Image Processing HW2

Report section:

In this assignment, I implement canny edge detection from scratch on those four images I found online. For the two color images, I will convert them into gray-scale images by the following formula,

```
red * 0.3 + green * 0.59 + blue * 0.11
```

This formula has been tested highly effective comparing to simply add Red, Green and Blue, and divide them by three. For a very long time, I thought the correct way to covert image to gray-scale is sum and divide by three.

The reason why I choose the Trump image and Musk image is not only because they are controversial, funny person but also because the images possess clear edges. For example, the black suit of Trump has great contrast with the white cloud in the background. Furthermore, the blue Twitter logo and the white bird or hole also create boundaries.

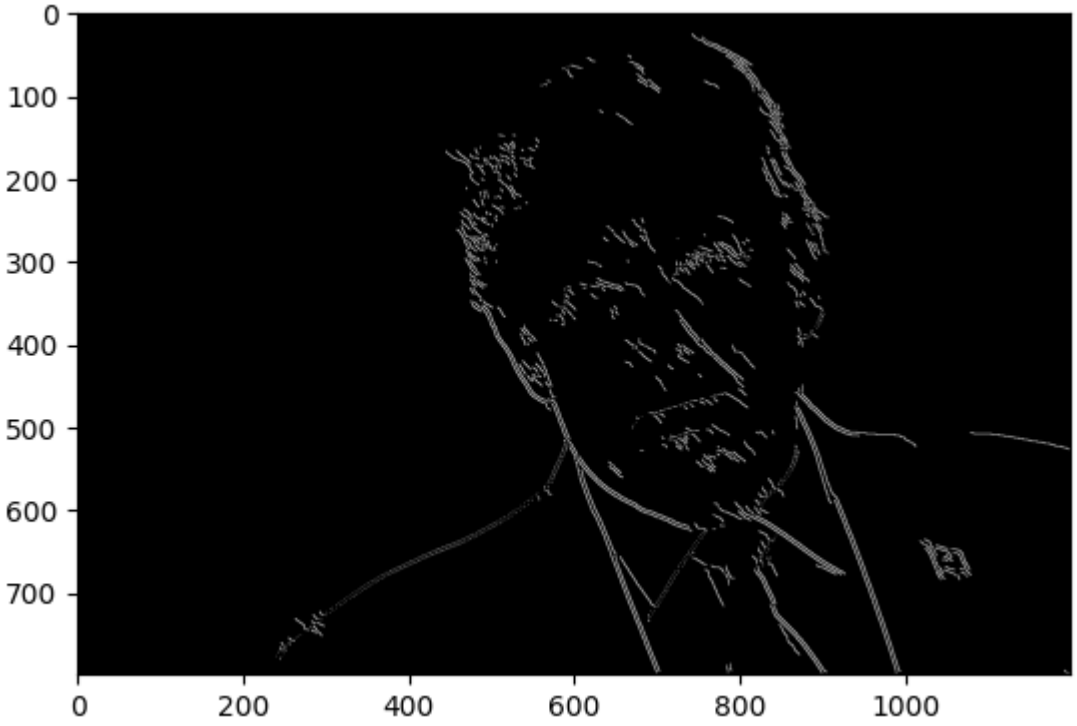
First of all. Let me explain how to run my model and set the high threshold, low threshold for my model. As the figure below indicates, first initializes a class name Canny with the required parameters. Second, directly run the built-in-function, detect_edge, then we can receive the results of canny edge detection.

At first, I try to use the same parameter as cv2. However, the high and low threshold have to be manually modified for EVERY SINGLE image. Therefore, I change the “high threshold” to “threshold1” and “low threshold” to “threshold2”, indicating the percentile of the gradient magnitude we want to reject. For instance, if the largest gradient magnitude is 100 and the threshold1 is 0.15, we set the high threshold as 15. In addition, if the threshold2 is 0.5, means we set the low threshold at 7.5 (15 * 0.5). To further demonstrate the results, we take the trump image as the first example.

Image Processing HW2

```
canny = Canny(gray, threshold1= 0.15, threshold2= 0.05, gaussian_sigma=1, gaussian_k_size=5 )
output = canny.detect_edge()
```

