NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA

Project REPORT

on

"HANDWRITTEN CHARACTER RECOGNITION USING NEURAL NETWORK"

Submitted in partial fulfilment of the requirement for the award of Degree of

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by:

Karthik P S	1NT18CS068
Nikhil Rajkoti L	1NT18CS107
H Puneeth Shetty	1NT18CS049

Under the Guidance of
Dr. Vani V
Professor, Dept. of CS&E, NMIT

Department of Computer Science and Engineering

(Accredited by NBA Tier-1)

2021-22

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE UNDER VTU, BELGAUM)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project work entitled "**HANDWRITTEN CHARACTER RECOGNITION**" is successfully completed by **Karthik P S(1NT18CS068), Nikhil Rajkoti L(1NT18CS107), H Puneeth Shetty D(1NT18CS049)** during the period **2021-2022.** It is certified that all the corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The project has been approved as it satisfies the requirements prescribed for Bachelor of Engineering Degree.

Name and Signature of internal guide	***Name and Signature of the HOD***	***Name and Signature of the Principal***
[Dr. Vani V]	**[Dr. Sarojadevi H]**	**[Dr. H C Nagaraj]**

ACKNOWLEDGEMENT

A mini project is a golden opportunity for learning and self-development. We consider ourselves very lucky and honored to have a so many wonderful people lead us through in completion of this mini project.

We would like to thank **Dr. N.R. Shetty**, Director, NMIT, for providing such a wonderful environment to carry our Mini project.

We would like to express our gratitude to **Dr. H.C Nagaraj**, principal of NMIT, for his support in bringing this Mini project to completion. We wish to express our gratitude to the head of the department, **Dr. Sarojadevi H**, Dept. of CSE for providing a congenial working environment.

I wish to convey my deep sense of gratitude to **Dr. Vani V**, Assistant Professor, CSE Department, NMIT for her valuable guidance through the conduction of this mini project.

I take this opportunity to extend my sincere thanks to all remaining staff of our department for their necessary help and co-operation.

-

ABSTRACT

CNN (Convolutional neural network) is effectively implemented for character recognition and it is one of the best performing deep learning method. The purpose of this project is to recognize 26 English alphabet (A-Z). CNN can be used directly for recognition or also for extracting features in character recognition is discussed in this project. The model is trained over a dataset containing images of alphabet of size 28pixels. Data sets are used to train the neural network. The trained network is used for classification and recognition.

Handwritten character detection is a technique or ability of a computer to receive and interpret intelligible handwritten output. The algorithm used in character recognition is mainly divided into image pre-processing, feature extraction and classification. CNN is found efficient for handwritten character recognition. In this algorithm CNN employs for the classification of individual characters.

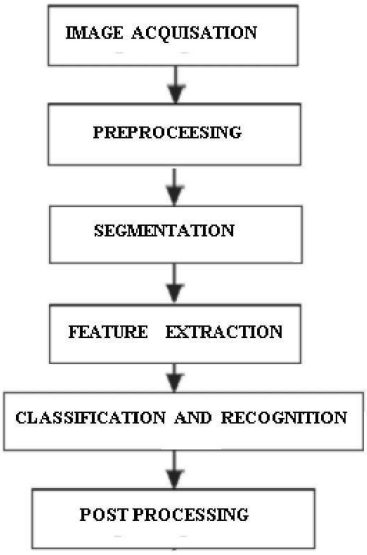
TABLE OF CONTENTS

INTRODUCTION

The main aim of this project is to design handwritten character recognition using neural networks. Handwriting character recognition is the ability of a machine to receive the handwritten characters as input and predict it. Handwritten character recognition works in stages of preprocessing, segmentation, feature extraction and recognition using neural network.

Pre-processing is a process in which the image is carried out to make it ready for segmentation, in segmentation the character is segmented individually and then feature extraction technique is applied and finally the convolutional neural network algorithm is used for recognition and the extracted features are given as input for neural networks. The characters are resized into 28×28 pixel. Handwritten character recognition technique applications is found in optical character recognition and more advanced intelligent character recognition systems. Handwritten character recognition has been one of the most fascinating and challenging area, in the field of pattern recognition. It contributes immensely to the advancement of automation process and it improves the interface between man and machine in many applications.

BLOCK DIAGRAM:



First the input image is provided and it is converted into gray scale image and is sized as (28×28) pixels CNN is used for recognition and the features are extracted from the input image and given to trained CNN model which recognizes and given desired output.

