Handwritten Character Recognition with Neural Network

In this machine learning project, we will recognize handwritten characters, i.e, English alphabets from A-Z. This we are going to achieve by modeling a neural network that will have to be trained over a dataset containing images of alphabets.

Project Prerequisites

Below are the prerequisites for this project:

1. Python (3.7.4 used)
2. IDE (Jupyter used)

Required frameworks are

1. Numpy (version 1.16.5)
2. cv2 (openCV) (version 3.4.2)
3. Keras (version 2.3.1)
4. Tensorflow (Keras uses TensorFlow in backend and for some image preprocessing) (version 2.0.0)
5. Matplotlib (version 3.1.1)
6. Pandas (version 0.25.1)

Steps to develop handwritten character recognition

Download Project Code

Please download project source code: **Handwritten Character Recognition with Neural Network**

```
1.   import matplotlib.pyplot as plt
2.   import cv2
3.   import numpy as np
4.   from keras.models import Sequential
5.   from keras.layers import Dense,
     Flatten, Conv2D, MaxPool2D, Dropout
6.   from keras.optimizers import SGD, Adam
7.   from keras.callbacks import
     ReduceLROnPlateau, EarlyStopping
8.   from keras.utils import to_categorical
9.   import pandas as pd
10.  import numpy as np
11.  from sklearn.model_selection import
     train_test_split
12.  from sklearn.utils import shuffle
```

- First of all, we do all the necessary imports as stated above. We will see the use of all the imports as we use them.

Read the data:

```
1.   data = pd.read_csv(r"D:\a-z
     alphabets\A_Z Handwritten
     Data.csv").astype('float32')
2.
3.   print(data.head(10))
```

- Now we are reading the dataset using the **pd.read_csv()** and printing the first 10 images using **data.head(10)**

Jupyter Handwritten_text_recognition Last Checkpoint: 20 hours ago (unsaved changes)

Logout

File   Edit   View   Insert   Cell   Kernel   Widgets   Help

Trusted   Python 3 O

Run   Code

Out[234]:

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	...	0.639	0.640	0.641	0.642	0.643	0.644	0.645	0.646	0.647	0.648
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10 rows × 785 columns

In [ ]:
```
1
```

In [189]:
```
1  # Reshaping the data in csv file so that it can be displayed as an image...
2
3  train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
4  train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
5  test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
```

In [190]:
```
1  print("Train data shape: ", train_x.shape)
2  print("Test data shape: ", test_x.shape)
```

Train data shape: (297960, 28, 28)
Test data shape: (74490, 28, 28)

In [99]:
```
1  word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',17
```

(The above image shows some of the rows of the dataframe data using the head() function of dataframe)

Split data into images and their labels:

```
1.   X = data.drop('0',axis = 1)
2.   y = data['0']
```

Splitting the data read into the images & their corresponding labels. The '0' contains the labels, & so we drop the '0' column from the data dataframe read & use it in the y to form the labels.

Reshaping the data in the csv file so that it can be displayed as an image

```
1.   train_x, test_x, train_y, test_y =
     train_test_split(X, y, test_size = 0.2)
2.
3.   train_x = np.reshape(train_x.values,
     (train_x.shape[0], 28,28))
4.   test_x = np.reshape(test_x.values,
     (test_x.shape[0], 28,28))
5.
6.   print("Train data shape: ",
     train_x.shape)
7.   print("Test data shape: ", test_x.shape)
```

- In the above segment, we are splitting the data into training & testing dataset using train_test_split().
- Also, we are reshaping the train & test image data so that they can be displayed as an image, as initially in the CSV file they were present as 784 columns of pixel data. So we convert it to 28×28 pixels.

```
1.   word_dict =
     {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G
     24:'Y',25:'Z'}
```

Plotting the number of alphabets in the dataset

```
1.    y_int = np.int0(y)
2.    count = np.zeros(26, dtype='int')
3.    for i in y_int:
4.        count[i] +=1
5.
6.    alphabets = []
7.    for i in word_dict.values():
8.        alphabets.append(i)
9.
10.   fig, ax = plt.subplots(1,1, figsize=
      (10,10))
11.   ax.barh(alphabets, count)
12.
13.   plt.xlabel("Number of elements ")
14.   plt.ylabel("Alphabets")
15.   plt.grid()
16.   plt.show()
```

