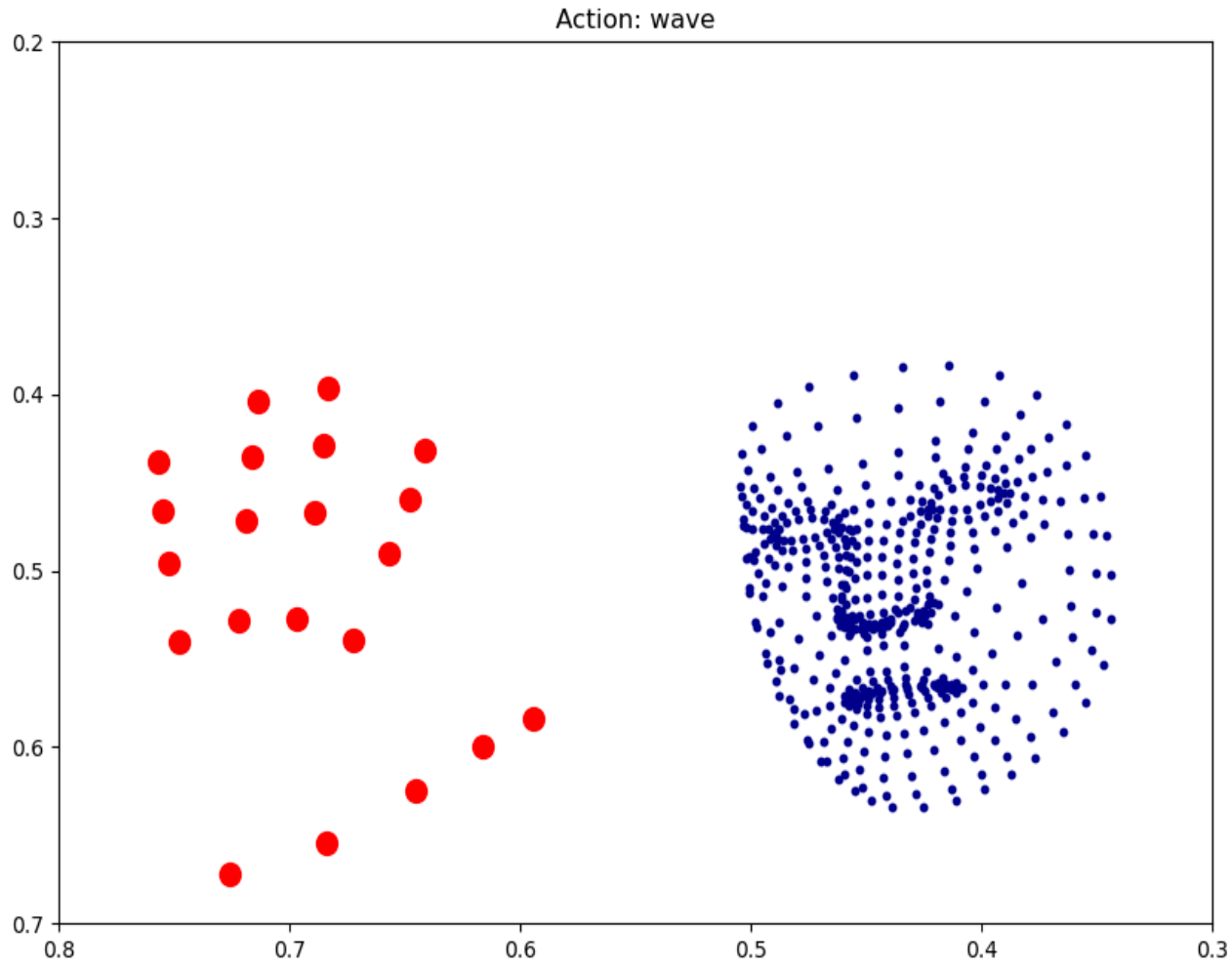
ACTION RECOGNITION

A case study on building and deploying an action recognition model to facilitate human-computer interactions



Author: Liam Moorfield-Yee

Date: June, 2023

RESEARCH AIM:

Action recognition is the process of interpreting human motion using computer vision. Use cases for the technology are broad with applications across, but not limited to, security systems, human-computer interactions, human behaviour analysis, and entertainment and gaming.

