香 港 大 學

THE UNIVERSITY OF HONG KONG

Final Report

Hand Gesture Recognition System

Author: Dhruv Agrawal

UID: 3035344405

Supervisor: Dr. K.K.Y. Wong

Second Examiner: Dr. A.T.C. Tam

Course: COMP4801 Final Year Project

Date: 29 April 2020

Abstract

The use of hand gestures is an active area of research in Human Computer Interaction. The construction of Hand Gesture Recognition Systems and Hand Pose Estimation Systems has become essential with this interest in new means of HCI. In this study, a pipeline composed of a hand pose estimator followed by a hand gesture classifier is employed to perform Hand Gesture Recognition. Current state-of-the-art models are not portable or applicable in online condition due to complex input and long execution times respectively. This study aims to show that the joint locations from hand pose estimation can aid the classification process alleviating the aforementioned problems by inferencing a larger amount of information in a smaller network. Secondly, an application feasible for online use is also be developed for better user experience. The hand pose estimator, called *HandNet*, uses a novel branched architecture to maximize the amount of contextual information available to the deeper layers of the network. The output of the pose estimators is then passed on to a pose-to-gesture classifier. Due to the comparatively smaller input dimension, the pose-to-gesture classifier can be much simpler and execute in a small time. In our study, we experiment with K-means clustering and Random Forests. Our tests show that Random Forests perform the best with test accuracy of 99.6% on the MSRA dataset.

Acknowledgements

I would like to take this opportunity to express my gratitude to my final year project supervisor, Dr. K.K.Y. Wong, without whose support this project would not have been feasible. I would also take this opportunity to thank Dr. A.T.C Tam for acting as my second examiner and providing inciteful comments. Furthermore, I thank the department of Computer Science for accepting this project proposal and for providing the GPU farm servers required for the development of this project.

Contents

List of Figures

List of Graphs

List of Tables

List of Acronyms *(In alphabetical order)*

AI	Artificial Intelligence
CNN	Convolutional Neural Network
DIP	Distal Interphalangeal
DNN	Deep Neural Network
FCC	Fully Connected Component
GPU	Graphical Processing Unit
GUI	Graphical User Interface
MCP	Metacarpophalangeal
PIP	Proximal Interphalangeal
RGB-D	Red, Green Blue & Depth
RNN	Recurrent Neural Network

Chapter 1: Introduction

1.1 Background: Brief overview of Hand Pose Estimation Systems

Hands are our most important tools when it comes to interacting with the world. Even in a quiet library, two people are able to communicate with little constraint on their communicational facilities due to an extensive use of hand gestures that garners for the lack of verbal conversation. Hand Gestures also see applications in presentations and conversing in presence of a large language carrier. Perhaps the most significant application of effective hand gestures is in Sign Language[19]. With a clear distinct set of hand gestures, sign language allow conversing with people with auditory impairment. Thus, we clearly see that humans use hand gestures to convey a large amount of information. This means that in order to achieve Artificial Intelligence (AI) that is capable of seamlessly interacting with humans, it is vital that we are able to encode our inherent ability to convey information with the use of our hands into the AI. A system allowing for such information inference is called a Hand Gesture Recognition System[2][3].

In order to achieve a marketable Hand Gesture Recognition System, it is essential to first create a system that allows the computer to detect, and regress the position and orientation of a hand from a camera input. Such a system is called a Hand Pose Estimation System[2]. This system will be capable of analyzing a video input and regressing the 3D locations of all the palm and finger joints in the video stream[3]. It has invited a large amount of interest and a proportionate amount of research has been conducted into developing such a system.

1.2 Background: Brief overview of Hand Gesture Estimation Systems

The current research community often uses Hand pose estimation and hand gesture recognition interchangeably. Hand pose estimation is almost always used to refer to regressing the location of hand joints[2]. However, the term hand gesture is sometimes used as a synonym for hand pose[18] and some other times used to refer to the class or label assigned to a particular posture[2]. In this study, we refer to the latter meaning for hand gesture.

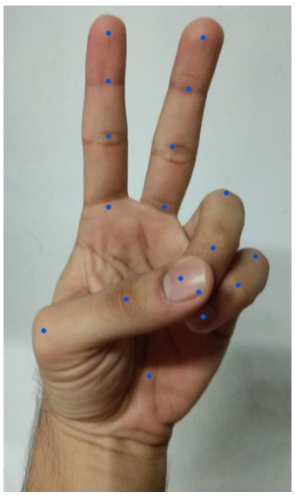
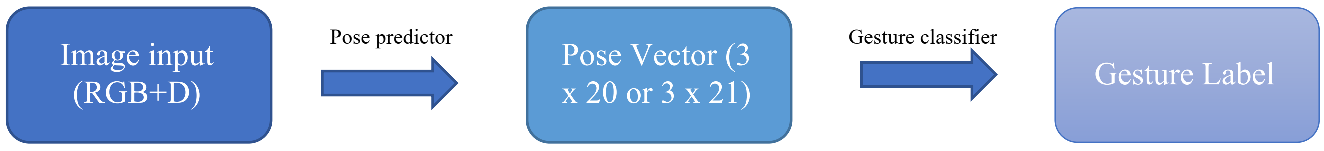


Figure 1.1 Hand gesturing the number 2

For example consider the pose in Figure 1.1. In this study, hand pose estimation refers to locating the hand joints, shown as blue dots in the figure. In contrast, labeling the image as “2” is referred to as hand gesture recognition.

With this differentiation, one can see that hand gesture recognition is indeed a harder problem when compared to hand pose estimation. Hand gesture recognition not only involves understanding the configuration of the hand in the image but also to infer meaning from this configuration. Due to the variability of hands caused by the different sizes and shapes, a single gesture can appear very different. Therefore, a hand gesture recognition system has to be invariant of these differences and infer meaning from a generalized model of a hand. When discussing hand gestures, there are again two types. Firstly, the gesture can be static (Fig 1.1). Such gestures can be depicted through a single image[6][8][9][10]. The other type of hand gestures are dynamic [4][5][7][12]. This means they involve some form of hand motion in order to convey the meaning. For example clapping or waving. Due to the motion involved, a single image is no longer sufficient and we use videos to capture these gestures. Since dynamic gestures add motional and temporal complexities into the system, classification of such gestures even is rendered more challenging. Lastly, due to the change in file format, even storage limits become actual concerns. This leads to finding a balance between the dataset disk size and the quality of the videos when constructing the videos.
In this study, a system built for hand pose estimation is used to aid the hand gesture recognition by providing a normalized version of hand pose to the recognition system. It is believed that normalization of hand pose can reduce the variability of hands. This should in turn improve the performance of the classifier since it has to learn lesser amount of information about hand poses. We use a Random Forest

as gesture classifier. Unlike most gesture classifiers, we use pose vectors generated by hand pose estimation system to perform the classification. This helps us significantly reduce the amount of input data required by the classifier. Finally, we believe that using a pose vector might be beneficial because it reduces the redundancy in training hand pose estimation and hand gesture recognition and also removes the noise being forwarded to the classifier. Our final pipeline is illustrated in the graph below.



Graph 1.1: Proposed Pipeline

1.3 Objectives, Approach and Deliverables

This project attempts to build a Hand Gesture Recognition System with two key features. First, the project aims to minimize the amount of input data required by a Hand Gesture Recognition System. Therefore, the project focuses on using a single RBG-D sensor as the input source. The project posits that using a strong Hand Pose Estimation model will allow the complete system to recover from a lack of information available. Secondly, the project aims to find evidence of feasibility of using a hand gesture recognition system in online conditions. This requires that the total execution time of the algorithm is sufficiently low to avoid a noticeable lag in any hand gesture based application.

The deliverables of this project will include a Graphic User Interface (GUI) application that allows for user-friendly interaction with the underlying machine learning models. Furthermore, this project will also deliver a novel machine learning architecture that can perform hand gesture recognition effectively.

1.4 Outline of the report

The report is divided into 6 chapters. Chapter 2 discusses related works that have been conducted related to Hand pose estimation and hand gesture recognition. We discuss Hand Pose Estimation Models such as *DeepPrior++* and *CrossInfoNet* and Hand Gesture Classifier as presented in [3]. It also features other implementations of Hand Gesture classification including an application involving finger writing in mid-air.

Next, Chapter 3 presents the Methodology. It discusses the dataset choices, system architecture and performance measurement heuristics. It focuses on the branched architecture of *HandNet* and the common feature extraction network that allows for faster executions. The second part of the section focuses on predicting gesture labels from provided pose vectors. We define the K-means clustering model and Random Forest model being used. We then discuss the advantages of using a pose vector instead of an image as the input.

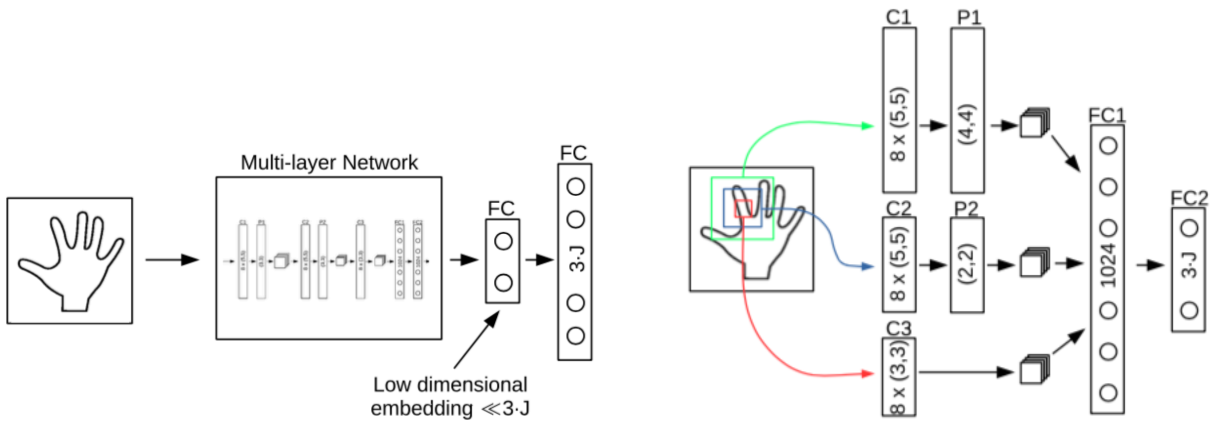
Chapter 4 focuses on some of the results gathered by this study. We first discuss the differences between some of the implementations and then we look average loss, average execution time, average joint error of our implementations. We also discuss the difference introducing a common feature extraction layer has on the total execution time. We then discuss the performance of our final proposed pipeline. We achieve impressive accuracy of 99.6% over MSRA dataset using *CrossInfoNet* model coupled with a Random Forest. Lastly, it also includes discussion of challenges and limitations faced by this study and the current status of the project.

Chapter 5 lists some recommendations for future work related to pose-to-vector prediction paradigm. The final chapter, Chapter 6 Conclusion, provides a summary to this report highlighting the important aspects of this project.

Chapter 2: Related Works

2.1 Hand Pose Estimation

Khan et. al. [1] provides a thorough literature review of Hand Pose estimation techniques that do not employ Deep Learning Algorithms. [1] divides the process of hand gesture estimation into three key steps: 1) Extraction Method; 2)Feature Extraction; and lastly 3)Classification. Extraction refers to extracting the hand segment from the image background. This allows the system to isolate the hand and remove the noise introduced from the background textures. Next, feature extraction refers to extracting contextual information such as Centre of Gravity, Hand Contours and Silhouettes, and Fingertips position. Lastly, Classification refers to using the extracted features to make a prediction of the depicted hand gesture in the image. Different classification systems greatly affect the overall accuracy of the complete system. Some of the frequently employed non-deep learning models include Fuzzy C-Means Clustering and Genetic Algorithms [1].



