Indian Gesture Recognition

- Kausik Lakkaraju

Background:

In India, there are more that 63 million people who suffer from significant hearing loss. According to a study by WHO in 2017, 5 million children in India were suffering from hearing and speech impairment [1]. Using implants is not the only solution for hearing loss. Not everyone can afford installing these implants. Moreover, these implants may lead to loss of remaining hearing ability in the implanted ear. After implantation surgery, if there's any infection near the ear, it will lead to serious complications [2]. Also, there are not many people who could translate sign language to regular speech. Another alternative for the impaired person to write the message he/she likes to convey which is cumbersome and difficult to do when there is an emergency. Hence, there is a need to come up with a solution which is easy to implement by a common user (who does not have much technical knowledge) and cost-effective.

Problem:

A lot of research has been happening in the field of AI. Neural Networks are helping people to solve complex problems in real life situations. They can learn and model relations between inputs and outputs that are non-linear and complex. They also help in revealing hidden relations. They are being widely used for fraud detection, voice recognition, disease

diagnosis, robotic control systems and in computer vision to interpret raw images and photos [3].

For image classification, Convolution Neural Networks (CNN) are the most widely used form of Neural Networks for classification problems. However, there are few problems with training CNN on gesture classification.

- We need a dataset for training our model. Though, American Sign Language (ASL) dataset is readily available (On websites like Kaggle), you cannot find many on Indian Sign Language.
- If we want to classify all the letters of Indian Sign Language, we must train the model over 26 classes with around 1000 images in each class. To train the model on such large number of images, it will take a lot of time and memory.
- If we train our model such that it can recognize only my hand, others who want to use this project will have a different skin tone or different lighting.

I tried addressing all these problems with my project.

Related Work:

A lot of research has been done in this area before. One of the recent papers [4] took an approach like mine. However, they used edge detection after skin color detection. Another recent paper used Microsoft Kinect to capture the live image and performed segmentation using K-means Clustering. They used SVM for classification. They trained the model on 138 gestures [5].

Most of the research approaches used Microsoft Kinect as it is used to perform high quality 3D scans of small or larger objects. It can track the object in 3D space.

Data Sources:

We don't need an external dataset for this project. It can be created by us easily. I wrote the code for capturing 1200 snaps from a single gesture. If the lighting is uniform, it will capture almost 1 snap per second. We can flip the images after creating them so that the gesture can be recognized even when camera captures the live image in flipped manner.

Approach:

My main motivation for this project is a paper which was published in 2018 [6]. This project is partly based on that and some other references like [7] where similar approach was used.

To train the model on large dataset, it is better to preprocess the images before training. OpenCV is a very well-known library which is used in the field of Computer Vision. I used that to preprocess the images before training them. For training the model I used CNNs.

My project can be split into 8 parts:

1) Setting hand histogram:

In this step, the camera(live) captures your hand histogram. Histogram can be considered as a graph or plot which gives overall idea about intensity distribution of the image. See fig.1. You can see that the darker pixels are at both the ends. In the middle region, it is lighter. When we place our hand in the highlighted area in the camera frame and press a button, it will convert the highlighted part of the image to HSV (Hue, Saturation, and Value) and its histogram will be calculated. Then we normalize the pixels (Change range of pixel intensity values). Image normalization is done to increase the contrast which aids in improved feature extraction or image segmentation. Then it will calculate back projection. If we are trying to get the histogram of our hand, back projection method can be used to find skin colored parts in the

image. For Back Projection, you calculate the histogram model of a feature and then use it to find this feature in an image. I removed the Gaussian and Salt-and-Pepper noise from the image and applied thresholding. I applied binary thresholding with Otsu to convert the pixels to 0 and 1 and to remove the remaining noise. After we see our hand correctly identified in the frame, we can press another button which will save this histogram.

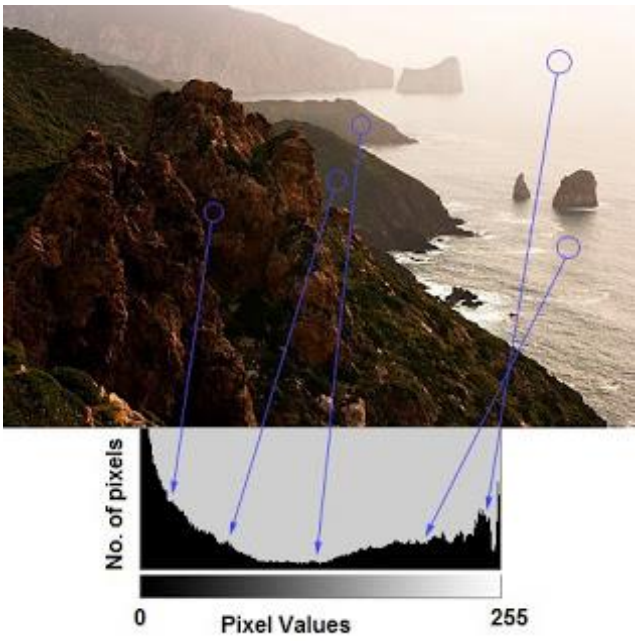


Fig.1: Histogram of an image (Taken from OpenCV documentation)

2) Creating gestures:

As mentioned before, this project needs no external data sources. We can create our own gestures in no time provided we save the histogram correctly and the lighting is appropriate. We make use of sqlite3 which can be used to create database and execute sql commands. This program accesses our saved histogram so that it can recognize our hand correctly. I created a .db file (database) for storing different gesture ids and gesture names. This file can be opened using DB browser for sqlite. We create a

folder called gestures for storing our data. We can give gesture Id and name. It will start capturing the images of our hand. The preprocessing is similar to that of hand histogram. We access the saved histogram, calculate back projection and remove noise. We find Contours. Contours are curves joining the continuous points along the boundary having same color or intensity. We define a contour retrieval method as well. In this program, I used RETR_TREE. CHAIN_APPROX_NONE is Contour approximation method which takes all boundary points. This program takes 1200 snaps of our gesture. This will be split into training, testing and validation sets later.

3) Flipping Images:

This is just a simple part which accesses our gestures folder and its contents. It will take each image and flip it and save it as a separate image. At the end we will have 2400 images in each gesture. We flip the images so that the model will be more robust and can recognize the image even when it is in different orientation. We can perform other rotations and translations as well. This property of CNN is called invariance.

4) Load Gestures:

We convert the images to numpy array and append this array with the image label to a list. We shuffle these images so that same kind of images (like same orientation) are not together. We access each image and label separately and store it in different variables. We split these images into training set, testing set and validation set in ratios 80%, 10% and 10% respectively (I have tried different proportions and it did not seem right. These proportions are like standard).

