CS 663: Digital Image Processing

Assignment 1

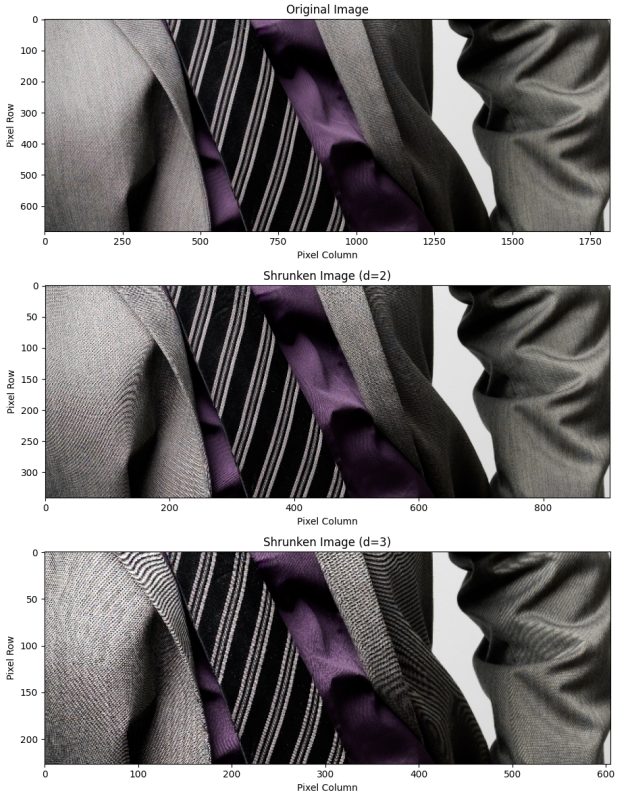
Pratiek Sonare (22B2440)
Entenuka Jogarao (22B2702)
Instructor: Suyash P. Awate

August 29, 2025

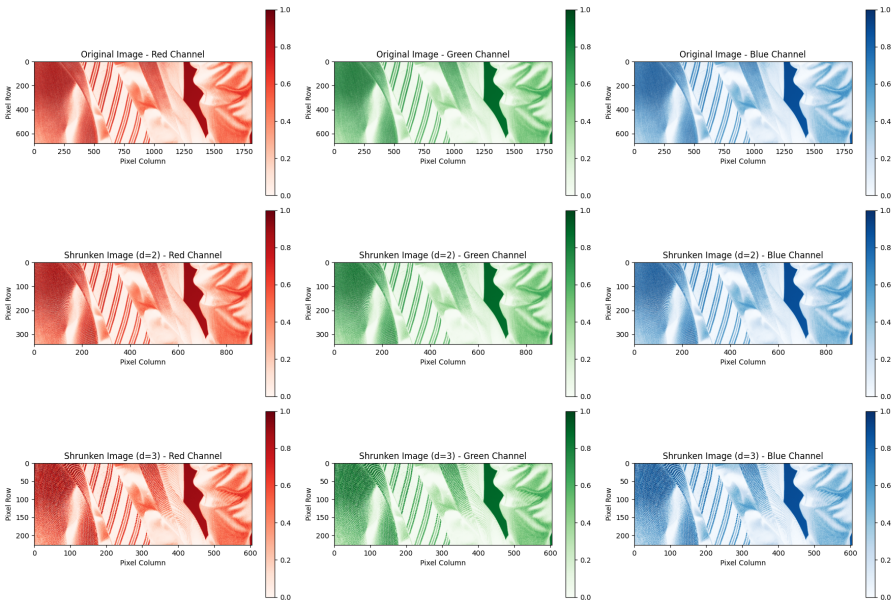
1 Image Subsampling and Interpolation

1.1 Image Shrinking

Function: `myImageShrink()`
Summary: We shrink the image by skipping every d-th sample, effectively sub-sampling the image. This results in Moire effect - where we see a distorted image with black pixels Distortions are clearly visible by zooming in the images shrunk by d=2 and d=3



(a) Original and Shrinked Images (with Moire Effect)



(b) RGB channels for each subsampled image

Figure 1: Image Shrinking results.

1.2 Nearest Neighbor Interpolation

Function: `myNearestNeighborInterpolation()`
Summary: Enlarging the image to { 300(M-1) + 1, 300(N-1) + 1 } from { M, N } pixel range using Nearest Neighbour interpolation — each new pixel takes the value of the nearest pixel from the original image. The original image was in gray-scale, and the intensities are colour coded using the `jet` colormap.

1.3 Bilinear Interpolation

Function: `myBilinearInterpolation()`
Summary: Given a pixel location (x, y) in the resized image, mapped to fractional coordinates in the original image, let If the intensities of the four neighboring pixels are

$$f(x_1, y_1), \quad f(x_2, y_1), \quad f(x_1, y_2), \quad f(x_2, y_2),$$

then the interpolated value at (x, y) is computed as

$$f(x, y) = f(x_1, y_1)(1-dx)(1-dy) + f(x_2, y_1)(dx)(1-dy) + f(x_1, y_2)(1-dx)(dy) + f(x_2, y_2)(dx)(dy).$$

1.4 Bicubic Interpolation

Function: `myBicubicInterpolation()`
Summary: Bicubic interpolation considers a 4×4 neighborhood of pixels around $(x_{\text{int}}, y_{\text{int}})$. The interpolation value is computed as $f(x, y) = \sum_{m=-1}^{2} \sum_{n=-1}^{2} f(x_{\text{int}}+m,\ y_{\text{int}}+n)\, w(m-dx)\, w(n-dy)$, where $w(\cdot)$ is the cubic kernel.

