CSEE5590 - Python and Deep Learning for Engineering & Science

Project Increment – 3

Team number: 5

Project Title

Sign Language for Numbers

Team Members

Vasim Shaikh

Shireesha Maddi

Harshita Patil

Prabhanjan Trivedi

University of Missouri-Kansas City

MS – Computer Science, 2020-21

5000 Holmes St,

Kansas City,

MO 64110

The Story and its details:

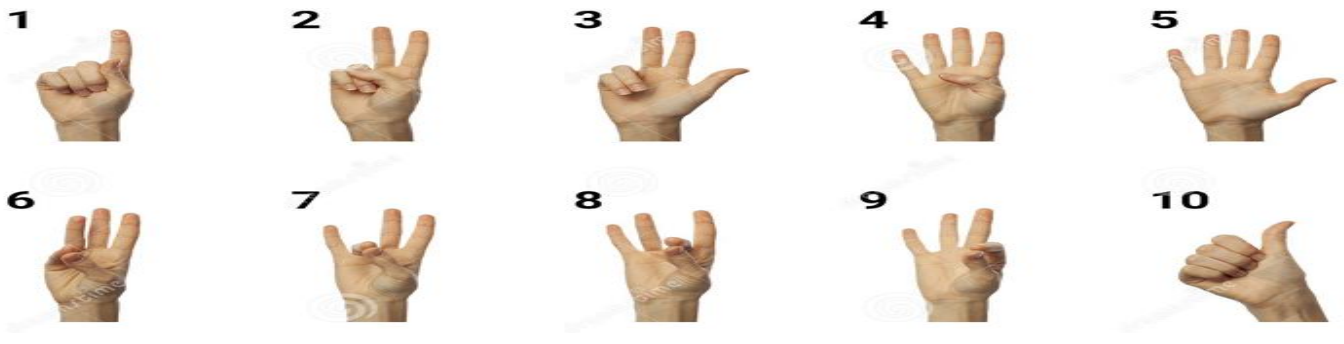
There have been several milestone reached in innovation and a great deal of technology advancement and research has been done to help the individuals who are physically challenged. Supporting the reason, Deep learning and computer vision can be utilized in an extensive way for community of people who can’t speak or listen.

The Sign language is very important for people who have hearing and speaking deficiency generally called Deaf and Mute. It is the only mode of communication for such people to convey their messages and it becomes very important for people to understand their language.

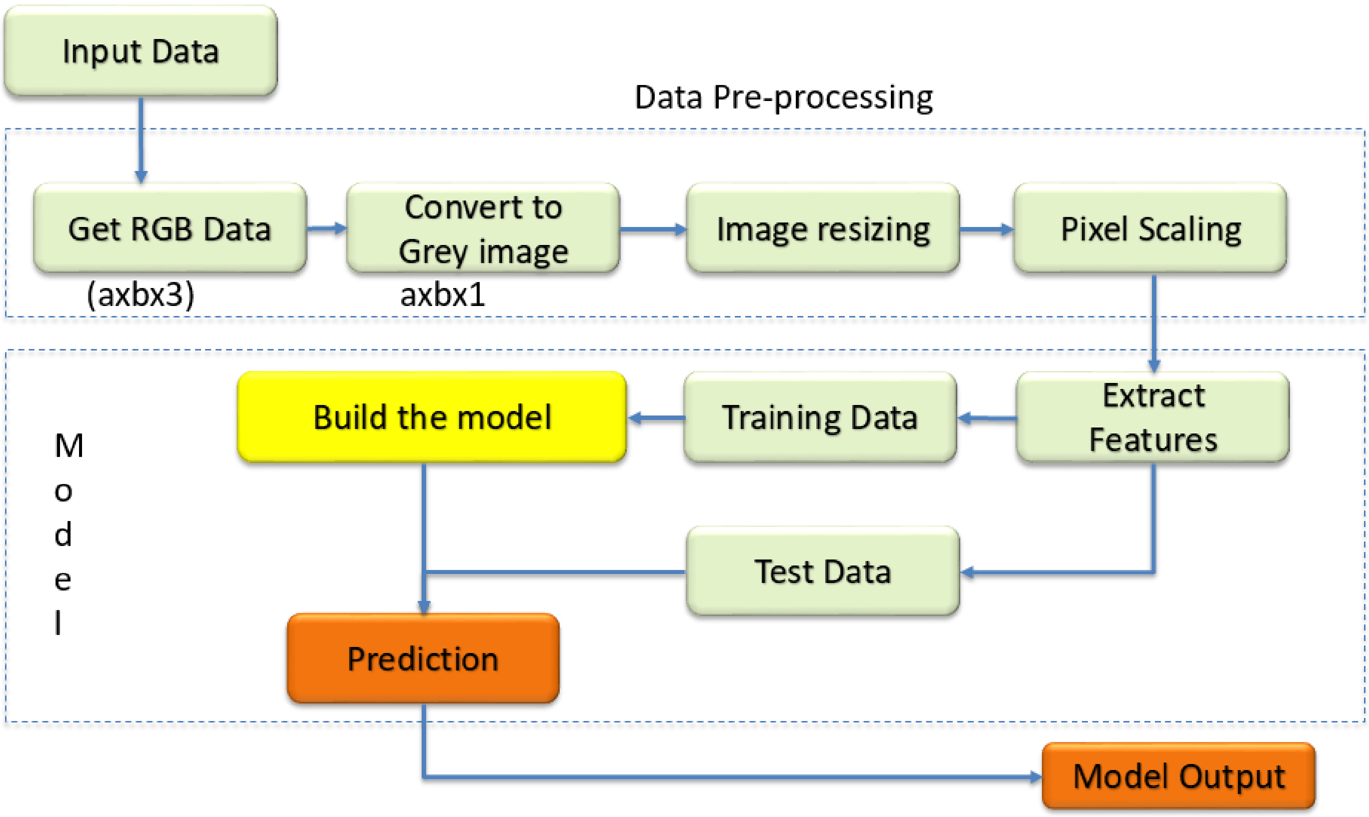
This project proposes the method or algorithm for an application which would help to recognize the different signs using palm/finger gestures. This will ease and make a better communication channel to rest of the world.

The Data and its details:

- Sign language datasets for hand gesture recognition for numbers
- Source of data – Kaggle – Link in reference section
- Sample hand signs are in below picture
- Total number of training images – 1500 images for each sign = 16,500
- Covering hand images 0 – 9 and unknown (Total classes – 11)
- Input data type – RGB Images(3 channels)
- Assumption – All input images are front facing

