

INTERNSHIP: PROJECT REPORT

Internship Project Title	Automate identification and recognition of handwritten text from an image
Name of Student	Chinmay M. Haridas
Name of the Company	TCS iON
Name of the Industry Mentor	Debashish Roy
Name of the Industry Mentor	Prof.Anilkumar Bhadgale
Name of the Institute	Pune Vidhyarthi Griha's College of Engineering,Pune

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
14 Oct 2021	30 Dec 2021	210	Jupyter Notebook, Anaconda Navigator.	IAM dataset, Tensorflow 2.6.0, Keras – 2.6.0, Jupyter Notebook, GPU of NVIDIA GEFORCE GTX 1050

Project Report:

- **Title of the Project** - Automate identification and recognition of handwritten text from an image.
- **Introduction** - This is TCS iON-RIO 210-Industry Project. This project is basically focused on identification and recognition of handwritten text from an image. It is based on enhancement of optical character recognition system. Optical character recognition or optical character reader is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo or from subtitle text superimposed on an image. An optical character recognition problem is basically a type of image-based sequence recognition problem. And for sequence recognition problem, most suited neural networks are recurrent neural networks (RNN) while for an image-based problem most suited are convolution neural networks (CNN). To cop up with the OCR problems we need to combine both of these CNN and RNN. The purpose of OCR in text identification and recognition from an image is to extract handwritten data to digital data. So, one can easily handle this digital data by editing, adding new information in that text. OCR is developing field of Computer Vision, Pattern Recognition and Artificial Intelligence.
- **Objective and Aim** - To develop machine learning algorithms in order to enable entity and knowledge extraction from documents with handwritten annotations, with an aim to first identify handwritten words on an image and then recognize the characters to transcribe the text.

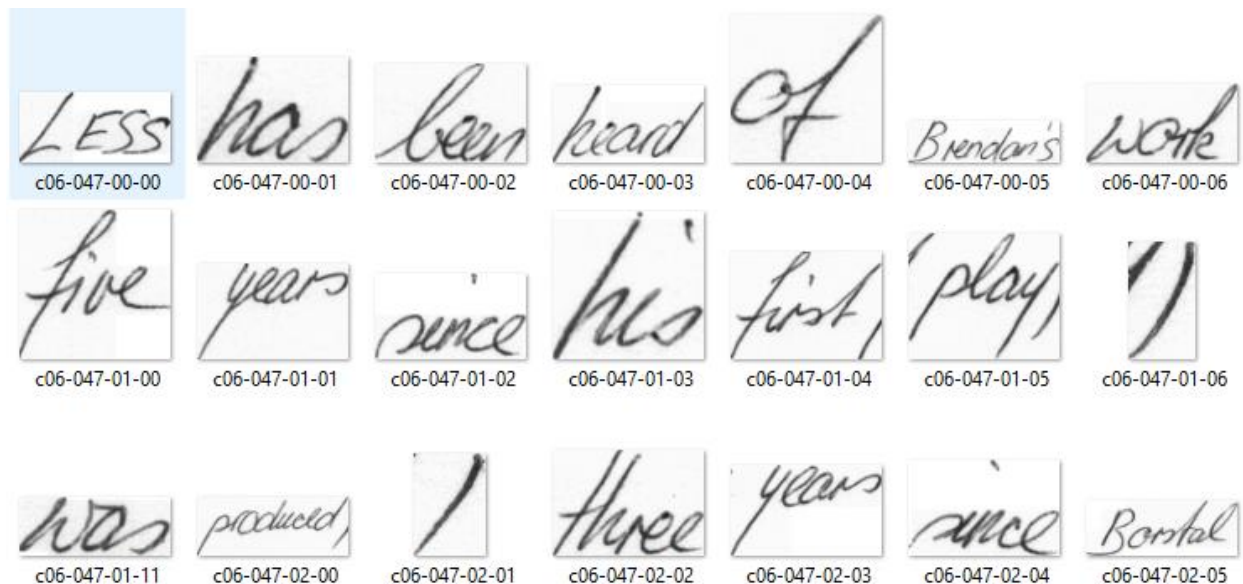
Solution Approach –

An optical character recognition problem is basically a type of image-based sequence recognition problem. And for sequence recognition problem, most suited neural networks are recurrent neural networks (RNN) while for an imagebased problem most suited are convolution neural networks (CNN). To cop up with the OCR problems we need to combine both of these CNN and RNN. So, I used Convolutional Recurrent Neural Network (CRNN) to tackle the both the problems

