

# CSL7360: Computer Vision

## DOCUMENT READER - IMAGE TO TEXT

### Project Report

Anurag Jain (MT19AI014)

Siddharth Yadav (MT19AI011)

## I. Introduction

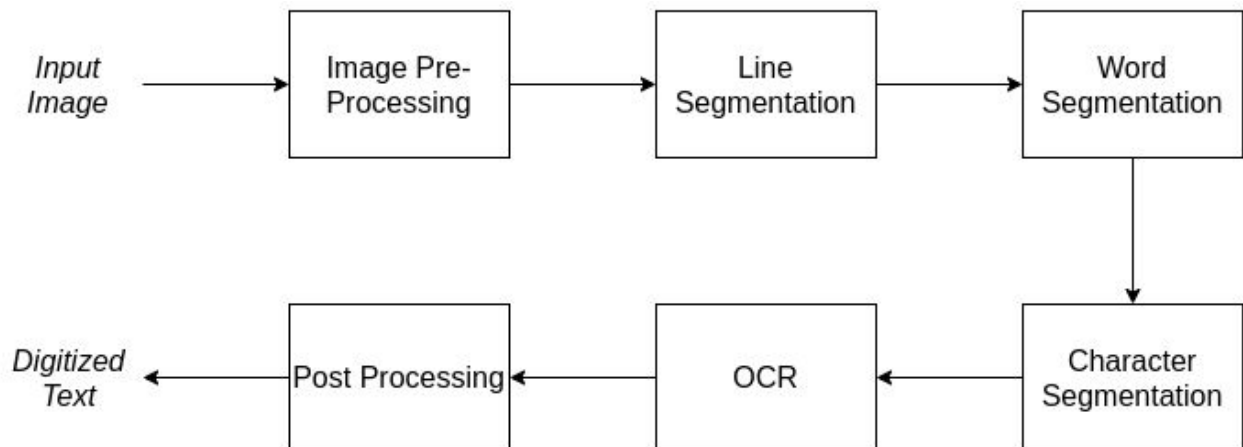
Document Reader is a well known problem in the field of computer vision and pattern recognition, where the objective is to convert the images with english text to editable text files. We are required to extract all the text from the image so that it can be used for searching purposes and also create an overlay so that text can be copied and pasted from the images saved as editable text documents. The main motivation behind this task is the huge time complexity of analyzing the hard copies of the documents, with the digitization of the documents the human effort and the time complexity will reduce exponentially. In this project, we are trying to develop such an application which can solve this problem. For this project we are going to focus only on the english text, and the documents that contain only textual information. A pipeline using basics of image processing and convolutional neural network has been developed, which is described in the further sections.

## II. Software Requirements

- **C++** programming language: For application code
- **Python** programming language with **Pytorch** library: For deep neural network training
- **OpenCv** module: For image processing
- **Linux** operating system (Ubuntu-18.04)

## III. Approach

The broad model pipeline is shown in the block diagram below:



The description of each of the algorithmic steps followed:

1. **Image Pre-processing**: This step includes a sequence of operations that are done on the input image in order to make it better for extracting the segments. These operations include normalization of the grayscale image, thresholding the image to binarize it, and then finding

the slant angle of the text within the image and then rotating the image with this angle in order to get the actual text aligned with the horizontal line.

2. **Line Segmentation:** We have experimented with two different approaches to segment the lines from the binary pre-processed (inverse-thresholded) image. In the first approach we compute the histogram of the image along the y-axis, computing the projection giving the number of pixels in the corresponding row which are greater than the threshold, and using this histogram information we extract the lines, that only the line region in the image will contain information and rest area will be black. In the second approach we dilate the image using a large horizontal structuring element, which makes the whole line text to become one segment and then we extract all the lines.
3. **Word Segmentation:** For each of the segmented line regions, we dilate the image such that the characters of the same word tend to form a single contour, then we segment all of these contours to get word segments corresponding to each line.
4. **Character Segmentation:** For each word segment, we find the contours which gives us the character segments corresponding to the word.
5. **OCR:** In OCR each of the character segments is fed to the deep neural network and the output of the network gives the predicted character value. For OCR we have trained a simple convolutional neural network using pytorch framework, whose details are provided further.
6. **Post-processing:** In this phase we concatenate the character predictions to form the words, and concatenate the words to produce the line text and finally concatenate the line text to produce the final digitized textual output.

