Gesture recognition using machine learning

Final report

Matthew Fen Yi Yong
765353, yongf@student.unimelb.edu.au

Yong Yick Wong
796429, yongw4@student.unimelb.edu.au

Tsz Kiu Pang
965374, tszkiup@student.unimelb.edu.au

Supervisor: Professor Jonathan H. Manton

Executive Summary: *Human gesture recognition requires the need for accurate representations of human subjects involving spatial-temporal features. Human pose estimation enables localization of human body joints which could be used to capture human motion. The goal of this project is to explore the use of human pose estimation to capture sufficiently accurate features of humans for machine learning models in gesture recognition. We developed a Proof of Concept (PoC) system that is able to recognise four (4) dynamic gestures. Our overall approach incorporates four main stages: (1) extraction and collection of human key-points from videos using OpenPose, a real time human key-point detection software, (2) generation of synthetic data through data augmentation techniques for model training, (3) development and optimisation of a Long Short Term Memory (LSTM) model for time series classification of human key-points and (4) deployment of our model as a desktop and web application. Our system achieved an average accuracy of 72.37% on a test data set and an average recognition delay of 0.7 seconds.*

Contents

1 Introduction

As humans, we influence the world through our bodies. We express our emotions and actions through body posture and orientation. For computers to be partners with humans, they have to perceive and understand our behavior - recognising our expressions, gestures and movements.

Human Pose Estimation (HPE) is a field of research in developing robust algorithms and expressive representations that can capture human pose and motion. Using Human Pose Estimation, a computer is able to localize human body joints, as coordinates of a person's nose or eyes. There has been significant advancements in the growth human pose estimation, now achieving the possibility of running in real-time[1] such as OpenPose (Cao et al., 2019), AlphaPose (Fang et al., 2016) and DensePose (Güler et al., 2018). This has led to an extension of it's application to many fields such as medical technology (K. Chen et al., 2018) and sports (Huang et al., 2019).

However, human gestures are cognitively complex Cartmill et al. (2012), making acquisition of kinematic data for gesture recognition a challenging task. Motivated by the successes of Book and Fisher (2018) and Ko et al. (2019), our goal is to explore the use of 2D Human Pose Estimation output as the **only** extracted feature that will be fed into our supervised learning framework to recognise gestures.

In this project, our group has been working towards developing a basic Proof-of-Concept (PoC) dynamic gesture recognition system based on a 2-dimensional (2D) Human Pose Estimation (HPE) using a single video camera.

Moreover, there is demand for running machine learning applications on low-powered devices such as mobile phones with stringent constraints on the latency, memory and computational cost (Kenneth Research, 2020), particularly for video processing tasks Bhardwaj et al. (2019). Exploring the possibility of running system on such devices, we investigated techniques to achieve decent gesture recognition performance with minimal but sufficient resources, particularly using a reduction of video frames.

This motivates a sub-goal of the project, that is to investigate the possibility of using fewer frames to recognise gestures.

[1] on specified hardware

2 Literature Review

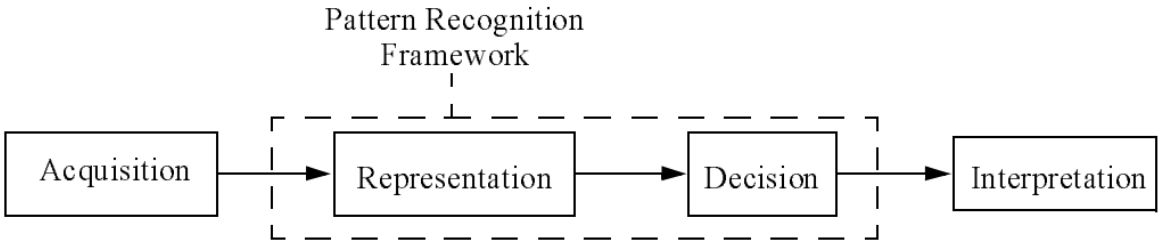


Figure 1: General flow of a human gesture recognition system (Allevard et al., 2005).

According to Allevard et al. (2005), a general framework for human gesture recognition systems start with (1) the *acquisition process* of human kinematic data in numerical form, (2) *representation and decision processes*, converting numerical data into meaningful representations and making decisions (3) *interpretation process* that gives meaningful symbols from outputs of the decision making process.

Data acquisition for human gesture recognition is a challenging task due to it's reliance on spatial-temporal information (G.Bosworth et al., 2019). There exists two approaches: (1) device-based and (2) vision-based (Mkom et al., 2013). Using device-based approaches, acquisition of human data in kinematic form is collected from strapped on devices such as the *CyberGlove* (Oz and Leu, 2005). These devices measure kinematic changes of the human subject such as changes in joint angles (Dong et al., 2015) that represents kinematic information of the human subject when gesturing. However, a major drawback is it's need for appropriate attachment to the subject's body in which changes in placement, development costs of devices and degradation of electronic quality causes potential problems in accurate data acquisition (Ahmed et al., 2018).

An alternative, is to use a vision-based approach, utilising cameras and image processing techniques in capturing human gestures. Due to it's unconstrained nature of only needing a camera, it has a major advantage of being able to capture a spatial-temporal representation through acquisition of image frames without the reliance on external hardware devices. A major feature is the use of Convolutional Neural Networks (CNN), that have shown success in extracting features from image data for gesture classification (Lai and Yanushkevich (2020), V. and R. (2020)).

However, studies have shown that detailed image data such as hand shape are not required in interpreting human gestures (Bragg et al. (2019), Poizner et al. (1981)). In light of this, there have been successes in using *human pose estimation* as main extracted features for human

gesture recognition. This reduces the dimensionality of raw pixel data while maintaing an accurate representation of the human subject (Malhotra et al., 2019). Moryossef et al. (2020) achieved accuracy of at least 87%-97% and Ko et al. (2019) achieved accuracy of 93.28% using human pose estimation software for gesture recognition.

There exists various machine learning models used in performing gesture recognition. X. Wang et al. (2012) proposed the use of *hidden markov models* in recognising dynamic hand gestures. Dong et al. (2015) proposed the use of ensemble methods such as *random forest*, achieving 90% accuracy. Rwigema et al. (2019) proposed dynamic time warping techniques based on sensor data. One of the most popular model used is the Long Short Term Memory Network (LSTM), due to it's effectiveness in dealing with sequential data using a recursive mechanism (Hochreiter and Schmidhuber (1997), Lipton et al. (2015)).

Bragg et al. (2019) highlights the lack of human gesture recognition data available online that sufficiently represents human gestures. Ko et al. (2019) and Molchanov et al. (2015) recommended the approach of data augmentation in order to diversify data-sets for human gesture recognition. Kim and O'Neill-Brown (2019) has shown that the effect of synthetic data augmentation improves the performance of American Sign Language Recognition (ASLR). Ko et al. (2019) noted the use of *sub-frame sampling*, where frames are sampled (in order) randomly for training, improving model generalisation performance.

Our approach utilises human pose estimation, synthetic data augmentation techniques and an LSTM model (variant of RNNs) to perform successful gesture recognition. This has the combined advantages of: (1) avoiding use of external hardware devices, (2) avoiding use of raw image data, using numerical outputs from human pose estimation, (3) achieving decent performance with limited data-set size using data augmentation and (4) the use of LSTMs, which has shown effectiveness in performing sequential classification tasks.

3 Project Overview

3.0.1 Specifications

This project is to build a PoC gesture recognition system[2] capable of recognizing a set of four (4) gestures. With reference to Figure: 2, the selected gestures are based on the signs[3]: $\{$*Ambulance*, *Help*, *Hospital*, *Pain*$\}$ from the Australian Sign Language (Auslan)[4].

[2]Hereof, this term "system" should be contextually clear in refering to the team's gesture recognition system throughout this document.

[3]The sign is non-static. Please refer to the website: http://www.auslan.org.au/ for its actual video.

[4]http://www.auslan.org.au/

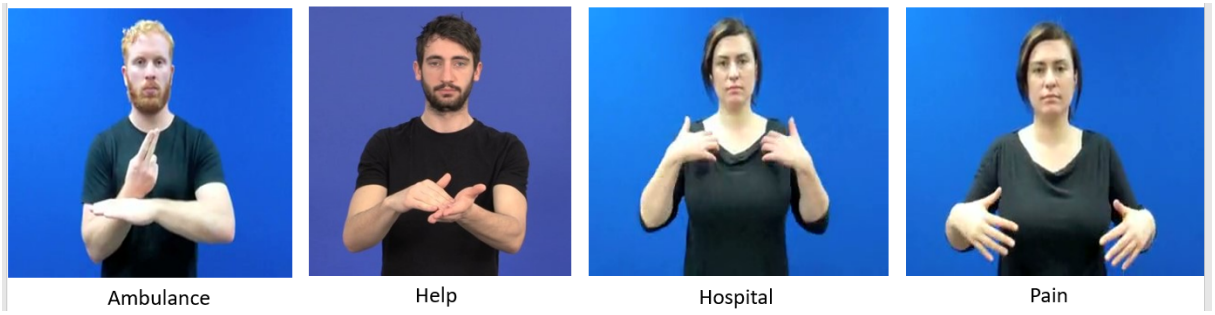


Figure 2: The set of Auslan Signs that will be used as example human gestures.
Note. The images correspond to Auslan signs. Retrieved from auslan signbank (http://www.auslan.org.au/), licensed under CC BY-NC-ND 4.0.

The selected Auslan signs have been reappropriated in this project to demonstrate our gesture recognition system only for research purpose. With reference to Figure: 3, the set of selected gestures is as follows: {Gesture A, Gesture B, Gesture C, Gesture D}. For convenience, we shall interchangeably use "gesture" with "sign", and the user of the system who is performing the sign could also be referred as the signer.

