Canny Edge Detector

Harman Singh Chawla
Net ID: hsc367 University ID: N14381201

Source Code and Executable File

- The source code file is named “HarmanCVProject1.py” and is located in the same folder as this PDF. (Python 3.7 with OpenCV and Numpy dependencies)

- Since I am working in Python, generating a *.exe would be an unnecessary task. So, I have written a small shell command that will help you run my code. Simply drag and drop the “run.sh” file in a Terminal window. If it throws an error, try running the command “chmod +x run.sh”

- In the following slides, you will find the output images for the two test images provided (also available in the “Output” folder for further examination) and the source code.

Output: Lady Image

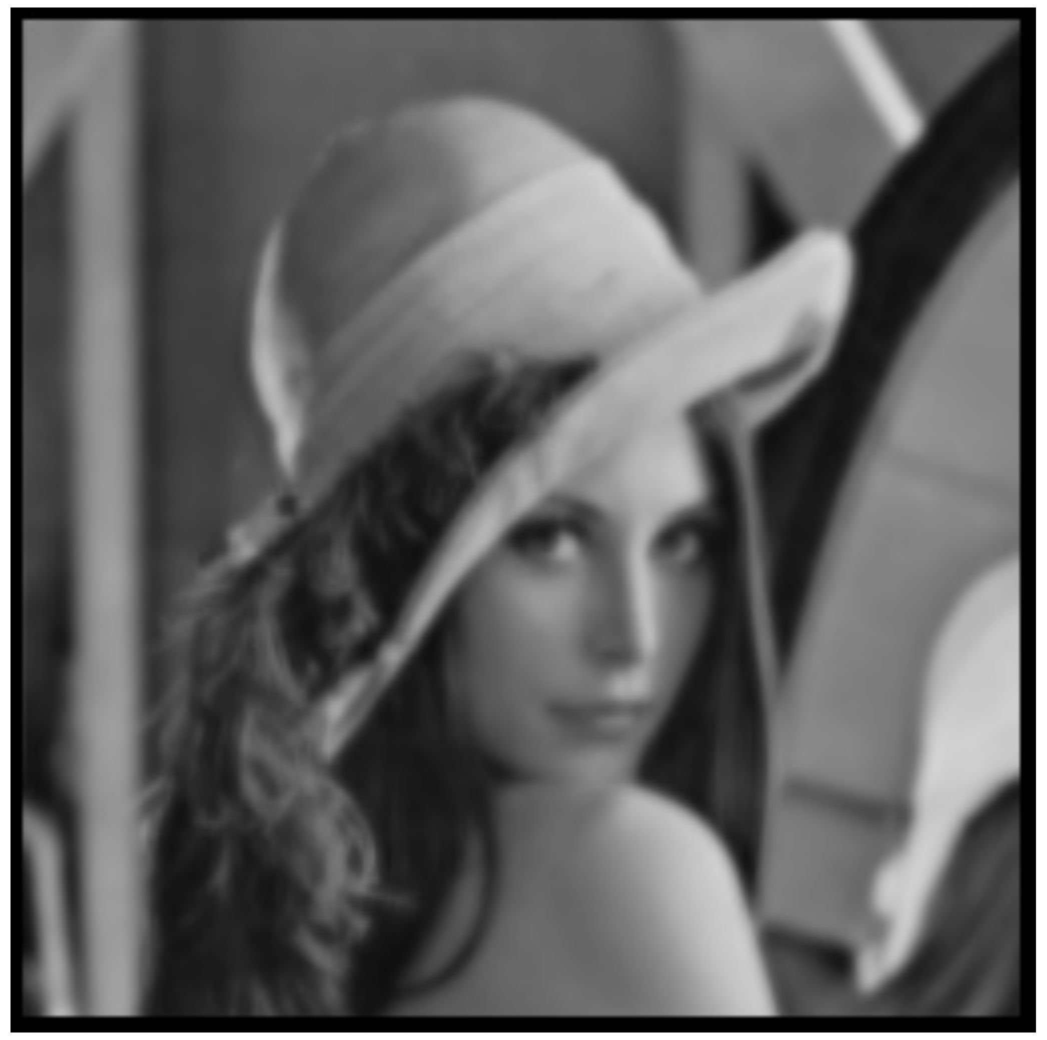


Fig 1a. Gaussian Filter

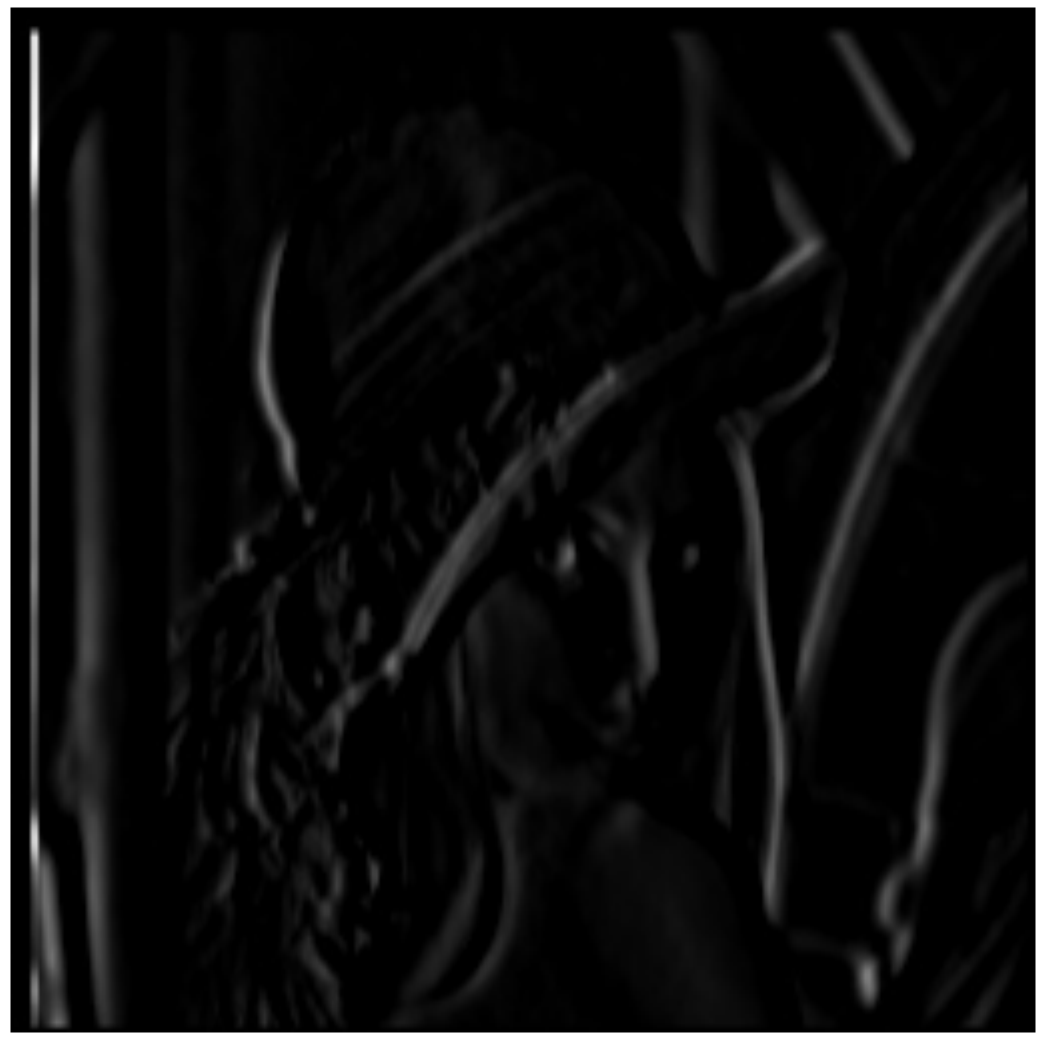


Fig 1b. Horizontal Gradient

