

|  |
| --- |
| **Project Report 2 :**  **Text Image Segmentation for Optimal OCR (Optical character recognition)** |
| **CZ4003 –** **COMPUTER VISION** (Semester 1, AY 2020/2021)  WILSON THURMAN TENG  U1820540H |

Table of Contents

[1. **Introduction** 2](#_Toc57409793)

[1.1 Overview 2](#_Toc57409794)

[1.2 Dependencies 2](#_Toc57409795)

[1.3 Program Structure 2](#_Toc57409796)

[1.4 Tesseract Setup 3](#_Toc57409797)

[2. **How does Tesseract OCR work?** 4](#_Toc57409798)

[3. **Metrics** 6](#_Toc57409799)

[3.1 Jaccard Similarity 6](#_Toc57409800)

[3.2 Cosine Similarity 7](#_Toc57409801)

[3.3 Levenshtein Similarity 7](#_Toc57409802)

[4. **Algorithms** 9](#_Toc57409803)

[4.1 Otsu algorithm 9](#_Toc57409804)

[4.2 Skew angle algorithm 10](#_Toc57409805)

[5. **Pre-processing** 11](#_Toc57409806)

[5.1 Step 0: Image Upscaling 11](#_Toc57409807)

[5.2 Step 1: Obtain the illumination and shadow of the image 12](#_Toc57409808)

[5.3 Step2: Remove illumination/shadow from image 12](#_Toc57409809)

[5.4 Step 3: Rotate image (if required) 13](#_Toc57409810)

[5.4 Conclusion 13](#_Toc57409811)

[6. **Results** 14](#_Toc57409812)

[6.1 ‘sample01.png’ results (Best score achieved – 100%) 14](#_Toc57409813)

[6.1.1 Otsu Thresholding 15](#_Toc57409814)

[6.1.2 Adaptive Thresholding 16](#_Toc57409815)

[6.2 ‘sample02.png’ results (Best score achieved – 99.04%) 16](#_Toc57409816)

# 1. Introduction

## Overview

In this report, various algorithms are implemented to improve the accuracy of optical character recognition (OCR) on 2 images, “sample01.png” and “sample02.png. OCR aims to recognize texts in imaged documents and is one of the earliest computer vision techniques that have been commercialized successfully. An open-source OCR software, [Tesseract](https://github.com/tesseract-ocr/tesseract) is used.

OCR usually involves a series of image processing and recognition tasks including:

1. Text image binarization to convert a colour/grayscale image into a binary image with multiple foreground regions (usually characters).
2. Connected component labelling to detects each binarized character region.
3. Character recognition by using classifiers (E.g. pre-trained neural network).

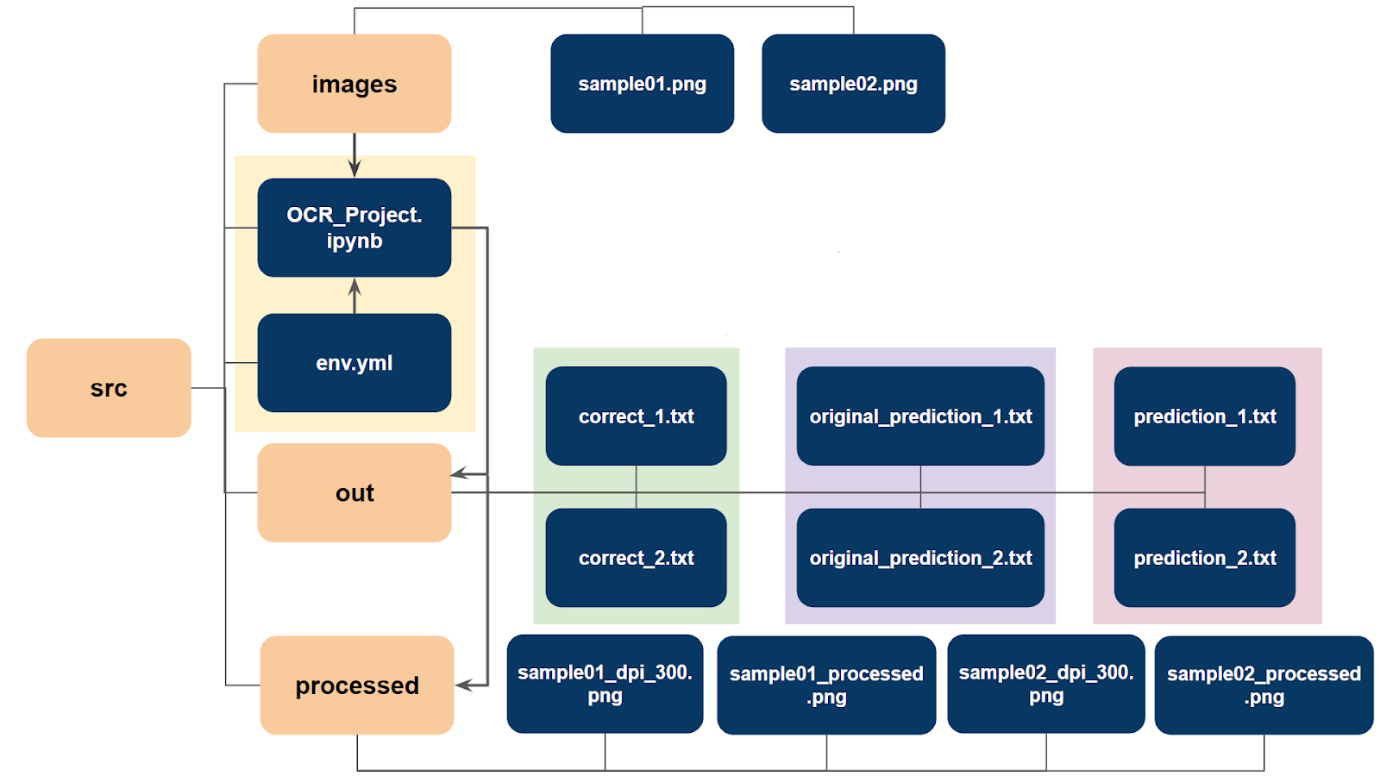
## 1.2 Dependencies

The following tools will be used in the implementations of this lab report:

