Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year: 2023), B.Sc. in CSE (Day)

Course Title: Machine Learning Lab

Course Code: CSE-412

Section: 201 D7

Lab Project Name: Handwritten Character Recognition with Neural Network.

Student Details

	Name	ID
1.	Estiak Hasan Emon	201002059
2.	Shamima Aktar	201002265
3.	Touhidul Islam	201002024

Submission Date : 25th June 2023
Course Teacher's Name : Afroza Akter

[For Teachers use only:Don't Write Anything inside this box]

Lab Project Status

Marks: …… Signature: .....

Comments: .....Date: .....

Contents

Chapter – 1

1. Introduction:

Handwritten character recognition, also known as optical character recognition (OCR), is the process of converting handwritten or printed text into digital form. It has numerous applications, such as digitizing historical documents, automated form processing, and even assisting visually impaired individuals in reading printed text.

OCR involves computer software designed to translate images of typewritten text into computerized format. It began as a field of researching artificial intelligence and machine vision.
Though academic research in the field continues the focus on OCR has shifted to implementation of proven techniques. Neural networks, particularly deep learning models, have revolutionized the field of character recognition, achieving impressive levels of accuracy and robustness.
These models are capable of automatically learning complex patterns and representations from large amounts of labeled data, making them well-suited for handwritten character recognition tasks.

2. Motivation behind this project:

Handwritten character recognition with neural networks provides efficient and accurate solutions for digitizing historical documents, automating form processing, enhancing accessibility for the visually impaired, enabling handwriting-based authentication, aiding handwriting analysis and forensics, supporting education and language learning, and automating data entry and information extraction tasks.

This technology offers tangible benefits such as improved productivity, enhanced security, increased accessibility, and streamlined data processing across various industries and applications. Neural Networks are recently being Used in various kind of pattern recognition. Handwritings of different person are different.

Handwritten Character recognition is an area of pattern recognition that has become the subject of research
High recognition accuracy and minimum training time of handwritten English characters using neural network is an open problem.

3. Objectives:

- To develop a system that can accurately recognize and classify handwritten characters.
- To handle variations in handwriting styles and variations in writing quality.
- To handle variations in handwriting styles, different writing utensils and variations in writing quality.

- To recognize and classify characters from various writing systems.

4. Features of proposed system:

- Detecting alphabets from dataset.
- Detecting alphabets from external image.

Chapter – 02

1. Requirements:

System Requirements:

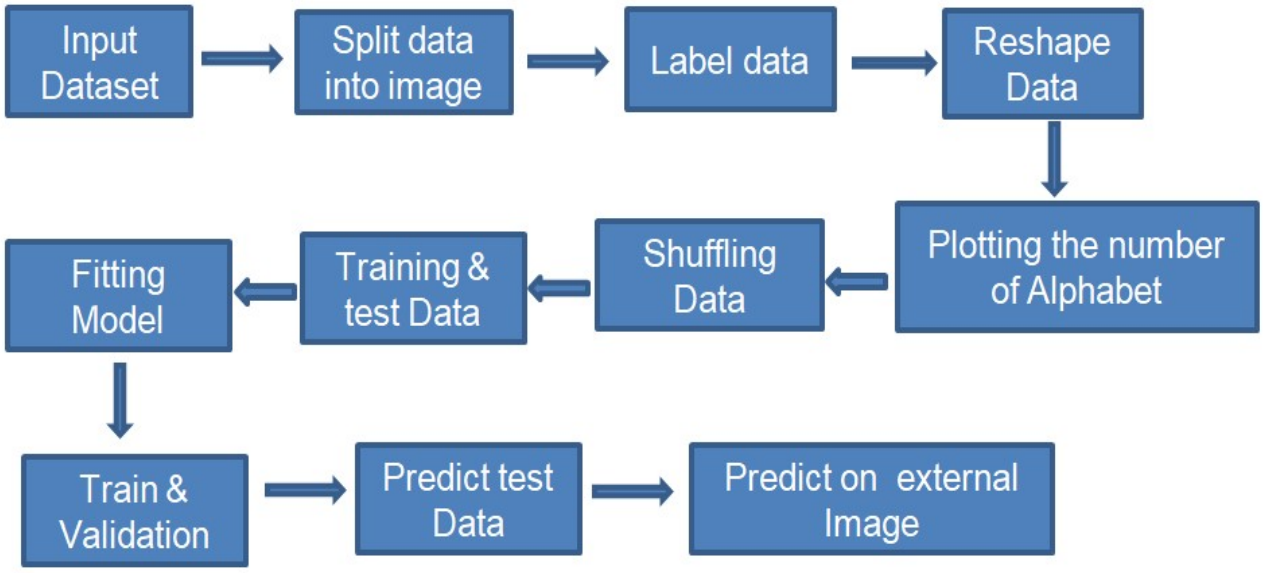
- Google Colab
- Using pandas
- Using numpy
- Some Library
- Dataset
- Tools: Lucid Chart for draw Methodology
- Code Editor: Google Colab
- Latest browser : Chrome, Firefox, etc
- Operating System : Any (Linux, Windows, Mac etc).

