VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SOFT COMPUTING – SWE1011

PROJECT COMPONENT – REVIEW 3

TITLE:

COMPARISON STUDY ON CONVOLUTION NEURAL NETWORK (CNN) AND FUZZY LOGIC FOR CHARACTER RECOGNITION

TEAM MEMBERS:

SRIJAN KUMAR AIRON | 19MIS1119

NIRANJANA O | 19MIS1156

INTRODUCTION

When you have to deal with so many hardcopies, textual images, and PDFs on your job or in your business, you need optical character recognition (OCR) technology.

OCR is used to convert the non-editable soft copies into the editable text documents. Therefore, you don't need to retype the text in the images, magazines, or PDFs manually to make the required changes to them. OCR is a tool that quickly converts such files into editable text versions.

For instance, when you receive an electronic image of a brochure, use Optical Character Recognition (OCR) to convert it into a fully editable file so you can easily update its text.

The important reasons why you should use OCR technology are:

- Make Your Immutable Files Searchable
- Prevent Human Errors
- Save time and Money
- Save Space

The recognition of optical characters is known to be one of the earliest applications of Convolution Neural Networks and fuzzy logic, which partially emulate human thinking in the domain of artificial intelligence. The recognition of characters from scanned images of documents has been a problem that has received much attention in the fields of image processing, pattern recognition and artificial intelligence. In this paper, a simplified neural approach to recognition of optical or visual characters will be portrayed and discussed and we present a solution for the semi-automatic design of a Fuzzy classifier for letters and digits to be applied on the automatic recognition of cars license plates on unstructured conditions.

Handwriting recognition is the ability of a machine to receive and interpret handwritten input from multiple sources like paper documents, photographs, touch screen devices etc. Recognition of handwritten and machine characters is an emerging area of research and finds extensive applications in banks, offices and industries. Once input image of character is given to proposed system, and then it will recognize input character which is given in image. Recognition and classification of characters are done by Neural Network. The main aim of this project is to effectively recognize a particular character of type format using the Convolution Neural Network approach.

Characters are then identified using one of two algorithms:

1. Pattern recognition- OCR programs are fed examples of text in various fonts and formats which are then used to compare, and recognize, characters in the scanned document.
2. Feature detection- OCR programs apply rules regarding the features of a specific letter or number to recognize characters in the scanned document. Features could include the number of angled lines, crossed lines or curves in a character for comparison. For example, the capital letter “A” may be stored as two diagonal lines that meet with a horizontal line across the middle.

When a document is put to visual recognition, it is expected to be consisting of printed (or handwritten) characters pertaining to one or more scripts or fonts. This document however, may contain information besides optical characters alone. For example, it may contain pictures and colors that do not provide any useful information in the instant sense of character recognition. In addition, characters which need to be singly analyzed may exist as word clusters or may be located at various points in the document. Such an image is usually processed for noise-reduction and separation of individual characters from the document. It is convenient for comprehension to assume that the submitted image is freed from noise and that individual characters have already been located (using for example, a suitable clustering algorithm). This situation is synonymous to the one in which a single noise-free character has been submitted to the system for recognition.

MOTIVATION

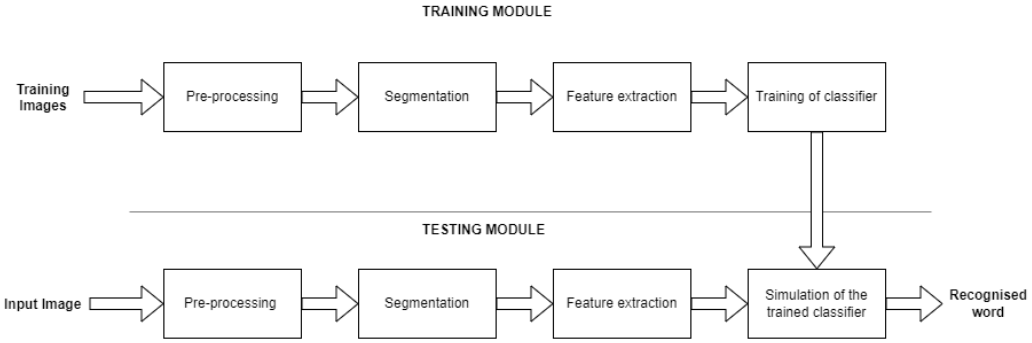
Optical character recognition (OCR) plays an important role in transforming printed materials into digital text files. These digital files can be very helpful to kids and adults who have trouble reading. That's because digital text can be used with software programs that support reading in a variety of ways.

The recognition of optical characters is known to be one of the earliest applications of Convolution Neural Networks and fuzzy logic, which partially emulate human thinking in the domain of artificial intelligence. The recognition of characters from scanned images of documents has been a problem that has received much attention in the fields of image processing, pattern recognition and artificial intelligence. In this paper, a simplified neural approach to recognition of optical or visual characters will be portrayed and discussed and we present a solution for the semi-automatic design of a Fuzzy classifier for letters and digits to be applied on the automatic recognition of cars license plates on unstructured conditions.

