Optical Character Recognition

A Udacity Machine Learning Nanodegree Capstone Project

Definition

Project Overview

In the modern world, we maintain order through digital organization. Be it expense forecasting reimbursement requests, or discovering exactly where our personal budget went wrong, an electronic record of monetary transactions is becoming a vital part of everyday finance. Unfortunately, to sustain a comprehensive database of all relevant dealings one must manually record each entry or limit the purchases to one credit/debit card account.

The proposal being presented here is to offer an alternative solution, the creation of a program that will convert an image of a receipt to a digital replication with an accurate depiction of both the characters location and ASCII values. The technology to recognize numbers and letters has been pioneered by Yann LeCun and others in the MNIST (LeCun, Cortes, & Burges) and EMNIST datasets (Cohen, Afshar, Tapson, & Schaik). Meanwhile, the ability to locate, rotate, and crop a document from an image is widely popular in applications such as OneNote, Evernote, and Google Drive (O'Neill, 2016). By uniting these two principal functions with a blend of Decision Tree and Tensor Flow algorithms we can hope to provide a backbone to future applications such as monetary recordkeeping, collective expensive allocation, and automatic document verification.

Problem & Solution Statements and Benchmarks

Problem Statement

The problem is to convert a standard cell phone receipt image into a digital reproduction including a display of correctly located characters derived from their respective ASCII values. The project objective is to create a functioning pipeline for optical character recognition with better than random guessing accuracy. In an effort to keep the scope at a manageable level the second dataset and testing will be limited to multiple images of one receipt. This receipt contains both upper and lowercase letters, fading ink, and human-made pen marks drawn in the vertical direction. In summary, an OCR nightmare.

Solution Statement

This could potentially be solved by locating the receipt using the Canny Edge Detection function and then dividing the image into lines of text and finally character blocks. These blocks would be manipulated using techniques suggested by Yann LeCun in an effort to make them match the EMNIST dataset as closely as possible. The new set of character images would then be analyzed along with a normalized representation of their original size and location in a Decision Tree matrix. The algorithm would separate images into real, recognizable characters and dead space or unrecognizable figures before passing the first category onto a Tensor Flow algorithm. This algorithm classifies inputs as letters A to Z or digits 0 to 9. Finally, the letters or numbers would be placed on a reproduction of the original image in the same location as the block from which they were derived and displayed to the user.

Benchmarks

Current commercial Optical Character Recognition programs achieve character by character accuracy rates between 81% - 99% (Holley, 2009) on standardized printed newspapers. This means that when the network is trained on a dataset of character images it correctly classifies 81% - 99% of the validation set from that same database.

Within the OCR process, the program must locate groups of text that are of interest. This task could be completed in several ways, one of which provides a great level of accuracy is human labeling. Apps such as Receiptmate ask the user to physically highlight areas of text (O'Neill, 2016). This process is essentially a binary labeling procedure, the area in question is text or is not text. Andrej Karpathy analyzed his own binary labeling and found human accuracy to be around 94% (Karpathy, 2014). Therefore, it can be inferred that a good benchmark for locating groups of text that are of interest is 94%.

Evaluation Metrics

The vision of this project is to obtain an accuracy similar to existing benchmarks but on a poorly printed and photographed restaurant receipt. Unfortunately, state-of-the-art programs can require teams of engineers and millions of dollars in resources. To keep things in perspective we will break the evaluation down into three parts. First, the Decision Tree's binary evaluation of whether or not an image contains recognizable text. Second, the TensorFlow network's ability to correctly classify images into one of the 47 categories. Third, the overall ability of the program to convert blocks of pixels from the image to their corresponding character or blank space.

The Decision Tree's ability to select areas of interest and blank spaces is critical to the successful reproduction of the image. Because other options exist for identifying areas of text we will require the algorithm to compete with its most fearsome competitor, a human annotator. Objective studying of binary labeling found human accuracy to be around 94% (Karpathy, 2014). Therefore, 94% accuracy will be the threshold used to determine if the algorithm is performing successfully or not.

The TensorFlow network is expected to be held to the standards of current character by character recognition systems. This means that when evaluated on the test portion of the same dataset it was trained on, EMNIST, it should perform with an accuracy between 81% -99%.

The overall ability of the program to convert blocks of pixels to corresponding classifications is obviously the most interesting metric of the project and will be optimized as much as possible. For evaluation purposes, we will create a system that produces a result that is, at a minimum, better than random guessing. The 10 digits (0-9) and 52 letters (26 uppercase and 26 lowercase) will be categorized into 47 classes. When evaluating the system, one more category will be considered to accommodate the dead space. Therefore, overall classification accuracy is expected to be greater than 1/48 or 2.08%.

To calculate this number the DT algorithm and TF network accuracies will be multiplied together follow the formula and examples seen below.

Formula:

$$Accuracy_{DT} \times Accuracy_{TF} = Accuracy_{Overall}$$

Example:

$$\frac{50\ blocks\ correctly\ identified\ by\ DT}{100\ total\ blocks} \times \frac{25\ blocks\ correctly\ classified\ by\ TF}{50\ total\ blocks}$$

$$50\%_{DT} \times 50\%_{TF} = 25\%_{Overall}$$

Analysis

Datasets

EMNIST

There will be two datasets used for this project, one for each of the two separate machine learning algorithms. The task of the first dataset is to teach a TensorFlow network to recognize uppercase letters, lowercase letters, and digits that may be present in purchase receipt.

First is the EMNIST or Extended MNIST is derived from the Special 19 Database and uses the same conversion paradigm as its more well-known counterpart, MNIST. EMNIST is a collection of datasets containing handwritten digits, uppercase and lowercase letters. It is also divided into 6 different architectures, for this project we will use the By_Merge dataset. This dataset is intended to address the similarity between some uppercase and lowercase letters. If the two forms of the letter are similar enough to create confusion, their labels were merged. This results in an architecture consisting of 47 classes.

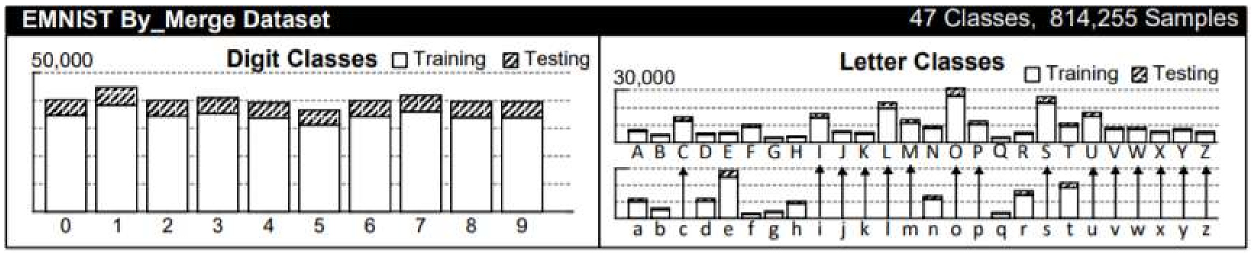


Figure 1 - Dataset Architecture

This merging of classing changes the task of the network from a 62-classification problem (By_Class) to a 47 (By_Merge) and thus eliminates 24.19% of the categories. The figures below are excerpts from the EMNIST Read_me that demonstrate the numerical results from testing the two architectures (Cohen, Afshar, Tapson, & Schaik).

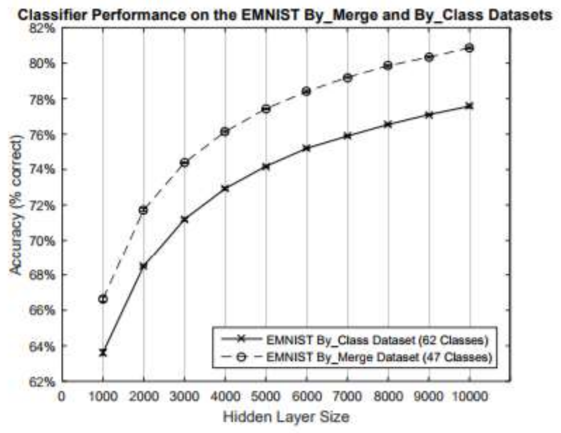


Fig. 4. **Classifier performance on the EMNIST By_Class and By_Merge Datasets.** The classification results for OPIUM-based classifiers of varying sizes applied to the two full EMNIST datasets are shown in the above figure. Ten trial of each experiment were performed. Both datasets contain all 814,255 training, with the By_Class dataset containing the full 62 classes and the By_Merge dataset merging certain letter classes to result in 47 classes. The results show that the merged 47-way classification task with the By_Merge dataset outperforms the By_Class dataset at all hidden layer sizes, although both suffer from significant variations in the letter classes accuracy from trial to trial which is overshadowed by the much larger set of samples in the digit classes.