(a) (b)

Figure 2.1 Architecture of the Deep-Prior model from [2]. (a) depicts the complete architecture of the primary deep learning model and (b) shows the structure *ORRef* Figure taken from [2]

However, with the recent increase in the popularity of the Deep Neural Networks(DNNs), a majority of the Hand Pose Estimators attempt to use CNNs for performing the aforementioned Extraction, Feature Extraction and Classification. This decreases the amount of the data preprocessing carried out by the researchers and also allows for the model to regress its own data processing parameters with low exterior intervention. In such a scenario, researchers primarily focus on the different architectural features of the DNNs. One of the most popular such architecture was introduced by Oberweger [2] called *Deep-Prior*. This architecture has two salient features. First, they use a pinch layer as a second last layer of the network. This can be seen in Figure 2.1 (a) when the model moves from multi-layer network to FC. They argue that since the latent output space is not purely three dimensional, a pinch layer allows the network to learn a lower dimensional embedding intrinsically. This then helps increase the accuracy of their prior network. Next, they train several refining networks, one for each joint, called *ORRef.* This is visualized in Figure 2.1 (b). These networks use overlapping crops, centered at the join location regressed by the first network, of the original input image. This network refines the regressed joint location and can be applied multiple times in order to achieve a more accurate result.

More recently, Du et al [17] theorized that by decomposing the task into palm pose estimation and finger pose estimation, we can use cross information to improve the performance of the model Therefore *CrossInfoNet*, split the task into the two subtasks and trains a separate subnetwork for each task. Since the two tasks are mutually distinct, [17] argues that the noise in one of the network will act as valuable information in the other network. Therefore, after performing a primary feature extraction they propose the two subnetworks exchanging their noise information and use this noise in further feature extraction.

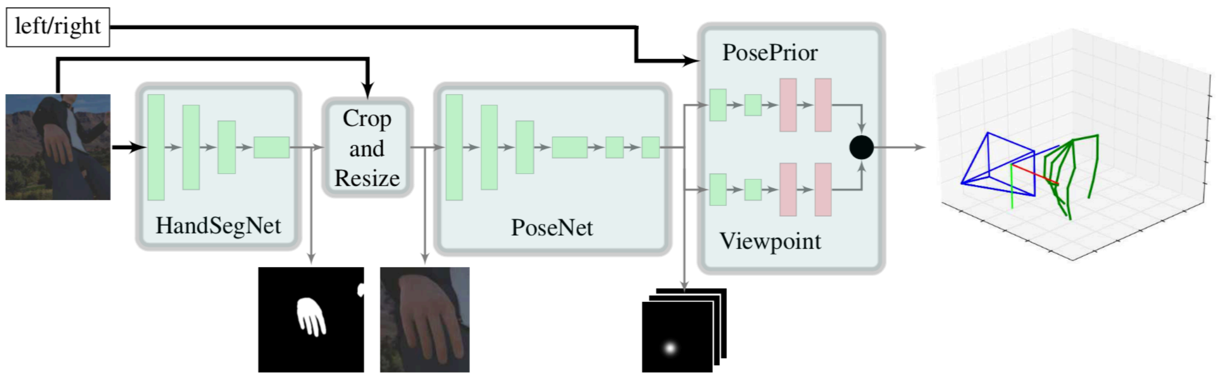


Figure 2.2 Architecture of the model proposed in [22]. Taken from [22]

[22] divide the pose estimation task into three discrete steps. The first subnetwork is responsible for creating a mask of the hand (see figure 2.2). Once the complete hand has been detected, they crop and resize to the area in the image corresponding to the mask[22]. This allows them to remove a large amount of background noise in the image. The cropped image is then processed by the *PoseNet* [22]. It is responsible for detecting key points in the image required for creating the pose. Finally, an identical stream estimate canonical coordinates and the viewpoint.

2.2 Hand Gesture Recognition

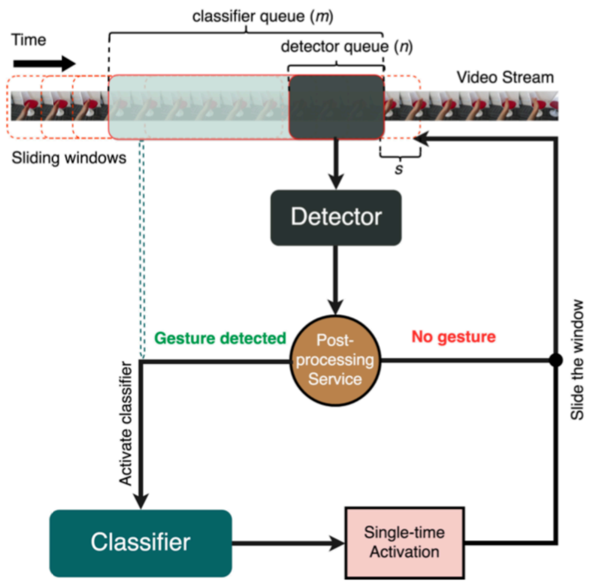


Figure 2.3 Architecture of the model proposed in [3]. Taken from [3]

Köpüklü[3] focuses on creating a model that is suitable for online applications. It uses an online video stream as input. In order to avoid the problem of running a costly hand gesture recognition system continuously over the video stream, [3] uses a *detector* and *classifier* combination. Refer to Figure 2.2 for the architecture of the model. The *detector* is a lightweight DNN that is executed continuously over the video stream in order to detect a hand gesture. If a hand gesture if detected, it then activates the *classifier* network that takes in a larger section of the video as the input as makes the prediction of the hand gesture depicted in the video stream. The detector network is able to achieve 98+% recall rate of EgoGesture Dataset[3][4][5].

Chang et al. [15] instead attempt to limit the size of the problem by focusing only on the fingertip instead of the complete hand. They perform frame by frame fingertip detection. By detecting the fingertip of the index fingertip in each frame, they create a trajectory of the movement of the fingertip. They then use this trajectory to predict the letter written in air by the use. Markussen, et al [16] build a word-gesture keyboard that can type by detecting word shaped gestures in midair. They adapt already existing touch based word-gesture algorithms to work in mid- air. They then projects users' movement onto a display and use pinch as a word delimiter. This approach greatly reduces the total number of unique hand gestures that the model has to learn. Since the method is based on locating only the fingertip on index finger, this approach differs from more traditional hand gesture classification approaches.

For implementation of *HandNet,* we use a modification of *DeepPrior*++ [2] as a base comparison model. Similar to the idea in [3], we aim to minimize execution time by reducing the execution time of the main network by preprocessing the input. [3] achieves this by using a combination of detector and classifier, in our implementation, we use a combination of hand pose regressor and hand gesture classifier to reduce the input size to the main network.

2.3 Summary

Current state of Hand Gesture Recognition suffers from limitations on portability and execution times. A number of papers attempt to alleviate these limitations by reducing the complete problem set. [2] proposes using a pinch layer for allowing a more natural embedding to be learnt. On the other hand [3] reduces processing time by breaking the complete network into a light and heavy network.
For our implementations, we use [2] as a basis for comparison and build upon subnetwork approach of [3].

Chapter 3: Methodology

3.1 Introduction

In this section, we first discuss the details of the datasets used and the architecture of the machine learning model. Next, we discuss the heuristics used in order to assess the performance of the different models. Towards the end of this section, we discuss the implementation details and the proposed project timeline and milestones.

The data available to a machine learning model is the determining factor in the final performance of the model. Therefore a large amount on research was conducted in deciding which publicly available datasets were used in the implementations of the architecture. The key factors affecting the choice of a dataset include dataset size, hand pose variability and image quality of samples. Based on these factors NYU hand pose dataset [6] and NVIDIA Dynamic Hand Gesture Dataset [7] were selected as the primary datasets of choice. Besides the aforementioned datasets, the architecture was also implemented on a number of other datasets. The details of these other datasets is discussed in the following chapter.

The architecture of the model is the key innovation of this project. This study theorizes that by transforming the process of hand joint regression into a hierarchical system, a better performance can be achieved with similar model sizes. This assumption is based on the fact that the position of each hand joint is not truly free from the position of other hand joints. For example, the position and orientation of the wrist almost completely decides the position of the finger bases. Finger bases in turn limit the area in which each finger joint can lie in. Therefore, by using a hierarchical approach the position of wrist joint can aid the model regress the position of the finger bases.

The models were implemented on the HKU GPU Farm that provides a NVIDIA GeForce GTX 1080 Ti GPU for the training of the models.

3.2 Overview of Datasets Available

Since the hand pose estimation problem has been made extremely popular due to the wide availability of good depth sensors, a large number of hand pose datasets are available. The datasets fall into two main categories: 1) Static images and 2) Dynamic videos [6][7][12]. The static images depict static hand gestures such as depicting numbers using fingers. On the other hand, dynamic videos provide hand gestures that require hand movement such as waving hand. While the static images offer the advantage of having a smaller file size and easier to generate, they suffer from a low number of real world applications. Since the biggest focus of the Hand Gesture Recognition is its implementation in

Human Computer interaction, it is essential that the recognition model can handle gestures that involve movement. This makes the dynamic video models more suitable for Hand gesture recognition.

However, dynamic videos have three primary drawbacks. Firstly, since the data is in form of videos, the file size of a single sample is much large than that of images. Due to storage limitation, severe down sampling and compression of the videos is required. This results in poor quality videos which effect the amount of information that the model can extract from the data. Secondly, shooting videos of hand gestures requires a large time. In contrast, static hand poses are captured by shooting a video of a static hand and extracting frames from the video feed. This results in the fact that most of the video datasets have a smaller number of samples when compared to static images datasets[4][5][8]. Thirdly, a machine learning model that is large enough to ingest the complete video file needs to have a very large input layer. This has a ripple effect and the sizes of the other layers are also large. This results in an enormous number of total parameters to be trained.

We now look at some of the important datasets used in this project:

3.2.1 **NYU Hand Pose dataset [6]:** This dataset contains 8252 test-set and 72757 training-set frames of captured RGB-D data with ground-truth hand-pose information. All the images comprise of various hand poses. For each hand pose, Kinect data from three different angles is captured. Finally, this dataset is presently well regarded among Hand Pose researchers due to the high variability of Hand Poses captured in this dataset. Therefore, it is used as the principle dataset in the first phase of the project that focuses on predicting the hand pose from a given RGB or depth picture.

3.2.2 **ICVL Hand Pose dataset [8]:** This dataset annotates 16 joint locations with (x,y,z). Coordinates available for each image. The x and y coordinates are measured in pixels while z coordinate is measured mm.

3.2.3 **MSRA Hand Pose dataset [9]:** This dataset contains images from 9 subjects' right hands are captured using Intel's Creative Interactive Gesture Camera. Each subject has 17 gestures captured and there are about 500 frames for each gesture. The biggest advantage of this dataset is that it is one of the few datasets that provides both the gesture labels and the hand pose coordinates for each sample. This is useful in the second phase of this project that focuses on using the pose parameters to predict the labels. Due to this unique characteristic, the MSRA dataset is used as the principle dataset in the second phase of the project.

3.2.4 **Multiview Hand Pose Dataset [10]:** This dataset captures hand pose from different angles. This dataset not only provides the 3D hand joint locations for each image but also provides the bounding boxes for the hands in the images.

3.2.5 **EgoGesture[4][5]:**This dataset contains 2,081 RGB-D videos, 24,161 gesture samples and 2,953,224 frames from 50 distinct subjects. The authors define 83 classes of static or dynamic gestures focused on interaction with wearable devices.

3.2.6 **NVIDIA Dynamic Hand Gesture Dataset [7] :** nvGesture is a dataset of 25 gesture classes, each intended for human-computer interfaces. The dataset has 1532 weakly segmented videos, which are performed by 20 subjects at an indoor car simulator. The dataset is then split into training (70%) and test (30%) sets, resulting in 1050 training and 482 test videos [13].

3.2.7 **BigHand 2.2M Benchmark Hand Dataset [12]:** BigHand dataset is one of the largest hand pose datasets available. It contains 2.2M images captured from 10 different subjects using kinematic 6D electromagnetic sensors[13].

