INTERPRETING AMERICAN SIGN LANGUAGE USING MACHINE LEARNING

SIRISHA KASIBHOTLA

AKRITI BHANOT

NICHOLAS CHIN

KRISHNA PRIYA JYESTA

ADEYEMI OMISAKIN

INSPIRATION BEHIND THE PROBLEM:

Sign languages across the world are different. There are 300 distinct sign languages across the world. Today, around one million people use American Sign Language (ASL) as their main way to communicate, according to Communication Service for the Deaf.

According to Communication Services for the Deaf:

- 98% of deaf people do not receive education in sign language
- 72% of families do not sign with their deaf children
- 70% of deaf people don't work or are underemployed
- 1 in 4 deaf people has left a job due to discrimination

But more is being done to include ASL users. According to NAD, in recent years several government departments have ensured important information reaches the deaf and hard of hearing community by creating videos and/or establishing a hotline in American Sign Language. Even the Internal Revenue Service (IRS) has its own ASL YouTube channel.

There are also more resources appearing online and for mobile devices to help us communicate more easily. A Netherlands-based start-up has developed an artificial intelligence (AI) powered smartphone app for deaf and mute people, which it says offers a low-cost and superior approach to translating sign language into text and speech in real time.

The easy-to-use innovative digital interpreter dubbed as "Google translator for the deaf and mute" works by placing a smartphone in front of the user while the app translates gestures or sign language into text and speech. Every day thousands of local businesses around the globe face problems with providing their services to deaf. The app, called GnoSys, uses neural networks and computer vision to recognize the video of sign language speakers, and then smart algorithms translate it into speech. Affordable and always available interpreter services are in huge demand in the deaf community.

It will help drive inclusivity at the workplace by removing communication barriers between the disabled and able. Interpreters aren't usually available, and also could be expensive. Pen and paper approach are just a bad idea: it's very uncomfortable, messy, time-consuming for both deaf and hearing person.

Apps can be used on multiple devices which can translate as quick as the person speaks, translate any sign language and can be plugged into many products, such as video chat applications, AI assistants, etc. It just requires a camera on the device facing the signing person, and a connection to the internet. It provides the same services only an experienced sign language interpreter could

do. Speakers of American Sign Language (ASL) have been fortunate in that a number of startups and research projects are dedicated to translating ASL in real time.

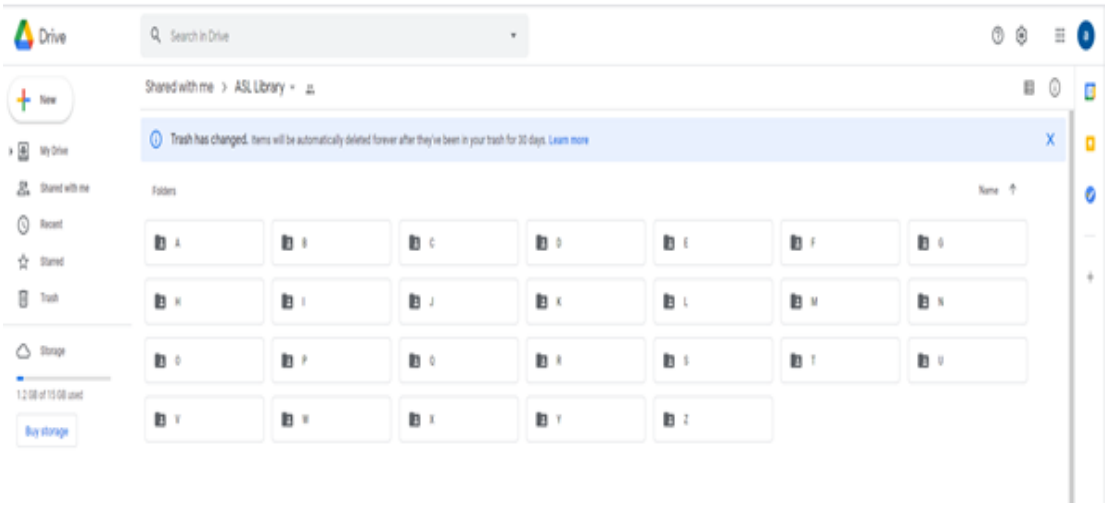
Our goal was to build a mobile application that uses a camera to capture a person signing the ASL alphabet, and translate it in real time. There are many other apps which require purchase with similar functionalities & our goal is to make our app available for everyone free of cost. This would involve:

- Gathering data
- Training a machine learning model to recognize the ASL alphabet
- Building the user interface.