- preprocessing: In this process the given colour image is converted into gray scale image. Colour image has 3 channels red, green and blue known as RGB. This is converted to grayscale image which has only one channel so as it avoid the unwanted noise in the image. The given image may be of any sized this lead to loss of accurate prediction so resize it to (28×28) pixels.
- Feature extraction: This is the process of transforming the input data into a set of features which represent the input data. These features contain the relevant information about the input data so that the desired task can be performed. After resizing the image, pixel values are obtained in the form of 1D array which represents values between 155 and 0 based on pixel intensity.
- Classification: CNN is used as classifier for classifying the handwritten character from the input image. It consists of input layer, output layer end multiple hidden layers. The hidden layer consists of convolutional layer pooling layer and fully connected layer.

CONVOLUTIONAL NEURAL NETWORK:

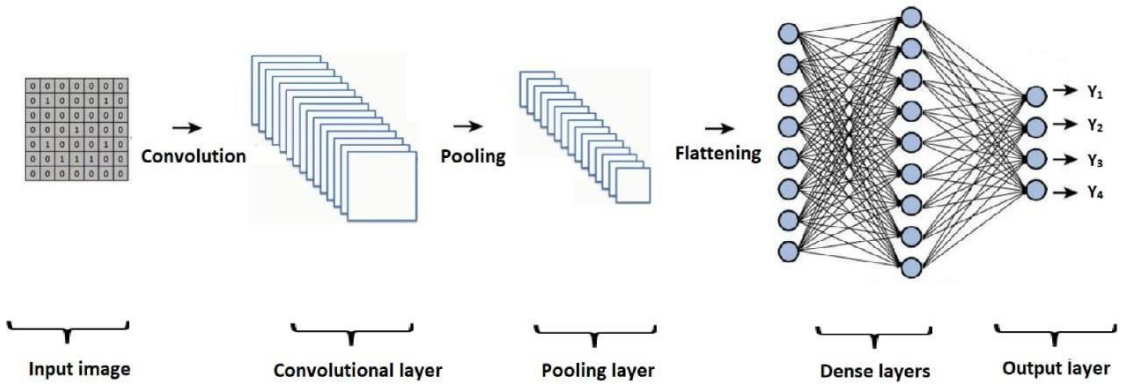
Convolutional neural network is a multilayer perceptron and is used to recognize 2D shapes with high degree of invariance.

CNN model consists of 3 layers

- convolutional layer
- pooling layer
- fully connected layer

1. convolutional layer-
   - This is done by multiplexing each image pixel by corresponding feature pixel.
   - This uses kernel side to get convolution input as output.
   - It extracts features as input.
   - As the number of convolutional layer increases the efficiency and parameters also increases.
2. pooling layer-
   - This layer is mainly used to discarded some unimportant information which reduce the calculation by less parameters.
   - Basically maxpooling is used, which means presence of strong feature.
3. Fully connected layer-

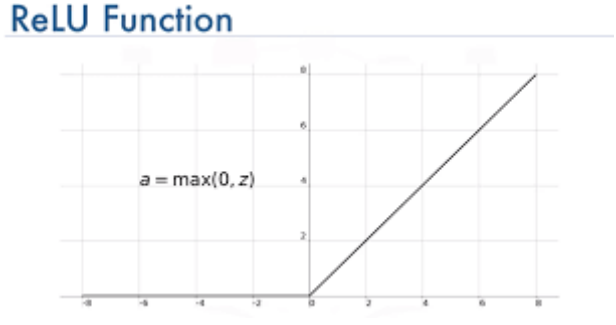
- This layer is used to reduce the affect to the features that are bought by pixel position when extracting as input.



CNN MODEL

ReLu:

- ReLu (Rectified linear unit) activation function is used.
- main advantage of ReLu function is that it doesn't activate all neuron at the same time
- ReLu layer is always used in hidden layer.
- This removes every negative value from filtered image and replace it with 0.
- Non linearity is introduced in this and also it doesn't have upper limits, so this is used as activation function.