3.3 Model Architecture: Hand Pose Estimation System

As discussed earlier, the performance of any model depends on the inherent architecture of the model. In this section we propose a novel architecture that uses the underlying hierarchy of the positions of the hand joints in order to aid the joint position regression process. In order to understand this approach, it is imperative to first understand the geometry of the hand. [14] describes this geometry very thoroughly. They compare the difference between a 3D coordinates representation of the hand joints against a hierarchical model designed by them. In essence, use of 3D coordinates to define the hand joint locations leads to loss of information regarding the relation between the hand joints. Therefore, each hand joint has to be located independently. However, such a situation is not entirely true. Since the joints are connected, there are constraints on the possible locations of one joint given the position of another joint. For example, consider Proximal Interphalangeal (PIP) joint, as shown in Figure 3.1 below.

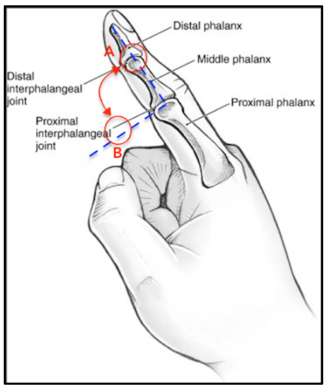


Figure 3.1. Representation of the finger joints

Clearly, after fixing the position of that joint, the Distal interphalangeal (DIP) joint can only take the positions shown by the red arrow in the figure. This means that the position of the DIP joint is bounded by locations A and B in the image since we fixed the position of the PIP joint. Given this observation, deduce that knowing the position of the base joints can aid the model to predict the location of the distal joints.

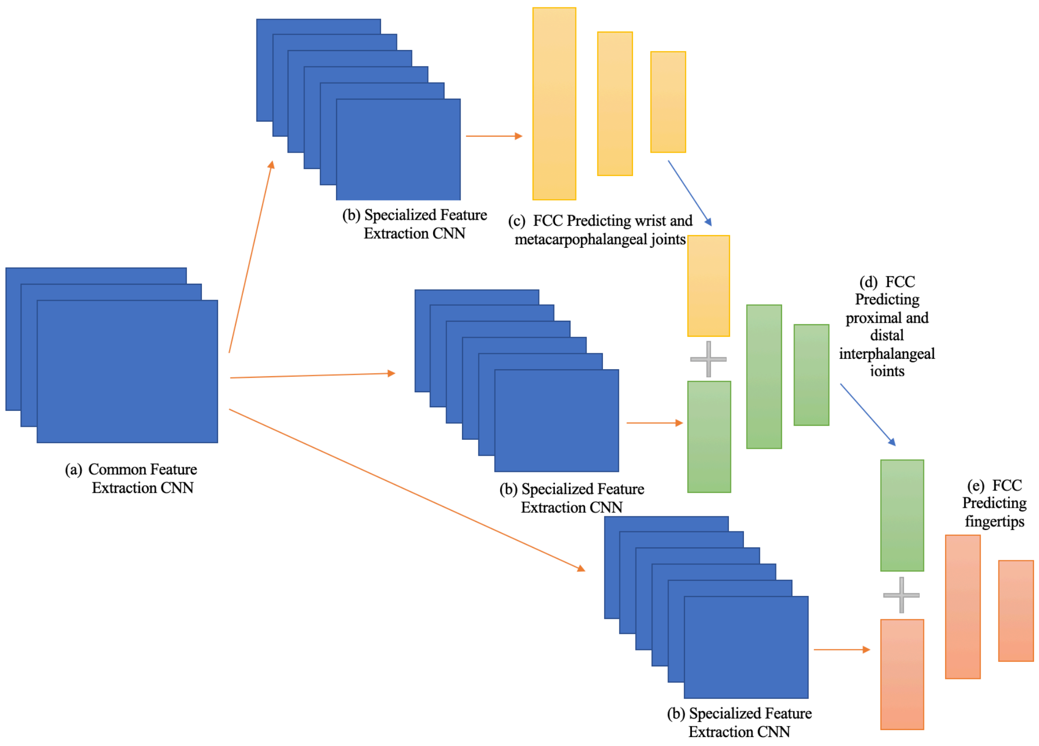

(b) Specialized Feature Extraction CNN
(c) FCC Predicting wrist and metacarpophalangeal joints
(d) FCC Predicting proximal and distal interphalangeal joints
(a) Common Feature Extraction CNN
(b) Specialized Feature Extraction CNN
(e) FCC Predicting fingertips
(b) Specialized Feature Extraction CNN

Figure 3.2. Proposed Model Architecture (a) This is a common feature extraction CNN. (b) These secondary CNN networks extract features related to their respective joints. (c) This Fully Connected Component makes prediction of the Wrist joint and the MCP joints. (d) This Fully Connected Component makes prediction of PIP joints and the DIP joints.(e) This Fully Connected Component makes prediction of the locations of the finger tips.

Based on this assumption, we design our model to predict the hand joints in three steps. In the first step, only the Wrist joint and the Metacarpophalangeal joints (MCP). In the second step, we predict the Proximal Interphalangeal joints (PIP) and the Distal Interphalangeal joints (DIP). Finally in the third step, we predict the locations of the finger tips. The summary of the network architecture is illustrated in the figure below.

The input is first passed into a common feature extraction layer. This layer is depicted by (a) in Figure 3.2. This layer is responsible for extracting features that are common for the next three steps. The main function of this layer is to reduce the time complexity and number of trainable parameters in the network. By making the starting extraction layer common for the three steps, the basic features are

extracted only once. If instead a completely independent network was trained for each step, the total size of the network will be closer to thrice the present size. This adds a significant time penalty in my testing. Hence, we instead use a common extraction layer first.
Next, we perform a branching in our network. The rest of the network is divided into three sub networks with similar architecture. This is illustrated as (b) in the figure. These networks extract the features that are more relevant to their respective sub-networks. Lastly, we have sections (c), (d) and (e) of the network. They regress the locations of (i) Wrist joint and the MCP joints, (ii) PIP joints and the DIP joints and, (iii) the locations of the finger tips respectively.
The earlier discussed idea of using the locations of base joints to predict the locations of distal joints is applied here. As illustrated in the figure, part (c) first predicts locations of Wrist joint and the MCP joints and next passes these locations to part (d). Part (d) uses this information in addition to the information received from its special feature extraction network to in turn predict the locations of PIP joints and the DIP joints. It then passes the locations of all joints that have been located to the last part , part (e). Part (e) then performs a function congruent to Part (d) to predict the locations of finger tips. This results in the regression of all the joints.

3.4 Model Architecture: Gesture recognition System

Given the plethora of models available for pose regression, it is desirable to use them perform gesture prediction as well. Since an original image has a lot of noise in terms of background and occlusions, we propose that hand pose vectors will serve as better inputs. This is due to key differences between images and pose vectors.

3.4.1 Advantages of pose vectors over images

The following are the advantages of using pose vectors we found in place of images.

3.4.1.1 Smaller input dimension

Firstly, the images have a much large input dimension (usually 480 x 640) compared to pose vectors (usually 3 x 20 or 3 x 21). This smaller dimension allows for faster learning for any model.

3.4.1.2 More invariants

Secondly, it is easier to manipulate pose vectors to have more invariants across samples. In particular, in this project, we perform conversion for x,y,z real world coordinates or u,v,d pixel coordinates to r, θ, z being hierarchical coordinates. In this system, we always make the wrist base to be (0,0,0). Then we calculate the bone vectors as shown in the figure below. By using bone vectors, we are able to regulate the length of each bone to match the average bone length over the NYU/MSRA datasets.

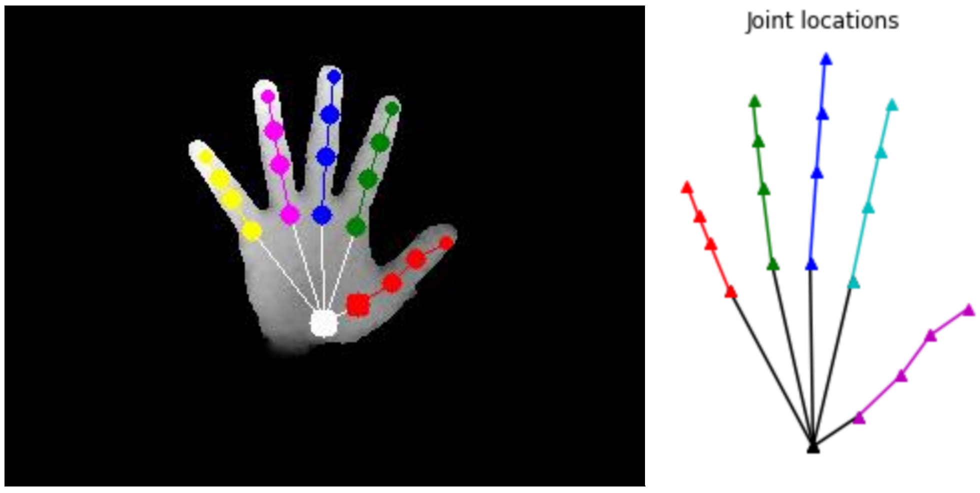


Figure 3.3: Bone vectors measured in r, θ, z coordinate system

The z measures the depth same as the x,y,z and u,v,d systems. Finally θ measures the angle between two adjacent bone vectors. This allows the model to more clearly illustrate the degree by which each finger joint is curled. For a given pose, we measure r, θ,z coordinates for 20 different bones. In our testing, these 20 bones serve as ample information to predict the gesture. However [] also experiment with 23 and 25 bones which are also regularly used models.

3.4.1.3: No occlusions and Camera Viewpoints

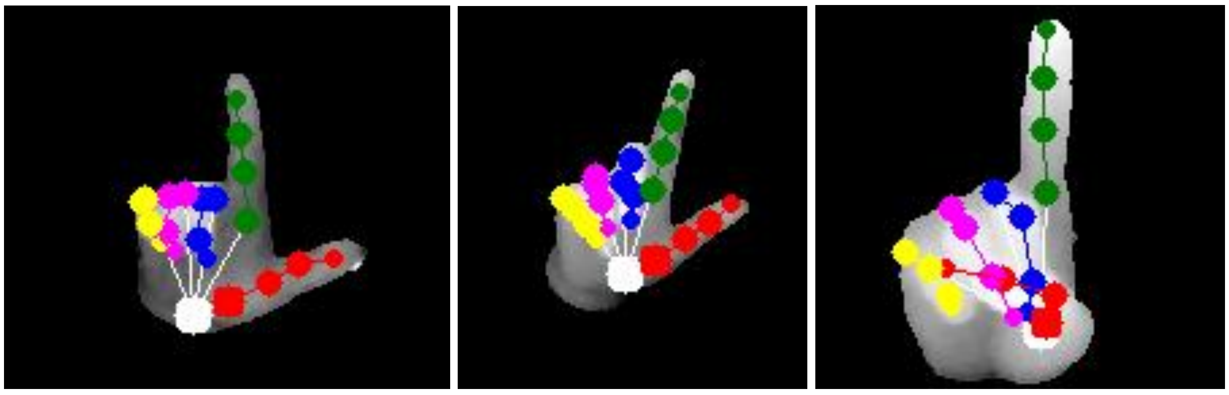
Finally, the last advantage of the pose vectors is that they are free of camera viewpoints and occlusions. This is because, for every sample, the pose vector always contains locations of all bones. This is very different from images where few joints maybe completely hidden due to the camera viewpoint and thus have to be completely estimated by the model. This serves as the largest advantage of using hand pose vectors. It allows the gesture prediction model to completely offload the learning process of working with occlusions to the hand pose regressor. This makes training the gesture predictor much simpler. The rationale behind this choice is that a hand pose regressor necessarily needs to learn how to deal with occlusions. To redundantly force a gesture predictor to learn to work with occlusions only results learning the same information twice. By using pose vectors, this need completely disappears. Furthermore, this allows for simpler machine learning models to be used as gesture predictor.

In this project, we experiment with two models for gesture prediction: 1) K-means Clustering and 2) Random Forests. The two models give very different results that will be discussed in chapter 4: Results. Both the models were chosen for their particular characteristics as discussed below.