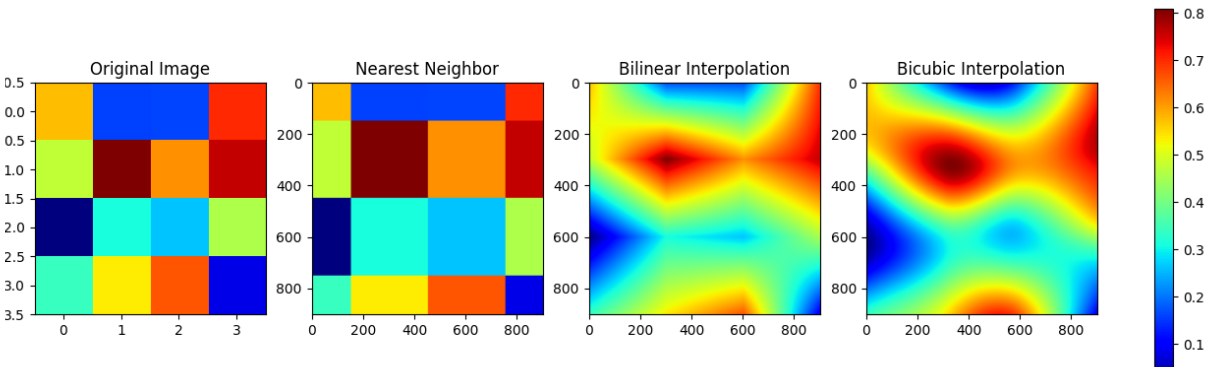


Figure 2: Image enlargement using NN, Bilinear, Bicubic interpolation

1.5 Image Rotation

Function: `myImageRotationUsingBilinearInterp()`
Function: `myImageRotationUsingNearestNeighborInterp()`

Summary: We calculate the rotation matrix for an input angle (here, 5 deg) and find the dot product with the image matrix to find the rotated image coordinates. Since, these coordinates may not always correspond to integer values, we interpolate them using NN / Bilinear interpolation methods to achieve the final image. Notice how empty spaces are blacked out after rotation!

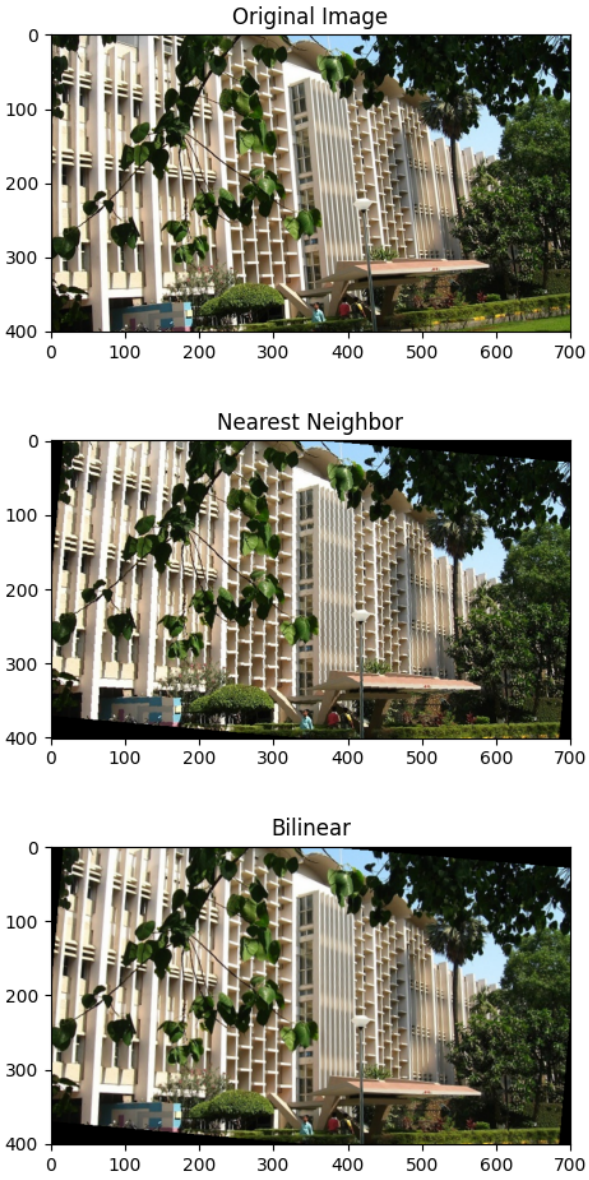


Figure 3: Image rotation using NN, bilinear interpolation.

1.6 Downsampling and Upsampling

Summary: To upsample or enlarge the subsampled CT scan images, we use and compare NN, Bilinear and Bicubic interpolation methods.
Key Observations: We find the NN method has the highest RMSE, also visible in the difference plot, while we see little to no difference in the bilinear and bicubic plots.

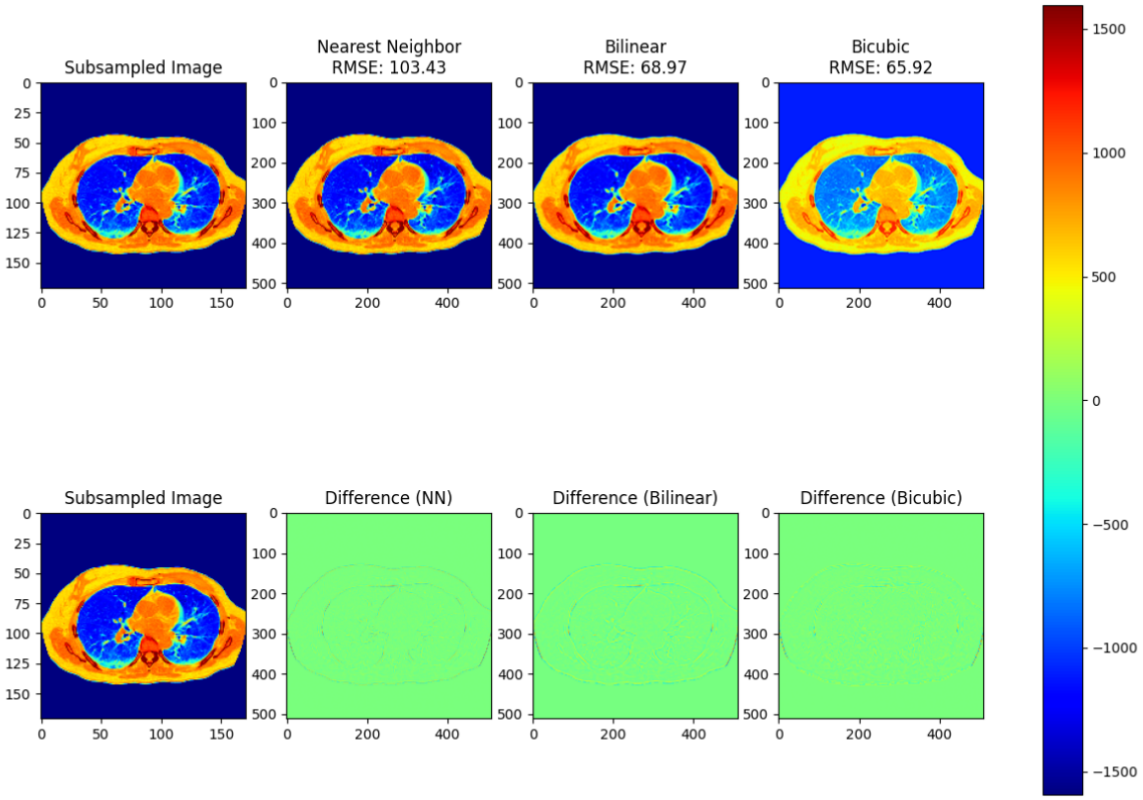


Figure 1: Chest CT Scans interpolation with difference using NN, Bilinear and Bicubic

Conclusion: For this Chest CT Scan, Bilinear and Bicubic interpolation methods work best since they result in the lowest RMSE and visible difference!