* Python v3.6.11 (Source code tested on this version)
* Conda v4.9.2
  + Dependencies listed in ‘src/env.yml’
* Jupyter Notebook
* Tesseract v5.0.0-alpha.20200328

## 1.3 Program Structure

The files in the source code are arranged as follows:



* **Images** contain the original images.
* **Out** contains the text files of the images (ground truth text, original prediction, prediction after processing).
* **Processed** contains the processed images by *‘OCR\_Project.ipynb’*.
* *OCR\_Project.ipynb’* is the script which contains all the processing code’
* *‘env.yml’* is the yaml dependency file for Anaconda.*’*

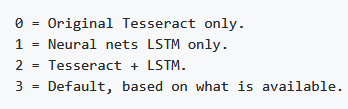
## 1.4 Tesseract Setup



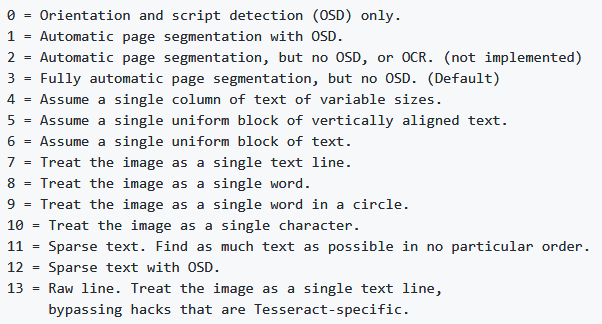
The configuration above is defined for the Tesseract engine to ensure that the results in this report are replicable.



A whitelist of the above characters are also used to ensure that the prediction does not introduce any unwanted special characters.

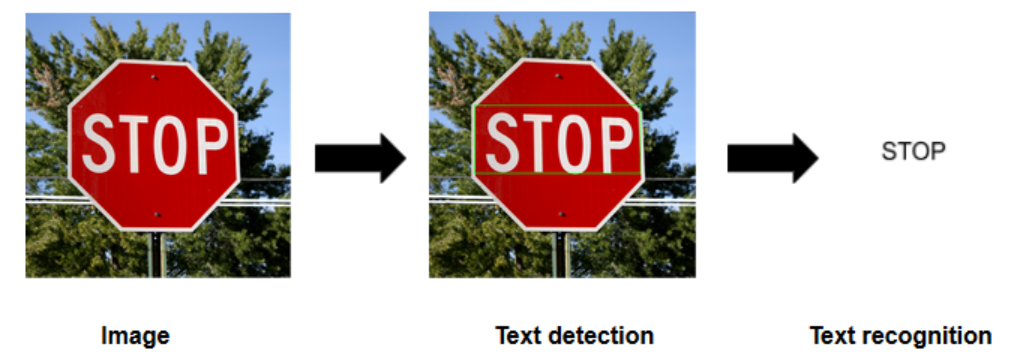


**OEM** refers to the OCR Engine Mode. In this setup, 3 is chosen which uses the Default.

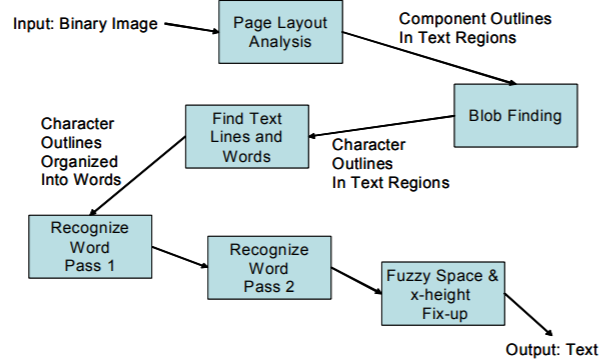


**PSM** refers to the Page Segmentation Mode. In this setup, the default mode 3 is selected. This mode features automatic page segmentation but there will be no orientation and script detection.

# 2. How does Tesseract OCR work?



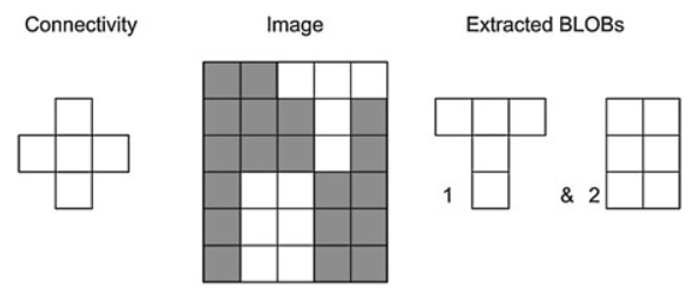
Optical character recognition, widely known as OCR, is an open-source technology developed for converting text from a document or image into editable and searchable text.



The main stages of Tesseract OCR are as follows:

1. Connected Component Analysis
2. Text Line Segmentation
3. Word Segmentation
4. Word Recognition

**Connected Component Analysis**



The first step is a connected component analysis where the outlines of a component in a given image are uniquely identified. As shown below, the outlines of the white blocks are gathered/nested together. Afterwards, they are extracted into Blobs (Binary Large Object).

**Text Line Segmentation**



Blobs are organized into text lines and a baseline is established. This is done through partitioning the blobs into groups with a reasonably continuous displacement for the original straight baseline.

**Word Segmentation**



Afterwards, the lines and regions are analysed for **fixed pitch** or **proportional text**. When a fixed pitch is detected, it is likely that there is a kind of character spacing. In this case, the word is then segmented or chopped off into characters.

**Word Recognition**

|  |  |
| --- | --- |
|  |  |
| With chop points | After chops |

After which, Tesseract identifies concave vertices of a polygonal approximation of the outline and uses chop points to separate the joined characters. The chop is only executed if there is an improve in the confidence of the result.

Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down the page.

