CS 474

Gabriella Charalampidi, Gustavo Rubio

Programming assignment 1

Gabriella: Questions 1 and 2, Image, main

Gustavo: Questions 3 and 4

Theory

Sampling:

1. Initially the user is requested to select the sampling factor. The program will keep looping until the user gives a valid factor.
2. I'm creating a `subsampled_array = []` that will hold the pixels of the subsampled image. In order to get these pixels I run a nested for loop. The outer loop (`for row in range(0, self.height, factor)`) iterates over the rows of the image with a step of 'factor' (2/4/8). Similarly the inner loop (`for col in range(0, self.width, factor)`) iterates over the columns of the image with a step of 'factor' (2/4/8). I then appends all the rows of the image to the subsampled_array.
3. I'm creating a `resized_array = []` that will hold the pixels of the output image. Again I will use a nested for loop. The outer loop (`for row in subsampled_array`) iterates over the rows of the subsampled_array, and the inner loop (`for pixel in row`) iterates over the pixels of each row. For each pixel in the current row, the pixel value is replicated horizontally 'factor' times. The expanded_row is then replicated vertically 'factor' times and appended to the resized_array.
4. Lastly the attributes of the image are updated.

Quantization:

1. Initially the user is requested to select the quantization factor. The program will keep looping until the user gives a valid factor.
2. Integer division is used to calculate the offset(`offset = 256 // quantization_level`) and a scaling factor is calculated by scaling the quantized pixel values back up to the 0-255 range(`scaling_factor = 255 /(quantization_level - 1) if quantization_level > 1 else 255`)

3. I then create a list that includes the new pixel values (`new_pixel_values = [round(i * scaling_factor) for i in range(quantization_level)]`.
   For example if the quantization level is 4 this calculation will happen:
   - i=0, new_pixel_value = (0*64) + 32 = 32
   - i=1, new_pixel_value = (1*64) + 32 = 96 etc.

   The new pixel values will be [32, 96, 160, 224].
4. In order to modify the pixels I run a nested for loop. The outer loop (`for row in range(self.height)`) iterates over the rows of the image and the inner loop (`for col in range(self.width)`) iterates over the columns of the image. Once I retrieve the pixel's value (`pixel_value = self.get_pixel(col, row)`), I determine in which 'bucket' the pixel will fall into(`index = min(pixel_value // offset, quantization_level - 1)`). Lastly, I set that pixel in the image (`self.set_pixel(col, row, new_pixel_values[index])`).

Histogram equalisation:

1. The user is prompted to type in the name of the image they want to equalize (ex: 'f_16.pgm'). The function `histogram_equalization(image)` will then be called with the single parameter of the file name.
2. The function `read_from_file(filename)` will then be called to determine the important information of the image: the width, height, max_value, and the pixel data itself. It will not do so if the file is of an incorrect type. This will be done by determining the header first. Then the entirety of the pixel data is read. Once the data is read, it is mapped into a numpy format to quickly determine the width, height, and maximum value of the data (`width, height = map(int, line.split())`)(`maxval = int(f.readline().strip())`)

3. The function `apply_histogram_equalization` is then called to turn the pixel_data into a histogram (`hist, _ = np.histogram(img_data, bins=max_val + 1, range=(0, max_val))`). The histogram is put into the form of it's cumulative sum (`cdf = hist.cumsum()`). This cumulative sum is then normalized with respect to the maximum value of the image (`cdf_normalized = (cdf - cdf.min()) * max_val / (cdf.max() - cdf.min())`). Once that is complete, the new normalized data is mapped back into the file data (`equalized_img_data = cdf_normalized[img_data]`). This is returned.
4. The new file data is outputted to a file with a new name via the `write_to_file` function.

