Final Report of Traineeship Program 2023

On

"Sign Language Recognition Project"

KELLA NAVEEN KUMAR

MEDTOUREASY

28th April 2023

ACKNOWLDEGMENT

I would like to thank the team of MedTourEasy. The traineeship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Deep Learning; and also, for personal as well as professional development. I am very obliged for having a chance to interact with so many professionals who guided me throughout the traineeship project and made it a great learning curve for me. Firstly, I express my deepest gratitude and special thanks to the Training & Development Team of MedTourEasy who gave me an opportunity to carry out my traineeship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Deep Learning profile and training me in the same so that I can carry out the project properly.

TABLE OF CONTENTS

ABSTRACT

The goal of this project was to build a neural network able to classify which letter of the American Sign Language (ASL) alphabet is being signed, given an image of a signing hand. This project is a first step towards building a possible sign language translator, which can take communications in sign language and translate them into written and oral language. Such a translator would greatly lower the barrier for many deaf and mute individuals to be able to better communicate with others in day-to-day interactions.

Most research implementations for this task have used depth maps generated by depth camera and high-resolution images. The objective of this project was to see if neural networks are able to classify signed ASL letters using simple images of hands taken with a personal device such as a laptop webcam. This is in alignment with the motivation as this would make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

American Sign Language (ASL) is the primary language used by many deaf individuals in North America, and it is also used by hard-of-hearing and hearing individuals. The language is as rich as spoken languages and employs signs made with the hand, along with facial gestures and bodily postures.

A lot of recent progress has been made towards developing computer vision systems that translate sign language to spoken language. This technology often relies on complex neural network architectures that can detect subtle patterns in streaming video. However, as a first step, towards understanding how to build a translation system, we can reduce the size of the problem by translating individual letters, instead of sentences.

In this notebook**,** we will train a convolutional neural network to classify images of American Sign Language (ASL) letters. After loading, examining, and pre-processing the data, we will train the network and test its performance.

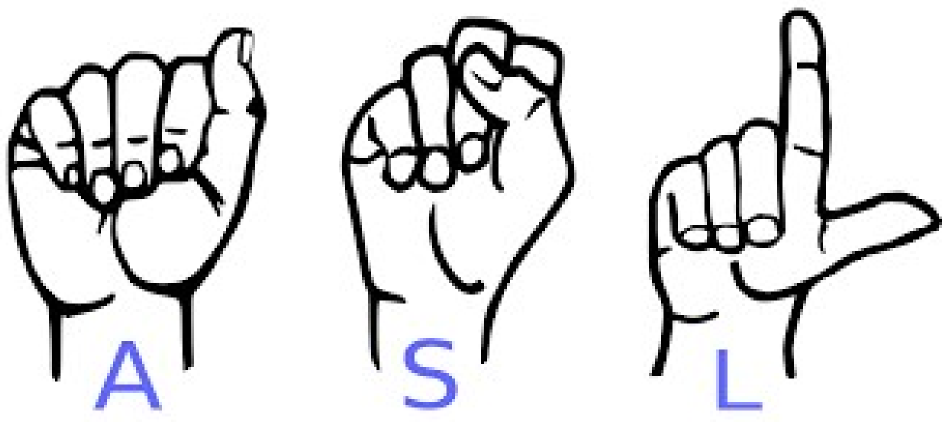
1.INTRODUCTION

1.1 About the Company

MedTourEasy, a global healthcare company, provides you the informational resources needed to evaluate your global options. MedTourEasy provides analytical solutions to our partner healthcare providers globally. it helps you find the right healthcare solution based on specific health needs, affordable care while meeting the quality standards that you expect to have in healthcare. MedTourEasy improves access to healthcare for people everywhere. It is an easy-to-use platform and service that helps patients to get medical second opinions and to schedule affordable, high quality medical treatment abroad.

1.2 About the Project

American Sign Language (ASL) is the primary language used by many deaf individuals in North America, and it is also used by hard-of-hearing and hearing individuals. The language is as rich as spoken languages and employs signs made with the hand, along with facial gestures and bodily postures.