# 3. Metrics

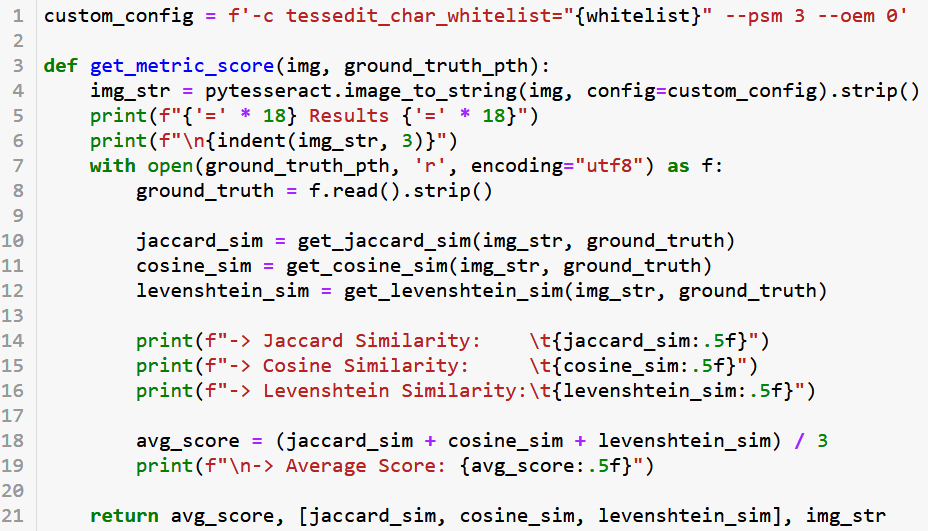
The similarity metrics used in this report lean heavily towards those used in Natural Language Processing (NLP), which is suitable for this task.

In this project, the following metrics are used:

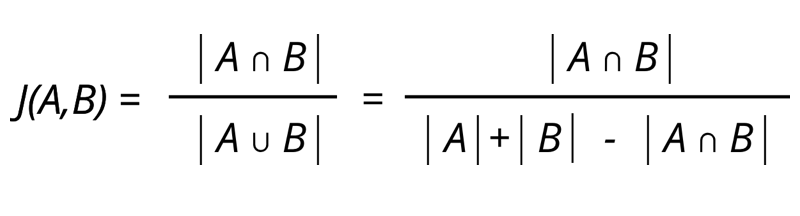
1. Jaccard Similarity
2. Cosine Similarity
3. Levenshtein Similarity

The 3 metrics are then averaged, and the mean score is used to compare the performance of the pre-processing performed.

**Source code:**



## 3.1 Jaccard Similarity

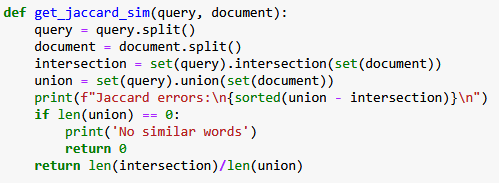


Jaccard similarity computes the percentage of common words between 2 strings over all possible words from both strings.

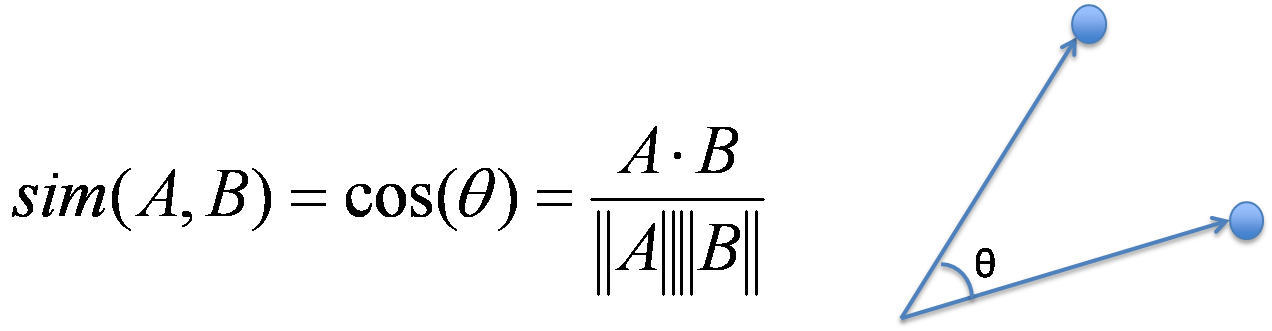
Jaccard Similarity is used in NLP to find out the similarity between 2 strings by the words used in those 2 strings. Punctuation and stop words (i.e. Common words used in the language, pronouns are an example of stop words) are often removed before calculating the Jaccard similarity. Words are also converted to their base form (i.e. past tense word ‘done’ is changed to ‘do’).

However, in this project, since we are also interested in converting punctuation accurately, we split the passage by whitespaces and use these tokens to compute the Jaccard similarity.

**Source Code:**

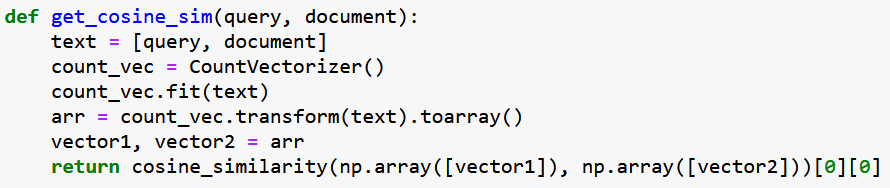


## 3.2 Cosine Similarity



The cosine similarity assumes both strings can be represented in the vector space and takes the normalized dot product of these 2 vectors.

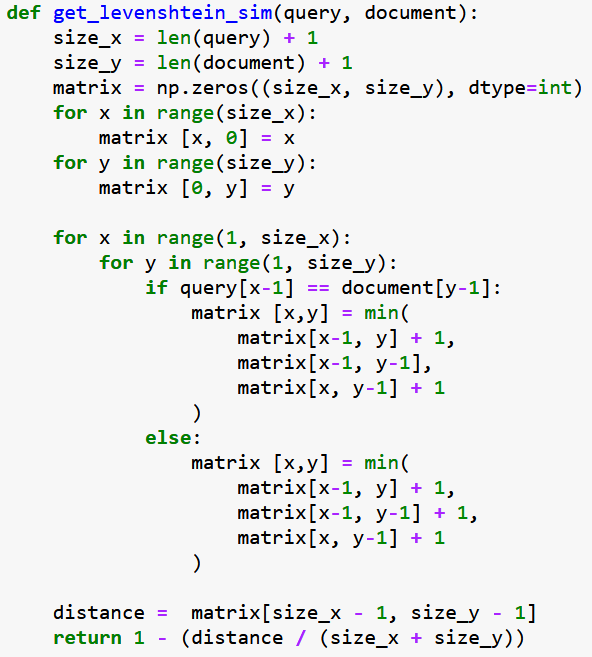
**Source Code:**



## 3.3 Levenshtein Similarity

To compute the Levenshtein similarity between 2 strings, we compute the amount of insertion, deletion or substitution operations necessary to convert 1 string into the other. The amount of change the string needs to undergo is then computed as the percentage of changes over the combined length of both strings. Levenshtein similarity is then computed by subtracting this percentage from 1.