Figure 2 - EMIST By_Class vs. By_Merge

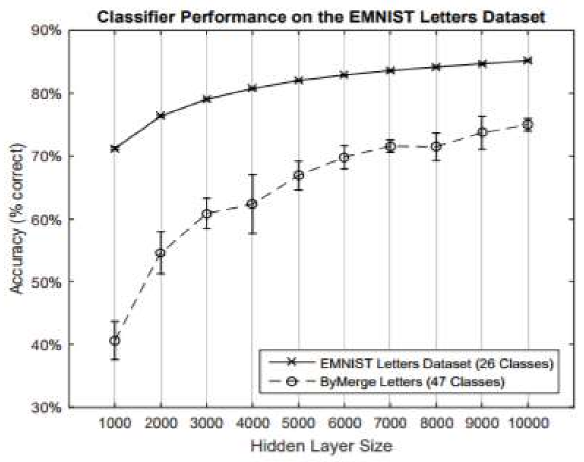


Fig. 5. **Classifier performance for the EMNIST Letter Dataset.** The classification performance achieved on the EMNIST Letters dataset over a range of hidden layer sizes is presented above. The accuracy is measured in percentage of characters correctly classified and was calculated over ten independent trials of the experiment. For comparison, the performance of the same network on the letters contained in the EMNIST By_Merge dataset is also shown. This represents the performance of the mixture of uppercase and lowercase letters found in that dataset. It is clearly visible from the above plot that the variance due to uppercase and lowercase classifications is drastically reduced with the EMNIST Letters dataset.

Figure 3 - EMNIST By_Class vs. By_Merge (Letters only)

Every entry in the dataset is a label and image from the Special 19 Database modified following Mr. LeCun example. “The original black and white (bilevel) images from NIST [a derivative of Special 19] were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain gray levels as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.” (LeCun, Cortes, & Burges)

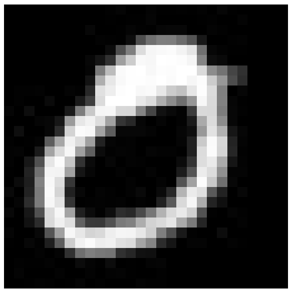


Figure 4- EMNIST image (Label: 25)

Receipt Database

The task of the second dataset is to train the algorithm to recognize which images are text and which are white spaces in the receipt or background structures in the image (dead space). To do this the dataset needs to consist of real images broken down into character size chunks and labeled in such a way that we can determine if the captured object belongs in one of our 47 categories. Currently, no dataset exists that can be easily manipulated for this task. As a result, the second dataset (receipt database), has been hand-made for this project.

Ten smartphone pictures were taken of the selected receipt and divided into character-blocks using the same process as the final image processing program. These blocks were then labeled by the human-eye resulting in 10,666 entries and 1,558 characters. The blocks and labels were saved into a *.pkl* format along with the respective height, width, horizontal location, and vertical location all normalized to the image from which they originated.

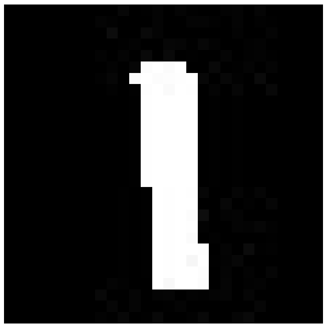


Figure 5 - Receipt Database (Label: 76)

Height (normalized)	1.2
Width (normalized)	.97
Horizontal Location (normalized)	.05
Vertical Location (normalized)	.11
Label	76

Table 1 - Receipt Database (Label: 76)

The idea behind the labeling was to retain only the blocks that were legible and classifiable after preprocessing. If the engineer could not identify the character or it was not contained in one of the 47 classes it was given a ‘32’ or dead-space identification. This logic aims at training the Decision Tree algorithm to give its TensorFlow complement the best chance possible at correctly classifying the image.

The distribution of labels is represented in the figures below. As can be seen in figure 6 the quantity of blocks that did not contain a recognizable character greatly outweighed other labels. However, even among the character-containing blocks, there is a very uneven distribution with some labels not being represented at all. This is not expected to impose an issue for the problem statement presented in this project for two reasons. First, the scope of the project is limited to one receipt analyzed through multiple images. Therefore, if a specific ASCII value is not represented in the training distribution it should also be absent in the testing set as it can be assumed that the character was not present on the receipt. Second, the decision tree algorithm is only concerned with the binary question “This is an image of a character, this is not an image of a character”, it will never see the complete value distribution. The task of training an algorithm or network to identify specific ASCII values is left to the EMNIST dataset, which contains a much more complete distribution.

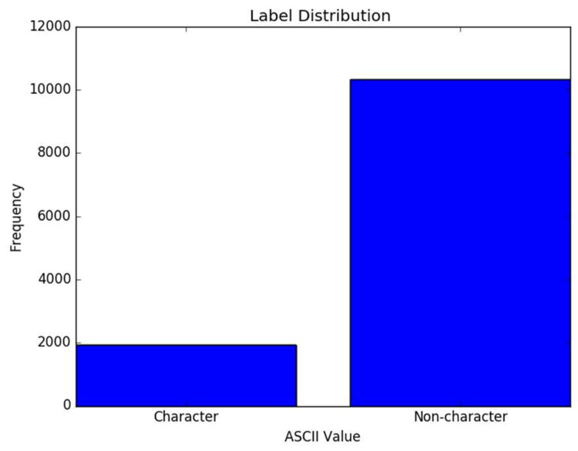


Figure 6 - Receipt Dataset Binary Distribution

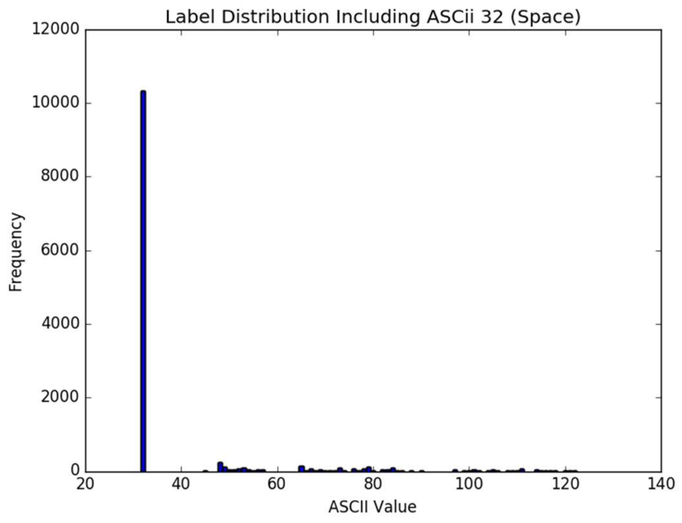


Figure 7 - Receipt Dataset ASCii Value Distribution

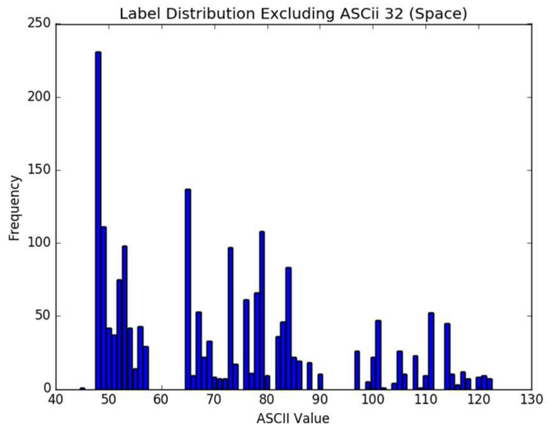


Figure 8 - Receipt Dataset Distribution (without #32)

Inputs

The inputs for the project are images of the selected restaurant receipt. They will be feed into the program as JPEGs because that is the camera-phone default. However, other formats such as BIP and PNG would also be accepted.

In testing scenarios, the same images, but in a labeled format, will be selected from the second dataset and feed into the algorithms.

The inputs for the Decision Tree algorithm will be the block image along with the normalized dimensions and locations.

The TensorFlow network requires only the block image as an input.

