HANDWRITTEN TEXT RECOGNITION

Kartik Saini, Khushi Sharma, Akshaj Agarwal, Kishan Jayan

[#]*Computer Science Engineering-Artificial Intelligence Machine learning Department*

ABES Engineering College

Khushi.20B1531039@abes.ac.in
Akshaj.20b1531044@abes.ac.in
Kartik.20B1531055@abes.ac.in
Kishan.20B1531001@abes.ac.in

Abstract

The goal of this project is to give and output that displays the best textual representation of given handwritten characters. A system is also described that was built to recognize handwriting of many languages. The input to the system is an ink along with the language it should be interpreted in. Our primary focus is to build this system for mobile devices using touch screens but it can be built for powerful machines for recognition in the cloud as well. We aim to use machine learning as much as possible. We built results from existing techniques. Some challenges in handwriting recognition are like strong variability in writing style between different groups of people, variability in writing speed, sloppiness and ambiguities like accidental apostrophe or dot. Combination of time and position based interpretation of input is implemented to recognize overlapping writing and delayed marks. A segmenter is trained as a filter to determine which hypothetical cut points are valid character segmentation. CNN (Convolutional Neural Network) as a powerful feature can also be applied to extract feature of a handwritten character and linear SVM (Support Vector Machine) as a high end classifier. In this project we would be creating a model that would be used to read handwriting digits, characters and words from the image using the concept of CNN and an essential technique known as elastic distortion that vastly expand the size of training set.

KEYWORDS: Segmenter, CNN (Convolutional Neural Network), SVM (Support Vector Machine), Machine Learning, Character Segmentation, Elastic Distortion, Character Segmentation.

Introduction

Handwriting Recognition or Handwritten Text Recognition is the ability of a computer to receive and interpret handwritten input from sources like documents, photographs, touch-screen and other devices. The image of the written text may be sensed "off line" from a piece of paper by optical character recognition. Similarly, the movements of the pen tip may be sensed "on line", for example by a pen-based computer screen surface. There exists several techniques to recognize handwriting depending on the language but some of them are accurate and some not. Most of the existing systems use machine learning mechanisms such as neural networks. Some of the frequently used techniques involves feature extraction,segmentation, classification, training on dataset, recognition. Handwriting recognition has been a challenging task because recognition depends on various factors like different people have different style of writing, size and shape of characters varies from person to person. Language dependency is another challenge as the shape of the characters takes new forms. Also some accidental dots or apostrophe may create ambiguity during recognition. But vast research is still going on and improvements are being made on the handwriting recognition system.

Problem Statement

The goal is to create a model which can recognize the digits, it can be extended to letters and an individual's handwriting. The major goal of the proposed system is understanding Convolutional Neural Network, and applying it to the handwritten recognition system.

SUMMARY OF DIFFERENT RESEARCH-ARTICLES BASED ON HANDWRITTEN TEXT RECOGNITION

PAPER NAME	DATASET	TECHNIQUE	RESULT
[1]Handwriting Recognition on Form Document Using Convolutional Neural Network and Support Vector Machines (CNN-SVM)	NIST SD 19 2nd edition dataset It consists of numeral, uppercase, lowercase and merger of uppercase and lowercase.	CONVOLUTIONAL NEURAL NETWORK AND SUPPORT VECTOR MACHINES	CNN as a powerful feature extraction method applied to extract the feature of the handwritten characters and linear SVM using L1 loss function and L2 regularization used as end classifier.
[2] Multi-Language Online Handwriting Recognition	Single-Character Experiments on **UNIPEN-1** -R01/V07 version of the UNIPEN-1 data with a fixed 2:1 random partition into training and test data Word Recognition Experiments on **IAM-OnDB -** performed experiments on the test set IAM-OnDB t2 of the IAM database [38] that contains 3,859 items.	PREPROCESSING, SEARCH LATTICE CREATION, LATTICE DECODING, TRAINING	Presented the architecture of a real-world, multi-script and multi-language online handwriting recognition system. The system combines several existing and some new components. A key emphasis of the system is on the re-use of components across many scripts and languages, which makes the problem tractable as an engineering problem
[3]Offline Handwriting Recognition Using LSTM Recurrent Neural Networks	The training dataset consisted of 118 scanned pages of handwritten medieval Latin texts from two sources - KNMP Chronicon Boemorum and Stanford CCCC	Recurrent Neural Networks, Connectionist Temporal Classification approach, Sequence-to-Sequence Learning approach.	Seqence to Sequence learning approach was better at predicting short words, but CTC model had higher accuracy predicting longer words.
[4]Fast multi-language LSTM-based online handwriting recognition	**IAM-OnDB** contains forms of handwritten English text acquired on a whiteboard. **IBM-UB-1**contains free form cursive handwritten pages in English. And their internal dataset.	Bidirectional long short-term memory recurrent neural networks	Described the online handwriting recognition system that is currently in use at Google for 102 languages in 26 scripts. The system is based on an end-to-end trained neural network. Recognition accuracy of the new system improves by 20–40%