DATA COLLECTION:

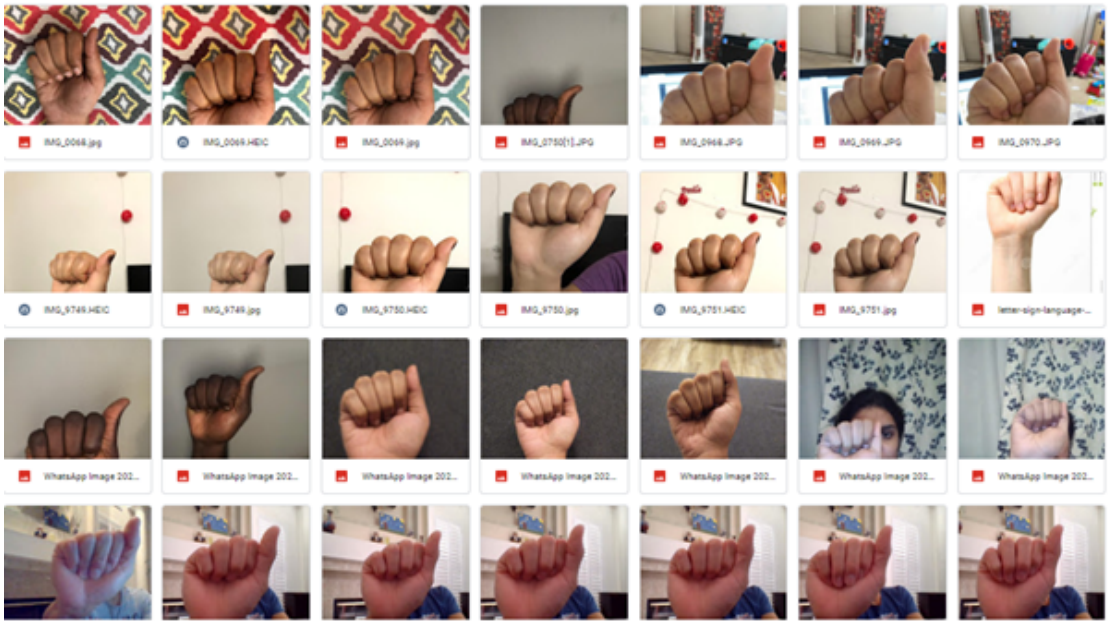
During the dataset formation our team decided to create our own dataset, all the five members of our team added the images of the ASL alphabets in the Shared Google Drive folder which we named as ASL Library. We also supplemented our dataset with additional images from Kaggle and Google images. The total data set contains 5408 images where each alphabet folder contains close to 200 images. After the train test split of 80:20 the training dataset contains a total of 4326 images. And the test dataset contains 1082 images.

THE GOOGLE DRIVE ASL ALPHABET STRUCTURE:



Google drive ASL alphabet structure

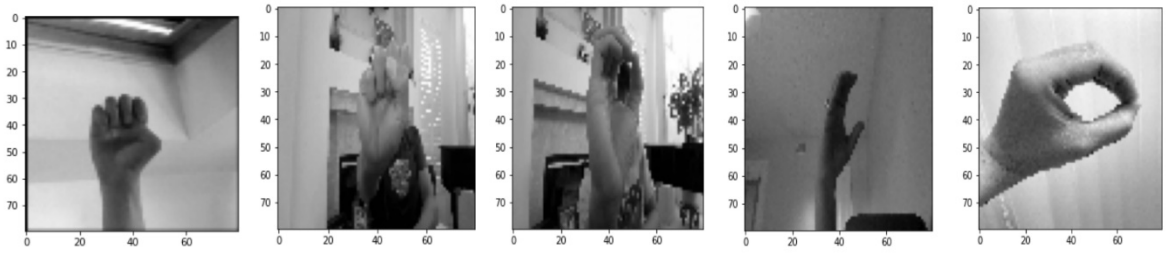
The google drive folder with the Alphabet Images for letter A



Google drive with alphabet images for letter A

The training dataset has 4326 total images with the images being resized to 80 by 80 pixels and converted to grayscale for storage and processing purposes. The input layer of the model will take images of size (80,80,1) where 80,80 are height and width of the image respectively while 1 represents the colour channel of the image for grayscale. The output layer of our model will have 26 neurons for 26 different letters, and the activation function will be softmax since it is a multiclass classification problem.