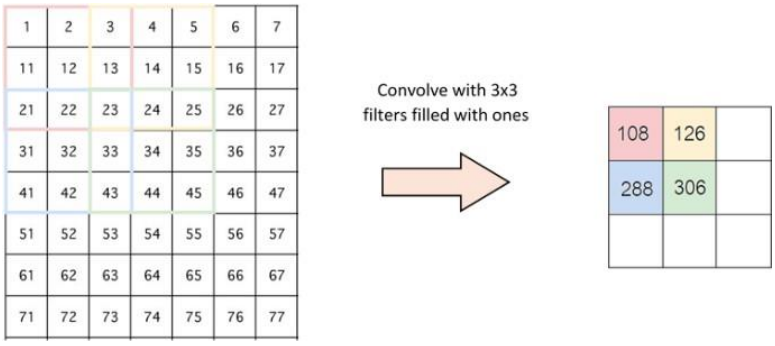


$$RELU(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; if \; x >= 0 \end{cases}$$

STRIDES

- Stride is the number of pixels shifts over the input matrix.
- When the stride is 1 then we move the filters to 1 pixel at a time.
- When the stride is 2 then we move the filters to 2 pixels at a time and so on.

The below figure shows convolution would work with a stride of 2



PADDING

- **Padding** is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN.
- For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero. If, however, the zero padding is set to one, there will be a one pixel border added to the image with a pixel value of zero.

SOFTMAX ACTIVATION FUNCTION

- The Softmax activation function calculates the relative probabilities. That means it uses the value of Z21, Z22, Z23 to determine the final probability value.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

- Here, the Z represents the values from the neurons of the output layer. The exponential acts as the non-linear function.Later these values are divided by the sum of exponential values in order to normalize and then convert them into probabilities

LITERATURE SURVEY

- "**Handwritten Character Recognition of MODI Script using Convolutional Neural Network Based Feature Extraction Method and Support Vector Machine Classifier**"

  - by solley Joseph ; Jossy George

Implementation of a feature extraction method using CNN auto encoder for MODI script character recognition is discussed in the paper. The extracted features are then subjected to Support Vector Machine (SVM) for the purpose of classification. MODI Script is an ancient Indian script and was used for writing Marathi until 1950.The MODI script has 46 characters, of which 36 are consonants and ten vowels. A feature extraction method using Backpropagation Neural Network and structure similarity was implemented for the recognition of MODI script and an accuracy of 93.5% was achieved. The method is implemented using an auto encoder with CNN for the feature extraction and SVM for classification. The original image (60*60 pixel size) is given as the input and data augmentation method will be applied to it. After the data augmentation, the data is passed through the CNN auto encoder for the extraction of features. Feature extraction is performed on the input image of 60*60 pixel size, using the CNN auto encoder. In the proposed method, an auto encoder with CNN is implemented for the feature extraction of MODI character recognition. The data set for the experiment consists of 4600 MODI characters, of which 3220 are used as train samples and 1380 as test samples (70:30 ratio). Handwritten characters written by different persons were used for the study. After pre-processing, the grayscale image was size-normalized to fit in a 60x60 pixel box. The output of the encoder is a 300-dimensional vector and thus, 300 features were extracted from the input image (of size 60x60). The classification is performed using Support Vector Machine with Radial Basis Function (RBF) kernel. The accuracy achieved is 99.3%.

- **"Optical Character Recognition for English Handwritten Text Using Recurrent Neural Network,"**

  -by R.Parthiban1,R.Ezhilarasi2,D.Saravanan3

Optical Character Recognition (OCR) passes on the which means of perceived English characters just as digits that perhaps pictures of manually written content, or might be simply PC content text styles of different kinds. It is an application programming which examinations and forms a picture record to perceive productively the characters present inside it. The picture record can be a written by hand and examined photograph. It makes an interpretation of pictures into unmistakable machine encoded editable content. It perceives just those characters for which the framework has been prepared for utilizing explicit arrangement calculation. This proposed framework introduces a Recurrent neural network for recognize English handwritten text. The proposed system's flow chart that consist of four main stages with output image. It RNN layers and a last Connectionist Temporal Classification (CTC) layer. Resize picture to 30*30 pixels and Convert to grayscale structure. Alter pixels force. Include cushioning of 2 pixels all sides.it is a grayscale picture of size 128×32. Randomly split data into training and testing set. The OCR has effectively distinguished the characters (digits and letters in order) from the picture of a manually written paper for example address of individual and composed digits. The Recurrent layer will decide the yield of neurons of which are associated with nearby locales of the contribution through the count of the scalar item between their loads and the district associated with the information volume. The amended direct unit (generally abbreviated to ReLu) plans to apply an 'elementwise' actuation capacity. The Long Short-Term Memory (LSTM) execution of RNNs is utilized, as it can spread data through longer separations and gives more vigorous preparing qualities than vanilla RNN. The RNN yield succession is mapped to a network of size 32×80. The IAM dataset comprises of 79 distinct characters. The output image is given as formal printed text. Word exactness rate (WAR) level of words with all characters effectively recognized.