**Source Code:**



# 4. Algorithms

## 4.1 Otsu algorithm

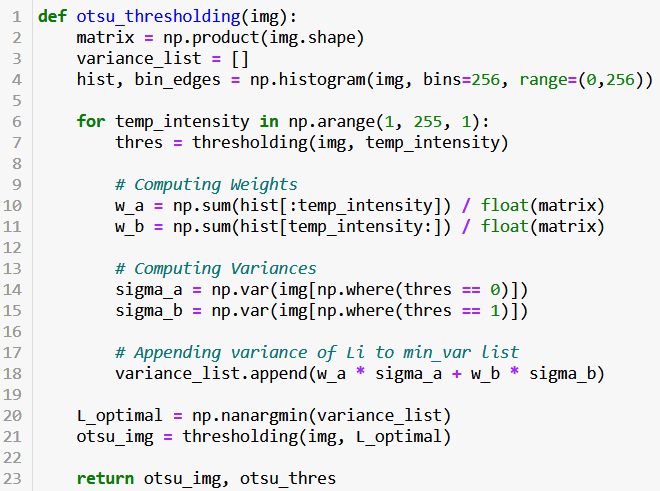
Otsu is used to define a globally optimal threshold under the assumption that the intensity distribution is bimodal. An exhaustive search to minimize intra-class variance is performed on all possible threshold values. The minimization results in the maximization of inter-class variance which allows us to arrive at the desired threshold.



**Li** corresponds to the intensity that is tested.  
**wa** and**wb** corresponds to the probabilities of the two classes separated by a threshold. This is computed as the percentage of pixels in a class.  
**σa** and **σb** corresponds to the standard deviation of the two classes.



**L\*** corresponds to the globally optimal threshold which is computed as the intensity value which minimizes intra-class variance.

**Source code:**  


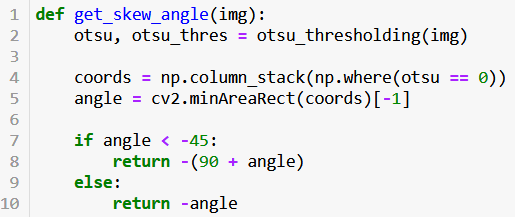
## 4.2 Skew angle algorithm

This algorithm aims to correct the skew angle of the image under the assumption that the text is written in paragraphs which will result in a rectangular bounding box.

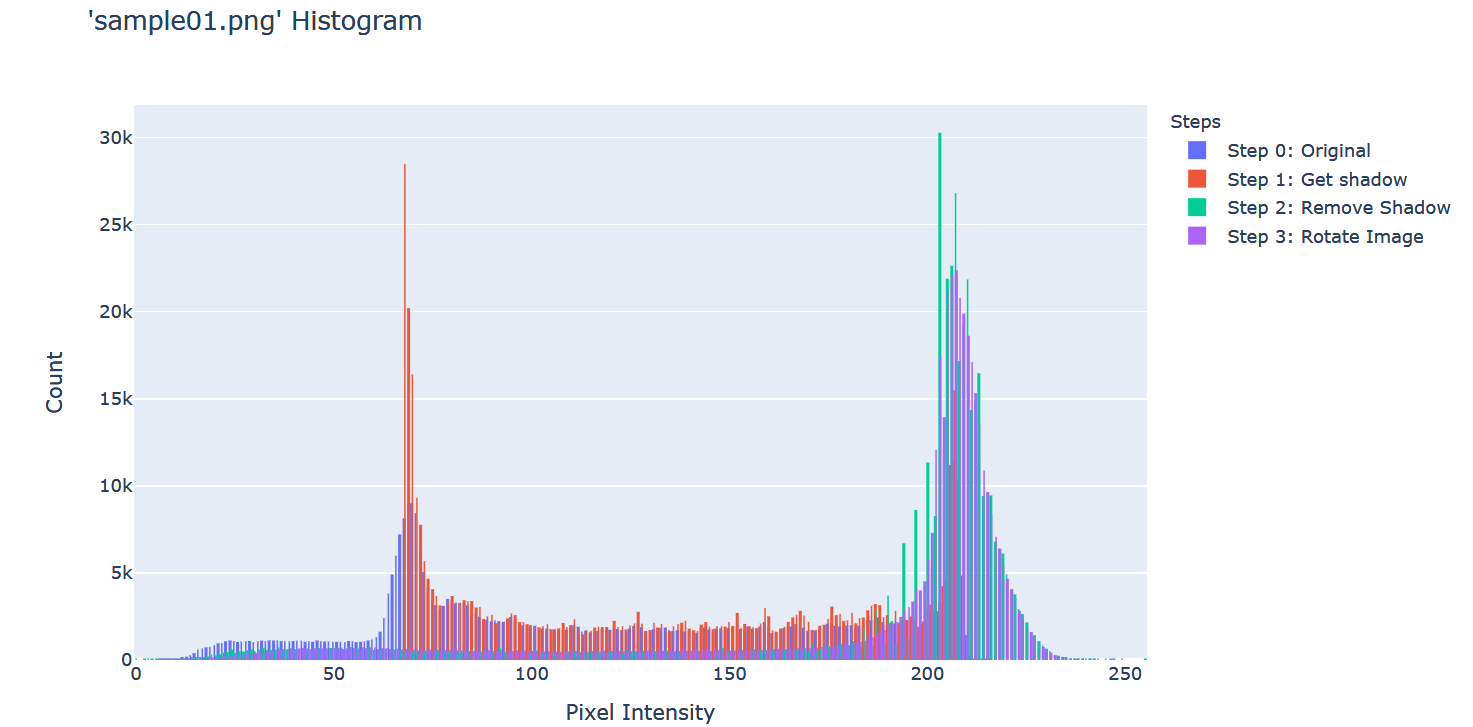
This is done by first performing Otsu thresholding to the image. Coordinates of the pixel of characters (intensity == 0) are then extracted. A minimum area rectangle is computed from these coordinates, and we obtain the skew angle of this computed rectangle.