3.4.2 Different Models Used

3.4.2.1 K-means Clustering models

K-means clustering method forms has two main characteristics that made it a good candidate as gesture predictor. Firstly, it can easily create required number of clusters. Therefore, in this project we always keep the number of clusters to be a whole number multiple of the number of gestures in the dataset (17 gestures of the MSRA dataset). Next, we can easily label any cluster by simple majority rule. Afterwards, for predicting on new samples, we find the closest centroid to the sample and then assign the label depending the cluster selected. Lastly, k-means clustering uses distance between the centroids and the samples to create the clusters. Therefore, we believe in theory that similar gestures will have the smallest mean distance. This is illustrated in the figure below.



(m) (n) (p)

Figure 3.4: (m), (n) and (p) are taken from the MSRA dataset. (m) and (n) are images of the gesture "L" made by two different people while (p) is the gesture "1" made by the same person as (m). After accounting for rotation, (m) and (n) will be very similar while (p) will be clearly different

For the figure above, we define the following two variables:

$$\Delta b_i = \left|\left(b_i^x - b_i^y\right)\right| \; where\ b_i^x\ is\ the\ i-th\ bone(r, \theta, z)\ of\ sample\ x$$

$$\text{Pose difference p. d. (x, y)} = \sum_{i=1}^{20} \Delta\, b_i$$

Therefore p.d. becomes the total distance between two samples. In the figure, Δb_i will be small for every i between sample m and sample n because they are similar. However, when comparing sample m and sample p, the bones for the thumb will have large Δb_i. Therefore, we get $p.d.(m,n) \leq p.d.(m,p)$ and $p.d.(m,n) \leq p.d.(n,p)$. Therefore, samples m and n should be clustered together and p should fall into a different cluster.

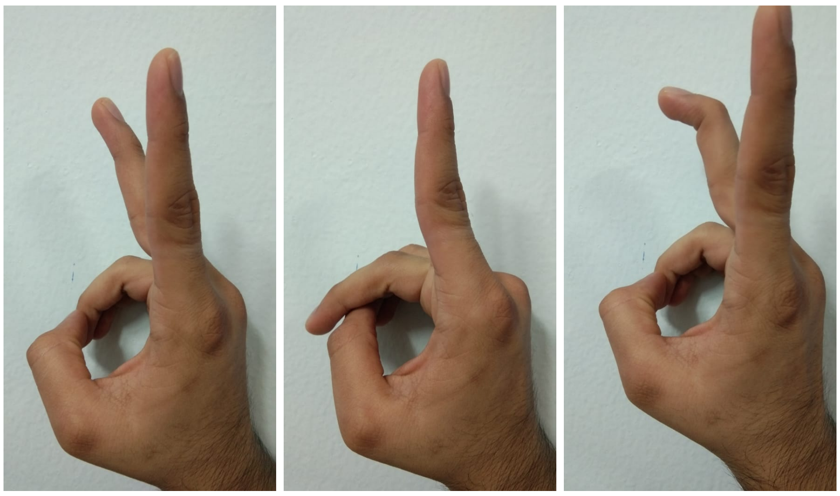


Figure 3.5 (a) Illustrates the symbol “2” seen from a side view. Both the index and middle fingers are straight parallel to the palm. (b) Illustrates the symbol “1” seen from a side view. The index finger is perpendicular to the palm while the middle finger is perpendicular to the palm. (c) Illustrates the symbol “2” seen from a side view. . Both the index and middle fingers are straight parallel to the palm. Only the tip of the middle finger is perpendicular to the palm. But does not affect the resultant gesture.

Secondly, we also observed that the contribution of all the finger joints is not the same. For example, rotating a finger about the knuckle joints results in a larger change in pose compared to rotating about the DIP. This is seen in the figure 3.4 above. Figure (a) gives the base pose, figure (b) is the result of rotating the middle finger at the knuckle and figure (c) is the result of rotating at the DIP. Clearly, figure (a) and figure (c) still depict the same gesture (number 2) while figure (b) shows a different gesture(number 1). Therefore, it is necessary to build this observation in the clustering model. We achieve this by assigning custom weights to different bones. In our experiments, we preprocess the pose vectors by assigning 0.5 weight to wrist bones, and knuckles, 0.3 weight to MIP and 0.2 to DIP.

Lastly, to investigate the importance of different coordinates ($r\ vs\ \theta\ vs\ z$), we formed different clusters using subsets of the available coordinates. We tested using all three coordinates, $only\ r\ and\ \theta$, and $only\ using\ \theta$. In each case, we found that the best model had similar accuracy of about 30.85%. These two subsets were chosen due to specific reasons. Firstly, we eliminated using depth coordinate (z) because we wanted to test the importance of inferring 3D coordinates instead of 2D coordinates during pose regression stage. Since, the depth coordinate is same in both the x,y,z and r, θ, z coordinate systems, if the subset has comparable accuracy, we can instead only regress 2D coordinates from the image. Secondly, we try using only the angular coordinate θ because r coordinate only represents the length of the bone. Since for a generic hand model, the bone length would always be constant, it would never contribute to the bone difference Δb_i, since it will always evaluate to 0. Therefore, we can safely remove it from the model directly. However, note that since θ depends on both x and y coordinates, this does not help us further reduce the prediction space for the pose regressor. It only helps simplify the clustering model and speed up the training process.

In our testing, the best accuracy using K-means clustering was found to be 30.85%. Compared to 5.88% accuracy of random assignment, we see the model has learnt some information. However, the accuracy is not enough to be used in real life deployment. More details are discussed in the next section.

3.4.2.2 Random Forest models

Random Forests were chosen as the second model because they excel at forming threshold values for separating two gestures. In particular, we believe that this thresholding ability should help the forests to distinguish between a finger has been lifted and a finger is curled. Depending on this learnt information, the forest should be able to the distinguish between different gestures based on the different combinations of fingers being lifted and being curled. This intuition is confirmed by the excellent performance of random forests on the MSRA dataset with 99.96% training accuracy and 99.6% test accuracy.

3.5 Performance Measurement Heuristics

After discussing the architecture and workflow of the models, we next look at the performance measurement heuristics used throughout is project.

The project has been divided into 2 separate phases, Hand Pose Estimation System development and Hand Gesture Recognition System development. The two phases differ in the type of problem they attempt to solve. Hand pose estimation is a regression type problem. This means that the locations of each joint can take any value between a smooth interval. On the other hand, hand gesture recognition is a classification type problem. It attempts to label each gesture with an appropriate name from the set of classes already provided by the dataset. Due to this major difference, the measurement heuristics differ for the two phases.

In the first phase, since the problem is of a regressive nature, we use a L2-distance based error measurement. The L2-distance can be expressed in the following equation:

$$error = \frac{1}{n} \times \frac{1}{3 \times 20} \times \sum_{i=0}^{all\ samples} (y_i - \hat{y}_i)^2 \tag{3.1}$$

In the above equation y_i is a vector containing the actual locations of all the joints for a particular sample and $\hat{y}_i$ is a vector containing the prediction made by our model. By taking the difference of the two errors, we can get the error made by our network. Next we square the difference in order to make the loss function invariant of the sign (+/-) of the error. Finally, we calculate the above defined loss function for each sample in our dataset and sum all the resulting values to get the total error made by our model. Finally n is the total number of samples in the training dataset and 3×20 reduces the error from per pose to per joint since there are 20 joint locations being regressed and we measure x, y and z coordinates for each. Using such an error function enforces the training method to try to minimize the distance between the actual position and the predicted position by our model.

For the second phase, we have a classification problem. Therefore we use the accuracy heuristic. The accuracy heuristic is simply the number of mislabeled samples by our model. We compare the actual label and predicted label for each sample. If the labels do not match, we add 1 to our total error. In addition to accuracy, recall and precision heuristics might also be considered in the future. Since the project is still Phase 1 development, these different heuristics are still yet to be completely explored. More research will be conducted on the appropriate performance heuristics at the start of the Phase 2.

3.6 Development Details

The project has been primarily spilt into two phases. The first half of the project focuses on Hand Pose Estimation and the second half of the project focuses on Hand Gesture recognition. Currently, the project has finished Phase 1 and will now move on to developing Hand Gesture Recognition in Phase 2. The details of the two phases are as follows:

3.6.1 Phase 1: Hand Pose Estimation Problem

For this phase, a lightweight hand pose estimating model was developed. The model was based on the observation that the movements of the hand joints are primarily hierarchical and this fact can be to refine the estimations made by the model. The implementation details of these models are as follows:

3.6.1.1 Implementation Details

This is an implementation of the architecture discussed in section 3.3. The implementation slightly differs from the earlier described network clarity of execution. Here we first discuss the major aspects of the implementation. The architecture consists of three branches. The first branch is responsible for regressing the position of the palm joint as well as the MCP joints. The second branch then uses the positions predicted by the first branch in addition to its own feature extraction layer in order to detect the PIP and DIP joints. Note that unlike the architecture discussed in section 3.3, this layer also acts as a refining layer for the joints detected in the first stage. This is achieved by adding the set of joints predicted by the first layer to set of joints predicted by the second layer. Lastly, the third layer computes the position of the fingers tips using the output of the second layer in addition to its own feature extraction layer. Also note that similar to the second layer, the third layer also acts as a refining layer for the joints predicted by the preceding two layers. Hence, the output of the third layer is the complete set of joints in the hand.

The key models developed are *HandNet6, HandNet7, HandNet9* and *HandNet10*. *HandNet6* combines the initial feature extraction stage for the three subnetworks. This is comparable to the common feature extraction layer discussed in the architecture in section 3.3. This design allows to significantly reduce the number of parameters that need to be trained. This also as an effect of significantly lowering the execution time of the model. *HandNet7* embed a generic hand model into the architecture itself. This change in design allows to restrict the output latent space from purely 3D to 2.5D. We believe that this reduction in the output latent space should have a positive effect to the performance of this model.

HandNet9 and *HandNet10* used ResNet50 and ResNet18 [20] as common feature extraction networks. Since ResNet architecture makes it easier for larger networks to learn the identity function [20], this characteristic allowed *HandNet* architecture to learn larger amount of information without overfitting to the data. *HandNet9* and *HandNet10* differ only in the size of the ResNet networks used with *HandNet9* using a 50 layer deep network while *HandNet10* only uses a 18 layer deep network.

Further details of the implementation will be covered in section 4, when we discuss the results measured using different models created during the iterations.

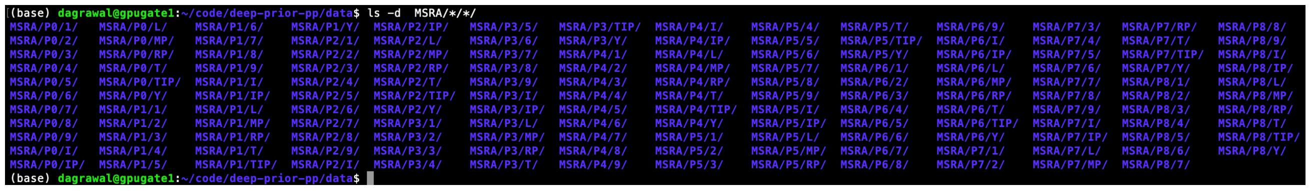
3.6.2 Phase 2: Hand Gesture Recognition Problem

As discussed earlier, we tested two different models for gesture recognition model. In this subsection, we discuss the implementation details for both the models and training procedure for the CrossInfoNet architecture.

3.6.2.1: *CrossInfoNet* Model Provided

Before, discussing the implementations of the K-means clustering and Random Forests, we first briefly discuss the training conducted for the *CrossInfoNet* model provided by their authors in [17]. For training on the MSRA dataset, the authors provide scripts that train the model over samples from 8 of the 9 participants and use the 9th person as the validation set for testing purposes. For our training, we train the model twice, once with person 0 acting as the validation set and once person 1 acting as the validation set. Then the results of the two training cycles are combined to give the final model. [17] suggest training for 9 cycles and combining the nine training results. However, in our experiments training over only 2 cycles showed consistent performance. Finally, [17] offers scripts for testing the models and generating the images for the samples. We then modified these scripts to accept a single image as an input and then output the resultant image, plots and pose vector. The pose vector is then passed on to the gesture classifier.