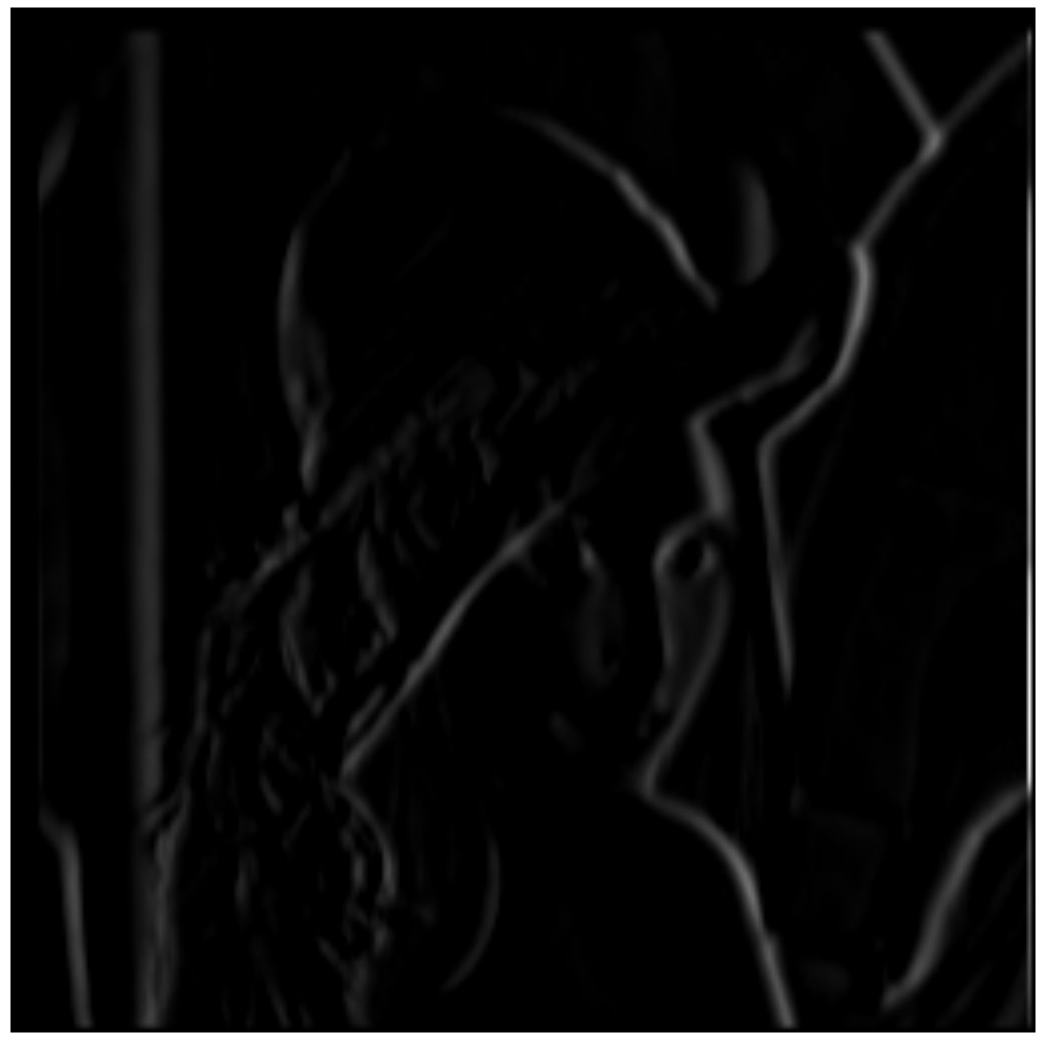


Fig 1c. Vertical Gradient

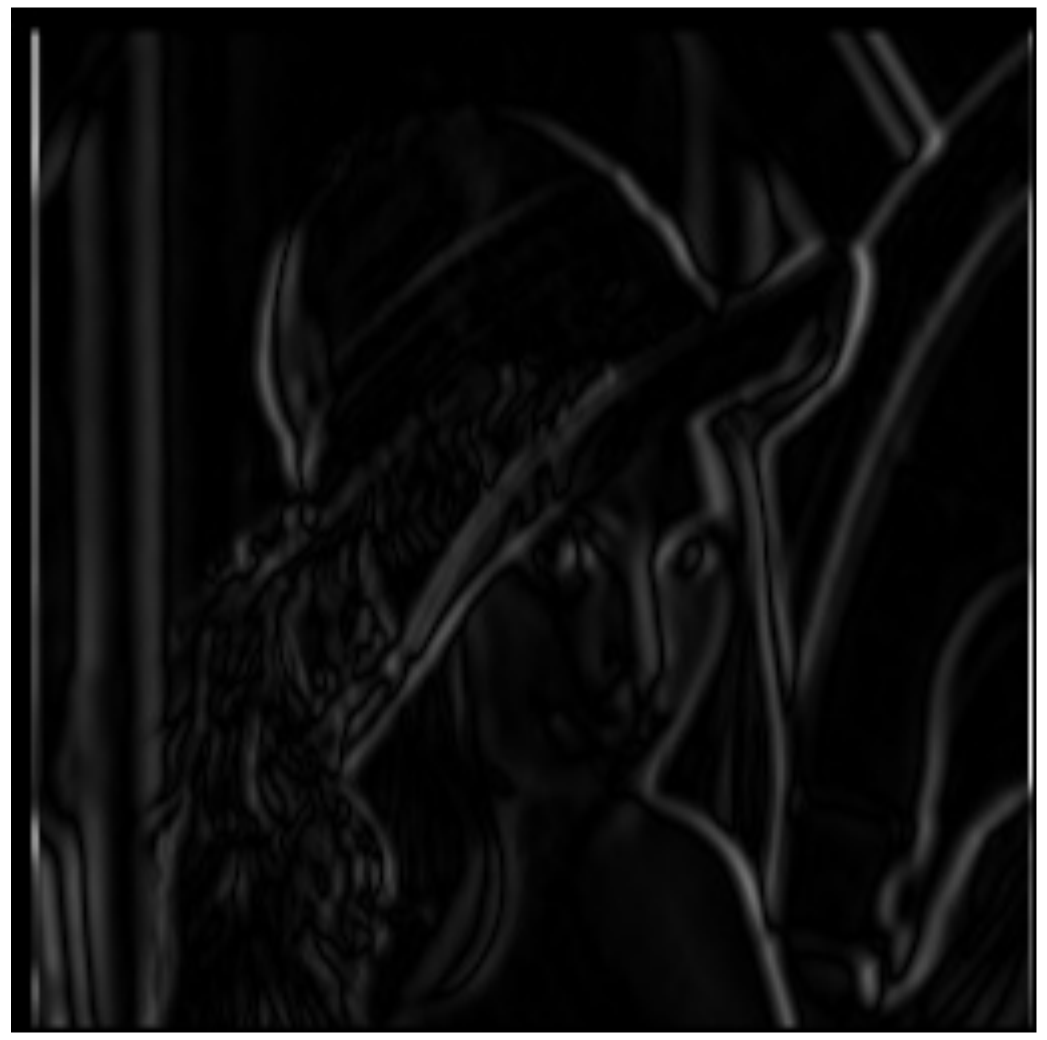


Fig 1d. Gradient Magnitude

Fig 1. Original Image

Output: Lady Image

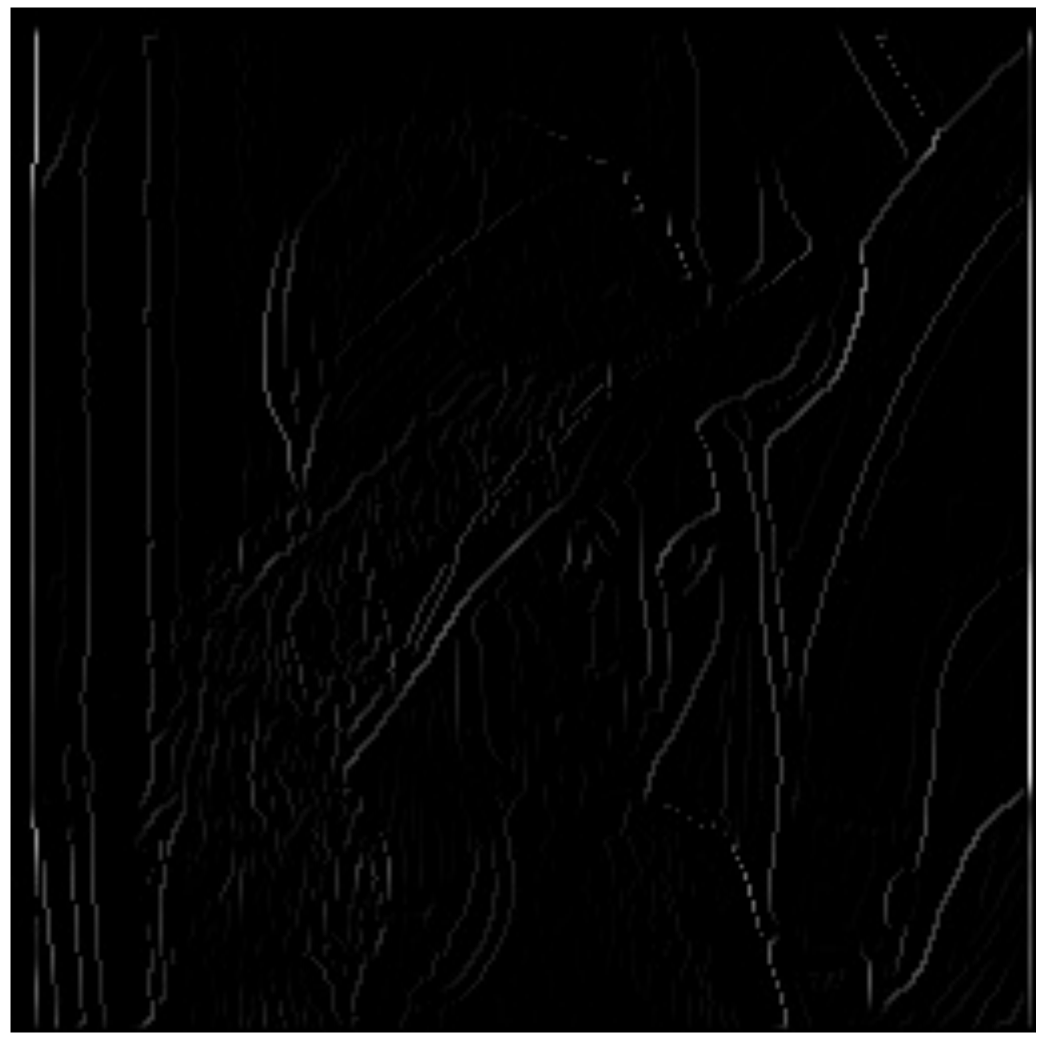


Fig 1e. Suppressed Image

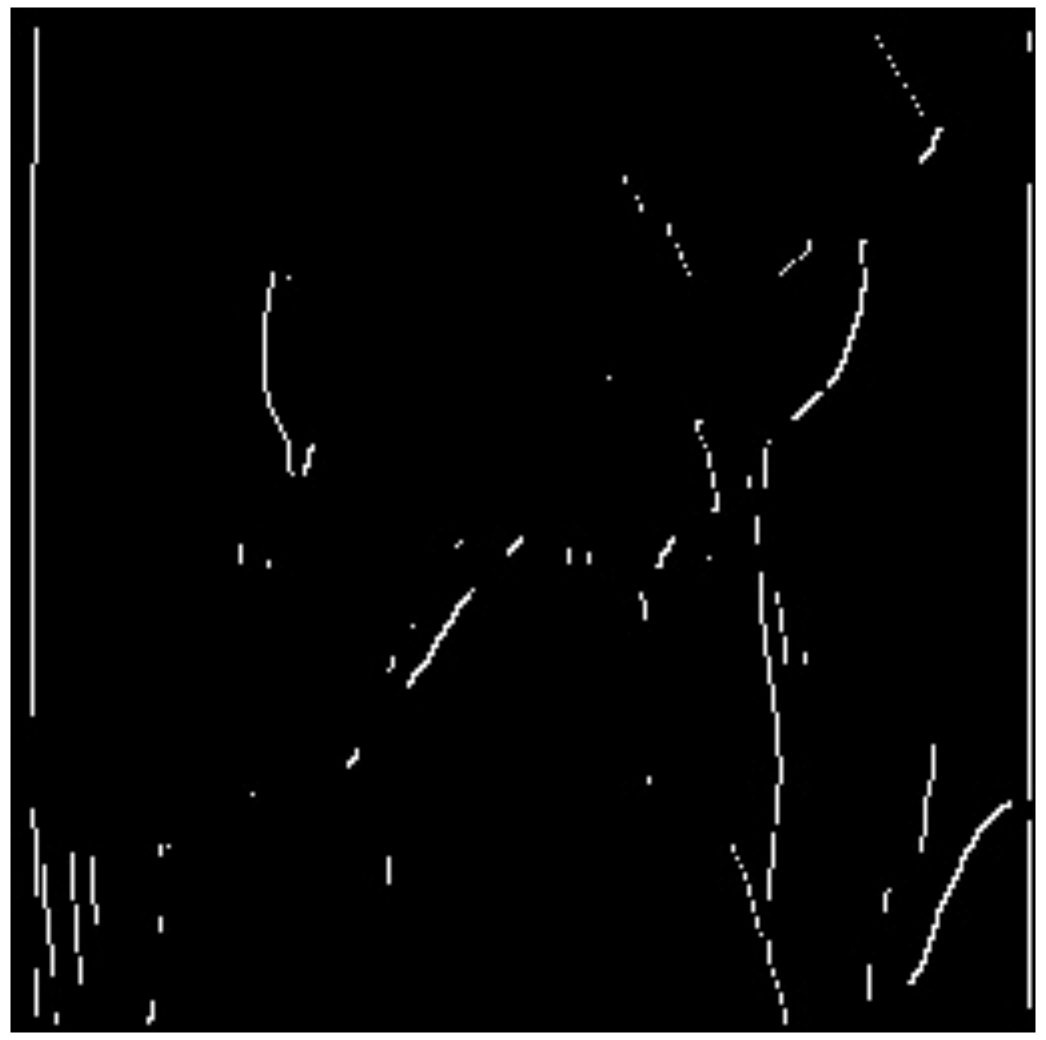


Fig 1f. Threshold 10%

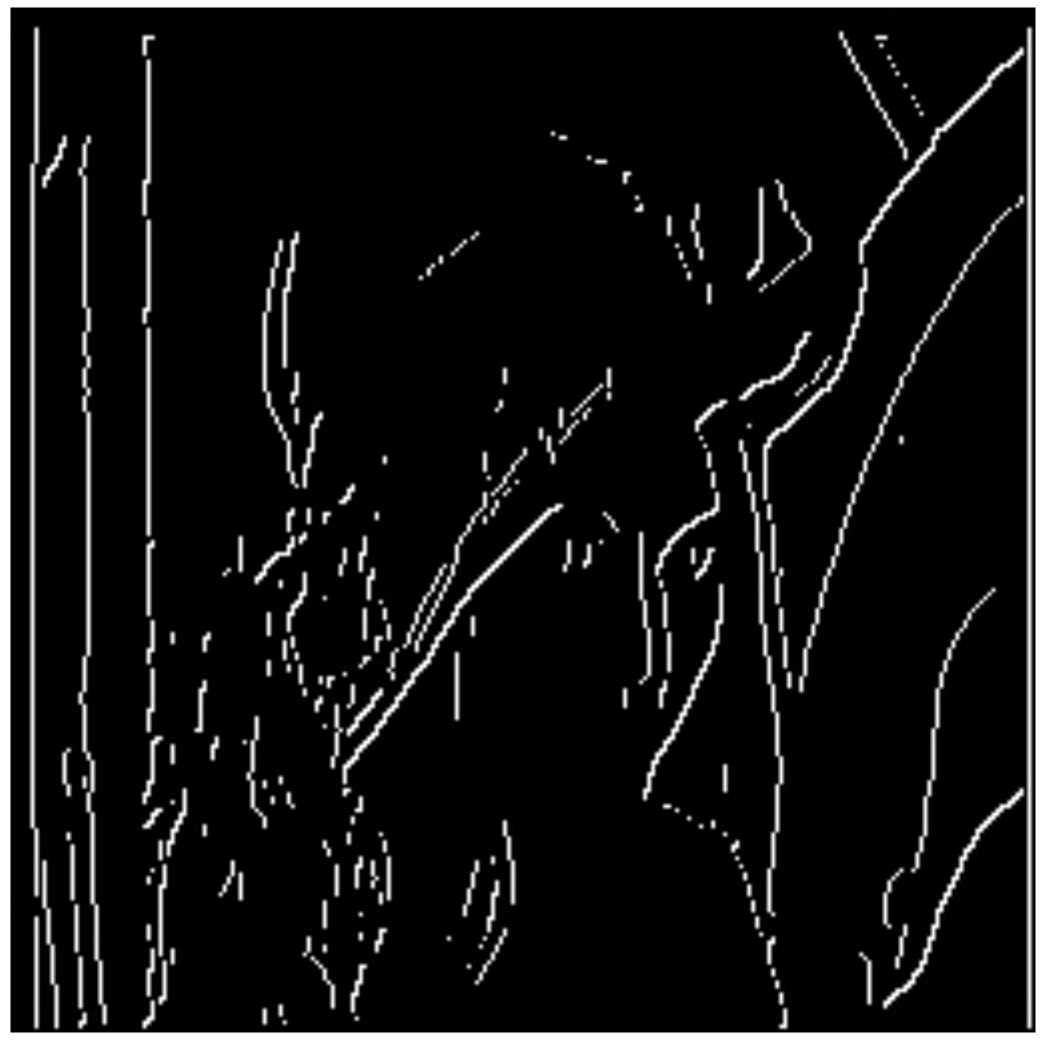


Fig 1g. Theshold 30%

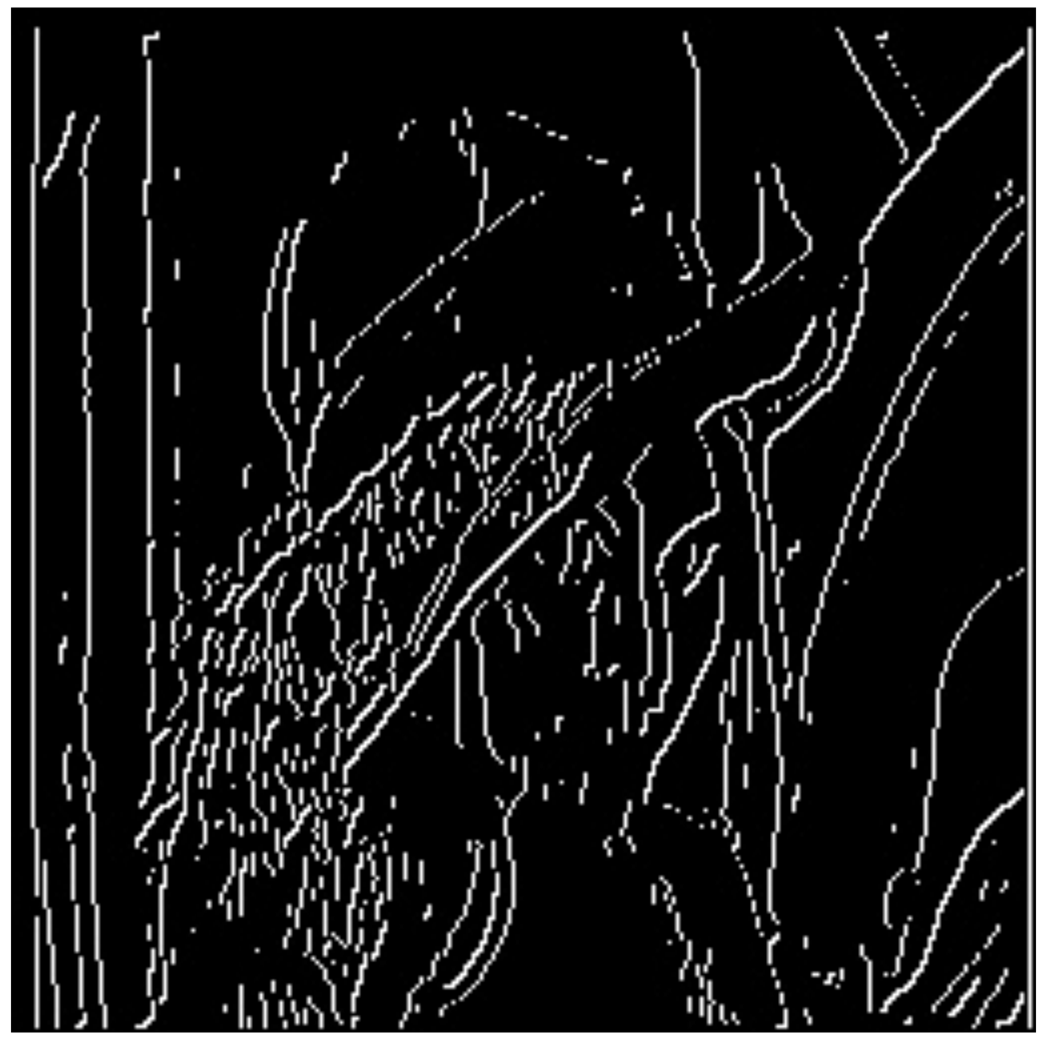


Fig 1h. Theshold 50%

