of the joint rotations these represent:

- Flexion (controls bend): rotation around the local x-axis.
- Rotation: rotation around the y-axis.
- Abduction (controls spread): rotation around the z-axis.

How these angles interact, is what gives a final posed hand.

The final pose estimation needs to be able to pose a variety of hand poses. It should be adept at posing simple hands, such as an open-palmed hand, as well as more complex hands, such as hands with multiple bends in the fingers.

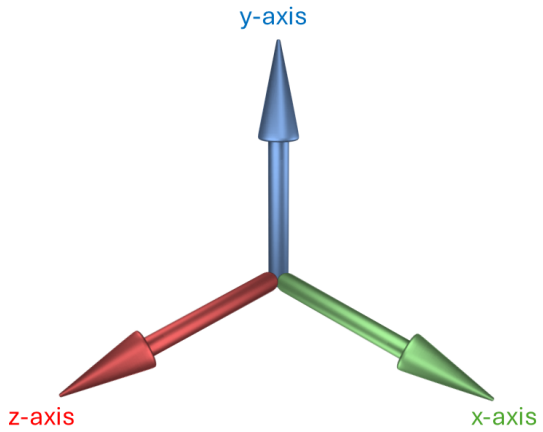


Figure 2.3: Labelled 3D Axes

2.3 Neural Networks

As mentioned earlier, all three sections of the pipeline use machine learning to achieve their respective goals. The specific type of machine learning used in all three is neural networks. A neural network is a method that trains a computer to learn in a way resembling a human brain, hence the "neural" part of the name.

2.3.1 Convolutional Neural Networks

A convolutional neural network or CNN is a type of neural network that performs best with image, speech or audio inputs. They consist of three types of layers:

- Convolution layers
- Pooling layers
- Fully connected layers

2.3.1.1 Convolution Layers

These are the core building blocks of a CNN. The layer uses a filter or kernel, which is a matrix of weights, to move across the input image. An example of this can be seen in Figure 2.4.

The process begins by sliding the kernel over the width and height of the image, going over the total image multiple times. At each position, an elementwise multiplication operation is applied between each element of the kernel and the corresponding elements at said location. This produces an output called a feature map.

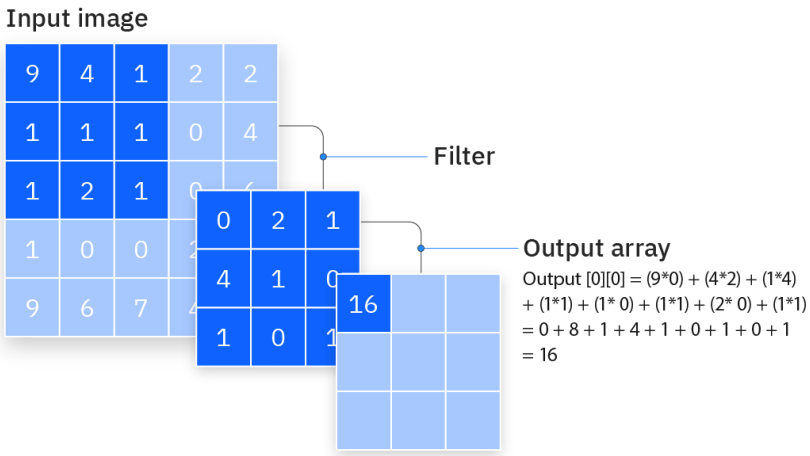


Figure 2.4: Example of a Convolution Layer [2]

2.3.1.2 Pooling Layers

Pooling layers are used to reduce dimensionality, reducing the number of parameters in the input. Similar to convolution layers, pooling operations sweep a filter across the input, but in this case without weights.

There are two main types of pooling:

- **Max pooling**: As the filter moves across the input, it selects the pixel with the maximum value.
- **Average pooling**: As the filter moves across the input, it calculates the average value in the area.

2.3.1.3 Fully Connected Layers

In the previous layers mentioned, the pixels of the input image are not directly connected to the output layer. However, in fully connected layers, each node in the output connects directly to a node in the previous layer.

2.3.2 Classification vs Regression

For CNNs, there are two main types of tasks - classification and regression tasks. A classification task is one in which the model predicts a discrete class label. A regression task is one where the model predicts a continuous output. In this project both types are used, the hand detection is a classification task, whilst keypoint detection and pose estimation are regression tasks.

2.3.3 Training

During the creation of a neural network, a key step is training. This involves feeding the model input data repeatedly and comparing the produced output with the desired output, which is often called the ground truth. A single iteration of training is called an epoch. After each epoch, the model alters its internal parameters based on what it has "learnt" to improve the results for the next epoch.

Generally, a model is trained on a dataset split into three parts:

- **Training set**: Used by the model to set internal weights.
- **Validation set**: Used to estimate the performance of the model between epochs.
- **Test set**: Used to confirm the performance of the model after training is completed.

These different sets are kept distinct from each other; it ensures that the model is learning for a general spread of data and not just data that it has seen.

2.3.4 Loss Functions

A loss function is used during the training of a machine learning model to track its performance. During training, this is optimised to reach a particular goal usually minimising the function's output value, meaning a lower loss typically means a better model. Loss functions work by taking the predicted output of a model as well as the target output. Some operation is then applied to these outputs to achieve some numerical value.

2.3.4.1 L1-Norm Loss

A popular type of loss function is L1-norm, which is used for regression tasks. The L1 norm is denoted by $||x||_1$ where x is some vector such that:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

More simply, L1-norm calculates the sum of all absolute values in some vector x. Within neural networks, it is used to calculate the total sum of differences between the

true output p and the predicted output q such that $||p-q||_1$ is used - a smaller difference in the outputs means a smaller loss.

2.3.4.2 Cross-Entropy Loss

Another popular type of loss function is cross-entropy loss which is used for classification tasks. The cross-entropy loss C is defined as:

$$C(p,q) = -\sum_{x \in classes} p(x) \log q(x)$$

Where $p(x)$ is the true probability of x and $q(x)$ is the predicted probability of x.

2.3.5 Evaluation Metrics

When evaluating a model during training, a specific metric is chosen. For classification tasks, the accuracy is measured, this being how many outputs are correctly predicted. For regression tasks, the error is calculated, being a measure of the difference between the predicted and true outputs.

2.3.5.1 Accuracy

For a given class C_1, the accuracy A for the class C_1 is defined as:

$$A = \frac{\text{Total number of inputs correctly predicted as class } C_1}{\text{Total number of inputs belonging to class } C_1}$$

For the total accuracy amongst all classes $C_1, C_2, ..., C_n$ the total accuracy A across all classes is defined as:

$$A = \frac{\text{Total number of correctly predicted inputs}}{\text{Total number of inputs}}$$

2.3.5.2 Error

The error for a given input can be calculated in many ways. A common method is using the mean absolute error. This calculated the same as the L1-norm (as in 2.3.4) and then averaged across all inputs to the model.

2.3.6 Learning Rates

During the training of a neural network, it is constantly being optimised using some optimisation algorithm. A learning rate is a tuning parameter that determines the rate

at which the algorithm updates. The learning rate regulates the weights of a neural network concerning the loss.

This parameter needs to be tuned during experimentation. If a learning rate is too low, it will take a very long time to converge on the optimal value, requiring longer training, or sometimes getting stuck at a local minimum, never achieving a true minimum. If a learning rate is too high, it will cause the model to jump past minima.

2.3.6.1 Adaptive Learning Rates

An optimisation can be made to learning rates by making them adaptive, this increases a model's performance and reduces learning time. One way this can be achieved is by setting up a scheduler that after a set number of epochs, known as the patience, where the model has stopped improving, the learning rate is decreased by some set factor.

2.3.7 Rectified Linear Unit

The rectified linear unit or ReLU is a type of activation layer that can be used in neural networks. An activation layer is one that for a given input outputs a low value for low inputs, and larger values for those above a certain threshold. In the case of ReLU it outputs 0 for any negative value, otherwise, it outputs the input such that:

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

The final output can be seen in Figure 2.5.

The purpose of ReLU is to introduce non-linearity to a model increasing its complexity, and allowing it to learn more complex representations of data.

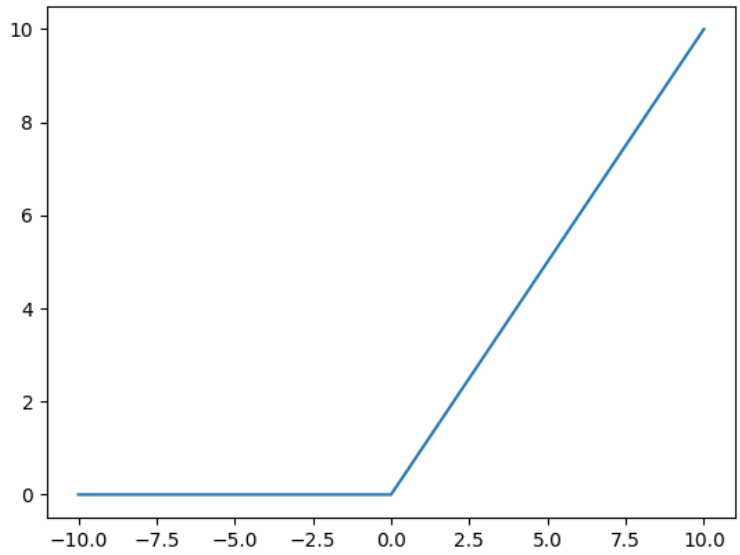


Figure 2.5: Graph demonstrating the ReLU function

2.3.8 Dropout

Dropout is a technique used in neural networks in which nodes in a neural network are dropped out as seen in Figure 2.6. All connections to a dropped node are temporarily removed, creating a new network architecture out of the parent network. Nodes are dropped by some probability p.

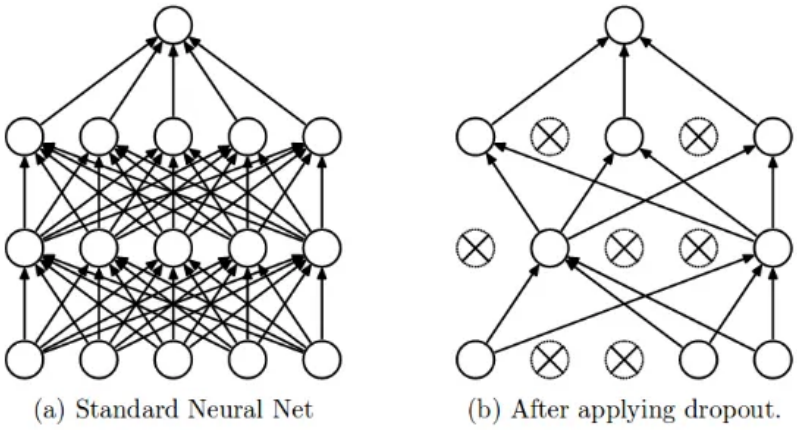


Figure 2.6: Dropout applied to a neural network [17]

The main purpose of dropout is to prevent overfitting. This is the concept of a model adapting to be specialised only to its training data, such that it does not do well on data

not included in the training dataset.

2.3.9 Softmax Functions

A softmax function is one used in multi-classification problems. The function returns a probability distribution over the predicted output classes. The resulting decimal probabilities will end up adding up to 1.0. For example in the case of the hand detection model, for labels null, left and right the model may return $[0.12, 0.66, 0.22]$ indicating the image contains a left hand.

2.3.10 Batch Normalisation

Batch normalisation is a normalisation method applied between layers of a neural network. Normalisation is a data pre-processing method used to bring numerical data to a common scale without distorting its shape. Batch normalisation helps models to learn and adapt quickly.

2.3.11 Upsampling

Upsampling is a technique for creating a larger-resolution image. There are multiple types of upsampling, but the only one used in the project is bilinear upsampling. In bilinear upsampling, upsampling is achieved by using nearby pixels to calculate the output using linear interpolations.

2.3.12 Transfer Learning

Although CNNs are typically trained from scratch, previously trained models can be used as a jumping-off point. This is called transfer learning. In transfer learning, you take a previously trained model, usually trained on a large dataset that is infeasible to do in quick time due to hardware limitations. From the weights of this model, a new model can be trained using a comparatively smaller dataset and still achieve good results.

2.3.13 Early Stopping

If a model is trained for too many epochs, this may lead to a model overfitting. To mitigate this, early stopping may be applied. This concept involves stopping training prematurely before completing all epochs. It's implemented when, after a certain number of epochs known as "patience", the model does not demonstrate any improvement. Typically this is based on the loss but may also be based on accuracy or error.

2.4 ResNet-50

A large part of this project is being able to extract features from images reliably. A model that can extract good features from an image can better identify patterns relating to a wanted output. Multiple similar projects (such as [21, 22]), have used the ResNet-50 architecture [10] as a backbone of their architectures to extract features from their images. Like in these examples, this is also used here due to its powerful feature extraction capabilities.

ResNet-50 is a CNN that is 50 layers deep. It has been trained on millions of images from the Imagenet dataset [8] which is a large dataset of images, largely used for object detection. By using ResNet-50 as a backbone for feature extraction along with its pre-trained weights, the accuracy of these models is greatly increased, in turn, this allows for less training and training data being needed to achieve a well-performing model. (For a full architecture diagram see Figure 2.7).

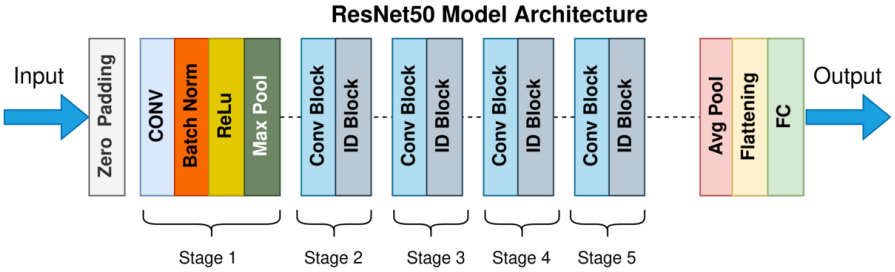


Figure 2.7: ResNet-50 Architecture [3]

2.5 MANO

MANO (Hand Model with Articulated and Non-rigid defOrmations) [14], as previously stated, is the name of the hand model chosen for the project. This was due to it being widely available and being used majorly in multiple papers (such as [12, 21, 22]) leading to a lot of resources that help to use it, as well as its high ease of access to their data.

MANO is a 3D model learned from approximately 1000 3D scans of hands, taken from 31 subjects. The model, according to the creators, is realistic, captures non-rigid shape changes, compatible with standard graphics packages, and can fit any human hand. This versatility is what makes it so great for tracking a hand in 3D space.

