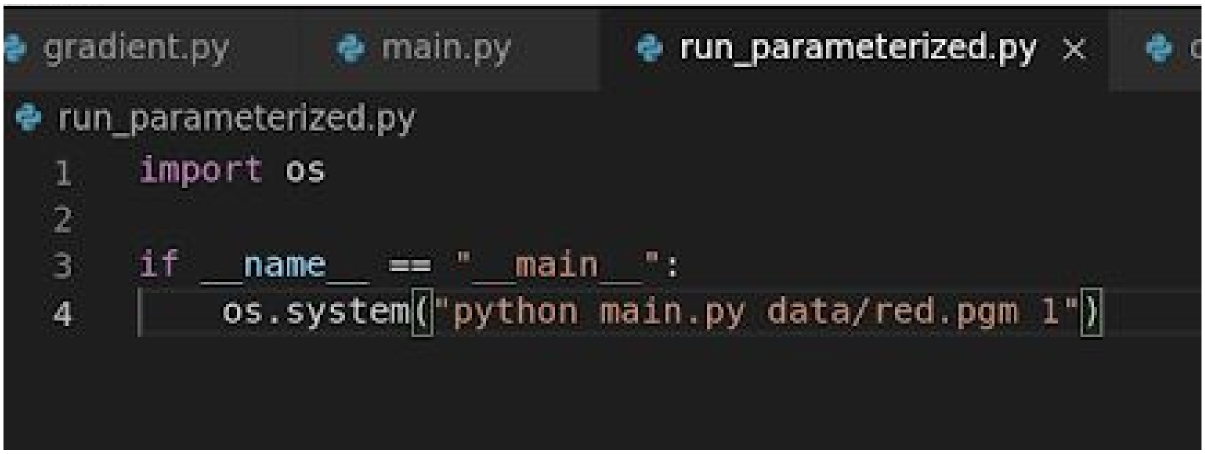
Vincent Lee   I pledge my honor that I have abided by the Stevens Honor System.

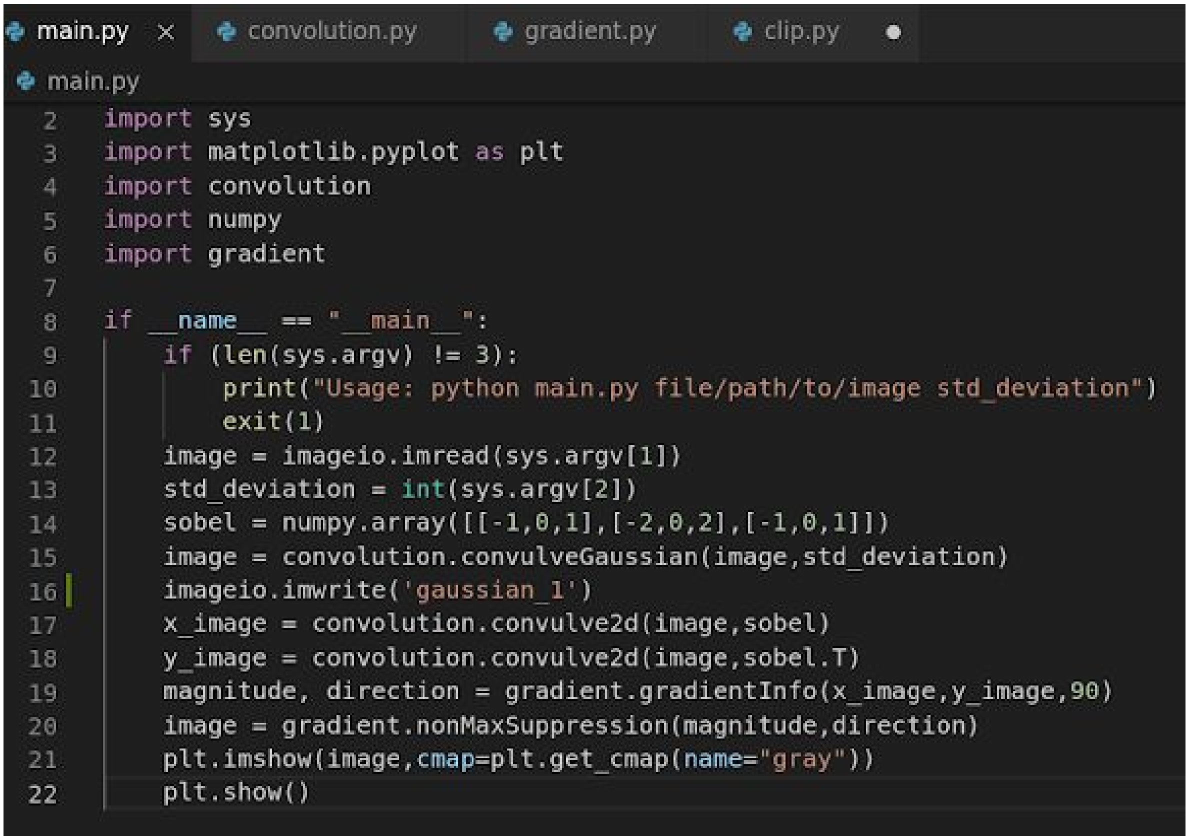
To run parameterized use run_parameterized.py which will just call main.py doing the edge detection on red.pgm with sigma 1. All results are at the end of this document.



```python
import os

if __name__ == "__main__":
    os.system("python main.py data/red.pgm 1")
```

