ASL Detection

Group Members:

* Harbir Singh Sahi – 0797037
* Ravi Ravi - 0795606
* Pratva - 0766531
* Manpreet Kaur - 080080
* Simnranjeet Singh – 0793112

Introduction:

Over 5% of the world’s population or 430 million people – require rehabilitation to address their disabling hearing loss, out of these (432 million children and 34 million children).

[ <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss> ]

For context, A person is said to be impaired when they have problems hearing anything greater than 35 dBs, out of these 80% of the people with live in low- and middle-income countries where accessibility is often a big concern. It has also been noted that hearing loss is directly proportional to the age but it is not the only factor that drives it. Some of the other major causes of hearing loss and deafness are:

* Genetic and other various pre-natal conditions.
* Chronic ear infections
* Smoking
* Trauma or other work-related hazards.

The goal of the project is to help the deaf and hearing impaired communicate effectively with others who may not understand ASL.

Technical Objective:

The end goal as stated above was to recognize gestures made by the live video camera. To achieve that one would have to build a model that can recognize and correctly label the image with very good accuracy. The accuracy in question should also be improved through feedback and continuous learning and further training.

Proposed System:

We propose to build a system using deep learning techniques, more specifically a convolutional neural network (CNN) to train a model that can accurately recognize hand gestures images and translate them into text. The said model would be trained on a large dataset of labelled hand gesture images and the accuracy of the model is continuously improved through feedback and further training.

The project will work like following – it would capture live video feed from a camera then isolating the region of interest (ROI) on the screen that contains the user’s hand. The said hand would then be segmented from the rest of the picture and pre-processed to remove any noise in the background. Next the segmented hand would be compared to pre-defined set of hand gestures, each of which is associated with a specific word or a phrase.

The small pipeline for the model will be: input image – model – Prediction.

Advantage over existing solutions:

We searched on GitHub and internet for an existing project that captured all aspects of the ASL, the video motions and images for a wide variety of gestures but majority of the projects that were made were limited to either just the alphabet or a combination of alphabets and numbers, we could not find a single project that even tried to include any form of sentences or words into the training. Some of the existing solutions that we took insights from are listed as following:

* <https://www.eecg.utoronto.ca/~jayar/mie324/asl.pdf>

This model used CNN and had peak accuracy of about 80%

* <https://github.com/emnikhil/Sign-Language-To-Text-Conversion>

This model encapsulates what we aim to do to some extent – but only uses conjunction of alphabets to form words which is often not the case in ASL.

Exploratory Data analysis:

EDA plays a crucial role in understanding the characterizes of the data used and in identifying any potential issues or challenges that may need to be addressed during the model training process.

For this project we had proposed to take a large dataset of alphabets to train our model but that data had couple of problems with it plus we had to make our own datasets for the phrases and words anyway so we decide to make the entire dataset ourself. As of now the data that the model was trained on has following labels – (0-9) numbers, (A-Z) alphabets and about 20 phrases / words which can be used to recognize simple sentences.

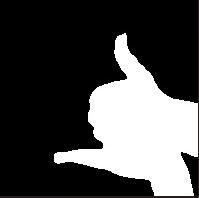
EDA on this dataset for the most part consisted of distributing the images across different classes, quality of the images that were used in the dataset. The most important part was perhaps pre-processing the data which in our case was resizing the images – normalizing the pixel values and augmenting the pictures we had already taken using ImageDataGenerator library in python to further strengthen our dataset by applying rotation, scaling, flipping and gray scaling.

Image Sample and Description of Each Class Label:

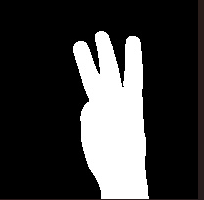
As mentioned, before we have a lot of different labels for our data so we will be sharing some of the more prominent ones that are unique in some way.



