The importance of communication (ASL Translator capstone project)

<u>Final Report</u>

Source: CanStockPhoto.com,

1. Problem statement:

Many people in North America (USA and Canada) don't have the ability to communicate verbally, so they have to use other ways of communication, like the American Sign Language (ASL). The problem with this type of communication is that many people don't know how to interpret it or use it. So what can we do about it?

2. Background:

American Sign Language (ASL) is the natural language of around 500,000 hearing impaired people in the US and Canada. A "natural" language is a language that is learned as a first language in childhood. However, not all people with this problem learn ASL as their first language.

The National Center for Health Statistics estimates that 28 million Americans (about 10% of the population) have some degree of hearing loss. About 2 million of these 28 million people are classified as deaf (they can't hear everyday sounds or speech even with a hearing aid). Only about 10% of these 2 million people were born deaf. The other 90% became deaf later in life. Moreover, according to the World Federation of the Deaf, there are more than 70 million deaf people worldwide. More than 80% of them live in developing countries.

Many hearing people are fluent in ASL. Sign language has become more and more popular in recent years and many hearing people are registering for high school and college ASL classes. And according to The Rhode Island Commission on the Deaf and Hard of Hearing (RICDHH) ASL is the third most used language in the United States, after English and Spanish.

But, what is Sign Language?

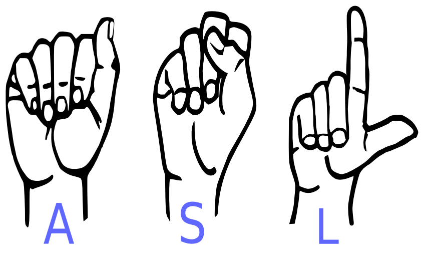
According to Wikipedia, ASL is a complete and organized visual language that is expressed by both manual and nonmanual features.

ASL is not English at all. ASL is a distinct language with its own syntax and grammar and has been developed over many years by deaf people as a means of communication.

Deaf people who use ASL see this language as not only a means of communication, but a source of cultural unity and pride. Due to the importance of sign languages around the world, there must be a simple way to be able to communicate with each other as simple as any daily life activity.

3. Goal:

To create a tool, to help people with a hearing loss disability, to communicate with another person by translating a message emitted in the American Sign Language (ASL) into text.



Source: Wikipedia by Psiĥedelisto

4. Dataset:

There are 146,627 images divided into 3 datasets. The datasets were gathered from different sources.

- <u>Data set No. 1 - Name: **ASL,** source: Kaggle:</u>

  a) Number of images: 84,000
  b) Number of classes: 28 (Alphabet and delete, space)
  c) Format: Color images in size 200 x 200

- <u>Data set No. 2 - Name: **MNIST,** source: Kaggle:</u>

  a) Number of images: 34,627 (train: 27455, test: 7172)
  b) Number of classes: 24 (Alphabet except 'J' and 'Z')
  c) Format: A csv file of grayscale images of size 28 x 28

- <u>Data set No. 2 - Name: **APPLE_CAM,** source: Self made:</u>

  a) Number of images: 28,000

b) Number of classes: 28 (Alphabet and delete, space)
c) Format: Grayscale images in size 28 x 28

5. Data wrangling, EDA, and preprocessing:

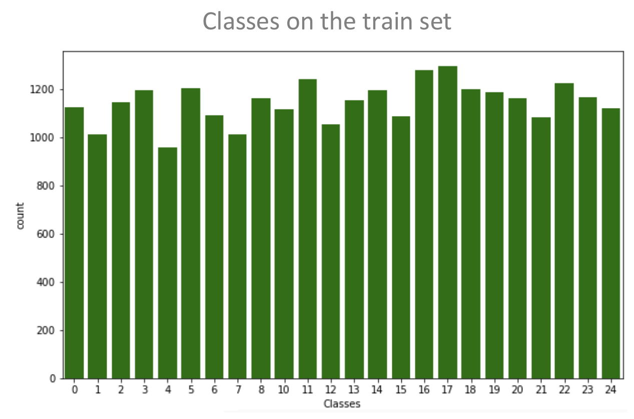
During the Data wrangling and EDA step, I checked the integrity of the three datasets that I had. Additionally, since the sets had different format, I needed to process and reshape to be able to integrate all of them together into a big csv file.