- Here we are only describing the distribution of the alphabets.
- Firstly we convert the labels into integer values and append into the count list according to the label. This count list has the number of images present in the dataset belonging to each alphabet.
- Now we create a list – alphabets containing all the characters using the values() function of the dictionary.
- Now using the count & alphabets lists we draw the horizontal bar plot.

```
18  plt.show()
```
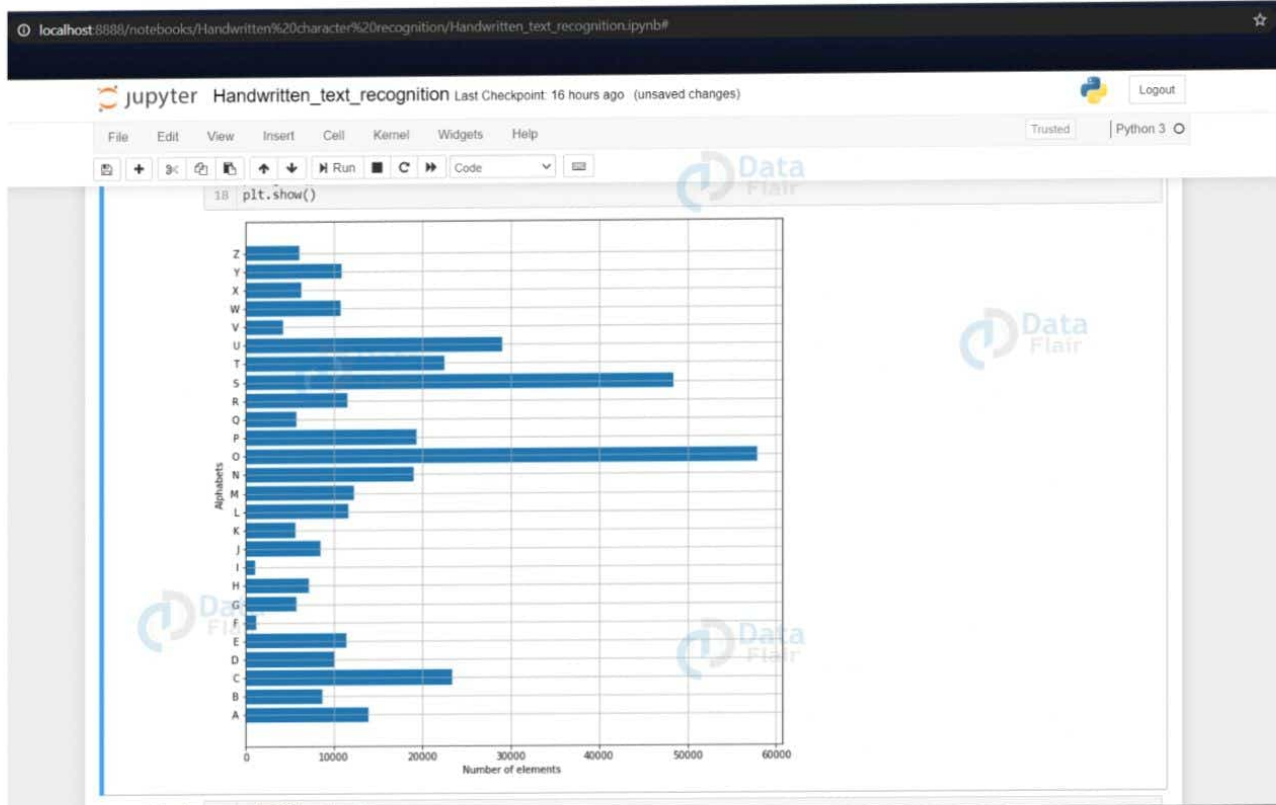

Shuffling the data

```
1.   shuff = shuffle(train_x[:100])
2.
3.   fig, ax = plt.subplots(3,3, figsize =
     (10,10))
4.   axes = ax.flatten()
5.
6.   for i in range(9):
7.       _, shu = cv2.threshold(shuff[i], 30,
     200, cv2.THRESH_BINARY)
8.       axes[i].imshow(np.reshape(shuff[i],
     (28,28)), cmap="Greys")
9.   plt.show()
```