With the rise of large language models, this project focuses on creating a deep learning model to identify human gestures to further develop human-computer interactions. As a large portion of communication is non-verbal, having a model to identify a person's non-verbal communication is important to develop systems which can fully assess all aspects of an interaction before responding. This will allow for more personalised and precise communication between AI models and humans.

The select actions were chosen due to their distinct and clear motions, and unambiguous meanings. These actions formed a strong baseline model with an overall precision and recall of 97%+ on both the test and train data. The structure laid out in this project provides a framework to easily collect additional data and continually add more actions to future models. The six actions included in the current algorithm are:

1. Wave
2. Kiss
3. Heart
4. Finger
5. Salute
6. Idle

DATA:

The raw data collection process consisted of **manually** generating 210 raw video clips for the 6 different actions (wave, kiss, middle finger, heart, salute, and idle), with each video sample consisting of 40 frames. Once the video clips were generated, every frame in each video sample was processed using Google's mediapipe holistic model to identify and extract key landmarks on the human body.

Each landmark extracted contains a corresponding x, y, & z value, with a visibility value included for the body's pose landmarks. In total there were 543 landmarks extracted by the model in every frame. These landmarks are then used as data points for the training of our machine learning models. Taking into account each x, y, z, & visibility values for each landmark, there are 1662 data points generated for each frame.

Body Groupings & Associated Landmarks

- Number of Pose landmarks = 33 (each with an x, y, z & visibility value) = 132 data points
- Number of Face Landmarks = 468 (each with an x, y, z value) = 1404 data points
- Left Hand Landmarks = 21 (each with an x, y, z value) = 63 data points
- Right Hand Landmarks = 21 (each with an x, y, z value) = 63 data points

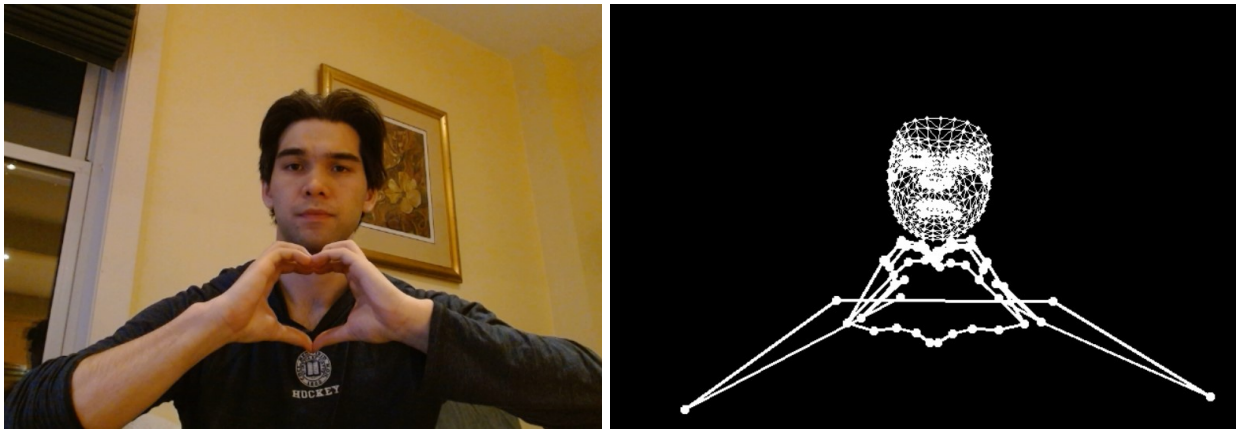
The landmark data for each video frame is then flattened, concatenated, and saved as a numpy array. The reasons why we process the images with the Mediapipe model are below:

1. Using the location of a person's body part within the camera frame substantially reduces the amount of data needed to train the model. If we passed through all of the pixel data in a frame to the model, it would result in 554x more data. This drastically increases model complexity, CPU usage, and runtime. The default resolution on a web camera is 640.0 by 480.0 pixels - that's a total of 307,200 pixels. With each pixel having 3 colour channels, the total amount of features per frame would be 921,600 - substantially more than our 1662 landmark values.

2. Training an object recognition model is a difficult task in and of itself, and Mediapipe's holistic model provides a stable object detection algorithm which works in a variety of different environments. By utilising Mediapipe's holistic model, we do not need to be concerned with external factors, such as background and lighting, which would otherwise impact our model's performance.

