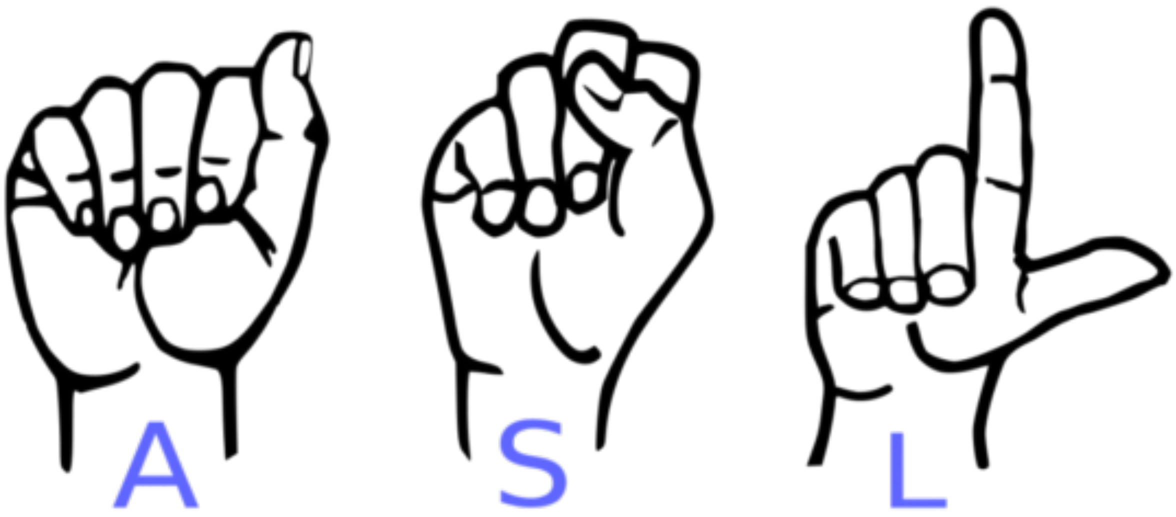

American Sign Language Hand Gesture Recognition

Rawini Dias Follow
Dec 14, 2019 · 13 min read ★

This project and blog was a joint effort by Rawini Dias, LaShay Fontenot, Katie Grant, Chris Henson, and Shivank Sood.
Please visit our Github repository for the project implementation code.

Overview

While there are new and accessible technologies emerging to help those with hearing disabilities, there is still plenty of work to be done. For example, advancements in machine learning algorithms could help the deaf and hard-of-hearing even further by offering ways to better communicate using computer vision applications. Our project aims to do just that.

We sought to create a system that is capable of identifying American Sign Language (ASL) hand gestures. Since ASL has both static and dynamic hand gestures, we needed to build a system that can identify both types of gestures. This article will detail the phases of our project.

Project Summary

Goal: Build a system that can correctly identify American Sign Language signs that corresponds to the hand gestures

Method: The static sign language data for our project was in the form of images. We trained a Convolutional Neural Network (CNN) to identify the signs represented by each of these images. The dynamic sign language dataset we used was collected by a LeapMotion Controller (LMC) and was in the form of (x, y, z) coordinates of each joint of each hand collected every few milliseconds. We feature engineered this data to get useful relative motion data which was then trained on classical classification models to identify the specific sign pertaining to each LMC input.

Applications: Our proposed system will help the deaf and hard-of-hearing communicate better with members of the community. For example, there have been incidents where those who are deaf have had trouble communicating with first responders when in need. Although responders may receive training on the basics of ASL, it is unrealistic to expect everyone to become fully fluent in sign language. Down the line, advancements like these in computer recognition could aid a first responder in understanding and helping those that are unable to communicate through speech.

Another application is to enable the deaf and hard-of-hearing equal access to video consultations, whether in a professional context or while trying to communicate with their healthcare providers via telehealth. Instead of using basic chat, these advancements would allow the hearing-impaired access to effective video communication.

Performance: The proposed model for the still images is able to identify the static signs with an accuracy of 94.33%. Based on our analysis of the dynamic signs, we realized the need to identify if the sign is a one-handed or two-handed sign first, and then identify the sign itself. The final model we propose for the dynamic signs is capable of identifying the one-handed signs with an accuracy of 88.9% and the two-handed signs with an accuracy of 79.0%.