Part of the MANO model is its pose features, which consist of 48 values θ, representing 16 3D rotation angles such that $\theta_i \in \{-\pi, \pi\}$. The first set of angles represents

the global rotation of the hand. The remaining 15 represent the angle deformations at predefined joints in the hand. By manipulating these values, a wide range of hands can be posed with the model.

2.6 Biomechanical Constraints BMC

As previously stated in 2.1 BMC involves a system that uses real-world constraints to optimise predicted keypoint annotations. The BMC losses consist of a combination of three losses based on bone length, root bones and joint angles. When referring to bones this refers to the connections between each joint spanning along a finger.

2.6.1 Bone Length Loss

This loss surrounds the length of each of the "bones". For a given bone i there is a defined interval $[b_i^{\min}, b_i^{\max}]$ of the valid bone lengths. If the bone length of the given bone is outside this interval then a penalty is applied.

2.6.2 Root Bone Loss

This loss checks the validity of the structure and shape of the palm. This is done by calculating the curvature of the hand c_i and the angular distance between neighbouring bones ϕ_i.

A positive value of c_i represents an arched hand, such as when the pinky and thumb touch. A flat hand has no curvature.

The curvature and angular distance are then constrained with a valid range $[c_i^{\min}, c_i^{\max}]$ and $[\phi_i^{\min}, \phi_i^{\max}]$ to calculate the final loss, penalising when either lies outside this range.

2.6.3 Joint Angle Loss

This loss surrounds the articulation of individual fingers. The flexion and abduction angles are calculated for each joint. For each angle calculated, there are penalties applied if any lie outside a pre-defined range of valid angles. These valid ranges are taken from the constraints calculated from real-world hands.

Chapter 3

Development

3.1 Pipeline

As previously mentioned the final goal of my project is to create a pipeline that connects three machine-learning models. During the process of developing the pipeline, I individually developed each stage of hand detection, keypoint detection and pose estimation. After forming the individual stages, these are then combined into a pipeline, which is demonstrated with a visualisation tool. The following sections describe the processes taken and the methodologies used.

3.2 Hand Detection

3.2.1 Datasets

In this model, three distinct datasets are utilised. For the "null" dataset, images taken from the PASS dataset [4] were used. This was chosen over the popular ImageNet dataset due to PASS not including any images of humans. This is advantageous because if I were to use a dataset with humans such as ImageNet, this could confuse the model into predicting images with hands as null due to these being present in the null dataset.

For the hand datasets, three datasets were utilised - Freihand, and both a real and synthetic dataset from *Simon et al.* [15]. By using multiple datasets, the data is diversified, allowing the model to generalise to a wider range of different hands.

The use of synthetic hands in datasets has been proven to improve the results of related machine-learning models [19]. A main factor for this is that it provides a wider range of distinct hands to learn from.

For the previously mentioned hand datasets, some additional pre-processing was

applied to fit the needs of training. For the Freihand dataset, due to it only containing right hands, the images are mirrored across the y-axis to achieve left hands. The other two datasets already contained a mix of left and right hands. However, this was then further split into a mixture of full and randomly zoomed-in images. Doing this allows the model to recognise hands by themselves and within a wider context.

3.2.2 Model Architecture

The architecture of the model for this draws inspiration from the approach from "minimal-hand" by *Zhou et al.* [21], which as previously stated utilised ResNet-50 as the backbone of their hand detection system.

Following the initialisation, the final classification layer from ResNet-50 is removed, and in its place, a customised structure is introduced. To begin with, there is a fully connected layer, serving to reduce dimensionality to 128 features. Afterwards, a ReLU layer is applied to introduce non-linearity, followed by a dropout layer with a probability of 0.2 to reduce overfitting. To finalise the model's architecture, another fully connected layer is added to produce the final three outputs of null, left and right. A softmax function is used to extract the probabilities to get a final result. The full architecture can be seen in Figure 3.1.

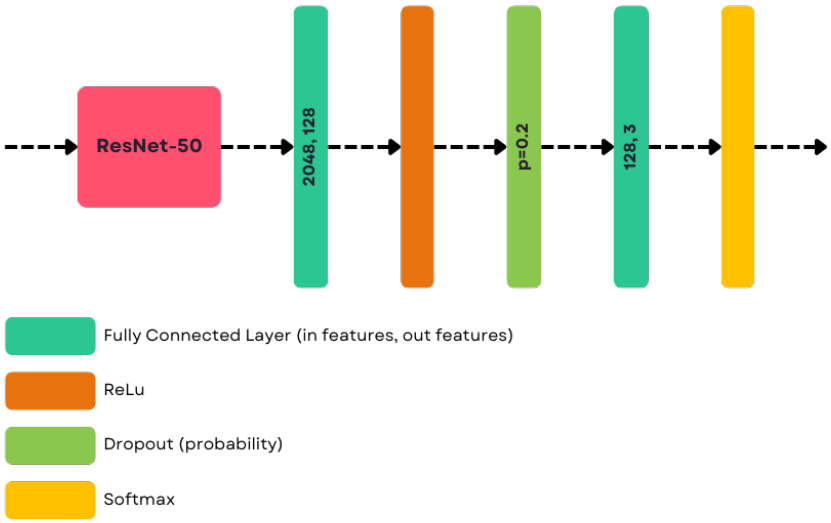


Figure 3.1: Hand Detection Architecture

3.2.3 Data Preparation

Before passing data to the model the data first had to be prepared.

- **Resizing**: Every image used was resized to be 224×224 pixels, this being the same size as images in the Freihand dataset. This is done such that all images are of a consistent size for training.

- **Conversion**: Each image was converted to a tensor to be compatible with PyTorch models.

- **Normalising**: To normalise the images the values are divided by 255, representing the maximum colour being 255, this allows values to be within the range of 0 to 1.

- **Data splitting**: Data is split into training, validation and test sets, in a ratio of $0.7 : 0.2 : 0.1$.

3.2.4 Data Augmentation

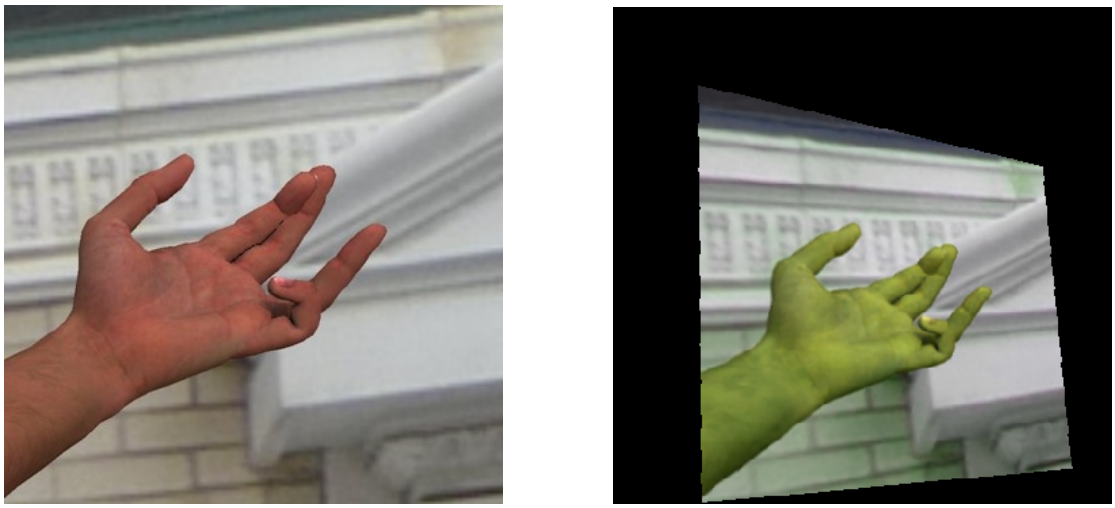
Tellez et al.[18] has shown that image models generalise better when training inputs are augmented. Therefore, images are randomly augmented in the training set. Three distinct augmentations are applied:

- **Colour Jitter**: This affected the brightness, contrast, saturation and hue of pixels randomly. Doing this can limit overfitting to certain colours in an image.

- **Gaussian Blur**: By applying blur to images, the model is prevented from unintentionally learning noise patterns. This also allows the model to further generalise to images of varying quality.

- **Random Perspective and Affine**: These apply random transformations such as rotation and scale allowing the model to recognise further orientations of hands. This is particularly important as from the viewpoint of a camera a hand can take any orientation.

By applying these augmentations, the model can generalise to a wider variety of hands. An example of these augmentations being applied can be seen in Figure 3.2.



(a) Before augmentation (b) After augmentation

Figure 3.2: An Example of Applying Augmentations to Hand Images

3.3 Keypoint Detection

3.3.1 Datasets

In this model, two datasets were employed: Freihand and HOnnotate [9]. Both contain real hands along with annotations for keypoints. By using multiple datasets, the model is diversified, reducing the risk of the model overfitting to a specific dataset.

A key advantage of HOnnotate is that it includes images of hands interacting with objects. For these images often a hand has some occlusions by said object. The keypoints for these images are still included and accurate to where the hidden parts of the hand would be. Therefore, by using this dataset, the model should be able to adapt to partially occluded hands, reducing the error on hands predicted.

For these datasets, some pre-processing is done before passing them to training.

For Freihand, the keypoints are converted to normalised values, such that they are in the range of $\{0,1\}$ referring to where in the frame a coordinate is. In addition to this, the previously mentioned mirrored images are used (see 3.2.1), to convert the keypoint labels for these mirrored images, the following is done:

$$y_i^L = 1 - y_i^R$$

with y^L and y^R being y coordinates for left and right hands respectively.

For HOnnotate, the majority of frames had multiple hands in them. Therefore, I decided to use the more prominent hand, decided by which is closer to the camera.

Random cropping on hands is also applied to make the images square by randomly cutting off a portion of the image to the left and/or right of the hand in the frame.

3.3.2 Model Architecture

The architecture of this model draws inspiration from the approach employed by *Mueller et al.*[12], which uses ResNet-50 as a backbone.

Following the initial ResNet layers, two sets of layers are added, which contain a convolution layer, followed by batch normalisation and then ReLU. The first set takes in 2048 inputs and has 512 outputs. The second set takes in 512 inputs and has 256 outputs. Next, an additional convolution layer is employed that has both an input and output size of 256. Afterwards, bilinear upsampling is applied, followed by an average pooling layer. Finally, a fully connected layer is used that takes in 256 inputs and outputs 63 (21×3) outputs representing the 3D coordinates of the hand keypoints.

The full architecture can be seen in Figure 3.3.

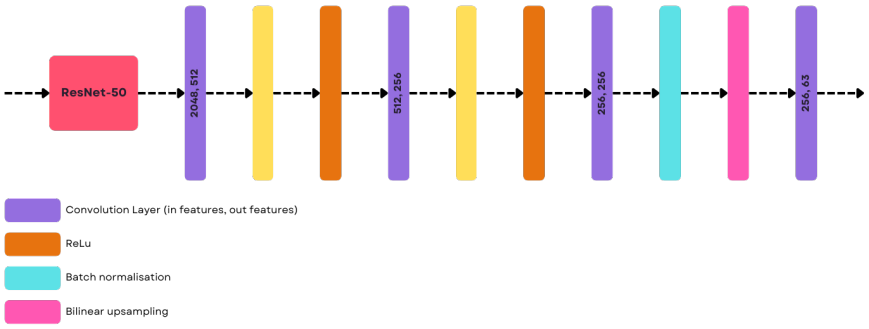


Figure 3.3: Keypoint Detection Architecture

3.3.3 Data Preparation

Before passing data to the model the data first has to be prepared, the same preparations are used as with the hand detection model here (see 3.2.3).

3.3.4 Data Augmentation

As with the hand detection model, data augmentation is applied. Similar augmentations are used as in the hand detection step (see 3.2.4), however, no size or perspective transformations are done. This is due to the keypoints being viewpoint-dependent, a

different viewpoint means different keypoints. If any size or perspective transformations were to be applied, then the keypoint labels for any data with these applied would be inaccurate.

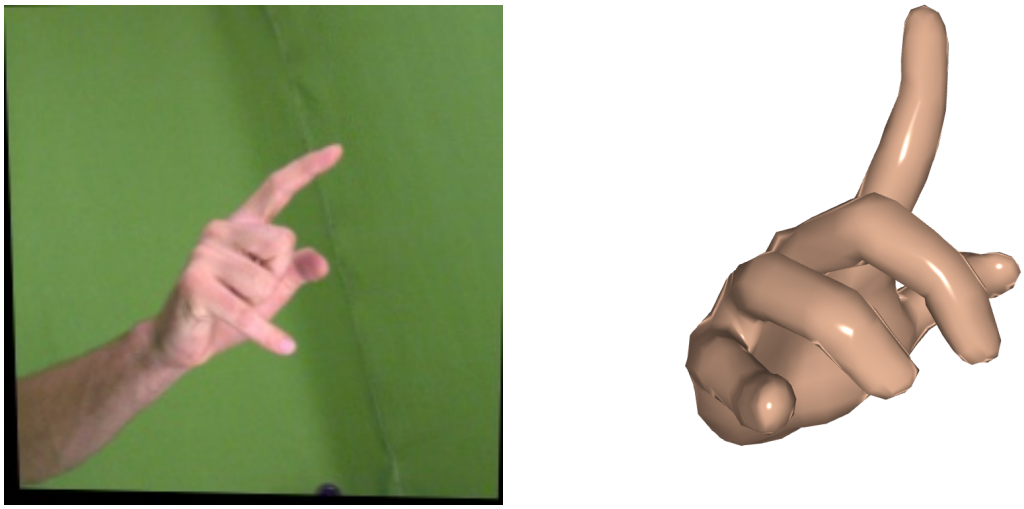
3.4 Pose Estimation

3.4.1 Initial Approach

The initial design of the pose estimation section was a mapping function. This design was based on the idea that by utilising the keypoint coordinates to find angles, it should be possible to calculate the pose angles.

This approach adapted the BMC angles loss function to be a mapping function. It took in the 21 keypoints and outputted joint angles. However, as this method was tested it became clear that just mapping via angles was not enough for posing the MANO model. The method was able to get similar hands to what was expected but often had errors that made hands unappealing. After testing adding some offsets to the joint angles, it became clear that a simple mapping function is insufficient for fitting a pose from the keypoints. This was most likely due to the complex nature of the MANO model which would be difficult to account for with the mapping approach.

For example, as can be seen in Figure 3.4, it is clear that it is the correct hand being posed, but there are multiple errors. The fingers are spread more than in the input image. The joints, especially those controlling the end of the fingers are bent less than they should be. The end of the index finger is bent backwards, which is not typically possible in a human hand. All of these errors make a pose that is below what would be deemed an acceptable accuracy to the original hand.



(a) Input Hand (b) Mapped Hand

Figure 3.4: An Example of Mapping Method Results

Therefore, I subsequently chose to use the machine learning approach used in the final pipeline. With the selected machine learning approach, a model can be created that can account for the complexity of the MANO model and correctly pose a wide range of complex hands.