This method is suitable for images where intensities of the characters and its background are easily separable which allows the computed rectangle to be accurate to the passage.

**Source Code:**

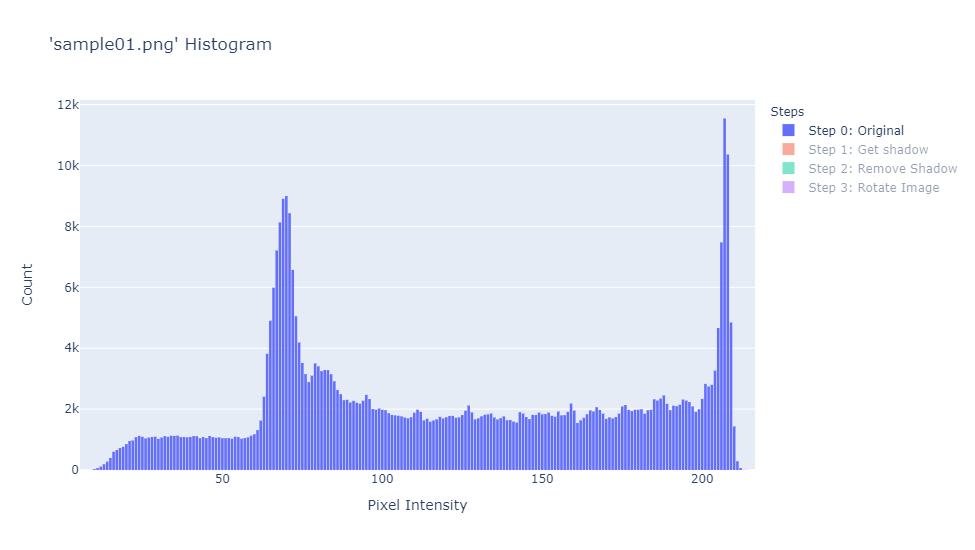


# 5. Pre-processing



The histograms used in this section are from ‘sample01.png’. For an interactive experience with these histogram plots, either ‘OCR\_Project.ipynb’ or ‘OCR\_Project.html’ can be explored.

## 5.1 Step 0: Image Upscaling

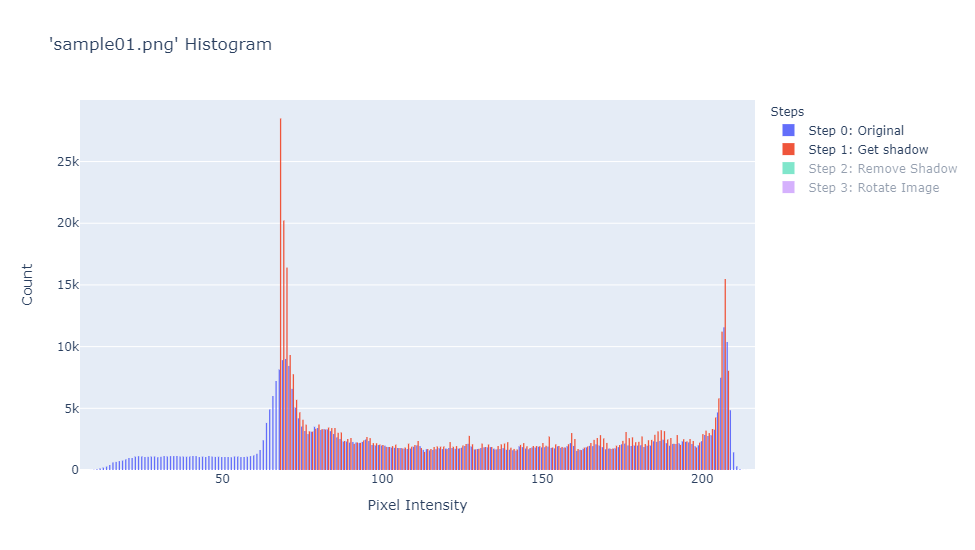


The image is first upscaled between 1.4-1.5x. This is to ensure the image has at least 300 dpi as recommended by Tesseract.

The benefits to upscaling are two-fold:

* This allows for subsequent algorithms to use larger kernels which introduces more flexibility to the shape of kernels that can be used.
* Interpolation between old pixels may create new pixels that fill in gaps between characters.

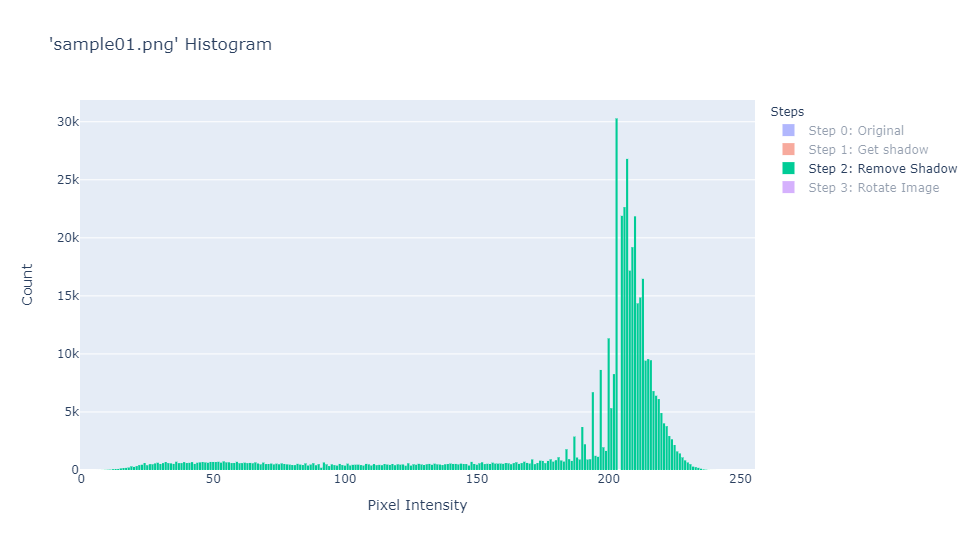
## 5.2 Step 1: Obtain the illumination and shadow of the image



This is achieved by applying Median Blur with a large kernel size to the image. This allows the resultant image to ignore the optical characters and produce the gradient of the image, which explains the illumination and shadow of the image well.