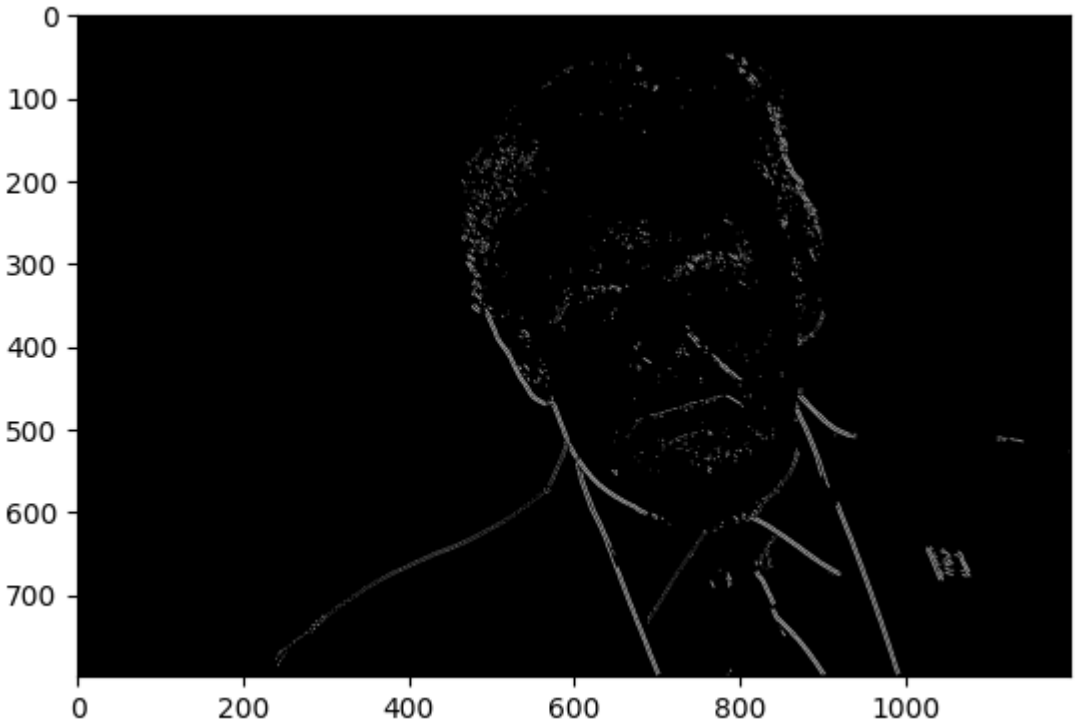
1. threshold1= 0.15, **threshold2= 0.05**, gaussian_sigma=1, gaussian_k_size=5
   Trump's **face and suit** are pretty **clear**. We can also see the wrinkle of his face.



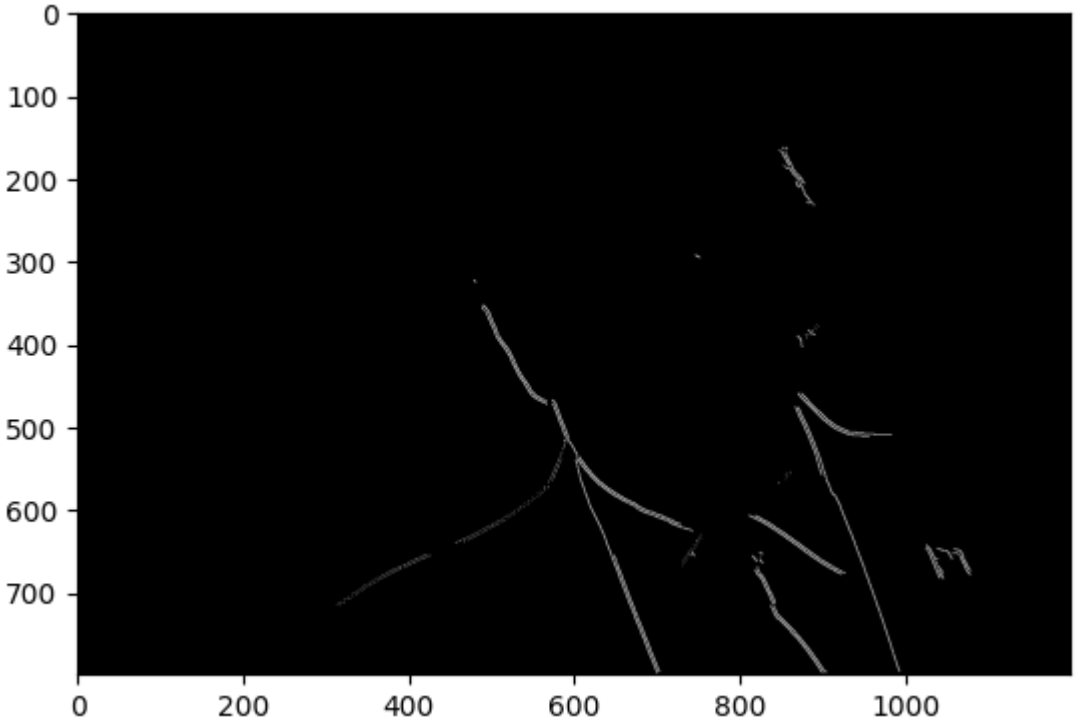
2. threshold1= 0.15, **threshold2= 0.95**, gaussian_sigma=1, gaussian_k_size=5

After increasing the low threshold, the **small edges** such as wrinkle on his face is **disappeared**.