Static Signs

Data Collection & Pre-Processing

Using the Sign Language MNIST dataset from Kaggle, we evaluated models to classify hand gestures for each letter of the alphabet. Due to the motion involved in the letters J and Z, these letters were not included in the dataset. However, the data includes approximately 35,000 28x28 pixel images of the remaining 24 letters of the alphabet. Similar to the original MNIST hand drawn images, the data contains an array of grayscale values for the 784 pixels in each image. One of these images is shown below.

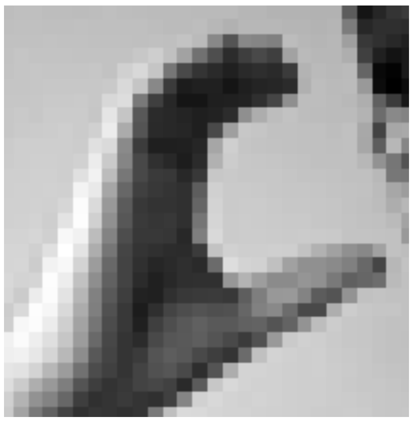

Figure 1: Sample image of the letter "C" from the training dataset

Learning/Modeling

We used a Convolutional Neural Network, or CNN, model to classify the static images in our first dataset. Our first goal when building the neural network was to define our input layer. A 28x28 image contains 784 pixels each represented by a grayscale value ranging from 0 (black) to 1 (white). By converting each image to a series of numbers, we transform the data into a format the computer can read.

Once the input layer has been prepared, it can be processed by the neural network's hidden layers. The architecture of our neural network can be seen below.

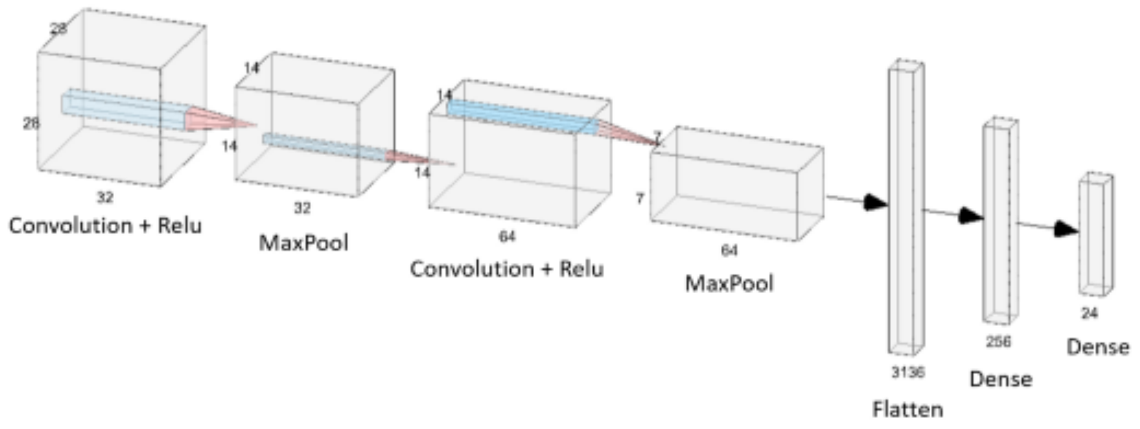

Figure 2: The architecture of the Convolutional Neural Network

The first hidden layer is composed of several nodes each of which takes a weighted sum of the 784 input values. The weighted sum of inputs is then input into an activation function. For our network, we used a rectified linear unit, or ReLU.

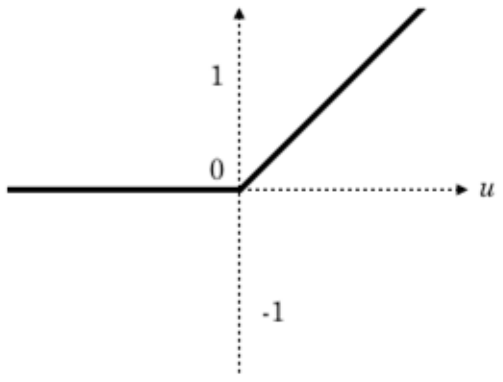

Figure 3: ReLU activation function

The above graph shows that the ReLU will output 0 when the input is negative, but will not change the input otherwise. The outputs from the ReLU will serve as the inputs to the next hidden layer in the network.