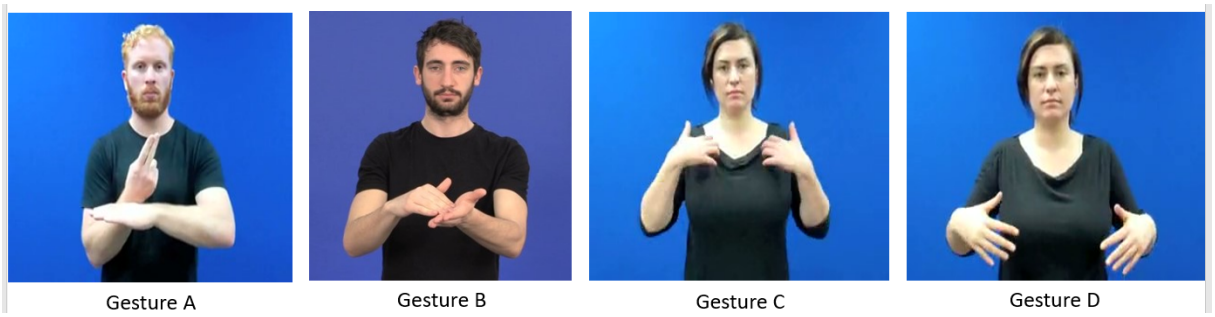


Figure 3: The set of gesture to be recognized.
Note. The images correspond to Auslan signs and have been reappropriated as gestures. Retrieved from auslan signbank (http://www.auslan.org.au/), licensed under CC BY-NC-ND 4.0.

This recognition is a dynamic word-level (or isolated) gesture recognition, where "dynamic" means the gesture of interest has non-static movements, and "isolated" means it is not a sentence-level.

This system is to achieve an average of 80% test accuracy evaluated on a dataset that is not used to train the system. This is based on the following similar work: Hosain et al. (2020), D. Li et al. (2020) and Book and Fisher (2018), where a 93% average accuracy is achieved on GMU-ASL51 benchmark[5], a 89.92% average accuracy is achieved on WLASL-100 bench-

[5]GMU-ASL51 collected by Hosain et al. (2019) has 13107 samples of 51 word level classes from 12 distinct subjects of different height, build and signing.

mark[6] a 93% average accuracy is achieved on self-created dataset, respectively. This suggests that an average accuracy above 80% is achievable. The specified accuracy of our system is chosen to reflect the fact that there is no pre-existing dataset of videos of the selected gestures of comparable dataset size to build the system.

3.0.2 Assumptions

Development of the system is based on the following assumptions:

1 There is a one-to-one correspondence between the gesture and the hand-and-body movement. This is because the selected gestures are based on Auslan with Australia-wide dialect. In this context, the Auslan signs are unique, thereby we impose the same on our set of gestures. This assumption reflects that recognizing a physical action is context-specific in general. For example, a hand-waving movement constitutes nothing other than a physical movement until it is put into context.

2 The maximum time taken for a signer to complete a single gesture is 2.5 seconds. This is based on the selected gestures in Specifications.

3 Only a single signer of the gesture is allowed and recorded by a single recording device[7] at a given time. This is by construction.

4 The gesture being performed must be visucentric[8]. This includes but not limited to the following[9]:

   a The signer who is performing the gesture must be recorded frontal and from waist-up in a fixed upright position at an eye-level with the recording device.

   b The gesture being performed must be recorded completely and naturally, that is, no cut-offs[10] and no obstructions[11].

   c The visibility of the signer and the gesture should be sufficiently high, and shall not be interfered throughout during the recording. This is judged based on the ability

[6] Word-Level American Sign Language (ASL) video dataset containing 100 words performed by over 100 signers.

[7] Hereof, all the recording devices used to record the gesture will be collectively referred as the ”camera”.

[8] Definition (Wikitionary): Designed in a manner that is visually optimal for deaf and hard-of-hearing people; pertaining to designs that focus on visual access

[9] These are based on the works of *FILMING THE SIGNERS(S)* (n.d.) and Sprawls (n.d.)

[10] Hands are moving outside the range of the camera.

[11] There exists scenery or people blocking the view of the gesture.

of an observer[12] to detect the object (gesture). Such ability depends on a combination of factors including but not limited to object contrast and size, background, brightness (luminance) and structure (texture), and distance between the image and the observer. For example, close-ups and long-distance recording shots should be avoided.

This Assumption (4) is reasonable as the selected gestures in Specifications involve only the upper-human body.

5 The camera set-up must be fixed throughout during the recording. This includes but not limited to:

   a The camera must be physically stationary at all times when recording the gestures.

   b The camera must maintain one single mode of operation at all time when recording the gestures. For example, zooming-in during the gesture-recording is not allowed.

These Assumptions (5a, 5b) are imposed to minimize the skews, distortions and perspective-changing during the recording which by construction, our system is not designed to handle these.

6 The system must be able to achieve online recognition of gestures, with a maximum delay of 2.5 seconds given minimal hardware requirements.

   a Due to our reliance on human pose estimation software, the following are the minimal requirements needed to run online recognition of gestures:

      * Nvidia CUDA[13] enabled GPU card with at least 1.6 GB available.
      * At least 2.5 GB of free RAM memory for human pose models.
      * Installation of the *cuDNN*[14] library (GPU-accelerated library of primitives for deep neural networks).

In the event of the assumptions imposed above being violated:

1 **Assumptions 1, 2, 3, 5:** The output of the system should not be considered as valid.

2 **Assumption 3, 4:** The output of the system should not be considered as valid, or the system will flag that no human is detected.

3 **Assumption 6:** The system is unable to perform online human gesture recognition due to the inability to run human pose estimation in real time.

[12] Here, it could be a human or a camera.

[13] Parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).

[14] Contains highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers

4 Method

In this section, we will address the methodologies used in creating a POC system to show the effectiveness of using human pose estimation in gesture recognition. The following is an outline of the subsections covered in the methods section:

- Problem Formulation and Set-up - Reformulation of the problem of human gesture recognition.
- Human Pose Estimation - Overview on using OpenPose for human keypoint detection.
- Data Preparation - Collection and processing of video data for model training.
- Model Development - Designing, optimising and evaluating our gesture recognition model.
- System Deployment - Deployment of our model as both a desktop and web application.

4.0.1 Problem Formulation and Set-up

The problem of human gesture recognition is formulated as a time-series multi-class classification problem. Given a set of *m* classes (gestures): $\vartheta = \{G_1, G_2, ..., G_m\}$ and a video, $\mathbf{V}$ consisting of *N* frames: ${\Omega_N}^{15} = \{F_1, F_2, ..., F_N\}$, the main goal of this project is to identify the class to which the video belongs to. Thus, the objective is to learn the conditional probability $p(G_i|\mathbf{V})$ for $G_i \in \vartheta$.

Refer to Equation: 1. This probability, ${p_N(G_i|\mathbf{V})}^{16}$ could be learned using a neural network, f which looks at all the frames in the video, which each of the frame could be processed by a function, g to extract the feature to predict:

$$p_N(G_i|\mathbf{V}) = f\left(g(F_1), g(F_2), ..., g(F_N)\right) \tag{1}$$

An overview of this approach can be seen in Figure 4, where a video stream can be broken down into a series of frames. Then, a human key-point detection software is used to extract the key-point x,y coordinates for each frame. This converts a series of frames into a series of key-point arrays of x,y coordinates. Finally, the list of key-point coordinates are sent to a deep learning model that outputs a classification of the gesture as well as its probability.

[15]The subscript, *N* is the number of frames.
[16]The subscript, *N* is the number of frames.

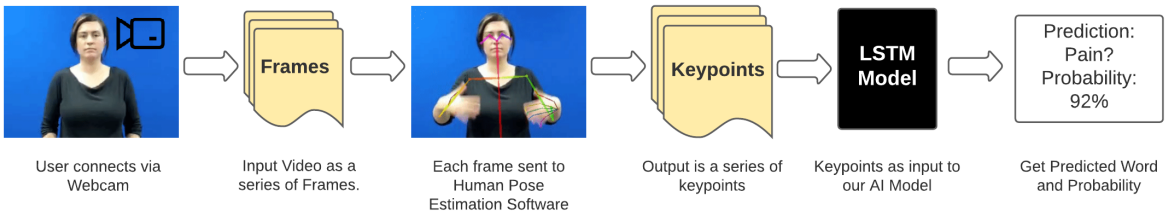


Figure 4: High level system view, using input video stream and outputs a classified gesture. *Note. The human images correspond to an Auslan sign. Retrieved from Auslan Signbank (http://www.auslan.org.au/), licensed under CC BY-NC-ND 4.0.*

The second focus of this project is to use the same neural network, f and the feature-extracting function, g to look at only a fraction of the N frames at inference time to make the prediction. Only every j^{th} frame: $\{F_1, F_j, F_{2j}, ..., F_{\hat{N}}\}$ where $1 \leq j \leq \hat{N}, \hat{N} < N$ of the video, $\mathbf{V}$ are processed. This is reflected in Equation: 2.

$$p_{\hat{N}}(G_i|\mathbf{V}) = f\left(g(F_1), g(F_j), g(F_{2j})..., g(F_{\hat{N}})\right) \tag{2}$$

As a heuristic, refer to Equation: 3, the error, ϵ_F between the probabilities, $p_N(G_i|\mathbf{V}), p_{\hat{N}}(G_i|\mathbf{V})$ is used to investigate the possibility of using fewer frames to recognise gestures.

$$\epsilon_F = |p_{\hat{N}}(G_i|\mathbf{V}) - p_N(G_i|\mathbf{V})| \tag{3}$$

In this project, the gestures of interest take a maximum of 2.5 seconds to complete by Assumption: 3.0.2. This corresponds approximately to 75 frames[17]. This will be the first setting. We choose to reduce it to 35 frames by considering every 2^{nd} frame for the second setting. So, we have two settings: $\Omega_{75} = \{F_1, F_2, ..., F_{74}, F_{75}\}$ and $\Omega_{35} = \{F_1, F_3, ..., F_{33}, F_{35}\}$.

The neural network, f used is called Long Short-Term Memory (LSTM) developed by Hochreiter and Schmidhuber (1997). Please refer to Appendix: 9.1 for the background knowledge on LSTM. The feature-extracting function, g is the Human Pose Estimation (HPE) which will be discussed subsequently.

[17]This assumes a 30 frame-per-second (FPS).

4.0.2 Human Pose Estimation - OpenPose

Our model relies on human key-point extraction as feature extraction of human gestures. We used OpenPose, an open source human key-point detector developed by Cao et al. (2019) - capable of detecting multi-person human key-points given a video feed in real time [18].