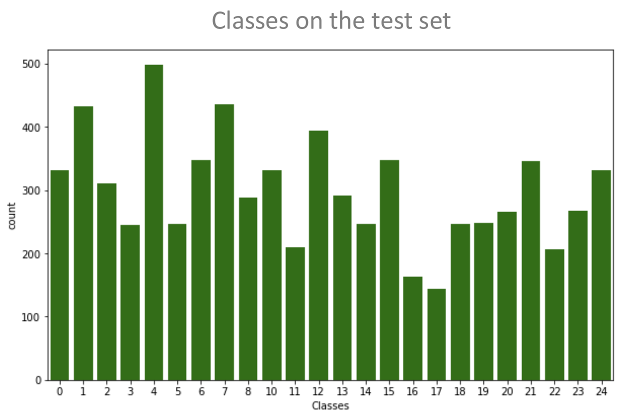
The first dataset that I checked was the **MNIST** set, which was originally saved as two csv files. One file was for training (train_MNIST) and the other one for testing (test_MNIST). Both data sets only had 24 classes as labels, so I was expecting to have some class imbalance by the end of this step.

The results for checking this first dataset are as follows:

- Missing values: 0
- Duplicated values: 0
- Values out of range (0-255): 0
- Data balance: some imbalance (~200 samples difference between the minimum and maximum counts for classes)

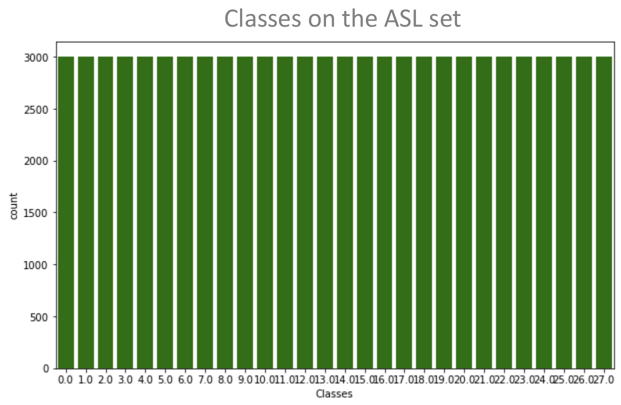
The count plots for the train_MNIST and test_MNIST show the balance in every set. The train set is the only MNIST set that I will use for training, so the class balance in the test set is not relevant.



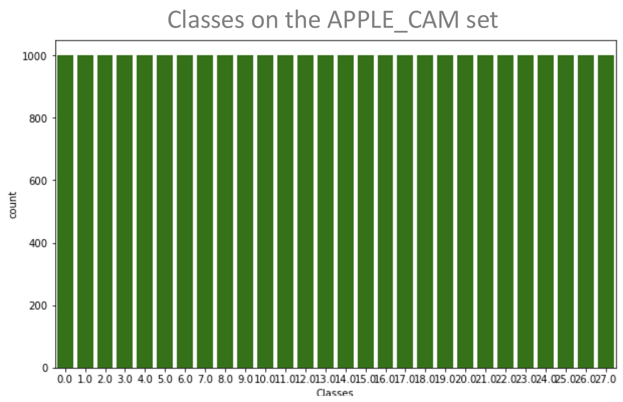


The next step was to check the **ASL** dataset. This set was conformed of 84,000 color images of 200 x 200 pixels. In this case, I decided to standardize the format of the data to grayscale images of 28 x 28 pixels in size. The reason for doing this was to handle a smaller number of features (pixels) and make it easier for the CNN to train on the data.

The **ASL** data was in a picture format, so I used a utility from the keras library to read the images and transform them, eventually, to a csv file.
After doing the transformations, I checked the balance of the classes for this set. The results are as follows: There are 3,000 images per class.



Finally, I used the same process, as for the **ASL** set, to transform the images in the **APPLE_CAM** set to a csv file. This set contained 1,000 images per class. The following classes balance count plot shows the results.