2 Thresholding

Input Images: The input images are in the .png and .tif format. All these images have varying maximum pixel intensities. Libraries such as `scikit-image` expect pixel intensities in the range of [0-255] (uint8 format). Hence, the intensities were scaled appropriately.

2.1 Manual Thresholding

Function: `myManualThreshold()`
Summary: By taking the average of RGB channels, we first convert the input image to grayscale and then input a threshold of 175, where intensity at a given pixel if ¡ 175, then it is assigned 0 (black) otherwise 255 (white)

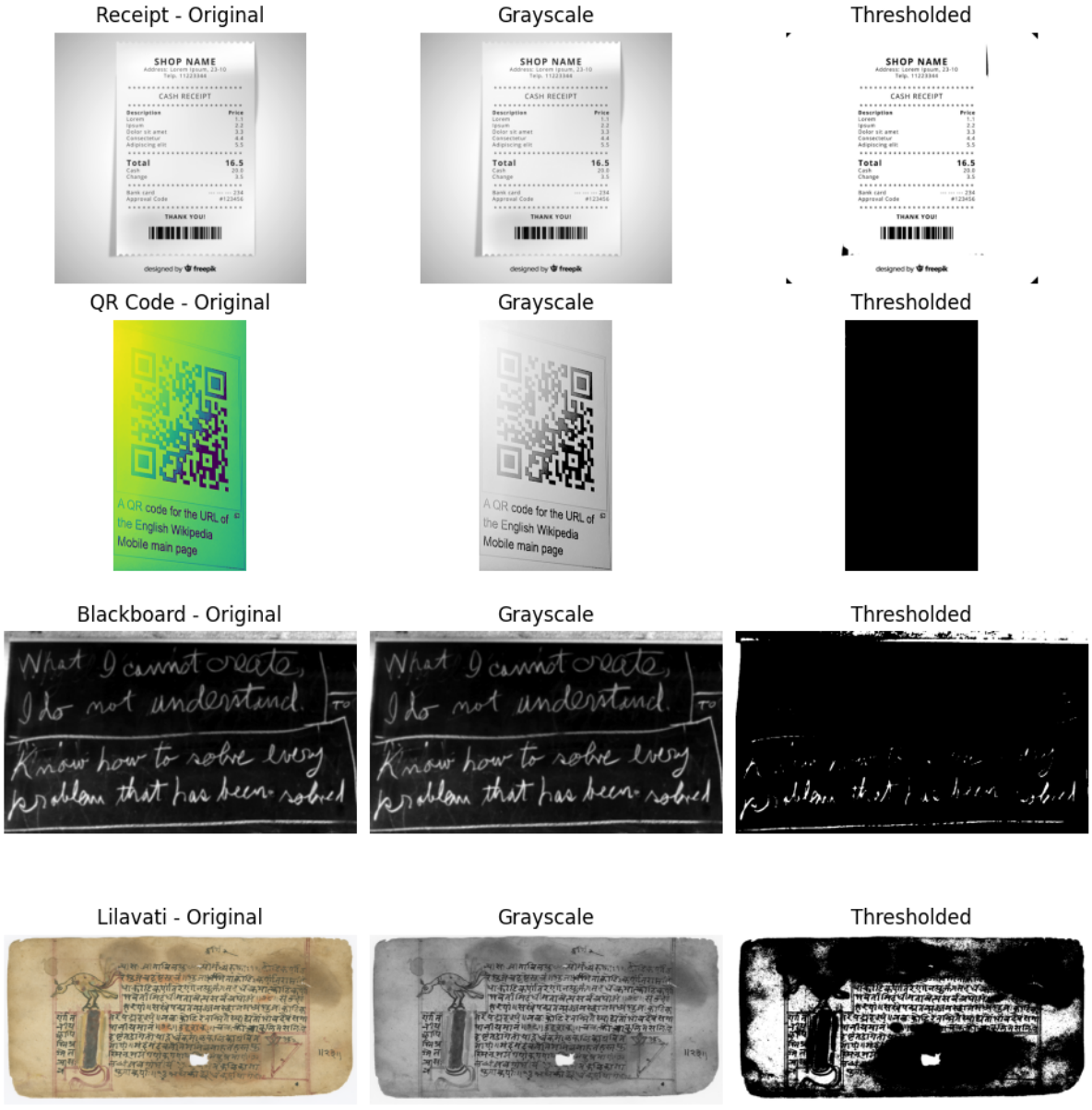


Figure 2: Manual thresholding results

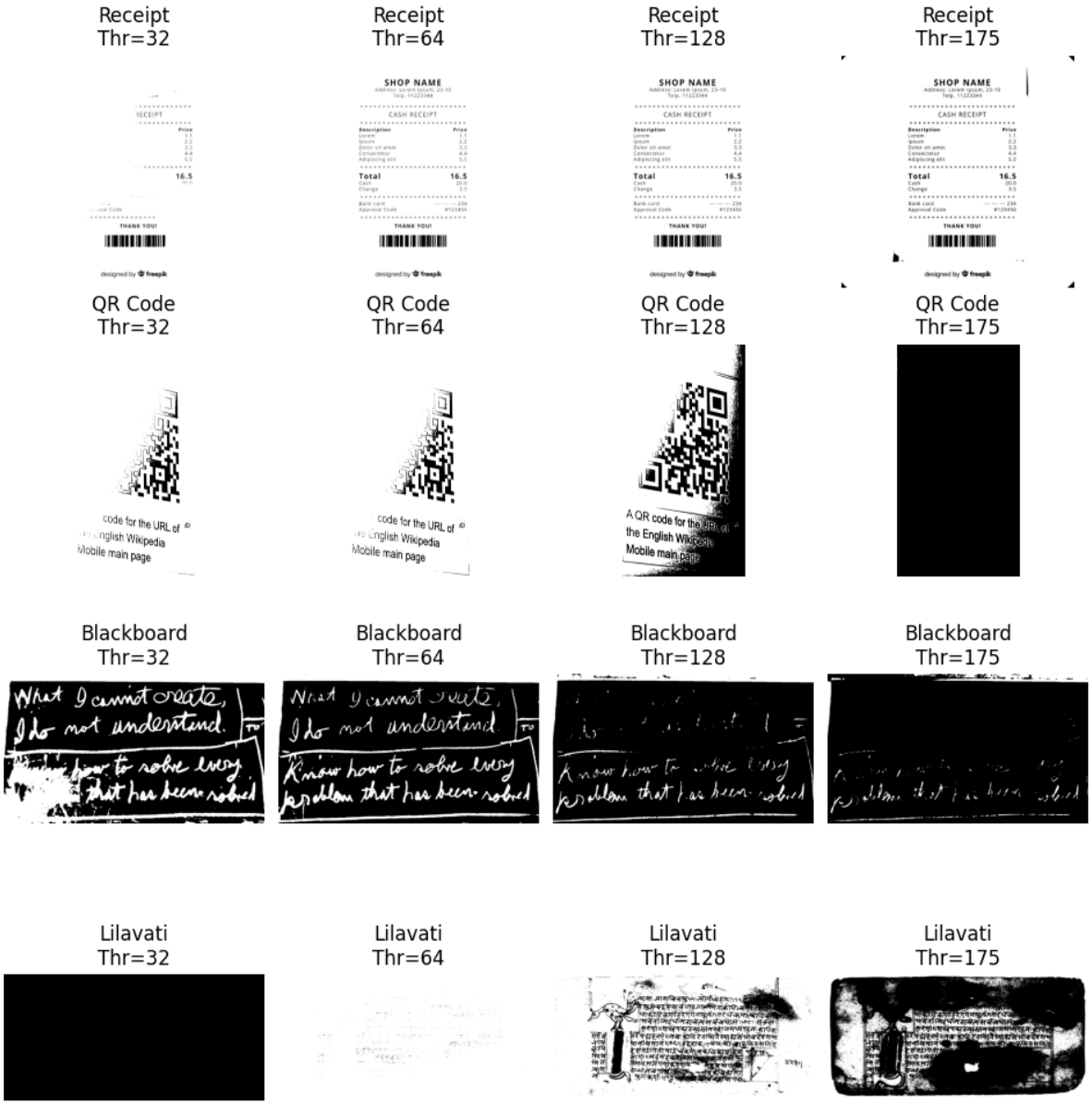


Figure 3: Comparing images for different thresholds

Key Observations: Thr=75 works best for the receipt, it may not work the best for the rest of the images where the QR or written text is not clearly visible, meaning even if they are written in white, the pixel intensity < 175 hence giving a black color! However, for lighter shades of shalk, they are classified as white when you bring down the threshold to Thr $= 32$

Conclusions: Lower threshold values can correctly classify lighter shades of white, however undermining black colors at the same time (notice Lilavati.tif) and vice-versa.

