Netaji Subhas University of Technology

ITD12

Image Processing

PRACTICAL FILE

2016UIT**2611**

Lakshay Mehra

INDEX

1. Color image to RGB channels and Grayscale image

April 24, 2019

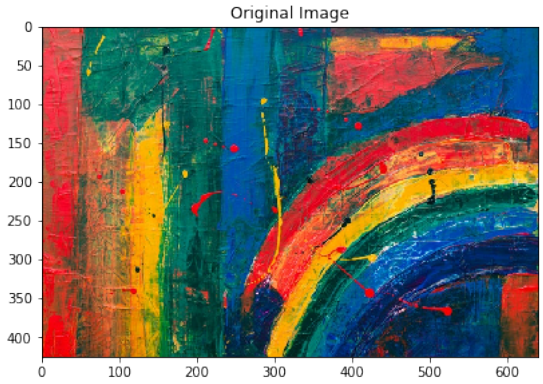
In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

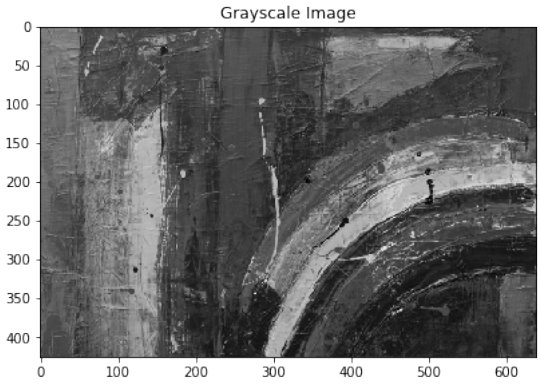
In [2]:
```
cimg = cv2.imread("images/image.jpg", cv2.IMREAD_COLOR)
cimg[:, :, 0], cimg[:, :, 2] = cimg[:, :, 2], cimg[:, :, 0].copy()
```

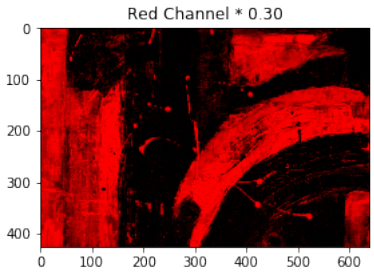
In [3]:
```
red, blue, green = cimg.copy(), cimg.copy(), cimg.copy()
red[:, :, 1:] = green[:, :, 0] = green[:, :, 2] =  blue[:, :, :2] = 0
```

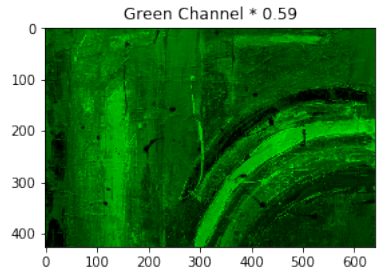
In [4]:
```
gimg = np.asarray(0.30 * red[:, :, 0] + \
                  0.59 * green[:, :, 1] + \
                  0.11 * blue[:, :, 2], dtype=np.int)
```

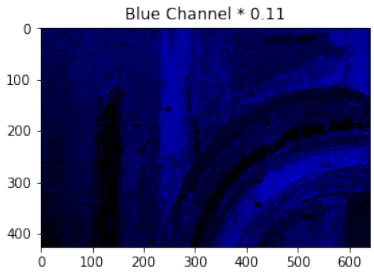
In [7]:
```
fig = plt.figure(figsize=(15, 10))

fig.add_subplot(221)
plt.imshow(cimg)
plt.title("Original Image")

fig.add_subplot(222)
plt.imshow(gimg, cmap="gray")
plt.title("Grayscale Image")

fig.add_subplot(234)
plt.imshow(red)
plt.title("Red Channel * 0.30")

fig.add_subplot(235)
plt.imshow(green)
plt.title("Green Channel * 0.59")

fig.add_subplot(236)
plt.imshow(blue)
plt.title("Blue Channel * 0.11")

plt.show()
```


Original Image
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


Grayscale Image
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


Red Channel * 0.30
0
100
200
300
400
0
100
200
300
400
500
600


Green Channel * 0.59
0
100
200
300
400
0
100
200
300
400
500
600


Blue Channel * 0.11
0
100
200
300
400
0
100
200
300
400
500
600

2. Distance Measure

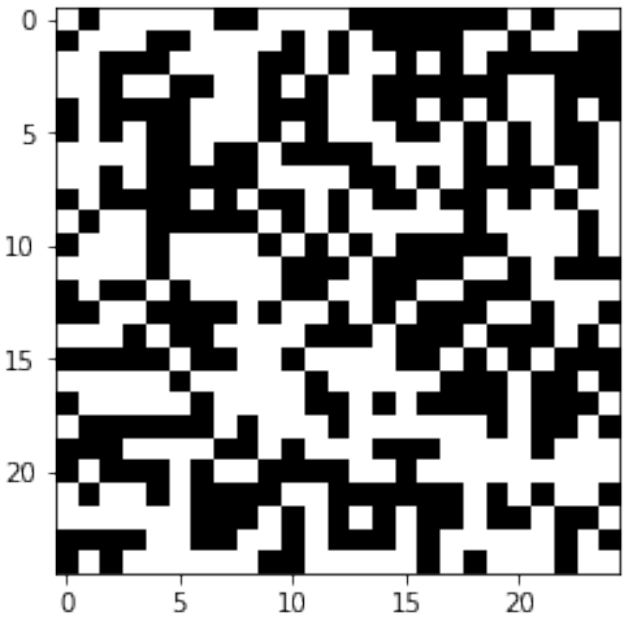
April 24, 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
class Pixel():
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

In [6]:
```
shape = (25, 25)
img = np.floor(np.random.random(shape) + 0.5)
plt.imshow(img, cmap="Greys")
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x7fe386b63e48>
```



In [7]:
```
def euclid(p1, p2):
    return ((p1.x - p2.x) ** 2 + (p1.y - p2.y) ** 2) ** 0.5
```

```
        def manhattan(p1, p2):
            return abs(p1.x - p2.x) + abs(p1.y - p2.y)

        def chessboard(p1, p2):
            return max(abs(p1.x - p2.x), abs(p1.y - p2.y))
```

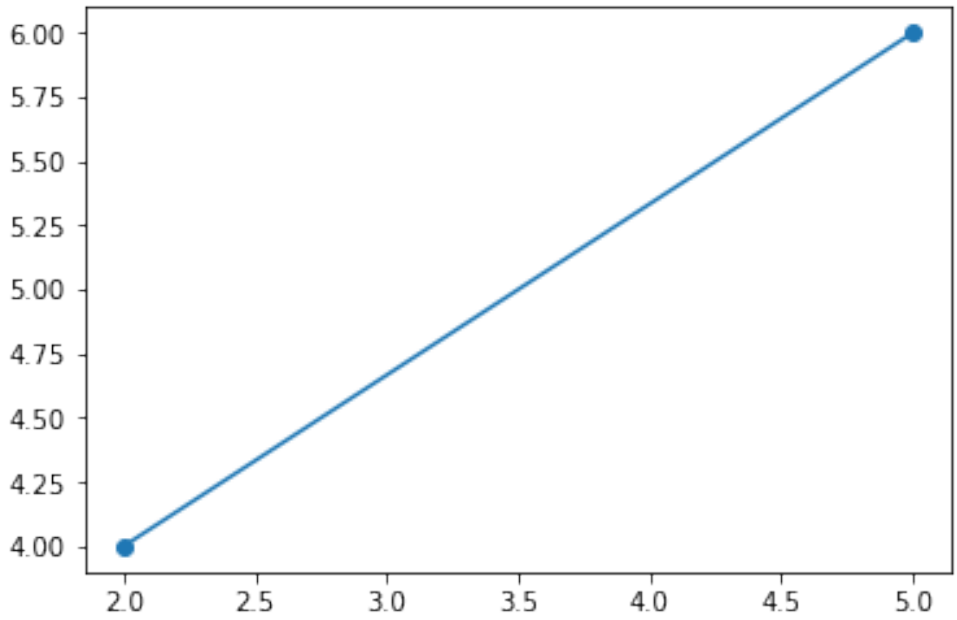
```
In [9]: x, y = [int(x) for x in input("Pixel 1: ").split()]
        p = Pixel(x, y)

        x, y = [int(x) for x in input("Pixel 2: ").split()]
        q = Pixel(x, y)
```

```
Pixel 1:  2 4
Pixel 2:  5 6
```

```
In [17]: print(euclid(p, q))

         plt.plot([p.x, q.x], [p.y, q.y], marker='o')
```

```
3.605551275463989
```

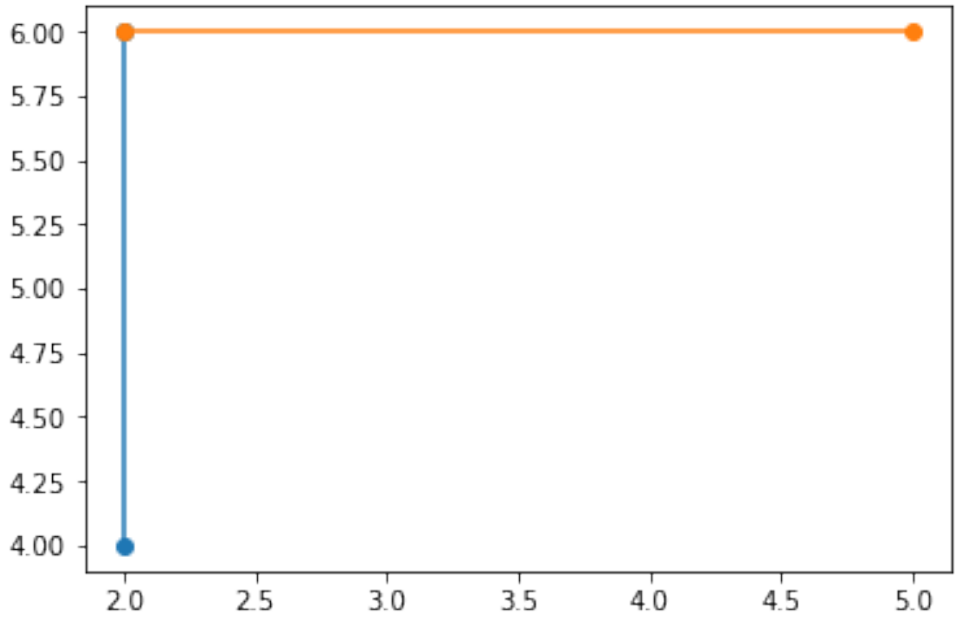
```
Out[17]: [<matplotlib.lines.Line2D at 0x7fe3868c9240>]
```




```
In [20]: print(manhattan(p, q))

         plt.plot([p.x, p.x], [p.y, q.y], marker='o')
         plt.plot([q.x, p.x], [q.y, q.y], marker='o')
```

5

Out[20]: [<matplotlib.lines.Line2D at 0x7fe3867c19b0>]



In [21]: print(chessboard(p, q))