5) Training the model:

This is the core part of the project where we train our CNN model to classify different gestures. We build a sequential neural network using Keras. CNNs are well known for extracting appropriate features from an image and classifying different groups of images. They are even being used in sentiment analysis now. There are many types of architectures of CNN like AlexNet, VGG-16, etc. I used 16 filters in the 1st layer, 32 in 2nd, 64 in 3rd and 128 for final dense layer. Filters capture patterns. In 1st layer, we capture edges or corners, In the next layers we combine these patterns to make bigger patterns. As we keep increasing the filter size, it will capture many combinations. Higher the number of filters, higher the number of abstractions that our network will be able to extract features from images. At each convolution layer, we add an activation function. I used ReLU for this. The activation function determines whether each neuron in the network should be activated or not. ReLU does a great job at introducing the non-linearity. We use non-linear activation function so that backpropagation is possible (Derivative can be calculated). Whereas it is not possible to backpropagate if linear activation function is used as they cannot be derived. We apply a Pooling layer after each Convolution layer. Convolution layers record precise position of the features. This will become a problem when the same kind of image is either tilted or flipped. We use Pooling so that it can be downsampled by changing stride of the convolution across the image (Stride can help the filter move in other directions by certain value). I used maxpooling here which takes the maximum value in each patch of feature map. Convolution layers are just used for feature extraction. Fully connected layers are responsible for classifying the images at the end. They are also called dense layers. At the end we use a softmax activation function which will give us the probabilities and classifies the images accordingly. We later use this to recognize the actual gesture in the next part. We convert each

of the train and valid images to numpy arrays and train and fit the model. I have used the Tensorboard library so that our network and its performance can be examined after training. We also calculate the error at the end by using evaluate() function from Keras library. We save the model in .h5 format so that when we train again, the model will monitor the validation set accuracy. We can even change it so that it will monitor the training accuracy instead. I trained the model for 100 epochs with batch size of 500.

6) Recognizing the gesture:

We access the saved hand histogram and our saved keras model. We follow same image processing techniques that we used in histogram part. We predict the label of the preprocessed image using the predict function from Keras. Our model will recognize gestures with different probabilities. We want to pick one with more probability. I gave that value as 80% here. We can set it however we want to. Our model will predict the class number that is gesture id. We use this gesture id to get the gesture name from our database file. We display the text on a black space which can be created by populating a frame with zeros. The gesture that we display in from of the camera will be recognized by the model and the corresponding text will be displayed on the black board.

7) Getting model reports:

We plot the confusion matrix on a graph. To plot this, we use test set images and labels. Average prediction time will also be calculated. I will discuss this in detail in the evaluation part. This method was described in [8].

8)Displaying the gestures:

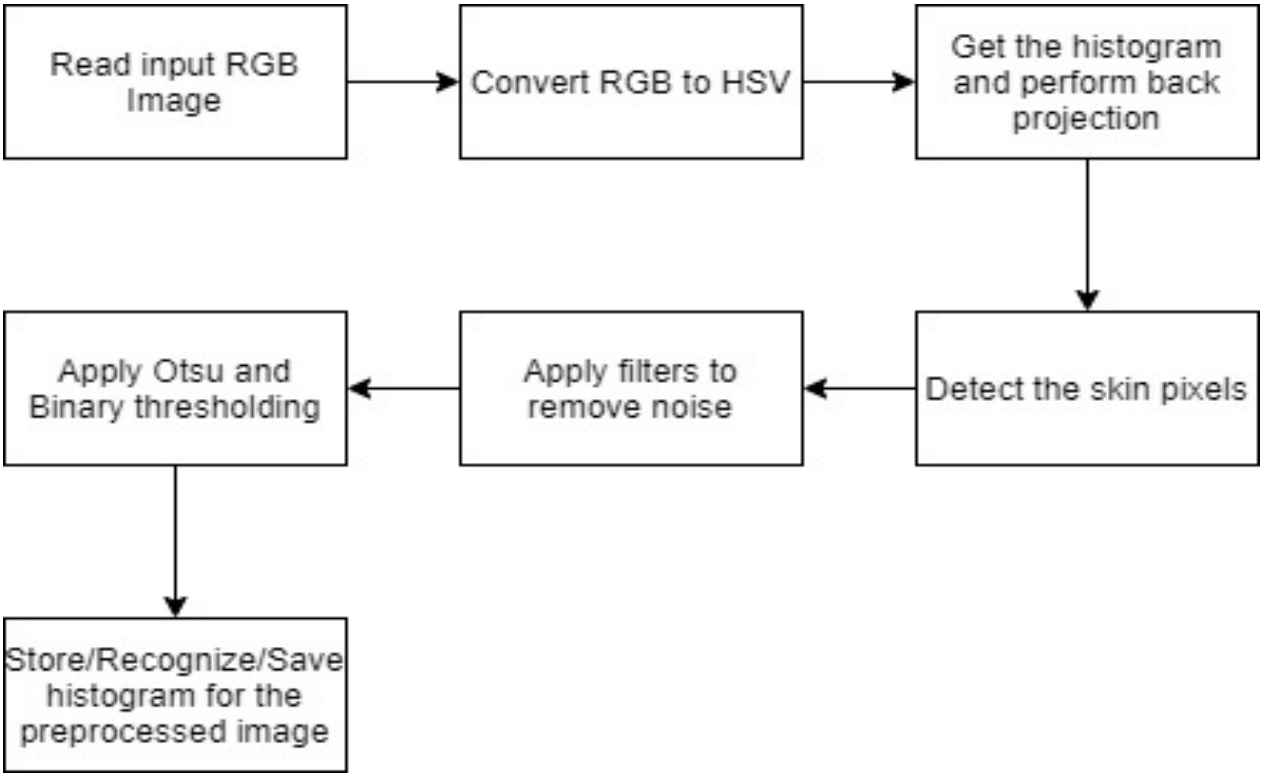
I have trained the model with 29 gestures. You can add your gestures and train it. You can display all the gestures that were you used by using

the code display_all_gestures.py. It will automatically save the image which consists of all the gestures later.

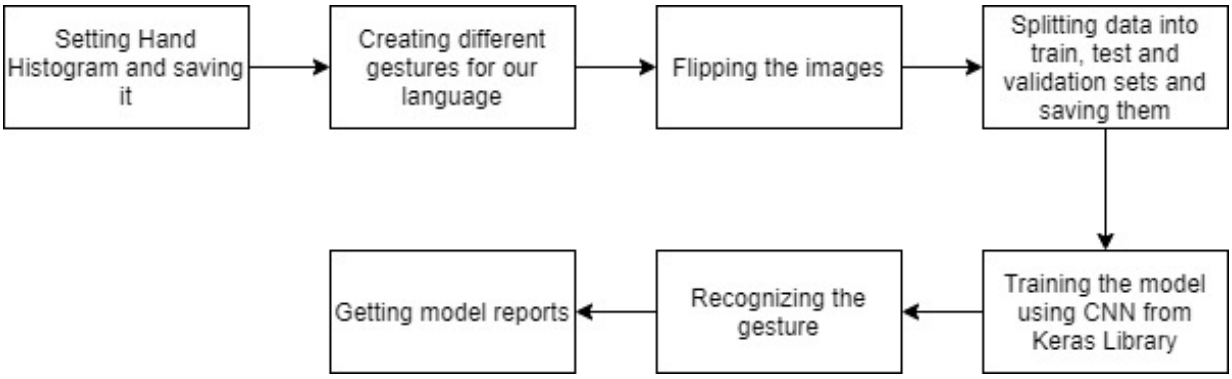
Block Diagrams:

I used draw.io to draw the block diagram of the whole process. This will be like the summary of the approach that I have explained above.

