AI6121 Computer Vision Assignment 1

Chen Yongquan (G2002341D)
Nanyang Technological University, School of Computer Science and Engineering

Contents

Introduction

Histogram equalization is non-linear contrast enhancement technique widely used in scientific imaging to improve the details of image features. For instance, microscopy images of biological cells can be processed to surface subcellular components that are easily missed due to poor contrast. In the medical sector, histogram equalization is done on X-ray images to better define regions of interests like white spots around the lung cavity. In some cases, tumors may also appear on X-ray images as poorly contrasted white lumps. Outside of the scientific community, histogram equalization can be used to improve contrast in overexposed or underexposed photos, though it may result in unrealistic representations of the original scene as lighting ambiance is distorted.

The goal of histogram equalization is to produce an output image with a flat histogram by redistributing pixel levels in the original image across the entire intensity range. In effect, an original image with poor contrast and pixel values occupying a narrow range of values on the intensity scale would be stretched across the full dynamic range, thereby improving contrast between pixels.

There are a few parameters we can set for histogram equalization:

1. L, the number of bins
2. r_min, the minimum range value that the transform function maps to
3. r_max, the maximum range value that the transform function maps to

Additionally, we can set the color space in which we perform the equalization. Possible options normally separate out the luminance channel from the color channels as colors can be warped out of their original hue if we equalize each color channel individually. Examples include the CIE color spaces like CIELAB and CIELUV, YUV color spaces like Y'CbCr, or HSL. In our report, we opt for CIELAB by default and present a comparison with Y'CbCr.

The main steps for vanilla histogram equalization are as follows:

1. Calculate the global histogram for input image. With L bins, the number of pixels in each bin is n_k, where $k = [0, L-1]$.
2. Normalize the histogram to $[0,1]$ as a probability histogram.

$$MN = width \times height = total\ no. of\ pixels$$
$$p_r(r_k) = \frac{n_k}{MN}$$

3. Calculate the cumulative probability function corresponding to p_r:

$$cdf_r(r_k) = \sum_{j=0}^{k} p_r\left(r_j\right)$$

4. Map every pixel in the input to the output image with the transformation function given by:

$$s_k = (L-1)cdf_r(r_k)$$

Implementation

The HE algorithm for this reported is programmed in Python in the attached Jupyter notebook. The snippets of code below demonstrate the main steps of the HE algorithm:

1. Probability histogram

```
def histogram(im, bins):
  freqHist = [0] * bins
  probHist = [0] * bins
  kdeBin = [0] * bins
  imData = im.reshape(im.shape[0] * im.shape[1])
  pixelCount = len(imData)
  binSize = 256/bins
  for pixelValue in imData:
    freqHist[math.floor(pixelValue / binSize)] += 1
  for i in range(bins):
    probHist[i] = freqHist[i] / pixelCount

  intensities = list(range(256))
  kde = stats.gaussian_kde(imData)
  kde = kde(intensities)
  for i in range(256):
    kdeBin[math.floor(i / binSize)] += kde[i]
  return freqHist, probHist, kdeBin
```

2. Cumulative distribution function

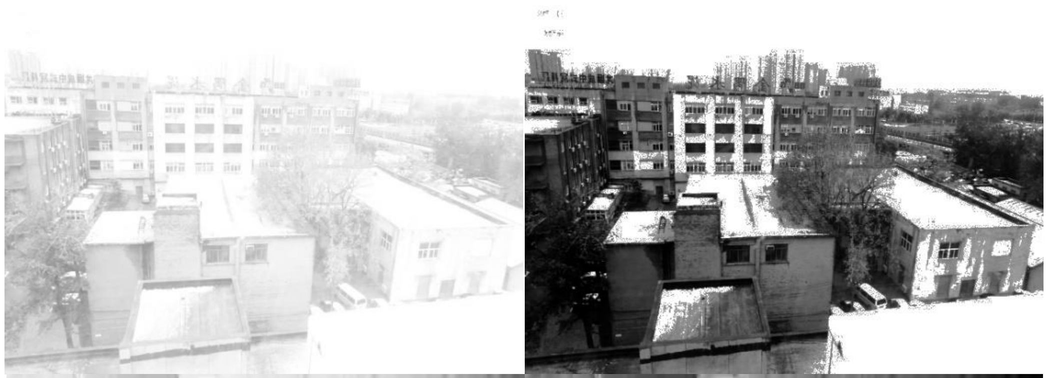
```
def cdf(probHist):
  cdf = []
  cumul = 0
  for i in probHist:
    cumul += i
    cdf.append(cumul)
  return cdf
```

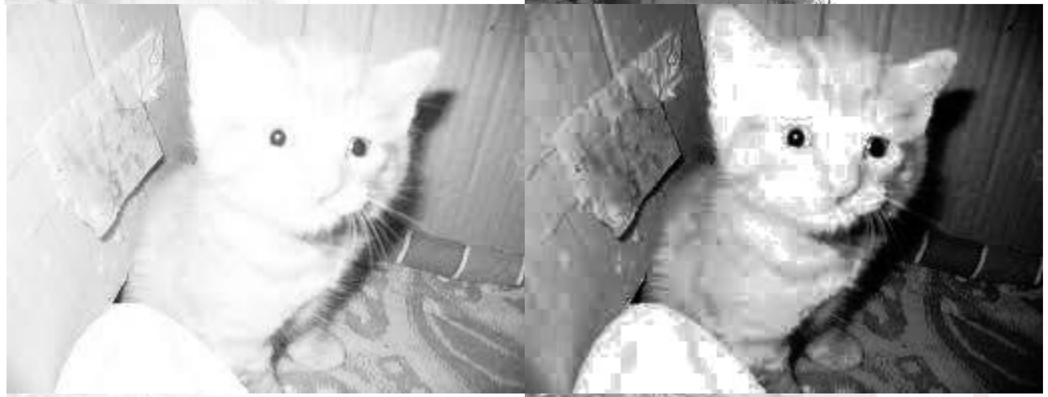
3. Transformation mapping

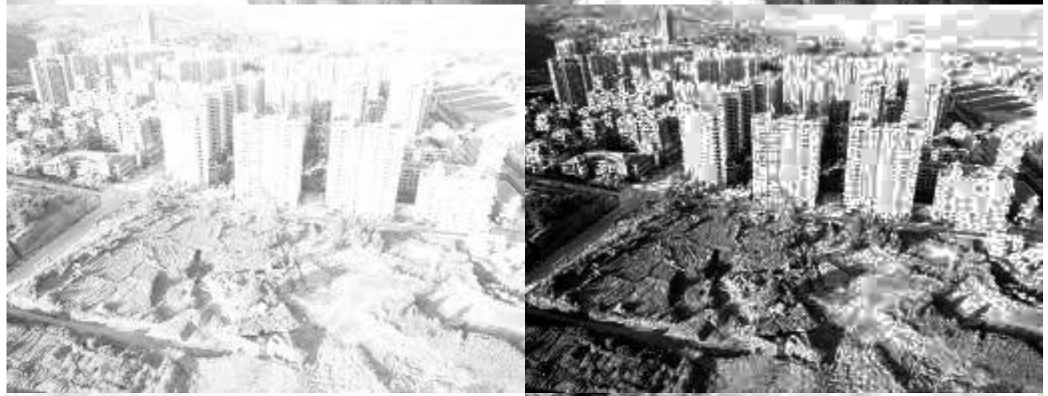
```
def histEq(im, cdf, bins, r_min, r_max):
  r = r_max - r_min
  binSize = 256/bins
  outIm = np.zeros((im.shape[0], im.shape[1]), np.uint8)
  transform = [0] * len(cdf)
  for i in range(len(cdf)):
    transform[i] = int(round((cdf[i] * r) + r_min))
  for i in range(im.shape[0]):
    for j in range(im.shape[1]):
      outIm[i][j] = transform[math.floor(im[i][j] / binSize)]
  return outIm, transform
```

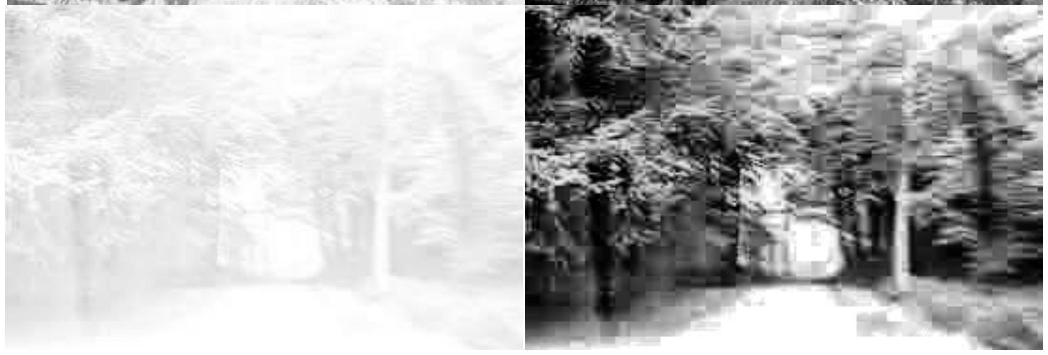
Results

The histogram equalization in this report was performed on the lightness L^* channel of the CIELAB color space by default. The $L*$ channels for the original samples are shown on the left while the enhanced samples are shown on the right. Equalization was done using 256 bins for the output range of $[0,255]$.

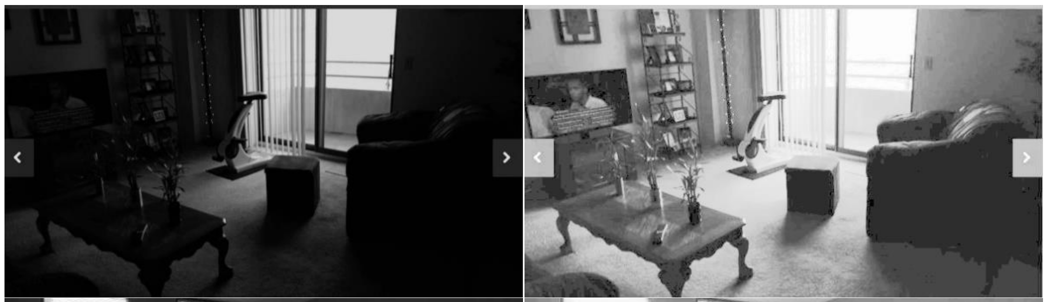

1.


2.

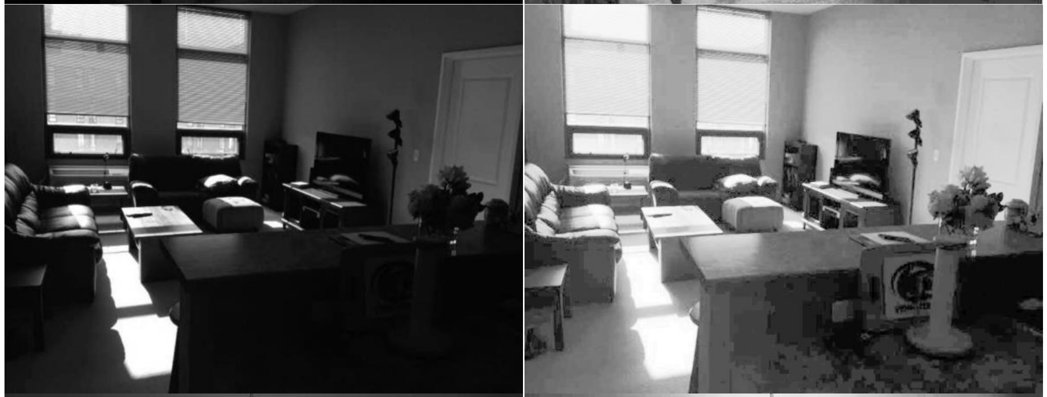

3.


4.

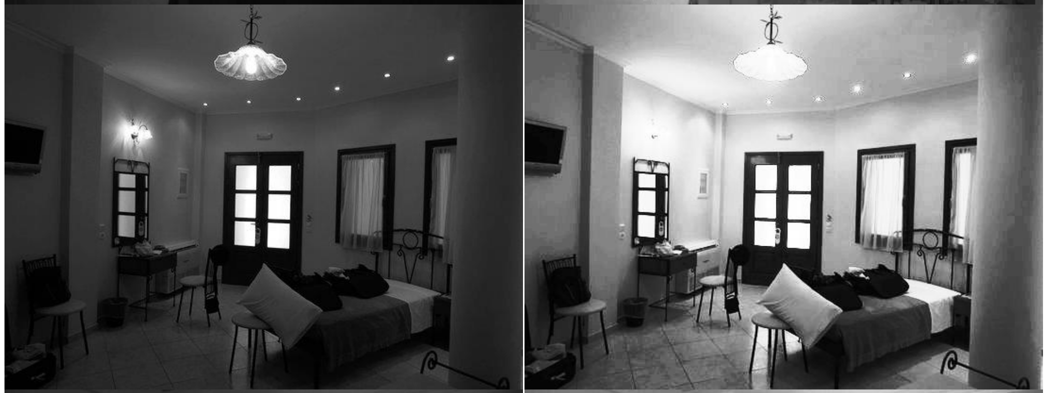
5.



6.



7.



8.

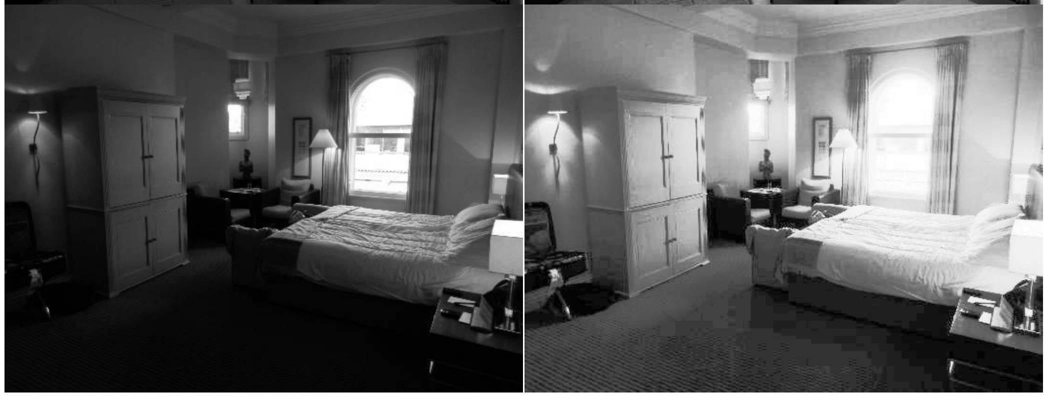


Figure 1. Comparison of L channel for original and enhanced samples*

We can see that details initially obscured by poor contrast are made clearer after processing. For instance, in sample 3, we can make out individual levels and windows on the buildings in the enhanced version while they were hard to differentiate in the original due to narrow contrast in the higher intensity range. Similarly, in sample 5 we can make out the person on the television in the enhanced sample, while in the original the feature was obscured by poor contrast in the darker ranges.

