

Internship Documentation

Abhey Arora

May 2020

1 Introduction

This project is about building an OCR for handwritten text. At first I experimented with EMNIST dataset available on Kaggle. Got good results on character level recognition. But, text recognition will be very hard for cursive handwriting if we use a model on character level.

2 Initial approach

First we will preprocess the image. Then using line segmentation we will segment the different lines from the document using

Now that we have lines from the document we can use word segmentation to get text on word level So, basically we segment all the words separately and then try to predict it using our model.

2.1 Working on documents

The code works on both '.png' files as well as '.pdf' files. If the file is a pdf it converts it to different png files for each page using wand library and then performs the mentioned algorithm.

This required me to install ImageMagick and GhostScript because these are required for the wand package which I used to convert pdf to images.

3 Pre-processing

1. Converting image to grayscale
2. Increasing contrast of the image
3. Otsu Thresholding for binarization

4 Line segmentation

4.1 Line segmentation Algorithm 1

This algorithm calculates the row sums for the input image and then finds out its peaks and then uses a A* search algorithm to locate a path of line from left side of peak row to the right side of peak row.

The algorithm is described in detail in the paper <https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/LineSegmentation.pdf>. The code regarding this is in the file linesegm.py

4.1.1 Skew correction

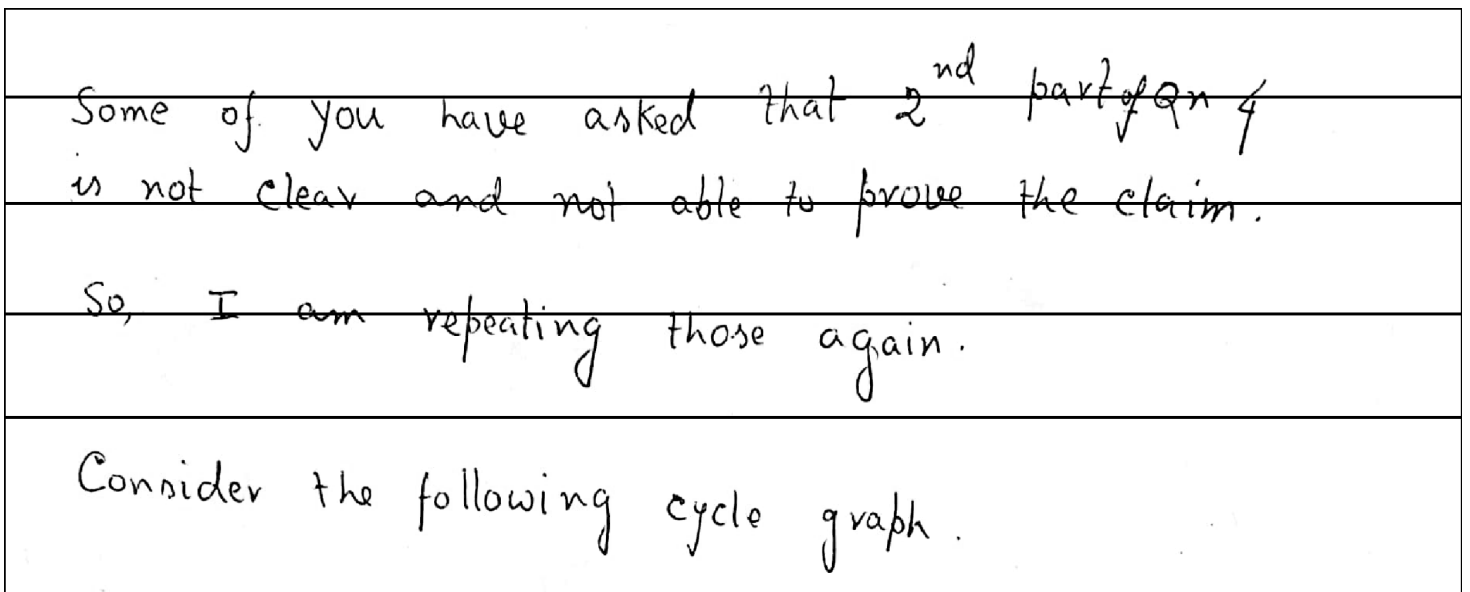
It is normal that when people write on a blank paper they don't follow straight lines exactly so there text may be a little skewed as you can see in the following image.