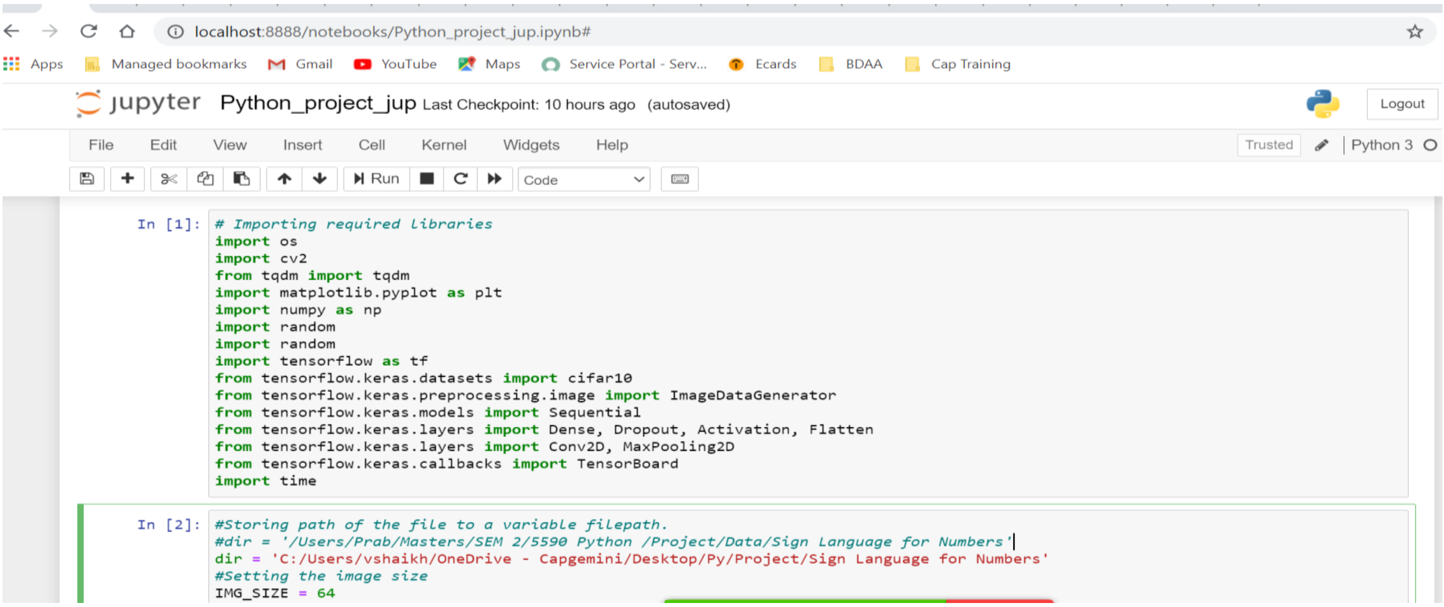


The Model Building Blocks:



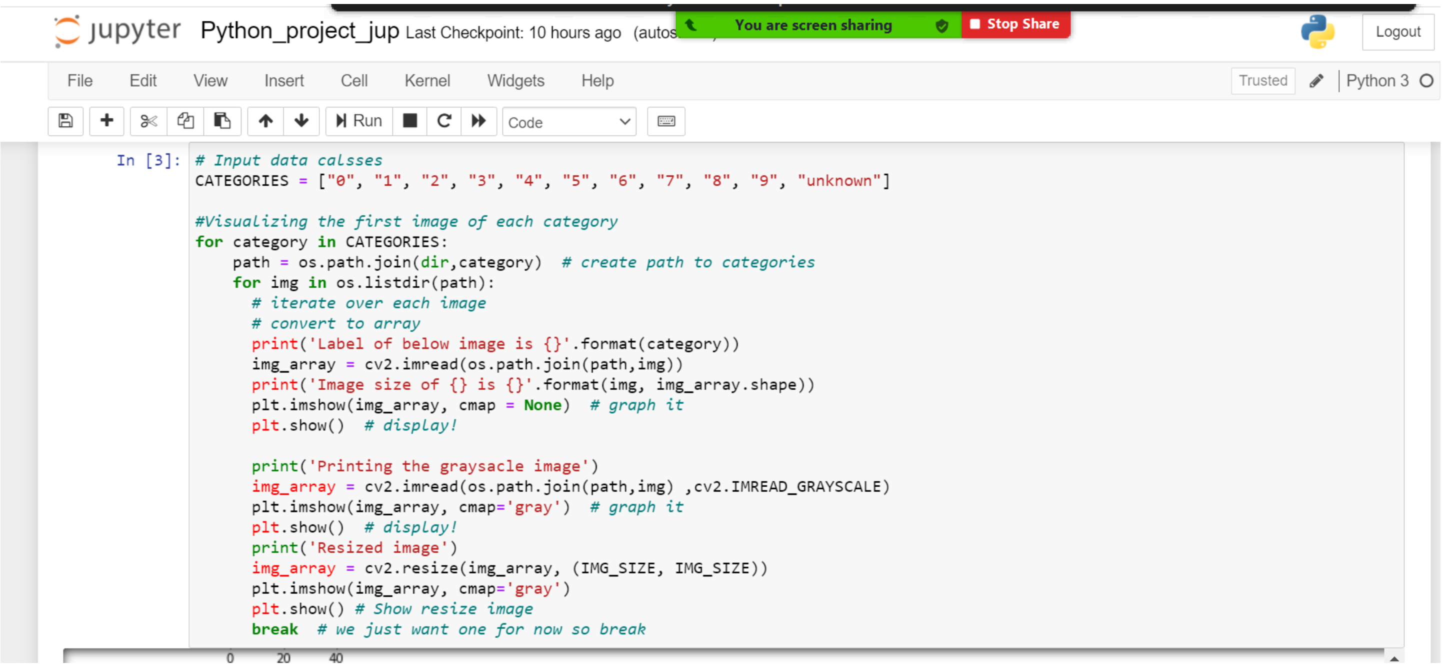
1. Blocks highlighted in green are completed
2. Build the model - Model 1 (CNN)implementation is complete and evaluation is in progress
3. Prediction and ensemble modelling – Next steps

Working screens from project:

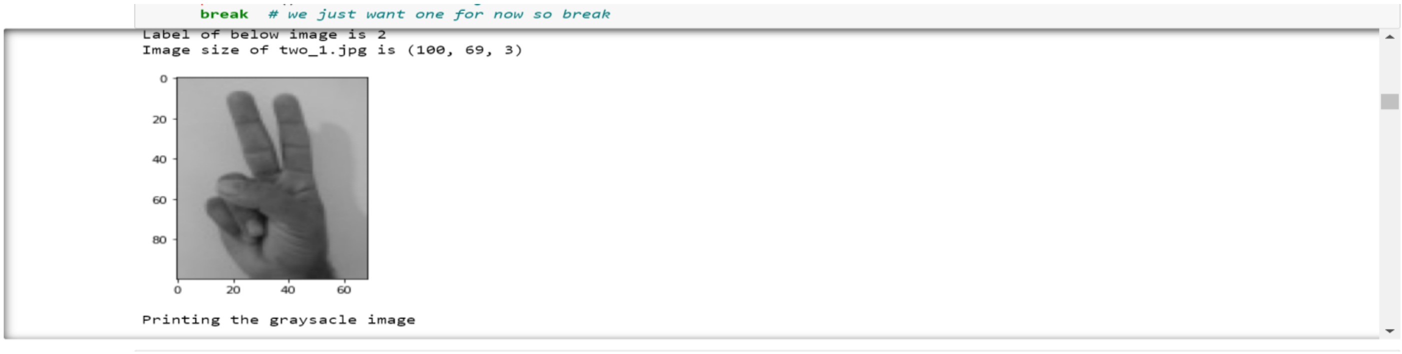


```
In [1]: # Importing required libraries
        import os
        import cv2
        from tqdm import tqdm
        import matplotlib.pyplot as plt
        import numpy as np
        import random
        import random
        import tensorflow as tf
        from tensorflow.keras.datasets import cifar10
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
        from tensorflow.keras.layers import Conv2D, MaxPooling2D
        from tensorflow.keras.callbacks import TensorBoard
        import time
```

```
In [2]: #Storing path of the file to a variable filepath.
        #dir = '/Users/Prab/Masters/SEM 2/5590 Python /Project/Data/Sign Language for Numbers'
        dir = 'C:/Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers'
        #Setting the image size
        IMG_SIZE = 64
```