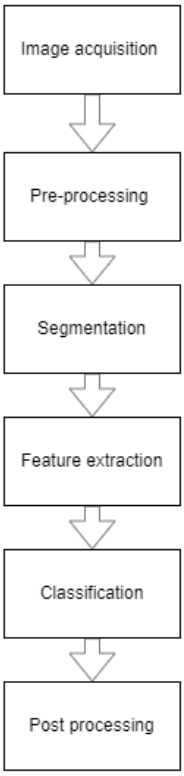
Neural network with their remarkable ability to derive meaning from complicated or imprecise data can be used to extract pattern and detect trend that are too complex to be noticed by either human or other computer techniques. A trained neural network can be thought of as an “expert” in the category of information it has been given to analyze. CNN is implemented to recognize the characters from a test dataset. The main focus of this work is to investigate CNN capability to recognize the characters from the image dataset and the accuracy of recognition with training and testing. CNN recognizes the characters by considering the forms and contrasting the features that differentiate among characters.

BLOCK DIAGRAMS FOR ALL SUBSYSTEMS

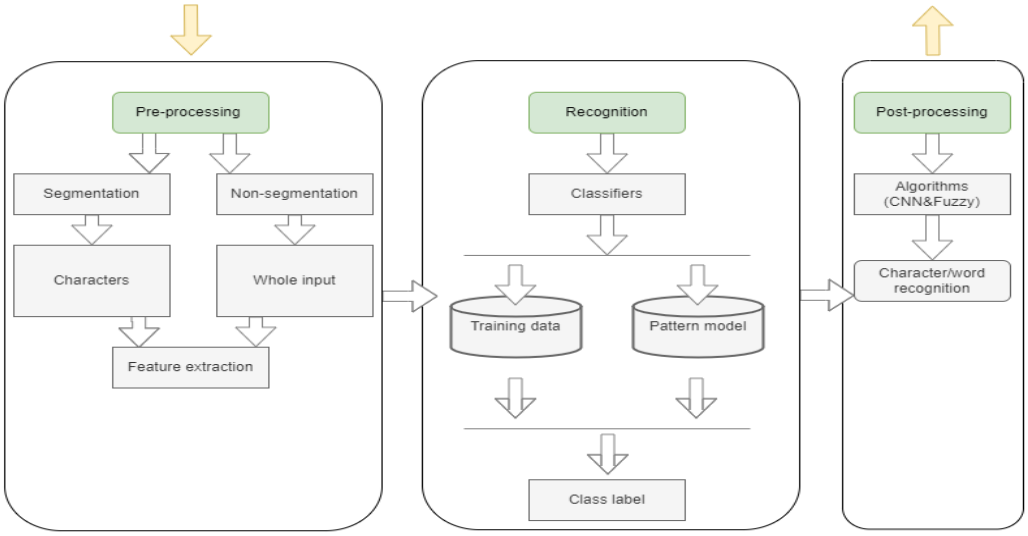
(i) Training and testing:



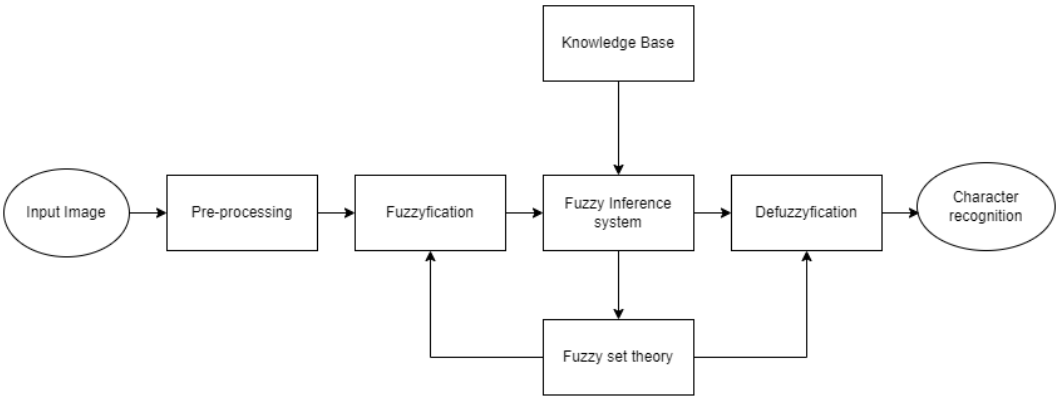
(ii) Image processing:



