

Introduction

The goal of this project is to implement working implementation of **Optical Character Recognition (OCR)** for reading images of receipts. Specifically, the goal of this OCR implementation is to take an image of a receipt, recognize each characters within each line, and create a text file that contains text written on the receipt while correctly keeping each lines and words within the line separate from each other.

There are number of reasons for picking OCR as the topic of the project. However, the definitive reason for choosing OCR was the fact that it seemed to be one of the most suitable topic for the project. OCR touches multiple topics introduced in CSC420 lectures. Majority, if not all of the OCR implementations with moderately high accuracies utilize neural networks as its main techniques. To train neural network models, OCR algorithms need to extract features from the image of each letters, and these features could be extracted through one or more feature extraction techniques from the lecture. For example, for this particular implementation of OCR, Harris Corner detection is used to get number of corners in each character image. In order to extract characters from the image of receipts, it may be necessary to preprocess the image so that images of individual characters could be extracted accurately. One of the most obvious process required would be making images of receipt as straight as possible in case image was taken with in-plane rotation. As a solution to this problem, some form of transformation is commonly used to distort the image so it becomes straight. Thus, numerous techniques used in OCR touches upon multiple topics introduced in CSC420 lectures.

Another reason that drove the idea to implement OCR was the fact that it is one of the most popular research topics in image processing using neural networks. Due to its popularity, various research institutes and companies already have provided numerous techniques for the OCR to the public (e.g., Google with Tesseract OCR and attention-OCR which is included in Tensorflow). The benefits of this is clear; there are multiple reliable sources to refer to when implementing OCR for the project.

Among these resources, there is one paper in particular that OCR algorithm for this project is closely based on. The reasoning for choosing this research paper as reference was the fact that all other research papers either touches upon highly advanced concepts or requires heavy workload that would be impossible for one person to do. While this research paper also requires high workload, majority of the workload seemed to not come from extracting features and processing images, but from training the neural networks.

Feature extraction and preprocessing techniques used in OCR implementation from the research paper is essentially same as topics from CSC420 that are being used in the project's OCR. They both use number of corners (estimated using Harris corner method), sum of pixels in four different quadrants of the image, and convex hull area as features used for the neural networks. Also, some parts of the preprocessing steps are the same; to extract each letter, binary representation of the image (where black and white have been inverted) is used. The main difference between OCR from the research paper and the project's OCR is that project's OCR comes from the fact that research paper's methods assumes very optimistic set of input images. To accommodate for more realistic cases of input images, this project OCR includes additional steps such as straighten out the each sentences in the image using affine transformation to assist in line detection.

METHODS

The methods used to implement the project's OCR closely follows the method from research paper written by Ranjan Jana and two others that was included in International Journal of Computer Applications Technology and Research¹. However, number of changes were made to make their method more applicable to the purpose of this project, which is extracting characters from a receipt. The methods used for the project could be divided into three parts. Preprocessing, Feature extraction, and Neural Network utilization.

1. Preprocessing

Preprocessing is one of the most important steps to ensure OCR's accuracy as its main goal is to allow program to take portions of the image that corresponds to each characters so they can be passed to the next step, which is feature extraction. To achieve its goals, Preprocessing is further broken down into two parts: making image as upright as possible and recognizing the presence of characters. Note that all of the functions related to preprocessing step are implemented in **ImageProcessing.py**.

Majority of preprocessing steps of available OCR implementation, including this project's OCR and the OCR implementation this project refers to, starts off by turning input images into a binary image where pixel values are inverted. For example, if figure 1 is the input image, the algorithm starts off by turning figure 1 into figure 2. This exact functionality is implemented in **getBinaryImage** function.

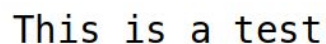
The image shows the text "This is a test" in a standard black font on a white background.

Figure 1: input image

The image shows the text "This is a test" in white font on a black background, representing a binary (inverted) image.

Figure 2: first step of preprocessing

Despite its simplicity, this step is very crucial to OCR because black pixels with zero values are used to distinguish separate lines and characters from each others. They can even be used to identify white spaces. All of this is done by simple step of summing all of the pixel values in each row and columns. If pixel value of one of the rows add up to 0, then there is either line separation or a white space. If pixel value of one of the columns add up to 0, then there is either letter separation or a space. Two functions (**extractLines** and **extractCharacter**) are associated with this task. They both follow same pattern; they check row or columns in the input image and cut portions of the image that corresponds to one line or one letter. They do this by using a variable that stores index next to a row or column that summed to 0. Idea is simple. If current row or column of the image sums to 0 and index stored by variable is not the current row or column index, then there is a line or a character starting from index stored by variable and to current index. But, if two indices are same, then this row or column is not part of a line or a character. Recognizing a space between characters is done by keeping track of how much distance exists between two characters. If there is enough distance for a normal character to fit in, then there is a white space.