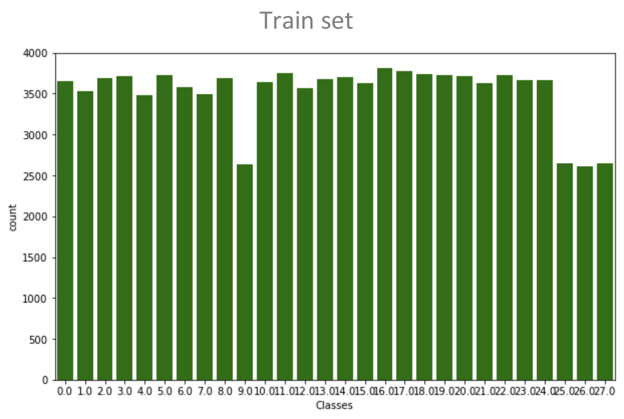


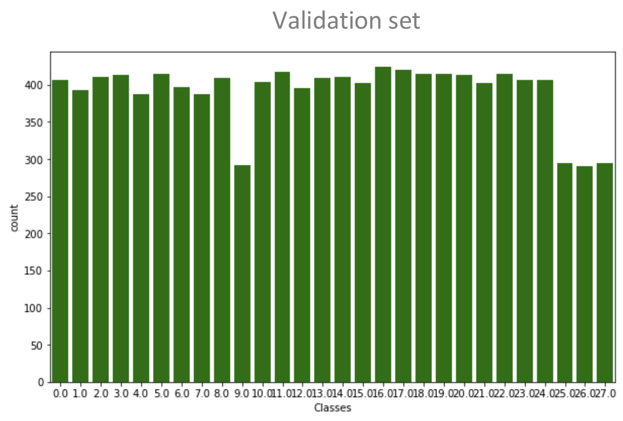
After doing all the processing of the datasets, I integrated them into a big dataset and saved the file in the .csv format to use anytime.

The integrated dataset had the following information:

- Integrated set: 139,455 samples
- Train set: 109,455 samples (reduced the integrated set by 30,000 images)
- Train_validation split:
  - Train set: 98509 samples
  - Validation set: 10946 samples
- Test set: 37,172 samples (30,000 images from the integrated set + MNIST test set)

The class balance was good, except for the classes 9 for letter 'J', 25 for letter 'Z', 26 for 'delete' and 27 for 'space'. These four classes were not present in the MNIST dataset, because of this, the imbalance was expected. The count plot for the class balance of the train and validation sets shows as follows.





The final step in preprocessing the data was to encode the labels and normalize the feature values. First, the reason for encoding the labels as one-hot encoded was because this is the correct shape of the labels that the CNN needs as input.

On the other hand, the motivation for normalizing the feature values (pixels), was to make it easier for the CNN to handle values with a range of [0,1] instead of a range of [0,255].

Additionally, the CNN needs the features to be in a shape of (samples, width pixels, height pixels, channels). So, I reshaped the train, validation, and test features to (-1,28,28,1).

After all these steps I had all my data ready for training a CNN. My goal was to set up a CNN and achieve 100% accuracy in predictions on the test set, which had 37,172 samples.

6. Some basics on Convolutional Neural Networks (CNN):

Now, before going into the modeling, let's take a quick overview of a Convolutional Neural Network (CNN).

A CNN is basically made of an input layer, a defined number of hidden layers, and an output layer (see Fig. 1). In this case, the input layer is every image fed into the network. These images are passed into the hidden layers for filtering, reshaping and, basically, for creating a feature map. This map will eventually be passed to the output layer which contains the labels or classes for our classification problem.

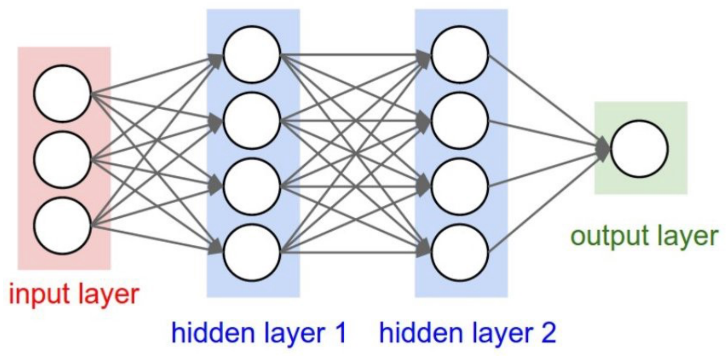


Fig. 1
Source: i2 Tutorials

The following image (Fig. 2) gives a wonderful and clear overview of a simple CNN.