- Now we shuffle some of the images of the train set.
- The shuffling is done using the shuffle() function so that we can display some random images.
- We then create 9 plots in 3×3 shape & display the thresholded images of 9 alphabets.
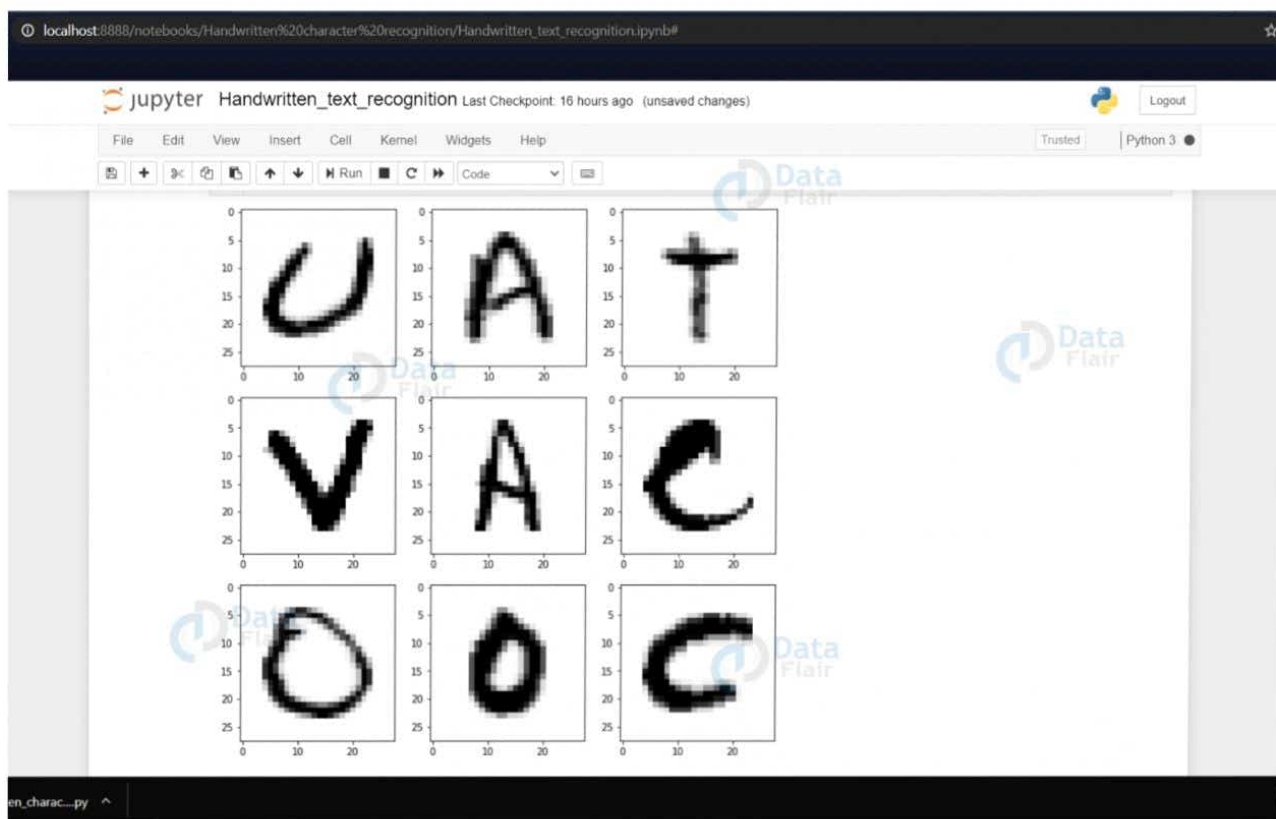
(The above image depicts the grayscale images that we got from the dataset)

Data Reshaping

Reshaping the training & test dataset so that it can be put in the model

```
1.    train_X =
      train_x.reshape(train_x.shape[0], ... x.shape[1], train_x...
2.    print("New shape of train data:  ", train...

3.

4.    test_X = test_x.reshape(test_x.shape[0],
      test_x.shape[2],1)
5.    print("New shape of train data: ", test_X...

6.

7.

8.    Now we reshape the train & test image data
      put in the model.

9.

10.   New shape of train data:  (297960, 28, 28
11.   New shape of train data:  (74490, 28, 28
```

Now we reshape the train & test image dataset so that they can be put in the model.