The image above is a pre-processed image of the gesture that translates to the phrase ‘I Love You’



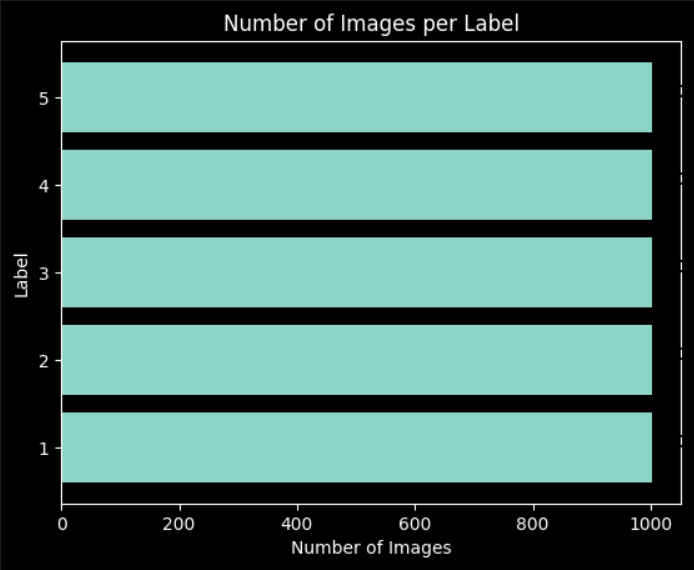
Meanwhile this image is used to convey the word ‘Help’



This symbol is used to describe the number 6 which is coincidentally very similar to letter ‘W’.

Distribution of the dataset:

As mentioned, before we have a lot of labels so showing that they all contain the same number of images would be little challenging and visually not pleasing, so we read the first few folders in our training dataset to see how much data they contain:



This gives us one very important detail – that the data that is being input into the model is balanced and not skewed which is a very important detail as to make any model good, we need to provide it with useful and good data and this is a very important first step as mentioned in the EDA process.

As for the dimension of the images being used in the dataset, they are all set to be 200 pixels by 200 pixels and the reason for that is two-fold:

1. 200 x 200 according to our testing was the ideal size because it made computation take not hours and we could provide feedback to train the model again in less than an hour which was very good.
2. If we made the image size too small, we tested that the model accuracy suffered a lot which could have been due to the fact that there was not enough information to learn meaningful features. Plus, this also reduces the CNN’s learning capacity to learn more complex features from the images.

Image Processing:

As we mentioned before, before we put any images in the model, we must make the images as good as we can. This was achieved by using a lot of tools provided to us by OpenCV-python library.

* Using OpenCV we resized the captured image into suitable dimensions, in this case it would be 200x200 pixels.
* Then the said images were normalized to reduce bias and avoid units of each color.
* CV2.videocapture() function to create a video capture object which captures video frames from the default camera setting.
* CV2.flip() to used to flip the frame horizontally, since the camera is usually mirrored.
* ROI (region of interest) is converted to grayscale and a gaussian blur is applied using CV2.gaussianBlur() to remove noise and smoothen the image.
* The function we used in the code cal\_accum\_avg() is used to calculate the accumulated weighted average of the background over time. This was done by gradually blending the current frame with the previous one.
* Segment\_hand() function is called to segment the hand gesture from the background , this is done by using 2 techniques called threshold and contour.
* The resized image is converted into RGB color format using the CV2.cvtColor() and reshaped into a 4 Dimensional array using np.arrray, which is necessary so that we can put into the neural network.
* The final prediction by the model is done using model.predict() which returns us a probability of the each label for the input images, which in turn is displayed as text.

Data Partitioning:

Typically, the CNN is split into 3 datasets namely – test train and validation, our folders that we uploaded into GitHub is split into only 2 folders namely Test and train but the code snippet we have splits the test data we have into validation data, this was an oversight by us at the beginning of the project so instead of rearranging the folders or automating it in some way, we just essentially split the test dataset into 2 which works as validation, this is an inbuilt feature of ImageDataGenerator() which when specified takes certain parts of the the files to be labelled as ‘test’ ‘train’ or ‘validation’ as specified by the user.

