Exploring Light-Weight Solutions for Real-Time Sign Language Interpretation

Patrick Fuller, Nathaniel Woodward

Abstract

There is little research available in the field of Sign Language Interpretation through the use of Artificial Intelligence. Though many companies and organizations have shown interest in this topic, as the benefits for hard-of-hearing and deaf employees are obvious, advances in research have been few and far between. We seek to unify the data that is available on this topic and to compare and contrast the methods used by different researchers and their effectiveness to develop our own approach. The goal is to balance and ideally improve baseline results on character recognition while maintaining a light-weight nature that would allow this research to be viable in real-time applications on portable and mobile devices.

I. Introduction

Communication is vital in every-day life, and the workplace is no exception. For those that are hard of hearing or deaf, communication with co-workers can become difficult without cooperation from other individuals and a plethora of tools. Even a change in workplace arrangements can become more difficult than otherwise would. Individual steps by employers as well as legislation included in the Americans with Disabilities Act (ADA) has sought to minimize some of these hardships and provide an equal opportunity for all individuals. Providing tools to make communication and work efficient for these employees is a worthy investment, however, not always foolproof. Some traditional solutions, like an American Sign Language (ASL) interpreter, are still helpful, but can become costly for the business and still is only as available and reliable as the individual providing interpretation. This situation is the inspiration for our work, as we look to synthesize the research in the space of A.I. regarding sign language translation, and provide our own solution. Although this topic seems as if it would garner a lot of attention from the computer science community, there is very little research that is publicly available. That being said, corporate giants such as Google and NVIDIA have shown interest in this topic and are currently developing their own applications [6]. Many students and professionals have shared their research, as well as their methodologies and results with the public to help build a healthy foundation for which future researchers can grow [2]. Some hobbyists see this as a fun computer vision problem to solve, while some researchers are in fact devoted to solving sign language translation. However, those that are looking to solve it directly are clearly working towards an application and have no published such research and advancements. Additionally, we found that any researcher or paper working at this scale and level of dedication were working towards gesture recognition of dynamic signs. While we will discuss the approaches used in this specific space later in the paper, our decision is to focus on recognition of individual characters.

The reason we decided to focus specifically on character recognition is that solutions to gesture recognition require a specific setup and level of hardware not available to both us, the authors developing the model and application, and to the users, who we would like to use our application in a mobile and fluid setting, not fixed in one location with the accuracy being dependent on several fixed metrics. Additionally, we see character recognition as secretly very powerful. Gesture recognition can be extremely powerful, but is limited to the vocabulary you train it on. There is the potential of up to 10,000 "signs". However, there are over 180,000 words in the English dictionary. How do these systems account for spelling words letter by letter (finger spelling), which accounts for 7-8% frequency of words signed?

Fingerspelling sequential words is not a common practice in fluent communication among deaf and hard-of-hearing individuals, but letters can be recognized and signed very quickly, and is a more realistic approach to be used as the base of an application to be used in a workplace.

II. Related Work

A. Optical Input

One of the largest hurdles facing Sign Language translation is how to recognize that a sign is being made. Some researchers have opted for a "Palm-Point" solution. This method involves recognizing key points in a signers hand taken from a stream of images and then comparing those points to a preprocessed image of a sign [3]. This method is similar to how facial recognition systems are developed. In fact, one research group used a facial recognition algorithm called You Only Look Once (YOLO) as a means to build their recognition network [5]. In 2018, the team at Coviu utilized this algorithm coupled with some training images that were taken from various YouTube videos and began training a convolutional neural network (CNN) that they built with PyTorch. The goal was to accurately translate the Australian Sign Language. This model gave an accuracy rating in real time of approximately 86%. This project is open sourced so others can build from it [5].

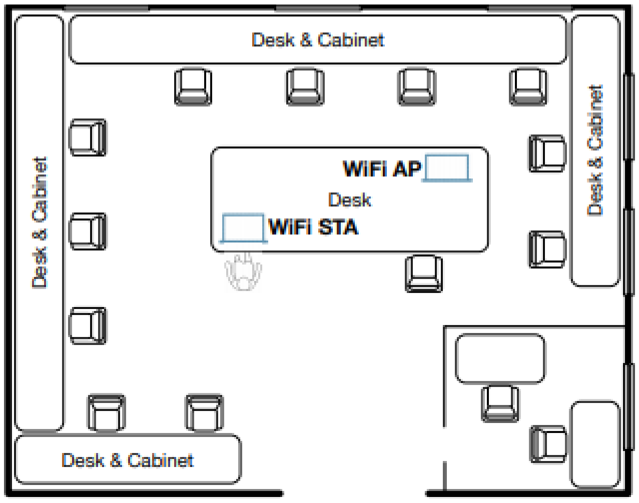
Google's AI team had developed an application in 2019 that utilizes a more technically complicated model, incorporating augmented reality to better predict signs when a signers hands are skewed in the eyes of the camera [6]. Their team had found that it is easier to identify a palm or fist rather than individual fingers. This also helped them to use a more lightweight algorithm, a non-maximum suppression algorithm to be exact, which freed up some resources that can be used in other aspects of the model. Utilizing this algorithm, as well as a single shot detector model called BlazePalm, the team was able to achieve an accuracy rating of around 96% in static images [6]. This number dropped drastically when given data in a real world scenario with dynamic recognition to an accuracy rating of 87%. This project has been open sourced.