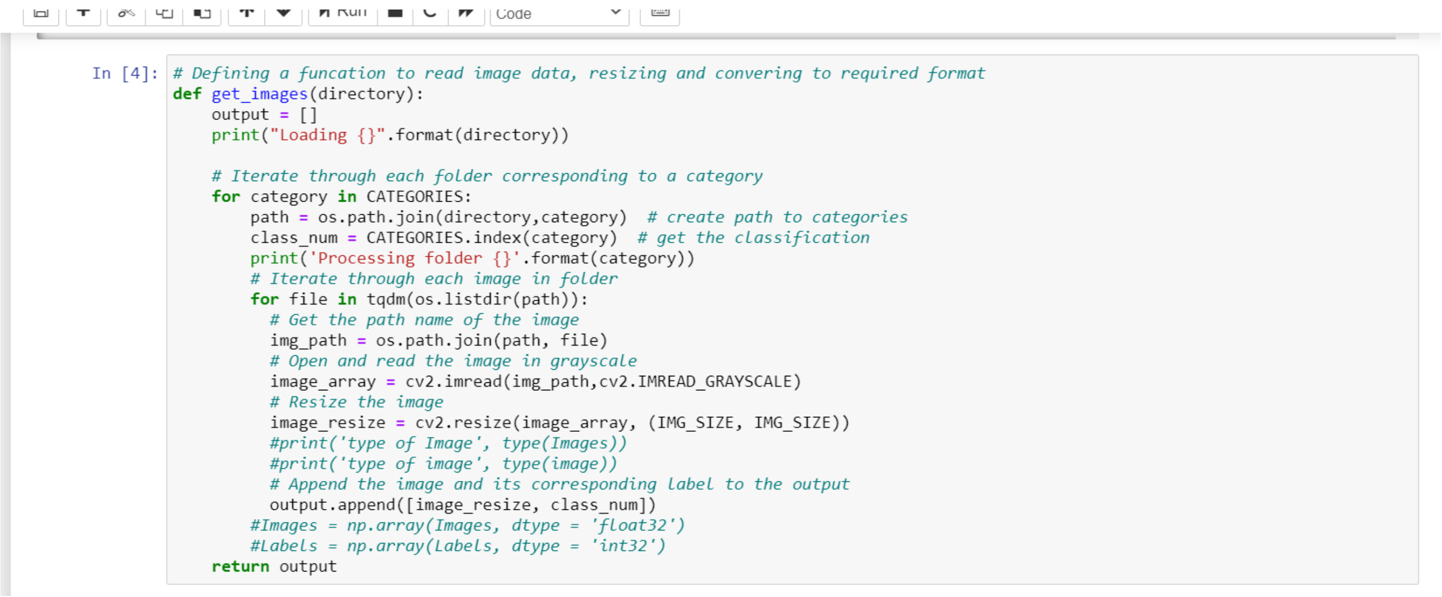


```
In [3]: # Input data calsses
        CATEGORIES = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "unknown"]

        #Visualizing the first image of each category
        for category in CATEGORIES:
            path = os.path.join(dir,category)  # create path to categories
            for img in os.listdir(path):
              # iterate over each image
              # convert to array
              print('Label of below image is {}'.format(category))
              img_array = cv2.imread(os.path.join(path,img))
              print('Image size of {} is {}'.format(img, img_array.shape))
              plt.imshow(img_array, cmap = None)  # graph it
              plt.show()  # display!

              print('Printing the graysacle image')
              img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
              plt.imshow(img_array, cmap='gray')  # graph it
              plt.show()  # display!
              print('Resized image')
              img_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
              plt.imshow(img_array, cmap='gray')
              plt.show() # Show resize image
              break  # we just want one for now so break
```

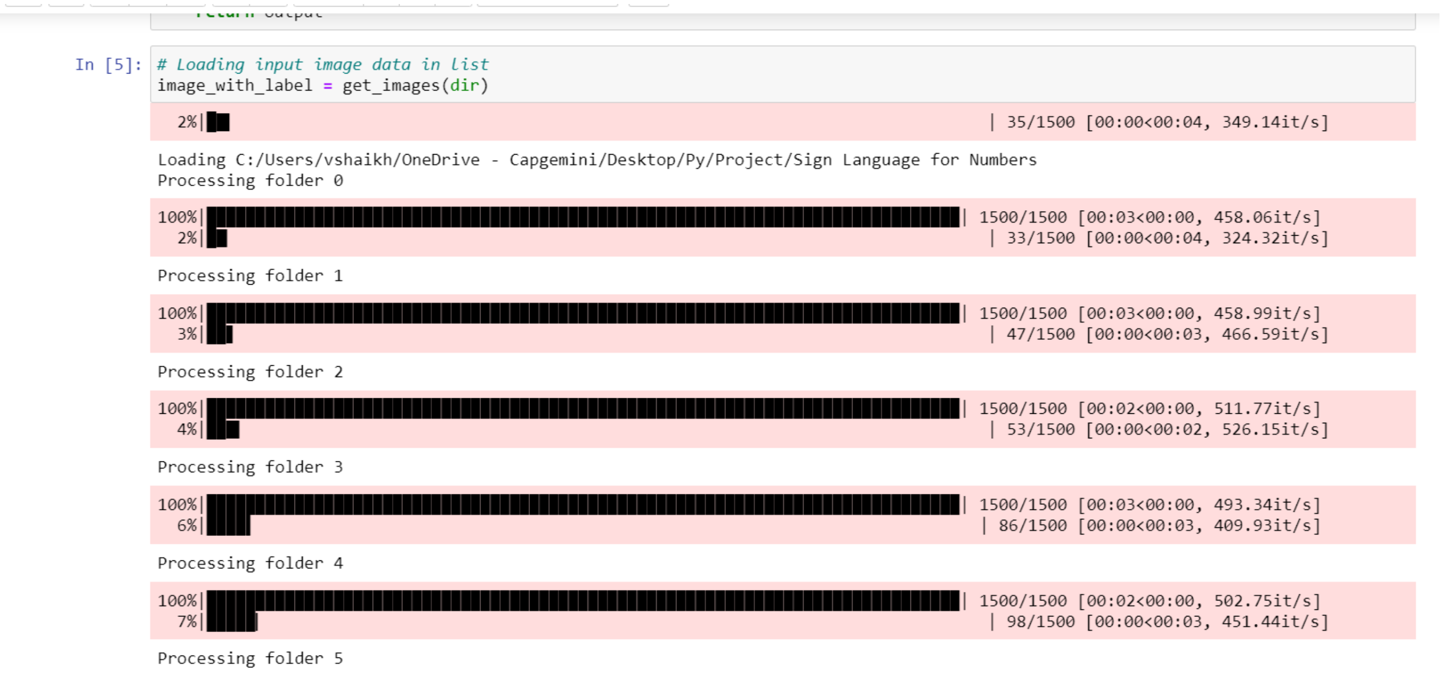


```
        break  # we just want one for now so break
Label of below image is 2
Image size of two_1.jpg is (100, 69, 3)
```

```
Printing the graysacle image
```



