Creating a Video Classification Neural Network Architecture to Map American Sign Language Gestures to Computer Cursor Controls

Rohan Hajela

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-131
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-131.html

May 15, 2022

Acknowledgement

I would like to thank Professor Brian Barsky for giving me the opportunity to join the lab and work on this project over the last couple of years. His advice and feedback have been instrumental to our results on this project. I would also like to acknowledge my team members for this research project: Siddharth Bansal, Jatearoon Boondicharern, Tucker Cullen, Weiyu Feng, Kexin Guo, and Hanqi Zhang for collaborating with me throughout the whole experience and their hard work and eort towards this project.

Creating a Video Classification Neural Network Architecture to Map American Sign Language Gestures to Computer Cursor Controls

Rohan Hajela

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Brian Barsky
Research Advisor

15 May 2022

(Date)

* * * * * * *

Professor Eric Paulos
Second Reader

13 May 2022

(Date)

Creating a Video Classification Neural Network Architecture to Map American Sign Language Gestures to Computer Cursor Controls

Abstract

Creating a Video Classification Neural Network Architecture to Map American Sign Language Gestures to Computer Cursor Controls

by

Rohan Hajela

Masters of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Brian Barsky, Chair

Professor Eric Paulos, Co-chair

Although hand gesture image classification techniques have become fairly powerful, operating with high-accuracy and-low latency, video classification of dynamic gestures lag behind. To explore into this, we have researched into classifying American Sign Language which leverages handshape, palm orientation, movement, location, and expression signals for its gestures. Creating a model that can classify these gestures in real time pose many useful usecases such as a real-time sign language translator or an ASL gesture to English reverse dictionary.

We have built a prototype of a visual based assistant that utilizes video classification on American Sign Language (ASL) gestures to act as short cuts for common commands on computers. We have trained our video classification Ml model with a industry standard level of accuracy and precision, and have configured several common command mappings in this prototype. The system is designed to be scalable, and easy to download and use, and can support the addition of new commands or integration into different operating systems. The goal of this research was to explore various techniques and architectures that would enable a real time hand gesture video classification and to build this model in a modular way where it could be easily applied to other usecases as well.

Contents

List of Figures

Acknowledgments

I would like to thank Professor Brian Barsky for giving me the opportunity to join the lab and work on this project over the last couple of years. His advice and feedback have been instrumental to our results on this project. I would also like to acknowledge my team members for this research project: Siddharth Bansal, Jatearoon Boondicharern, Tucker Cullen, Weiyu Feng, Kexin Guo, and Hanqi Zhang for collaborating with me throughout the whole experience and their hard work and effort towards this project.

Chapter 1

Introduction

1.1 Motivation

With the advance in neural networks, image classification techniques have greatly improved. There are countless papers on effective strategies to perform image classification regarding the neural network architectures, optimization methods, datasets, etc. [16]. Specifically with Hand Gesture Classification, there has been lots of focused research towards image processing techniques, feature selection, and neural network architecture in making high-accuracy low-latency models that can classify hand gestures in real time [11] [25] [27]. Although static image hand gesture classification have become fairly robust, video classification of hand gestures are still in progress, with many industry leading models result in 50% accuracy or below [5] [14]. As such, the motivation to build a robust video classification model that could operate in real time arose. We specifically decided to focus on American Sign Language gesture classification as it was a popular publicly used gesture set that had interesting motion and facial expressions for the model to capture. A model like this could have wide use cases such as a real-time sign language translator or an ASL gesture to English reverse dictionary.

1.2 Background

Working under Professor Barsky's lab, I have been working on the Assistive Technology Cursor Control (ATCC) project for the past 2 years. The goal of our project has been to create assistive technologies to aid or replace mouse and track pad schemes for cursor controls. Previously, we have identified and created various hand tracking algorithms, and worked on image filtering techniques to increase the accuracy and latency of the image capturing.

For this project we have decided to create a video based gesture recognition software rather than the previous image hand tracking models to enable more complex gestures with motion and temporal continuity. The eventual outcome of this project is an application that can be downloaded and configured on a computer that leverages the ML models to classify gestures to trigger shortcuts within the computer to increase accessibility. Specifically, the

implementation we picked utilizes hand tracking and pose detection via Google MediaPipe to identify key points on the hands and body across the 3d coordinate space on the webcam. We passed these key points into a convolutional neural network (CNN) with long short-term memory (LSTM) architectures to provide temporal context into the model.

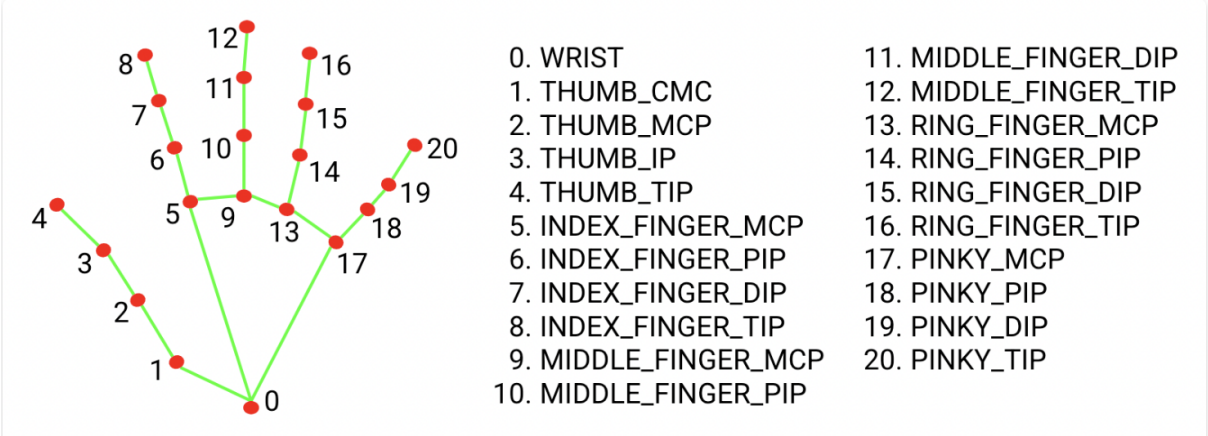

Figure 1.1: Hand keypoints detected by Google MediaPipe [21]

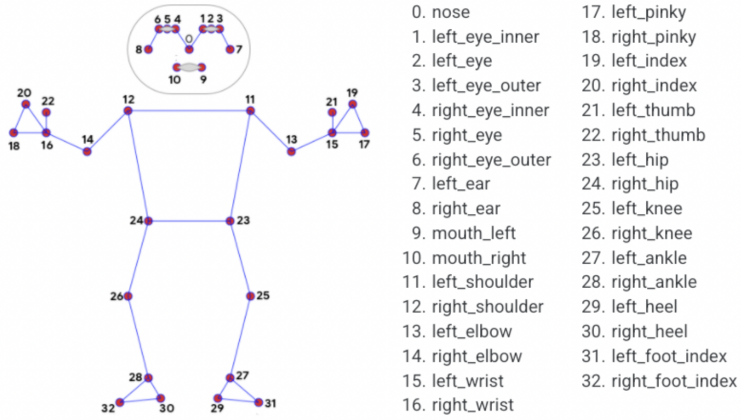

Figure 1.2: Body keypoints detected by Google MediaPipe [22]

We decided to use keypoint detection via Google MediaPipe as passing in the coordinates of the keypoints rather than the whole image enabled the model to be more lightweight. This allowed the model to have a lower latency, which was integral to creating a functional assistant application that could execute tasks in real time. The above figure displays the various keypoints detected by Google MediaPipe. We used these keypoints on both hands,

as well as several keypoints from the pose detection such as nose, ears, and shoulders to give spacial context relative to the hands to create 108 keypoints per frame that was passed in to our eventual model.