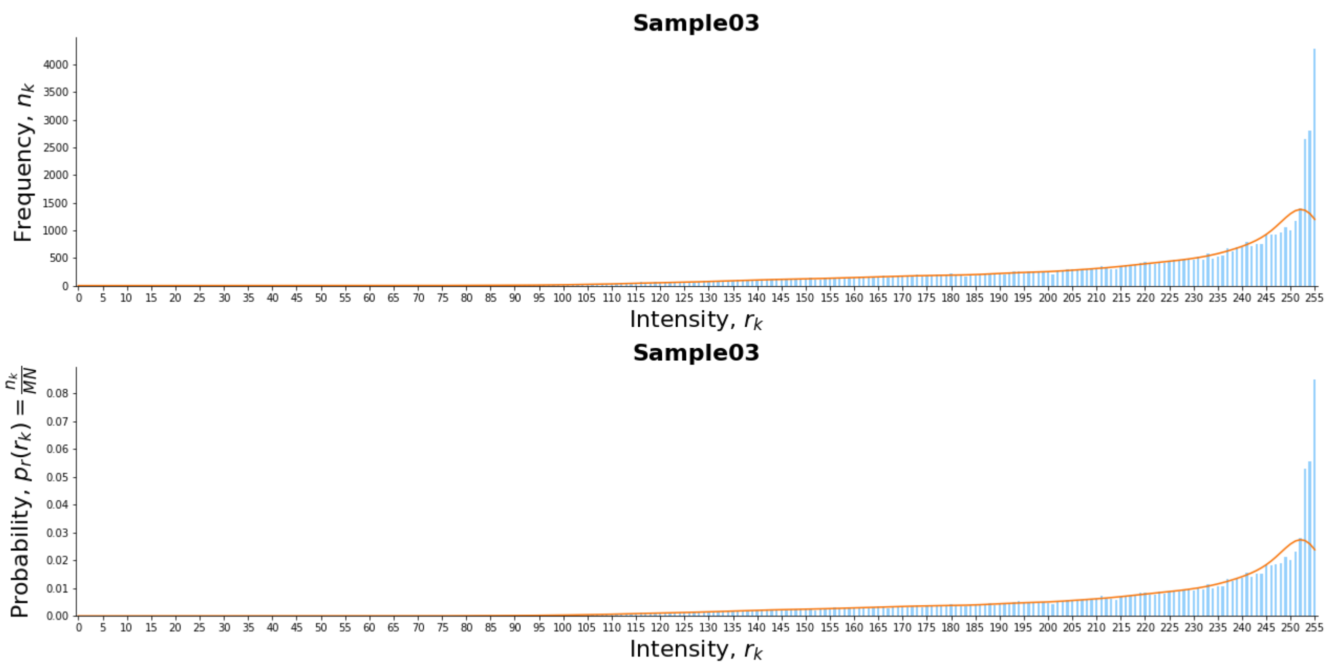


Figure 2. Frequency and probability histograms for original sample image 3

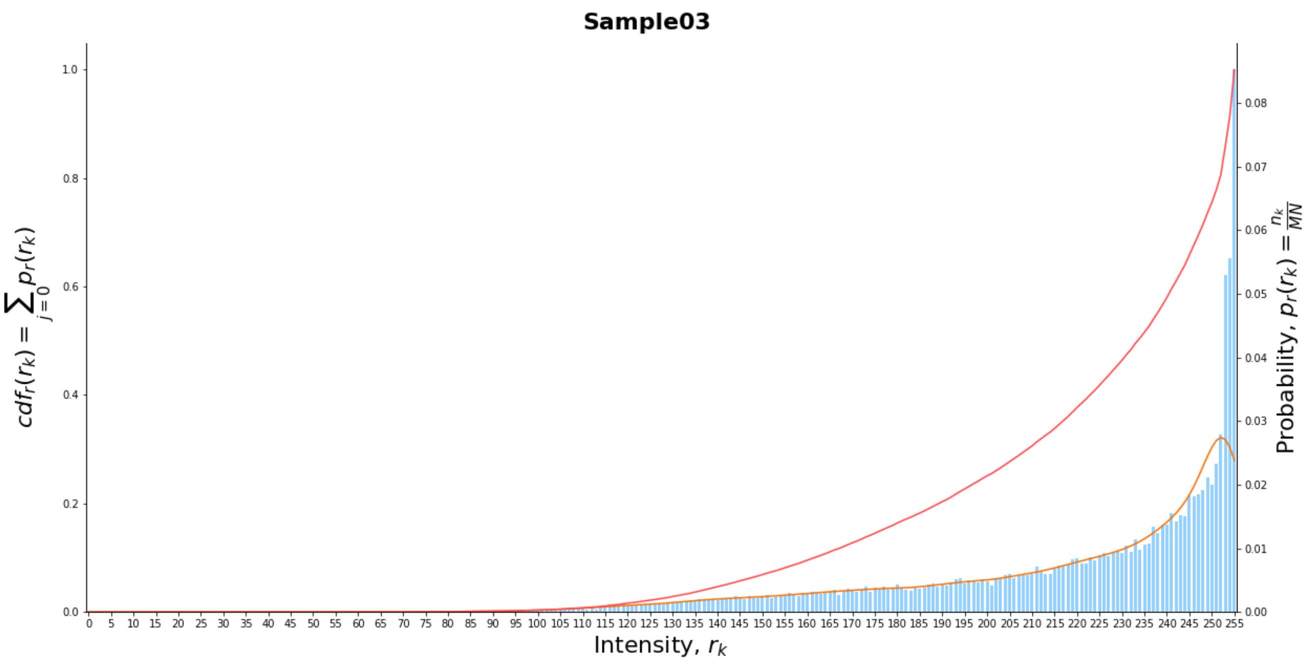


Figure 3. Cumulative distribution function for original sample image 3.
Red line denotes cdf. Orange line is the kernel density estimation.
(Note: cdf figures incorrectly show r_j as r_k in the summation equation)

In Figure 2 and Figure 3, we can see that in the original sample image 3, pixel values were increasingly concentrated in the brighter range of the intensity scale, shown by the histogram bars occupying only the latter region of the graphs and the increasing gradient on the cdf curve. The graphs for the frequency and probability histograms are essentially the same except that they are on different y-axis scales; the probability histogram being normalized to $[0,1]$.

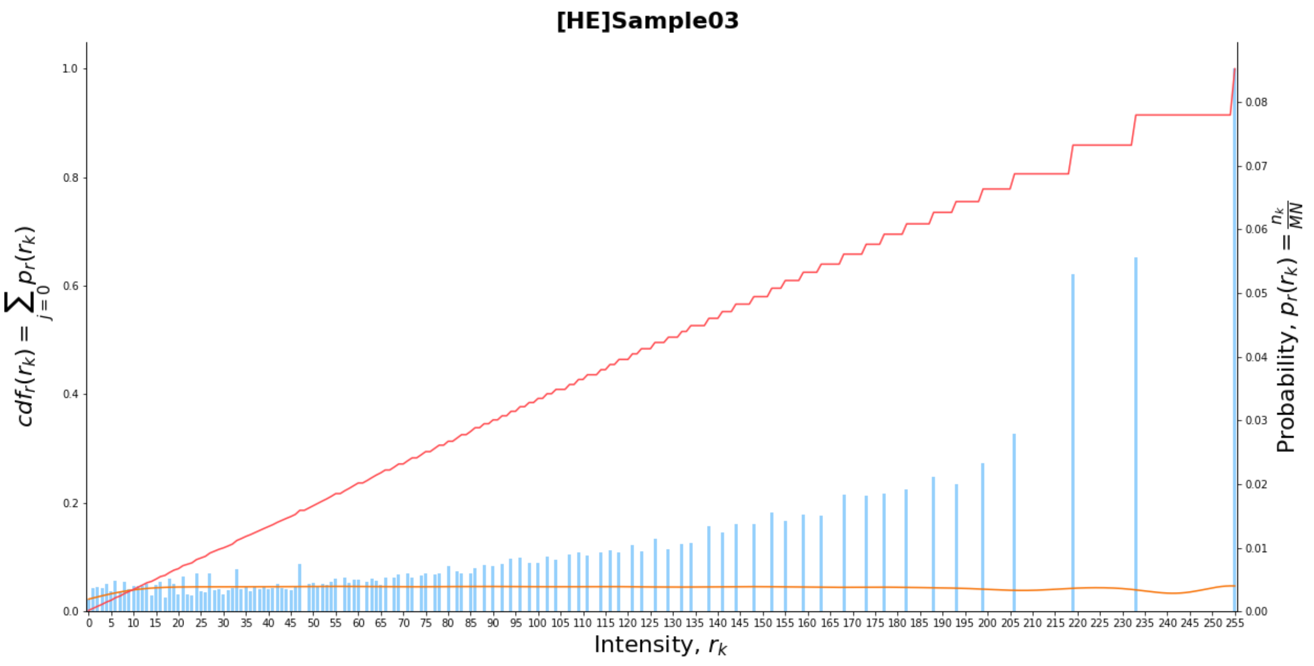


Figure 4. Cumulative distribution function for enhanced sample image 3

Based on Figure 4, we can interpret histogram equalization as a transformation applied on the cdf of the image, where we get the new pixel value by multiplying the original pixel value with its respective cdf value. By transforming the curve into a linear $y = x$ line, cumulative pixel count increases by the same gradient per bin, which essentially means a flat histogram. On a continuous range of intensity values, the cdf value is the integral of the probability histogram with respect to r. This would give us a continuous transformation for the entire range of pixel values. However, in digital imaging, pixels are usually represented as discrete 8-bit values or 12-bit and 14-bit values in RAW images. Thus, there would be gradation steps on the resultant cdf curve when “stretching” the histogram and the final histogram would not be completely flat. Using a higher bit depth can alleviate the issue a little as pixels are split more evenly across a wider range of values and there would be less bins with a disproportionately large number of pixels.

From the figure, we can basically see that pixel bins have been “stretched” from the brighter region of the intensity scale to the darker region and there are approximately equal number of dark pixels as bright pixels, which translates to better contrast in the output image.

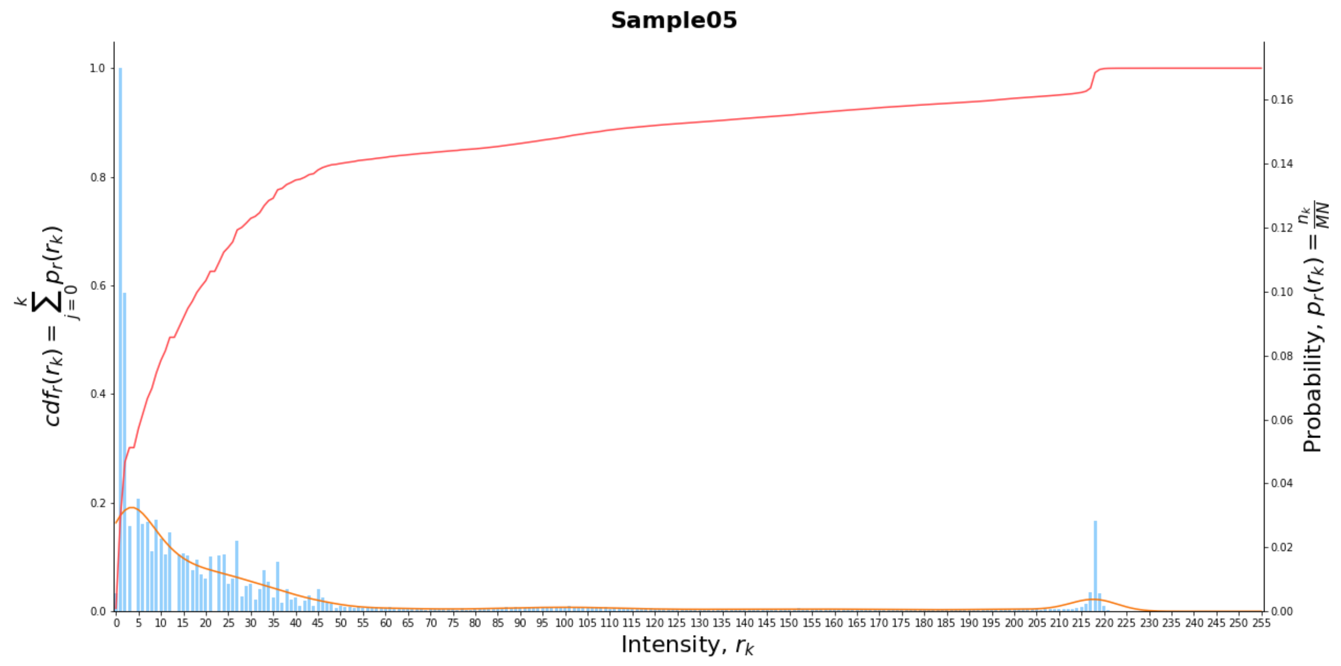


Figure 5. Cumulative distribution function for original sample image 5

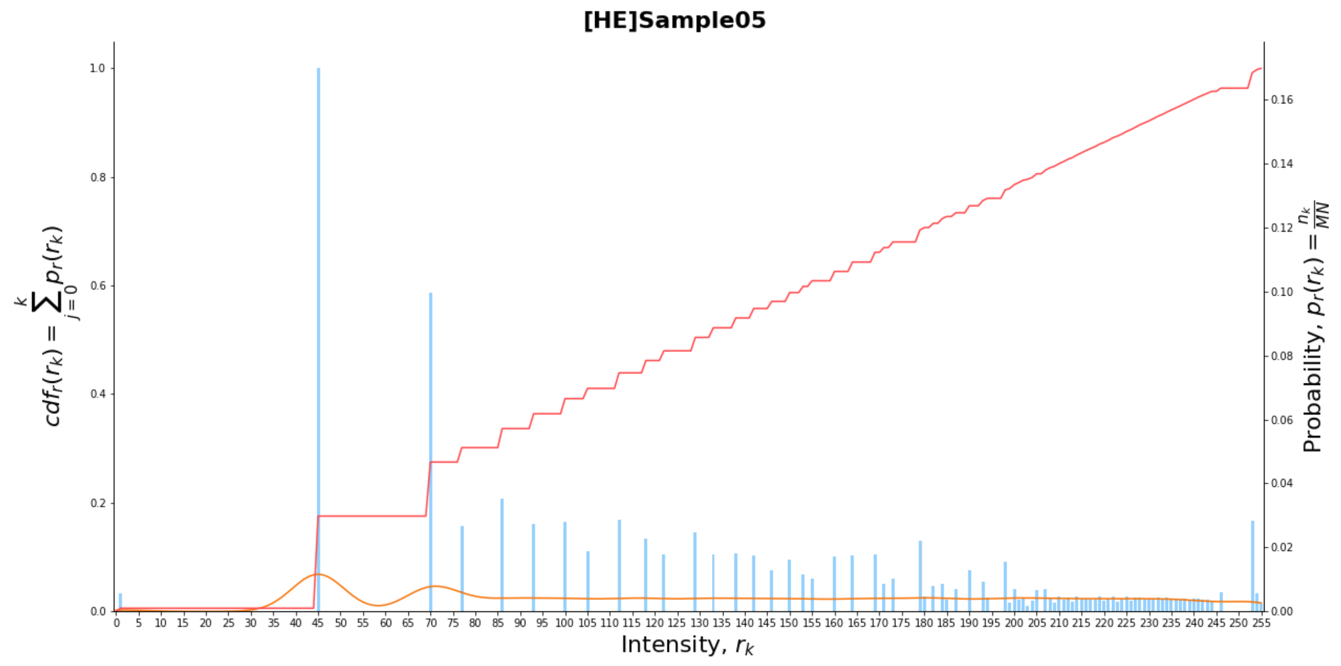


Figure 6. Cumulative distribution function for enhanced sample image 5

Likewise, for sample image 5, we can see that pixel values are “stretched” from the darker regions to the brighter regions. Because discrete bins cannot be split into smaller continuous intensity values, we see the large “step” at the beginning of the cdf curve, but it still approximates the linear $y = x$ line.