A few researchers based in China have developed a method that utilizes Microsoft Kinect as an image input [1]. The Kinect SDK ships with a 3D trajectory model that can accurately track hand motions in real time. Given variables such as gesture speed and subtle differences in signing, the team implemented a linear resampling function that normalized the trajectories predicted by the model [1]. Once these trajectories have been normalized, the application then compares these inputs to a set of gallery images and computes the Euclidean distance as a recognition metric. After 5 iterations of training, this model was able to achieve an accuracy rating of ~96% [1].

B. Other Methods

Amazingly, there are many other ways to format input for translation. Although an optical input seems to be the most intuitive, a plethora of other techniques have been utilized to further the translation of sign language.

A team of researchers based out of the College of William and Mary have devised a way to accurately predict signs using Wi-Fi signals. This model takes Channel State Information (CSI) that is reflected from a Wi-Fi signal into a receiver. What this requires is both a router and a receiver dedicated only to this task, and cannot be one that is in use for any other task. The router and receiver must be placed on either side of the individual signing, and packets of trivial data are sent from one to the other. Shown below is (SignFi)'s floor plan in testing.



(a) Lab Environment

The disruptions in the Wi-Fi signal from the individual between the two devices are recorded in the CSI information and stored to create training data for specific signs. After building a set of training data on their own, they feed the input from the CSI's into a 9 layer CNN. In order to speed up the training model, batch normalization is used. Batch normalization is also used in conjunction with shuffling the training data to greatly reduce overfitting. The next step of this model is to send the data through a Rectified Linear Unit (ReLU) layer. The ReLU operates as a threshold to further reduce the noise of the original inputs. After going through an Average-pooling Layer, a Dropout Layer, a Fully-Connected Layer, and a Softmax Layer, the classification is able to begin. In a home environment, this model was able to achieve an accuracy rating of ~99%. This project is open sourced and available for all to use [4].

A hot topic for discussion in this field is the use of wearable technology as a way to circumvent the need for optical input. Though the use of this technology is not as easy to develop as the previously mentioned methods, the results that are produced from this are unparalleled. Some researchers have been able to achieve 100% accuracy through the use of wearable technology, a feat that has yet to be achieved by any other method [4].

C. *Future Methods*

While it is impossible to predict exactly what advances will be made in the coming years, there are some fairly logical conclusions that can be drawn from the current state of ASL translation. Many of the technologies and innovative methods of interpreting signs discussed in this paper are very new; simply put, they have a lot of room for improvement. However, the raw potential of some methods are obvious. The ground work for the near future seems to have been laid out for how sign language translation tools can be created with enough accuracy to be commercial. The biggest hurdle for the technology now seems to be having enough of a demand behind the technology for full development. There are many proof of concepts that demonstrate the capabilities of the various technologies, but all of them lack the data and hardware needed to scale to a meaningful size where it could be deployed in the workplace and other settings. Therefore, this section will primarily focus on the methods that have the biggest potential to be used in the future, as well as the details of what is needed to bring these technologies to a meaningful scale.

In the present state, methods involving optical input from some form of camera to classify an ASL character have lower accuracies to methods employing wearable technology. Thankfully, for settings with less technology and only a laptop camera at hand, there is still room for growth. This can be attributed a Google open-source technology called MediaPipe. Instead of many attempts which use the pixels of a static image as the input layer of a model, MediaPipe offers a concise alternative. Provided an image containing a hand, MediaPipe uses machine learning to plot 21 points on the image indicating key points on the hand including fingertips, knuckles, and the center of the wrist. What makes this technology so unique is that each point is technically three-dimensional. In addition to an x and a y value given to each key point on the hand, there is a third value indicating depth relative to other key points on the hand as well. This added dimension, in conjunction with recognition of which finger is the thumb, in theory allows enough information for a carefully trained model to detect an ASL character from any hand angle- front or back.