#### **OCR training details:**

1. Initially we used a basic two layer neural network for the classification but the accuracy was not satisfactory so we decided to use LeNet architecture.
2. We modified the original LeNet architecture for improved results. In our modified LeNet architecture a 64×64 grayscale image passes through the first convolutional layer with 6 filters having size 5×5 and a stride of one. Then an average pooling layer with a filter size 2×2 and a stride of two is applied. Next, there is a second convolutional layer with 16 feature maps having size 5×5 and a stride of 1. The fourth layer is again an average pooling layer with filter size 2×2 and a stride of 2. The fifth layer is a fully connected convolutional layer with 1024 feature maps each of size 1×1. The sixth layer is a fully connected layer with 256 units. Finally, there is a fully connected softmax output layer with 62 possible values corresponding to the 62 classes.
3. The original LeNet architecture used Tanh activation function but we used ReLu because it gave better classification accuracy.
4. We ran 50 epochs for training the network. Cross Entropy loss was used as the loss function and ADAM optimizer with a learning rate of 0.001 was used for optimization purposes.
5. Top 1, Top 2, Top 3 accuracy were calculated on test and training datasets and are presented below in the table.

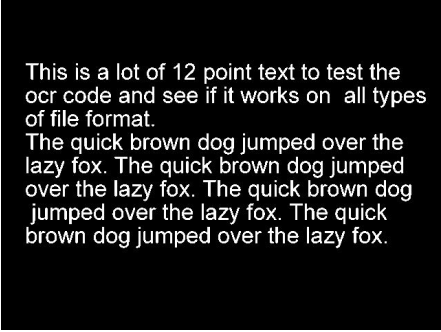
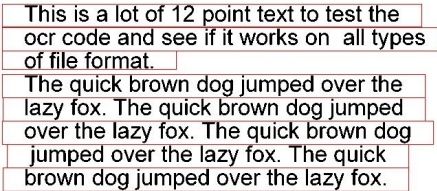
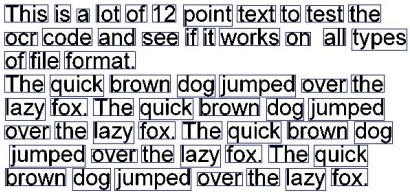
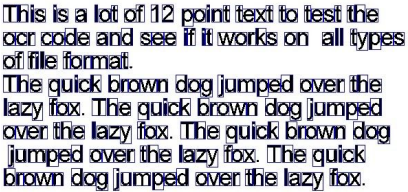
## **IV. Dataset**

For the training of OCR, we have used a 62 class dataset which consists of the digits(0-9) and alphabets(A-Z, a-z) segments from computer fonts with variations: normal, bold, italics. This dataset has been downloaded from site: [The Chars74K image dataset](#)

For the document level dataset, we have captured 30 images, containing high quality images like screenshots and HD images, and few noisy, low quality images having slant in text and dark or colourful backgrounds.

## V. Experiments and Results

The intermediate results of the application on a sample input image are shown below:

<p>This is a lot of 12 point text to test the ocr code and see if it works on all types of file format.</p> <p>The quick brown dog jumped over the lazy fox. The quick brown dog jumped over the lazy fox. The quick brown dog jumped over the lazy fox. The quick brown dog jumped over the lazy fox.</p> <p>Input image</p>	 <p>Pre Processed Image</p>	 <p>Segmented Lines</p>
 <p>Segmented Words</p>	 <p>Segmented Characters</p>	<p>Digitized Output:</p> <p>This is a lot of 1 2 point text to test the oCr Code and see lf it Works on all types of file format</p> <p>The quick boowdn doq jumped over the lazy fox The qu9ck brown doq jumped over the lazy fox The quick boowdn doq jumped over the lazy fox The qu1ck boowdn doq jumped over the lazy fox</p> <p>Digitized Output</p>

The total word level evaluation results and the training results statistics are tabulated below:

Words	Count	Percentage
Correct	1122	56.72%
1-Deviation	395	19.96%
2-Deviation	274	13.85%
3-Deviation +	187	9.45%

	Training Accuracy	Test data Accuracy
Top 1	95.55%	90.08%
Top 2	95.83%	98.80%
Top 3	100%	100%

For viewing detailed result analysis including the extraction (segmentation) results and the detection results, for each of the document image refer to the following sheet: [Cv Project: Results Evaluation](#)

## VI. Things to do further

In future in extension to the current pipeline, we can try out the following things:

1. Use of the RNN to get the word level predictions.
2. Analyse the results, and devise some approach to handle merged & splitted segments.
3. Increase the training data in order to predict special characters like comma, brackets, etc.