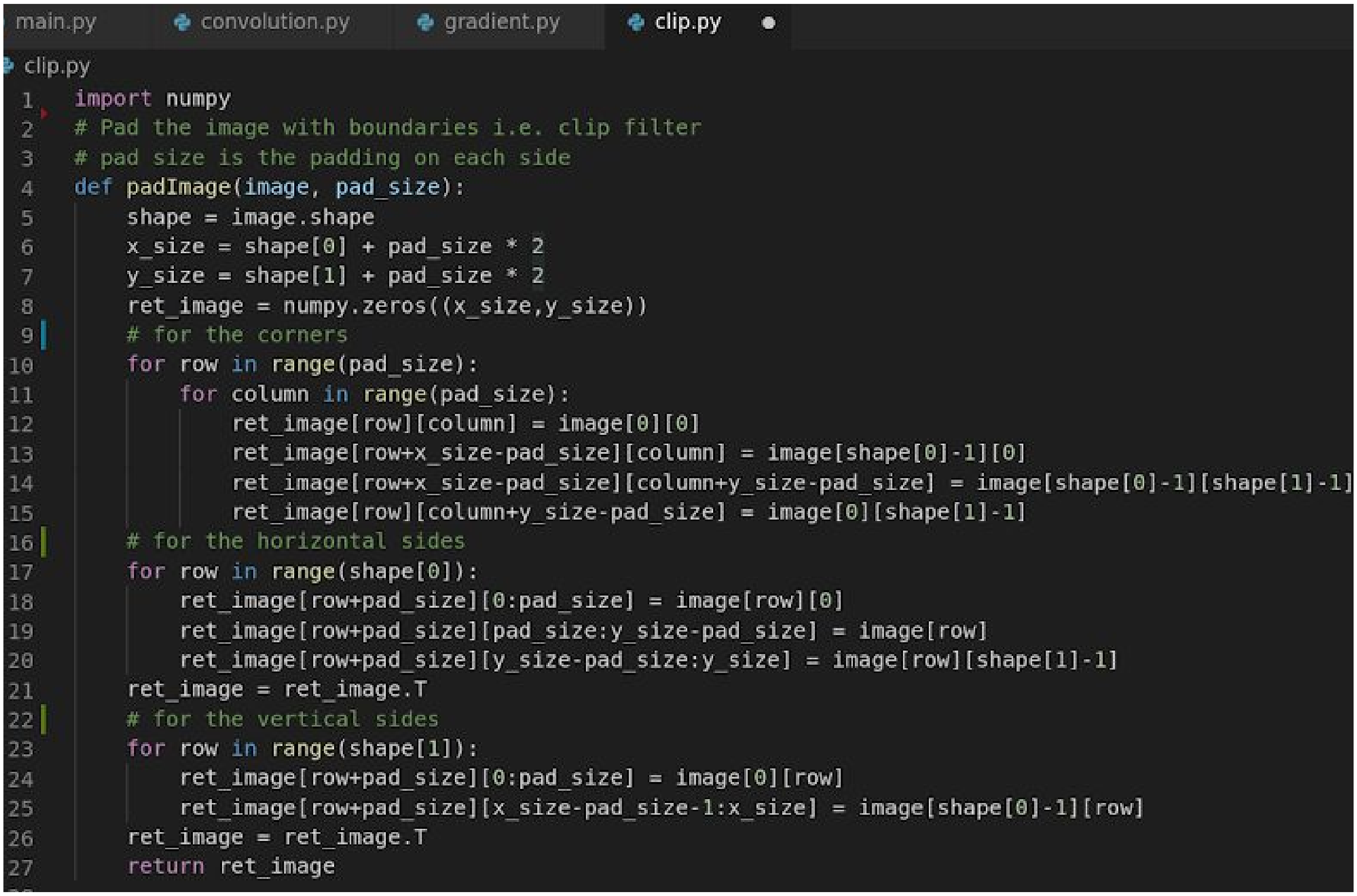
Main.py is the driver it takes the file path and sigma as parameters and calls the helper functions which do the image processing.



