

**GestHome: A User-defined  
Postures Detection for Smart Home  
Term 2 Final Report**

**Theodore Fabian Rudy  
Christopher Albert Priatko**

Supervised by

**Prof. Michael R. Lyu**

Computer Science and Engineering  
The Chinese University of Hong Kong  
May 22, 2023

# Contents

<b>Acknowledgement</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Overview . . . . .	5
1.2 Background . . . . .	5
1.3 Objectives . . . . .	8
1.4 Glossary . . . . .	9
<b>2 Related Work</b>	<b>12</b>
2.1 Object Detection . . . . .	12
2.1.1 Two-Stage Detection Framework . . . . .	13
2.1.2 One-Stage Detection Framework . . . . .	14
2.1.3 Other Methods . . . . .	16
2.2 Face Recognition . . . . .	16
2.2.1 One-Shot Learning and Siamese Neural Network . . . . .	17
2.3 Pose Estimation and Keypoints Analysis . . . . .	19
2.4 Action Recognition . . . . .	22
2.4.1 Skeletal-based Action Recognition . . . . .	24
2.5 Long Short-Term Memory . . . . .	25

<b>3</b>	<b>Implementation</b>	<b>27</b>
3.1	General Framework . . . . .	27
3.2	Stage 1: Object detection and Face Recognition . . . . .	31
3.2.1	Object Detection . . . . .	31
3.2.2	Face Recognition . . . . .	31
3.3	Stage 2: Pose Estimation and Action Recognition . . . . .	35
3.3.1	Pose Estimation . . . . .	35
3.3.2	Problem encountered . . . . .	37
3.4	Action Recognition . . . . .	38
3.5	Website building . . . . .	39
3.5.1	Specification . . . . .	39
3.5.2	System Architecture . . . . .	44
3.5.3	System component . . . . .	48
3.5.4	User Interface Design . . . . .	50
3.5.5	Challenges in Website Implementation . . . . .	54
<b>4</b>	<b>Evaluation</b>	<b>56</b>
4.1	Stage 1 . . . . .	56
4.1.1	Object Detection . . . . .	56
4.1.2	Face recognition . . . . .	57
4.2	Stage 2 . . . . .	59
4.2.1	Pose Estimation . . . . .	59
4.2.2	LitePose . . . . .	60
4.2.3	Movenet . . . . .	61
4.2.4	BlazePose . . . . .	62
4.2.5	LSTM Model . . . . .	63

<i>CONTENTS</i>	3
4.2.6 Action Recognition . . . . .	66
4.3 Overall evaluation . . . . .	68
4.3.1 Accuracy evaluation . . . . .	68
4.3.2 Speed evaluation . . . . .	71
4.3.3 Website performance . . . . .	73
<b>5 Conclusion</b>	<b>74</b>
5.1 Summary . . . . .	74
5.2 Problem Encountered . . . . .	75
<b>6 Distribution of Work</b>	<b>77</b>
<b>Bibliography</b>	<b>79</b>



# Acknowledgement

First, we would like to thank God, for giving us this amazing opportunity to have this project that we would never think we are able to do. It is only by His Grace that we are able to do and finish this project.

We would like to express our deepest gratitude to Professor Michael R. Lyu for giving us the opportunity to be our supervisor for our Final Year Project. We would also like to express our deepest gratitude to Mr. Huang Jen-tse for giving us invaluable knowledge, advice and guidance during our Final Year Project. Last but not least, we would also express thanks to Mr. Duan Haodong for advice and help in the integration of PYSKL with our project.

# Chapter 1

## Introduction

### 1.1 Overview

The focus of this final year project is to utilize different aspects of computer vision to process human gestures and/or poses captured by video feed and turn it into command preset in the computer. This report describes the introduction to the topic, the work done throughout the whole year, with the focus on the second semester, and all the implementation challenges that was faced.

### 1.2 Background

With the development of technology, more of the things that are faced in daily life has become easier. One of the example of that aspect is smart technology. This mechanism turn traditional sensors and actuators to be connected through network, enabling it to increase the capability of achieving centralised control over devices, which brings increase in life quality, well-being, safety, productivity, energy efficiency and many more [75]. Not only that this mechanism allow it to be connected through network, the devices connected to the network will also have the capability to have computing-like functionalities [77].

The evolution of this smart technology happens with more integration of artificial intelligence (AI) into the system. Bowes et al. (2012) described that it was in the second generation smart home technology that artificial intelligence started to be used. They mentioned that the implementation of smart home technology in the generation uses AI-based devices, where it is able to detect changes of environment, monitor health condition and many more [13]. In general, AI was mostly used for analytical functionality and was more described as 'reactive' to a predefined trigger actions. Only in the third generation of smart home technology that AI is more utilised to have the capability to inter-operate with other devices and have multiple functionalities. Not only limited in the usage of behavioural analysis to predict user needs and optimization, but this also enable integration with other devices, making it possible to capture, process and transmit data among devices within network [60]. This generation also mark the emerging system of voice-over interface bringing life to home automation system, enabling it to be more interactive.

With the usage of voice-over interface for smart home system being made available for public by different providers. Google Nest and Amazon Echo being two examples of speech recognition in a form of virtual assistant being implemented. As mentioned in the previous sentence, the functionalities of virtual assistant reduces things that is needed to do by the users, but also to enable people who are disabled or elderly to access and dynamically operate and put control over smart things in the home [100]. Despite its power and capability to control centralised system for smart home, the development for interface for this interaction is pretty much stopped here.

Along with the development of voice controlled smart home system, there are others subjects that earn recognition over its progress and evolution. A subject in particular that has gotten much attention is is computer vision.

Traditionally a subject where the task involved in it are mostly tasks like feature extraction engineered by human in a specific algorithm, its capability of computer to process images and videos, increases significantly. Rather than defining the features and analysis of the input and output, the involvement of deep learning in the subject automate the tasks of feature engineering, extraction and classification into one process involving neural network. The neural network learns the value of features that differentiate one image to another, and automatically adjust the weight and parameter for it to recognise the generalised features and pattern for the thing detected. *Figure 1.1* explains more of the work flow and the difference between them.

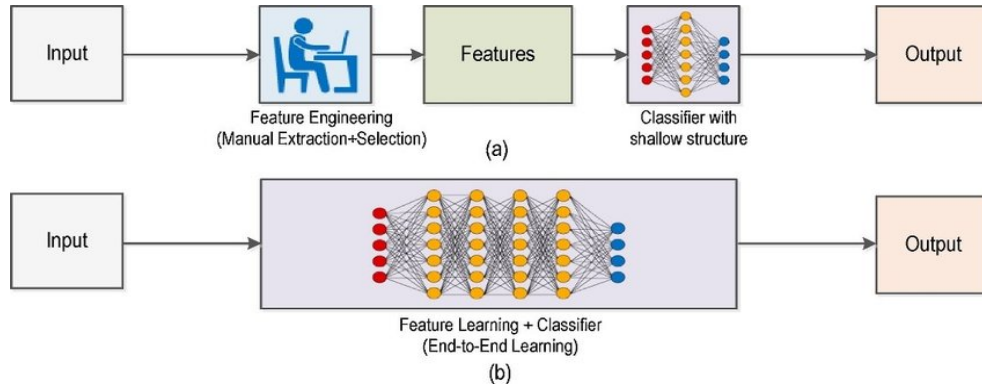


Figure 1.1: Difference in flow of traditional computer vision and modern (deep learning) computer vision, adopted from (Walsh et al., 2019) [91].

With more tasks and applications that are implemented using modern computer vision, the growth of computer vision is formidable. This growth enables a lot of enterprises to implement computer vision tasks, such as autonomous vehicles or so we call it self-driving car. Similar to what was mentioned in the third generation of smart home technology earlier, autonomous vehicles uses different sensors and AI as a part for them to enable the interaction between them with the whole system without human intervention. However, Computer Vision plays a huge part in it, where some of smaller tasks that is able to be

solved through modern computer vision. Despite all this, there are a lot of potential that Computer Vision has to offer, and more methods and techniques used to increase performance, accuracy, precision, and others.

## 1.3 Objectives

The goal of GestHome, our project, is to utilize computer vision to make virtual assistant to be controllable by gestures, which is customizable by user. By that, we will not only be needing feature limited to gesture detection or in other words, action recognition, but to also employ person detection to understand if a person gets into a frame of the camera. With that too, two stage detection will be needed in this area, for person detection to pass the face detection to the face recognition system. Once the user is recognized, then the user gestures can be detected.

With a prototype demo of GestHome finished last semester, our main objective this term will be:

1. Develop a working program from the prototype, with both stages smoothly combined into one program
2. Research on various methods of improving the performance of GestHome, either in accuracy or frame rate
3. Add quality-of-life features that bring significant improvements to the program and our users

At the end of the semester two, our goal is to produce a working smart home system without any issue, with the architecture design shown in *Figure 1.2*

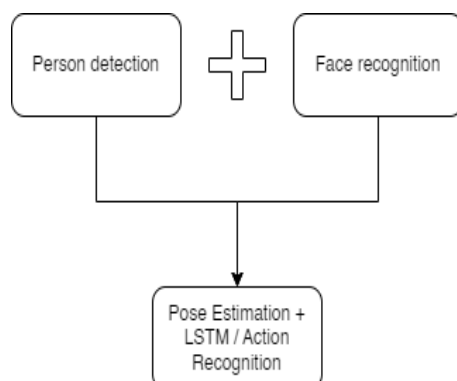


Figure 1.2: Overall architecture design of the project

## 1.4 Glossary

Terminologies that are not used as much and more specific in term of computer vision or AI explained in *Table 1.1*

Table 1.1: Terms in the papers that are used

Terms	Explanations
Action Recognition	A task in computer vision to identify actions of a person identified in the video or image.
Artificial Intelligence	Systems or machines that mimic human intelligence to perform tasks and can iteratively improve themselves based on the information they collect.
Bounding Box	A rectangle determined by coordinates of a certain object detected on an image. Usually used in object detection task to describe the spatial location of an object.

*Continued on next page*

Table 1.1 – *Continued from previous page*

<b>Terms</b>	<b>Explanations</b>
Computer Vision	A field with the focus of enabling computers to process and understand visual side, including videos and images.
Convolutional Neural Network	A Deep Learning that assume input as images, with the capability to differentiate the objects that contained in the image.
Deep Learning	A part of artificial intelligence and machine learning where neural networks are designed to learn and improving on its own by examining the algorithm. This remove the need to supervise the training process in machine learning.
LSTM	A recurrent neural network that has the ability to not only process a single frame of image or a single data point, but also entire sequences of data.
Machine Learning	A part of artificial intelligence utilizing the use of data for the algorithm to learn a certain task, gradually improving its accuracy in the task.
Neural Network	A computing system that have the resemblance of human brain, mimicking the way brain send neural signal to one another. Each signal passed in the system will adjust the weight on each neurons. This weight output from one neuron will affect the next node in the connection.

*Continued on next page*

Table 1.1 – *Continued from previous page*

<b>Terms</b>	<b>Explanations</b>
Object Detection	A task in computer vision designed to detecting instances based on the features available to classify the class in digital images and/or videos.
Pose Estimation	A task in computer vision designed to predict and track the location of a person or object. For human pose estimation, the things that are tracked are keypoints of human joint, where its unity will resemblance of a human pose.
Recurrent Neural Network	A type of neural network where the architecture allow it to have a cycle in between. This allows output of the network to depend on the prior elements within the sequence. This technique is used for ordinal or temporal problem, such as natural language processing, image captioning, language translation, etc.
YOLO	An abbreviation for You Only Look Once, a revolutionary object detection system that utilize deep learning to achieve real-time object detection.



# Chapter 2

## Related Work

### 2.1 Object Detection

The interest in object detection started in 2012, during the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [51]. This competition focused on building an efficient object detection algorithm using the ImageNet database as the training dataset for the algorithm. In this competition, Krizhevsky et al. (2012) has decided to approach the challenge by building an object detection method using Convolution Neural Network (CNN), called AlexNet. With CNN, AlexNet manage to perform much better compared to its rivals, hence the rise of CNN on object detection algorithm.

Furthermore, research on Deep Neural Networks has accelerated after ILSVRC 2012, and one improvement they have made is on the training. Since deep neural networks are harder to train, Kaiming et al. [36] has developed a residual learning framework to ease the training of deeper neural networks. Also, the residual networks manage to gain accuracy from the increase depth. The framework has helped on development of deep neural network, especially for many visual recognition tasks.

### 2.1.1 Two-Stage Detection Framework

There are two famous framework on CNN-based object detection algorithm. The first one is two-stage detection framework. Two-stage detection framework works by using region proposals to create Region of Interest (ROI) [31]. These ROIs then converted to feature maps using CNN, which then resize to extract the features that will be used by SVM for object detection and bounding box regressor to adjust the bounding box size. This method is more time efficient since the algorithm does not need to detect all parts of an image; instead, it focuses on parts that have high chance of an object present. This method created a model family called R-CNN model family. In this project, we are analyzing two examples from the family, which are R-CNN and Fast R-CNN.

R-CNN is the first CNN-based object detection algorithm after Krizhevsky's algorithm. [30]. The algorithm works in 3 different modules. The first module generates around 2000 region proposals. After that, for each region proposal, the algorithm extracts a dimensional feature vector using CNN. Finally, the algorithm utilize SVM to classify the object present in the region proposal. This algorithm achieved better accuracy than Krizhevsky's algorithm; however, R-CNN still has several drawbacks, mainly in the training and testing speed. Since it uses feature extraction from each object proposal and we have around 2000 region proposals, it consumes a lot of space, roughly hundreds of gigabyte of storage. Furthermore, during testing, the algorithm is slow to detect the object in an image, taking around 47 seconds per image.

Due to these issues present in R-CNN, Ross Girshick, the creator of R-CNN, decided to modify parts of R-CNN to improve the performance and make the algorithm more time efficient, which he called Fast R-CNN (Fast Region-based CNN) [29]. Fast R-CNN utilize similar object detection algorithm as

R-CNN; however, with Fast R-CNN, instead of using 2000 region proposals to extract feature vectors with CNN, Fast R-CNN only uses one fixed-size region that has been through Region of Interest (RoI) pooling to extract feature vectors. Furthermore, Fast R-CNN uses a more optimized softmax classifier and bounding box regressors in one stage instead of a softmax classifier, SVM, and regressors in three different stages. Because of this, Fast R-CNN manages to reduce the time for object detection and model training, going from 84 hours to 9.5 hours, while simultaneously keeping similar or in some cases better accuracy when compared to R-CNN.

Besides aforementioned algorithms, there are other algorithms that is derived from the R-CNN family. Algorithms such as Faster R-CNN[69] and T-CNN[44] aims to improve the performance of R-CNN based algorithms. Faster R-CNN improves the performance on Fast R-CNN algorithm by using deep convolutional network[35] to determine the region proposals instead of selective search[87], used in Fast R-CNN. Whereas T-CNN focuses on utilizing temporal and contextual information such that it improve the algorithm performance when applied to videos.

### 2.1.2 One-Stage Detection Framework

Although Fast R-CNN manage to greatly improve the performance of R-CNN, it still struggles with lengthy training time. With that, a new architecture for object detection has arised, in the form of one-stage detection framework. While two-stage detection framework has two stages for the object detection (using region proposals, object classification, and bounding box regression), one-stage detection framework only uses feature extraction network and convolution layers for predicting the object and adjusting the bounding box. This meant that one-stage detection framework uses less stages for object detection, and thus

improve the performance of the algorithm. However, there are problems in one-stage detection framework, mainly in its inability to detect smaller objects. In this project, we are analyzing two of the most famous one-stage detection framework model, Single Shot Multibox Detector (SSD) and You Only Look Once (YOLO).