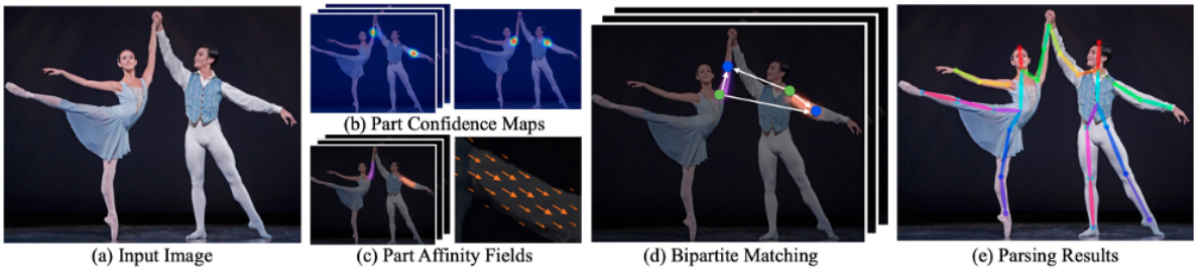


Figure 5: OpenPose - Multi-person key-point detection using part affinity fields.
Note. Adapted from Cao et al. (2019) (Fig.2, p. 2), licensed under CC BY-SA 4.0.

OpenPose features detection on body, face, hand and foot key-points. The hand key-point estimation model is attributed to the work of Simon et al. (2017) - allowing for hand key-point estimation given a 2D image view. To perform mult-person detection, OpenPose uses a bottom up approach, where key-points are first detected and then later joined together to form the overall skeleton output. First given an image, it uses two Convolutional Neural Networks (CNNs) in parallel - one performing human part detection and the other, part-affinity fields used to represent degree of associations between parts. Then, it uses a greedy parsing algorithm to perform bipartite matching between its key-points to generate skeleton pose output.

[18] using hardware that meets compute specifications

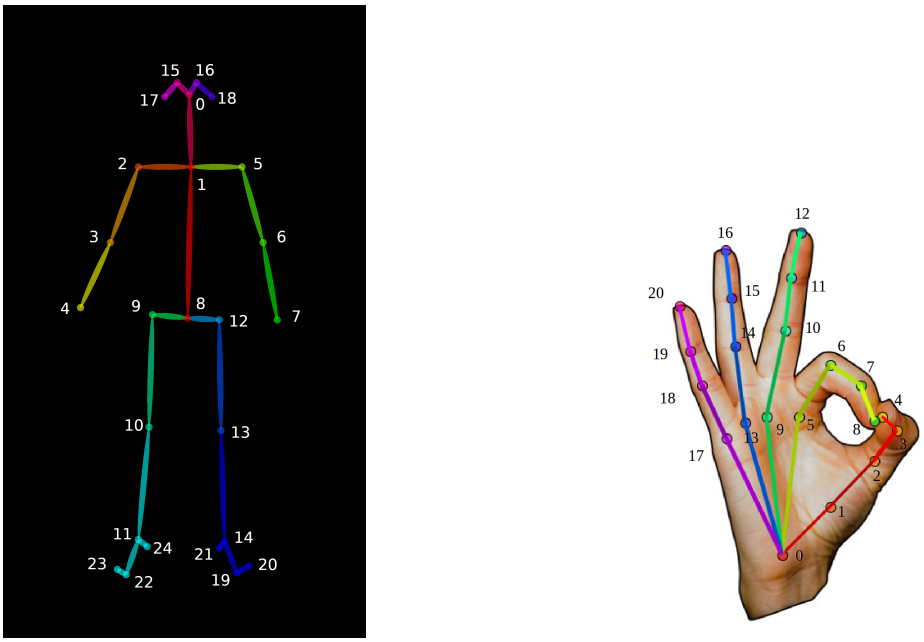


Figure 6: From Cao et al. (2019), OpenPose output mappings. *Retrieved from https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/output.md, licensed under CC BY-SA 4.0.*

For our system, we used a selected subset of key-points portrayed in figure 6. Selection of these features are influenced by noting that parts such as wrists, elbows and shoulders carry significant weight in recognising human gestures (Moryossef et al., 2020), (Mocialov et al., 2017). The following is a list of human key-points we extracted per frame:

- In total, we used 98 x,y key-points [19] (including both hand and human body).
- Of the 98 key-points, 14 were upper body pose key-points (points 1 - 7).
- Remaining 84 being key-points from both the left and right hand.
  - 21 Key-points from right hand.
  - 21 Key-points from left hand.

[19] Key-point here indicates a single x coordinate or y coordinate.

4.0.3 Data Preparation

Our model requires the availability of videos showing humans gesturing the four[20] Auslan inspired gestures for model training. Upon scrapping the web, the amount of videos collected were both insufficient and unsuitable for model training. Moreover, videos were constricted to having elbows in frames when gesturing - a limitation of OpenPose's hand model for hand key-point detection (Simon et al., 2017). Thus, we resorted to recording a data-set of ourselves gesturing these four signs to train our model. We recorded up to 400 raw videos, each gesture having 100 videos each. We set a limit of 3 seconds per video to perform the full sequence of a gesture - ending up with 75 frames per video[21].

After recording, collecting and labeling these videos, we performed data augmentation to increase variability in training data and to synthesize more data. Performing data augmentation on image data reduces over fitting in models. (J. Wang and Perez, 2017), (Kim and O'Neill-Brown, 2019). Moreover, Park and Sohn (2020) showed that performing data augmentation on pose estimated key-points helped increase accuracy in their sign language recognition model. Motivated by these examples, our approach in doing data augmentation comes in three forms: (1) re-processing videos to have variations in gesture speeds (2) using classical image processing techniques on raw frames (3) performing key-point perturbation on processed key-points.

(2) Classical Image Processing: As shown in Figure: 7, flipping, rotation and shearing were chosen to augment the videos. Flipping was to reflect the left-handedness and right-handedness. Rotation operation was randomly parameterized from $(-10°, +10°)$[22] because it was shown that this parameter range improved the accuracy by at least 2% (Kim and O'Neill-Brown (2019)). Shearing was to account for the difference in the spatial variation of the motion of a given gesture signed by different people. For example, the degree of the hand rotation might be different. Shearing operation was randomly parameterized from (-0.3, 0.3)[23] (Hu et al. (2019)).

(3) Key-point Perturbation: Since the keypoints estimated by HPE are inputs to the gesture recognition, this could be augmented in the form of perturbation. Gaussian noise sampled from $N(0, 0.005^2)$ on the keypoints (x,y)-coordinates was introduced (De Coster et al. (2020)).

[20] help, pain, hospital, ambulance

[21] using camera FPS = 30

[22] negative value for anticlockwise rotation, otherwise positive

[23] Shear the image along the horizontal axis with certain rate.

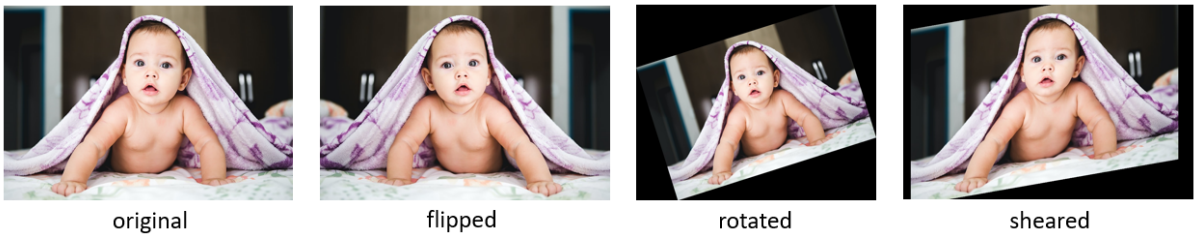


Figure 7: Various Image Transformations

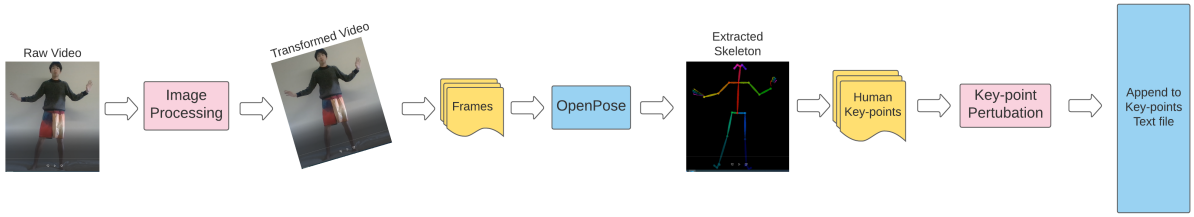


Figure 8: Data processing pipeline setup, converting raw videos and appending to key-point lists at the end.

To do so, we have implemented an automated data processing pipeline (Figure 8). First, raw videos with labels are pushed into the pipeline into an image processing block - transforming the frames as part of the data augmentation. These frames are then sent to OpenPose - which estimates the human key-points and render them as a skeleton. The series of 75 frames are now converted into a series of 75 key-point lists, each list having 98 key-points coordinates[24]. Finally, we apply key-point perturbation by adding each key-point. After processing, the 75 key-point list is appended to a text file and labeled for training. We repeat this over all video data-sets each having different video speed alterations.

Including both raw and augmented videos, we collected up to 30,000 input videos as training data for all classes in the 75-frame setting. See the Problem Formulation in Section: 4.0.1, for the 35-frame setting, we obtained a total of 60,000 inputs using the collection from 75-frame setting by sampling at every 2^{nd} frame of the 75-frame video.

4.0.4 Model Development

In this section, we describe the choice of the model architecture to recognise a gesture, the hyperparameters that we considered, the procedure to model-testing.

[24]Refer to the human pose section for the full list of extracted key-points.

