Homework #2

Jacob Taylor Cassady

CECS 627: Digital Image Processing

3.5 DO THE FOLLOWING:

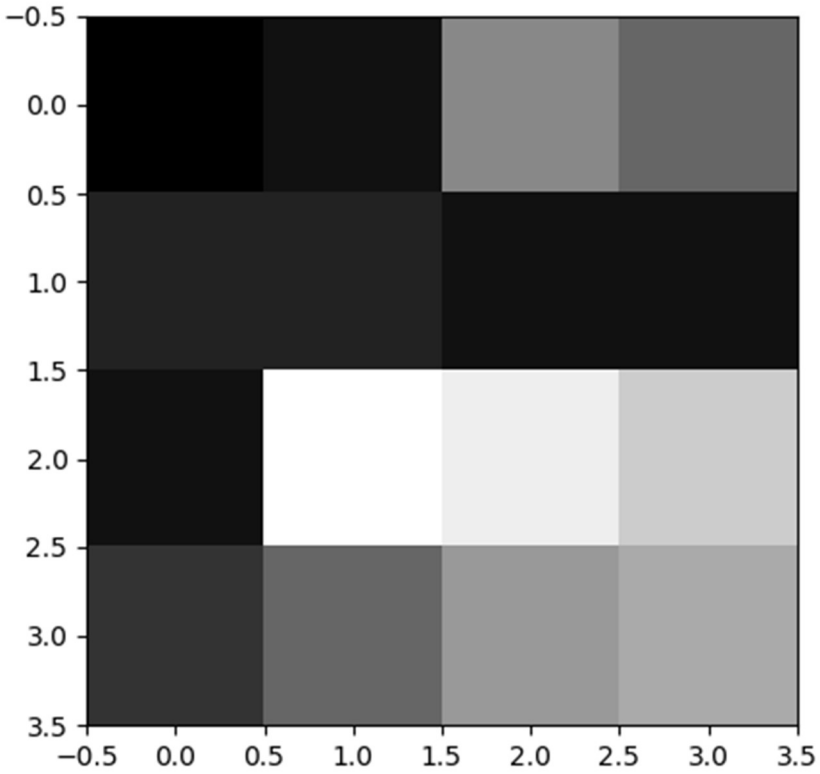
Purpose a method for extracting the bit planes of an image based on converting the value of its pixels to binary:

```
def calculate_bit_planes(image, gray_scale=True, bits_per_pixel=8):
    bit_planes = None

    # If the image is grayscale...
    if gray_scale:
        # Retrieve the image shape
        rows, columns = image.shape
        # Create bit planes of all zeros
        bit_planes = np.zeros((rows, columns, bits_per_pixel))

        # Update bit plane values by iterating over pixels
        for row_index, row in enumerate(image):
            for column_index, pixel in enumerate(row):
                for bit_plane_index, bit_plane in enumerate(bit_planes):
                    bit_plane[row_index, column_index] = (pixel & 2**bit_plane_index) >> bit_plane_index

    # Retrun produce bit planes
    return bit_planes
```

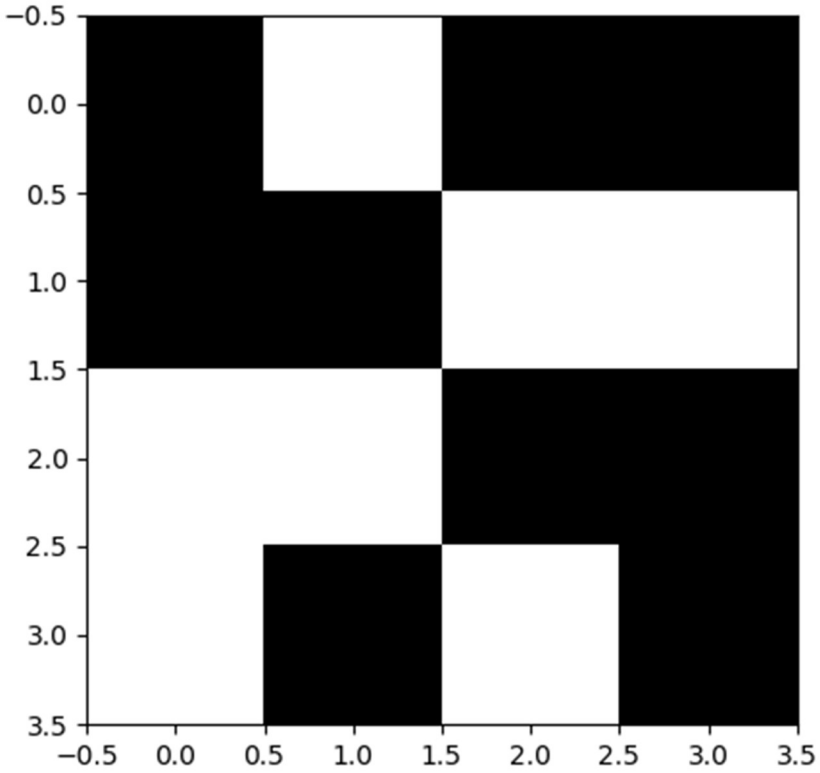
The above method was written in Python utilizing the NumPy library. Bit planes are generated by iterating over each pixel and then iterating over a bit mask for each corresponding bit plane.