Single Shot Multibox Detector (SSD) is one of the object detection algorithm that utilize one-stage detection framework [55]. It uses VGG16 Network for feature extraction and several convolution layers for detections, predictions, and bounding box adjustments. By using feature extraction and convolution layers for its model, SSD manages to achieve higher mAP in PASCAL VOC2012 and COCO dataset compared to any R-CNN model family. At the same time, SSD manages to achieve higher performance than Faster R-CNN (46 FPS vs 7 FPS).

Another famous one-stage detection framework model is You Only Look Once (YOLO) [68]. The model is designed based using 24 convolutional layers followed by 2 fully connected layers. It also utilize DarkNet during training. The model used in YOLO is similar R-CNN, where it proposes potential bounding boxes and score it using convolutional features. However, unlike R-CNN, YOLO only uses 98 bounding boxes instead of 2000. Furthermore, YOLO model has spatial constraints to help mitigate multiple detections of the same object and it combines individual components to a single optimized model, unlike R-CNN. This means that YOLO manages to achieve higher performance than Faster R-CNN (45 FPS vs 7 FPS); however, it comes at slightly lower accuracy, with around ten percent lower mAP compared to Faster R-CNN (73.2 vs 63.4).

### 2.1.3 Other Methods

Although two-stage and one-stage detection framework model are mostly used for object detection, there are some popular object detection algorithms that uses neither framework. Yang et al. [98] propose a new method of using multi-task learning [72] to improve the performance of existing object detection method. To demonstrate that, the research uses YOLOv5 as the base of this model, with the addition of multi-task learning such that this model is able to perform object detection and semantic segmentation at the same time. This model manages to achieve higher accuracy (around 9 - 11 percent higher) compared to other semantic segmentation algorithm such as PSPNet [104], BiSeNet network [99], and ASPP network [19].

## 2.2 Face Recognition

Face recognition (FR) is a method of detecting and identifying a person with their face and it has been used in many areas[94]. Interest of FR started with the introduction of Eigenface approach [86]. It developed to research of FR in multiple areas, from low-dimensional representation to local-feature based [94]. However, in 2012, a new approach of FR using deep learning has started with the winning of AlexNet [51]. Several famous Deep-learning based FR [7, 66, 83, 76, 47, 74, 5] are reviewed in this paper. BlazeFace [7] develops its' model from [55] with additional modifications such as enlarging receptive field size compared to other neural network architectures (such as MobileNet [[40],[73]]), using feature extractions *BlazeBlocks*, reducing the Pooling Pyramid Network architecture [43], and implement suppression algorithm to reduce error from larger spatial resolution. YOLO5Face [66] redesign YOLOv5 to be used for face recognition with modifications to the architecture such as the use of SPP, PAN, and SILU

activation function. DeepFace [83] implements a deep neural network and affine transformation to the align and represent step in typical FR framework to reduce error. LightFace [76] implements the typical face recognition framework in the background in TensorFlow and Keras to make it lightweight. It also provides mainstream face detection models [47, 74, 5] for users to use. Dlib [47] is a toolkit developed to build machine learning models with functions such as classifications and regressions. It also has several functions for FR and face detection in Python. FaceNet [74] removes the intermediate bottleneck layer used in previous deep-learning based FR and uses a trained deep convolutional network to optimize, improving the performance of the model. OpenFace [5] uses CLNF [4] as the base of OpenFace model, with some changes. These changes include: adding a validation step using CNN, using separate point distribution set for eyes, lips, and eyebrows, and additional features such as head pose and eye-gaze estimation.

### **2.2.1 One-Shot Learning and Siamese Neural Network**

One-shot learning is another method of object classification in Machine Learning by predicting from only one image per category. Research on one-shot learning has started in the 2000s, with the usage of bayesian approach for one-shot learning [27] [58]. The one-shot learning method proposed is used to differentiate object classes, such as bicycles and person, cars and motorcycles, and so on. Later on, another approach for one-shot learning emerge, utilising transfer learning instead of bayesian approach [53]. With transfer learning, the model gain knowledge from each iteration and transfer it to the next iteration, which makes it more effective in one-shot learning and this approach shows impressive performance compared to previous approaches. Because of the transfer

knowledge capability, one-shot learning has been utilized for handwriting and signature recognition, where the model performs really well in recognizing handwriting and signatures [52]. With the advancement of neural networks on image detection in early 2010s and how effective the model is [51], researchers are finding ways to incorporate neural networks in one-shot learning. A special type of neural networks is Siamese Neural Network (SNN). SNN was first introduced in 1994, where it was used to detect if a signature is original or forged [14]. SNN is able to do that thanks to one of its prominent feature, the identical neural network structure. SNN contains 2 or more identical neural network structure, where it takes 2 inputs and calculate the similarities of both inputs from the feature vectors [84]. The similarities then run through a loss function to determine if both images are the same based on the similarity value. Because of that, there has been numerous research on face recognition and identification using SNN [49] [102]. There are advantages of using SNN in one-shot learning. The biggest advantage is that SNN does not need to train its' model when there are new classes added to the database, compared to other neural networks where there is a need to retrain the model when there is a new class added [56]. Another advantage is that for SNN to work well, it only needs a single image, whereas other neural networks might need a lot of data to be able to accurately detect. These advantages are here due to the SNN's structure of using similarity value of two images. However, there are disadvantages of using SNN in one-shot learning compared to other neural networks. The main disadvantage is in computational power, where SNN requires higher computational power during training due to the identical neural network structure [10].

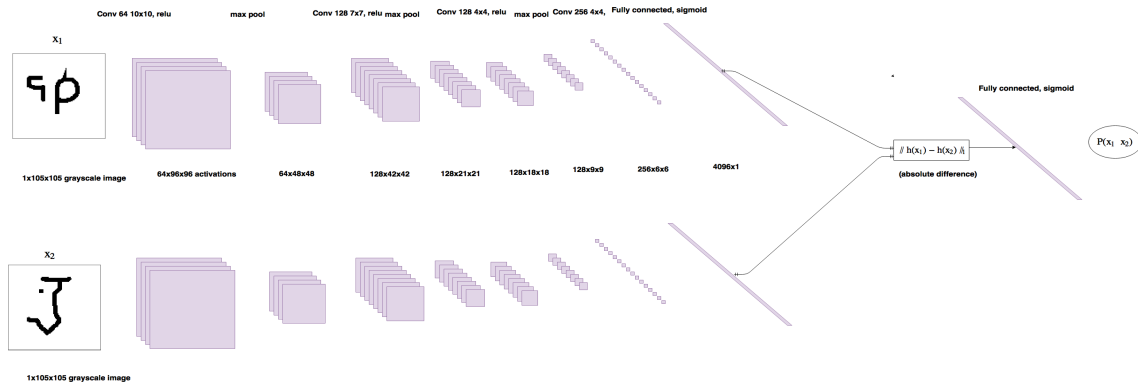


Figure 2.1: Architecture of Siamese Neural Network

## 2.3 Pose Estimation and Keypoints Analysis

Pose Estimation and Keypoints Analysis has been an interest in computer vision. This subject focuses on how to predict and track the position of a person, either in an image or a video. During the last few years, there have been many advances in this subject and new pose estimation algorithms are being made, either to improve the performance or accuracy of pose estimation. Currently, the most famous approach of pose estimation are top down approach and bottom up approach. We will discuss both strengths and weaknesses of both approach in this paper, and show some examples of it.

The top down approach relies on the model detecting the presence of human being in a frame, then determines the keypoints to outline the pose, with the advantage in accuracy but suffers in performance. Examples of top down approach in pose estimation [26, 25, 92, 2, 59, 32, 67, 6, 81] are reviewed in this paper. RMPE [26] utilize CNN for pose estimation and Pose NMS for pose redundancy elimination, which is implemented and improved in AlphaPose [25] by using SIKR to accurately localize keypoints and Pose-aware Identity Embedding to simultaneously track poses. HRNet [92] focuses on making a pose



estimation and object detection using high-resolution images by making high-resolution convolution stream. YOLOPose [[2],[59]] build their model by using YOLO [68] as the object detection framework; however, both differ in their pose estimation. Amini, et al. [2] utilize a 6D Transformer to perform pose estimation and keypoints regression [1], whereas Maji, et al. [59] utilize loss function to perform pose and keypoints estimation. Both algorithm shows noticeable accuracy improvements compared to similar algorithms, although both are incomparable to each other due to their objective and dataset difference. DensePose [32], tackles the same issue with a different approach. Where most models focus on developing their framework first, DensePose focus on gathering a dataset, consisted of dense correspondences between SMPL model [57] and person appearing in COCO dataset, then use the resulting dataset to build a CNN object detection model, based on Mask R-CNN [34]. Improvements were made in another version of DensePose [67] where they create a lighter, less layer R-CNN model, which resulted in increase in model performance. BlazePose [6] differs from typical top down approach models by detecting the torso or face instead of whole body. To complement this approach, Bazarevsky et al. utilize a fast face detector[7] and design a pose estimation model partly inspired by Stacked Hourglass approach [63]. Pose Estimation Transformer (POET) [81] extends the work done in DETR [17] with the addition of a transformer-based architecture to predict human poses in parallel and a set prediction loss that is a linear combination of simple sub-losses for classes, keypoint coordinates, and visibilities. It contains three main elements, a CNN backbone using ResNet [36], a encoder-decoder transformer based on [17, 89], and pose prediction head.

Bottom up approach relies on the model detecting human keypoints, then classify which keypoints belong to the same person and connect them. The method has an advantage of performance compared to top down approach, but

suffers from accuracy. Examples of bottom up approach [21, 95, 3, 28, 15] are reviewed in this paper. HigherHRNet [21] uses similar backbone as HRNet [92] with addition of high-resolution feature pyramid to predict high-resolution heatmaps that are beneficial for detecting small person, which is more efficient compared to HRNet. LitePose [95] provide more improvements to HRNet [92] and HigherHRNet [21] by converting multi-branch architecture in HigherHRNet to single-branch architecture using gradual shrinking, removing redundancies found in HigherHRNet. Furthermore, LitePose also use fusion deconv head and large kernel convolution to enhance its' capacity. MoveNet divides the model into three parts: encoder, mapper, and decoder, and uses a combination of Convolution, Max Pooling, Dense, and Upsampling layers in the model. Furthermore, it utilizes LeakyReLU as the activation function after each convolution layer. Disentangled Keypoint Regression (DEKR) [28] utilize disentangled representations [9] for the representation to accurately learn the keypoint region, therefore the predicted keypoints are inside the keypoint regions. OpenPose [15] utilize Part Affinite Fields (PAF) for body parts association in pose estimation. The research is based on [16], with the modifications on the model mainly in PAF, by increasing network depth but removing body part refinement stage, improving performance and accuracy of the algorithm.

With the massive amount of research conducted around pose estimation, it is noted that the models made are getting increasingly more complex. Because of this, Xiao et al. [96] researched the possibility of making simple baselines for pose estimation models by using a simpler model, consisting of a few deconvolutional layers added on a backbone network, ResNet, similar to the one in [36]. The research showed that the simple model is capable of pose estimation with a promising result, managed to perform slightly better than the winner of COCO2017 keypoint Challenge's models. The result of this result showcased

that a simple model can be as effective as a complex one in pose estimation, which our research will keep in mind during building our model. Furthermore, another research [62] compares two popular pose estimation model from top down and bottom up approach, BlazePose [6] and OpenPose [15], and found out that BlazePose manage to surpass OpenPose in terms of real world performance. This research demonstrate that, even though bottom up approach are generally has better performance compared to top down approach, with the right architecture, a top down model can surpass bottom up model in terms of performance.

## 2.4 Action Recognition

Action recognition, also known as human activity recognition, is the process of identifying and classifying human actions or activities from a video stream or a series of images [37]. This task is challenging due to the need to determine when the action started and ends. Even though it is challenging, but it is also an important research topic in computer vision and has various applications such as surveillance, sports analysis, and health monitoring. The interest started with the usage of holistic features [12], then progressed to local features, and finally with deep learning with [51] popularize the technique.

Deep learning has shown remarkable success in action recognition due to its ability to automatically learn high-level features from raw data, without the need for handcrafted features. Convolutional Neural Networks (CNNs) have been the most widely used deep learning approach for action recognition. The usage of this approach improves the accuracy as CNN performs well with images [42]. Despite the current situation where Masked AutoEncoders (MAE) [33] are becoming the State-of-the-arts solution for action recognition tasks, like

in this case, VideoMAE V2-g is able to achieve 99.6 3-fold accuracy in UCF-101 dataset, 95.9 Top-5 accuracy in Something-Something V2 dataset, and 88.1 accuracy in HMDB-51 dataset [93], CNN still is a very fundamental approach that inspired the emergence of masked autoencoder. Not only that, CNN itself is still proven to be competitive in terms of accuracy, as one of the research that use CNN as their approach, is able to achieve one of the best accuracy in UCF-101 dataset, with 98.2 accuracy. This shows how CNN is relevant to the task.]

Some researches [50, 82] focus on hand gesture detection and recognition as part of action recognition. Kopuklu et al. [50] create a hand gesture detection using lightweight CNN and hand gesture classification using deep CNN. Sung et al. [82] improves on an existing hand skeleton tracker model, *MediaPipe Hands* [103], with modifications to improve keypoint accuracy and the ability to do a 3D keypoints estimation. And for classification, it combines heuristic methods and neural networks to classify gestures.

Other method that is available for action recognition is through Graph Convolutional Network (GCN). Even though this neural model is more recent than CNN, but this model allows faster training time and higher predictive accuracy in some of the tasks that are able to utilize graph [48]. It's able to make the model to be semi-supervised learning, due to its nature of how everything is able to be parameterized. This causes flexibility and eliminate the limit of normalization of matrix in layer-wise propagation, by turning the layer-wise propagation into vector. In reality, this approach is also generalization of CNN.

$$f(H^{(l)}, A) = \sigma \left( AH^{(l)}W^{(l)} \right)$$

Figure 2.2: ReLU Layer-wise propagation rule

$$h_{v_i}^{(l+1)} = \sigma \left( \sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)} \right)$$

Figure 2.3: Graph Convolutional Layer-wise propagation rule

With these two different approach, generally it is used for different tasks. Almost every tasks for action recognition uses CNN, as it's bale to utilize image well, and able to separate instance for picture. To name some of the tasks that uses CNN in action recognition, there are Temporal Action Detection ([18], [78]) ; Action Classification ([106], [79], [101]); Spatio-Temporal Action Detection ([24], [39]). CNN also works well in case where there is a need to recognize the interaction between human and object and many other tasks. However, even though CNN is able to take care of skeletal-based action recognition, GCN works better comparably, that most of the State-of-the-art solution in this task are using GCN rather than CNN.

### 2.4.1 Skeletal-based Action Recognition

Skeletal-based Action Recognition is different, since the task require precise mapping of joints, where it might not be able to accurately locate joints, since it might be influenced by noises that may received from the input video data, such as background color, clothing and other noises. Because of that, most of the time, this task would use keypoint extraction mentioned in section 2.3 to get the joints mapping. This resulted the input for this task to be in form of

graphs, where GCN is more suitable for this result <sup>1</sup>. Since the data that is passed through this task is in vector form, this causes the algorithm able to achieve high accuracy and near real-time performance.

Recent works of skeletal-based action recognition [20, 23, 54, 97] are discussed in this paper. Channel-wise Topology Refinement Graph Convolution Network (CTR-GCN) [20] improves on Graph Convolution Network model by simultaneously learning the topologies and channel-specific correlations using channel-wise topologies. PYSKL [23] is a open-source toolbox, based of PyTorch, and designed to support different action recognition models from GCN and CNN, such as CTR-GCN [20] and 3D Convolutional Networks [85]. PYSKL is also developing a variation of ST-GCN [97] with modified architecture to more reflect the current solution, named ST-GCN++. Hierarchically Decomposed Graph Convolutional Network (HD-CGN) [54] creates sets of joint nodes for edge extraction, highlights the dominant edge sets, and apply a new ensemble method that uses only joint and bone stream.

