

INTERNSHIP: PROJECT REPORT

Name of the Student	Mudit Yadav
Project Title	Automate identification and recognition of handwritten text from an image
Name of the Company	TCS iON
Name of the Industry Mentor	Debashish Roy
Name of the Institute	SRMIST, Kattankulathur, Chennai

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
26/04/2021	19/06/2021	125	Google Colab – Jupyter Notebook, Anaconda Navigator (Jupyter Notebook).	Kaggle Dataset platform (IAM dataset), Tensorflow 2.0, Keras – 2.3.0, Google Colab Virtual GPU or NVIDIA Tesla K80 GPU

Project Synopsis:

Title of the Project - Automate identification and recognition of handwritten text from an image.

Introduction - This is TCS iON-RIO 125-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 image-based problem most suited are convolution neural networks (CNN). To cope 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 both the problems.

To implement my project I used Kaggle dataset (IAM dataset), Google Colab Virtual GPU, Tensorflow 2.0, Keras 2.3.0, OpenCV, Numpy, Scikit library, Jupyter Notebook and python language.

We can break the implementation of CRNN network into following steps:

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

1) Setting Up kaggle –

This is an optional method to run this model. This method is only for use of GPU on Google Colab fastly. If one wants to use GPU on local machine then this step is not required. If we upload the dataset on Google Drive and use this data for training purpose it takes 462 seconds per epoch and if we upload same dataset on kaggle and used on Google Colab it takes nearly 224 seconds per epoch. It means it takes half the time as compared to Google Drive so I used kaggle to load dataset in Google Colab.

2) Collecting Dataset –

This is one of the main tasks to implement our model effectively. The features of data provided in the project guidelines match with IAM dataset. IAM dataset has 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 a 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.

3) 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. This 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.

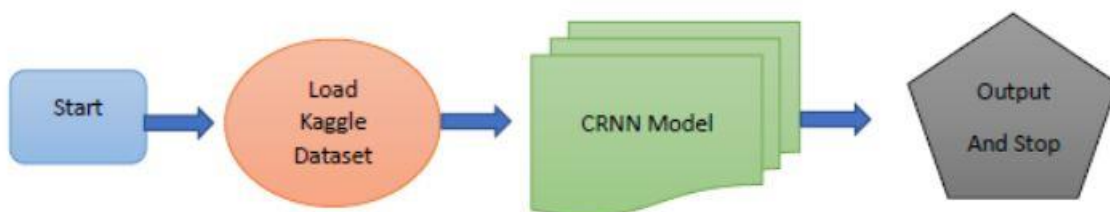
Assumptions:

The assumptions considered as follows:

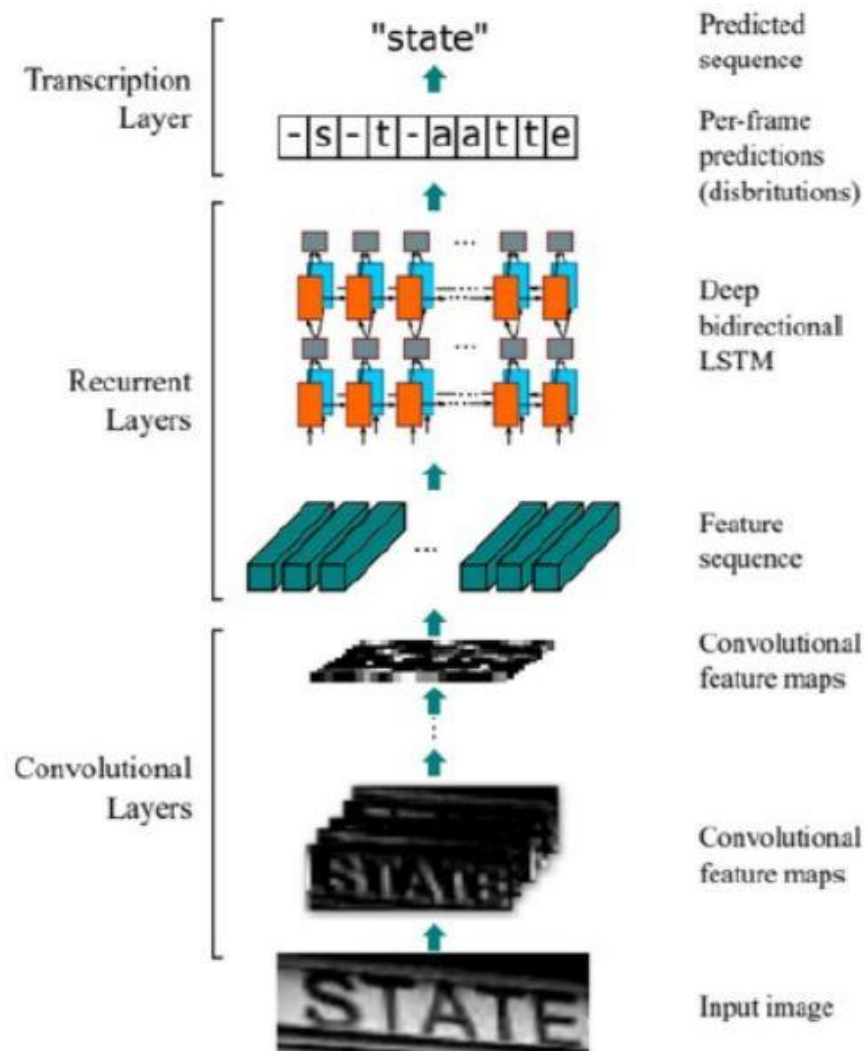
- 1) The handwritten text across the image must be in English.
- 2) The image should not be tilted.
- 3) Only image is provided for text recognition.
- 4) All machine dependencies must be installed properly