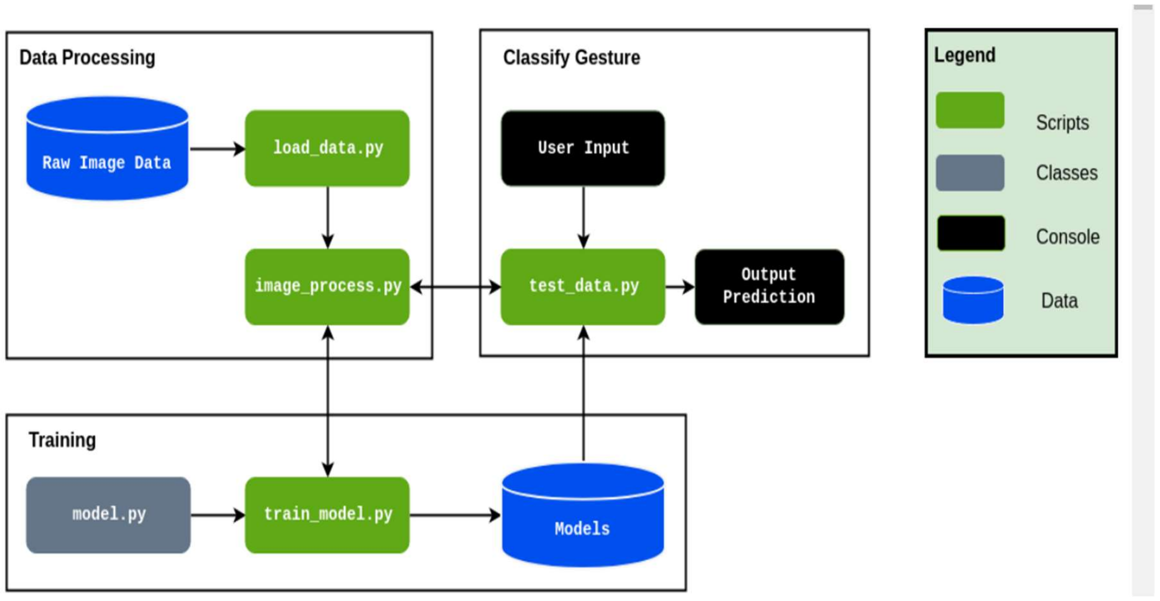


A lot of recent progress has been made towards developing computer vision systems that translate sign language to spoken language. This technology often relies on complex neural network architectures that can detect subtle patterns in streaming video.

However, as a first step, towards understanding how to build a translation system, we can reduce the size of the problem by translating individual letters, instead of sentences.

2.METHODOLOGY

2.1 Flow of the Project



As shown in Figure , the project will be structured into 3 distinct functional blocks, Data Processing, Training, Classify Gesture. The block diagram is simplified in detail to abstract some of the minutiae:

• **Data Processing:** The load data.py script contains functions to load the Raw Image Data and save the image data as NumPy arrays into file storage. The process data.py script will load the image data from data.npy and pre-process the image by resizing/rescaling the image, and applying filters and ZCA whitening to enhance features. During training the processed image data was split into training, validation, and testing data and written to storage. Training also involves a load dataset.py script that loads the relevant data split into a Dataset class. For use of the trained model in classifying gestures, an individual image is loaded and processed from the filesystem.

• **Training:** The training loop for the model is contained in train model.py. The model is trained with hyperparameters obtained from a config file that lists the learning rate, batch size, image filtering, and number of epochs. The configuration used to train the model is saved along with the model architecture for future evaluation and tweaking for improved results. Within the training loop, the training and validation datasets are loaded as Data loaders and the model is trained using Adam Optimizer with Cross Entropy Loss. The model is evaluated every epoch on the validation set and the model with best validation accuracy is saved to storage for further evaluation and use. Upon finishing training, the training and validation error and loss is saved to the disk, along with a plot of error and loss over training.

• **Classify Gesture:** After a model has been trained, it can be used to classify a new ASL gesture that is available as a file on the filesystem. The user inputs the file path of the gesture image and the test data.py script will pass the file path to process data.py to load and pre-process the file the same way as the model has been trained.

2.1 Language and Platform used:

Language: Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Important Features of Python:

1. Free and Open source
2. Easy to code
3. Easy to read
4. Object-Oriented Language
5. GUI-Programming Support
6. High Level Language
7. Extensible Feature
8. Easy to Debug
9. Python is a Portable Language

10. Python is a Integrated Language
11. Interpreted Language
12. Large Standard Library
13. Dynamically Typed Language
14. Frontend and backend development
15. Allocating Memory Dynamically

Platform: Jupyter Notebook

The Jupyter Notebook is an open-source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