Algorithms and Techniques

Canny-Edge-Detector

The Canny-Edge-Detector is a popular and useful technique for extracting structural information from images. John F. Canny developed the operator in 1986 as well as produced the paper, *Computational Theory of Edge Detection* explaining why the technique works. The procedure produces images similar to the example below (Wikipedia) and proved to be useful for tasks including, locating the receipt, locating lines of text, and locating text within lines of text.

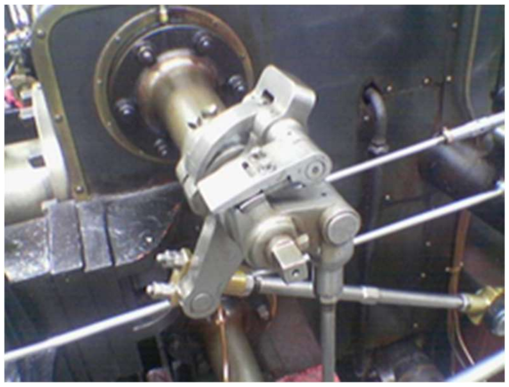


Figure 9 - Original Image

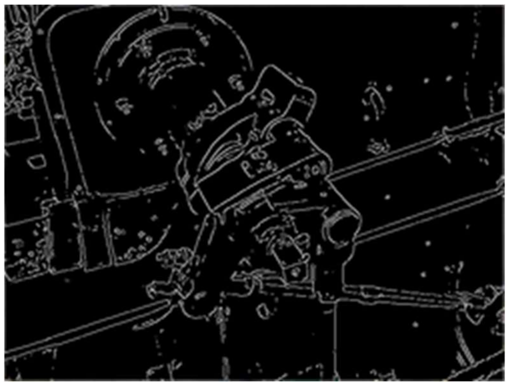


Figure 10 - The Canny edge detector applied to a color photograph of a steam engine.

Because every image will have different lighting and resolution properties the program needs to have a dynamic of selecting values for the upper and lower thresholds. Through trial and error testing done by Adrian Rosebrock, the following formulas proved to provide the desired result (Rosebrock, 2014).

$$image\_median = median(image\_pixel\_values)$$

$$Threshold1 = \max(0, 1 - \sigma) \quad x \quad image\_median$$

$$Threshold2 = \min(255, 1 + \sigma) \quad x \quad image\_median$$

LeCun Image Manipulation

For images that are not a part of the EMNIST dataset, it is necessary to process them in such a way that they match the training images. To complete this LeCun's lead was followed in all aspects except for the border. We found that size normalizing at 24x24 pixel box provided better results than its 20x20 counterpart.

An excerpt from LeCun's writings in the NIST dataset is provided below for reference.

"The original black and white (bilevel) images from NIST [a derivative of Special 19] were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field." (LeCun, Cortes, & Burges)

Bayesian Optimization

Bayesian optimization is a derivative-free optimization method that resembles human-tuning selection. Within this a Gaussian Process and Acquisition Function are combined to create a regression of possible function scores and expected quantities of improvements. Credits for the inspiration and implementation of this technique need to be given to Yurii Shevchuk and his blog *NeuPy* (Shevchuk, 2016).

The figure below provides an example of the Gaussian process' prediction of the optimum value to test next based on previous results. For every iteration past values were entered into the Bayesian optimization function, fitted to a Gaussian curve, plotted, and used to select the next 'guess' of the program. In this specific case, the optimum number of outputs for the first convolutional network was being determined based on its score (accuracy minus training time).

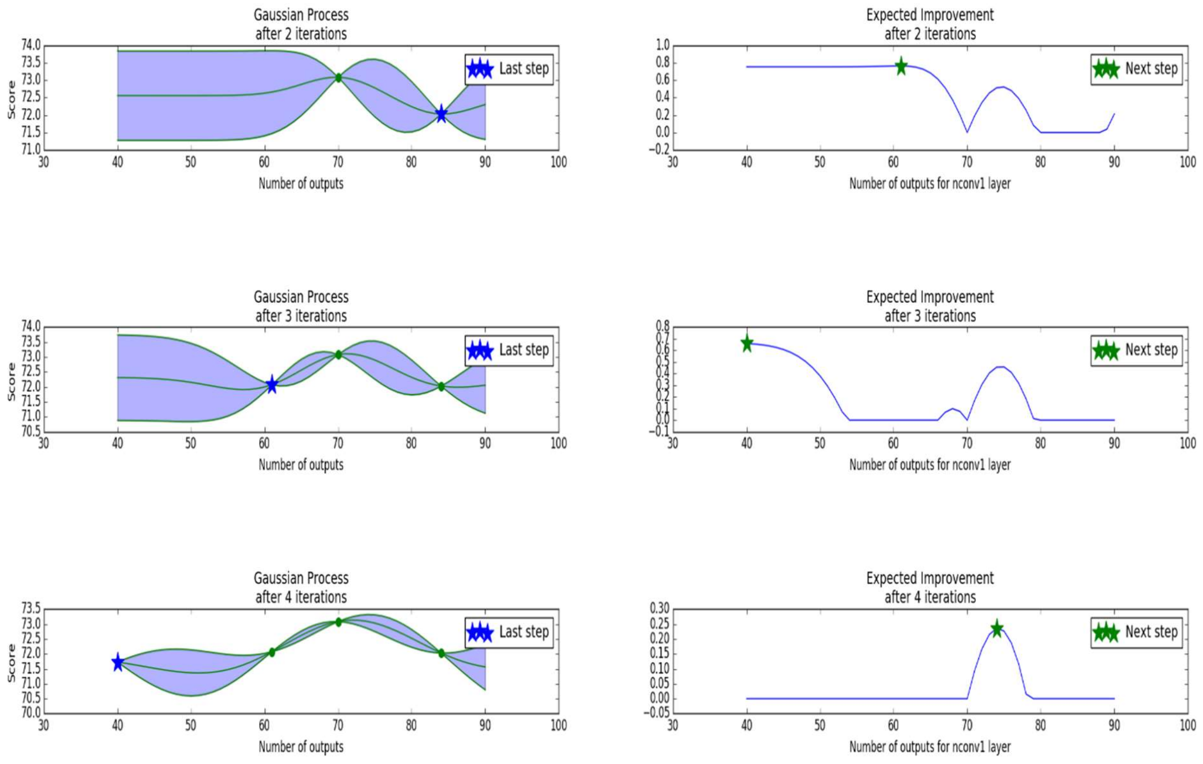


Figure 11 - Bayesian Optimization of Convolutional Layer Outputs

Decision Tree Classifier

A decision tree classifier is a method of analyzing past behaviors and using them to make predictions. The model creates a *tree* that consists of *yes/no branches,* which ultimately lead to a result *leaf,* that is outputted.

In the case of this project, several machine learning algorithms were tested for training time, prediction time, accuracy when asked to identify blocks containing recognizable characters. Of the group, the decision tree classifier performed the best producing both accurate results and speedy processing times.

Entries from the receipt database were used to train this algorithm. The labels of every entry were modified from an ASCII value to a 0, indicating a character or 1, indicating a dead space.

Hyperparameter selection was based on an extensive search utilizing a combination of a Bayesian optimization process and grid search. The optimized parameters include max_depth, criterion, splitter, min_samples_split, and min_samples_leaf.

TensorFlow Network

TensorFlow is an opensource library used to create neural networks that complete tasks that mimic human logic and reasoning. For this project, it was used to create a convolutional network consisting of convolution, max pooling, normalization, and fully connected layers.

Before training images from the EMNIST dataset were normalized and mirrored across the vertical direction. The normalization was done to improve the effectiveness of learning, while the mirroring was to ensure the character was recognizable to the human eye.