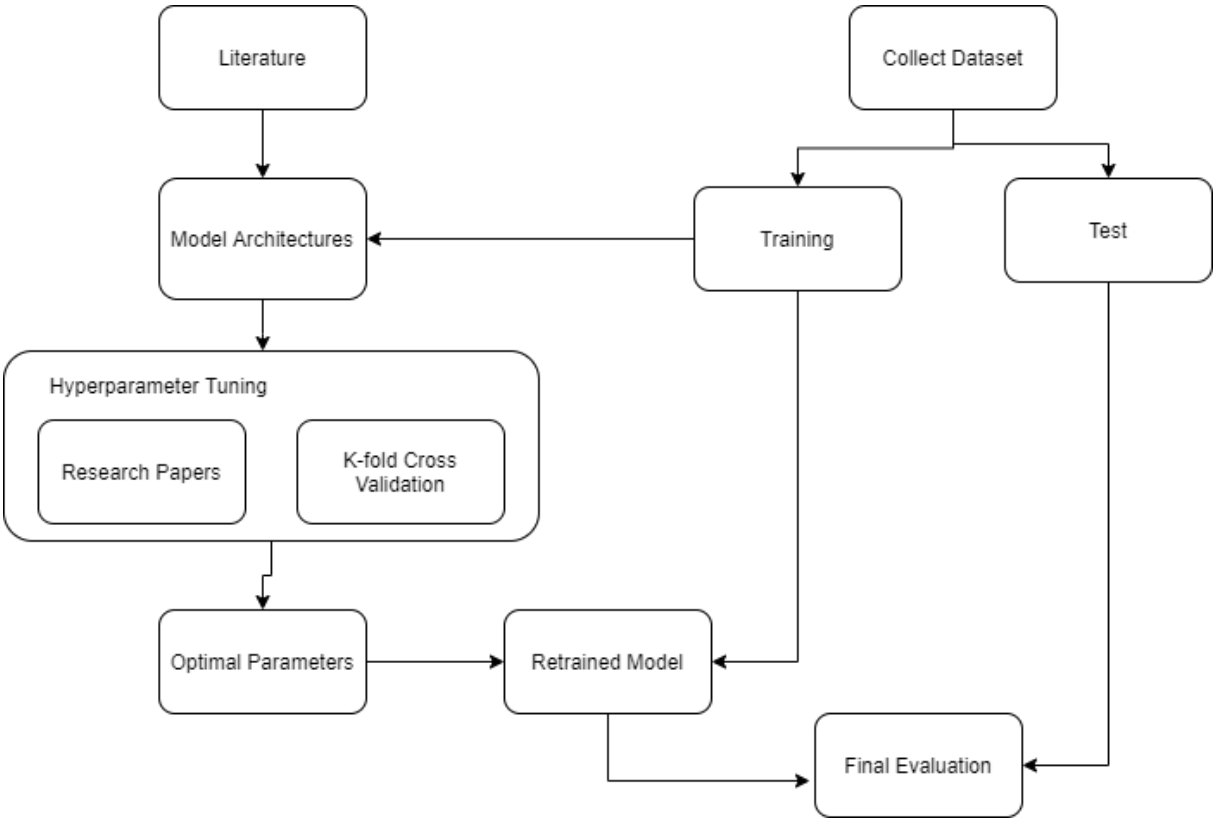


Figure 9: Model development process to a refined model

1. Model Architecture and Hyperparameter Selection
2. Hyperparameter Tuning via K-fold Cross Validation
3. Model Testing

An overview of the model development process can be viewed in Figure 9. We consulted the literature where similar works were done to determine the architecture of the model (and the hyperparameters). Then, we performed K-fold cross validation to tune and find the optimal hyperparameters. Finally, we used these optimal hyperparameters to train the model.

1. Model Architecture and Hyperparameter Selections

Book and Fisher (2018) and Mocialov et al. (2017) achieved accuracies of 93.0% and 99.99% respectively for similar applications. Leveraging on their model architectures, we decided to experiment with a maximum of two layers of LSTM, each layer with the number of hidden units ranging from: $\{32, 64, 128\}$, while fixing the rest of the hyperparameters. In other words,

we treated the number of LSTM layers and the number of hidden units as hyperparameters to be optimized (tuned). The rest of the fixed hyperparameters will subsequently be discussed.

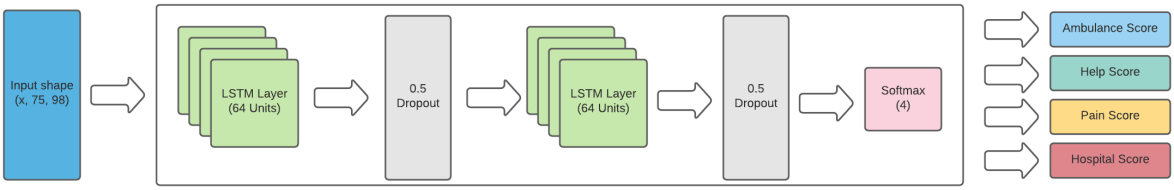


Figure 10: Model network architecture, composed of two layers of LSTMs, Dropouts and a Softmax layer.

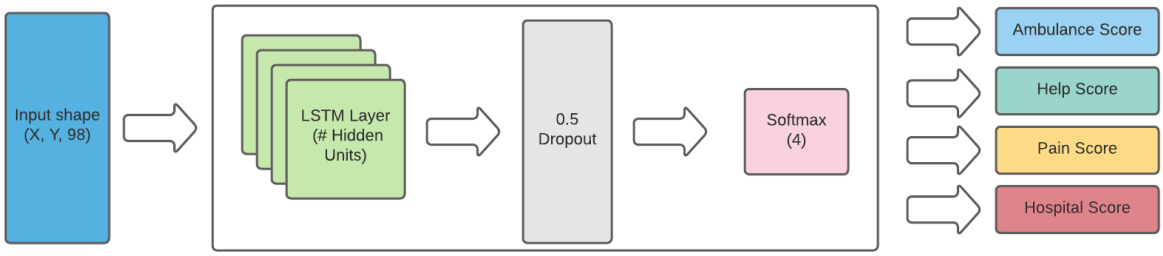


Figure 11: Model network architecture, composed of one layer of LSTM, Dropout and a Softmax layer.

The two architectures are shown in Figures: 10, 11. The model takes in as input a vector of shape $(X, Y, 98)$ where X represents the batch size for training, Y is the number of frames and 98, the number of key-points per key-point list. A Batch size of 32 was chosen (Reimers and Gurevych (2017)).

The layer activation function for the LSTM layer was hyperbolic tangent function. This was because due to resource constraint, we decided to implement using NVIDIA CUDA® Deep Neural Network library (cuDNN), a GPU-accelerated library of primitives for deep neural network. This restricted the choice of the activation function to the hyperbolic tangent function.

We applied a dropout ratio of 50% after each LSTM layer. This choice was suggested by G.E.Hinton et al. (2012) to reduce errors in model. Dropout is a technique that randomly drops output units from a layer during the training process, preventing models from co-adapting too much (Srivastava et al., 2014).

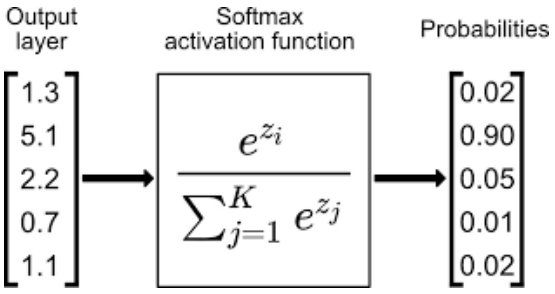


Figure 12: Softmax: Outputs from network are squashed into probabilities (Radecic, 2019)

Softmax ((Radecic, 2019)) was used in the final output layer to label probability scores for each class. Given a list of layer outputs, it squashed all results into a list of estimated probabilities p, where $p \in [0, 1]$ as shown in Figure: 12

For training, the model uses the cross-entropy loss: $J(\theta) = -\left(\sum_{c=1}^{M} y_{o,c} log(p_{o,c})\right)$, where M is the number of gesture classes, y is a binary indicator if gesture label c is the correct classification for observation o and p is the predicted probability observation o of gesture c.

All the neural networks were trained for at least 200 epochs with Early Stopping. The number of epochs was determined empirically. We splitted the training data into a training and validation set so that we could compare the validation loss and training loss. With reference to Figure: 13, the number of epochs where the validation loss starts deviating from the training loss indicates that the model has started to overfit the training data. We denote this epoch number as the critical point. As such, this critical point will be the optimal epoch for training. We did this for each of the model under consideration to obtain these critical points. 200 number of epochs for the training was obtained which was significant higher than all the critical points. This was to ensure the the minimum validation loss will be reached. The training was stopped by Early Stopping when the deviation of the validation loss and the training loss was significant.

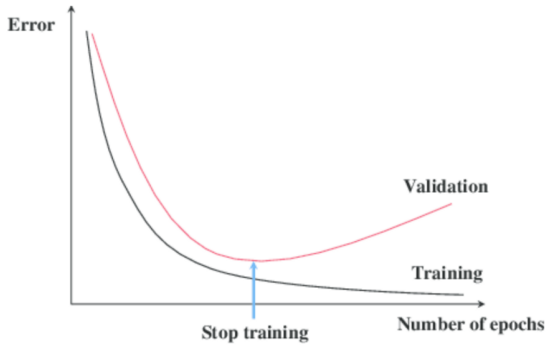


Figure 13: A made-up example that shows the general change in validation and training losses with the number of epochs.

The Adam optimisation algorithm was used with learning rate = 0.003 and decay = 0.00005 with the learning rate reduced by a factor 10 whenever the validation accuracy did not improve by at least 0.01 for 10 epochs (De Coster et al. (2020)). The Adam optimiser is known to be computationally efficient and uses little memory when training models (Kingma and Ba, 2015).

2. Hyperparameter Tuning via K-fold Cross Validation

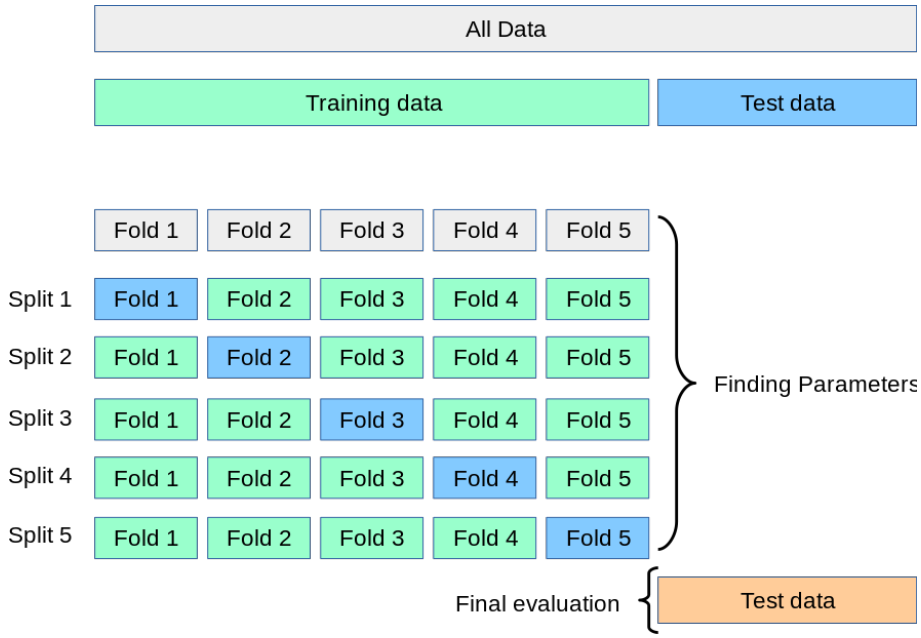


Figure 14: Illustration of k-fold cross validation, splitting data-set into k-folds and performing evaluation for all k configurations (Pedregosa et al., 2011).