3

3. Steganography

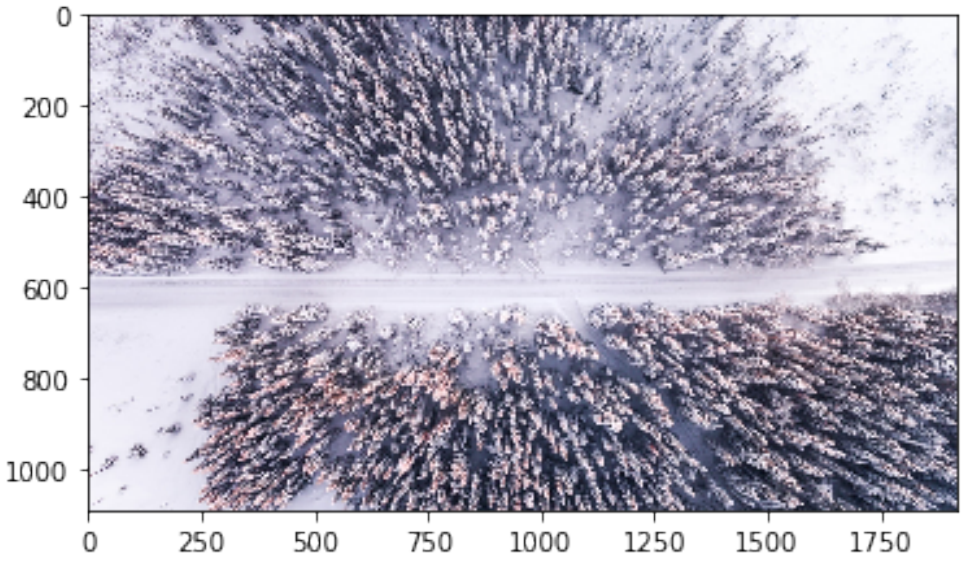
April 24, 2019

```
In [1]: import numpy as np
        import cv2
        import matplotlib.pyplot as plt
```

0.1 Reading Large Image

```
In [2]: img = cv2.imread('large.jpeg')
        img = np.array(img)
        img[:, :, 0], img[:, :, 2] = np.array(img[:, :, 2]), np.array(img[:, :, 0])
        plt.imshow(img)
        # img.shape
```

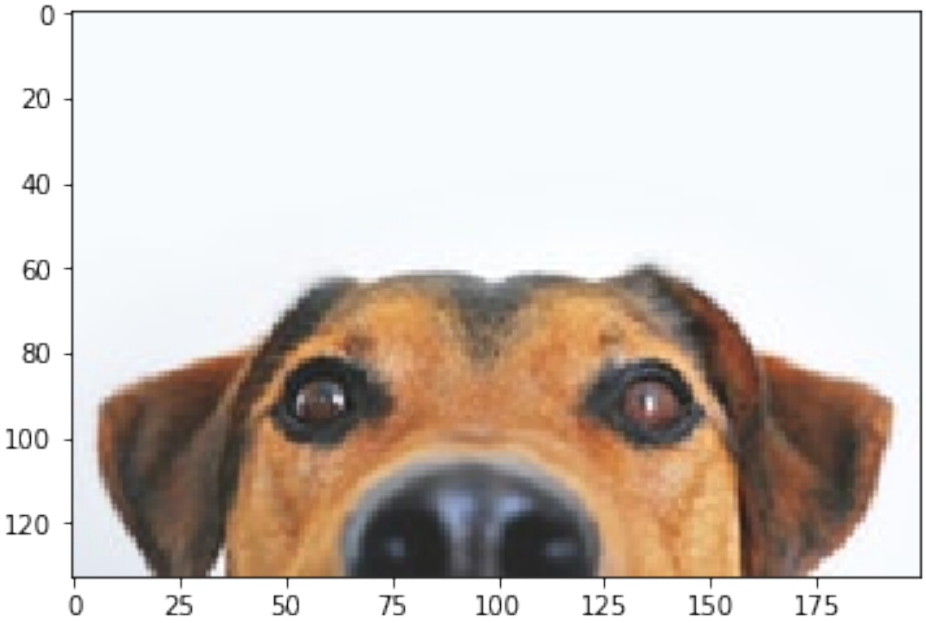
```
Out[2]: <matplotlib.image.AxesImage at 0x7ff838aa1e80>
```



0.2 Reading Small Image

```
In [3]: img2 = cv2.imread('small.jpeg')
        img2 = np.array(img2)
        img2[:, :, 0], img2[:, :, 2] = img2[:, :, 2], img2[:, :, 0].copy()
        plt.imshow(img2)
```

Out[3]: <matplotlib.image.AxesImage at 0x7ff8389cc940>



In [4]:
```
print(img.shape)
print(img2.shape)
```

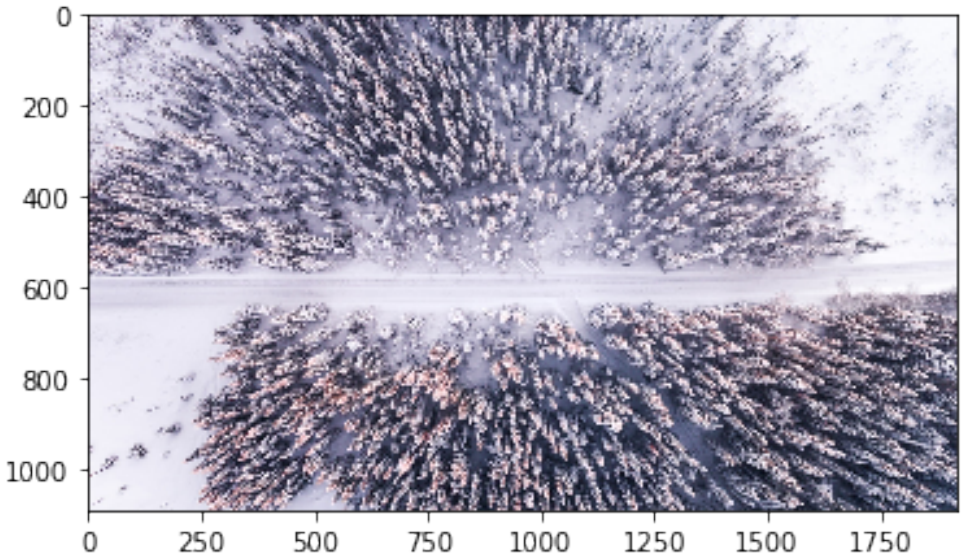
```
(1092, 1920, 3)
(133, 200, 3)
```

In [5]:
```
def hiding_image(img, img2):
    for k in range(3):
        for i in range(img2.shape[0]):
            col = 0
            for j in range(img2.shape[1]):
                c = 0
                while c != 8:
                    if img[i][col][k] % 2:
                        img[i][col][k] -= 1
                    img[i][col][k] += img2[i][j][k] % 2
                    img2[i][j][k] //= 2
                    col = col + 1
                    c = c + 1
    return img
```

In [6]:
```
steg = img.copy()
to_hide = img2.copy()
steg = hiding_image(steg,  to_hide)
```

In [7]:
```
plt.imshow(steg)
```

Out[7]: <matplotlib.image.AxesImage at 0x7ff83813f2b0>



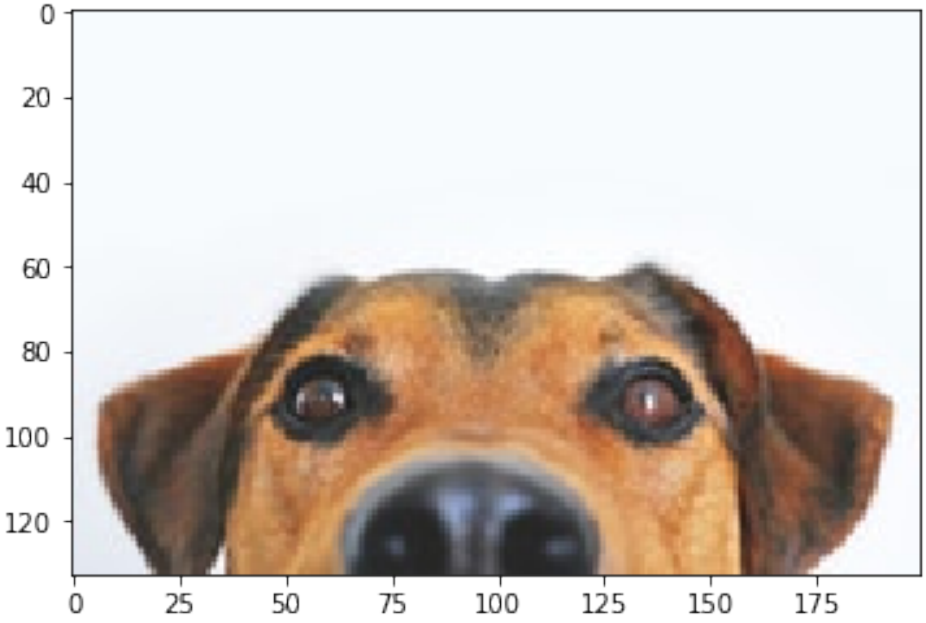
0.3 Extracting Image

```
In [8]: def extraction(steg, shape):
            ext = np.zeros(shape, dtype = int)
            for k in range(3):
                for i in range(shape[0]):
                    for j in range(shape[1] * 8):
                        if j % 8 == 0:
                            pro = 1
                        else:
                            pro *= 2
                        ext[i][j // 8][k] += steg[i][j][k] % 2 * pro
            return ext
```

```
In [9]: # Extracting from steganographic image
        extract = extraction(steg, img2.shape)
```

```
In [10]: plt.imshow(extract)
```

Out[10]: <matplotlib.image.AxesImage at 0x7ff8381252e8>

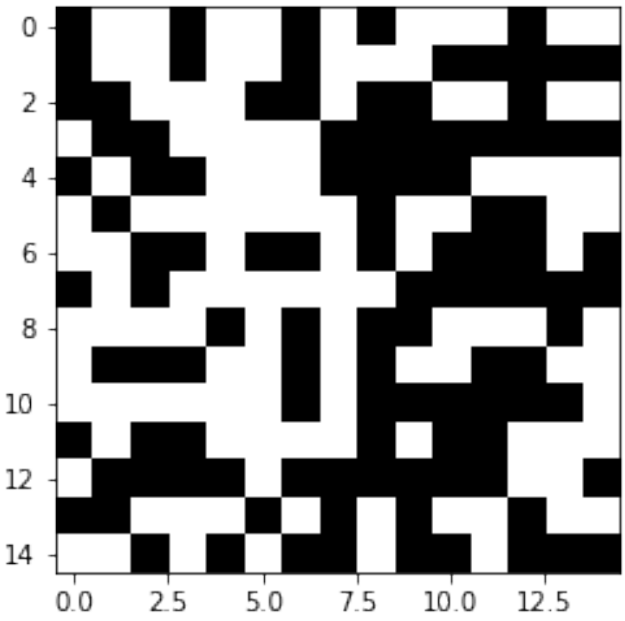

0
20
40
60
80
100
120
0
25
50
75
100
125
150
175

4. Image connectivity

Lakshay Mehra

April 24, 2019

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: dim = (15, 15)

In [3]: # Generating a random image
        img = np.floor(np.random.random(dim) + 0.5)

In [4]: plt.imshow(img, cmap="Greys")