For the classification part of the model, we decided to use a TensorFlow based convolutional neural network (CNN). Every frame of the video adds a set of 108 keypoints to the data stack, and once a gesture is completed all the frames and keypoints from that gesture are coalesced together to create a 2d tensor (keypoints vs. time) to pass into the model. The model ultimately creates a probability distribution on the likelihood of a gesture being a specific American Sign Language (ASL) gesture from our dataset and returns the most probable one.

```
{"up": 0, "weather": 1, "loud": 2, "hello": 3, "dark": 4,
"cancel": 5, "search": 6, "sound": 7, "network": 8,
"eight": 9, "end": 10, "one": 11, "top": 12, "quiet": 13,
"start": 14, "five": 15, "bottom": 16, "open": 17,
"right": 18, "two": 19, "six": 20, "bright": 21, "down":
22, "left": 23, "keyboard": 24, "seven": 25, "enter": 26,
"ten": 27, "close": 28, "aim": 29, "nine": 30, "click":
31, "add": 32, "key": 33, "message": 34, "text": 35,
"remove": 36, "four": 37, "three": 38, "calculator": 39}
```

Figure 1.3: Custom Curated American Sign Language Gestures

For the input space, we decided to use a custom curated subset of relevant American Sign Language (ASL) gestures. To allow our application to handle a higher variety of gestures, we wanted to use gestures that had motion involved instead of the standard static hand gestures. This made the application more robust as there could be a higher variety and richness of gestures utilized to be able to be mapped into various actions on the computer. However, this also added complexity as video classification is less fleshed out in the industry than static image recognition. We decided to pick the Word Level American Sign Language (WLASL) dataset specifically since there was a wide variety of public datasets on the gestures already available across variety of hand types and backgrounds resulting in a well balanced dataset [29]. Given enough data, our model could also handle the addition of custom made gestures as well.

1.3 Related Work

There has been various other solutions researched to handle gesture based classification. For example, there has been research from Stanford that leveraged CNN's to classify ASL

alphabet gestures[12]. This approach used a YOLO approach and passed in a 500x500 pixel image into a CNN that would get classified into one of the 24 letters in the English alphabet (j and z were removed since those signs included motion and the rest were static). The top-1 results of the various models were in the 60%-70% range and the top-5 accuracy was roughly 90%. This approach was limited for our specific project scope since it didn't handle video classification with signs that had motion.

Towards creating an actionable, usable gesture controllers there have been approaches such as the chordal controller [4]. The chorded keyboard allows users to input keys with just one hand, by allowing various combination of keys to represent various keys on the keyboard. This allows a one hand to cover every letter to make it a functional device. By leveraging gestures rather than individual letters, we were able to produce more robustness than the chorded keyboard since the number of gestures that can be done far outnumber the limited combinations that can be made on the keyboard.

On the image processing side, other work has been done in filtering and augmenting the image to improve hand detection. There has been substantial work from the East China University of Science and Technology on utilizing background subtraction, RGB color segmentation, and image masks to detect the palm and hand [7]. This solution uses the color of the hand pigmentation and separates colors from the backgorund that don't match the pigmentation color to extract the hand from the image. This solution works is lightweight and allows for fast image processing, but does run into some noise and accuracy issues when hand pigmentation colors are similar to the background. Additionally, this lends itself to a YOLO approach where the extracted hand pixels would be passed into the model, which didn't work for our proposed approach as using pixels in the neural network was limited in handling temporal context.

On the neural network end, there has been different explorations into various types of neural network architectures that can handle gesture classification, while leveraging temporal context. Our architecture was developed off of the work from Australian National University where they compared 2d CNNs (VGG16), 3d CNNs (I3D) [20]. The results of the 3d CNN far exceeded the 2d model since it was able to use all the frames in succession to provide the model with the data over time and as a result we decided to focus on 3d neural networks that leveraged temporal continuity for our approach as well. The work out of the Australian National University passed in video clips as the input into the neural networks, which we found was too slow for real-time evaluation and so we decided to focus on keypoint based solutions to improve the latency.

Method	WLASL100			WLASL300		
	top-1	top-5	top-10	top-1	top-5	top-10
VGG-GRU	25.97	55.04	63.95	19.31	46.56	61.08
I3D	**65.89**	**84.11**	**89.92**	**56.14**	**79.94**	**86.98**

Figure 1.4: Gesture Classification Results from Australian National University

Chapter 2

Architecture

2.1 System Objectives

The goal of this project is to build a functional video gesture classification that can detect American Sign Language gestures in real time. The prototype was built in collaboration with others in the lab, specifically: Tucker Cullen, Weiyu Feng, Hanqi Zhang, and Kexin Guo [8]. To that end, key objectives for the system are as listed:

- **Accuracy** of the Computer Shortcuts Executed by the Assistant
- **Latency** between Gestured Sign and Computer Shortcut Executed
- **Robustness** of Model in Processing Different Signs and Backgrounds
- **Scalability** in Handling Additional Gestures and Computer Shortcuts
- **Feasibility** of Obtaining and Processing Training Data to Create a Functional Model

Accuracy

In order to make an effective assistant that can aid the user in navigating the computer, accuracy in an important component. The goal of the assistant is to make sure that whenever a user wants to gesture for a shortcut, that correct shortcut is executed. A large part of this is improving the model classification accuracy in understanding the gestures performed. We experimented with different architectures to improve the top-1 accuracy of the model towards industry standards. Beyond that, we worked on testing different parameter settings of the assistant, and the filtered mapping of gestures to actions to improve the ultimate actions accuracy.

Latency

Beyond accurately executing the user's shortcuts, the model also needs to be functional enough to work in real-time so that command execution can happen immediately after a gesture is signed. To do this we focused on making improvements to the various components of the model such as the webcam interface, the image processing, and the neural network to make it as light as possible. This was done in trade-off against the accuracy loss suffered by making the system less complex.

Robustness

The assistant needs to be able to handle a variety of different gestures, with different hand structures, backgrounds, lighting and rotations. In order to make sure it avoided the background noise, we decided to go with a keypoint based hand and pose detection. This kept the model focused on the hand orientations and the motion rather than confounding variables in the data. It also made the model more lightweight as it only had to process keypoints into the neural network rather than full video streams. Keypoint detection was done through MediaPipe which added some delay onto the image processing side, but we mitigated that through sampling at a subset of frames and only on motion detection.

Scalability

We wanted to build the system in a scalable way so that users across different operating systems and webcams would be able to leverage the assistant. Additionally we wanted to make it easy to add in new shortcuts for users to add beyond the initial set we created for the prototype. We made sure to build the different components to pass information asynchronously so all parts would work in parallel with each other. Lastly, we wanted to make sure the image processing happened in an efficient manner so the stream of videos and data wouldn't slow the assistant down.