## 2.5 Long Short-Term Memory

Unlike the other sections discussed in this material, Long short-term Memory (LSTM) is not a part of computer vision. LSTM is a neural network, part of AI. Rather than continuously passing input that is processed in the previous layer, LSTM as the part of Recurrent Neural Network, allows feedback connection. This feedback connection allows the network to process entire sequences of data, with example of speech or video. The architecture of this neural network also allows the unit to protect memory content, while also processing activation pattern step-by-step and add them to the memory after processing [38]. This

---

<sup>1</sup>There are cases where CNN is also used, but it requires more sophisticated technique to extract the keypoints

behaviour allows the network to process short-term memory that can last for a lot of steps. LSTM network are well suited for issues that can be solved by gradient-based approaches, such as classification, prediction on time series and many else. Despite it's older age compared to most techniques, LSTM is proved to be effective, even for the recent implementations. In 2018, one team consisted of five neural network trained on a single layer, 1024-units LSTM model is trained to play a game of Dota 2, a game that is well known for a complex mechanics and strategies and is able to beat teams of amateur human teams [64, 11]. In 2019, an Artificial Intelligence called AlphaStar, built on LSTM network beats one of the world strongest professional player in one of the most complex video game, Starcraft 2, won convincingly with the score of 5-0 [90]. This prove that this network is still well used in the industry. Not only that this model is able to be used in time series prediction, but it is also used in human action recognition, speech recognition, rhythm learning and music composition, handwriting recognition, market prediction and more tasks that is able to be solved through artificial intelligence, especially healthcare.

# Chapter 3

## Implementation

### 3.1 General Framework

With the object detection and pose estimation model sorted, we have decided on a general framework for the model we use on GestHome, shown in *Figure 3.1*. The flow of the model will go as follow

1. A person enters the Field of View of a camera-equipped smart home device, making them the input for our model
2. The model will detect the person and using object detection, it will create a bounding box around them
3. The model will scan the image inside the bounding box and using facial recognition, it will search for a face inside and match it to a database
4. When the person's face match with the face in database, the model will use pose estimation to extract keypoints
5. With the extracted keypoints, the model uses LSTM for action recognition



6. With the action recognize, the program will do the command related to the action

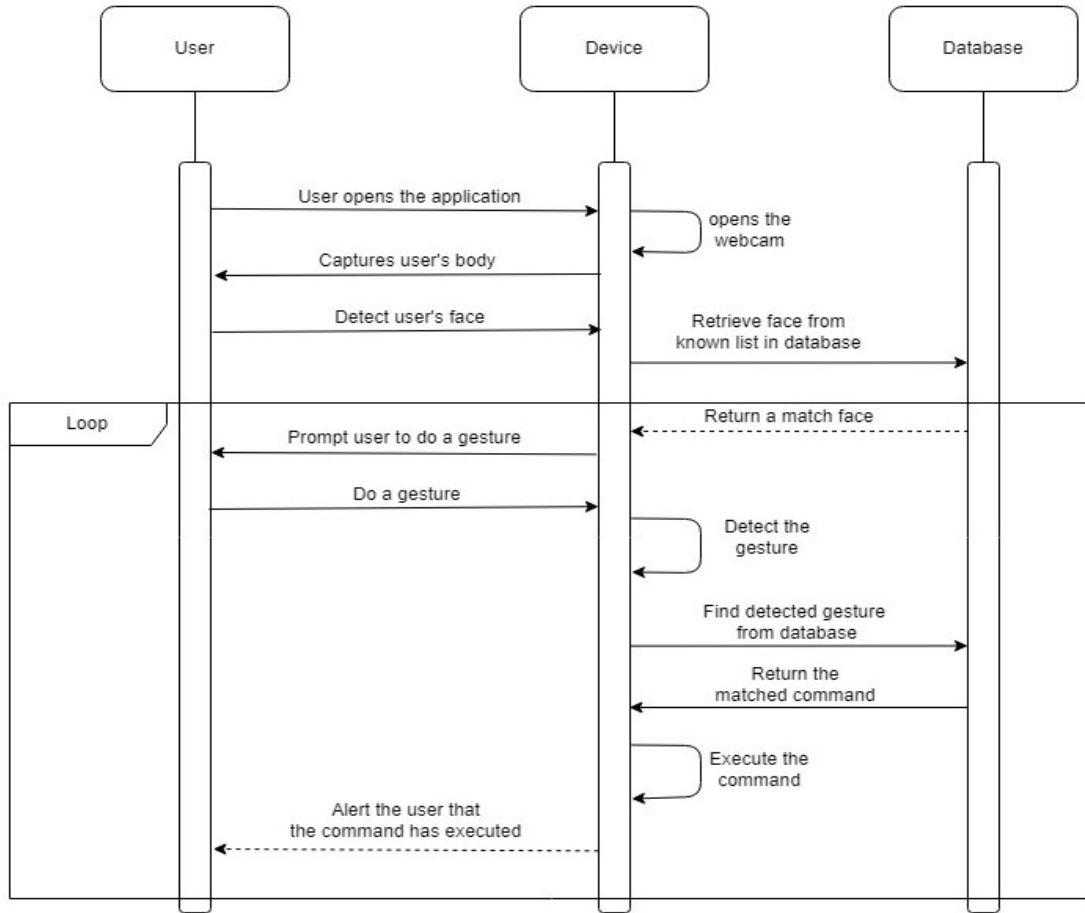


Figure 3.1: Sequence Diagram of the General Framework

From *Figure 3.1*, initially, we are planning to create a single stage model, with all components of the model (Object detection, Face recognition, Pose estimation) in one, streamline pipeline. The problem comes during combining of these components. Previous researches combines either object detection and face recognition [66] or object detection and pose estimation [[59],[2]], but not both of them at the same time. Here, we began the research more on multi-task

learning, as a part that is often used by different researcher to solve this issue.

Multi-task Learning in itself is a subsidiary of machine learning, where multiple task are learnt together in parallel through a shared model. This approach reduces the data that are needed to train machine learning tasks, reducing overfitting and allows optimization on training on related tasks. Not only that, but multi-task learning also allows model for different tasks to be trained faster, and has the capability to reuse the previously known to augment the model for more complex tasks. There are different methods of multi-task learning, with some of the most used one named task grouping and overlap, and transfer of knowledge. Because of its flexibility, multi-task learning are used in different tasks that are solvable through deep learning. In Computer Vision, multi-task learning are mostly used in changing the architecture, for the architecture to partition the network into task-specific and allows generalization while minimizing negative transfer [22].

In some cases, multi-task learning can hinder the performance of the model. Depending on the tasks, multi-task learning can be outperformed by single-task learning. Furthermore, tackling a large and diverse task at the same time can be a challenge for multi-task learning, shown in [88]. This meant that depending on the tasks, using multi-task learning might be less efficient than just relying on single-task learning. Furthermore, research that involve multi-task learning applies it only on the the head of the architecture, consisting of at most 2 or 3 tasks. Because of that, it is not yet possible for us to build a whole architecture using multi-task learning, that is capable to accommodate every task inside the architecture.

With multi-task learning not as efficient and more power-consuming than we first thought, we decided to split the framework into 2 stages, as shown in *Figure 3.2*. The 2 stages mentioned are: Object Detection and Face Recognition

in stage 1, and Pose Estimation and Action Recognition in stage 2. Since both stages are related and crucial to the smart home system, we have decided on combining both algorithms into one, using Flask as our back-end framework for the project. Furthermore, to store the necessary data for our program to run, we decided to use MongoDB as the database. The scheme of our program is shown in *Figure 3.3*.

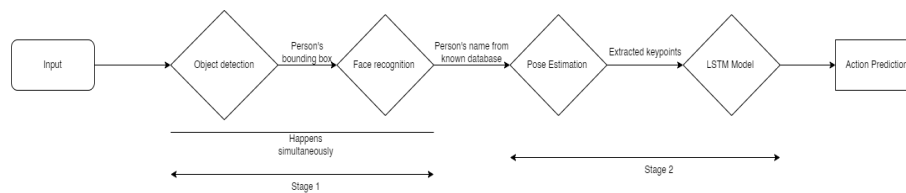


Figure 3.2: General framework of GestHome

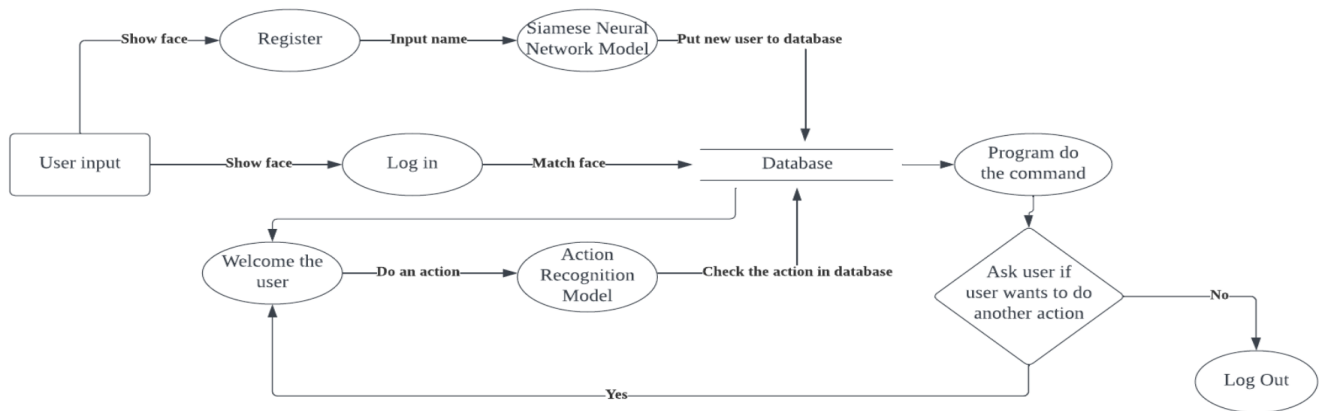


Figure 3.3: Data Flow Diagram of GestHome

## 3.2 Stage 1: Object detection and Face Recognition

### 3.2.1 Object Detection

With the goal of human detection, we have selected potential object detection models that is suitable for our research, which are YOLO, Faster R-CNN, and SSD. Furthermore, since our objective is to implement the model on smart home devices, our main goal is to find the most efficient model that is available for real-time detection. Based on previous research, we have decided to utilize YOLO for object detection due to its performance. Compared to R-CNN, YOLO is much more suitable and provides much better real-time performance. Whereas with SSD, although SSD provides slightly higher accuracy compared to YOLO, we ultimately chose YOLO due to better performance. In the middle of our research time, we also discovered that there is a newly published paper and repository on the newest iteration of YOLO, YOLOv7. Before, we have been doing trials on YOLOv5, which is the current iteration of YOLO with the most update and community support, despite not having an official paper. This means there are 2 versions of YOLO that we can use in this project and because of that, we have decided to do inference runs on both models, YOLOv5 and YOLOv7, and we will evaluate which model suits GestHome the most based on speed and accuracy.

### 3.2.2 Face Recognition

Face recognition is necessary to identify the person. There are a few face recognition models we are able to choose to be used in GestHome [74, 66, 47, 5]. In the end, we decided to implement dlib [47] to our framework as the face recogni-

tion model. We implement face recognition system that is available in dlib due to its' simple model, which is beneficial for our performance-minded framework, and the built-in face recognition functions is sufficient for GestHome.

We notice that dlib performs pretty well in detecting our faces; however, if we want to add more faces, we have to change both the code and database to include that person. Furthermore, we also need to train the model again whenever a new person is added to the database, which is inefficient for a smart home system with the possibility of adding new faces. Because of that, we have implemented a one-shot learning, which enables us to add new images to the dataset without the need to retrain our model. One-shot learning utilize Siamese Neural Network (SNN), a special type of Convolutional Neural Network. We implemented the SNN based on *Figure 3.4*, which takes an input of 2 105x105 images, run it thru a series of neural networks, feature vector, and sigmoid loss function, with the result of the similarity value of both images. After that is done, we compare the similarity value to our determined value; if the similarity value is higher than the determined value, that means both images match and vice versa.

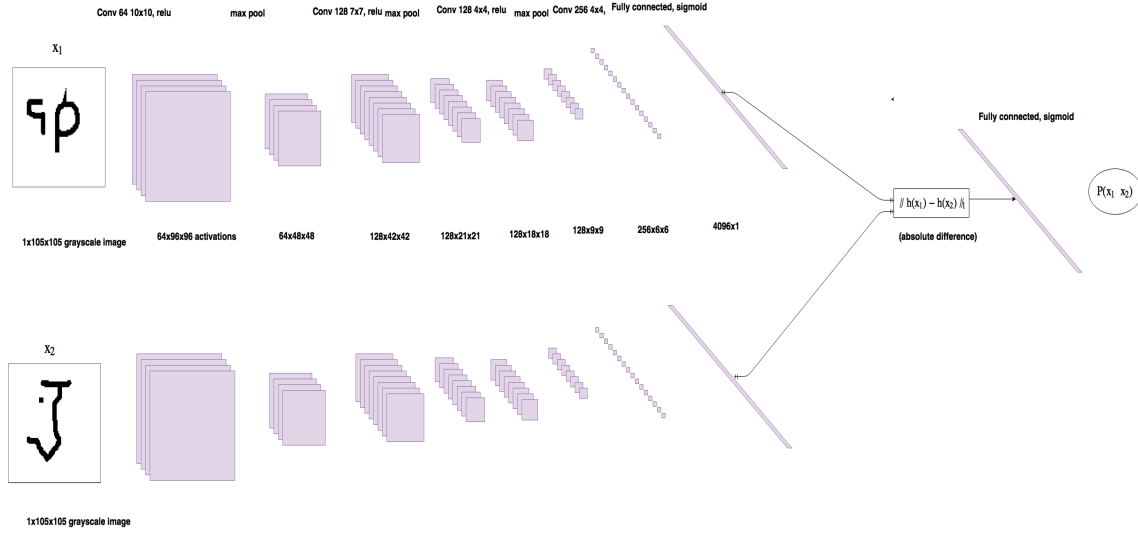


Figure 3.4: The architecture of the Siamese Neural Network

With increase interest on Siamese Neural Network (SNN), there are several implementations of SNN for face recognition with slight different models between each implementations [8] [70] [41]. We decided to implement those three models and compare the accuracy results of the 3 models. We will refer Behera's model [8] as Model 1, Renotte's model [70] as Model 2, and Jain's model [41] as Model 3.

As we can see from the three figures, *Figure 3.5* managed to perform the best from all three models, reaching up to 90 percent accuracy, whereas the other two stays in around 70 percent of accuracy. Although Model 1 managed to beat Model 2 and 3, we encounter an issue where Model 1 might randomly not function as intended, and it resulted in a 50 percent accuracy, similar to a coin flip. This is something we would like to avoid, since having unreliable model will render GestHome useless to the user. We also encounter similar issue in Model 2, which means that our best option is using Model 3 as our face recognition model for GestHome.