3.IMPLEMENTATION

3.1 Data Collection

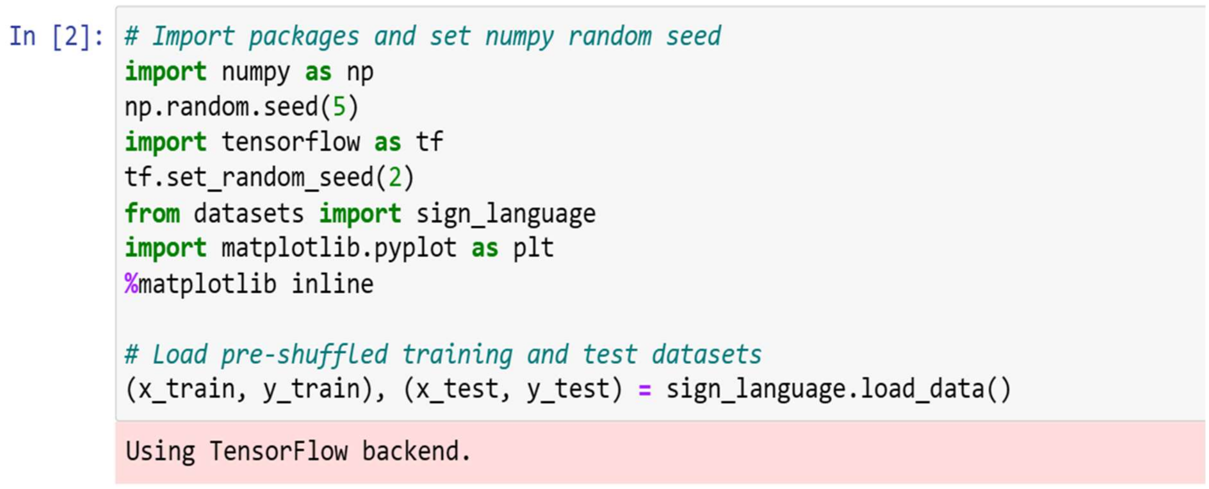
The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the ASL Alphabet provided by MedTourEasy organisation. The dataset is comprised of 87,000 images which are 200x200 pixels. There are 29 total classes, each with 3000 images, 26 for the letters A-Z and 3 for space, delete and nothing. These photos were then cropped, rescaled, and labelled for use.

A self-generated test set was created in order to to investigate the neural network's ability to generalize. Five different test sets of images were taken with a webcam under different lighting conditions, backgrounds, and use of dominant/non-dominant hand. These images were then cropped and pre-processed.

3.2 Load the training and test data

In the code cell below, we load the training and test data.

- x_train and x_test are arrays of image data with shape (num_samples, 3, 50, 50), corresponding to the training and test datasets, respectively.
- y_train and y_test are arrays of category labels with shape (num_samples,), corresponding to the training and test datasets, respectively.

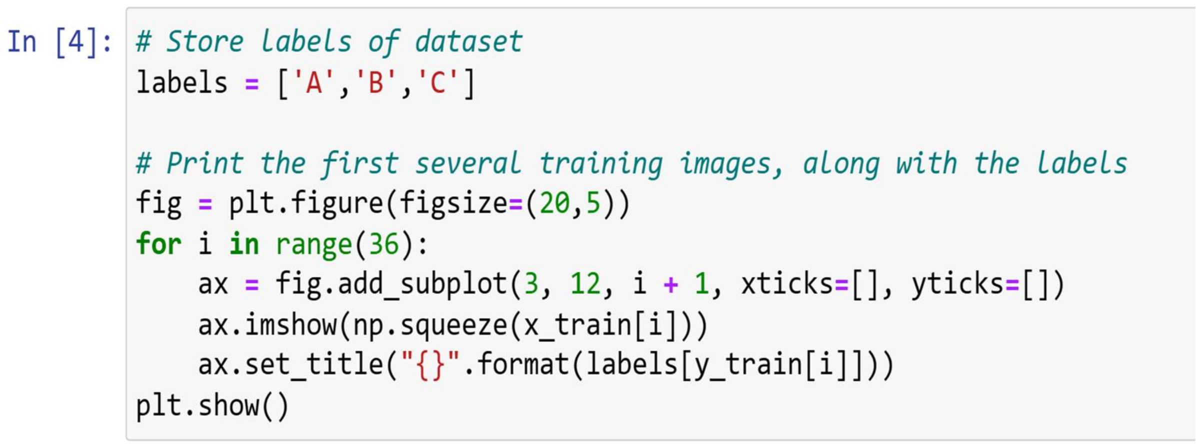


```
In [2]: # Import packages and set numpy random seed
        import numpy as np
        np.random.seed(5)
        import tensorflow as tf
        tf.set_random_seed(2)
        from datasets import sign_language
        import matplotlib.pyplot as plt
        %matplotlib inline

        # Load pre-shuffled training and test datasets
        (x_train, y_train), (x_test, y_test) = sign_language.load_data()
```