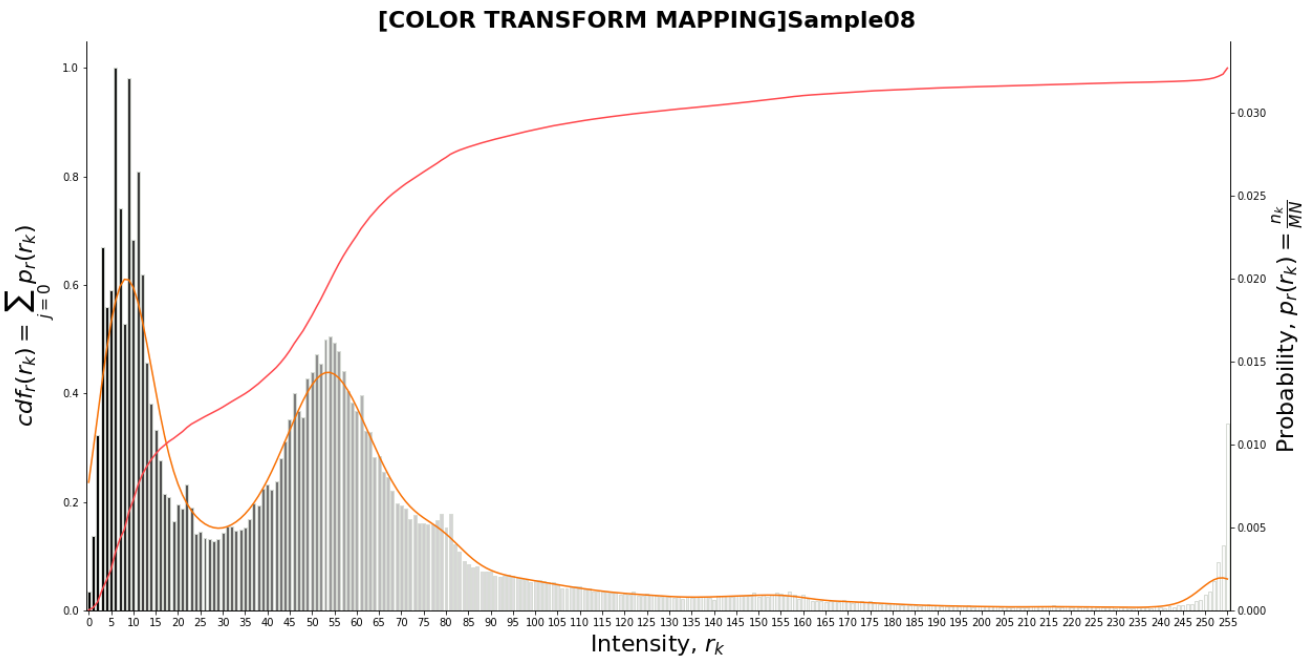


Figure 7. Cumulative distribution function for original sample 8. Bars are colored with their transformed value.

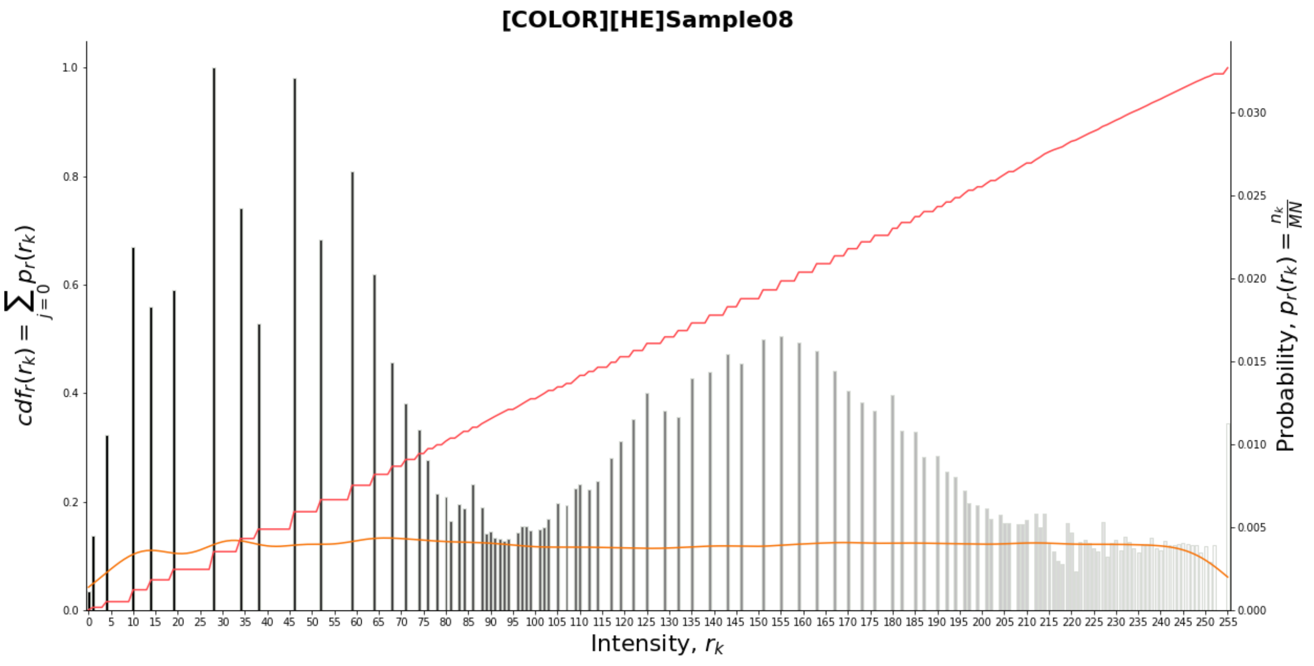


Figure 8. Cumulative distribution function for enhanced sample 8.

From Figure 7, we can see that histogram equalization transforms pixel values progressively and successive bins are only reassigned equal or larger bin values. Thus, a brighter pixel would not be suddenly become darker than another originally darker pixel in the output image, and vice versa. We can however see that bins can be binned together with other bins in order to achieve a linear gradient as shown in the latter part of Figure 7, where bins above 100 are compressed to above 210 in Figure 8. This means that we are losing some image information whose pixels lie within that range.

Comparison on different bin sizes

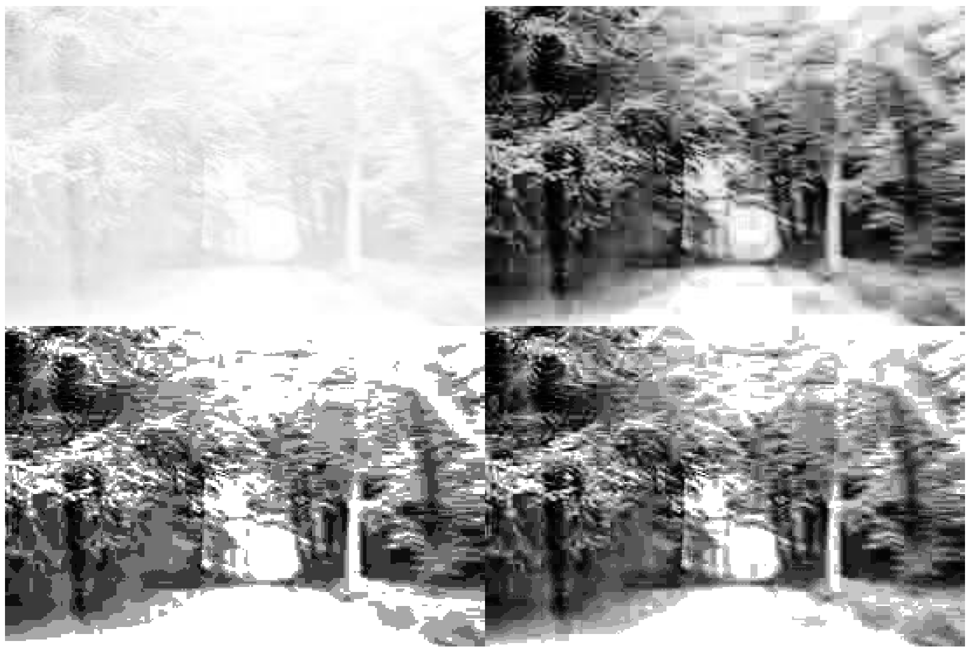


Figure 9. Clockwise from top left: original, 64 bins, 32 bins, 16 bins

Testing histogram equalization with different bins sizes for along the intensity range $[0,255]$ shows that output image demonstrates increasing banding gradients as there are lesser intensity levels to map to than the original range and more pixels are binned together as a result.

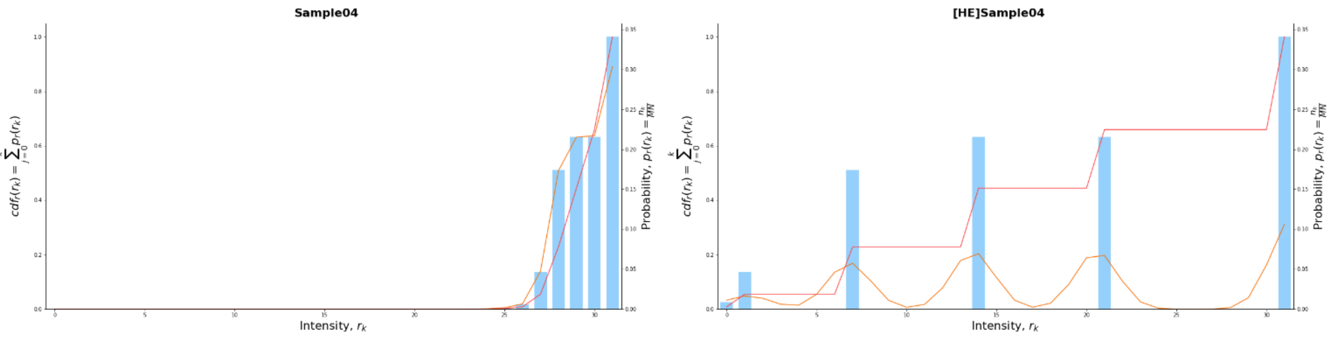


Figure 10. Cumulative distribution function for original and enhanced sample 4 using 32 bins

Looking at the cumulative distribution function plots for sample 4 in Figure 10, we see the “steps” becoming more pronounced than in the plots when 256 bins are used. This in turn translates to larger gaps on intensities between the final output bins. Visually, we see these as distinct bands and sharp gradients of color in the final output. This as explained previously is due to the discrete gradations of intensities of digital images and as suggested can be lessened by using and equalizing in higher bit-depth.

Comparison on different output ranges

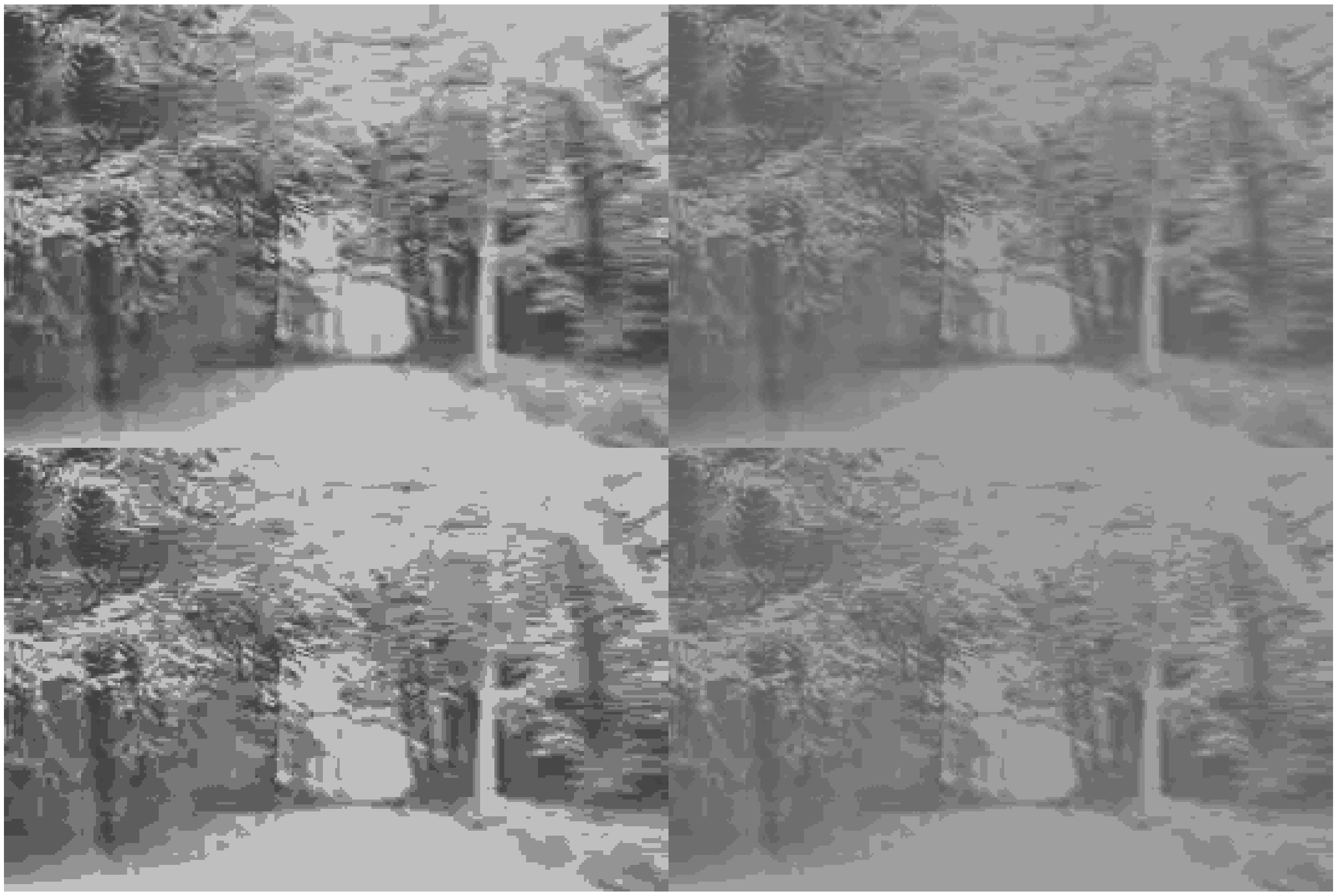


Figure 11. 1st row: Output image for 64 bins ranging 64-191 and 96-159.
2nd row: Output image for 32 bins ranging 64-191 and 96-159

From Figure 11, by limiting the output dynamic range to 128 and 64 respectively, we see that limiting the output range limits the contrast enhancement as a result because pixel values are "stretched" or "compressed" along a narrower range than the full 8-bit range. This can prove to be detrimental for the original image if the original histogram already spans across the entire 8-bit range and compressing the dynamic range can instead reduce contrast in some areas of the image. Meanwhile, we still see the effects of more pronounced banding and contouring as bin sizes decreases from 64 to 32 bins.