Using Model 3 as our face recognition model leads to another problem,

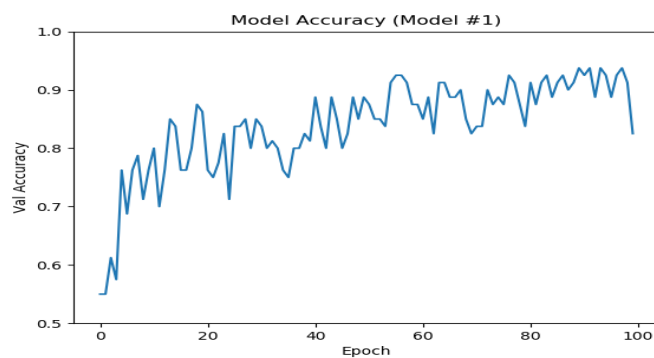


Figure 3.5: Accuracy result of Model 1

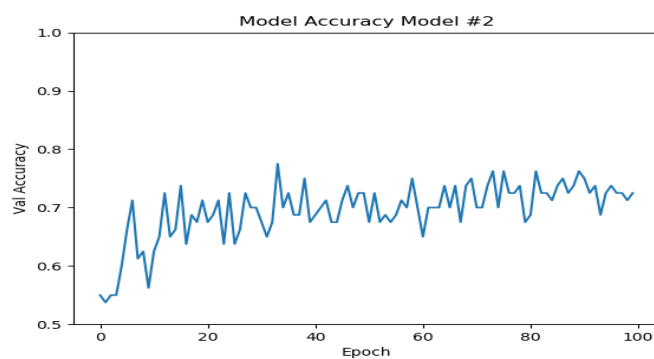


Figure 3.6: Accuracy result of Model 2

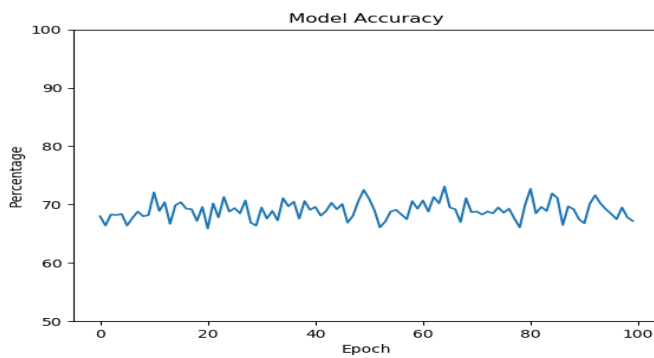


Figure 3.7: Accuracy result of Model 3

accuracy. 70 percent accuracy is not as high as expected, especially when it involves security, since we want to ensure that our system can only be accessed by listed individuals. Because of that, we have built a system where instead of validating once, it will validate the user multiple times. This idea can be done by capturing the user face for multiple times, with each time, the picture of the user is compared to the pictures in our database. Then we keep a record of the comparison result, and determined the user name based on the highest amount of certain name appeared in the record. We found that this is a good compromise on the accuracy issue since the multiple validations happen for only around 3 to 5 seconds, which we believe is not too long for the user to wait.

While SNN is better for GestHome than dlib, it is unable to do face detection. Because of that, we have decided to combine dlib for the face detection and SNN for the face recognition. To increase the accuracy of dlib correctly detecting faces, we utilize dlib HOG and Linear SVM, based of [71]. With dlib, we detect faces that appears in the picture, crop the face, and use the cropped picture as the input for SNN.

## 3.3 Stage 2: Pose Estimation and Action Recognition

### 3.3.1 Pose Estimation

Another crucial part of our model is pose estimation. Similar to object detection, we began to find different papers and implementations that are available on GitHub. Our criteria in choosing the libraries of implementations and papers are

1. Performance in Low Performance Device (using CPU)



2. Accuracy and Precision
3. Maturity of the model
4. Existing paper and conference entry of the paper
5. Community support on the model

Ultimately, we managed to get several pose estimation models that suits our criteria, such as AlphaPose, OpenPose, LitePose, Yolo-Pose, BlazePose, HR-Net, HigherHRNet, Mo0veNet, PoseResnet, and Lightweight Openpose. The models chosen have different architecture, with a good combination of top-down approach and bottom-up approach, as shown in *Table 3.1*. This proves that, contrary to popular belief regarding bottom-up approach has better performance than top-down approach, there are top-down approaches that can match the performance of bottom-up approaches. To mimic the goal of this research, we have decided to run inference of the aforementioned models on our machines. However, in that stage, we encounter some issues about some of the models.

Table 3.1: Approaches of different pose estimation tools

Method	Model architecture
<b>HRNet</b>	Top-down
<b>HigherHRNet</b>	Bottom-up
<b>BlazePose</b>	Bottom-up
<b>AlphaPose</b>	Top-down
<b>LitePose</b>	Top-down
<b>Openpose</b>	Bottom-up
<b>PoseResnet</b>	Top-down
<b>YOLO-Pose</b>	Top-down
<b>Lightweight Openpose</b>	Bottom-up
<b>Movenet</b>	Bottom-up

### 3.3.2 Problem encountered

During our inference testing, we encounter several issues with some of the pose estimation models. HRNet, HigherHRNet, and LitePose utilize similar architecture for their models, and from their research paper and GitHub page, they develop their models using multiple NVidia GPUs. This meant it is impossible for us to do an inference testing of their models on our machine. We have tried to do inference run of the models on CUHK CSE GPU Server, and the issue still persists. Although LitePose has a mobile version, we could not find a source code for it, hence we could not replicate it to run on our machine. Furthermore, some models are unable to run inference on our machine, hence we remove it from our inference testing. For models that manage to run, we have done an inference run and use their pre-trained model to measure the performance. From these constraints, we have decided in testing the performance of 6 models [15, 6, 25, 95, 65, 3].

## 3.4 Action Recognition

As the main part of the project, having a robust action recognition system is important. However, there is still a constraint for us to make sure GestHome to work in lower computational devices. In the previous semester, we build our own action recognition model, using LSTM. Through this approach, we provided 4 pre-determined poses that is trained over LSTM, so it's able to predict the pose shown the movement that is used by the users.

Due to the nature of current model, there are ways for us to improve the model. This doesn't necessarily means that we need to change everything, but we have decided to do keep the keypoint extraction as is. This decision also was made clear due to BlazePose accuracy.

In the current model, LSTM can be said that it is not sufficient due to number of poses that it can currently detect, and the fact that there are inaccuracies in trial, where if a motion can be detected even with the slightest resemblance of it. This also brings a point where the model might find it hard to distinguish two similar motions.

There are two candidate for Action Recognition model, each are trying to achieve different goals that we have. The models that we are focusing to evaluate in this semester are MotionBERT [105] and PYSKL [23]. One of the goal is to get increase the accuracy to the action recognition system, so the algorithm won't register a wrong movement. Both models also uses different method to solve this issue, where both of them also uses different techniques to get the motion accurately. That is why, we will be evaluating the use case of each models, to figure out which model should we use to replace the current LSTM model we have.

## 3.5 Website building

To connect both stages of GestHome as a coherent program, we have decided to implement a website using Flask as the framework. We have designed a few pages for GestHome, where it covers the functionality of the program, such as the home page, log in page, face recognition page, and so on. We have added pages if unexpected things happen, such as if a user is not recognized by GestHome, the user will have options to either go back to the main menu, try to log in again, or register his/her face to the system.

### 3.5.1 Specification

#### External Interface Requirements

- **User Interface**

The design of this user website complies with the design guidance based on Google Chrome, Microsoft Edge and Mozilla Firefox. Important user interface qualities that fulfill the guidelines are:

1. **Minimalist Design**

Having minimalist design is helpful for the user, since it makes user having easier time when using the application. Reducing the unimportant components to exist in the design helps the design to look cleaner and improve visibility for the website.

This aspect of quality can be seen throughout the entire website pages, where for every pages, it has minimal design and easy to access functionalities. With black and white design, it looks pleasing for the user too.

2. **Concise**

In addition to having minimal design, the functionalities of the website is concise, preventing over-clarifying. Labels of functionalities are clearly defined, makes user understand the functionalities of the website faster.

### 3. Error recovery and prevention

With the complexity of different parts coming together, it is easy for website to have errors. It is important for the users to not see any errors, so they are able to experience the website without any confusion.

- **Hardware Interface** The web page should be accessible by any device with a browser application. However, for the hosting server, it is required to be used in any operating system with Linux kernel, as the functionalities of the websites are dependent on the libraries that are available on Linux.
- **Communication Interface** This web page uses internet connection (by any means, whether it is WiFi, modem or LAN). In addition to that, this website requires availability of camera and permission given to access the camera.

## Functional Requirements

### 1. Login

Summary	Login to system if the user has an account in the database. Else, the user has to register for an account
Actor	User
Trigger	Login Button
Scenario	The website will prepare the user to open the camera. Once ready, the system will try to match the face of the user with the existing face in the system local database.
On success	The user will get in to the platform in their account
On failure	The user will be guided to the failed login page

### 2. Register

Summary	Register account to the system
Actor	User
Trigger	Register button
Scenario	When pressed, the system will prepare to take the photo of user's face. After putting the User name, the system will take the user's face, then immediately log the user in.
Post-condition	Log the user in and add the user into the user database

### 3. Failed login

Summary	Giving options for user to do further action
Actor	User
Trigger	Failed login
Scenario	When the login is failed because the user does not exist in the user database, then it will give option to try to login again or register the face
Post-condition	Execute the selected button

## 4. Motion Detection

Summary	Giving options for user to do further action
Actor	User
Trigger	Motion Detection button
Pre-condition	Must be log in to the registered account in the system
Scenario	When this button is pressed, the system will capture the motion made by the user. The system will then capture ten movements before getting the aggregated motion wanted to be made by the user. The option of the motions will be explained below.
Post-condition	Put the motion into the action queue in the database

## 5. Motion Detection Execution

Summary	Execute the action given by the user through Motion Detection option
Actor	API listener
Trigger	New entry of movement in queue database
Pre-condition	Need to have a new entry placed in MongoDB
Scenario	API listener will always have the script running to check if there is a new entry in the queue. If there's no queue, then it will keep listening to the database. Else, it will match the action with the associated motions, then it will call 3rd party API associated to the motion.
Post-condition	The result of the API will be spoken by the text to speech.

## **Non Functional Requirements**

1. Usability
  - Intuitiveness
2. Security
  - Permission required for camera access
  - GDPR-compliant
3. Reliability
  - Cloud database, available for 99.995
  - Software Testing
4. Availability
  - Detailed logging for every execution

## **Design Constraints**

- Requires a stable internet connection
- Requires access to webcam
- Video will look like stuck after execution