Feasibility

When building the model, we wanted to ensure that we would be able to get effective datasets that could be trained in a reasonable time frame and produce meaningful results. We also wanted to make sure that the dataset and gestures chosen were standard for the users to sign. After reviewing various dataset options and published literature, we decided to use the Word Level American Sign Language (WLASL) dataset. American Sign Language is the predominant sign language used in North America with over 1 million people using it [28]. This made it a good candidate for a gesture set that would be easy for users to know and use for the assistant. Additionally, there were lots of public datasets for ASL, namely WLASL 100, 300, 1000 with many videos of the different gestures that set up a good dataset for the model.

2.2 System Design

The prototype was developed to be split into 4 modular parts (Webcam, ImageProcessing, ML Model, and System Controller), that worked functioned independently as shown below. Each module sends its information asynchronously to the next so that work can be done in parallel without bottlenecking the assistant. The webcam captures the video stream in real time and sends it to the MediaPipe API to extract the keypoints. These keypoints are sent to the ML Model which outputs a probability distribution across the various gestures. This probability distribution is passed into the system controller to execute the relevant action.

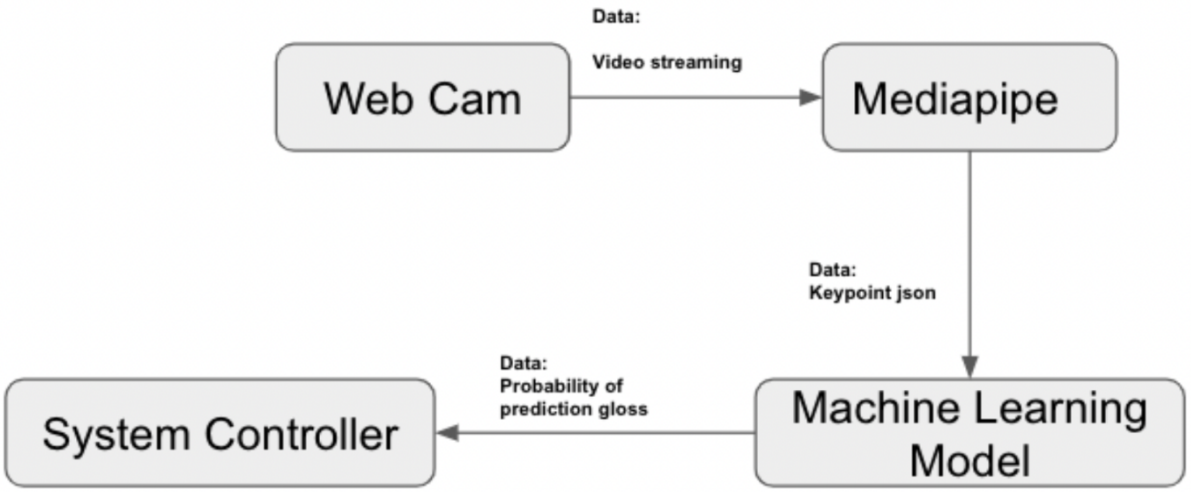

Figure 2.1: Prototype System Architecture

2.3 Webcam Interface

For the webcam component of the system we wanted to make sure it could handle streaming the video to the rest of the system in real time. We tried various methods such as sampling a subset of frames, inserting a start/stop sign, and trying to detect motion so that it could run more intelligently and not just every frame. The webcam was housed in an Electron Javascript application to run as an independent application on the operating system [9]. It serves as the user interface for the application, displaying the results and functions of the application. It consistently collects video stream from the user and opens up an API to fetch the processed video as needed by the application. Below is an image of our prototype webcam interface that shows the application window, the webcam stream, and a view of the keypoints drawn for the user.

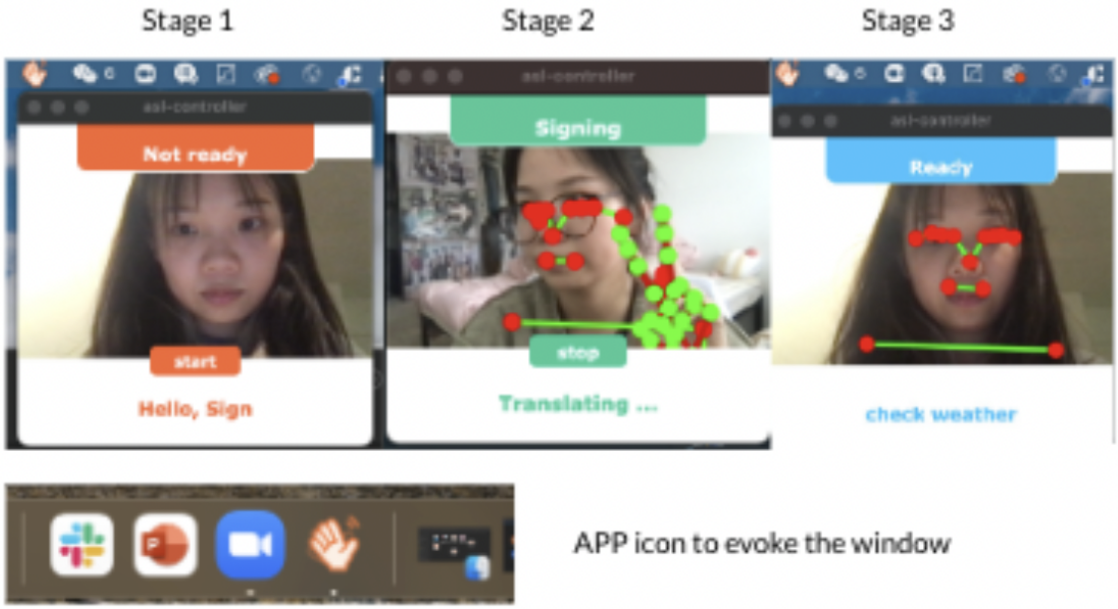

Figure 2.2: Webcam Interface. Stage 1 is the resting start screen, Stage 2 is the display during gesture motion, and Stage 3 shows the display once a command has been identified.

Failed Approach

Repeatedly sending the full video every frame to the application was fairly demanding on the application. As such we tried to figure out ways to reduce that bandwidth. The first approach was to lower the data size of the video being passed in by trying to downscale the image, grayscale the image, and crop out the relevant parts of the image. Changing the video type to make it less data intensive did help the latency of the application, however it also hurt the accuracy of the model as our frames now had less resolution and more noise being passed in. As such, we didn't think that trade off justified this approach. We also tried to build in some internal logic to "detect motion" and only capture the frames in that time span instead of constantly looping through. Our plan was to design a hueristic to detect when there was significant motion in the frame, and then start capturing and saving video frames from then until motion was at rest again. We tried this through a multitude of ways:

1. Comparing pixel values between frames to calculate numeric distance change between 2 frames.

2. Leveraging background subtraction and color segmentation based on human skin pigmentation we tried to detect significant changes in the hand outline over 2 frames [30].

3. Using a subset of easy to process keypoints like center of palm to detect significant distance changed across the sum of those keypoints [24].

All of these approaches fell short in achieving our goal to improve latency of the model by reducing frames captured. Firstly, all of these approaches were fairly noisy, and would sometimes incorrectly detect motion start or end without the user doing so, which would ruin the current gesture being performed. Additionally, even though the number of frames being passed in did reduce as a result of these efforts, all of these approaches add layers of post video processing in detecting the pixel values and the changes on a per frame level that ended up actually increasing the end to end latency of the application.