3.4.2 Datasets

Unfortunately, unlike the other models, multiple datasets could not be used, due to not being able to find multiple ones readily available with annotations for both MANO annotations and keypoints. Therefore, only the Freihand dataset is used in this section. Hence, this also means only a right-hand pose can be estimated in this system, however, the methods used here can be applied if the model were to be trained on labelled left-hand data. The use of only one dataset also may mean the model may not adapt to as large a range of hands as would be preferable. However, the Freihand dataset is very large with over 30,000 unique hands, with many unique poses. Therefore, the model should still adapt somewhat well to poses, despite only using hands from a singular dataset.

3.4.3 Model Architecture

The model architecture for this section also takes inspiration from “minimal-hand” by *Zhou et al.*. As in their pose estimation model, a fully connected model is used.

This begins by applying batch normalisation to allow the model to better generalise to similar poses in different positions of a frame, this layer takes in 21×3 inputs corresponding to the 21 3D keypoints of a hand. The proceeding layers alternate between fully connected layers and batch normalisation layers with feature sizes $\{2048, 1024, 523, 256, 128\}$. Finally, a fully connected layer followed by a ReLU activation function, with an input of 128 and an output of 48 (16×3) representing the estimated pose features.

The full architecture can be seen in Figure 3.5.

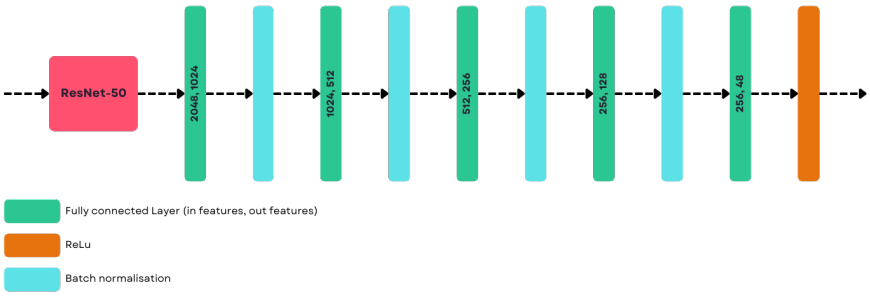


Figure 3.5: Pose Estimation Architecture

3.4.4 Data Preparation

For the input keypoint coordinates taken from Freihand the normalised coordinates calculated in 3.3.1 are used.

Once again the data is split into training, validation and test sets, in a ratio of 0.7 : 0.2 : 0.1.

3.5 Visualisation

To evaluate the performance of the pipeline a visualisation system was built, that combines the three stages. The system uses OpenCV to load images or capture video. For a given frame, it is passed into the hand detection model. If a hand is detected, the image is then passed into the keypoint detection model. The keypoints detected are overlayed onto the input image and displayed in an OpenCV window. Finally, the pose is estimated depending on which hand, in this case, it only estimates a pose if a right hand is detected. This is done by passing in the keypoints found in the previous step, the model outputs 48 pose features which are applied to a loaded right-hand MANO model. The model is then visualised in an Open3D window. The final label, the overlayed keypoints, and the 3D visualisation are all shown simultaneously in real-time.

Some processing decisions are also added to make the pipeline run smoother. Instead of running on every incoming frame, it is run on every third frame to not overload the device the pipeline is running on. If the frame rate is very high, the pipeline could end up not being able to keep up, thus slowing down the process. The reduction in predicted frames also reduces the jitter between calculated frames which becomes very obvious and unsatisfying at high frame rates. The incoming images were also cropped to a square resolution, making them better suited to the models since they were trained on square-shaped images.

During the development of the project, a WSL (Windows Subsystem Linux) container on a Windows machine is used. Because of this, there was no direct access to a camera from the Windows machine in this development area. To get around this, a virtual camera is used to pass a camera feed from the Windows device through to the WSL container so the visualisation step could still use a live video capture.

3.6 Technologies

During the development of the project I used various technologies:

- **Python** was chosen due to it being one of the most widely used languages for machine learning, being used by most codebases I explored during my initial research.
- **PyTorch** [13] is a machine learning library originally developed by Meta AI and is used vastly by various members of the field. PyTorch was chosen over other libraries, due to its familiarity and being used in most codebases explored during my initial research. In addition to this, it allows for learning to be run on a device's GPUs allowing for faster, more efficient training.
- **Pillow** [7] is a Python library for image processing. In this project, it is used during training for loading and manipulating images due to its ease of interaction with PyTorch.
- **Open3D** [20] is an open-source library for 3D data. It allows for visualising 3D data in real-time as well as parallelisation, making it a great library for visualising the pose estimation section of the pipeline.
- **OpenCV** [5] is a library primarily used for computer vision. Although computer vision is its main purpose, in this case, I use it for loading images and live camera capture.

Chapter 4

Experimentation

To achieve the best possible models, parameter optimisation was applied. This chapter explains the various experiments employed for each section to find the best parameters.

4.1 Hand Detection

4.1.1 Constants

The Adam optimisation function [11], which is popular amongst many machine learning training methods, is used across all experiments.

The loss was consistent among experiments, using cross-entropy loss.

The dataset size is kept constant throughout experiments at a size of 30,000, using the same data across experiments.

All experiments used a plateau-based scheduler, that had a patience of 2 epochs and decreased the learning rate by a factor of 2 upon reaching this patience.

Early stopping with a patience of 10 is applied and training runs for a maximum of 50 epochs.

4.1.2 Learning Rates

Multiple learning rates are explored to achieve the best results from optimisation.

The experiments here test values in the range of 10^{-1} to 10^{-5} decreasing by a factor of 10 each time.

The optimal learning rate is selected by:

- *Speed*: How fast the loss decreases, as in a learning rate that takes fewer epochs to reach the same lowest value.

- *Noise*: How much the loss varies between epochs, a better learning rate will have a smoother curve.
- *Final loss*: The loss achieved after the final epoch, a lower loss indicates a better model.
- *Accuracy*: The accuracy of the model, a better learning rate will produce a model with a higher accuracy.

4.1.3 Post Training Experiments

For the hand detection model, some extra experiments outside of training are done. This was done to further refine the results of the model when being used on sequential data. For the following experiments, only accuracy is used as a measure of a better result.

4.1.3.1 Detection Window

The process involves exploring a window, consisting of predictions used to determine the final prediction of a frame. For instance, with a window size of 5, the last five (inclusive of the current) recorded frames are considered, and the most frequently predicted label among these frames is returned as the final result.

The motivation behind this is to counteract random error frames, under the assumption that what is in a frame is likely to remain constant for a set amount of frames.

4.1.3.2 Null Thresholding

Experimentation was conducted on applying thresholding to classify null frames. This involved determining if the probability of null exceeded the predefined threshold. If the threshold is exceeded, the frame is labelled as null; otherwise, it is labelled as left or right based on whichever had the highest probability.

By applying thresholding the hope is to make more ambiguous frames still have accurate predictions.

4.2 Keypoint Detection

4.2.1 Constants

The Adam optimisation function is used throughout all experiments.

The error was kept as the mean absolute error for each experiment.

The dataset size was kept constant throughout experiments at a size of 30,000, using the same data across experiments.

All experiments used a plateau-based scheduler, that had a patience of 2 epochs and decreased the learning rate by a factor of 2 upon reaching this patience.

The experiments use early stopping with a patience of 10 and run for a maximum of 60 epochs.

4.2.2 Learning Rates

As in 4.1.2, except instead of accuracy the error of the model is measured, a better learning rate will produce a model with a lower error.

This experiment was repeated for each of the loss types to ensure the best possible was chosen, specific to each of the three.

4.2.3 Loss Types

Multiple types of losses are experimented with - L1-norm and two more that combine L1-norm with losses calculated using losses from BMC.

From BMC, only angle and combined constraints were experimented with.

The learning rate used for each model was chosen based on the results from the previous learning rate experiments.

The optimal loss is determined by:

- *Speed*: How long it takes the total loss to approach a minimum, optimal weighting should take fewer epochs to close in on a minimum.
- *Error*: The model's error, a better learning rate will produce a model with a lower error.
- *Look of hands*: How the connected keypoints look compared to the actual hand. A better loss will closely resemble the actual hand as well as have minimal abnormalities such as weird bends in fingers.

4.3 Pose Estimation

4.3.1 Constants

The Adam optimisation function is used throughout all experiments.

The dataset size is kept constant throughout experiments at a size of 30,000, using the same data for each experiment.

The error was kept as the mean absolute error for all experiments.

The loss function is constant throughout the experiments, being a combination of two functions. The first is the L1-norm on solely the first output, which represents the global rotation. The second is the L1-norm on the total outputs. The purpose of having an additional loss on the global rotation is so the model may prioritise having the hand face the correct direction. This gives a better-looking hand even if the individual joint rotations are not as accurate.

The experiments use early stopping with a patience of 25 and run for a maximum of 350 epochs.

All experiments used a plateau-based scheduler, that had a patience of 5 epochs and decreased the learning rate by a factor of 2 upon reaching this patience.

4.3.2 Learning Rates

As in 4.2.2.

4.3.3 Loss Weighting

This experiment explores multiple weightings for the combination of loss functions. This is done such that for a given weighting $[x, y]$, the output loss will be $x \times G + y \times T$, where G is the loss calculated on the global rotation and T is the loss calculated on the total outputs.

The optimal loss weighting is determined by:

- *Speed*: How long it takes the total loss to approach a minimum, optimal weighting should take fewer epochs to close in on a minimum.
- *Error*: The model's error, a better weighting will produce a model with a lower error.
- *Complexity of Poses*: How the model performs on more complex poses, for example, multiple bent fingers. A better weighting should adapt better to complex hand poses.

Chapter 5

Evaluation

In this section, the results from the experiments are evaluated, from the individual sections and the overall pipeline.

5.1 Hand Detection

5.1.1 Learning Rates

From the learning rates experiments, a learning rate of 10^{-4} was found to perform the best.

In terms of loss, as seen in Figure 5.1, it performs better than all others reaching a minimum loss value of about 0.0001.

For accuracy, as seen in Figure 5.2, it by far outperforms all other losses reaching a total accuracy of 0.89 across all classes during training. For the unique classes, null has an accuracy of 0.999, left has an accuracy of 0.829 and right has an accuracy of 0.855.

This learning rate also begins to plateau fairly quickly after around 5 epochs. Finally, there is minimal noise in both loss and accuracy, with minimal variability between each epoch.

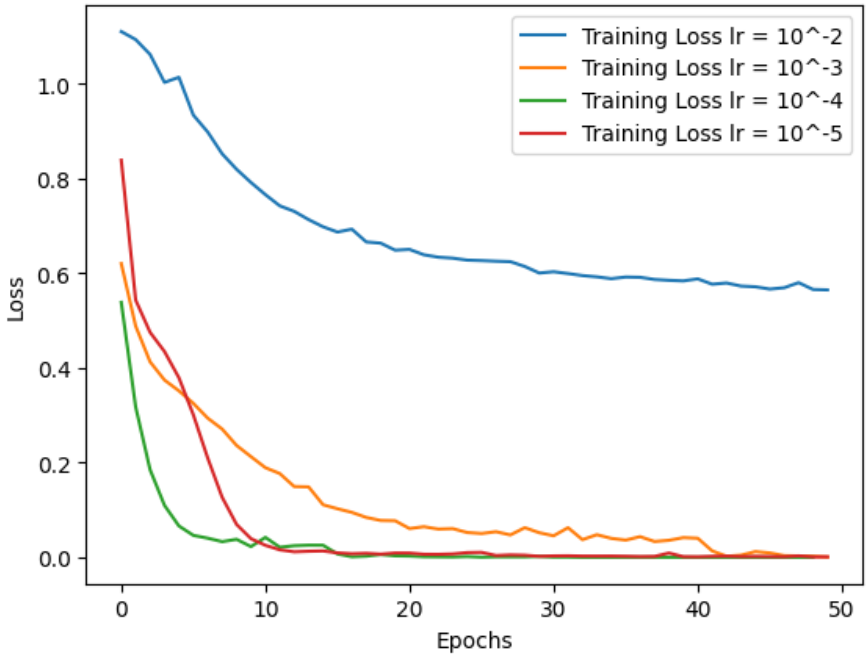


Figure 5.1: Training Loss for Learning Rates - Hand Detection

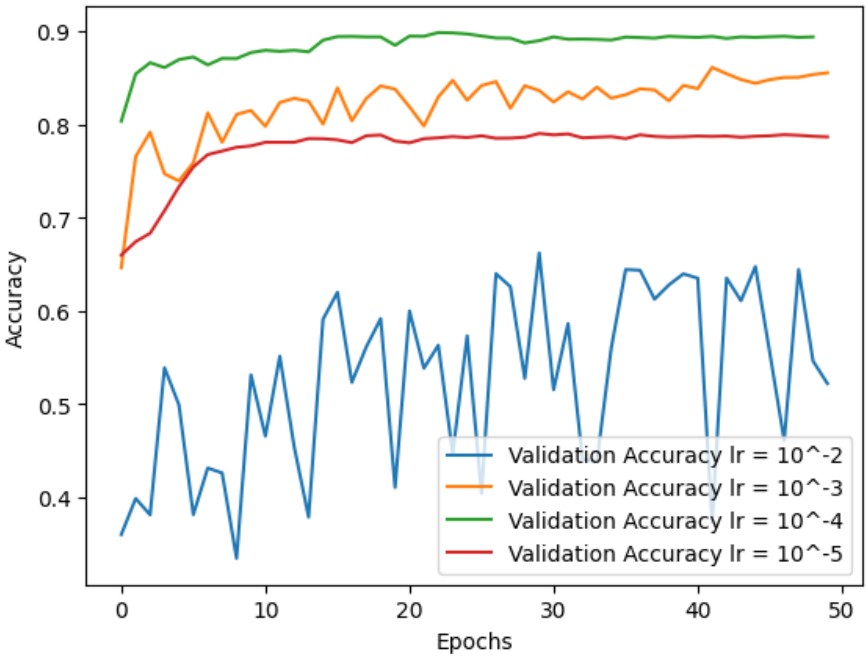


Figure 5.2: Training Accuracy for Learning Rates - Hand Detection

5.1.2 Post Training Experiments

5.1.2.1 Window

For the window, a size of 5 was seen to be good. This window size can reduce noise from the output switching between different labels whilst still updating the hand result in good time.

5.1.2.2 Null Thresholding

For the thresholding value, 50% was observed to be a reasonable value. This means that the model must be 50% certain that it is a null frame, otherwise the most confident hand is predicted instead. This allows for more ambiguous frames to still be predicted as hand frames.