### 3.5.2 System Architecture

#### Architecture Diagram

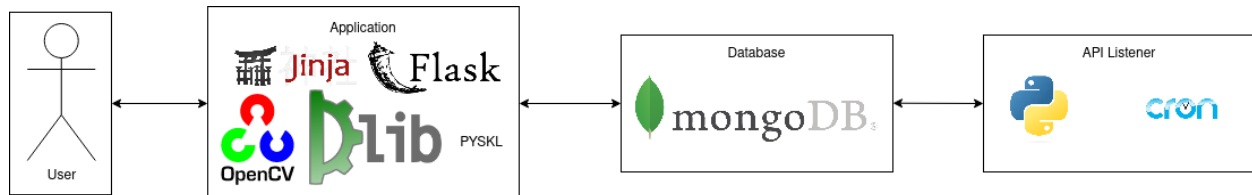


Figure 3.8: Architecture diagram of the system

#### Data Flow Diagram

- Login

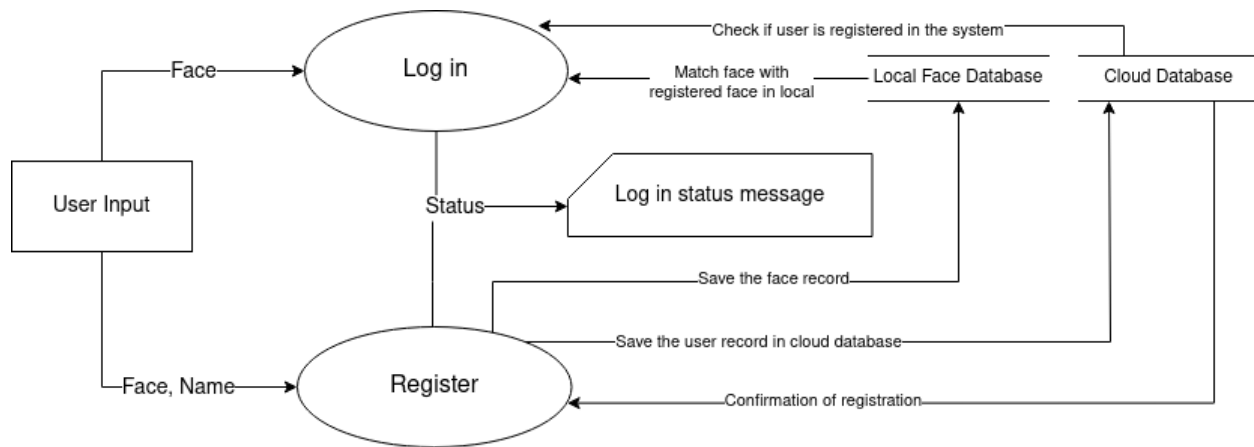


Figure 3.9: Login flow of the system

- Motion detection

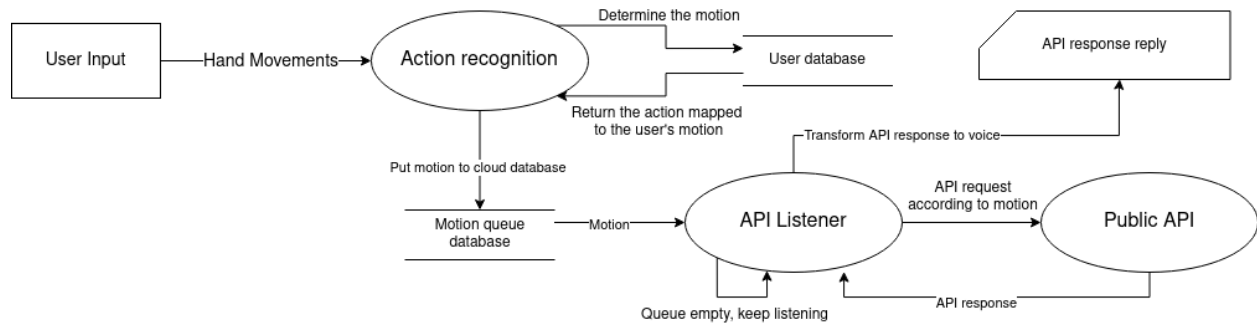


Figure 3.10: Motion detection flow of the system

## Sequence Diagram

- Login

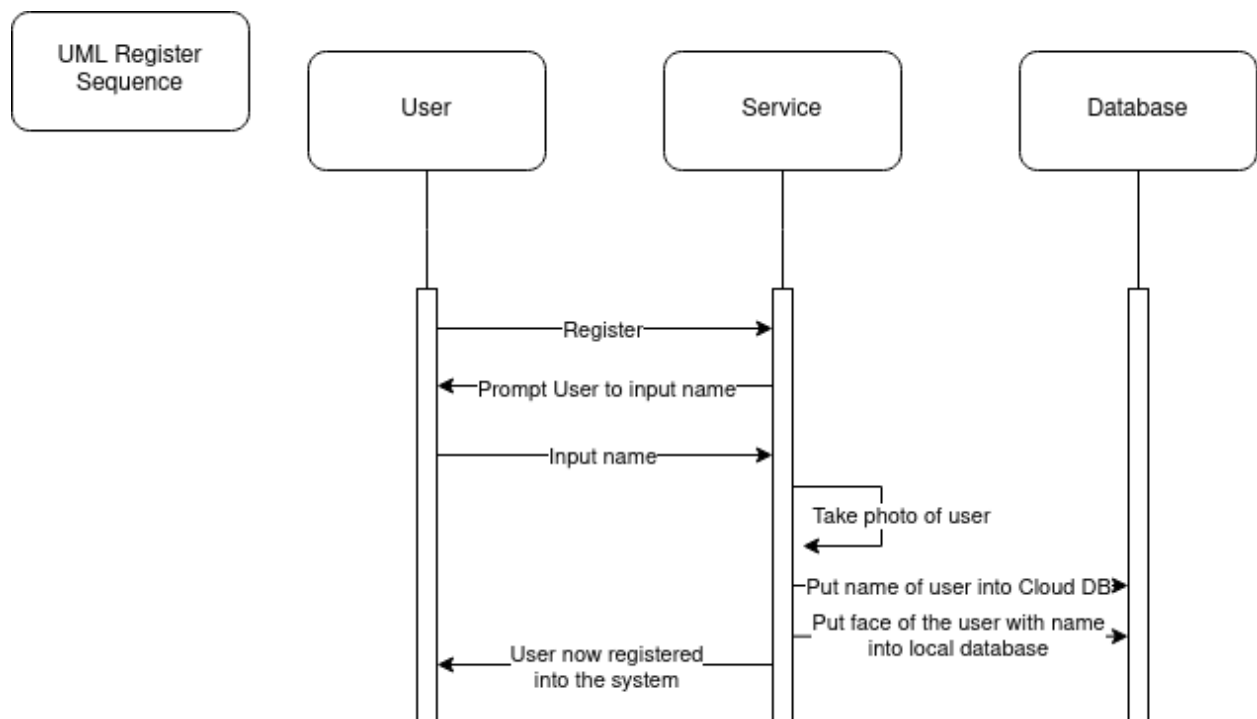


Figure 3.11: UML Sequence Diagram for Login

- Register

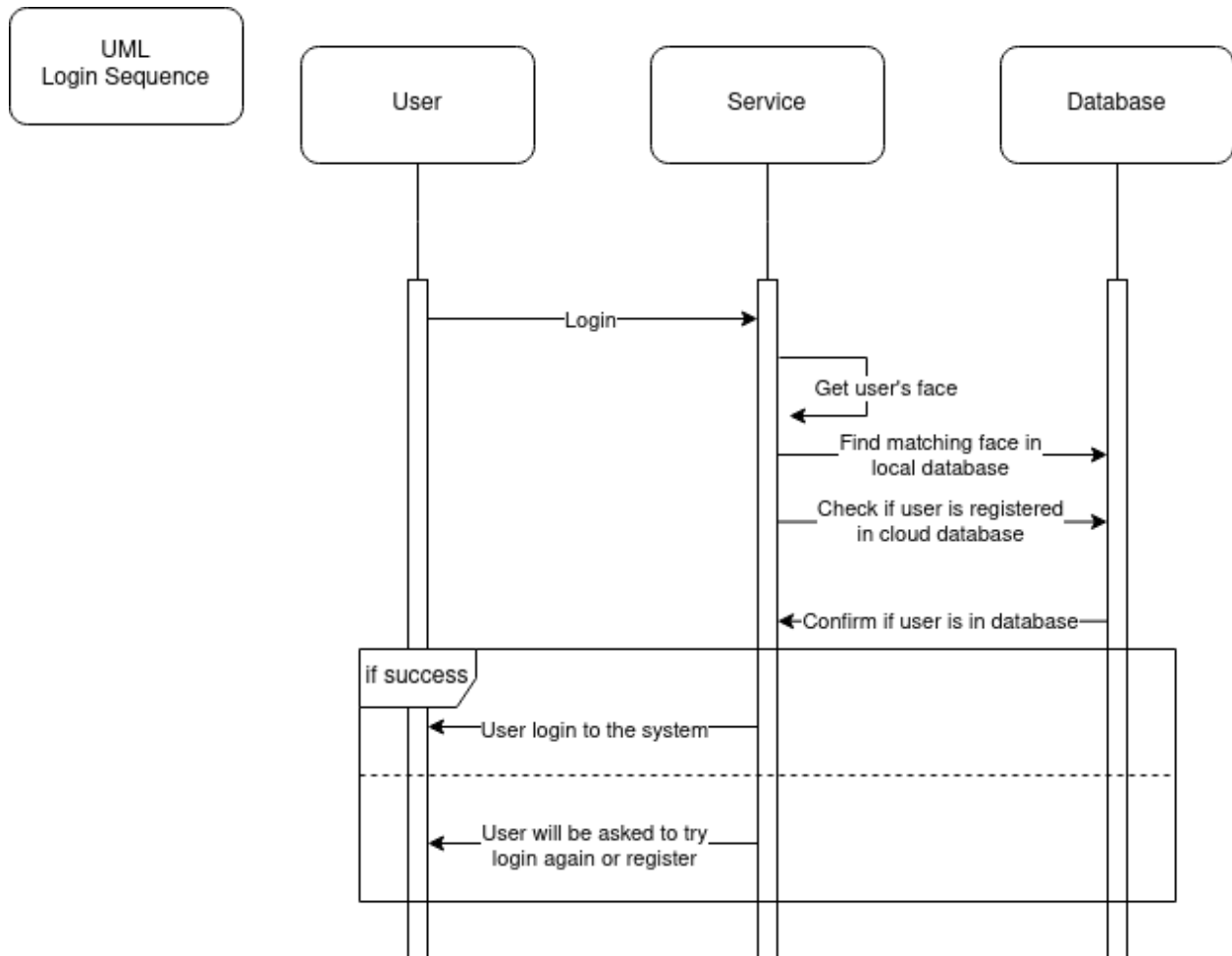


Figure 3.12: UML Sequence Diagram for Register

- Motion Detection

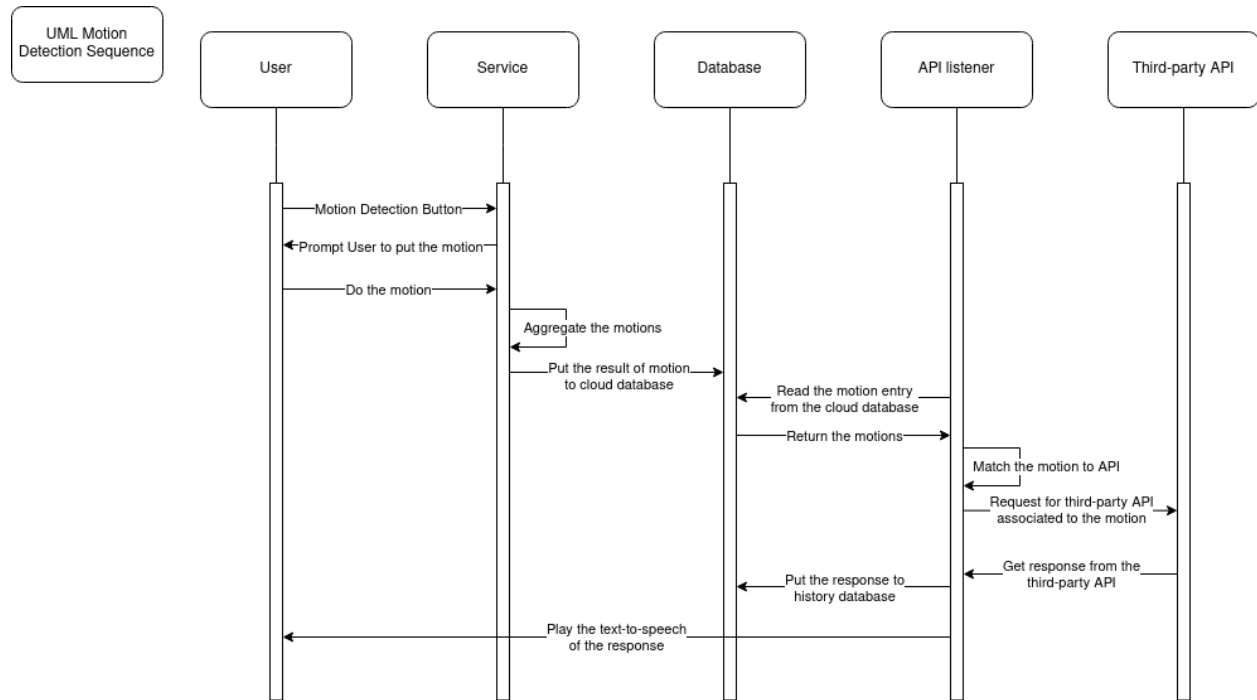


Figure 3.13: UML Sequence Diagram for Motion Detection

### 3.5.3 System component

#### OpenCV

OpenCV helps us in the development of both stages. Here, OpenCV also enable the usage of camera usage for the main functionalities.

#### Flask Framework

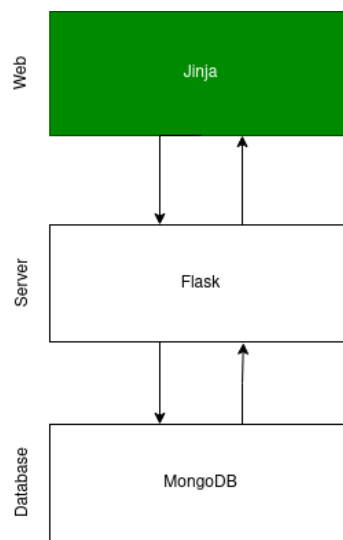


Figure 3.14: Caption

- Database: MongoDB to store our data, in particular, the user data, the action queue, and history database.
- Front-end: Jinja is the template engine that is fully integrated through flask, enabling it to use the state and component from backend to be viewed.
- Back-end: Flask is a full-stack framework based on Python, where it enables REST API usage and operations, handling HTTP request and responses, and integrate it with the Jinja

## API Listener

- Database: Since the database is available on the cloud, this allow the API listener to be run on the other device. This API listener is dependent on the database, since it only execute the command once there is a new entry to the database.
- Text-to-Speech: Every result of API processed will be read using text-to-speech through the API host speaker.
- Third-party API: The API listener uses different public APIs that have some functionalities.

Preset movements:

Motion	API
Dislike	Holiday API ( <a href="https://holidayapi.com/">https://holidayapi.com/</a> )
Fist	Hong Kong Weather API ( <a href="https://www.hko.gov.hk/">https://www.hko.gov.hk/</a> )
Like	Cryptocurrency API ( <a href="http://api.coinlayer.com">http://api.coinlayer.com</a> )
Mute	Useless Facts API ( <a href="https://uselessfacts.jsph.pl">https://uselessfacts.jsph.pl</a> )
OK	Sunrise-Sunset API ( <a href="https://api.sunrise-sunset.org">https://api.sunrise-sunset.org</a> )
Palm	Air Quality API ( <a href="http://api.airvisual.com">http://api.airvisual.com</a> )
Peace	Bible Verse API ( <a href="https://bible-api.com">https://bible-api.com</a> )
Stop Sign	Tech Facts API ( <a href="https://techy-api.vercel.app">https://techy-api.vercel.app</a> )
Three-Left	USD-HKD Currency API ( <a href="https://cdn.jsdelivr.net">https://cdn.jsdelivr.net</a> )
Two Up	Number Trivia API ( <a href="http://numbersapi.com">http://numbersapi.com</a> )

Table 3.2: Available preset movements to API

If any of the movements are not registered to the API, then the listener will response that the movement given has not bound to any API.

### 3.5.4 User Interface Design

#### Landing Page

**Welcome to GestHome!**

Log In

Register

#### After Login Page

**Welcome Back, Albert!**

What would you like to do?

Motion detection

Log Out

## Register Page

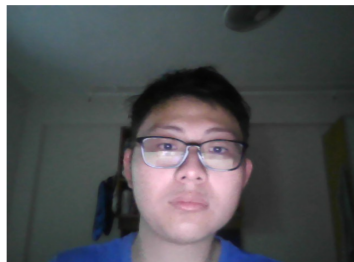
**Please enter your name:**

## Login Page

**Please wait a moment**

while we are analysing your face...





## Login Failed Page

**I am sorry, but I couldn't recognise you...**

What would you like to do instead?

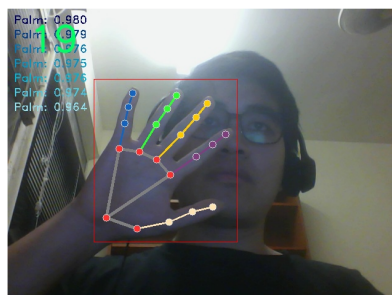
[Log in again](#)[Register](#)[Back to Home](#)

## Motion Detection Part

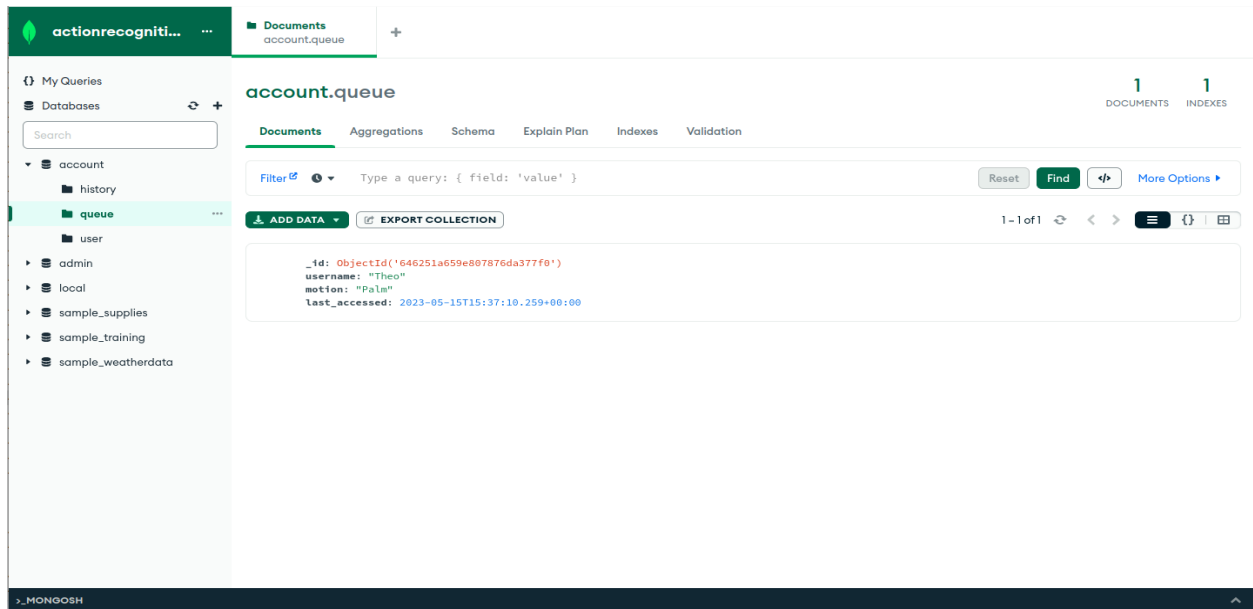
← → ↻ 127.0.0.1:5000/detect\_motion? ☆ ⌂ ⬇ ⌵ ≡

**Please wait a moment**

while we are analysing your move...