- **"Offline Kannada Handwritten Character Recognition Using Convolutional Neural Networks"**

-by Ramesh G;Ganesh N Sharma;J Manoj Balaji;Champa H.N

This paper deals with the various pre-processing techniques involved in the character recognition with different kind of images ranges from a simple handwritten form based documents and documents containing colored and complex background and varied intensities. In this, different preprocessing techniques like skew detection and correction, image enhancement techniques of contrast stretching, binarization, noise removal techniques, normalization and segmentation, morphological processing techniques are discussed. In this paper hybrid Hidden Markov Model (HMM) model is proposed for recognizing unconstrained offline handwritten texts. In this, the structural part of the optical model has been modelled with Markov chains, and a Multilayer Perceptron is used to estimate the emission probabilities. In diagonal feature extraction has been proposed for offline character recognition. It is based on ANN model. Two approaches using 54 features and 69 features are chosen to build this Neural Network recognition system. To compare the recognition efficiency of the proposed diagonal method of feature extraction, the neural network recognition system is trained using the horizontal and vertical feature extraction methods. It is found that the diagonal method of feature extraction yields the recognition accuracy of 97.8 % for 54 features and 98.5% for 69 features.

- **"A Handwritten Character Recognition Algorithm based on Artificial Immune"**

-by Yuefeng Chen, Chunlin Liang*, Donghong Yang;Lingxi Peng*;Xiuyu Zhong

In order to improve the rate of character recognition and decrease the time of recognition training, referencing to immune biological principle, a handwritten character recognition algorithm based on artificial immune is proposed. The antigen and memory cell in the artificial immune system are described. The equations of clone selection principle and of evolving memory cell are established. Based on artificial immune theory, a handwritten character recognition algorithm based on artificial immune (HCRA) is proposed. The HCRA steals the merit of the adaptive mechanism of biology immune response. In the HCRA algorithm, after the antigen presentation, the input characters to be recognized are corresponding to the antigens of immune system. The memory cells set that can recognize characters are corresponding to the antibodies of immune system. The HCRA algorithm simulates the immune memory mechanism of biology immune system. After the immune training and learning, it evolves memory immune cells, and significantly improves speed and accuracy of recognition. The algorithm is a supervised classification learning methods. The HCRA algorithm has a higher recognition speed and accuracy than the neural network model in handwritten character recognition.

SOFTWARE TOOLS

Essential hardware and software devices used

Hardware: The hardware used in this project is a computer system
Software: The software used in this project is visual code studio. It is a code editor made by Microsoft for windows , Linux and macOS. It can be used with variety of programming languages, including Java, JavaScript, Python and C++.
We used the Jupyter Notebook application within visual code studio to implement the code for our project. It allows to create and edit documents that display the input of a python script.
The packages used:
OpenCV: Open-Source Computer Vision Library is a library of programming functions mainly aimed at real-time computer vision.
NumPy: It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.
Keras : Keras is an open-source software library that provides a Python interface for Artificial Neural Networks. Keras acts as an interface for the TensorFlow library. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a slack channel.
Tensorflow: It is an open source artificial intelligence library, using data flow graphs to build models. It allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for: Classification, Perception, Understanding, Discovering, Prediction and Creation.
Matplotlib: Matplotlib is a plotting library available for the Python programming language as a component of NumPy, a big data numerical handling resource. Matplotlib uses an object oriented API to embed plots in Python applications.
Pandas: pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

STEPS:

1. Importing the packages.
2. Importing the dataset and creating the corresponding list of all the classes of these images.
3. Splitting the data into training and testing for plotting the bar plot.
4. Reshaping the train and test image data in the CSV file.
5. Convert floating point values into integer values.
6. Plot horizontal bar graph using the count of alphabets.
7. Reshape the train and test image dataset.
8. Train the CNN model.
9. Print the training and validation accuracy.
10. Convert BGR representation into RGB representation.
11. Convert BGR to gray scale and apply threshold.
12. Resize and reshaping the image.
13. Predicting the character.
14. Closing all the windows.

CODE IMPLEMENTATION

- Importing the libraries:

Firstly, we input all the libraries necessary for code implementation.

```
[1]
import numpy as np
import matplotlib.pyplot as plt
import cv2
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.utils.np_utils import to_categorical
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import SGD, Adam
```

- Reading the data:

Reading the dataset using the pd.read_csv() and printing the first 10 images using data.head(10).

```
[2]
data = pd.read_csv(r"A_Z Handwritten Data.csv").astype('float32')

[3]
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]
```

The above image shows 10 rows and 785 columns(10x785).