1. Uploading Dataset
2. Collecting Dataset
3. Preprocessing Data
4. Creating Network Architecture
5. Defining Loss Function
6. Training Model
7. Testing and Prediction

Collecting Dataset -

This is one of the main task to implement our model effectively. The features of data provided in the project guidelines matches with IAM dataset. IAM dataset have cursive handwriting, poor image quality generated from scanned documents and skewed images. So, I decided to go with IAM dataset for this project. This is large dataset total of 1.09 GB (115320-Images). Here I have used only 7850 images for the training set and 876 images for validation dataset. This data contains text image segments which look like images shown below:



Validation dataset

```
a01-000u-00-00 ok 154 408 768 27 51 AT A
a01-000u-00-01 ok 154 507 766 213 48 NN MOVE
a01-000u-00-02 ok 154 796 764 70 50 TO to
a01-000u-00-03 ok 154 919 757 166 78 VB stop
a01-000u-00-04 ok 154 1185 754 126 61 NPT Mr.
a01-000u-00-05 ok 154 1438 746 382 73 NP Gaitskell
```

Preprocessing Data-

Now we have our dataset, to make it acceptable for our model we have to use preprocessing of our dataset. We have to preprocess both input images and output labels. To Preprocess input images we

have to follow the below steps:

- Read the image and convert it into a gray-scale image.
- Make each image of size (128, 32) using padding.
- Expand image dimension as (128,32,1) to make it compatible with the input shape of architecture
- Normalize the image pixel values by dividing it with 255

To preprocess the output labels follow the below steps:

- Read the text from the words.txt file. This file contains every image text.
- Encode each character of a word into some numerical value by creating a function.
- Compute the maximum length from words and pad every output label to make it of the same size as the maximum length. This is done to make it compatible with the output shape of our RNN architecture. In preprocessing we need further two lists. One is for label length and other is for input length to our RNN. These two lists are important for our CTC loss. Label length is the length of each output text label and input length is the same for each input to the LSTM layer which is 31 in our architecture.

4) Creating Network Architecture – Input shape for our architecture having an input image of height 32 and width 128. Here we used seven convolution layers of which 6 are having kernel size (3, 3) and the last one is of size (2,2). And the number of filters is increased from 64 to 512 layer by layer. Two max-pooling layers are added with size (2, 2) and then two max-pooling layers of size (2, 1) are added to extract features with a larger width to predict long texts. Also, we used batch normalization layers after fifth and sixth convolution layers which accelerates the training process. Then we used a lambda function to squeeze the output from conv layer and make it compatible with LSTM layer. Then used two Bidirectional LSTM layers each of which has 128 units. This RNN layer gives the output of size (batch_size, 31, 63). Where 63 is the total number of output classes including blank character.