5.1.3 Overall Evaluation

The final hand detection model uses a learning rate of $10^{-}4$ with a window of 5 and a threshold value of 50%.

When run on the test set, the final result of the model gives a high accuracy across all three classes as can be seen in Figure 5.3. The null class has an exceedingly high accuracy which is to be expected as none have hands and are more visually distinct. The hand classes have slightly lower accuracies but that is not surprising due to the similarities between hands. However, the model does successively identify images as hands almost exclusively, with nearly none being predicted as null. The accuracies of the model can be seen in Table 5.1, as can be seen in the table, the final model achieves a final accuracy of 89.57%.

Label	Null	left	Right	Total
Accuracy	99.90%	86.60%	82.02%	89.57%

Table 5.1: Table of Hand Detection Accuracies for Final Model on Test Data

The model can consistently make a correct prediction a majority of the time. However, the model does have an issue in that when a body is in the frame, the model has a higher bias towards hands even if none are present in these cases. This is most likely due to there not being any bodies in the null dataset but plenty in the hand datasets, the model probably learns that a body or some skin showing in a frame means a hand is likely in the frame. This problem could be mitigated in the future by adding some images with bodies but no hands, such as some form of head-shot dataset, into the null dataset, this should reduce the bias toward hand frames.

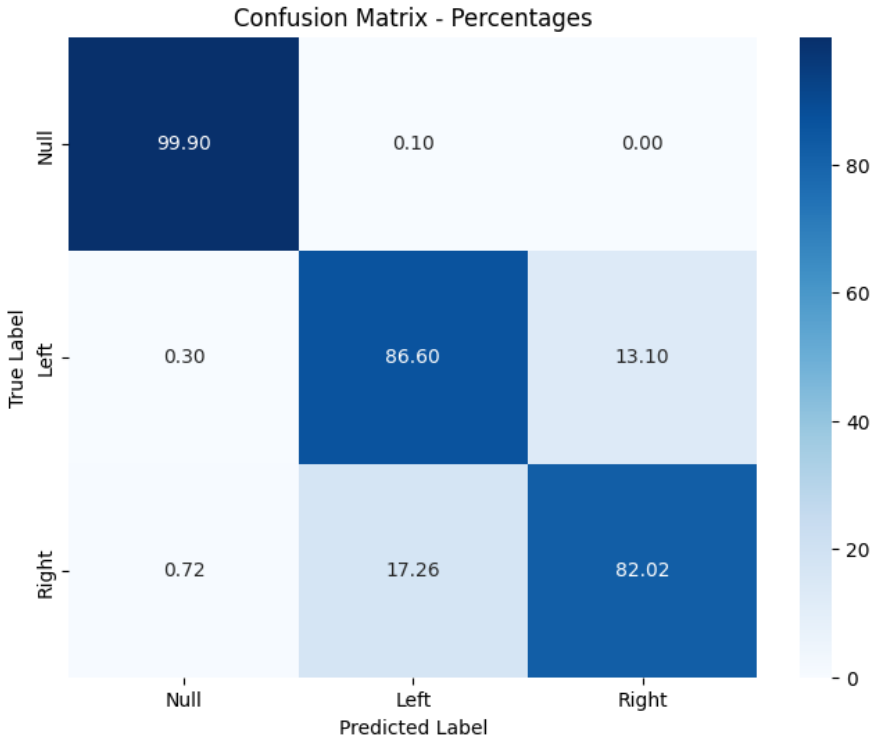


Figure 5.3: Confusion Matrix for the Final Model - Hand Detection

5.2 Keypoint Detection

5.2.1 Learning Rates

This subsection details the learning rate experiments for each of the loss types.

5.2.1.1 L1-Norm

From this experiment, a learning rate of 10^{-3} performed the best.

The loss ends up being the lowest out of all the learning rates as can be seen in Figure 5.4, the final loss achieved during training is 0.211.

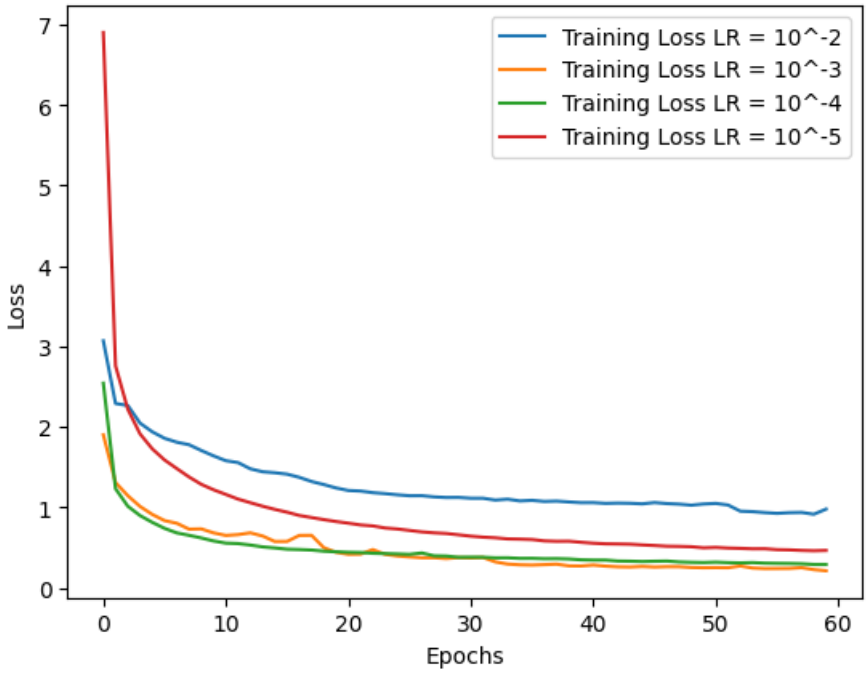


Figure 5.4: Training Loss for L1-Norm Learning Rates - Keypoint Detection

In terms of error, as seen in Figure 5.5, after around epoch 30, it has the lowest error for the majority of epochs, with a learning rate of 10^{-4}, this is most likely a point where the learning rate was decreased. It achieves a final training error of 0.0252.

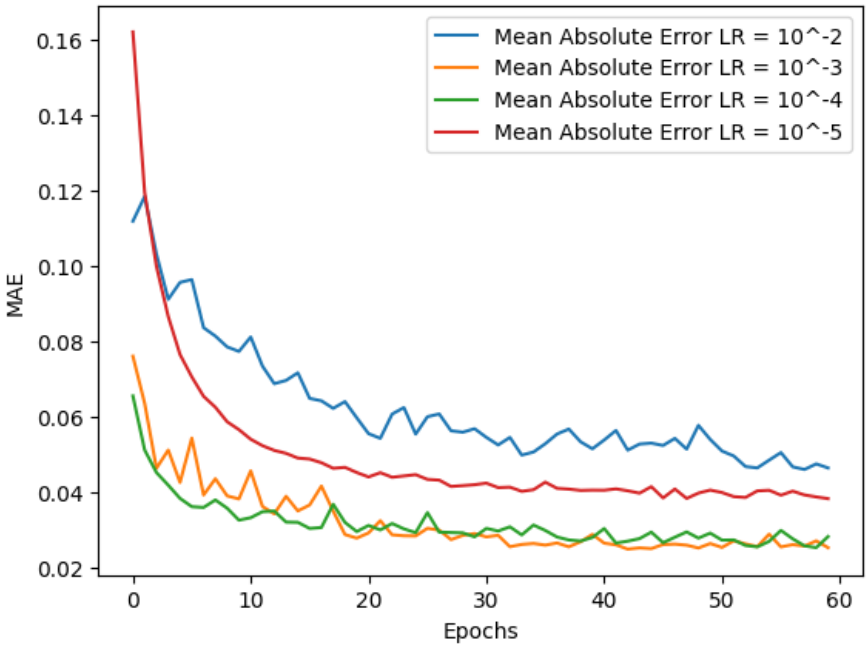


Figure 5.5: Training Error for L1-Norm Learning Rates - Keypoint Detection

5.2.1.2 Angle Loss

From this experiment, a learning rate of 10^{-4} was found to perform best.

In terms of loss, as can be seen in Figure 5.6, it is consistently lower than all other learning rates after the first few epochs. All other learning rates are not remotely comparable to that of 10^{-4}. The final loss achieved by the learning rate of 10^{-4} is 0.316, which is notably over half that of the next-best loss at 0.658.

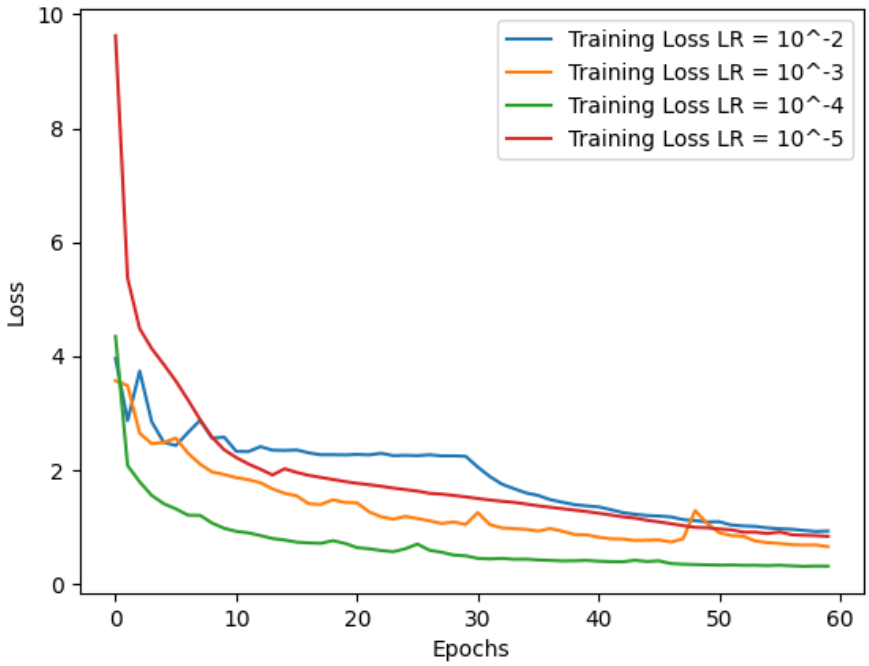


Figure 5.6: Training Loss for Angle Loss Learning Rates - Keypoint Detection

In terms of error, as can be seen in Figure 5.7, it has the lowest error throughout all the epochs. It achieves a final error of 0.0306.

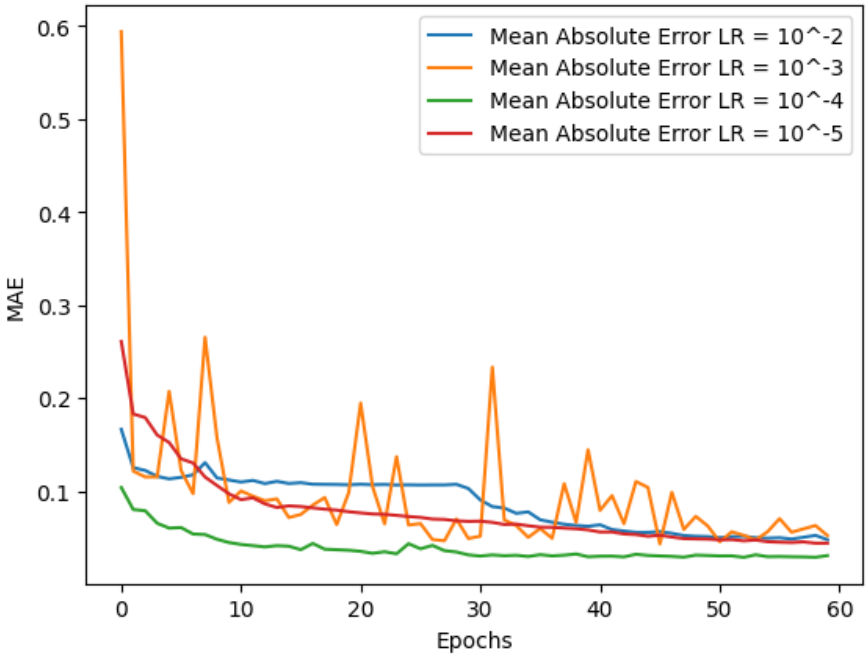


Figure 5.7: Training Error for Angle Loss Learning Rates - Keypoint Detection

After around 20 epochs, the model begins its plateau. There is also very little noise between epochs.

5.2.1.3 Combined BMC Loss

From this experiment, a learning rate of 10^{-4} performed the best.

In terms of loss, as can be seen in Figure 5.8, it is consistently lower than all other learning rates after the first few epochs. All other learning rates are not remotely comparable to that of 10^{-4}. The final loss achieved by the learning rate of 10^{-4} is 0.807.

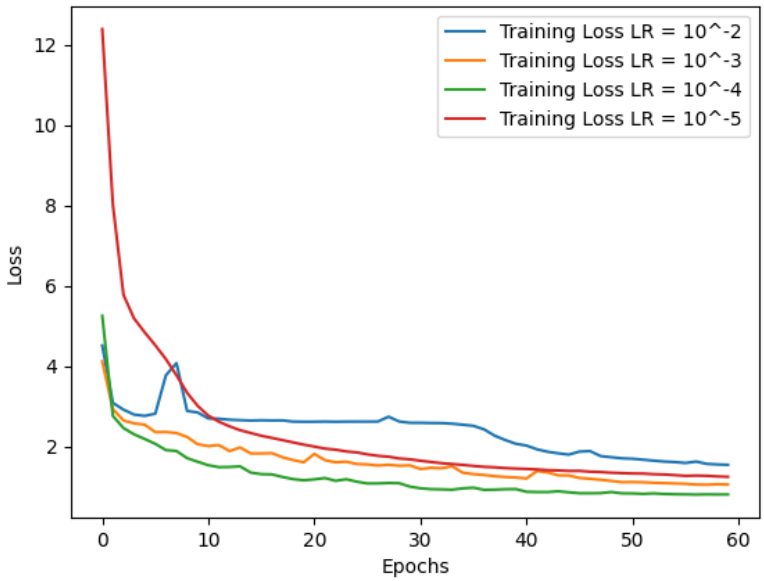


Figure 5.8: Training Loss for BMC Loss Learning Rates - Keypoint Detection

In terms of error, as can be seen in Figure 5.9, it has the lowest error throughout all the epochs. It achieves a final error of 0.0354.