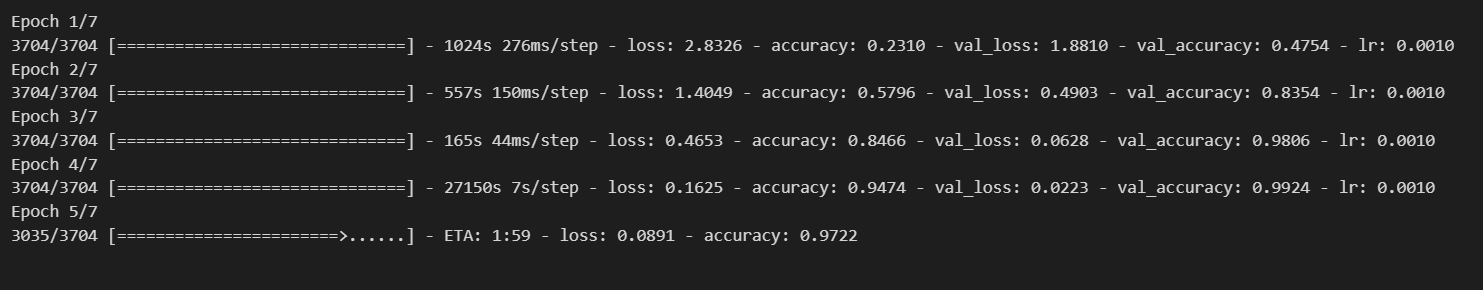
The split we chose on for this model was 80% of the data being used in training and the rest 20% of the data being split into the testing and validation evenly.

Data Modelling:

Unfortunately, we did not make a graph representing the accuracy metrics of our model but we can glean this information from the model.compile() function from tensorflow module which under the ‘accuracy’ metrics provides us with the following parameters:

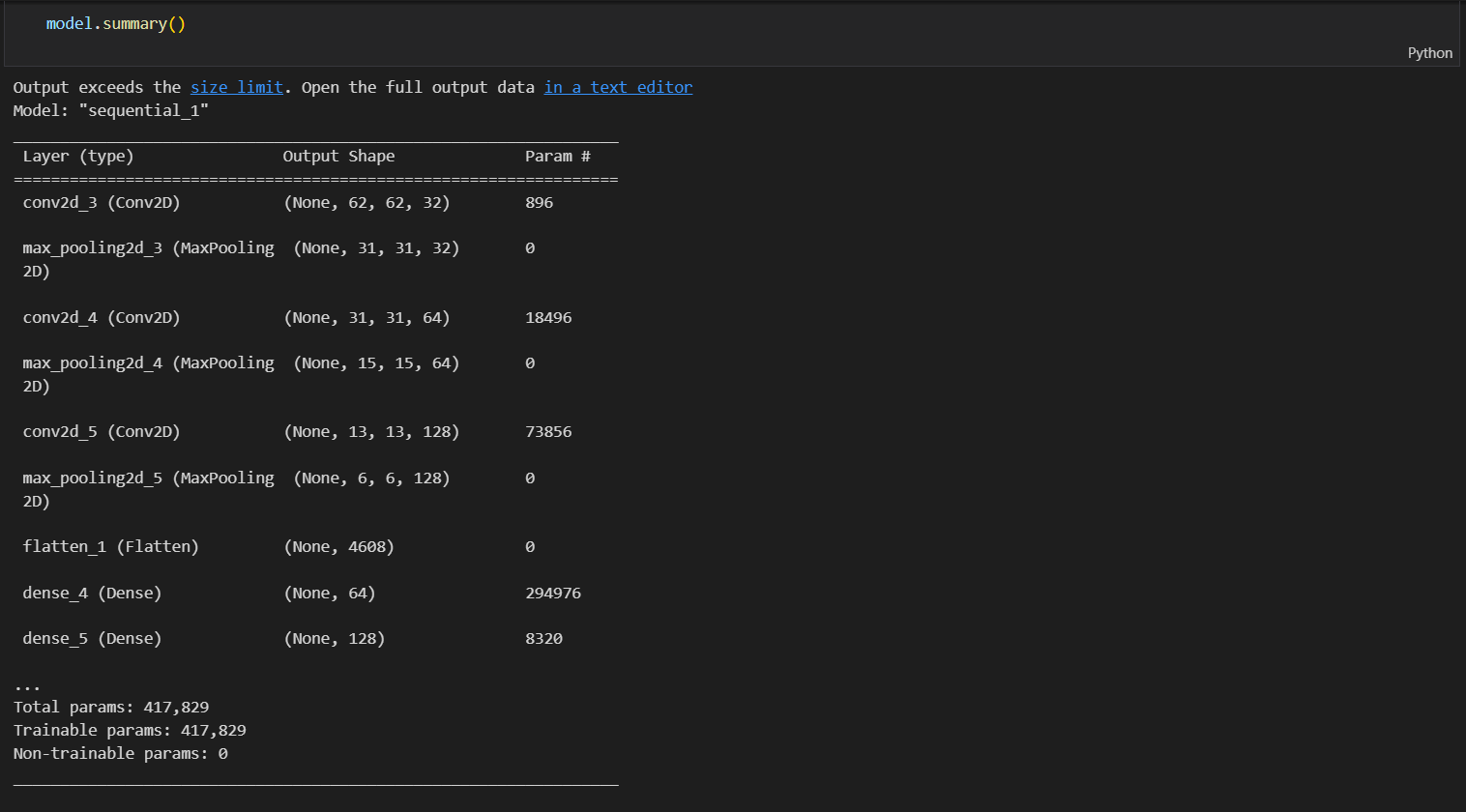
‘Test accuracy’, ‘test loss’, ‘validation accuracy’, ‘validation loss’

From the successful compilation of the model we have the following outcome:



An accuracy of 97% which is really good for an image classifier.

The summary of the model we made can also be seen using the tensorflow module by using model.summary() function, which is as following:



Discussion:

As shown above the image classification has a really good accuracy of 97% which means the model is really good at the task for which it was meant for ( image classification) , the dictionary that contains the labels can always be expanded , for example right now we have about 40 labels in it but using feedback learning and inputting new images into the model we can always train the model to include more phrases and make it more usable. The only slight drawback of the model is that since it uses differences in light intensity to segment the hand from rest of the image , the value of threshold function needs to be changed according to the brightness of room where the final product is being used as was seen during the live demonstration of our model in the classroom.

Conclusion:

All things considered, the model performs really well in real life testing while classifying the images that it already contains, which is a great success but at the same time the deliverable that was agreed upon could not be reached which was identifying the motion of the hands that are used for more complicated words. An attempt was made to try that but an accuracy of less than 50% means that the method used to attempt the recognition was not upto par or had some shortcomings. Looking at the bigger picture we can train the model through feedbacks to translate simple sentences, words and phrases.

References:

<https://github.com/ravi-392/capstone>

The link above contains all the relevant files to the project.

Also here are some other projects which we took insights from while making our own:

<https://www.ripublication.com/ijaer18/ijaerv13n9_90.pdf>

<http://noiselab.ucsd.edu/ECE228-2020/projects/Report/64Report.pdf>

Other documentations we took reference from while completing the project:

OpenCV-python: <https://docs.opencv.org/4.x/index.html>

ImageDataGenerator <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator>

Numpy <https://numpy.org/doc/>

Contributions:

Pratva , Manpreet , Simran : were involved in making the CNN model that was used and in supplying the model with pictures of various different images.

Harbir , Ravi : Worked on image processing , hyper parameter tuning and improving ROI.