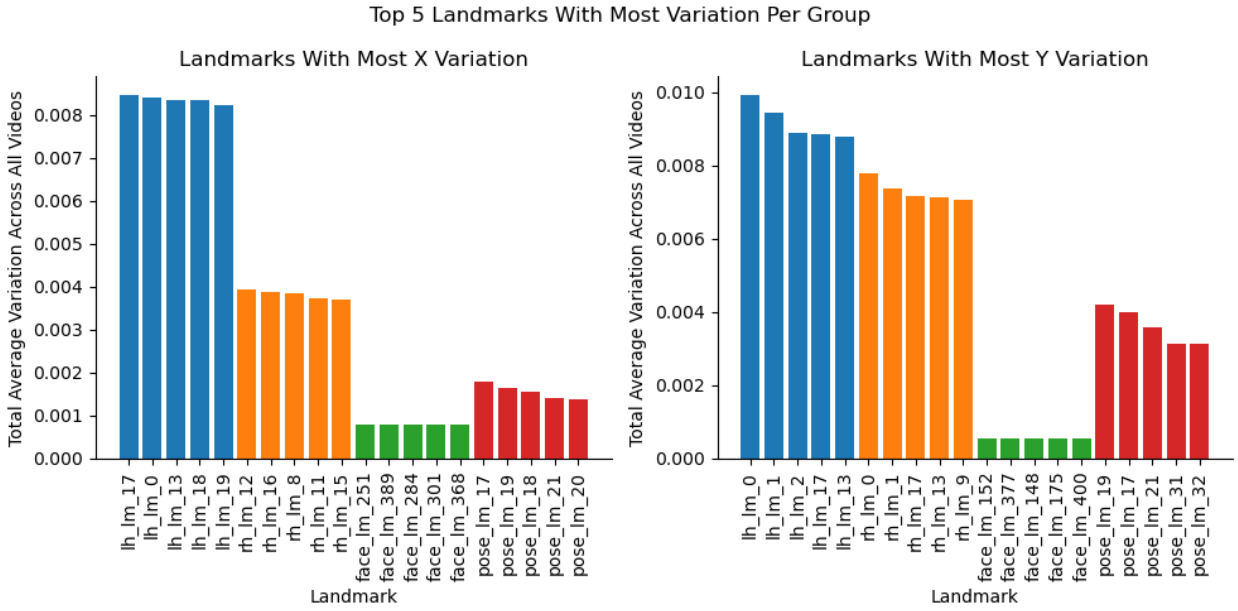
3. By only passing through body location data, there is no other external noise. This provides clean data for our model to detect actions and gestures.

Below is a video frame before processing and post processing.



IMPORTANT FEATURES (LANDMARKS)

When evaluating whether a feature will be a good explanatory variable, a common method is to look at how much variation each feature has. The rationale is that the more variation a feature has across different classes, the better it will be at distinguishing between them (i.e. it's carrying useful information). In a similar vein, a feature with NO variation across each class is not carrying any information to help the model discern between the classes. With this in mind the below two graphs calculate the amount of X & Y variance each landmark has across every video. The top five landmarks from each grouping are then plotted.



lh_lm = left hand landmark; rh_lm = right hand landmark; face_lm = face landmark; pose_lm = pose landmark