1) The preprocessing phase can be represented this way:



2) The representation of whole procedure:



Demonstration:

Here I will give a practical demonstration with the help of few screenshots. As I mentioned before this project consists of 8 parts (8 files). I will give the demonstration for creating gestures and training it from scratch and also test the already stored gesture and finally get the model reports and confusion matrix for the test set. I will also demonstrate the way to get additional details and the network structure using Tensorboard.

A. Creating additional gestures and training the model:
   Re-train the model only if you add new gestures or else you can directly go to the B part.

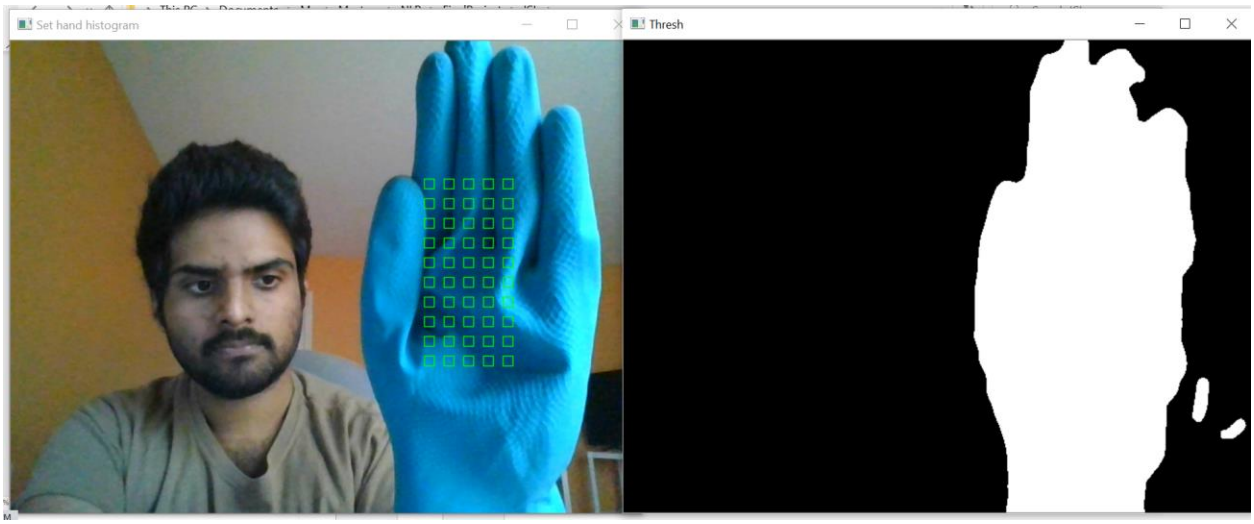
i. Download the necessary requirements by creating a pip environment and installing requirements from the txt file that I have provided in the project directory by using the command:

```
pip install -r requirements.txt
```

ii. Set hand histogram using the command:

```
python set_hand_hist.py
```

You will be able to see a camera frame opened. Place your hand in the region where you can see green boxes. Make sure your hand covers all the boxes. I used green gloves so that the gesture can be easily recognized. If we use the skin, some noise maybe added depending on the tone. It is optimal to use gloves and optional, of course. When the frame opens, Press 'C' so another frame with other pixels masked will be displayed. Make sure you are able to see your hand clearly marked in the other frame and then press 'S' to save that histogram.

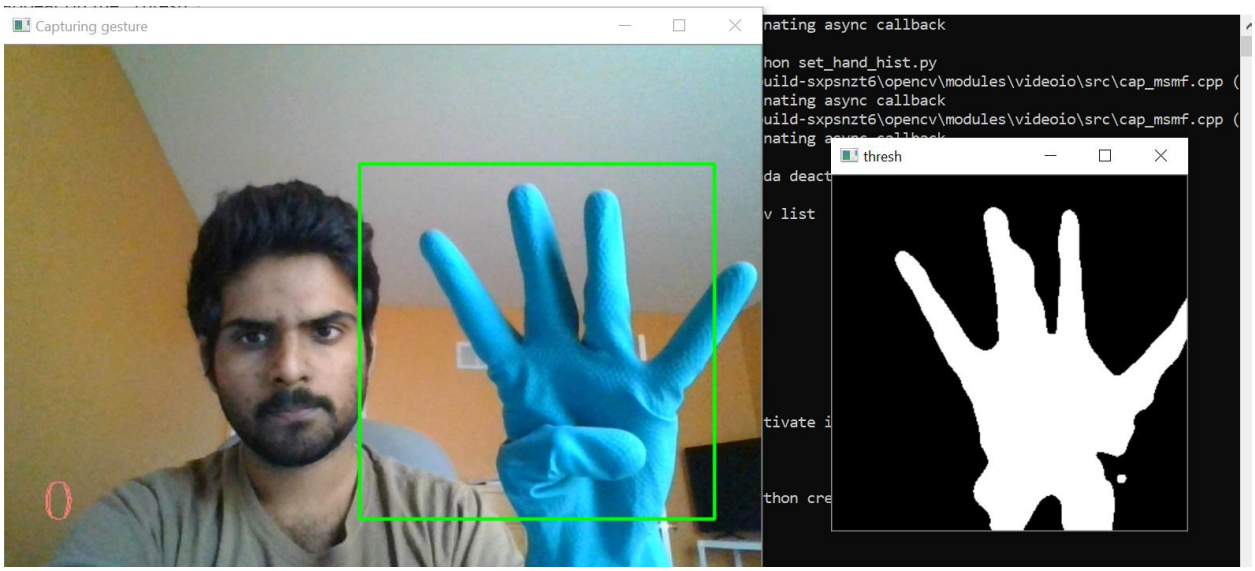


iii. If you want to create your own gesture other than the ones I created, you can use create_gestures.py file by typing the command:

```
python create_gestures.py
```

When you run this command, it will ask you to type a gesture id and the name of that gesture. I have created gestures from 0-

28. So, you can start from 29. Place your hand in the green box and when you are ready, press ‘C’. After few seconds the camera starts capturing images. It will roughly capture 1 image per second like that 1200 images will be captured. You will be able to see the counter while it’s capturing the images.



iv. After creating the gestures, flip all the images by entering the following command:

```
python flip_images.py
```

v. Split the images into train, test and validation sets and load them using the command