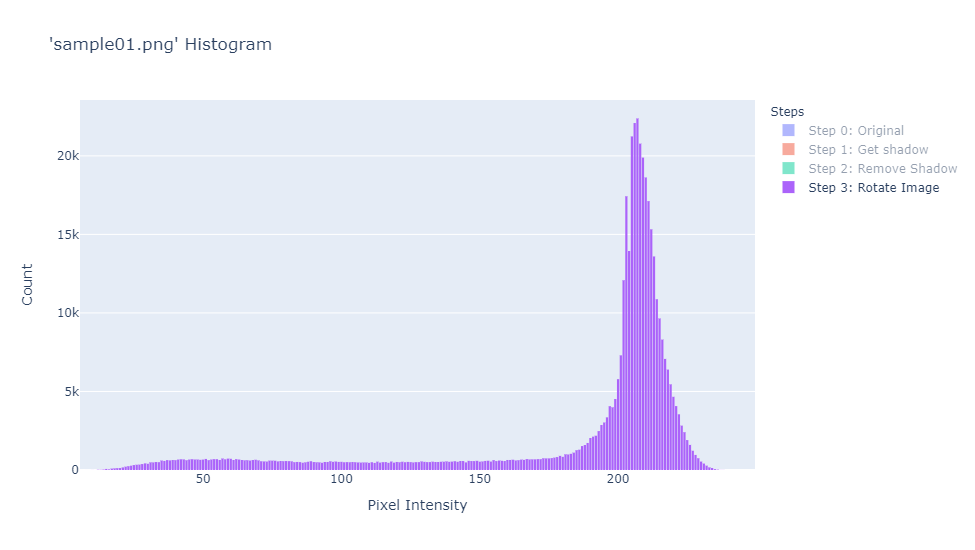
It can be observed from the histogram plot above that the shadow and the original image overlap for many intensities, hence we can conclude that a global threshold technique would be insufficient to identify an optimal threshold.

## 5.3 Step2: Remove illumination/shadow from image



This step reduces the effect of illumination and shadows. The original image is divided by the shadow obtained in the previous step and contrast stretched to the original [0,255] range. This darkens the previously illuminated parts of the image, while the dark parts of the image remain at similar intensities.

## 5.4 Step 3: Rotate image (if required)



Finally, in the last stage of pre-processing, we utilize the skew angle algorithm explained in *section 4.2* and apply an affine transformation to our original image.

It is interesting to note that the histogram of the rotated image follows a much smoother curve as compared to before rotation.

