# How-To Guide

CMPT 310 - D200 Introduction to Artificial Intelligence, Summer 2025

Group 6

Daniel Shi, Farzan Ustad and Andy Wang

## 1. Project Overview

Our project goal was to build an AI powered tool that can extract purchase totals from scanned and photographed receipts, then return them in batch via a CSV file. We used Python with OpenCV for image processing and Pytesseract to convert the receipt images into text. Post-processing techniques were used to clean the OCR output and accurately identify the total amounts.

## 2. Step-by-Step Implementation

### Folder Structure

Data folder
- dataOut folder contains results.csv file which is our output
- gdt folder contains all the solution for the images
- image folder is the image dataset used in the AI model

Document
- Contains project proposal, Milestones 1 and 2, and the demo poster

src/preprocess
- All our code for preprocessing images

src/textExtraction
- text extraction functions, full scale test function and two lists for true and false ‘total’

src/Training_set
- cross validate function

### Cleaning the Database

When we first tested our text detection algorithm, we found some images had poor resolution and were extremely difficult (even for humans) to read. We went through our data set and deleted all the images with broken text, unusual patterns, tears, or smudges that made the text difficult to examine. This process removed the noisiest and un-usable images, ensuring that our OCR pipeline was trained and tested only on images with a reasonable chance of recognizing text.

### Preprocessing

Why is preprocessing necessary?

Preprocessing was quite necessary because raw receipt images often contain a lot of noise, skewness, uneven lighting, and low contrast, which all together can cause our model to misread characters and return a lower accuracy. By applying steps like grayscalling, binarizing, and skew correction, we were able to enhance text clarity and improve the contrast between characters and the background of the receipt. This overall ensures that we can more accurately detect and interpret text on receipts. Without this crucial step in our program, our OCR results would be far less reliable, leading to more errors in extracting the total amounts.

### Hyper Parameter Training (Cross Validation)

Our Cross validation code takes two parameters max_files and n_splits. Max files is the number of unique images that our training will use while n_splits is the amount of folds used. For every hyper parameter test, images will go through our full pipeline to return a result. We first start off with a base hyper parameter combination, and each parameter is separately adjusted to be tested. Finally we return a combination of every separately tested parameter that produces the highest accuracy. We create K-folds using Scipy learn and after iterating every fold we return the mean of all the fold accuracies. This is the accuracy that's later used to determine the optimal config.

```
Using 50 receipts for sequential CV (max_files=50)
  4% (50 of 1250) |======                                     | Elapsed Time: 0:03:13 ETA:   1:17:23B
aseline accuracy: 0.5400
Baseline params: {'clip_limit': 2.0, 'tile_grid_size': (16, 16), 'bin_method': 'otsu', 'block_size': 15, 'C': 5}
```

### Convert to Grayscale

```
Optimizing clip_limit...
  8% (100 of 1250) |===========                               | Elapsed Time: 0:06:23 ETA:   1:13:34
  clip_limit = 1.0: 0.4600
 12% (150 of 1250) |================                          | Elapsed Time: 0:09:35 ETA:   1:10:18
  clip_limit = 2.0: 0.5400
 16% (200 of 1250) |======================                    | Elapsed Time: 0:12:46 ETA:   1:07:03
  clip_limit = 3.0: 0.5200
 20% (250 of 1250) |============================              | Elapsed Time: 0:15:57 ETA:   1:03:48
  clip_limit = 4.0: 0.4800
   No improvement. Keeping clip_limit = 2.0

Optimizing tile_grid_size...
 24% (300 of 1250) |==================================        | Elapsed Time: 0:19:07 ETA:   1:00:35
  tile_grid_size = (8, 8): 0.5800
 28% (350 of 1250) |========================================  | Elapsed Time: 0:22:18 ETA:   0:57:20
  tile_grid_size = (16, 16): 0.5400
 32% (400 of 1250) |==============================================| Elapsed Time: 0:25:29 ETA:   0:54:10
  tile_grid_size = (32, 32): 0.4200
   Improved! tile_grid_size: (16, 16) -> (8, 8)
   Score: 0.5400 -> 0.5800
```

