CS 674
Assignment 1

Bryson Lingenfelter Nate Thom

Submitted: September 25, 2019
Due: September 25, 2019

0.1 Division of Work

0.1.1 Coding

Bryson: Quantization, Histogram Equalization
Nate: Sampling, Histogram Specification

0.1.2 Writing

Bryson: Quantization, Histogram Equalization
Nate: Sampling, Histogram Specification

1 Technical Discussion

1.1 Sampling

The first technique implemented was Sampling. This method reduces the variation of pixel values in an image. This is accomplished by reducing the sample rate of an image by some factor. The original image has a factor of one, while an image where every second pixel is sampled has a factor of two. Note that an image with a factor of two will be half the size of the original. Once the image has been sampled it must be resized to its original size. This results in duplicate pixel values, and thus a smaller variation in pixel values.

The operation of sampling is achieved as follows

$$Sampled_{X,Y} = Image_{X,Y}$$

$$Sampled_{X+1,Y+1} = Image_{X+(1*Factor),Y+(1*Factor)}$$
$$Sampled_{X+2,Y+2} = Image_{X+(2*Factor),Y+(2*Factor)}$$
$$...$$
$$\forall X \in \frac{Dimensions_X}{Factor} \wedge \forall Y \in \frac{Dimensions_Y}{Factor}$$

Finally, resize the image to original dimensions.

1.2 Quantization

Quantization compresses an image by reducing the number of grey channels rather than reducing the number of pixels. Our input images are grey-scale images with 1-byte intensity values, allowing 256 possible grey levels.

We performed quantization by applying $r_{new} = \lfloor \frac{r_{old}}{256/Q} \rfloor$ for each pixel, where r is the pixel intensity value and Q is the new number of grey levels. When we output the images, we rescale the intensity values back to $[0, 255]$ by multiplying the result with the denominator after applying the floor function.

1.3 Histogram Equalization

Histogram Equalization is a method for redistributing the intensity values of an image such that there is a roughly even distribution of intensity values across all pixels. This is done by creating a random variable for pixel intensity and remapping the probability density function of this variable to be uniform by integrating the probability distribution function from $-\infty$ to x: $F_X(x) = \int_{-\infty}^{x} f_X(a)da$. Because images are not continuous data, this is done in practice by calculating the following sum for all intensity levels k:

$$r_k = 255 \sum_{i=0}^{k} \frac{n_i}{n}$$

Where n is the number of pixels in the image and n_i is the number of pixels with intensity i, making $\frac{n_i}{n}$ the discrete estimate of $f_X(a)$, the probability of a pixel having intensity value i. The sum returns a probability density between 0 and 1, which is scaled to be a valid pixel value by multiplying by 255 and taking the floor of the resulting real number. r_k is then the new pixel intensity for pixels of intensity k.

1.4 Histogram Specification

Histogram Specification is another method for redistributing intensity values of an image to match a probability distribution. Rather than distributing the pixel values normally (along a normal curve), we distribute them along some other predefined distribution. This is done by identifying the mapping functions that normally distribute the pixel values of the original image and the predefined distribution. Now that both transformation functions for attaining the normal distribution are known, all we need to do is find the inverse of the transformation from predefined distribution to normal.

With this knowledge we can simply transform the original image to be normally distributed, then compute the inverse transformation of the normal distribution to our predefined probability function.

In mathematical notation:

Compute equalization transformation: $s = T(r)$
Compute equalization transformation: $v = G(z)$
Compute I_2 (The input image distributed along target distribution) through $z = G^{-1}(s)$ or $z = G^{-1}(T(r))$

2 Implementation Details

2.1 Sampling

Sampling was implemented in Python using the integer division floor operator to index the row and column dimensions of the input image. The function takes as input some image and a sampling factor. The sampling factor governs how many of the original images pixels will be sampled and used in the output result. Every F pixels are sampled, where F is the provided factor. A new image object is now created. This image only contains the sampled pixel values, so it is a fraction of the original size. Finally, this new image is resized to match the original images dimensions and it is returned.

2.2 Quantization

Quantization was implemented in Python using the integer division operator to compute floor. Our quantization function takes two parameters, an image and a desire number of grey channels. The image is modified in place and also returned by the function. We first compute the divisor, then in a nested for loop replace each intensity value in the provided image with the result of integer division of the intensity value by the divisor, multiplied by the divisor.