New shape of train data: (297960, 28, 28, 1)

New shape of train data: (74490, 28, 28, 1)

```
1.    train_yOHE = to_categorical(train_y,
      num_classes = 26, dtype='int')
2.    print("New shape of train labels: ",
      train_yOHE.shape)
3.
4.    test_yOHE = to_categorical(test_y,
      num_classes = 26, dtype='int')
5.    print("New shape of test labels: ",
      test_yOHE.shape)
```

Here we convert the single float values to categorical values. This is done as the CNN model takes input of labels & generates the output as a vector of probabilities.

Now we define the CNN.

What is CNN?

CNN stands for Convolutional Neural Networks that are used to extract the features of the images using several layers of filters.
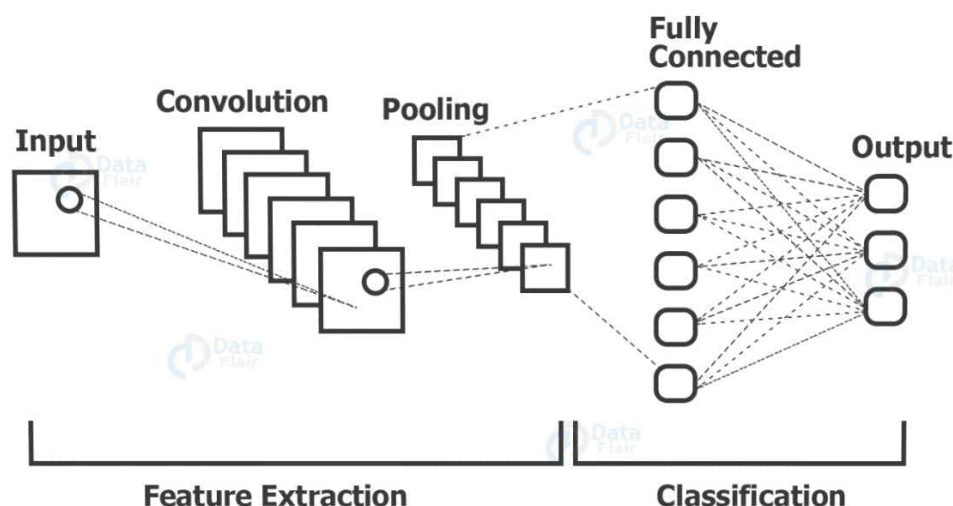
The convolution layers are generally followed by maxpool layers that are used to reduce the number of features extracted and ultimately the output of the maxpool and layers and convolution layers are flattened into a vector of single dimension and are given as an input to the Dense layer (The fully connected network).

The model created is as follows:

```
1.   model = Sequential()
2.
3.   model.add(Conv2D(filters=32,
     kernel_size=(3, 3), activation='relu',
     input_shape=(28,28,1)))
4.   model.add(MaxPool2D(pool_size=(2, 2),
     strides=2))
5.
6.   model.add(Conv2D(filters=64,
     kernel_size=(3, 3), activation='relu',
     padding = 'same'))
7.   model.add(MaxPool2D(pool_size=(2, 2),
     strides=2))
8.
9.   model.add(Conv2D(filters=128,
     kernel_size=(3, 3), activation='relu',
     padding = 'valid'))
10.  model.add(MaxPool2D(pool_size=(2, 2),
     strides=2))
11.
12.  model.add(Flatten())
13.
14.  model.add(Dense(64,activation ="relu"))
15.  model.add(Dense(128,activation ="relu"))
16.
17.  model.add(Dense(26,activation
     ="softmax"))
```

Above we have the CNN model that we designed for training the model over the training dataset.
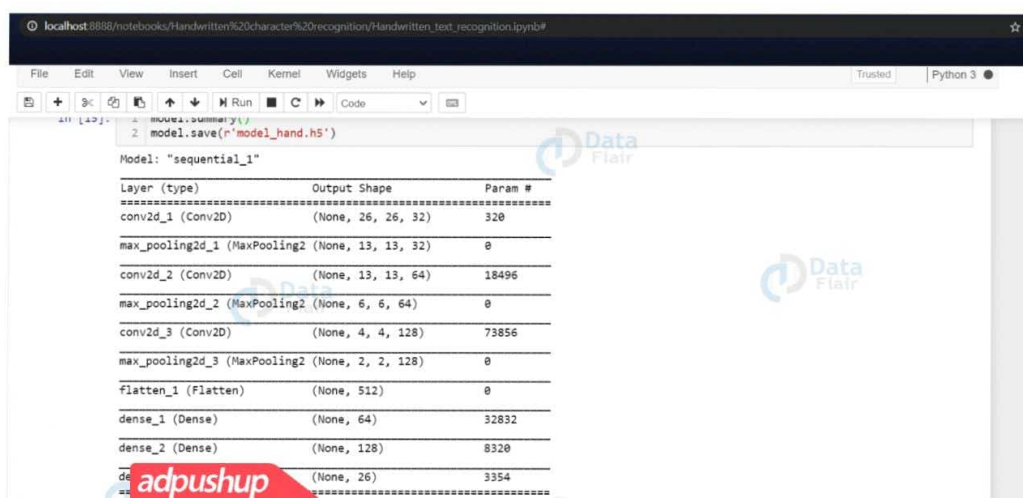
Compiling & Fitting Model

```
1.   model.compile(optimizer =
     Adam(learning_rate=0.001),
     loss='categorical_crossentropy', metrics=
     ['accuracy'])
2.
3.   history = model.fit(train_X, train_yOHE,
     epochs=1,  validation_data =
     (test_X,test_yOHE))
```