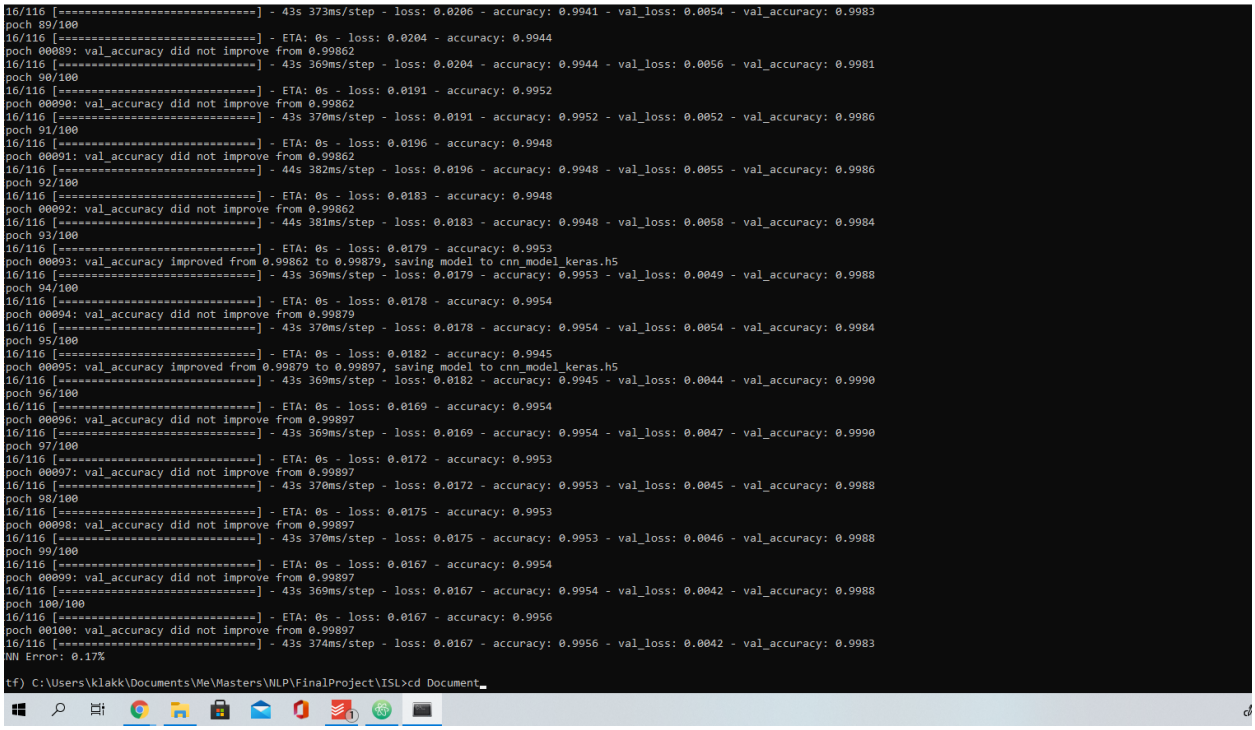
```
python load_images.py
```

vi. To see all the gestures stored in the ‘gestures’ folder,

```
python display_all_gestures.py
```

vii. To train the model,

```
python cnn_keras.py
```



B. Testing and Recognizing the gestures:

i. The test set that we created can be used to get the model reports. To get those you can use the following command:

`python get_model_reports.py`

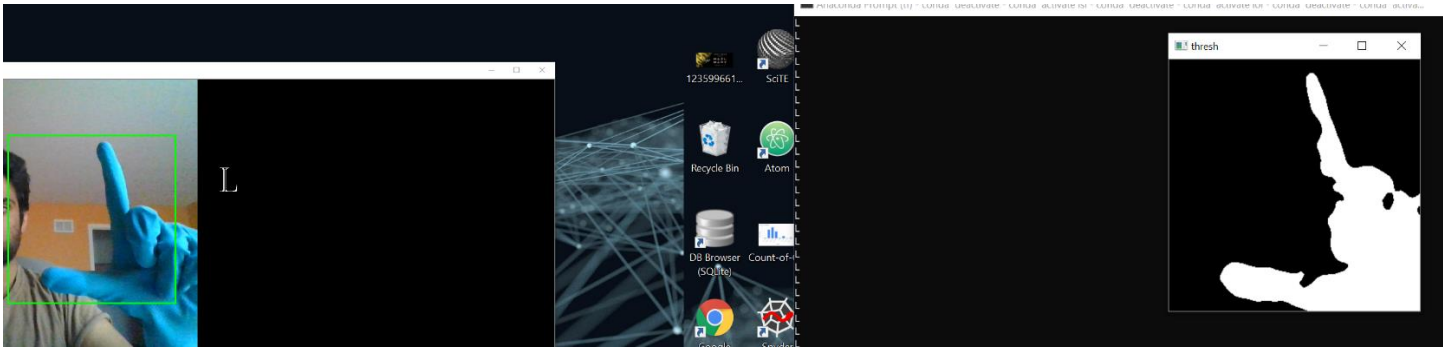
You will get f1 score, precision, recall and the confusion matrix will be saved as an image file in our main project directory.

ii. If your lighting or glove color changes, you can set histogram again. You can even use your hand. To set histogram follow pt. ii from Part-A.

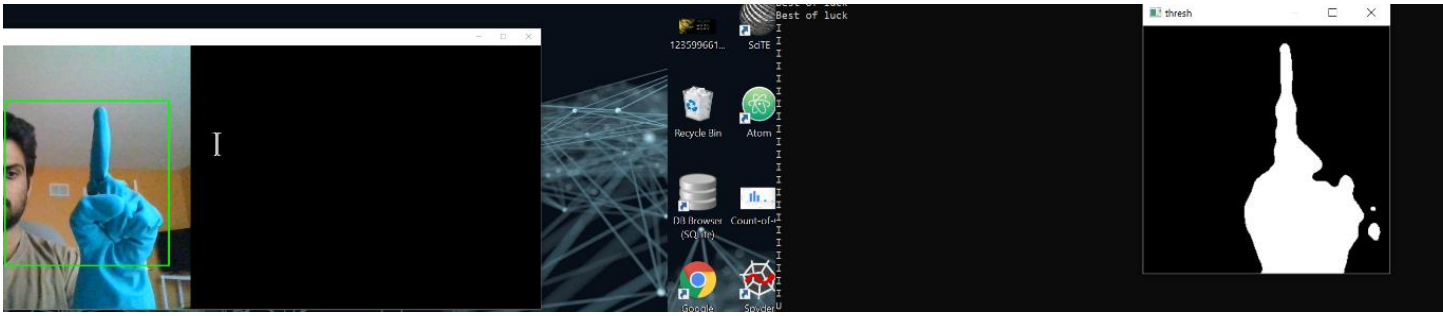
iii. Now to recognize the gesture run this command:

`python recognize_gesture.py`

Place your hand inside the green box for that gesture to be recognized. Refer to Fig. 2 on page 15.