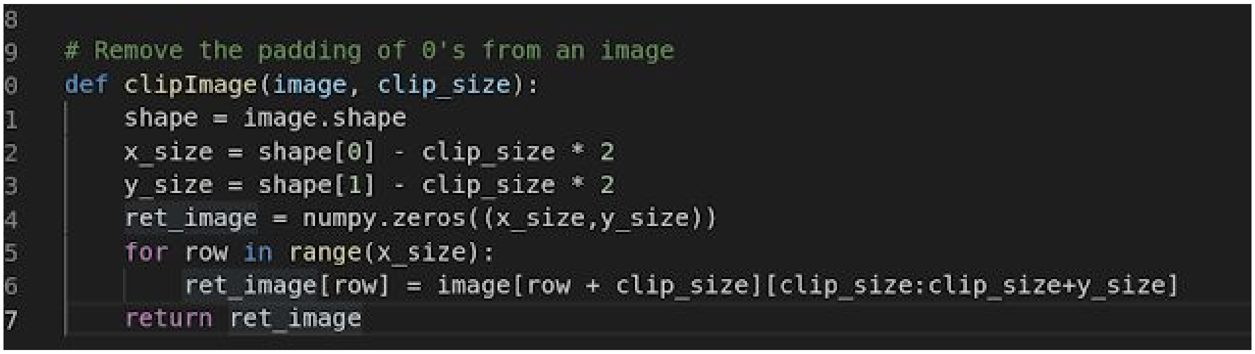
```python
import sys
import matplotlib.pyplot as plt
import convolution
import numpy
import gradient

if __name__ == "__main__":
    if (len(sys.argv) != 3):
        print("Usage: python main.py file/path/to/image std_deviation")
        exit(1)
    image = imageio.imread(sys.argv[1])
    std_deviation = int(sys.argv[2])
    sobel = numpy.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    image = convolution.convulveGaussian(image,std_deviation)
    imageio.imwrite('gaussian_1')
    x_image = convolution.convulve2d(image,sobel)
    y_image = convolution.convulve2d(image,sobel.T)
    magnitude, direction = gradient.gradientInfo(x_image,y_image,90)
    image = gradient.nonMaxSuppression(magnitude,direction)
    plt.imshow(image,cmap=plt.get_cmap(name="gray"))
    plt.show()
```

Lets start with padImage and clipImage which most other functions call, this is in the clip.py file. padImage takes in the amount of padding you want on one side of the image and initializes a new matrix of that size copying over the pixel values of the original and extending it towards the end by numpy array splicing in order to be fast. Transpose is used to add to the vertical components. Clip Image does array splicing to get rid of padding.

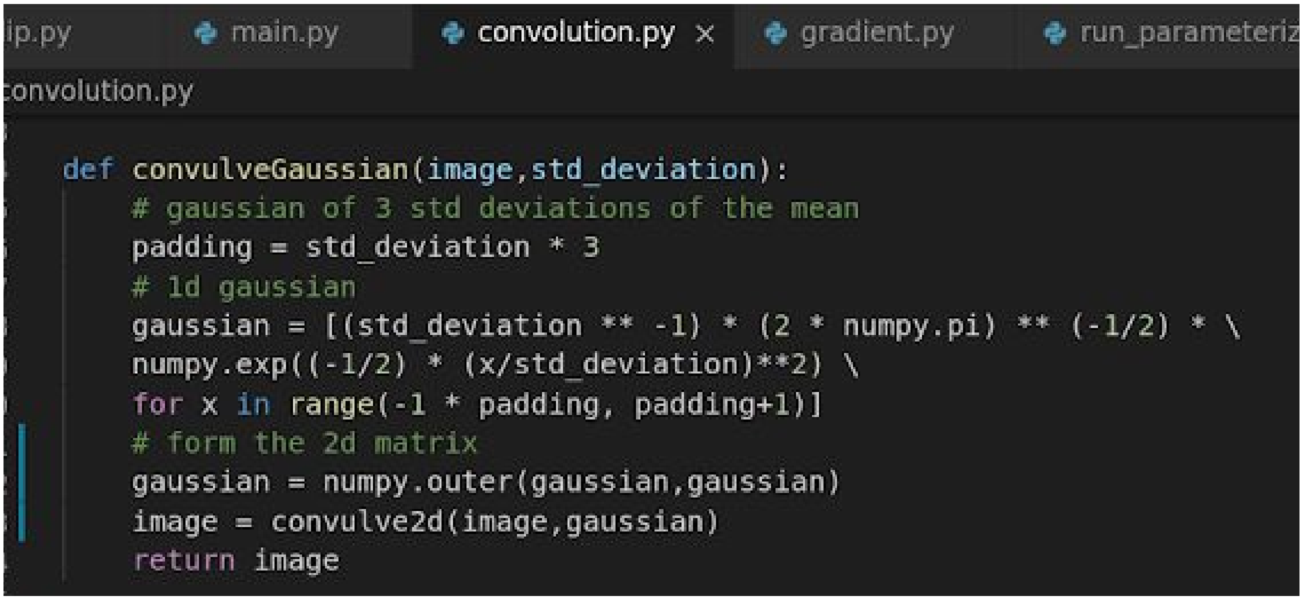


```
import numpy
# Pad the image with boundaries i.e. clip filter
# pad size is the padding on each side
def padImage(image, pad_size):
    shape = image.shape
    x_size = shape[0] + pad_size * 2
    y_size = shape[1] + pad_size * 2
    ret_image = numpy.zeros((x_size,y_size))
    # for the corners
    for row in range(pad_size):
        for column in range(pad_size):
            ret_image[row][column] = image[0][0]
            ret_image[row+x_size-pad_size][column] = image[shape[0]-1][0]
            ret_image[row+x_size-pad_size][column+y_size-pad_size] = image[shape[0]-1][shape[1]-1]
            ret_image[row][column+y_size-pad_size] = image[0][shape[1]-1]
    # for the horizontal sides
    for row in range(shape[0]):
        ret_image[row+pad_size][0:pad_size] = image[row][0]
        ret_image[row+pad_size][pad_size:y_size-pad_size] = image[row]
        ret_image[row+pad_size][y_size-pad_size:y_size] = image[row][shape[1]-1]
    ret_image = ret_image.T
    # for the vertical sides
    for row in range(shape[1]):
        ret_image[row+pad_size][0:pad_size] = image[0][row]
        ret_image[row+pad_size][x_size-pad_size-1:x_size] = image[shape[0]-1][row]
    ret_image = ret_image.T
    return ret_image
```