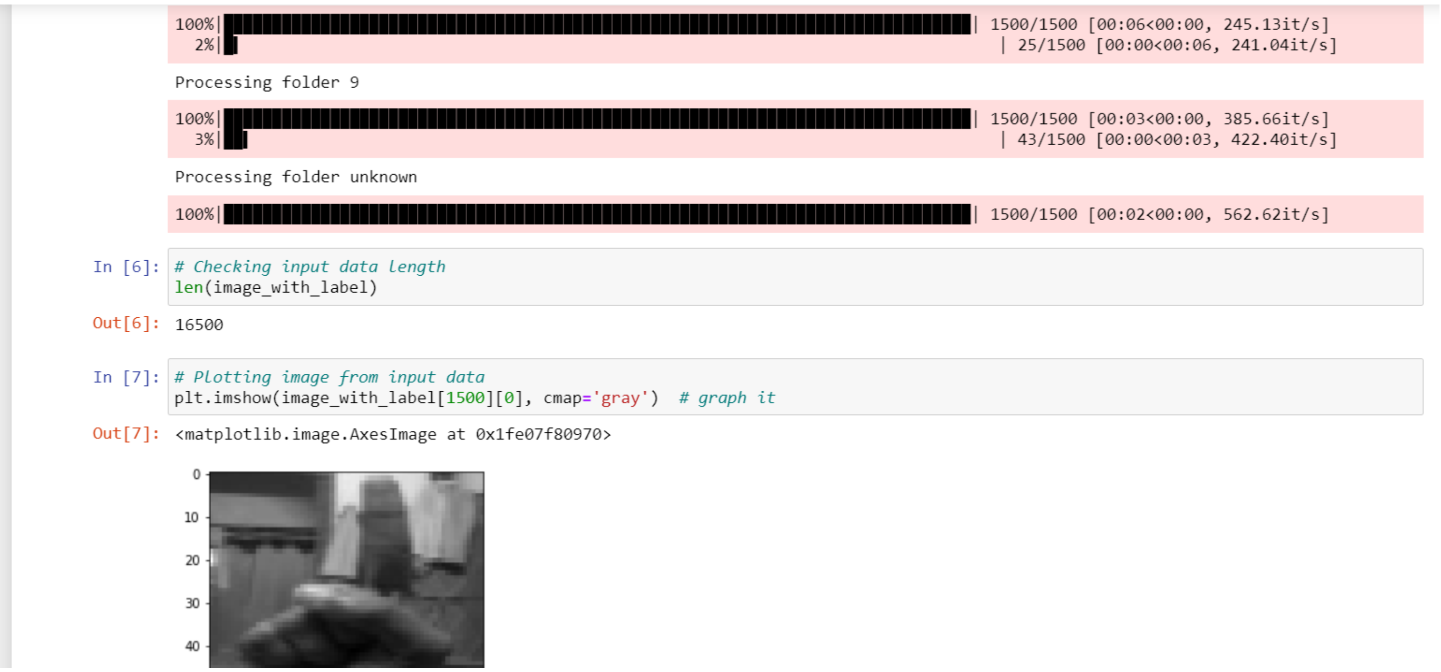
```
In [4]: # Defining a funcation to read image data, resizing and convering to required format
        def get_images(directory):
            output = []
            print("Loading {}".format(directory))

            # Iterate through each folder corresponding to a category
            for category in CATEGORIES:
                path = os.path.join(directory,category)  # create path to categories
                class_num = CATEGORIES.index(category)  # get the classification
                print('Processing folder {}'.format(category))
                # Iterate through each image in folder
                for file in tqdm(os.listdir(path)):
                  # Get the path name of the image
                  img_path = os.path.join(path, file)
                  # Open and read the image in grayscale
                  image_array = cv2.imread(img_path,cv2.IMREAD_GRAYSCALE)
                  # Resize the image
                  image_resize = cv2.resize(image_array, (IMG_SIZE, IMG_SIZE))
                  #print('type of Image', type(Images))
                  #print('type of image', type(image))
                  # Append the image and its corresponding label to the output
                  output.append([image_resize, class_num])
                #Images = np.array(Images, dtype = 'float32')
                #Labels = np.array(Labels, dtype = 'int32')
            return output
```



```
    return output
```

In [5]:
```
# Loading input image data in list
image_with_label = get_images(dir)
```

```
  2%|██                                                                    | 35/1500 [00:00<00:04, 349.14it/s]
Loading C:/Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers
Processing folder 0
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:03<00:00, 458.06it/s]
  2%|██                                                                    | 33/1500 [00:00<00:04, 324.32it/s]
Processing folder 1
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:03<00:00, 458.99it/s]
  3%|██                                                                    | 47/1500 [00:00<00:03, 466.59it/s]
Processing folder 2
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:02<00:00, 511.77it/s]
  4%|███                                                                   | 53/1500 [00:00<00:02, 526.15it/s]
Processing folder 3
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:03<00:00, 493.34it/s]
  6%|████                                                                 | 86/1500 [00:00<00:03, 409.93it/s]
Processing folder 4
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:02<00:00, 502.75it/s]
  7%|█████                                                                 | 98/1500 [00:00<00:03, 451.44it/s]
Processing folder 5
```



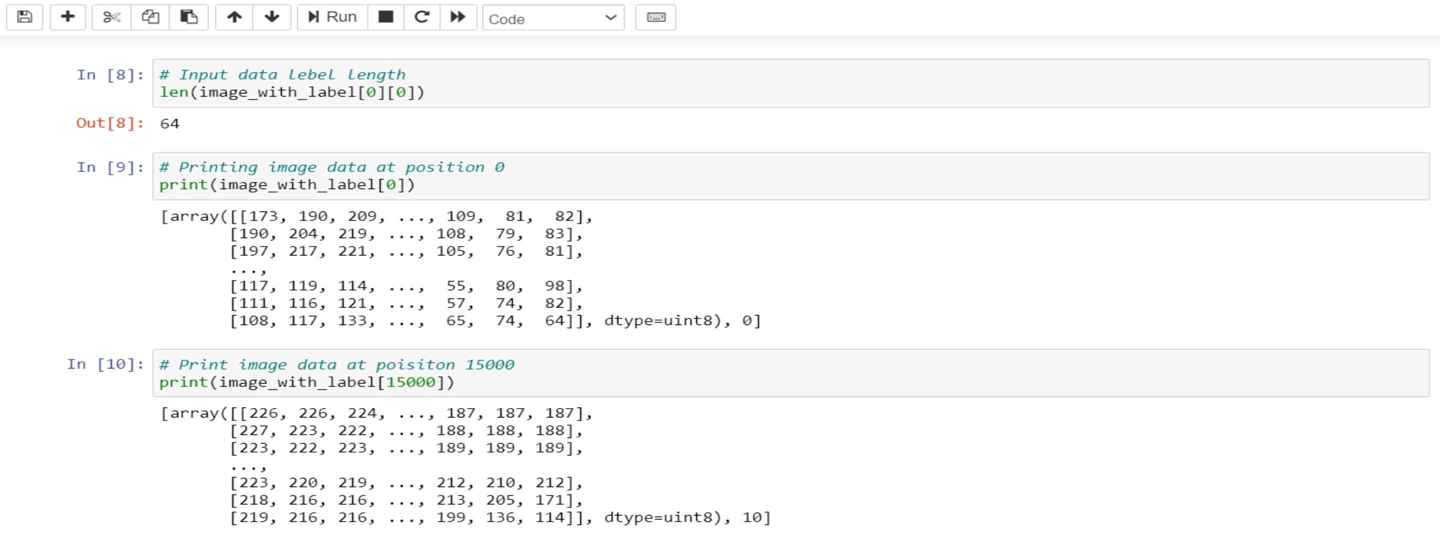
```
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:06<00:00, 245.13it/s]
  2%|█                                                                     | 25/1500 [00:00<00:06, 241.04it/s]
Processing folder 9
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:03<00:00, 385.66it/s]
  3%|██                                                                    | 43/1500 [00:00<00:03, 422.40it/s]
Processing folder unknown
100%|████████████████████████████████████████████████████████████████████| 1500/1500 [00:02<00:00, 562.62it/s]
```

In [6]:
```
# Checking input data length
len(image_with_label)
```

Out[6]:
```
16500
```

In [7]:
```
# Plotting image from input data
plt.imshow(image_with_label[1500][0], cmap='gray')  # graph it
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x1fe07f80970>
```