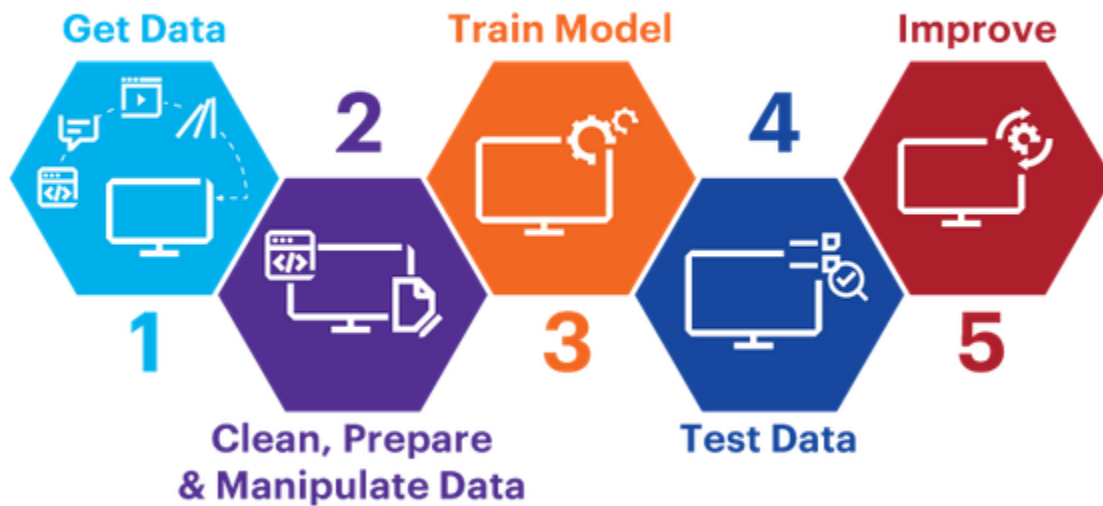
5) Defining Loss Function – Now we have prepared model architecture, the next thing is to choose a loss function. In this text recognition problem, we will use the CTC loss function. CTC loss is very helpful in text recognition problems. It helps us to prevent annotating each time step and help us to get rid of the problem where a single character can span multiple time step which needs further processing if we do not use CTC. A CTC loss function requires four arguments to compute the loss, predicted outputs, ground truth labels, input sequence length to LSTM and ground truth label length. To get this we need to create a custom loss function and then pass it to the model. To make it compatible with our model, we will create a model which takes these four inputs and outputs the loss.

6) Training Model – To train the model I used Adam optimizer. Also, we can use Keras callbacks functionality to save the weights of the best model on the basis of validation loss. In model.compile(), I have only taken y_pred and neglected y_true. This is because I have already taken labels as input to the model earlier. Labels as input to the model earlier. Now train our model on 7850 training images and 876 validation images.

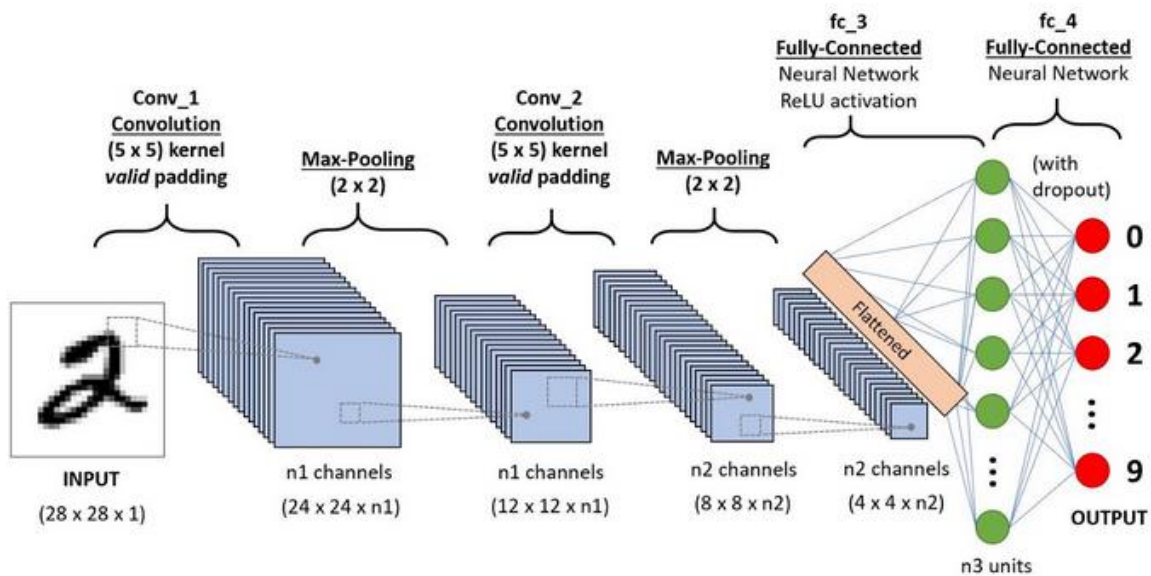
7) Testing and Prediction – Our model is now trained with 7850 images. Now its time to test the model. We cannot use our training model because it also requires labels as input and at test time we cannot have labels. So to test the model we will use "act_model" that we have created earlier which takes only one input: test images. As our model predicts the probability for each class at each time step, we need to use some transcription function to convert it into actual texts. Here I used the CTC decoder to get the output text. I used Jaro Distance & Ratio method to test accuracy. This approach I used for solving this problem. I also used other approaches but when we increase the complexity of image then this models not good so I continued with the CRNN model.