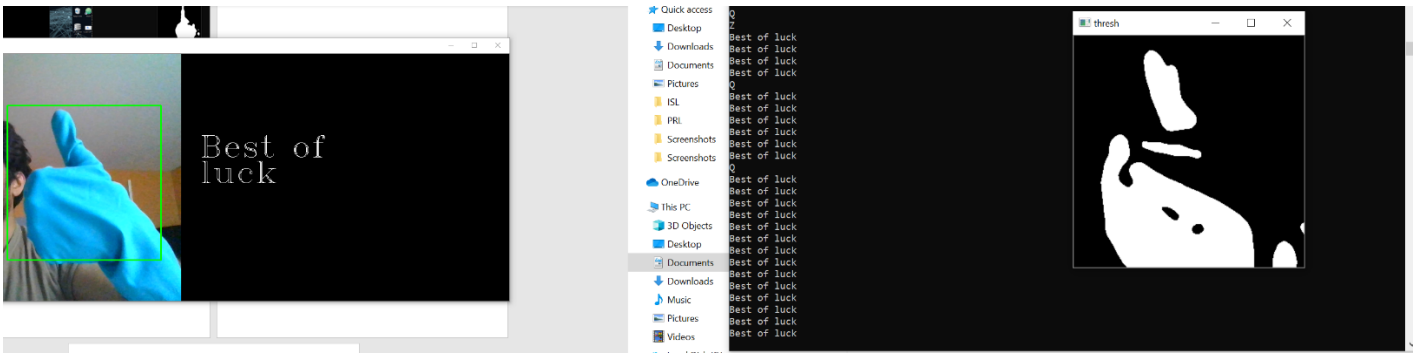


Letter ‘L’ being recognized by the model



Letter ‘I’ being recognized by the model

The word ‘Namaste’ being recognized by the model(I had to take my hand off in the last minute to capture a screenshot).You can see it displayed in the terminal when it was recognized.



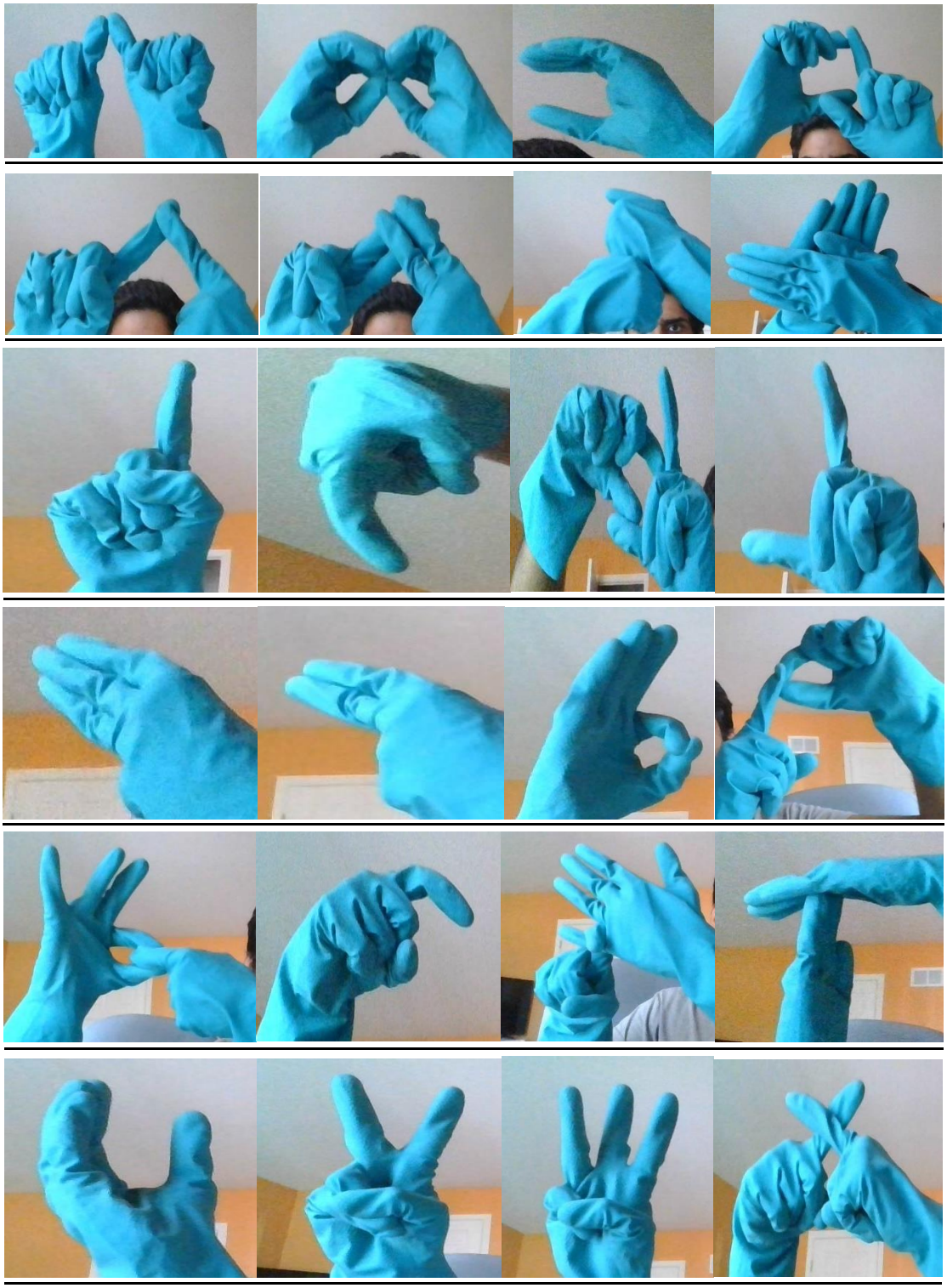
The sentence ‘Best of Luck’ being recognized by the model.

C. Get Graphs from TensorBoard:
Run the command,
tensorboard --logdir="logs"

and navigate to the url it gives you (like http://127.0.0.1:6006)

When I was training the model, I already specified the TensorBoard logs to be stored in a directory called “logs” in the project directory.

Gestures



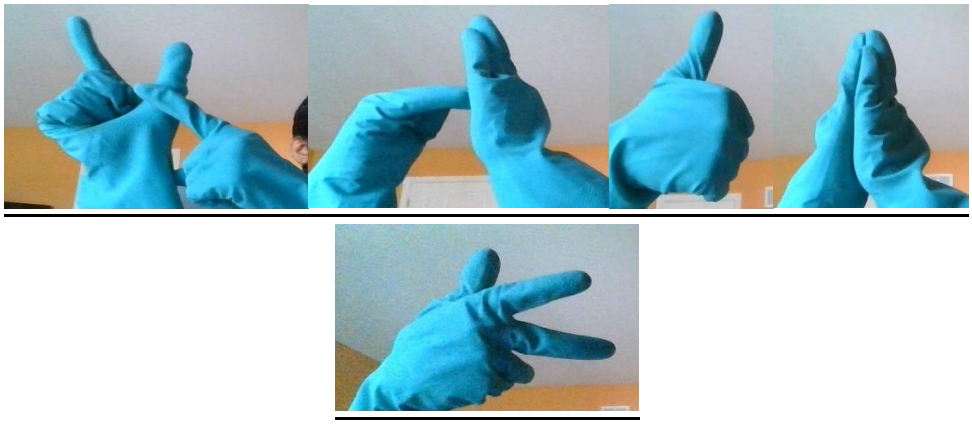


Fig.2. Order of Gestures (Left to Right):
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
ALL THE BEST, NAMASTE, WHAT.