To better understand how each hidden layer transforms the data, we can visualize each layer's outputs. Our first layer had 32 channels, so the process described above was repeated 32 times. This allows the network to capture several features in each image. If we input the image depicting the letter "C" that was shown previously, we obtain the following set of 32 outputs.

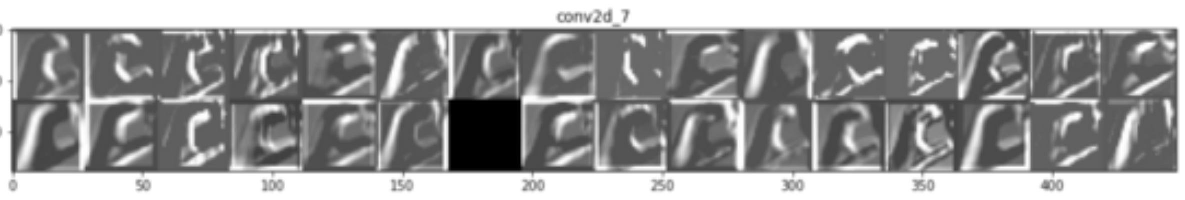

Figure 4: Outputs from the first hidden layer

Here, we see how each channel transforms the image a little differently. Based on these images, it appears the network is extracting information about the edges and general shape of the person's hand. As the data continues to move through the hidden layers, the neural network attempts to extract more abstract features. Below are the outputs of the fourth hidden layer. These images are much less interpretable to the human eye, but will be very useful to the network as it attempts to classify the image into 1 of 24 potential classes.

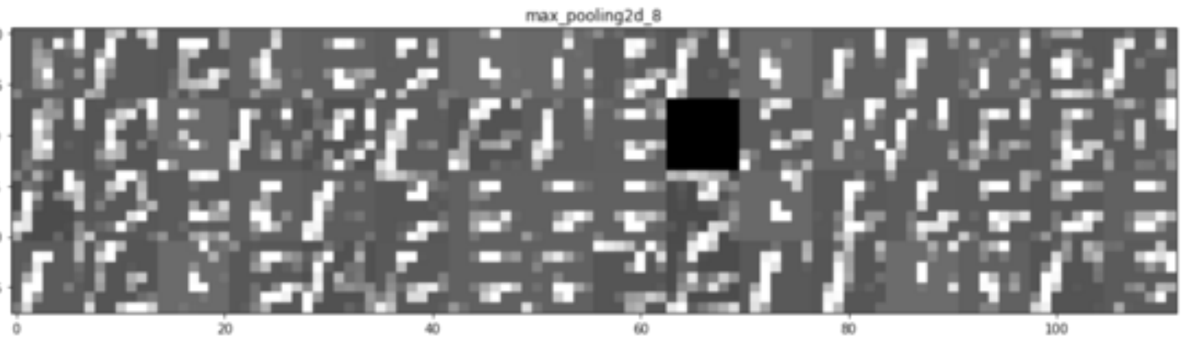

Figure 5: Outputs from the fourth hidden layer

Once the data has passed through the Convolution and MaxPool layers of the neural network, it enters the Flatten and Dense layers. These layers are responsible for reducing the data to one dimension and identifying an image's class.

After the CNN's architecture was defined, we attempted to optimize the model's performance by selecting an appropriate value for the number of epochs. Earlier we mentioned that each node takes a weighted sum of its inputs. The weights applied to each input are learned through the training process and updated with each epoch. An epoch is a single pass through all of the training data. On the first epoch, the neural network estimates a value for each weight. For each subsequent epoch, the neural network updates these weights with values that reduce overall loss. Generally, more epochs result in more accurate classifiers; however, more epochs also produce more complex models. Using a validation set, we determined that 10 epochs provided us with the best balance between accuracy and complexity.

Results

The training and validation datasets used to build and optimize the model contained 80% of the original data. The remaining 20% (~7,000 samples) was reserved for model testing. When this test data was input to the model, it achieved 94.33% accuracy. To further understand the strengths and weaknesses of this model, we created a confusion matrix.