Image Processing HW2

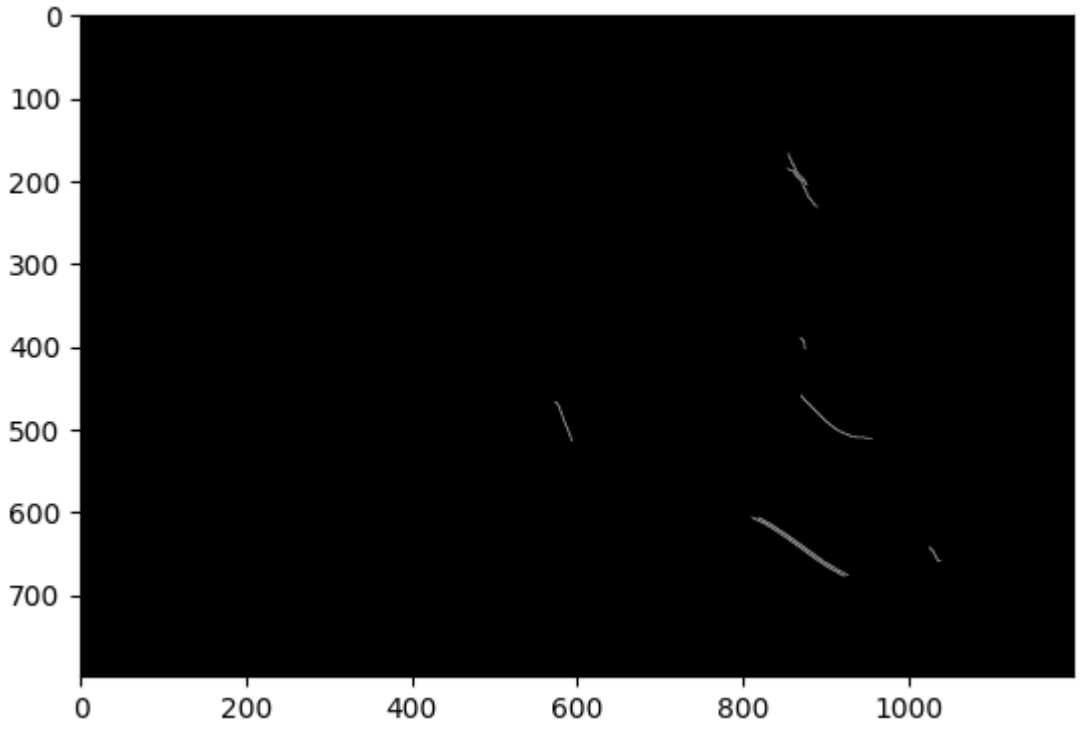


3. **threshold1= 0.5**, threshold2= 0.05, gaussian_sigma=1, gaussian_k_size=5
   Increasing the high threshold ratio to 0.5, only the **most significant edges** are **preserved**.



4. **threshold1= 0.85**, threshold2= 0.05, gaussian_sigma=1, gaussian_k_size=5
   Set the high threshold ratio to 0.85, we can **barely see the shape** of the face and suit.