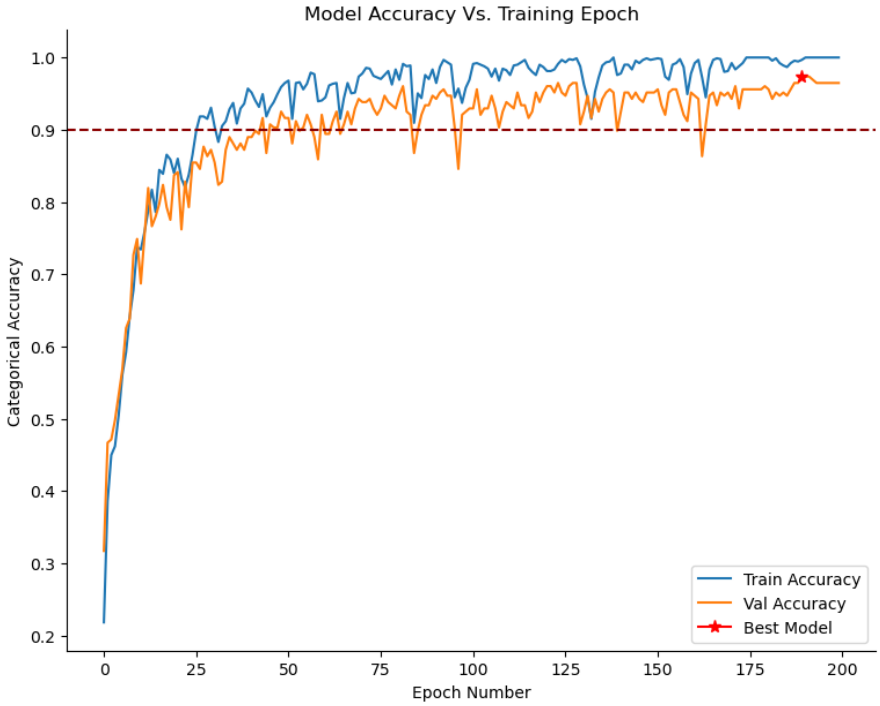
We can see that the right and left hand landmarks have the highest variation, as one would expect as the actions being performed are hand gestures. The pose landmarks, which sparsely encompass the entire body, also have some variability. The pose landmarks which have the most variability unsurprisingly are landmarks 17-22, which are all located on the hands.

It should be highlighted that the face landmarks have very little variability. This is important because the face landmarks make up the vast majority of all landmarks (468 out of 543). As these are not great predictors, we actually removed all of them in our final model, only keeping the hands and pose landmarks, which drastically reduced model complexity and actually increased model performance. Do note that not all face location data was thrown out with the face landmarks as the pose grouping does contain a handful of sparsely located landmarks on the head.

BUILDING A MODEL

Due to the sequential nature of videos, a Long Short-Term Memory (LSTM) Neural Network was trained and deployed due to its ability to retain information from previously seen data. This enables the model to consider the entire 40 frames in each video when making a classification.

The final model performed exceptionally well with a 99% train accuracy, 97% validation accuracy, and a 99% test accuracy. The model's accuracy vs training epoch is below:



The LSTM architecture was the same used within Nicholas Renotte's action recognition tutorial, which can be found here, and learned well across multiple different data sets.

MODEL DEPLOYMENT

To showcase the algorithm, the "main.py" script can be run as a stand-alone application, or be utilised as a backend to send user location and action data to a custom built Unity program. The data flowing into Unity then controls the location and behaviour of an animated 3D character, which is programmed to respond to each action differently. If you would like to see a demonstration please reach out to me at lmoorfieldyee@gmail.com.

NEXT STEPS

The structure laid out in this project provides a framework to easily collect additional data and continually add more actions to future models. In addition, this project provides a proof-of-concept for interactive digital characters which can understand and respond to human body language. The applications for this are broad, and any next steps are largely dependent on what type of problems are to be solved. With that being said, the broad universal improvements to the model and code are as follows:

- Normalise landmark position data to a fixed point on the body (i.e. the nose) in order to improve model accuracy on unseen data. Right now, the model is being trained on location data, meaning that a large part of its learning is coming from memorising where a specific action is taking place in the frame. This is an issue because if a person performs an action in an area that is not covered in the test data (i.e. at the very edge of the frame), the model will struggle with classification. By normalising each landmark value relative to a person's nose, this should help the model focus more on the path of the action being performed, and less on where the action is being performed.
- Add a video collection function to the main.py application to store and collect any new video data when an action is performed. Any additional data collected can be evaluated for accuracy, and then used to re-train the model.
- Change stored data from numpy arrays to sparse matrices. Currently each video frame (both raw and processed data) is stored as its own numpy array which is taking up a massive amount of space. The 1,260 raw videos are currently occupying ~42GB. Updating my code to ingest, transform, and store data as sparse matrices will save a lot of space.
- Deploy my application on the cloud or find a way to compress my algorithms to reduce CPU usage.
- Connect application to ChatGPT to allow verbal conversations. This is actually already done, but I need to figure out: 1) how to run the programs in parallel, & 2) how to speed up voice to text and generative response processes (it currently takes far too long to get a response).
- Automate the entire ML pipeline. Yeah lots to do…