Project Diagrams:

1. Block Diagram:



2. CNN Network Model Architecture-





- 1) Convolutional layers, which extract a feature sequence from the input image
- 2) Recurrent layers, which predict a label distribution for each frame
- 3) Transcription layer, which translates the per-frame predictions into the final label sequence.

LSTMs deal with both Long Term Memory (LTM) and Short Term Memory (STM) and for making the calculations simple and effective it uses the concept of gates.



1. **Forget Gate:** LTM goes to forget gate and it forgets information that is not useful.
2. **Learn Gate:** Event (current input) and STM are combined together so that necessary information that we have recently learned from STM can be applied to the current input.
3. **Remember Gate:** LTM information that we haven't forget and STM and Event are combined together in Remember gate which works as updated LTM.
4. **Use Gate:** This gate also uses LTM, STM, and Event to predict the output of the current event which works as an updated STM.

Algorithms:

Model = CNN + RNN + CTC loss

Our model consists of three parts:

1. The convolutional neural network to extract features from the image
2. Recurrent neural network to predict sequential output per time-step
3. CTC loss function which is transcription layer used to predict output for each time step. Model Architecture Here is the model architecture that we used.

Model Architecture-

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 128, 1)]	0
conv2d (Conv2D)	(None, 32, 128, 64)	640
max_pooling2d (MaxPooling2D)	(None, 16, 64, 64)	0
conv2d_1 (Conv2D)	(None, 16, 64, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 8, 32, 128)	0
conv2d_2 (Conv2D)	(None, 8, 32, 256)	295168
conv2d_3 (Conv2D)	(None, 8, 32, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 4, 32, 256)	0
conv2d_4 (Conv2D)	(None, 4, 32, 512)	1180160
batch_normalization (Batch Normalization)	(None, 4, 32, 512)	2048
conv2d_5 (Conv2D)	(None, 4, 32, 512)	2359808
batch_normalization_1 (Batch Normalization)	(None, 4, 32, 512)	2048
max_pooling2d_3 (MaxPooling2D)	(None, 2, 32, 512)	0
conv2d_6 (Conv2D)	(None, 1, 31, 512)	1049088
lambda (Lambda)	(None, 31, 512)	0
bidirectional (Bidirectional)	(None, 31, 512)	1574912
bidirectional_1 (Bidirectional)	(None, 31, 512)	1574912

Algorithm of CRNN Model-

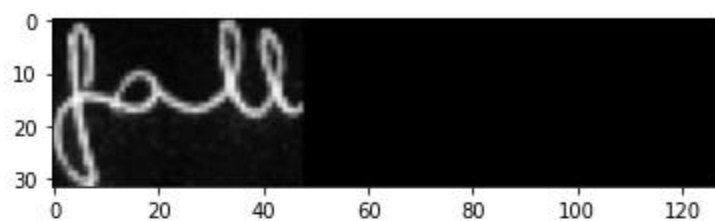
1. Input shape for our architecture having an input image of height 32 and width 128.
2. Here we used seven convolution layers of which 6 are having kernel size (3, 3) and the last one is of size (2,2). And the number of filters is increased from 64 to 512 layer by layer.
3. Two max-pooling layers are added with size (2, 2) and then two max-pooling layers of size (2, 1) are added to extract features with a larger width to predict long texts.
4. Also, we used batch normalization layers after fifth and sixth convolution layers which accelerates the training process.
5. Then we used a lambda function to squeeze the output from conv layer and make it compatible with LSTM layer.
6. Then used two Bidirectional LSTM layers each of which has 128 units. This RNN layer gives the output of size (batch_size, 31, 63). Where 63 is the total number of output classes including blank character.