Some of you have asked that 2nd part of Qn 4 is not clear and not able to prove the claim.

So, I am repeating those again.

Consider the following cycle graph.

The following image shows the result if I apply line segmentation without applying skew correction.



Because of the skew the lines haven't been detected properly

The following images show line segmentation on the same image after it has been corrected for skew. The angle by which image needs to be rotated can be determined using canny lines and hough transform on the image.

Some of you have asked that 2nd part of Qn 4 is not clear and not able to prove the claim.

So, I am repeating those again.

Consider the following cycle graph.

If I implement line segmentation on this image we get:

Some of you have asked that 2nd part of Qn 4 is not clear and not able to prove the claim.

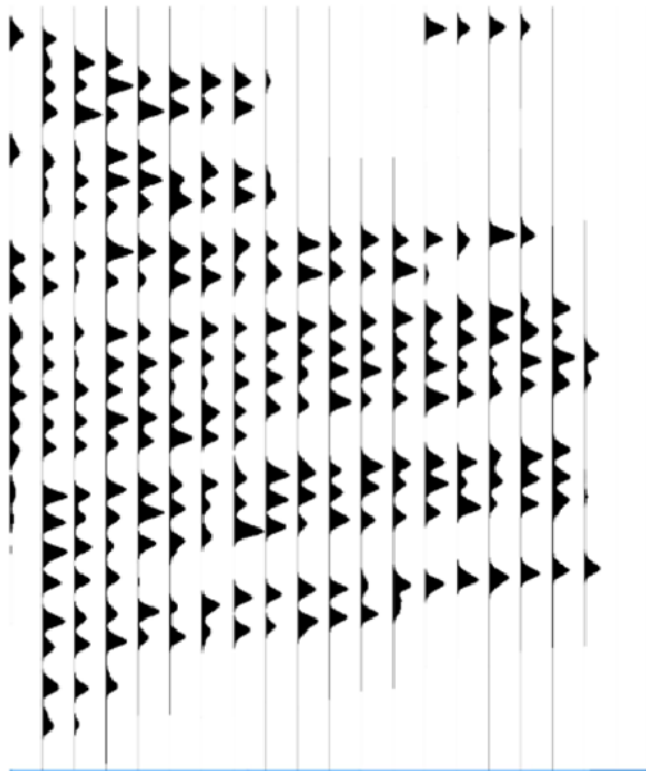
So, I am repeating those again.

Consider the following cycle graph.

We can see that the first line is still not identified correctly. This is because it was skewed in the opposite direction compared to other lines. This brings us to a disadvantage of this line segmentation algorithm because we need to be able to deal with different types (skewed to different degrees) of lines in the same image. So, we will be using a different line segmentation algorithm further.

4.2 Line segmentation algorithm 2

First we will divide the image into 20 chunks as can be seen in the image below and take smoothed projection profiles of pixels in those chunks:



Join valleys of a chunk to the closest valley of the previous chunk. Here, valleys are the lowest points that lie in between two peaks.

Some examples that show that this algorithm works well.

Place a facility such that sum of
distances from facility to all users is
minimized.

The answer is bit tricky.

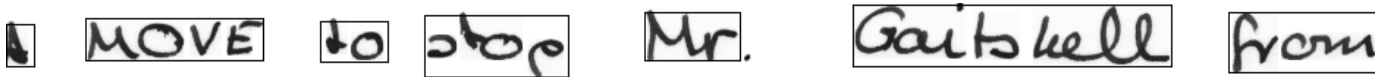
U_5 is the best choice of facility in this path as it act as median in this path.

But in cycle, from U_5 , U_9 is reachable in counter clockwise direction with distance 9 whereas in path above, U_9 is reachable at distance 10.