There is a problem for this method of extracting lines; it heavily assumes that each lines are going to be as straight as possible which is not always true in input image. When lines are not straight, multiple lines may be detected as a single line because sum of values in rows within these lines may never be zero (e.g., figure 3). To solve this problem of lines not getting separated properly, extra step of straightening out each extracted line estimations before putting them through second round of line extraction is implemented for this project's OCR. The function responsible for the task of straightening out the image is **adjustImageRotation**. It achieves its goal by utilizing OpenCV function **cv2.minAreaRect** to

¹ <https://pdfs.semanticscholar.org/dd13/1b9105c16e16a8711faa80f232de01df02c1.pdf>

find rectangle of minimum area enclosing in the image (i.e., rectangle surrounding all of non-zero pixels). Then it calculates transformation matrix that will rotate the estimated rectangle to upright orientation. Then it applies affine transformation introduced in the lecture to warp the image using calculated transformation matrix and make lines as straight as possible. To do this, `cv2.warpAffine` is used.



Figure 3: unsuccessful line extraction as lines were not straight



Figure 4: result from `adjustImageRotation` function



Figure 5 and 6: results of second round of line extraction



Figure 7: final extraction of first letter of figure 3

2. Feature Extraction

After extracting each characters from the image, multiple features must be obtained from that image to provide identifications on which letter this image belongs to. For this project, the idea of features were heavily influenced by the same research paper mentioned earlier, but changes were made to allow this OCR to work in more general case. All of the functions responsible for feature extraction is implemented in `GetFeatures.py`, and main function that gets all features at once is `getFeatures`.

The essence of the feature extraction used for this project is dividing the image into four quadrants and getting features from each quadrant. First set of features is percentage of non-zero pixels in the quadrant. In the research paper this project is referring to, only the numbers of non-zero pixels were used, but in real life, each input images will feature characters of different sizes which means just using number of non-zero pixels as features might result in an error for a case in which character of larger font shares similar number of non-zero pixels as other character of smaller font. So instead, this project calculates the percentage of non-zero pixels in the selected dimensions as it should remain similar for same character, regardless of its font.

The second set of features is number of corners in each one of the quadrants. To obtain the number of corners, Harris corner detection method from the lecture (which utilizes both horizontal and vertical gradients along with cornerness score) is used. The research paper only calculates number of corners in the whole image, but for this project, number of corners are also calculated for each one of the quadrants for increased accuracy. The actual function responsible for this task is `getCornerCount`, and it uses `cv2.cornerHarris` to obtain set of cornerness scores for each pixels (if cornerness score in a pixel is over the threshold, which is 200 for this project, there is a corner).

The final set of features is percentage of the convex hull area in each quadrant. Convex hull is not one of the concepts introduced in the lecture, but it is useful in determining the shape of the letter. `getConvexArea` is the function that estimates this set of features. It works by finding a convex hull of non-zero pixels in given numpy matrix (that is, a smallest convex set that contains all of the points corresponding to non-zero pixels), and estimating the area it encloses. These convex hulls for each quadrants would be different for each letters because all of the letters have unique shapes. This function borrows help from `cv2.convexHull` to get convex hull from non-zero pixels and `cv2.contourArea` to get measure of area enclosed by convex hull.

3. Neural Network Utilization to Predict Characters

To implement neural networks this project heavily followed steps provided by Tensorflow's tutorial² as I am unfamiliar with technical side of the neural network implementation (my knowledge of neural network only comes from contents of this course). All of the code regarding neural networks portion and character extraction is implemented in `OCR.py`.

The first step of neural network utilization was to create a database that the model will use to train itself. To do this, same format of training model file from Tensorflow's tutorial was utilized along with features of characters extracted from train image sets. The train image sets are similar to those used by the research paper this project refers to, but training image they used are feature capital letters separated from lower case letters. To take account of the fact that sentences appearing on the receipt are combinations of lower and upper case letters, a change was made accordingly.

```
A a B b C c D d E e F f G g H h I i J j K k L l M m N n O o
P p Q q R r S s T t U u V v W w X x Y y Z z
0 1 2 3 4 5 6 7 8 9 * / + - # . , ( ) : = @
```

Figure 8: one of the training images for noto mono font featuring one of different orderings of letters used to train

The reason for using following patterns of images (as opposed to images of individual characters) as train images for this project is because it forces algorithm to apply same methods it will use for character prediction to train image sets. This makes individual features to be more useful for neural network's purpose, ultimately increasing the accuracy.