- Here we are compiling the model, where we define the optimizing function & the loss function to be used for fitting.
- The optimizing function used is Adam, that is a combination of RMSprop & Adagram optimizing algorithms.
- The dataset is very large so we are training for only a single epoch, however, as required we can even train it for multiple epochs (which is recommended for character recognition for better accuracy).

```
1.   model.summary()
2.   model.save(r'model_hand.h5')
```

Now we are getting the model summary that tells us what were the different layers defined in the model & also we save the model using **model.save()** function.

```
Jupyter  Handwritten_text_recognition  Last Checkpoint: 16 hours ago  (unsaved changes)            Logout

File   Edit   View   Insert   Cell   Kernel   Widgets   Help                          Trusted  | Python 3 O

In [52]:   1  print("The validation accuracy is :", history.history['val_accuracy'])
           2  print("The training accuracy is :", history.history['accuracy'])
           3  print("The validation loss is :", history.history['val_loss'])
           4  print("The training loss is :", history.history['loss'])

           The validation accuracy is : [0.975057065486908]
           The training accuracy is : [0.94837564]
           The validation loss is : [0.0921439248674415]
           The training loss is : [0.1830781325123433]

In [99]:   1  word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',17

In [79]:   1  #Making model predictions...
           2
           3  pred = model.predict(test_X[:9])
           4  print(test_X.shape)

           (74490, 28, 28, 1)

In [162]:  1  fig, axes = plt.subplots(3,3, figsize=(8,9))
           2  axes = axes.flatten()
           3
           4  for i,ax in enumerate(axes):
           5      img = np.reshape(test_X[i], (28,28))
           6      ax.imshow(img, cmap="Greys")
           7      pred = word_dict[np.argmax(test_yOHE[i])]
           8      ax.set_title("Prediction: "+pred)
           9      ax.grid()
          10
          11  y = []
          12  for i in pred:
          13      y.append(np.argmax(i))
          14  print("Predicted values: ", y)
          15
          16  y_labels = []
```
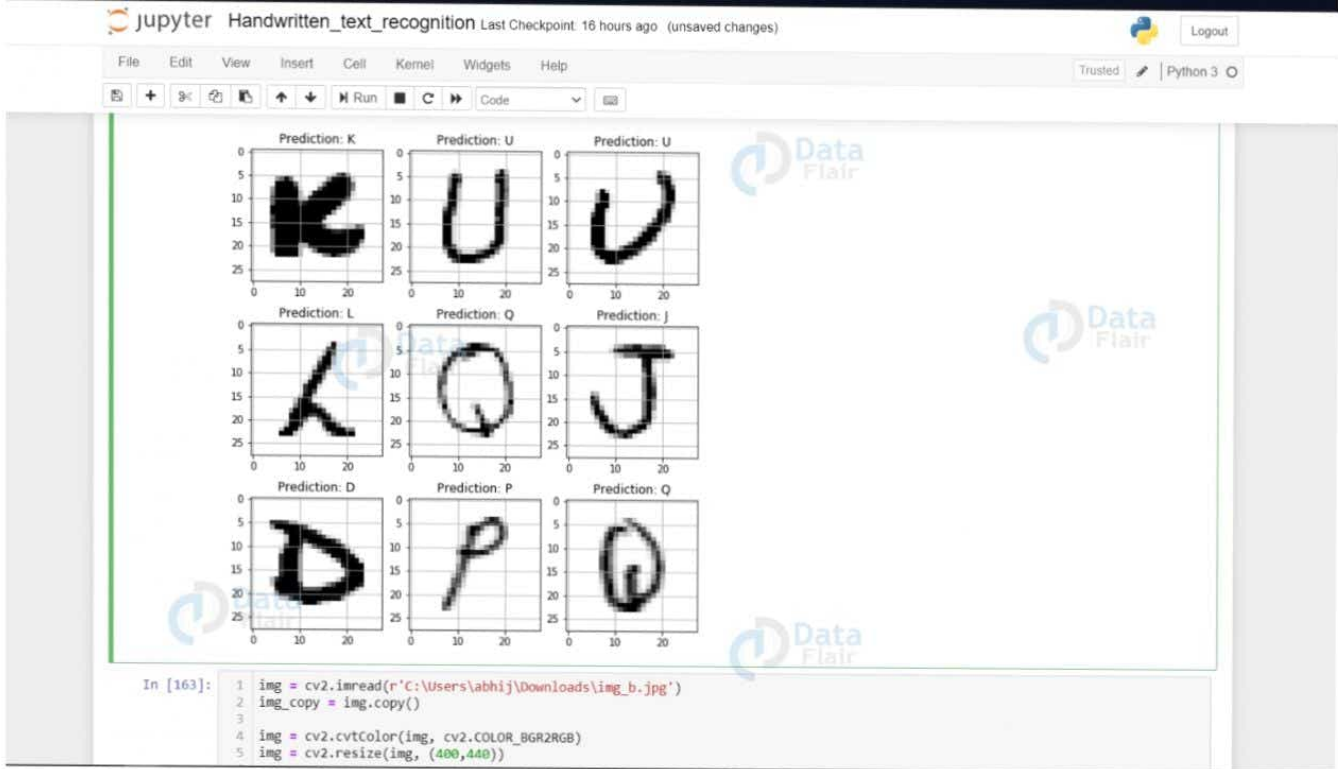
Doing Some Predictions on Test Data

```
1.   fig, axes = plt.subplots(3,3, figsize=
     (8,9))

2.   axes = axes.flatten()

3.

4.   for i,ax in enumerate(axes):
5.       img = np.reshape(test_X[i],
     (28,28))
6.       ax.imshow(img, cmap="Greys")

7.

8.       pred =
     word_dict[np.argmax(test_yOHE[i])]
9.       ax.set_title("Prediction: "+pred)
10.      ax.grid()
```