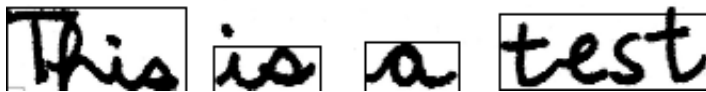
We can see that the algorithm correctly identified almost all of the lines. Two observations that can be made from the above image is that the algorithm can't deal with subscripts/superscripts and since the last line gets merged with the line above that, this algorithm works well if the lines begin from the left hand side of the page.

5 Word segmentation

Word segmentation can be done by just finding the contours in image of a line. The following is a line from IAM dataset on which word segmentation was performed



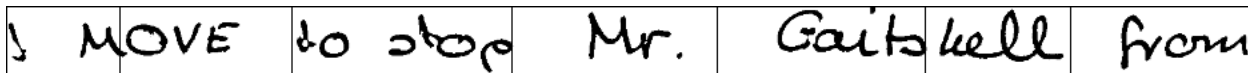
The following is a image of a line I wrote myself and took a photo.



In almost all images I experimented with the word segmentation part was good except words which are small in height and contain the letter 'i'. Because in that case the algorithm fails to include the dot of 'i' in the word. Some examples are 'in' or 'is'.

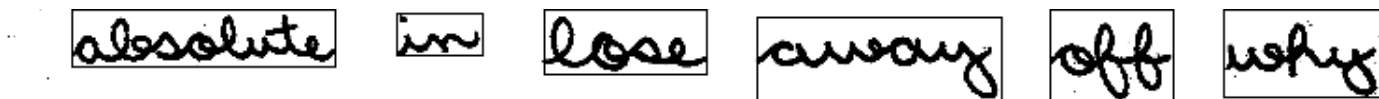
5.1 Another way to segment Words

I thought of using line segmentation algorithm horizontally for word segmentation algorithm but it failed badly because if length of a word is long and it has even a little space between two of its characters then it will detect it as a line because height of image of a line is less.



5.2 Word segmentation issue

I solved the issue with word segmentation discussed above that the algorithm does not detect the dot of letter 'i' by just increasing the height of the box containing the word by 10 pixels.



Before using this, the dot was not included in the bounding box but now we can see it is. After this the word images are resized to be an input for the neural network model

6 Word Prediction

6.1 Initial Model

First the image is resized to 128 X 32. Code for training on word level data <https://github.com/githubharald/SimpleHTR>. The model in this uses 5 CNN layers, 2 RNN layers and a CTC layer.

6.2 CNN layers

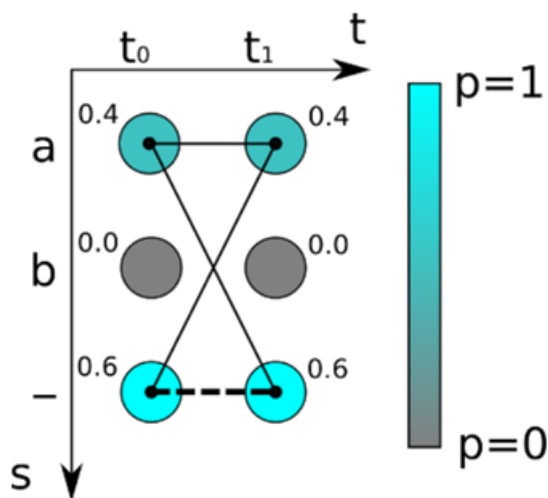
After the input is resized and fed to the model, it first goes through the convolutional neural network layers which have 5X5 and 3X3 convolutional filters along with max pooling. This is used to transform the input image to 32X256, where 256 was the number of channels in the last convolutional layer.

6.3 RNN layers

Further the model uses 2 cells of bidirectional LSTM implementation of RNN. This is used to transform the word image to the shape 32X80. This can be interpreted as probability of occurrence of the 80 characters for each time step. So, the word length has to be smaller than 32.

6.4 CTC layer

This layer is used to decode the RNN output. A word will be formed by using 32 characters, one for each time step and '-' will be used for a blank word. So, if the output is supposed to be 'abbbcc-c—..' consisting of all blanks after 'a'. This is decoded as 'abcc' because if a character repeats without a blank in between then that is ignored. Otherwise if the next character in the sequence is different then it is not ignored.