In [8]:
```
# Input data lebel length
len(image_with_label[0][0])
```

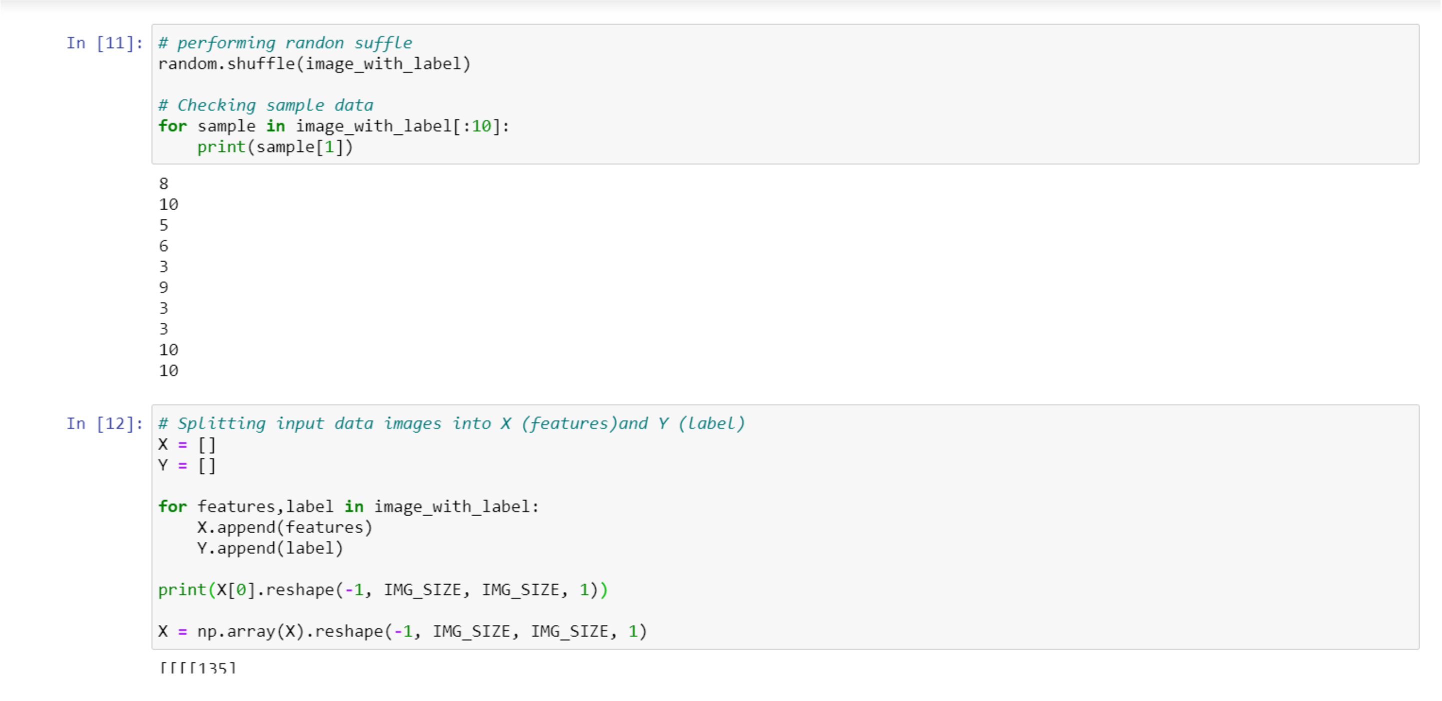
Out[8]:
```
64
```

In [9]:
```
# Printing image data at position 0
print(image_with_label[0])
```

```
[array([[173, 190, 209, ..., 109,  81,  82],
       [190, 204, 219, ..., 108,  79,  83],
       [197, 217, 221, ..., 105,  76,  81],
       ...,
       [117, 119, 114, ...,  55,  80,  98],
       [111, 116, 121, ...,  57,  74,  82],
       [108, 117, 133, ...,  65,  74,  64]], dtype=uint8), 0]
```

In [10]:
```
# Print image data at poisiton 15000
print(image_with_label[15000])
```

```
[array([[226, 226, 224, ..., 187, 187, 187],
       [227, 223, 222, ..., 188, 188, 188],
       [223, 222, 223, ..., 189, 189, 189],
       ...,
       [223, 220, 219, ..., 212, 210, 212],
       [218, 216, 216, ..., 213, 205, 171],
       [219, 216, 216, ..., 199, 136, 114]], dtype=uint8), 10]
```

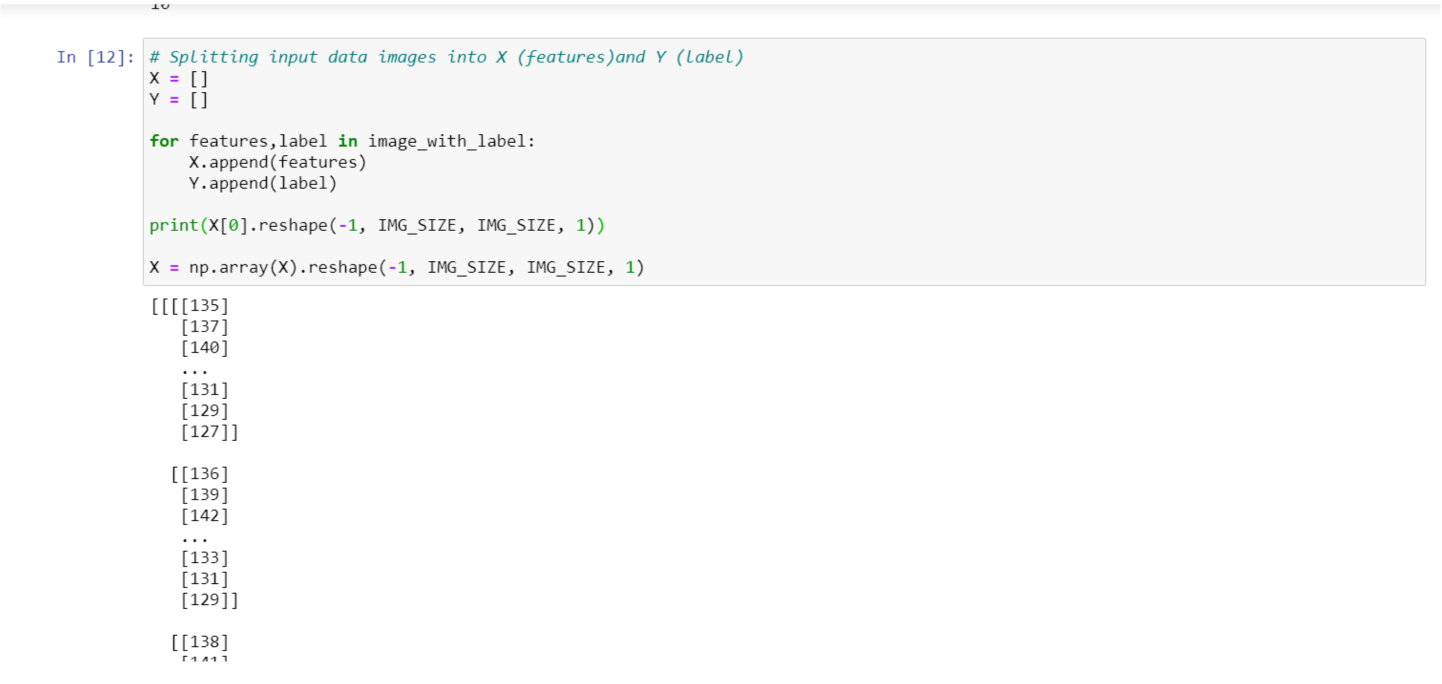


```
In [11]: # performing randon suffle
         random.shuffle(image_with_label)

         # Checking sample data
         for sample in image_with_label[:10]:
             print(sample[1])
```

```
8
10
5
6
3
9
3
3
10
10
```

```
In [12]: # Splitting input data images into X (features)and Y (label)
         X = []
         Y = []

         for features,label in image_with_label:
             X.append(features)
             Y.append(label)

         print(X[0].reshape(-1, IMG_SIZE, IMG_SIZE, 1))

         X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

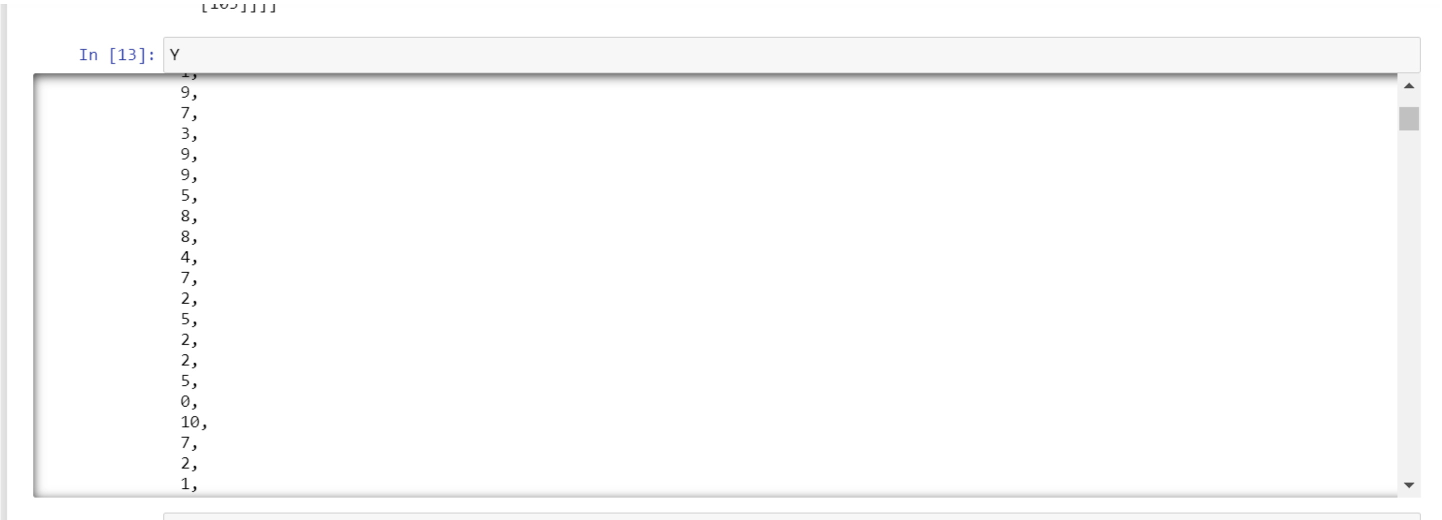
```
[[[[135]
```