- Split data into images and their labels:

```
[4]
X = data.drop('0',axis = 1)
y = data['0']
```

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

- Reshaping the data:

To display the data as image we reshape the data in csv file. Splitting the data into training & testing dataset using train_test_split(). Initially csv file has 784 columns of pixel data. Here we convert it into 28x28 pixels.

```
[5]
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:  (297960, 28, 28)
Test data shape:  (74490, 28, 28)
```

The labels are present in the form of floating point values, so we convert it into integer values and create a dictionary word_dict to map the integer values with the characters.

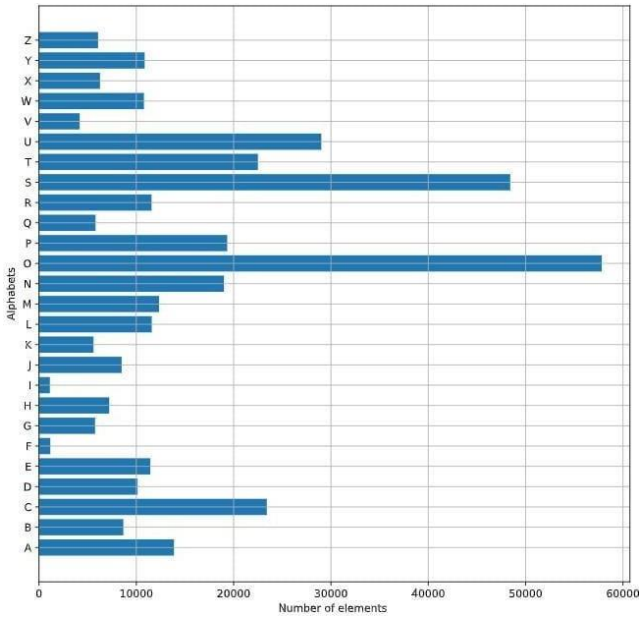
```
[6]
word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',
15:'P',16:'Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}
[7]
```

- Plotting the number of alphabets in the dataset:

Converting the labels into integer values an append into the count list according to the label. This count list has the number of images belonging to each alphabet which is present in the dataset.

```
[7]
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()
```

Create list of alphabets containing all the characters using the values function of the dictionary. Using the count of alphabets plot a horizontal bar graph.



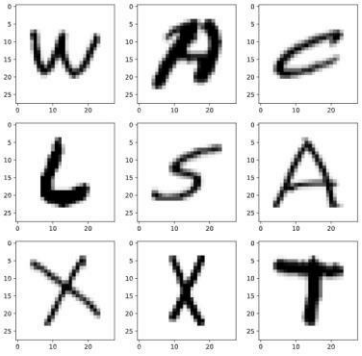
➢ Shuffling the data:

Shuffled some of the images of trained set. The shuffling is done using shuffle() function to display some random images and then create 9 plots in 3x3 shape and display the threshold image of 9 alphabets.

```
[8]  ▷ M↓
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()
```

Output:

The below image shows the gray scale images that is obtained from the dataset (3x3 shape).



- Data Reshaping:

Reshaping the training and test dataset to train the model.

```
[21]
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 test data: ", test_X.shape)

New shape of train data:  (297960, 28, 28, 1)
New shape of test data:  (74490, 28, 28, 1)
```

New shape of training data and testing data is of 28x28 pixel and training data is of 30%, testing data is of 70% out of total dataset.

```
[10]
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:  (297960, 26)
New shape of test labels:  (74490, 26)
```

Converting single float value into categorical values. This is done by CNN model which takes input of labels and generates output.

CNN MODEL

- Training the model:

```
[11]
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"))
```

CNN model which is designed for training the model over training dataset. Sequential model allows us to build a model layer by layer. First 2 layers are Conv2D layers. The model is of 32 layers (filters) of kernel size 3x3 matrix. ReLu activation function is used and input size is 28x28 pixels maxpooling is done.