Outcome:

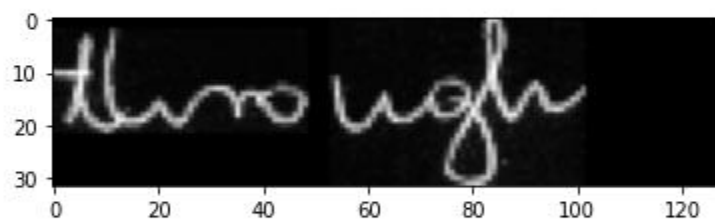
The algorithm is able to detect and segment handwritten text from an image. The model successfully able to detect maximum words, which makes it about 99.69% accurate while implementation and testing.

As we can see the model is accurate and successfully able to extract the handwritten text. The model identify and recognize the text from the image as follows.

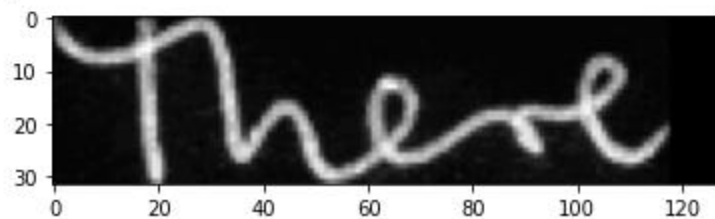
```
original_text = fall  
predicted text = fall
```



```
original_text = through  
predicted text = through
```



```
original_text = There  
predicted text = There
```



Accuracy and Loss Plots:

```
Epoch 00024: val_loss did not improve from 3.41522
Epoch 25/28
982/982 - 58s - loss: 0.0341 - accuracy: 0.9932 - val_loss: 3.4709 - val_accuracy: 0.5982

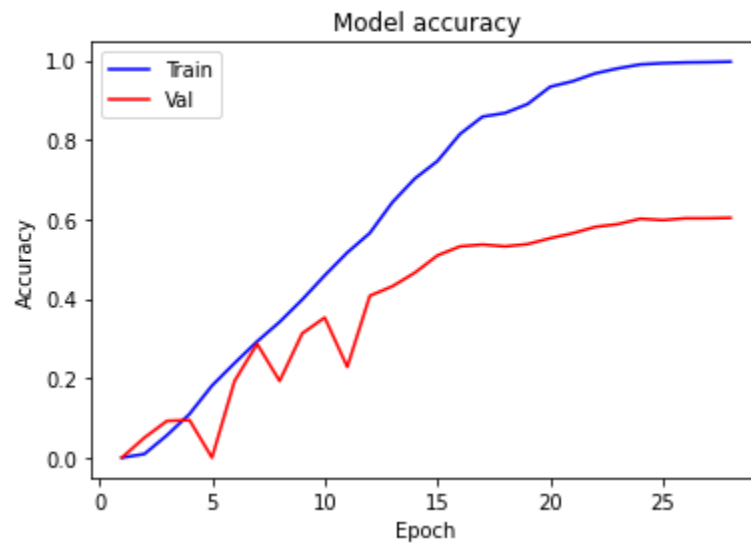
Epoch 00025: val_loss did not improve from 3.41522
Epoch 26/28
982/982 - 58s - loss: 0.0260 - accuracy: 0.9950 - val_loss: 3.5296 - val_accuracy: 0.6027

Epoch 00026: val_loss did not improve from 3.41522
Epoch 27/28
982/982 - 58s - loss: 0.0213 - accuracy: 0.9958 - val_loss: 3.5096 - val_accuracy: 0.6027

Epoch 00027: val_loss did not improve from 3.41522
Epoch 28/28
982/982 - 59s - loss: 0.0179 - accuracy: 0.9969 - val_loss: 3.4994 - val_accuracy: 0.6039
```