[5]HANDWRITTEN CHARACTER RECOGNITION USING CNN	EMNIST dataset which consists of English alphabets and numbers are made use of to train the neural network.	Normalizing the pixel values.	The work is extended by adding some more datasets to the EMNIST dataset of characters from the Tamil language and training The model.
[6]HANDWRITTEN TEXT RECOGNITION: with Deep Learning and Android	The EMNIST dataset is a collection of handwritten alphanumeric derived from the NIST Special Database 19.	Using modern day techniques like neural networks to implement deep learning	The aim of our project is to make an application for mobile devices that can recognize handwriting using concepts of deep learning.
[7]Handwritten character recognition using convolutional neural network	NIST	In this research, a deep learning technique CNN is implemented for handwritten character recognition.	It was found that the accuracy obtained from 200 training images as 65.32% is improved gradually with increasing training images
[8]Handwritten Character Recognition Using Deep-Learning	NIST	OpenCV for performing Image processing and have used Tensorflow for training a the neural Network.	Designed a image segmentation based Handwritten character recognition system. An Android application was developed using which the user can click a photo of handwritten text using their camera.
[9]Online handwriting recognition system using UNIPEN-online handwritten training set and MNIST training set.	MNIST and UNIPEN.	Back Propagation, Segmentation, Multi Neural Networks, Elastic Distortion	The proposed model possessed the capability of creating efficient and flexible recognition system for large pattern set such as English Character set etc
[10]Accurate Acoustic based handwriting recognition system using deep learning	Acoustic signals generated by pen.	Segmentation, Classification, Word Suggestion.	THE WordRecorder extracts the valid temporal-frequency of the handwriting sound to recognize the user's input letter and uses word suggestion algorithm.

[11]Handwriting Recognition using Artificial Intelligence and Image Processing	The dataset consist of handwritten texts for evaluating machine learning models.	Image Acquisition and Digitization, Pre-processing, Segmentation	After training the machine learning model with the dataset, it was able to give an accuracy of 83.4%.
[12]RNN based Online Handwriting Recognition in Devanagari and Bengali Scripts using horizontal zoning.	In online handwriting recognition systems data is collected from graphic tablets	Horizontal Zoning (Zone Segmentation Approach), LSTM, BLSTM	Using the segmentation process both the scripts gave and accuracy of 99.40%(Devanagari) and 97.67% (Bengali)

CODE:

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.utils import to_categorical
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

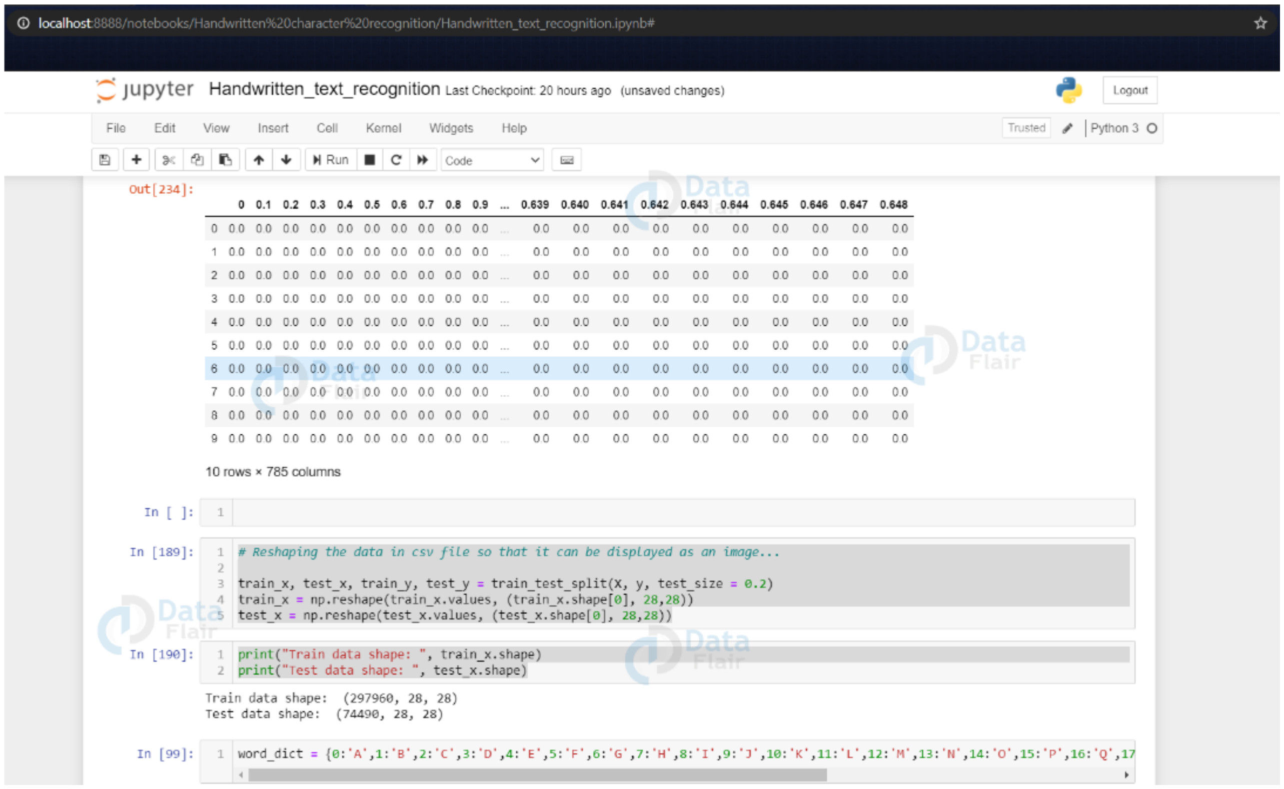
First of all, we do all the necessary imports as stated above. We will see the use of all the imports as we use them.