The number of LSTM layers and the number of hidden units were to be optimized (tuned). The maximum number of LSTM layers was two, the number of hidden units considered were: $\{32, 64, 128\}$. Note that for two layers, the number of hidden units was the same for both layers. So, a total of six (6) models was considered. This is summarised in Table: 1

Table 1: Set of model configurations used for both 35-frame and 75-frame settings.

Model (35 or 75)	**1**	**2**	**3**	**4**	**5**	**6**
No. of LSTM layers*	1	1	1	2	2	2
No. of Hidden Units	32	64	128	32	64	128

* The layers are with hyperbolic tangent activation function.

We performed K-fold cross validation over for these six (6) models in Table: 1 to determine the optimal hyperparameters, hence the optimal model under consideration. K-fold cross validation

was employed to avoid conducting the learning of the hyperparameters on the same training data. This was because the model that would just repeat the labels of the samples that it had just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. So, K-fold cross validation is procedure used in obtaining the refined model's "average" performance given different configurations of training and test data used.

With reference to Figure: 14, K-fold cross validation first splits the entire data-set into k mutually exclusive subsets $D_1, D_2, ..D_k$ of approximately equal size. Then, for each split, we train the model on all chosen k-1 segments and test it on the k'th segment. The model's performance (validation accuracy) is stored and we repeat the process again over all k configurations. We chose k = 10, as it sits within the range of commonly suggested k values (Bergstra et al., 1995).

The average validation accuracies obtained via Cross Validation for all the models were compared. The highest accuracy achieved will be the optimal model in terms of accuracy under consideration.

3. Model Testing

Once the final model was chosen and trained, we evaluated it on a test dataset that was not seen during the training. To build this test dataset, we invited people outside of the team to record themselves by without any further instructions from the team.

However, the number of the collected test videos was insignificant. We employed the same strategy as in Data Preparation in Section: 4.0.3. We chose a different set of image transformations consisting Y-rotation, Z-rotation [25] and Zoom-out, as shown in Figure: 15

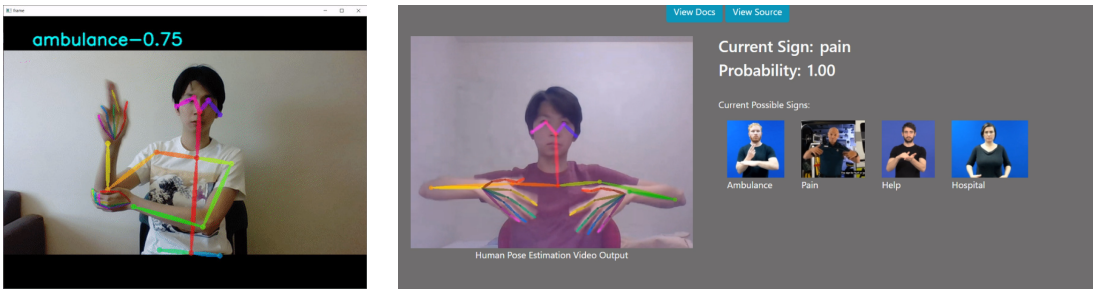
Figure 15: Various Image Transformations for Testing Stage

As in Section: 4.0.3, the rotation operations were randomly parameterised from $(-10°, +10°)$, and the Zoom-in operation was randomly parameterized from ratio: $(1, 1:5)$. In total, we were able to synthesize up to 10,000 raw videos for the test set.

[25] Image rotation about the y-axis and the z-axis respectively

4.0.5 System Deployment

One of the biggest achievements that we have accomplished was the development of an application that deploys our working model. Our model was deployed both as a web and desktop application - featuring capability of performing gesture recognition from a live camera feed.



(a) Desktop Application **(b)** Web Application

Figure 16: Demonstration of our deployed application in both forms.

Recognition through a live camera feed requires the need for a specialised data structure to store key-points dynamically. Thus, both applications fundamentally rely on a core data structure in storing key-points dynamically for classification - **a rolling window**.

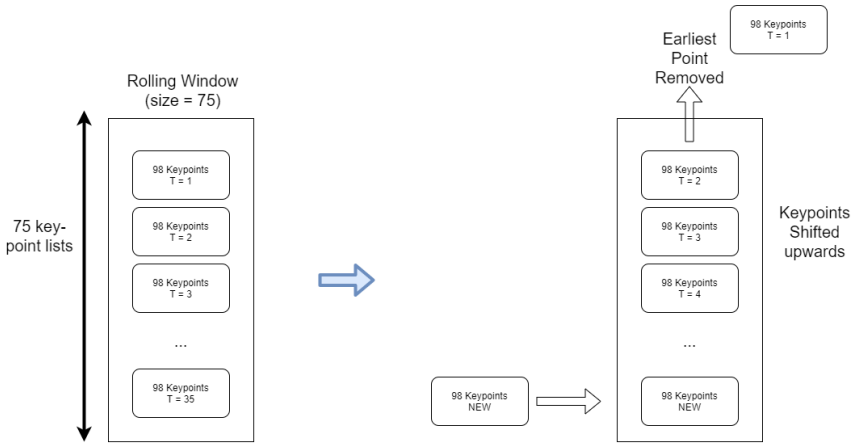


Figure 17: Illustration of the rolling window data structure.

The rolling window acts similar to a shift register for lists of key-points. It has length 75 to store 75 captured key-point lists ready for model classification, each having 98 key-points. First, we instantiate the rolling window with 75 zero valued key-point lists. Then, for each new key-point

list generated from OpenPose, it is inserted to end of the window (index = 75) while all other key-point list's index decrements by 1. The earliest key-point list (index = 0) is popped out of the window. These queue like operations enable our model to receive a sequence of 75 key-point lists (in order) all the time for classification. Refer to 9.3 for it's code implementation.

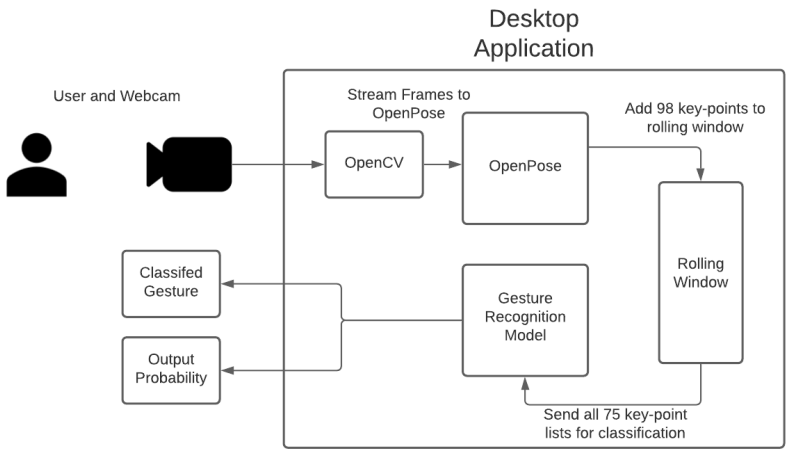


Figure 18: Working principle of the desktop application

The desktop application was developed using OpenCV-python and OpenPose's Python API. The working principle starts with the user connecting through a webcam. Then, frames are streamed individually at a fast rate to the OpenPose processor, which outputs corresponding list of key-points per frame. Then, the key-point lists are added to the rolling window data structure. For each added key-point, the model performs classification over the 75 key-point lists currently in the rolling window. The resulting classification and score are then displayed on a graphical user interface for the user to see.

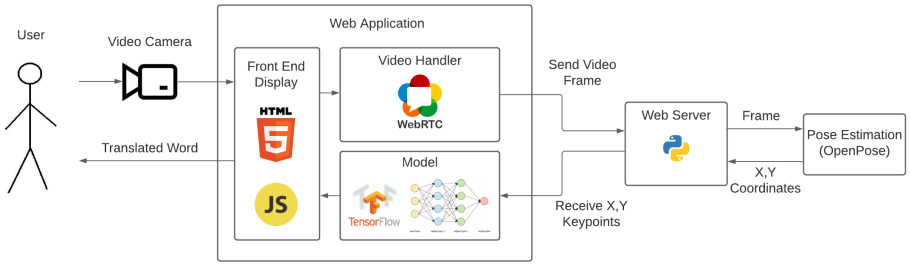


Figure 19: Block diagram view of the web application

Similarly, the web application performs the same core operations, but uses the user's browser

to perform the following tasks: (1) connect and stream live user frames from webcam to server (2) perform gesture recognition on browser. The reason for decision (2) was to limit the usage of computation resources on our server, thus dedicating it run only as an OpenPose processing server.

From Figure 19, the user connects to their webcam through a front-end display. Then, we use WebRTC, an open framework for real-time communication on the web to handle streaming of video frames to our OpenPose server. It is built on top of the User Datagram Protocol (UDP) rather than the typical Transmission Control Protocol (TCP) as we do not need reliability in transmission of frames for faster speed (Sequeira, 2017). The OpenPose processing server receives and translates frames into key-points, which are then transmitted back to the user's browser via websockets. Websocket is a full duplex communication channel that operates at a lower latency as compared to HTTP (Lubbers and Greco, 2017). The browser receives these key-points, adds them to a rolling window and performs classification, outputting the gesture and it's probability. The gesture recognition model is deployed on the browser using Tensorflow.js a Javascript based framework for machine learning on browsers.