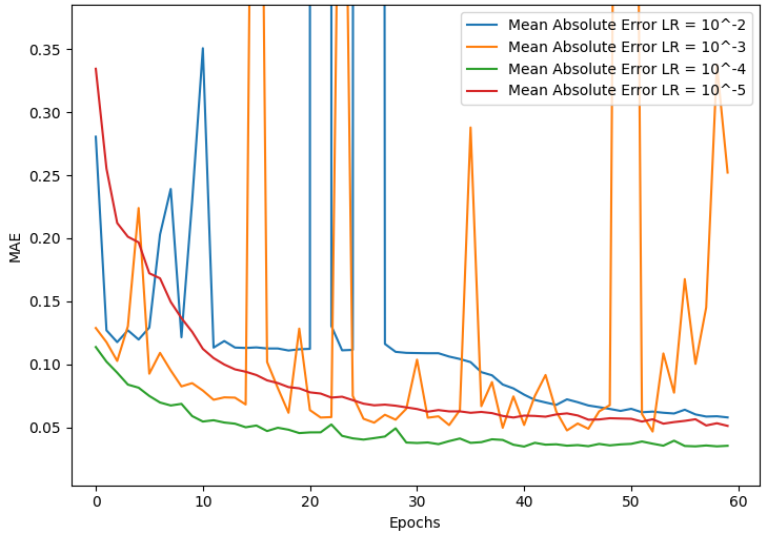


Figure 5.9: Training Error for BMC Loss Learning Rates - Keypoint Detection

After around 30 epochs, the model begins its plateau. There is also very little noise between epochs.

5.2.2 Loss Types

From the loss-type experiment, the angle loss was found to be the most suitable.

In terms of loss, shown in Figure 5.10, all three follow a similar shape. All begin to plateau after around 40 epochs. The L1 Loss and angle loss are close together, with L1 being slightly lower, whilst the combined BMC loss is distinctly higher. However, this is to be expected and does not necessarily reflect the performance of the loss type.

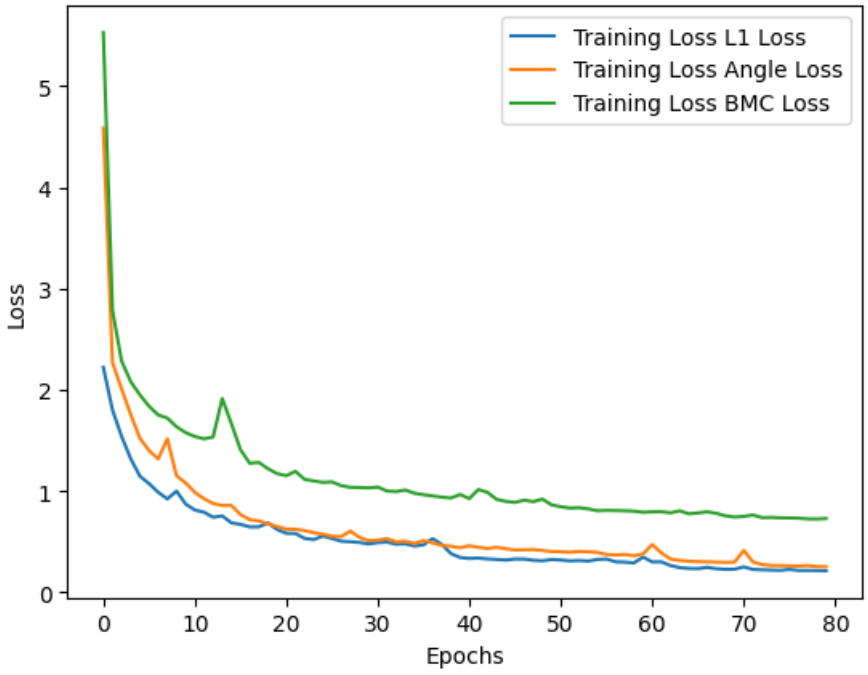


Figure 5.10: Training Loss for Loss Types - Keypoint Detection

For the error, shown in Figure 5.11, L1 loss and angle loss errors are very close together for most of the epochs, meanwhile, the combined BMC loss error is consistently higher. The L1 loss and angle loss both produce a similar low final error of 0.281 and 0.251 respectively, whilst the combined BMC loss has a slightly higher final error of 0.323.

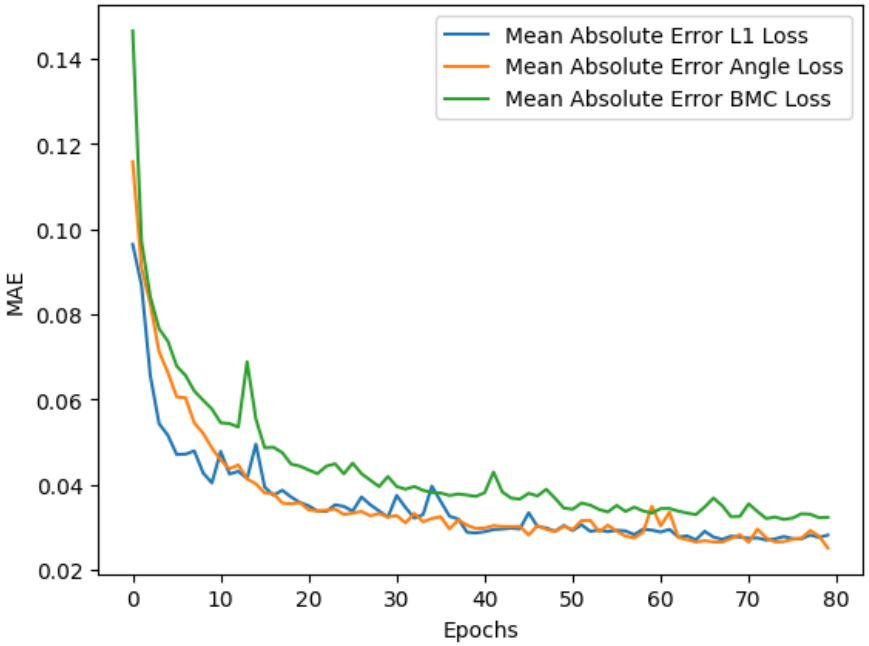


Figure 5.11: Training Error for Loss Types - Keypoint Detection

Figure 5.12 shows a histogram of the errors on the test set for the L1 loss and angle loss. As can be seen in the figure, the angle loss has a lower mean error at 0.0318 compared to the L1 loss at 0.355.

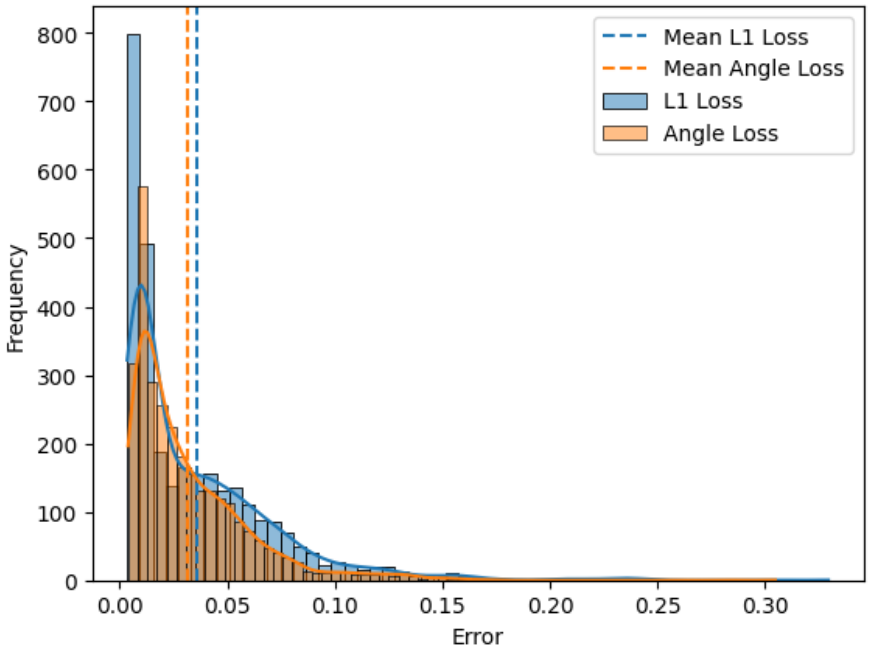


Figure 5.12: Testing Error for L1 and Angle Losses - Keypoint Detection

The combined BMC loss underperforms when compared to the other loss types

with its distinctly higher error. It also consistently incorrectly predicts the position of the base of each finger.

The L1-norm loss and angle loss are far more comparable, with the loss values being fairly similar for both and following a similar curve and the error between the two is quite small. In terms of the final look of hands, both produce good-looking hands, but the angle loss is capable of correcting mistakes not picked up by the L1-norm. Based on the look of hands being better and the other two parameters being similar, the angle loss is judged to be the better loss.

A visual comparison can be seen in Figure 5.13. By taking a closer look, some of the discrepancies mentioned above can be seen:

- In the L1 model, the index, middle and ring fingers are all curved in a manner that is not usually possible. In the angle model, these have been corrected into straight fingers. This not only makes the results look better but also more accurate.

- In the combined BMC model, the base of the middle and ring fingers are higher than they should be. Not only is this inaccurate but also causes the palm to seem to be more curved than it should be.

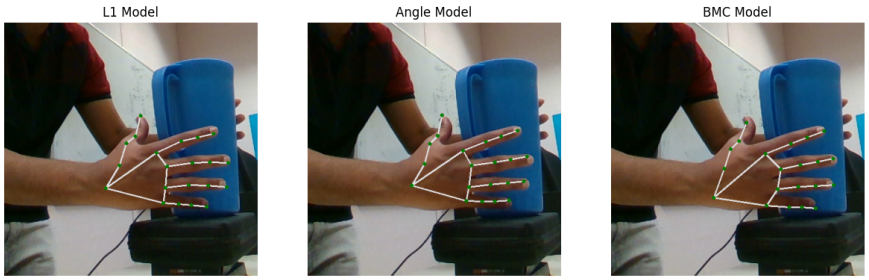


Figure 5.13: Example Result from Each Loss Type - Keypoint Detection

5.2.3 Overall Evaluation

The final keypoint detection model uses the angle loss and a learning rate of 10^{-4}.

The final results when the final model is run on the test set are tight on the predictions, as can be seen in Figure 5.14, the final error achieved is low at a value of 0.0331. There is a wider margin of error on the x coordinates, especially when compared to the y and z. This most likely is from predicting the opposite hand than intended, however, it is still within a good margin for the majority of predictions, as can be seen in Figure 5.15.

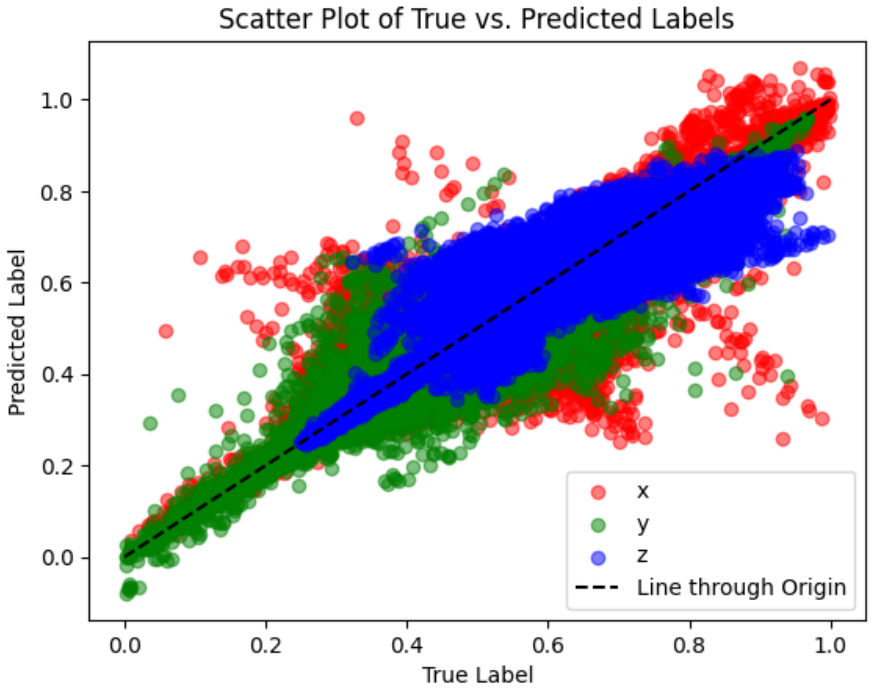


Figure 5.14: Results of Final Model on Test Set - Keypoint Detection

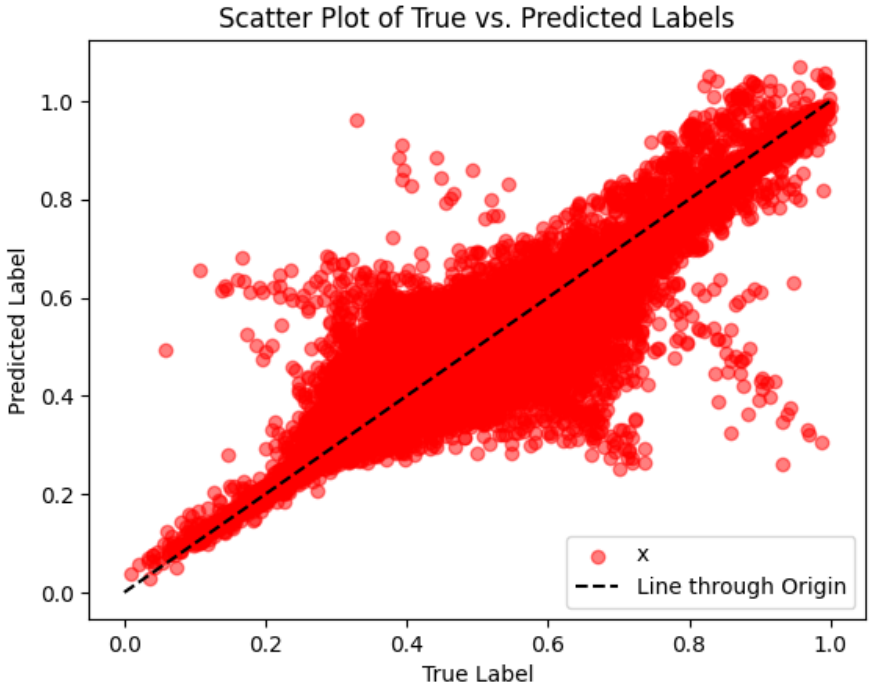


Figure 5.15: X-Coordinate Results of Final Model on Test Set - Keypoint Detection

A collection of example results can be seen in Figure 5.16. As demonstrated in these examples the predictions made are accurate to the hand to a close degree. When there

are errors, these tend to be minimal and the general shape of the hand is still portrayed accurately.

Figure 5.16: General Example Keypoint Detection Results

The model can predict complex poses reliably, with high precision. The use of the angle model reduces abnormalities in hands which further enhances its capabilities. The model is also capable of predicting hands both with self-occlusions and object occlusions. The model in these cases is capable of predicting the position of the occluded fingers accurately in a large number of cases. Some examples of results on hands with occlusions can be seen in Figure 5.17.