Output: Zebra Crossing Image

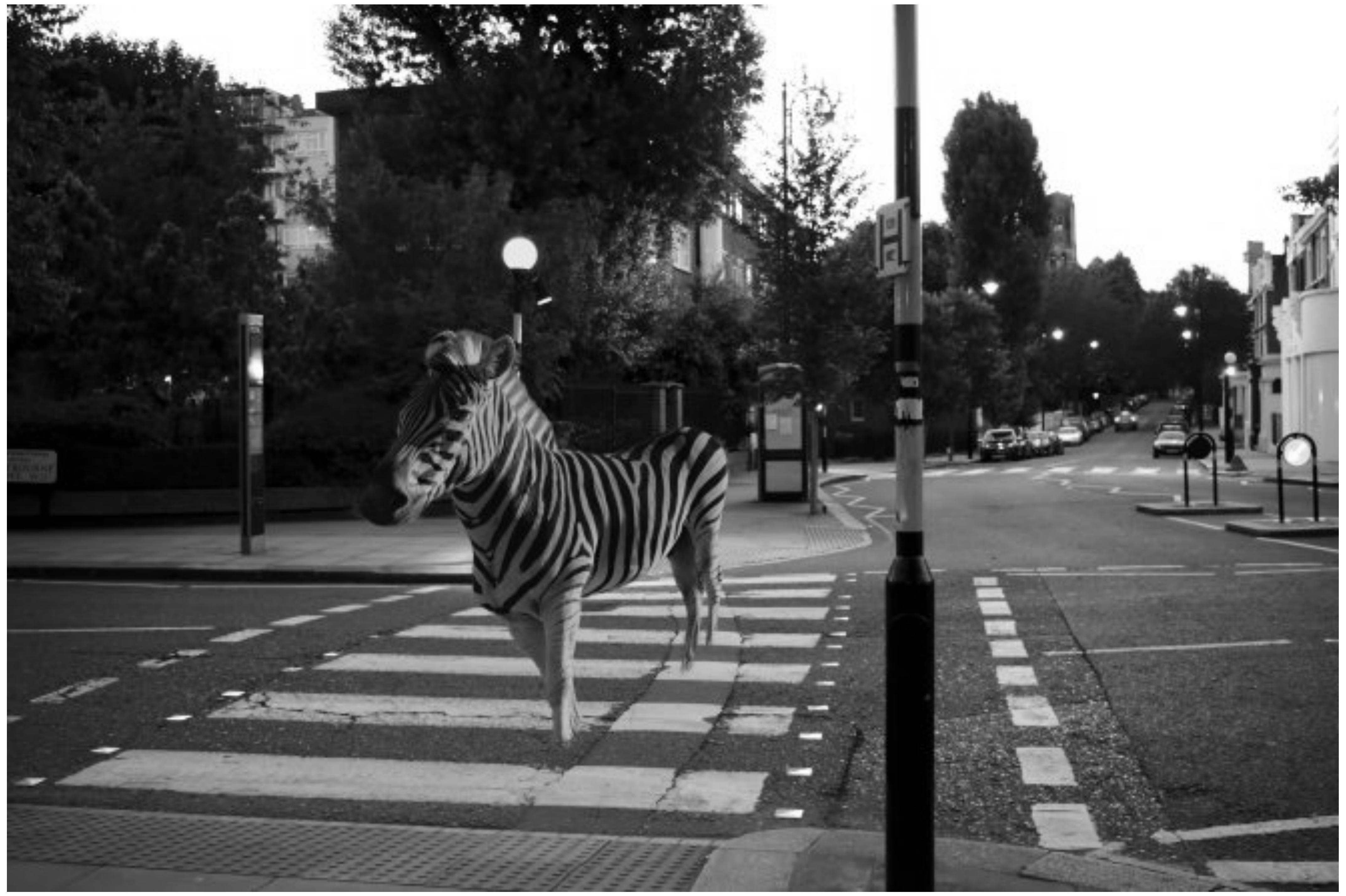


Fig 2. Original Image

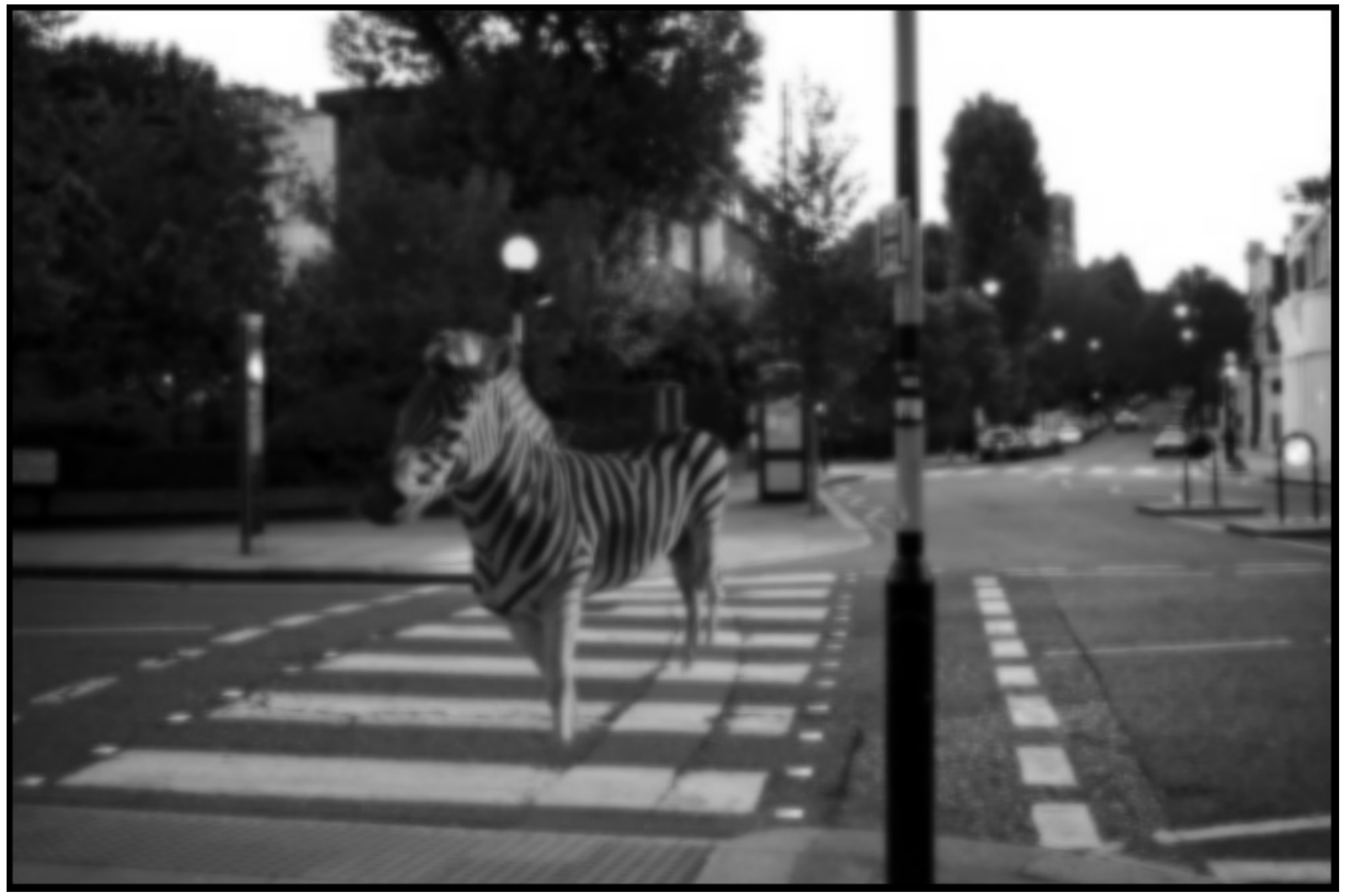


Fig 2a. Gaussian Filter

Output: Zebra Crossing Image

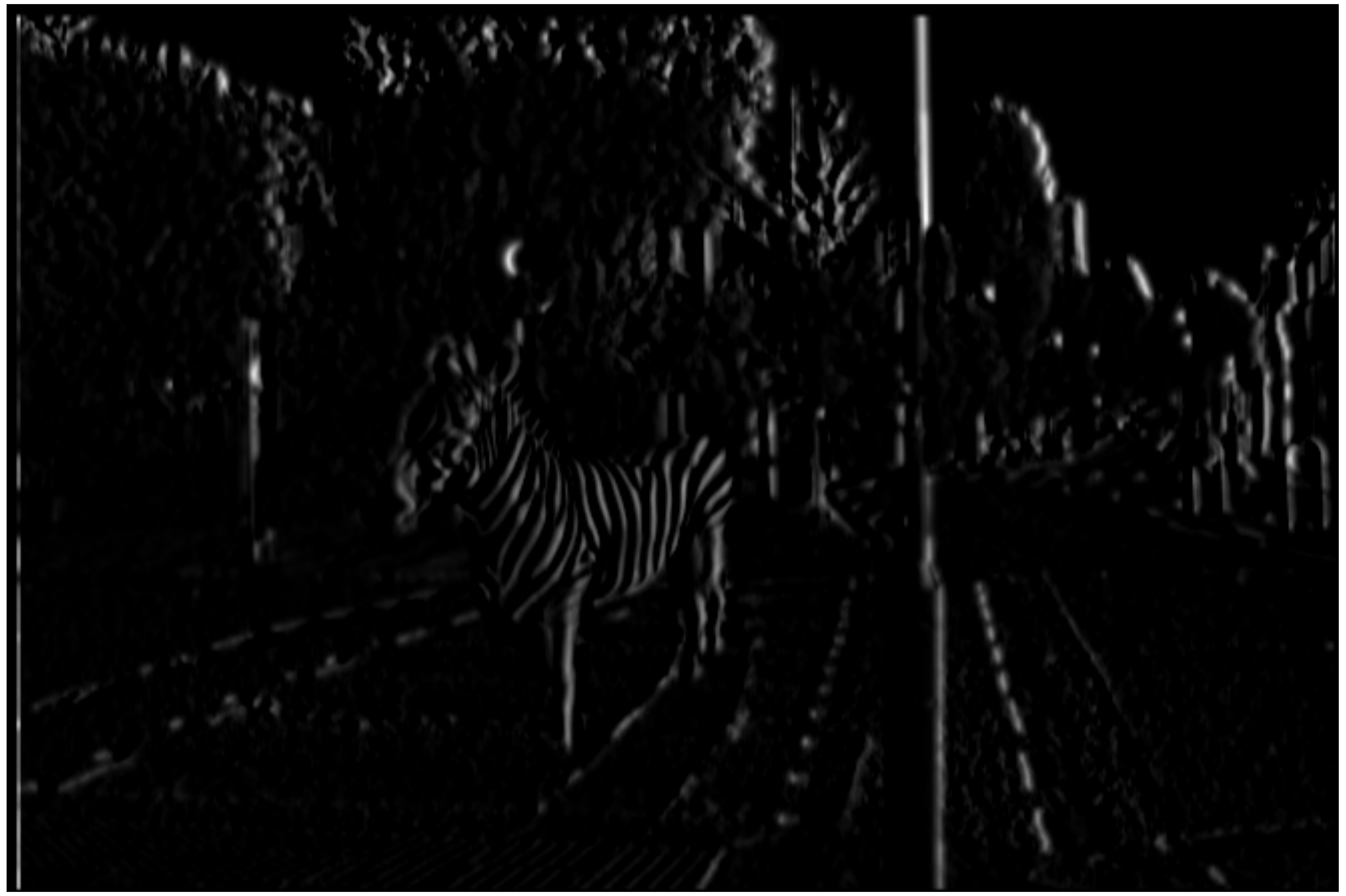


Fig 2b. Horizontal Gradient Gx

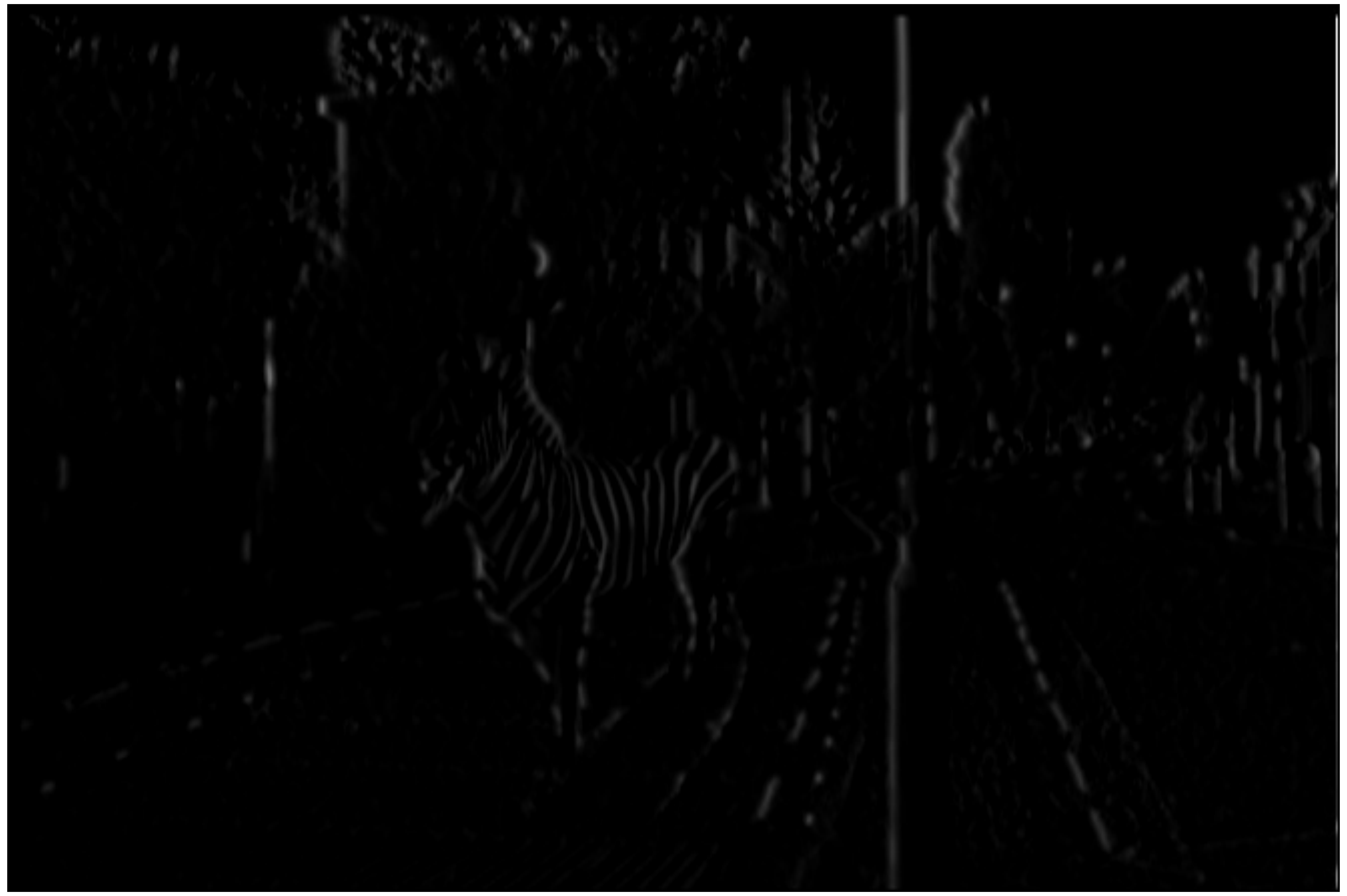


Fig 2c. Vertical Gradient Gy

Output: Zebra Crossing Image

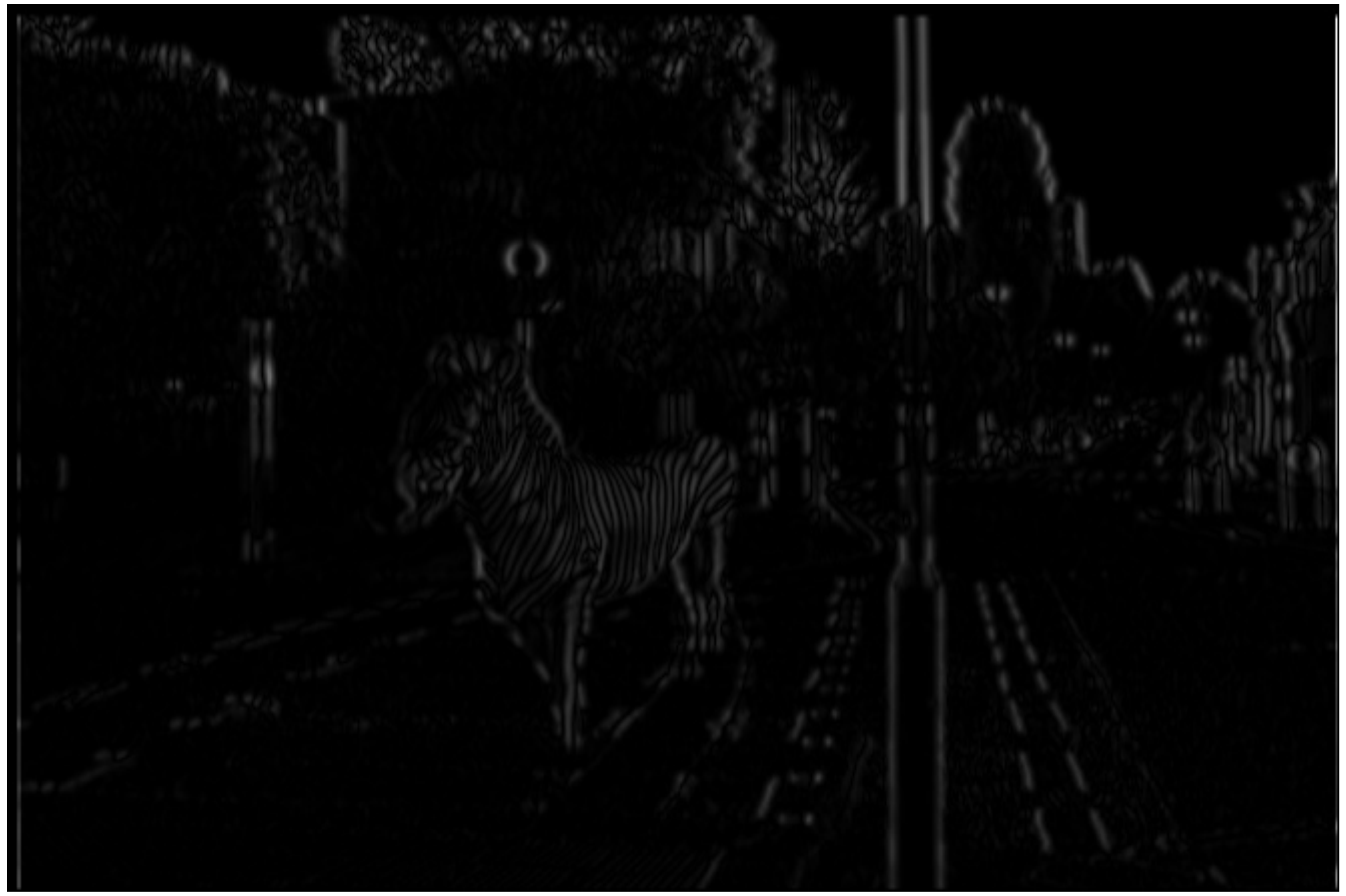


Fig 2d. Gradient Magnitude