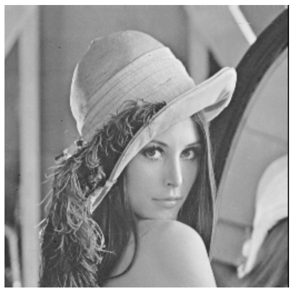
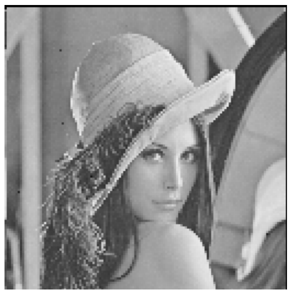
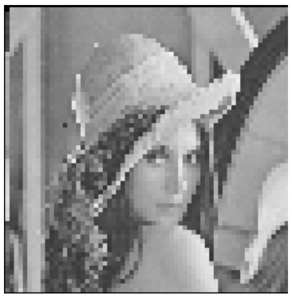
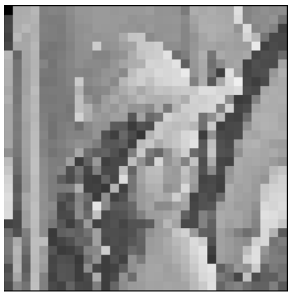
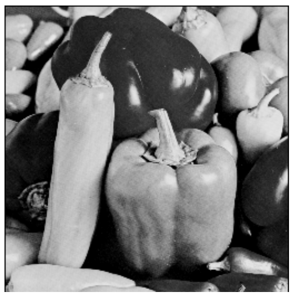
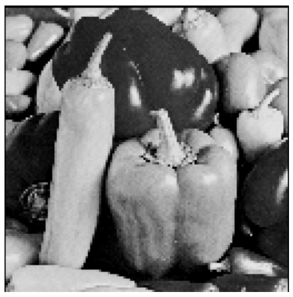
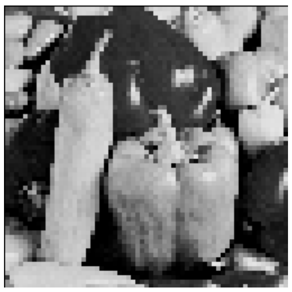
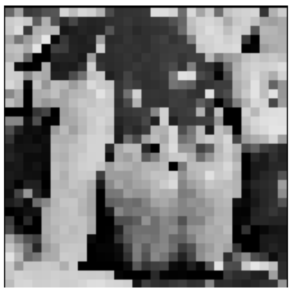
Histogram specification:

1. The user is prompted for two files: the file that will have it's histogram edited, and the file from which we will acquire a new target histogram. The `apply_histogram_specification` function is called using these two as parameters.
2. The `read_from_file` function is called twice to acquire the pixel data of the two files. There is no difference form the previous `read_from_file` function.
3. The `apply_histogram_specification` function is called with the two pixels data sets of the files as parameters. Both data sets are converted into histograms and then their cumulative sums are found, similar to the equalization process. But here, the two data sets are interpolated (`spec_data = np.interp(cdf[input_data], target_cdf, np.arange(max_val + 1))`) to create a new histogram for the output file. This new data is returned.
4. The new data is outputted as a new pgm file, similar to the previous histogram process.

Results and Discussion

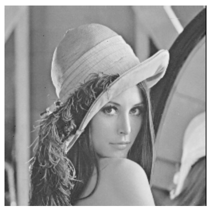
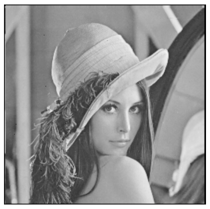
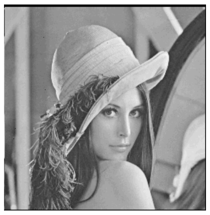
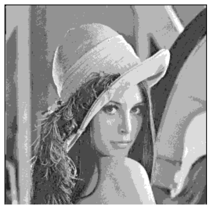
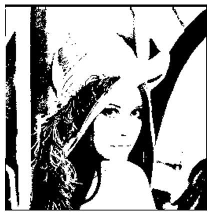
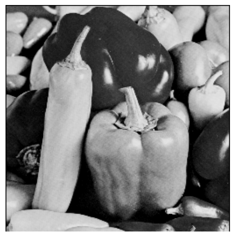
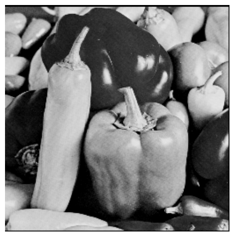
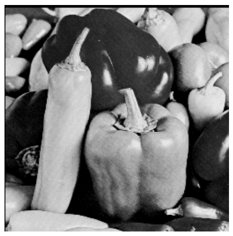
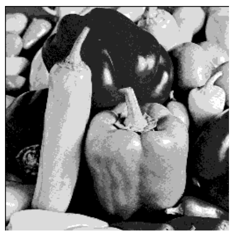
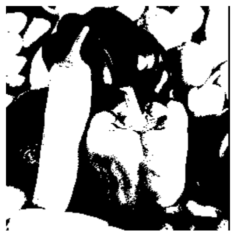
1. **Sampling**

   The subsampling process reduces the number of pixels in the image by selecting every nth pixel in both horizontal and vertical dimensions, depending on the specified factor. For example if an image is 256*256 and we subsample it by a factor of 2, then it will be 128*128. The larger the factor, the smaller the dimensions will get.
   During the resizing process, the images are scaled back to their original dimensions but keep the pixel values from the subsampled state. This is achieved by duplicating the pixels horizontally and vertically in blocks, each block corresponding to a pixel in the subsampled image.
   Although the images are restored to their initial dimensions, they are of a lower resolution and this is evident in the table below.