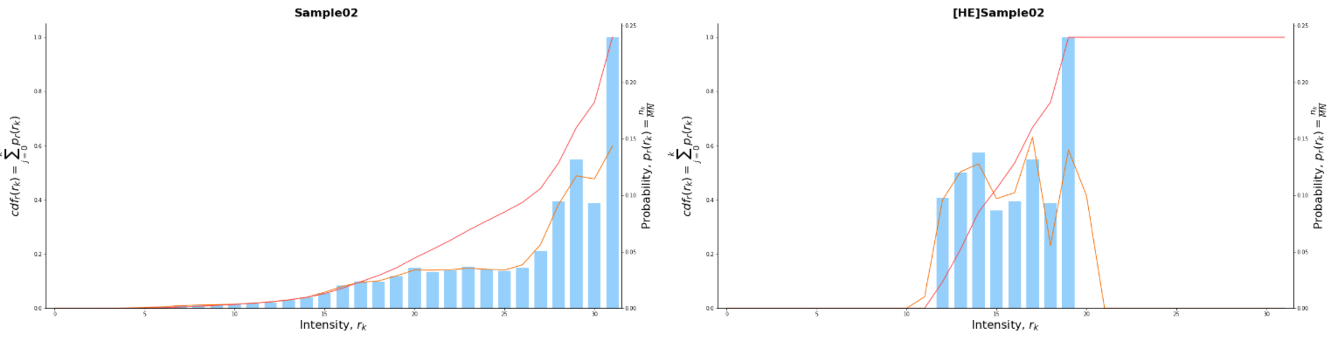


Figure 12. Cumulative distribution function for original and enhanced sample 2, using 32 bins, outputted to 96-159

In Figure 12, we that equalizing to a narrower range of intensities than the original will result in more bins being merged together resulting in loss of visual information. This is apparent in Figure 11 where the smooth gradations near the top left corner in the original have either become two shades of grey bands in the 128 range output or an entire grey patch in the 64 range one.

The above comparisons on bin sizes and output range lead us to the trivial conclusion to always equalize along the full range of intensities and use as many bins as intensity levels possible. This is inconsequential for most users but should be noted when equalizing higher bit depth images.

Comparison on different color space

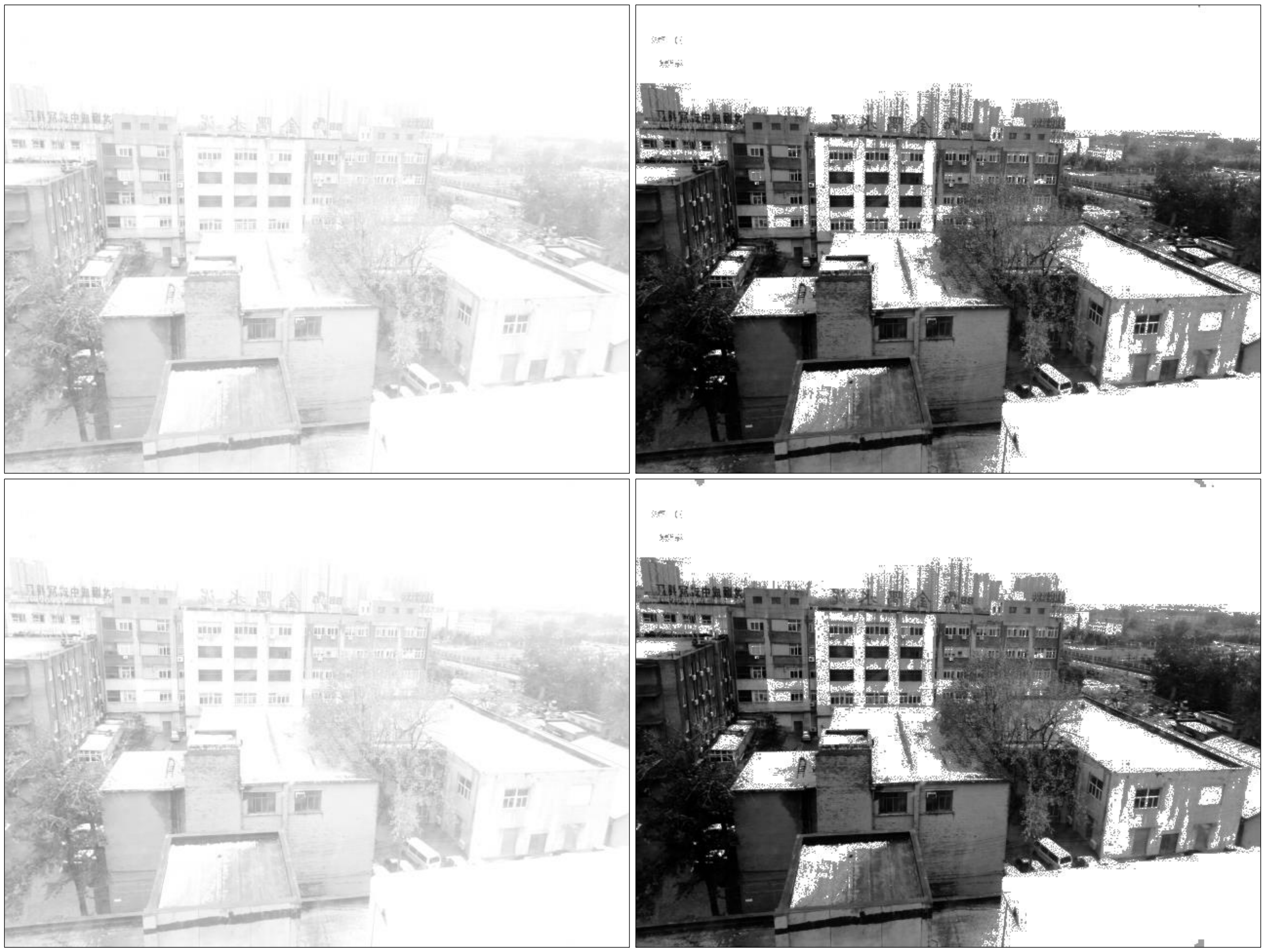


Figure 13. 1st row: CIELAB L, CIELAB equalized L*. 2nd row: Y'CbCr luma, Y'CbCr luma equalized*

Both CIELAB and Y'CbCr separate their luminance channels from color channels. However, CIELAB's *L** channel is a true achromatic luminance value while the Y' luma channel is influenced by chromaticity at high chroma levels. From Figure 13, by the naked eye, we cannot detect any obvious difference between the *L** and luma channel images, except a slight difference in contrast when overlaid on top of the other. However, histogram equalization can in some cases amplify artifacts and flaws in the original image, thus in the equalized luma image we see a triangular pixel block at the bottom right $(580, 475)$ which isn't present in the equalized $L*$ image. These artifacts then appear in the final output image when channels are merged back with their color channels. The offending pixels have a value of $(255, 254, 255)$ in the original RGB image, $(255, 255, 255)$ in the $L*$ image and $(254, 254, 254)$ in the luma image. Thus, at first glance both CIELAB's and Y'CbCr's luminance channels may not vary much but histogram equalization may blow up any subtle differences unobservable to the human eye in the final output.

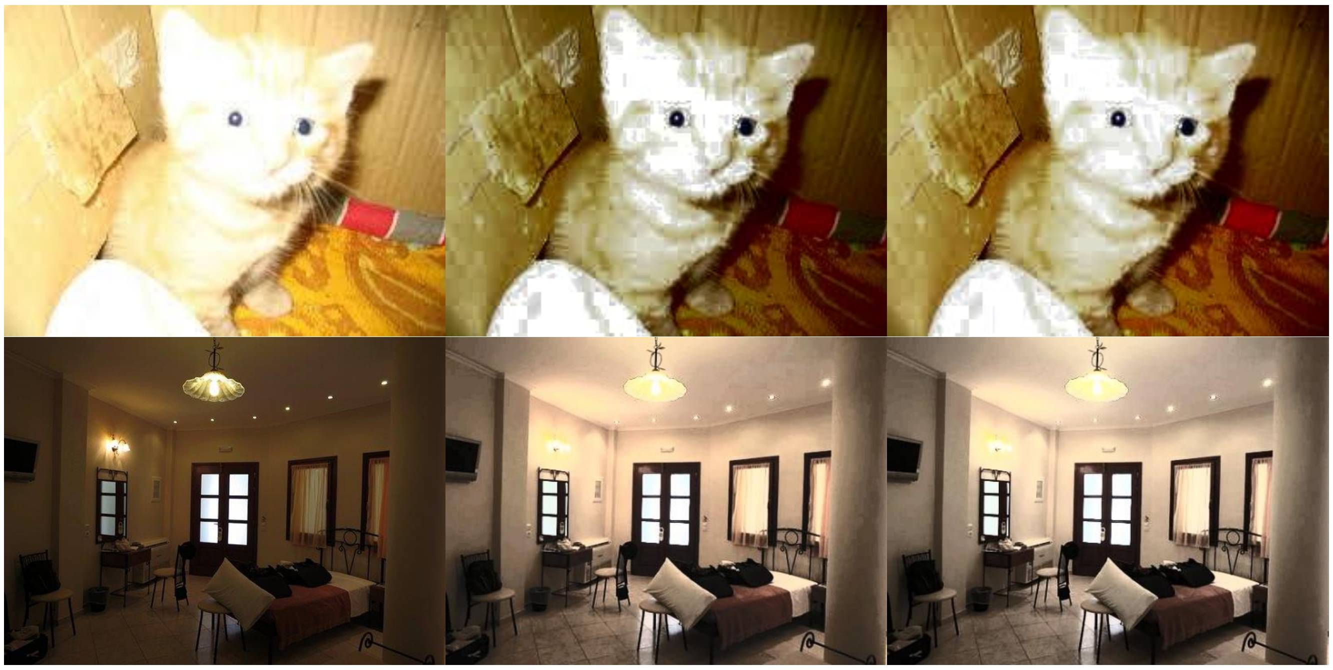


Figure 14. 1st row: Original sample 2, CIELAB equalized, Y'CbCr equalized.
2nd row: Original sample 7, CIELAB equalized, Y'CbCr equalized.

Additionally, equalizing the $L*$ or luma channels may eventually present obvious differences when they are merged back into the color channels and converted back into RGB. Figure 14 shows the color differences between images equalized in CIELAB and Y'CbCr color space. Although the final output image looks more saturated in when equalized in Y'CbCr than CIELAB for sample 2, the reverse is true for sample 7. Such differences should be noted by users when performing equalization on color images when deciding which color space to use, or just perform hue and saturation adjustment separately after luminance equalization.

Discussion

Advantages

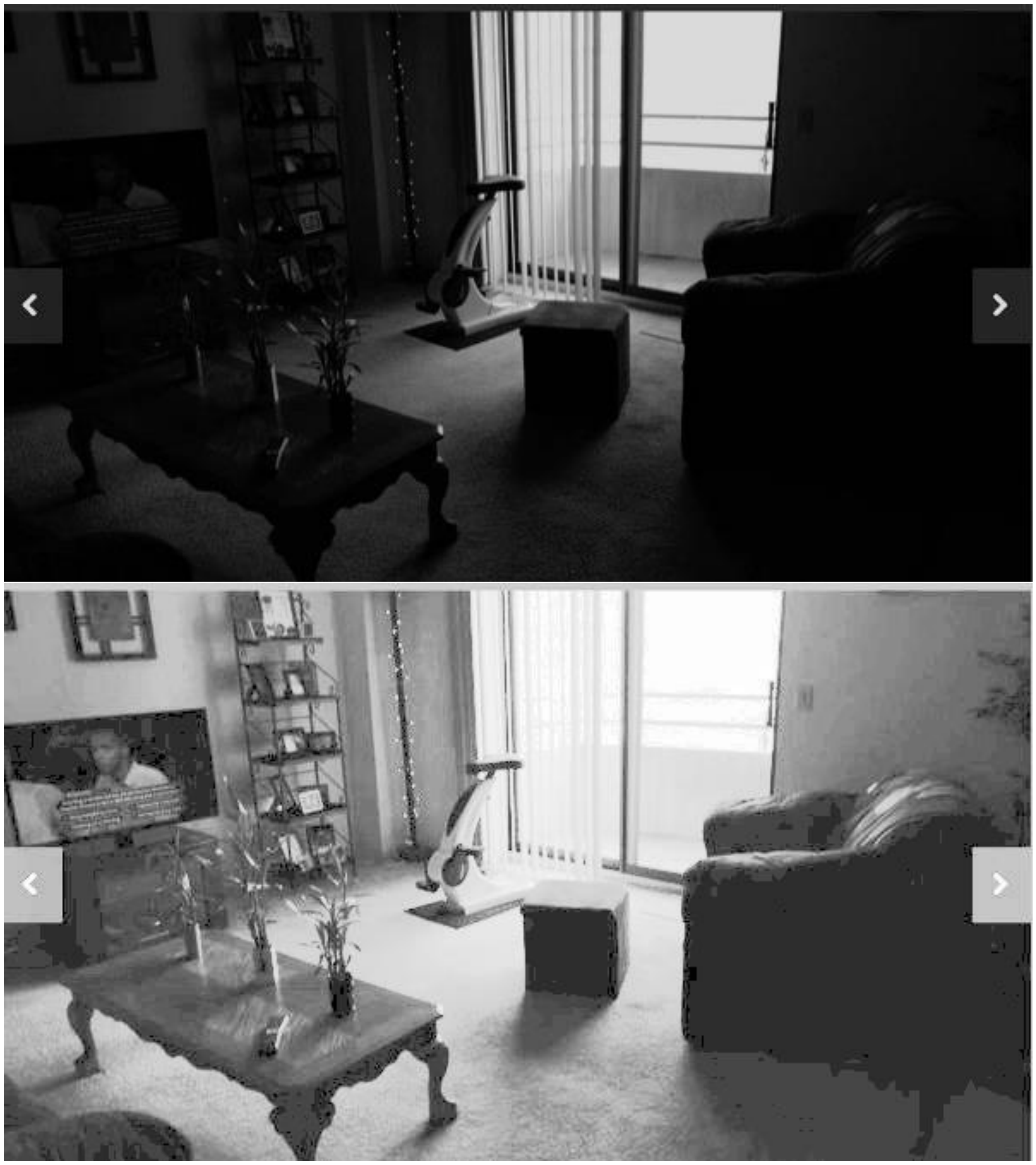


Figure 15. Full-sized original and enhanced sample 5

1. At first glance, histogram equalization has brightened up the entire scene in sample 5 which allows us to easily identify the picture frames on the display stand at the back and the program showing on the television. By enhancing contrast, image features can be more easily detected.