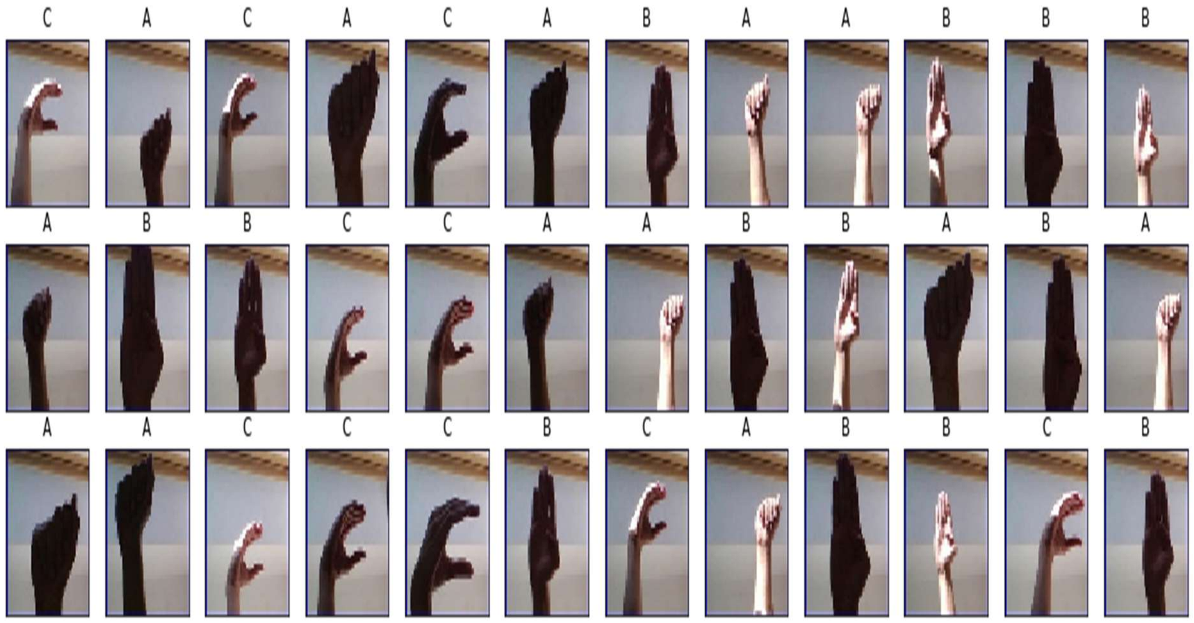
```
Using TensorFlow backend.
```

3.3 Visualise the Training Data

Now we'll begin by creating a list of string-valued labels containing the letters that appear in the dataset. Then, we visualize the first several images in the training data, along with their corresponding labels.



```
In [4]: # Store labels of dataset
        labels = ['A','B','C']

        # Print the first several training images, along with the labels
        fig = plt.figure(figsize=(20,5))
        for i in range(36):
            ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
            ax.imshow(np.squeeze(x_train[i]))
            ax.set_title("{}".format(labels[y_train[i]]))
        plt.show()
```



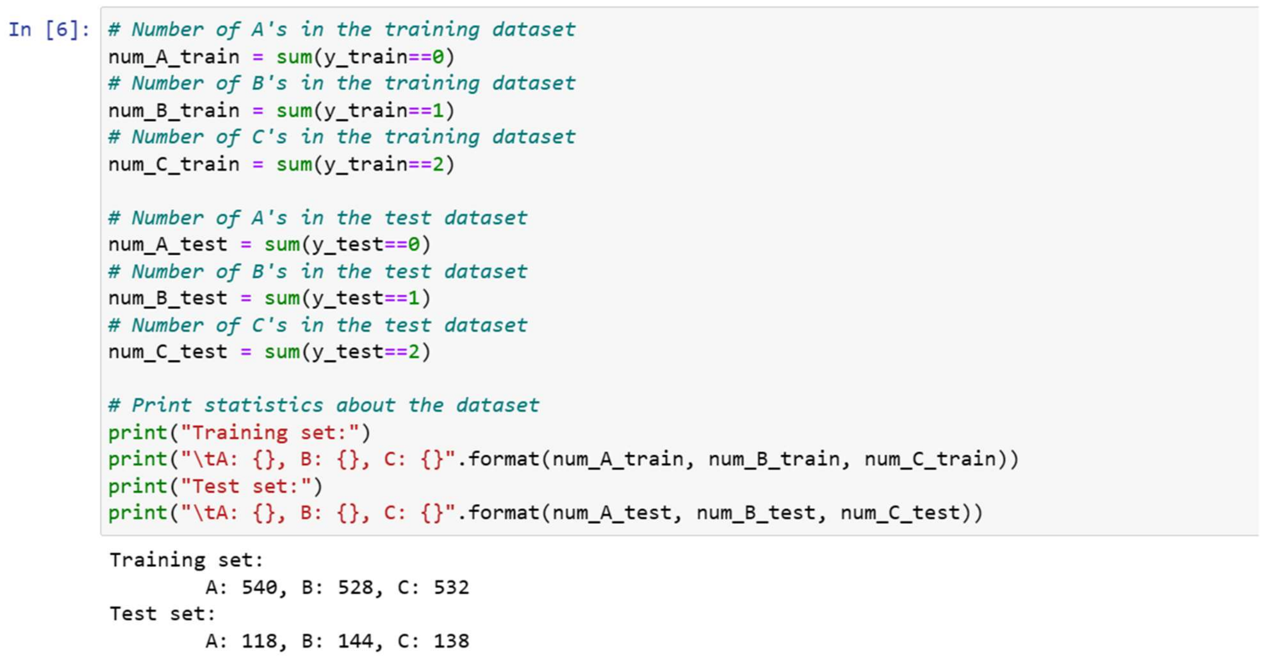
3.4 Examine the Dataset

Let's examine how many images of each letter can be found in the dataset.