(iii) Pre-processing, Recognition, Post-processing



(iv) Fuzzy logic



(v) CNN logic

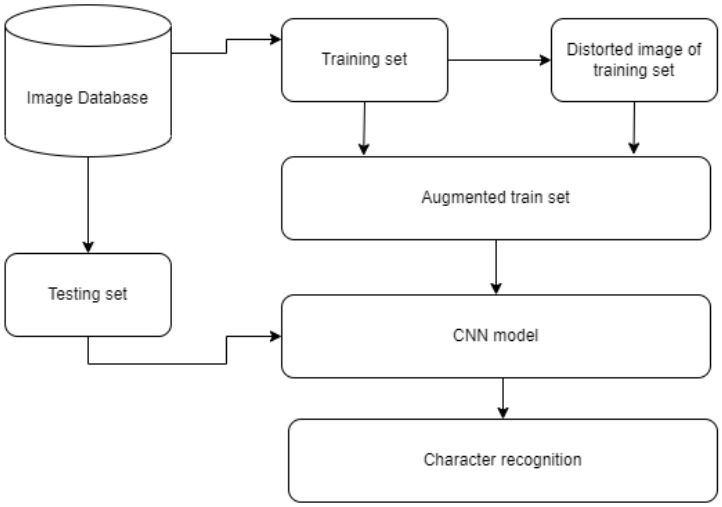
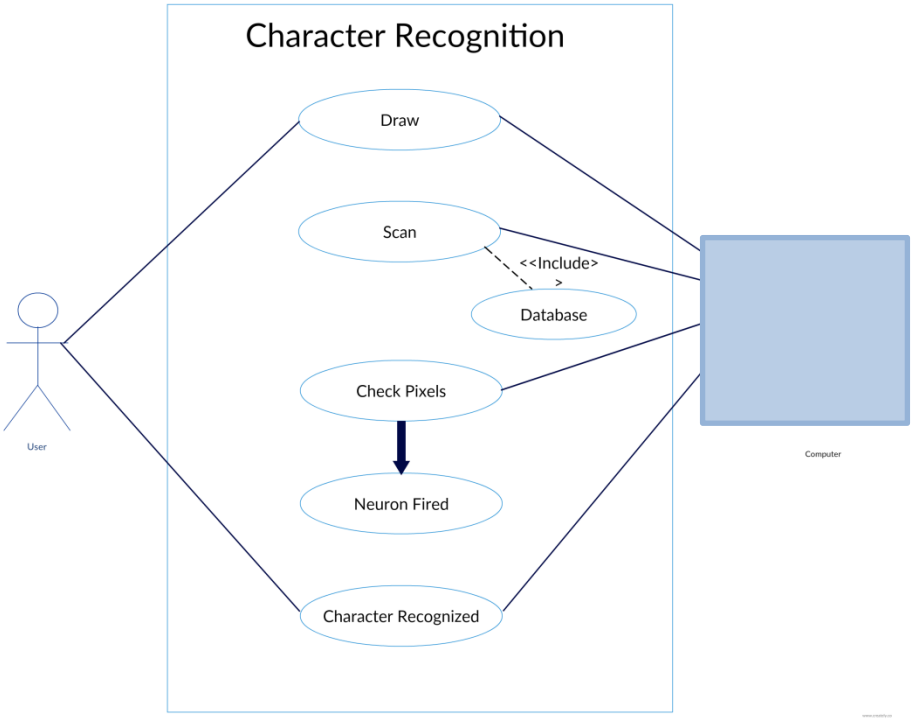


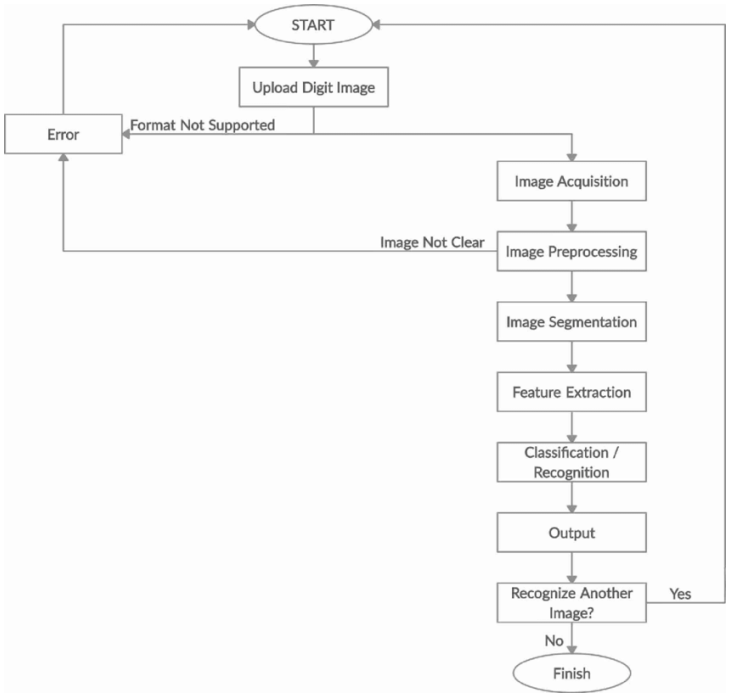
Image Database
Training set
Distorted image of training set
Augmented train set
Testing set
CNN model
Character recognition

UML DIAGRAMS

1. USE CASE DIAGRAM



2. ACTIVITY DIAGRAM



EXPLANATION OF THE BLOCK DIAGRAMS AND UML DIAGRAMS

(i) TRAINING AND TESTING MODULE

The training and testing module/subsystem's block diagram defines the working process taking place in the model. In the training process, images are trained according to the features and dimensions. In the pre-processing stage, the images included in the dataset are analyzed and is characterized accordingly. It is then divided into segments according to the characters defined and analyzed. The feature is extracted then and the algorithm is trained. In the testing process, images are inputted and the character is analyzed and the dimensions are analyzed. It is then divided according to the alphabet observed and compared. The algorithm is simulated to make the prediction and to define the accuracy and the alphabet is predicted.