Project Diagrams:

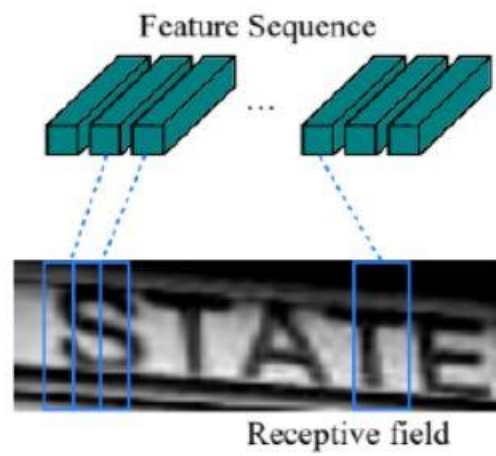
1) Block Diagram:



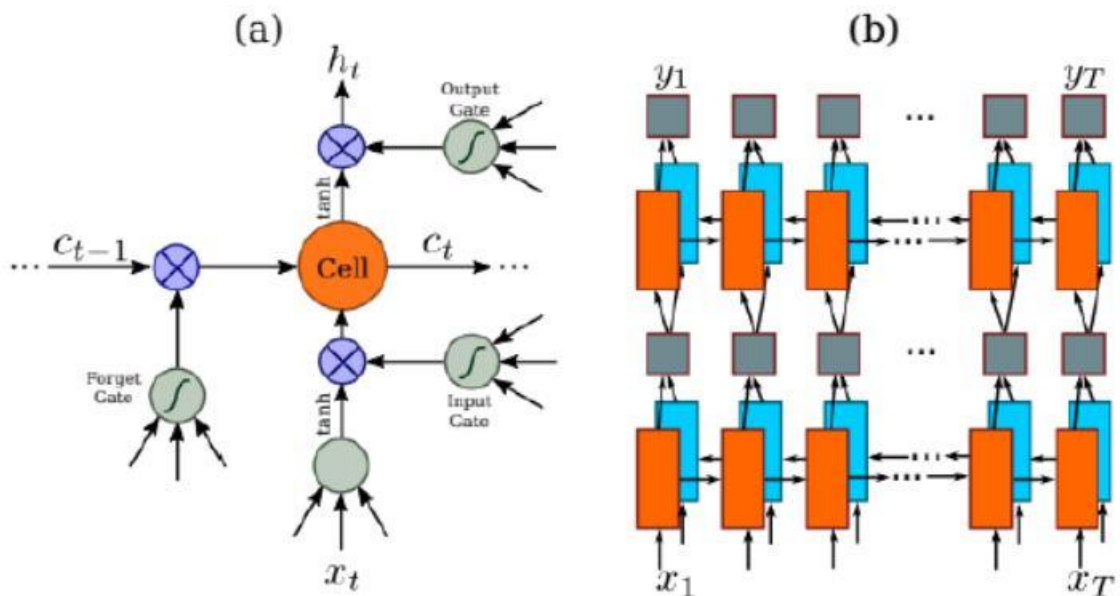
2) CRNN Model Diagram:



3) Respective Field:



4) LSTM:



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.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 32, 128, 1)	0
conv2d_1 (Conv2D)	(None, 32, 128, 64)	640
max_pooling2d_1 (MaxPooling2D)	(None, 16, 64, 64)	0
conv2d_2 (Conv2D)	(None, 16, 64, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 8, 32, 128)	0
conv2d_3 (Conv2D)	(None, 8, 32, 256)	295168
conv2d_4 (Conv2D)	(None, 8, 32, 256)	590080
max_pooling2d_3 (MaxPooling2D)	(None, 4, 32, 256)	0
conv2d_5 (Conv2D)	(None, 4, 32, 512)	1180160
batch_normalization_1 (Batch Normalization)	(None, 4, 32, 512)	2048
conv2d_6 (Conv2D)	(None, 4, 32, 512)	2359808
batch_normalization_2 (Batch Normalization)	(None, 4, 32, 512)	2048
max_pooling2d_4 (MaxPooling2D)	(None, 2, 32, 512)	0
conv2d_7 (Conv2D)	(None, 1, 31, 512)	1049088
lambda_1 (Lambda)	(None, 31, 512)	0
bidirectional_1 (Bidirectional LSTM)	(None, 31, 512)	1574912
bidirectional_2 (Bidirectional LSTM)	(None, 31, 512)	1574912
dense_1 (Dense)	(None, 31, 79)	40527
Total params: 8,743,247		
Trainable params: 8,741,199		
Non-trainable params: 2,048		

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 91.72% accurate while implementation and testing.

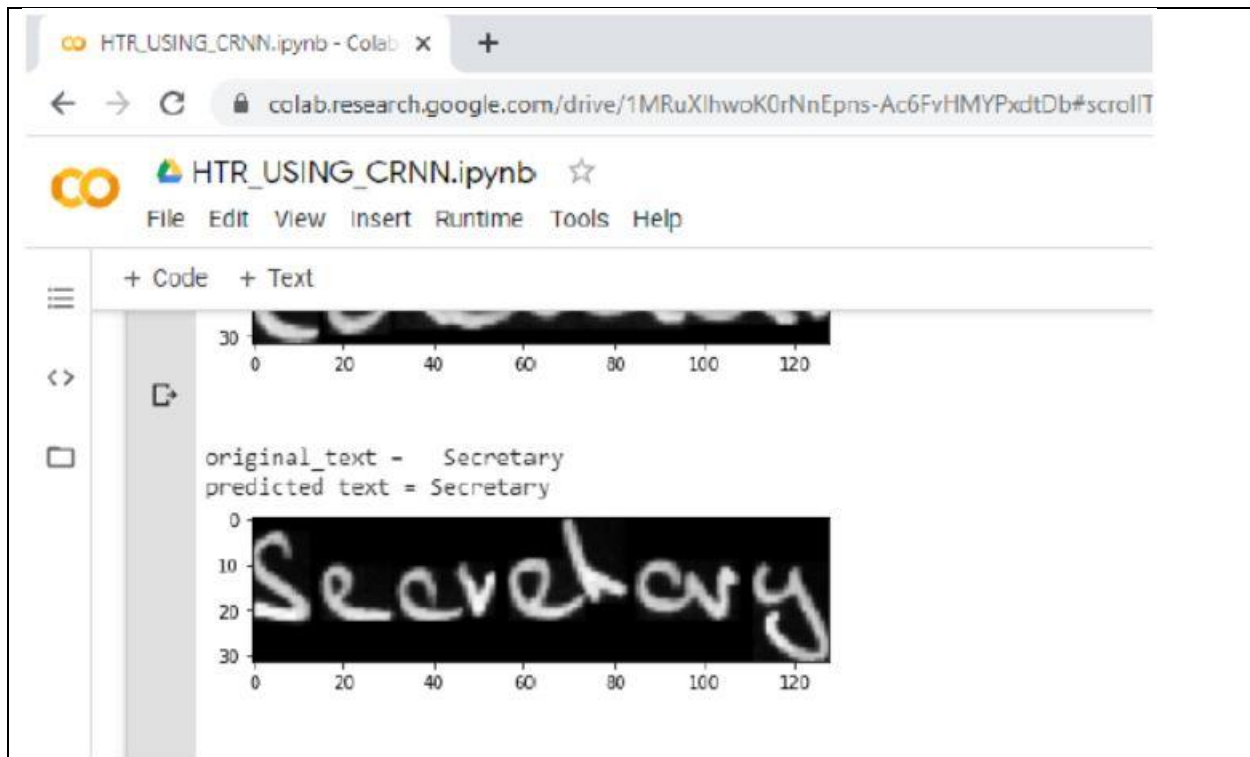
For example the input image having the handwritten text is given as following:



The Model pre-process the image removes the noise and using CRNN algorithm predict the text.

Extracted Output: Secretary

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.