Out[4]: <matplotlib.image.AxesImage at 0x7f973628d588>
```



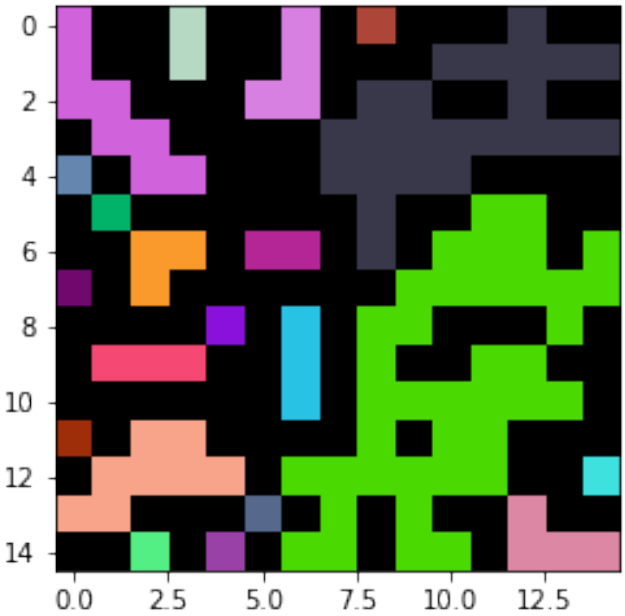
0.1 Four connected

```
In [5]: vis = np.zeros(dim, dtype=bool)

In [6]: def four_way(out, i, j, color):
```

```
        if i < 0 or i >= dim[0]:
            return
        if j < 0 or j >= dim[1]:
            return
        if vis[i][j] or img[i][j] == 0:
            return

        vis[i][j] = True
        out[i][j] = color

        four_way(out, i - 1, j, color)
        four_way(out, i + 1, j, color)
        four_way(out, i, j - 1, color)
        four_way(out, i, j + 1, color)
```

In [7]:
```
out = np.zeros(dim + (3, ), dtype=int)

for i in range(dim[0]):
    for j in range(dim[1]):

        if vis[i][j]:
            continue

        color = np.random.randint(0, 255, 3)

        four_way(out, i, j, color)

plt.imshow(out)
```

Out[7]: <matplotlib.image.AxesImage at 0x7f9736238160>



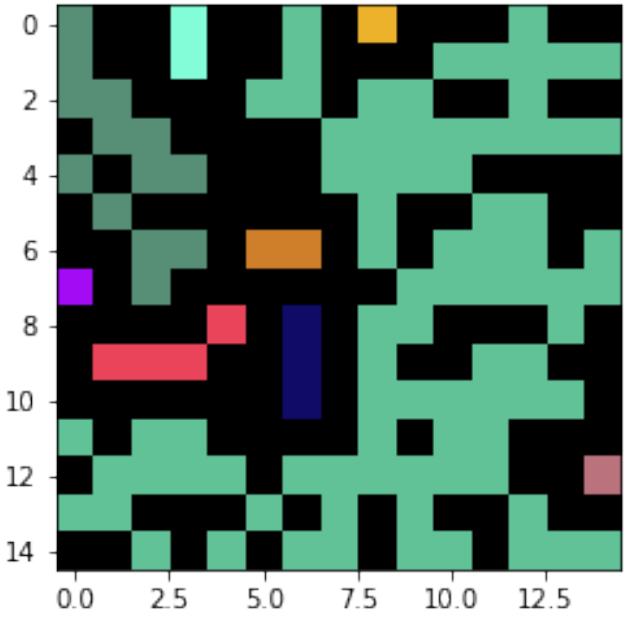
0.2 Eight connected

```
In [8]: vis = np.zeros(dim, dtype=bool)
```

```
In [9]: def eight_way(out, i, j, color):

            if i < 0 or i >= dim[0]:
                return
            if j < 0 or j >= dim[1]:
                return
            if vis[i][j] or img[i][j] == 0:
                return

            vis[i][j] = True
            out[i][j] = color

            eight_way(out, i - 1, j, color)
            eight_way(out, i + 1, j, color)
            eight_way(out, i, j - 1, color)
            eight_way(out, i, j + 1, color)

            eight_way(out, i - 1, j - 1, color)
            eight_way(out, i - 1, j + 1, color)
            eight_way(out, i + 1, j - 1, color)
            eight_way(out, i + 1, j + 1, color)
```

```
In [10]: out = np.zeros(dim + (3, ), dtype=int)

         for i in range(dim[0]):
             for j in range(dim[1]):

                 if vis[i][j]:
                     continue

                 color = np.random.randint(0, 255, 3)

                 eight_way(out, i, j, color)

         plt.imshow(out)
```

```
Out[10]: <matplotlib.image.AxesImage at 0x7f97361a94a8>
```

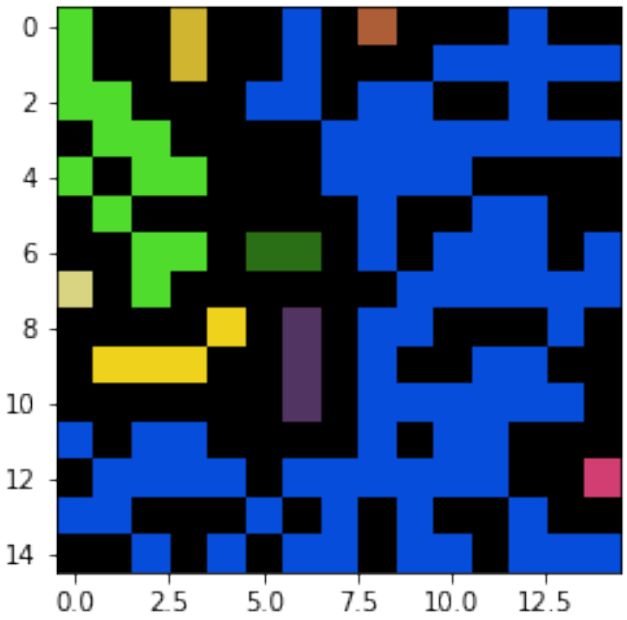


0.3 m-connected

```
In [11]: vis = np.zeros(dim, dtype=bool)

In [12]: def m_way(out, i, j, color):

             if i < 0 or i >= dim[0]:
                 return
             if j < 0 or j >= dim[1]:
                 return
             if vis[i][j] or img[i][j] == 0:
                 return

             vis[i][j] = True
             out[i][j] = color

             m_way(out, i - 1, j, color)
             m_way(out, i + 1, j, color)
             m_way(out, i, j - 1, color)
             m_way(out, i, j + 1, color)

             m_way(out, i - 1, j - 1, color)
             m_way(out, i - 1, j + 1, color)
             m_way(out, i + 1, j - 1, color)
             m_way(out, i + 1, j + 1, color)

In [13]: out = np.zeros(dim + (3, ), dtype=int)
```

```
        for i in range(dim[0]):
            for j in range(dim[1]):

                if vis[i][j]:
                    continue

                color = np.random.randint(0, 255, 3)

                m_way(out, i, j, color)

        plt.imshow(out)
```

Out[13]: <matplotlib.image.AxesImage at 0x7f973610e2e8>



5. Component Labeling Algorithm

April 24, 2019

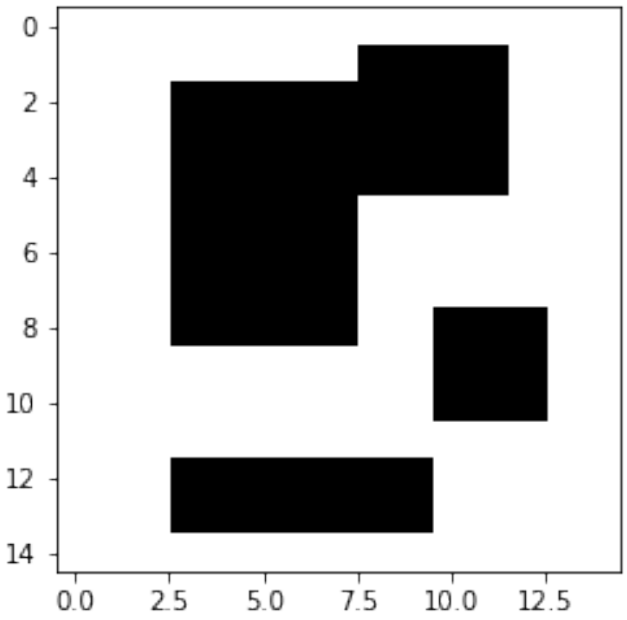
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
dim = (15, 15)
img = np.zeros(dim).astype(int)

img[2:9, 3:8] = 1
img[2:5, 3:12] = 1
img[1:4, 8:12] = 1

img[12:14, 3:10] = 1

img[8:11, 10:13] = 1
print(img)
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 1 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

In [3]:
```
plt.imshow(img, cmap="Greys")
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x7f28ef990710>
```



In [4]:
```
def component_labelling(img_in):
    mapping = {}
    label = 0

    img = img_in.copy()

    def up(i, j):
        if (i == 0):
            return 0
        else:
            return img[i - 1][j]

    def left(i, j):
        if (j == 0):
            return 0
        else:
            return img[i][j - 1]

    # First Pass

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):

            if img[i][j] == 0:
                continue

            if (up(i, j) == 0 and left(i, j) == 0):
                label += 1
```

```
                    img[i][j] = label
                elif (up(i, j) != 0 and left(i, j) == 0):
                    img[i][j] = img[i - 1][j]
                elif (up(i, j) == 0 and left(i, j) != 0):
                    img[i][j] = img[i][j - 1]
                else:
                    a = img[i - 1][j]
                    b = img[i][j - 1]

                    if a != b:
                        a, b = min(a, b), max(a, b)
                        mapping[b] = a

                    img[i][j] = a

        first_out = img.copy()

        # Second Pass

        for i in range(img.shape[0]):
            for j in range(img.shape[1]):

                p = img[i][j]

                if p in mapping.keys():
                    img[i][j] = mapping[p]

        second_out = img.copy()

        return first_out, second_out
```

```
In [5]: first, second = component_labelling(img)
        print("\nAfter first pass:\n")
        print(first)

        print("\nAfter second pass:\n")
        print(second)
```

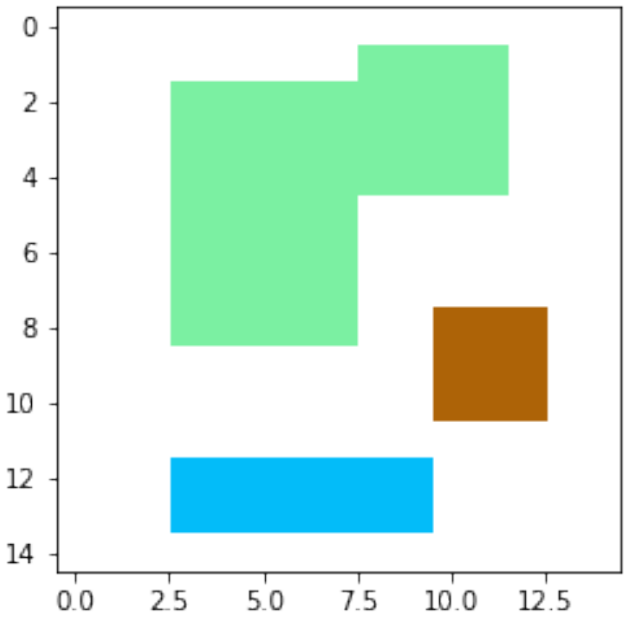
```
After first pass:

[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 2 2 2 2 2 1 1 1 1 0 0 0 0]
 [0 0 0 2 2 2 2 2 1 1 1 1 0 0 0 0]
 [0 0 0 2 2 2 2 2 1 1 1 1 0 0 0 0]
 [0 0 0 2 2 2 2 2 0 0 0 0 0 0 0 0]
 [0 0 0 2 2 2 2 2 0 0 0 0 0 0 0 0]
 [0 0 0 2 2 2 2 2 0 0 0 0 0 0 0 0]
 [0 0 0 2 2 2 2 2 0 0 3 3 3 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 3 3 3 0 0 0]
```