2.2 Otsu Thresholding

Function: `myOtsuThreshold()`
Summary: Otsu's method is an automatic image binarization technique that determines an optimal threshold by analyzing the image histogram and selecting the value that *minimizes the within-class variance* (or equivalently, *maximizes the between-class variance*) between foreground and background. This makes it adaptive to the image's intensity distribution, eliminating the need for a manually chosen threshold.
Key Observations: We observe better results for all four input images. Another key observation is that, all images have different Otsu Thresholds.

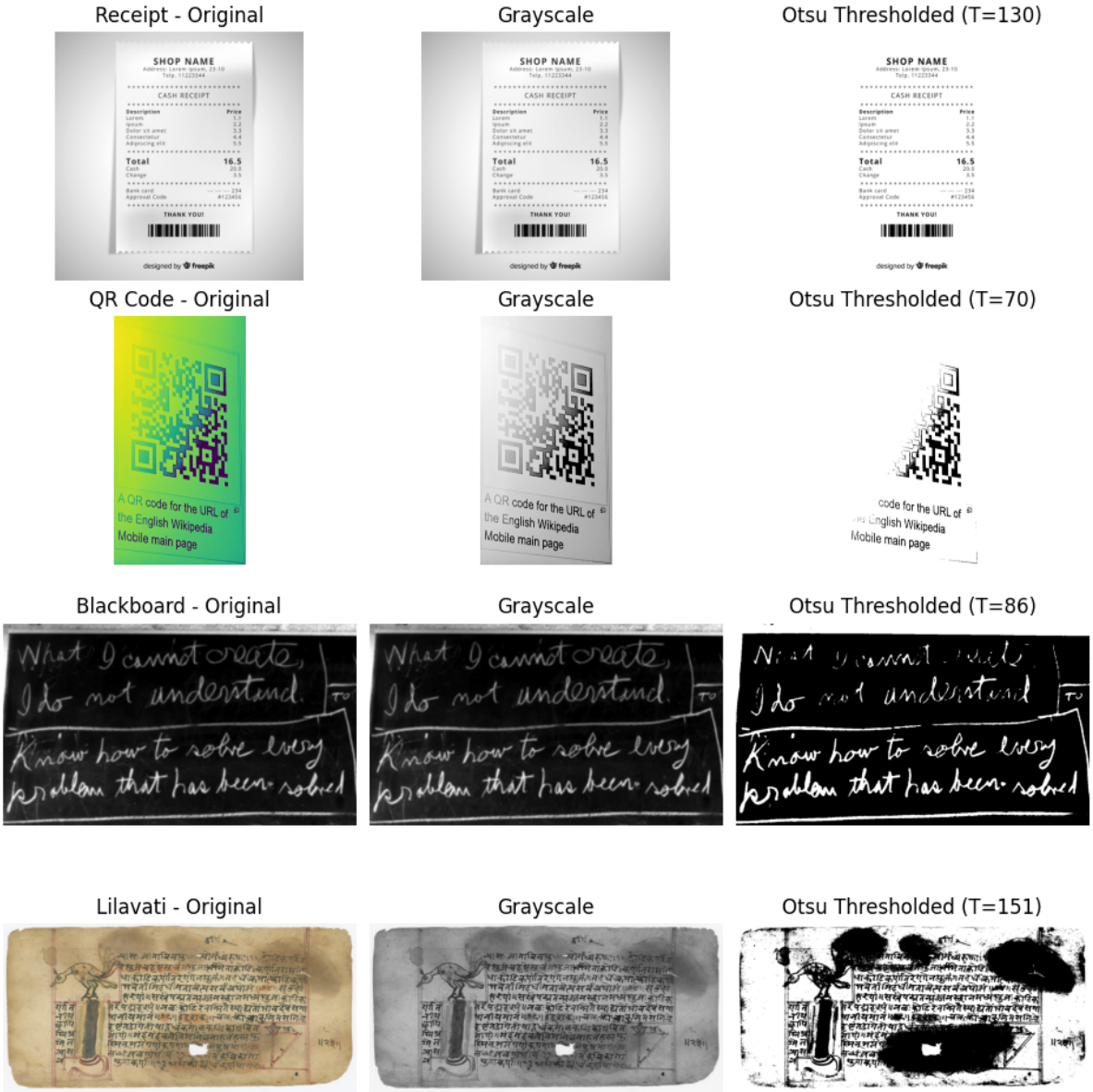


Figure 4: Otsu Threholded Images

2.3 Local/Adaptive Thresholding

Function: `myAdaptiveThreshold()`
Summary: We use Niblack's Adaptive Thresholdin method here which is a *local adaptive thresholding* technique used to binarize images with varying illumination or background. Instead of a single global threshold, the threshold at each pixel is computed based on the local mean and standard deviation in a sliding window centered at that pixel. The threshold is defined as:

$$T(x, y) = m(x, y) + k \cdot \sigma(x, y)$$

where $m(x, y)$ and $\sigma(x, y)$ are the local mean and standard deviation of the neighborhood around pixel (x, y), and k is a user-defined parameter (typically in the range -0.5 to -0.2).

Figure 5: Niblack's Adaptive Thresholding

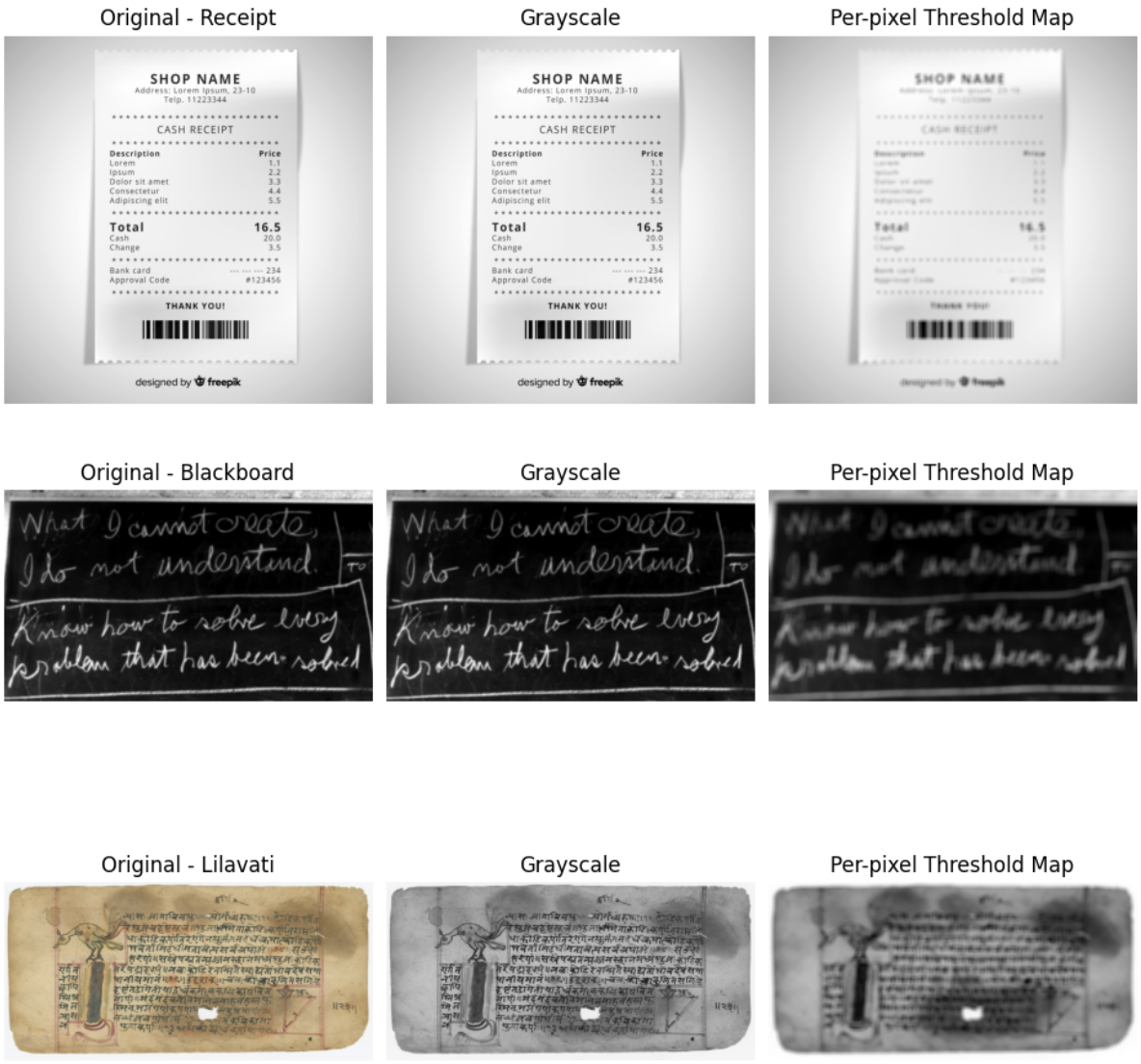


Figure 6: Perpixel threshold values of adaptive thresolding

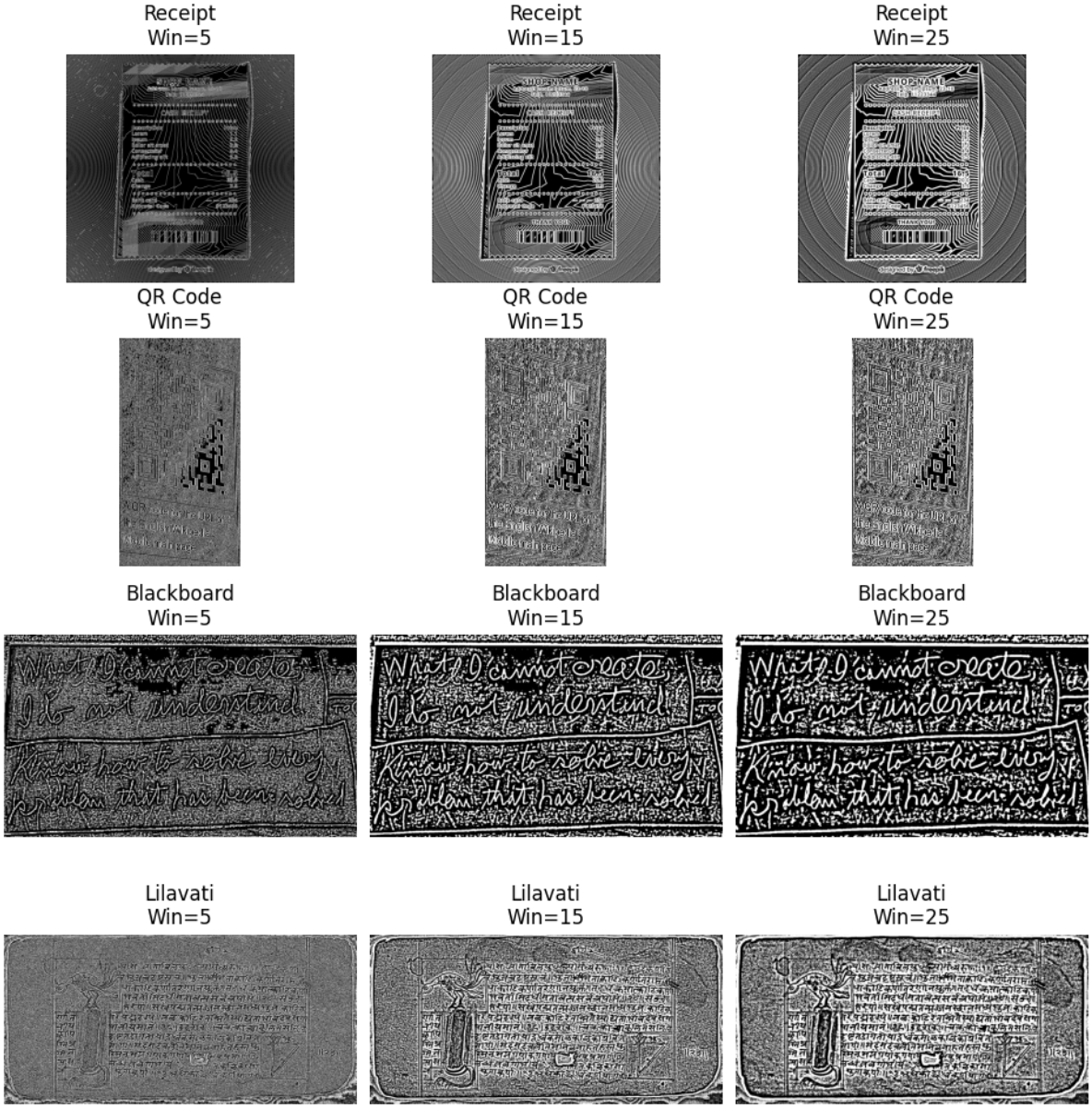


Figure 7: Comparing images for different thresholds

3 Contrast Enhancement

3.1 Linear Contrast Stretching

Function: `myLinearContrastStretch()`
Summary: Simply applying linear contrast by linear scaling the luminance channel of the original image does not render the