Successful Approach

We ended up going with a combination of 2 simple solutions that helped us achieve our goal of reducing frames captured by the webcam interface. Firstly, we implemented n-frame sampling that processed 1 out of every n frames to be passed into the model. After brief testing of different hyperparameter values for n, we found that 5 was a good middle ground between helping eliminate frames, but also maintaining enough info and context to create meaningful predictions. We also added a start and stop button to the interface so that users could self identify when a gesture started and ended so only the frames in between would be captured. This is similar in nature to how Siri can be enabled on the Mac via the click of the shortcut button on the magic bar, or through the microphone activation of "Hey Siri" [13]. Both of these configurations are available on the application and toggleable based on the user preference.

2.4 Hand Recognition

From the webcam interface, we passed in the video stream to our image processing component to convert the video frames into data usable for the model. The neural network architecture we built took in a Pandas Dataframe and converted that into a TensorFlow tensor. Our approach in the hand recognition was to to find a representation of the hand every frame on a streaming basis, with each new frame being processed and appended to the current Pandas DataFrame to be flushed into the model upon gesture completion.

Failed Approach

For hand tracking there are several different options on how to capture the hand details for the model. One previous implementation we tried was a You Only Look Once (YOLO) hand tracking model [10]. In previous years of the lab, we had built a hand tracking model that worked via a YOLO model that tracked the palm location and drew the hand contour behind it via color segmentation and background subtraction. The benefit of YOLO models

is that on the hand-tracking side, it operates at a lower latency as the image doesn't need to be preprocessed to find the hand or keypoints, and instead the entirety of the image is passed in. However, on the model side there was far too much variety being passed in with the whole image and it significantly decreased the accuracy of the model. Different colored backgrounds, clothes, skin colors etc. would all be detected as different by the model. This would require significantly more training data to offset this noise, which was not feasible to obtain or train the model on. Additionally, we noticed that YOLO models could not handle rotations of gestures well and so if gestures were performed at an angle to the camera, accuracy dropped as well. As a result we wanted to use something more lightweight that could be more in tune with gesture rotations and less biased by background noise.

Successful Approach

To achieve a more lightweight hand recognition image processing approach, we decided to use a keypoint detection solution. We decided to use Google MediaPipe Holistic pipeline for this as it was one of the most streamlined keypoint detection algorithms for 3d poses released [1].

Figure 2.3: MediaPipe demo showing 3d hand pose keypoint recognition

For American Sign Language gestures there a 5 parameters that matter: handshape, palm orientation, movement, location, and expression/non-manual signals [26]. Handshape and

palm orientation can be detected and modeled using MediaPipe's hand keypoint detection. Movement is tracked by seeing the changing values over several frames and combining them into a 3d tensor. Location is important since some ASL gestures mean different things if they are done in front of the face, to the side of the body, or in front of the chest. Additionally, gestures can also mean different things based on the facial expression used during the signing. As such, we need to also track the face and the body in our keypoint detection, which we can also do with the MediaPipe pipeline.

```
[
    'thumbCMC', 'thumbIP', 'thumbMP', 'thumbTip',
    'indexDIP', 'indexMCP', 'indexPIP', 'indexTip',
    'middleDIP', 'middleMCP', 'middlePIP', 'middleTip',
    'ringDIP', 'ringMCP', 'ringPIP', 'ringTip',
    'littleDIP', 'littleMCP', 'littlePIP', 'littleTip',
    'wrist',
    'leftEar', 'leftEye', 'nose', 'rightEar',  'rightEye',
    'leftElbow', 'leftShoulder', 'leftWrist', 'neck', 'rightElbow', 'rightShoulder', 'rightWrist'
]
```

Figure 2.4: List of the 33 different keypoint categories used in the model

Using MediaPipe we extracted 33 different keypoint categories for the model to use. 21 different keypoint categories helped map the hand, where we wanted many keypoints to track more precise movements. The other 12 around the body and face served as anchors to tie location and expression through. The figure below shows our model extracting the 21 hand keypoints from the live webcam video stream.

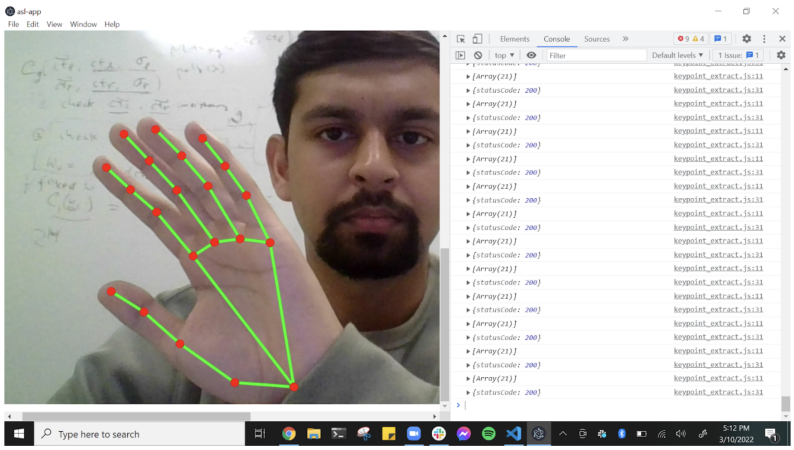

Figure 2.5: 21 keypoints detected across the hand

In total we tracked these 21 keypoints across both hands with both they X and Y coordinates as seperate values. We similarly took the X,Y coordinates of the left and right of the

body and face categories to get a total of 108 keypoints (plus the label for training) passed into the Pandas DataFrame for the eventual neural network tensor.

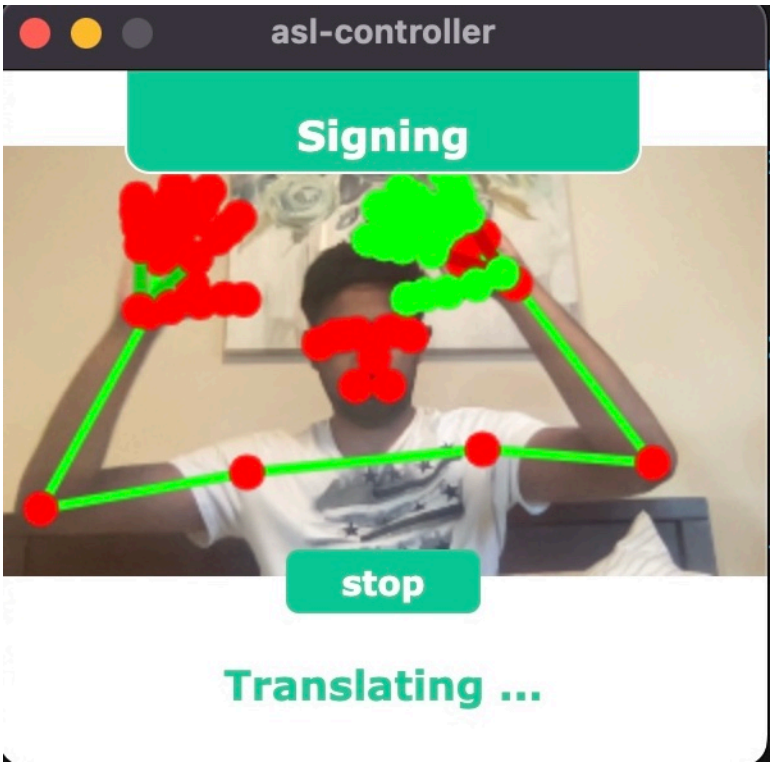

Figure 2.6: All keypoints detected across the body

2.5 Neural Network Integration

The neural network decisions and results will be discussed in full technical detail in the upcoming chapter, Models. At a high level, we wanted to design a neural network architecture that would be light weight enough to support real-time classification, and could also leverage the temporal context to understand the gesture motion over time. Additionally, the model had to be scalable and flexible in being able to accept in different sized tensors (108 keypoints * number of frames in gesture). We eventually use a TensorFlow nn.Linear class at the end of the model to get a probability distribution across all our gesture classes that is passed into our logic controller to convert into the relevant action. For our dataset we decided to use the public WLASL dataset. They had labeled videos for a set of 100, 300, 1000, and 2000 words that we passed through MediaPipe to get the keypoints from and passed those into the model. In the end we decided to make a custom curated set of 100 words from the WLASL 2000 that would be used for our model.