Read the data:

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

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)



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 [ ]:

In [189]:

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

train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
```

In [190]:

```
print("Train data shape: ", train_x.shape)
print("Test data shape: ", test_x.shape)
```

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

In [99]:

```
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:

```
X = data.drop('0',axis = 1)
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

```
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)

train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))

print("Train data shape: ", train_x.shape)
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.

```python
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
24:'Y',25:'Z'}
```

All the labels are present in the form of floating point values, that we convert to integer values, & so we create a dictionary word_dict to map the integer values with the characters.

Plotting the number of alphabets in the dataset

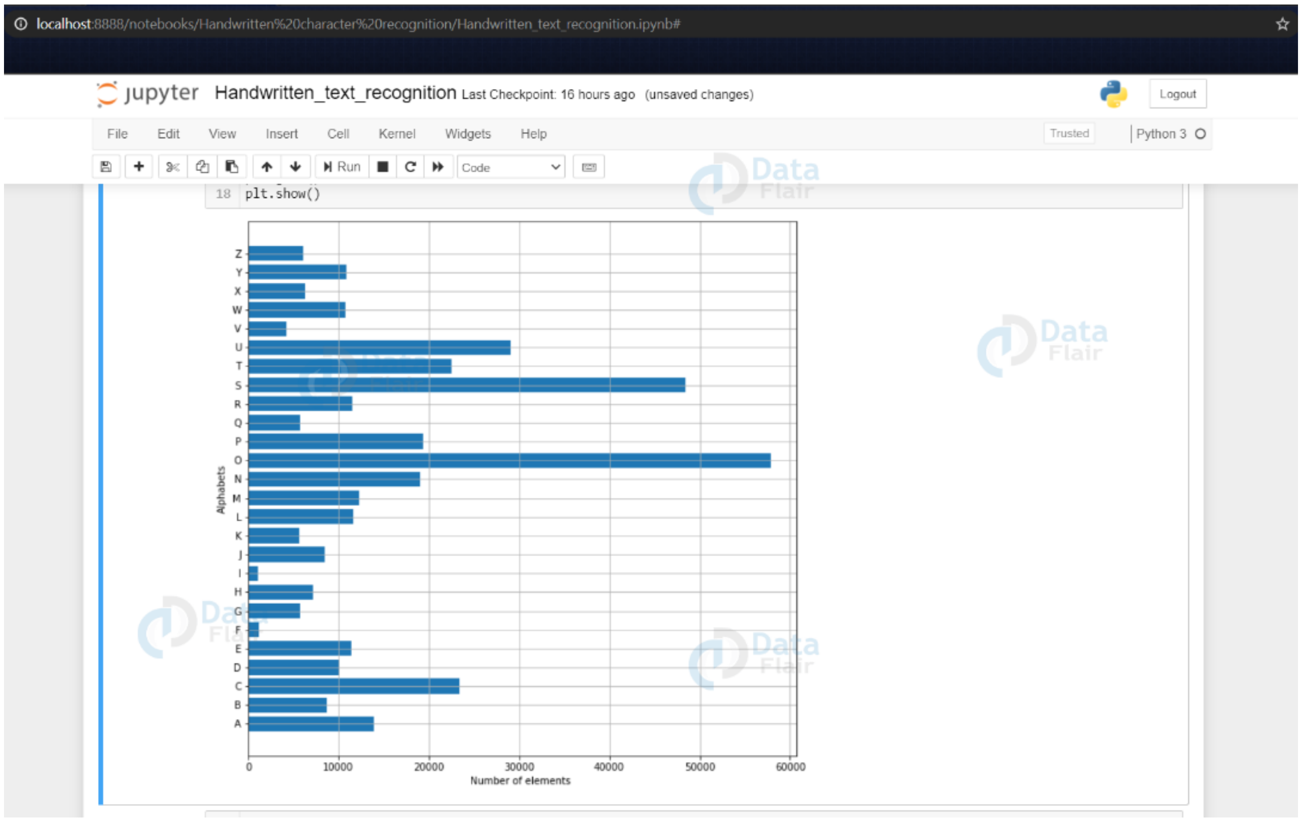
```python
y_int = np.int0(y)
count = np.zeros(26, dtype='int')
for i in y_int:
    count[i] +=1

alphabets = []
for i in word_dict.values():
    alphabets.append(i)

fig, ax = plt.subplots(1,1, figsize=(10,10))
ax.barh(alphabets, count)

plt.xlabel("Number of elements ")
plt.ylabel("Alphabets")
plt.grid()
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.