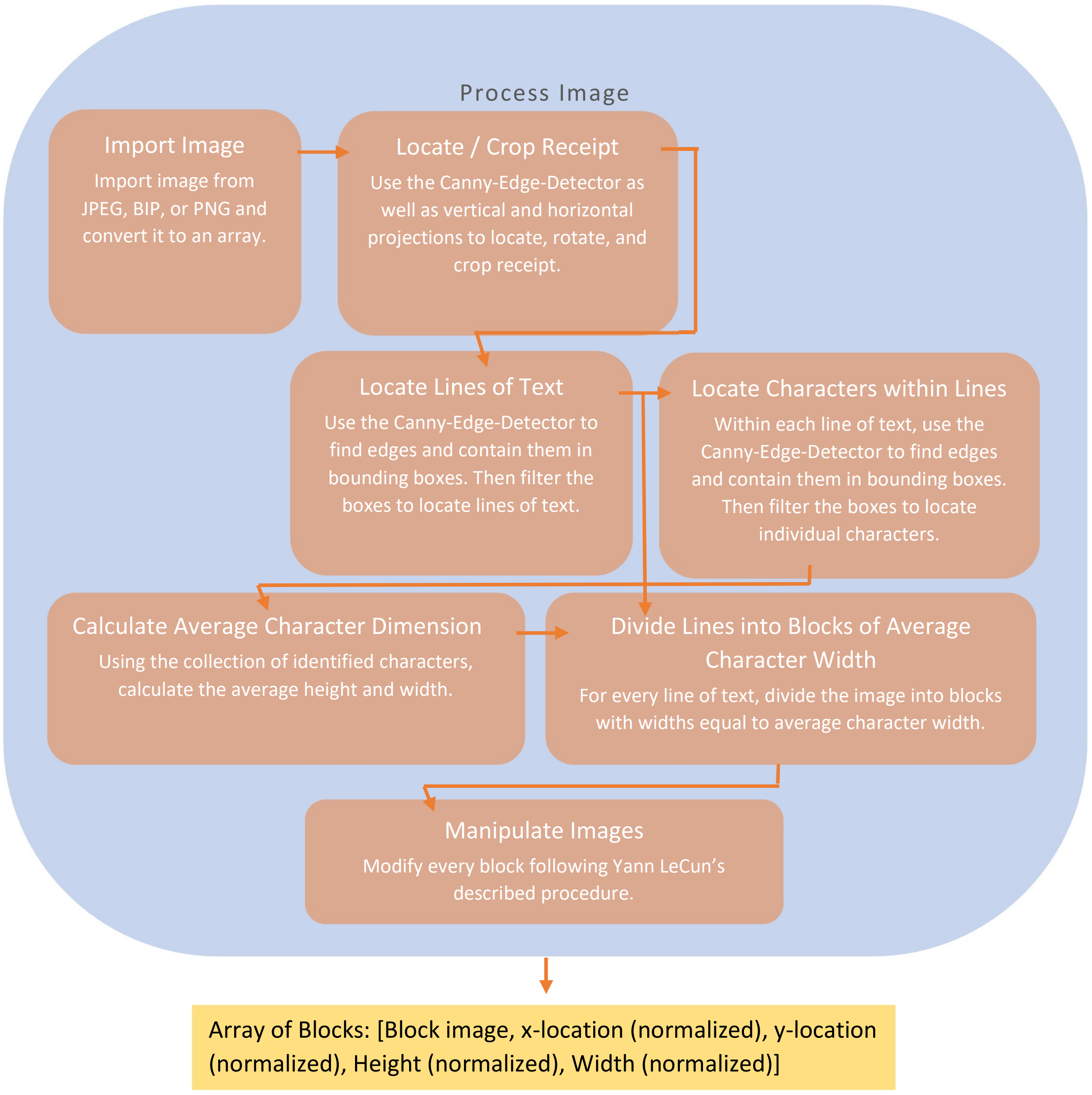
Before testing ASCII labels from the receipt dataset were converted to the 0-46 class system used by the EMNIST dataset. If a label could not be contained in one of the 47 categories it was not used in testing.

The selection of layer outputs was determined using a combination of a grid search and Bayesian optimization process.