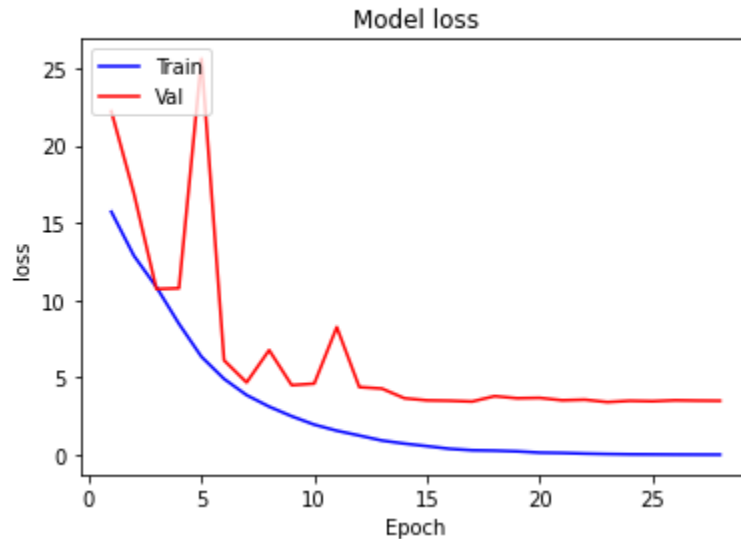
Accuracy-

```
In [43]: plotgraph(epochs, acc, val_acc)
```



Loss-

```
In [39]: plotgraph(epochs, loss, val_loss)
```



Enhancement Scope:

The enhancement scope of this project are follows:

- 1) The accuracy of the model can increased with predefined models and powerful machine learning GPU processors can be used to attain a good percentage of accuracy.
- 2) In future we can use this algorithm with more than one particular language.
- 3) This Model can be used in paragraph extraction if we increase the CNN layers and RNN layers and preprocess the data well.
- 4) This Model can be used in extraction of text from video if we can join CRNN and OpenCV concepts together.

References:

- [1] <https://arxiv.org/pdf/1507.05717.pdf>
- [2] <https://software.intel.com/content/www/us/en/develop/training/course-artificial-intelligence.html>
- [3] <https://software.intel.com/content/www/us/en/develop/training/course-machine-learning.html>

- [4] https://www.python-course.eu/machine_learning.php
- [5] <https://numpy.org/doc/>
- [6] <https://software.intel.com/en-us/ai/courses/deep-learning>
- [7] <https://www.tensorflow.org/tutorials/images/classification>
- [8] https://www.tensorflow.org/tutorials/text/text_classification_rnn
- [9] <https://www.tensorflow.org/tutorials/images/cnn>
- [10] <https://www.tensorflow.org/tutorials/keras/classification>
- [11] <https://www.tensorflow.org/tutorials>
- [12] <https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html>
- [13] <http://www.fki.inf.unibe.ch/databases/iam-handwriting-database/download-the-iam-handwriting-database>
- [14] <https://towardsdatascience.com/setting-up-kaggle-in-google-colab-ebb281b61463>
- [15] <http://www.fki.inf.unibe.ch/DBs/iamDB/data/words/>
- [16] <https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa>
- [17] <http://ufldl.stanford.edu/housenumbers/>
- [18] <http://yann.lecun.com/exdb/mnist/>
- [19] <https://www.coursera.org/learn/neural-networks-deep-learning/home>
- [20] <https://medium.com/@arthurflor23/handwritten-text-recognition-using-tensorflow-2-0-f4352b7afe16>

Link to Code and executable file:

https://github.com/Chinmay1009/Object-Charachter-Recognition/blob/main/OCR_Project.ipynb