```
plt.show()
```

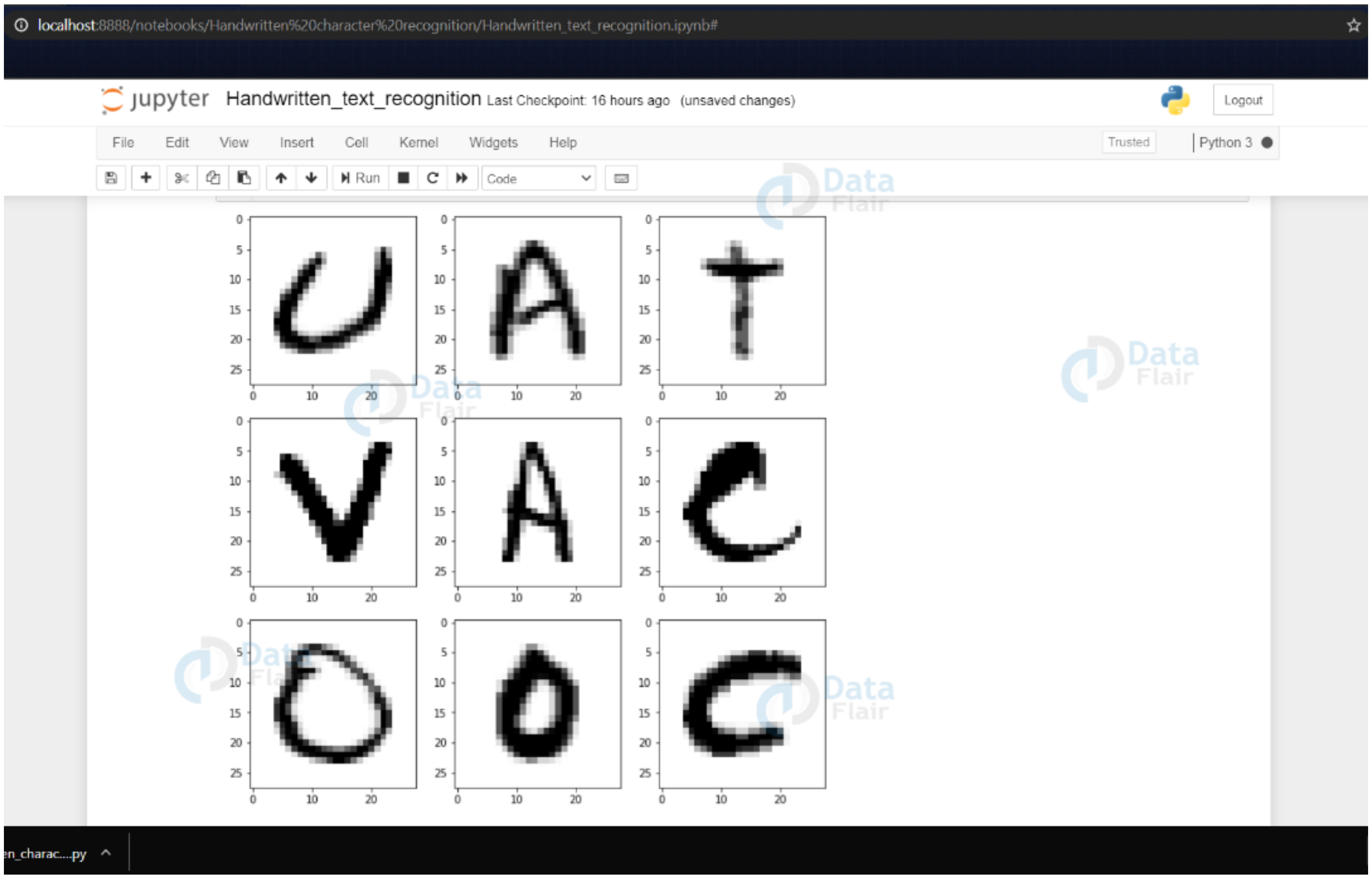
Shuffling the data

```
shuff = shuffle(train_x[:100])

fig, ax = plt.subplots(3,3, figsize = (10,10))
axes = ax.flatten()

for i in range(9):
    _, shu = cv2.threshold(shuff[i], 30, 200, cv2.THRESH_BINARY)
    axes[i].imshow(np.reshape(shuff[i], (28,28)), cmap="Greys")
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.


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
en_charac....py

(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

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

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


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)
```

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)

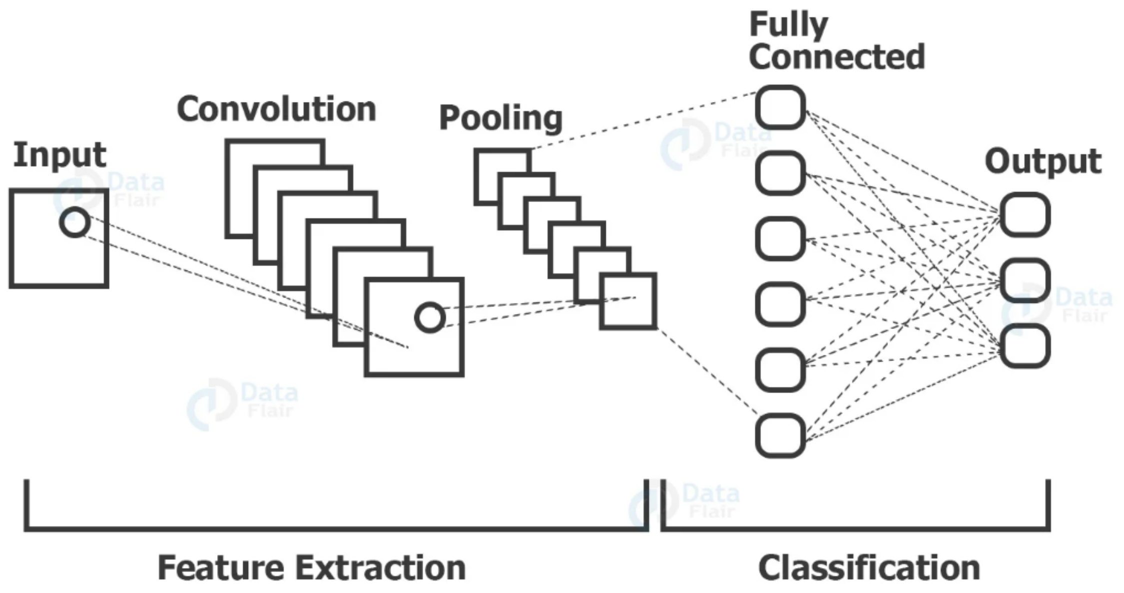
```
train_yOHE = to_categorical(train_y, num_classes = 26, dtype='int')
print("New shape of train labels: ", train_yOHE.shape)

test_yOHE = to_categorical(test_y, num_classes = 26, dtype='int')
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:

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))

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

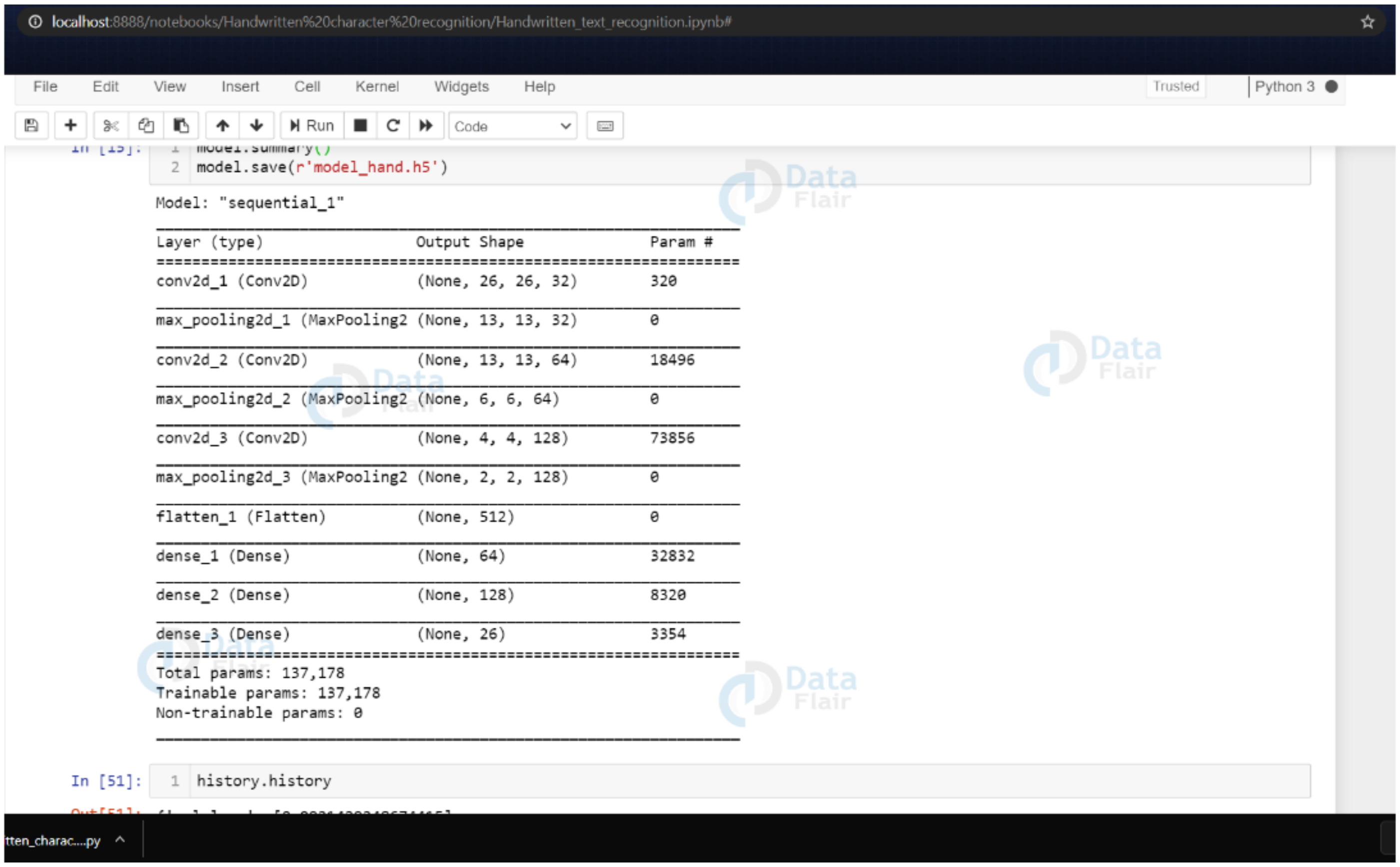
```
model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

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).

```
model.summary()
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.