optimal result. Since out of the 1.4M pixels in the image, around 9000+ pixels lie in the 0.5 to 1.0 pixel range.

Hence, we must mask and ignore these pixels with luminance > 0.5 to get our final contrasted image!

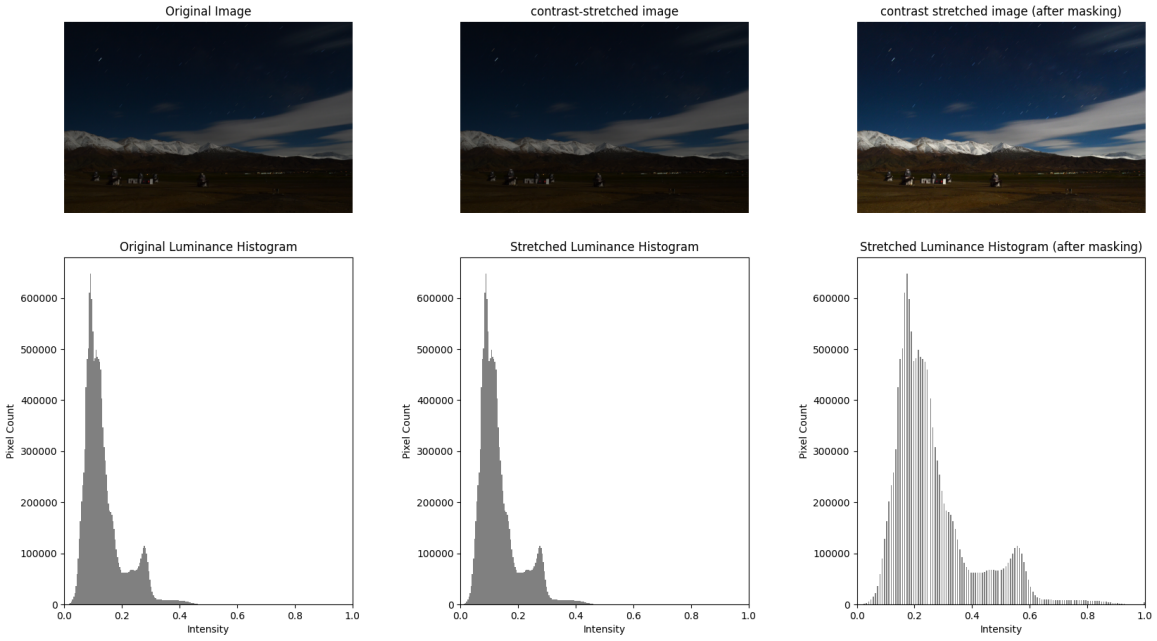


Figure 1: Linear contrast stretching results.

3.2 Histogram Equalization

Function: `myHistEqualize()`
Summary:

Given an image in RGB space, we first convert it to HSV and operate on the luminance component $V(x, y) \in [0, 1]$. A binary mask is defined as

$$M(x,y) = \begin{cases} 1, & V(x,y) \le T, \\ 0, & V(x,y) > T, \end{cases}$$

where T is the luminance threshold (e.g., $T = 0.5$). Only pixels with $M(x, y) = 1$ are equalized.