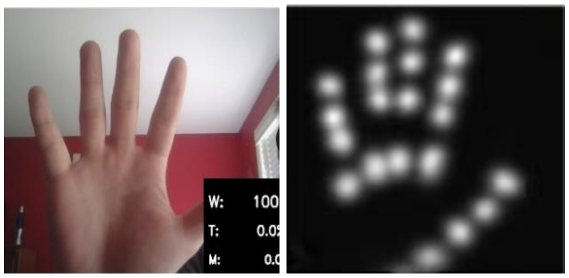
Moving forward, a model recognizing ASL characters from static images could be trained on location data of key points of the hand, including their depth. Where Google has used MediaPipe to demonstrate the ability for it to be used to recognize a few common gestures with high accuracy, massive amounts of data are needed to bring this to practice. Not to mention, if full gesture recognition for comprehensive sign language translation is the goal, other methods such as SignFi will be more effective in the future than using MediaPipe, which is not bound to the scope of a stationary camera with fixed view. Therefore, what we are suggesting is that MediaPipe has greater potential to be used specifically in ASL character translation specifically. It is a portable tool that can be used on a phone or on a laptop, and provides more concise and relevant input than a grid of pixels does without any of the noise. What we can possibly see is MediaPipe being used to develop, with enough data, the most accurate ASL character recognition model yet from an image as input. This will be useful as a lite translation tool, but we see more potential in gesture recognition elsewhere - in SignFi and wearables, though the latter may take longer.

III. METHODOLOGY

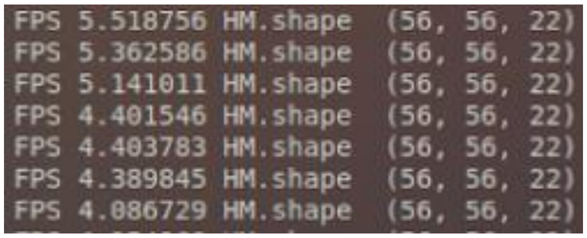
A. *Initial Decisions*

Originally, we had hoped to build off of Google's implementation of the MediaPipe framework and work it into our model. The idea was to use MediaPipe's hand Keypoint detection capabilities to build a library of keypoint references with which we can compare captures of our video stream to and make predictions. This proposed flow would fit nicely into our already built prediction system and greatly improve our accuracy rating. When diving into this improvement strategy, we realized that we had too main complications: lack of GPU resources which MediaPipe relied on, and time. The time constraint was the most pressing issue to be addressed, since with our deadline fast approaching we had limited time to learn and master the complicated setup process to get MediaPipe running on our system. It quickly became apparent to us that MediaPipe was not the right choice, as in addition to these complications, the MediaPipe application takes a video as input and outputs a video with the skeleton of the hand drawn over using the detected keypoints. This does not fit into our goal of using it in a real-time application, let alone get it to the point where we could implement it into our model. That being said, the time and effort we put into MediaPipe was not wasted, as it gave us a clear goal in how to improve our prediction accuracy: key points.

The keypoint strategy made the most sense the more we discussed it. Rather than feeding our model a stream of 64 x 64 x 3 pixels and comparing each pixel to our test data, we could use roughly 22 keypoints generated as our input layer, and inference with it from a new model that would be trained with keypoints estimated from our training data images. The process is shown below, with screenshots taken from our application.



Not only would this make our model more efficient in resource consumption, it would also become more lightweight and portable which fits nicely into this projects ultimate end goal of being used as a working program for the hearing-impaired professional. With all of that considered, our next logical step was to find a hand keypoint generator that was both easy to understand and manipulate and that would fit nicely into our pre-existing model. We found just that in a hand key point detector model published on GitHub by "erezposner" [site]. This program used native OpenCV functions to locate and draw hand keypoints. Our original program was written using OpenCV, so this was a promising candidate. Within a day we were able to fully implement this model into our program's workflow. Unfortunately, the original implementation used pose pairs to mark the keypoints on the hand, which is a very accurate method to mark points but also quite slow. The reason it was implemented this way was to draw the skeleton using the coordinates of each pair of points. Additionally, with this format, the order in which the pairs are arranged also indicate which coordinates corresponds to each finger, thumb wrist. This means the potential for higher accuracy is there. However, we found that with this model we were getting a framerate of roughly ~5fps, which is abysmally slow considering we are looking for a real world implementation.


```
FPS 5.518756 HM.shape  (56, 56, 22)
FPS 5.362586 HM.shape  (56, 56, 22)
FPS 5.141011 HM.shape  (56, 56, 22)
FPS 4.401546 HM.shape  (56, 56, 22)
FPS 4.403783 HM.shape  (56, 56, 22)
FPS 4.389845 HM.shape  (56, 56, 22)
FPS 4.086729 HM.shape  (56, 56, 22)
```