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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_2 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_3 (Dense)              (None, 26)                3354
=================================================================
Total params: 137,178
Trainable params: 137,178
Non-trainable params: 0
_________________________________________________________________
```

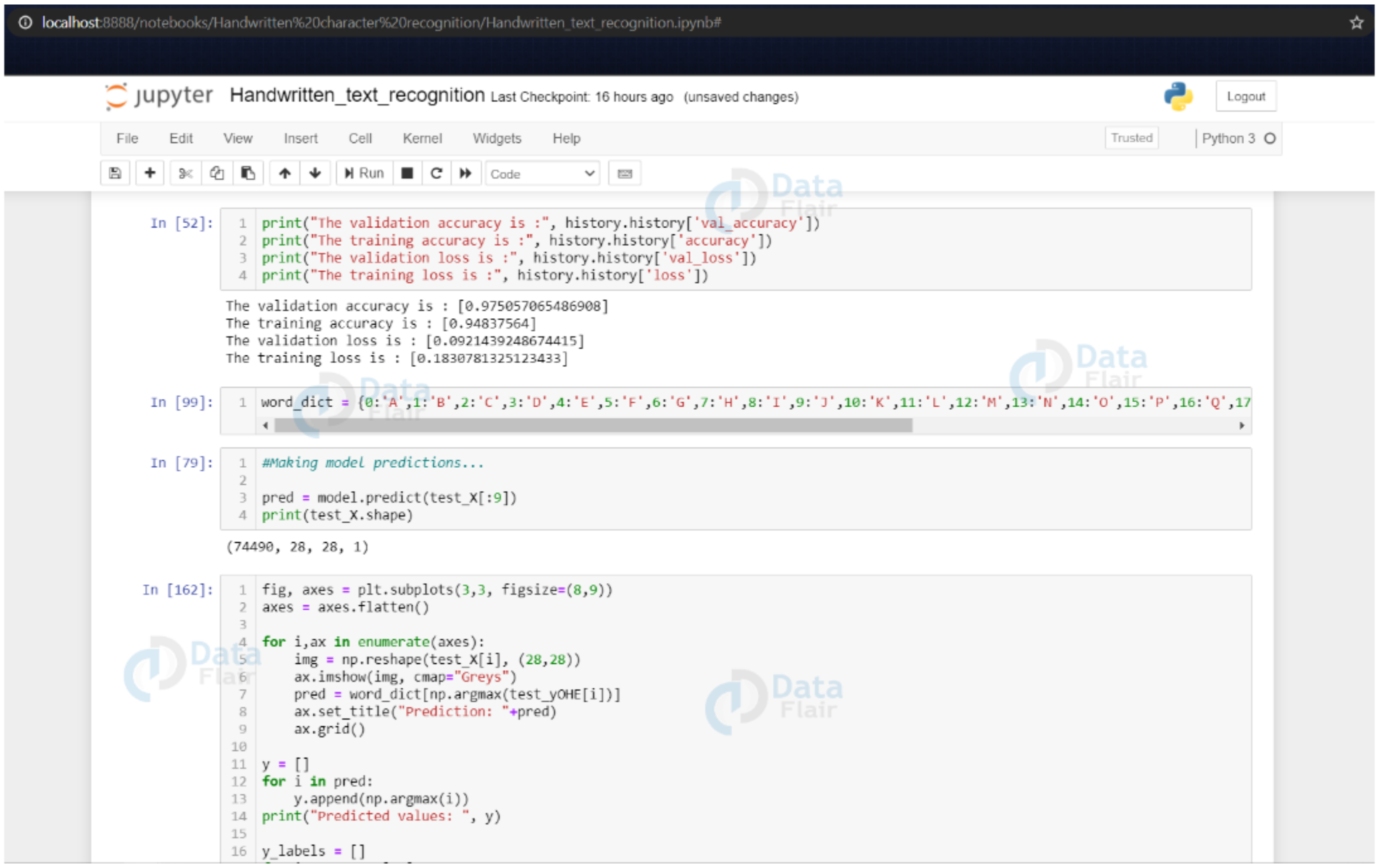
```
In [51]: history.history
```

(Summary of the defined model)

Getting the Train & Validation Accuracies & Losses

```
print("The validation accuracy is :", history.history['val_accuracy'])
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])
```

In the above code segment, we print out the training & validation accuracies along with the training & validation losses for character recognition.



```
In [52]: print("The validation accuracy is :", history.history['val_accuracy'])
         print("The training accuracy is :", history.history['accuracy'])
         print("The validation loss is :", history.history['val_loss'])
         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]: 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]: #Making model predictions...

         pred = model.predict(test_X[:9])
         print(test_X.shape)

(74490, 28, 28, 1)
```

```
In [162]: fig, axes = plt.subplots(3,3, figsize=(8,9))
          axes = axes.flatten()

          for i,ax in enumerate(axes):
              img = np.reshape(test_X[i], (28,28))
              ax.imshow(img, cmap="Greys")
              pred = word_dict[np.argmax(test_yOHE[i])]
              ax.set_title("Prediction: "+pred)
              ax.grid()

          y = []
          for i in pred:
              y.append(np.argmax(i))
          print("Predicted values: ", y)