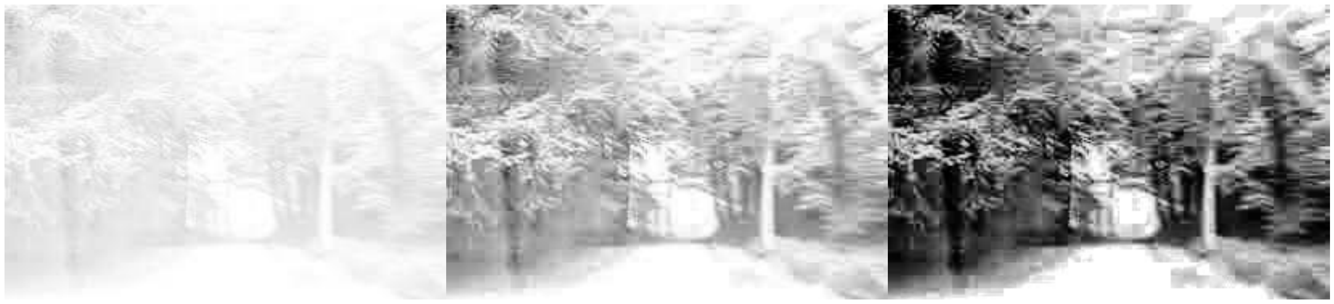
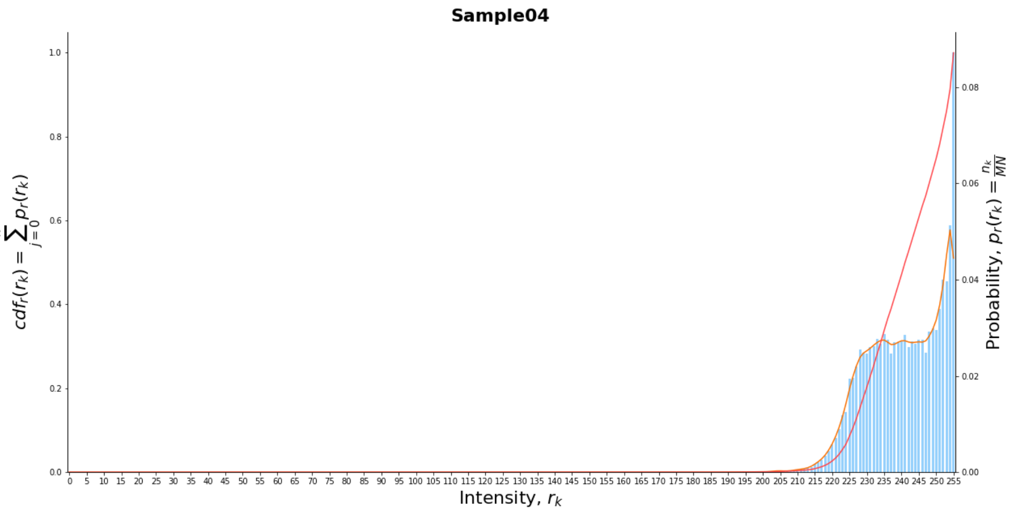
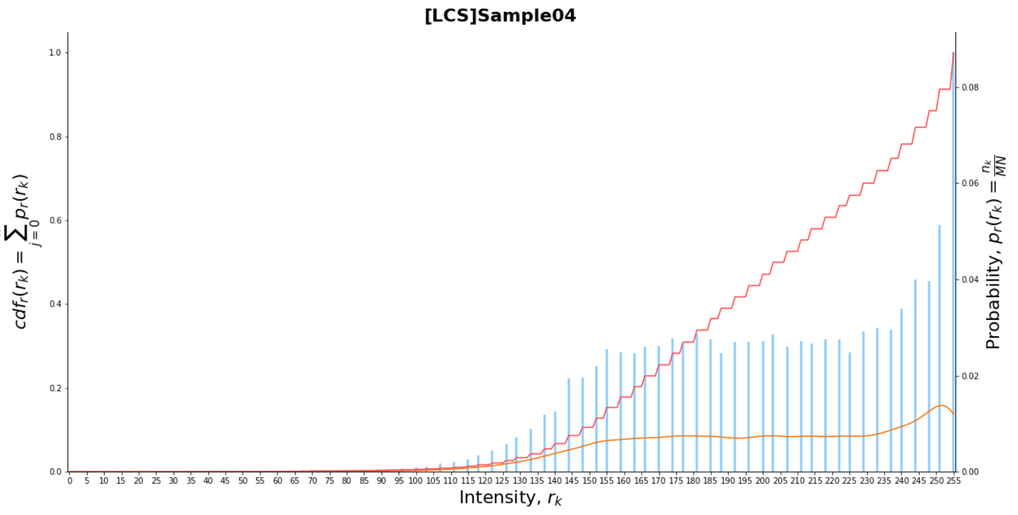


Figure 16. From left: Original sample 4, linearly contrast stretched, histogram equalized

2. Histogram equalization can give better results than linear contrast stretching when the original image contains outlier pixel values or already spans a large part of the intensity range. Even when such limitations are not present, histogram equalization can still provide better results than linear contrast stretching as shown in Figure 16. In Figure 17, the histogram doesn't look to be completely stretched to intensity 0, but actually there are 3 pixels with value 186 in the original image at $(2, 25)$, $(3, 14)$ and $(31, 2)$ which are stretched to 0 in the final output. The histogram equalized image is not affected by such "outliers" and they are binned together with other bins to stretch the histogram further, giving better contrast.





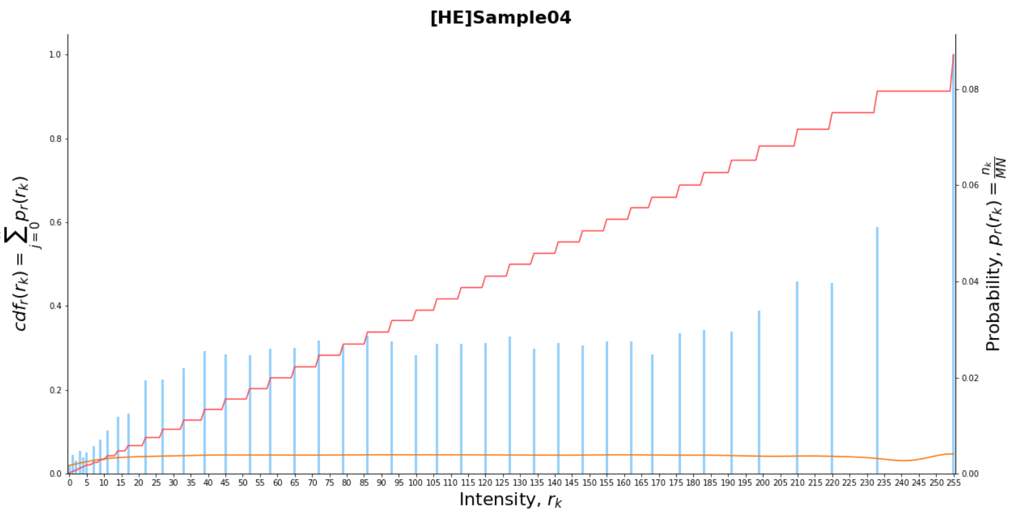


Figure 17. Cumulative distribution function for sample 4.
From top on previous page: Original, linear contrast stretching, histogram equalization

Disadvantages

1. Histogram equalization is not without its cons, and one of the more obvious one is how it amplifies artifacts present in the original images such as noise from the capture device or blocks due to image compression. For instance, in Figure 15, we see pixelated blocks around the sofa in the enhanced image, while in Figure 18 we see similar blocks on the counter in the foreground as JPEG compression did not preserve the gradual gradients in the original dark region. Meanwhile, the equalized sample 1 in Figure 18 presents noisy artifacts which were not obvious in the original image. Buildings in the distant background are also wrongly equalized to grey levels resulting in them misrepresented as being a part of the building structure with the signs. It should also be noted that vanilla histogram equalization cannot recover visual information that are not available in the original image which explains why there are large white patches in the foreground.

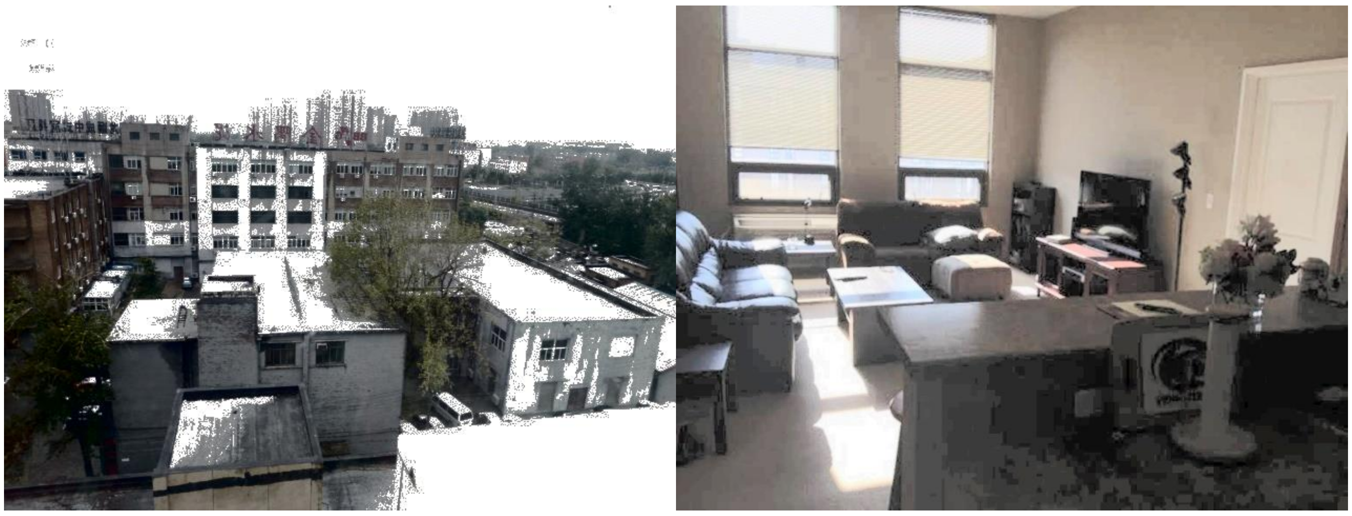


Figure 18. Histogram equalized samples 1 and 6 merged back into color channels

The reason for pixelated blocks in the final output, as explained is because such blocks are already present in the original image, albeit with single level intensity variations that render them unobservable by the human eye. They can be a result of lossy compression like JPEG compression but can also be due to device limitations when capturing scenes with high dynamic range like sunsets. In such cases, when the natural gradients of the sunrays are represented with standard dynamic range of 8-bit values, banding may be apparent in captured images. Such artifacts may then be further exaggerated by the equalization process as individual band intensities are stretched apart from one another. Therefore, histogram equalization would give better results when performed on higher bit-depth images like RAW images.

2. Another disadvantage of the global histogram equalization technique is that pixel values across the entire image are equalized using the global histogram. Thus, if the overall image is mostly dark but there is a small area with well-contrasted gradients across the entire intensity range, bright pixels in this area would be compressed towards the brighter bins resulting in pixels in the brighter ranges being binned together and loss of visual information. Notice in figure 17, the intricate grooves on the ceiling light fixture are barely visible in the enhanced image even though details around the bed and makeup table are better contrasted. Similar loss of details is observable around the lighting above the makeup table where the second bulb has mostly disappeared in the latter image. Therefore, by using localized histograms for the transformation function can prevent local contrast from being overblown. Such a technique is referred to as adaptive histogram equalization.

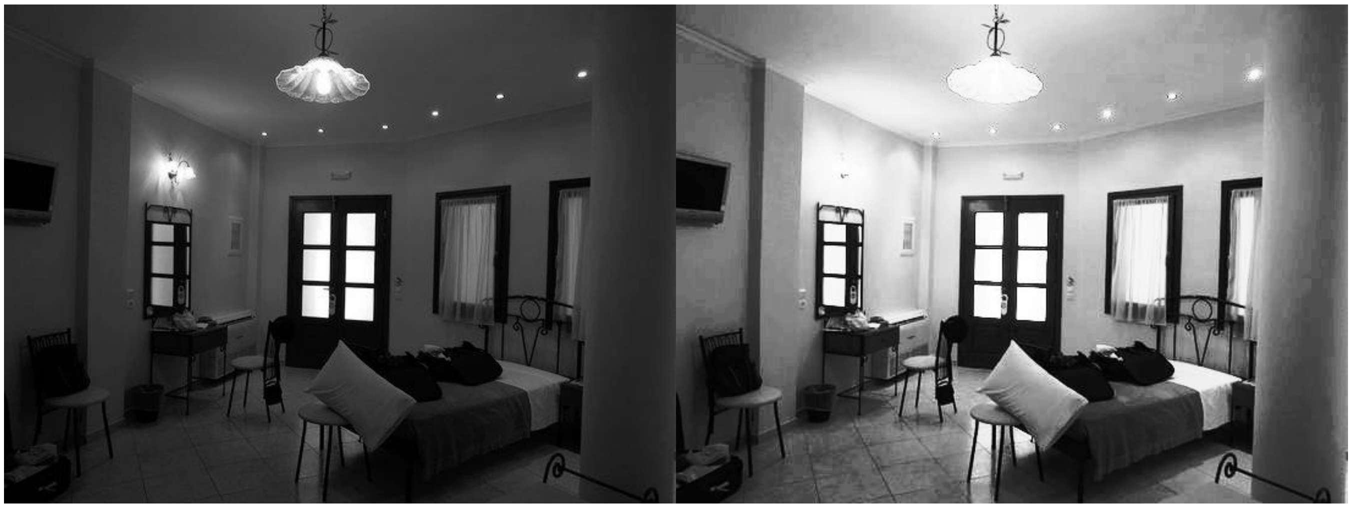


Figure 19. From left for sample 7: Original L, Histogram Equalized L*,*

3. Finally, histogram equalization can result in an unrealistic look on the output image as the entire scene looks equally bright and the lighting ambiance in the final image can look entirely different from the original like in Figure 19. Though such a filter may look appealing to some as it resembles HDR tone mapped images albeit with poorer saturation of colors. This is a mostly trivial issue though as the histogram equalization is mostly used in the scientific community to allow researchers and medical practitioners to easily identify hidden structures in their images, which may be of interest, though as previously stated in point 1, the process can result in false positives where background noise is amplified and misrepresented as actual structures.