Remember that dataset has already been split into training and test sets for you, where x_train and x_test contain the images, and y_train and y_test contain their corresponding labels.

Each entry in y_train and y_test is one of 0, 1, or 2, corresponding to the letter’s 'A', 'B', and 'C', respectively.

We will use the arrays y_train and y_test to verify that both the training and test sets each have roughly equal proportions of each letter.



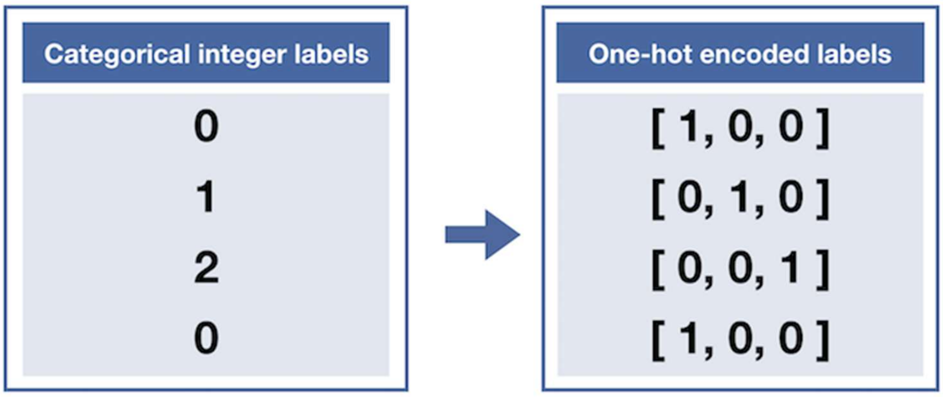
```
In [6]: # Number of A's in the training dataset
        num_A_train = sum(y_train==0)
        # Number of B's in the training dataset
        num_B_train = sum(y_train==1)
        # Number of C's in the training dataset
        num_C_train = sum(y_train==2)

        # Number of A's in the test dataset
        num_A_test = sum(y_test==0)
        # Number of B's in the test dataset
        num_B_test = sum(y_test==1)
        # Number of C's in the test dataset
        num_C_test = sum(y_test==2)

        # Print statistics about the dataset
        print("Training set:")
        print("\tA: {}, B: {}, C: {}".format(num_A_train, num_B_train, num_C_train))
        print("Test set:")
        print("\tA: {}, B: {}, C: {}".format(num_A_test, num_B_test, num_C_test))
```

```
Training set:
        A: 540, B: 528, C: 532
Test set:
        A: 118, B: 144, C: 138
```

3.5 One-Hot Encode the Data

Currently, our labels for each of the letters are encoded as categorical integers, where 'A', 'B' and 'C' are encoded as 0, 1, and 2, respectively. However, recall that Keras models do not accept labels in this format, and we must first one-hot encode the labels before supplying them to a Keras model.

This conversion will turn the one-dimensional array of labels into a two-dimensional array.

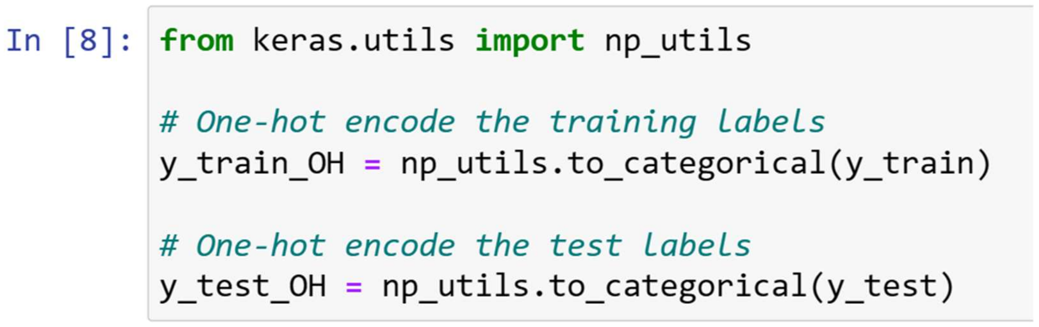


Each row in the two-dimensional array of one-hot encoded labels corresponds to a different image. The row has a 1 in the column that corresponds to the correct label, and 0 elsewhere.