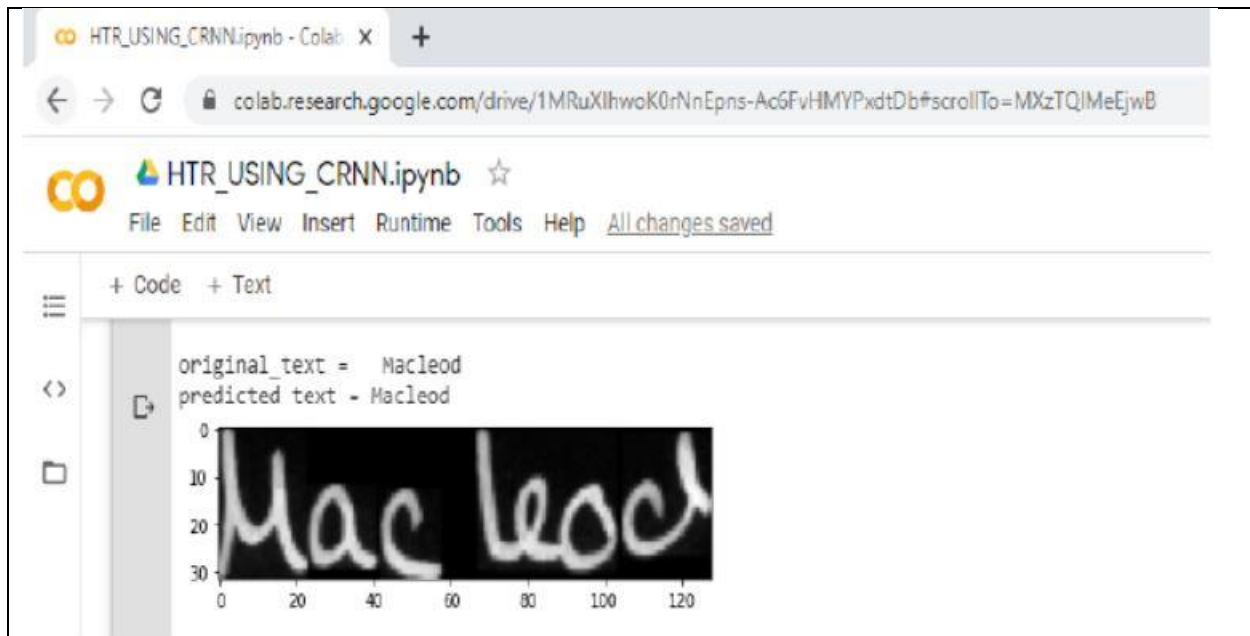
Whereas another image having the handwritten text is given as following:



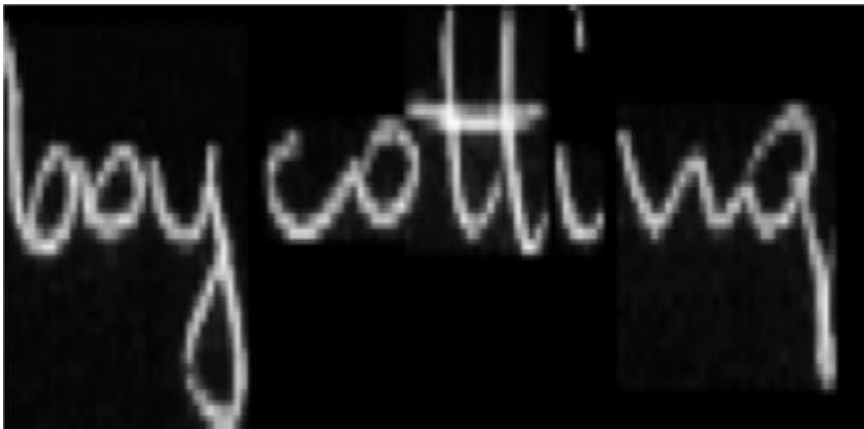
The model processes the image removes the noise from the image the convolution recurrent neural network algorithm predicts the text.

Extracted Text : **MacLeod**

As we can see the model is quite accurate and successfully able to extract the handwritten text. The model predicts and extracts the text from the image as follows:



Whereas another image having the handwritten text is given as following:



The model processes the image removes the noise from the image the convolution recurrent neural network algorithm predicts the text.

Extracted Text : **boycotting**

As we can see the model is quite accurate and successfully able to extract the handwritten text. The model predicts and extracts the text from the image as follows:



Exceptions considered:

The exceptions considered are as follows:

- 1) The text across the input image must be of the same color not multicolor handwritten text.
- 2) The image doesn't have too aggressive multicolor backgrounds across the text of the image.
- 3) The image doesn't have any kind's objects in the background across the text of the image.
- 4) The image should not be tilted or rotated.

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.

Link to Code and executable file:

<https://github.com/my23701/internship>