2.3 Histogram Equalization

Equalization was implemented in Python using 3 functions. `get_histogram` takes an image as input and returns the histogram for that image, a 256 length list with each index containing the number of pixels with that index's intensity level. This is done using a nested for loop which iteratively updates the list for each pixel in the image. `histogram_eq` takes an image and a 256 length list as input, where the list contains a new intensity value for each index's intensity level. This is also done using a nested for loop, this time setting each pixel in the image to the value contained in the map at the index of the old intensity value. Finally, `histogram_norm` uses the histogram from `get_histogram` to produce a map for `histogram_eq` by computing $r_k = 255 \sum_{i=0}^{k} \frac{n_i}{n}$ where n_i is the sum of values from the histogram and n is the image width multiplied by the image height.

2.4 Histogram Specification

Histogram Specification was implemented in Python using 3 functions. The first function call is to the `histogram-spec` function. This function takes as input an image, the path to the target image, and the name of the output file. We first find the histogram of the input image with `get_histogram` (as described in the above subsection), then we also find the histogram of the target image. The mapping of both histograms to the normal distribution is computed with the map function described above. These values are stored in `pixel_map_1` and `pixel_map_2`. Knowing both distributions, we can simply use the inverse mapping from the normal distribution to the target distribution. The inverse mapping is accomplished by storing the values in `pixel_map_2` as keys in a dictionary where the values are the pixel values 0-255. The final map, `pixel_map_3`, contains the mapping from the input image's pixel values to the closest available pixel in the target image. The closest pixel is located searching the inverse mapping dictionary's keys and finding the value closest to the desired value. The input image is now transformed using the `histogram_eq` function. Finally, the output image is returned.

3 Results and Discussion

3.1 Sampling

Experimentation for sampling was completed on the Lenna and Peppers images. We sampled with factors of 2, 4, and 8. The difference in quality between the original image and the factor 2 image are relatively small, mostly a loss of sharpness is apparent in Lenna. Beginning in the factor 4 image, we begin to lose clear distinction between facial features in Lenna and sharpness is lost in Peppers. Finally, a factor 8 sampling leads severe loss of intensity changes between objects in both images.

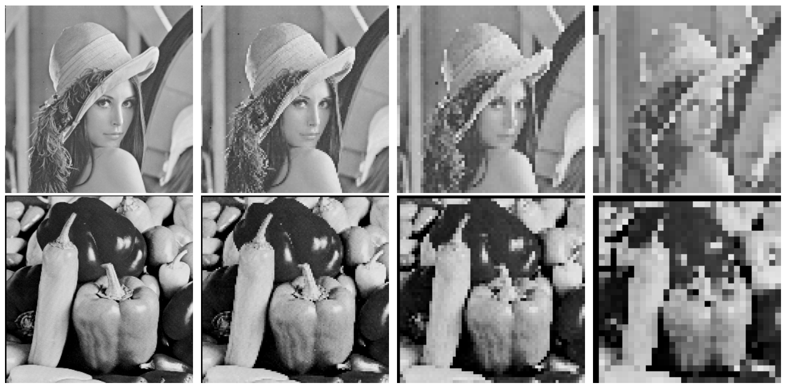

Figure 1: Sampling results on the Lenna and Peppers images, with factor 2 sampling on the left and 8 factor sampling on the right.

3.2 Quantization

For the Lenna and Peppers images, we created reduced the number of grey levels from 256 to 128, 32, 8, and 2. The difference is quality is not substantial for 128 and 32 grey levels, but once the number of channels drops to 8 there is noticeable posterization.

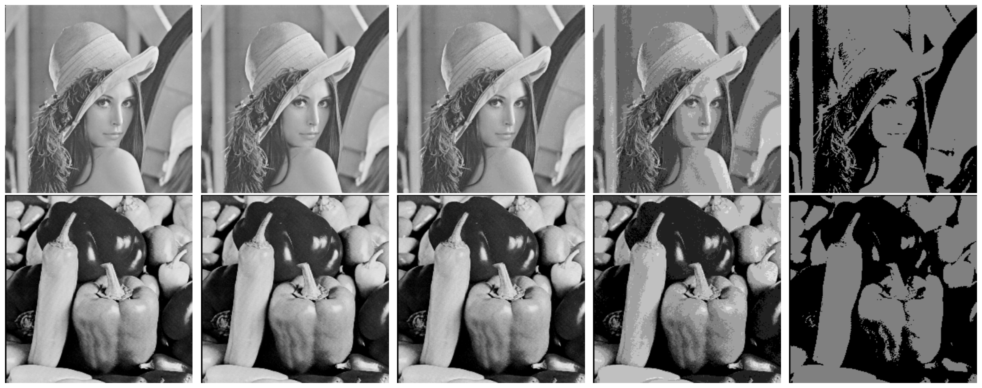

Figure 2: Quantization results on the Lenna and Peppers images, with 256 grey levels on the left and 2 grey levels on the right.