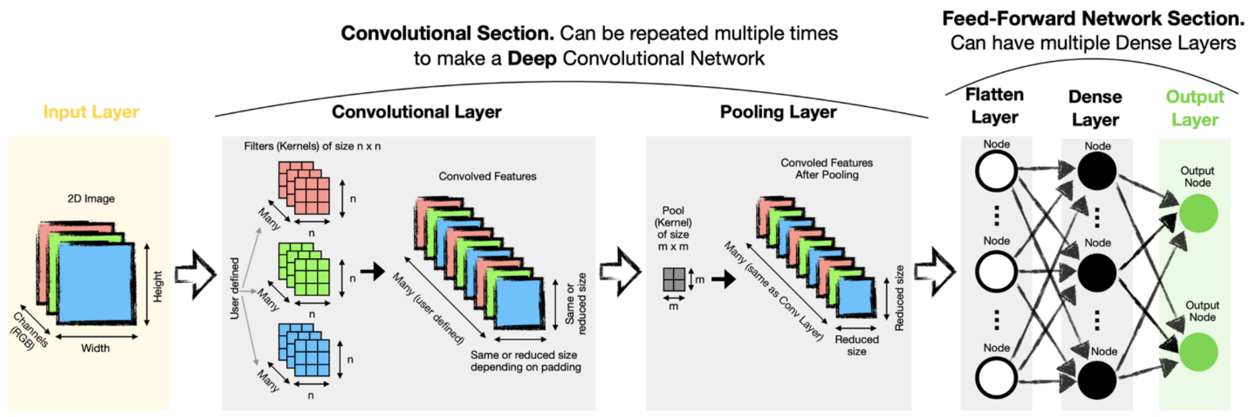

Convolutional Section. Can be repeated multiple times to make a Deep Convolutional Network
Feed-Forward Network Section. Can have multiple Dense Layers
Input Layer
Convolutional Layer
Pooling Layer
Flatten Layer
Dense Layer
Output Layer
2D Image
Height
Width
Channels (RGB)
Filters (Kernels) of size n x n
User defined
Many
Convolved Features
Many (user defined)
Same or reduced size
Same or reduced size depending on padding
Pool (Kernel) of size m x m
Convoled Features After Pooling
Many (same as Conv Layer)
Reduced size
Node
Output Node

Fig. 2
Source: TDS by Saul Dobilas

As mentioned previously, a CNN has an input layer which contains the data that will be fed into the network. These images will be fed into a hidden layer, which could be a convolutional layer, a pooling layer, among others. The purpose of a convolutional layer is to pass filters (kernels) over the images and try to learn features from them. Normally, intial convolutional layers will learn simple shapes, like lines and edges. As the network gets deeper, other convolutional layers will learn more complex shapes to make the network more robust and able to classify the images better.

On the other hand, a pooling layer, will pass a kernel of size m x m on every region of the image, which will get the maximum or the average value of the pixels on this part of the image. The decision on whether to take the maximum or the average value of this part of the image depends on the type of pooling layer that is selected during the configuration of the network. A good example of a max pooling layer is shown on Fig. 3.

After defining all the hidden layers in the convolutional section, which will determine the feature map for the network, it is necessary to flatten the values into an array which then will be the input of a Dense layer. The main property of a Dense layer is that it has all its nodes connected. Because of this, it is better to avoid using them on early stages of the network to prevent overfitting.

Finally, the last Dense layer in the network is connected to the output layer, which contains the classes (in this case 28) that we have for classification.

The following image shows an example of a Max pooling layer. In this case, the size of the kernel (the small purple window passed to every region of the image) is 2 x 2. Additionally, the stride = 2, which indicates that the kernel will slide 2 pixels every time it moves to take the next maximum value in that region. In this example, the maxpool layer will reduce the size of the image from 4 x 4 to 2 x 2.