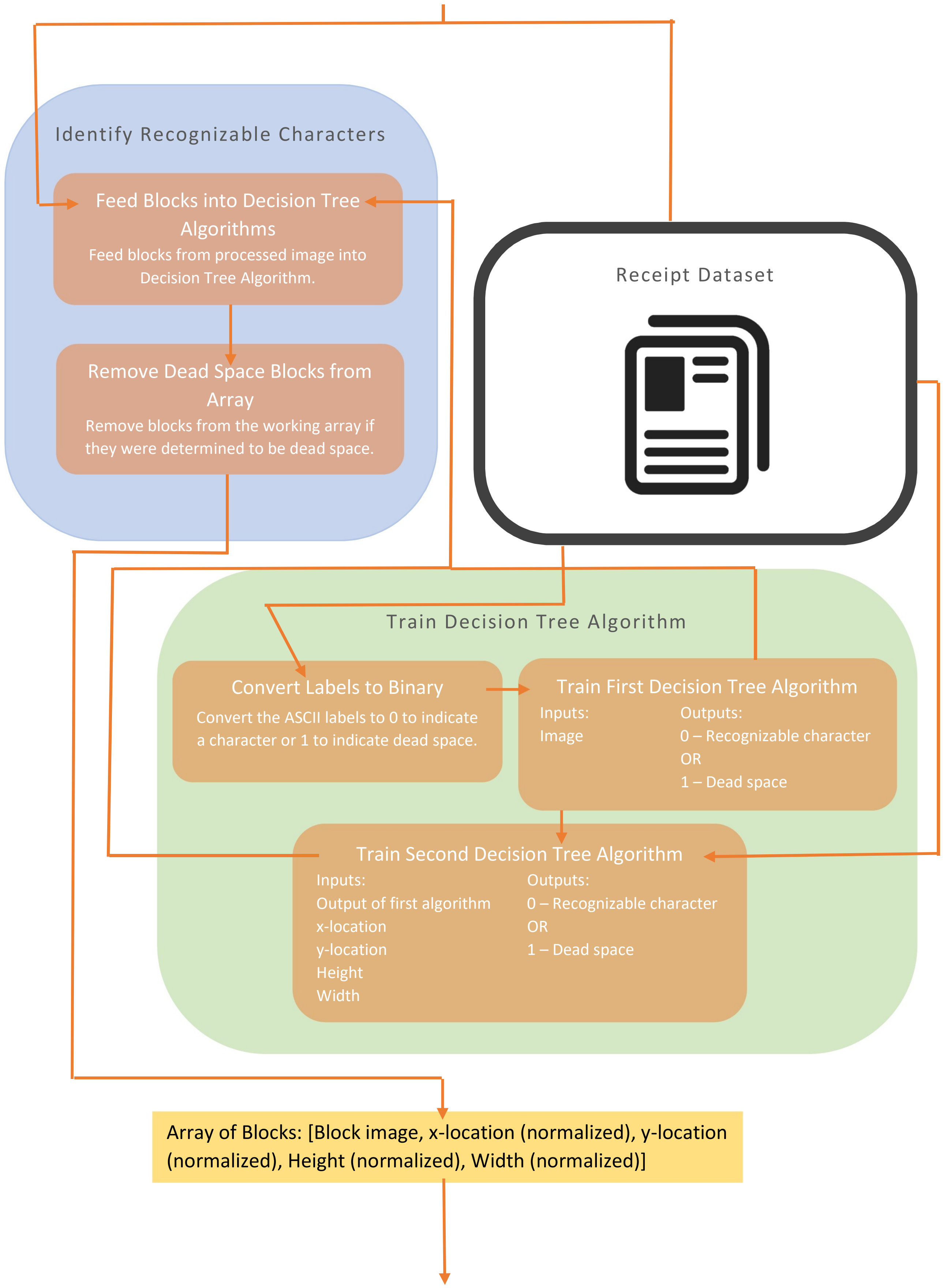
Methodology

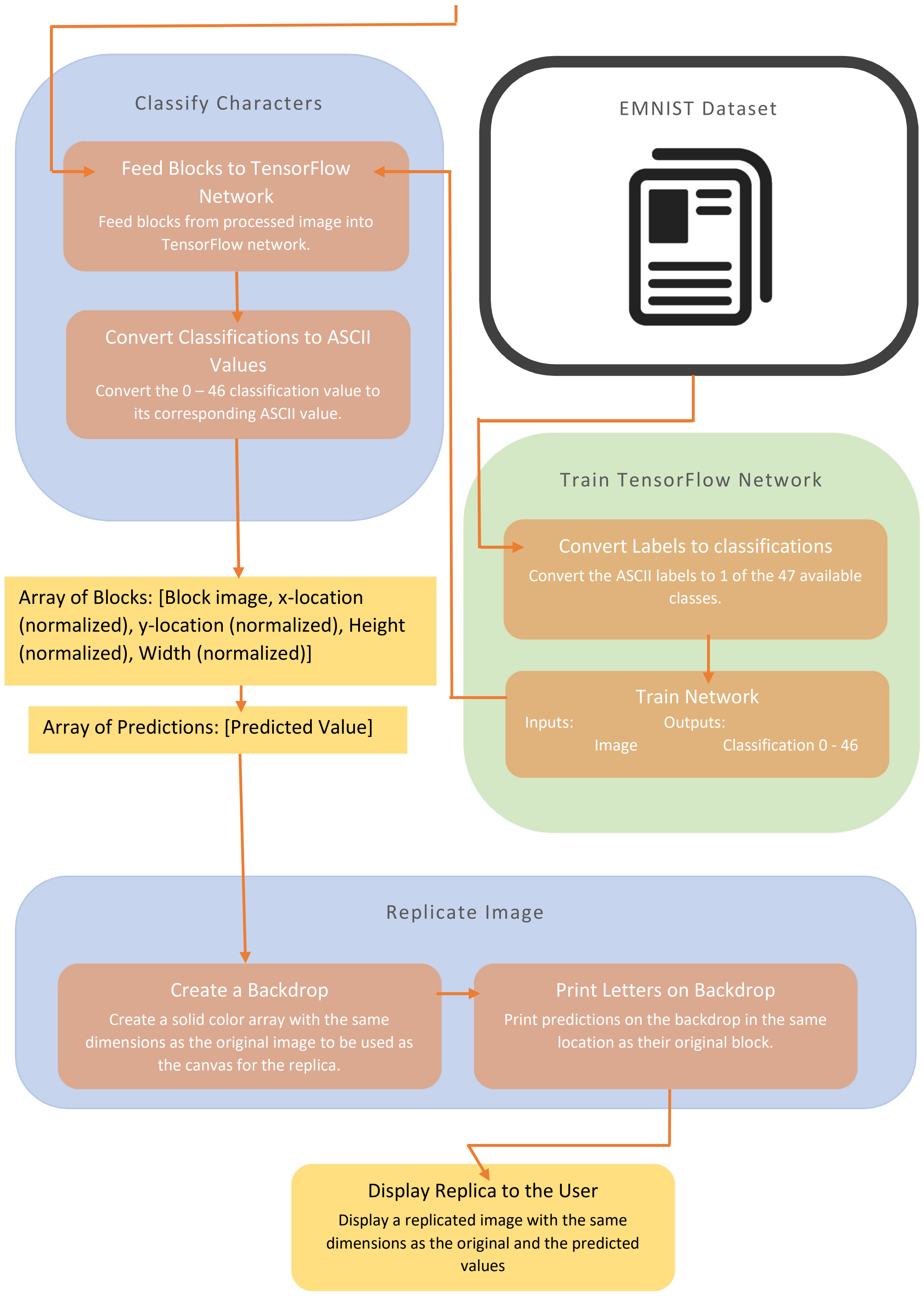
Project Overview

The design for this project will be a single direction pipeline that processes the raw image and outputs a digital replica.



Process Image
Import Image
Import image from JPEG, BIP, or PNG and convert it to an array.
Locate / Crop Receipt
Use the Canny-Edge-Detector as well as vertical and horizontal projections to locate, rotate, and crop receipt.
Locate Lines of Text
Use the Canny-Edge-Detector to find edges and contain them in bounding boxes. Then filter the boxes to locate lines of text.
Locate Characters within Lines
Within each line of text, use the Canny-Edge-Detector to find edges and contain them in bounding boxes. Then filter the boxes to locate individual characters.
Calculate Average Character Dimension
Using the collection of identified characters, calculate the average height and width.
Divide Lines into Blocks of Average Character Width
For every line of text, divide the image into blocks with widths equal to average character width.
Manipulate Images
Modify every block following Yann LeCun's described procedure.
Array of Blocks: [Block image, x-location (normalized), y-location (normalized), Height (normalized), Width (normalized)]


Identify Recognizable Characters
Feed Blocks into Decision Tree Algorithms
Feed blocks from processed image into Decision Tree Algorithm.
Remove Dead Space Blocks from Array
Remove blocks from the working array if they were determined to be dead space.
Receipt Dataset
Train Decision Tree Algorithm
Convert Labels to Binary
Convert the ASCII labels to 0 to indicate a character or 1 to indicate dead space.
Train First Decision Tree Algorithm
Inputs:
Image
Outputs:
0 – Recognizable character
OR
1 – Dead space
Train Second Decision Tree Algorithm
Inputs:
Output of first algorithm
x-location
y-location
Height
Width
Outputs:
0 – Recognizable character
OR
1 – Dead space
Array of Blocks: [Block image, x-location (normalized), y-location (normalized), Height (normalized), Width (normalized)]



Classify Characters
Feed Blocks to TensorFlow Network
Feed blocks from processed image into TensorFlow network.
Convert Classifications to ASCII Values
Convert the 0 – 46 classification value to its corresponding ASCII value.
EMNIST Dataset
Train TensorFlow Network
Convert Labels to classifications
Convert the ASCII labels to 1 of the 47 available classes.
Train Network
Inputs: Image
Outputs: Classification 0 - 46
Array of Blocks: [Block image, x-location (normalized), y-location (normalized), Height (normalized), Width (normalized)]
Array of Predictions: [Predicted Value]
Replicate Image
Create a Backdrop
Create a solid color array with the same dimensions as the original image to be used as the canvas for the replica.
Print Letters on Backdrop
Print predictions on the backdrop in the same location as their original block.
Display Replica to the User
Display a replicated image with the same dimensions as the original and the predicted values

Image Processing

Implementation

The implementation of the image pre-processing was largely inspired by two sources. First, Roland Szabo's writing in *A Novel Machine Learning Based Approach for Retrieving Information from Receipt Images* (Szabo) and second Ozhiganov Ivan's *Applying OCR Technology for Receipt Recognition* (Ivan, 2016). The process begins by loading the image and then slowing rotating it 20 degrees in either direction in small 0.3 degrees increments. With each step, a Canny-Edge filter is applied, the horizontal projection is taken and the standard deviation calculated. The angle where the text is horizontally aligned should correspond with the highest standard deviation or separation of white horizontal lines and black horizontal lines. After this angle is determined, the vertical and horizontal projections are analyzed for spikes indicating the edges of the receipt. The image is then cropped at these points and passed on to the next step.

The cv2 findContours function and subsequently approxPolyDP function are used to create boxes around text. These boxes are then passed through a series of filters that combine and eliminate entries until only complete lines of text remain. Next, a similar procedure is followed for each line of text, but this time breaking it up into individual characters. Finally, these identified characters pass through a counter and are used to calculate the average height and width of the text in this image.

Now that the program has a better idea of the font size it will go back and break each individual line of text down into characters or blocks using a window width equal to the average text width. It starts on the left side of the image and slowly slides the window mentioned above to the right until it finds a location where it is not intruding on a box created by the findContours/approxPolyDP functions. It then claims this section of the image as a block continues stepping rightward until it can safely identify another. This helps to recover text that was not well defined by the findContours function and would have otherwise been left out.

The figure below shows an example of a line broken down utilizing the estimated font size. As you can see the area between the 'o' and 'r' in 'Hora' did not separate correctly. This was due largely in part to the angle with which they appear. The issue also manifested if the font size was larger than the average text with, for example, the title on the receipt might be written in bigger text than the items purchased. To compensate for such issues, any block that is more than double the window width passes through the findContours/approxPolyDP functions for the third time. This pass is usually able to identify characters in the blocks breaks the section of the line down accordingly.

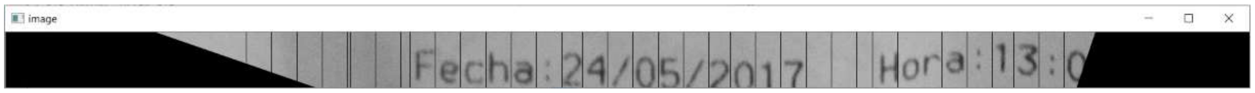


Figure 12 - The Division of a Line of Text

After breaking the image down into blocks the final step of the processing section is to manipulate the block pixels to match the 28x28 style used by the TensorFlow network. This process starts converting the pixels to binary and then resizing it to 24x24. Afterwards, it is placed on a 28x28 canvas and centered based on the center of mass.

Refinement
The image pre-processing started out by only containing findContours and approxPolyDP functions. However, it quickly became obvious that alone these would not yield the accuracy necessary to complete the project. Thus, the process was then broken down into two parts. First, filtering the boxes into lines of text and second, dividing those lines into characters. The number of correctly identified characters improved but several were still being left out because the grayness and blur in the image prevented the Canny-Edge-Detector and by consequence the findContours function from detecting them. To overcome this the font size was determined by calculating the average height/width of the text and used as a reference for splitting of the lines of text into blocks. This final combination of filters and font assumption proved to yield an acceptable accuracy.