At two grey levels, it is still clear what the images are of but many features are obscured. The Peppers image retains almost all its features, with all the major objects remaining clearly distinguishable even with just 2 grey channels. The Lenna image fares slightly worse, with most facial features almost entirely removed. We suggest that this can be explained by the two images' histograms; the Peppers image has a mostly bimodal histogram, with peaks on either side of an original grey level of 128. The original Lenna image, however, for the most part has intensity values greater than 128 as well as 3 to 5 distinct modes.

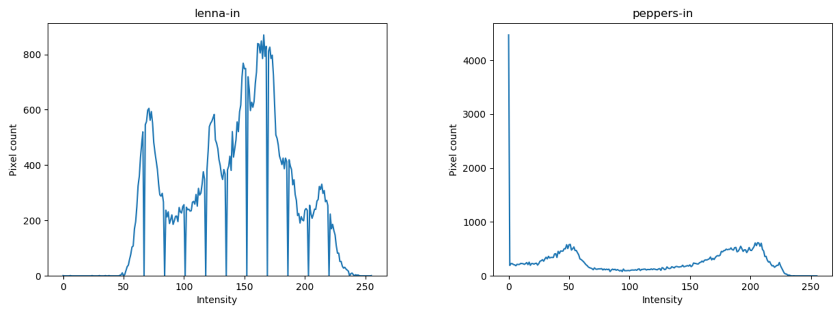

Figure 3: Intensity histograms for Lenna (left) and Peppers (right). With the exception of the large number of pure black pixel values in the Peppers image, the distribution appears mostly biomodal with a center at 128. The Lenna image, however, has several modes and is skewed to the right.

To improve quantization results we tried changing the quantization of an equalized image (see next section for details) rather than the original image. This had little to no impact for some images, but for others it substantially improved the 2-grey level images. As previously noted, the histogram for the Peppers image is already well balanced and so the results were not substantially improved. For the unbalanced Lenna image, however, the facial features are substantially more clear.

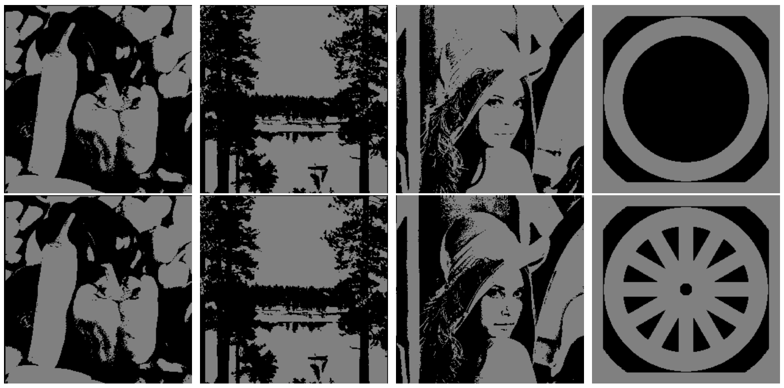

Figure 4: Comparison between 2-grey level versions of original images (top) and equalized images (bottom). For some images, such as the peppers and boat images, the effect is negligible. For others, however, the equalization drastically improves quality.

3.3 Histogram Equalization

We performed histogram equalization on two images, F16 and Boat. The intensity values for F16 are mostly concentrated between 175 and 225. Boat has a bimodal intensity distribution, with peaks at 40 and 200. We performed equalization on both images.