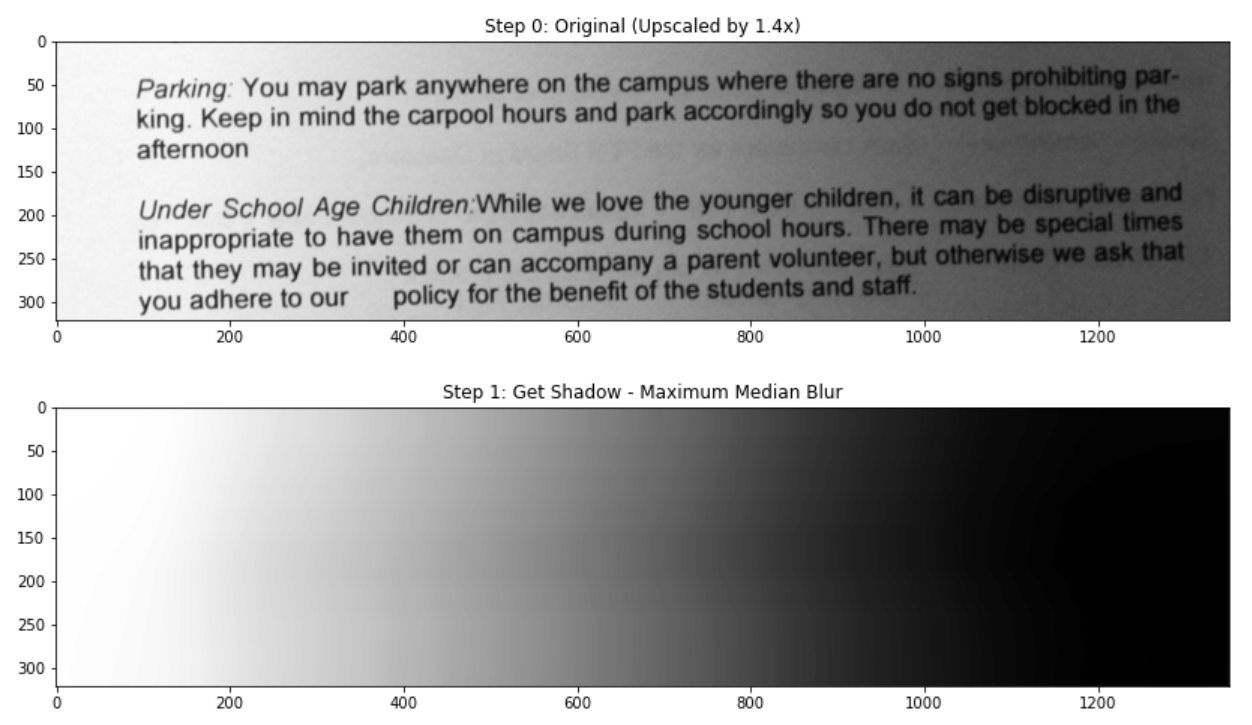
## 5.4 Conclusion

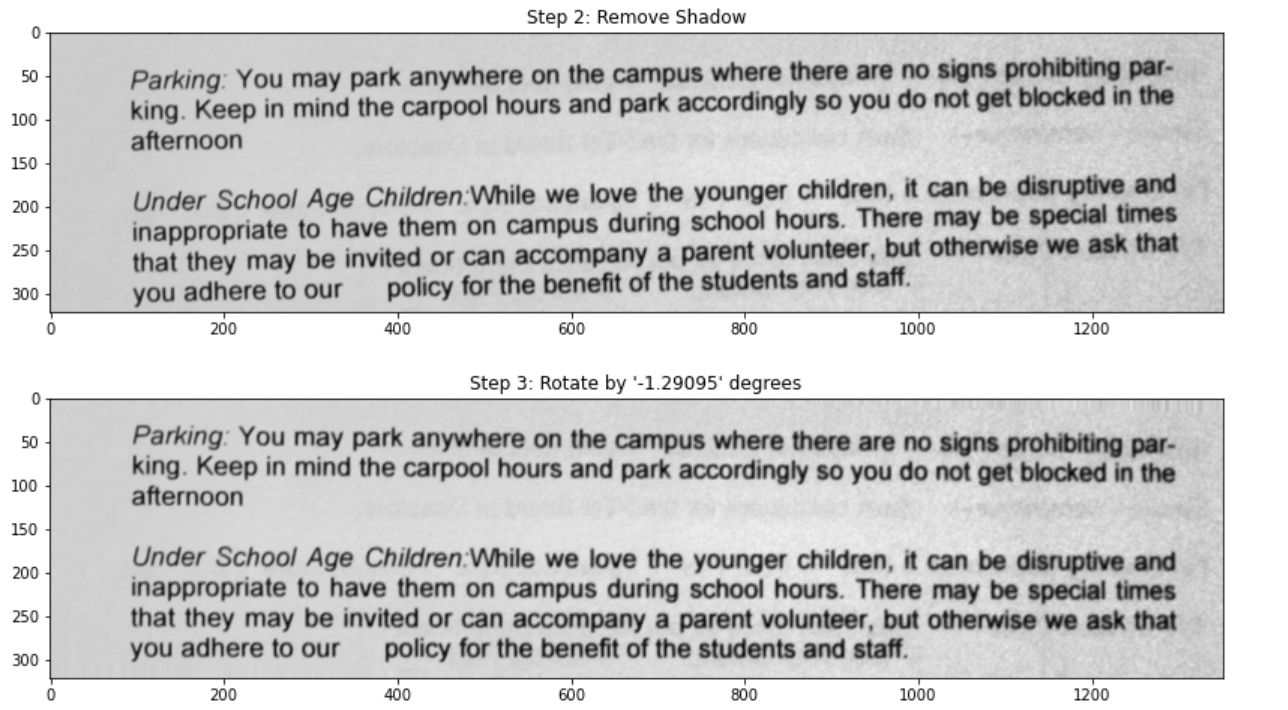
It can be observed in that the final histogram obtained is not bimodal, therefore binarizing the image may not be useful at all for obtaining accurate OCR results.

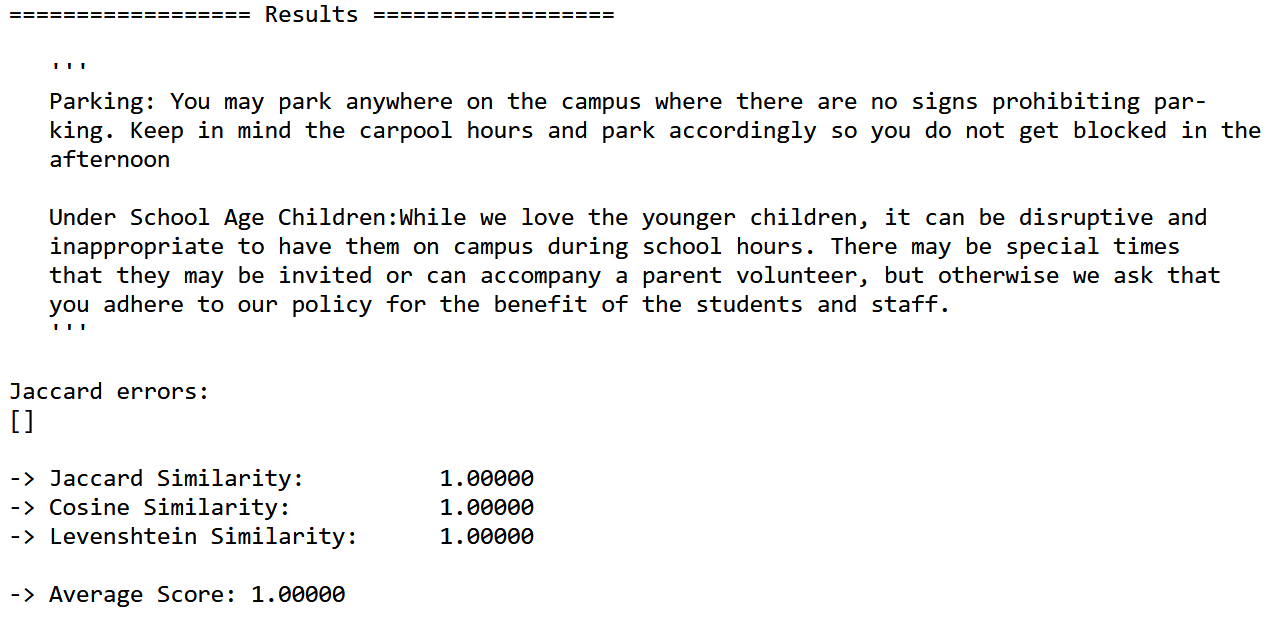
# 6. Results

## 6.1 ‘sample01.png’ results (Best score achieved – 100%)

The following show the results of the pre-processing steps outlined in the previous section as well as the score achieved.