- Here we are creating 9 subplots of (3,3) shape & visualize some of the test dataset alphabets along with their predictions, that are made using the **model.predict()** function for text recognition.

```
In [163]:    1  img = cv2.imread(r'C:\Users\abhij\Downloads\img_b.jpg')
             2  img_copy = img.copy()
             3
             4  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
             5  img = cv2.resize(img, (400,440))
```

Doing Prediction on External Image

```
1.    img =
      cv2.imread(r'C:\Users\abhij\Downloads\img.
2.    img_copy = img.copy()

3.

4.    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGI
5.    img = cv2.resize(img, (400,440))
```

- Here we have read an external image that is originally an image of alphabet 'B' and made a copy of it that is to go through some processing to be fed to the model for the prediction that we will see in a while.

- The img read is then converted from BGR representation (as OpenCV reads the image in BGR format) to RGB for displaying the image, & is resized to our required dimensions that we want to display the image in.

```
1.    img_copy = cv2.GaussianBlur(img_copy,
      (7,7), 0)
2.    img_gray = cv2.cvtColor(img_copy,
      cv2.COLOR_BGR2GRAY)
3.    _, img_thresh = cv2.threshold(img_gray,
      100, 255, cv2.THRESH_BINARY_INV)
4.
5.    img_final = cv2.resize(img_thresh,
      (28,28))
6.    img_final =np.reshape(img_final,
      (1,28,28,1))
```

- Now we do some processing on the copied image (img_copy).
- We convert the image from BGR to grayscale and apply thresholding to it. We don't need to apply a threshold we could use the grayscale to predict, but we do it to keep the image smooth without any sort of hazy gray colors in the image that could lead to wrong predictions.
- The image is to be then resized using **cv2.resize()** function into the dimensions that the model takes as input, along with reshaping the image using **np.reshape()** so that it can be used as model input.

```
1.    img_pred =
      word_dict[np.argmax(model.predict(img_fin:
2.
3.    cv2.putText(img, "Dataflair _ _ _ ", (20
      cv2.FONT_HERSHEY_TRIPLEX, 0.7, color =
      (0,0,230))
4.    cv2.putText(img, "Prediction: " + img_pr
      (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, c
      = (255,0,30))
5.    cv2.imshow('Dataflair handwritten charac
      recognition _ _ _ ', img)
```