```
 [0 0 0 0 0 0 0 0 0 0 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 4 4 4 4 4 4 4 0 0 0 0 0]
 [0 0 0 4 4 4 4 4 4 4 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]

After second pass:

[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 0 0 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 0 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 0 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 4 4 4 4 4 4 4 0 0 0 0 0]
 [0 0 0 4 4 4 4 4 4 4 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

In [8]:
```
img = second
color = np.zeros(img.shape + (3, )).astype(int)

mapping = { 0: [255, 255, 255] }

for i in range(img.shape[0]):
        for j in range(img.shape[1]):

            val = img[i][j]

            if val in mapping.keys():
                color[i][j]  = mapping[val]
            else:
                mapping[val] = np.random.randint(0, 255, 3)
                color[i][j]  = mapping[val]
```

In [9]:
```
plt.imshow(color)
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x7f28ef8a1fd0>
```


0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5

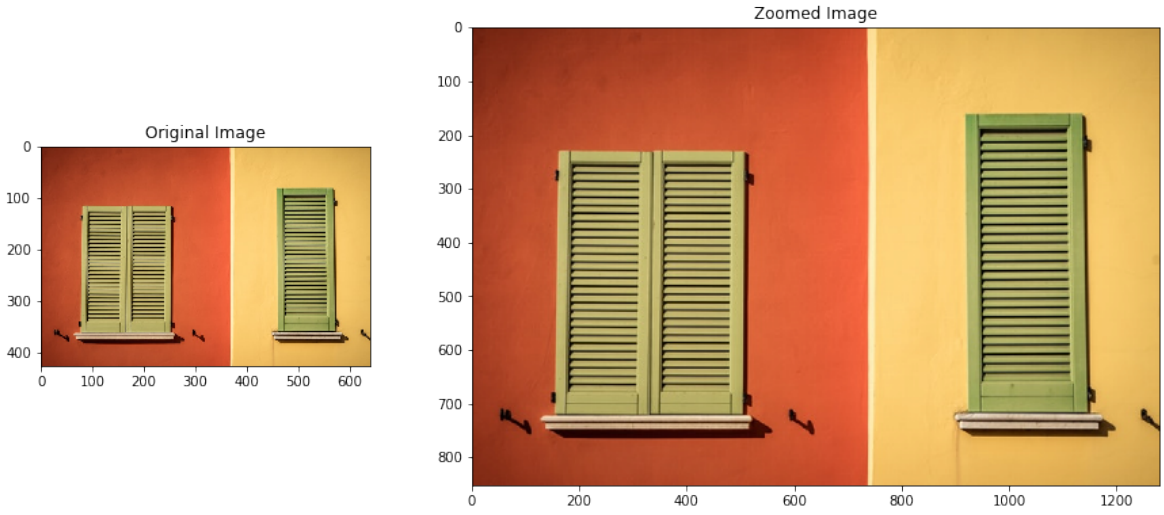
6. Zooming and Shrinking

April 24, 2019

```
In [1]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

In [2]: img = cv2.imread("image.jpeg", cv2.IMREAD_COLOR)
        img[:,:,0], img[:,:,2] = img[:,:,2], img[:,:,0].copy()

In [3]: out = np.zeros((img.shape[0] * 2, img.shape[1] * 2, img.shape[2]), dtype=np.uint8)
```

0.1 Zooming

```
In [4]: for i in range(img.shape[0]):
            out[2 * i, ::2] = np.copy(img[i, :])

In [5]: for k in range(out.shape[2]):
            for i in range(img.shape[0] - 1):
                for j in range(out.shape[1]):

                    out[2 * i + 1][j][k] = out[2 * i][j][k] // 2 + out[2 * (i + 1)][j][k] // 2

In [6]: for k in range(out.shape[2]):
            for i in range(out.shape[0]):
                for j in range(img.shape[1] - 1):
                    out[i][2 * j + 1][k] = out[i][2 * j][k] // 2 + out[i][2 * (j + 1)][k] // 2

In [7]: out[-1, :, :] = np.copy(out[-2, :, :])
        out[:, -1, :] = np.copy(out[:, -2, :])

In [8]: fig = plt.figure(figsize = (20, 20))

        ax = fig.add_subplot(142)
        ax.imshow(img)
        ax.set_title('Original Image')

        bx = fig.add_subplot(122)
        bx.imshow(out)
        bx.set_title('Zoomed Image')

        plt.show()
```



0.2 Shrinking

```
In [9]: out_s = np.copy(img[::2, ::2, :])
```

```
In [10]: fig = plt.figure(figsize = (20, 20))

         ax = fig.add_subplot(121)
         ax.imshow(img)
         ax.set_title('Original Image')

         bx = fig.add_subplot(143)
         bx.imshow(out_s)
         bx.set_title('Shrinked Image')

         plt.show()
```

7. Contrast Stretching

Lakshay Mehra

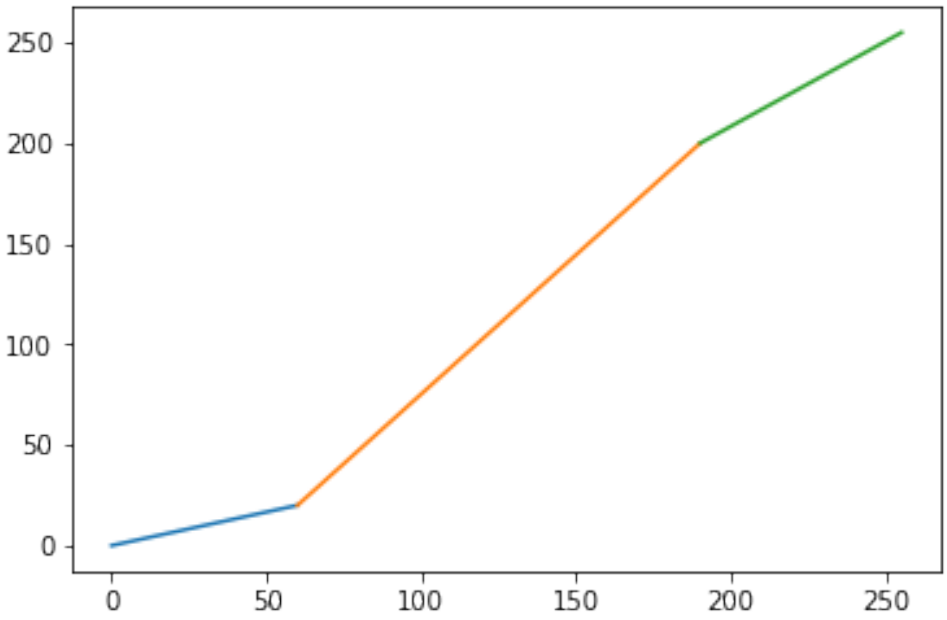
April 24, 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
img = cv2.imread("low_contrast.jpeg", cv2.IMREAD_COLOR)
img[:,:,0], img[:,:,2] = img[:,:,2], img[:,:,0].copy()
```

In [3]:
```
def compute(x, r1, s1, r2, s2):

    if x <= r1:
        return int(s1 / r1 * x)
    elif x <= r2:
        return int((s2 - s1) / (r2 - r1) * (x - r1) + s1)
    else:
        return int((255 - s2) / (255 - r2) * (x - r2) + s2)
```

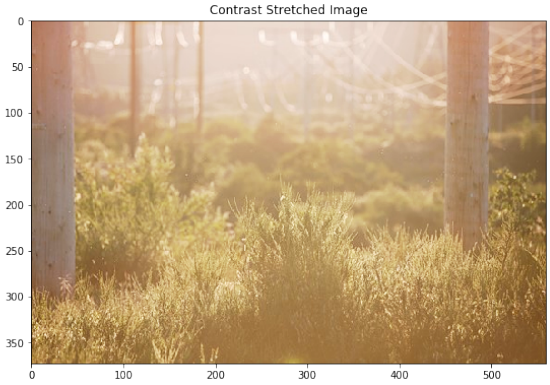
In [8]:
```
r1, s1 = 60, 20
r2, s2 = 190, 200

plt.plot(np.linspace(0, r1, 1000), np.linspace(0, s1, 1000))
plt.plot(np.linspace(r1, r2, 1000), np.linspace(s1, s2, 1000))
plt.plot(np.linspace(r2, 255, 1000), np.linspace(s2, 255, 1000))
plt.show()
```




```
In [9]: img_s = np.zeros(img.shape, dtype=np.int)

        for k in range(3):
            for i in range(img.shape[0]):
                for j in range(img.shape[1]):

                    img_s[i][j][k] = compute(img[i][j][k], r1, s1, r2, s2)
```

```
In [10]: fig = plt.figure(figsize = (20, 10))

         ax = fig.add_subplot(121)
         ax.imshow(img)
         ax.set_title('Original Image')

         bx = fig.add_subplot(122)
         bx.imshow(img_s)
         bx.set_title('Contrast Stretched Image')

         plt.show()
```


Original Image
0
50
100
150
200
250
300
350
0
100
200
300
400
500


Contrast Stretched Image
0
50
100
150
200
250
300
350
0
100
200
300
400
500

8. Image Negatives

Lakshay Mehra

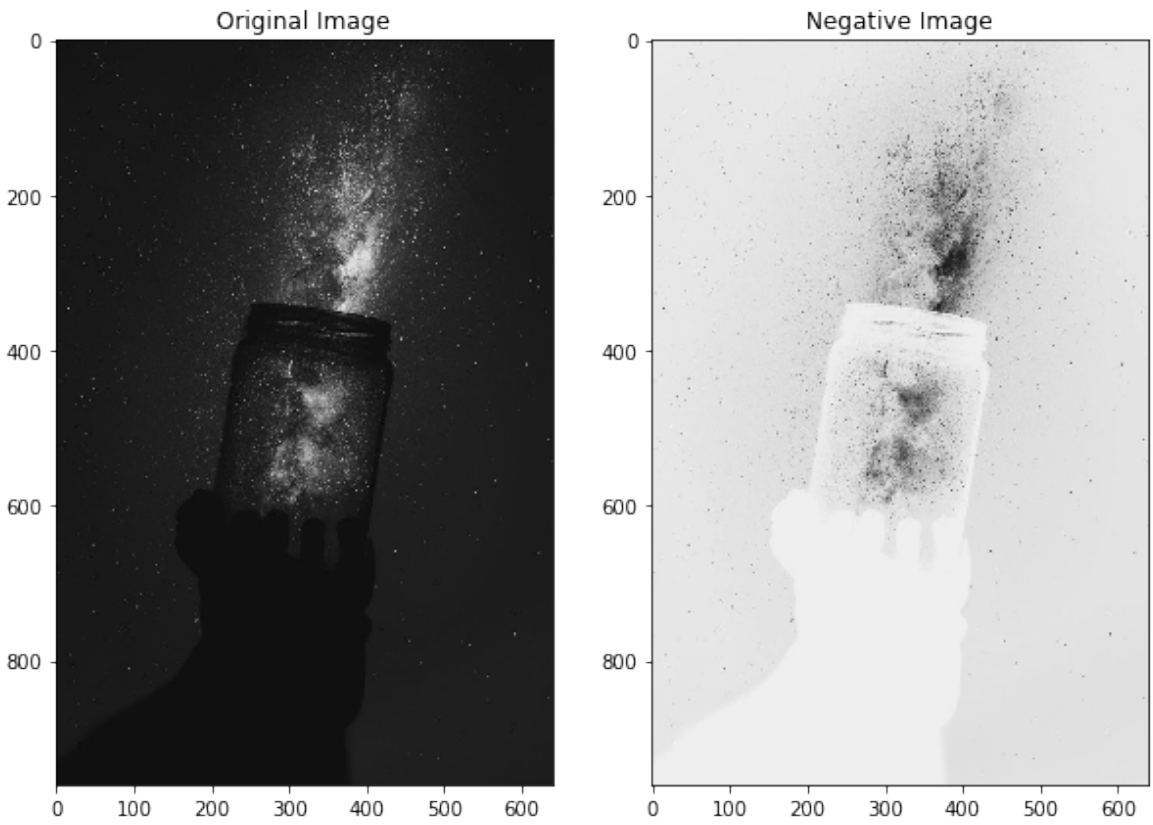
April 24, 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
img = cv2.imread("dark.jpeg", cv2.IMREAD_GRAYSCALE)
```