Sample Images from our dataset



Sample Images from dataset

Therefore, the shape of the training dataset is

```
X_train.shape
```

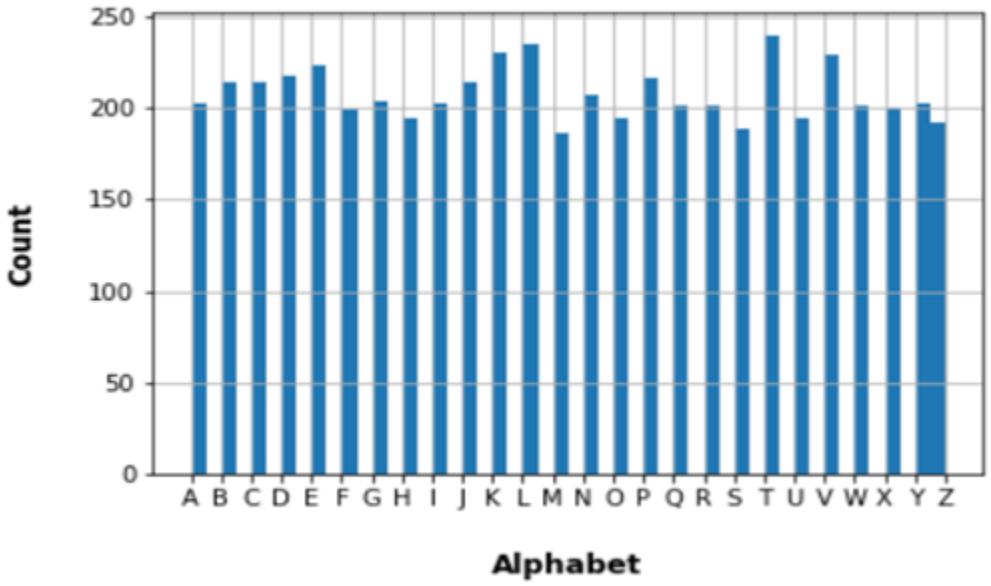
```
(4326, 80, 80, 1)
```

Similarly, the shape of the test dataset is

```
: X_test.shape

: (1082, 80, 80, 1)
```

Count of Alphabet Letters from the total dataset. We can see that there is a uniform distribution of the Alphabet letters.



Distribution of alphabet images in ASL Library

One of the challenges that we faced while training our model was that we needed more data for the individual alphabets as many alphabets were looking similar to each other, since there are 26 distinct alphabet characters so more data is needed to train the model . To solve that issue, we added more images to each of our alphabet folders which we had labelled with the alphabet character. We added our own images, images from Kaggle and google images.
Post the dataset creation we downloaded our ASL library dataset to our system and converted each of the images to a numpy array along with its label. So, for example, an image in the folder A was converted to an array, was appended into the "image_arrays" list and simultaneously had its respective letter appended to the list "labels".

METHODOLOGY:

The image library was imported to local computers to be analyzed and for model building. Because we would eventually be training a model on hand symbols with their respective letter, this was a supervised learning experiment where we would need labeled data. Labeling each of the image folders with their respective letter would help with this task. Utilizing the Jupyter Notebook Environment (Python) and libraries such as *Glob, Numpy, and CV2,* we imported the library into

the notebook and converted each of the images to a numpy array with its label. So for example, an image in the folder A was converted to an array, was appended into the "image_arrays" list and simultaneously had its respective letter appended to the list "labels".

```
#working import/array code
#WORKING CODE 11/13
pic_lib = r'C:\Users\nchin\ASL_Project\ASL_Library'

letter_list = []

for item in os.listdir(pic_lib):
    letter_list.append(item)

extensions = ("*.jpg", "*.jpeg")

image_arrays = []
labels = []

for letter in letter_list:
    for ext in extensions:
        for index, pic in enumerate(glob.glob(pic_lib + "\\" + letter + "\\" + ext), start = 1):

            #file name to label
            labels.append(letter)

            #split file path and file name
            path, filename = os.path.split(pic)

            #New array Name
            newFileName = str(letter)+str(index)

            img = cv2.imread(os.path.join(pic_lib, pic), 0)

            new_img_array = cv2.resize(img, dsize=(80, 80))
            image_arrays.append(new_img_array)
```