Hardware Requirements:

- Processor Pentium or higher version.
- Ram 1 GB or above.
- Hard Disk 150 MB or above.

Chapter -03

1. Methodology:



Chapter-04

1.Implementation:

1. Import the method:

```
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.

2. 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)

3. 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 frame read & use it in the y to form the labels.

4. 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.

```
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:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X',
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.

5. Plotting the number of alphabets in the dataset:

```
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.**

6. Shuffling the data:

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

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

for i inrange(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 threshold images of 9 alphabets.**

7. Data Reshaping:

```
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.

CNN stands for Convolution Neural Networks that are used to extract the features of the images using several layers of filters.

The convolution layers are generally followed by maypole layers that are used to reduce the number of features extracted and ultimately the output of the maypole 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.

8. Compiling and 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')
```

Using this we get the model summary that tells us what were the different layers defined in the model & also we save the model using model.save() function.

9. 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.

10. Doing Some Predictions on Test Data:

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

for i,ax inenumerate(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.

10. Doing Some Predictions on External Image:

```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/drive/MyDrive/ML DATASET/images.jpg')
img_copy = img.copy()

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

img = cv2.resize(img, (400, 440))

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))

img_pred = word_dict[np.argmax(model.predict(img_final))]
cv2.putText(img, "Probably _ _ _ ", (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(img)
```

The code performs handwritten character recognition on an image. It starts by loading an image and creating a copy of it.

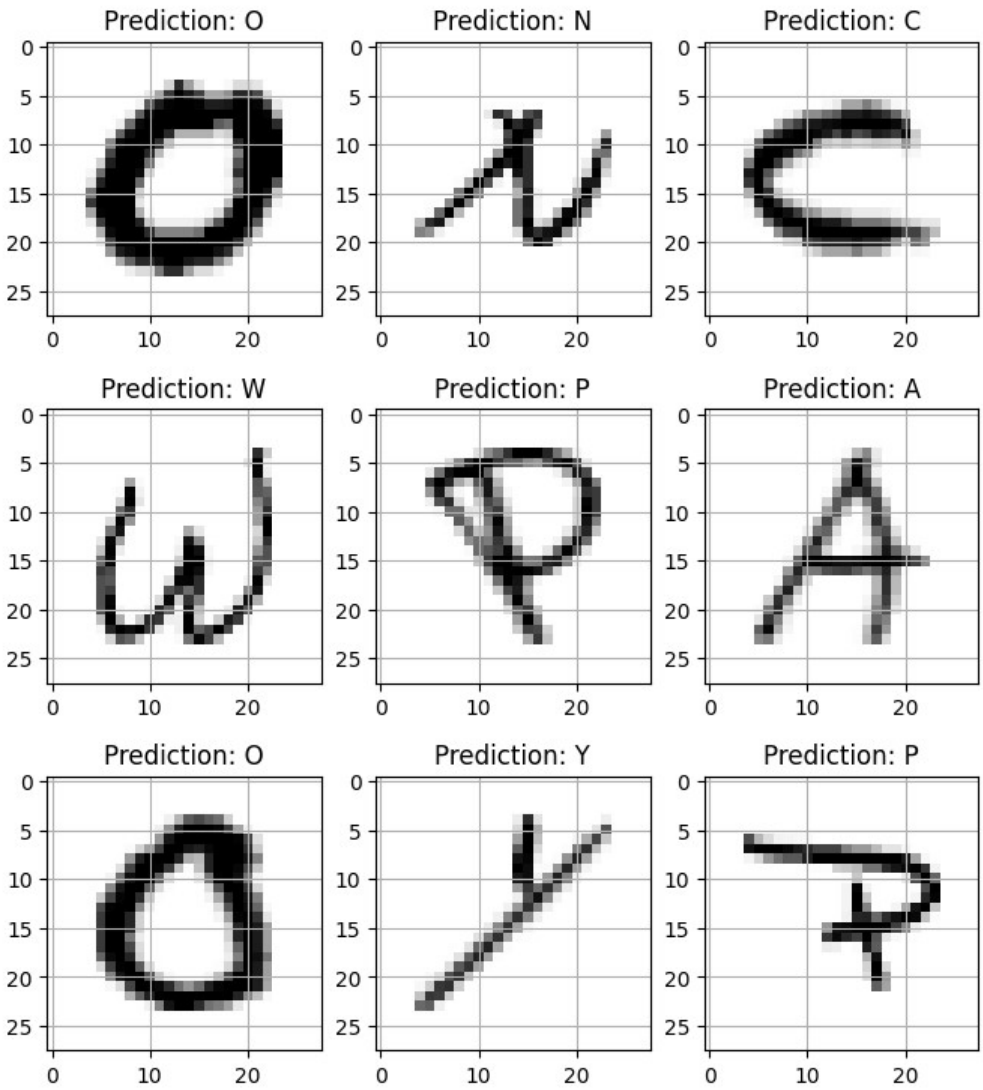
The image is then converted from BGR to RGB format and resized to a specific dimension.
The copied image is blurred using a Gaussian blur filter. The blurred image is then converted to grayscale.