(ii) IMAGE PROCESSING

In the image processing stage, the image is inputted by the user and the images are trained according to the features and dimensions. In the pre-processing stage, the images included in the dataset are analyzed and is characterized accordingly. It is then divided into segments according to the character's defined and analyzed. The feature is extracted then and the algorithm is trained. It is then categorized into the alphabets observed and analyzed and then the post-processing analysis is done where the algorithm is used to produce the accuracy rate of the alphabet predicted.

(iii) PRE-PROCESSING, RECOGNITION, POST- PROCESSING

In the pre-processing, recognition and post-processing module's block diagram, a detailed illustration of the processes are given. In the pre-processing phase, two types of processing are done which is segmentation and non-segmentation that doesn't involve categorization. In the segmentation phase, it is categorized into characters observed and hence feature is extracted. And in the non-segmentation phase, the character is observed and but not categorized. From the observation is done, the feature is extracted. Then the observations are compared from both the processes.

In the recognition phase, the classifiers are introduced and then the dataset in the training process is analyzed and the pattern model is observed from the inputted image by the user. And then it is labeled according to the observation.

In the post-processing phase, the algorithms are introduced which are fuzzy logic and CNN used for our model. And then the accuracy of the predicted output is analyzed. And hence, the result is given to the user.

(iv) FUZZY LOGIC

In the fuzzy logic process, the image is inputted from the user and it's then introduced to the pre-processing stage. Using fuzzy set theory, fuzzyfication and Defuzzyfication is observed. The algorithm is introduced and fuzzyfication is done. Fuzzyfication is the process of converting a crisp input value to a fuzzy value that is performed by the use of the information in the knowledge base. Defuzzyfication is the process of obtaining a single number from the output of the aggregated fuzzy set. It is used to transfer fuzzy inference results into a crisp output. In other words, defuzzyfication is realized by a decision-making algorithm that selects the best crisp value based on a fuzzy set. And hence, the character recognition is done with the accuracy rate predicted.

(v) CNN PROCESS

In the CNN process, the image is analyzed and observed from the training dataset. The distorted image set is then augmented and categorized accordingly. And then, via this the CNN model is trained according to the dataset trained. In the testing phase of CNN, the input image is accessed and it's then the CNN model is used to predict the alphabet identified. And hence, the result is obtained.

(vi) USE-CASE DIAGRAM

In the use case diagram, the user inputs the image. And then use cases are defined in the use case subject. It is then scanned and is categorized and analyzed from the database that has been trained earlier. And then the system defines the pixels and the algorithm is introduced to find the accuracy of the prediction of the character recognized. And hence, the output is released and then user is made to see it.

(vii) ACTIVITY DIAGRAM

In the activity diagram, the image is inputted by the user and image processing takes place where the image is acquainted and the features are defined according to the dimensions. The input image is gone through image pre-processing phase where segmentation and non-segmentation takes place in-order to observe the feature and extract the feature categorically. And then the algorithm i.e. CNN/fuzzy logic is introduced for defining the accuracy of the predicted character. And the output is made visible to the user.

SYSTEM WORKING AND OUTPUT

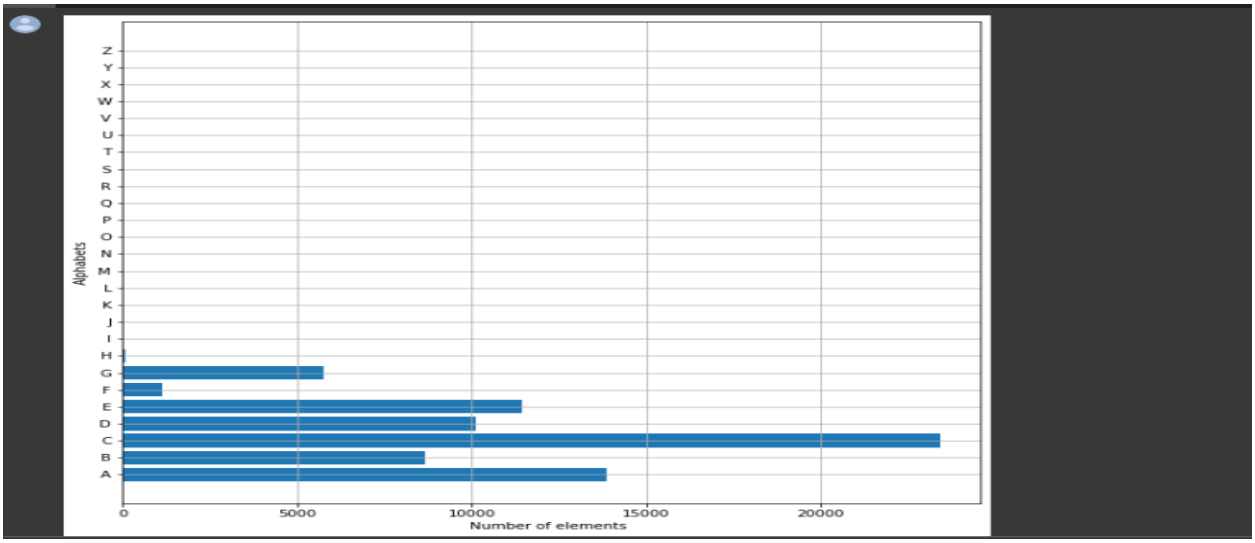
(i) CNN

```
[45] data = pd.read_csv(r"A_Z Handwritten Data.csv").astype('float32')
     print(data.head(10))
```