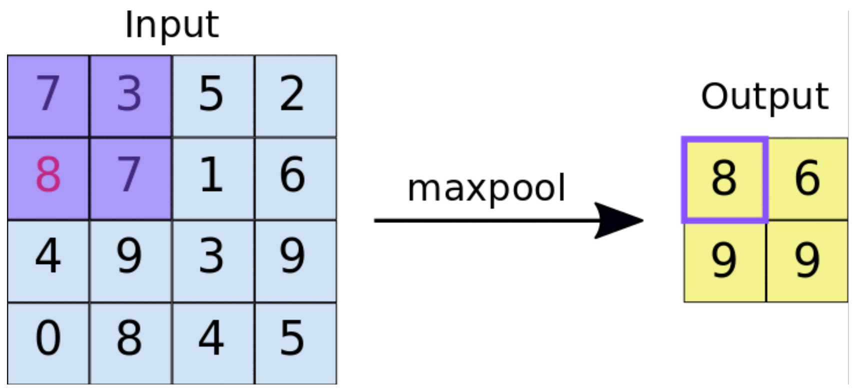
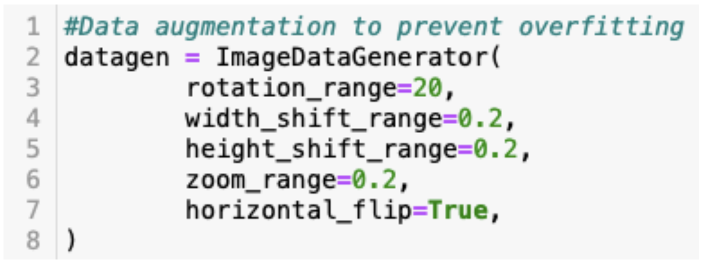


Fig. 3
Source: NumPyNet

7. Modeling:

The modeling step had one goal, to create a CNN to achieve 100% accuracy in the test set. Considering this goal, I set up six different CNNs to try to achieve it.

The training set had almost 100,000 samples, but I decided to increase this number, for four of the six CNNs that I tried, by using data augmentation. I increased the number of images by rotating, shifting, and flipping random images in the set. The following image shows the data augmentation used.



```
#Data augmentation to prevent overfitting
datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
)
```

Fig. 4

Additionally, I used learning rate reduction and different number of epochs to try and achieve a higher accuracy.

At the end of the trial of the different CNNs, I decided to use the sixth CNN trained. This CNN achieved 99.8% of accuracy and had a more stable condition through every epoch while increasing accuracy and decreasing the train and validation losses. Moreover, it didn't seem to have any overfitting. The following graph shows the progress on the train and validation losses during every epoch for the best CNN.

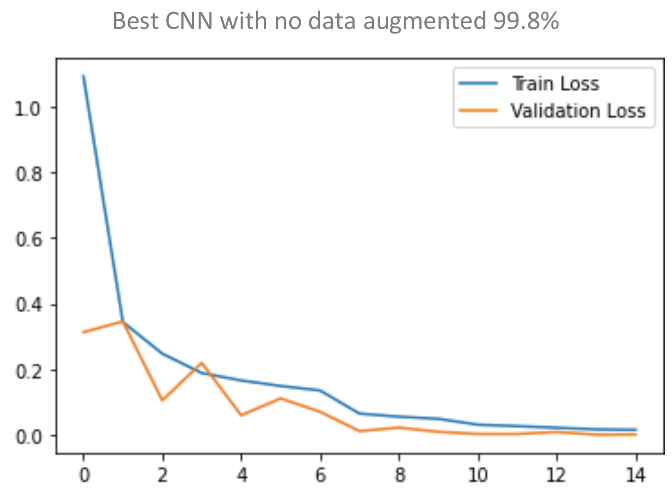


Fig. 5

The following table shows the results and parameters for each of the six CNNs.

CNN	Data augmented	Number of parameters	Reduce learning rate	Epochs	Train Accuracy	Validation Accuracy	Test accuracy	Number misclassified
1	No	210,204	No	5	90.5%	91.1%	80.3%	-
2	Yes	210,204	No	5	33.9%	42.1%	35.6%	-
3	Yes	6,719,260	No	5	42.5%	55.0%	46.2%	-
4	Yes	1,830,428	Yes	5	71.5%	80.4%	75.6%	-
5	Yes	1,830,428	Yes	25	96.9%	98.4%	97.5%	948/37,172
6	**No**	**1,830,428**	**Yes**	**15**	**99.5%**	**99.9%**	**99.8%**	**71/37,172**

Table 1

The best CNN (sixth in the table) made only 71 misclassifications while doing predictions for the 37,172 samples in the test set. The following heat map shows the results in classifications for the best CNN on the test set. The classes with more misclassifications were the letter 'M' and letter 'Y'.