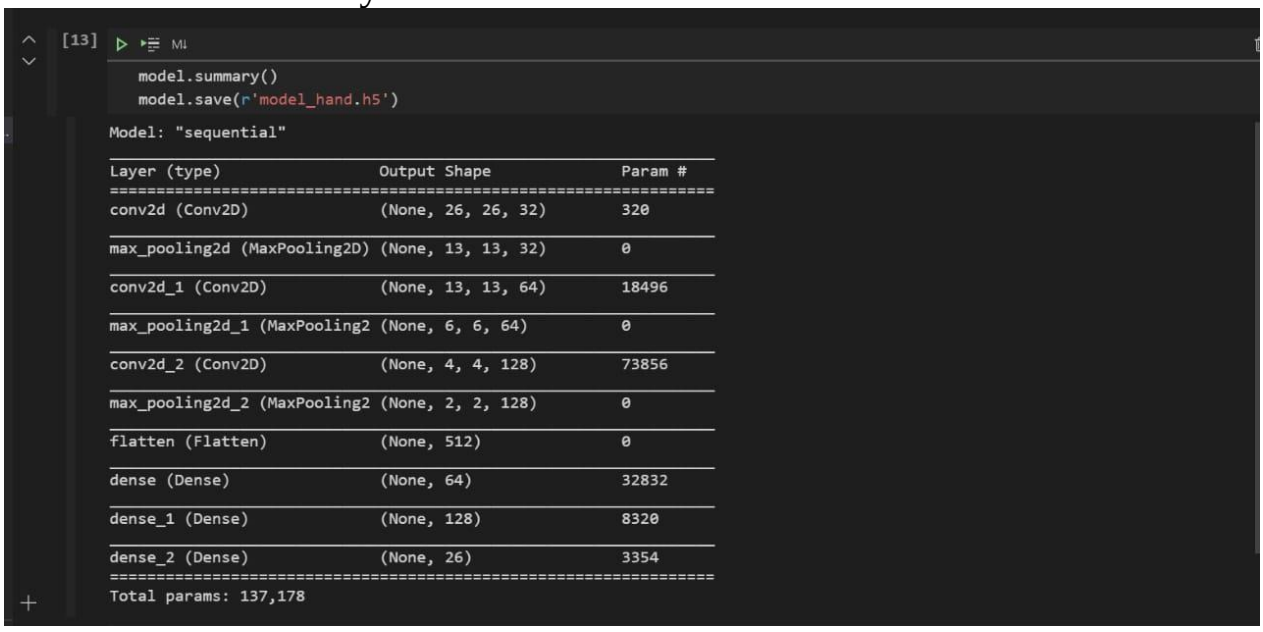
- Compiling and fitting model:

```
[12]
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))

9312/9312 [==============================] - 209s 21ms/step - loss: 0.3818 - accuracy: 0.9069 - val_loss: 0.1063 - val_accur
acy: 0.9687
```

The trained model is compiled to define the optimizing function and loss of function to be used for fitting. The optimizing function used is Adam. Adam is an adaptive learning rate optimization algorithm that is designed specially for training the deep meural networks. This is a combination of RMSprop and Adagram optimizing algorithm.

As the dataset is very large we are training per only a single epoch, however, as required we can even train it for multiple epochs, which is recommended for better accuracy.

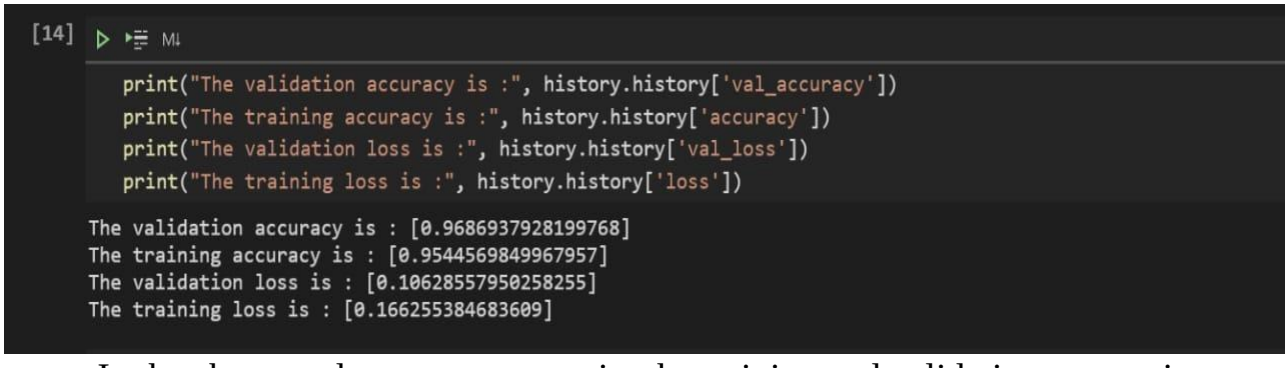
- Model summary



```
[13]
model.summary()
model.save(r'model_hand.h5')

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 64)                32832
_________________________________________________________________
dense_1 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_2 (Dense)              (None, 26)                3354
=================================================================
Total params: 137,178
```

Getting the summary of the model that defines the different layers of the model and save the model using model.save() function.