Identifying Recognizable Characters

Implementation
A vital part of the pipeline is to identify which parts of the image are important and which are not. This is accomplished by passing every block through a series of decision tree algorithms. The first algorithm analyzes only the image. It then outputs an engineered feature of character – 0 or dead space – 1. This feature is combined with the normalized height, width, vertical location, and horizontal location and delivered to a second decision tree algorithm. The final binary decision of this algorithm (character or dead space) is used to decide if the block will be passed on to be classified by the TensorFlow network or if it will be removed from the pipeline.

Refinement
The original plan contained only one algorithm to complete this task. However, it was found that by analyzing the image alone the maximum accuracy attainable was around 80%. The second algorithm was added to incorporate more physical information like the original size and location of the block as well as the engineered featured we had created in the first algorithm. Using these two decision trees together proved to be very effective and consistently demonstrates an accuracy of 99% or better.

Classifying Characters

Implementation
To classify images a TensorFlow network containing convolutional, max-pooling, normalizing, and fully-connected layers is utilized. This network inputs 28 by 28-pixel images and outputs and 0-46 classification. The classification is then mapped to its corresponding ASCII value and delivered to the image replication part of the program.

Refinement
This network began by using the 68 categories in the By_Class architecture and a small three-layer network. However, when tested on the EMNIST testing set it only produced results between 60 – 70%. The addition of layer complexity and the use of the By_Merge architecture (47 categories) greatly improved the accuracy to ~85%.

Replicating Image

Implementation
The final deliverable of the program is to display a black and white image to the user that mimics the information and layout of the original receipt. To complete this task, first, a black canvas with the same dimensions as the image is created. Then, each block that passed through the TensorFlow network is

located on the canvas and its respective character is printed with a font size that fills the extent of the block. Finally, side-by-side images of the cropped receipt and the replica are presented to the user.

Refinement

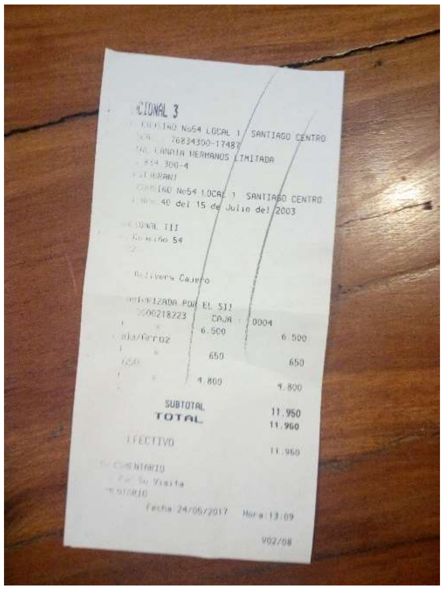
When the replication process was first implemented all of the characters were printed with the same font size. This gave some images a very distorted feel as the text would be far too big or small for the canvas size. The program was later refined to make the font size variable based on the size of the image block that was used in the classification process.

Results

Model Evaluation and Validation

The evaluation of this model was done on three images from the receipt dataset (30% of the dataset). The cumulative accuracy for the Decision Tree selection process was 99.8%. The cumulative accuracy for the TensorFlow network was 44.7%. This gives the overall program accuracy 44.6% which is not as good as state-of-the-art models but significantly beats our benchmark of random guessing.

Below are the three images with their corresponding results.



CIONAL 3
[illegible] INO No54 LOCAL 1 SANTIAGO CENTRO
[illegible] 76834300-17487
[illegible] CANAIA HERMANOS LIMITADA
[illegible] 300-4
[illegible]URANT
[illegible] INO No54 LOCAL 1 SANTIAGO CENTRO
[illegible] 40 del 15 de Julio del 2003
[illegible]CIONAL III
[illegible] 54
[illegible] Cajero
[illegible]IZADA POR EL SII
[illegible]00218223 CAJA : 0004
6.500
[illegible]/Arroz 6.500
650
650
4.800
4.800
SUBTOTAL 11.950
TOTAL 11.960
EFECTIVO 11.960
[illegible]ENTARIO
[illegible] Su Visita
[illegible]
Fecha 24/05/2017 Hora 13:09
V02/08

Figure 13 - Original Image

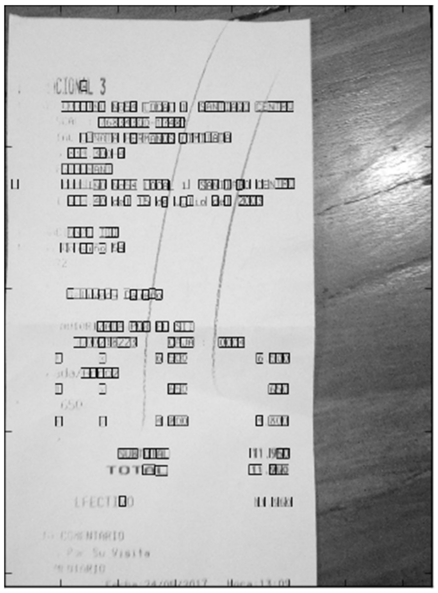
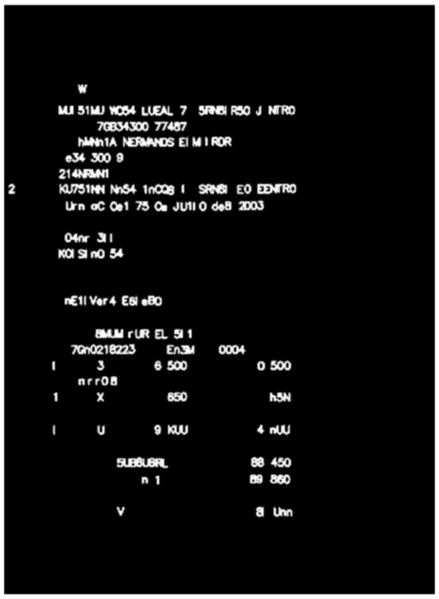
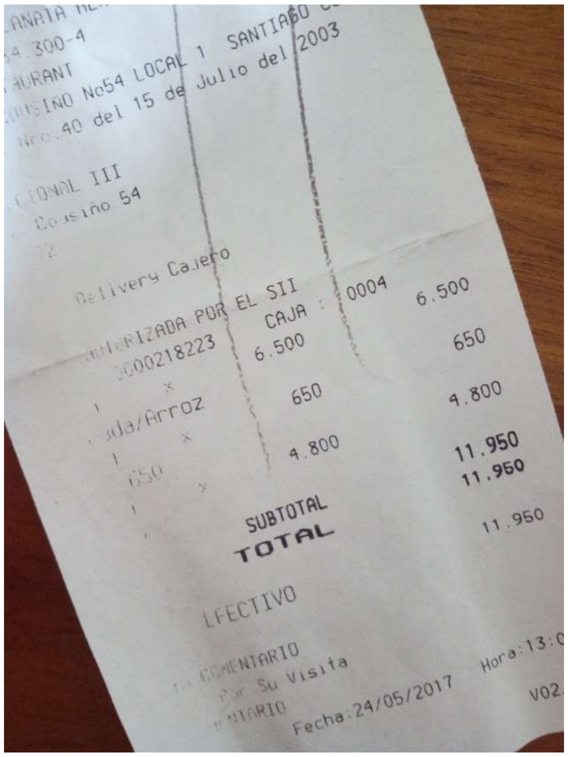


Figure 14 - Located Characters



```
           W
      MJI 51MJ WC54 LUEAL  7   5RNBI R50  J  NITRO
             7GB34300 77487
         hMNn1A  NERMANDS EI M I ROR
       e34  300  9
      214NRMN1
2     KU751NN NnS4  1nCQ8  I    SRNBI   E0  EENTRO
       Urn  oC  Oe1  75  Oe  JU1I 0  de8  2003

        04nr  3I I
      KOI SI nO 54

        nE1I Ver4  E6I eB0

              BMUM rUR EL  5I 1
          7Gn0218223        En3M      0004
     I          3             6  500              0  500
          n r r 0 8
     1          X               650                   h5N

     I          U             9  KUU              4  nUU

                  5UB8UBRL                 88  450
                        n  1               B9  860

                   V                          8I  Unn
```

