Abstract: The main aim of the project is Handwritten Text Recognition (HTR). HTR is the task of transcribing images of handwritten text into the digital text. In HTR, the text is written, captured by a scanner and then the resulting images are processed as input to return its text format. We need openCV and CNN for achieving this task. Our goal is to design a model that transcribes the images to text with great accuracy.

Keywords – CNN, Handwritten Text Recognition (HTR), openCV, Transcription

I. INTRODUCTION

Now a day’s people have been using ebooks which do not occupy any space and ample copies can be carried comfortably. So there is a need for making more ebooks available. We have more number of handwritten texts available all over the world which need protection be safeguarded. By transcribing them we can increase the availability of ebooks. And also instead of striving hard to protect old texts which are hand written, they can be digitilized and stored as soft copies with ease.

We will apply the machine learning techniques in order to find the digital form of handwritten text from their scanned images. We will take the help of the readymade datasets that contain pixel values of scanned images as the inputs and we will be able to find the text in it. We can also extend this project for different languages and writing styles.

Problem Statement: To accurately predict the text from a scanned handwritten text image using Machine Learning algorithms.

For this we need to assume that all the images that contain same letters have same features and we can conclude that an image having those features contains that alphabet. However this hypothesis is ideal and may not come true always in practical.

II. DATASETS

The data plays a very important role in machine learning. The past data is used to predict the future outcome. The relevant data can be downloaded from the internet.

The data that is related to our project that is HTR consists of pixel values. The format of the data files is csv (Comma Separated Values). Each row represents an image and contains a lable in the first column and followed by 784 pixel values for 28 X 28 images.

The data related to our project will have thousands of instances. The data that we use for our project is obtained from kaggle.

Two types of data will be taken
✓ One for English alphabets
  A_Z Handwritten Data [1] and
✓ The other for digits - MNIST dataset [2]

DATA VISUALIZATION

For dataset_1 the shape will be (372450, 785) and for dataset_2 the shape will be (42000, 785).

A_Z Handwritten Data:

The dataset will comprise of multivariate data of English alphabets. Here the dataset will have a label and pixel values which lie from 0 to 255. There will be total of 327450 instances, and the total number of attributes is 784 and a label.

0_9 Handwritten Data:

The dataset will comprise of multivariate data of digits from 0 to 9. Here the dataset will have a label and pixel values which lie from 0 to 255. There will be total of 42000 instances, and the total number of attributes is 784 and a label.

Fig 1. Bar diagram showing lable sizes of different labels
Table 1. First five instances of the dataset

<table>
<thead>
<tr>
<th>label</th>
<th>pixel0</th>
<th>pixel1</th>
<th>pixel2</th>
<th>pixel3</th>
<th>pixel4</th>
<th>pixel5</th>
<th>pixel6</th>
<th>pixel7</th>
<th>pixel8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>

III. ARCHITECTURE

The problem of converting a handwritten text which is in the form of pixels into its digital form is a data-driven approach. The data which is already collected can be used for extracting the features of each letter. The availability of more powerful machine learning algorithms introduces an efficient and better approach to solve this problem.

The project is divided into two modules.

A Segmentation module in which an image is taken as input, letters are detected, bounded, cropped, resized and then segmented and a Training module where prediction occurs.

The output of the segmentation module is the input of the training module.

IV. METHODOLOGY

The research methodology in this project include,

- Visualizing and understanding the data
- Choosing a suitable model
- Agreeing on a common evaluation metric
- Training and Testing the models
- Implementing the final model
- Analyzing the result

V. EXPERIMENTAL RESULTS

A. Segmentation Module

Segmentation model is very important for this project as its output will be the input of the other module.

1) Read the image

We have many ML libraries like Pillow, openCV etc. for performing operations on images. Here openCV is used to read and manipulate images. An image is read and then stored in multiple copies for performing different operations.

After reading the image is plotted in its shape to make sure it is read perfectly. That image contains letters that need to be images each cropped into 28 x 28 images by the end of segmentation model.

Fig 4. An image of dimensions 351 X 232 pixels

2) Detecting the letters

Object detection is a Computer vision technique that detects certain components from an image or a video. It makes use of Machine Learning and Deep Learning Algorithms to yield good results. Detecting the letters is same as detecting objects. We need to apply some standard filters to the input image for achieving this task.

Step 1: Convert a BGR image to Grayscale image.

An image with 3 channels is a BGR image but a Grayscale image consists of a single channel. A channel is a third dimension of an image.
Step 2: Applying Gaussian blur to the Grayscale image.
This step is done to remove any noise and disturbances in
the image. If the image is blurred the colour intensities can
be recognised easily from the image. The blurring is
technically called Gaussian Blur in the Computer Vision.

Step 3: Otsu Thresholding → A standard step for object
detection
It is calculation of the measure of spread for the pixel
levels each side of the threshold.

Step 4: Finding and Drawing contours
The ‘findcontours()’ and ‘drawcontours()’ are the methods
used for finding and drawing contours which are generally
borders of a detected object in an image.
The drawcontours need image, detected contours, colour
and dimensions of the border as the parameters.

Step 5: Storing co-ordinates for rectangular bounding
Boundingrect() gets the list of x, y co-ordinates of top left
point of the image, width and height allowing us to
drawimages in the order of detected objects. We need to
sort in the order of x co-ordinate of the top left corner to
order them. All these lists are stored in a list and sorted
with a base of list[i][0]th elements.