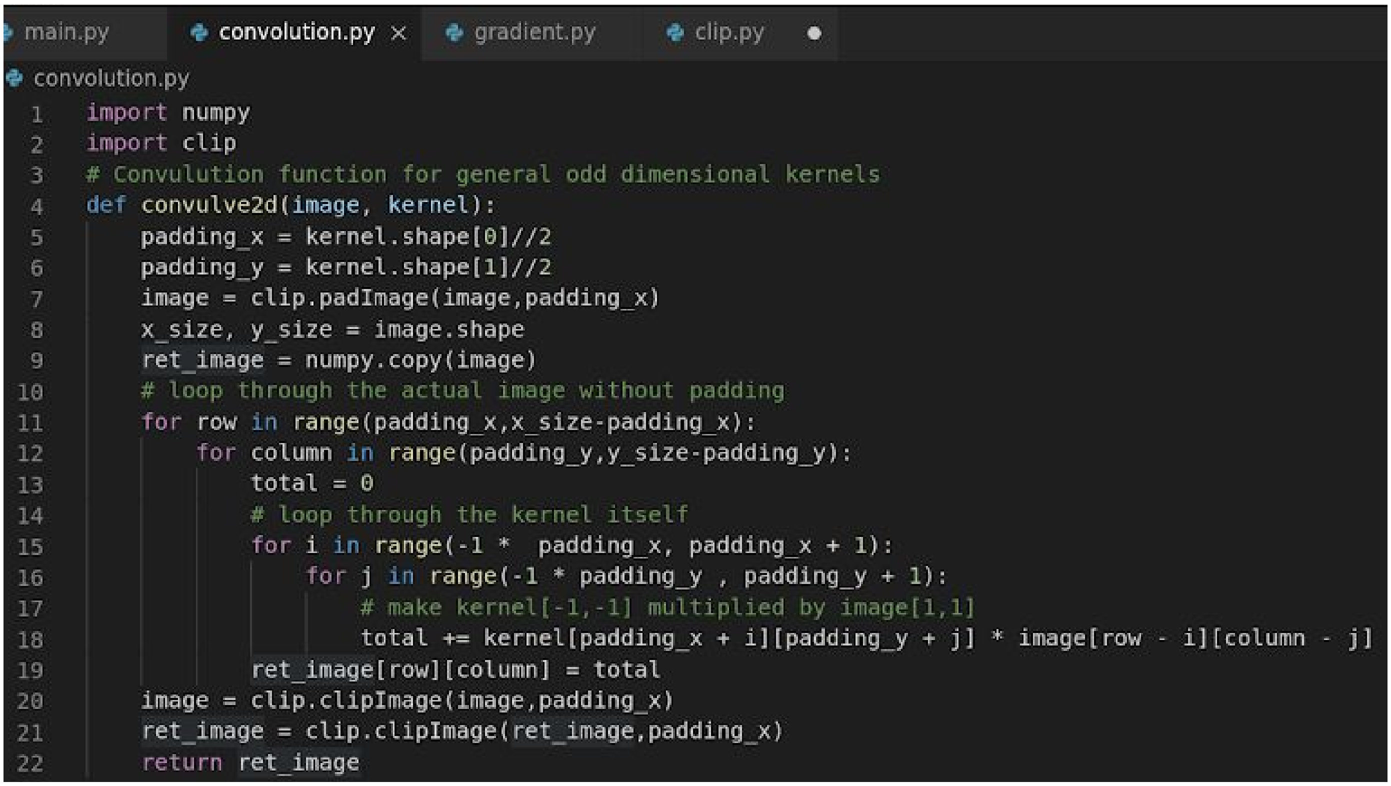
```
# Remove the padding of 0's from an image
def clipImage(image, clip_size):
    shape = image.shape
    x_size = shape[0] - clip_size * 2
    y_size = shape[1] - clip_size * 2
    ret_image = numpy.zeros((x_size,y_size))
    for row in range(x_size):
        ret_image[row] = image[row + clip_size][clip_size:clip_size+y_size]
    return ret_image
```