Image Processing HW2



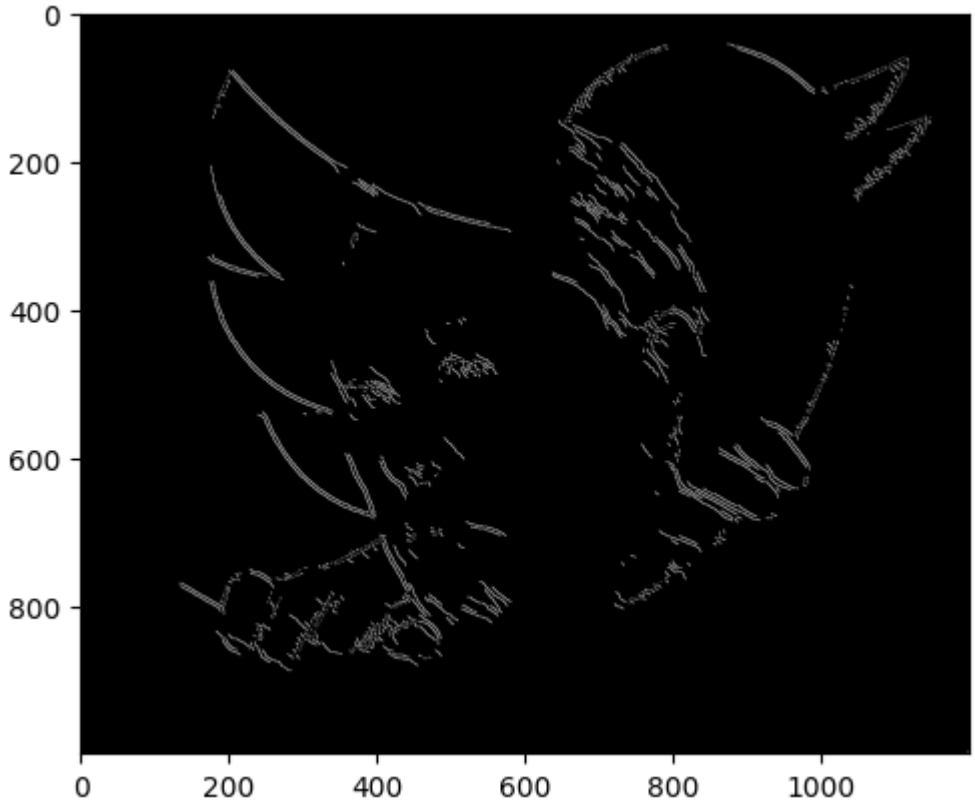
The experiment tells us that we **SHOULD NOT** set the **high threshold** or the high threshold ratio too high because if we only accept top 5% of the magnitude, there will left almost nothing.

Next, the experiment goes to Elon Musk.

1. threshold1= 0.25, **threshold2= 0.05**, gaussian_sigma=1, gaussian_k_size=5

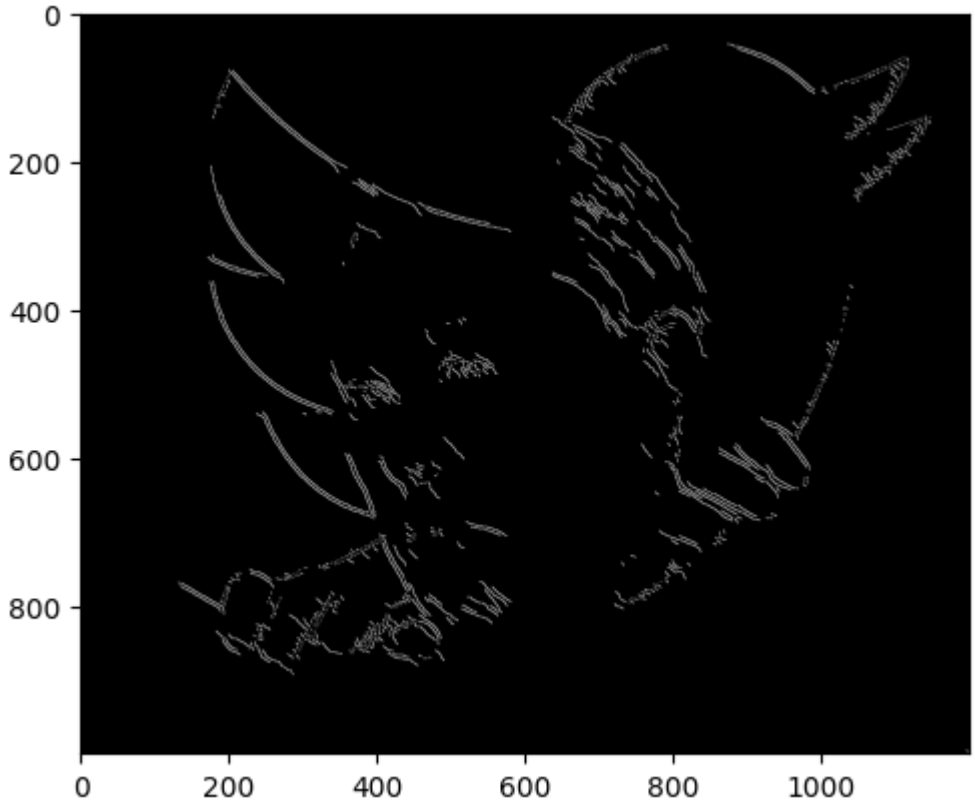
Some part of the **edge** of the Twitter bird are **not continuous**. Thus, let's try lower the low threshold2 ratio.

Image Processing HW2



2. threshold1= 0.25, **threshold2= 0.001**, gaussian_sigma=1, gaussian_k_size=5
   The modification of threshold2 is **NOT** particularly **helpful**. The output image looks the same as before, so probably lower the threshold1 ratio would be a better idea.