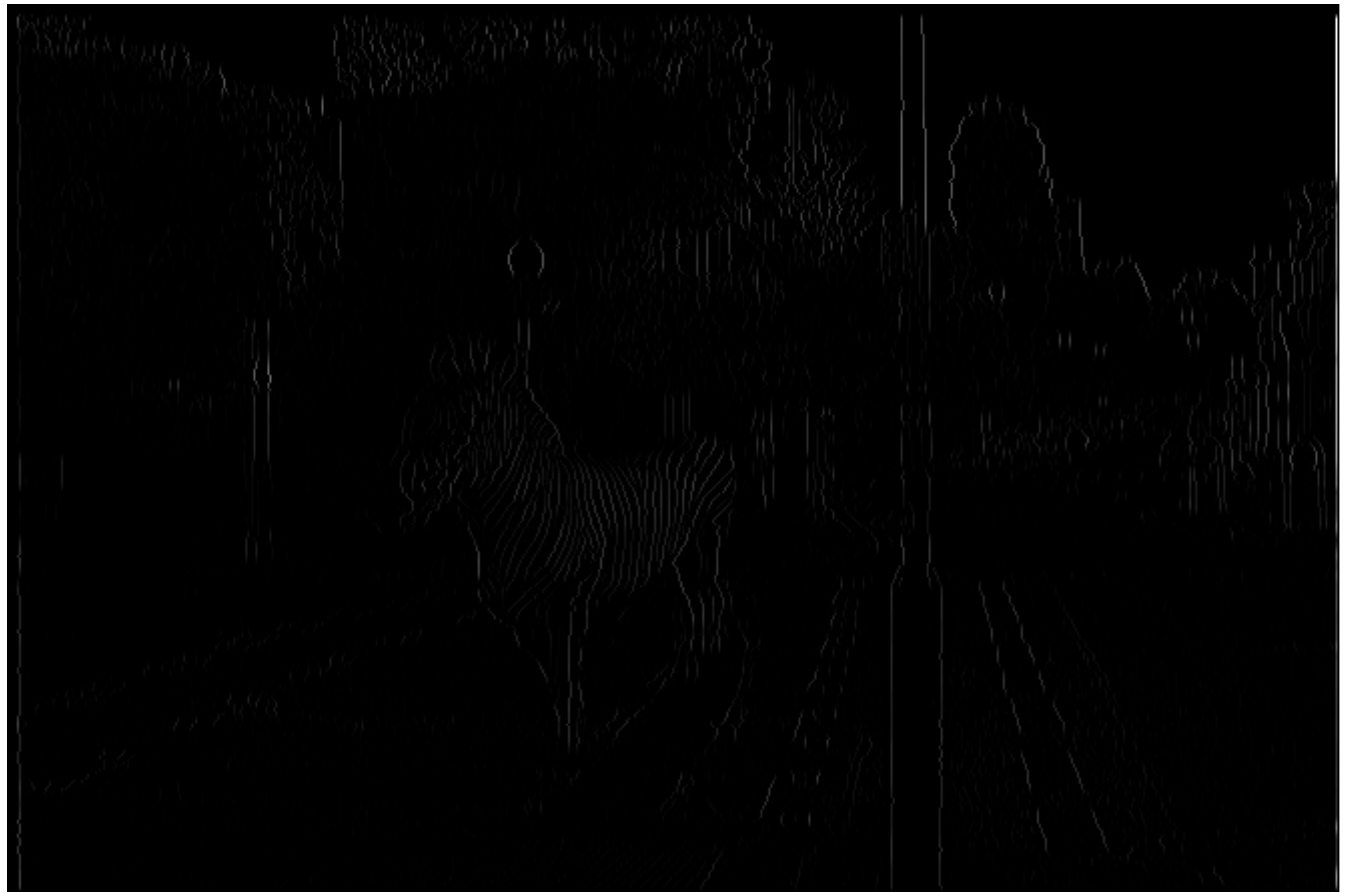


Fig 2e. Suppressed Image

Output: Zebra Crossing Image

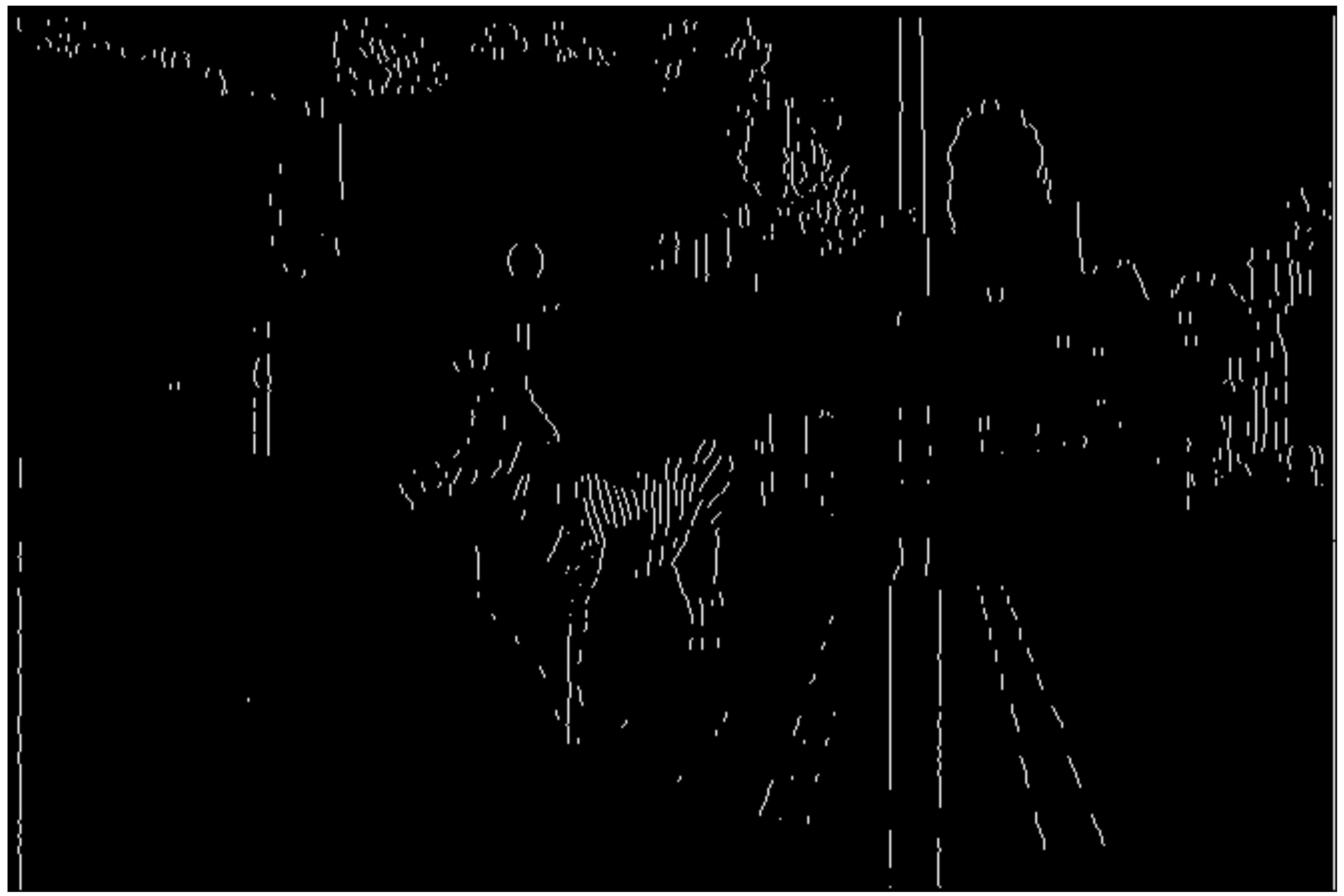


Fig 2f. Threshold 10%

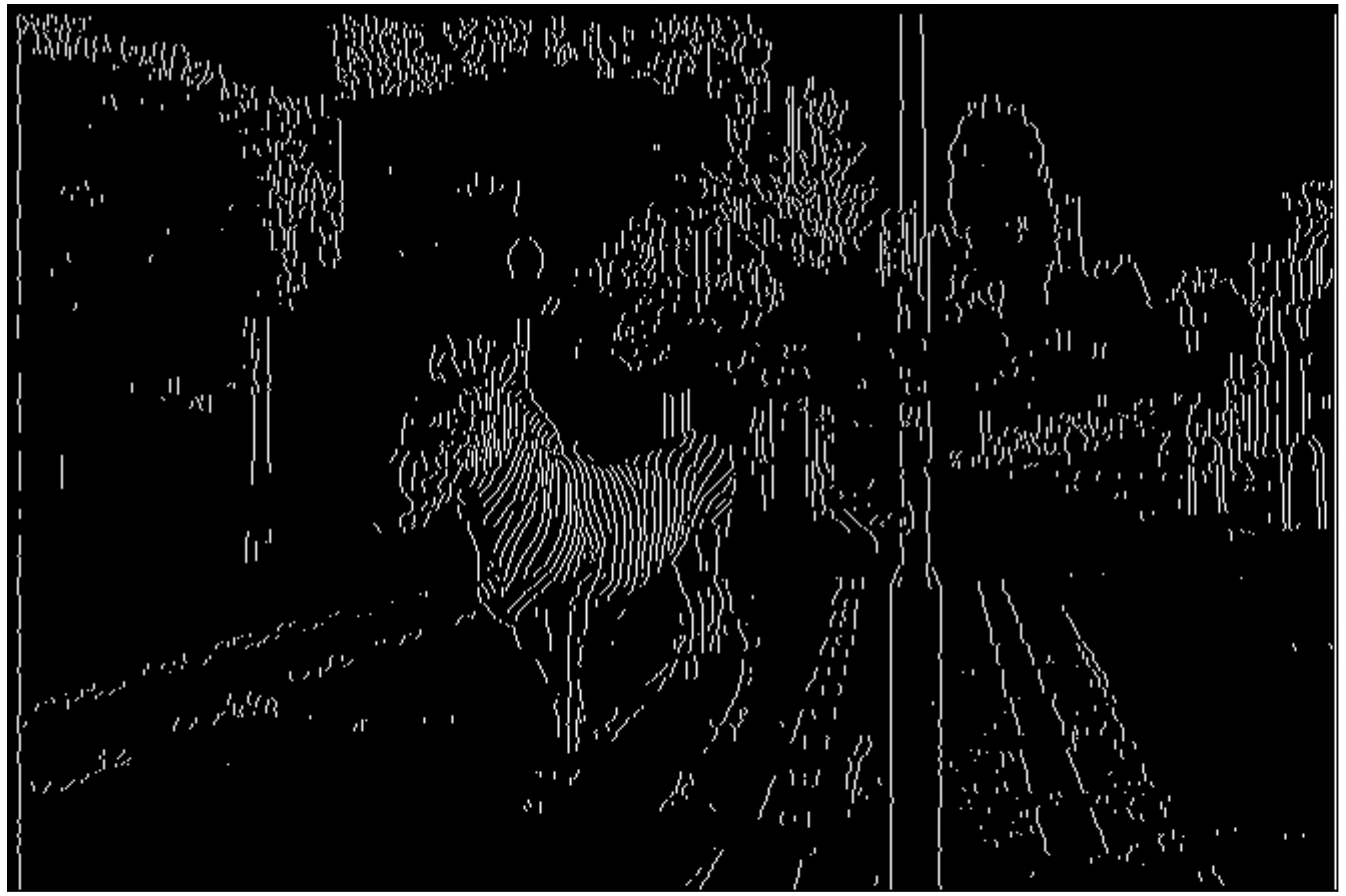


Fig 2g. Theshold 30%

Output: Zebra Crossing Image

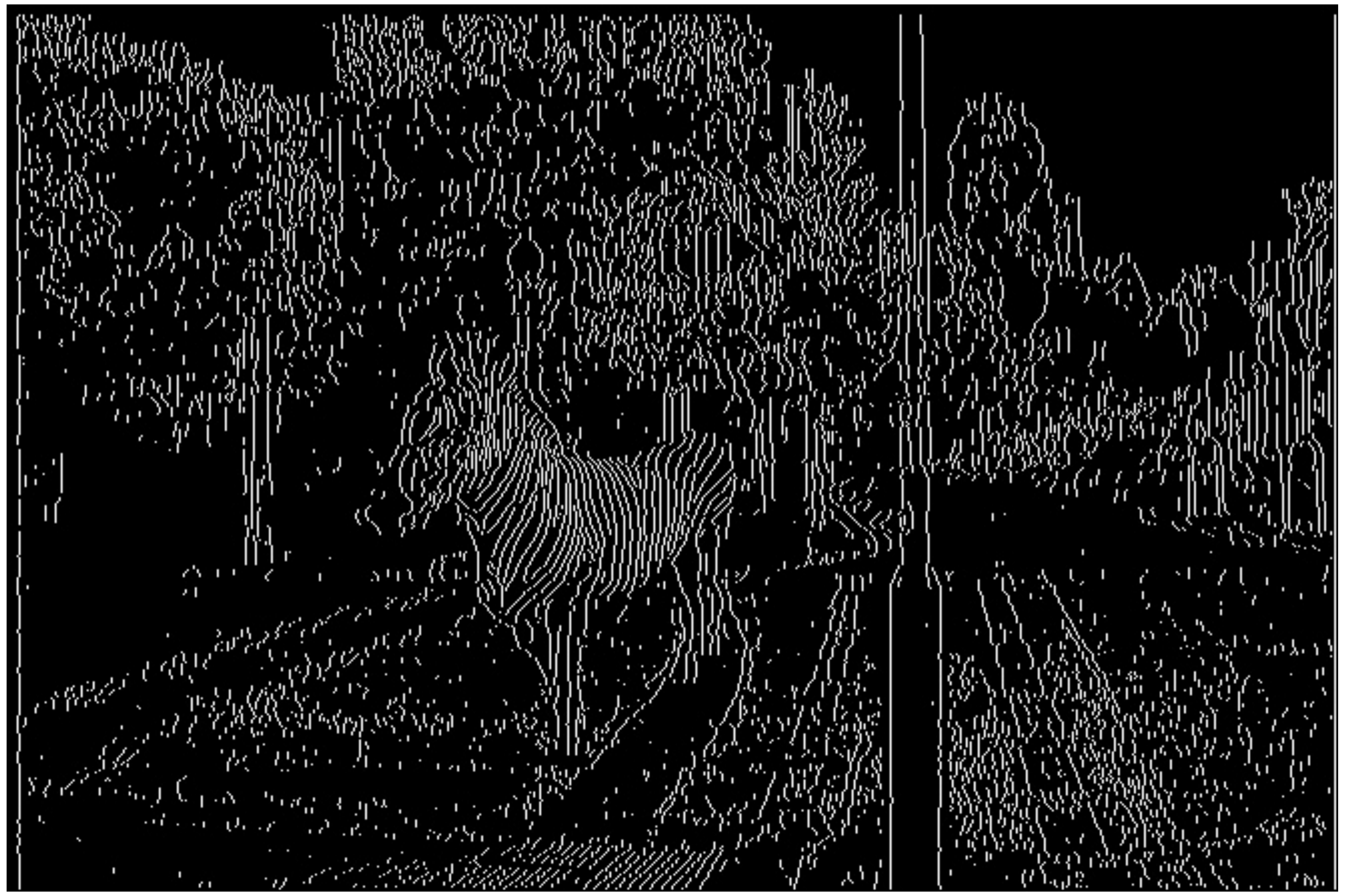
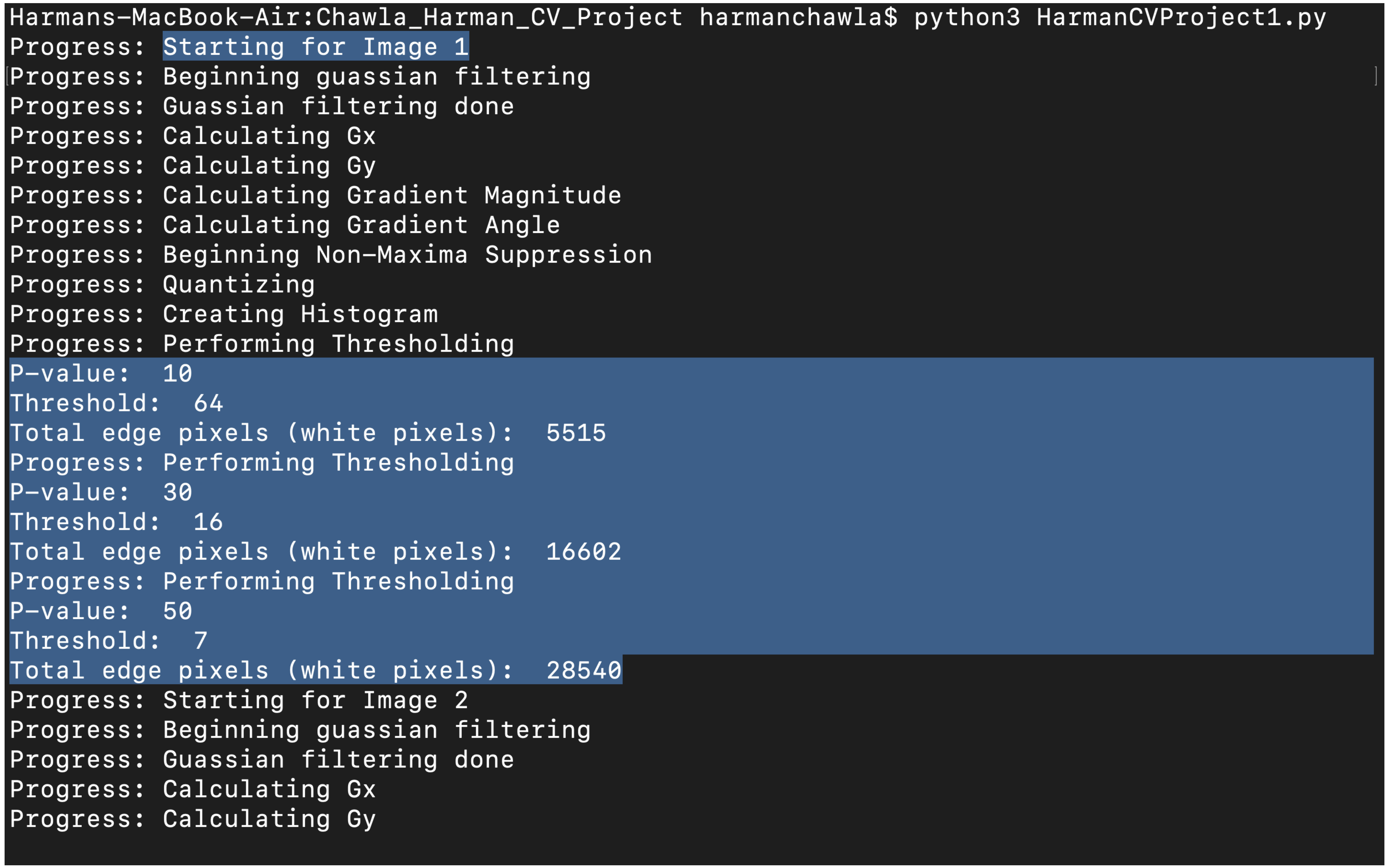


Fig 2h. Theshold 50%

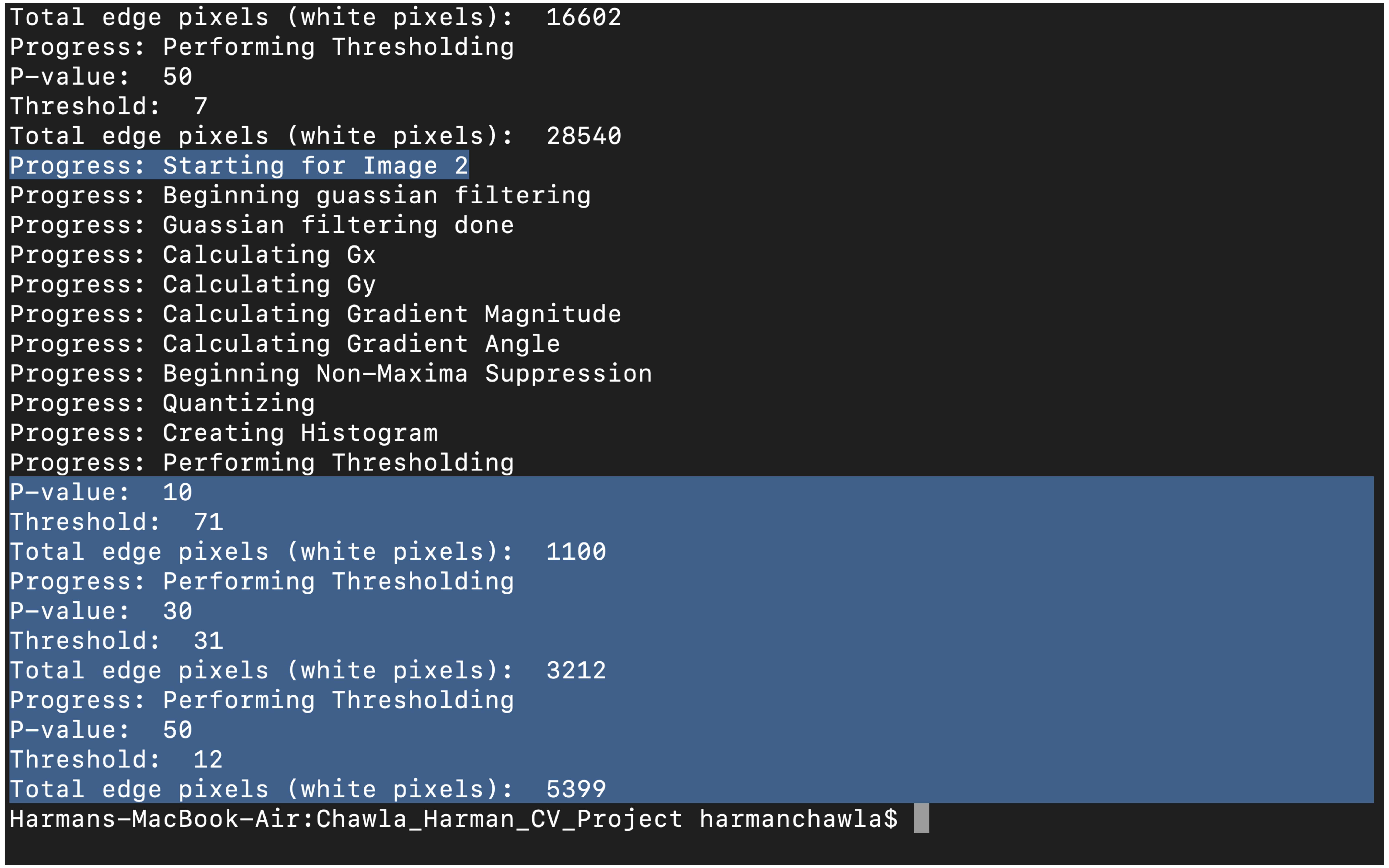
Thresholds and Edge Pixels



```
Harmans-MacBook-Air:Chawla_Harman_CV_Project harmanchawla$ python3 HarmanCVProject1.py
Progress: Starting for Image 1
Progress: Beginning guassian filtering
Progress: Guassian filtering done
Progress: Calculating Gx
Progress: Calculating Gy
Progress: Calculating Gradient Magnitude
Progress: Calculating Gradient Angle
Progress: Beginning Non-Maxima Suppression
Progress: Quantizing
Progress: Creating Histogram
Progress: Performing Thresholding
P-value:  10
Threshold:  64
Total edge pixels (white pixels):  5515
Progress: Performing Thresholding
P-value:  30
Threshold:  16
Total edge pixels (white pixels):  16602
Progress: Performing Thresholding
P-value:  50
Threshold:  7
Total edge pixels (white pixels):  28540
Progress: Starting for Image 2
Progress: Beginning guassian filtering
Progress: Guassian filtering done
Progress: Calculating Gx
Progress: Calculating Gy
```