Convolve Gaussian is done through a 2D convolution. The 1D gaussian is found by list comprehension using the gaussian formula. Then I do an outer multiplication to find a square filter matrix and then apply the 2d convolution with this filter.



```python
def convulveGaussian(image,std_deviation):
    # gaussian of 3 std deviations of the mean
    padding = std_deviation * 3
    # 1d gaussian
    gaussian = [(std_deviation ** -1) * (2 * numpy.pi) ** (-1/2) * \
    numpy.exp((-1/2) * (x/std_deviation)**2) \
    for x in range(-1 * padding, padding+1)]
    # form the 2d matrix
    gaussian = numpy.outer(gaussian,gaussian)
    image = convulve2d(image,gaussian)
    return image
```

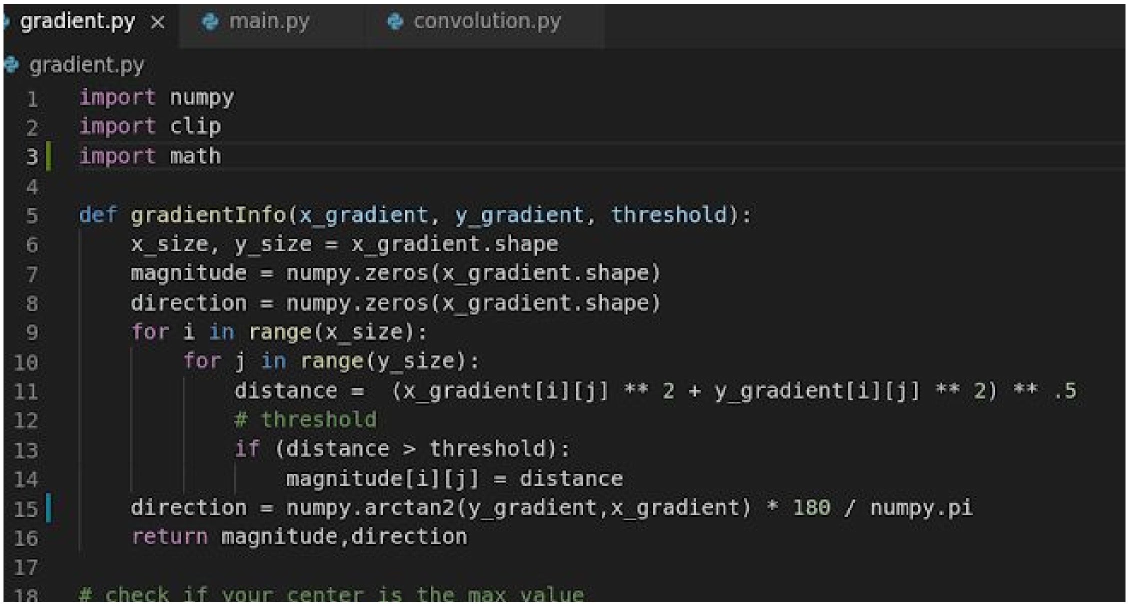
Convulve2d does the convolution, It iterates through all the pixels in the image with regards to the padding and applies the convolution equation. Note this is not linear mapping but convolution. It pads and removes the padding before and after respectively.



```python
import numpy
import clip
# Convulution function for general odd dimensional kernels
def convulve2d(image, kernel):
    padding_x = kernel.shape[0]//2
    padding_y = kernel.shape[1]//2
    image = clip.padImage(image,padding_x)
    x_size, y_size = image.shape
    ret_image = numpy.copy(image)
    # loop through the actual image without padding
    for row in range(padding_x,x_size-padding_x):
        for column in range(padding_y,y_size-padding_y):
            total = 0
            # loop through the kernel itself
            for i in range(-1 *  padding_x, padding_x + 1):
                for j in range(-1 * padding_y , padding_y + 1):
                    # make kernel[-1,-1] multiplied by image[1,1]
                    total += kernel[padding_x + i][padding_y + j] * image[row - i][column - j]
            ret_image[row][column] = total
    image = clip.clipImage(image,padding_x)
    ret_image = clip.clipImage(ret_image,padding_x)
    return ret_image
```

Then the sobel filter is applied in main.py to get two gradient images.

The magnitude and direction of the image is then done in gradientInfo by passing in the two gradient images. The magnitude is the distance of the two gradients and the direction is done by numpy.arctan2 in order to preserve quadrants. The threshold was chosen as 90 as that is what removed a good amount of the texture in the kangaroo image while maintaining the kangaroo itself. The other images did not really need the threshold as their was not a lot of texture differences.