3.6.2.2: MSRA Dataset Details



```
(base) dagrawal@gpugate1:~/code/deep-prior-pp/data$ ls -d  MSRA/*/*/
MSRA/P0/1/   MSRA/P0/L/    MSRA/P1/6/    MSRA/P1/Y/  MSRA/P2/IP/   MSRA/P3/5/   MSRA/P3/TIP/  MSRA/P4/I/    MSRA/P5/4/   MSRA/P5/T/    MSRA/P6/9/    MSRA/P7/3/   MSRA/P7/RP/   MSRA/P8/8/
MSRA/P0/2/   MSRA/P0/MP/   MSRA/P1/7/    MSRA/P2/1/  MSRA/P2/L/    MSRA/P3/6/   MSRA/P3/Y/    MSRA/P4/IP/   MSRA/P5/5/   MSRA/P5/TIP/  MSRA/P6/I/    MSRA/P7/4/   MSRA/P7/T/    MSRA/P8/9/
MSRA/P0/3/   MSRA/P0/RP/   MSRA/P1/8/    MSRA/P2/2/  MSRA/P2/MP/   MSRA/P3/7/   MSRA/P4/1/    MSRA/P4/L/    MSRA/P5/6/   MSRA/P5/Y/    MSRA/P6/IP/   MSRA/P7/5/   MSRA/P7/TIP/  MSRA/P8/I/
MSRA/P0/4/   MSRA/P0/T/    MSRA/P1/9/    MSRA/P2/3/  MSRA/P2/RP/   MSRA/P3/8/   MSRA/P4/2/    MSRA/P4/MP/   MSRA/P5/7/   MSRA/P6/1/    MSRA/P6/L/    MSRA/P7/6/   MSRA/P7/Y/    MSRA/P8/IP/
MSRA/P0/5/   MSRA/P0/TIP/  MSRA/P1/I/    MSRA/P2/4/  MSRA/P2/T/    MSRA/P3/9/   MSRA/P4/3/    MSRA/P4/RP/   MSRA/P5/8/   MSRA/P6/2/    MSRA/P6/MP/   MSRA/P7/7/   MSRA/P8/1/    MSRA/P8/L/
MSRA/P0/6/   MSRA/P0/Y/    MSRA/P1/IP/   MSRA/P2/5/  MSRA/P2/TIP/  MSRA/P3/I/   MSRA/P4/4/    MSRA/P4/T/    MSRA/P5/9/   MSRA/P6/3/    MSRA/P6/RP/   MSRA/P7/8/   MSRA/P8/2/    MSRA/P8/MP/
MSRA/P0/7/   MSRA/P1/1/    MSRA/P1/L/    MSRA/P2/6/  MSRA/P2/Y/    MSRA/P3/IP/  MSRA/P4/5/    MSRA/P4/TIP/  MSRA/P5/I/   MSRA/P6/4/    MSRA/P6/T/    MSRA/P7/9/   MSRA/P8/3/    MSRA/P8/RP/
MSRA/P0/8/   MSRA/P1/2/    MSRA/P1/MP/   MSRA/P2/7/  MSRA/P3/1/    MSRA/P3/L/   MSRA/P4/6/    MSRA/P4/Y/    MSRA/P5/IP/  MSRA/P6/5/    MSRA/P6/TIP/  MSRA/P7/I/   MSRA/P8/4/    MSRA/P8/T/
MSRA/P0/9/   MSRA/P1/3/    MSRA/P1/RP/   MSRA/P2/8/  MSRA/P3/2/    MSRA/P3/MP/  MSRA/P4/7/    MSRA/P5/1/    MSRA/P5/L/   MSRA/P6/6/    MSRA/P6/Y/    MSRA/P7/IP/  MSRA/P8/5/    MSRA/P8/TIP/
MSRA/P0/I/   MSRA/P1/4/    MSRA/P1/T/    MSRA/P2/9/  MSRA/P3/3/    MSRA/P3/RP/  MSRA/P4/8/    MSRA/P5/2/    MSRA/P5/MP/  MSRA/P6/7/    MSRA/P7/1/    MSRA/P7/L/   MSRA/P8/6/    MSRA/P8/Y/
MSRA/P0/IP/  MSRA/P1/5/    MSRA/P1/TIP/  MSRA/P2/I/  MSRA/P3/4/    MSRA/P3/T/   MSRA/P4/9/    MSRA/P5/3/    MSRA/P5/RP/  MSRA/P6/8/    MSRA/P7/2/    MSRA/P7/MP/  MSRA/P8/7/
(base) dagrawal@gpugate1:~/code/deep-prior-pp/data$ 
```

Figure 3.6: MSRA file structure

The file structure of the MSRA dataset is shown in the figure 3.6 above. Originally, the annotation file is provided for a particular gesture by a particular person. We first combined all these files to have a single csv file with all the required annotations. The provided annotations use real world x,y,z coordinates. These were then converted into r,θ,z coordinate system according to the equations below.

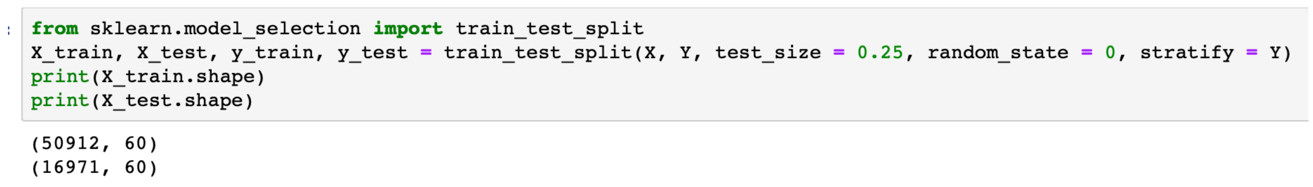
$$r = \sqrt{x^2 + y^2} \tag{3.2}$$

$$\theta = \tan^{-1}\frac{y}{x} \tag{3.3}$$

$$z' = z \tag{3.4}$$

In equation 3.4 above, z' is the z coordinate in r,t,z coordinate system and z is the depth coordinate in x,y,z coordinate system.

Next, we split the dataset into training and testing dataset. [17] suggest using one of the participants as the validation set. We used their approach for training the CrossInfoNet model. However, for training the Random Forest, we opted to create shuffled subsets of the complete dataset. This is shown in the figure 3.7 below. The total number of samples is 67883 with 50912 in the training set and 16971 in the test set.



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0, stratify = Y)
print(X_train.shape)
print(X_test.shape)
```