Screenshot of Processing the images and adding labels

The code above was used to create these two corresponding lists of arrays that had been resized to an 80 x 80 pixel image. The images were also converted into a grayscale image for storage and processing purposes. Using a stratified split of the labelled variables, the data was split into a 80:20 train test ratio. Later on when we added more images and had a similar number of images for each letter, this step was much more necessary to ensure that the model was training and testing on an equal distribution of data to the original data set. The X value arrays that represent the pixel colors were then normalized to a 0 to 1 scale because of the optimizers (Gradient Descent, Adam) that we use in the model. The values are all divided by 255 and can help to improve the convergence speed of the algorithm.

Model Complexity and Hyperparameters

The complexity of data is objective but is also difficult to quantify. For example, images of hands performing sign language is probably more complex than the *MNIST* dataset but is less complex than the images that a self-driving car is processing. As has been documented with many machine learning techniques, much of the model design can be very ad-hoc, requiring specific modifications to a model based on the data but we tried to be as systematic as possible. The approach we took was to start with a more complex model in terms of the number of convolutions and filters, choice of optimizer, number and size hidden layers, and incrementally decreasing some of those values

(if necessary) to find the right compromise between overfitting and underfitting to the data. A 3 x 3 filter size was used as standard throughout the filter layers. It seemed to gather enough detail from each image while not increasing the parameter number too high. We used the relu activation function for the convolutional and dense hidden layers because it is computationally efficient and allows the network to converge very quickly. The output dense layer used the softmax activation function because it can handle multiple classes well and normalized the outputs for each class between 0 and 1 and divides by their sum, giving the probability of the input value being in a specific class. For the scope of this project, we created a Machine Learning algorithm to be able to identify each of the letters in an efficient and effective manner. Due to their proficiency with image classification and some preliminary testing, we used convolutional neural nets (CNN). During some preliminary testing, machine learning classifier models (MLP) and CNNs were run side by side on the same dataset with similar levels of complexity and the CNNs outperformed the MLPs by over 15% in most scenarios.

Initial Models and Choosing the Right One

Gradient Descent was effective as an optimizer during some initial testing but could be painfully slow especially when running on our local computers/laptops. For the remainder of the models, we used *Adam* (adaptive moment estimation). The typical structure of a CNN includes a layer (or layers) of filter maps with the number of convolutional layers increasing (and oftentimes doubling) after each layer. This has been proven to be an effective method to acquire details and then subsequently put those details together to make and identify more structured "images". To avoid overfitting and looking at other CNN models, we did not want to start with too high of a number of filters and started with 32. The image below details one of the first CNN models that we tried.

```
In [92]:
m = "model1"
m = Sequential()

m.add(Conv2D(32, (3, 3), input_shape=(80,80,1)))
m.add(Activation('relu'))

m.add(Conv2D(64, (3, 3)))
m.add(Activation('relu'))
m.add(Conv2D(64, (3, 3)))
m.add(Activation('relu'))
m.add(MaxPooling2D(pool_size=(2, 2)))


m.add(Conv2D(128, (3, 3)))
m.add(Activation('relu'))
m.add(Conv2D(128, (3, 3)))
m.add(Activation('relu'))
m.add(Conv2D(128, (3, 3)))
m.add(Activation('relu'))
m.add(Conv2D(128, (3, 3)))
m.add(Activation('relu'))
m.add(MaxPooling2D(pool_size=(2, 2)))
```