- Now we make a prediction using the processed image & use the np.argmax() function to get the index of the class with the highest predicted pr Using this we get to know the exact

```
1.    while (1):
2.        k = cv2.waitKey(1) & 0xFF
3.        if k == 27:
4.            break
5.    cv2.destroyAllWindows()
```

- Here we are setting up a waitKey in a while loop that will be stuck in loop until Esc is pressed, & when it gets out of loop using cv2.destroyAllWindows() we destroy any active windows created to stop displaying the frame.
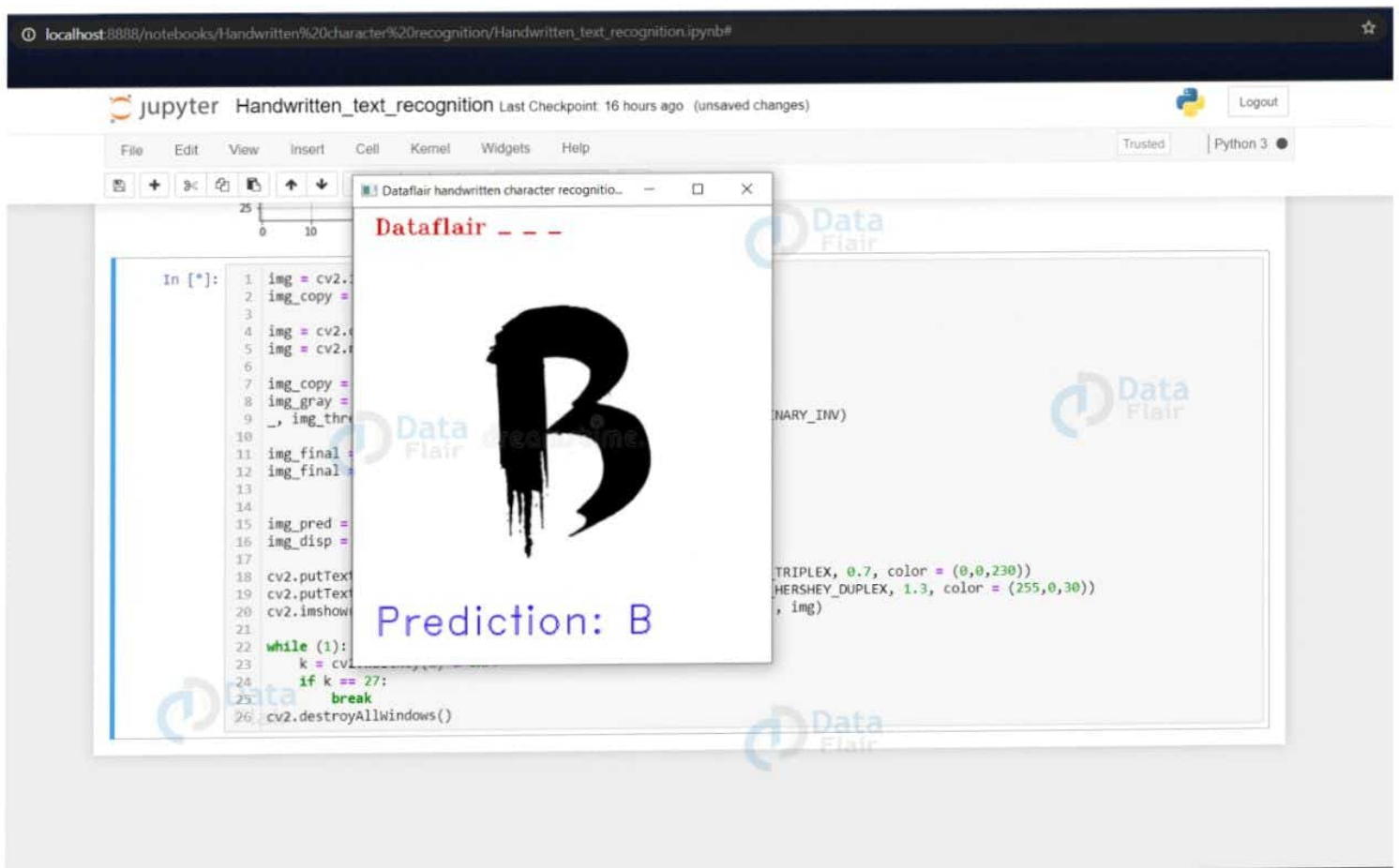
Conclusion
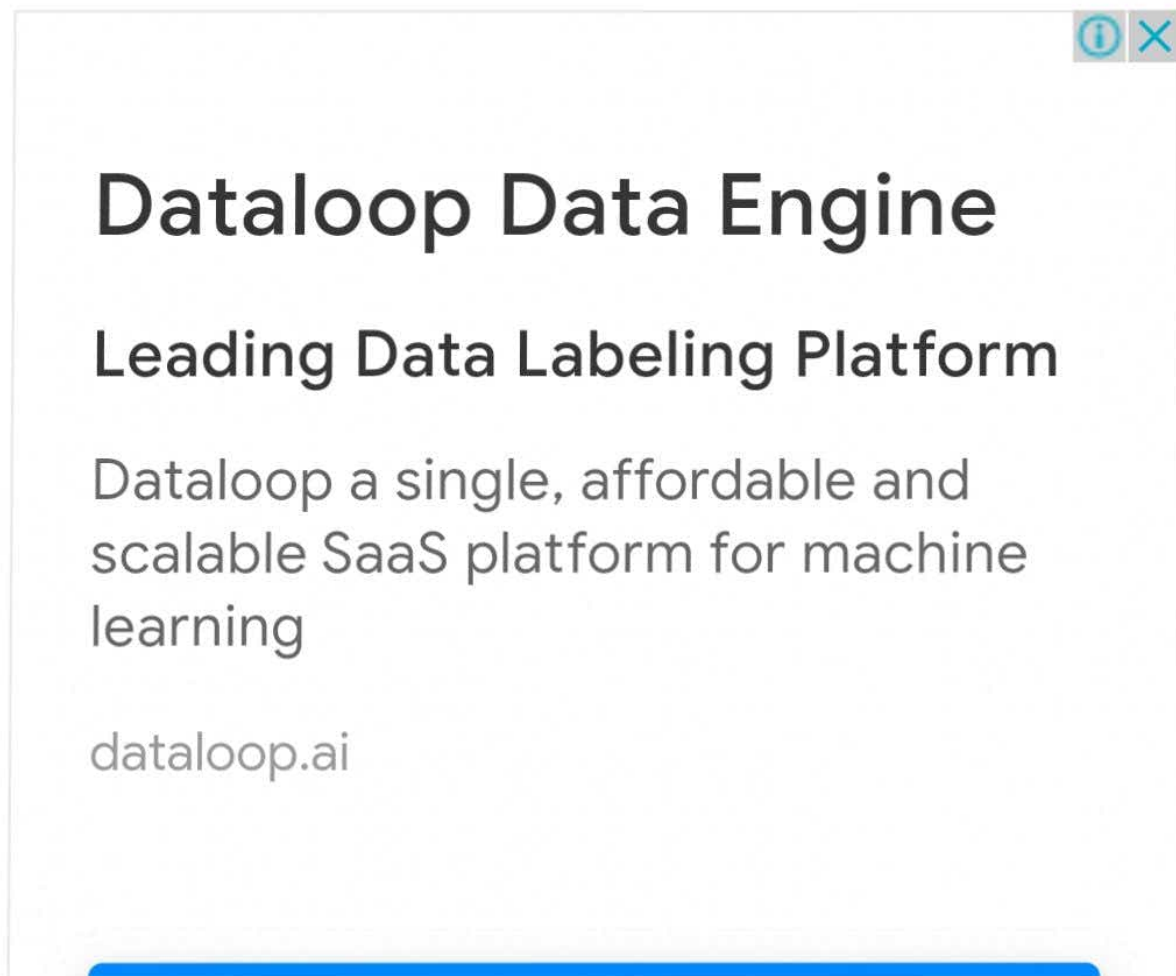
We have successfully developed Handwritten character recognition (Text Recognition) with Python, Tensorflow, and Machine Learning libraries.

Handwritten characters have been recognized with more than 97% test accuracy. This can be also further extended to identifying the handwritten characters of other languages too.