Figure 15 - Image Replication



ANAIA
4 300-4
URANT
USIÑO No54 LOCAL 1 SANTIAGO
.40 del 15 de Julio del 2003

IONAL III
Cousiño 54

Delivery Cajero

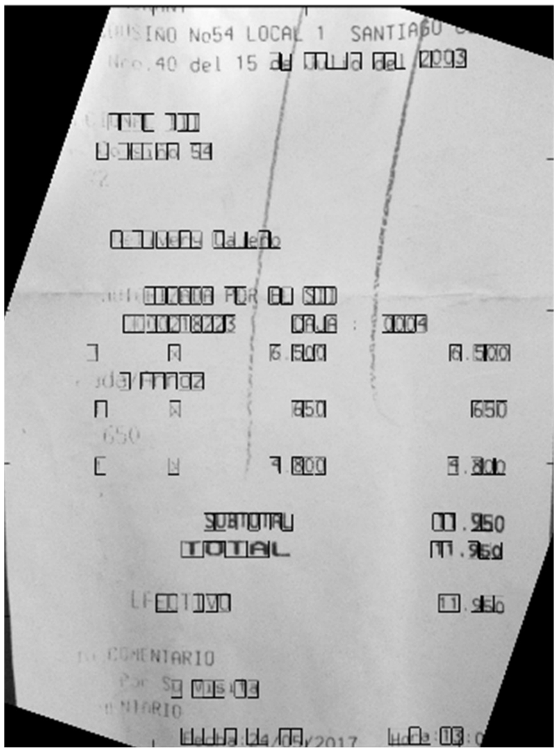
RIZADA POR EL SII
000218223 CAJA : 0004
1 x 6.500 6.500
da/Arroz
1 x 650 650
650
1 x 4.800 4.800

SUBTOTAL 11.950
TOTAL 11.950

EFECTIVO 11.950

COMENTARIO
Por Su Visita
NTARIO
Fecha:24/05/2017 Hora:13:
V02

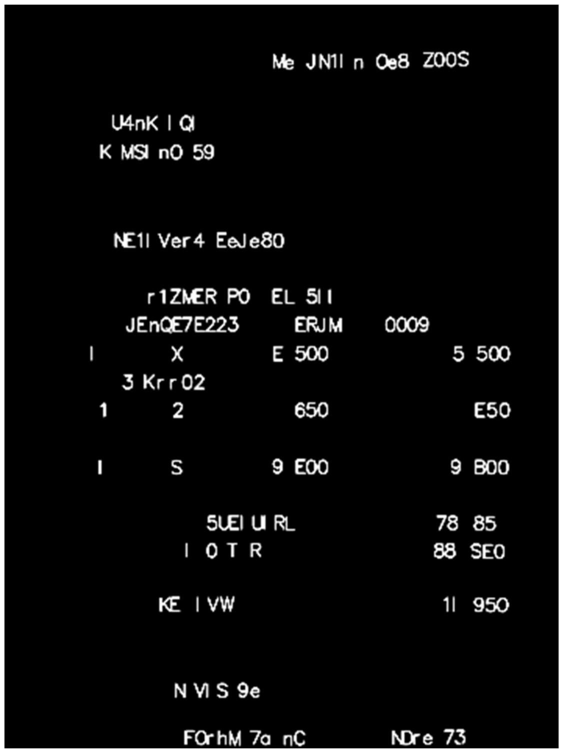
Figure 16 - Original Image



USIÑO No54 LOCAL 1 SANTIAGO
.40 del 15

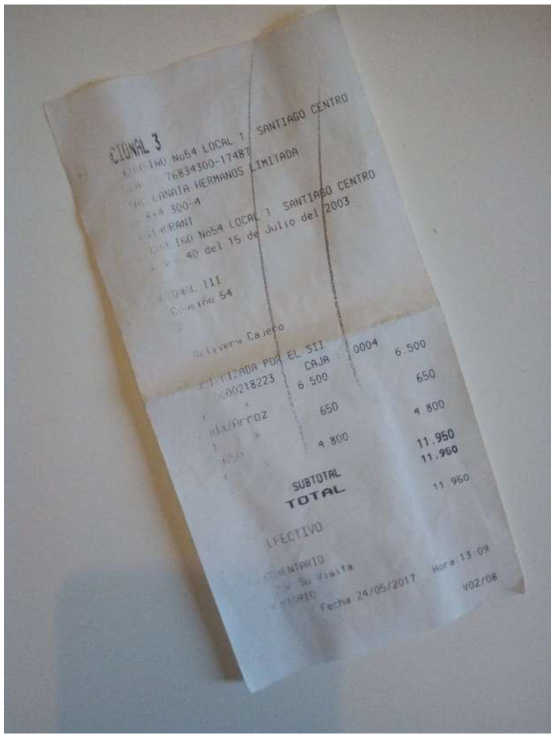
COMENTARIO
NTARIO
2017

Figure 17 - Located Characters



```
                  Me JN1l n Oe8 Z00S

   U4nK I Ql
  K MSl nO 59


   NE1l Ver4 EeJe80

      r1ZMER P0  EL 5l l
    JEnQE7E223     ERJM    0009
I       X        E 500          5 500
    3 Krr02
1       2          650             E50

I       S        9 E00          9 B00

           5UEl Ul RL           78 85
         I  O T R               88 SE0

       KE  I VW                 1l  950


         N Vl S 9e

          FOrhM 7a nC       NDre 73
```

Figure 18 - Image Replication



```
CIONAL 3
[illegible]INO No54 LOCAL 1  SANTIAGO CENTRO
     76834300-17487
[illegible] LANAIA HERMANOS LIMITADA
[illegible] 300-4
[illegible]RANT
[illegible]INO No54 LOCAL 1  SANTIAGO CENTRO
[illegible] 40 del 15 de Julio del 2003

[illegible]ONAL III
Cosino 54

Delivery Cajero

[illegible]IZADA POR EL SII     0004
[illegible]00218223      CAJA
                          6.500
  1    x     6 500
[illegible]a/Arroz                 650
   1   x     650
                                4.800
   1   x     4 800
                               11.950
                               11.950
        SUBTOTAL
        TOTAL                  11.950

EFECTIVO

[illegible]ENTARIO
[illegible] Su Visita
[illegible]ARIO   Fecha 24/05/2017   Hora 13:09
                                V02/08
```

Figure 19 - Original Image

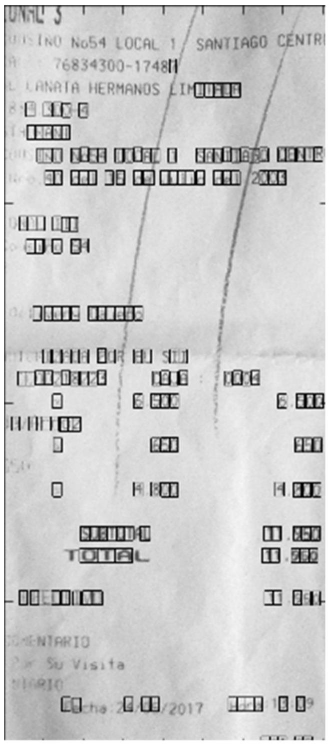
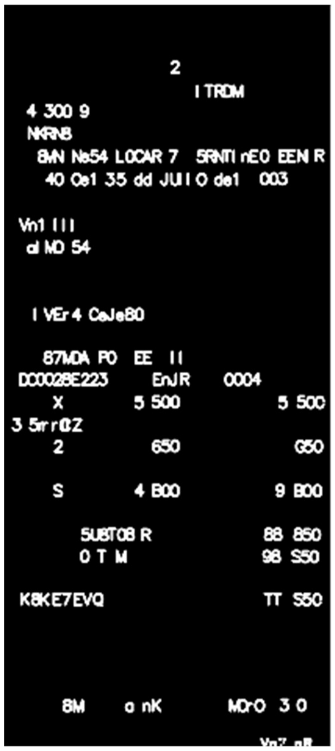


Figure 20 - Located Characters



```
                2
                  I TROM
 4 300 9
 NKRNB
  8MN No54 LOCAR 7   5RNTI nE0 EEN R
   40 Os1 35 dd JUI I O de1   003

Vn1 I I I
 al MO 54


 I VEr 4 CaJe80

  87MDA P0   EE   I I
DC0028E223      EnJR     0004
    X          5 500              5 500
3 5rrr0Z
    2           650                G50

    S          4 B00              9 B00

       5UBT08 R                 88  850
       0 T M                    98  S50

K8KE7EVQ                        TT  S50




      8M      o nK        MOrO  3 0
                                  Vn7 o⊞
```

Figure 21 - Image Replication

When testing the TensorFlow network it received an 87% accuracy on the EMNIST dataset. This exceeds its benchmark of 81% that was seen by the newspaper print models. However, when testing on the receipt dataset the accuracies were closer to 43%. This large difference shows a disconnect in image processing between the two datasets and could be an area of improvement in the future. In the context of this project, the task was to surpass 81% on a testing set from the same dataset as which the network was trained. We will say that the final program was successful but with room for improvement.