Screenshot of CNN Model 1

Model 3

```
model3= Sequential()

model3.add(Conv2D(64, (3, 3), input_shape= X_train.shape[1:]))
model3.add(Activation('relu'))

model3.add(Conv2D(64, (3, 3)))
model3.add(Activation('relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))

model3.add(Flatten())
model3.add(Dense(500))

model3.add(Dense(350))
model3.add(Activation('relu'))

model3.add(Dense(200))
model3.add(Activation('relu'))
model3.add(Dropout(rate=0.5))

model3.add(Dense(26))
model3.add(Activation('sigmoid'))

model3.compile(loss="categorical_crossentropy",
               optimizer="adam",
               metrics=['accuracy'])

result3 = model3.fit(X_train,Y_train, batch_size=300, epochs=15, validation_steps=1, validation_split = 0.2)
```

Screenshot of CNN Model 3(Best Model)

The above structure doubled the number of layers and filters at each of the following convolutional layers. We also tested the use of hidden layers before the classification output layer. Current methodologies recommended taking advantage of more hidden layers with less neurons. After stacking convolutional layers, we implemented a hidden layer at a time with 200 neurons in each hidden layer and adjusted after each run. Our most complicated model involved a structure similar to the one shown above with 3 hidden layers. Our simplest model decreased the first layer's filter count to 20 and had one more layer with two 40 layer filters and no hidden layers. Once a narrower range of parameters were determined, we then changed ones that typically make smaller changes like learning rate, and batch size. Early stopping was also used to ensure that models weren't being overfit to the data. After slowly converging on a model, it turned out that the factor that made a huge difference was the addition of using *Dropout* as a regularization technique to reduce overfitting.

RESULTS:

In a recognition system, the accuracy rate of the recognition is a great concern. The classification report shows that our model is able to perform classification with a high accuracy 98.03%.
From 5408 images 4326 images were taken for training and 1082 images were taken for testing. The table below shows us the accuracy rate for each alphabet. Precision is the ratio of true positives to sum of true positives and false positives. Recall is the ratio of true positives to sum of true positives and false positives. F1- score represents harmonic mean of recall and precision. Support is the number of occurrences in each class.