```
(50912, 60)
(16971, 60)
```

Figure 3.7: Creating train and test subsets of MSRA

Since Random Forests do not require the data to be normalized, we opted to not normalize the data. However, the data was normalized before being fed to the K-means Clustering model.

3.6.2.3: Clustering Model Used

For the clustering model, we used the implementation provided by [] to perform the learning. For our first model, we used the labels provided by the MSRA dataset directly using all the x,y and z coordinates. We trained model for *1000,000 iterations* with *k = 17*. The training process took 72 hours roughly. The output provided two lists. First list contained the *clusters labels* for all the samples while the second label contained the *coordinates for the centroids*. For the rest of the models, we always transformed the pose vectors to r, θ, and z coordinates and trained for *1000,000 iterations*. To test the complete pipeline, in addition to using the provided labels, we also performed clustering using the predictions from the *CrossInfoNet* [17] model discussed in Chapter 2 and the Hand Net model that we developed. Finally, as discussed in section [], we also experimented with different weight assignments to r, θ, and z coordinates ranging between 0.3 to 2 and with different subsets of r, θ, and z coordinates. The results are summarized in the following sections.

3.6.2.4: Random Forest Model Used

Next, for the random forest models, we always used r, θ, and z coordinates. The models were created using the *sklearn* library in python. Since the input space is only 60 dimensional, we found that there was no need to use GPUs to train the system. In our testing, we used number of trees to be 5, 11, 25 and 100. Random forest with 100 trees was discarded for the large training time it required. Finally, we found forest with 11 trees to be best for training time vs accuracy. Therefore, we always used 11 trees for the rest of the models. Similar to the Clustering model, we again trained the forests using outputs from *CrossInfoNet* and *HandNet* in addition to the provided labels. The models using *CrossInfoNet* and the original labels perform almost identically with accuracy of about 99%.

3.7 Summary

In designing the models for this project, we made a key observation based on the underlying hierarchy of hand joint locations. Using this observation, we model our architecture to predict the joints in a branched system. Each branch of the architecture is responsible of a separate subset of the hand joints. Finally, each branch shares information with the following branch in order to aid the prediction of the rest of the hand joints.

Next, we proposed using pose-to-vector prediction paradigm that was able to make maximum advantage of the learning done by pose estimators to carry out gesture prediction. We discussed the advantages of using pose in place of images and the simpler models required in this setting. For our implementation, we found Random Forests to be the best. When coupled with *CrossInfoNet* model, our pipeline is able to deliver impressive accuracy of 99.6% over the MSRA dataset.

This section covered the theoretical observations and models proposed by this project. The next section will look at the actual implementation and results obtained by this project.

Chapter 4: Results

4.1 Introduction

In this chapter we discuss some of the interesting results observed from the currently implemented models. We shortly discuss the choices for the internal architecture of the different subnetworks discussed in section 3.5.1.1. We then discuss the differences in architecture over the different iterations of the model. Lastly, we compare the results outputted by the models.

4.2 Implemented Architecture

In section 3.5.1.1, we briefly discussed the difference between the implemented models and the proposed model by this study. In this section, we detail the changes implemented to the models over a number of iterations. In section 3.5.1.1, we also discussed a key property of our network, that being the proposed architecture is independent of the actual details of each subnetwork. Therefore, during our implementations, it is required to select a design for the architecture of each subnetwork. In this study we used the model in [2] called *Deep-prior* as a base model. We split that model and use the segmented subnetworks in place of different subnetworks in our proposed model. We refer to the resulting model as *HandNet*. By doing so, we believe that we can measure the difference in performance and time penalty between *Deep-prior* and *HandNet.*

In this section we discuss an interesting observation made when comparing results from two different iterations of *HandNet*. We refer to these two iterations as *HandNet1* and *HandNet2* respectively. The summary of the implementation details of *HandNet1* can be found in the Figure 4.1.

Notice that the architecture in Figure 4.1 is very similar to the architecture of the originally proposed model. The difference lies in the first feature extraction layer for each branch. *HandNet1* features an independent primary feature extraction layer. The rationale behind this variation was to confirm the earlier discussed assumption that the basic features should be similar across the three branches of the network. By comparing the performance of *HandNet1* and *HandNet2*, we can conclusively decide the effect of the primary feature extraction. In order to conduct this experiment, the implementation of *HandNet2* was kept completely identical to the original model.

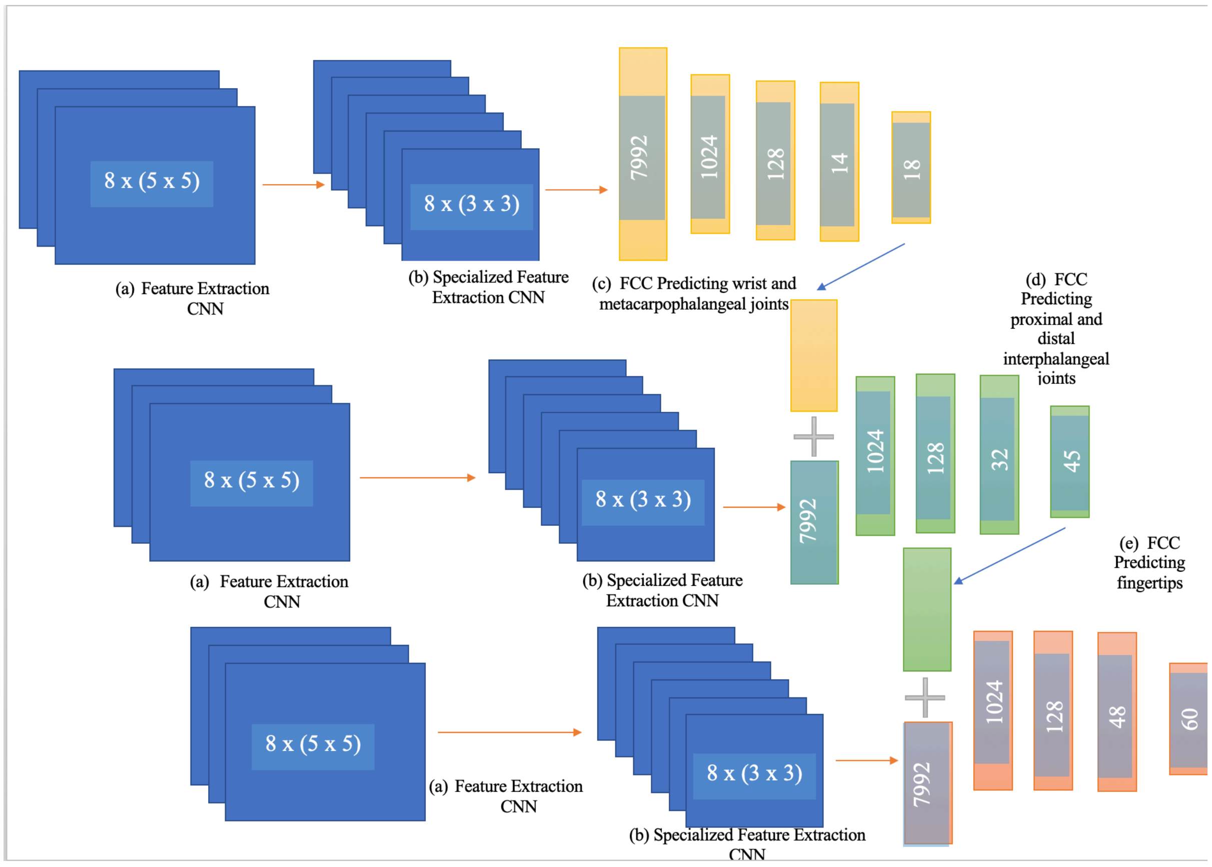

8 x (5 x 5)
(a) Feature Extraction CNN
8 x (3 x 3)
(b) Specialized Feature Extraction CNN
7992
1024
128
14
18
(c) FCC Predicting wrist and metacarpophalangeal joints
(d) FCC Predicting proximal and distal interphalangeal joints
1024
128
32
45
(e) FCC Predicting fingertips
1024
128
48
60

Figure 4.1. Implemented architecture of *HandNet1*. Architecture of *HandNet1* omits the common feature extraction layer discussed in the original layer and hence has approximately thrice the number of trainable parameters

In the above figure, we also see that the common feature extraction layers use 8x5x5 filters for the CNN layers. It performs three such convolutions. Next, the specialized feature CNN uses a 8x3x3 filter and performs convolution only once. For both common features and specialized features, maximum pooling with stride 2 is performed after every convolution. The sizes of the FCC layers are as seen in the figure. Note that the last FCC layer outputs the complete joint coordinates vector of length 60.

The two models were trained on NYU Hand pose dataset discussed in section 3.2. The images in the NYU Hand Pose Dataset have 36 annotated joints. However, for this implementation, we only regress the locations of 20 of the joints. See figure 4.2 below for illustration of the hand model that we regress. Figure 4.2 (a), shows the actual image captured and figure 4.2 (b) shows the 36 joint locations that are annotated in the dataset. In this study we only regress the locations of the joints marked in red in figure 4.2 (b). For each of the four fingers, we predict the MCP, DIP, PIP and finger tips. However, for the thumb only MCP, DIP and thumb tip are predicted. In addition of to these joints, a central palm joint is predicted as well. This joint acts as the base location of the hand.

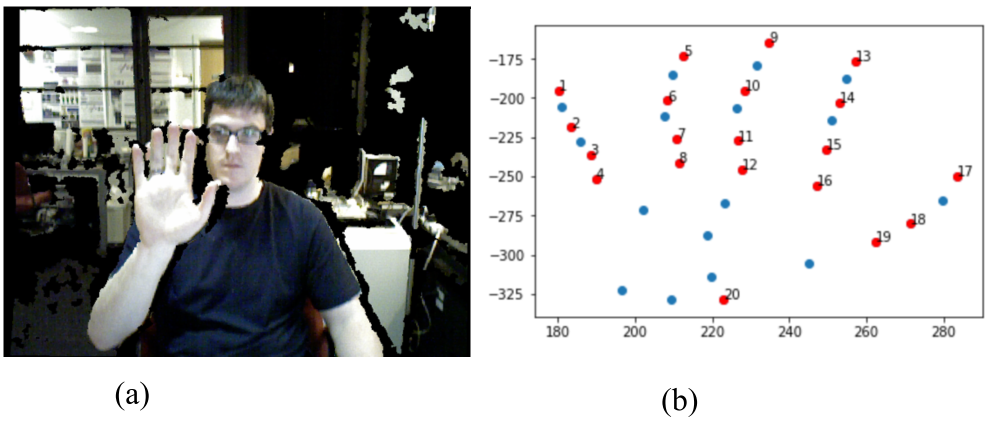


Figure 4.2 Illustration of the hand model predicted. (a) depicts the actual image captured and (b) graphs the 36 hand joints annotated in NYU dataset. (a) is taken from NYU dataset(`train/rgb_1_0000001.png`)

The training was conducted on the training sub dataset using Adam optimizer and the min squared error loss (L2 loss) discussed earlier. The generalized performance was measured on the test sub dataset of NYU dataset. We notice that the average loss for HandNet1 is 120.7 with standard deviation of 9.12 while for HandNet2 it is 127.23 with standard deviation of 10.99. For the calculation of the error, x, y and z coordinates are measured in millimeters in the real world. The difference between the two models is 5.8%. This difference is within the margin of variation associated with being due to the randomness with which the input is presented to the models during training. The complete statistics of the comparison of the results of the two models is presented in Table 4.1 below.

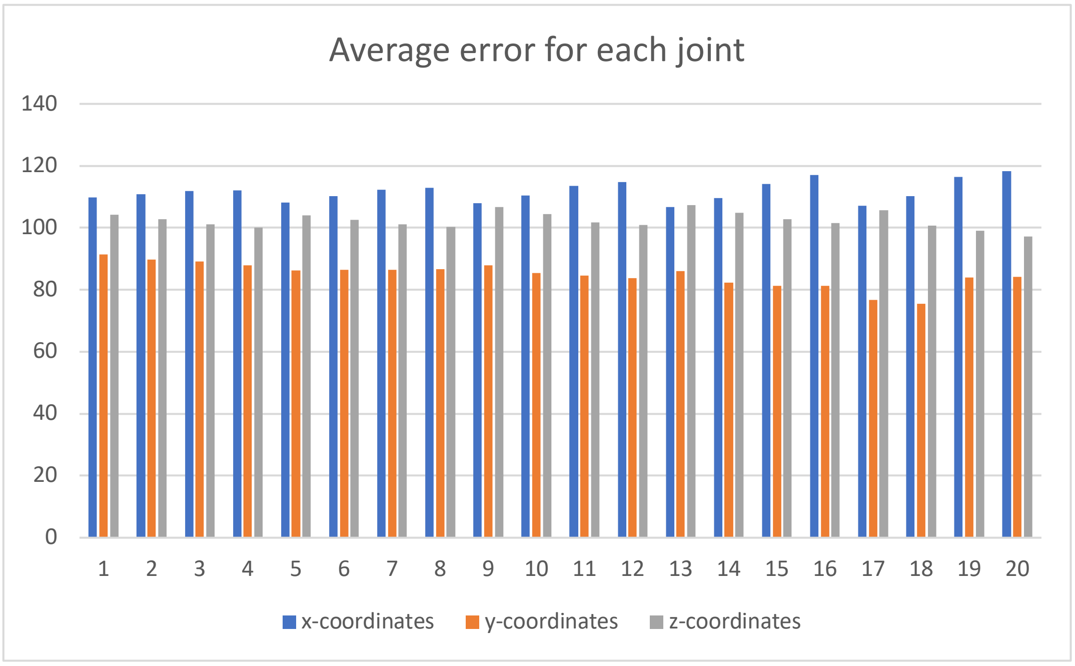
Table 4.1. Results of *HandNet1* and *HandNet2*.

	Average	Standard Deviation	Minimum Error	Maximum error
HandNet1	120.7	9.1214	89.1276	147.2234
HandNet2	127.23	10.9903	95.4217	164.3338
Deep-Prior-ORRef	19.5	N/A	N/A	N/A
Tompson et. al.	21	N/A	N/A	N/A

The average pose error for *Deep-Prior-ORRef* and *Tompson et. al.* are presented as listed in [2].

Since the difference in error is within the expected margins, we can conclude that using a common feature layer indeed has little effect on the overall performance of the model. This also confirms that the primary feature extraction layers in *HandNet1* must be extracting similar features from the input. Based on these results, we can conclusively argue that given the decrease in the total number of trainable parameters and the low execution times, using a common feature extraction layer should be preferred over independent feature extractors for the three branches. This finding helps this study focus on the second objective of building a lightweight deep learning model capable of similar performance as more complex models.

Graph 4.1. Average Error measured in mm per joint for Hand Net 1



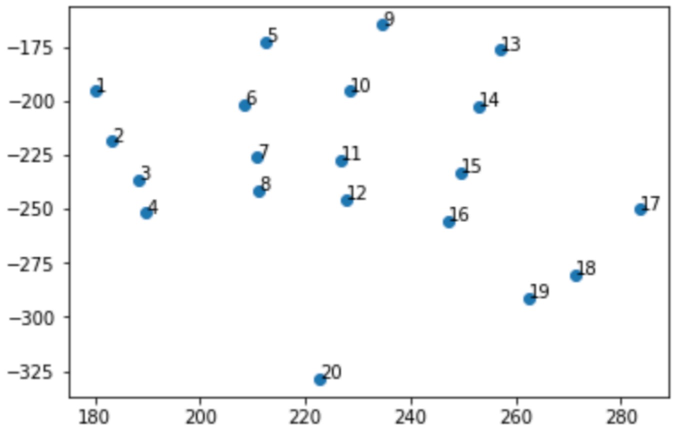


Figure 4.3 Joint numbers of graph 1.

Lastly, we looking at graph 1, we clearly see that the model struggles is regressing the z-coordinate for all the joints when compared to the y-coordinates. On average, error in z-coordinate is 1.25 times the error in y-coordinate. This is also theoretically justifiable since depth is not measured as a completely independent dimension. Due to occlusion in hand joint locations, the depth information for a number of joints is not captured by the samples.

This study reaches the conclusion that by applying such refinements to a model architecture, it might be possible to build a model achieving the abovementioned objective. This is clear from analyzing the execution times of *HandNet1* and *HandNet2* as seen in Table 2 below. By using a common feature layer, *HandNet2* decreases execution time by 0.2 seconds which is 13.79% of the original 1.45 seconds of *HandNet1*.

Table 4.2. Execution Times of *HandNet1* and *HandNet2*

	Average Execution Time (CPU) as measured on jupyter lab
HandNet1	1.45 seconds
HandNet2	1.25 seconds

4.3 Gesture prediction System

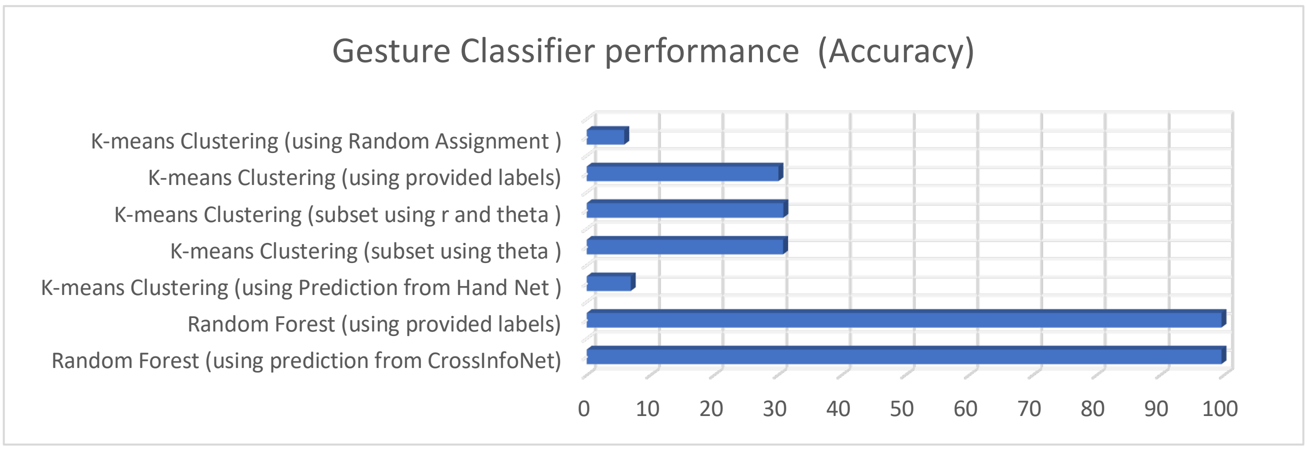
The accuracies for different models trained have be summarized in the table below. As stated earlier, the random forest trained over the original labels performs the best. More importantly, the random forest trained over *CrossInfoNet[17]* predictions has virtually the same output accuracy. This serves as positive evidence for our original hypothesis that an accurate pose contains most of the required information for regressing static gestures.

Table 4.3. Gesture Classifier performance

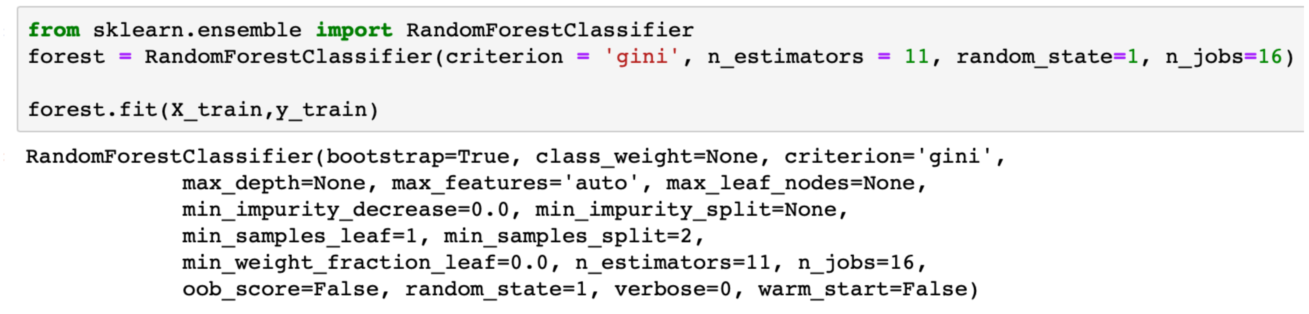
Model	**Accuracy (in percentage)**
Random Forest (using prediction from *CrossInfoNet*)	99.6
Random Forest (using provided labels)	99.3
K-means Clustering (using Prediction from Hand Net)	6.9
K-means Clustering (subset using theta)	30.81
K-means Clustering (subset using r and theta)	30.85
K-means Clustering (using provided labels)	30.08
K-means Clustering (using Random Assignment)	5.88

The above table has also been illustrated in the graph below. From the graph, it is clear that even the clustering method significantly out performs random assignment. However, we see that its performance is much worse compared to simple Random Forest. We believe that is this can be due to non-convergence or bad starting centroids. Given that we are trying to form 17 clusters from data that has very small range of values (0 to 2π for θ), this could be the reason for convergence of the k-means clustering method. For example, if two of the initial centroids come from the same gesture, this could split the samples unevenly. Therefore, we believe that in future work it will be fruitful to try better unsupervised clustering methods and try custom initial centroids. Secondly, we believe that the position of the thumb can be acting as noise in the system. Since the thumb is free to take a larger latent space for any gesture, this could be adding to the distance between two samples of the same gesture. Therefore, adding custom weights to the thumb joints might also help the model perform better.

Graph 4.2 Gesture Classifier performance (Accuracy)



However, based on the excellent performance of the Random forests, we opted to use them for our final model pipeline. Therefore, for an end-to-end approach, we use the *CrossInfoNet* model to perform the pose regression and then we use Random Forests as described in the figure 4.4 below to the make the prediction.

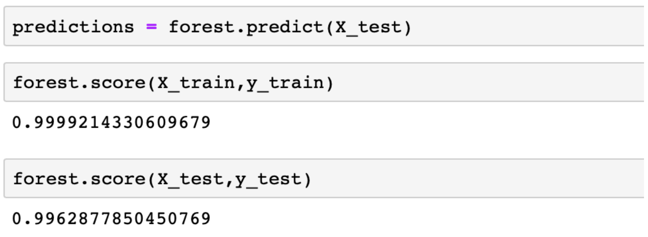