—

Step 1: Discretization. The masked pixel intensities are quantized into L discrete levels ($L = \text{num\_bins}$):

$$v_q = \text{round}\big(V(x,y) \cdot (L-1)\big), \quad v_q \in \{0, 1, \ldots, L-1\}.$$

—

Step 2: Histogram and CDF. The histogram of quantized values is

$$h(k) = \#\{(x, y) \mid v_q(x, y) = k,\ M(x, y) = 1\}, \quad k = 0, 1, \ldots, L-1.$$

The cumulative distribution function (CDF) is

$$\mathrm{CDF}(k) = \sum_{i=0}^{k} h(i).$$

We normalize it as

$$\hat{F}(k) = \frac{\mathrm{CDF}(k)}{\max_j \mathrm{CDF}(j)}.$$

—

Step 3: Equalization mapping. Each masked intensity v_q is mapped using

$$v'_q = \hat{F}(v_q)\,(L-1).$$

The equalized luminance is then

$$V'(x, y) = \begin{cases} \dfrac{v'_q}{L-1}, & M(x, y) = 1, \\ V(x, y), & M(x, y) = 0. \end{cases}$$

Thus, histogram equalization redistributes the intensity values of masked pixels so that their histogram becomes approximately uniform, enhancing contrast in dark regions while preserving unmasked regions.

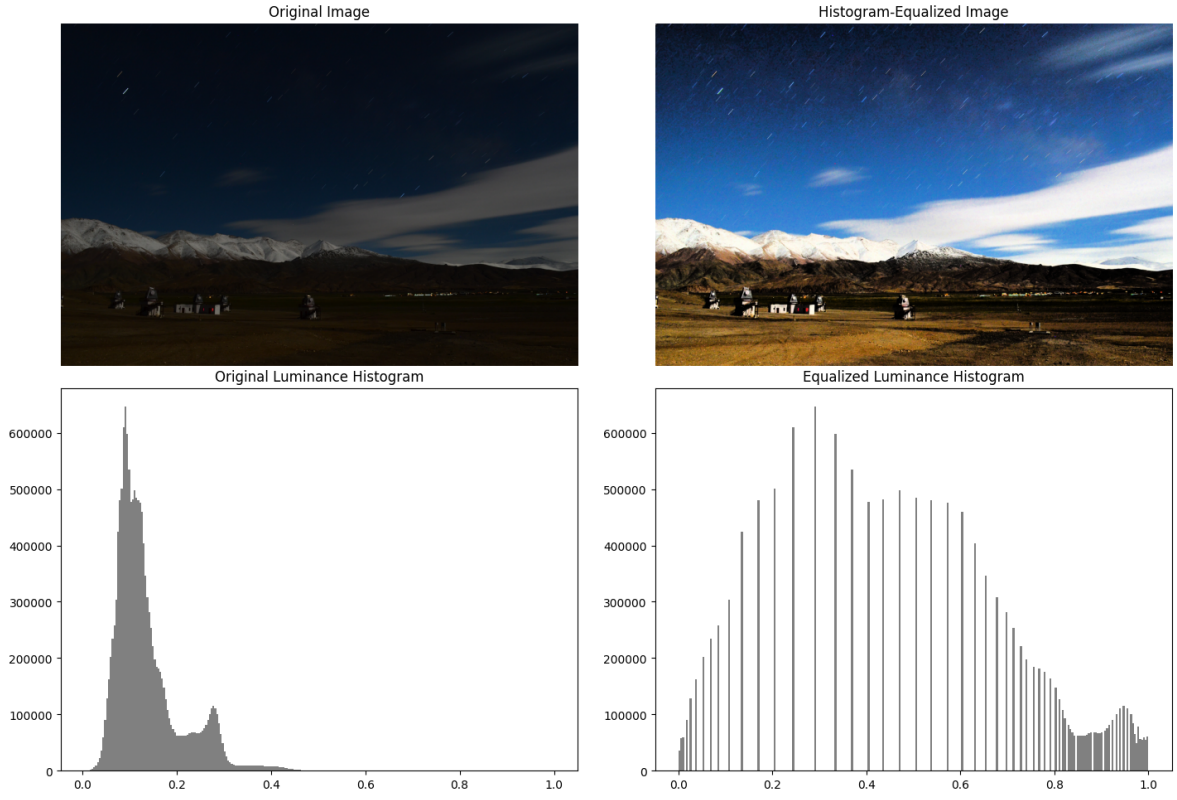


Figure 2: Linear contrast stretching results.

3.3 CLAHE

Function: `myCLAHE()`
Summary:

Given an image in CIELab color space, with luminance $L(x, y) \in [0, 100]$ and chroma channels $a(x, y), b(x, y)$, we apply CLAHE on the luminance channel while preserving chroma.

CLAHE adaptively enhances local contrast by equalizing histograms within small tiles, while limiting contrast amplification using the clip limit parameter γ.