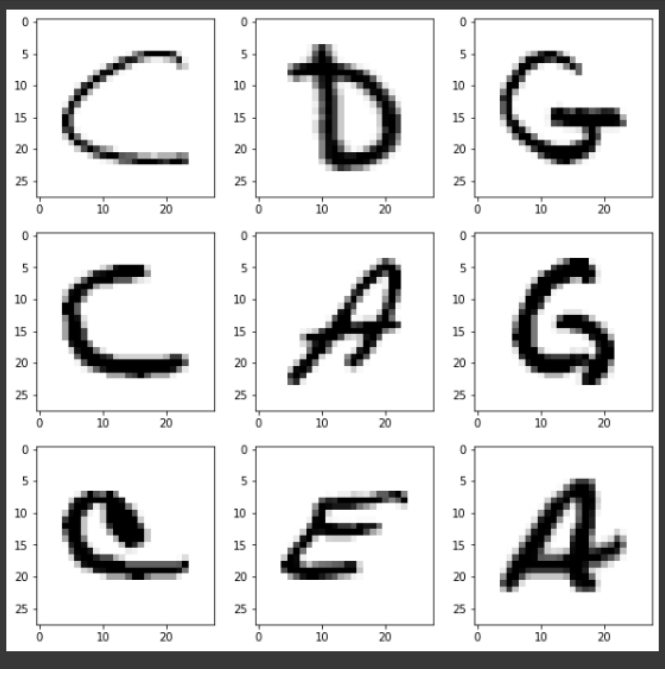
```
        0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  ...  0.639  0.640  0.641  \
0   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
1   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
2   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
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
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
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
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
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
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
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

     0.642  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
1      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
3      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
5      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
7      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
9      0.0    0.0    0.0    0.0    0.0    0.0    0.0

[10 rows x 785 columns]
```

```
[47] 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)
```

```
Train data shape:  (59617, 28, 28)
Test data shape:  (14905, 28, 28)
```




```
[50] 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()
```



```
[51] 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)
```

```
New shape of train data:  (59617, 28, 28, 1)
New shape of train data:  (14905, 28, 28, 1)
```

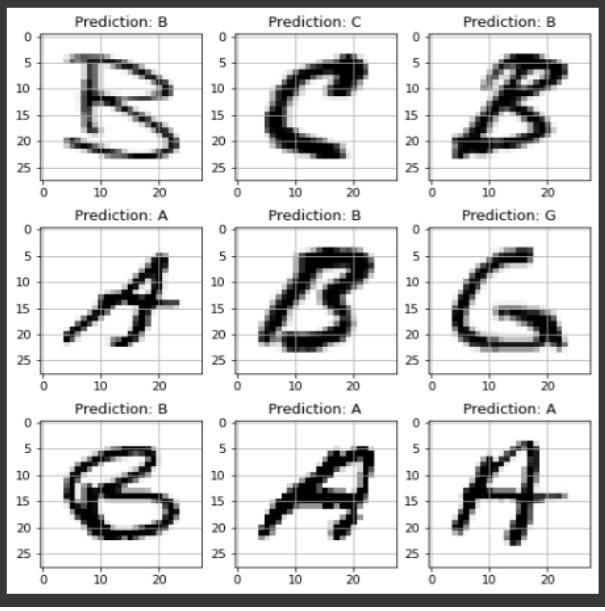
```
[52] train_yOHE = to_categorical(train_y, num_classes = 26, dtype='int')
     print("New shape of train labels: ", train_yOHE.shape)
     test_yOHE = to_categorical(test_y, num_classes = 26, dtype='int')
     print("New shape of test labels: ", test_yOHE.shape)
```

```
New shape of train labels:  (59617, 26)
New shape of test labels:  (14905, 26)
```

```
[57] 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'])
```