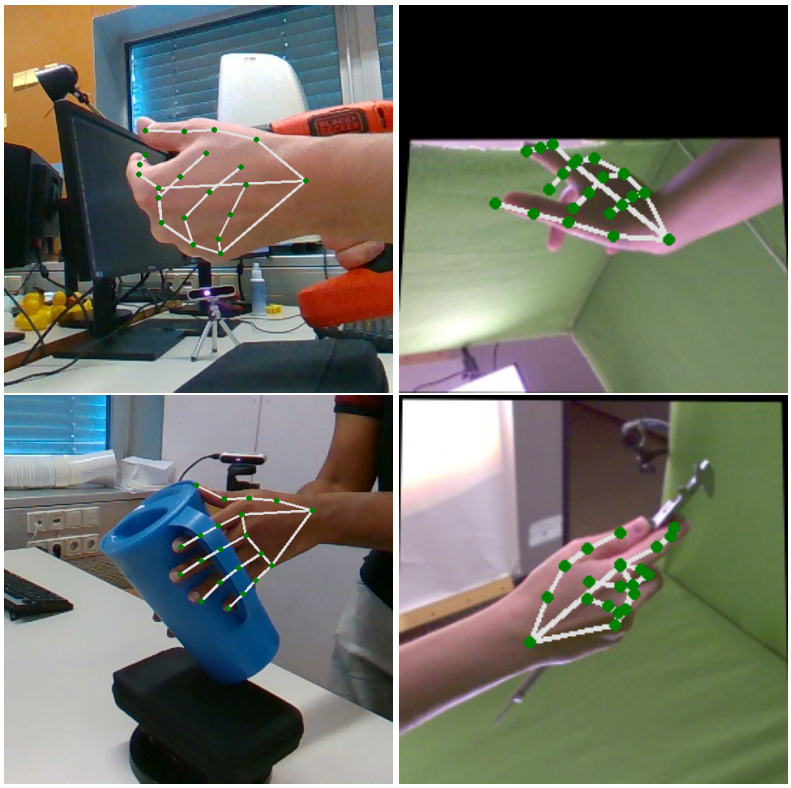


Figure 5.17: Example Keypoint Detection Results on Hands with Occlusions

The model appears to falter at hands at weird angles such as the fingers being pointed directly at the camera. However, this can be deemed not a significant problem though, unusual perspectives can also be difficult for humans to decipher, so the model being unable to do so is not surprising.

5.3 Pose Estimation

5.3.1 Learning Rates

From the learning rates experiment, a learning rate of 10^{-3} was seen to perform the best.

For the loss, as seen in Figure 5.18, during most of the first around 200 epochs, it is comparable to a learning rate of 10^{-4}. However, after this point, the loss achieved from a learning rate of 10^{-3} is by far lower than all the other learning rates, achieving a final loss of 1.21.

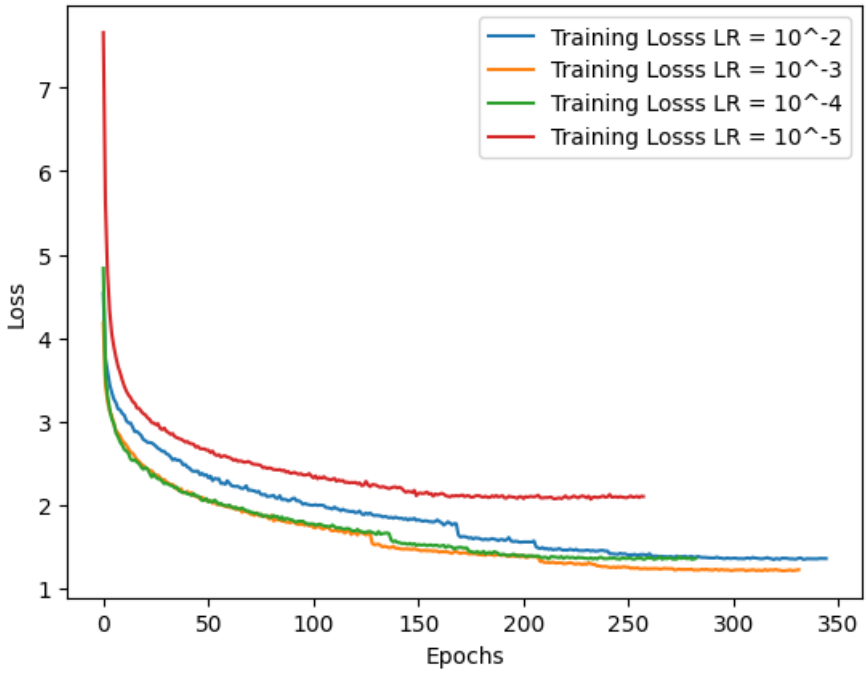


Figure 5.18: Training Losses for Learning Rates - Pose Estimation

For the error, as seen in Figure 5.19, it is once again comparable to a learning rate of 10^{-4}, this time for roughly 250 epochs. After this point, the error is the lowest of all the learning rates, achieving a final error of 0.0746.

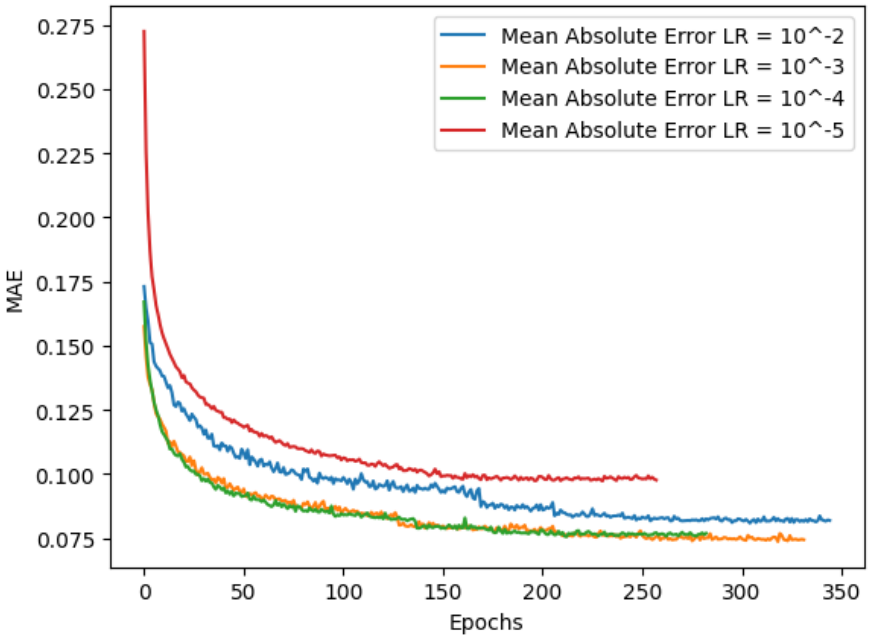


Figure 5.19: Training Errors for Learning Rates - Pose Estimation

5.3.2 Loss Weighting

From this experiment, a weighting of $[1,2]$ was found to be the most appropriate.

For the loss, as shown in Figure 5.20, the rate at which the loss decreases across all the weightings is quite similar, the only significant difference in this experiment appears to be the offset in the magnitude of the loss. In general, for a given weighting $[i, j]$, a bigger value for $i + j$ results in a bigger loss value, which is to be expected. Therefore, this is not a very useful metric for the best weighting.

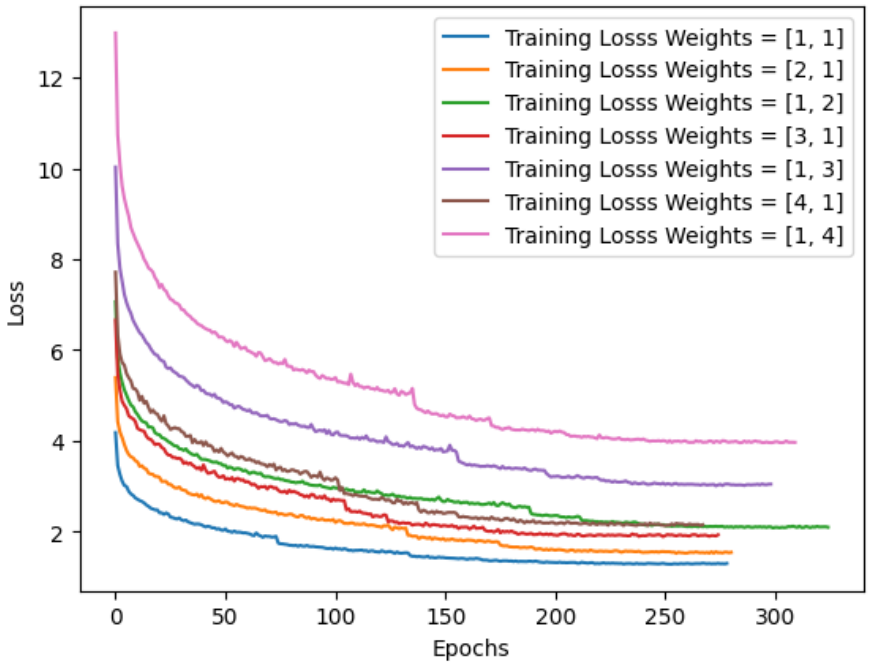


Figure 5.20: Training Losses for Weighting - Pose Estimation

For the error, as seen in Figure 5.21, there are clearer differences in the results than with the loss. In general, for any weighting $[i, j]$, a weighting such that $i < j$ gives a better result than if $i > j$. The weightings with the three lowest errors are $[1,2]$, $[1,3]$ and $[1,4]$.

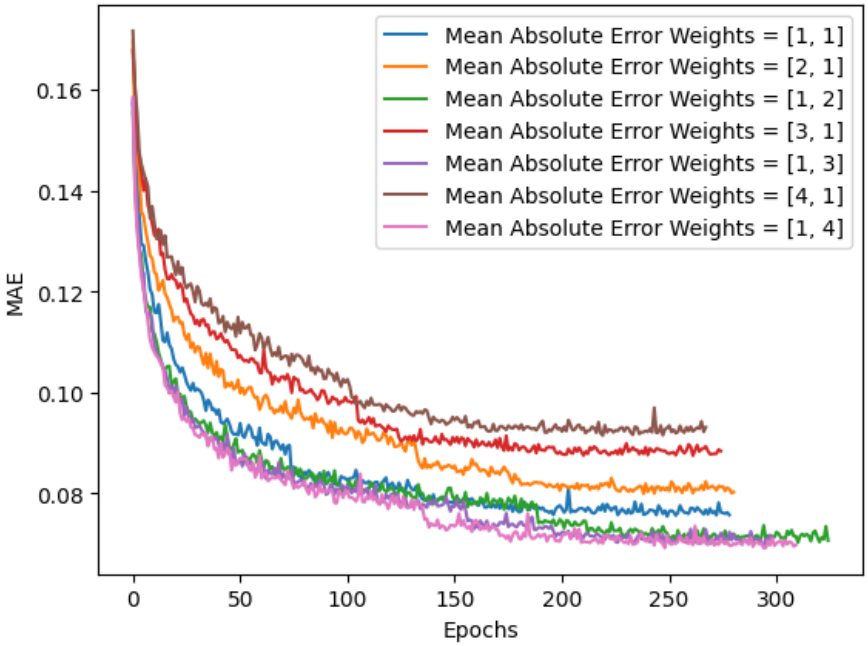


Figure 5.21: Training Errors for Weighting - Pose Estimation

The weightings with the three lowest errors were further explored as they were very similar. Note that for the following graphs, the extreme errors have been cut off to allow for better visuals.

Firstly, the overall error of both the global rotation as well as joint rotations are examined, the results can be seen in Figure 5.22. The weighting of $[1,4]$ gives the lowest mean error at around 0.0832, then the weighting of $[1,2]$ follows with a mean error of around 0.0845, and the weighting of $[1,3]$ is the highest at around 0.0847.

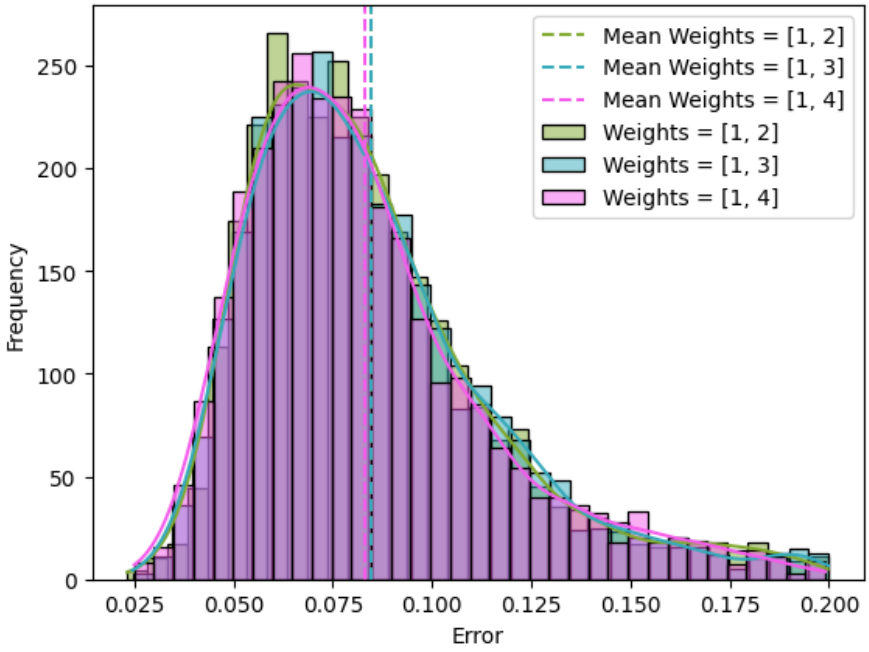


Figure 5.22: Histogram of Overall Errors for Best Weightings - Pose Estimation

Next, just the rotation errors are analysed, the results can be seen in Figure 5.23. The weighting of $[1,2]$ has the smallest mean error at around 0.0508, then the weighting of $[1,4]$ is next at around 0.0553, and the weighting of $[1,3]$ is the highest at around 0.0610.