5 Results and Observations

5.0.1 Model Selections via K-fold Cross Validation

Table 2: Set of model configurations used for both 35-frame and 75-frame settings.

Model (35 or 75)	**1**	**2**	**3**	**4**	**5**	**6**
No. of LSTM layers*	1	1	1	2	2	2
No. of Hidden Units	32	64	128	32	64	128

* The layers are with hyperbolic tangent activation function.

Table 3: Accuracy Comparison of Different Model for 35-frame via K-Fold Cross Validation.

Model (35)	**1**	**2**	**3**	**4**	**5**	**6**
Average Accuracy,%	95.58	98.21	99.27	97.30	99.02	99.44
Standard Deviation	0.3625	0.1888	0.1336	0.3319	0.0900	0.1165

Observations:

This section presents the results on the model selections via K-fold Cross Validation. Recall that we have two different settings resulting from 35-frame and 75-frame. Each setting has six

Table 4: Accuracy Comparison of Different Models for 75-frame via K-Fold Cross Validation.

Model (75)	**1**	**2**	**3**	**4**	**5**	**6**
Average Accuracy,%	93.05	96.27	98.33	94.27	97.56	98.72
Standard Deviation	0.7879	0.3487	0.2817	0.2693	0.3735	0.3500

Table 5: Model Size Comparison of Different Models for 35-frame and 75-frame.

Model (35 or 75)	**1**	**2**	**3**	**4**	**5**	**6**
Model Size (kilobyte)	225	521	1397	332	917	2949

(6) models with different hyperparameters in the form of the number of LSTM layers and the number of hidden units. This is summarized in Table: 2. For example, Model 1 has one LSTM layer with 32 hidden units, and so on.

From both Tables: 3 and 4, the reported average accuracies achieve above 90%. It is safe to claim that these values are inflated. This is because these validations were evaluated on data synthesized by the team.

Nevertheless, these results provide insights in which combination of the hyperparameters is optimal in terms of the accuracy. In both 35-frame and 75-frame settings, Model 6 which has the highest number of LSTM layers and hidden units achieved the highest accuracies as shown in Tables: 3, 4.

However, we have chosen to deploy Model 5 in both settings where it has two LSTM layers and 64 hidden units for this Gesture Recognition System. This is because in 35-frame setting, we observe that Model 3, 5, and 6 achieve equally decent performance where the accuracy exceeds 90%, but the model sizes are different as shown in Table: 5. A similar observation is made in 75-frame setting. Model 5 offers the smallest model size for an equally decent performance.

5.0.2 Final Evaluation on Test Data-set

Table 6: Test Accuracy for Each Gesture using Model 5 in 35-frame setting.

Gesture	**A**	**B**	**C**	**D**
Average Accuracy,%	68.15	67.62	65.05	61.10
Standard Deviation	7.907	5.095	1.669	7.151

Observations:

Table 7: Test Accuracy for Each Gesture using Model 5 in 75-frame setting.

Gesture	**A**	**B**	**C**	**D**
Average Accuracy,%	75.49	73.88	71.06	69.08
Standard Deviation	9.220	7.025	2.028	7.473

This section presents the result on the average accuracy of the deployed[26] model in recognising individual gesture evaluated on a test dataset. This is obtained for both 35-frame and 75-frame setting. Please refer to Section: 4.0.4 for the procedure to obtain this test dataset. The deployed model is Model 5 as justified in Section: 5.0.1 above.

By comparing the average accuracies presented in Tables: 6 and 7, we observe that the overall accuracy in 75-frame setting is higher than in 35-frame setting. Also, not all gestures were equally well recognized. The recognition of *Gesture A* was the most accurate, while the recognition of *Gesture D* was the least accurate.

5.0.3 Scalability of the Multiclass Classification

Table 8: Variation of the Test Accuracies with the Number of Gestures using 35-frame Model.

No. of Gesture	**2**	**3**	**4**
Average Accuracy,%	79.37	71.19	65.48
Standard Deviation	3.132	1.531	1.910

Observations:

This section presents the scalability of the Model in terms of the number of gestures. This is evaluated using Model 5 in 35-frame setting. The model was retrained[27] only for the specified number of gestures, and evaluated on a test dataset that consisted only the corresponding gestures.

From Table: 8, we observe that the accuracy performance degrades as the number of gestures increases. Note that the reported accuracy was the average accuracy across the gestures rather than for an individual gesture.

[26]This is the final model deployed for the Gesture Recognition System.

[27]Formally, we consider a model that is trained to recognize a x number of classes to be different than a model that is trained to recognize y number of classes for $x \neq y$ even if the model structure is the same.

5.0.4 Model Classification Latency

Table 9: Latency metrics 35 and 75 frame models with respect to GPU memory allocation.

Allocated memory	**CPU only**	**256 MB GPU memory**	**512 MB GPU memory**	**Elastic* GPU growth**
35 Frame latency (s)	1.2	0.63	0.92	1.5
75 Frame latency (s)	1.6	0.85	1.1	2.3

* Tensorflow's automated GPU memory allocation according to model's requirements.

Observations:

To perform evaluation on the model's computational speed, we utilised the desktop application together with the rolling window data structure for online classification. Here, we define a successful classification when a model is able to classify and sustain a given gesture for 10 consecutive frames. To simulate model's performance on different computational devices, we allocated various amounts of GPU memory for the model to use when performing its classification.

Table 9 presents the measure of computational resources needed to perform effective classification of gestures. This is evaluated using Model 5 for both 35 and 75 frame setting and measured its classification latency based on different memory allocations.

We see that both models show similar decrease in latency with higher GPU memory allocated. However, we observe that latency starts to increase again after GPU allocation of 256 Megabytes and continues to grow as seen in the elastic GPU growth scenario.

6 Discussion

The trade-off between the model size and the average validation[28] accuracy shows that Model 5 which has the two LSTM layers, each with 64 hidden units is the optimal model among all the models under consideration. Empirically, this suggests that for a given number of hidden units, greater depth in terms of the layers does seem to result in better model performance. We could appeal to the universal approximation theorem which states that depth-2 networks with suitable activation functions can approximate any continuous function[29] on a compact domain to any desired accuracy (Lu et al. (2017), Barron (1994)).

[28]This is performed using the K-fold Cross Validation.

[29]In this project, the tangent activation function is used.

However, this does not include the existence of better models whether it is the choice of the hyperparameters or entirely different architectures such as CNN-LSTM (Mutegeki and Han (2020)) and Transformer (Girdhar et al. (2019)). There are two reasons. First, the universal approximation theorem does not consider the algorithmic learnability of the hyperparameters, hence it does not guarantee a way to train a model to reach the theoretical performance. Second, the model configurations considered in this project leveraged findings reported in the literature[30]. The transferability of model hyperparameters or model architecture that resulted in a decent model for one application to another similar application is not guaranteed.

From the final evaluation, the deployed model achieved an average test accuracy less than the specified 80% accuracy in both 35-frame and 75-frame settings. In contrast to the achieved validation accuracy which exceeded 90%, the relatively low test accuracy indicates the trained models generalised poorly, implying a possibility of over-fitting. This is because the models were trained on data that was synthesised by transformations that were completely different to the transformations applied to synthesise test data. So, the training data may not be sufficiently (statistically) representative.

There is a major defect in the experimental procedure performed in evaluating the test accuracy. Since the test data in actuality was collected from a small invited group of people and then increased in size by augmenting the collected gestures with various video transformation techniques, the test data itself may not be sufficiently representative, and is less "challenging" where significant occlusions and missing parts are absent. So, it remains unknown how well the trained model will perform if it was to be deployed in any reasonable real-world environments. Also, in the studying of the generalization bounds of multi-class classification algorithms, convergence rates of the existing generalization bounds are usually $\mathcal{O}(K^2/\sqrt{n})$, where K and n are the number of classes and size of the training data, respectively (J. Li et al., 2018). As argued above, the generalization error could not be estimated. As such, it is also premature to assert that our training data size is sufficiently large.

In light of the arguments above, if there exists a better training data in terms of size and representation, the average test accuracy of approximately 70% achieved could be improved. On this basis, Human Pose Estimation (HPE) as the extracted feature could be used to recognise a gesture. Rwigema et al. (2019) had success in achieving at least 80% test accuracy using the UTD-MHAD dataset and Moryossef et al. (2020) achieved at least 84.3% on the German Sign Language Public DGS Corpus in doing gesture recognition.

Also, from the final evaluation, the test accuracy in the 75-frame setting is higher than the accuracy achieved in the 35-frame setting. This indicates that there is a information loss when the video frame is reduced from 75-frame to 35-frame[31]. Also, from the same result, not all gestures

[30] See Method Section: 4.0.5.

[31] See Method Section:4.0.5 where the 35-frame is obtained by sampling the 75-frame at equal interval.

were recognized equally accurate. Thus, the significance of the frame selection differs among the gestures. Problems of frame sub-sampling and selection allude to the fact we lack of the necessary domain knowledge and the experience with modelling the underlying problem with neural network. Nevertheless, the lowest test accuracy achieved in 35-frame setting only differs the corresponding test accuracy achieved in 75-frame setting by 7.98%. Moreover, under certain conditions, the generalization error bound of a vanilla RNN has a polynomial dependency on the sequence length[32] (M. Chen et al., 2019). So, it is possible to approximate the final representation learned by the model in 75-frame setting using fewer frames while obtaining an approximately similar performance.

As for the scalability of the multiclass classification, the results obtained are not definitive to assert that the model performance will degrade with an increasing number of classes. This is because due to time and resource constraints, the data collection is only limited to a maximum four classes (gestures) which is insignificant to extrapolate the variation of the model performance with the number of classes. Thus, the results give little basis for the assertion. However, as shown above, this generalization bound, $\mathcal{O}(K^2/\sqrt{n})$, where K and n are the number of classes and size of the training data respectively, indicates that for a given training size, n, the model performance will degrade as the number of classes increase. This is consistent with D. Li et al. (2020) where model accuracy dropped from 89.98% to 66.31% as the number of classes increased from 100 to 2000 for classification.

(a) Sample raw image of gesture **(b)** OpenPose's noisy output

Figure 20: An example where OpenPose fails to detect hand key-points accurately (right), despite having a clear representation in the RGB image (left).