Our next candidate was the "Model Based Tracker" written by FORTH. This program utilized heat maps to gauge depth and posturing of the subjects hands in real time as well return the keypoints of the hand. This sounded like a perfect fit for us. This program used all of the libraries that we had already loaded into our original model and supplied us with a good estimator function and some other OpenCV hacks. What separated this model from the previous option is that the heat map of the hand was calculated first, and then calculated the key points from the heat map. Since calculating the exact location of key points is too slow to run in real time, the final approach we decided to try was use just the heat maps using the estimator function on our original training data to train the character classification data. The heatmap contains the shape of the hand, and should be distinct enough to be able to separate individual fingers. Considering this preprocessing step was the most robust option we had that would fit into our real-time application goal, we decided to give it a go.

B. *Model Training*

The first step to train our new model is to preprocess all of our training data for our new expected input format. To avoid a slow feedback loop and repeat the same lengthy preprocessing, we perform the operation once on all of our training data and save it back to the disk. Going forward, we can train using just this new data. The original source training data comes from kaggle.com, as the "ASL Alphabet" dataset. This dataset comes with 87,000 training examples, 3,000 of each classification. The 29 possible classes includes all 26 letters in the English alphabet, and three special characters to add additional functionality to our application; the special characters are space, delete, and nothing.

First, we separated the training data into 29 folders names by the class's name. Next, using the keypoint estimator, we transformed every image to an image of the heat map of the hand. To finally assemble the dataset to be used for training, at the top of our training script, we iterated through each of the folders named by each of the classes. Then, using the name of the folder to get the correct class label (0-28) from a list with the correct mapping, we appended to a pandas dataframe the pre-processed image as a numpy array in the "X" data structure, and the label number as the target in the "y" data structure.

To have baseline to compare our new results to, we use the same architecture for our convolutional neural network. The input size is fixed as a 64 x 64 image. We do a forward pass of each image through our 9 stack of convolutional layers. Each convolutional layer has a receptive field of 3 x 3, sub-sampling a small portion of the original image with the hopes that we can pick out edges and draw conclusions from the details yielded here. Paired with the convolutional layers we apply a parallel series of dropout layers. The point of dropout layers are to randomly select nodes in the network that will not provide information and weights set to 0, or "dropped out". We apply these layers in hopes that it helps our model generalize patterns and to prevent overfitting. Following the convolutions, the model contains a fully connected layer bringing the dimensionality of the layers slowly to a linear layer containing only the number of nodes as there are classes that we are to classify. Finally, once we have a single layer with 29 node values, we apply a final softmax activation function. Softmax takes the value in a node and divides it by the weighted sum of all of the other node's, or classes, values. This activation function works such that all of the resulting values in every node add up to one, with each value weighted accordingly relative to the initial values. In essence, we create a probability distribution of what class the model is predicting.

Our training objective is simply to minimize the cross-entropy loss across the probability distribution produced by the model for a given input, and, a one-hot vector with all 0's and a 1 in the index of the corresponding class to the input. The cross entropy loss can be thought of as the difference between the two vectors. The closer and more confident the model is in the correct class, the lower the loss. Ideally, the loss converges and stays at a low value and does not spike back up, but this can happen quite often. We also took a few special considerations implemented for our training. The first is simply saving checkpoints of the model, and choosing the one with the highest validation accuracy at the end of training. Secondly, to try to encourage progress even towards the end of training, we lower the learning rate more and more as a function of the derivative of the loss function over time. That way, if the loss function's rate of change decreases, so does the learning rate, preventing our model from flying over the theoretical minimum for how we designed the problem, and only increasing the loss.

Lastly, in line with our efforts to prevent overfitting and generalize our model as best as possible, we used the keras function fit_generator(). This function allows you to pass in a data augmentation function along with your data to apply before training. We defined a function that randomly rotates the image by a degree of anywhere between negative twenty and twenty degrees. While this may bring down accuracy technically on the training data, the model has a more general knowledge of the task and is less affected by things like angle of the hand, lighting in the room, skin color, or location of the hand relative to the camera.