2.6 Operating System Control Logic

The goal of operating system is to take in the oncoming stream of gesture probabilities from the model and execute the current action accordingly. We set up the logic controller with a mapping of gestures to actions that it can call, and have a seperate computer controls file of the actual code for those actions that the logic handler calls upon. This lets users add their own actions or use the flow for different operating systems since only the computer controls file would have to change. If no gesture is detected (the probabilities across the highest gestures are too low) then no action is performed. As this was only a prototype we didn't build out functions for all the words in our dataset, but selected roughly a dozen that could showcase the demo effectively. This would tie the system together and make it possible for the application to detect a gesture and execute a computer command from end to end.

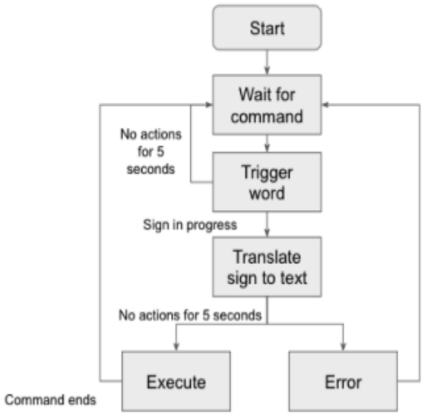

Figure 2.7: End to End Controller Logic

Failed Approach

Initially, we wanted to build a generalized logic handler that could take any stream of gestures and act accordingly. For example, if the first probability distribution displayed "volume", then it would wait for the next gesture model classification which might be "up" and then increase the volume. We did this by creating input states and sliders for the various actions, so "volume" would necessitate an up, down, or a value after it, "open" would require an application name after, etc. Although this was useful in creating a smarter more generalized system it made the logic too complex and hard coded to the specific actions. As such, we didn't think this approach was scalable enough to handle adding new actions or making adjustments and the complexity wasn't worth the benefit.

We also tried implementing majority based systems through successive runs of the model. The way neural network voting systems work is that they use the majority result of multiple

classifiers to increase the overall accuracy of the system [23]. For example, if you have 3 independent classifiers that all have a 60% accuracy - by running all 3 successively you only need 2 of the models to predict the right one to get the correct result.

$$\text{P(at least 2 models correct)} = \text{P(2 models correct)} + \text{P(3 models correct)}$$

$$\text{P(at least 2 models correct)} = \binom{3}{2} 0.6 * 0.6 * 0.4 + \binom{3}{3} 0.6 * 0.6 * 0.6$$

$$\text{P(at least 2 models correct)} = 0.432 + 0.216 = 0.648$$

Hence, you increase the overall accuracy by combining the results in majority with each other. Although this approach does seem promising, and does warrant future consideration the first issue we ran into is that we didn't have independent runs of the model to use. We tried using successive frames, for example if a gesture was 90 frames long we tried passing in frames 0-70, 10-80, and 20-90 into the model and took the majority against those 3. However, since there were so many frames in common the successive runs very similar to eachother and didn't contribute anything to the majority. Splitting the gesture into 3 separate runs, 0-30, 30-60, 60-90 also didn't work since the individual segments became too short to carry meaningful context. Given these constraints and the fact that the successive runs were adding more latency to the system we decided to move away from the majority based approach.

Successful Approach

For our final prototype we decided to keep the logic handler simple by making gestures have 1-to-1 mappings with specific actions. For example, as a sample command we mapped the "bird" gesture to opening Twitter or "weather" searching for the current weather on Google. For the most part, we would use the highest probability from the distribution (top-1) and call the action accordingly. As the system was designed to be asynchronous, we kept a running stream of the probability distributions and the action mappings. This allowed us to make it possible for multi gesture actions to be created in future development. Additionally, we set up protocols to check for consecutive results of the same action (ie. back to back "weathers") and can selectively choose to drop the second execution if it is not relevant.

We also set up a version of action "fuzzy matching" to make our system more reliable in selecting the right action. Fuzzy matching is the process by which 2 strings are compared to check for similarity to enable selection of options by using approximately matching strings instead of matching the option entirely [15]. For example the strings "Aple" or "Appl" can get fuzzy matched to "Apple". As many signs in ASL have similar characteristics, we tried to help the system pick the correct one by matching the gestures into actions that were relevant at that time. For example, the sign "four" or "now" has no meaning to the model when used without any other gestures around it, and so if the model is predicting one of those 2, it is likely a mistake and one of the subsequent results in the probability distribution are probably more relevant. And so through our fuzzy matching, our logic handler goes through

the top 5 predicted gestures in order to see if there is a meaningful action associated to them currently, and picks the first one that has one. If none of the top 5 have any meaningful actions, then it returns no action. This allows for our system to assign different actions to the same gesture based on the state of the system, and also makes it more reliable in picking a relevant action when detecting a gesture.

Chapter 3

Models

3.1 Overview

The goal of the neural network models was to create a lightweight model that could make classification inferences of the video streams of gestures in real time. We wanted to make sure our models could process frames in continuity to leverage the temporal context to be able to classify gestures in motion. Industry research for static gestures has had good results, but for dynamic gestures there is still improvements that need to be made, and so we decided to test several architectures to see how they integrated with our system.

3.2 Failed Approaches

LSTM

One of the first models we tried was an Recurrent Neural Network (RNN) based Long Short Term Memory (LSTM) model. A RNN model takes in the result of previous frame to make successive predictions [19].

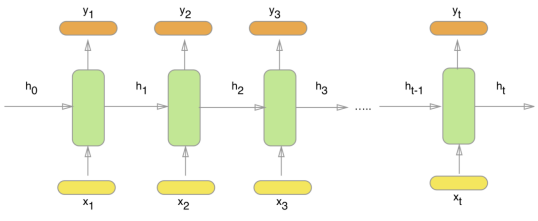

Figure 3.1: RNN Example Architecture

Our approach was to train a LSTM model based off the RNN structure so we could pass successive frames into the model and use that to make a more accurate prediction.

LSTM's are a specific type of recurrent network that uses memory cells within the network to keep previous results and layers towards future predictions [3]. We decided to build a LSTM model based off the work from Lee et al. and Joshi et al. [5] [14]. The work from Lee had promising results of over 90% accuracy across a series of 30 gestures (12 of them being dynamic and 18 being static). One key aspect of their approach is that they used an external Leap Motion Sensor that provided rich 3d poses from multiple different angles in real time. This is important for RNN models since they use previous frames for future predictions and so having more rich frames, and having more data available strengthens the RNN relationship and leverages it better. We found for our dataset, it was tough to get enough data to train these meaningful relationships since the same gesture might be signed in different ways or at differnet speeds with different signers, and so we would need a much larger dataset (or a custom created one) to get meaningful results which was not feasible at this time. Our results were comparable to the Joshi et al. paper of roughly 20-30% accuracy, which wasn't a strong enough for meaningful results for our application.