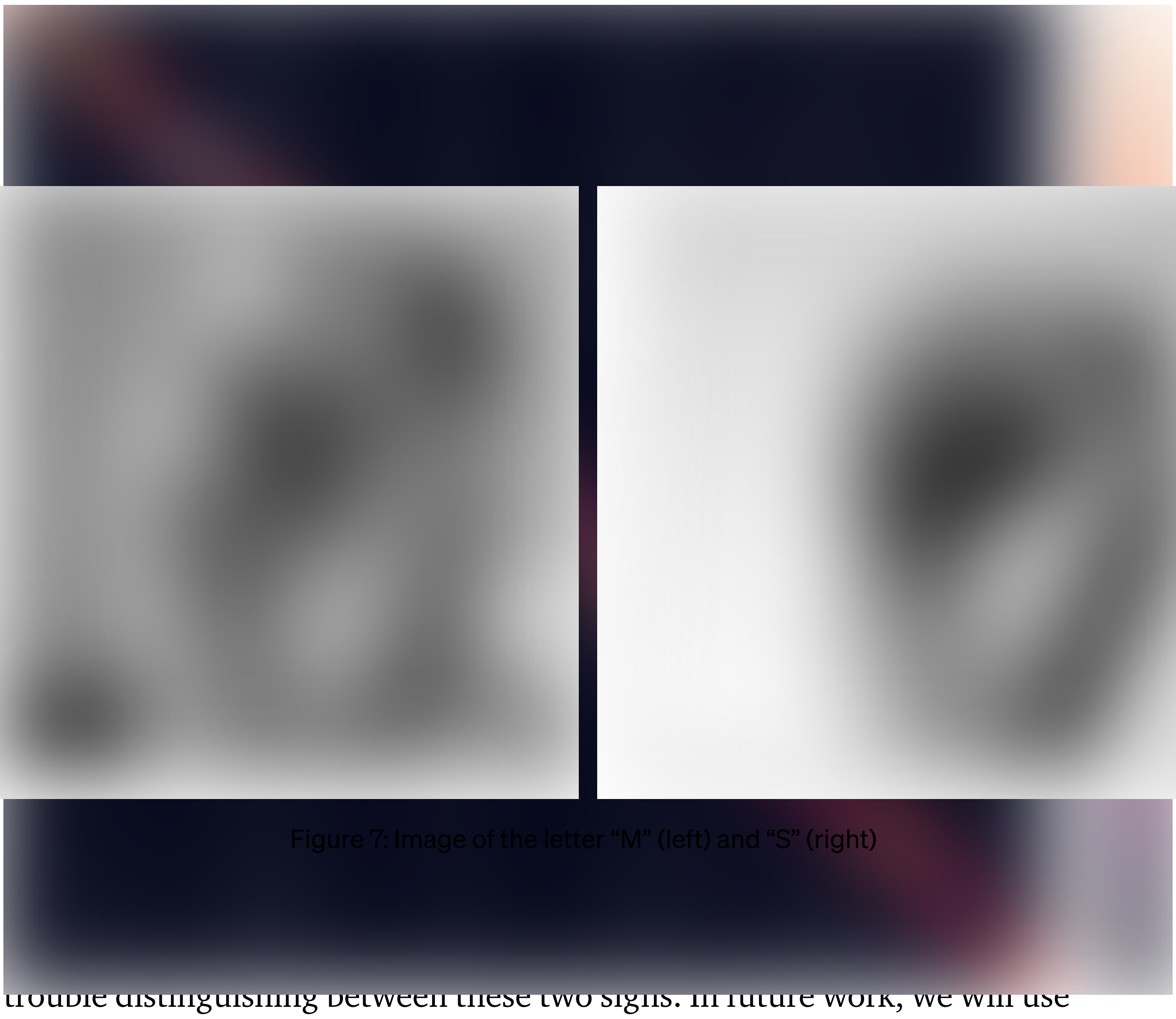
Figure 6: Confusion matrix of the CNN's outputs on the test data

From the confusion matrix, we see that the two signs most commonly confused are the letters "M" and "S". Images of each of these signs are shown below.