Evaluation:

I have used many evaluation metrics for this project. Though the numbers look very good here, there were some issues that I encountered which I will specify in the next section.

1)From model reports:

As I mentioned before model reports give us the F1 score, Precision, Recall and Support values. We can also obtain the confusion matrix from test data.

Confusion Matrix: It consists of True Negative (TN) (Actual value is False and predicted as False), False Positive (FP) (Actual value is False but predicted as True), False Negative (FN) (Actual value is True but predicted as False), True Positive (TP) (Actual value is True and predicted as True).

Precision = TP / (TP + FP)

When model predicts as True, how often is it actually correct?

Recall = TP / (TP + FN)

When it is actually True, how often does the model predict correctly?

F1 = 2 * (precision * recall / (precision + recall))

Weighted average between precision and recall. Useful in dealing with unbalanced samples.

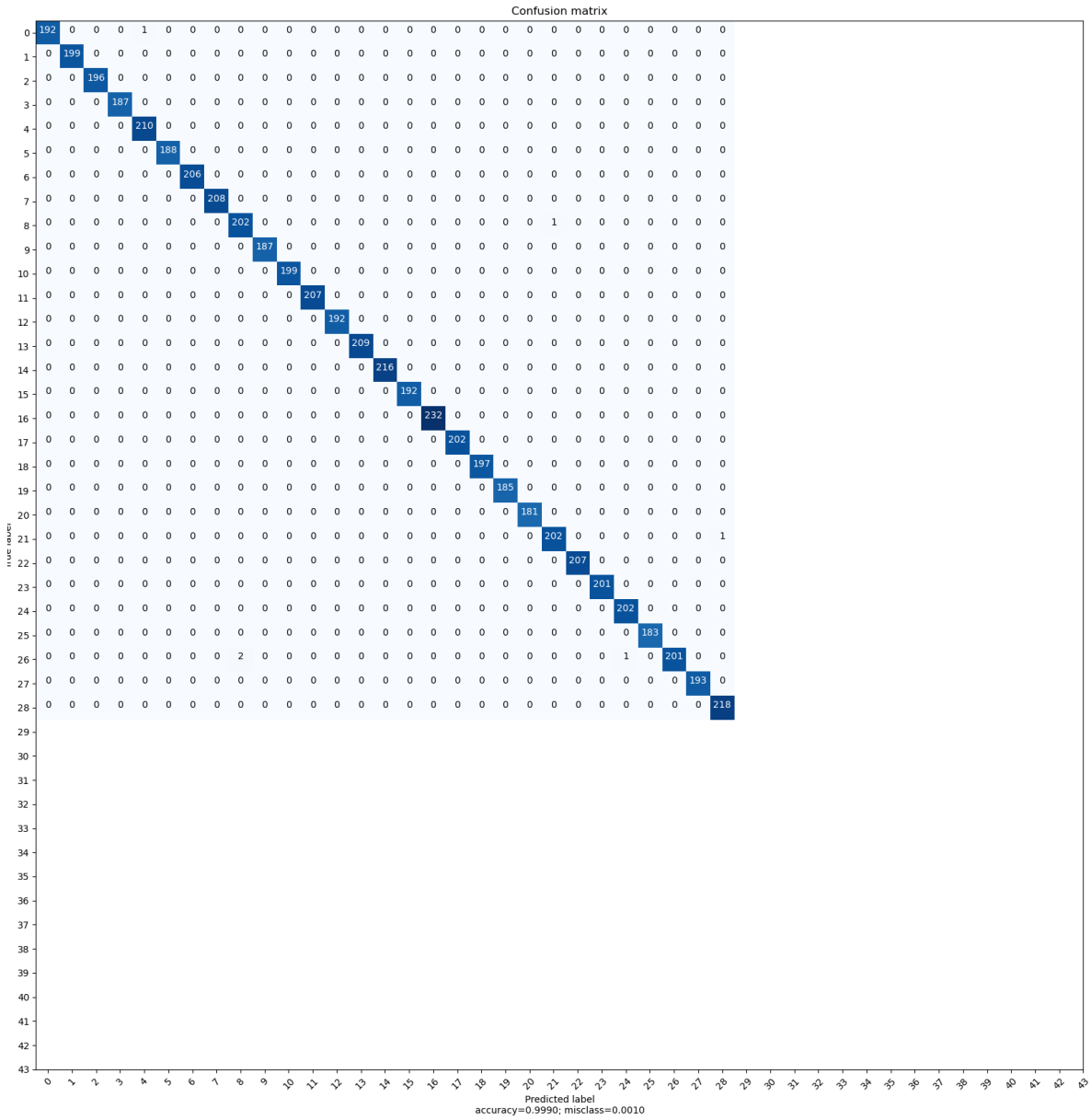
Support is the number of occurrences of each class in true targets.

For more information refer to [9].

Classification Report

	precision	recall	f1-score	support
0	1.00	0.99	1.00	193
1	1.00	1.00	1.00	199
2	1.00	1.00	1.00	196
3	1.00	1.00	1.00	187
4	1.00	1.00	1.00	210
5	1.00	1.00	1.00	188
6	1.00	1.00	1.00	206
7	1.00	1.00	1.00	208
8	0.99	1.00	0.99	203
9	1.00	1.00	1.00	187
10	1.00	1.00	1.00	199
11	1.00	1.00	1.00	207
12	1.00	1.00	1.00	192
13	1.00	1.00	1.00	209
14	1.00	1.00	1.00	216
15	1.00	1.00	1.00	192
16	1.00	1.00	1.00	232
17	1.00	1.00	1.00	202
18	1.00	1.00	1.00	197
19	1.00	1.00	1.00	185
20	1.00	1.00	1.00	181
21	1.00	1.00	1.00	203
22	1.00	1.00	1.00	207
23	1.00	1.00	1.00	201
24	1.00	1.00	1.00	202
25	1.00	1.00	1.00	183
26	1.00	0.99	0.99	204
27	1.00	1.00	1.00	193
28	1.00	1.00	1.00	218
accuracy			1.00	5800
macro avg	1.00	1.00	1.00	5800
weighted avg	1.00	1.00	1.00	5800