CNN Detector Network

One solution that we experimented with to make the model more lightweight was a CNN Detector Network. One issue with real-time classifiers for video streams is that running the classifier on multiple frames is taxing on the system and increases the latency. The CNN Detector Network architecture (shown below) addresses that by putting in an additional self-wrapping CNN layer that checks for meaningful changes to the input before rerunning the full classifier, reducing the number of frames classified [18]. This way our application could have a continuous stream of gestures being signed, and it would be able to use the network to detect changes in gesture signed and classify the new gesture accordingly.

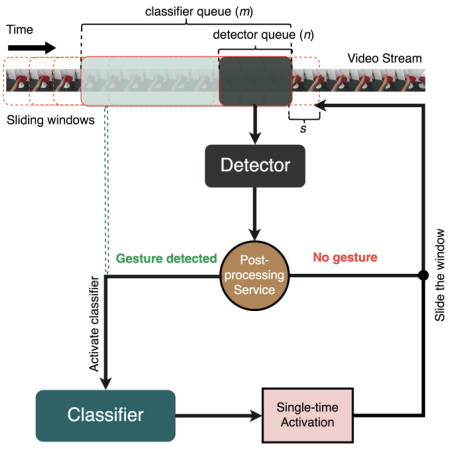

Figure 3.2: Detector Framework Architecture

We found this model tough to integrate effectively since there are several ASL commands that have 2 distinct movements to them in succession, for example raising the left hand up and then rotating the right hand after. Our detector network would evaluate this as 2 separate gestures instead of the original larger one. Additionally, it was tough to get effective training on the detector threshold since frame to frame there weren't much differences in the continuous video stream, and the model would need a larger time horizon and more comprehensive understanding on ASL to detect when a gesture starts and stops. In effect, it didn't perform much better than using a simple approach to see when the hands left the screen, or trying to detect when the hands stop motion and it added complexity and latency to the model so we moved away from this approach. That being said, there is still useful approaches in this space that would be necessary if future applications want to be able to run a continuous video stream instead of segmenting the frames upon gesture completion.

3.3 I3D

One of the successful models we used was the I3D model based off the work from Carreira and Zisserman [6]. I3D is a Two-Stream Inflated 3D ConvNet that is modeled off the previous 2D ImageNet ConvNet. The figure below shows the architecture of the I3D model.

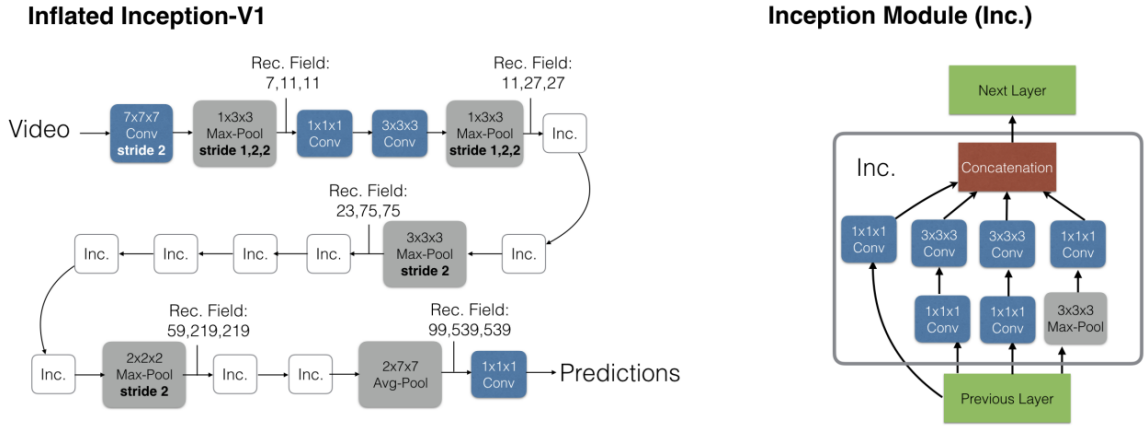

Figure 3.3: I3D Model Architecture

The recurrent part of the model occurs in the inception module which takes in the previous layer towards its current evaluation. Since the video stream is a series of ongoing frames each inception module has the one previous behind it and it can use all the recurrent modules together to detect motion and trends over a larger frame interval. The original model was trained on the Kinematics Human Action Video Dataset which has videos of people doing various tasks such as mowing a lawn or washing dishes which does require temporal

context [17]. One key difference between the initial implementation and our approach was that with an action like mowing a lawn, the action is consistent across the multiple frames. In contrast, with the ASL gestures they can have multiple components to them across the different frames, such as an action with the left hand followed by an action of the right arm, followed by a facial expression. As such, more previous frames are needed to provide a stronger temporal context, so we made that adjustment by adding in more recurrent layers to the I3D framework. We found that I3D had good accuracy (64.9% top-1 accuracy across the test set) but it had more latency than we were hoping (0.55 seconds inference speed in real time).

3.4 SPOTER

Ultimately the model we ended up using was the SPOTER model, the Sign Pose-Based Transformer Model [2]. This is a Tranformer Based neural network that takes the keypoints over frames and predicts a gesture accordingly. Below is a figure that displays the architecture of SPOTER as shown from the Boháček" paper.

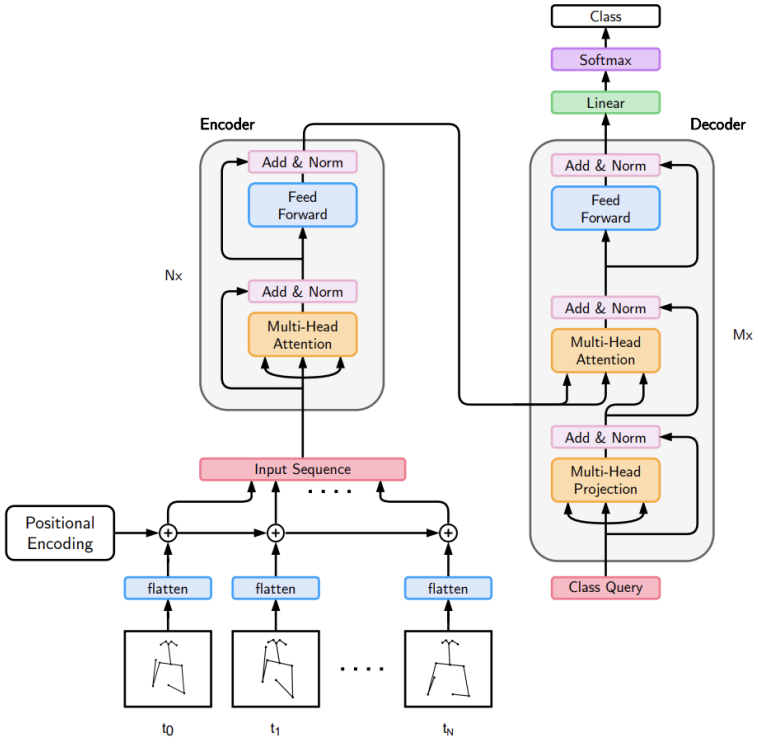

Figure 3.4: SPOTER Transformer Architecture

The transformer model adds positional encoding vectors to the keypoint data in order to encode the temporal order for the model to eventually read. From there it leverages the standard transformer structure of the encoder-decoder modules. The encoder module has a multiheaded attention layer that can look at the keypoint data across the different frames and leverage it in its calculations, weighting it by "similarity" which is computed by the dot product across the positions. All the attention layers are combined together with their inferences into a layer normalization, which is passed into a feed forward and another layer normalization. This allows the value of this overall module to leverage all the positions with each other, and especially prioritize temporal continuity by weighting more temporally close frames with each other. The decoder module is similarly built. The bottom figure shows how the encoder-decoder structure connects to eachother to output the probability distributions.