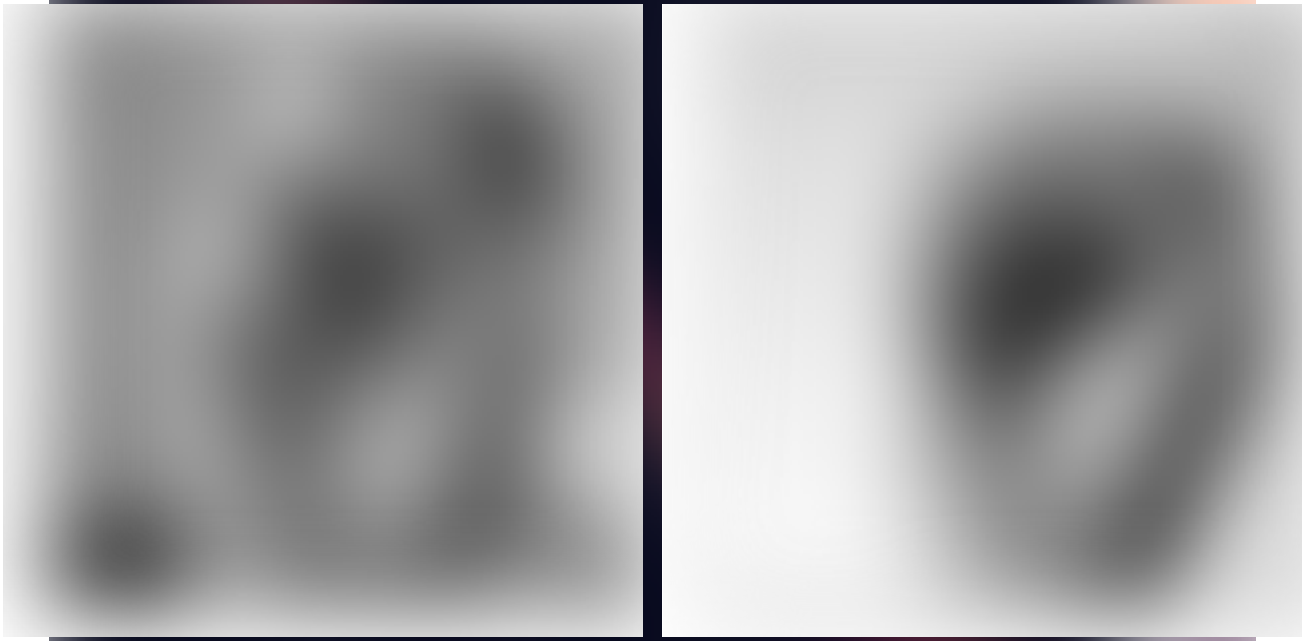
Figure 7: Image of the letter "M" (left) and "S" (right)

trouble distinguishing between these two signs. In future work, we will use images with higher resolution that allow for more intricate details to be extracted from the images. Hopefully, this will further improve the accuracy of our model. An additional limitation of this model was its inability to recognize moving signs, such as the letters "J" and "Z". In the next section, we explore a data source that is better equipped to recognize dynamic signs.

Dynamic Signs

Understanding the Data

The second phase of our project will focus on dynamic signs (i.e. moving signs). This dataset consists of 25 subjects each performing the same 60 ASL signs with both their left and right hands using a LeapMotion Controller (LMC). Therefore, this dataset has 60 different ASL signs (or class labels) that we are trying to accurately predict.

The LMC device records the position of the fingers, joints, palm, wrist, and arm every 0.04 seconds. In other words, the LMC acquires spatial coordinates of the skeleton joints of the hands and how these coordinates vary with time. Our second dataset is made up of these coordinate points.

Let us try to understand the nature of this data in more detail using Figure 9.

Figure 9: Pictorial representation of the hand as seen by the LeapMotion Controller

The metacarpals, proximal, intermediate, and distal bones refer to the four different bones of an anatomical finger. The Leap Motion dataset gives us the (x, y, z) coordinates of each of these bones in each finger every 0.04 seconds for the duration of the sign. It also gives us the coordinates for the palm, wrist, and arm. Altogether, these coordinate points as a function of time provides discriminating information that can be used to identify the type of hand gesture (or ASL sign).

Feature Engineering

Realizing that we needed our dataframes for each test subject to be comparable, we first transformed each dataframe by taking the difference of each successive row of coordinates, giving the distance that each part of the hand (x, y, and z coordinates) traveled in between each measurement that the Leap Motion device recorded. This captured movement in intervals of approximately .04 seconds for the 54 parts of the hand identified during the motion capture. Each of these transformed data frames consisted of anywhere from 398–1203 time intervals, each with 162 columns of coordinate data.