For instance,

- 0 is encoded as [1, 0, 0],
- 1 is encoded as [0, 1, 0], and
- 2 is encoded as [0, 0, 1].

One-hot encoding is easy to do using keras

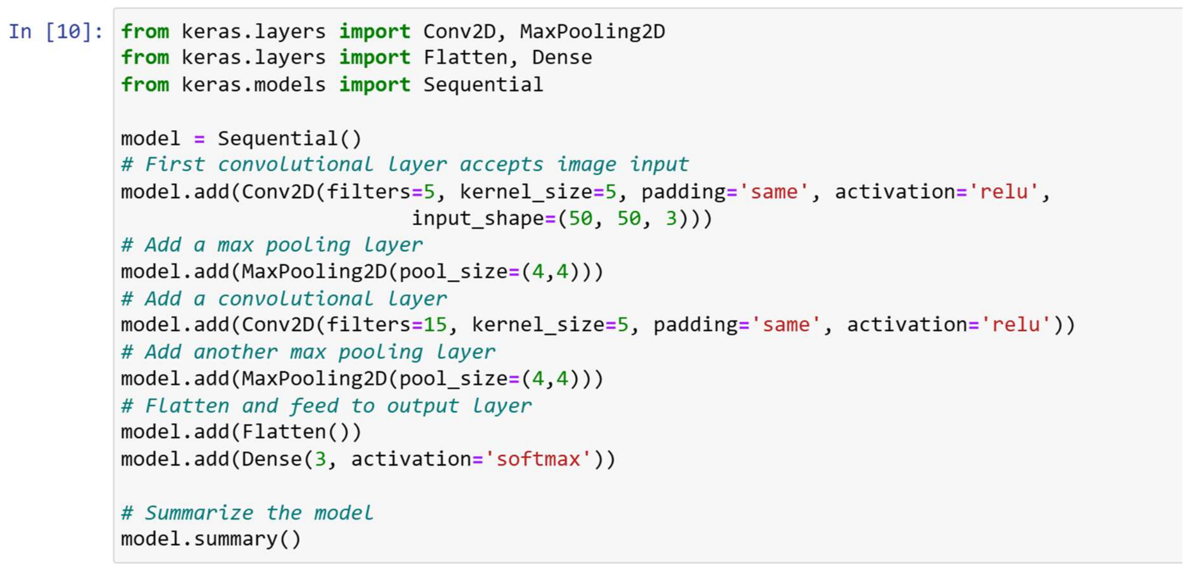


```
In [8]: from keras.utils import np_utils

        # One-hot encode the training labels
        y_train_OH = np_utils.to_categorical(y_train)

        # One-hot encode the test labels
        y_test_OH = np_utils.to_categorical(y_test)
```

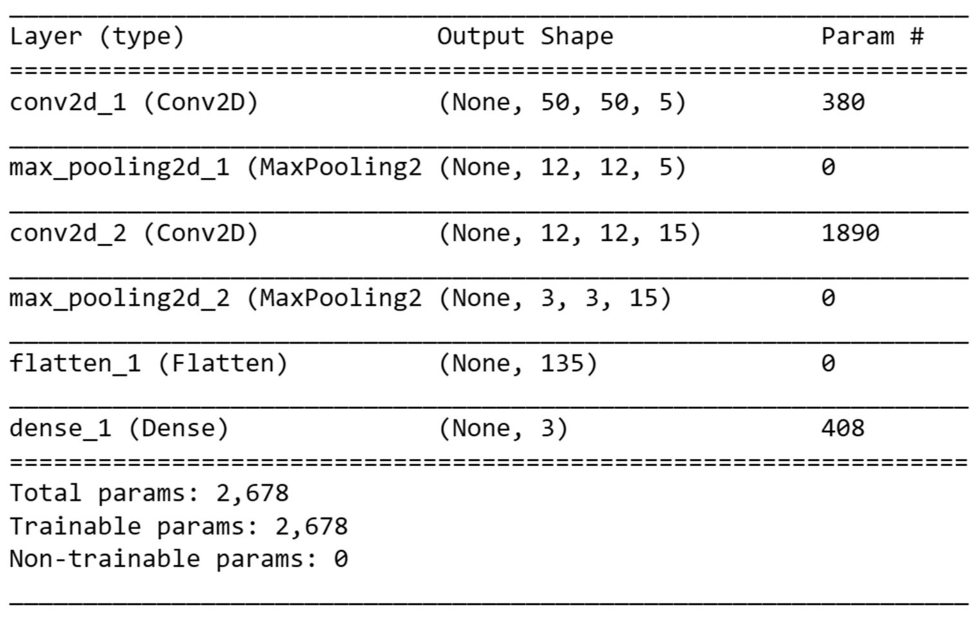
3.6 Define the Model

Now it's time to define a convolutional neural network to classify the data.