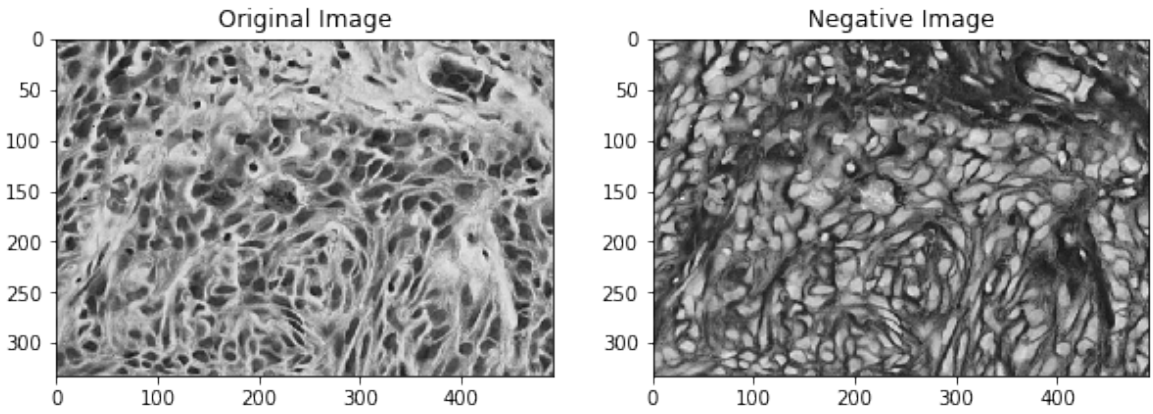
In [3]:
```
n = 8
L = 1 << n

img_neg = (L - 1) - img
```

In [4]:
```
fig = plt.figure(figsize = (10, 10))

ax = fig.add_subplot(121)
ax.imshow(img, cmap="gray")
ax.set_title('Original Image')

bx = fig.add_subplot(122)
bx.imshow(img_neg, cmap="gray")
bx.set_title('Negative Image')

plt.show()
```



Applying image negative on the cancerous tissue

```
In [5]: img = cv2.imread("lung_cancer_tissue.png", cv2.IMREAD_GRAYSCALE)

In [6]: n = 8
        L = 1 << n

        img_neg = (L - 1) - img

In [7]: fig = plt.figure(figsize = (10, 10))

        ax = fig.add_subplot(121)
        ax.imshow(img, cmap="gray")
        ax.set_title('Original Image')

        bx = fig.add_subplot(122)
        bx.imshow(img_neg, cmap="gray")
        bx.set_title('Negative Image')

        plt.show()
```

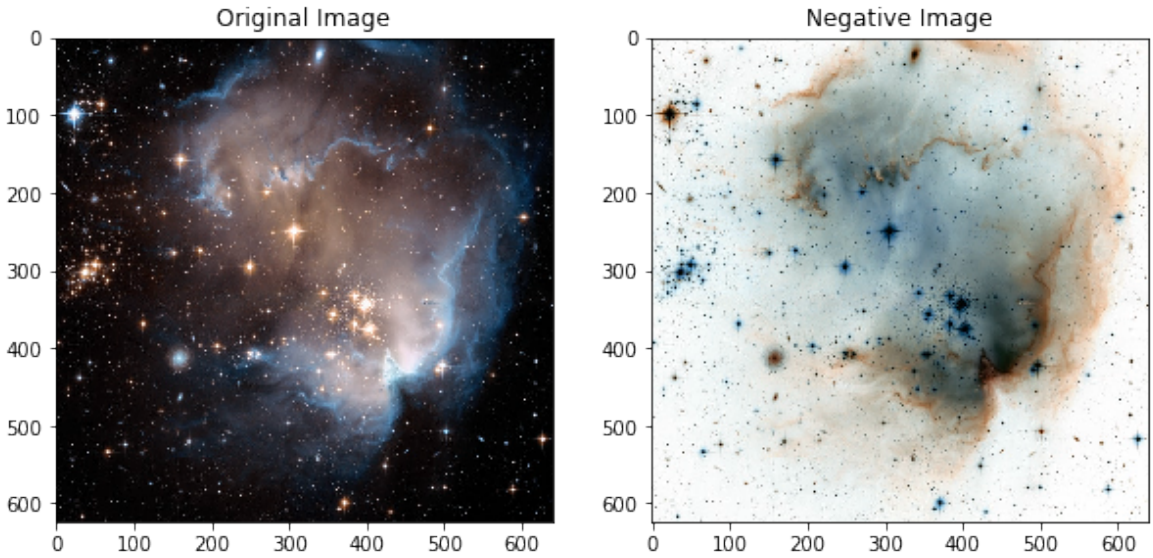


Applying image negative on a color image

```
In [8]: img = cv2.imread("galaxy.jpg", cv2.IMREAD_COLOR)
```

```
In [9]: n = 8
        L = 1 << n

        img_neg = (L - 1) - img
```

```
In [10]: fig = plt.figure(figsize = (10, 10))

         ax = fig.add_subplot(121)
         ax.imshow(img, cmap="gray")
         ax.set_title('Original Image')

         bx = fig.add_subplot(122)
         bx.imshow(img_neg, cmap="gray")
         bx.set_title('Negative Image')

         plt.show()
```



9. Low and high pass filter

April 24, 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv2.imread("images/image.jpeg", cv2.IMREAD_COLOR)
img[:, :, 0], img[:, :, 2] = img[:, :, 2], img[:, :, 0].copy()
```

In [3]:
```
l_mask = np.array([[1, 1, 1, 1, 1],
                   [1, 1, 1, 1, 1],
                   [1, 1, 1, 1, 1],
                   [1, 1, 1, 1, 1],
                   [1, 1, 1, 1, 1]])

l_scaling = 1 / 25

h_mask = np.array([[-1, -1, -1],
                   [-1,  10, -1],
                   [-1, -1, -1]])

h_scaling = 1 / 9
```

In [4]:
```
def padding(x, size):

    if size == 0:
        return x

    x = np.append(x[:, 0:1, :], x, axis=1)
    x = np.append(x, x[:, -1:, :], axis=1)
    x = np.append(x[0:1, :, :], x, axis=0)
    x = np.append(x, x[-1:, :, :], axis=0)

    return padding(x, size - 1)
```

In [5]:
```
def convolve(img, mask, scaling=1):

    '''assuming mask to be square with odd side'''

    ps = (mask.shape[0] - 1) // 2

    m = img.shape[0] + 2 * ps - mask.shape[0] + 1
```

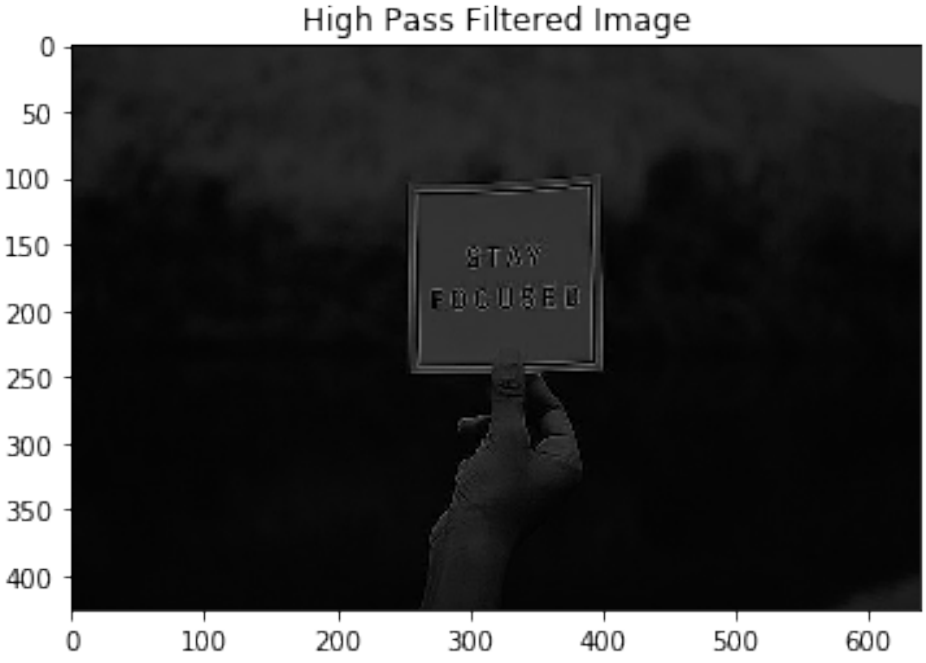
```
            n = img.shape[1] + 2 * ps - mask.shape[1] + 1

            img = padding(img, ps)

            res = np.zeros((m, n, img.shape[2]))

            for k in range(img.shape[2]):
                for i in range(ps, m - ps + 1):
                    for j in range(ps, n - ps + 1):
                        res[i][j][k] = np.sum(img[i-ps:i+ps+1, j-ps:j+ps+1, k] * mask)

            res *= scaling
            res //= 1

            res = np.minimum(255, res)
            res = np.maximum(0  , res)

            return res.astype(int)
```

In [6]:
```
out_l = convolve(img, l_mask, l_scaling)
out_h = convolve(img, h_mask, h_scaling)
```

In [7]:
```
plt.imshow(img)
plt.title("Original Image")
plt.show()

plt.imshow(out_l)
plt.title("Los Pass Filtered Image")
plt.show()

plt.imshow(out_h)
plt.title("High Pass Filtered Image")
plt.show()
```


Original Image
STAY
FOCUSED
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


Los Pass Filtered Image
STAY
FOCUSED
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


High Pass Filtered Image
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
STAY
FOCUSED

10. Gaussian Filter

Lakshay Mehra

April 24, 2019

0.1 Importing Libraries

```
In [1]: import cv2
        import numpy as np
        import matplotlib.pyplot as plt
```

0.2 Input variance and mask size

```
In [2]: variance = float(input())
```

```
2
```

```
In [3]: n = int(input())
```

```
7
```

```
In [4]: assert n % 2 == 1
```

0.3 Creating mask

```
In [5]: mask = np.zeros((n, n))
```

```
In [6]: for i in range(n):
            for j in range(n):

                x = i - n // 2;
                y = j - n // 2;

                mask[i][j] = np.exp(-1 * (x**2 + y**2) / (2 * variance))
```

```
In [7]: c = np.ceil(1 / mask[0][0])
        print(f"c = {c}")
```

```
c = 91.0
```

0.3.1 Gaussian Distribution

```
In [8]: print("Gaussian mask", end="\n\n")
        print(np.around(mask, decimals=3))
```

