|  |  |
| --- | --- |
| **Name of the Student** | Aryan Saxena |
| **Internship Project Title** | Automate Identification and Recognition of Handwritten Text from an Image |
| **Name of the Company** | TCS iON |
| **Name of the Industry Mentor** | Anamika Chatterjee Ma’am and Debashis Roy Sir |
| **Name of the Institute** | Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune |
| **Name of Academic Mentor** | Prof. R.G. Khalkar Ma’am |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Start Date** | **End Date** | **Total Effort (hrs.)** | **Project Environment** | **Tools used** |
| 17th June 2020 | 31st July 2020 | 210 | Python, Deep Learning, IAM Handwriting Dataset | Google Colab, Jupyter Notebook, GitHub, Google Drive, Tensorflow, Keras |
| **Project Synopsis:**   * **Title of the Project:** Automate Identification and Recognition of Handwritten Text from an Image. * **Introduction:** The aim of the TCS iON RIO-210 internship is to develop an end to end project by automating the identification of Handwritten Texts from images. 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 CNN and RNN and develop a model accordingly. The libraries such as PyTesseract are very efficient in reading typed text from images but it lacks and performs poorly in case of Handwritten Text. So we will try to develop a model that will perform OCR of handwritten data to digital data. So, one can easily handle this digital data by editing, adding new information in that text.   * **Objective and Aim: -** To develop a model to perform OCR over Handwritten Text to enable entities for 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. We will be using Deep Learning techniques like CNN, RNN, Softmax, CTC, ReLU, etc. | | | | |
| **Solution Approach:**  First of all, I selected a Handwriting Database to work upon, from a variety of options I finally chose IAM Dataset specifically IAM Lines Dataset as the images are comparatively easy to work upon. Then I divided the data subset I selected into, train, test, validation sets. Then the images which we will be using must be in the same image dimension so we will be using Image reduction techniques.  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 CNN and RNN. So, I used Convolutional Recurrent Neural Network (CRNN) to tackle the both the problems. For the last layer, I will be trying the Softmax function.  To implement my project, I will be using IAM Handwriting Dataset (Lines) which I will be storing in Google Drive, Google Colab as my coding environment, libraries like Tensorflow 2.0, Keras 2.3.0, OpenCV, Numpy, Scikit library, Jupyter Notebook and python language.  The complete implementation of the project can be divided into the following major steps:   1. Collecting the Dataset 2. Uploading the dataset into the Google Drive. 3. Preprocessing the data. 4. Dividing the dataset into train, test and validation sets. 5. Creating the defining the model/network architecture. 6. Training the model. 7. Saving the model. 8. Testing the model. 9. Prediction. 10. Plotting the loss and accuracy plots.   **Step 1:**  This is one of the main tasks 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. I have used around 4900 images for training and testing.    **Step 2:**  Uploading the data subset into the Google Drive.    **Step 3:**  After fetching the dataset, we will preprocess the data.    **Step 4: Dividing the dataset into train, test and validation sets.**  After splitting we have 3240 train images, 835 test and validation images each.    **Step 5:**    **Step 6:**    **Step 7:**    **Step 8:**    **Step 9:** | | | | |
| **Assumptions:**  The assumptions considered are as follows:   1. The handwritten text across must be in English. 2. The text across the input image must be clearly handwritten in order to achieve good results. 3. All machine dependencies must be installed properly. | | | | |
| **Project Diagrams:**   1. Logical flow of our model:      1. Flow Chart of Process | | | | |
| **Algorithms:**  **Model = CNN (ReLU activation) + RNN + Softmax + CTC loss**  Our model consists of mainly 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 I used.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Model: "sequential\_1"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  zero\_padding2d\_1 (ZeroPaddin (None, 115, 115, 1) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv1 (Conv2D) (None, 58, 58, 32) 832  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_1 (Activation) (None, 58, 58, 32) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  pool1 (MaxPooling2D) (None, 29, 29, 32) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2 (Conv2D) (None, 29, 29, 64) 18496  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_2 (Activation) (None, 29, 29, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  pool2 (MaxPooling2D) (None, 14, 14, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv3 (Conv2D) (None, 14, 14, 128) 73856  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_3 (Activation) (None, 14, 14, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  pool3 (MaxPooling2D) (None, 7, 7, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten\_1 (Flatten) (None, 6272) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_1 (Dropout) (None, 6272) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense1 (Dense) (None, 512) 3211776  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_4 (Activation) (None, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_2 (Dropout) (None, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense2 (Dense) (None, 256) 131328  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_5 (Activation) (None, 256) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_3 (Dropout) (None, 256) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  output (Dense) (None, 50) 12850  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_6 (Activation) (None, 50) 0  =================================================================  Total params: 3,449,138  Trainable params: 3,449,138  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  None  The algorithms I used are explained as follows:  There are four layered concepts we should understand in convolutional neural networks:   1. Convolution Neural Network 2. Rectified Linear Unit 3. Pooling Layers 4. Full Connectedness (Fully Connected Layer)   **Convolution of an Image:**  Convolution has the nice property of being translational invariant. Intuitively, this means that each convolution filter represents a feature of interest (e.g pixels in letters) and the Convolutional Neural Network algorithm learns which features comprise the resulting reference (i.e. alphabet).  