```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(criterion = 'gini', n_estimators = 11, random_state=1, n_jobs=16)

forest.fit(X_train,y_train)
```

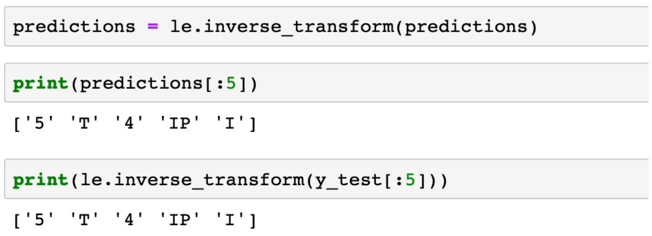
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=11, n_jobs=16,
            oob_score=False, random_state=1, verbose=0, warm_start=False)
```

Figure 4.4: Random Forests initialized in jupyter using sklearn

The train and test accuracy of using Random Forests is shown in the figure 4.5 below. We get a training accuracy of 99.99% and test accuracy of 99.62%.



```
predictions = forest.predict(X_test)

forest.score(X_train,y_train)
0.9999214330609679

forest.score(X_test,y_test)
0.9962877850450769
```

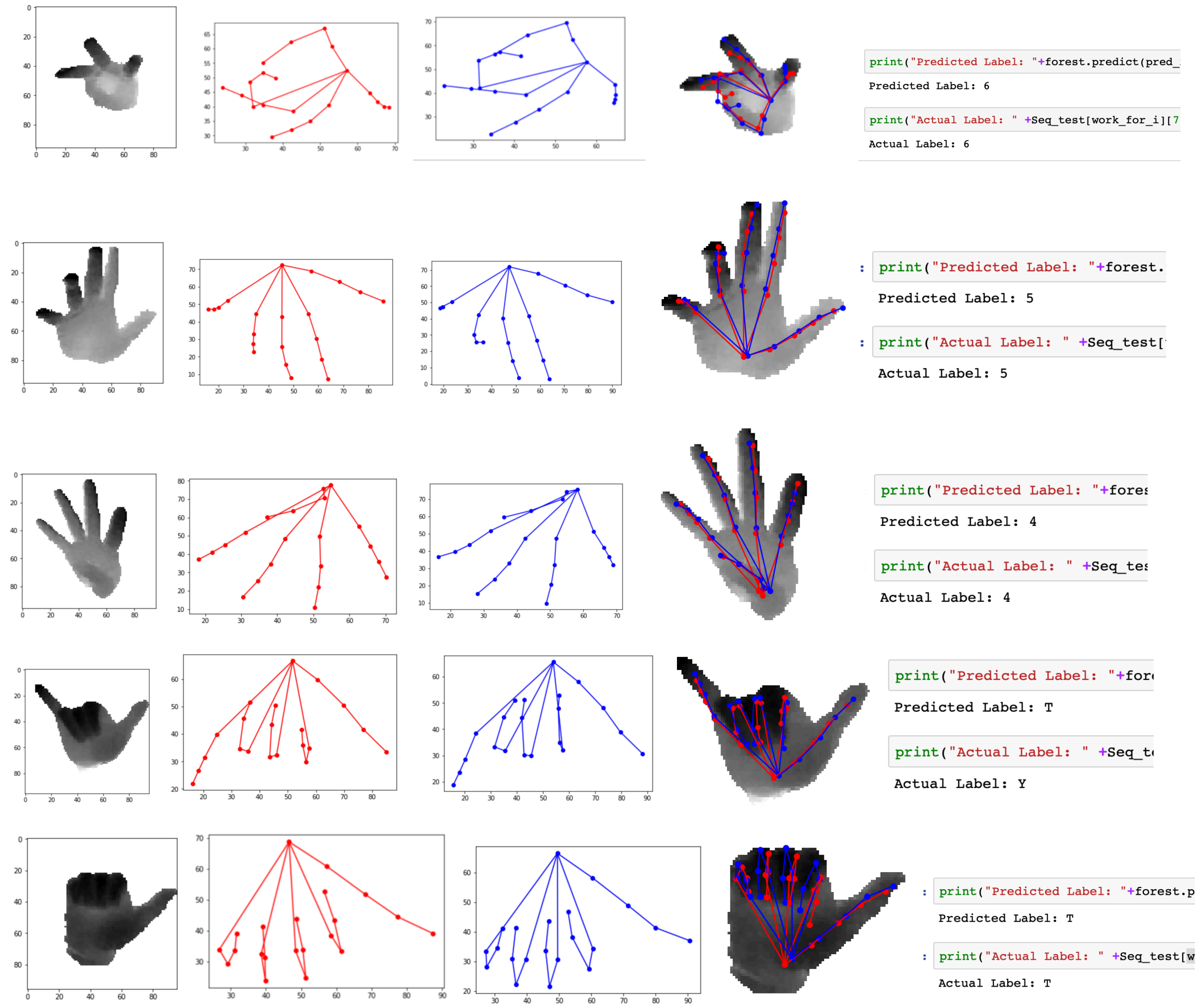


```
predictions = le.inverse_transform(predictions)

print(predictions[:5])
['5' 'T' '4' 'IP' 'I']

print(le.inverse_transform(y_test[:5]))
['5' 'T' '4' 'IP' 'I']
```

Figure 4.5 (a) Output of forest.score() and (b) prediction on the first 5 samples in the dataset.

Next, we present the output of the pipeline for some of the samples in the MSRA dataset.

Original image | Predicted pose | Actual pose | Output image | Predicted label



```
print("Predicted Label: "+forest.predict(pred_
Predicted Label: 6

print("Actual Label: " +Seq_test[work_for_i][7
Actual Label: 6
```

```
: print("Predicted Label: "+forest.
Predicted Label: 5

: print("Actual Label: " +Seq_test[
Actual Label: 5
```

```
print("Predicted Label: "+fores
Predicted Label: 4

print("Actual Label: " +Seq_tes
Actual Label: 4
```

```
print("Predicted Label: "+for
Predicted Label: T

print("Actual Label: " +Seq_t
Actual Label: Y
```

```
: print("Predicted Label: "+forest.p
Predicted Label: T

: print("Actual Label: " +Seq_test[w
Actual Label: T
```

Figure 4.6: Sample outputs of the pipeline

From the figure above, we see that the model generally performs very well. In the figure, it only mislabels the fourth sample as ‘T’ instead of ‘Y’. However, given the high similarity between ‘Y’ and ‘T’ seen in fourth and fifth sample, this is to be expected in certain cases. We believe that if deeper trees were constructed, this mislabeling might also be corrected.

4.4 Challenges and Limitations

We next look at some of the challenges faced with Hand Pose Estimation.

4.4.1 Large Latent Space

Firstly, note that the fingers are the most “free” parts of the human body. Due to the three finger joints, MCP joint, PIP joint and DIP, the fingers are capable of fine movements and have a large volume that each finger can potentially occupy. Since large degree of freedom for the fingers results in a large latent output space for a model, this severely limits the possible accuracy of a hand pose estimators [14]. As mentioned earlier, by transforming the data from 3D to 2.5D, we try to minimize the latent output space.