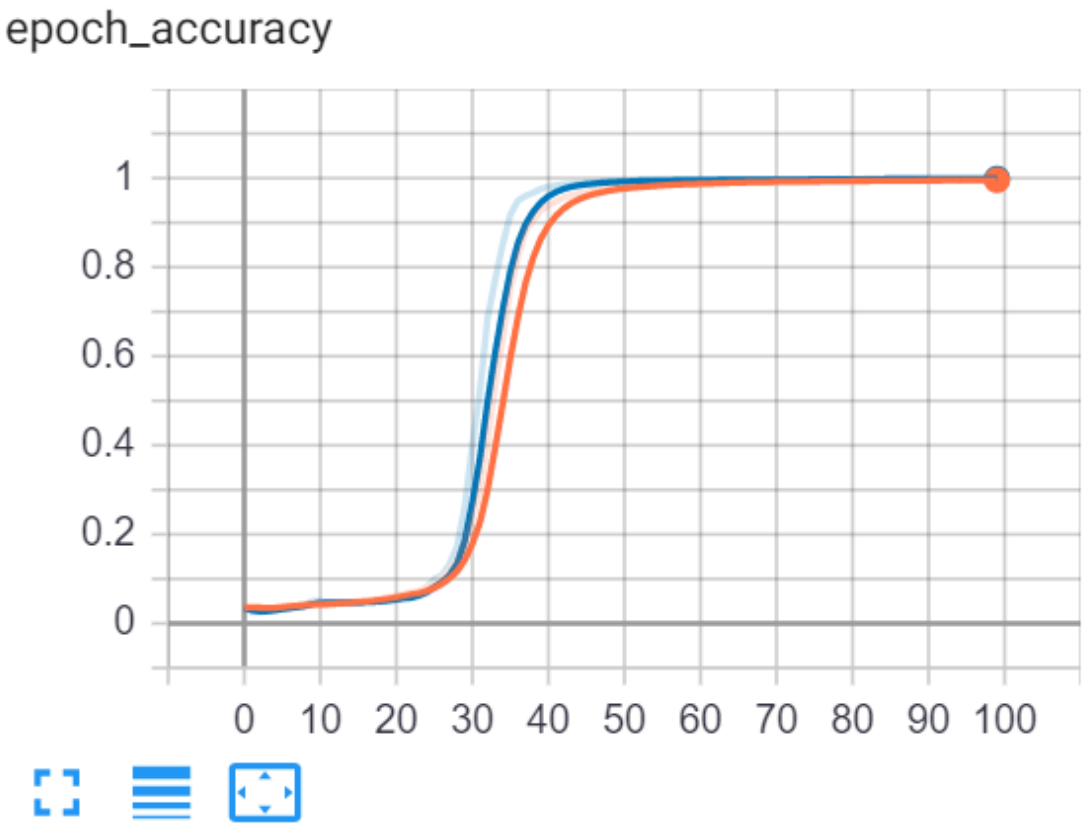
These is the report that I got. You can see that the precision, recall and f1-scores are almost 1 except for few gestures. For example, gesture number 8 has a precision of 0.99. Gesture 8 is ‘I’. This might have happened due to noise in any of the images in the test set. The accuracy was also found out to be 1. Also remember that the testing set was also collected at the time of taking snaps for training set. The actual performance can be observed while we try to recognize the gesture.



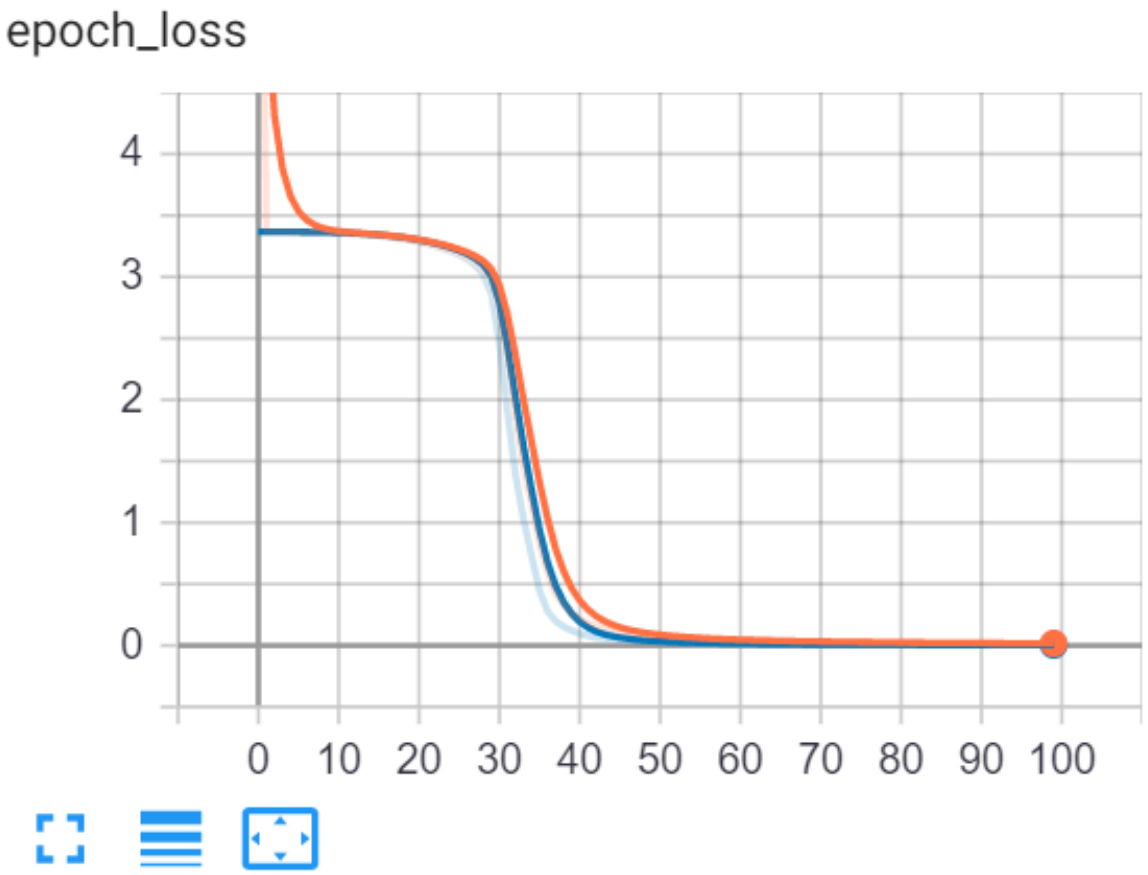
The above image is the confusion matrix that I have obtained. Please zoom to examine it clearly. On the Y-axis, you can see the ground truth and on X-axis you can see the predicted class. For example, the 1st row is the true gesture for the letter ‘A’. It is predicted correctly 192 times and was predicted as gesture number 4, which is E, once. You can see that the numbers are different as we have shuffled and taken few from each gesture for test dataset. These scores look very promising as well.

2)From TensorBoard:

As mentioned in the demo, when you go to the specified url (http://localhost:6006/, in my case), you can first see two graphs: epoch_accuracy and epoch_loss.