Also, there is a major defect in the experimental procedure in evaluating the scalability of the multi-class classification. The evaluation could only be performed on the selected gestures that are in our training database. There could be some edge cases (gestures) that will break the system. This illustrates the challenge to scale the system to recognize arbitrary gestures. For

[32]It is equivalent to the number of frames in this project.

example, due to the limitation of OpenPose, there are certain gestures such as this arbitrary gesture in Figure 20 that OpenPose is not able to recognize accurately. Here, despite having a clear image input, OpenPose is unable to distinguish minute finger positions when there is hand to hand interaction Simon et al. (2017). To fix this, De Coster et al. (2020) suggested to retrain the OpenPose model specifically for human gesture recognition, rather than using it purely as a fixed feature extractor, which is beyond the scope of this project.

In terms of computational performance, we note that our system heavily relies on human pose estimation software for key-points - implying that GPU resources are shared between both OpenPose and our model. Our system uses an Nvida GRID P40-3Q virtual GPU - containing approximately 8 GB of total GPU RAM[33], and we observed that allocating 3.2% (256 Mb) of it's memory was sufficient for lowest latency while sufficiently reserving the rest for human pose estimation. Thus, we see that the major limitation of our system's recognition speed heavily relies on the performance of OpenPose in doing human pose estimation. A possible work around in having to scale our system to perform faster recognition would be the use of a distributed network of GPUs - using tools such as Apache Spark[34] and Kubernetes[35] for compute cluster management.

7 Conclusions and Recommendations

In this work, we built a gesture recognition system that recognized four (4) isolated gestures in a continuous stream of video data. We were able to train a LSTM model that utilized only human keypoints as the extracted feature estimated by an open-source project, OpenPose developed by Cao et al. (2019). To improve the latency of the recognition system, we also explored the possibility of using a fraction of the available video frame to perform the inference. This resulted in two settings, where we had 75-frame and 35-frame to perform the inference.

In the 75-frame setting, an average accuracy of approximately 70% was achieved where the LSTM model was trained using 30,000 synthesized videos. In the 35-frame setting, an average accuracy of approximately 65% was achieved using 60,000 synthesized videos. In either case, the built gesture recognition system did not achieve the specified accuracy of at least 80%. Nevertheless, the results indicate that a neural network trained using features extracted by a pose estimation technique is able to recognize a gesture adequately.

Nevertheless, this illustrates the challenge to collect a sufficient amount of training data of high quality. Various data augmentation techniques such as using Generative Adversarial Networks

[33]Random Access Memory

[34]Library for distributed systems programming

[35]An open-source system for automating deployment, scaling, and management of containerized applications.

(Jain et al. (2019)) and using the spatial properties of videos (Ko et al. (2019)) could be explored to expand the training dataset. There exist other modalities such acceleration forces and angular velocity captured by sensors (Chevalier (2016)) that could be used in conjunction with human keypoints to perform recognition. Also, different sets of feature could be combined which may leads to model such as CNN-LSTM where the accuracy could be further improved beyond what is possible when considering either feature set in isolation (De Coster et al. (2020)).

Although the drop in the accuracy performance when inferring using the reduced 35-frame setting from 75-frame setting is considerable, the improvement in the inference time by 25.5% warrants the consideration of using fewer frames to train a neural network. Nevertheless, the richer information of 75-frame at the disposal could be used to complement the model trained using 35-frame. This leads to the idea of having a compute-heavy teacher which looks at all the frames in the video used to train a compute-efficient student which looks at only a small fraction of frames in the video (Bhardwaj et al. (2019)).

Finally, the results indicate that not all the gestures could be recognized equally well based on equal interval frame sampling. Deploying reinforcement learning agent to select which frames to process could be explored (Y. Chen et al. (2018)). Moreover, the assumption on all gestures taking the same amount of duration to complete is strong leading to a fixed-length rolling window. The relaxation of this assumption could be explored through the use of dynamic rolling window.

8 Acknowledgements

First and foremost, we would like to thank Professor Jonathan for supervising us in throughout project journey, providing great guidance and help.

We would also like to extend our gratitude to Mr. Lucas Barbuto from the Melbourne School of Engineering IT (MSE-IT) Department, in helping us set up MSE-IT Virtual Machines in order to provide us with computational resources and power to train and deploy our artificial intelligence model.

Last but not least, we would like to thank Mr. Amit Moryossef for his helpful insights and knowledge on system development and model enhancement for the purpose of gesture recognition.

References

Ahmed, M. A., Zaidan, B. B., Zaidan, A. A., Salih, M. M., & bin Lakulu1, M. M. (2018). A review on systems-based sensory gloves for sign language recognition state of the art between 2007 and 2017.

Allevard, T., E.Benoit, & L.Foulloy. (2005). Hand posture recognition with the fuzzy glove.

Barron, A. (1994). Approximation and estimation bounds for artificial neural networks.

Bergstra, J., Yamins, D., & Cox, D. . (1995). A study of crossvalidation and bootstrap for accuracy estimation and model selection.

Bhardwaj, S., Srinivasan, M., & Khapra, M. M. (2019). Efficient video classification using fewer frames.

Book, D., & Fisher, G. (2018). Signnet: A neural network asl translator.

Bragg, D., Koller, O., Bellard, M., Berke, L., Boudreault, P., Braffort, A., Caselli, N., Huenerfauth, M., Kacorri, H., Verhoef, T., Vogler, C., & Morris, M. R. (2019). Sign language recognition, generation, and translation: An interdisciplinary perspective.

Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., & Sheikh, Y. A. (2019). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Cartmill, E. A., Beilock, S., & Goldin-Meadow, S. (2012). A word in the hand: Action, gesture and mental representation in humans and non-human primates.

Chen, K., Gabriel, P., Alasfour, A., Gong, C., Doyle, W. K., Devinsky, O., Friedman, D., Dugan, P., Melloni, L., Thesen, T., Gonda, D., Sattar, S., Wang, S., & Gilja, V. (2018). Patient-specific pose estimation in clinical environments.

Chen, M., Li, X., & Zhao, T. (2019). On generalization bounds of a family of recurrent neural networks.

Chen, Y., Wang, S., Zhang, W., & Huang, Q. (2018). Less is more: Picking informative frames for video captioning.

Chevalier, G. (2016). Lstms for human activity recognition.

De Coster, M., Van Herreweghe, M., & Dambre, J. (2020). Sign language recognition with transformer networks. *Proceedings of the 12th Language Resources and Evaluation Conference*, 6018–6024. https://www.aclweb.org/anthology/2020.lrec-1.737

Dong, C., Leu, M. C., & Yin, Z. (2015). American sign language alphabet recognition using microsoft kinect.

Fang, H.-S., Xie, S., Tai, Y.-W., & Lu, C. (2016). Rmpe: Regional multi-person pose estimation.

Filming the signers(s) [Accessed: 2020-11-4]. (n.d.).

G.Bosworth, R., E.Wright, C., & R.Dobkins, K. (2019). Analysis of the visual spatiotemporal properties of american sign language.

G.E.Hinton, N.Srivastava, A.Krizhevsky, I.Sutskever, & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

Girdhar, R., Carreira, J., Doersch, C., & Zisserman, A. (2019). Video action transformer network.

Güler, R. A., Neverova, N., & Kokkinos, I. (2018). Densepose: Dense human pose estimation in the wild.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory.

Hosain, A. A., Santhalingam, P. S., Pathak, P., Rangwala, H., & Kosecka, J. (2019). Sign language recognition analysis using multimodel data.

Hosain, A. A., Santhalingam, P. S., Pathak, P., Rangwala, H., & Kosecka, J. (2020). Finehand: Learning hand shapes for american sign language.

Hu, B., Lei, C., Wang, D., Zhang, S., & Chen, Z. (2019). A preliminary study on data augmentation of deep learning for image classification.

Huang, Y., Sun, B., Kan, H., Zhuang, J., & Qin, Z. (2019). Followmeup sports: New benchmark for 2d human keypoint recognition.

Jain, V., Aggarwal, S., Mehta, S., & Hebbalaguppe, R. (2019). Synthetic video generation for robust hand gesture recognition in augmented reality applications.

Kenneth Research. (2020). Gesture recognition and touchless sensing market : Industry development scenario and forecast. Retrieved November 7, 2020, from https://www.marketwatch.com/press-release/gesture-recognition-and-touchless-sensing-market-industry-development-scenario-and-forecast-2020-09-07

Kim, J., & O'Neill-Brown, P. (2019). Improving american sign language recognition with synthetic data.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization.

Ko, S.-K., Kim, C. J., Jung, H., & Cho, C. (2019). Neural sign language translation based on human keypoint estimation.

Lai, K., & Yanushkevich, S. N. (2020). Cnn+rnn depth and skeleton based dynamichand gesture recognition. https://arxiv.org/pdf/2007.11983v1.pdf.

Li, D., Rodriguez, C., Yu, X., & Li, H. (2020). Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison. *The IEEE Winter Conference on Applications of Computer Vision*, 1459–1469.

Li, J., Liu, Y., Yin, R., Zhang, H., Ding, L., & Wang, W. (2018). Multi-class learning: From theory to algorithm. *NeurIPS*.

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning.

Lu, Z., Pu, H., Wang, F., Hu, Z., & Wang, L. (2017). The expressive power of neural networks: A view from the width. *CoRR*, *abs/1709.02540*. http://arxiv.org/abs/1709.02540

Lubbers, P., & Greco, F. (2017). Html5 websocket: A quantum leap in scalability for the web.

Malhotra, P., Maniar, C. K., Sankpal, N. V., & Thakkar, H. R. (2019). Indian sign language recognition system using openpose.

Mkom, S., Zain, J. M., & Majid, M. A. (2013). *Vision-based sign language recognition systems : A review*.

Mocialov, B., Turner, G., Lohan, K., & Hastie, H. (2017). Towards continuous sign language recognition with deep learning.

Molchanov, P., Gupta, S., Kim, K., & Kautz, J. (2015). Hand gesture recognition with 3d convolutional neural networks.

Moryossef, A., Tsochantaridi, I., Aharoni, R., Ebling, S., & Narayanan, S. (2020). Real-time sign language detection using human pose estimation.

Mutegeki, R., & Han, D. S. (2020). A cnn-lstm approach to human activity recognition. *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 362–366. https://doi.org/10.1109/ICAIIC48513.2020.9065078

Olah, C. (2015). Https://colah.github.io/posts/2015-08-understanding-lstms/ [Accessed: 2020-10-29].