```
Gaussian mask

[[0.011 0.039 0.082 0.105 0.082 0.039 0.011]
 [0.039 0.135 0.287 0.368 0.287 0.135 0.039]
 [0.082 0.287 0.607 0.779 0.607 0.287 0.082]
 [0.105 0.368 0.779 1.    0.779 0.368 0.105]
 [0.082 0.287 0.607 0.779 0.607 0.287 0.082]
 [0.039 0.135 0.287 0.368 0.287 0.135 0.039]
 [0.011 0.039 0.082 0.105 0.082 0.039 0.011]]
```

0.3.2 Multiplying mask by "c"

```
In [9]: mask = np.around(mask * c).astype(int)
```

```
In [10]: print(mask)
```

```
[[ 1  4  7 10  7  4  1]
 [ 4 12 26 33 26 12  4]
 [ 7 26 55 71 55 26  7]
 [10 33 71 91 71 33 10]
 [ 7 26 55 71 55 26  7]
 [ 4 12 26 33 26 12  4]
 [ 1  4  7 10  7  4  1]]
```

0.3.3 Dividing by normalization factor

```
In [11]: n_factor = sum(sum(mask))
         print(f"Normalization Factor: {n_factor}")
```

```
Normalization Factor: 1115
```

```
In [12]: mask = mask / n_factor
```

```
In [13]: print(mask)
```

```
[[0.00089686 0.00358744 0.00627803 0.00896861 0.00627803 0.00358744
  0.00089686]
 [0.00358744 0.01076233 0.02331839 0.02959641 0.02331839 0.01076233
  0.00358744]
 [0.00627803 0.02331839 0.04932735 0.06367713 0.04932735 0.02331839
  0.00627803]
 [0.00896861 0.02959641 0.06367713 0.08161435 0.06367713 0.02959641
  0.00896861]
 [0.00627803 0.02331839 0.04932735 0.06367713 0.04932735 0.02331839
```

```
 0.00627803]
[0.00358744 0.01076233 0.02331839 0.02959641 0.02331839 0.01076233
 0.00358744]
[0.00089686 0.00358744 0.00627803 0.00896861 0.00627803 0.00358744
 0.00089686]]
```

0.4 Applying Gaussian Filter

0.4.1 Defining functions

```
In [14]: def padding(x, size):

             if size == 0:
                 return x

             x = np.append(x[:, 0:1], x, axis=1)
             x = np.append(x, x[:, -1:], axis=1)
             x = np.append(x[0:1, :], x, axis=0)
             x = np.append(x, x[-1:, :], axis=0)

             return padding(x, size - 1)
```

```
In [15]: def convolve(img, mask):

             '''assuming mask to be square with odd side'''

             ps = (mask.shape[0] - 1) // 2

             m = img.shape[0] + 2 * ps - mask.shape[0] + 1
             n = img.shape[1] + 2 * ps - mask.shape[1] + 1

             img = padding(img, ps)

             res = np.zeros((m, n))

             for i in range(ps, m - ps + 1):
                 for j in range(ps, n - ps + 1):
                     res[i][j] = np.sum(img[i-ps:i+ps+1, j-ps:j+ps+1] * mask)

             return res.astype(int)
```

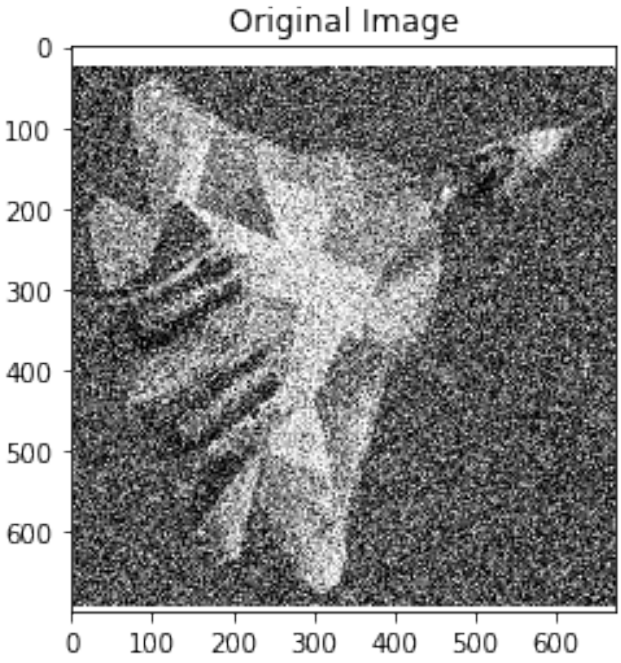
0.4.2 Filtering Image

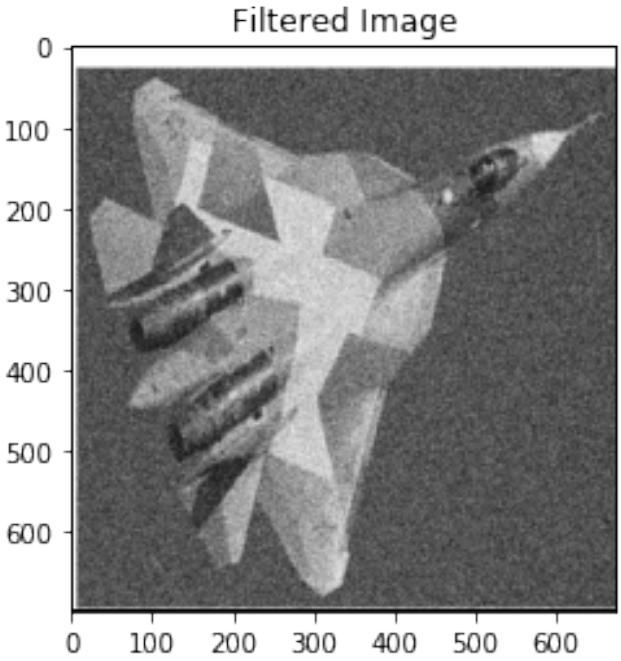
```
In [16]: img  = cv2.imread("noise.png", cv2.IMREAD_GRAYSCALE)
         img2 = convolve(img, mask)
```

0.4.3 Displaying Image

```
In [17]: plt.imshow(img, cmap='gray')
         plt.title("Original Image")
         plt.show()
```

```
plt.imshow(img2, cmap='gray')
plt.title("Filtered Image")
plt.show()
```





11. Edge Detection

April 24, 2019

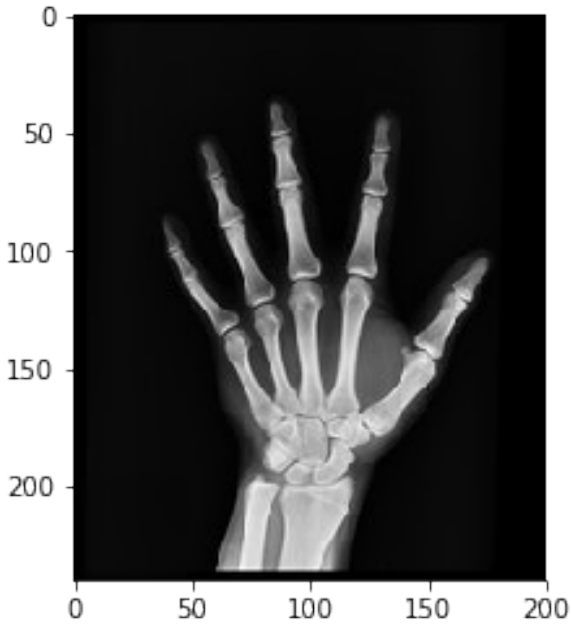
In [1]:
```
import cv2, math
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
```

In [3]:
```
img = cv2.imread("./images/bone.jpg", cv2.IMREAD_GRAYSCALE)
```

In [4]:
```
plt.imshow(img, cmap="gray")
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x7f2ca296b518>
```



In [5]:
```
def padding(x, size):

    if size == 0:
        return x
```

```
            x = np.append(x[:, 0:1], x, axis=1)
            x = np.append(x, x[:, -1:], axis=1)
            x = np.append(x[0:1, :], x, axis=0)
            x = np.append(x, x[-1:, :], axis=0)

            return padding(x, size - 1)
```

```
In [6]: def convolve(img, mask):

            '''assuming mask to be square with odd side'''

            ps = (mask.shape[0] - 1) // 2

            m = img.shape[0] + 2 * ps - mask.shape[0] + 1
            n = img.shape[1] + 2 * ps - mask.shape[1] + 1

            img = padding(img, ps)

            res = np.zeros((m, n))

            for i in range(ps, m - ps + 1):
                for j in range(ps, n - ps + 1):
                        res[i][j] = np.sum(img[i-ps:i+ps+1, j-ps:j+ps+1] * mask)

            return res.astype(int)
```

```
In [7]: Gy = convolve(img, sobel_x)
        Gx = convolve(img, sobel_y)

        angle = np.zeros_like(img)
```

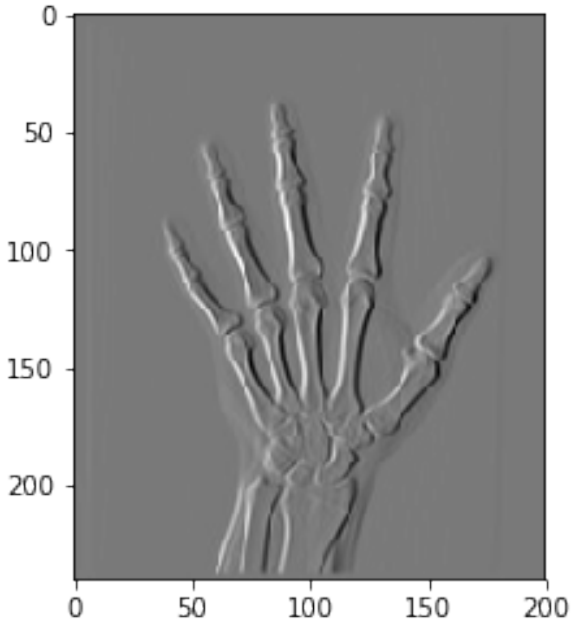
```
In [8]: for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                if Gx[i][j] == 0:
                    angle[i][j] = 90
                else:
                    angle[i][j] = int(math.atan(abs(Gy[i][j] / Gx[i][j])) * 180 / math.pi)
```

```
In [9]: angle
```

```
Out[9]: array([[90, 90, 90, ..., 90, 90, 90],
               [90, 90, 90, ..., 90, 90, 90],
               [90, 90, 45, ..., 90, 90, 90],
               ...,
               [90, 26, 59, ..., 90, 90, 90],
               [90, 26, 56, ..., 90, 90, 90],
               [90, 45, 59, ..., 90, 90, 90]], dtype=uint8)
```

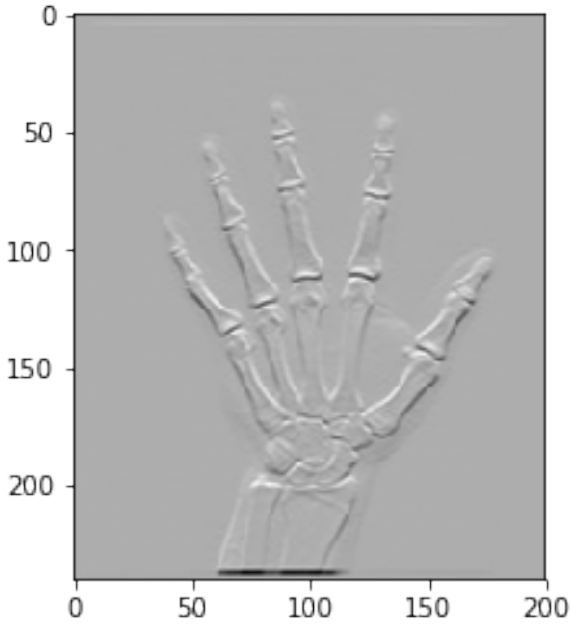
```
In [10]: plt.imshow(convolve(img, sobel_x), cmap="gray")
```