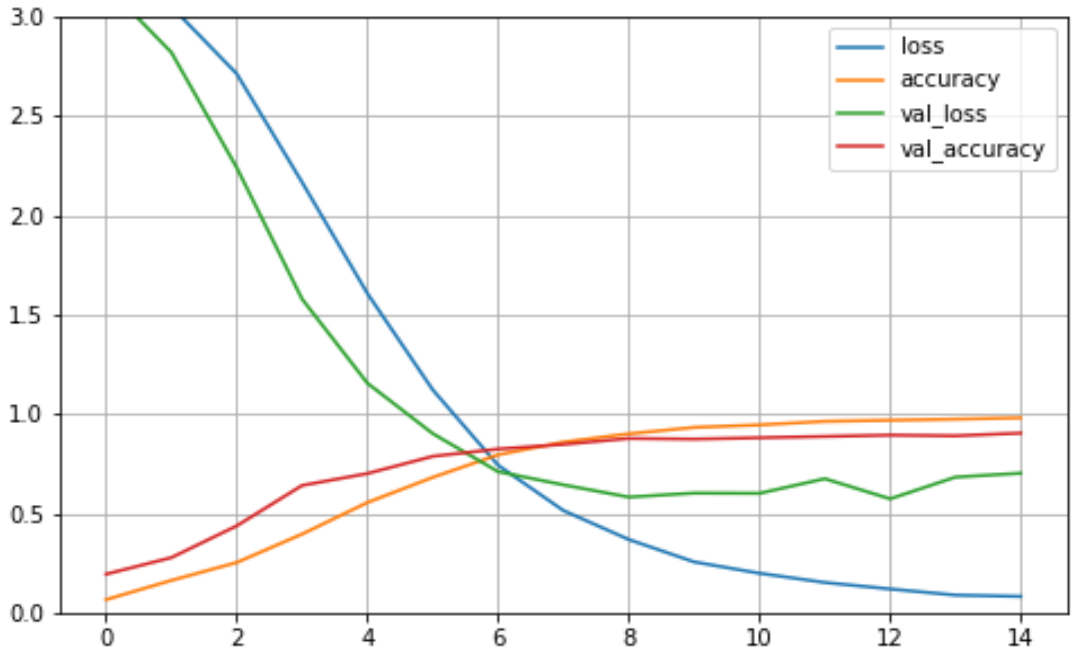
```
Epoch 1/15
12/12 [==============================] - 53s 4s/step - loss: 3.3697 - accuracy: 0.0650 - val_loss: 3.1578 - val_accuracy: 0.193
3
Epoch 2/15
12/12 [==============================] - 55s 5s/step - loss: 3.0511 - accuracy: 0.1618 - val_loss: 2.8201 - val_accuracy: 0.276
7
Epoch 3/15
12/12 [==============================] - 56s 5s/step - loss: 2.7125 - accuracy: 0.2520 - val_loss: 2.2400 - val_accuracy: 0.436
7
Epoch 4/15
12/12 [==============================] - 57s 5s/step - loss: 2.1663 - accuracy: 0.3951 - val_loss: 1.5786 - val_accuracy: 0.640
0
Epoch 5/15
12/12 [==============================] - 56s 5s/step - loss: 1.6064 - accuracy: 0.5540 - val_loss: 1.1547 - val_accuracy: 0.700
0
Epoch 6/15
12/12 [==============================] - 56s 5s/step - loss: 1.1213 - accuracy: 0.6806 - val_loss: 0.9011 - val_accuracy: 0.786
7
Epoch 7/15
12/12 [==============================] - 57s 5s/step - loss: 0.7416 - accuracy: 0.7945 - val_loss: 0.7093 - val_accuracy: 0.823
3
Epoch 8/15
12/12 [==============================] - 58s 5s/step - loss: 0.5135 - accuracy: 0.8584 - val_loss: 0.6428 - val_accuracy: 0.846
7
Epoch 9/15
12/12 [==============================] - 57s 5s/step - loss: 0.3678 - accuracy: 0.9000 - val_loss: 0.5815 - val_accuracy: 0.876
7
Epoch 10/15
12/12 [==============================] - 57s 5s/step - loss: 0.2549 - accuracy: 0.9321 - val_loss: 0.6013 - val_accuracy: 0.873
3
Epoch 11/15
12/12 [==============================] - 59s 5s/step - loss: 0.1978 - accuracy: 0.9448 - val_loss: 0.6003 - val_accuracy: 0.880
0
Epoch 12/15
12/12 [==============================] - 57s 5s/step - loss: 0.1513 - accuracy: 0.9627 - val_loss: 0.6740 - val_accuracy: 0.886
7
Epoch 13/15
12/12 [==============================] - 60s 5s/step - loss: 0.1187 - accuracy: 0.9682 - val_loss: 0.5725 - val_accuracy: 0.893
3
Epoch 14/15
12/12 [==============================] - 59s 5s/step - loss: 0.0879 - accuracy: 0.9737 - val_loss: 0.6818 - val_accuracy: 0.890
0
Epoch 15/15
12/12 [==============================] - 60s 5s/step - loss: 0.0814 - accuracy: 0.9803 - val_loss: 0.7016 - val_accuracy: 0.903
3
```

Screenshot of CNN Model 3 Accuracy & Loss

	precision	recall	f1-score	support
A	0.79	1.00	0.88	34
B	0.79	0.63	0.70	43
C	0.88	0.84	0.86	51
D	1.00	1.00	1.00	47
E	0.73	0.60	0.66	40
F	1.00	0.97	0.99	35
G	0.83	0.95	0.89	37
H	0.93	0.88	0.90	32
I	0.81	0.61	0.69	41
J	0.86	0.92	0.89	61
K	0.80	0.90	0.85	52
L	0.96	0.83	0.89	54
M	0.97	0.79	0.87	39
N	0.73	0.86	0.79	37
O	0.88	0.91	0.89	32
P	0.97	1.00	0.99	37
Q	0.93	1.00	0.96	41
R	0.84	0.96	0.90	45
S	0.95	1.00	0.98	42
T	1.00	0.91	0.95	46
U	1.00	0.89	0.94	28
V	0.94	0.96	0.95	53
W	0.95	1.00	0.97	38
X	1.00	0.95	0.97	41
Y	0.89	1.00	0.94	31
Z	0.96	1.00	0.98	45
accuracy			0.90	1082
macro avg	0.90	0.90	0.90	1082
weighted avg	0.90	0.90	0.90	1082

Screenshot of Classification Report for CNN Model 3

The model was trained for a dataset size of 4326 images. The model was trained for 15 epochs. The initial training accuracy and validation accuracy increases drastically till the 8th epoch. The accuracy then attains a plateau limit as the epochs increase. The maximum validation accuracy attained is 90.33% at the 15th epoch. Whereas the maximum training accuracy attained is 98.03%.