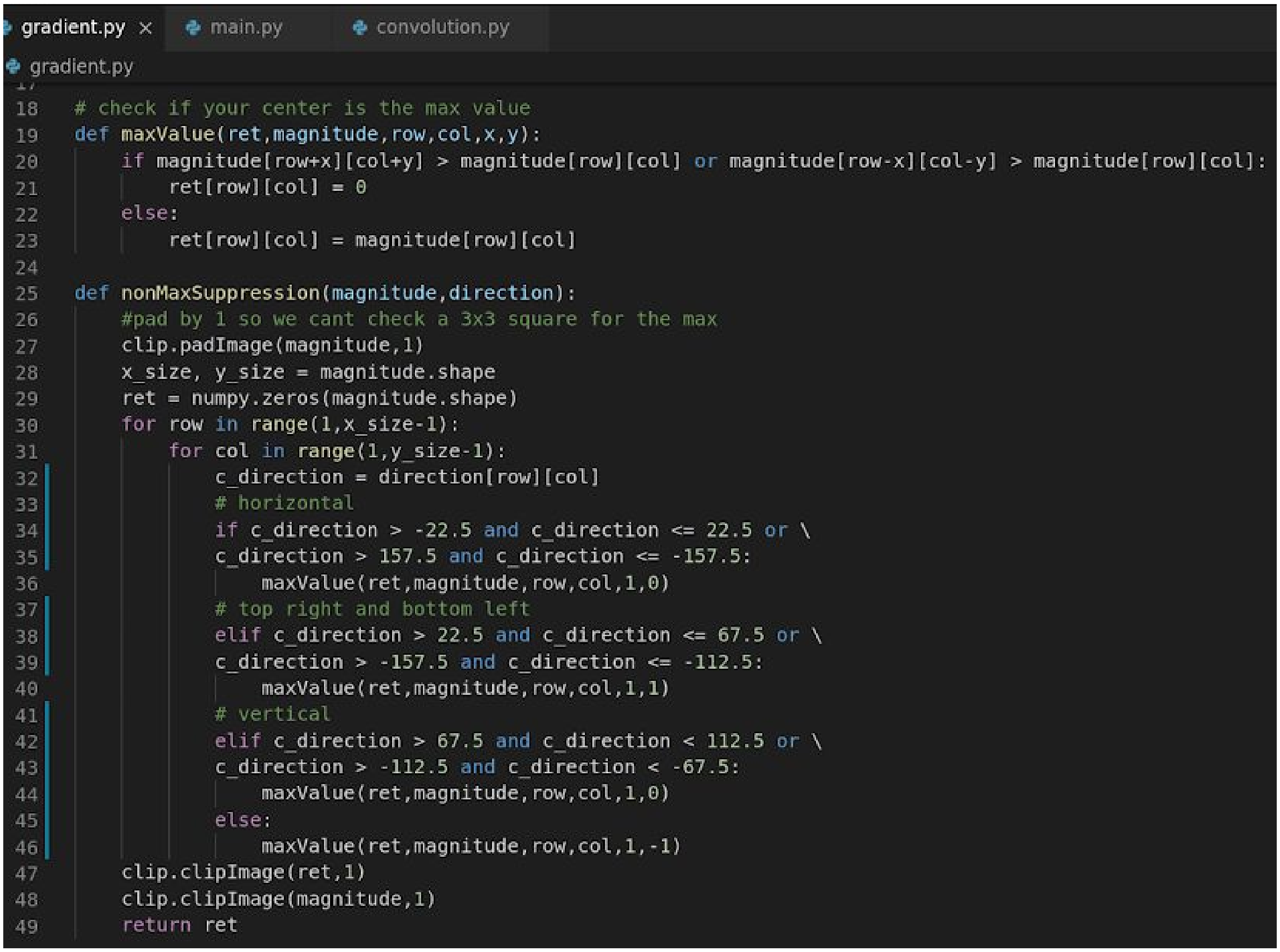


```
import numpy
import clip
import math

def gradientInfo(x_gradient, y_gradient, threshold):
    x_size, y_size = x_gradient.shape
    magnitude = numpy.zeros(x_gradient.shape)
    direction = numpy.zeros(x_gradient.shape)
    for i in range(x_size):
        for j in range(y_size):
            distance =  (x_gradient[i][j] ** 2 + y_gradient[i][j] ** 2) ** .5
            # threshold
            if (distance > threshold):
                magnitude[i][j] = distance
    direction = numpy.arctan2(y_gradient,x_gradient) * 180 / numpy.pi
    return magnitude,direction

# check if your center is the max value
```