For the above image, if we consider there are only 2 timesteps, then

$$P(a) = P(" - a") + P("aa") + P("a -") = 0.64$$

$$P(-) = P("- -") = 0.36$$

During training, the layer calculates the probability of true word and tries to maximize it. Given a batch, loss function is negative of log sum of probabilities of true words.

During validation or testing, this layer decodes the path using a greedy approach by selecting the most likely character at each time step which is also known as best path decoding.

6.5 Different variations of models tried

1. Increasing CNN layers to 6 or 7.
2. Changing RMSProp optimizer to Adam optimizer. Increases accuracy but increases the time taken for training to complete.

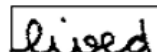
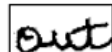
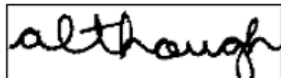
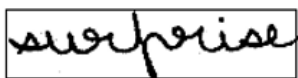

7 Performance of various models

7.1 Performance metrics

1. Character error rate: The number of errors in predicting a word is calculated by calculating the edit distance of predicted word and actual word. Character error rate is the ratio of number of errors made and total number of characters in actual words.
2. Word accuracy: The proportion of words which are predicted accurately.

7.2 Model performance

Model	Character error rate	Word accuracy
7 layer model Adam optimizer 1	23.774510%	29.729730%.
7 layer model Adam optimizer 2	24.641176%	30.324324%.
7 layer model	24.754902%	31.081081%.
6 layer model Adam optimizer	29.166667%	27.027027%
6 layer model	24.754902%	31.081081%.
5 layer model	33.333333%	14.864865%.

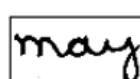
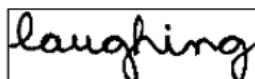
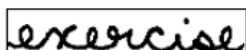
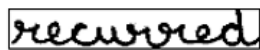
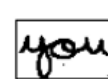
 

► Predictions:

[ERR:2] "surprise" -> "surforise" [OK] "although" -> "although"

[ERR:1] "out" -> "ouct" [ERR:1] "lived" -> "lised"

[OK] "and" -> "and"

► Predictions:

[ERR:1] "recurred" -> "recursed" [ERR:2] "exercise" -> "ercercise"

[OK] "laughing" -> "laughing" [OK] "may" -> "may"

[ERR:4] "you" -> "spore"

We can see that the highest word accuracy we are able to obtain is approximately 31%. Looking at the predictions on the test set I could see that a lot of words were pretty close to the actual ones. Like "lived" got predicted as "lised", "recurred" as "recursed". These words can be corrected using a spell checker. Using a spell checker just provides us with a accuracy boost as it mainly depends on how close the predicted word is to actual.

7.3 Model performance with spell checker

Model	Character error rate	Word accuracy
7 layer model Adam optimizer 1	18.382353%	52.702703%.
7 layer model Adam optimizer 2	18.382353%	51.351351%.
7 layer model	22.058824%	47.297297%.
6 layer model Adam optimizer	26.470588%	47.297297%
6 layer model	23.529412%	50.000000%.
5 layer model	31.862745%	31.081081%.

surprise

although

out

lived and

► Predictions:

[OK] "surprise" -> "surprise"

[OK] "although" -> "although"

[OK] "out" -> "out"

[OK] "lived" -> "lived"

[OK] "and" -> "and"

recurred

exercise

laughing

may

you

► Predictions:

[OK] "recurred" -> "recurred"

[OK] "exercise" -> "exercise"

[OK] "laughing" -> "laughing"

[OK] "may" -> "may"

[ERR:4] "you" -> "spore"

Using a spell checker improves the performance of 7 layer model with Adam optimizer to close to 53%, which was the highest I was able to achieve. The difference between 7 layer model Adam optimizer 1 and 2 is that the second one was trained for a few more epochs than the first one.

As told earlier spell checker only helps if the model predicts a word pretty close to the actual one. But here, the model predicts "you" as "spore" which is not too close and also is an actual word in the dictionary. So, spell checker doesn't help for that word at all.