Image Processing HW2



3. **threshold1= 0.05**, threshold2= 0.05, gaussian_sigma=1, gaussian_k_size=5
   It turns out the bar is too low, **too many unnecessary details** appear.
   However, the outline of the **Twitter bird** finally **close to continuous**.

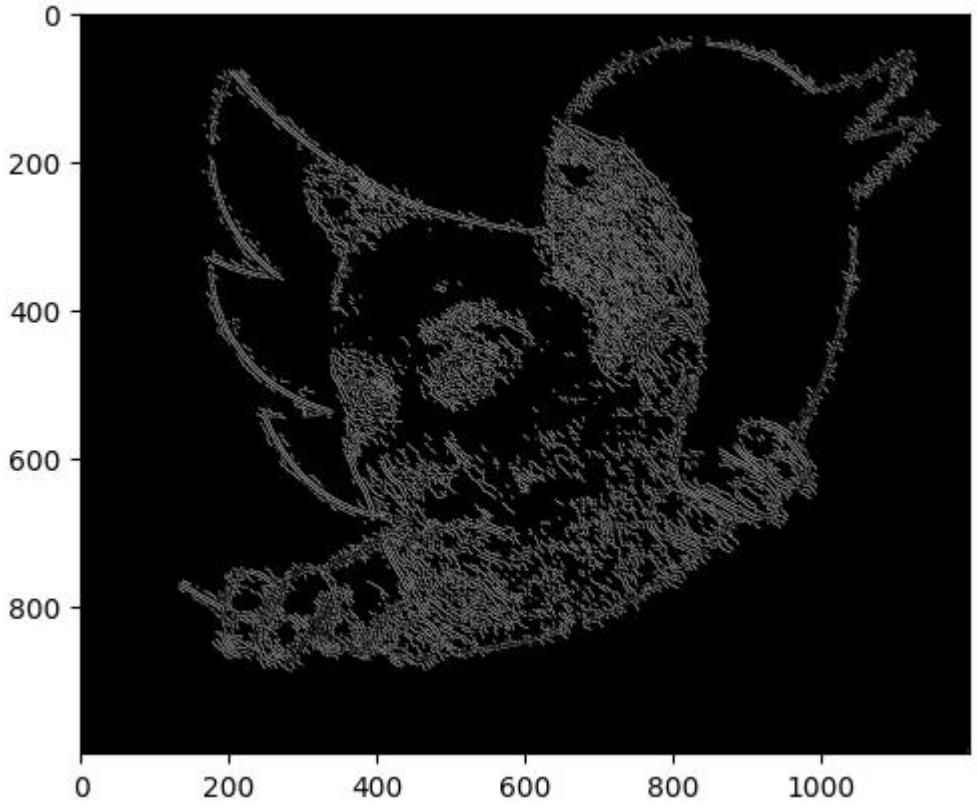
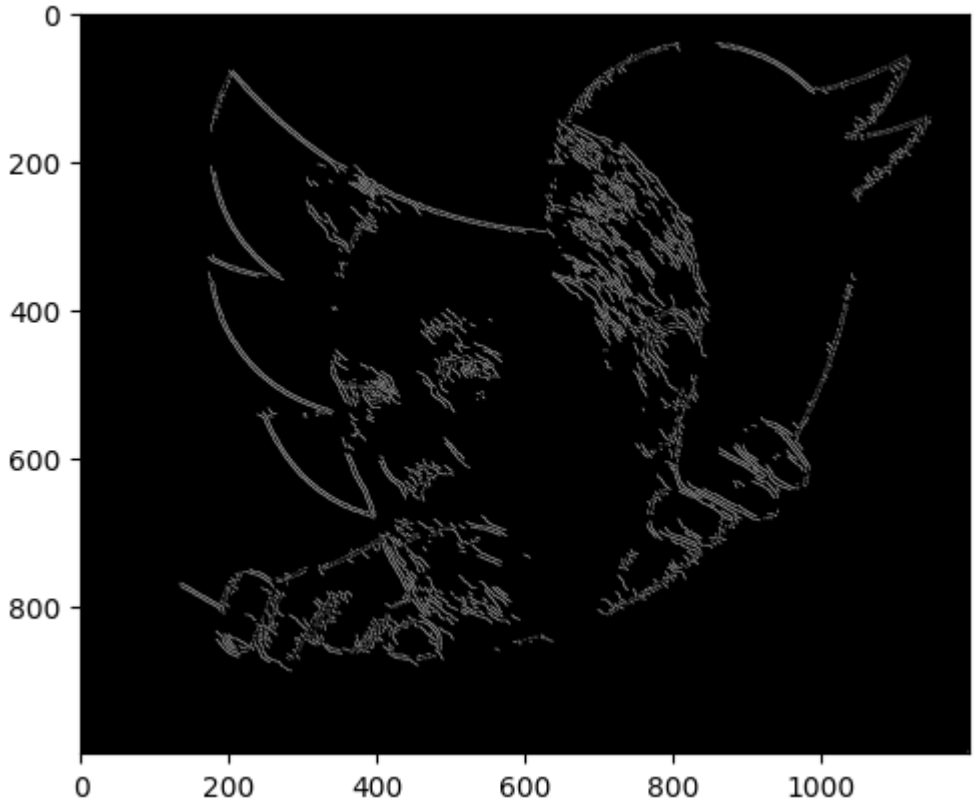