          y_labels = []
```

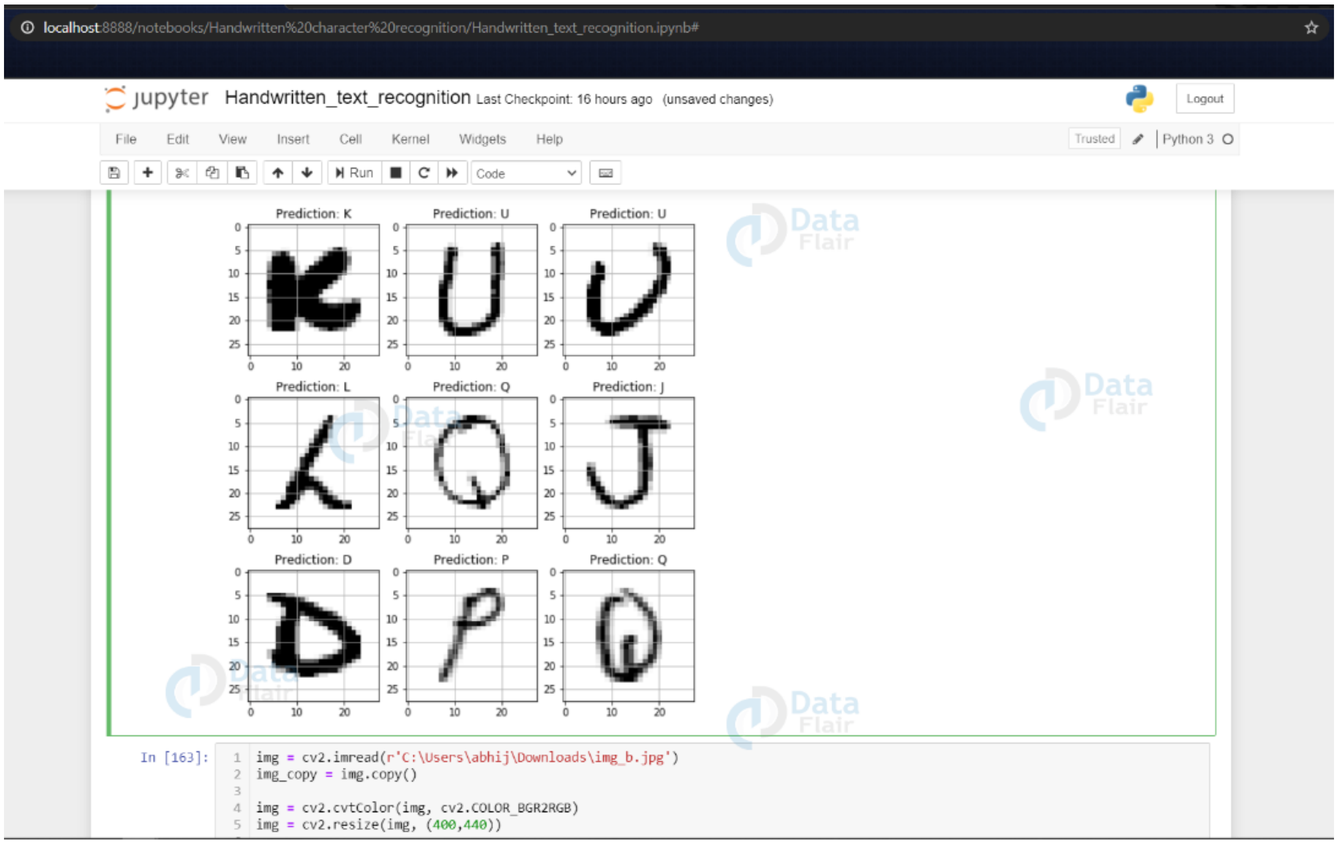
Doing Some Predictions on Test Data

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

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

    pred = word_dict[np.argmax(test_yOHE[i])]
    ax.set_title("Prediction: "+pred)
    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.


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Prediction: K
Prediction: U
Prediction: U
Prediction: L
Prediction: Q
Prediction: J
Prediction: D
Prediction: P
Prediction: Q
In [163]:

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

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))
```

Doing Prediction on External Image

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

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
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.

```
img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)

img_final = cv2.resize(img_thresh, (28,28))
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.

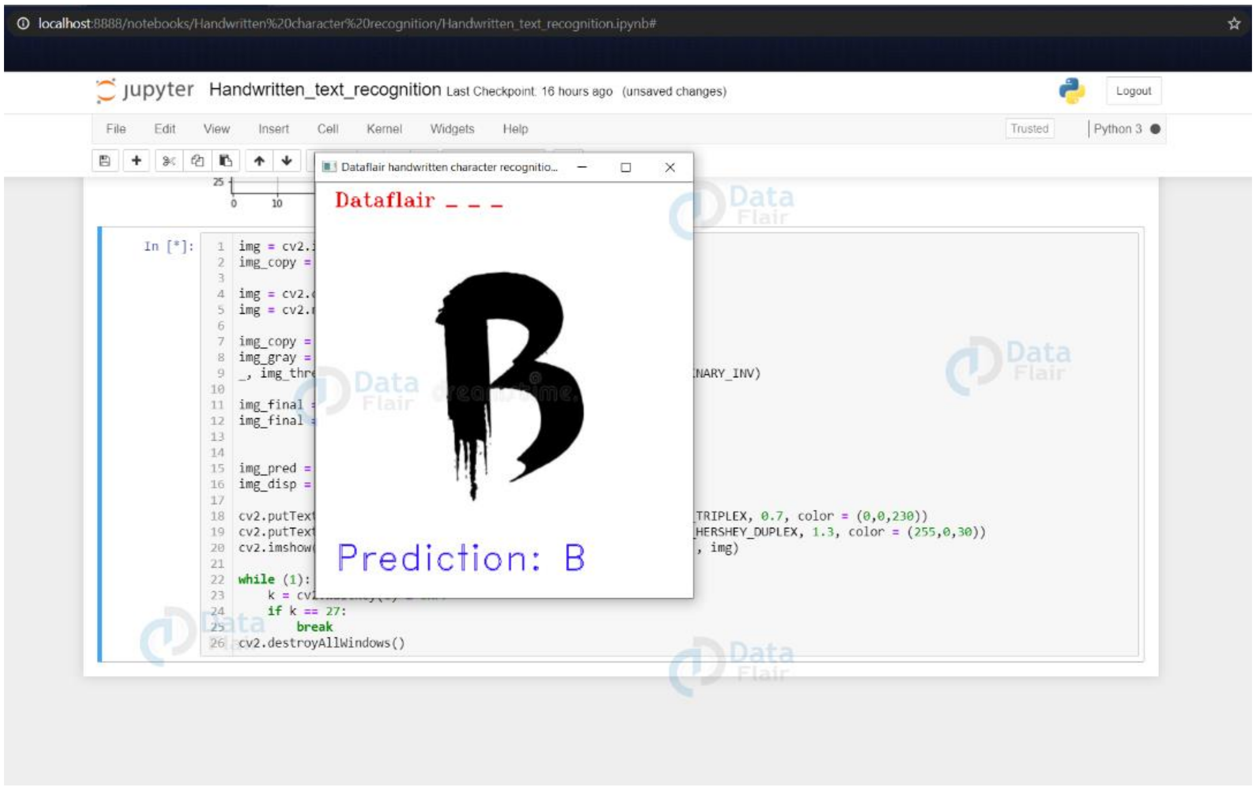
```
img_pred = word_dict[np.argmax(model.predict(img_final))]

cv2.putText(img, "Dataflair _ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow('Dataflair handwritten character 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 probability. Using this we get to know the exact character through the word_dict dictionary.
This predicted character is then displayed on the frame.

```
while (1):
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
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.


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Dataflair handwritten character recognitio...
Dataflair _ _ _
Prediction: B