[Stop Detecting](#)

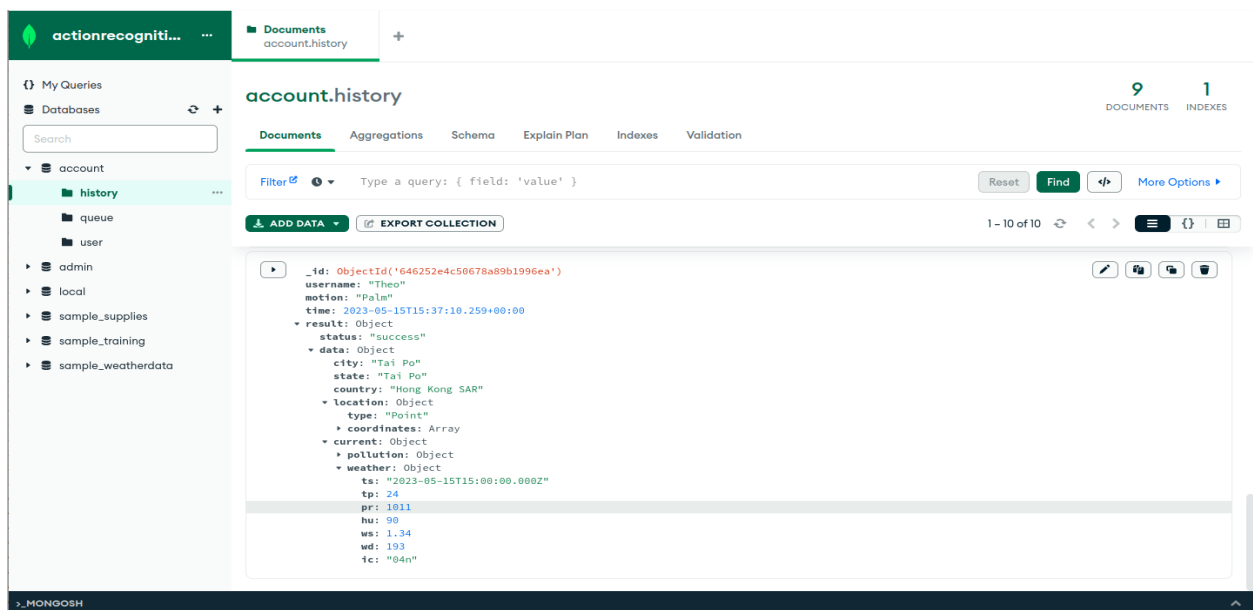
## Queue Database Entry (Before API listened)



The screenshot shows the MongoDB Compass interface. On the left, the 'Databases' sidebar lists 'account', 'history', 'queue', 'user', 'admin', 'local', 'sample\_supplies', 'sample\_training', and 'sample\_weatherdata'. The 'queue' collection is selected. The main panel displays the 'account.queue' collection with 1 document and 1 index. The document content is:

```
{
  "_id": ObjectId("646251a659e807876da377f0"),
  "username": "Theo",
  "motion": "Palm",
  "last_accessed": "2023-05-15T15:37:10.259+00:00"
}
```

## History Database (After API listened)



The screenshot shows the MongoDB Compass interface. On the left, the 'Databases' sidebar lists 'account', 'history', 'queue', 'user', 'admin', 'local', 'sample\_supplies', 'sample\_training', and 'sample\_weatherdata'. The 'history' collection is selected. The main panel displays the 'account.history' collection with 9 documents and 1 index. The document content is:

```
{
  "_id": ObjectId("646252e4c50678a89b1996ea"),
  "username": "Theo",
  "motion": "Palm",
  "time": "2023-05-15T15:37:10.259+00:00",
  "result": {
    "status": "success"
  },
  "data": {
    "city": "Tai Po",
    "state": "Tai Po",
    "country": "Hong Kong SAR"
  },
  "location": {
    "type": "Point",
    "coordinates": [
      [101.1, 90]
    ]
  },
  "current": {
    "pollution": {
      "weather": {
        "ts": "2023-05-15T15:00:00.000Z",
        "tp": 24,
        "p": 1011,
        "hu": 90,
        "ws": 1.34,
        "wd": 193,
        "fc": "04n"
      }
    }
  }
}
```

### 3.5.5 Challenges in Website Implementation

#### Flask integration with OpenCV

As OpenCV is not officially supported by Flask, making use of the webcam and everything requires us to make some different tweaks in Python. With this tweaks, we manage to make the camera to work. However, due to variable locality of the OpenCV video capture, this doesn't allow several cases of usage to use camera twice in a row. This might also cause the camera to stop working and the whole website to crash.

By isolating each instances of video capturing in OpenCV to each functionality that uses them, we are able to utilize every functionality without any error caused by the OpenCV error.

#### Stream Context

To explain more about this, it requires a bit of explanation on the code. In all the functions that make use OpenCV, and some of the processing through it, it is only possible for Flask to put the video to Jinja by generator function. This will allow the video to be changed into jpeg format that is able to be shown to the front end easily, frame by frame, so it looks like a video. However, with this generator function, it does not allow any function to return two values freely.

At first, we tried to have return statements in the generator. However, the more we use it, the more we realize that return in generator function only acts as StopIteration, where it is not flexible to be used. In case where the generator function only have limited amount of result, it would be easy to resolve this issue. The problem is, the OpenCV will accept input all the time until

the condition for stop video capture is achieved. This result that the returned value will include the output of the OpenCV camera output. Another thing is that this method forces the user to do every operations twice. Another method that we tried is to jsonify the result, thinking that it would help the result to be processed after. However, this would result more errors, as view function are not able to return json, as the data is not active to receive response context.

Our last resort is to look at the Flask documentation to see if there is any thing that we can use to enable this function easily. Reading the documentation, we found Streaming Context, where the basic functionality is to invoke the generator function to generate data and pass the function data result to a direct response object. With this functionalities, it bypasses the need to have generator and view. For every JPEG frame sent to the Jinja template engine, then it will wait until the request data produced by that generator function to be returned, by letting the request context active along with the view function. This what enables the request data to be passed after the view function iteration.

## Performance Drop

Despite our trial on each stages run in a separate window of OpenCV having good performances, both in FPS and accuracy, packaging the model into a format into h5 format or any format that support the model to be accessible through Flask, significantly reduces the performance. This part will be later discussed in the evaluation part for stage 1 in more detail.

# Chapter 4

## Evaluation

### 4.1 Stage 1

#### 4.1.1 Object Detection

With our concerns regarding YOLOv5 and YOLOv7, we decided to do inference testing on both models to determine which model suits the best for our use case.

```
video 1/1 (103/109) C:\Users\dorer\Documents\FYP\yolov5\WIN_20221029_23_27_06_Pro.mp4: 192x320 1 person, 1 tv, 73.0ms  
video 1/1 (104/109) C:\Users\dorer\Documents\FYP\yolov5\WIN_20221029_23_27_06_Pro.mp4: 192x320 1 person, 1 tv, 69.0ms  
video 1/1 (105/109) C:\Users\dorer\Documents\FYP\yolov5\WIN_20221029_23_27_06_Pro.mp4: 192x320 1 person, 1 tv, 69.0ms  
video 1/1 (106/109) C:\Users\dorer\Documents\FYP\yolov5\WIN_20221029_23_27_06_Pro.mp4: 192x320 1 person, 1 tv, 73.0ms
```

Figure 4.1: Inference testing using YOLOv5

```
video 1/1 (104/109) C:\Users\dorer\Documents\FYP\yolov7\WIN_20221029_23_27_06_Pro.mp4: 1 person, 1 tv, Done. (101.0ms) I  
nference, (2.0ms) NMS  
video 1/1 (105/109) C:\Users\dorer\Documents\FYP\yolov7\WIN_20221029_23_27_06_Pro.mp4: 1 person, 1 chair, 1 tv, Done. (9  
7.0ms) Inference, (1.0ms) NMS
```

Figure 4.2: Inference testing using YOLOv7

From *Figure 4.1* and *Figure 4.2*, we can see that YOLOv7 manages to pick up a detail that YOLOv5 miss (A chair), which indicates that YOLOv7 has higher precision than YOLOv5. However, it is slower when compared to YOLOv5 ( 70ms vs 100ms, both models are using smallest possible pre-trained

model) during inference testing. Because our project mainly focus on the performance of the models instead of precision or accuracy, we decided to use YOLOv5 for our project.

### 4.1.2 Face recognition

Early on, we figured out that the face recognition that is based on dlib is really easy to use. What we need to do is only to put a picture of a person into a known list, then the algorithm will draw similarities that is read on the camera through OpenCV. From there, it is able to the compare it and give annotations on the face that is detected on the camera.

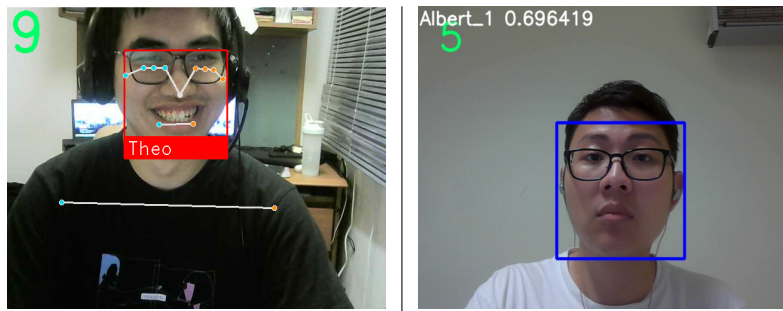


Table 4.1: Face Recognition using dlib only (Left) vs dlib and Siamese Neural Network (SNN) (Right)

One issue that we found on this part is that the face encoding that is detected might delay the performance of the framework in stage 1, since it takes some time to detect the face. Not only that, but this face recognition is pretty inconsistent in time given too. Because of that, we have decided to include One-shot learning with Siamese Neural Network (SNN) to improve it's ability to do face recognition. With SNN, GestHome manage to held performance similar to one without SNN. As shown in *Figure 4.3*, performance of face recognition algorithm with dlib and SNN manages to achieve a stable 8-9 FPS when a face

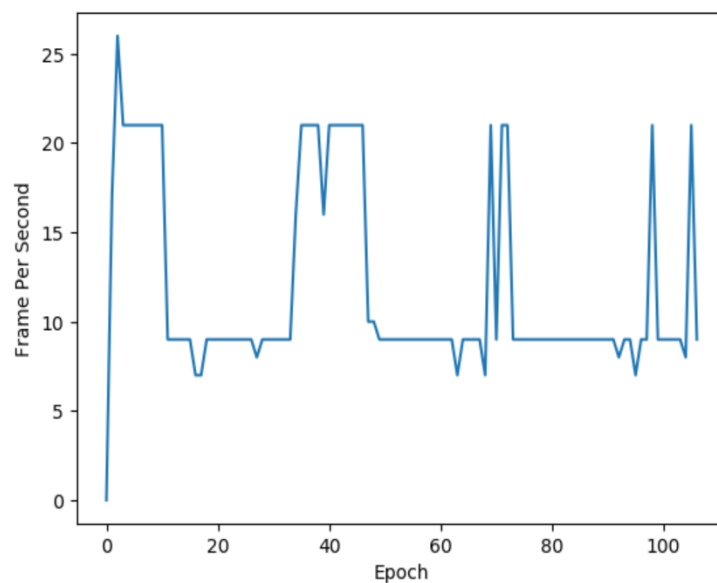


Figure 4.3: Performance record of dlib and Siamese Neural Network (SNN) Based of Frames Per Second (FPS)

is detected. This shows that SNN does not affect the overall performance of the face recognition system, while at the same time increase the accuracy, from 40-50 percent when using only dlib to 60-70 percent when using dlib and SNN, as shown in *Table 4.2*.

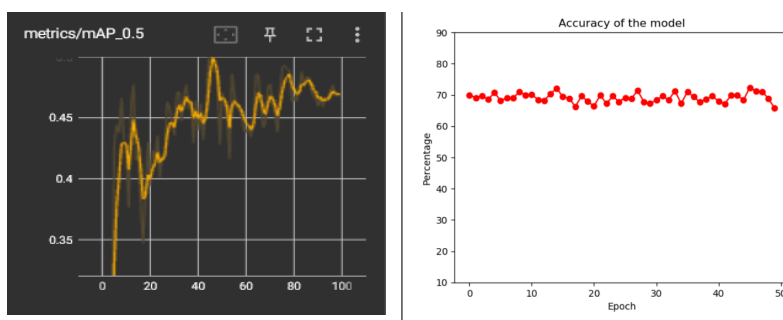


Table 4.2: Accuracy when using dlib only (Left) vs dlib and Siamese Neural Network (SNN) (Right)

## 4.2 Stage 2

### 4.2.1 Pose Estimation

For models that manage to run, we have done an inference run and use their pretrained model to measure the performance. *Table 4.3* shows the performance of these models, with Frames per Second (FPS) as the measurement of performance (Higher FPS equals to better performance). We notice that OpenPose, Lightweight OpenPose, and AlphaPose are not suitable for our use case due to the low real-life performance (0 and 3-4 FPS) and all three models are more suitable for devices with a dedicated GPU, which is not the intended target of our research. Because of that, for our research, we have decided to focus on the remaining 3 models, LitePose, BlazePose, and MoveNet.

Table 4.3: Performance of different algorithms

Method	CPU/GPU <sup>†</sup>	FPS
<b>OpenPose</b>	CPU	0
<b>BlazePose</b>	CPU	15
<b>AlphaPose</b>	CPU	3-4
<b>LitePose</b>	Phone	30-35
<b>Lightweight Openpose</b>	CPU	3-4
<b>MoveNet</b>	CPU	14-16

<sup>†</sup>Devices used:

Phone: Samsung Galaxy Note 10 with Snapdragon 855

CPU 1: AMD Ryzen 5 Pro 3500U with Radeon Vega 8

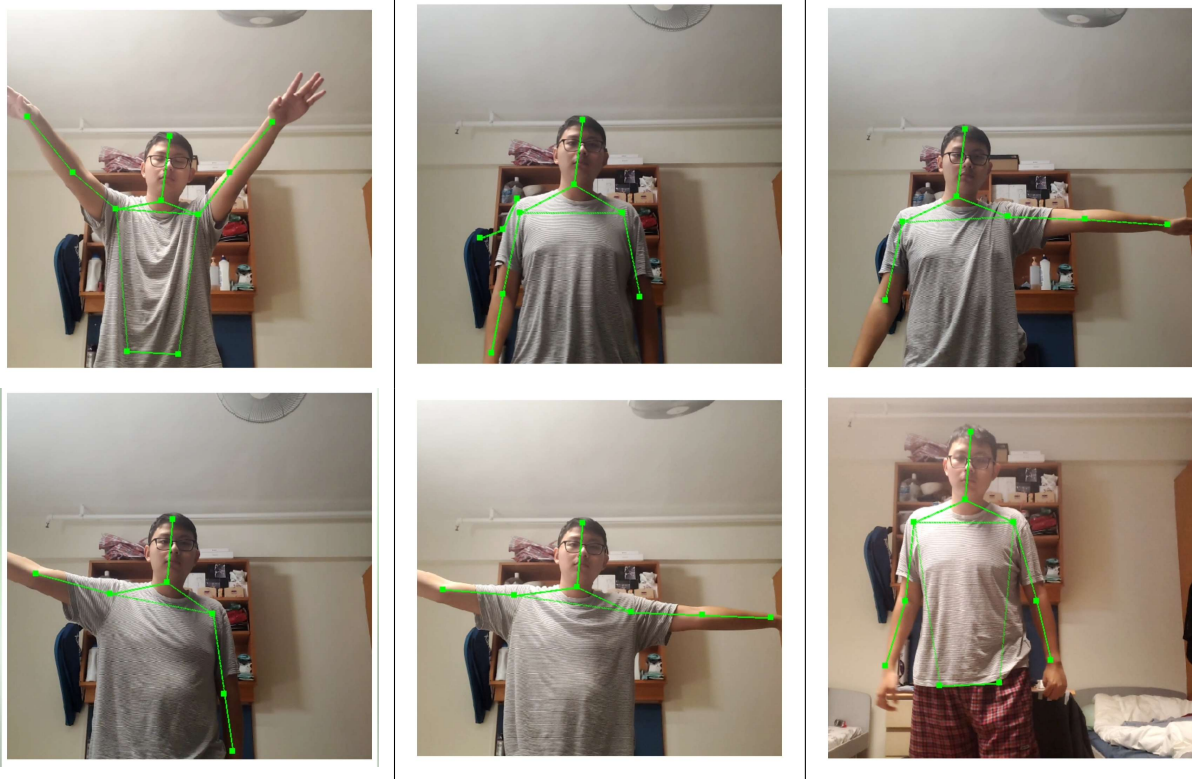
CPU 2: Intel Core i5-7300HQ with NVidia GTX 1050



### 4.2.2 LitePose

LitePose [95] is the most efficient architecture when it comes to pose estimation. Originally inspired by HRNet, this pose estimation architecture is able to remove redundant parts and improving the overall architecture, through Fusion Deconv and Large Kernel Convs, enabling it to perform in low computing devices. This also allows the architecture to perform with much lower latency compared to its predecessor while not sacrificing its accuracy.