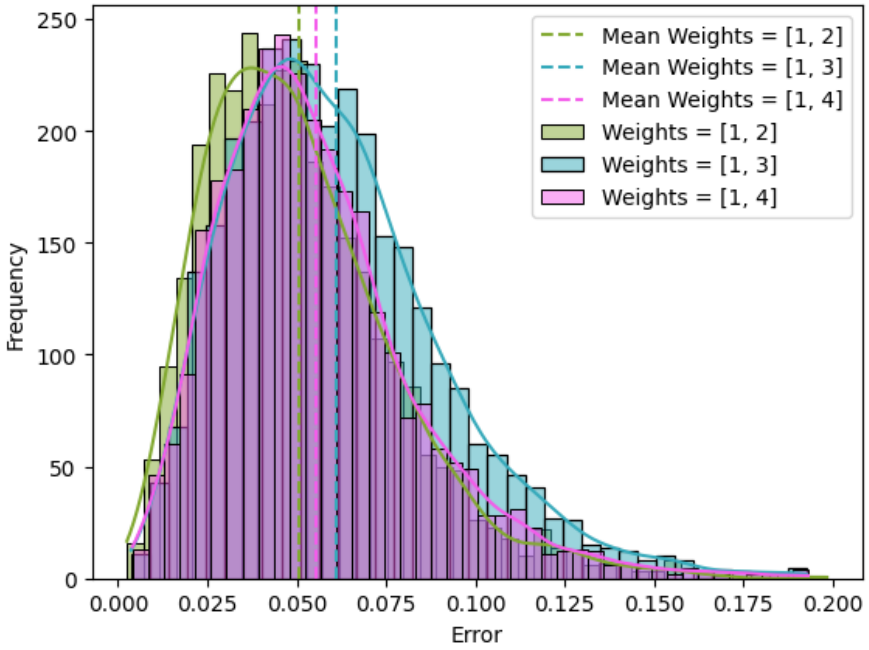


Figure 5.23: Histogram of Rotation Errors for Best Weightings - Pose Estimation

As the mean overall errors are all within 2% of each other the difference is deemed

non-significant and thus a poor metric. Therefore, the final decision is based upon the mean rotation error, due to there being a larger difference between each, which is why the weighting of $[1,2]$ is chosen.

5.3.3 Overall Evaluation

The final pose estimation model uses a learning rate of 10^{-3} with a loss weighting of $[1,2]$.

When run on the test set, the model in general can estimate the pose accurately with correct global rotation. It has achieved a low error of 0.0832.

As can be seen in Figure 5.24, the joint rotations consistently have a small error in their predictions. Also seen in the figure, there are outliers regarding the global rotation. Although these can be at worst the inverse of the true value, the outliers occur very little. As shown in Figure 5.25, the non-outlier predictions are very close to the true rotations. The model chosen also has had these minimised as much as possible to reduce the likelihood of encountering one of these outliers.

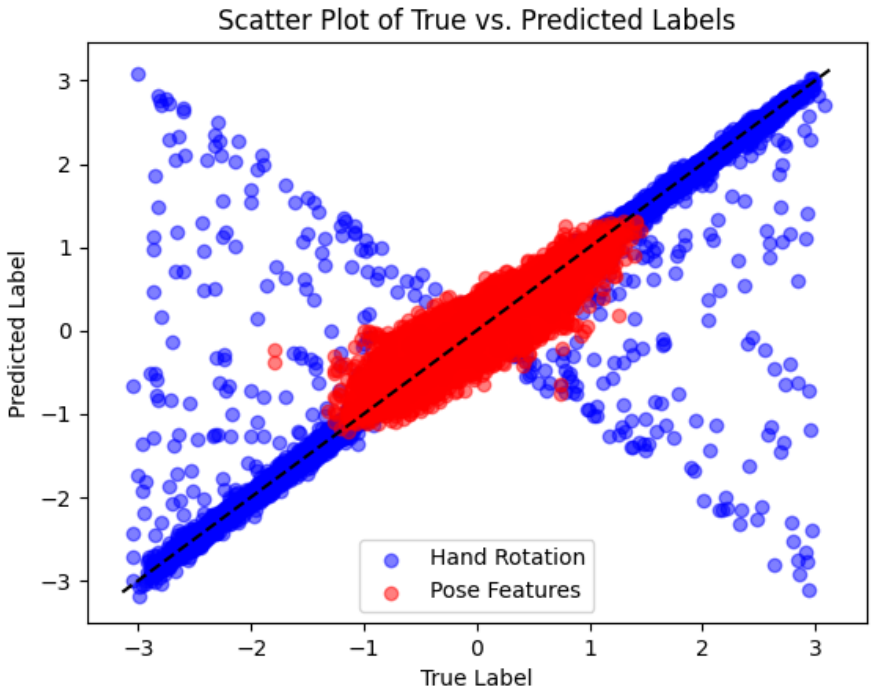


Figure 5.24: Results of Final Model on Test Set - Pose Estimation

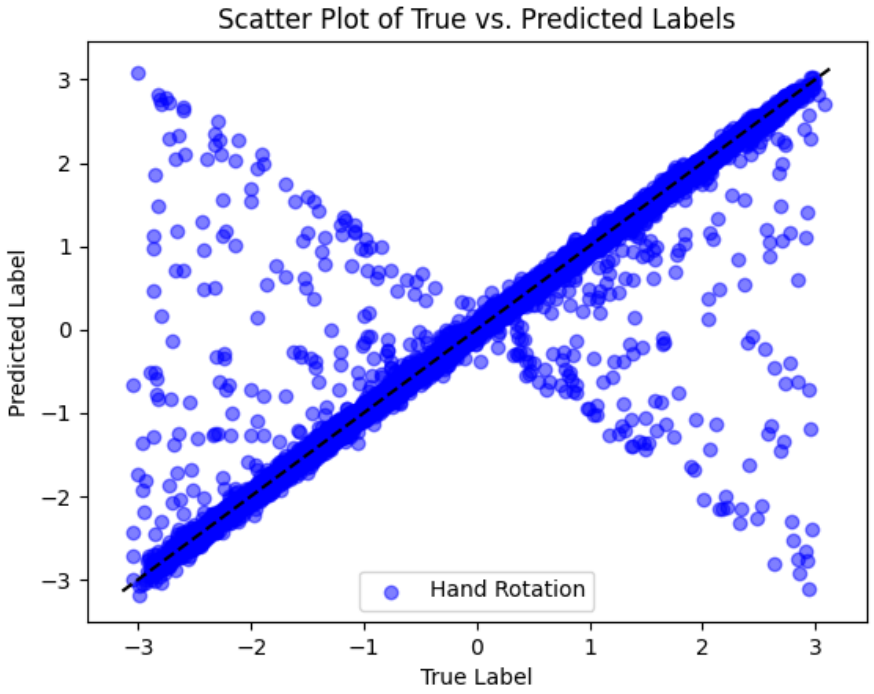


Figure 5.25: Global Rotation Results of Final Model on Test Set - Pose Estimation

Some examples of the results can be seen in Figure 5.26. As can be seen in these examples the model can predict a pose accurately, giving a final pose that is very close to the inputted hand.

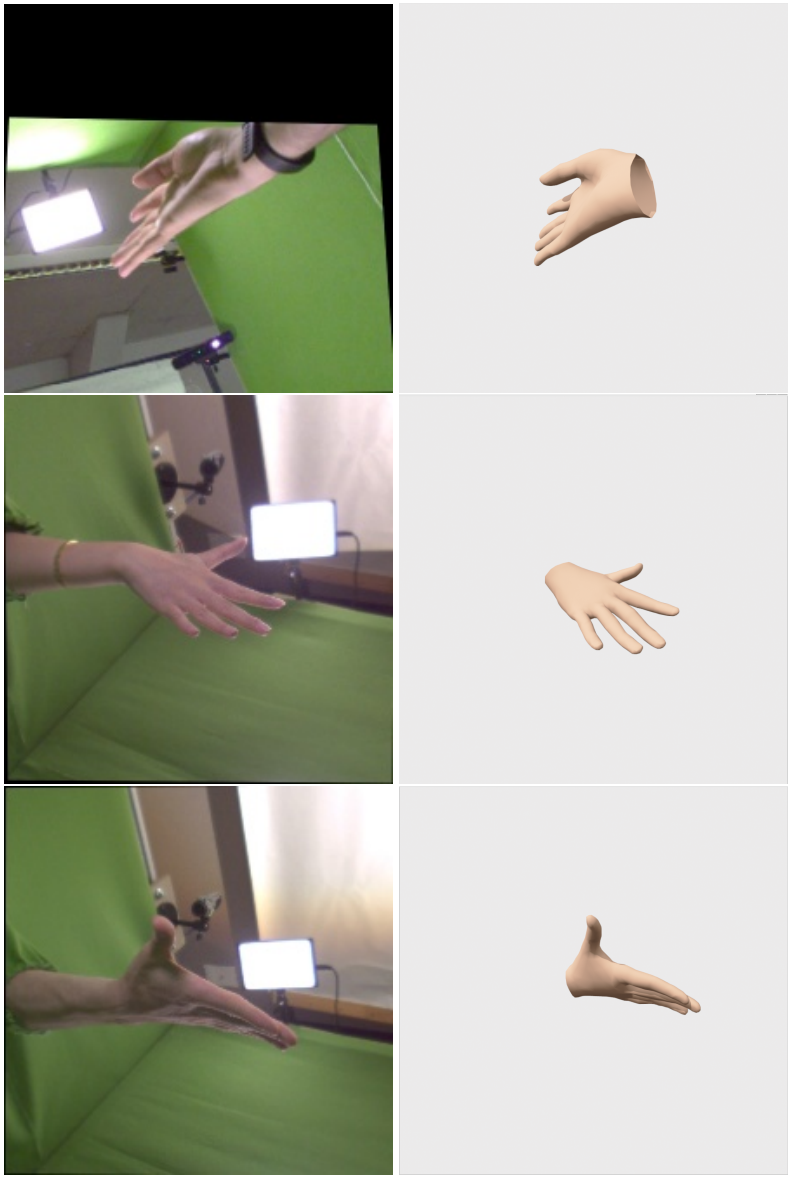


Figure 5.26: Example Pose Estimation Results - (input on the left, output on the right)

The model does sometimes bend fingers more than it should, or have larger gaps between fingers than in the input image. Some examples of this can be seen in Figure 5.27. The poses however are still very close to the original hand despite these errors. Furthermore, as can be seen in Figure 5.28, these errors are present in the true poses taken directly from the dataset, which subsequently transferred to the final trained model. The estimated poses and the true poses taken from the dataset are highly close further indicating this is an issue with the data used in training and not the method implemented to estimate poses. Therefore, to fix and improve the outputs of the model, higher-quality annotations would need to be collected.

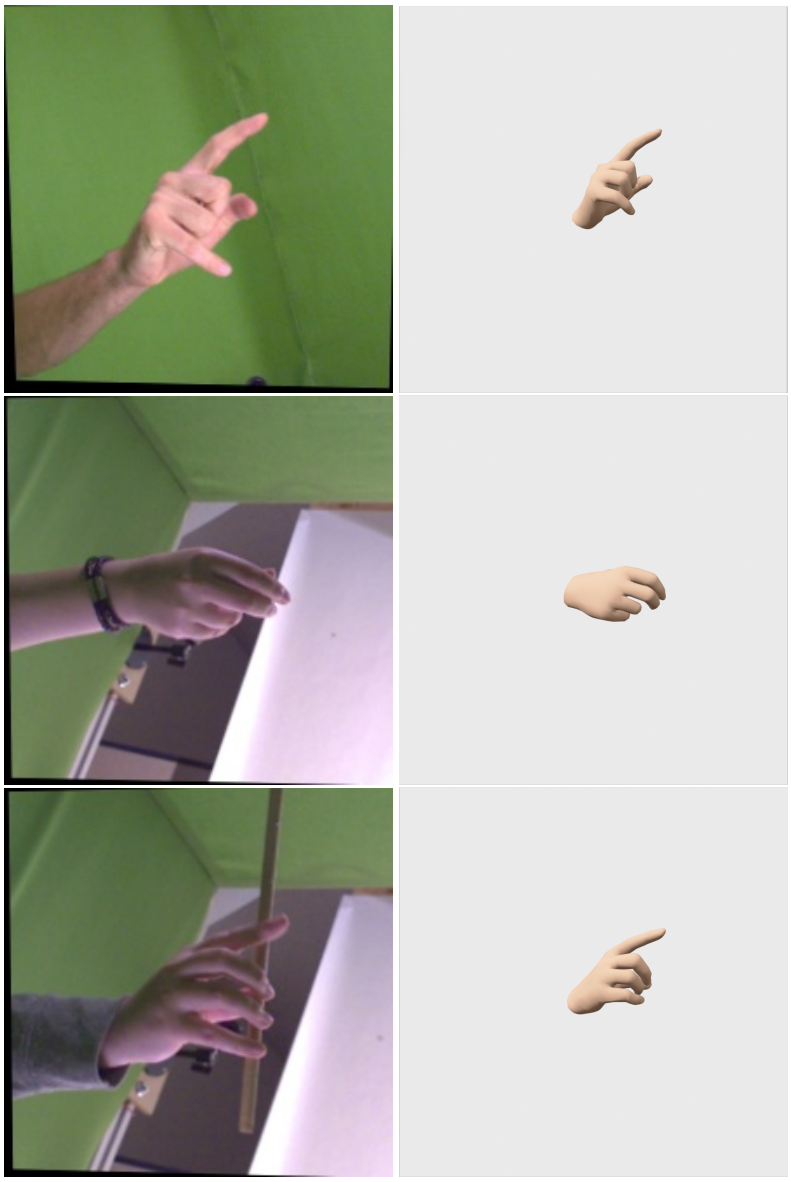


Figure 5.27: Example Pose Estimation Results with Extra Bends in Fingers - (input on the left, output on the right)

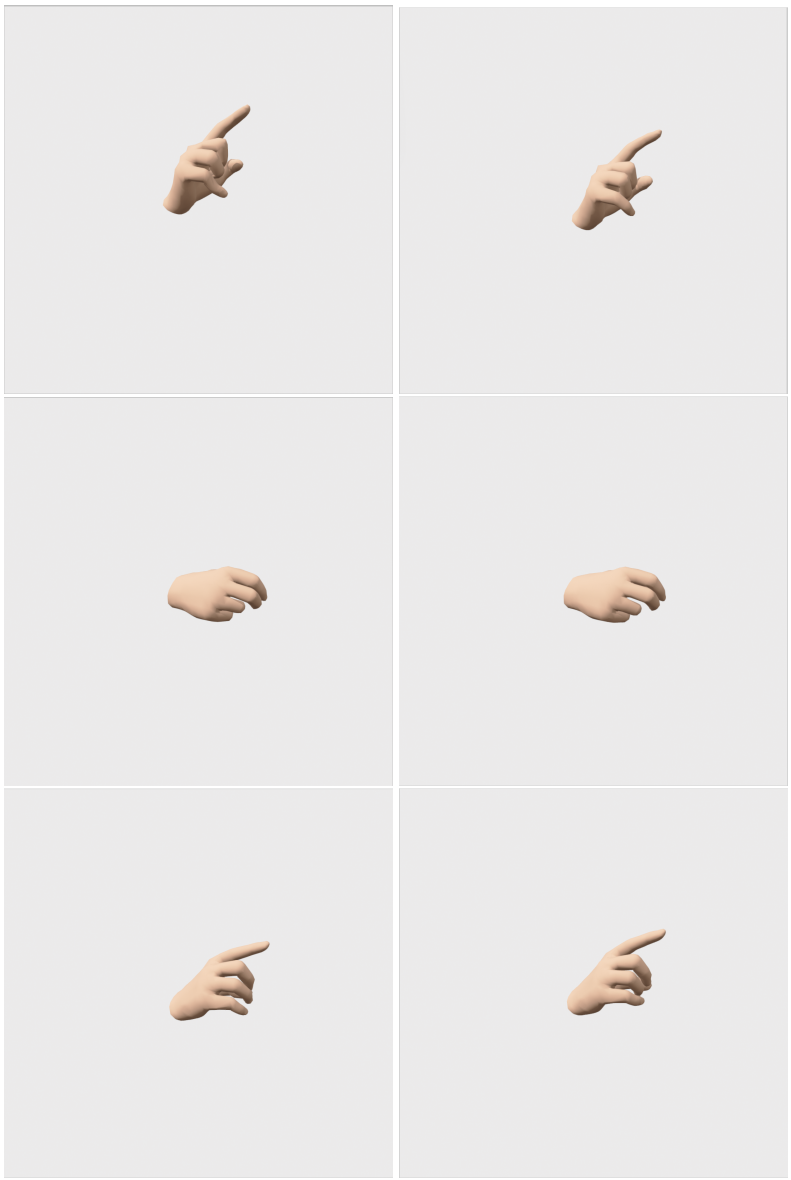


Figure 5.28: Pose Estimation Results compared with True Pose - (true pose on the left, estimated pose on the right)

5.4 Overall Pipeline

In general, the total pipeline is capable of detecting hands well in a large range of cases. The final output is consistently accurate, with the most important interaction between the keypoint detection and pose estimation steps yielding high-quality results. Some examples of this are demonstrated in Figure 5.29 which shows captures from the visualisation of the pipeline. The pipeline also rarely has any false triggers of the keypoint detection step and infrequently sends the wrong hand label to the pose estimation step.