A thresholding operation is applied to the grayscale image to create a binary image with inverted colors. The binary image is resized and reshaped to match the input dimensions expected by the model for prediction. The model predicts the handwritten character in the image using the pre-trained model and a word dictionary. The predicted

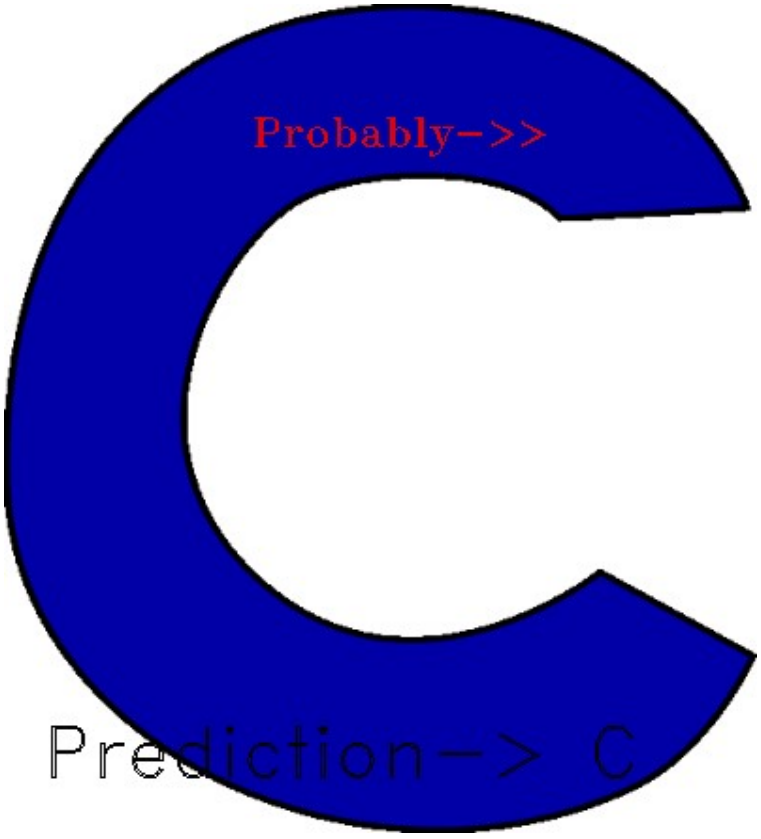
character is added as text on the original image. Finally, the modified image is displayed using the `cv2_imshow()` function in Google Colab.

2. Result:

Predictions on Test Data:



Predictions on External Data:



Chapter-05

1. Scope of future work:

The future scope of handwritten character recognition with neural networks involves improving accuracy, handling complex languages and scripts, advancing online recognition capabilities, incorporating contextual understanding, exploring transfer learning and few-shot learning techniques, integrating with emerging technologies such as AR and VR, and optimizing for mobile and edge computing applications.

- These advancements will enhance the reliability, versatility, and efficiency of handwritten character recognition, expanding its applications and making it more accessible to a wide range of industries and users.

- Using different types of language.

- Prediction multiple letters in one time.

2. Conclusion:

In conclusion, handwritten character recognition with neural networks is a powerful and versatile technology that enables the accurate conversion of handwritten text into digital form. It has numerous applications across diverse fields, including historical document digitization, form processing automation, accessibility for the visually impaired, handwriting-based authentication, handwriting analysis and forensics, education and language learning, and data entry and information extraction. With its ability to efficiently process large volumes of data and provide accurate results, neural network-based handwritten character recognition is revolutionizing industries and improving efficiency, accessibility, and security. A system for recognizing handwritten English characters will be developed. By achieving these objectives, a handwritten character recognition system can find applications in various domains, including document digitization, form processing, OCR, and NLP tasks involving handwritten text.

3. References:

https://en.wikipedia.org/wiki/Handwriting_recognition

https://www.researchgate.net/publication/259486519_Handwritten_Character_Recognition_using_Neural_Network

http://csjournals.com/IJCSC/PDF1-2/30..pdf?fbclid=IwAR3FjW2bsNG3snuhYtQz_CQahFLk7hiYFvun9PtGuiMUt8ZTOb5WOurYuzY