IV. RESULTS

A. *Evaluation Results*

To evaluate the results of our model, we use the metric of exact match. What this means is that we don't use the probability distribution from the output of the model, but simply the index of the maximum argument in the output. The index with the largest value is the class that the model thinks has the highest likelihood of being the correct class. Using our class/index mapping we can easily see as a letter what the prediction is, and calculate a 0.0, or a 1.0 as the score for the one sample.

The comprehensive score we calculate is simply the sum of this exact match of prediction divided by the number of samples we test it on. This is simply the number of predictions it guesses correct. Note, that these results are much higher than training accuracy reported, as cross-entropy is a much less forgiving metric. However, in the context of an application setting, we just care that the model gets the prediction right once the training is done.

We test using the asl_alphabet_test folder provided in the kaggle.com dataset. Our previous baseline model scored a **96.5%**, only faltering on the difference between U, and V, signs that look similar as they do in sight as well. This newly trained model, using heat maps to train, scored only **89.7%**. Upon investigation, we find that the new model trained on heat maps easily predicted the difference between U, and V, but struggled on signs where the "shadow" of the hand occupies the same space but the depth of the fingers relative to each other are different. While the results are not bad, and it is very light-weight, it is simply not an improvement on the previous baseline due to the drawback discussed.

While I don't think as it stands this research makes an immediate improvement, it shows potential for using predicted heat maps to better predictions. The way it can better discern between signs as long as the fingers are distinguishable to the heat map is very promising. Furthermore, through analysis, we identify that the primary hindrance to using heat maps is that it will not be effective for signs where the hand occupies the same outline. What we draw from this is that the heat maps generated should be used as information for what parts of the original image we should "attend" to. Attention is a concept used in Computer Vision and Natural

Language Processing that allows the model to weight certain pieces of the input more than others. In Computer Vision specifically we can think of this as focusing on certain regions of the image with higher resolution. Therefore, given the heat map values, we can use the greyscale value for any pixel as how much we “attend to” the values in the corresponding pixel in the original image.

B. Future Work

What we hope to achieve in the future is a model that returns approximately 99% accuracy. To attain this, we need to improve our heatmap training as well as the keypoint detection. We propose to do this is by overlaying the values of the heatmap image (a value between 0 and 255) and the original RGB values of the input image. What this would achieve is allowing us to measure the lighting of the subject’s area and the contour of the hand. These factors have contributed to making life difficult for our program as it can be hard to distinguish between certain hand gestures through the heatmap alone (signs for e, m, and n are very similar in the view of the heatmap). With more data on the aforementioned factors, we can help our program to find more discernable aspects to the subject’s hand. We would also like to find a piece of hardware with a more powerful GPU so that we can run at a faster framerate. After we achieve those very attainable goals, the next checkpoint would be to implement actual word recognition using Natural Language Processing (NLP). With this, our original program as an application will grow in potential. With an accurate character recognition system in place, we can use this along with some NLP techniques to synthesize words and eventually sentences to allow for an accurate, real-time translation which can be used to effectively communicate with co-workers in presentations, meetings, or even in a text to audio extension of this application.

After all of this has been completed, the next piece of the puzzle is to pack all of this up into a nice, clean, application that can be used by anyone with a hearing deficiency to allow them the right to communicate with their target audience. We hope to prototype this application with the deaf community and to see what they like or dislike about our tool. After that, it is just a matter of slight improvements that will eventually culminate into a functioning, marketable tool that will help those of us who have been denied an easy and accessible tool to help them.

References

[1] Chai, Xiujuan et al. “Sign Language Recognition and Translation with Kinect.” (2013).

[2] Coviu, “How we used AI to translate sign language in real time,” *Medium,* 12-Nov-2018. [Online]. Available: https://medium.com/@coviu/how-we-used-ai-to-translate-sign-language-in-real-time-782238ed6bf.

[3] Gago, Jennifer Joana & Vasco, Valentina & Łukawski, Bartek & Pattacini, Ugo & Tikhanoff, Vadim & Victores, Juan & Balaguer, Carlos. (2019). Sequence-to-Sequence Natural Language to Humanoid Robot Sign Language.

[4] Ma, Yongsen, et al. "Signfi: Sign language recognition using wifi." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2.1 (2018): 1-21.

[5] Nurul Khotimah, Wijayanti & Susanto, Y.A. & Suciati, Nanik. (2017). Combining decision tree and back propagation genetic algorithm neural network for recognizing word gestures in Indonesian Sign Language using Kinect. 95. 292-298.

[6] V. Valentin Bazarevsky and F. Zhang, “On-Device, Real-Time Hand Tracking with MediaPipe,” *Google AI Blog*, 19-Aug-2019. [Online]. Available: https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html.