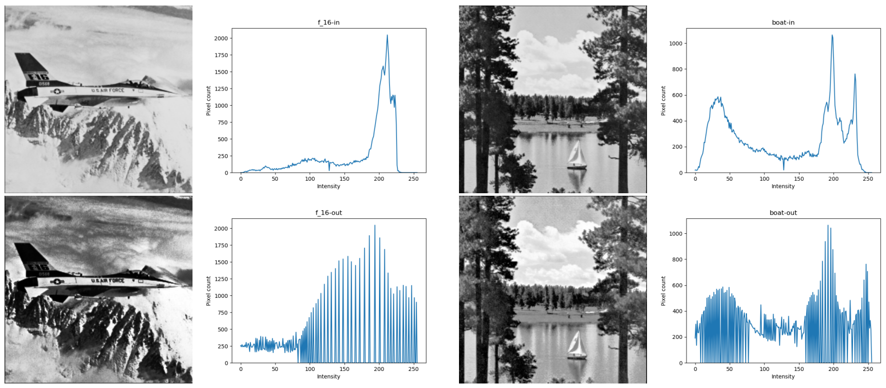

Figure 5: Histogram equalization on the F16 and Boat images. On top are the original images and histograms, on bottom are the images and histograms after equalization.

After Equalization, the background detail in the F16 image is much more clear. The background clouds and mountain had very little contrast, but after equalization have a much larger intensity range and as a result far more clear texture. The plane itself already had a reasonable amount of contrast, so the effect was somewhat smaller. This image had a far more skewed distribution than Boat, so it makes sense that equalization has a larger effect.

The Boat image has more contrast than the F16 image, containing dark trees in the foreground and light clouds and water in the background. As a result equalization has a smaller effect on this image. It adds a small amount more contrast to both sides of the distribution; the trees are a little brighter, the clouds stand out from the sky a bit more, and the reflections in the water are more clear.

Because equalization is a global, automatic technique, not every change is positive. The F-16 text on the plane's vertical stabilizer is a bit harder to make out, and the number below it is effectively unreadable. This likely happened because the background white on the vertical stabilizer is still one of the darker shades in the image because the background is so washed out. When the intensity levels are equalized, these grey levels are compressed and some detail is lost.

3.4 Histogram Specification

We performed Histogram Specification (matching) on the F16 and Boat images using the Peppers and SF images as target distributions. Peppers is a fairly uniform histogram with a large preference towards black pixels. The SF distribution has pixel values primarily between 75 - 175, it is a relatively low-contrast image.

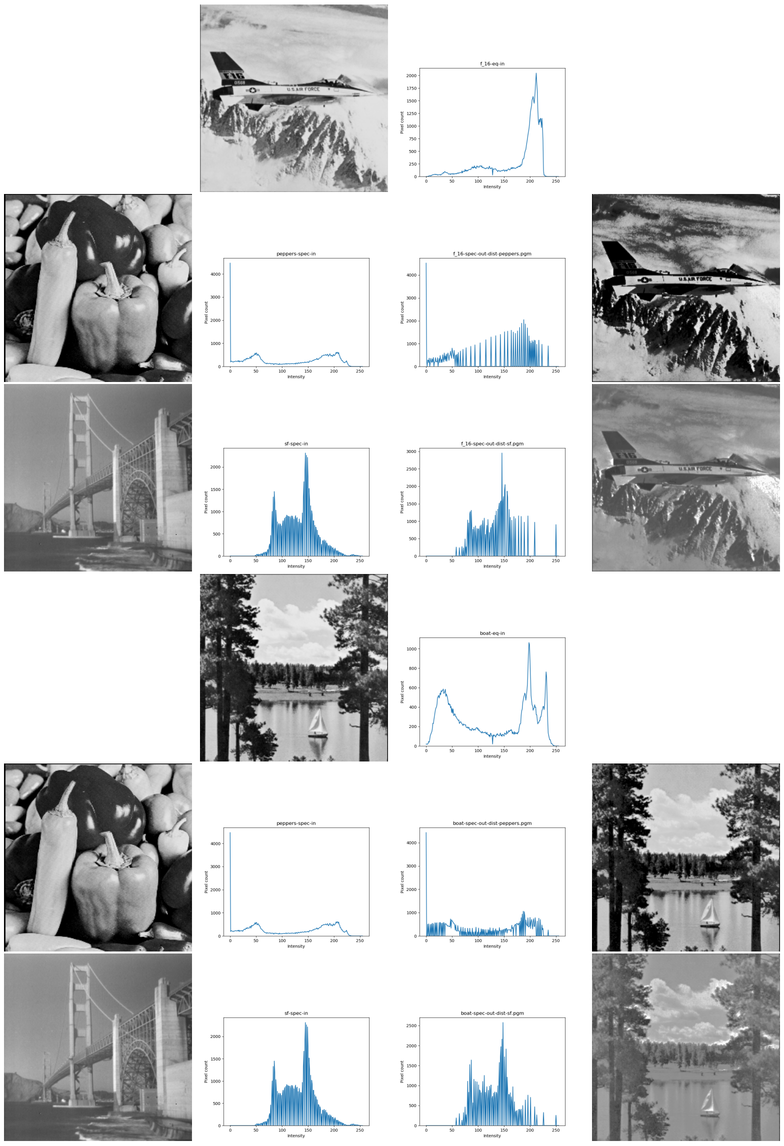

Figure 6: Histogram Specification on the F16 and Boat images. The image to be adjusted is shown first, it is followed by the image to be used for specification and the results.