Possible Improvements

Adaptive Histogram Equalization
As previously explained, adaptive histogram equalization alleviates the issue of loss of visual information due to excessive contrast enhancement by using localized histograms for transformation mapping instead. Thus, already well-contrasted regions will not be negatively affected by the equalization process due to poor contrast in the overall image.

There are several approaches for adaptive histogram equalization. They can be classified into two main classes, the tile-based approach and the sliding window approach. In both approaches, the main steps involve calculating the histograms and cumulative probabilities for each individual contextual region and then using the cumulative probability to map the intensity transformation.

The tile-based approach divides the image up into individual blocks based on the specified grid size and calculate a localized histogram for each block based on the pixels contained within. This can produce a distinct difference in contrast between neighboring tiles if their histograms are not uniform or similar. To reduce the effect of tiling in the output image, interpolation is done for every pixel when performing the transformation using the closest cumulative distribution function output. For pixels in the center of the image, the cumulative probabilities of the closest tiles from their top-left, top-right, bottom-left and bottom-right are bilinearly interpolated to give the final transformation coefficient. Pixels lining the top and bottom borders have the cumulative probabilities of the closest tiles from their left and right linearly interpolated, while the left and right border pixels use the closest top and bottom tiles. Pixels in the corner tiles just use the cumulative probabilities from their respective tiles directly. (Zuiderveld, 1994)

The sliding-window approach calculates the localized histogram for every pixel separately as the window slides across the image. This approach does not require any interpolation, thus number of operations per window or tile is lesser. However, the total number of operations for the entire algorithm increases drastically as the number of histograms that need to be calculated is as many as the pixel count.

	Tile-based approach	**Sliding-window approach**
1.	For every tile in image, calculate probability histogram. $O\left(\frac{MN}{W^2} \times W^2\right) = O(MN)$	For every pixel in image, calculate probability histogram. $O(MN \times W^2)$
2.	Calculate cdf for every tile and interpolate for every pixel	Calculate cdf for every pixel
3.	Perform transformation with interpolated cdf	Perform transformation with cdf

Given W as the tile or window width, we can see that time complexity for calculating cdf the for tile-based approach is in linear time while the sliding-window approach takes polynomial time. The time complexity for just interpolation is in constant time so the overall time complexity remains unchanged for the tile-based approach. Furthermore, when we add in clipping into the steps between 1 and 2 for contrast limited adaptive histogram equalization (CLAHE), computation cost for the sliding-window approach can shoot up exponentially. A comparison done between both approaches for CLAHE on a window size of 8×8 showed that the tile-based approach required only 6.25% number of operations for histogram computation and 1.56% for histogram clipping (Isaksson, 2017).

Thus, some optimizations have been proposed for the sliding-window approach to try to reduce its computational time (Wang & Tao, 2006). Firstly, instead of recalculating the entire histogram for every pixel, the trailing row or column can be subtracted from the previous histogram and the new leading ones added to it to obtain the new histogram. Thus, time complexity can be reduced from $O(MN \times W)$, though it is still in polynomial time. Another minor optimization that can be done is that instead of calculating the entire cdf across the entire intensity range, we need only the cumulative probability above or below a pixel intensity, depending on which is closer to either end of the scale. So, if a pixel lies below 128, we need only sum up the probabilities up to it from 0 to be used as the transformation coefficient. If a pixel lies closer to 255, we take the probabilities above it and subtract it from 1.0 for probabilities or the window size if using the frequency histogram. This would approximately half the total number of operations we need to perform for cdf computation. Finally, Wang & Tao proposed that by setting window size to a product of L and integral power of 2, we can eliminate any multiplication or division in the basic adaptive histogram equalization algorithm (without clipping). In their proposed algorithm,

$$s_k = L \times cdf_r(r_k) = L \times \frac{1}{W^2}\sum_{j=0}^{k} n^j = \frac{\sum_{j=0}^{k} n^j}{\frac{W^2}{L}}$$

$$Thus, when\ W^2 = L \times 2^p,$$

$$\frac{W^2}{L} = 2^p\ and\ p = \log\frac{W^2}{L}$$

Thereafter, since $\frac{W^2}{L}$ is a power of 2, we need only perform bit right shifting by p bits on the cumulative frequency histogram $\sum_{j=0}^{k} n^j$ to obtain the output s_k. However, such an algorithm implies the maximum possible value to be L instead of $L - 1$, which in the 8-bit color range would result in an intensity value of 256 which would have to be clipped to 255 in order to not result in integer overflow and flipped colors. This would mean that there would always be merging of bins and loss of intensity gradations in the last few bins. Thus, in our implementation we did not peruse this optimization trick.

The sliding window algorithm can also be parallelized as the calculations of histograms and cdf are independent for every row. Thus, at initialization we can split the image into as many horizontal blocks as there are available threads and perform the equalization separately for every block.

Our implementation of the contrast limited adaptive histogram equalization opts for the sliding window approach. In order to reduce code complexity, we have hard coded the number of bins to 256 across the entire 8-bit range from $0 - 255$. The code is displayed on the next page.

Implementation

```
def clahe(im, win_size, max_slope, clip_tolerance):
  pixelCount = win_size**2
  shift = int(math.log2((win_size**2)/256))

  imPadded = np.pad(im, int(win_size/2), mode = 'symmetric')
  imNew = np.zeros([im.shape[0],im.shape[1]], np.uint8)
  freqHist = np.zeros(256, np.uint32)

  for im_y in range(im.shape[0]):
    if im_y == 0:
      # Initialize freqHist with histogram at top-left pixel original image
      for windowRow in imPadded[1:win_size+1, 1:win_size+1]:
        for pixelValue in windowRow:
          freqHist[pixelValue] += 1
    else:
      # Subtract trailing row
      for pixelValue in imPadded[im_y, 1:win_size+1]:
        freqHist[pixelValue] -= 1
      # Add leading row
      for pixelValue in imPadded[win_size+im_y, 1:win_size+1]:
        freqHist[pixelValue] += 1

    # Perform HE mapping
    clahe_transform(imNew, im, 0, im_y, freqHist, pixelCount, max_slope, clip_tolerance)

    freqHistX = np.copy(freqHist)
    for im_x in range(1, im.shape[1]):
      # Subtract trailing column
      for pixelValue in imPadded[im_y+1:im_y+win_size+1, im_x]:
        freqHistX[pixelValue] -= 1
      # Add leading column
      for pixelValue in imPadded[im_y+1:im_y+win_size+1, im_x+win_size]:
        freqHistX[pixelValue] += 1
      # Perform HE mapping
      clahe_transform(imNew, im, im_x, im_y, freqHistX, pixelCount, max_slope, clip_tolerance)
  return imNew

def clahe_transform(imOut, imIn, im_x, im_y, freqHist, pixelCount, max_slope, clip_tolerance):
  freqHistClip = freqHist.astype(np.float64)
  clipValue = pixelCount/256 * max_slope

  while (np.amax(freqHistClip) - clipValue) > clip_tolerance:
    clipSum = 0
    for i in range(256):
      if freqHistClip[i] > clipValue:
        clipSum += freqHistClip[i] - clipValue
        freqHistClip[i] = clipValue
    freqHistClip = freqHistClip + (clipSum / 256)

  cHist = 0
  iPixel = imIn[im_y][im_x]
  if (iPixel <= 128):
    for i in range(iPixel + 1):
      cHist += freqHistClip[i]
  else:
    for i in range(iPixel + 1, 256):
      cHist += freqHistClip[i]
    cHist = pixelCount - cHist
  imOut[im_y][im_x] = round((cHist/pixelCount) * 255)
```

This is the code for contrast limited adaptive histogram equalization function which includes additional clipping to limit contrast enhancement. Clipping can be skipped by setting *max_slope* to L, which then gives the unclipped output for the sliding window adaptive histogram equalization algorithm.

[AHE] Results

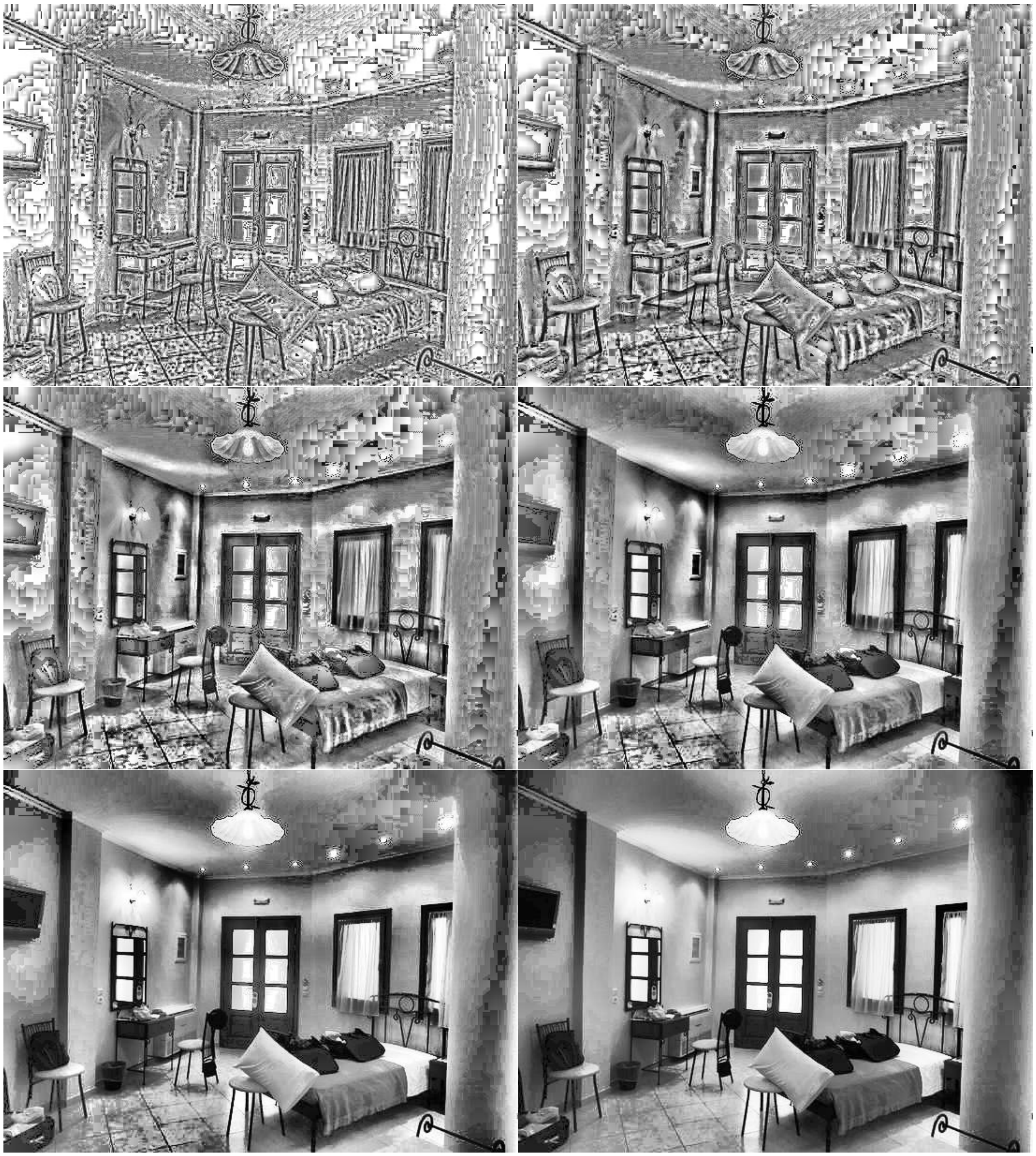


Figure 20. AHE enhanced sample 7. 1st row: 8×8, 16×16. 2nd row: 32×32, 64×64. 3rd row: 128×128, 256×256

As stated previously for Figure 19, our qualms with the global histogram equalization was that it compressed the dynamic range of well-contrasted regions and caused lost of details of the grooves on the lighting. By using a localized histogram for equalization, we can avoid such contextual regions being negatively impacted by an equalization using a global histogram with poor contrast. As seen in the output in Figure 20, using a grid size of 8, we had not only preserved the detailed grooves on the lighting, in fact we had enhanced their details up to the bulb itself, where they were barely visible due to glare from the bulb. On the other hand, such an output can be considered over-enhancement where not only actual image features are enhanced, but noise and artifact blocks are also amplified, and the overall image is generally noisy and not very useful for accurate identification of image features. Additionally, the visual representation has become more of a sketch than a realistic grayscale scene. This is because with a small grid size and pixel count, the total number of occupied bins in homogeneous regions drops proportionally, and when equalization is performed, these input pixels are "stretched" to the extremes of the intensity scale and smooth gradations of color over a larger spatial context in the original image are lost.

Meanwhile, setting too large a grid size would result in the same problem as the vanilla histogram equalization process as the local histogram will still be plagued by too many dark pixels saturating the lower bins. Referring to Figure 20, the grooves are barely visible for grid sizes 128 and 256, even though amplified blocky artifacts are largely reduced in the upper right regions.

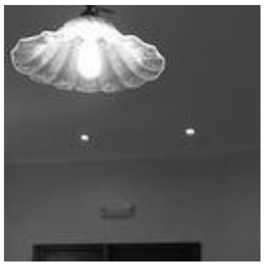


Figure 21. 128×128 crop

Empirically, grid sizes of 32 and 64 seems to give the best enhancement result where we can still see the grooves on the lighting clearly and even the rectangle grooves on the bottom of the door which can't be easily spotted in the vanilla output in Figure 19. The smaller grid size had marginally more artifacts than the other and edges are more pronounced.