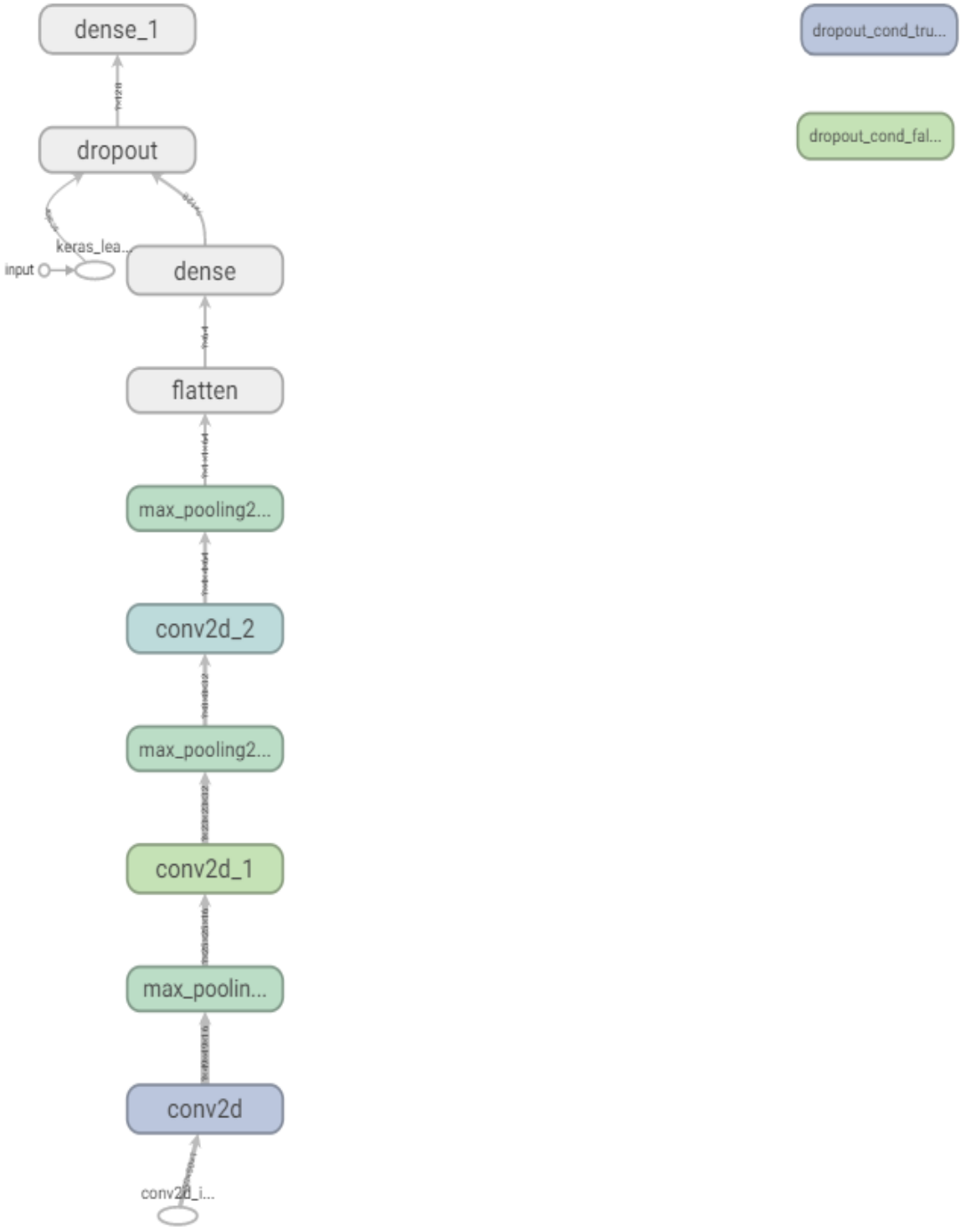


Blue line represents validation set and red line represents training set. You can see that both have almost reached 1.



You can see that as the epochs reached 100, the loss has reached almost 0.

<u>Note</u>: This indicates that 60-70 epochs would have been enough to train the model. After 70 epochs it's almost the same accuracy and loss throughout. I have reduced the epochs to 70 after this. The graphs are for 100 epochs, however.



This is the neural network structure that I used. This can also be obtained from TensorBoard under ‘Graphs’ tab after opening the url.

Discussion:

Under this section, I will discuss about the parts I liked about this project, the drawbacks and limitations, and also how this project can be extended in future.

The good part of this project is generating our own dataset. You can delete the whole gestures folder and capture your own gestures and train it. It won't take much time with proper lighting. Training is easy and fast even without a GPU as the images we stored are in Gray scale and do not have much details. This project can also be used by anyone without proper technical knowledge if they follow the instructions that I have mentioned in the demo part clearly.

My complaint with this approach is the noise when we are capturing or recognizing gestures. Though I used denoising filters, it is still prone to some outliers due to the similarity in color of our object of interest and some other object in the environment around us. That is the reason why I had to use Gloves. When I used my hand with excess lighting, it will almost appear close to the roof's color or some other color in the room. For the camera to properly recognize, it is better to use a distinct color like green, orange or red. Other issue that I faced was, in the source I referred to, they have used the regular CNN model. When I tried using AlexNet , our images are too small and AlexNet has many layers which try reducing the image. This became a hinderance. I ended up using the same network prescribed by the author. Coming from electronics background, I would still recommend the use of sensors for hands which recognize the posture of your hand accurately. Accelerometer can be used for this purpose. In the area of robotics, accelerometer is used to control drones using gestures. However, if we want the bot to process the image it is seeing, we probably need to use cameras. I hope high quality cameras can recognize the gestures properly than the regular

laptop camera that I used for this project. Also, while creating gesture the lighting will almost be uniform for all the 1200 snaps that we take. After flipping and shuffling, some of these will be taken into test dataset. Hence, we can see good values in our classification report. While recognizing the gesture at a later point of time, we can observe that the lighting is different. Though we set hand histogram again and save it, it still isn't that efficient.

In future, a UI can be built to this project and an app could be created. Different networks can be used. High definition cameras can be used to capture images better. More signs can also be created from different domains. Advanced methods like Neural Machine Translation (NMT) can be used along with attention layers. With proper modifications in terms of image quality and good equipment, we can use this project in many applications from assistance to speech and hearing-impaired people to controlling robots with gestures.

References:

[1] https://www.aljazeera.com/news/2017/12/15/listening-to-indias-hearing-impaired/?gb=true#:~:text=An%20estimated%2063%20million%20people,and%20speech%20impairment%20in%202016.

[2] https://www.fda.gov/medical-devices/cochlear-implants/benefits-and-risks-cochlear-implants

[3] https://www.sas.com/en_us/insights/analytics/neural-networks.html#:~:text=What%20they%20are%20%26%20why%20they,time%20%E2%80%93%20continuously%20learn%20and%20improve.

[4] Sajeena, A., Sheeba, O., & Ajitha, S. S. (2020, April). Indian sign language recognition using AlexNet. In *AIP Conference Proceedings* (Vol. 2222, No. 1, p. 030028). AIP Publishing LLC.

[5] Raghuveera, T., Deepthi, R., Mangalashri, R., & Akshaya, R. (2020). A depth-based Indian Sign Language recognition using Microsoft Kinect. *Sādhanā*, *45*(1), 34.

[6] Ss, Shivashankara & S, Dr.Srinath. (2018). American Sign Language Recognition System: An Optimal Approach. International Journal of Image, Graphics and Signal Processing. 10. 10.5815/ijigsp.2018.08.03.

[7] Bheda, Vivek, and Dianna Radpour. "Using deep convolutional networks for gesture recognition in American sign language." *arXiv preprint arXiv:1710.06836* (2017).

[8] https://www.kaggle.com/grfiv4/plot-a-confusion-matrix

[9] https://medium.com/@kennymiyasato/classification-report-precision-recall-f1-score-accuracy-16a245a437a5