Table 4.4: Litepose inference run on mobile



From *Table 4.4*, we can see that Litepose manages to estimate the poses correctly. However, with Litepose, there are moments where the model inaccurately estimate the poses, especially in regards to detecting background objects as part of a pose, shown in *Table 4.4*.

### 4.2.3 Movenet

MoveNet, released in 2022, is another architecture that is suitable for our project. The model offers great performance for real-time video, easy to use (offered as part of TensorFlow Hub), and it has different models for different goals. MoveNet uses a combination of Convolution layer, Max Pooling, Dense layer, and Up sampling layer. With Movenet, we manage to do an inference run on local device, and it manages to locate the keypoints correctly, as shown in *Table 4.5*. Compared to Litepose, it doesn't perform as smooth; however, pose estimation accuracy is better in Movenet compared to Litepose.

Table 4.5: Movenet inference run on CPU



### 4.2.4 BlazePose

Built for low end devices, BlazePose is another architecture fit for our project. Not only that it doesn't require a high computational device to operate it, it also has some perks, which are customizable through python, supports more keypoints and many others. However, we also find that there are times where this tool fall short, especially when handling pollution, where it is not able to determine correctly the keypoints on the body. This happens because of its nature of approach on bottom-up.

Table 4.6: BlazePose inference run on CPU



In conclusion, we found that LitePose, although have the best performance from *Table 4.3*, it suffers from slightly in accuracy, mentioning that the model is designed to be simpler than the original HRNet in favour of performance.

However, the biggest problem that LitePose have is the lack of support for non NVidia GPU machine, which makes it not suitable for our project. Meanwhile, BlazePose and Movenet perform very similar between each other; but, BlazePose manage to do this with more keypoints plotted (33 in BlazePose vs 17 in Movenet). Because of that, for our project, we are leaning towards using BlazePose as our pose estimation model, although we still keep Movenet as a second option.

### 4.2.5 LSTM Model

Succeeding in finding different pose estimation tools that are available to use, we began to find more on how to process those poses in a sequence where the poses detected connect together to form an action. Different researches points out that we are able to use and integrate Long Short-term Memory to our existing pose estimation. This is possible since the pose estimation system that we use enables us to only extract the keypoints from different body parts that are detected in the frame. In addition to that, for each and every single one of keypoints that are used in BlazePose are in form of arrays, this enables us to do concatenation for unification through numpy and thus enables us to have the structure to do training.

### Data Collection & Preparation

No model is able to be run in anything without the data. That is the realization that we had as we thought of this solution. Now that the keypoints are collected in form of numpy array, it is easier for us to collect sequences of image. Initially there are several datasets that are available and well known for action recognition, namely UCF101[80], Kinetics[46], Moments [61] and many more. However, we decided not to use those datasets for several reasons. First,

those dataset takes very long time to train, as the length / sequence that each dataset have are varying from 1 sec to 15 minutes. With the amount of data in one dataset, we believe that it will take some time to train all of them. Second, we envision that the actions that are to be detected in the model is simple gestures used in our daily lives, such as raising hand, swiping and rotating. As we look at the datasets, most of them contains obscure actions that we don't need and would be weird instead if implemented, such as breakdancing, high kick, tai chi.

Because of this reasons, we decided to take our own data. This could be done by recording 30 videos of 30 frames for each movements. This way, we would have sufficient data to train the activity for it to be recognizable. In a single frame, it contains the coordinates of keypoints extracted from that frame, in total 258 coordinates (x, y, z) for keypoints on pose, right hand and left hand captured through BlazePose. This frame by frame recording will imitate as if there are a certain movement done through the keypoints.

There is not much to be done for the data preparation before the model training, apart from splitting the data collected to be training and testing data.

## Model Designing

For us to have the machine learning model that has the capability to use every frame, it means that we need to use an algorithm that allows the training to go through the whole sequence that are available in the dataset we collected. Hence, we decided to use LSTM as the machine learning algorithm. Not only to just use one layer in the infrastructure, but also to use two layers to imitate multi-step forecasting for the movements. The rest of the architecture is explained through the *Figure 4.4*. Also, the architecture explained through Tensorboard would also help explain on the more detail on the graph, which

is shown in *Figure 4.5*. For this model, Adam is utilized in this model as the optimization technique since it can give one that can manage difficulties with sparse gradients and noise.

```
model = Sequential()
model.add(LSTM(128, return_sequences=True, activation='relu', input_shape=(30,258)))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
```

Figure 4.4: LSTM Model



# MotionBERT

MotionBERT was introduced in 2022, where it is trying to tackle the issue of heterogeneity of available data resource. With their method of functionality, they are able to utilize the different data that is able to be extracted in their algorithm, and able to use all the data in a way that helps them to have more accurate result of the motion. By having 3D pose representation, Mesh recovery



and skeletal-based action recognition, this model is able to become state of the art solution for 3D action recognition. This is what mentioned as unified way of learning human motion representation.

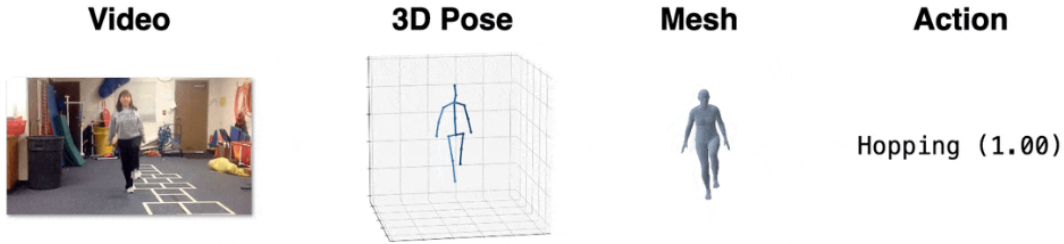


Figure 4.6: MotionBERT in action

The only problem with this model is that it is reliant on Alphapose. It in itself is not a problem, but since the way they got the data for the skeletal-pose data is through the output result of Alphapose, it means that it's only able to process the result after processing the video before. This causes this project to not be viable for real-time evaluation, especially mentioning the fact that Alphapose is not suitable for low computational device evaluation too.

## PYSKL (ST-GCN)

Although we mentioned earlier that PYSKL is a toolkit that is able to support multiple skeletal based action recognition, but it is also supported to be the official implementation of ST-GCN++, an improvement from ST-GCN [97], where they improved ST-GCN algorithm with improved architecture on spatial module and temporal module [23]. This modification help them to be able to reach the status of state of the art for both 2D and 3D action recognition based on NTU RGB+D.

Not only that, they also allow inference in CPU. Even though it is not compared to what is capable in NTU RGB+D model, but it has 15 pretrained



motions according to HaGRID [45]. The motions that are registered are Call, Dislike, Fist, Four, Like, Mute, OK, One, Palm, Peace, Rock, Stop, Three [Middle 3 Fingers], Three [Left 3 Fingers], Two Up.

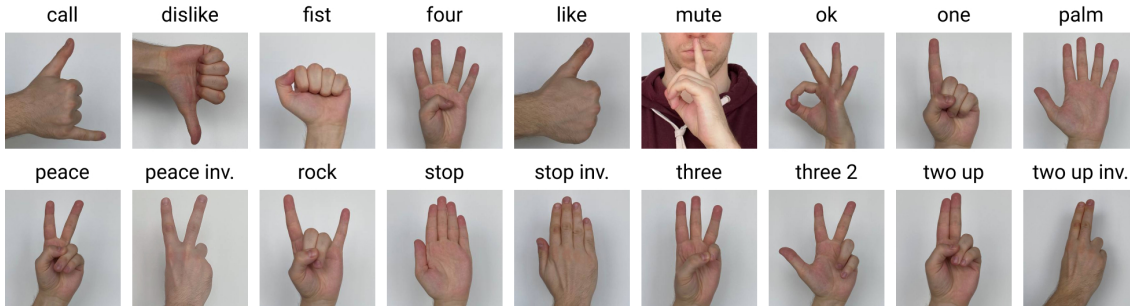


Figure 4.7: Gesture recognized through HaGRID

## 4.3 Overall evaluation

### 4.3.1 Accuracy evaluation

#### LSTM

On the training for the model, we thought that we need a lot of epochs for the model to learn., so we set it to around 1000 epochs. However, the training categorical accuracy reaches 99% accuracy pretty quickly. We decided to finish the training early in 60 epochs, so the model won't overfit. This decision is not just a hunch, as it is revealed in more iterations of model building that overfitting prone to happen quite quickly, and thus it is wise to use enough epoch for it to train. *Figure 4.9* and *Figure 4.10* describe the graph produced on the accuracy and loss on the training.

```
graph = 'handmp'
modality = 'j'

model = dict(
    type='RecognizerGCN',
    backbone=dict(
        type='STGCN',
        in_channels=2,
        gcn_adaptive='init',
        gcn_with_res=True,
        tcn_type='mstcn',
        num_stages=6,
        down_stages=[6],
        inflate_stages=[6],
        graph_cfg=dict(layout=graph, mode='spatial')),
    cls_head=dict(type='GCNHead', num_classes=40, in_channels=128))

test_pipeline = [
    dict(type='PreNormalize2D', threshold=0, mode='auto'),
    dict(type='GenSkeFeat', dataset=graph, feats=[modality]),
    dict(type='UniformSample', clip_len=10, num_clips=1),
    dict(type='PoseDecode'),
    dict(type='FormatGCNInput', num_person=1),
    dict(type='Collect', keys=['keypoint', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['keypoint'])
]
```

Figure 4.8: ST-GCN model, where it utilizes some of the parts from libraries that are supported by PYSKL

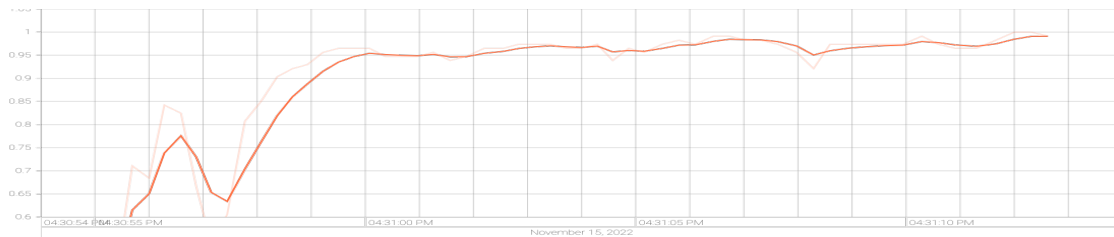


Figure 4.9: Training statistics on accuracy

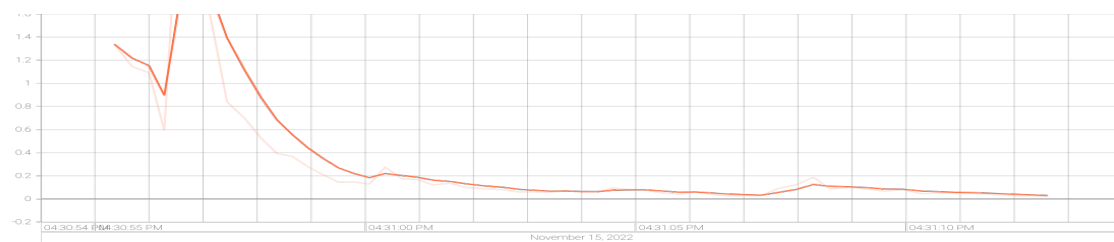


Figure 4.10: Training statistics on loss

As seen on the figures above, we can see that there are a significant drop in the accuracy in around 20 epochs. This drop is however, without any reason whatsoever since from there it will increase with stability and hitting constant accuracy above 95 percent.

```

ytrue
[1, 3, 0, 1, 0, 3]

yhat
[1, 3, 0, 1, 0, 3]

multilabel_confusion_matrix(ytrue,yhat)
array([[4, 0],
       [0, 2]],

       [[4, 0],
        [0, 2]],

       [[4, 0],
        [0, 2]]], dtype=int64)

accuracy_score(ytrue,yhat)
1.0

```

Figure 4.11: Confusion Matrix on the testing data

After the model is built, the thing that is needed to test is the accuracy. The way to test this accuracy is through a method called confusion matrix, to test whether the result of the model and expected value are the same. As we can see on *Figure 4.9*, our model is able to hit accuracy score of 1.0. This prove that the model that was built had a good performance in detecting the gestures correctly.

However, in the real application, the LSTM model can be inaccurate in detecting the motions. In case of there's no movement detected, sometimes the model will detect that the user is doing a rotating motion.

**PYSKL (ST-GCN)** PYSKL provides the checkpoint of the HaGRID, where the keypoints are extracted by MediaPipe Hands, then it will just predict the movement based through the ST-GCN++ model they have. It is able to accurately depict each motions and gestures that are made to the camera, and we found that it is working better compared to LSTM for hand movement. However, as we mentioned, this model is only limited to one hand only, whereas the LSTM model is able to depict the whole body movement.

### 4.3.2 Speed evaluation

Now that the part of stage 2 is ready, then the speed or the performance of the integration is able to be tested. Initially, it was pretty hard to determine accurately on the speed, since the speed varies a lot on whether a certain part is detected or not (it varies if only keypoints are detected, or when a motion that form the gesture trained in the model is detected). Most of the time a motion is registered in the camera, the system will print out the latest gesture that the user made. So in *Figure 4.12*, it can be seen that the latest motion that is made by the user is Swipe Right (SwipeR), and the FPS shown on user's CPU (AMD Ryzen 5 PRO 3500U) is showing 4 FPS. Despite its low number,

the interaction with camera shows that it is smoother than 4 FPS that was achieved in comparison done in *Table 4.3*. Trial on another CPU (Intel Core i5 7300HQ) shows that the performance of the model improves a lot, reaching 12 FPS.

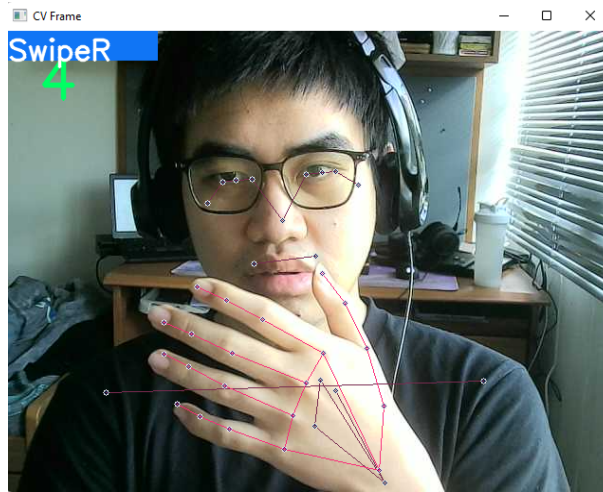


Figure 4.12: Stage 2 evaluation on speed (FPS is shown on top left corner)

With the integration from the both model in Flask, LSTM suffers from performance issue both in FPS, notably 50% cut in FPS, from the OpenCV evaluated (9 FPS) to less than 6 FPS. Although the model doesn't suffer from accuracy cut, but this performance cut add more delay in the website. However, as PYSKL doesn't need to be converted to different format to run, it is able to run in Flask without performance issue, able to achieve 14 FPS.

Accuracy wise, since the dataset that PYSKL is trained with, HaGRID, it is able to detect hand motion more accurately compared to the LSTM. The contributing factor to this is that the size of dataset it is trained with. The previous LSTM model, since we made the dataset ourselves, it is limited in the training data, where each motion only have the size of 4 MB. If we see the HaGRID dataset [45], we can see that each motion will have around 40 GB of

HD Video as the whole dataset. This difference in dataset size also shows that the more data that is used, the more accurate the model will be.