```
10
```

```
In [12]: # Splitting input data images into X (features)and Y (label)
         X = []
         Y = []

         for features,label in image_with_label:
             X.append(features)
             Y.append(label)

         print(X[0].reshape(-1, IMG_SIZE, IMG_SIZE, 1))

         X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

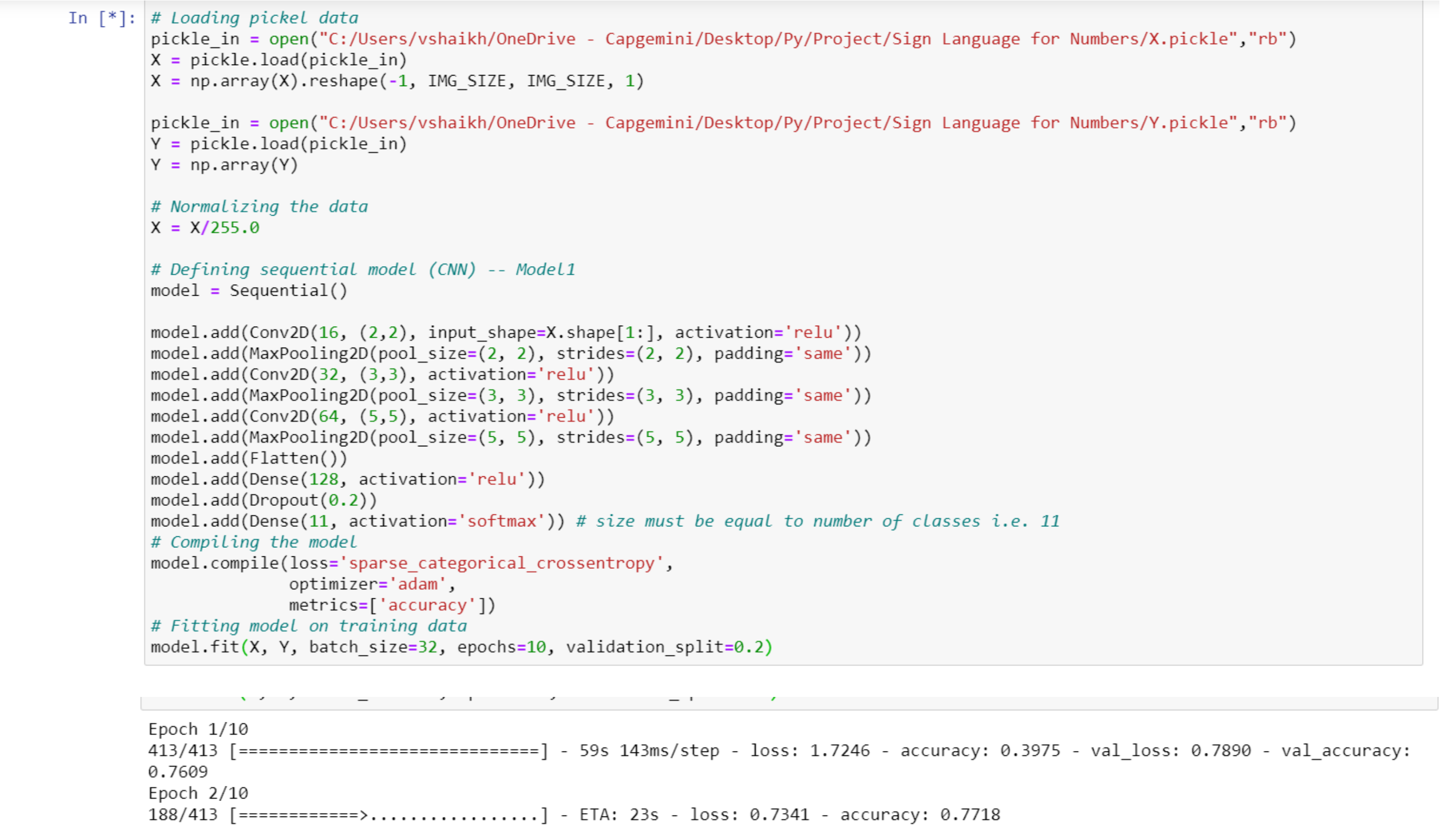
```
[[[[135]
   [137]
   [140]
   ...
   [131]
   [129]
   [127]]

  [[136]
   [139]
   [142]
   ...
   [133]
   [131]
   [129]]

  [[138]
   [141]
```



```
    [105]]]]
```