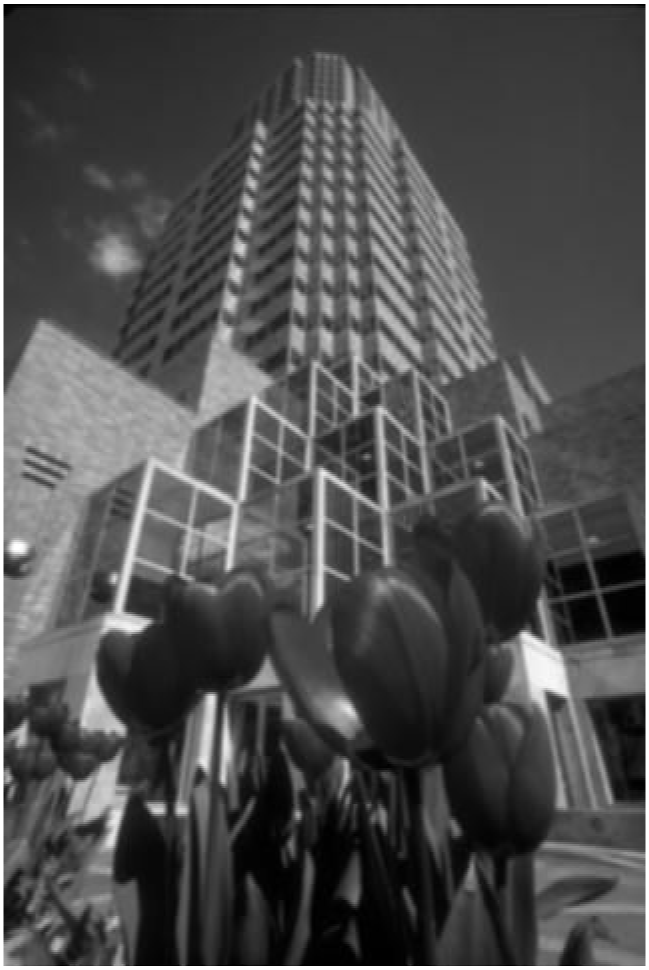
Non max suppression is done by looping through the the direction matrix. If the gradient is vertical as in between 67.5 and 112.5 or -67.5 and -112.5 degrees I look along the vertical axis to check if it is a maximum. The gradient is always perpendicular to the edge so we always look along its direction. We do this for the other directions, horizontal, and the two verticals, looking along the axis of the direction to find the max. The splitting of the circle is done in degrees allowing for 8 separate components.

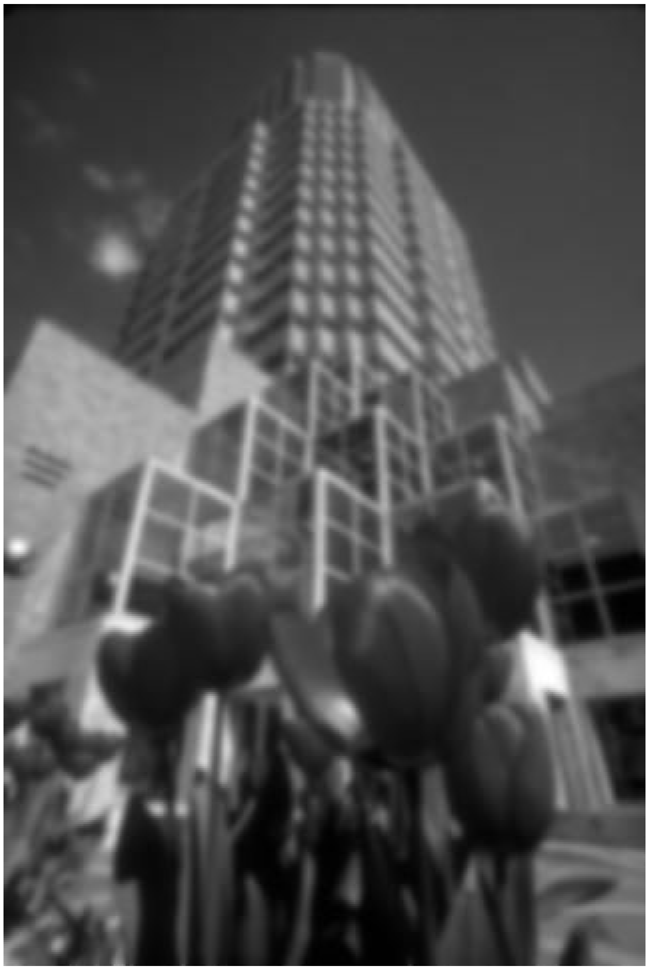


```python
# check if your center is the max value
def maxValue(ret,magnitude,row,col,x,y):
    if magnitude[row+x][col+y] > magnitude[row][col] or magnitude[row-x][col-y] > magnitude[row][col]:
        ret[row][col] = 0
    else:
        ret[row][col] = magnitude[row][col]

def nonMaxSuppression(magnitude,direction):
    #pad by 1 so we cant check a 3x3 square for the max
    clip.padImage(magnitude,1)
    x_size, y_size = magnitude.shape
    ret = numpy.zeros(magnitude.shape)
    for row in range(1,x_size-1):
        for col in range(1,y_size-1):
            c_direction = direction[row][col]
            # horizontal
            if c_direction > -22.5 and c_direction <= 22.5 or \
            c_direction > 157.5 and c_direction <= -157.5:
                maxValue(ret,magnitude,row,col,1,0)
            # top right and bottom left
            elif c_direction > 22.5 and c_direction <= 67.5 or \
            c_direction > -157.5 and c_direction <= -112.5:
                maxValue(ret,magnitude,row,col,1,1)
            # vertical
            elif c_direction > 67.5 and c_direction < 112.5 or \
            c_direction > -112.5 and c_direction < -67.5:
                maxValue(ret,magnitude,row,col,1,0)
            else:
                maxValue(ret,magnitude,row,col,1,-1)
    clip.clipImage(ret,1)
    clip.clipImage(magnitude,1)
    return ret
```