- Train and validation accuracies and losses



```
[14]
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 : [0.9686937928199768]
The training accuracy is : [0.9544569849967957]
The validation loss is : [0.10628557950258255]
The training loss is : [0.166255384683609]
```

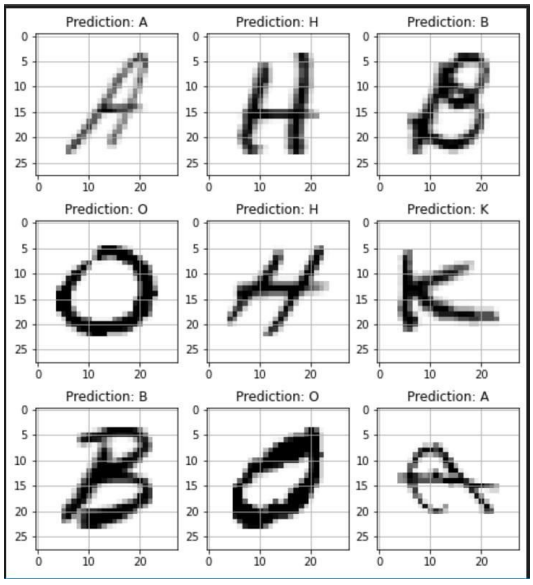
In the above code segment, we print the training and validation accuracies along with the training & validation losses for character recognition.

- Predictions on test data

```
[15]
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()
```

Creating 9 subplots of (3,3) shape and visualize some of the test dataset alphabets along with their predictions, that are made using the model.predict() function for text recognition.

Output



- Prediction of final image

Read an external image that is the original image of alphabet 'Z' and allowed it to go through some processing to be fed to the model for the prediction.

The image read is converted from BGR representation to RGB for displaying the image and resize to our required dimensions.

No we do some processing on the copied image (img_copy). Now convert the image from BGR to GRAY scale and apply threshold to it.

The image is resized using cv2.resize() function and reshaped using np.reshape() function into the required dimension that the model can take as input.

Now the prediction is made using the processed image and np.argmax() function is used to get the index of class with the highest predicted probability. Using

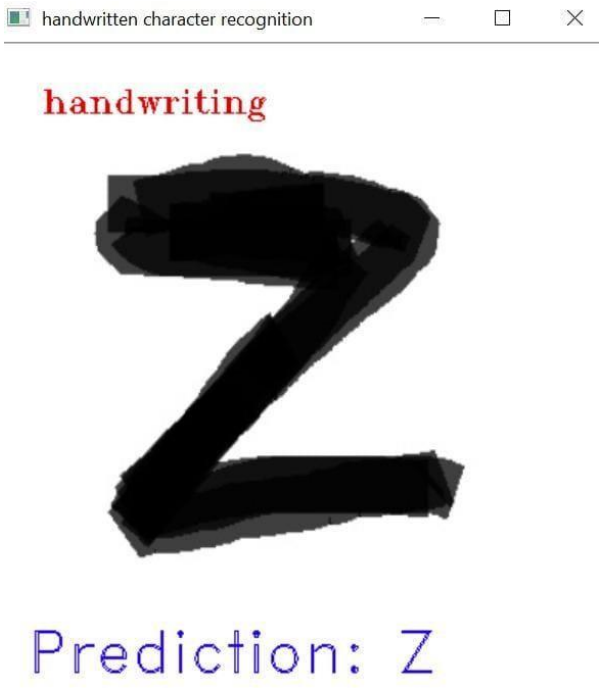
this we get to know the exact character through the word_dict dictionary.

The predicted character is displayed on the frame.