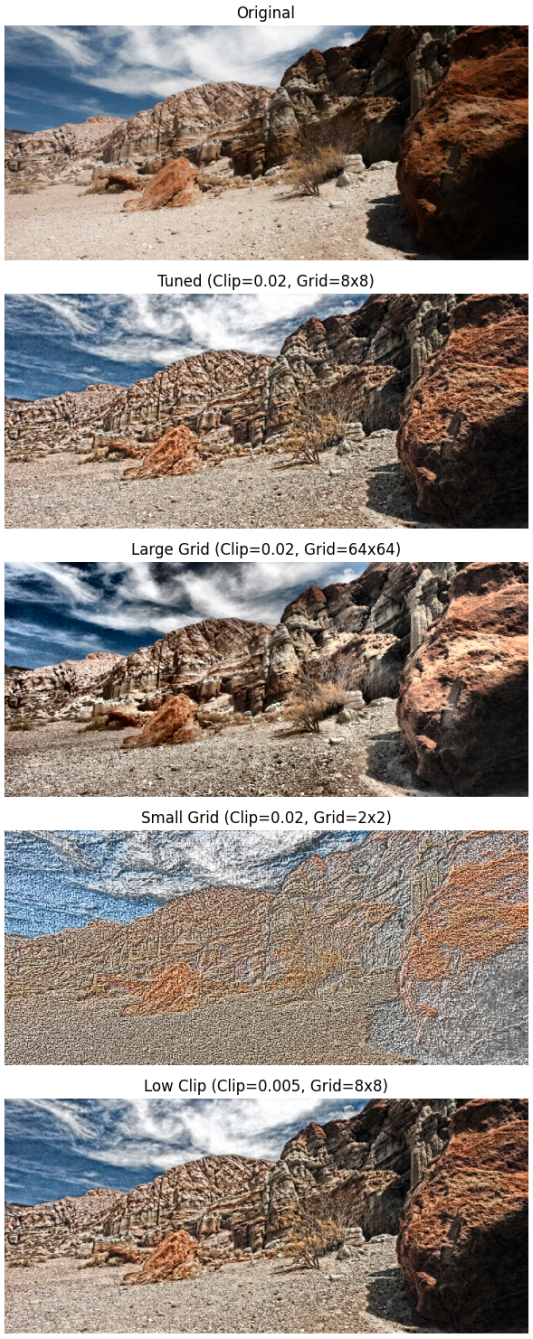


Figure 3: CLAHE results for different grid size and clip

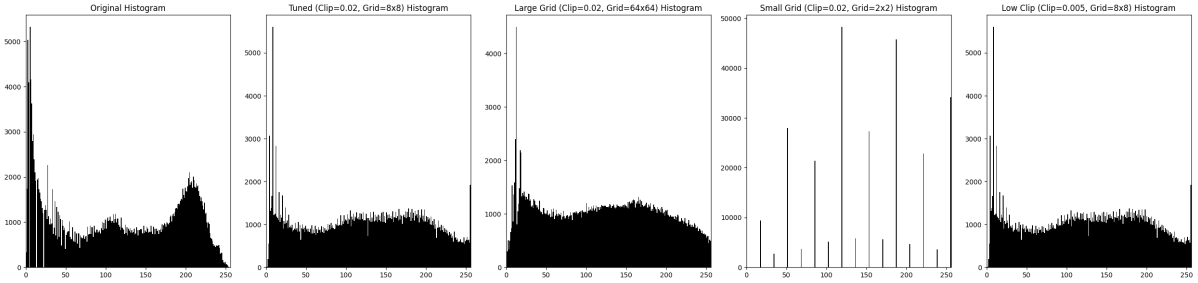


Figure 4: Linear contrast stretching results.

3.4 Histogram Matching

Function: `myHistMatch()`
Histogram Matching:

Given a source image I_s and a reference image I_r, the goal of histogram matching is to transform the intensity distribution of I_s so that it resembles that of I_r. The process is carried out independently on the L, a, and b channels of the Lab color space, while ignoring black background pixels.

—

Step 1: Conversion to Lab color space. The input images are transformed:

$$I_s^{Lab} = \text{RGB2Lab}(I_s), \quad I_r^{Lab} = \text{RGB2Lab}(I_r).$$

—

Step 2: Foreground masking. Background pixels are removed by thresholding the luminance channel:

$$M_s(x,y) = \begin{cases} 1, & L_s(x,y) > 5, \\ 0, & \text{otherwise}, \end{cases} \qquad M_r(x,y) = \begin{cases} 1, & L_r(x,y) > 5, \\ 0, & \text{otherwise}. \end{cases}$$

Only pixels with mask value 1 are considered in subsequent steps.

—

Step 3: Histogram and CDF computation. For each channel $c \in \{L, a, b\}$, compute the histograms:

$$h_s^c(k) = \#\{(x,y) \mid I_s^c(x,y) = k,\ M_s(x,y) = 1\},$$

$$h_r^c(k) = \#\{(x,y) \mid I_r^c(x,y) = k,\ M_r(x,y) = 1\}.$$

The cumulative distribution functions (CDFs) are

$$\text{CDF}_s^c(k) = \sum_{i=0}^{k} h_s^c(i), \quad \text{CDF}_r^c(k) = \sum_{i=0}^{k} h_r^c(i).$$

Both are normalized to $[0, 1]$.

—

Step 4: Mapping function. A mapping is constructed by matching equalized source intensities to the closest reference intensities:

$$T^c(v) = \arg\min_j \left|\text{CDF}_s^c(v) - \text{CDF}_r^c(j)\right|.$$

—

Step 5: Transformation of source image. Each foreground pixel in the source channel is remapped:

$$I_s^{c'}(x,y) = T^c\big(I_s^c(x,y)\big), \quad \text{if } M_s(x,y) = 1.$$

Step 6: Reconstruction. The modified Lab image is

$$I_s^{Lab'} = \big(I_s^{L'},\, I_s^{a'},\, I_s^{b'}\big),$$

and the final RGB image is obtained by

$$I_s^{RGB'} = \text{Lab2RGB}(I_s^{Lab'}).$$

Thus, histogram matching aligns the intensity distributions of the source image to the reference image, producing a perceptually closer appearance while ignoring background pixels.

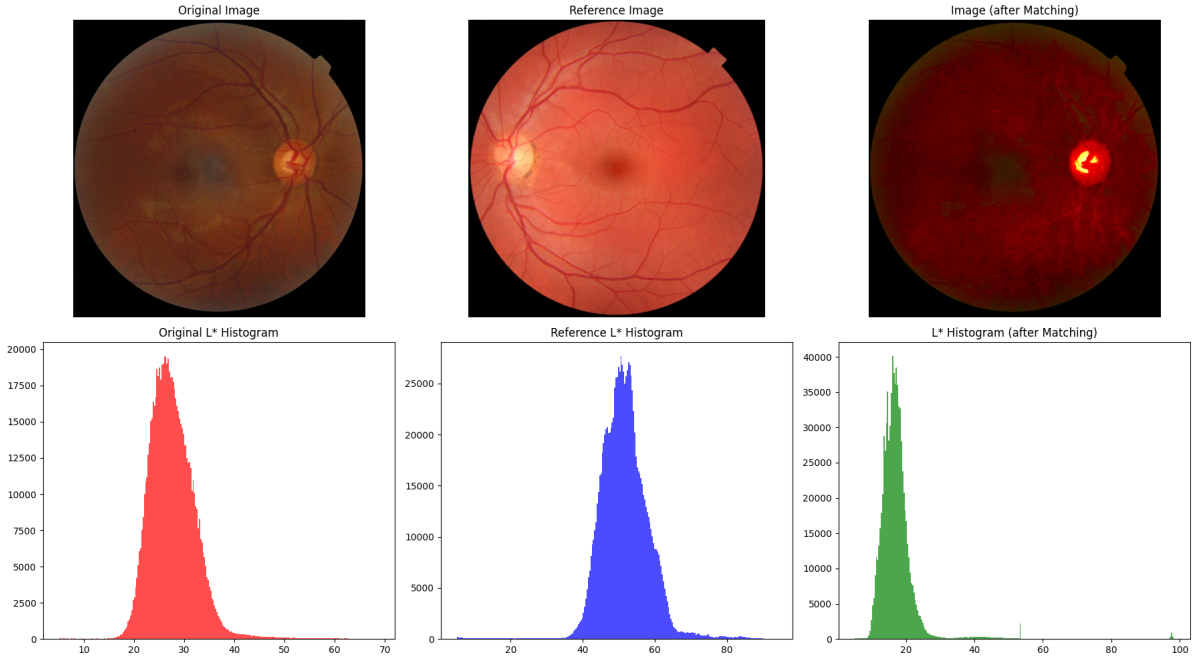


Figure 5: CLAHE results for different grid size and clip