This network accepts an image of an American Sign Language letter as input. The output layer returns the network's predicted probabilities that the image belongs in each category.



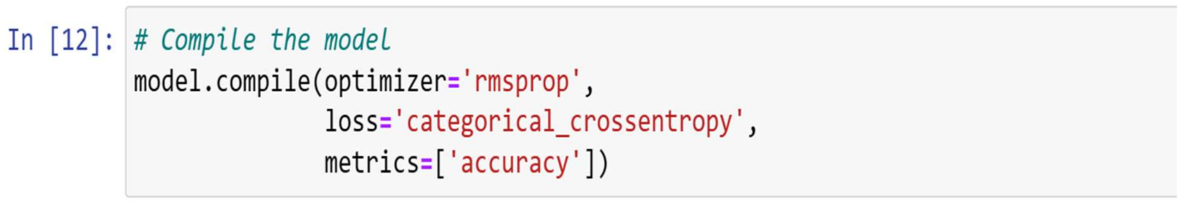
```
In [10]: from keras.layers import Conv2D, MaxPooling2D
         from keras.layers import Flatten, Dense
         from keras.models import Sequential

         model = Sequential()
         # First convolutional layer accepts image input
         model.add(Conv2D(filters=5, kernel_size=5, padding='same', activation='relu',
                                 input_shape=(50, 50, 3)))
         # Add a max pooling layer
         model.add(MaxPooling2D(pool_size=(4,4)))
         # Add a convolutional layer
         model.add(Conv2D(filters=15, kernel_size=5, padding='same', activation='relu'))
         # Add another max pooling layer
         model.add(MaxPooling2D(pool_size=(4,4)))
         # Flatten and feed to output layer
         model.add(Flatten())
         model.add(Dense(3, activation='softmax'))

         # Summarize the model
         model.summary()
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 50, 50, 5)         380
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 5)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 15)        1890
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 15)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 135)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 408
=================================================================
Total params: 2,678
Trainable params: 2,678
Non-trainable params: 0
_________________________________________________________________
```

3.7 Compile the Model

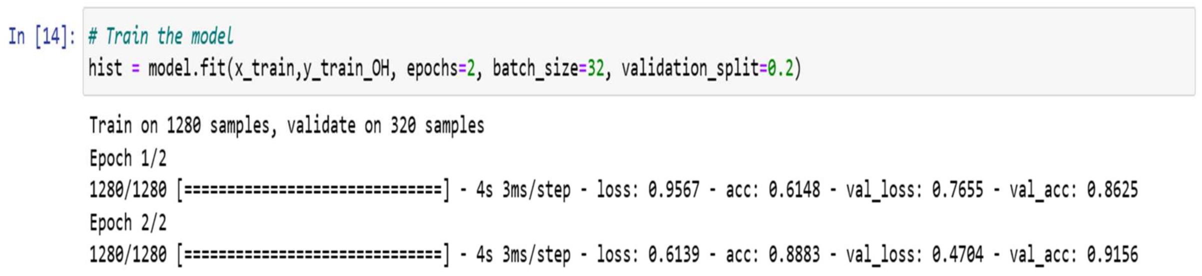
After we have defined a neural network in Keras, the next step is to compile it!'



```
In [12]: # Compile the model
         model.compile(optimizer='rmsprop',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
```

3.8 Train the Model

Once we have compiled the model, we're ready to fit it to the training data.

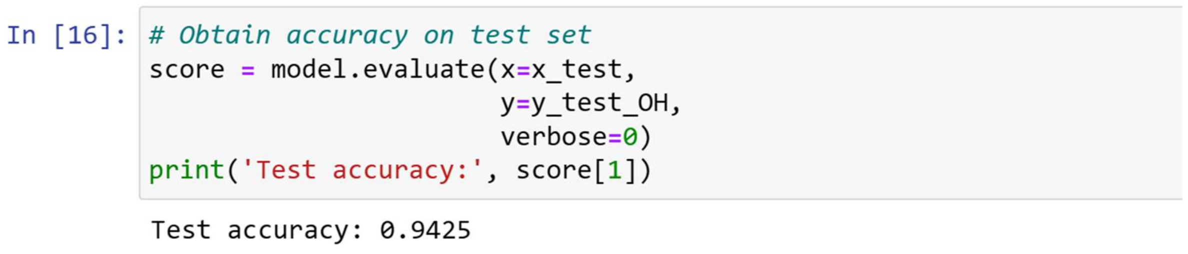


```
In [14]: # Train the model
         hist = model.fit(x_train,y_train_OH, epochs=2, batch_size=32, validation_split=0.2)

         Train on 1280 samples, validate on 320 samples
         Epoch 1/2
         1280/1280 [==============================] - 4s 3ms/step - loss: 0.9567 - acc: 0.6148 - val_loss: 0.7655 - val_acc: 0.8625
         Epoch 2/2
         1280/1280 [==============================] - 4s 3ms/step - loss: 0.6139 - acc: 0.8883 - val_loss: 0.4704 - val_acc: 0.9156
```

3.9 Test the Model

To evaluate the model, we'll use the test dataset. This will tell us how the network performs when classifying images it has never seen before!