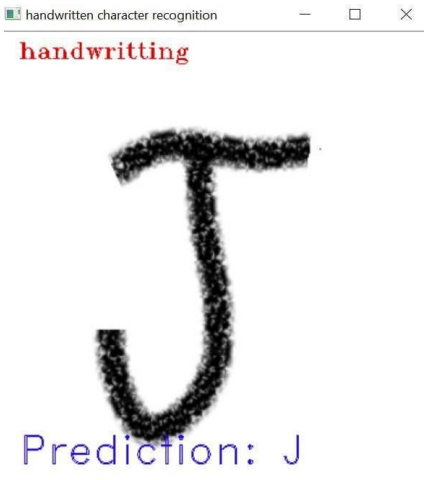
Here we are setting up a waitKey in a while loop that will be stuck in loop until Esc is pressed, and when it gets out of loop using cv2.destroyAllWindows() we destroy any active windows created to stop displaying the frame.

```
[19]
img = cv2.imread(r'z.jpg')
img_copy = img.copy()

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #color converting(bcs it will be color inverted)
img = cv2.resize(img, (400,440))

img_copy = cv2.GaussianBlur(img_copy, (7,7), 0) #removing edges
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY) #convertcolor converted to gray scale
_, 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, "handwriting ", (30,45), 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(' handwritten character recognition ', img)

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

- Predicted character



- Examples of prediction



CONCLUSION

Handwritten character recognition has been a challenging task in the past few years. Due to development of machine learning and deep learning in recent years enormous improvement is seen. In this project the recognition of handwritten characters is done using neural networks (CNN). CNN with the help of training data is allowed per classification and to recognize different handwritten characters. In our project the handwritten characters are recognized with the accuracy of 96.87% and prediction of characters was almost up to the mark.

This technique can be implemented to recognize other languages and also digits. Handwritten character recognition is of two types, online and offline handwritten recognition which can be used for recognizing other languages. Various pre-processing, segmentation methods, feature extraction process, classification techniques are discussed in detail. Many regional languages throughout world have different writing styles which can be recognized with HCR systems using proper algorithm and strategies. We have learning for recognition of English characters.

FUTURE SCOPE OF HANDWRITTING RECOGNITION

Electronic form filling

One of the applications of online handwriting recognition is electronic form filling. Internationally, the expenditure for entry of data from handwritten forms, notes and records is trillions of dollars. If we look at 2010 census of our country, more than fifty thousand enumerators were employed to collect data using handwritten forms, where they took six months to do this job. Further, it took two more years to feed this data into servers. So, in such applications, an immediate and direct conversion of handwritten data to typed data will result in reducing the huge cost and it will also increase the productivity.

Automatic conversion of prescription to typed form

It is one of the challenging tasks to understand the doctor's handwriting. This problem can be solved by using offline and online handwriting. In case of automatic conversion of prescription to typed form, online handwriting will be employed.

Biometrics and forensics

The handwriting recognition is highly applicable in writer identification, where it used in forensics and biometrics. So one of the applications of handwriting recognition is solving the ancient manuscript disputes, where the actual writer of the manuscript is identified based on certain features of writers' handwriting. In this way, it assists to avoid false claims of handwriting or manuscript.

Digitization of palm leaf manuscripts

As the offline handwriting recognition deals with understanding the handwritten text which is already written on papers or other writable surfaces or materials. So one of the applications of offline handwriting recognition is the digitization of ancient palm leaf manuscripts. This problem is not solved yet. It is not even solved for Latin script completely.

REFERENCES

- S. Joseph and J. George, "Handwritten Character Recognition of MODI Script using Convolutional Neural Network Based Feature Extraction Method and Support Vector Machine Classifier," 2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP), 2020, pp. 32-36, doi: 10.1109/ICSIP49896.2020.9339435.
- R. Parthiban, R. Ezhilarasi and D. Saravanan, "Optical Character Recognition for English Handwritten Text Using Recurrent Neural Network," 2020 International Conference on System, Computation, Automation and Networking (ICSCAN), 2020, pp. 1-5, doi: 10.1109/ICSCAN49426.2020.9262379.
- M. Rajalakshmi, P. Saranya and P. Shanmugavadivu, "Pattern Recognition-Recognition of Handwritten Document Using Convolutional Neural Networks," 2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS), 2019, pp. 1-7, doi: 10.1109/INCOS45849.2019.8951342.
- Yuefeng Chen, Chunlin Liang, Donghong Yang, Lingxi Peng and Xiuyu Zhong, "A handwritten character recognition algorithm based on artificial immune," 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), 2010, pp. V12-273-V12-276, doi: 10.1109/ICCASM.2010.5622270.
- G. Ramesh, G. N. Sharma, J. M. Balaji and H. N. Champa, "Offline Kannada Handwritten Character Recognition Using Convolutional Neural Networks," 2019 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE), 2019, pp. 1-5, doi: 10.1109/WIECON-ECE48653.2019.9019914.
- M. H. Alkawaz, C. C. Seong and H. Razalli, "Handwriting Detection and Recognition Improvements Based on Hidden Markov Model and Deep Learning," 2020 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA), 2020, pp. 106- 110, doi: 10.1109/CSPA48992.2020.9068682.