Original Image	Subsampled by 2	Subsampled by 4	Subsampled by 8
			
			

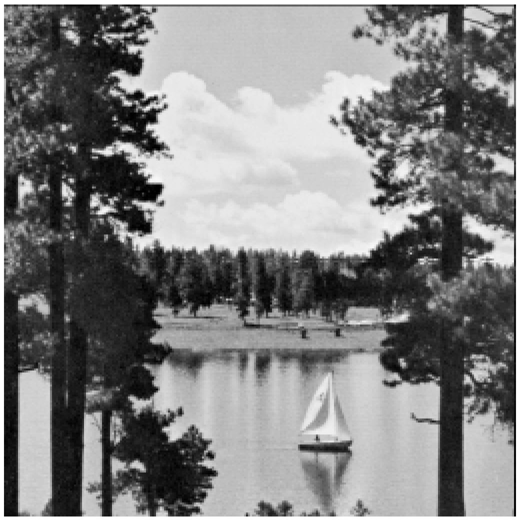
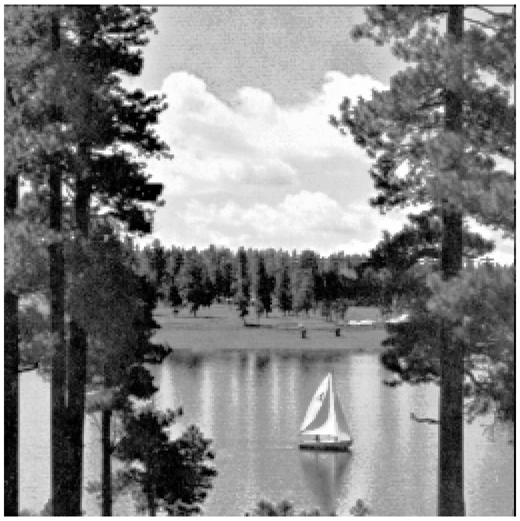
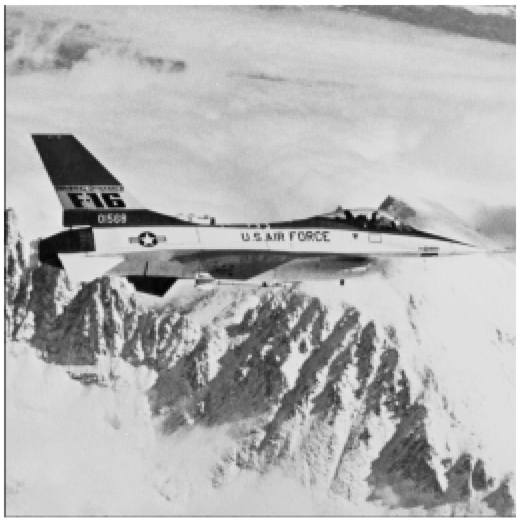
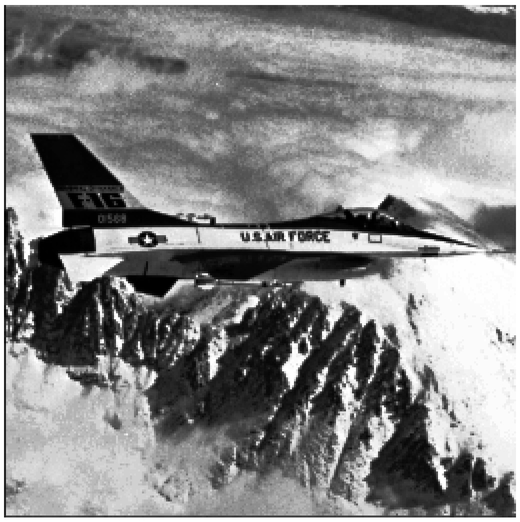
2. Quantization