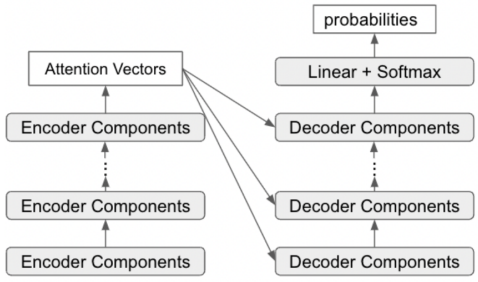

Figure 3.5: SPOTER Transformer Encoder Decoder Pipeline

Encoder Lay.	Decoder Lay.	heads	hidden dim.	feed-forward dim.	input dim.
6	6	9	108	2048	108

Figure 3.6: SPOTER Model Parameters

Overall, the model had 6 encoder layers, 6 decoder layers, 9 heads in the multihead attention layer. The neural network had an input of 108 for the 108 keypoints we extracted earlier from MediaPipe and a hidden dimension of 108, and a fast-foward dimension of 2048. When evaluating our model, we saw that the SPOTER Transformer model had slightly lower accuracy (59.8% vs 64.9% on the test set but the average inference time was much better (0.05 seconds vs 0.55 seconds. This is because SPOTER was much more lightweight in terms of parameters and structure than the I3D model. Because we valued real time inference for our application, we decided to use the SPOTER model for our neural network.

Originally when we were building our model we tested on the WLASL 100 dataset. We also ran an evaluation on the WLASL300 dataset and found the accuracy was a little lower (52.3%) which makes sense because there are more classes. For our final prototype, we trained

Model	I3D	Transformer
Number of parameters	12.4 million	5.92 million
Average inference time	0.55s	0.05s
Top-1 accuracy	64.9%	59.8%

Figure 3.7: Comparison of I3D and SPOTER statistics

custom datasets of words we thought were the most relevant for potential shortcut actions. We made a WLASL40_custom and WLASL100_custom accordingly. The WLASL40_custom actually had pretty poor accuracy since we didn't have enough training data across all the 40 classes so the model had heavy overfitting (shown in figure 3.9 training curves), but that should be fixable if we make our own training data in the future. We ended up using the WLASL100_custom which had a 54.3% test accuracy.

	WLASL100	WLASL300	WLASL40_custom	WLASL100_custom
Top-1 accuracy	59.8%	52.3%	39.7%	54.3%

Figure 3.8: SPOTER Top-1 Accuracy across WLASL Datasets

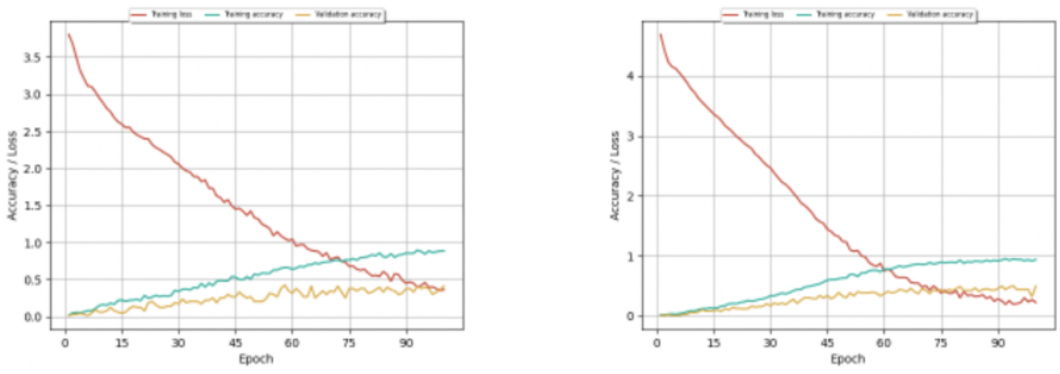

Figure 3.9: Training Curves for WLASL_40_custom (left) and WLASL_100_custom (right)

Chapter 4

Discussion

4.1 Current Capabilities and Limitations

At present, our prototype works end to end in capturing the video, passing it to the model, classifying the gesture, and performing the relevant action. There are currently about a dozen different actions setup within the logic controller such as "volume up", "check weather", "click", and "open browser".

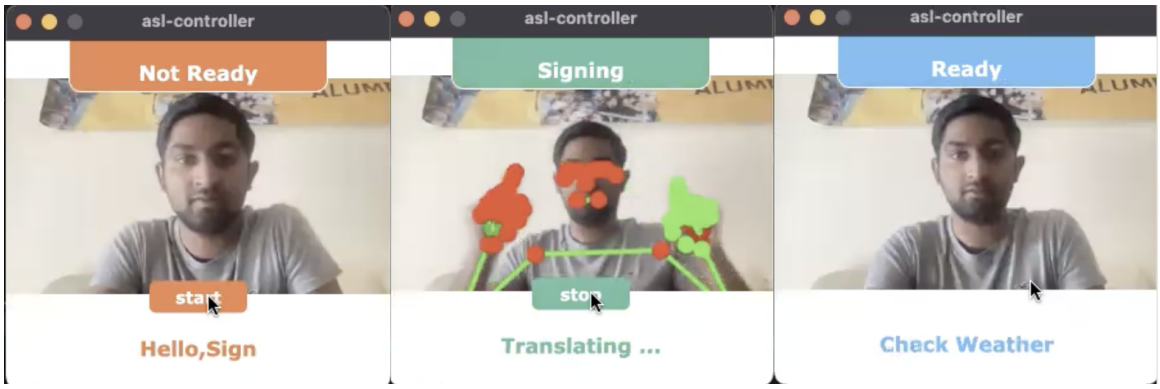

Figure 4.1: End to End example of "Check Weather"

The accuracy of the actual gestures in the application have a slightly lower accuracy than the posted test accuracy above. This is because there is some amount of over fitting in the data so certain signs get over represented by the classifier, especially when they have similar motions to the actual signs. There is also some level of noise with the key point detection since the angle of video being captured from the webcam is pretty narrow and up close, and so the 21 keypoints in the hand are all close to each other, leading to some bouncing and overlap. In terms of the actions inside the logic handler, they are pretty one-dimensional

and simple right now like "volume up" but a more robust controller could be built to handle more complex actions.

4.2 Future Work

In the future, work could be done to make the results more accurate, and make the application more functional overall. In terms of accuracy, first thing that can be done is improve the neural network. This could be done through a model that has more context into previous frames and facial expressions since that is a pretty key part of American Sign Language. The trade off is making sure that the increased model complexity doesn't hamper the inference time. Additionally, future work can be done in getting different angles of video streams or higher quality frames that have more keypoints so that the model can have more context towards the motioned gesture. Lastly, one of the main limiters we had currently was lack of training data in our dataset which resulted in overfitting. To remedy this we built a module into the application to record gesture video streams in real time through the ASL Controller and save that into training data, so that future users or developers can create their own training data to make more rich datasets. In terms of functionality, by allowing users to make their own training sets we can add more custom signs and diverse signs to encompass a greater variety of actions. Furthermore, since we built the logic handler with scalability in mind, users can add their own shortcuts as they see fit. The logic handler can be improved to allow for more generality so it is a more fluid experience to use the ASL controller assistant. In the paper, we mentioned many of the approaches that we tried that didn't work to the intended effect. Some of those approaches like motion detection still have some promise for future development, and so revisiting some of those approaches with more robust solutions could yield promising results in the future. Additionally, we could look into integrating other input parameters into the assistant such as eye tracking or voice recognition to make it more robust and higher fidelity in terms of user experience.