The Decision Tree section performed very well giving results of 99% or more and surpassing its benchmark of human binary labeling by 5%. Although, it should be noted that when the engineer labeled the receipt dataset they only gave non-dead space labels to characters they could recognize. In some cases, characters on the physical receipt were labeled as dead space due to poor resolution or lighting conditions. As a result, the 99% accuracy on the Decision Tree section of the program is misleading. The true accuracy is probably closer to its 94% human benchmark.

The program was designed with the scope of replicating one receipt and within this boundary it proves to be very robust. It was tested with images taken at different angles, with different lighting and backgrounds. In each case, the overall accuracy was between 40 and 45 percent. Unfortunately, when the model was tested with images of other receipts it did not perform well. When slicing the image into blocks it frequently selected an incorrect font size. This resulted in every block being a little too big or a little too small for the character it was trying to contain.

Justification

The results demonstrate a strong basis or starting point for an OCR receipt replication program. It has a clear pipeline for image processing and testing. It is able to locate, rotate, and crop images with different backgrounds and lightings. Overall this program completes its objective of replicating the image and provides a strong framework for future advancements. It outperforms its benchmarks of hand-labeling, newspaper print classification, and random-guessing.

Conclusion

Visualization of Overfitting

Below are processed images of receipts outside of our receipt dataset. As can be seen, the program does a horrible job of dividing up the image into blocks. The ironic part of this finding is that the pipeline was first built using these images. Later, the receipt dataset was created and the filters were adjusted to better process the restaurant receipt. It is now clear that some serious overfitting took place in the adjustment of the filters. I find the process of identifying areas of interest very intriguing and open to experimentation. In this case, the canny-edge-detector was heavily used along with assumptions about the geography of a receipt. However, while the assumptions were necessary to obtain an acceptable accuracy in our dataset, they did not generalize well to the greater population. Perhaps in future iterations and more dynamic approach would be appropriate, the use of machine learning algorithms earlier in the pipeline could very well make the program more practical in real life settings.

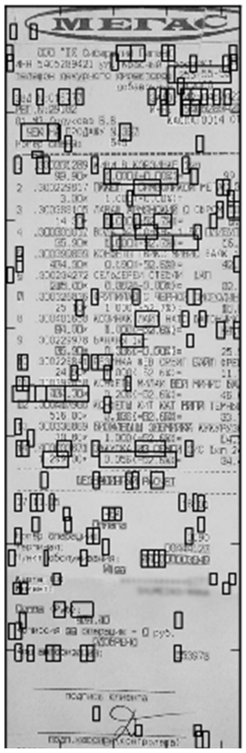


Figure 22 - Character Location

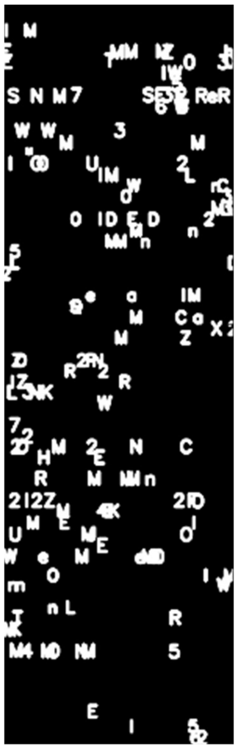


Figure 23 - Image Replication

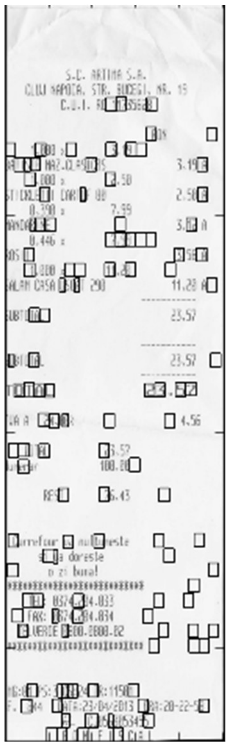


Figure 24 - Located Characters

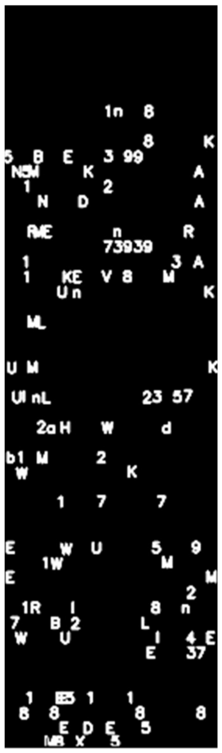


Figure 25 - Image Replication

Reflection

This project proved to be much more involved than what was originally thought. At first glance, it seems simple to train a network using a large, well-known dataset like EMNIST and then simply pass real images from a camera phone. However, dividing an image into blocks was not simple. Even following the examples of predecessors, hours of trial and error were necessary. Part of the problem was the quality of the chosen receipt. The fading ink made it difficult for even the human eye to detect the letter without context. Another issue was the dependency on other processes like the canny-edge-detector or findContoures function. If either one of these performed poorly the rest of the pipeline was a lost cause.

The introduction of the average font size was the key to obtaining high results. Nonetheless, I believe that it is in this section where the majority of the overfitting takes place. As mentioned above the identification of blocks was first built using two example images from online resources, after some tweaking was done to the findContoures filters and the average font size theory was implemented the results went from "not bad" to "unrecognizable". In spite of that, it appears that this should be a major area of focus if this project is to be used in a more generalized field.

When the proposal was written it was assumed that only one decision tree algorithm was necessary. The idea was that the image would be entered and a yes or no response would be outputted. In practice, this system guessed correctly about 80% of the time. In several cases a small 3 or 4-pixel wide image was misidentified as a character when it was stretched to fill the 28x28 input. When considering the data lost because it was deemed 'unrecognizable' by the engineer and the incorrect classifications of the TensorFlow network, 80% was not sufficient. Luckily, the answer was relatively simple, the yes/no response generated by the first decision tree algorithm would be used as an engineered feature for the second algorithm. These two algorithms in unison give great results almost always surpassing 99.5%

The construction of the TensorFlow network was based on the work done in my image-classification nanodegree project. With some trial and error, more layers were added and several hours of hyperparameter selection was done utilizing the Bayesian Optimization Theory. A major roadblock was found with the accuracy began to approach 80%. At this point the appearance of vanishing gradients completely destroyed the network. Several hours of research and coding were spent on finding a viable solution. Finally, the loss function was rewritten to add a small value to each gradient so that they would never fully reach zero. This solution, while effective, has a "band-aid" feel to it and is definitely an area for improvement in the future.

Overall the system is not perfect, but I feel that the biggest accomplishment is the construction of a complete pipeline from image input to replication. From this point forward there is lots of work to be had, but they will be modifications to existing sections of the program, not sections being started from scratch.

Improvement

There are three major areas of improvement the overfitting in the image preprocessing, the Tensorflow network, and the disconnect between the classification accuracy on the EMNIST dataset and the accuracy for the receipt dataset. All three of these areas have been mentioned about but will be reiterated here of completeness.

The image preprocessing sector severely overfitted to the single receipt present in the receipt dataset. To compensate for this the process should be altered to consider more dynamic approaches, possible the use of machine-learned algorithms instead of hard numbers like “average width”.

The TensorFlow network suffered from vanishing gradients. A small bias was added to the output of the loss function to prevent any value from being exactly zero. However, this is not the most elegant approach and in the future more research should be done to find how to best handle such event.

There is almost a 40% loss between the accuracy on the EMNIST dataset and the receipt dataset. This indicates that there is a serious difference in the preprocessing performed on the two banks of images. At this moment, it is not clear where the major difference lies but efforts to close this gap will have great impacts on the overall performance of the program.

Summary

In summary, this was a very captivating project. There were many moments where I found myself thinking “Oh I could just change this one thing” or “Oh, what if I…”. From TensorFlow to Python, I feel like I’ve learned as much in this project as in any structured coding course. I also have a better understanding of where I would like my career trajectory to take me. I feel my future lies in neural networks and the amazing ability they have to mimic human understanding of the world. In the future, I hope to fine tune this project more and write an in-depth / beginner-friendly blog to accompany it, with the idea of creating a springboard for future OCR projects.