Before running anything we convert our image to grayscale. We use Contrast Limited Adaptive Histogram Equalization (CLAHE), followed by grayscale. This function takes two hyperparameter inputs for CLAHE, which are clip_limit and grid_size. The clip limit sets a maximum allowed histogram bin height, this means lower values means less contrast boost

which is better for avoiding noise. Higher values for clip_limit means stronger contrast enhancement which can be helpful for faded text but can introduce unwanted distortion. Through our training we found that clip_limit = 2, grid_size = (16,16) procured the highest accuracy for our dataset. Although depending on the dataset oftentimes grid_size being (8,8) will produce the exact same accuracy.

**Biniarize the Image**

To biniarize our grayscale image, we take three hyperparameters. First is to choose our thresholding method, which is between adaptive and otsu. If adaptive thresholding is chosen, our other two parameters block_size and C (constant) become more important. Block_size dictates how local the threshold is, while the C parameter is a constant subtracted from our local mean for thresholding. A smaller C parameter means more pixels become black, while a larger C value means more pixels become white. Throughout our training on certain models adaptive thresholding has higher accuracy however on our current database Otsu thresholding produces a better accuracy.

```
Selecting binarization method with Adaptive tuning...
 36% (450 of 1250) |=======================================================                                                                    | Elapsed Time: 0:28:47 ETA:   0:51:10
  Otsu score: 0.5800
 40% (500 of 1250) |=============================================================                                                              | Elapsed Time: 0:31:57 ETA:   0:47:56
  Adaptive block_size=11, C= 2: 0.0200
 44% (550 of 1250) |===================================================================                                                        | Elapsed Time: 0:35:03 ETA:   0:44:37
  Adaptive block_size=11, C= 5: 0.0600
 48% (600 of 1250) |==========================================================================                                                 | Elapsed Time: 0:38:11 ETA:   0:41:21
  Adaptive block_size=11, C=10: 0.1600
 52% (650 of 1250) |================================================================================                                           | Elapsed Time: 0:41:22 ETA:   0:38:11
  Adaptive block_size=11, C=15: 0.2600
 56% (700 of 1250) |======================================================================================                                     | Elapsed Time: 0:44:36 ETA:   0:35:03
  Adaptive block_size=15, C= 2: 0.0800
 60% (750 of 1250) |============================================================================================                               | Elapsed Time: 0:47:47 ETA:   0:31:51
  Adaptive block_size=15, C= 5: 0.1200
 64% (800 of 1250) |===================================================================================================                        | Elapsed Time: 0:50:57 ETA:   0:28:39
  Adaptive block_size=15, C=10: 0.1600
 68% (850 of 1250) |=========================================================================================================                  | Elapsed Time: 0:54:08 ETA:   0:25:28
  Adaptive block_size=15, C=15: 0.3000
 72% (900 of 1250) |===============================================================================================================            | Elapsed Time: 0:57:38 ETA:   0:22:24
  Adaptive block_size=21, C= 2: 0.1200
 76% (950 of 1250) |=====================================================================================================================      | Elapsed Time: 1:00:50 ETA:   0:19:12
  Adaptive block_size=21, C= 5: 0.1800
 80% (1000 of 1250) |=========================================================================================================================  | Elapsed Time: 1:04:01 ETA:   0:16:00
  Adaptive block_size=21, C=10: 0.3200
 84% (1050 of 1250) |===============================================================================================================================  | Elapsed Time: 1:07:14 ETA:   0:12:48
  Adaptive block_size=21, C=15: 0.4000
 88% (1100 of 1250) |========================================================================================================================================  | Elapsed Time: 1:11:21 ETA:   0:09:43
  Adaptive block_size=25, C= 2: 0.1400
 92% (1150 of 1250) |==============================================================================================================================================  | Elapsed Time: 1:14:34 ETA:   0:06:29
  Adaptive block_size=25, C= 5: 0.1800
 96% (1200 of 1250) |====================================================================================================================================================  | Elapsed Time: 1:17:47 ETA:   0:03:14
  Adaptive block_size=25, C=10: 0.2600
100% (1250 of 1250) |===========================================================================================================================================================| Elapsed Time: 1:21:01 ETA:  00:00:00
  Adaptive block_size=25, C=15: 0.3800
   → Keeping OTSU
```