After this differencing, we next sought to derive features that captured information about the movement of the hand during the given time interval. We decided to take the mean and standard deviation of each of these columns. While this may see relatively simplistic, we found this a computationally cheap way to capture information.

Consider what the mean of each of these differenced columns represents. This is the average distance traveled by each part of the hand in each time interval. Likewise, taking the standard deviation of each of these columns represents the variation in this displacement. In other words, this serves as a proxy for velocity. (Surprisingly, adding the actual calculation for velocity at each point actually reduced accuracy!)

In addition to these two sets of features, we also experimented with identifying pairs of hand coordinates with strong correlation and using the polynomial weights as features for classification. While this produced incremental gains in accuracy in a reduced data set (10 classes as compared to all 60) this did not scale well to the full set of signs.

Another idea we had was to use the angles the fingers formed. We calculated the internal angles of the joints between distal and intermediate bones and the internal angles of the joints between intermediate and proximal bones. However, adding these angles to the previously derived mean and standard deviation features for each joint indicated multicollinearity in the independent variables. Using only the angle features derived resulted in acceptable classification accuracy on the reduced data set of 10 classes, but it did not extend well to the dataset with all 60 classes.

Learning/Modeling

Initially our team began model selection by looking at the subset of our data that consisted of numeric signs zero through ten, developing the features described above. What we realized is that this subset of signs could be easily distinguished by the fact that each of them only utilizes a single hand to complete the sign. Realizing this meant that the still left hand was only contributing noise to the data set, we removed all coordinates originating from the left hand and saw a significant gain in classification accuracy.

Now wanting to extend this to our full data set, we used both hands, again with the above features, and noted a significant decline in accuracy. In an attempt to identify where our model was unable to distinguish between different signs, we found that a better understanding of sign language would inform our model pipeline.

We first attempted to conditionally identify which signs utilized only one hand, with the intent of dividing our data set into two groups. Through both manually examining the signs and developing thresholds for our feature means, we split the data set into 22 “two-handed” signs and 38 “one-handed” signs by identifying data frames whose left handed attributes appeared to be still, as measured by the sum of mean absolute deviation in left-handed coordinates. This distinction however, is often far from clear.

As members of our team are far from fluent in American Sign Language, we had to do some research to understand more about the signs in our dataset. We set out to determine how many, and which hands were involved in signing each word. Fortunately for us, the website Signing Savvy offers an ASL dictionary complete with videos of the various ways to sign each word. For example, the word ‘bug’ can be signed in two different ways depending on the manner in which the word is being used. Both of these signs used only the right hand when signing. On the other hand, the word 'cost' involves both hands, but only one hand is in motion (video). Using this website, we were able to understand the application-based differences of the signs in our dataset. Of the 60 words in our dataset, there were 9 that could be signed using either both hands, or just the right hand (Car Drive, Come, Cost, Finish, Go, Happy, Hurt, Small, When).

Regardless of the difficulties associated with splitting our data set between one-handed and two-handed signs, we found that this methodology significantly increased our accuracy for the complete data set.

Model Selection

After creating the previously mentioned features, we experimented with several classifiers. Below are two boxplots, showing the performance of various models on our one-handed and two-handed signs. Ultimately, we decided that linear