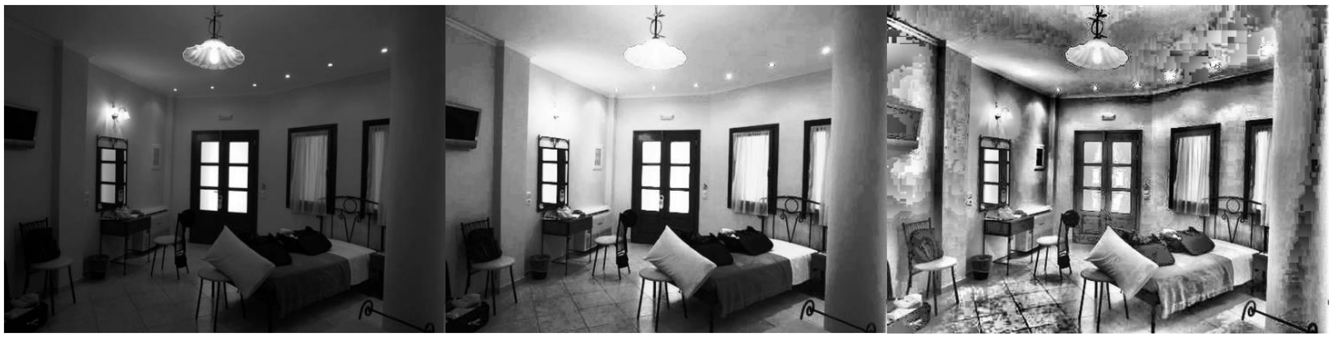


Figure 22. Original sample 7, vanilla enhanced, AHE enhanced 64×64

In Figure 22, we can see that image features in the AHE enhanced output like the bottom of the bed and the door carvings are clearer than in the vanilla enhanced version. Details are not lost but there is slight over-enhancement and blocky artifacts are clearly visible too. Additionally, the dark ambiance has been eroded even further and the scene looks uniformly lit in the AHE output. In Figure 23 on the following page, we see that the final cdf curve is smoother and without gradation "steps" as global histogram bins are "split" and equalized according to their local contextual region. Thus, we see less unfilled bins, pixels are more evenly distributed among all bins, maximum bin value also dropped from ~ 0.04 to ~ 0.007 and consequently the difference between the largest and smallest bin from ~ 0.04 to ~ 0.0065.

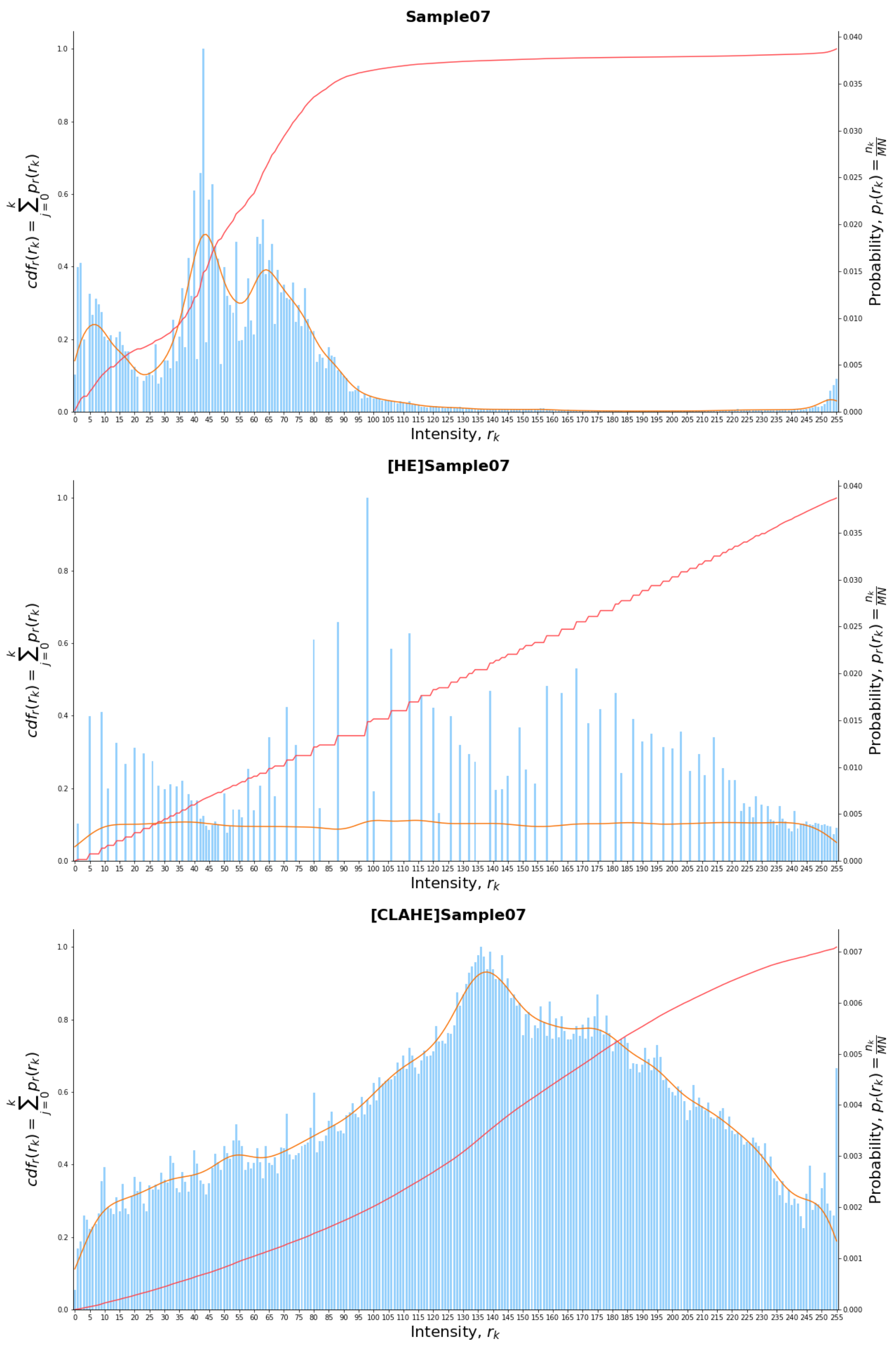


Figure 23. Cumulative distribution function for sample 7. From top: Original, vanilla, AHE enhanced 64×64

Contrast Limited Adaptive Histogram Equalization

Even though adaptive histogram equalization was able to preserve local contrast, it could also easily cause excess contrast enhancement and over-enhancement of noise and artifacts. Thus, by limiting contrast enhancement to a specified limit in our algorithm we can calibrate the balance between image enhancement and noise amplification. Because the degree of contrast enhancement is proportional to the slope of the local cdf function, we can reduce contrast enhancement by limiting the slope of the cdf function. And since the cdf function is the cumulative sum of every bin, the slope or gradient of the function at any bin is the size of that bin itself. For instance, given a 64×64 window containing 4096 pixels equalized using 256 bins, a bin containing $\frac{4096}{256}$ pixels would give a corresponding slope of 1 on the cdf function. Hence, by clipping the maximum bin size to a multiple of the average bin size across all bins, we can limit the maximum slope and thus contrast enhancement. It has been advised that a binary search algorithm be used for finding the ideal value, below a specified threshold, to clip at before performing a single clip operation (Zuiderveld, 1994). For our implementation, we just repeatedly iterate through all bins to find the excess sum and perform clipping until all bins are within our threshold of 1.0. Clipping is done by finding the excess sum of all pixels or probabilities above the clipping value and redistributing the sum across all bins. Doing so would sometimes push a bin above the threshold again and so multiple iterations may be required. Furthermore, because we are averaging the excess sum across all bins, we convert the discrete frequency histogram values into floating point values to allow for uniform redistribution.

[CLAHE] Results

Referring to Figure 25, on the following page, we can see that block artifacts increases as we unclamp the maximum slope progressively from 4 to 128. Above the slope limit of 8, image quality no longer appears to improve and in fact worsens because of increased artifacts. There doesn't appear to be any obvious difference between a slope of 64 and 128 except subtle intensity gradients along the block artifact at the top left corner. Thus, limits of 4 and 8 gave the best result over adaptive histogram equalization with no clipping, having much lesser artifacts and over-enhanced edges and preserved more of the darker gradients from the original. Using a limit of 4 seems to give a smoother output than the limit of 8, as can be seen from the pixelated block on the top right ceiling. However, the grooves on the door are less obvious as a result, so the optimum limit may lie between these 2. In general, CLAHE gave a better contrasted image than vanilla HE with minimal increased artifacts, though the resulting image looks less realistic and more akin to a tone mapped photo.

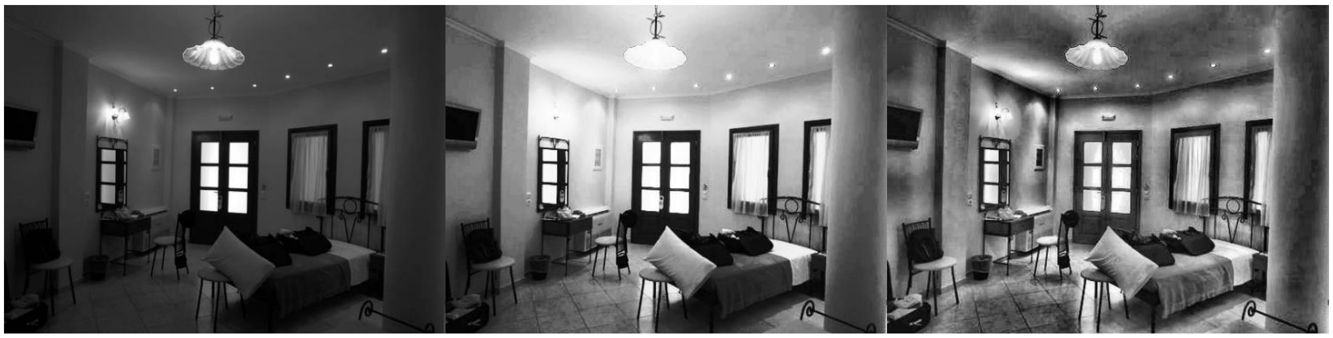


Figure 24. Original sample 7, vanilla enhanced, CLAHE enhanced 64×64 slope limit 8

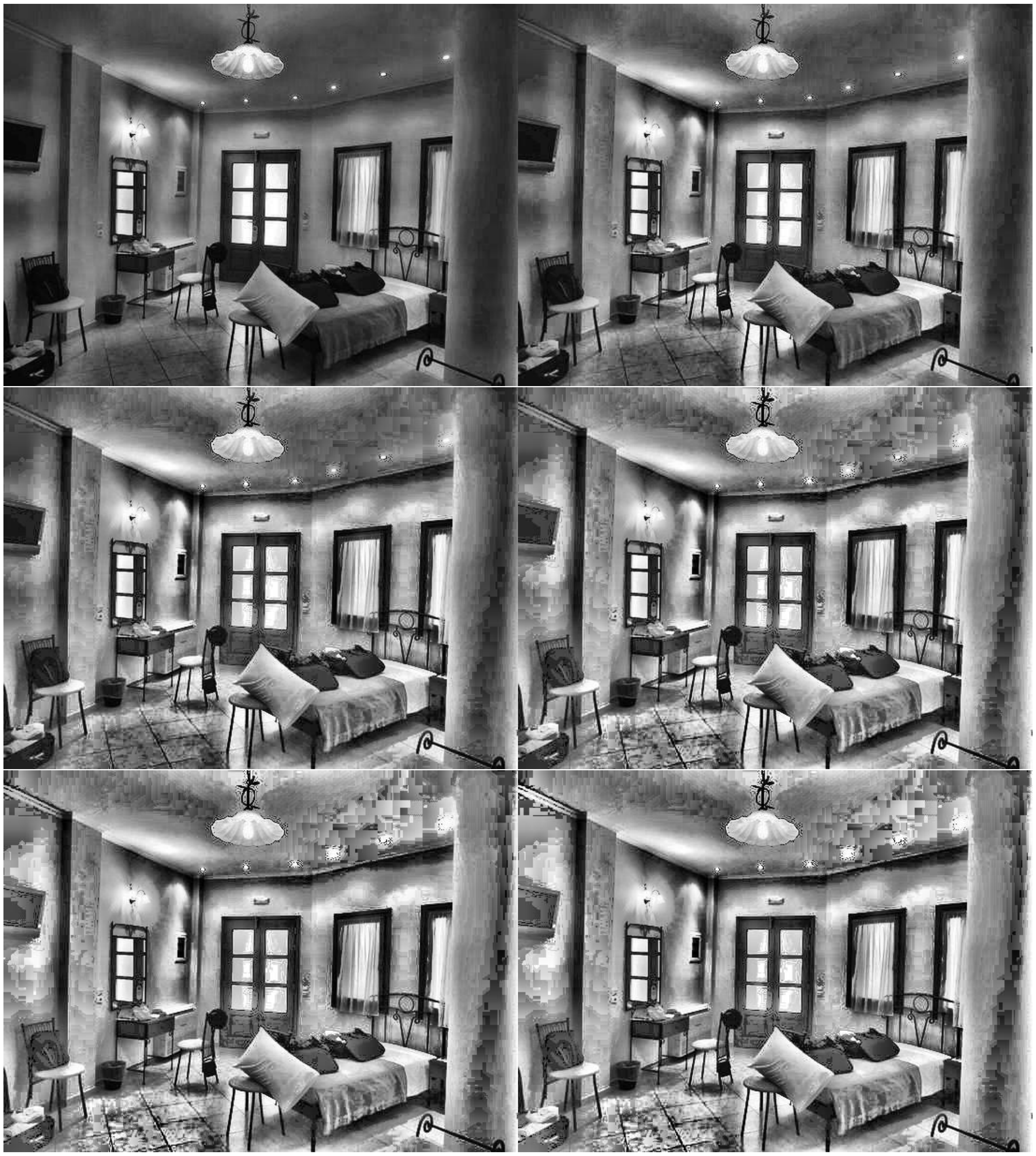


Figure 25. 64×64 CLAHE enhanced sample 7 limited at different maximum cdf slopes.
1st row: 4, 8. 2nd row: 16, 32. 3rd row: 64, 128.

Discussion

Adaptive histogram equalization is not without its disadvantages and as previously stated, the problem with our sliding window approach is the massive computational cost which increases exponentially with both image size and window size. Furthermore, setting a small clip limit which requires multiple clips would push computation time further out of feasible limits. The interpolated tile-based approach is what most image libraries like ImageJ and OpenCV used and though the results aren't exact copies of the sliding window approach, they are close enough approximations of them to be used out in the field where time is a practical constraint.