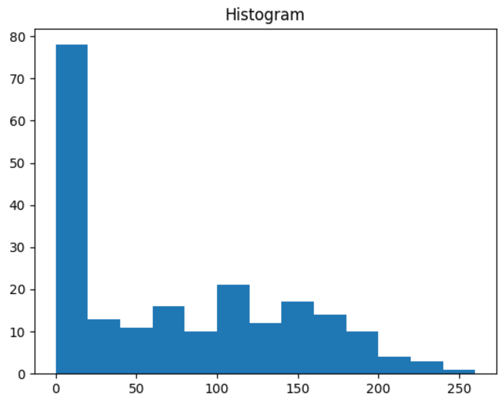
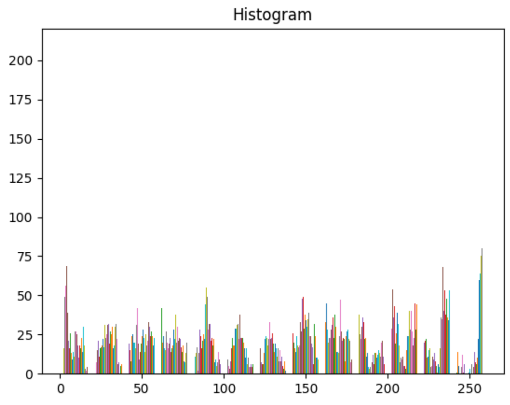
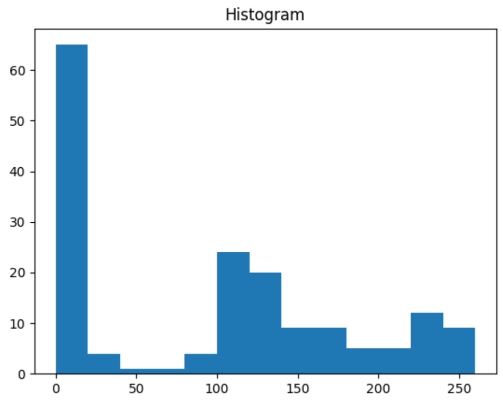
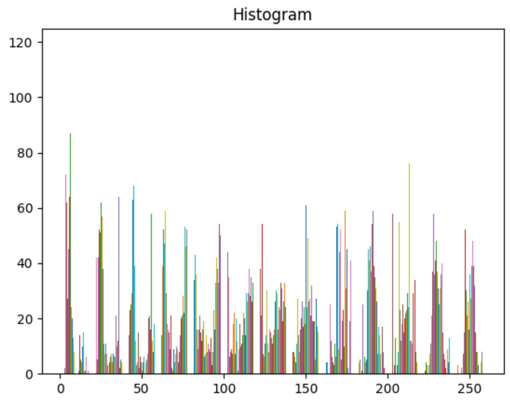
With a lower quantization level (e.g., 2), the image has a higher contrast, however, this comes at the expense of image details. As the quantization level increases, more details appear. It is also worth mentioning that the lower the quantization level the less bits are needed for the image representation. Taking that and the application's needs into consideration, the programmer has a tool and has to decide how they will handle it. Lower quantization levels mean less colours, less details but also lower storage needs. All these observations can be seen in the table below.

Original Image	Quantization factor: 128	Quantization factor: 32	Quantization factor: 8	Quantization factor: 2
				
				

3. Equalization

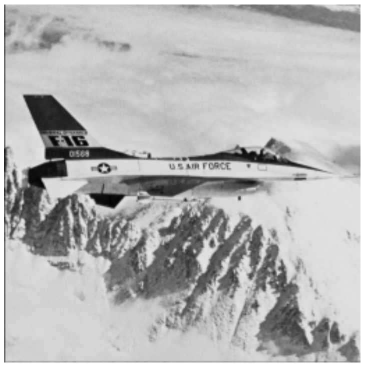
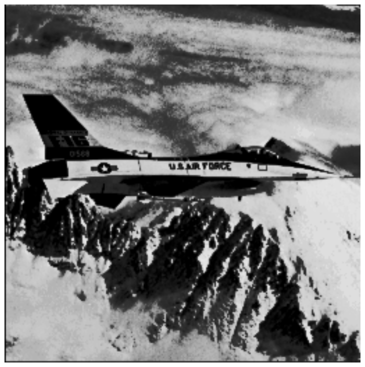
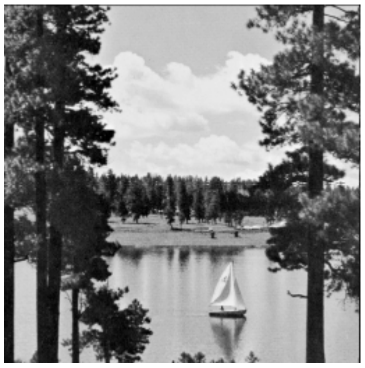
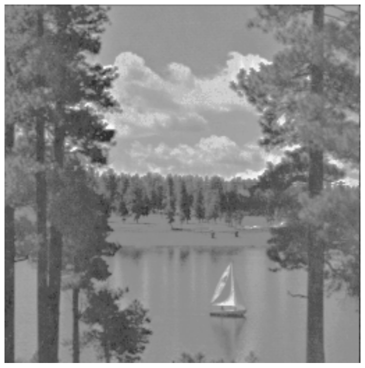
Equalization operations on the images have a noticeable effect in that the contrast of the images is increasing because the lightest lights of the image become lighter and the darkest darks become darker. This has the effect of bringing out details that were harder to see earlier, in these cases, the reflections in the water of the boat image and the details of the mountain sides in the f_16 image.