If the classification accuracy on the test dataset is similar to the training dataset, this is a good sign that the model did not overfit to the training data.



```
In [16]: # Obtain accuracy on test set
         score = model.evaluate(x=x_test,
                                y=y_test_OH,
                                verbose=0)
         print('Test accuracy:', score[1])

         Test accuracy: 0.9425
```

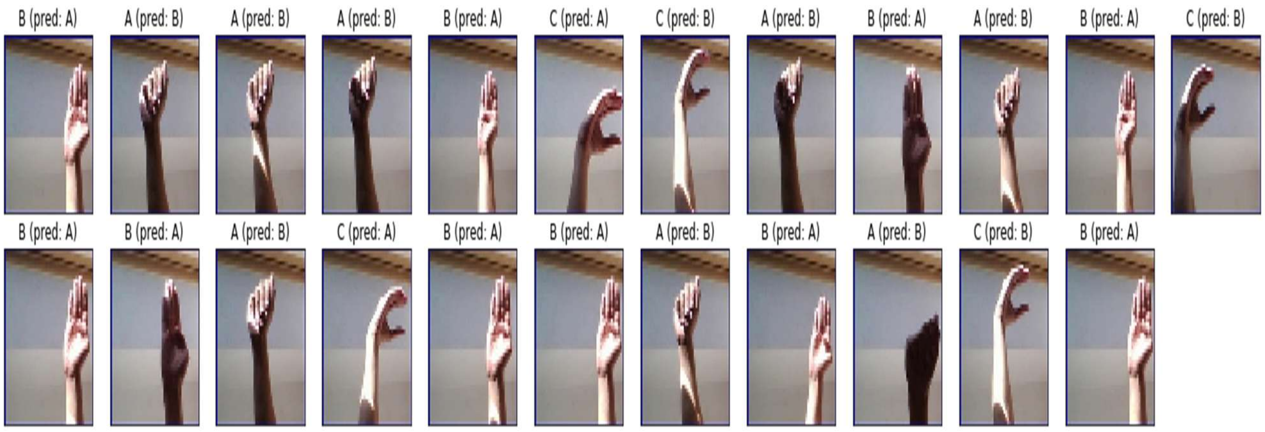
3.10 Visualise the Mistakes

Our network gets very high accuracy on the test set!

The final step is to take a look at the images that were incorrectly classified by the model. Do any of the mislabelled images look relatively difficult to classify, even to the human eye?

Sometimes, it's possible to review the images to discover special characteristics that are confusing to the model. However, it is also often the case that it's hard to interpret what the model had in mind!

```
In [18]: # Get predicted probabilities for test dataset
         y_probs = model.predict(x_test)

         # Get predicted labels for test dataset
         y_preds = np.argmax(y_probs, axis=1)

         # Indices corresponding to test images which were mislabeled
         bad_test_idxs = np.where(y_preds!=y_test)[0]
         # Print mislabeled examples
         fig = plt.figure(figsize=(25,4))
         for i, idx in enumerate(bad_test_idxs):
             ax = fig.add_subplot(2, np.ceil(len(bad_test_idxs)/2), i + 1, xticks=[], yticks=[])
             ax.imshow(np.squeeze(x_test[idx]))
             ax.set_title("{} (pred: {})".format(labels[y_test[idx]], labels[y_preds[idx]]))
```



4. Conclusion

The aim of this project is to predict the ASL alphanumeric hand-gestures in real time. The above work shows that it can be solved with better accuracy. We were able to achieve training accuracy of 91.56% and testing accuracy of 94.25%. Our model showed good accuracy while predicting results.

5. Feature Scope

1. We can develop a model for ASL word and sentence level recognition. This will require a system that can detect changes with respect to the temporal space.
2. We can develop a complete product that will help the speech and hearing-impaired people, and thereby reduce the communication gap.

6.References

Datasets & Images –
https://drive.google.com/uc?export=download&id=1o3Eu6DLIc2UYSV0dkUML8BUxsHfBE68J

About Sign Language Recognition –
https://www.eecg.utoronto.ca/~jayar/mie324/asl.pdf

ASL Recognition with Deep Learning Coding information –

https://www.kaggle.com/code/gargimaheshwari/asl-recognition-with-deep-learning/notebook