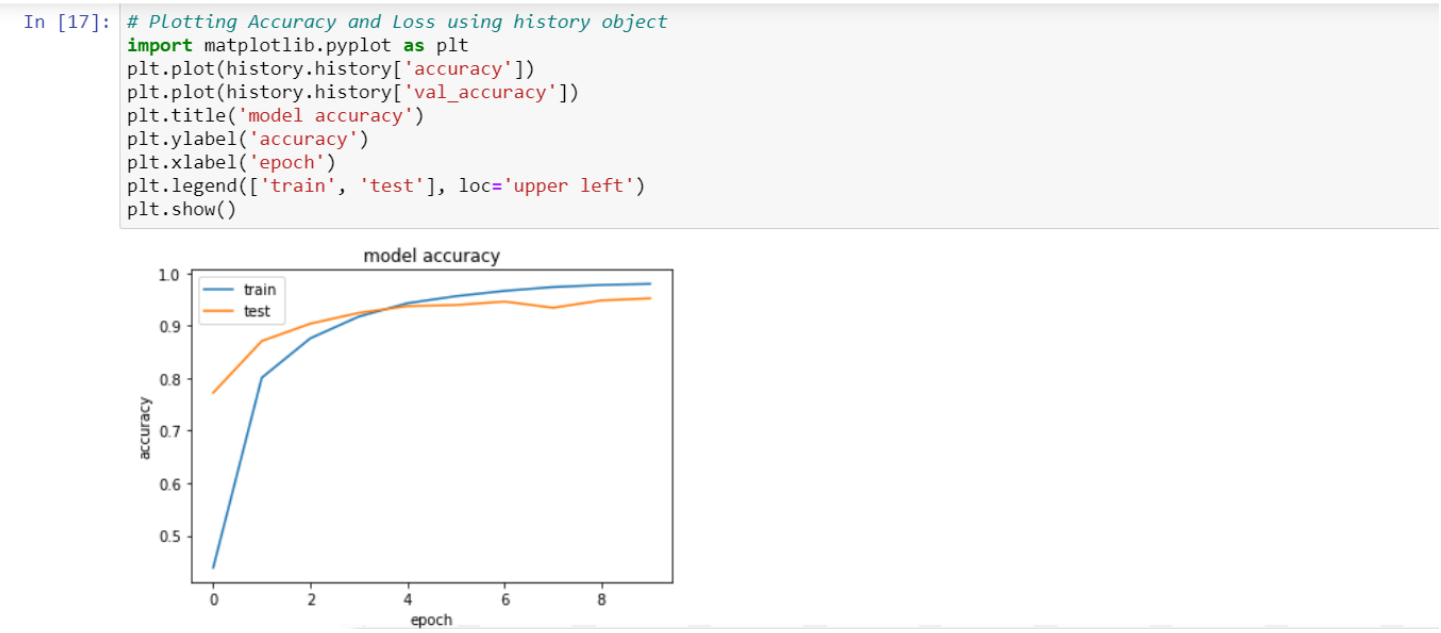
```
In [13]: Y
```

```
   1,
   9,
   7,
   3,
   9,
   9,
   5,
   8,
   8,
   4,
   7,
   2,
   5,
   2,
   2,
   5,
   0,
   10,
   7,
   2,
   1,
In [15]: # Using pickle to serialize the data
         import pickle
         pickle_out = open("C:/Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers/X.pickle","wb")
         pickle.dump(X, pickle_out)
         pickle_out.close()

         pickle_out = open("C:/Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers/Y.pickle","wb")
         pickle.dump(Y, pickle_out)
         pickle_out.close()
```

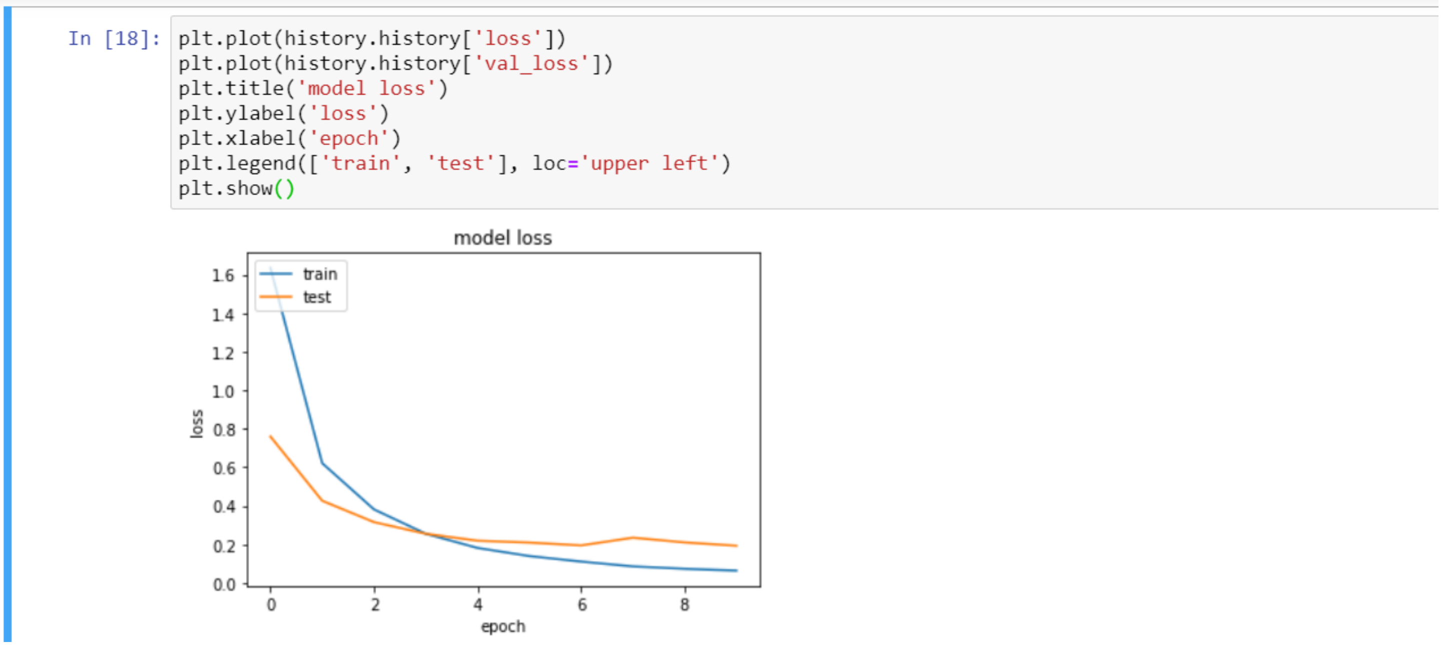


```
In [*]: # Loading pickel data
        pickle_in = open("C:/Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers/X.pickle","rb")
        X = pickle.load(pickle_in)
        X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)

        pickle_in = open("C:/Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers/Y.pickle","rb")
        Y = pickle.load(pickle_in)
        Y = np.array(Y)

        # Normalizing the data
        X = X/255.0

        # Defining sequential model (CNN) -- Model1
        model = Sequential()

        model.add(Conv2D(16, (2,2), input_shape=X.shape[1:], activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
        model.add(Conv2D(32, (3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(3, 3), strides=(3, 3), padding='same'))
        model.add(Conv2D(64, (5,5), activation='relu'))
        model.add(MaxPooling2D(pool_size=(5, 5), strides=(5, 5), padding='same'))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(11, activation='softmax')) # size must be equal to number of classes i.e. 11
        # Compiling the model
        model.compile(loss='sparse_categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])
        # Fitting model on training data
        model.fit(X, Y, batch_size=32, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
413/413 [==============================] - 59s 143ms/step - loss: 1.7246 - accuracy: 0.3975 - val_loss: 0.7890 - val_accuracy:
0.7609
Epoch 2/10
188/413 [============>.................] - ETA: 23s - loss: 0.7341 - accuracy: 0.7718
```



```
In [17]: # Plotting Accuracy and Loss using history object
         import matplotlib.pyplot as plt
         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()
```



```
In [18]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()
```