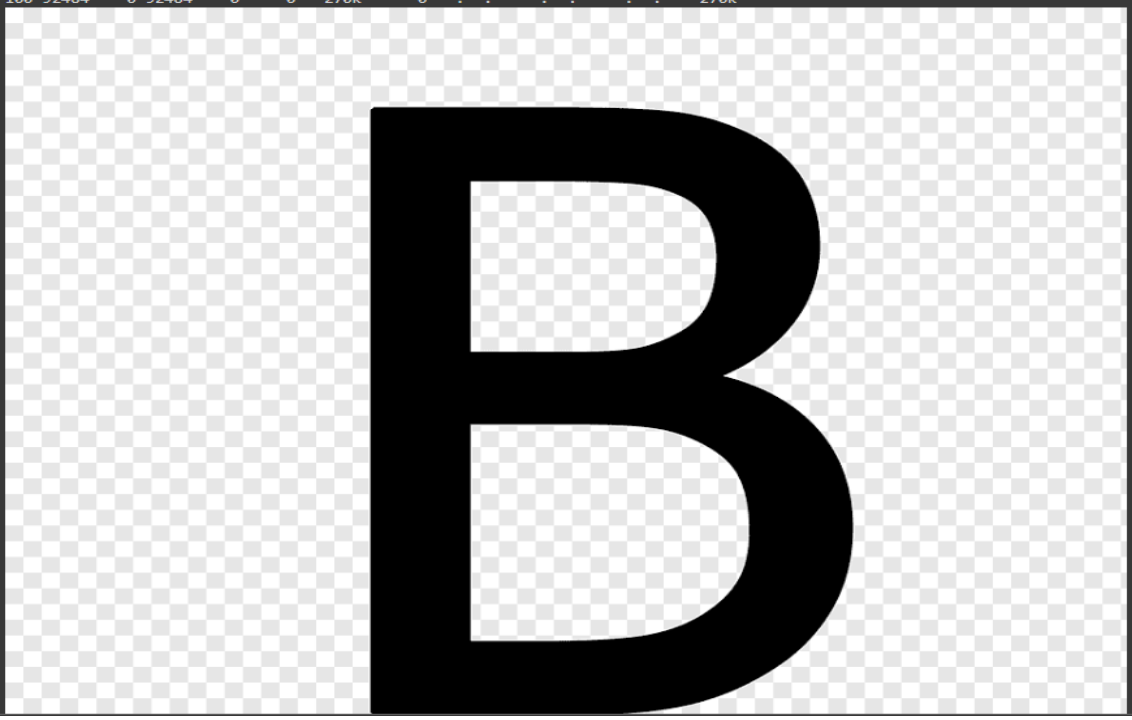
```
The validation accuracy is : [0.9869842529296875]
The training accuracy is : [0.9854739308357239]
The validation loss is : [0.04854567348957062]
The training loss is : [0.05045042186975479]
```

```
[58] 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()
```



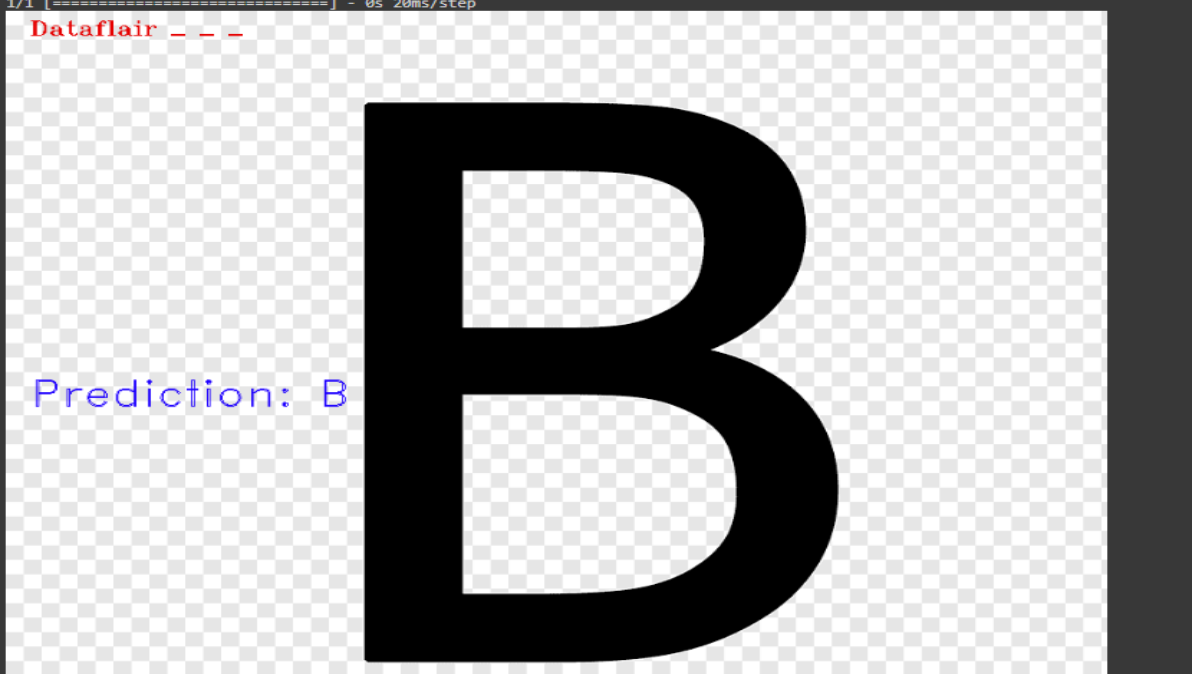
```
[79] !curl -o logo.png https://www.pngwing.com/en/free-png-bywri
     import cv2
     img = cv2.imread('img_b.png', cv2.IMREAD_UNCHANGED)
     cv2_imshow(img)
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 92484    0 92484    0     0   270k      0 --:--:-- --:--:-- --:--:--  270k
```



```
[81] 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(img)
```

```
1/1 [==============================] - 0s 20ms/step
```