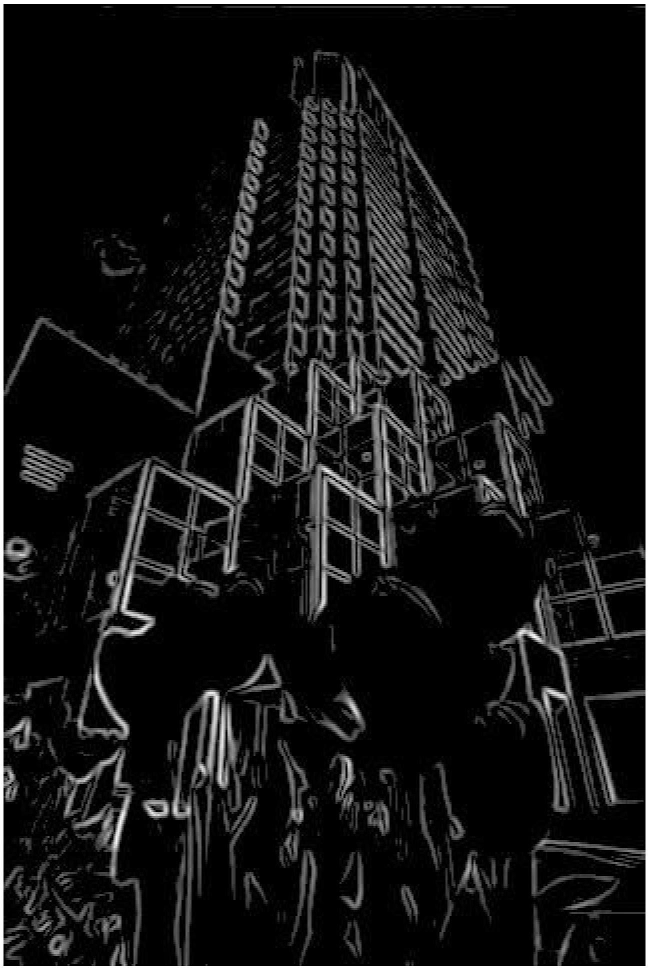
RESULTS



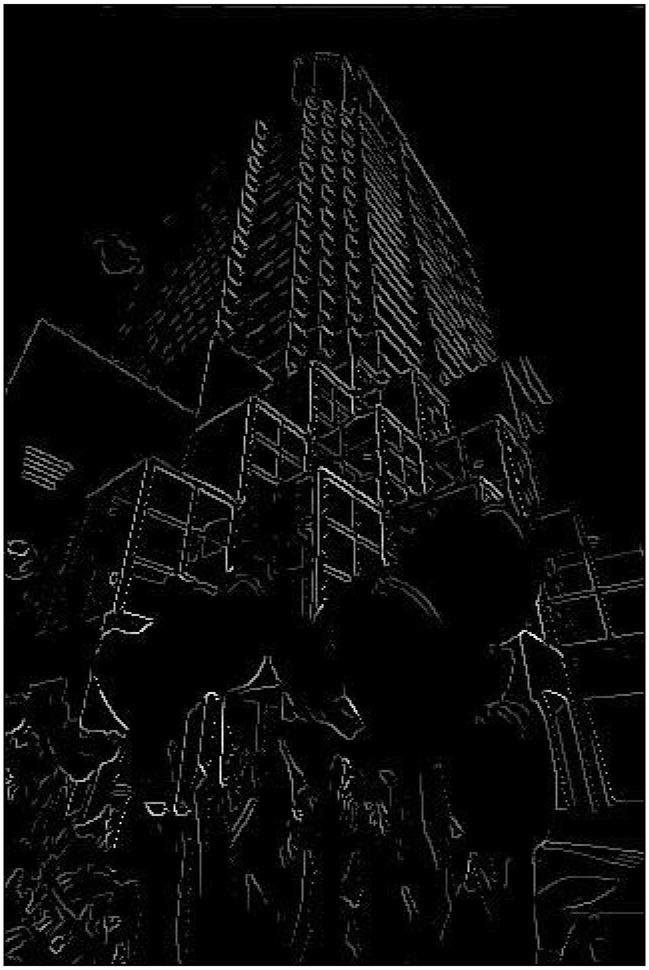


Red with sigma 1

Red with sigma 2



Red magnitude w/sigma 1



Red Non-max-suppression w/sigma 1