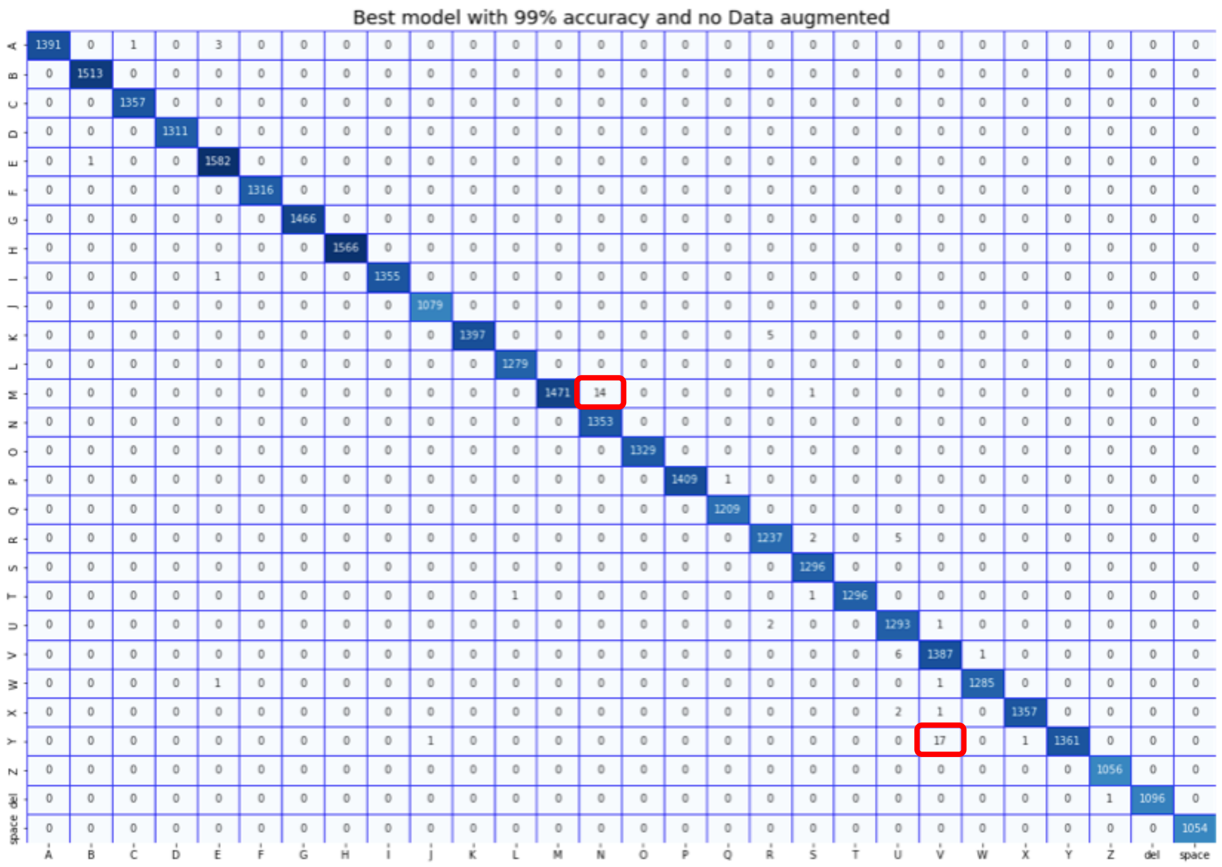


Fig. 6

I tried one more test with the best CNN. The intention of doing this test was to place the CNN in a challenge, that could simulate a real-life situation while using this CNN model. The test consisted in using the web cam on my laptop to try and do real time translations.

By using some of the utilities in the cv2 and cvzone libraries, I took real time video, which captured an infinite number of images and applied predictions to them. These predictions were shown on the screen of the laptop as the label that they corresponded to.

The following set of images show the results of this test.