From the table on the right, we see that it took 24 minutes just to process a small image like sample 6 using a window size of 256 and clipping limit of slope 8. For sample 5 with a dimension of $(600, 337)$ and roughly three time fewer pixels than sample 6 (202200 vs. 621530), the time to compute the AHE output was 8 minutes. Thus, for a 4K image with 8294400 pixels, rough 13.3 times that of sample 6, we can estimate that the time needed to process it with our CLAHE algorithm at 256×256 with a clip limit of slope 4 is 360 minutes or 6 hours, excluding overhead. The code can probably by streamlined in a low-level language like C++ with more efficient use of data structures and multithreading, but the time would most likely still be in hours. However, it should also be noted that as shown from our test results, using a larger window size may not guarantee a better contrasted image. Likewise, using a smaller clip size may also limit the effectiveness of the contrast enhancement.

Sample 6 (910, 683)	**Time Taken (mins)**
Global HE	<1
AHE 8×8	2
AHE 16×16	2
AHE 32×32	3
AHE 64×64	6
AHE 128×128	11
AHE 256×256	24
CLAHE 32×32, 4	6
CLAHE 32×32, 8	4
CLAHE 64×64, 4	8
CLAHE 64×64, 8	7
CLAHE 128×128, 4	13
CLAHE 128×128, 8	12
CLAHE 256×256, 4	27
CLAHE 256×256, 8	24
System Specifications: Intel Core i7-3770K @ 3.50GHz 16GB DDR3 RAM 1600MHz	

In fact, we have noticed that by using a low clip limit of 4 or 8 with the window size of 32×32, we were able to achieve comparable results to a higher window size of 256×256 with no clipping as shown in Figure 26 on the following page. Thus, by using a clip limit, we were able to reduce over-enhancement in the 32×32 AHE output, allowing us to reduce the optimum window size further with better or zero downgrade in output quality at a fraction of the time required of the higher window sizes.

Finally, we also provide a set of colored images of the enhanced outputs for sample 7, with the equalized $L*$ channels merged back with the $a*$ and $b*$ channels with no more additional adjustments done on color balance. Output from the CLAHE enhanced version using 32×32 window size with slope 4 seems to give the most visually appealing result with the slope 8 output having brighter contrast in the darker regions like the door. Noise is minimal in the slope 4 version while the slope 8 one is noisier. Such noise can be reduced by performing additional filtering on the output like denoising, blending with the original photo, or using a smoothing filter.

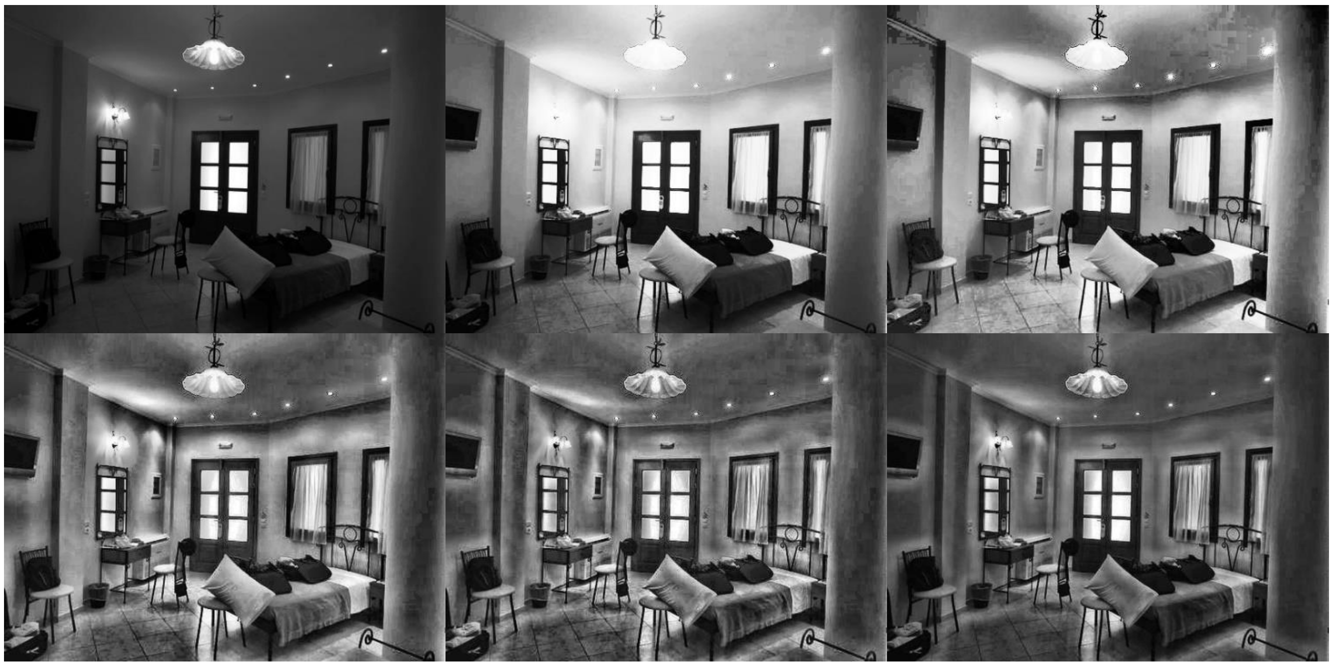


Figure 26: 1st row: Original, global HE, AHE 256. 2nd row: CLAHE 64 slope 8, CLAHE 32 slope 8, CLAHE 32 slope 4

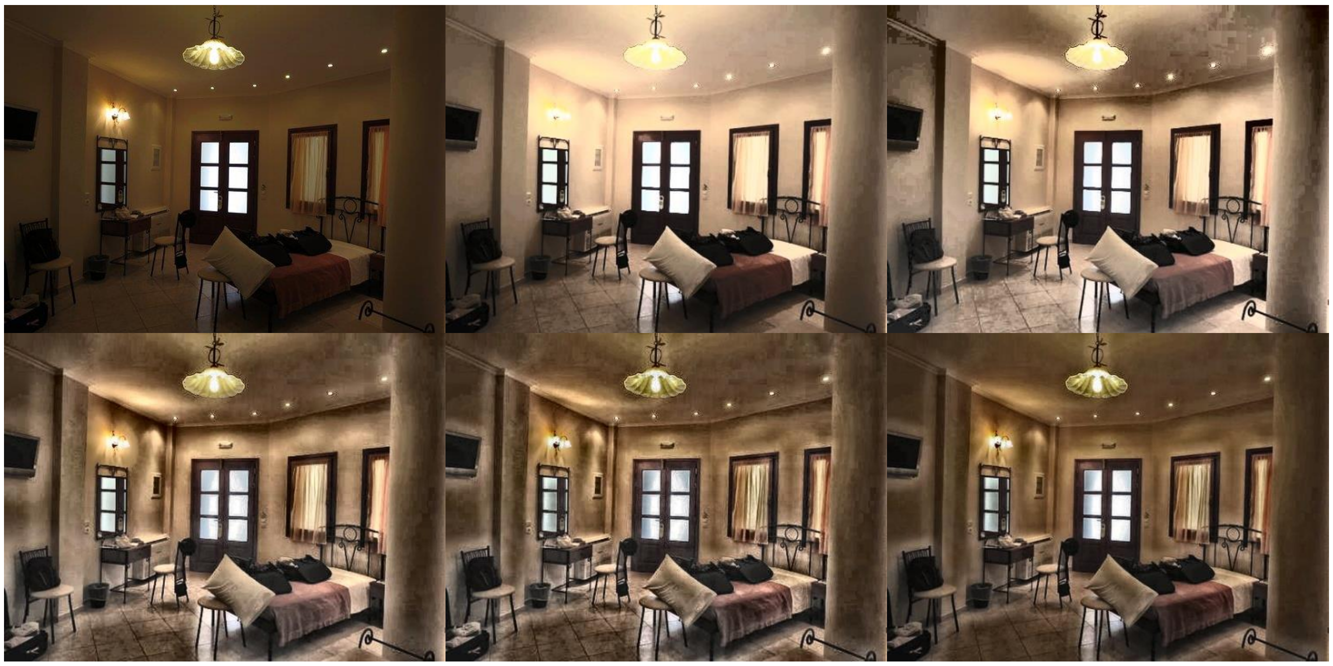


Figure 27: 1st row: Original, global HE, AHE 256. 2nd row: CLAHE 64 slope 8, CLAHE 32 slope 8, CLAHE 32 slope 4

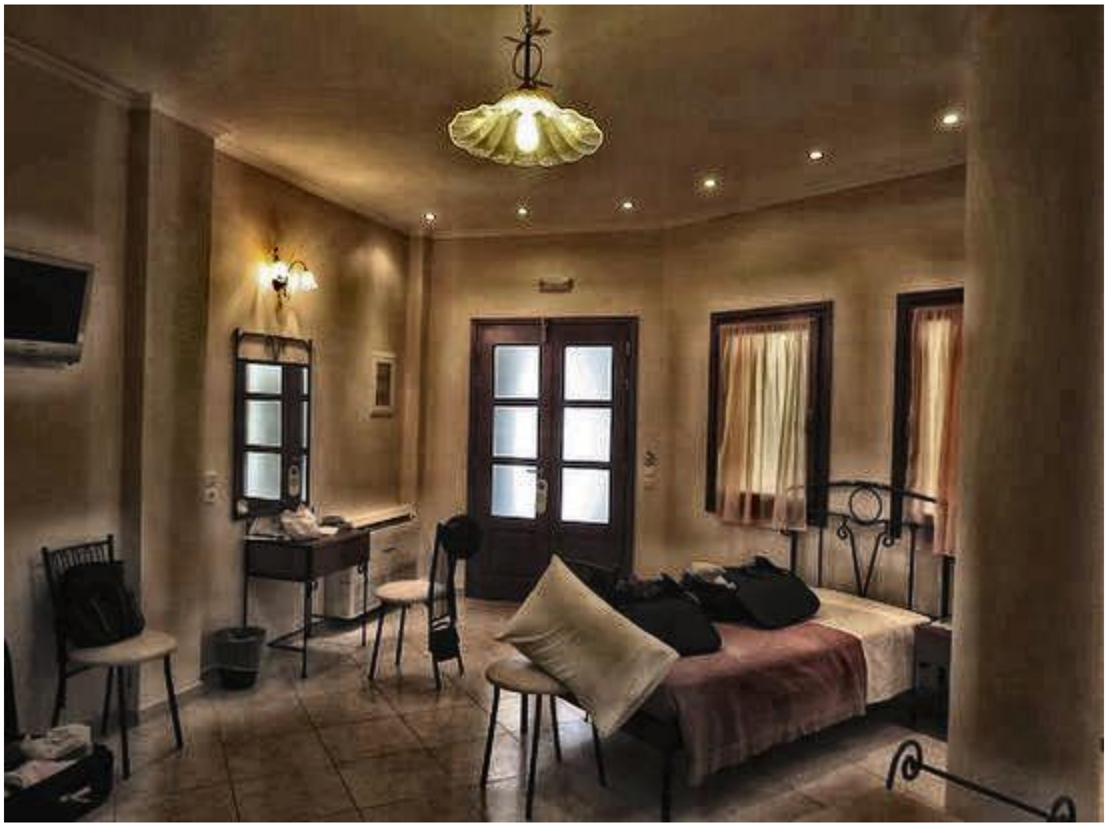


Figure 28. Full-sized CLAHE enhanced using 32×32 window size with slope 4

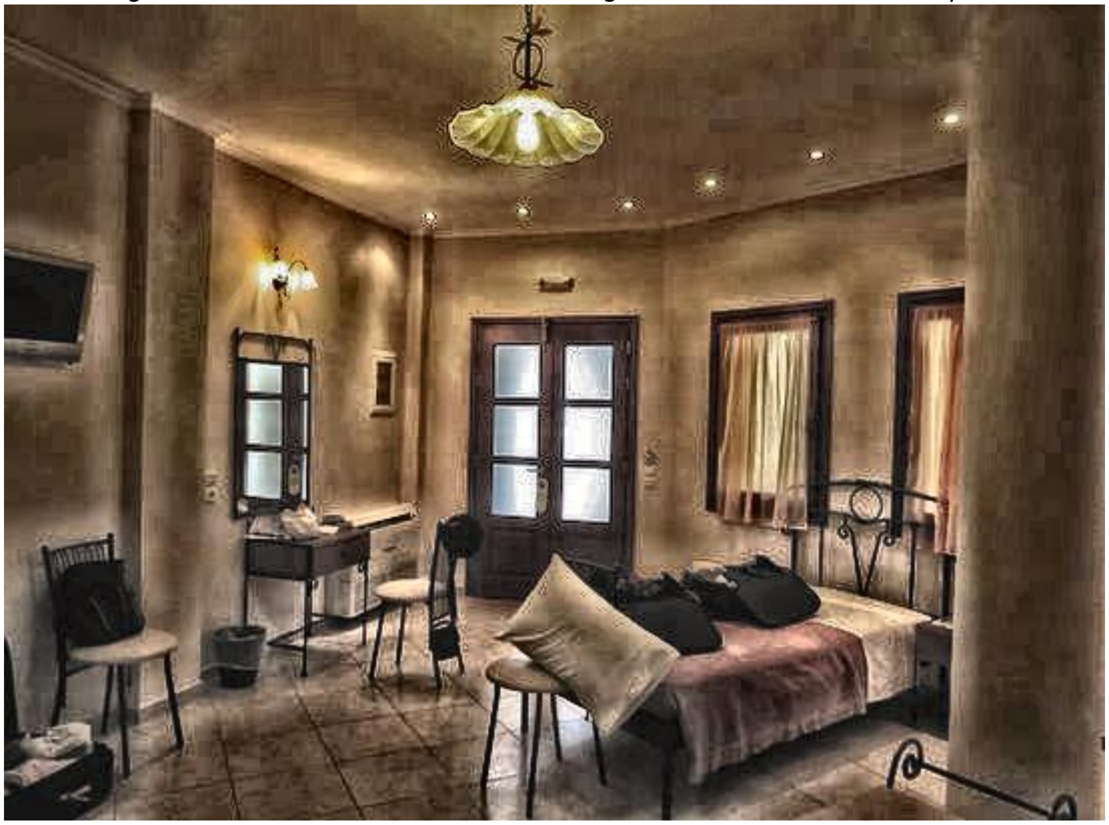


Figure 29. Full-sized CLAHE enhanced using 32×32 window size with slope 8

References

Isaksson, J. (2017). *FPGA-Accelerated Image Processing Using High Level Synthesis with OpenCL.* Retrieved from Digitala Vetenskapliga Arkivet: https://www.diva-portal.org/smash/get/diva2:1159832/FULLTEXT01.pdf

Wang, Z., & Tao, J. (2006). A Fast Implementation of Adaptive Histogram Equalization. *2006 8th international Conference on Signal Processing. 2.* IEEE. doi:10.1109/ICOSP.2006.345602

Zuiderveld, K. (1994). Contrast Limited Adaptive Histogram Equalization. In P. S. Heckbert, *Graphics Gems IV* (pp. 474-485). Academic Press.