We are very happy with the results of this transformation! When applying the Peppers transformation both test image's contrast improves. This is expected because the peppers distribution is mostly equalized with an extreme maxima at 0 and two small peaks. The most significant difference between the equalized images and the Peppers-specification images is that the latter have more black pixels. This is the expected result.

When applying the SF distribution, the images lose contrast. The gray tone that is found in SF is clearly represented in both F16 and Boat. Both output images share the central stack of pixel values, with very few pixels appearing outside of the 75-175 range.

4 Program Listing

4.1 Sampling

```
def sample(image, factor):
    image_width, image_height = image.size

    new_pixels = []
    for height_index in range(image_height // factor):
        for width_index in range(image_width // factor):
            new_pixels.append(image.getpixel((width_index*factor, height_index*factor)))

    new_image = Image.new('L', (image_width//factor, image_height//factor))
    new_image.putdata(new_pixels)
    new_image = new_image.resize((image_width, image_height))

    return new_image
```

4.2 Quantization

```
def change_quantization(image, levels):
    image_width, image_height = image.size

    divisor = 256 // levels
    for i in range(image_width):
        for j in range(image_height):
            image.putpixel((i,j), (image.getpixel((i,j))//divisor) * divisor)

    return image
```

4.3 Histogram Equalization

```
def get_histogram(image):
    image_width, image_height = image.size
    histogram = [0] * 256
    for i in range(image_width):
        for j in range(image_height):
            histogram[image.getpixel((i,j))] += 1

    return histogram

def histogram_eq(image, pixel_map):
    image_width, image_height = image.size
    for i in range(image_width):
        for j in range(image_height):
            image.putpixel((i,j), pixel_map[image.getpixel((i,j))])

def histogram_norm(image):
    image_width, image_height = image.size

    histogram = get_histogram(image)

    pixel_map = [0] * 256

    corrected_pixels = 0
    sum_r = image_width * image_height
    for i in range(256):
        corrected_pixels += histogram[i]
        pixel_map[i] = (corrected_pixels*255) // sum_r

    histogram_eq(image, pixel_map)

    return image
```

4.4 Histogram Specification

```
def histogram_spec(image, path_to_image_histogram, name):
    image_width, image_height = image.size

    histogram = get_histogram(image)
    plot_histogram(histogram, name + "-spec-in")

    target_dist_image = Image.open(path_to_image_histogram)
    target_dist_hist = get_histogram(target_dist_image)
    plot_histogram(target_dist_hist, "spec-dist-"+path_to_image_histogram.split("/")[-1])

    pixel_map_1 = [0] * 256
    corrected_pixels_1 = 0

    sum_r = image_width * image_height
    for i in range(256):
        corrected_pixels_1 += histogram[i]
        pixel_map_1[i] = (corrected_pixels_1*255) // sum_r

    pixel_map_2 = [0] * 256
    corrected_pixels_2 = 0

    for i in range(256):
        corrected_pixels_2 += target_dist_hist[i]
        pixel_map_2[i] = (corrected_pixels_2 * 255) // sum_r

    inverse_mapping_dict = {}
    for i in range(256):
        inverse_mapping_dict[pixel_map_2[i]] = i

    pixel_map_3 = [0] * 256
    for i in range(256):
        if pixel_map_1[i] in inverse_mapping_dict:
            pixel_map_3[i] = inverse_mapping_dict[pixel_map_1[i]]
        else:
            closest = min(list(inverse_mapping_dict.keys()),
                                 key=lambda x: abs(x-pixel_map_1[i]))
            pixel_map_3[i] = inverse_mapping_dict[closest]

    histogram_eq(image, pixel_map_3)
    plot_histogram(get_histogram(image), name + "-spec-out-dist-"
                   + path_to_image_histogram.split("/")[-1])

    return image
```