Image Processing HW2

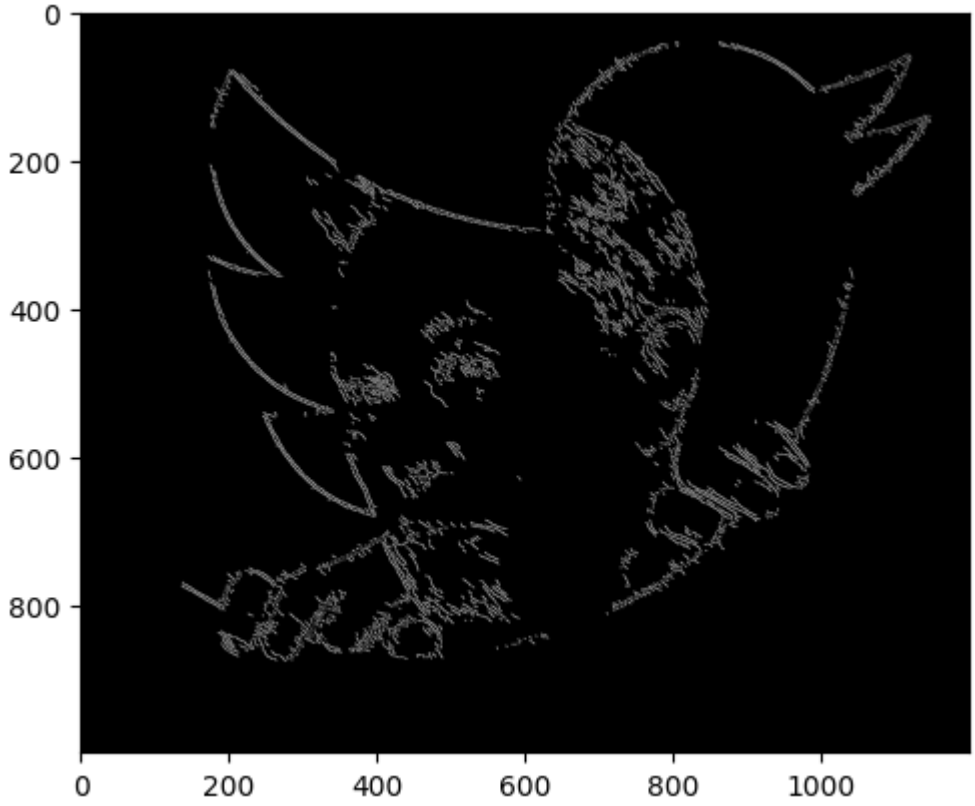
4. **threshold1= 0.15**, threshold2= 0.05, gaussian_sigma=1, gaussian_k_size=5
With those parameters, the **balance** between details and cleanness is the **best**.



Further testing for gaussian related parameters.