Oz, C., & Leu, M. C. (2005). Linguistic properties based on american sign language recognition with artificial neural networks using a sensory glove and motion tracker.

Park, C.-I., & Sohn, C.-B. (2020). Data augmentation for human keypoint estimation deep learning based sign language translation.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Poizner, H., Bellugi, U., & Lutes-Driscoll, V. (1981). Perception of american sign language in dynamic point-light displays.

Radecic, D. (2019). Softmax activation function explained.

Reimers, N., & Gurevych, I. (2017). Optimal hyperparameters for deep lstm-networks for sequence labeling tasks.

Rwigema, J., Choi, H.-R., & Kim, T. (2019). A differential evolution approach to optimize weights of dynamic time warping for multi-sensor based gesture recognition.

Sequeira, P. M. (2017). Webrtc – architecture & protocols.

Simon, T., Joo, H., Matthews, I., & Sheikh, Y. (2017). Hand keypoint detection in single images using multiview bootstrapping. *CVPR*.

Sprawls, P. (n.d.). Image characteristics and quality [Accessed: 2020-11-4].

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting.

V., A., & R., R. (2020). A deep convolutional neural network approach for static hand gesture recognition.

Wang, J., & Perez, L. (2017). The effectiveness of data augmentation in image classification using deep learning.

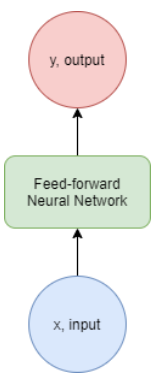
Wang, X., Xia, M., Cai, H., Gao, Y., & Cattani, C. (2012). Hidden-markov-models-based dynamic hand gesture recognition.

Williams, R. J., & Zipser, D. (1992). Gradient-based learning algorithms for recurrent networks and their computational complexity.

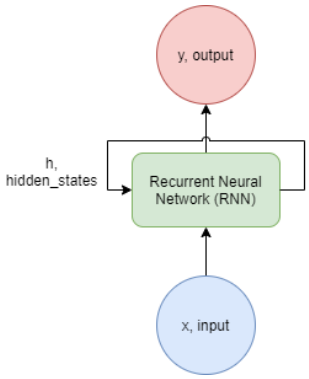
9 Appendix

9.1 Background to RNNs and LSTMs

As we are dealing with time-series classification, we rely on the use of Recurrent Neural Networks, often shortened to RNNs to handle sequential data information. RNNs are a class of neural networks - derived from the feed-forward neural networks. RNNs uses recurrence as a shared mechanism where the model can learn using both current and past data in a sequence.



(a) Structure of a feed-forward neural network.



(b) Structure of a recurrent neural network.

Figure 21: Comparing architectures of both feed-forward and recurrent neural networks

Referring to Figure 21, we see that the output of a typical feed-forward neural network only depends on a fixed length input. This makes it difficult to understand temporal dependencies between data points given a sequence. Alternatively, RNNs uses recurrence of it's past output - making the current output h_t a function of both it's current input x_t and it's past output h_{t-1}.

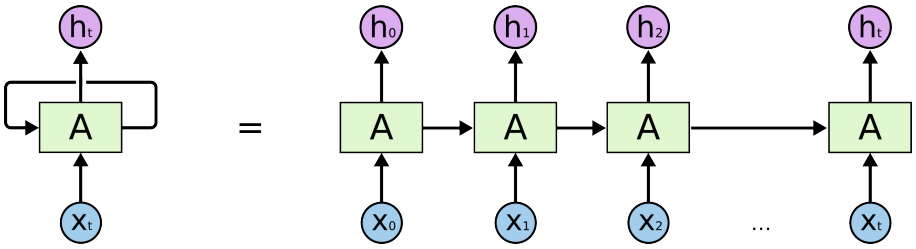


Figure 22: Rolled up RNNs (left) can be seen as a recurrence of it self in time (right) (Olah, 2015)

Given input x_t, hidden state output h_t and their respective weight matrices W_h and W_x, we can describe the recurrence relationship as:

$$h_t = tanh(W_h h_{t-1} + W_x x_t)$$

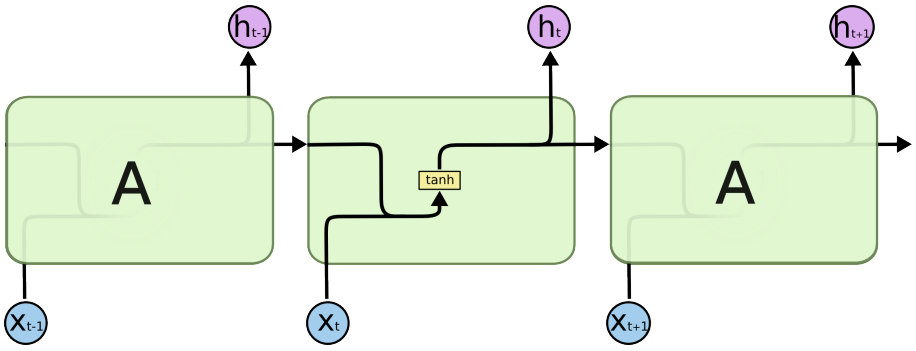


Figure 23: Illustration of recurrence in a chain of RNN cells (Olah, 2015)

While RNNs perform well in understanding temporal dependencies from past data points, they suffer from the vanishing gradient problem. During training of RNNs, the network uses the back-propagation through time (BPTT) algorithm - where error signals from training "flow back in time" for weight adjustments (Williams and Zipser, 1992). As the backpropogated error depends exponentially on it's weights, it has the possibility for weight gradients to vanish,

impeding proper training on points distant in a sequence.

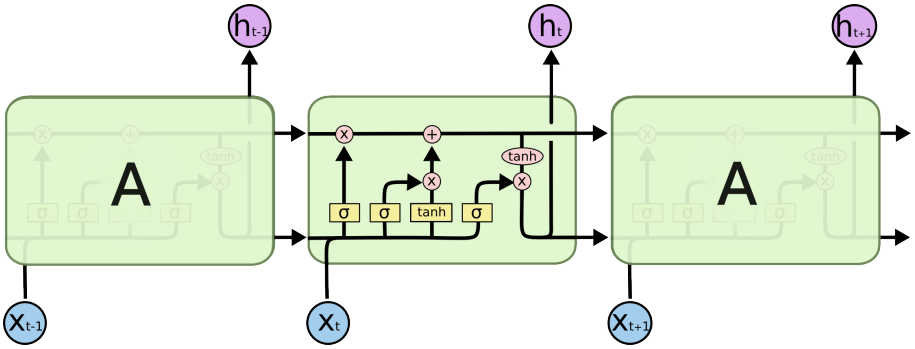


Figure 24: Deep look into an LSTM cell, using gating functions (Olah, 2015)

To address the vanishing gradient problem, Hochreiter and Schmidhuber (1997) proposed the use of Long Short Term Memory Networks (LSTMs) - a variation of classical Recurrent Neural Networks. The core feature of LSTMs, is the addition of a new parameter, known as the cell state (C_t) - that aims to store the global information of it's sequential data, ready for the cell to easily use, update or select from. To do so, it incorporates gating functions to control a cell's ability to (1) choose what to forget from cell's state, (2) update current cell's state, (3) select cell state to update hidden state output. Mathematically, these gates perform the following operations:

$$u_t = [h_{t-1}, x_t]$$

$$f_t = \sigma(W_f u_t + b_f)$$

$$i_t = \sigma(W_i u_t + b_i)$$

$$\tilde{C}_t = tanh(W_C u_t + b_c)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

where u_t represents the concatenation of past cell output h_{t-1} and current input x_t, f_t is the "forget-gate" output that controls whether to forget the current cell's state. i_t is the "input-gate", controlling whether to update current cell state using a potential new cell state candidate $\tilde{C}_t$ and C_t is the update operation of updating the current cell state. The operation "*" represents an element-wise multiplication. Finally, the final output of this cell h_t, is computed as the following for the next stage in recurrence:

$$o_t = \sigma(W_o u_t + b_o)$$

$$h_t = o_t * tanh(C_t)$$

9.2 Data Text Output Format for Training

This is a view of a sample X text, `X.txt` file used to store a flatten version of the mentioned 98 key-points from OpenPose. Each 75 rows in the X.txt file maps to a single output class (enumerated from dictionary above).

```
[x1 y1, x2 y2, x3, y3 ... x98, y98]  --> For Sign Pain, Frame 1
[x1 y1, x2 y2, x3, y3 ... x98, y98]  --> For Sign Pain, Frame 2
...
[x1 y1, x2 y2, x3, y3 ... x98, y98]  --> For Sign Pain, Frame 75
[x1 y1, x2 y2, x3, y3... x98, y98]  --> For Sign Help, Frame 1
[x1 y1, x2 y2, x3, y3 ... x98, y98]  --> For Sign Help, Frame 2
...
[x1 y1, x2 y2, x3, y3 ... x98, y98]  --> For Sign Help, Frame 75
```

And corresponding Y text, `Y.txt` file used to label each gesture output.

```
3 --> Indicates series of this 75 arrays represents the sign "pain"
1 --> Indicates series of this 75 arrays represents the sign "help"
```

9.3 Rolling Window Data Structure

This is the implementation of the described rolling window structure as a python class.

```
import numpy as np
class RollingWindow:
    def __init__(self, window_Width, numbJoints):
        # Features
        self.window_Width = window_Width
        self.numbJoints   = numbJoints
        # Create a 2D Matrix with dimensions
        self.points       =
        ↪  np.zeros(shape=(window_Width,numbJoints))
        # Frame to add pointer
        self.isReset      = True

    def getPoints(self):
        # Return the 2D matrix of points
        return self.points

    def addPoint(self, arr):
        # Check for same number of joints in input array before
        ↪  we insert
        if ( len(arr) != self.numbJoints ):
```

```
    print("Error! Number of items not equal to
     ↪  numbJoints = ", self.numbJoints)
    return False
# shift register;
# remove the oldest from the first index;
# add the most recent entry always enters from the
 ↪  "last" index
self.points = np.delete(self.points, 0, 0)
self.points = np.vstack([self.points, arr])
return True
```