r

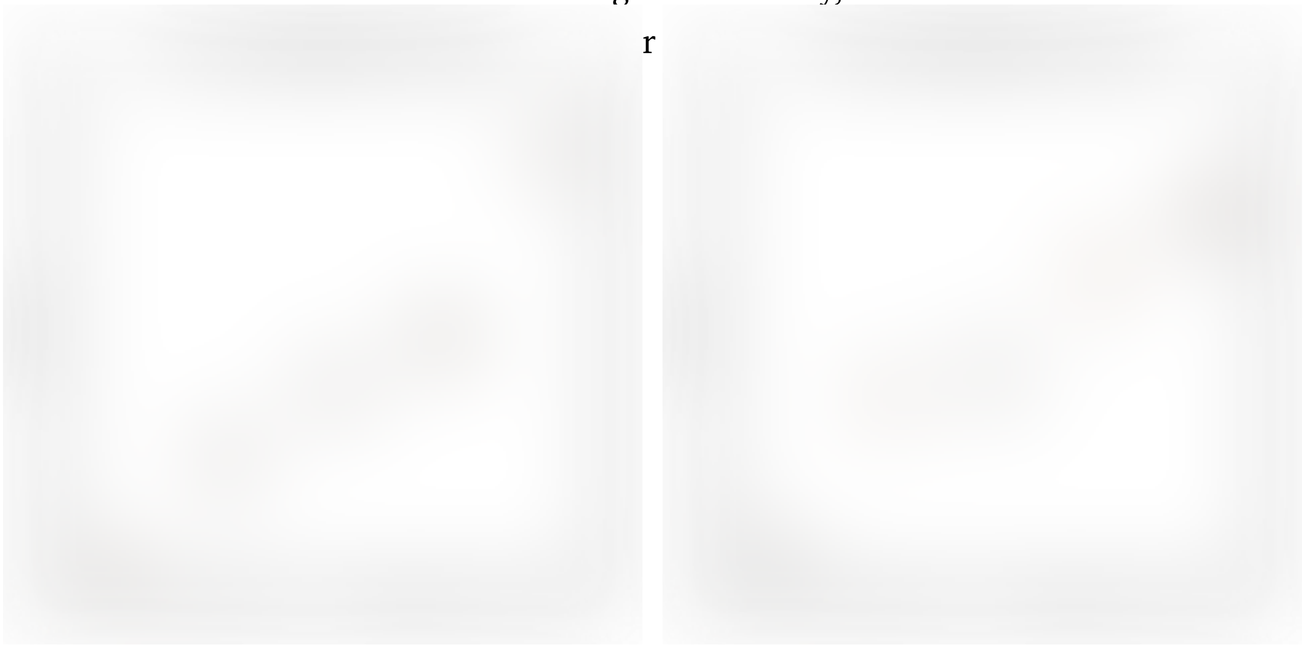

Figure 10: Comparison of model performance for one-handed signs (left) and two-handed signs (right)

Results

The plots in Figure 12 show the results of our model run one hundred times using LDA, each iteration taking a different randomly selected training/test split stratified by our classification label. Two pieces of information immediately stand out. First, the tactic of separating one vs. two handed signs is very useful. Second, our model accuracy has a relatively high standard deviation with regard to classification accuracy given a random set of training data. Considering the small sample size of 25 test subjects however, this should not be much of a surprise.

Figure 12: Distribution of classification accuracy for different train/test splits

The above plot demonstrates that our two-handed signs are systematically misclassified more often than our one-handed signs. The natural question is to identify the particular signs that may be problematic for our model. The most commonly confused signs are shown in the table below:

Table 1: Some of the signs that were misclassified with each other

Table 1 shows the signs that were misclassified with each other, i.e. the sign for 'come' was misclassified as either 'big' or 'with'. Similarly, the sign for 'red' was misidentified as 'cry' in our dataset. This was to be expected as these pairs of signs were very similar in motion.

Furthermore, the overall model misclassifications can be categorized into two groups.

1. Signs that are "over predicted". In other words, signs that are predicted when the actual sign is something else. This is the equivalent to false positives in a two-class problem.
2. Signs that are "under predicted". In other words, signs that are not predicted when they should be. This is the equivalent to false negatives in a two-class problem.

Below are two charts that identify these cases:

Table 2: The list of 'over-predicted' signs

Table 3: The list of 'under-predicted' signs

For example, the sign "Cold" (link to video) often fails to be predicted. Intuitively, what we are seeing is that because this sign involves both hands moving through a very small range of motion that it is very easy for our model to predict this in place of another sign.

In order to further visualize how these signs behave in our classification model, we generated ROC plots for each sign based on a “One Versus Rest” Classification (still using LDA), where for each individual sign we treat our model as working with a two class model (for instance “Cold” versus “not Cold”). Below we see the results for one of the most problematic signs:

Figure 13: ROC for the word ‘Coat’ versus the rest of the signs

further work with the LeapMotion API to enable real-time generation of data, feeding through the model, and identification of the word and/or numbers. This [illegible] labels it currently deals with.

In conclusion, we see this application having real potential in improving the lives of the hearing-impaired and as such it would be a worthy goal to continue development.

1. http://blog.leapmotion.com/getting-started-leap-motion-sdk/

2. https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847

3. https://www.kaggle.com/datamunge/sign-language-mnist#sign_mnist_test.zip

4. https://data.mendeley.com/datasets/c7zmhcfnyd/1