We have 4 steps for convolution:   * Line up the feature and the image * Multiply each image pixel by corresponding feature pixel * Add the values and find the sum * Divide the sum by the total number of pixels in the feature     The output signal strength is not dependent on where the features are located, but simply whether  the features are present. Hence, an alphabet could be sitting in different positions and the convolutional neural network algorithm would still be able to recognize it.  **Rectified Linear Unit (ReLU):** Transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.    The main aim is to remove all the negative values from the convolution. All the positive values remain the same but all the negative values get changed to zero as shown below:    Inputs from the convolution layer can be smoothened to reduce the sensitivity of the filters to noise and variations. This smoothing process is called sub sampling and can be achieved by taking averages or taking the maximum over a sample of the signal.  **Pooling Layer**: In this layer the shrink the image stack into a smaller size. Pooling is done after passing through the activation layer. We do this by implementing the following 4 steps:   * Pick a window size (usually 2 or 3) * Pick a stride (usually 2) * Walk your window across your filtered images * From each window, take the maximum value     We took window size to be 2 and we got 4 values to choose from. From those 4 values, the maximum value there is 1 so we pick 1. Also, note that we started out with a 7×7 matrix but now the same matrix after pooling came down to 4×4.  But we need to move the window across the entire image. The procedure is exactly as same as above and we need to repeat that for the entire image. Do note that this is for one filter. We need to do it for 2 other filters as well. This is done and we arrive at the following result:    Well the easy part of this process is over. Next up, we need to stack up all these layers!  **Stacking Up the Layers:**  So to get the time-frame in one picture we’re here with a 4×4 matrix from  a 7×7 matrix after passing the input through 3 layers – Convolution, Rectified Linear Unit and Pooling as shown below:    We further reduce the image from 4×4 to 2x2 to achieve this we have to perform the 3 operations in iteration after the first pass. So after the second pass we arrive at a 2×2 matrix as shown below:    The last layers in the network are fully connected, meaning that neurons of preceding layers are connected to every neuron in subsequent layers.  This mimics high level reasoning where all possible pathways from the input to output are considered.  Also, fully connected layer is the final layer where the classification actually happens. Here we take our filtered and shrieked images and put them into one single list as shown below:    So next, when we feed in, ‘X’ and ‘O’ there will be some element in the vector that will be high. Consider the image below, as you can see for ‘X’ there are different elements that are high and similarly, for ‘O’ we have different elements that are high:    Well, what did we understand from the above image is when the 1st, 4th, 5th, 10th and 11th values are high; we can classify the image as ‘x’. The concept is similar for the other alphabets as well – when certain values are arranged the way they are, they can be mapped to an actual letter or a number which we require.  **Prediction of Image Using Convolutional Neural Networks – Fully Connected Layer**  At this point in time, we’re done training the network and we can begin to predict and check the working of the classifier. Let’s check out a simple example:    We have a 12-element vector obtained after passing the input of a random letter through all the layers of our network. We make predictions based on the output data by comparing the obtained values with list of ‘x’ and ‘o’.    We just added the values we which found out as high (1st, 4th, 5th, 10th and 11th) from the vector table of X and we got the sum to be 5. We did the exact same thing with the input image and got a value of 4.56**.** When we divide the value we have a probability match to be 0.91! Let’s do the same with the vector table of ‘o’ now:    We have the output as 0.51 with this table. Well, probability being 0.51 is less than 0.91, isn’t it? So, we can conclude that the resulting input image is an ‘x’. And this is how prediction work is done. | | | | |
| **Outcome:**  The algorithm is able to detect and segment handwritten text from an image. The model successfully able to detect maximum words in a given line of sentence or words, which makes it about 90% accurate while implementation and testing. | | | | |
| **Accuracy and Loss Plots:**   1. **Accuracy:**     **2) Loss:** | | | | |
| **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:**  Several Enhancements that can be done in the project are:  1.The accuracy of the model can be 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. | | | | |
| **Link to Code and executable file:**  **Google Colab Notebook Link:**  <https://colab.research.google.com/drive/1mHrhvHTWxzcYeWF7Jvvelb-m7siq4wEY#scrollTo=lYImY9tL3u-5>  **Dataset Used Link:**  https://drive.google.com/drive/folders/1a3Yq4Dp19HyXoPVfYse30lu4TqfPCT05?usp=sharing  **GitHub Link:**  <https://github.com/aryansaxena40/tcs_ion_text_detection> | | | | |
| **References:**   1. <https://www.tensorflow.org/tutorials> 2. <https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html> 3. <http://www.fki.inf.unibe.ch/databases/iam-handwriting-database/download-the-iam-handwriting-database> 4. <https://towardsdatascience.com/setting-up-kaggle-in-google-colab-ebb281b61463> 5. <http://www.fki.inf.unibe.ch/DBs/iamDB/data/words/> 6. <https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa> 7. <http://ufldl.stanford.edu/housenumbers/> 8. <http://yann.lecun.com/exdb/mnist/> 9. <https://www.coursera.org/learn/neural-networks-deep-learning/home> | | | | |