EXPLANATION OF SYSTEM WORKING

(i) CNN

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. Now we are reading the dataset using the pd.read_csv(). 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. 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. 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. 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. 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. Now we reshape the train & test image dataset so that they can be put in the model. And 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. We have the CNN model that we designed for training the model over the training dataset. 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. 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). We print out the training & validation accuracies along with the training & validation losses for character recognition. And 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.

(ii) FUZZY LOGIC

OCA is a complex fuzzy system with six input variables, one output variable, twenty-three input membership functions, fifteen output membership functions and fourteen fuzzy rules. This complexity makes it difficult to estimate the output for a given set of inputs intuitively. The most productive testing technique for fuzzy systems is empirical. To test OCA, a text file of test data was supplied to the OCA simulator and generated outputs were compared to the expected outputs. A group of fourteen data sets was created as a starting point. This group is shown in the listing below. The input data sets correspond, in order, to the fourteen fuzzy rules in the fuzzy inference unit listing. Therefore, proper selection of input values X1 through X4, SOP and TERM should generate the outputs 0 through 9, SS1, SS2, SS3, and SS4, in that order. The values shown are center points for the fuzzy set specified by a given rule, and thus represent a set of input data that OCA should find easy to classify. This group of fourteen data sets was then duplicated and modified to check for proper operation across the range of character data specified in Tables 3, 5, and 6. The listing shows only the initial fourteen data sets. In each group of data sets, one input variable (say, X1) was taken to the lower limit, with all others held at the upper limit. Over the entire suite of 112 worst-case simulated data sets, OCA generated two outputs that would have required a second read. These two cases generated outputs that were outside the imposed definition of expected output ±0.13. Importantly, no data sets were incorrectly labelled as being a different character. These worst-case data sets would probably never be encountered in the real world, since transitions, SOP, and TERM would all tend to deviate from mean values in the same direction. Nonetheless, further manipulation of membership functions and rule definitions might make OCA even more robust.

CODE/PSEUDOCODE

FUZZY LOGIC PSEUDOCODE

```
$Variable Definitions: X1, X2, X3, and X4 are defined as
$transitions; the magnitude of change between local minima and maxima.

invar X1 "delta pixels" :2 () 30 [
 Small (@2,1, @11,1, @13,0),
 Med (@11,0, @13,1, @17,1, @19,0),
 Large (@16,0, @18,1, @19,1, @21,0),
 Max (@18,0, @20,1, @30,1)];

invar X2 "delta pixels" :-30 () -2 [
 Large (@-30,1, @-22,1, @-18,0),
 Med (@-21,0, @-19,1, @-13,1, @-11,0),
 Small (@-13,0, @-11,1, @-2,1)];

invar X3 "delta pixels" :2 () 30 [
 Small (@2,1, @6,1, @8,0),
 Med (@6,0, @7,1, @11,1, @13,0),
 Large (@10,0, @11,1, @13,1, @15,0),
 Max (@13,0, @15,1, @30,1)];

invar X4 "delta pixels" :-30 () -2 [
 Large (@-30,1, @-20,1, @-17,0),
 Med (@-20,0, @-18,1, @-13,1, @-11,0),
 Small (@-13,0, @-10,1, @-2,1)];

invar SOP "total pixels" :60 () 220 [
 Small (@60,1, @88,1, @104,0),
 Sp (@90,0, @99,1, @113,1, @120,0),
```

```
 Med (@97,0, @109,1, @128,1, @134,0),
 Large (@126,0, @133,1, @160,1, @176,0),
 Max (@167,0, @178,1, @220,1)];

invar TERM "data slices" :8 () 20 [
 Small (@8,1, @11,1, @12.3,0),
 Med (@12.1,0, @13,1, @14,1, @15,0),
 Large (@14,0, @15,1, @16,1, @18,0),
 Max (@16,0, @17,1, @20,1)];

outvar Char "Character" :0 () 14 * (
 one =1,
 two =2,
 three =3,
 four =4,
 five =5,
 six =6,
 seven =7,
 eight =8,
 nine =9,
 zero =10,
 SS1 =11,
 SS2 =12,
 SS3 =13,
 SS4 =14);

$!!! Fuzzy Rules !!!
$Defines relationships between input variables and output variables.

if X1 is Max and X2 is Med and X3 is Max and X4 is Large
 and SOP is Large and TERM is Max
 then Char is zero;
```

```
if X1 is Max and X2 is Large
 and SOP is Med and TERM is Small
 then Char is one;

if X1 is Med and X2 is Small and X3 is Med and X4 is Med
 and SOP is Sp and TERM is Small
 then Char is two;

if X1 is Max and X2 is Large
 and SOP is Large and TERM is Med
 then Char is three;

if X1 is Large and X2 is Med and X3 is Med and X4 is Small
 and TERM is Large
 then Char is four;

if X1 is Med and X2 is Small and X3 is Med and X4 is Med
 and SOP is Med and TERM is Med
 then Char is five;

if X1 is Max and X2 is Med and X3 is Small and X4 is Small
 and TERM is Large
 then Char is six;

if X1 is Small and X2 is Small and X3 is Large and X4 is Small
 and SOP is Small
 then Char is seven;

if X1 is Max and X2 is Med and X3 is Max and X4 is Large
 and SOP is Max and TERM is Max
 then Char is eight;
```

```
if X1 is Med and X2 is Small and X3 is Max and X4 is Large
 and SOP is Large
 then Char is nine;

if X1 is Small and X2 is Small and X3 is Med and X4 is Small
 and SOP is Med and TERM is Max
 then Char is SS1;

if X1 is Med and X2 is Med and X3 is Max and X4 is Med
 and SOP is Large and TERM is Max
 then Char is SS2;

if X1 is Med and X2 is Med and X3 is Max and X4 is Med
 and SOP is Max and TERM is Max
 then Char is SS3;

if X1 is Small and X2 is Small and X3 is Med and X4 is Small
 and SOP is Small and TERM is Max
 then Char is SS4

end
```