### Deskew the Image

This function deskews an image, taking in three hyper parameters on top of the image path. The three parameters are -limit, +limit and a delta. The image is rotated by the delta in both directions until it reaches the limits. Every rotation is scored using the sum of squared differences between adjacent rows. These parameters are not trained in our cross validation system since obviously a higher limit would result in a better accuracy, however time is lost in the process.

### Text Detection

We use the *pytesseract.image_to_data* function to detect the text and its relative location and to create a table. We then sort the table and group columns that have the same block number and line number. Before finding the total and date, we convert the grouped data into a list of strings.

This separate function reads the list line by line and identifies if there is any match to a list of keywords used in receipts that represent ‘total’. Then, we compare this line with another set of keywords used in receipts that do not represent ‘total’, such as “sub total” or “total without GST”. We then extract the number from the line of text and store it in a list.

```python
for i in range(len(lines)):
    line = lines[i]
    upper_line = line.upper()
    # Check if the line contains any of the true total keywords
    if any(true_kw in upper_line for true_kw in true_total_list):
        # check if the line contains any of the fake total keywords
        if not any(fake_kw in upper_line for fake_kw in fake_total_list):

            # extrac the total value from the line
            matches = re.findall(r'\d{1,3}(?:,\d{3})*(?:\.\d{2})|\$\d+(?:\.\d{2})?', line)
            if matches:
                cleaned = re.sub(r'[^\d\.]', '', matches[0])  # strip $, commas, etc.
                total_value = float(cleaned)
                GuessedTotal.append(total_value)
```

We then return the max of the list as our detected total.

This entire function will return 2 values, the total amount and the date from the receipt.

- Currently, we have not implemented date detection.

**Accuracy Test + CSV Output**

To test the accuracy of our code, fullScaleTest.py is a test function that reads a group of images, extracts text from each image, and stores the results in a list. This test function also reads the corresponding solution data file and stores it in a separate list.

We then use these lists to generate a CSV file. By analyzing the CSV file, we can identify errors made during text detection, unreadable images, and calculate the overall accuracy.

The CSV file path is cmpt310-rcpt-scan\data\dataOut\results.csv

```
100% (100 of 100) |########################################| Elapsed Time: 0:03:51 Time:  0:03:51
============================================================
Total files processed: 81
Total files with correct total: 56
Accuracy: 69.14%
============================================================
```

## 3. Challenges or Roadblocks

Even after we remove poor quality images based on human perspective, this does not dramatically increase the accuracy. We decided to continue to remove poor quality images that computers can't detect.

## 7. Changes from Original Plan

In our proposal we did not plan to have a test for accuracy, however based on feedback given on our last milestone we implemented it as a way to test the efficiency of our system. Another unplanned change we implemented cross validation. We did this in an attempt to increase the accuracy of our system which it has not yet, however we believe it has potential with a better cross validation implementation. Lastly, using a text detection(i.e. bounding boxes with CRAFT) separate from recognition was not deemed necessary by our group at this time.

## 9. Conclusion

Our Project demonstrates how OCR, combined with effective preprocessing techniques, can accurately extract purchase totals from receipt images. By cleaning our dataset, applying preprocessing steps like grayscalling, binarizing, and skew correction, we were able to enhance text clarity and improve the contrast between characters and the background of the receipt. The resulting data we get left with after these steps was significantly cleaner and more consistent, helping us attain a higher accuracy. On a future note, with further enhancements, such as advanced image enhancement or deep learning-based text recognition, this approach could be scaled into a one of a kind, real-world application.