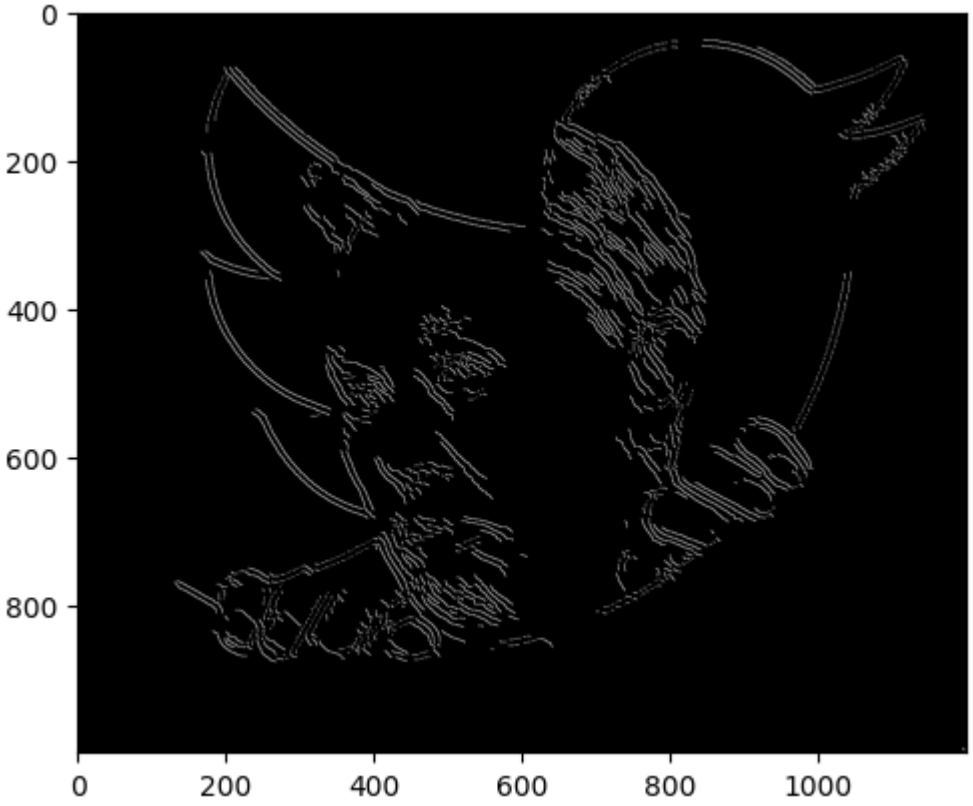
5. threshold1= 0.15, threshold2= 0.05, **WITHOUT gaussian blur**
Surprisingly, the result **WITHOUT** gaussian filter make **not** much of a **difference**. Therefore, the importance of the gaussian filter for canny edge detection might be re-considered when the gaussian sigma is low and the kernel size is small.

Image Processing HW2



6. threshold1= 0.15, threshold2= 0.05, **gaussian_sigma=5, gaussian_k_size=9**
Significantly **increase the sigma and kernel size** of gaussian filter makes the image more blur before entering Sobel filters. The blurred image has less detail, **less small irrelevant edges**, in the middle of Musk's face, but it tends to make the edges become **double edges**.

Image Processing HW2



Comparing with the canny edge detection performed by OpenCV library.

High threshold 200. Low threshold 100. Kernel size (5,5), Sigma=1
The result is incredibly good!!! After searching online, I think OpenCV might do some enhancing technique, since the results of canny edge detection performed by others are mostly the same as mine, a little messy, but can see the overall outline.

Image Processing HW2

Image Processing HW2

Code section:

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg


def convolve(img, kernel):
    h, w = img.shape
    k_size = kernel.shape[0]
    pad_one_side = (k_size - 1)//2
    h_pad = h + 2*pad_one_side
    w_pad = w + 2*pad_one_side

    img_pad = np.zeros((h_pad, w_pad))
    img_pad[pad_one_side : -pad_one_side, pad_one_side : -pad_one_side] = img

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            mat = img_pad[i : i + k_size, j : j + k_size]
            # print("mat = ", mat)
            img[i, j] = np.sum(np.multiply(mat, kernel))
    return img
```

```python
class Canny:
    def __init__(self, img, threshold1, threshold2, gaussian_sigma=1, gaussian_k_size=5, strong=255, weak=32):
        self.img = img
        self.threshold1 =threshold1
        self.threshold2 = threshold2
        self.gaussian_sigma = gaussian_sigma
        self.gaussian_k_size = gaussian_k_size
        self.strong = strong
        self.weak = weak

    def gaussian_kernel(self, k_size=5, sigma=1):
        tmp = (k_size - 1) / 2
        x, y = np.mgrid[-tmp:tmp+1, -tmp:tmp+1]
        kernel =  1 / (2.0 * np.pi * sigma*sigma) * np.exp(-((x*x + y*y) / (2.0*sigma*sigma)))
        return kernel
```

```python
    def gaussian_blur(self, img, k_size=5, sigma=1):
        kernel = self.gaussian_kernel(k_size, sigma)
        img_blur = convolve(img, kernel)
        return img_blur

    def sobel_edge(self, img):
        sx = np.array([[-1,0,1], [-2,0,2], [-1,0,1]])
        sy = np.array([[1,2,1], [0,0,0], [-1,-2,-1]])
        gradient_x = convolve(img, sx)
        gradient_y = convolve(img, sy)
        magnitude = np.hypot(gradient_x, gradient_y)
        magnitude = magnitude / magnitude.max() *255
        # magnitude = magnitude.astype(np.uint8)
        theta = np.arctan2(gradient_y, gradient_x)
        return magnitude, theta
```

Image Processing HW2