```
Out[10]: <matplotlib.image.AxesImage at 0x7f2ca28895f8>
```



In [11]:
```
plt.imshow(convolve(img, sobel_y), cmap="gray")
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x7f2ca2863588>
```



12. Histogram Equalization

April 24, 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv2.imread("images/histeq.jpg", cv2.IMREAD_GRAYSCALE)
```

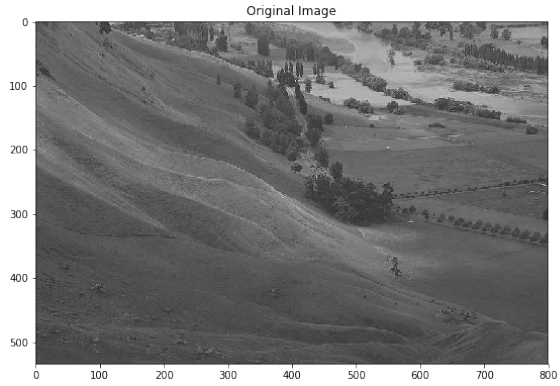
In [3]:
```
m, n = img.shape
h = [0.0] * 256

for i in range(m):
    for j in range(n):
        h[img[i][j]] += 1

h = np.array(h) / (m * n)
```

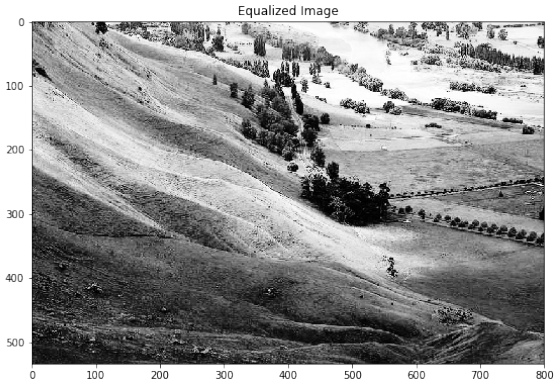
In [4]:
```
cdf = np.array([sum(h[:i+1]) for i in range(len(h))])
```

In [5]:
```
sk = np.uint8(255 * cdf)
```

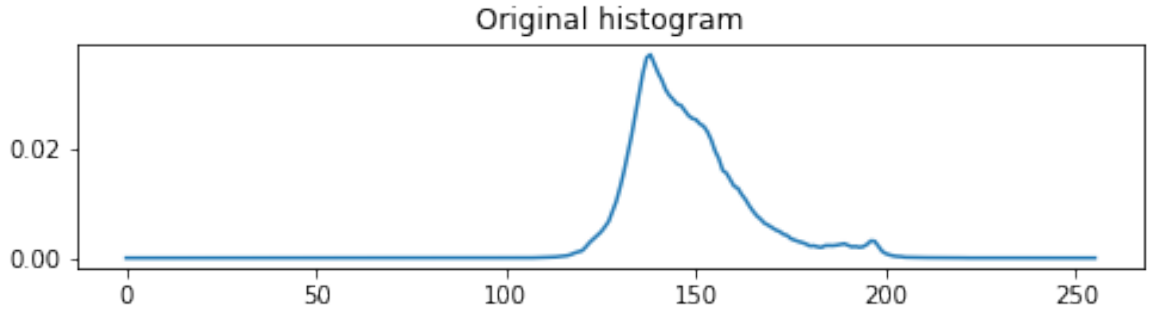
In [6]:
```
m, n = img.shape
IMG, H = np.zeros_like(img), [0.0] * 256

for i in range(m):
    for j in range(n):
        IMG[i, j] = sk[img[i, j]]
        H[IMG[i][j]] += 1

H = np.array(H) / (m * n)
```

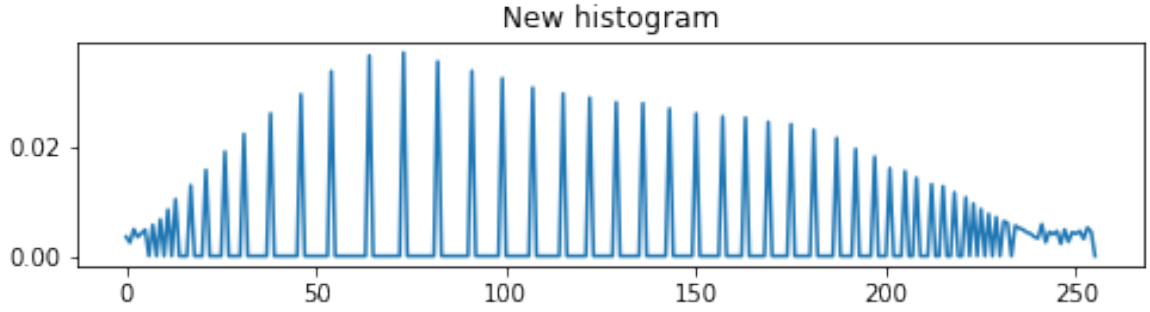
In [7]:
```
fig = plt.figure(figsize=(20, 10))

fig.add_subplot(121)
plt.imshow(img, cmap="gray")
plt.title("Original Image")

fig.add_subplot(122)
plt.imshow(IMG, cmap="gray")
plt.title("Equalized Image")

plt.show()
```

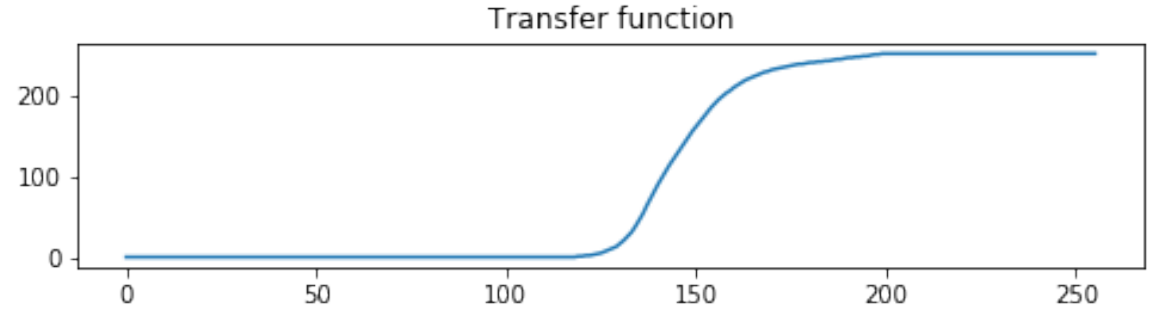




In [8]:
```
fig = plt.figure(figsize=(8,10))
fig.add_subplot(511)
plt.plot(h)
plt.title('Original histogram')

fig.add_subplot(513)
plt.plot(H)
plt.title('New histogram')

fig.add_subplot(515)
plt.plot(sk)
plt.title('Transfer function')

plt.show()
```


Original histogram
0.02
0.00
0
50
100
150
200
250


New histogram
0.02
0.00
0
50
100
150
200
250


Transfer function
200
100
0
0
50
100
150
200
250

13. Histogram Specification

April 24, 2019

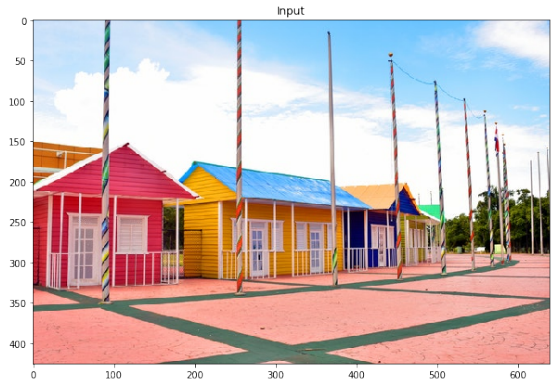
In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

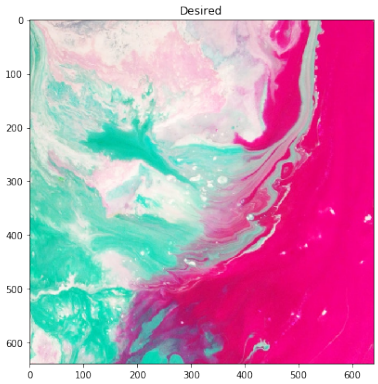
In [2]:
```
img = cv2.imread("images/input.jpg", cv2.IMREAD_COLOR)
img[:, :, 0], img[:, :, 2] = img[:, :, 2], img[:, :, 0].copy()

desired = cv2.imread("images/desired.jpg", cv2.IMREAD_COLOR)
desired[:, :, 0], desired[:, :, 2] = desired[:, :, 2], desired[:, :, 0].copy()
```

In [3]:
```
def getCvalue(IMG):
    df = [0.0] * 256

    for x in IMG.flat:
        df[x] += 1

    df = np.array(df) / np.sum(df)

    for i in range(1, 256):
        df[i] += df[i - 1]

    return np.uint8(255 * df)
```

In [4]:
```
target = np.zeros_like(img)

for k in range(img.shape[2]):

    GH = getCvalue(img[:, :, k])
    DH = getCvalue(desired[:, :, k])

    fmap = [0] * 256

    for i in range(256):
        for j in range(256):
            if DH[j] >= GH[i]:
                fmap[i] = j
                break

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            target[i][j][k] = fmap[img[i][j][k]]
```

In [5]:
```
fig = plt.figure(figsize=(25, 15))

    fig.add_subplot(221)
    plt.imshow(img)
    plt.title("Input")

    fig.add_subplot(222)
    plt.imshow(desired)
    plt.title("Desired")

    fig.add_subplot(223)
    plt.imshow(target)
    plt.title("Target")

    plt.show()
```





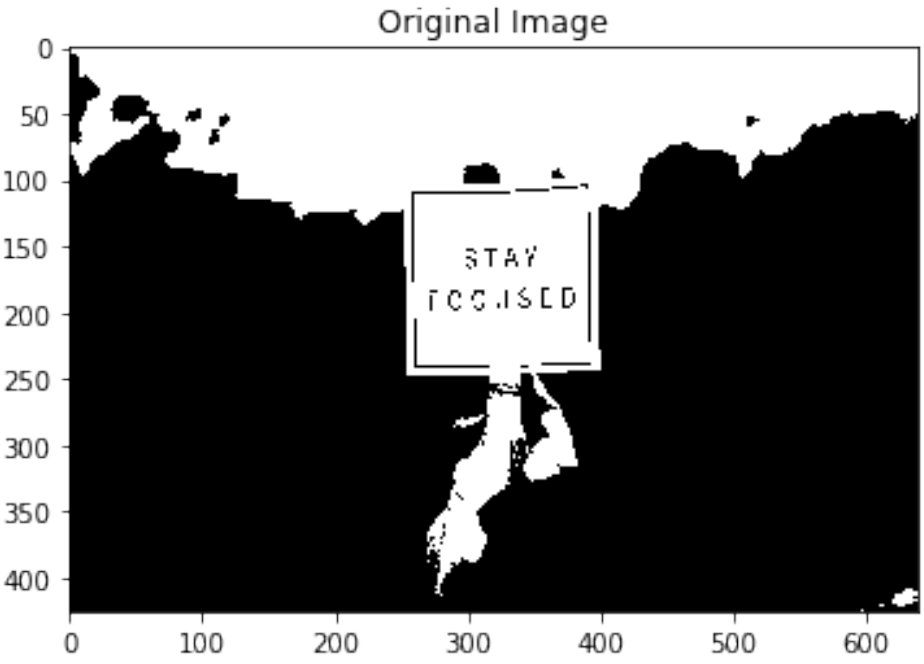
14. Dilation and Erosion

April 24, 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv2.imread("images/image.jpeg", cv2.IMREAD_GRAYSCALE)
```

In [3]:
```
img[img >= 127] = 255
img[img <  127] = 0

plt.imshow(img, cmap="gray")
plt.title("Original Image")
plt.show()
```



In [4]:
```
def brpad(img):
    img = np.append(img, img[:, -1:], axis=1)
    img = np.append(img, img[-1:, :], axis=0)
    return img
```

In [5]:
```
shape = img.shape
img = brpad(img)
```

0.1 Dilation