4.4.2 Varying hand models

Another challenge arise from the varying finger and palm sizes. The sizes of fingers vary from human to human. Besides the sizes of particular fingers, the ratio of the lengths of different fingers also varies amongst individuals. Easiest example of this being that many people have their index and ring fingers of almost same length while others have one clearly larger than the other. This results in a large number of possible human hand models [14]. This is less desirable while programming a generic hand model into a system. The system has to account for different variations of human hand and the different latent spaces that arises from these differences. To mitigate this difficulty, we calculate the mean bone length for every bone in a human hand across our dataset. We then use this mean bone length as a generic hand model.

4.3.3 Self Occlusion

Lastly, one of the most challenging problems faced during Hand Pose Estimation is due to self-occlusion[2]. Due to the large number of joints in such a small region, it is easy for fingers to block each other from the view of the camera. This can result in the system having to estimate position of a finger that is completely occluded from the view of the camera. We believe that using depth information provided by the depth sensors should help reduce the amount of ambiguity in the data.

4.4.4 Training Random Forest on GPU

During the development of this project, we found that there is an absence of implementations of Random Forests or Boosted Forests using parallelization provided by a GPU[]. This severely limited the size of the forests that we could try. We found an implementation provided by tensorflow[] that uses the GPU. However, at this time it is limited to only binary classification. Therefore, we believe if a GPU implementation is created, it might be worthwhile to try training larger Random forests and using more challenging datasets such as the NYU hand pose dataset [].

4.4.5 Dataset Limitations

During the development of this project faces two major limitations. Firstly, the image and video resolution is very low for a significant number of datasets available. This results in a large loss of information. This in turn effects the performance of any model. Secondly, for the second phase of this project, we required a dataset that provided both the labels and the annotations for the samples. However, we only found the MSRA dataset to have both types of data available. Since most modern learning models prefer using an end-to-end approach, we found that a pose-to-gesture research model has been under researched but holds significant potential. In future work, it might be advantageous to create pose vectors using accurate pose estimators, such as *Deep-prior++[6], CrossInfoNet[17]* and *hand3d[22],* on datasets with gestures labels.

4.4.6 Dataset and Hardware limitations

Due to the limited hardware available, each iteration of the models requires a significant amount of time to complete training. This acutely limits the number of the models that can be trained and tested. The study mitigates the effect of these limitations by using validation techniques while training and keeping the models as lightweight as feasible. By using validation techniques, one is able to monitor the performance of the model while it is training. This allows one to decide when the model has converged and approximate its real world performance while it is still training.

4.5 Project Schedule and Milestones

4.5.1 Project Schedule:

This table summarizes the different periods in the project and the main objectives of each period. Each period is in between 4 and 6 weeks in length.

Table 4.4: project schedule

Period	Period Task	Status
1 Sep 2019 – 30 Oct 2019	Complete HandNet 6-7	Complete
31 Oct 2019 – 01 Dec 2019	Complete HandNet 8-10	Complete
24 Dec 2019 – 15 Jan 2020	Modified MSRA dataset annotations to fit the required annotation system. Used K-means clustering for pose-to-gesture prediction	Complete
16 Jan 2020 – 14 Feb 2020	Used K-means clustering for pose-to-gesture prediction Used Random Forests for pose-to-gesture prediction	Complete
15 Feb 2020 – 31 Mar 2020	Testing complete pipeline using CrossInfoNet + Random Forests.	Complete
1 Apr 2020 – 18 Apr 2020	Testing and Debugging. Experimenting combinations of pose estimator and gesture predictor.	Complete
19 Apr 2020 – 30 Apr 2020	Real world testing example	Complete

4.5.2 Milestones and Completion Dates:

Listed below are the major milestones for this project.

Table 4.5: Major project milestones.

Date	Milestone
29 Sep 2019	Submission of Inception Deliverables.
30 Oct 2019	Phase 1 complete. A working hand pose estimator ready.
30 Nov 2019	First hand gesture recognition prototype ready.
15 Jan 2020	Second hand gesture recognition prototype ready.
2 Feb 2020	Submission of Elaboration Deliverables.
14 Feb 2020	Third hand gesture recognition model ready.
30 Apr 2020	Submission of Final Deliverables.

4.6 Summary

Through this project, we expect that we can successfully build an architecture that has been previously under researched. By assessing the performance of the models, we should be able to show that the earlier discussed observation can indeed provide valuable information that can be coded into the model. The project has been split into two phases.

In the second stage of this section, we displayed the results of our pose-to-vector predictions. We saw that Random Forests completely outperformed K-means clustering method. We were able to achieve state-of-the-art performance over the MSRA dataset with test accuracy of 99.6%.

Lastly, we explained some of limitations we faced during this study. The unavailability of required datasets became the biggest hurdle in this study. Therefore, it is one of the recommendations that we make in the next section. Furthermore, self-occlusion and low quality of images also slowed the learning process.

Chapter 5: Future Work

Based on the observations of this study, we make the following suggestion for more future work related to image to gesture labelling.

5.1 Pose and gesture annotated Datasets

As stated earlier, datasets that provide both annotations and gesture labels are very limited. Therefore, our first recommendation is to develop more datasets that provide both sets of required data. This should help encourage predicting gestures based on pose instead of an end to end approach using deep neural networks on images. Instead of creating new datasets, we instead recommend creating pose annotations for existing gesture labelled dataset. Given the excellent performance of many existing pose estimation models, this method will be very cheap to implement in both resources and time required.

5.2 Using Statistical Learning Models

In this study, we experimented with Random Forests and K-means clustering methods for pose-to-gesture prediction. Based on the excellent performance of the Random forests, we believe that other traditional statistical learning models might also perform very well. Simpler models have advantages of being faster in execution and more intuitive. Therefore, a study investigating their effectiveness in pose-to-gesture prediction might be very useful.

5.3 Experimenting with nested models

We believe that our initial observation of the hierarchy of hand joints can prove to be useful in future research. [14] and [23] have also come to similar conclusions and showed very good results. Therefore, trying deeper networks with nested architecture might give state-of-the-art results.

Chapter 6: Conclusion

Hand Gesture recognition is an exciting field for research in the computer vision. A successful hand gesture recognition system will enable a number of currently unexplored applications. Chief amongst these applications is Human-Computer Interaction. Allowing computers to infer meaning from hand and finger movements will open a new form of interaction with the computer that may also increase its accessibility to a larger number of users. Current implementations of this system suffer from two central limitations. Firstly, they require an array of sensors to make a reliable regression and secondly, they have large execution time delays. These limitations make the implementations unportable and unsuitable for real time applications respectively. It is to be noted that research is being conducted in order to mitigate these effects [16][6].

This project directly tackles both these limitations. The key objective of this project is to use a single RGB-D image to regress the locations of the hand joints. Secondly, the project also hopes to investigate the feasibility of using a lightweight model to make the system suitable for real time applications. Hence, the project looks to find an ideal balance between the performance and the speed of the system.

This study posits that this insight can be that the positions of hand joints is not truly independent. Therefore, by using the locations of base hand joints, we can regress the position of the distal hand joints. Building on this insight, we designed and implemented a novel architecture. The salient feature of the architecture is the branching system that occurs after the common feature extraction stage. These branches independently regress the positions of their assigned hand joints supplementing their data processing with the output from other branches. Another fact to note is that the general architecture is not dependent on any particular subnetwork. Any existing popular deep learning networks can be plugged in at the different parts of the main architecture.

For our final pipeline, we use the above described model or other hand pose estimation in addition with a gesture classifier such a K-means clustering[21] or Random Forests. In our testing, we found the best combination to be using the *CrossInfoNet* [17] model with Random Forests. We achieved accuracy of 99.99% over the training subset and 99.6% over the subset of the MSRA dataset. This excellent performance shows that we can use pose to vector prediction models reliably. Lastly, because most of the learning is offloaded to already excellent pose estimation models, we can safely use simpler learning models for this task.

Based on this observation, we recommend using traditional statistical learning models such as SVM and shallow Neural Networks for perform pose-to-vector prediction. Furthermore, such prediction models could be further combined with histogram frequency distribution models to be adapted to work with dynamic gestures and videos.

References

1. R. Z. Khan, and N. A Ibraheem. Hand gesture recognition: a literature review. International journal of artificial Intelligence & Applications, 3(4), 161,2012.
2. M. Oberweger, P. Wohlhart, and V. Lepetit,. Hands deep in deep learning for hand pose estimation. arXiv preprint arXiv:1502.06807,2015.
3. O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. arXiv preprint arXiv:1901.10323,2019.
4. Y. Zhang, C. Cao, J. Cheng, and H. Lu. Egogesture: a new dataset and benchmark for egocentric hand gesture recognition. IEEE Transactions on Multimedia, 20(5), 1038-1050,2018.
5. C. Cao, Y. Zhang, Y. Wu, H. Lu, and J. Cheng. Egocentric gesture recognition using recurrent 3d convolutional neural networks with spatiotemporal transformer modules. In Proceedings of the IEEE International Conference on Computer Vision (pp. 3763-3771),2017.
6. J. Tompson, M. Stein, Y. Lecun, and K. Perlin. Real-time continuous pose recovery of human hands using convolutional networks. ACM Transactions on Graphics (ToG), 33(5), 169,2014.
7. P. M. X. Y. S. Gupta, and K. K. S. T. J Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. CVPR, 2016.
8. D. Tang, H. Jin Chang, A. Tejani, and T. K. Kim. Latent regression forest: Structured estimation of 3d articulated hand posture. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3786-3793), 2014.
9. X. Sun, Y. Wei, S. Liang, X. Tang, and J. Sun. Cascaded hand pose regression. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 824-832), 2015.
10. F. Gomez-Donoso, S. Orts-Escolano, and M. Cazorla. Large-scale multiview 3D hand pose dataset. Image and Vision Computing, 81, 25-33, 2019.
11. O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll. Motion fused frames: Data level fusion strategy for hand gesture recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 2103-2111), 2018.
12. S. Yuan, Q. Ye, B. Stenger, S. Jain, and T. K. Kim. Bighand2. 2m benchmark: Hand pose dataset and state of the art analysis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4866-4874), 2017.

13. B. Doosti. Hand Pose Estimation: A Survey. arXiv preprint arXiv:1903.01013,2019.
14. C. Xu, and L. Cheng. Efficient hand pose estimation from a single depth image. In Proceedings of the IEEE international conference on computer vision (pp. 3456-3462), 2013.
15. H. J. Chang, G. Garcia-Hernando, D. Tang, and T. K. Kim. Spatio-Temporal Hough forest for efficient detection–localisation–recognition of fingerwriting in egocentric camera. Computer Vision and Image Understanding, 148, 87-96, 2016.
16. A. Markussen, M. R. Jakobsen, and K. Hornbæk. Vulture: a mid-air word-gesture keyboard. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 1073-1082). AC,2014.
17. K. Du, X. Lin, Y. Sun, and X. Ma. CrossInfoNet: Multi-Task Information Sharing Based Hand Pose Estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 9896-9905),2019.
18. P. Molchanov, S. Gupta, K. Kim, and J. Kautz. Hand gesture recognition with 3D convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 1-7),2015.
19. L. Pigou, S. Dieleman, P. J. Kindermans, and B. Schrauwen. Sign language recognition using convolutional neural networks. In European Conference on Computer Vision (pp. 572-578). Springer, Cham,2014.
20. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385,2015.
21. Charlier B., Feydy J., Glaunès J. A.. Retrieved from https://www.kernel-operations.io/keops/_auto_tutorials/kmeans/plot_kmeans_torch.html, 2020.
22. Zimmermann, C., & Brox, T. Learning to estimate 3d hand pose from single rgb images. In Proceedings of the IEEE International Conference on Computer Vision (pp. 4903-4911). 2017.
23. Wei, S. E., Ramakrishna, V., Kanade, T., & Sheikh, Y. (2016). Convolutional pose machines. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 4724-4732).