### 4.3.3 Website performance

The website is able to perform well with satisfying performance, where for each button press, it is able to respond to it under 0.2 seconds. For the login part, it will take 15 frame from the webcam, where each frame returns the username of the detected face. Based on the performance of the OpenCV on this part, it means that this part will take around 3 seconds before it is able to pass the aggregated username. Motion detection will take only 10 frames of majority action, which means it will take less than 1 seconds to record every motions (assuming there is motion read through camera).

# Chapter 5

## Conclusion

### 5.1 Summary

In this semester, we have done improvements on our face recognition and pose estimation algorithms which resulted in a better system, combine Stage 1 and Stage 2 of GestHome into a smooth system with the help of Flask as our back-end framework and other programs such as MongoDB, and added features that is necessary for GestHome. With utilizing one-shot learning for our face recognition, we have managed to improve the accuracy without sacrificing any performance, which shows how effective one-shot learning is and the suitability of one-shot learning for our case. Furthermore, swapping LSTM to PYSKL for our action recognition program prove to be the right choice as it gives more accuracy and performance increase.

With the completion of website and API listener, it means that all the functionalities that is expected to work in a Smart Home system has been completed, where it is able to execute the commands that is given by the user through camera, in which the API listener will reply to the user through text-to-speech.

## 5.2 Problem Encountered

Our research in developing a smart home system using face recognition and pose estimation has their own sets of problem. For face recognition, our biggest hurdle is in finding the right balance between accuracy and performance. Since our system will be applied to low power hardware, we have to find the right algorithm that performs decently accurate in recognizing faces but doesn't take too much power to run. This is a challenging task in face recognition since popular face recognition models utilize layers of Convolutional Neural Network (CNN) with various degree of complexity, which negatively impact the performance of GestHome. This is why our team has spent a good amount of time researching the most suitable face recognition model, since a suitable model matters a lot in our case. The same situation also happens in action recognition part of GestHome.

Another struggle that we found during our research is the necessary hardware to do inference testing on the models for face recognition and action recognition. With models of both subject consists of various amounts of neural networks, many times the model we have chosen needed to be run on powerful hardware, such as multiple GPUs. The hardware requirements on these models meant that we couldn't utilize them on our research even though some models looks promising in terms of accuracy, with many reaching state-of-the-art accuracy. This issue is amplified with the necessity of GestHome to be able to run in a low power hardware.

The action recognition module that we are using in the end product, which is PYSKL, has a limitation on which OS it is using. There are a certain dependencies on Linux, where it disables user to use it on different OS. This limits the host server to only available on Linux.



The website implementation on Flask takes time, because there are unforeseen errors that are not usual, and there is not much help available, leading us having to resort to the blindly experimenting through the official documentation or trying to debug ourselves. However, though it is difficult, we managed to put all the stages and features that we have into GestHome website.

Lastly, utilizing Flask as the framework of GestHome has negatively impact the performance of GestHome. In face recognition, we notice a drop in frames by up to 40 percent, from average of 9 FPS if we run the model outside of Flask vs 5 FPS if we run the model with Flask. The same problem is also encountered with the action recognition model. This creates a burden in our research since that means we have to be even more wary about the performance of our model.

## Chapter 6

# Distribution of Work

In this project, our team have contributed equally throughout the entire year. We have collaborated on creating the ideas for this project, doing research on similar projects, writing the codes , and . With the components of the project being very similar with each other, we decided that collaborating will be much better, rather than separating the projects into different components. Furthermore, with both of us having various knowledge on AI and Computer Vision, we manage to bring new perspective and ideas that might not be imagined by only one person. We have also regularly updated our professor and supervisor with the progress of our project and communicate to them regarding the difficulties we encounter when building our project. This way, our professor and supervisor know the progress of our project and they are able to help us with any issue arising and give helpful suggestions to our project. The details of the distribution of labour will be written in the following table:

Details of Work	Responsible Member
Designing the system architecture	Christopher Albert Priatko and Theodore Fabian Rudy
Research on various pose estimation system and face recognition	Christopher Albert Priatko and Theodore Fabian Rudy
Building the face recognition system	Christopher Albert Priatko
Building the pose estimation system	Theodore Fabian Rudy
Building the back-end infrastructure	Theodore Fabian Rudy
Designing the front-end	Christopher Albert Priatko and Theodore Fabian Rudy

# Bibliography

- [1] A. Amini, A. S. Periyasamy, and S. Behnke. T6d-direct: Transformers for multi-object 6d pose direct regression. *CoRR*, abs/2109.10948, 2021.
- [2] A. Amini, A. S. Periyasamy, and S. Behnke. Yolopose: Transformer-based multi-object 6d pose estimation using keypoint regression, 2022.
- [3] R. Bajpai and D. Joshi. Movenet: A deep neural network for joint profile prediction across variable walking speeds and slopes. *IEEE Transactions on Instrumentation and Measurement*, 70:1–11, 2021.
- [4] T. Baltrušaitis, P. Robinson, and L.-P. Morency. Constrained local neural fields for robust facial landmark detection in the wild. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 354–361, 2013.
- [5] T. Baltrušaitis, P. Robinson, and L.-P. Morency. Openface: an open source facial behavior analysis toolkit. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–10. IEEE, 2016.
- [6] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann. Blazepose: On-device real-time body pose tracking, 2020.

- [7] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus, 2019.
- [8] G. S. Behera. Face recognition using siamese network. <https://medium.com/wicds/face-recognition-using-siamese-networks-84d6f2e54ea4>, 2021.
- [9] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.
- [10] S. Benhur. A friendly introduction to siamese networks. <https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942>, 2020.
- [11] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- [12] A. F. Bobick and J. W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on pattern analysis and machine intelligence*, 23(3):257–267, 2001.
- [13] A. Bowes, A. Dawson, and D. Bell. Ethical implications of lifestyle monitoring data in ageing research. *Information, Communication & Society*, 15(1):5–22, 2012.
- [14] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature

- verification using a " siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.
- [15] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields, 2018.
- [16] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.
- [17] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.
- [18] Y. Chao, S. Vijayanarasimhan, B. Seybold, D. A. Ross, J. Deng, and R. Sukthankar. Rethinking the faster R-CNN architecture for temporal action localization. *CoRR*, abs/1804.07667, 2018.
- [19] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.
- [20] Y. Chen, Z. Zhang, C. Yuan, B. Li, Y. Deng, and W. Hu. Channel-wise topology refinement graph convolution for skeleton-based action recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13359–13368, 2021.
- [21] B. Cheng, B. Xiao, J. Wang, H. Shi, T. S. Huang, and L. Zhang. High-

- erhrnet: Scale-aware representation learning for bottom-up human pose estimation, 2019.
- [22] M. Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.
- [23] H. Duan, J. Wang, K. Chen, and D. Lin. Pyskl: Towards good practices for skeleton action recognition. *arXiv preprint arXiv:2205.09443*, 2022.
- [24] A. El-Nouby and G. W. Taylor. Real-time end-to-end action detection with two-stream networks. *CoRR*, abs/1802.08362, 2018.
- [25] H.-S. Fang, J. Li, H. Tang, C. Xu, H. Zhu, Y. Xiu, Y.-L. Li, and C. Lu. Alphapose: Whole-body regional multi-person pose estimation and tracking in real-time, 2022.
- [26] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu. Rmpe: Regional multi-person pose estimation, 2016.
- [27] L. Fe-Fei et al. A bayesian approach to unsupervised one-shot learning of object categories. In *proceedings ninth IEEE international conference on computer vision*, pages 1134–1141. IEEE, 2003.
- [28] Z. Geng, K. Sun, B. Xiao, Z. Zhang, and J. Wang. Bottom-up human pose estimation via disentangled keypoint regression, 2021.
- [29] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [30] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings*

- of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [31] T. Grel. Region of interest pooling explained. <https://deepsense.ai/region-of-interest-pooling-explained/>, 2017.
- [32] R. A. Güler, N. Neverova, and I. Kokkinos. Densepose: Dense human pose estimation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7297–7306, 2018.
- [33] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021.
- [34] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn, 2017.
- [35] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Computer Vision – ECCV 2014*, pages 346–361. Springer International Publishing, 2014.
- [36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.
- [37] S. Herath, M. Harandi, and F. Porikli. Going deeper into action recognition: A survey. *Image and vision computing*, 60:4–21, 2017.
- [38] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [39] R. Hou, C. Chen, and M. Shah. Tube convolutional neural network (T-CNN) for action detection in videos. *CoRR*, abs/1703.10664, 2017.



- [40] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [41] A. Jain. One shot face-recognition using siamese network, 2019.
- [42] Y. Jiabin, W. Fang, and Y. Jieru. A review of action recognition based on convolutional neural network. *Journal of Physics: Conference Series*, 1827(1):012138, mar 2021.
- [43] P. Jin, V. Rathod, and X. Zhu. Pooling pyramid network for object detection. *arXiv preprint arXiv:1807.03284*, 2018.
- [44] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, and W. Ouyang. T-CNN: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2896–2907, oct 2018.
- [45] A. Kapitanov, A. Makhlyarchuk, and K. Kvanchiani. Hagrid - hand gesture recognition image dataset, 2022.
- [46] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.
- [47] D. E. King. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758, 2009.
- [48] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

- [49] G. Koch et al. Siamese neural networks for one-shot image recognition. 2015.
- [50] O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, pages 1–8. IEEE, 2019.
- [51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
- [52] B. Lake, C.-y. Lee, J. Glass, and J. Tenenbaum. One-shot learning of generative speech concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36, 2014.
- [53] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011.
- [54] J. Lee, M. Lee, D. Lee, and S. Lee. Hierarchically decomposed graph convolutional networks for skeleton-based action recognition. *arXiv preprint arXiv:2208.10741*, 2022.
- [55] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.
- [56] I. Logunova. A friendly introduction to siamese networks. <https://serokell.io/blog/nn-and-one-shot-learning>, 2022.

- [57] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. Smpl: a skinned multi-person linear model. *ACM Trans. Graph.*, 34:248:1–248:16, 2015.
- [58] A. Maas and C. Kemp. One-shot learning with bayesian networks. 2009.
- [59] D. Maji, S. Nagori, M. Mathew, and D. Poddar. Yolo-pose: Enhancing yolo for multi person pose estimation using object keypoint similarity loss, 2022.
- [60] D. Marikyan, S. Papagiannidis, and E. Alamanos. A systematic review of the smart home literature: A user perspective. *Technological Forecasting and Social Change*, 138:139–154, 2019.
- [61] M. Monfort, A. Andonian, B. Zhou, K. Ramakrishnan, S. A. Bargal, T. Yan, L. Brown, Q. Fan, D. Gutfrund, C. Vondrick, et al. Moments in time dataset: one million videos for event understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–8, 2019.
- [62] S. Mroz, N. Baddour, C. McGuirk, P. Juneau, A. Tu, K. Cheung, and E. Lemaire. Comparing the quality of human pose estimation with blazepose or openpose. In *2021 4th International Conference on Bio-Engineering for Smart Technologies (BioSMART)*, pages 1–4, 2021.
- [63] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation, 2016.
- [64] OpenAI. Openai five. <https://blog.openai.com/openai-five/>, 2018.
- [65] D. Osokin. Real-time 2d multi-person pose estimation on cpu: Lightweight openpose. *arXiv preprint arXiv:1811.12004*, 2018.

- [66] D. Qi, W. Tan, Q. Yao, and J. Liu. Yolo5face: why reinventing a face detector. *arXiv preprint arXiv:2105.12931*, 2021.
- [67] R. Rakhimov, E. Bogomolov, A. Notchenko, F. Mao, A. Artemov, D. Zorin, and E. Burnaev. Making densepose fast and light, 2020.
- [68] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [69] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.
- [70] N. Renotte. Face recognition, 2021.
- [71] A. Rosebrock. Face detection with dlib (hog and cnn). <https://pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/>, 2021.
- [72] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.
- [73] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [74] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

- [75] S. Sepasgozar, R. Karimi, L. Farahzadi, F. Moezzi, S. Shirowzhan, S. M. Ebrahimzadeh, F. Hui, and L. Aye. A systematic content review of artificial intelligence and the internet of things applications in smart home. *Applied Sciences*, 10(9), 2020.
- [76] S. I. Serengil and A. Ozpinar. Lightface: A hybrid deep face recognition framework. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 23–27. IEEE, 2020.
- [77] M. Shafiq, Z. Gu, O. Cheikhrouhou, W. Alhakami, and H. Hamam. The rise of “internet of things” review and open research issues related to detection and prevention of iot-based security attacks. *Wireless Communications and Mobile Computing*, 2022:12, 08 2022.
- [78] Z. Shou, J. Chan, A. Zareian, K. Miyazawa, and S. Chang. CDC: convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos. *CoRR*, abs/1703.01515, 2017.
- [79] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.
- [80] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.
- [81] L. Stoffl, M. Vidal, and A. Mathis. End-to-end trainable multi-instance pose estimation with transformers. *arXiv preprint arXiv:2103.12115*, 2021.
- [82] G. Sung, K. Sokal, E. Uboweja, V. Bazarevsky, J. Baccash, E. G. Bazavan,

- C.-L. Chang, and M. Grundmann. On-device real-time hand gesture recognition. *arXiv preprint arXiv:2111.00038*, 2021.
- [83] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.
- [84] P. Tehria. One-shot image classification by meta learning. <https://medium.com/nerd-for-tech/one-shot-learning-fe1087533585>, 2021.
- [85] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.
- [86] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.
- [87] J. Uijlings, K. Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013.
- [88] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.
- [89] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

- [90] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>, 2019.
- [91] J. Walsh, N. O’ Mahony, S. Campbell, A. Carvalho, L. Krpalkova, G. Velasco-Hernandez, S. Harapanahalli, and D. Riordan. Deep learning vs. traditional computer vision. 04 2019.
- [92] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao. Deep high-resolution representation learning for visual recognition, 2019.
- [93] L. Wang, B. Huang, Z. Zhao, Z. Tong, Y. He, Y. Wang, Y. Wang, and Y. Qiao. Videomae v2: Scaling video masked autoencoders with dual masking, 2023.
- [94] M. Wang and W. Deng. Deep face recognition: A survey. *Neurocomputing*, 429:215–244, 2021.
- [95] Y. Wang, M. Li, H. Cai, W.-M. Chen, and S. Han. Lite pose: Efficient architecture design for 2d human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13126–13136, 2022.

- [96] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking, 2018.
- [97] S. Yan, Y. Xiong, and D. Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. *CoRR*, abs/1801.07455, 2018.
- [98] L. Yang, L. Lou, X. Song, J. Chen, and X. Zhou. An improved object detection of image based on multi-task learning. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*, pages 453–457, 2022.
- [99] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 325–341, 2018.
- [100] C. Z. Yue and S. Ping. Voice activated smart home design and implementation. In *2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST)*, pages 489–492, 2017.
- [101] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang. Real-time action recognition with enhanced motion vector cnns. *CoRR*, abs/1604.07669, 2016.
- [102] C. Zhang, W. Liu, H. Ma, and H. Fu. Siamese neural network based gait recognition for human identification. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2832–2836. IEEE, 2016.



- [103] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.
- [104] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.
- [105] W. Zhu, X. Ma, Z. Liu, L. Liu, W. Wu, and Y. Wang. Learning human motion representations: A unified perspective. *arXiv preprint arXiv:2210.06551*, 2022.
- [106] Y. Zhu, Z. Lan, S. D. Newsam, and A. G. Hauptmann. Hidden two-stream convolutional networks for action recognition. *CoRR*, abs/1704.00389, 2017.