```
In [*]:
img = cv2.i
img_copy =

img = cv2.
img = cv2.

img_copy =
img_gray =
_, img_thre                                                  NARY_INV)

img_final =
img_final =


img_pred =
img_disp =

cv2.putText                                                  TRIPLEX, 0.7, color = (0,0,230))
cv2.putText                                                  HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow(                                                  , img)

while (1):
    k = cv
    if k == 27:
        break
cv2.destroyAllWindows()
```

GAP ANALYSIS

After reading several Research Papers we have identified some shortcomings which can certainly be modified in the future.

[1] Some ROI are difficult to be recognized, it happened on ROI which contain the connected character. It is hard to do the segmentation with the CCL method because the CCL segmentation is based on the connectivity principle.

[4] Systems trained on IBM-UB-1 or IAM-OnDB alone perform significantly worse on our internal datasets, as our data distribution covers a wide range of use-cases not necessarily relevant to, or present, in the two academic datasets: sloppy handwriting, overlapping characters for handling writing on small input surfaces, non-uniform sampling rates, and partially rotated inputs. The network trained on the internal dataset performs well on all three datasets.

[5] Neural networks are quick to set up; however, they can be inaccurate if they learn properties that are not important in the target data.

[6] Without the use of EMNIST data set it would be practically impossible to achieve this accuracy .

[7] Exceeding the number of training images would lead to numerical errors and constraints on the CNN.

[8] Different people have different styles of writing. There are lot of characters like Capital letters , Small letters , Digits and Special symbols. Thus a large dataset is required.

[9] Use of spell checker and a voting module.

[10] This model only identifies upper case letters but not lower case letters.

[11] Use of this model can be extended to character recognition for other languages.

[12] Misclassification between two similar shaped characters.

CONCLUSION

1. Code:

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.

2. Reasearch:

Based on the 12 research papers we have read so far , we have observed that in order to tackle different challenges in handwriting recognition like variation in in style of writing, variation in shape and size of characters, accidental comas and dots, overwriting etc. different types of algorithms, architecture, methods have been used in those problems respectively to improve accuracy of recognition. there are variations too in accuracy of algorithms during training of models on different datasets and vice versa. Some of them are still not giving good accuracy but heavy research is still going on. Although Researchers have gained a significant progress in recognising handwritten text as compared to two decades back and is expected to improve in future. We will also try to implement some existing methods to compare accuracy on various datasets.

Acknowledgement

We take this opportunity to thank our teachers and friends who helped us throughout the project.

First and foremost, I would like to thank my guide for the project MR. RAHUL KUMAR SHARMA (Computer Science Department) for her/his valuable advice and time during development of project.

We would also like to thank Dr. Pankaj Kumar Sharma (HOD, Computer Science Department) for his constant support during the development of the project.

REFERENCES

[1] Darmatasia and Mohamad Ivan Fanany (2017). Handwriting Recognition on Form Document Using Convolutional Neural Network and Support Vector Machines (CNN-SVM).

[2] Daniel Keysers, Thomas Deselaers, Henry A. Rowley, Li-Lun Wang, and Victor Carbune(2016). Multi-Language Online Handwriting Recognition.

[3] Yaroslav Shkarupa, Roberts Mencis, Matthia Sabatelli (2016). Offline Handwriting Recognition Using LSTM Recurrent Neural Networks.

[4] Victor Carbune, Pedro Gonnet, ThomasDeselaers, Henry A, Rowley, Alexander Daryin, Marcos Calvo, Li-Lun Wang, Daniel Keysers, Sandro Feuz, Philippe Gervais (2020). Fast multi-language LSTM-based online handwriting recognition.

[5] Anandh Kishan, J. Clinton David (2018). Handwritten Character Recognition Using Cnn.

[6] Shubham Sanjay Mor, Shivam Solanki, Saransh Gupta, Sayam Dhingra, Monika Jain, Rahul Saxena (2019). HANDWRITTEN TEXT RECOGNITION: with Deep Learning and Android.

[7] Khandokar, Md M Hasan, F Ernawan, Md S Islam, M N Kabir (2020). Handwritten character recognition using convolutional neural network.

[8] Rohan Vaidya1 , Darshan Trivedi1 , Sagar Satra1, Prof. Mrunalini Pimpale2 (2018). Handwritten Character Recognition Using Deep-Learning

[9] Dũng Việt Phạm (2017). Online handwriting recognition system using UNIPEN-online handwritten training set and MNIST training set.

[10] Haishi Du, Ping Li , Hao Zhou , Wei Gong , Gan Luo , Panlong Yang(2018). Accurate Acoustic based handwriting recognition system using deep learning

[11] Sara Aqab, Muhammad Usman Tariq(2020). Handwriting Recognition using Artificial Intelligence and Image Processing

[12] Rajib Ghosh, Chirumavika Vamshi, Prabhat Kumar (2020). RNN based Online Handwriting Recognition in Devanagari and Bengali Scripts using horizontal zoning.