Find all the bit planes to the following 4-bit image:

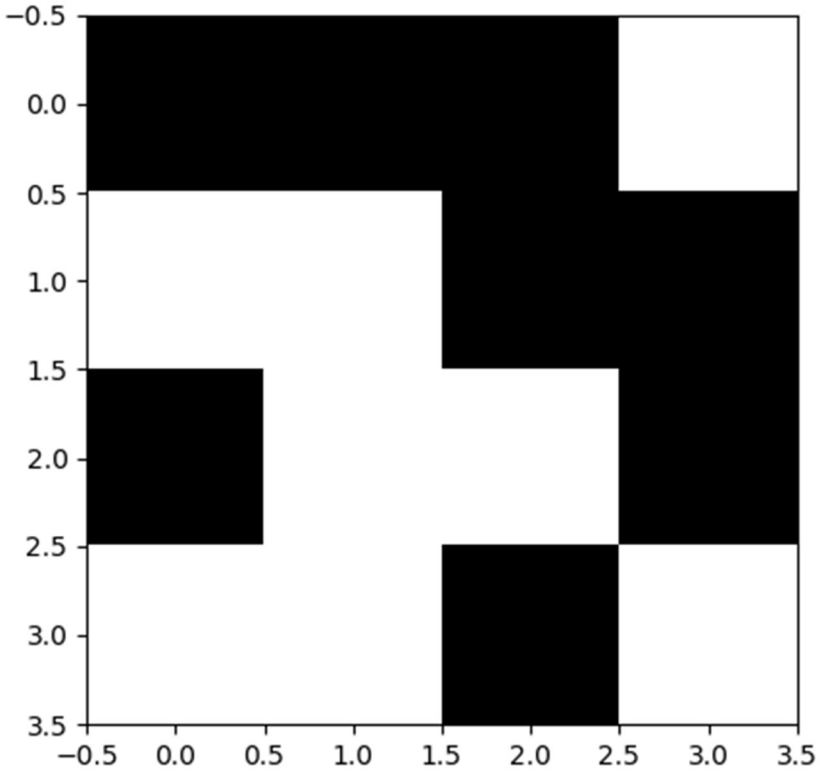
Input



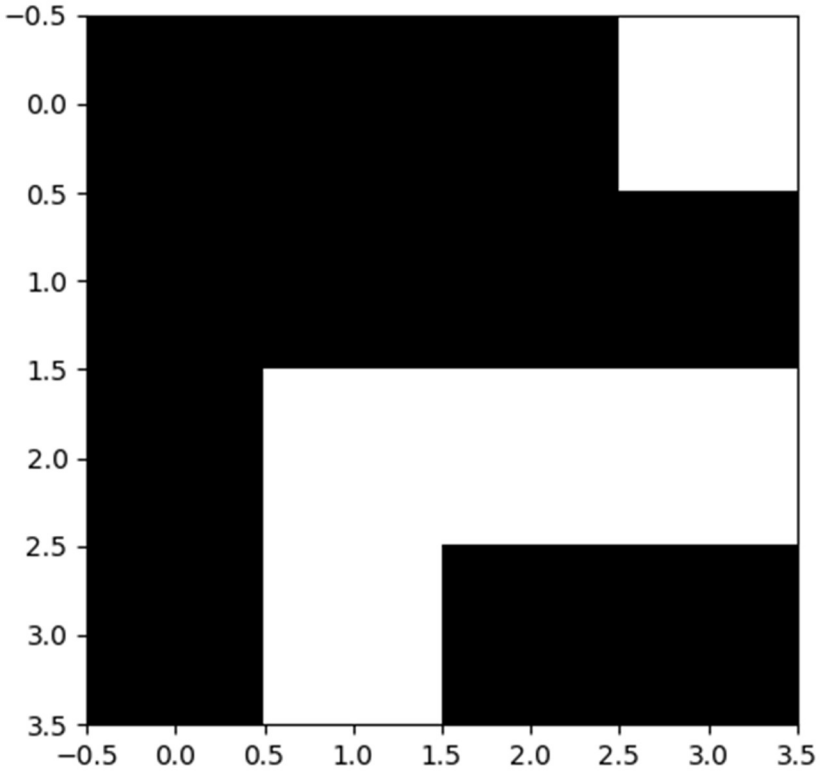
Bit Plane 0



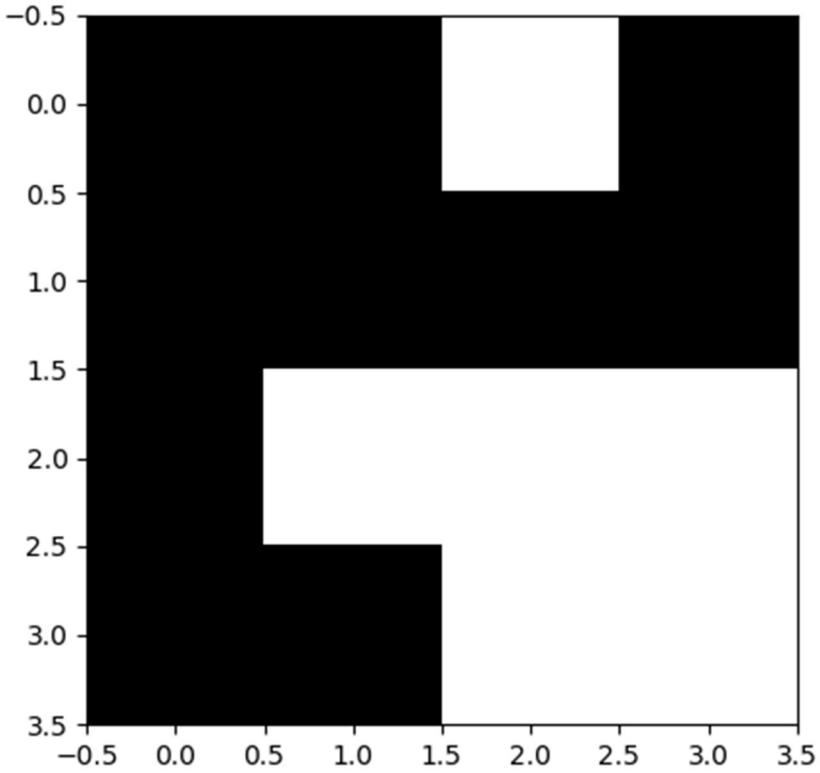
Bit Plane 1



Bit Plane 2



Bit Plane 3

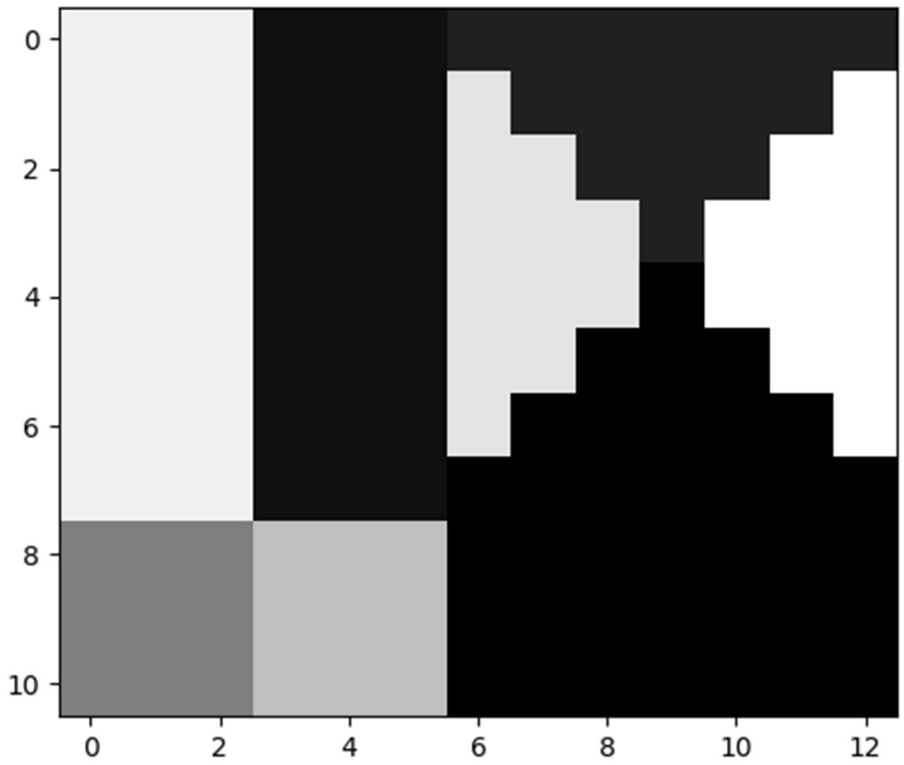


3.7

An image was generated to closely match the one described in the book. The generated pixel values are shown below.

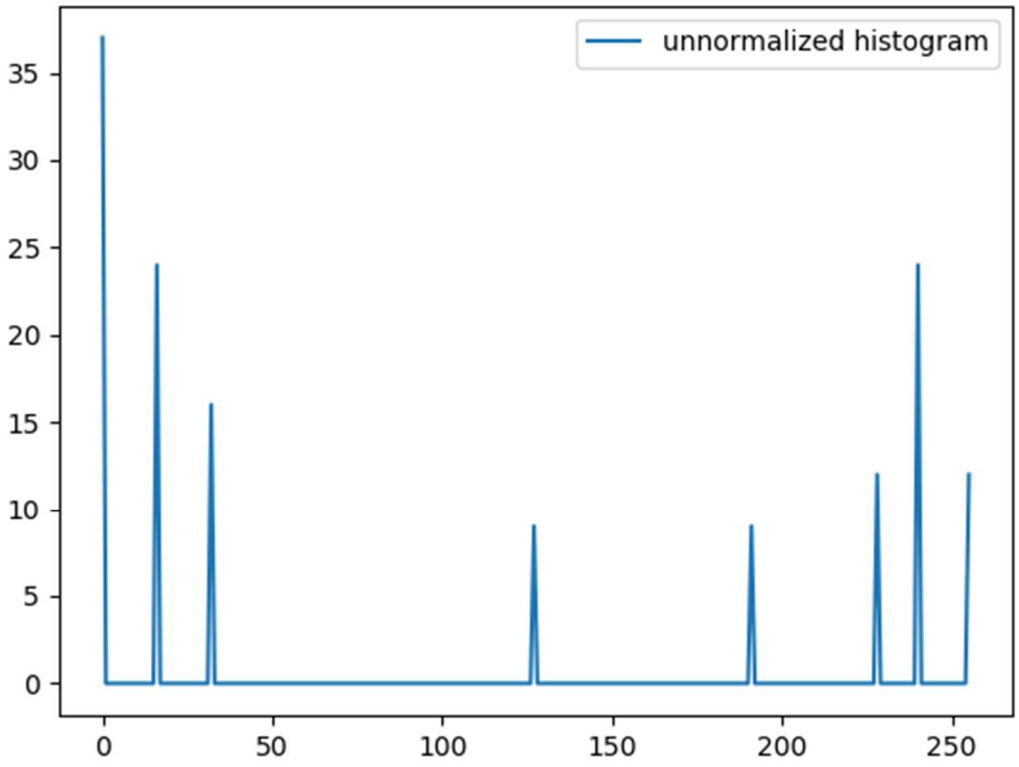
```
figure_image = np.array([
[240, 240, 240,  16,  16,  16,  32,  32,  32,  32,  32,  32,  32],
[240, 240, 240,  16,  16,  16, 228,  32,  32,  32,  32,  32, 255],
[240, 240, 240,  16,  16,  16, 228, 228,  32,  32,  32, 255, 255],
[240, 240, 240,  16,  16,  16, 228, 228, 228,  32, 255, 255, 255],
[240, 240, 240,  16,  16,  16, 228, 228, 228,   0, 255, 255, 255],
[240, 240, 240,  16,  16,  16, 228, 228,   0,   0,   0, 255, 255],
[240, 240, 240,  16,  16,  16, 228,   0,   0,   0,   0,   0, 255],
[240, 240, 240,  16,  16,  16,   0,   0,   0,   0,   0,   0,   0],
[127, 127, 127, 191, 191, 191,   0,   0,   0,   0,   0,   0,   0],
[127, 127, 127, 191, 191, 191,   0,   0,   0,   0,   0,   0,   0],
[127, 127, 127, 191, 191, 191,   0,   0,   0,   0,   0,   0,   0],
])
```

Although the triangles weren't perfect. It is very close to the image referenced in the book.