CONVOLUTION NEURAL NETWORK CODE:

```
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
13. data = pd.read_csv(r"A_Z Handwritten Data.csv").astype('float32')
14. print(data.head(10))
15. X = data.drop('0',axis = 1)
16. y = data['0']
17. train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
18. train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
19. test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
20. print("Train data shape: ", train_x.shape)
21. print("Test data shape: ", test_x.shape)
22. 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',1
    4:'O',15:'P',16:'Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}
```

```
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()
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()
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)
```

```
47. train_yOHE = to_categorical(train_y, num_classes = 26, dtype='int')
48. print("New shape of train labels: ", train_yOHE.shape)
49. test_yOHE = to_categorical(test_y, num_classes = 26, dtype='int')
50. print("New shape of test labels: ", test_yOHE.shape)
```

51. CNN

```
52. model = Sequential()
53. model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,
    1)))
54. model.add(MaxPool2D(pool_size=(2, 2), strides=2))
55. model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
56. model.add(MaxPool2D(pool_size=(2, 2), strides=2))
57. model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
58. model.add(MaxPool2D(pool_size=(2, 2), strides=2))
59. model.add(Flatten())
60. model.add(Dense(64,activation ="relu"))
61. model.add(Dense(128,activation ="relu"))
62. model.add(Dense(26,activation ="softmax"))
63. model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy',
     metrics=['accuracy'])
64. history = model.fit(train_X, train_yOHE, epochs=1,  validation_data = (test_X,test_yOHE)
    )
65. print("The validation accuracy is :", history.history['val_accuracy'])
66. print("The training accuracy is :", history.history['accuracy'])
67. print("The validation loss is :", history.history['val_loss'])
68. print("The training loss is :", history.history['loss'])
```

```
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()
img = cv2.imread(r'img_b.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))
from google.colab.patches import cv2_imshow
!curl -o logo.png https://in.pinterest.com/pin/447474912975790887/
import cv2
img = cv2.imread('img_b.jpg', cv2.IMREAD_UNCHANGED)
cv2_imshow(img)
c = cv2.waitKey()
```

```
93. if c == 27:
94.   cv2.destroyAllWindows()
95. img_pred = word_dict[np.argmax(model.predict(img_final))]
96. cv2.putText(img, "Dataflair _ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,
    0,230))
97. cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3,
    color = (255,0,30))
98. cv2_imshow(img)
99. while (1):
100.            k = cv2.waitKey(1) & 0xFF
101.            if k == 27:
102.               break
103.         cv2.destroyAllWindows()
```

REFERENCES

[1] William A. Gowan “Optical Character Recognition Using Fuzzy Logic”Freescale semiconductor, Inc. \

[2] Michael Hadourin “A Neural Network Implementation of Optical Character Recognition” Technical Report Number CSSE10-05 COMP 6600 – Artificial Intelligence Spring 2009.

[3] Sibigtroth, James M., "Creating Fuzzy Micros," Embedded Systems Programming, December 1991

[4] Aditya Panchwagh et.al “OPTICAL CHARACTER AND DIGIT RECOGNITION SYSTEM” e-ISSN: 2395-0056 Volume: 07 Issue: 08 | Aug 2020