Screenshot of CNN Model 3 Accuracy & Loss Curve

Due to the high architectural complexity and parametric variation of a CNN model, it is not feasible to evaluate all possible architectures and associated parameters (e.g., number of convolutional layers, kernel size, activation function, pooling method, etc.). In this project, a few representative CNN models with increasing numbers of layers and different parameters are compared to find the optimal model. As shown, the CNN model 3 is selected and their performance evaluations are compared.

Our results suggest that the CNN Model 3 is selected as the best prediction model among other models in terms of bias, variance with its computational efficiency and accuracy in interpretability.

CONCLUSION:

Overall, the project was able to meet our expectations quite well. Our initial goal was to create a system capable of recognizing and classifying ASL and we were able to do so with 98% training accuracy and 90% validation accuracy.

Though sign language can be implemented to communicate, the target person must have an idea of sign language which is not always possible. Hence, our project lowers this barrier, with our system the people who have hearing impairment will be able to communicate naturally with everyone around regardless of whether the other person knows the sign language or not.

Due to their proficiency with image classification and some preliminary testing we used convolutional neural nets (CNN) algorithms and the model recognized the sign languages with a high test accuracy of 90%.

The only noticeable issue was with the quality of the dataset exemplified in the course of training the model on the dataset of good quality as we created our own image dataset. However, this was solved by creating more data by group members and adding some from Google images and other sites like Kaggle as well as duplicating the dataset.

While there may be many applications to sign language systems, we chose to specifically tailor this project towards the deaf community. We do not present an ultimate solution, but we hope to show that it is possible to build simple and efficient systems to help bridge the gap between communities that share their thoughts through different means.

We managed to build the baseline model for identifying the alphabets at present but would like to improve it furthermore so that it can be used in conjunction with head and eye movements, and when all this is perfected, we shall release all the data and code under an open-source license for the research community

Future directions to further develop this model are as follows:

- In the current work, due to the limited time frame of the semester, the model was limited to tackle static signs. But the majority of the communications semantic content lies in movement, facial expression and stress placed on the sign. So, we are looking forward to widen the scope of the project for the development of a complete system which could be used as an effective tool for the disabled to communicate, in which all elements of sign language would be used
- We are also looking at the possibility of generating complete sentences from language and context models. This might be done using HMMs
- Also, we are looking to port this system to run on a portable embedded device that could be taken by a disabled person and used anywhere for interpretations
- And at the end we intend to looking into upgrading the system so it can accommodate the possibility where speech can be recorded and converted to sign language

REFERENCES:

Mitchell, R. E. (2006, November 6). How Many People Use ASL in the United States? Why Estimates Need Updating. Retrieved November 18, 2020, from https://www.gallaudet.edu/documents/Research-Support-and-International-Affairs/ASL_Users.pdf

Waterfield, S. (2019, April 15). ASL Day 2019: Everything You Need To Know About American Sign Language. Newsweek. https://www.newsweek.com/asl-day-2019-american-sign-language-1394695#:%7E:text=Today%2C%20around%20one%20million%20people,Communication%20Service%20for%20the%20Deaf.

P. (2018, October 26). Meet the new Google translator: An AI app that converts sign language into text, speech. The Economic Times. https://economictimes.indiatimes.com/magazines/panache/meet-the-new-google-translator-an-ai-app-that-converts-sign-language-into-text-speech/articleshow/66379450.cms

C. (2018a, November 12). How we used AI to translate sign language in real time. Medium. https://medium.com/@coviu/how-we-used-ai-to-translate-sign-language-in-real-time-782238ed6bf

C. (2018a, November 12). How we used AI to translate sign language in real time. Medium. https://medium.com/@coviu/how-we-used-ai-to-translate-sign-language-in-real-time-782238ed6bf