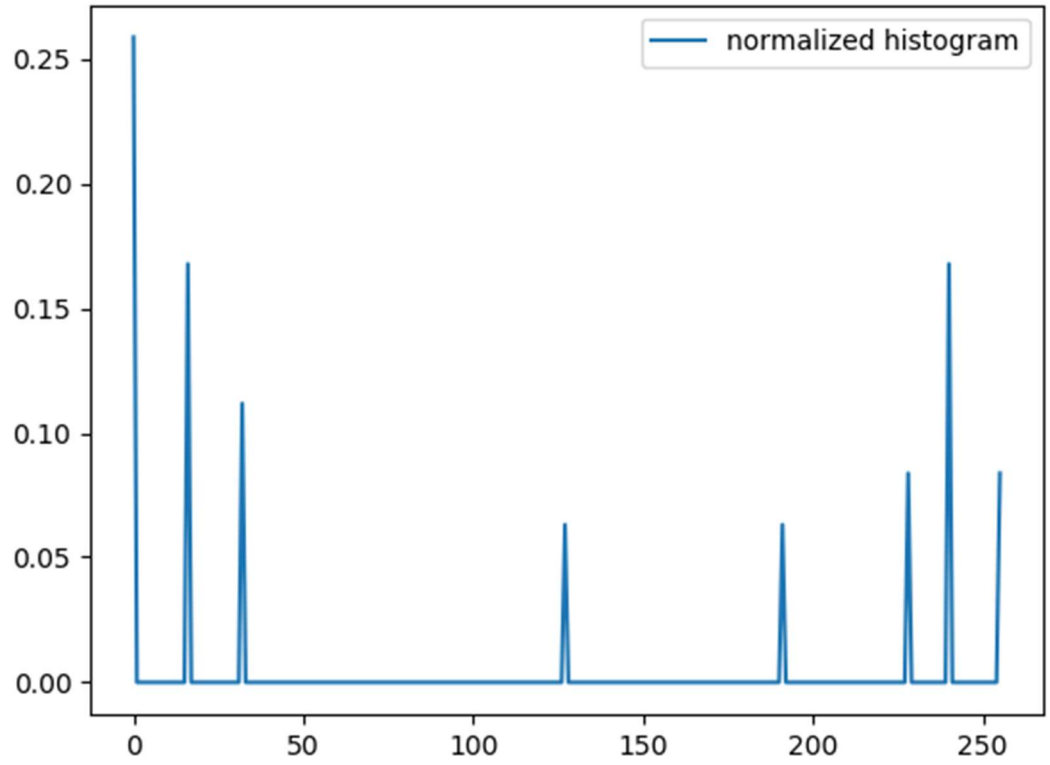
Unnormalized Histogram

The histogram for the image was calculated by iterating over the pixel values and keeping an array of bin counts for each possible pixel value.


unnormalized histogram
35
30
25
20
15
10
5
0
0
50
100
150
200
250

Normalized Histogram

The normalized histogram was obtained from the unnormalized histogram by dividing each bin count by the number of rows times the number of columns. As you can see the shape of the histogram did not change but the magnitude of each bin was normalized to match a probability density function.



3.14 AN IMAGE WITH INTENSITIES IN THE RANGE [0, 1] HAS THE PDF, P(R), SHOWN IN THE FOLLOWING FIGURE. IT IS DESIRED TO TRANSFORM THE INTENSITY LEVELS OF THIS IMAGE SO THAT THEY WILL HAVE THE SPECIFIED P(Z) SHOWN IN THE FIGURE. ASSUME CONTINUEOUS QUANTITES AND FIND THE TRANSFORM THAT WILL ACCOMPLISH THIS.

The transform function would be $z = L - 1 - r$ where L is the number of intensity levels in the image.

3.19 THE IMAGE IN FIG 3.30© IS OF SIZE 1000 X 683. ITS HISTOGRAM, LABELED "B" IN FIG. 3.30(B), WAS INTENTED TO BE UNIFORM, BUT THE RESULT IS NOT QUITE UNIFORM (OBSERVE THE SHAPE CHANGE AT THE LOW END OF THE INTENSITY SCALE). HOW DO YOU EXPLAIN THIS?

This is because of the relatively large number of pixels within the low end of the intensity scale. Histogram equalization utilizes the cumulative density function of the entire image which is dependent on pixel densities.