Work sharing/Module sharing between teammates:

Prabhajan Trivedi –

1. Worked on prepressing of data
2. Now working on building model 2

Harshita Patil –

1. Working on writing preprocessed data to CSV file so that it can be shared across multiple models that we are planning to implement.
2. Worked on building first model 1

Vasim Shaikh –

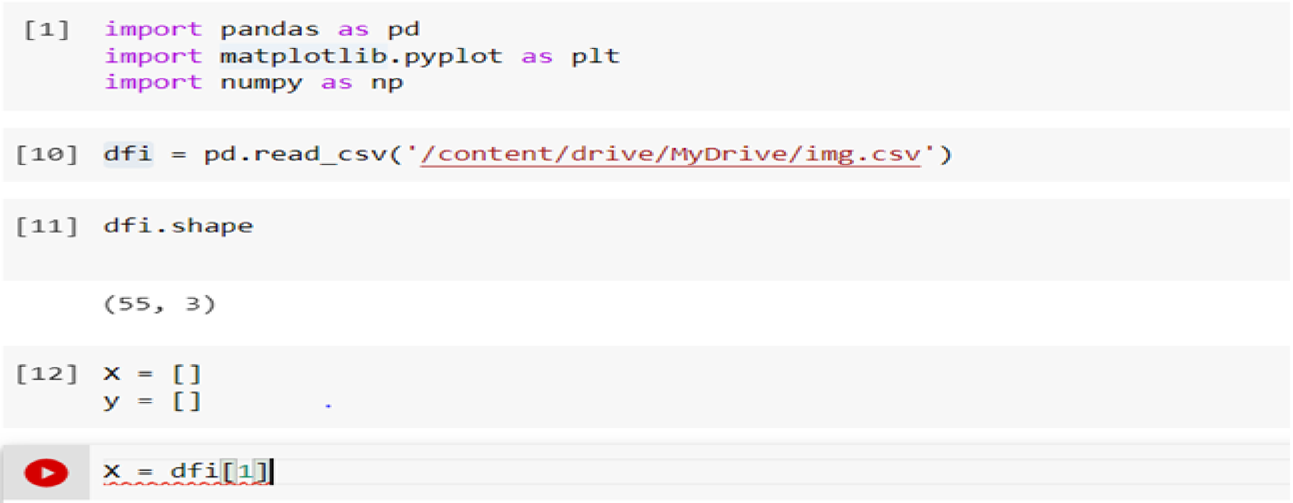
1. Used pre-processed data and worked on building first model and training the model 1
2. Worked on documentation and collaborating with team members

Shireesha Maddi –

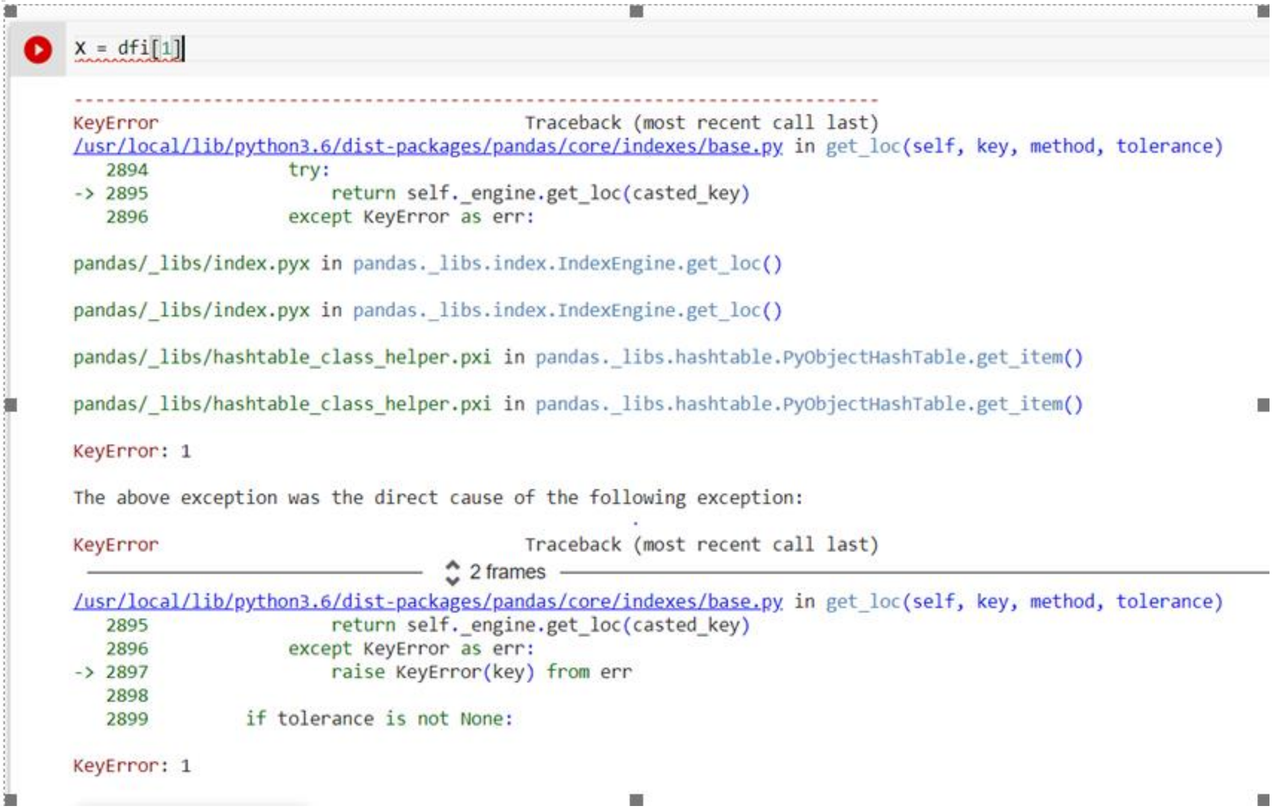
1. Worked on data preprocessing along with Prabhanjan T
2. Now working on model 3 and visualization

Any issues, blockages with the project:

1. Converting image data in to CSV so that it can used as an input for multiple models



```
[1]  import pandas as pd
     import matplotlib.pyplot as plt
     import numpy as np

[10] dfi = pd.read_csv('/content/drive/MyDrive/img.csv')

[11] dfi.shape

     (55, 3)

[12] x = []
     y = []

     x = dfi[1]
```



```
x = dfi[1]

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2894             try:
-> 2895                 return self._engine.get_loc(casted_key)
   2896             except KeyError as err:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 1

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                   2 frames
/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2895                 return self._engine.get_loc(casted_key)
   2896             except KeyError as err:
-> 2897                 raise KeyError(key) from err
   2898 
   2899         if tolerance is not None:

KeyError: 1
```

2. Unable to upload data to Github due to size issue

Next Steps –

1. Complete remaining model building and training
2. Implementation of ensemble model
3. Model evaluation

Github Link - https://github.com/vasimshaikh39/python2020

References:

1. https://www.kaggle.com/muhammadkhalid/sign-language-for-numbers
2. https://heartbeat.fritz.ai/introduction-to-machine-learning-model-evaluation-fa859e1b2d7f
3. https://data-flair.training/blogs/sign-language-recognition-python-ml-opencv/
4. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
5. https://www.kaggle.com/joshbeau/numerical-sign-language-recognition-tensorflow