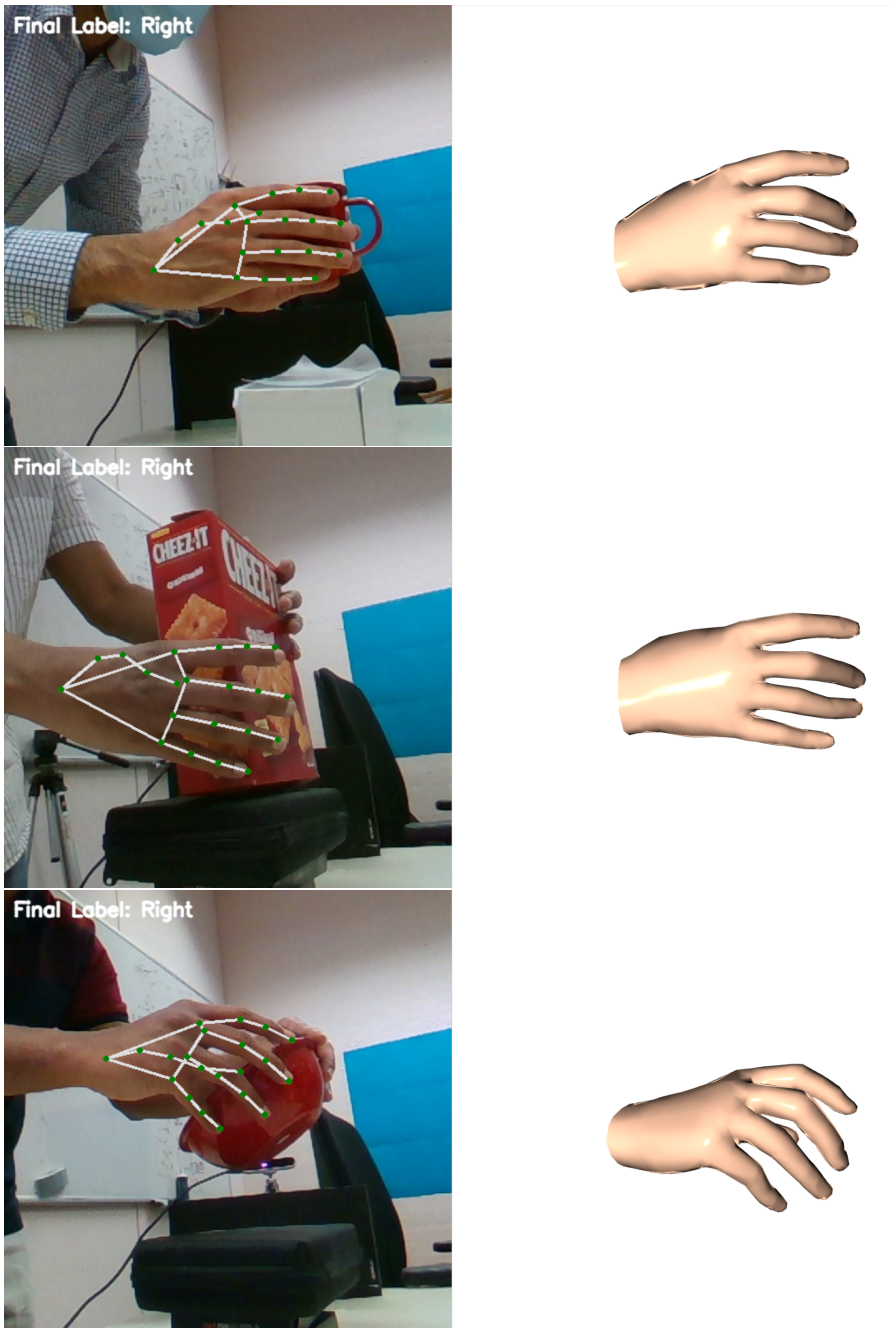


Figure 5.29: Example Results of the Pipeline

The pipeline is capable of running on a single input image and can be used to analyse a video in real-time, with hand detection, keypoint detection and pose estimation all happening smoothly as well as the visualisation.

The results given from the pipeline from the various steps are also highly consistent between frames. There is minimal variability between frames besides the actual movement, in other words, the results are largely smooth across a sequence of consecutive

frames. Nevertheless, there could still be room for improvement, currently, the keypoints for any given frame are calculated independently of any other frame, such that it does not take into account the previously predicted pose. An improvement could be introduced here by having the pipeline use the previously detected keypoints to inform the keypoints detected in the current frame. There could be some form of weighting added to the latest detected keypoints, resulting in a keypoint prediction that is somewhere between the latest and previously detected keypoints. In doing this, the keypoints should become smoother between frames, resulting in a smooth final product in consecutive frames.

5.4.1 Limitations

During the testing of the pipeline, some limitations became clear. The following subsections highlight some key limitations found, why they may have occurred and possible ways to resolve them in the future.

5.4.1.1 Performance on Dark Skin

During the testing of the pipeline, it has been noticed that the hands of people with a darker skin tone have noticeably worse detected keypoints and therefore worse 3D poses when compared to those of people with fair skin. I hypothesise that this may be due to the lack of diversity in the datasets used to train the keypoint detection. Freihand had hands solely of people with fairer skin and in HOnnotate there is a slightly wider range of skin tones present, but there is still a distinct lack of dark-skinned hands present in the dataset. The reason why the colour augmentation may not have covered this could be due to specific visual differences based on the darkness of the skin, which were unable to be picked up by said augmentations.

A lack of diversity in training has been proven to decrease the performance of machine-learning models when tested on racially diverse populations. For example, the “Gender Shades” project [6] found that when it came to various face detection machine learning models, the models consistently performed better on lighter skin with a large margin. The way to mitigate this is clearly to have a more diverse dataset. From research there are very few datasets with a large diversity, therefore any future work that would like to fix this would most likely need to create a new dataset with a large diverse set of hands. Alternatively, more experimenting with data augmentations could improve on areas not picked up by the model.

5.4.2 Image Aspect Ratio

While testing, the model would sometimes perform worse for keypoint detection on some aspect ratios. It appears to work better on a square aspect ratio than on more rectangular aspect ratios. This is most likely due to the keypoint detection model using datasets of only square images. This was done to ease the process of training due to having a set deadline and experiencing problems with PyTorch when trying to use images of different sizes.

Although the hand detection model was also trained on only square images, it does not have as large a problem in this aspect. Transformations applied to images during training may have helped avoid this problem - however, these transformations could not be used for the keypoint detection step.

This issue could be remedied by collecting images with transformations applied with annotations. However, this is not a large issue to cause major concern.

Chapter 6

Conclusion

6.1 Summary

The final pipeline I have created has achieved the expectations of my initial aims with the project. It can successfully take a frame through the process of detecting a hand, mapping keypoints, and finally posing a 3D model. It is capable of processing images through the pipeline and producing high-quality outputs. Although there are limitations with the pipeline as described in the evaluation chapter 5, it can offer a new starting point for future work.

6.2 Summary of Achievements

6.2.1 Hand Detection

The hand detection step has been trained to achieve an exceedingly high accuracy at 89.57%. It can differentiate right and left hands to a high degree, as well as detect hands in many different situations.

6.2.2 Keypoint Detection

The keypoint detection step has been trained to a low error of 0.0331. The model can detect the keypoints of hands with high precision in a high number of cases. It is also able to handle cases in which occlusions occur, both in situations where the occlusion is by an object and when the hand occluded part of itself.

6.2.3 Pose Estimation

The pose estimation has also been trained to have a low error, here being 0.0832. The model can pose a wide range of hands, being able to pose hands of both simple and complex hands.

6.2.4 Pipeline

The overall pipeline can work effectively as intended. The different parts interact with each other effectively, in particular, the poses estimated from the detected keypoints are consistently accurate. The pipeline is also able to run efficiently, making the predictions at each step in real time with minimal delay.

6.3 Proposed Future Work

As highlighted in the evaluation there are certain areas that I believe could be improved upon in future work, as well as some extra steps that may be considered to improve the pipeline.

For the hand detection model, currently, the model searches a whole frame and can only predict a singular hand. This is good for testing purposes, but future work may want to be able to predict on multiple hands within a frame. A future implementation could take a segmentation approach such that it segments any parts of the frame where a hand is detected. These segmented parts may also improve the efficiency of the keypoint detector by having zoomed-in hands. Also as mentioned in 5.1.3, the model could benefit from an expansion of the null dataset, specifically with bodies to reduce false positives.

For the pose estimation, one main extension would be to implement a left-hand version. As previously stated, I was only able to procure annotations for the right-hand MANO model, so only a right-hand pose estimation model was created. If any future work could get left-hand annotations, this extension would expand the potential scope of any possible uses of the pipeline.

As discussed in 5.4, the pipeline could be improved in the keypoint detection stage by considering previous frames. As mentioned in the evaluation, by considering previous frames, the keypoints detected across consecutive frames can become smoother and more consistent with each other.

In terms of datasets as mentioned in 5.4, a wider diversity, specifically in skin colour, would benefit the pipeline a lot. A wider diversity can only be a positive as it would

improve the accuracy of the model in both the hand detection and keypoint detection steps on wider diversity, which in turn would enhance the overall final output.

Another way datasets could be improved, as mentioned in 5.4, is by having labelled images with transformations applied. As explained in the evaluation, this could help improve the keypoint detection model, which seems to prefer images similar to the inputs specifically in terms of aspect ratio.

A final extension for the datasets, as mentioned in 5.3.3, is to acquire higher-quality annotations for the pose features. As explained in the evaluation, the current annotations used to train the pose estimation step are in some cases of a low quality, resulting in poses with incorrect bends in fingers. Although these bends are of minimal difference to the input hand, having better annotations would still result in an improved model regardless of whether or not this would only be a minor improvement.

In a future application, the pipeline could have the 3D hand model interact with the original image. The way it is currently, the 3D model is displayed in a separate window. A more interactive application though could have the 3D model overlayed onto the original image. This could further demonstrate the accuracy of the pipeline and better connect the final posed 3D model with the original input.

Bibliography

[1] URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker.

[2] URL: https://www.ibm.com/topics/convolutional-neural-networks.

[3] URL: https://commons.wikimedia.org/wiki/File:ResNet50.png.

[4] Yuki M. Asano et al. *PASS: An ImageNet replacement for self-supervised pretraining without humans*. 2021. arXiv: 2109.13228 [cs.CV].

[5] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[6] Joy Buolamwini and Timnit Gebru. "Gender shades: Intersectional accuracy disparities in commercial gender classification". In: *Conference on fairness, accountability and transparency*. PMLR. 2018, pp. 77–91.

[7] Alex Clark. *Pillow (PIL Fork) Documentation*. 2015. URL: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

[8] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[9] Shreyas Hampali et al. *HOnnotate: A method for 3D Annotation of Hand and Object Poses*. 2020. arXiv: 1907.01481 [cs.CV].

[10] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[12] Franziska Mueller et al. "Real-time Hand Tracking under Occlusion from an Egocentric RGB-D Sensor". In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2017. URL: https://handtracker.mpi-inf.mpg.de/projects/OccludedHands/.

[13] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[14] Javier Romero, Dimitrios Tzionas, and Michael J. Black. "Embodied hands: modeling and capturing hands and bodies together". In: *ACM Transactions on Graphics* 36.6 (Nov. 2017), pp. 1–17. ISSN: 1557-7368. DOI: 10.1145/3130800.3130883. URL: http://dx.doi.org/10.1145/3130800.3130883.

[15] Tomas Simon et al. *Hand Keypoint Detection in Single Images using Multiview Bootstrapping*. 2017. arXiv: 1704.07809 [cs.CV].

[16] Adrian Spurr et al. *Weakly Supervised 3D Hand Pose Estimation via Biomechanical Constraints*. 2020. arXiv: 2003.09282 [cs.CV].

[17] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[18] David Tellez et al. "Quantifying the effects of data augmentation and stain color normalization in convolutional neural networks for computational pathology". In: *Medical Image Analysis* 58 (Dec. 2019), p. 101544. ISSN: 1361-8415. DOI: 10.1016/j.media.2019.101544. URL: http://dx.doi.org/10.1016/j.media.2019.101544.

[19] Fan Zhang et al. *MediaPipe Hands: On-device Real-time Hand Tracking*. 2020. arXiv: 2006.10214 [cs.CV].

[20] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[21] Yuxiao Zhou et al. *Monocular Real-time Hand Shape and Motion Capture using Multi-modal Data*. 2022. arXiv: 2003.09572 [cs.CV].

[22] Christian Zimmermann et al. *FreiHAND: A Dataset for Markerless Capture of Hand Pose and Shape from Single RGB Images*. 2019. arXiv: 1909.04349 [cs.CV].