3) Bounding and Cropping
This part of code is added before cropping in order to add
spaces between letters. In the list that contains bounding
details, we add a space string “ “ if the distance between the
corresponding x co-ordinates is > 50 pixel values
representing they must be separated by a space.

And then Bounding and Cropping for non-spaces and
storing them in a list ‘img_lst’.

The elements in the list of detected objects may be the
boundingrect values of a detected object from the image or
a space string. The detected object is considered a letter
only if its height is greater than 20 pixels. (Our assumption)
Then for those which are considered letters we use
boundingrect values to crop the letter from another copy of
the original image stored in another variable and append
each of the cropped images which are in numpy array form
into a python list.

‘rectangle()’ draws a rectangular border around detected
letters which has the image, bounding values, colour and
width of the border.

4) Resizing
For each cropped image in the list, we resize them to 28 X
28 pixels. We do so because the output images from this
module which are going to be the input of the other module
must be images of size 28 X 28 pixels as the training data
of that module is of that format.
Each resized images must be converted to Grayscale to match requirements of training model. And then sent as input to the training model.

The subplot of the output of segmentation module looks like

![Segmentation Output](image)

**B. Training Module**

A model is trained by using past data and Machine learning Algorithms. It learns from the past data by feature extraction and patterns. In this project Convolutional Neural Networks are used. We split the data into training and testing in the ratio of 80:20.

The dataframe with attributes, the dataframe with only label column, train_size or test_size and shuffled are important parameters of train_test_split. Shuffled will be ‘True’ by default which shuffles the data before splitting. This method need not always return same output.

Scikit-learn library is used to change the form of data. We need to convert the attribute values to the float datatype and their labels into the categorical form to train them.

The attribute values are converted to floating point numbers ranging from 0 to 1 which earlier were 0 to 255. A pixel value of 0 will be 0.0000, a pixel value of 1 will be 1.0000, and a pixel value of 128 will be 0.5000.

Categorical form results in a list of size equals number of all possible labels in which an instance with label value $i$ will have $i = 1$ and other elements 0. Even these values must be in float.

The categorical form will look like this:

```python
Y_train[8]
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

Now we can proceed to train our data by using standard Machine Learning Algorithms. A sequence of hidden layers are created with some nodes in each of them. The first hidden layer is 2Dimensional Convolution layer with kernal size 5 X 5 and 32 nodes. The activation function used is ‘relu’. And then max pooled with 2 X 2 and over fitting is reduced using dropout of 0.3. Then two more layers are added with 128 and $n$ nodes respectively, where $n$ is the number of possible outputs (here $n = 36$).

Now we compile the model by ‘categorial_crossentropy’ loss function, ‘adam’ optimizer with a metric of ‘accuracy’.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_1 (Conv2D)</td>
<td>(None, 24, 24, 32)</td>
<td>852</td>
</tr>
<tr>
<td>max_pooling2d_1 (MaxPooling2D)</td>
<td>(None, 12, 12, 32)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_1 (Dropout)</td>
<td>(None, 12, 12, 32)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_1 (Flatten)</td>
<td>(None, 4608)</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 128)</td>
<td>589992</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 36)</td>
<td>4664</td>
</tr>
</tbody>
</table>

Total params: 595,428
Trainable params: 595,428
Non-trainable params: 0

<table>
<thead>
<tr>
<th>Model</th>
<th>Loss</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>0.1381</td>
<td>0.9626</td>
</tr>
</tbody>
</table>

The fitting of a model is shown below. It is passed with 1 iteration. 200 batch size and the dataset contains 331560 instances.

```python
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=1, batch_size=200, verbose=2)
```

Instructions for updating:
Use tf.cast instead.
Train on 331560 samples, validate in 82890 samples
Epoch 1/1
- 350s - loss: 0.3029 - acc: 0.9184 - val_loss: 0.1381 - val_acc: 0.962

**VI. RESULTS**

The above model will give us the train accuracy of 0.9184 and the test accuracy of 0.9626.

```python
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: {}% \% (scores[1]*100)\n".format(scores[1]*100))
```

Accuracy: 96.26%

<table>
<thead>
<tr>
<th>Model</th>
<th>Loss</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>0.1381</td>
<td>0.9626</td>
</tr>
</tbody>
</table>

**VII. CHALLENGES INVOLVED**

The challenges we have faced while modelling handwritten text recogniser are

- Letters like ‘i’and ‘j’which have break in them cannot be detected as a single letter.
(ii) If two letters touch each other (like in cursive writing), they are recognized as a single letter.

VIII. CONCLUSION

Convolutional Neural Network learns from the real time data and simplifies model by reducing the number of parameters and hence gives considerable accuracy.

Future Enhancements

We can increase the accuracy:

→ By taking huge datasets
→ By adopting much suitable algorithms
→ We can compile the model at more number of epochs.
→ Hyper-parameter Tuning (There are a lot of parameters that we can play with).
→ Use of deeper architectures

This application can be taken to next level by

→ Extending its scope to different writing styles
→ Extending its scope to different writing styles

ACKNOWLEDGEMENT

We would like to acknowledge the help of ‘JNTUA College of Engineering, Pulivendula’ for the kind support provided and our faculty and friends for the helpful discussions. We also would like to thank ‘kaggle’ who provided datasets.