4.3 Conclusion

In conclusion, this paper summarized the results of our efforts to build an end to end video classifer for ASL gestures as an assistive technology tool for users with potential impairments. To accomplish this we split the development into 4 parts: Webcam, ImageProcessing, ML Model, and System Controller that all worked asynchronously to pass information to one another. As a relatively novel solution to tihs problem, we experimented with many different industry standard approaches and detailed our results and rationales for moving forward with the subset we did. We did a heavy exploration into various neural network architectures to build a lightweight model that could make accurate predictions in real time, leveraging the temporal context within the WLASL dataset. We ultimately decided to use the SPOTER model with pretty reasonable accuracy and built our functioning prototype that works end to

end from the webcam to the inference to the executed shortcut. In aggregate this prototype serves as an exploration into building more assistive technology and will hopefully lay the groundwork out for future development in the space.

Bibliography

[1] *3D Hand Pose with MediaPipe and TensorFlow.js*. URL: https://blog.tensorflow.org/2021/11/3D-handpose.html.

[2] Matyás Boháček and Marek Hrúz. *Sign Pose-based Transformer for Word-level Sign Language Recognition*. URL: https://openaccess.thecvf.com/content/WACV2022W/HADCV/papers/Bohacek_Sign_Pose-Based_Transformer_for_Word-Level_Sign_Language_Recognition_WACVW_2022_paper.pdf.

[3] Jason Brownlee. *A Gentle Introduction to Long Short-Term Memory Networks by the Experts*. URL: https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/.

[4] Bill Buxton. *Chord Keyboards*. URL: https://www.billbuxton.com/input06.ChordKeyboards.pdf.

[5] C.K.M.Lee et al. "American sign language recognition and training method with recurrent neural network". In: *Expert Systems with Applications* 167 (). DOI: https://doi.org/10.1016/j.eswa.2020.114403. URL: https://www.sciencedirect.com/science/article/pii/S0957417420310745.

[6] Joao Carreira and Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. URL: https://arxiv.org/pdf/1705.07750.pdf.

[7] Zhi-hua Chen et al. "Real-Time Hand Gesture Recognition Using Finger Segmentation". In: *Hindawi* 2014 (2014). DOI: https://doi.org/10.1155/2014/267872. URL: https://www.hindawi.com/journals/tswj/2014/267872/.

[8] Tucker Cullen et al. *A Voice Assistant for the Voiceless: Computer-Vision Based Sign Language Recognition*. M.Eng. Capstone Report, University of California, Berkeley, May 2022.

[9] *Electron*. URL: https://www.electronjs.org/.

[10] Juan A. Figueroa, Heidy Sierra, and Emmanuel Arzuaga. *Real-Time Hand Detection with the use of YOLOv3 for ASL recognition*. URL: https://indico.cern.ch/event/809812/contributions/3391220/attachments/1834047/3004274/Real-Time_Hand_Detection_with_the_use_of_YOLOv3_for_ASL_recognition.pdf.

[11] Thippa Reddy Gadekallu et al. "Hand gesture classification using a novel CNN-crow search algorithm". In: (). DOI: https://doi.org/10.1007/s40747-021-00324-x. URL: https://link.springer.com/article/10.1007/s40747-021-00324-x.

[12] Brandon Garcia and Sigberto Alarcon Viesca. *Real-time American Sign Language Recognition with Convolutional Neural Networks*. URL: http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf.

[13] *How to Use Siri on the Mac: macOS Monterey/Big Sur/Catalina, Mojave, Sierra – MacBook Air/Pro/iMac*. URL: https://www.howtoisolve.com/use-siri-on-macos-sierra/.

[14] Nihar Joshi et al. *American Sign Language Recognition Using Computer Vision*. URL: https://web.eecs.umich.edu/~justincj/teaching/eecs442/projects/WI2021/pdfs/027.pdf.

[15] Krishna Prakash Kalyanathaya, Akila D., and Suseendran G. *A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP*. URL: https://www.researchgate.net/publication/333249900_A_Fuzzy_Approach_to_Approximate_String_Matching_for_Text_Retrieval_in_NLP.

[16] I. Kanellopoulos and G. G. Wilkinson. "Strategies and best practice for neural network image classification". In: *International Journal of Remote Sensing* (1997). DOI: https://doi.org/10.1080/014311697218719. URL: https://www.tandfonline.com/doi/abs/10.1080/014311697218719.

[17] Will Kay et al. *The Kinetics Human Action Video Dataset*. URL: https://arxiv.org/pdf/1705.06950.pdf.

[18] Okan Köpüklü et al. *Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks*. URL: https://arxiv.org/pdf/1901.10323.pdf.

[19] Simeon Kostadinov. *How Recurrent Neural Networks work*. URL: https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7.

[20] Dongxu Li et al. *Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison*. DOI: https://doi.org/10.48550/arXiv.1910.11006. URL: https://arxiv.org/abs/1910.11006.

[21] *MediaPipe Hands*. URL: https://google.github.io/mediapipe/solutions/hands.html.

[22] *MediaPipe Pose*. URL: https://google.github.io/mediapipe/solutions/pose.html.

[23] Nabil Moukafih, Ghizlane Orhanou, and Said El Hajji. "Neural Network-Based Voting System with High Capacity and Low Computation for Intrusion Detection in SIEM/IDS Systems". In: *Hindawi* 2020 (). DOI: https://doi.org/10.1155/2020/3512737. URL: https://www.hindawi.com/journals/scn/2020/3512737/.

[24] José Manuel Palacious et al. *Human-Computer Interaction Based on Hand Gestures Using RGB-D Sensors*. URL: https://www.mdpi.com/1424-8220/13/9/11842.

[25] Ashish Sharma et al. "Hand Gesture Recognition using Image Processing and Feature Extraction Techniques". In: *Procedia Computer Science* (). DOI: https://doi.org/10.1016/j.procs.2020.06.022. URL: https://www.sciencedirect.com/science/article/pii/S187705092031526X.

[26] *The 5 Parameters of ASL*. URL: https://www.mtsac.edu/llc/passportrewards/languagepartners/5ParametersofASL.pdf.

[27] Thiago R. Trigo and Sergio Roberto M. Pellegrino. "An Analysis of Features for Hand-Gesture Classification". In: (). URL: http://www.ic.uff.br/iwssip2010/Proceedings/nav/papers/paper_128.pdf.

[28] Sophia Waterfield. *ASL Day 2019: Everything You Need To Know About American Sign Language*. URL: https://www.newsweek.com/asl-day-2019-american-sign-language-1394695.

[29] *WLASL*. URL: https://dxli94.github.io/WLASL/.

[30] Hui-Shyong Yuo, Byung-Gook Lee, and Hyotaek Lim. *Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware*. URL: https://doi.org/10.1007/s11042-013-1501-1.