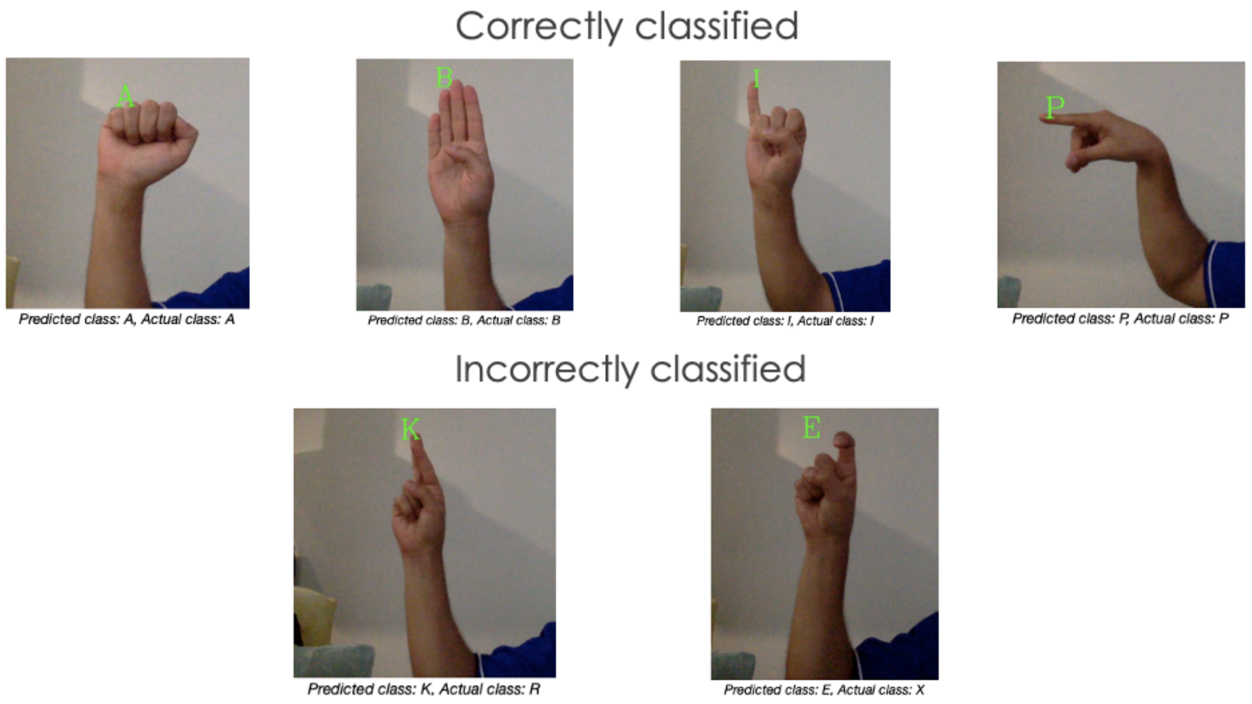

Correctly classified
A
B
I
P
Predicted class: A, Actual class: A
Predicted class: B, Actual class: B
Predicted class: I, Actual class: I
Predicted class: P, Actual class: P
Incorrectly classified
K
E
Predicted class: K, Actual class: R
Predicted class: E, Actual class: X

Fig. 7

The results were not as positive as expected. The predictions were correct on 4 of 6 letters, which means that, although, the CNN had accuracy of 99.8% on the test set, the model could not generalize well. Perhaps, more information is needed.

8. Conclusions and future work:

This project was a great opportunity to work with image recognition and understand the basics of convolutional neural networks.

I think that it was very beneficial to have as much information (images) as possible to train the model, but I was able to understand the challenge in finding a proper network to classify with high accuracy. Every component of the network plays an important roll in the classification process so, the better you understand these components, the better chance you have to improve its performance.

Additionally, by the end of the training process, I was able to see that the network trained without augmented data performed the best, but during the final challenge seemed to lose some classification power. This problem makes me wonder whether an improved version of the best model trained with augmented data, could get better results than the one with 99% accuracy. How important is the augmented data when you have ~100,000 images to train your model?

There is more area to explore with the real time video use to classify images. I understood that even though my model had almost perfect accuracy, this doesn't mean that it will perform well in any environment. Perhaps I need to acquire even more data to tackle this challenge.

As for the future steps to take, there is still a lot of work to do on this project. I will continue to work on the following:

- Gather more data
- Trial and error with augmented data
- Explore transfer learning
- Train a model on a wider range of classes
- Test results on real time video
- Create application
- Distribution