Rest of the technical processes of setting up the neural network model to training the model and making predictions of the model follow the instructions provided by Tensorflow as I did not have enough knowledges in machine learning to setup my own neural network from the scratch (which would be desired as it gives more freedom to make a model suitable for OCR). However, to give details of the model implementation, this project's neural network model contains two hidden layers with (200 nodes) that uses Rectified Linear Unit (ReLU) as their activation functions to mitigate the effects of vanishing gradient problem. It also uses cross entropy loss (which is based on the following equation: $H(p, q) = -\sum_i p_i \log q_i = -x \log y - (1-x) \log (1-y)$, p = set of ground truth label, q = set of predictions by model, x = ground truth label, and y = predicted probability) as its loss function. As for the training loop, AdamOptimizer is used as optimizer, batch size was kept moderately low (16) to ensure higher quality model, and number of epochs was set to 100 in attempt to create better model. To predict the word, the algorithm plugs list of feature measurements obtained from input image to the trained model. Once neural network word is outputted, algorithm converts it into an actual letter and puts it in a text file.

RESULTS AND DISCUSSIONS

The results of the algorithm is, in a simple word, disappointing. To test the accuracy of the OCR, two different sets of images were used; those similar to images used for referred research paper's OCR, and an image of receipt. When the images similar to those featured on the research paper this project refers to, the accuracy of this OCR was similar to the accuracy written on the paper. They both had over 70~80% accuracy for sentences written in fonts that were similar to fonts used for training but had low accuracy for sentences written in fonts that are not similar.

² https://www.tensorflow.org/tutorials/eager/custom_training_walkthrough

This is a test for the Neural Network.
Let us see if it gets over 0.8 accuracy.

Figure 9: input image with similar font as training images

This is a test for the Neural Network.
Let us see if it gets over 0.8 accuracy.

Figure 11: input image with very different font as training images accuracy

Ts1s is a tesQ fwr tse Neurql NeQwwrk.
LmN us smm if i5 ge5s over 0.8 accvracy.

Figure 10: the output from OCR with about 80% accuracy

Wbigige(eg(Sr(beSewre15e(wwrk,
kelxsseekFkl9elso(ek0,68((xk8(K

Figure 12: the output from OCR with about 20% accuracy

The reason as for why accuracy is low for input image with letters of significantly different fonts than the fonts used for letters in training image is simple; the shape is different for different types of fonts. Due to the difference in shapes, the number of pixels used in each quadrants differ, and number of corners in each characters might differ (e.g., letter “T” has extra set of corners in the second input image compared to the first input). The only aspect that might stay close among all of the fonts is are of the convex hull since general shape of the characters remain the same. Therefore, this is more of a problem stemming from the algorithm to extract features and the solution to this problem could be finding different set of features that are unaffected by aforementioned factors. There is also a problem stemming from the implementation which is absence of spaces in the result of second input. The amount of pixels provided for a space differs for each fonts, and some of the fonts give only half of the actual space given for normal character (which is the case for the second input). To solve this problem, another algorithm to detect spaces must be implemented.

Unfortunately, accuracy drops significantly when an image of a receipt is used as input.

```
Store #05666
3515 DEL MAR HTS, RD
SAN DIEGO, CA 92130
(858) 792-7040

Register #4 Transaction #571140
Cashier #56661020 8/20/17 5:45PM

wellness+ with Plenti
Plenti Card#: 31XXXXXXXXXX4553
1 G2 RETRACT BOLD BLK 2PK 1.99 T
SALE 1/1.99, Reg 1/4.69
Discount 2.70-

1 Items Subtotal 1.99
Tax .15
Total 2.14

*MASTER*
MASTER card * #XXXXXXXXXX5485
App #AA APPROVAL AUTO
Ref # 05639E
Entry Method: Chip

mQgpe wWEmmm
351t KZ( Rwl #+m,W+
G85 WNWG#, Kw m#3##
vm5+y pm2 V+4+
8@g7mQar 84 +raXm8k51gX 8G7eQw#
KaeK4ep 0mm+E1+H+ @/2#/lp Wp2GFA
ge41Xewww w4NK F1eUQ4
F4eC54 Kae2gx 3NXXCXXXXXX4GG2
l G2 #mTWgKH m+(+ mbR 2FR 1,+3 H
W8(e 5/Qc9@a W8g l/4c6@
R4egwwXQ W,Hw-
E KQe9s mwb5A5a4| g,+B
eaw ,Qm
TgQa4 2 l4
wmwmXEGw #c34
gw++Eg eay2 w 8XXXXXX454W6
6gg 8@w GPPW#Mw( 8CTK
#e7 0 GtE3BE
exQrp MaQKg2v Kh7g
```