```python
def non_max_suppression(self, magnitude, theta):
    h, w = magnitude.shape
    output_magnitude = np.zeros(magnitude.shape)
    angle = np.rad2deg(theta)
    angle[angle < 0] += 180
    PI = 180
    for row in range(1, h-1):
        for col in range(1, w-1):
            direction = angle[row, col]
            if (0 <= direction < PI/8) or (7*PI/8 <= direction <= PI):
                neighbor1 = magnitude[row][col+1]
                neighbor2 = magnitude[row][col-1]
            elif (PI/8 <= direction < 3*PI/8):
                neighbor1 = magnitude[row+1][col-1]
                neighbor2 = magnitude[row-1][col+1]
            elif (3*PI/8 <= direction < 5*PI/8):
                neighbor1 = magnitude[row+1][col]
                neighbor2 = magnitude[row-1][col]
            elif (5*PI/8 <= direction < 7*PI/8):
                neighbor1 = magnitude[row-1][col-1]
                neighbor2 = magnitude[row+1][col+1]
```

```python
            if magnitude[row, col] > neighbor1 and magnitude[row, col] > neighbor2:
                output_magnitude[row, col] = magnitude[row, col]

    return output_magnitude

def hysteresis_thresholding(self, magnitude):
    h, w = magnitude.shape
    output = np.zeros(magnitude.shape)

    high_threshold = magnitude.max() * self.threshold1
    low_threshold = high_threshold * self.threshold2

    strong_row, strong_col = np.where(high_threshold < magnitude)
    weak_row, weak_col = np.where((low_threshold < magnitude) & (magnitude <= high_threshold))
    output[strong_row, strong_col] = self.strong
    output[weak_row, weak_col] = self.weak
```

```python
top2bottom = output.copy()
for row in range(1, h-1):
    for col in range(1, w-1):
        if top2bottom[row][col] == self.weak:
            if top2bottom[row+1][col+1] ==self.strong or top2bottom[row+1][col] ==self.strong or top2bottom[row+
                top2bottom[row, col] = self.strong
            else:
                top2bottom[row, col] = 0

bottom2top = output.copy()
for row in range(h-2, 0, -1):
    for col in range(w-2, 0, -1):
        if bottom2top[row][col] == self.weak:
            if bottom2top[row+1][col+1] ==self.strong or bottom2top[row+1][col] ==self.strong or bottom2top[row+
                bottom2top[row, col] = self.strong
            else:
                bottom2top[row, col] = 0
```

Image Processing HW2

```
    right2left = output.copy()
    for row in range(1, h-1):
        for col in range(w-2, 0, -1):
            if right2left[row][col] == self.weak:
                if right2left[row+1][col+1] ==self.strong or right2left[row+1][col] ==self.strong or right2left[row+
                    right2left[row, col] = self.strong
                else:
                    right2left[row, col] = 0

    left2right = output.copy()
    for row in range(h-2, 0, -1):
        for col in range(1, w-1):
            if left2right[row][col] == self.weak:
                if left2right[row+1][col+1] ==self.strong or left2right[row+1][col] ==self.strong or left2right[row+
                    left2right[row, col] = self.strong
                else:
                    left2right[row, col] = 0
    sum_results = top2bottom + bottom2top + right2left + left2right
    sum_results[sum_results > 255] = 255
    return sum_results
```

```
def detect_edge(self):
    # output_lst = []

    # img_blur = self.gaussian_blur(self.img, self.gaussian_k_size, self.gaussian_sigma)
    # plt.imshow(img_blur, cmap="gray")
    magnitude, theta = self.sobel_edge(self.img)
    # plt.imshow(magnitude, cmap="gray")
    magnitude_nms = self.non_max_suppression(magnitude, theta)
    # plt.imshow(magnitude_nms, cmap="gray")

    output = self.hysteresis_thresholding(magnitude_nms)
    plt.imshow(output, cmap="gray")
    return output
```

```
img = mpimg.imread("data/musk2.jpg")
r, g, b = img[:,:,0], img[:,:,1], img[:,:,2]
gray = 0.3 * r + 0.59 * g + 0.11 * b
# gray = (r + g +b)/3
print(gray.shape)

canny = Canny(gray, threshold1= 0.15, threshold2= 0.05, gaussian_sigma=5, gaussian_k_size=9 )
output = canny.detect_edge()
# print(type(output[0][0]))
# magnitude_nms = canny.non_max_suppression(output, theta)
# plt.imshow(output, cmap="gray")
```

✓ 9.7s Python Python Python Python Python

```
(1000, 1200)
```

Image Processing HW2


```python
import cv2
img = cv2.imread("data/musk2.jpg")
print(img.shape)
img = cv2.resize(img, (600, 500))
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, ksize= (5, 5), sigmaX=1)
output = cv2.Canny(blur, 200,100)
cv2.imwrite("output.jpg", output)
cv2.imshow("img", output)
cv2.waitKey(0)
```