```
In [6]: IMGD = np.zeros(shape)
```

```
In [7]: for i in range(img.shape[0] - 1):
            for j in range(img.shape[1] - 1):
                IMGD[i, j] = 1 if np.sum(img[i:i+2, j:j+2]) // 255 else 0
```

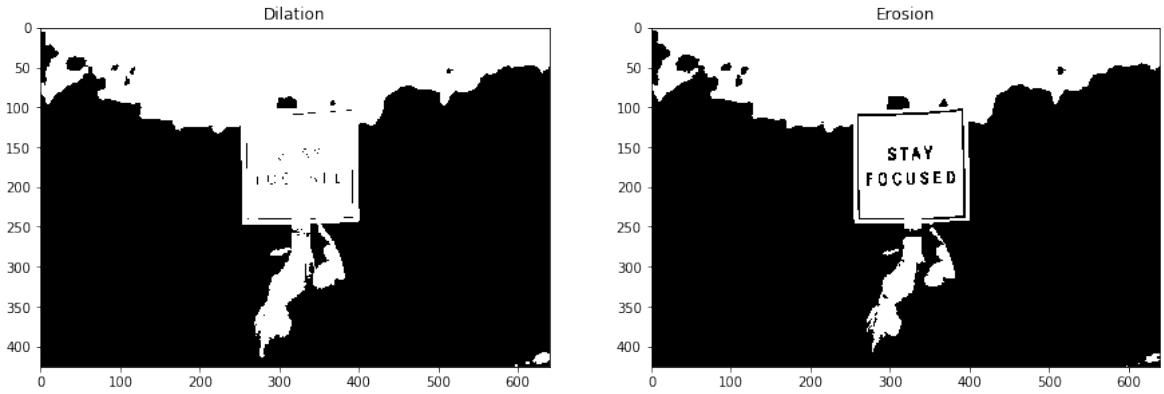
0.2 Erosion

```
In [8]: IMGE = np.zeros(shape)
```

```
In [9]: for i in range(img.shape[0] - 1):
            for j in range(img.shape[1] - 1):
                IMGE[i, j] = 1 if np.sum(img[i:i+2, j:j+2]) // 255 == 4 else 0
```

0.3 Output

```
In [10]: fig = plt.figure(figsize=(15, 10))

         fig.add_subplot(121)
         plt.imshow(IMGD, cmap="gray")
         plt.title("Dilation")

         fig.add_subplot(122)
         plt.imshow(IMGE, cmap="gray")
         plt.title("Erosion")

         plt.show()
```



15. Opening and Closing

April 24, 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv2.imread("images/morph.jpg", cv2.IMREAD_GRAYSCALE)
```

In [3]:
```
img[img >= 100] = 255
img[img <  100] = 0
```

In [4]:
```
img = np.append(img[:, 0:1], img, axis=1)
img = np.append(img, img[:, -1:], axis=1)
img = np.append(img[0:1, :], img, axis=0)
img = np.append(img, img[-1:, :], axis=0)
```

In [6]:
```
def dilation(IMG):
    m, n = IMG.shape
    RES = np.zeros((m - 1, n - 1))

    for i in range(m - 1):
        for j in range(n - 1):
            RES[i, j] = 255 if np.sum(IMG[i:i+2, j:j+2]) // 255 else 0

    return RES.astype(np.uint8)
```

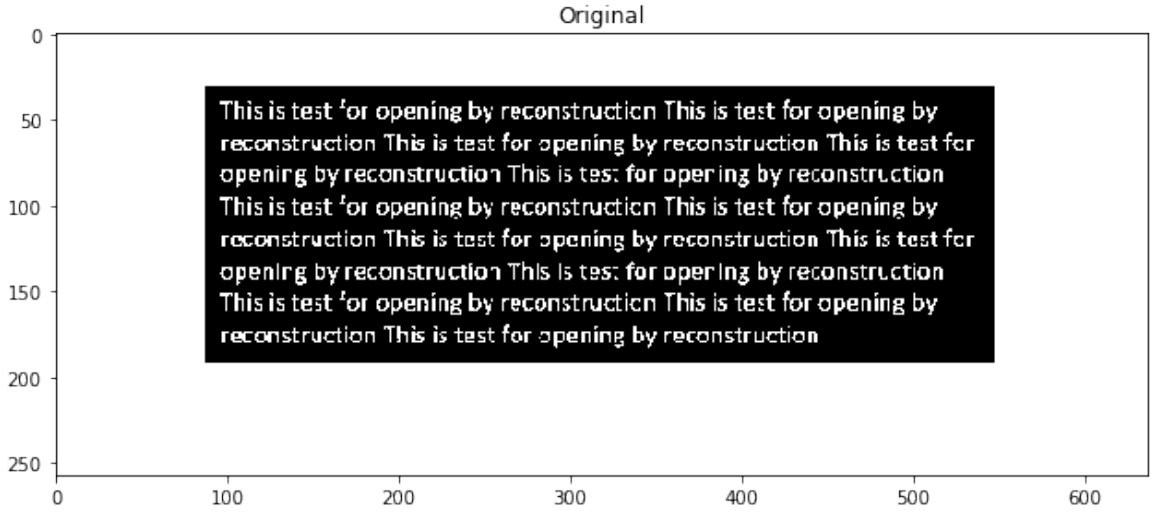
In [7]:
```
def erosion(IMG):
    m, n = IMG.shape
    RES = np.zeros((m - 1, n - 1))

    for i in range(m - 1):
        for j in range(n - 1):
            RES[i, j] = 255 if np.sum(IMG[i:i+2, j:j+2]) // 255 == 4 else 0

    return RES.astype(np.uint8)
```

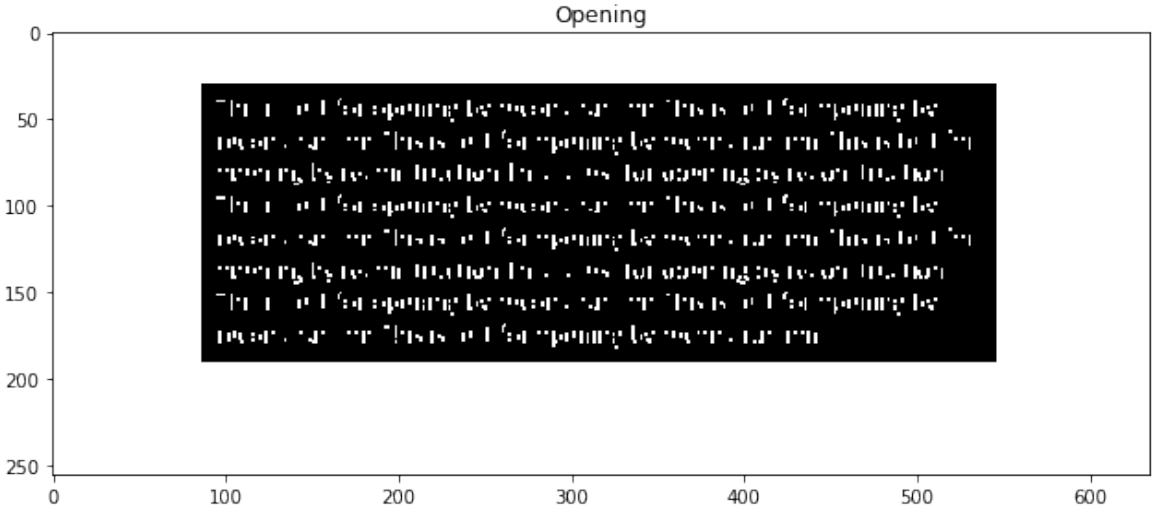
In [8]:
```
IMGO = dilation(erosion(img))
```

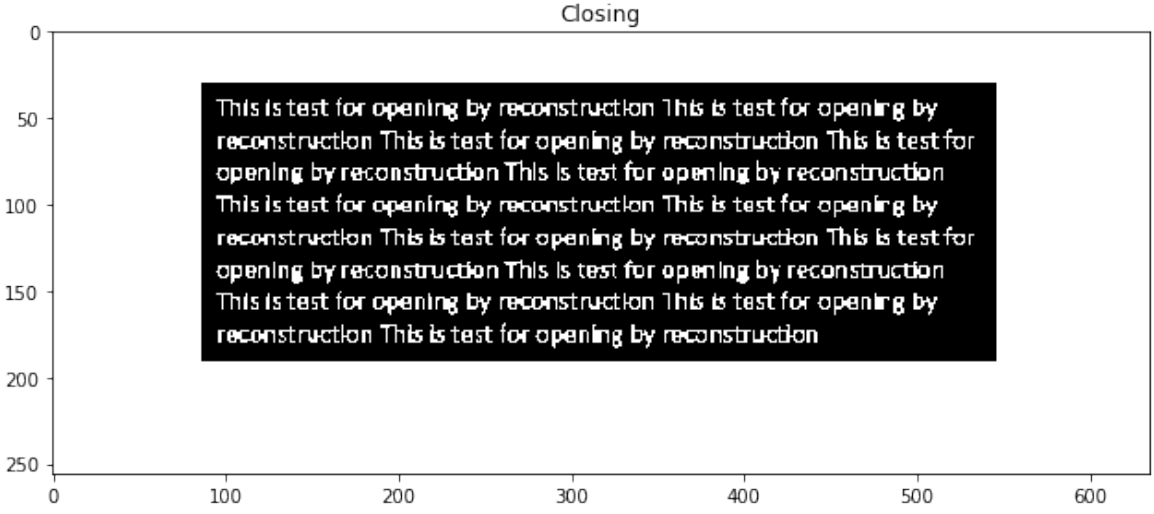
In [9]:
```
IMGC = erosion(dilation(img))
```

In [11]:
```
fig = plt.figure(figsize=(25, 15))

fig.add_subplot(311)
```

```
plt.imshow(img, cmap="gray")
plt.title("Original")

fig.add_subplot(312)
plt.imshow(IMGO, cmap="gray")
plt.title("Opening")

fig.add_subplot(313)
plt.imshow(IMGC, cmap="gray")
plt.title("Closing")

plt.show()
```


Original
This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction
0
50
100
150
200
250
0
100
200
300
400
500
600


Opening
0
50
100
150
200
250
0
100
200
300
400
500
600


Closing
This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction This is test for opening by reconstruction
0
50
100
150
200
250
0
100
200
300
400
500
600