The image on the left is image of a receipt used as input, and the image on the right is screenshot of the text file output from OCR algorithm. The results from OCR is not completely wrong, but the accuracy of the output is only about 30%; it would be hard to argue that this OCR extracts sentences from a receipt successfully. The reasoning for such result could be similar to the reasons as for why OCR had low accuracy for image of different fonts. Even though the fonts of the

training images were similar to the fonts featured in this receipt, this image of the receipt was rotated counterclockwise by around 5 degrees. While it was demonstrated that the algorithm takes account of such case can warps image so the texts can be as straight as possible, there is no guarantee that they will be perfectly upright. This will mean amount of pixels in each quadrants could be different than what is expected. Also, the resolution of the input image could affect feature extractions when calculating number of corners. For example, on a high resolution image, character “S” might show four corners at each end of the tails (as it has rectangle-esq shape at the end), but on a low resolution image it might only show two corners (as it could just appear as single dot at the end). Furthermore, wrong estimations of the characters by OCR are not all random; a lot of them share similar characteristics with characters that they were supposed to be. Therefore, low accuracy of OCR on an image of receipt mainly stems from the kinds of features extracted from the letters. Meaning, to improve the accuracy of OCR, better set of features must

be extracted. In fact, current result was an improvement made by changing from corner and convex hull feature extraction from whole image to corner and convex hull feature extraction from each one of four quadrants, and using percentage of non-zero pixels instead of number of non-zero pixels. These changes improved the accuracy by around 5~10%. Nevertheless, it is evident that this particular implementation of the OCR based on aforementioned research paper is quite not suitable for extracting texts from a receipt.

Main Challenges

There were various challenges when working on this project. One of these challenges was the fact that a lot of techniques used by many researchers are very advanced, especially when it comes to working with neural networks and getting features from images. Because the primary goal of this project is to create a working OCR, easier but less accurate approach had to be taken to make sure OCR works. Also, it was not possible to find a suitable open sourced training database when trying to increase the quality of model by increasing the number of training data. All of the open source database of the printed letters were over 50 Gb in size and training the model using these database would have required strong CPU/GPU and enough space in HDD, both of which my laptop lacked. Therefore, custom made dataset had to be made manually for the training at the cost of the quality of neural network model.

In terms of technical challenges, it was extremely difficult to increase the accuracy of OCR model when using it on image of receipts. While it was mentioned that kinds of features are the factor lowering the accuracy, another unmentioned factor was the quality of neural network. Therefore, in attempts to increase the accuracy in the model, different numbers of nodes and hidden layers, along with learning rate, number of epochs, and batch sizes were tested. However, due to lack of knowledge in optimal number of these factors in neural networks, I could not improve the overall accuracy. As an alternative solution, extra set of images were added for the training which tried to tackle different position in which a letter could be extracted. Because different letters have different vertical lengths, vertical position that a letter could show up in final extracted letter could be affected by taller letter that appeared in the same line. Doing this managed to increase the accuracy by 3%, which was not much, but still a improvement.

Conclusion and Future Work

For this project, I implemented OCR which was originally intended to work for images of receipt. This project has achieved partial success as the final product managed to get similar results as research paper this project was heavily referring to when testing images with correctly aligned images (with straight lines). However, this OCR only managed to achieve low accuracy of around 30% for its intended input, images of receipts; making this OCR unsuitable for extracting lines from receipts in most of the cases. If this project is going to be continued in the future, the tasks I have to complete is very clear. First, a new strong set of features must be extracted from the input images for both training and prediction. Current set of features are able to guide neural network model to make reasonable predictions only under certain circumstances, such as input image using similar fonts as training images and input image having letters in perfectly upright orientation. In real life scenario, this is not the case since various fonts could be used for receipts and the images of receipts could have been taken with in-plane rotation. Therefore, a stronger set of features that are unaffected by these factors must be found and utilized. Second, better neural network model that is most suitable for OCR must be created. Due to lack of technical knowledge in coding up neural networks in general, I was not able to create a custom model that is most suitable for doing tasks relating to OCR. Creating strong model would have helped the performance of OCR by a lot.