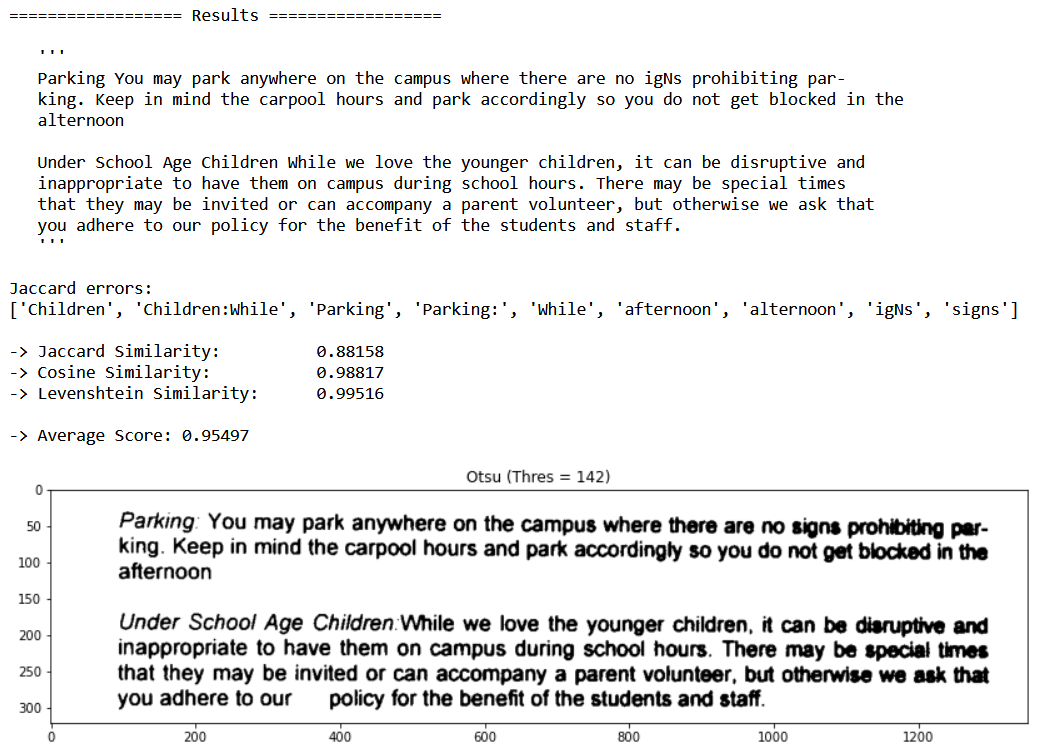






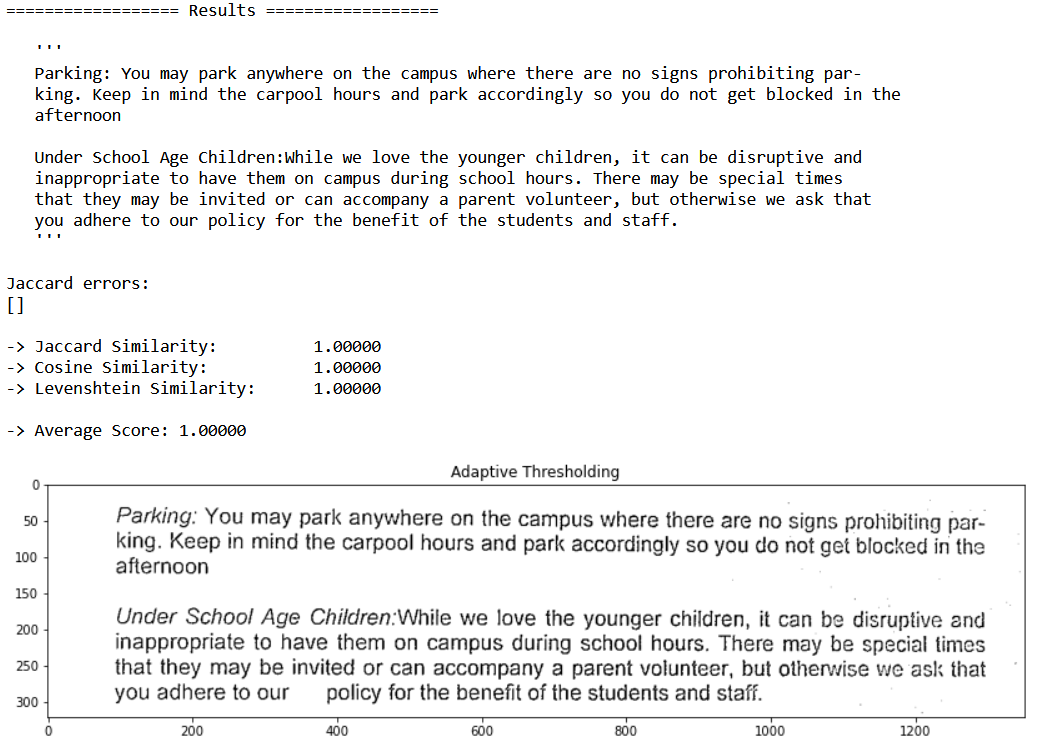
Experimentation with Otsu and adaptive thresholding on the processed image was also attempted but these binarization algorithms did not improve the score. This could be attributed to the unimodal distribution of the processed image observed.

### 6.1.1 Otsu Thresholding



It can be observed that Otsu thresholding results in thick strokes caused by the remaining slight skew. This affects the accuracy of the OCR. Eroding and dilation were also experimented with after Otsu thresholding to make the strokes thinner, but the results were not promising. The benefits of thinning the characters with wide strokes was offset by characters which are already sufficiently thin.

### 6.1.2 Adaptive Thresholding



Adaptive thresholding is also applied to the processed image. This allows for locally optimal thresholding which results in a clean binarized image.

## 6.2 ‘sample02.png’ results (Best score achieved – 99.04%)

|  |  |
| --- | --- |
|  |  |
|  |  |

As observed previously for ‘sample01.png’, binarization is not necessary to achieve high OCR accuracy if the processed image is clean. In this situation, only the quote before “damn” is missing from the prediction.

It was hypothesized that this “damn” quote missing could be attributed to the new Tesseract LSTM engine (oem 2) which would stipulate that the closing “ quote be accompanied by an open ‘ quote. However, after switching to the legacy engine (oem 0), the results remained the same.

Therefore, this quote missing can be attributed to the light ink in the picture.

# 6. Discussion (Optional)

This section discusses techniques to improve on OCR.

## 6.1 Types of image degradation

## 6.2