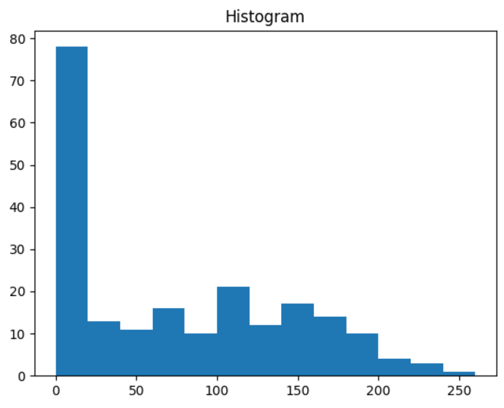
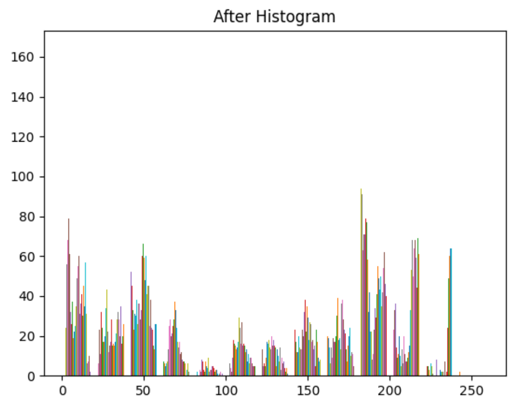
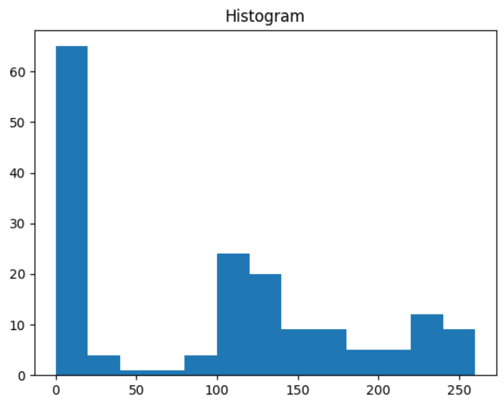
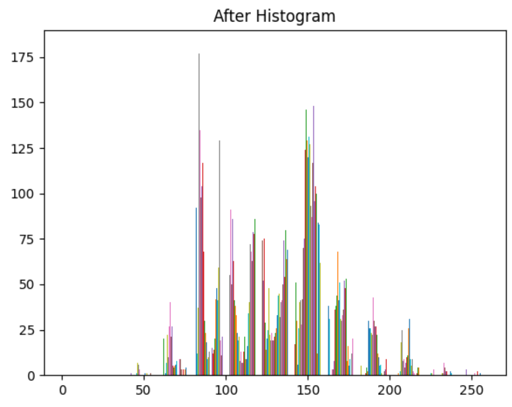
Original Images	Equalizated Images
 boat.pgm	 boat_equalizaed.pgm
 f_16.pgm	 f_16_equalized.pgm

Before Histogram	After Histogram
 f_16.pgm histogram	 f_16_equalized.pgm histogram
 boat.pgm histogram	 boat_equalized.pgm histogram

4. **Specification**

The specification operation has a clear effect on the output image depending on the range and distribution of dark and light values from the image we are deriving our histogram from. For examples, the peppers image has noticeably more black values than the f_16 image, so the output f_16 image ends up with a lot more black. The output for the boat image is a lot more muted and grey, just like the sf image.

Original Image	Histogram Image	Output Image
 f_16.pgm	peppers.pgm	 f_16_specified.pgm
 boat.pgm	sf.pgm	 boat_specified.pgm

Before Histogram	After Histogram
 f_16.pgm histogram	 f_16_specified.pgm histogram
 boat.pgm histogram	 boat_specified.pgm histogram