Fig 3a. Output for Zebra Crossing Image

Thresholds and Edge Pixels



```
Total edge pixels (white pixels):  16602
Progress: Performing Thresholding
P-value:  50
Threshold:  7
Total edge pixels (white pixels):  28540
Progress: Starting for Image 2
Progress: Beginning guassian filtering
Progress: Guassian filtering done
Progress: Calculating Gx
Progress: Calculating Gy
Progress: Calculating Gradient Magnitude
Progress: Calculating Gradient Angle
Progress: Beginning Non-Maxima Suppression
Progress: Quantizing
Progress: Creating Histogram
Progress: Performing Thresholding
P-value:  10
Threshold:  71
Total edge pixels (white pixels):  1100
Progress: Performing Thresholding
P-value:  30
Threshold:  31
Total edge pixels (white pixels):  3212
Progress: Performing Thresholding
P-value:  50
Threshold:  12
Total edge pixels (white pixels):  5399
Harmans-MacBook-Air:Chawla_Harman_CV_Project harmanchawla$
```

Fig 3b. Output for Lady Image

Source Code:

```
'''
Made by: Harman Singh Chawla
Net ID: hsc367
Student ID: N14381201

Fall Semester 2018
CV Project - Canny Edge Detector

OpenCV and Numpy is used to load the image,
matrix initialization and float to int conversion only.

Math library has been used to calculate square root
and inverse of tan (gradient angle)

Not used for any matrix multiplication
or any other built-in function.

NOTE: If you wish to run your own image on the program,
rename it as "image.jpg", uncomment lines 40 and 467
and run the program.
Please make sure the image is in the same directory,
else specify the path.

'''

import cv2
import numpy as np
import math

''' reading the input image from the file system '''
path1 = 'sample1.bmp'
path2 = 'sample2.bmp'

img1 = cv2.imread(path1, 0)
img2 = cv2.imread(path2, 0)

# Again, If you wish to run your own image as a test
# Remove the # from the statement below
# and the last statement of the program (line 367)
# img3 = cv2.imread('image.jpg', 0)

''' initializing the guassian kernel '''
gaussian = [[1, 1, 2, 2, 2, 1, 1],
                        [1, 2, 2, 4, 2, 2, 1],
                        [2, 2, 4, 8, 4, 2, 2],
                        [2, 4, 8, 16, 8, 4, 2],
                        [2, 2, 4, 8, 4, 2, 2],
                        [1, 2, 2, 4, 2, 2, 1],
                        [1, 1, 2, 2, 2, 1, 1]]
```

```
''' Step 1: Performimg guassian filtering on the input image '''

def gaussianFiltering(img, gaussian):

    print('Progress: Beginning guassian filtering')
    '''
        calculate the sum of our kernel (140)
        just to make the program independent of the kernel values
    '''
    sum = 0
    for i in range(0, 7):
        for j in range(0, 7):
            sum += gaussian[i][j]

    ''' normalizing the kernel -> won't need to normalize the final image'''
    for i in range(0, 7):
        for j in range(0, 7):
            gaussian[i][j] = gaussian[i][j]/sum

    ''' number of rows in the image '''
    rowSize = len(img)

    ''' number of columns in the input image '''
    columnSize = len(img[1])

    '''
        initializing an array of zeros to store the new
        values after gaussian filteri
    '''
    filterImage = np.zeros((rowSize, columnSize))

    ''' Not computing the guassian filter for a 3 pixel border '''
    x = 3
    y = 3
    rowLimit = rowSize - 3
    columnLimit = columnSize - 3

    '''
        go over each row and column in the image
        and calculate the mask value for each pixe
    '''
    for i in range(3, rowLimit, 1):
        for j in range(3, columnLimit, 1):
            filterImage[x][y] = convolution(img, gaussian, i, j, 3)
            y = y + 1

        x = x + 1
        y = 3

    normalizedFilterImage = np.zeros((rowSize, columnSize))
    ''' converting float values to int '''
    normalizedFilterImage = filterImage.astype(int)
    cv2.imwrite('Gaussian Filter.jpg', filterImage)
```

```
	print('Progress: Guassian filtering done')

	gradientOperator(filterImage)
	return


'''
	Performing convolution
	The method takes in 4 parameters: image, kernel,
	location (i, j) at which the result is stored and
	border.

	Border can be thought as the floor(len(kernel))
	and basically signifies the extent of the neighboring
	pixels to be taken into account.

	(a,b) -> top left starting pixel

'''

def convolution(image, kernel, i, j, border):
	mask = 0
	maskSize = len(kernel)
	a = i - border
	b = j - border
	for k in range(0, maskSize):
		for l in range(0, maskSize):
			mask += image[a][b] * kernel[k][l]
			b = b + 1

		b = j - border
		a = a + 1

	return mask


''' Method calculates 4 arrays: Gx, Gy, Gradient Magnitude and Gradient Angle '''
def gradientOperator(img):

	''' Using Prewitt's operator as the kernel here '''
	gxkernel = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]
	gykernel = [[1, 0, -1], [1, 0, -1], [1, 0, -1]]

	'''
		Starting from the fourth row and column.
		If we start from the first row and column (i=0, j=0)
		then edges will be detected at i=3 and j=3
		as these values were set to 0 in the gaussian filtering.
	'''
	x = 4
	y = 4
	rowLimit = len(img) - 5
	columnLimit = len(img[1]) - 5
```

```
gmImage = np.zeros((len(img), len(img[1])))
gaImage = np.zeros((len(img), len(img[1])))
gx = np.zeros((len(img), len(img[1])))
gy = np.zeros((len(img), len(img[1])))

''' Step 2: Calculating Horizontal Gradient Gx '''
print('Progress: Calculating Gx')
for i in range(1, rowLimit, 1):
        for j in range(1, columnLimit, 1):
                gx[x][y] = convolution(img, gxkernel, i, j, 1)
                y = y + 1
        x = x + 1
        y = 4

''' normalizing the horizontal gradient '''
gx = normalize(gx)
normalizedgx = np.zeros((len(img), len(img[1])))
normalizedgx = gx.astype(int)

''' saving the horizontal gradient as an image '''
cv2.imwrite('Gradient X.jpg', normalizedgx)

''' Step 3: Calculating Vertical Gradient Gy '''
print('Progress: Calculating Gy')
x = 4
y = 4

for i in range(1, rowLimit, 1):
        for j in range(1, columnLimit, 1):
                gy[x][y] = convolution(img, gykernel, i, j, 1)
                y = y + 1
        x = x + 1
        y = 4

''' normalizing the verticle gradient '''
gy = normalize(gy)

normalizedgy = np.zeros((len(img), len(img[1])))
normalizedgy = gy.astype(int)

''' saving the verticle gradient as an image '''
cv2.imwrite('Gradient Y.jpg', normalizedgy)

'''
        Calculating Gradient Magnitude
        gimage[i][j] = sqrt(gx[i][j]*gx[i][j] + gy[i][j]*gy[i][j])
'''
print('Progress: Calculating Gradient Magnitude')
rowSize = len(gx)
columnSize = len(gx[1])
temp1 = 0.0
temp2 = 0.0

for i in range(0, rowSize, 1):
```

```
                for j in range(0, columnSize, 1):
                        ''' Calculating the gradient mangnitude '''
                        temp1 = gx[i][j] * gx[i][j] + gy[i][j] * gy[i][j]
                        gmImage[i][j] = math.sqrt(temp1)
                        temp1 = 0

        ''' Normalizing the Gradient Magnitude '''
        gmImage = normalize(gmImage)

        printgmImage = gmImage.astype(int)

        ''' Saving the Gradient Magnitude as an Image '''
        cv2.imwrite('Gradient Mangnitude.jpg', printgmImage)

        ''' Calculating gradient angle '''
        print('Progress: Calculating Gradient Angle')
        for i in range(rowSize):
                for j in range(columnSize):
                        if gx[i][j] is 0:
                                gaImage[i][j] = 90.0
                        elif gx[i][j] is not 0:
                                gaImage[i][j] = np.rad2deg(np.arctan2(gy[i][j], gx[i][j]))

                                if gaImage[i][j] < 0:
                                        gaImage[i][j] = 360 - math.fabs(gaImage[i][j])

                        '''
                                NOTE: The list gaImage[][] at this time has both positive and
negative
                                values in degrees. We can't simply take the absolute value as -45
deg is not
                                the same as 45 deg. Thus the if statement above is required -
aids in
                                non-maxima supression.
                        '''

        ''' Since decimal values won't effect the non-maxima supression. '''
        nonMaximaSuppression(gmImage, gaImage)


''' Step 4: Performing Non-Maxima Suppression '''
def nonMaximaSuppression(magnitude, angle):

        print('Progress: Beginning Non-Maxima Suppression')
        rowSize = len(angle)
        columnSize = len(angle[1])
        quantize = np.zeros((rowSize, columnSize))

        ''' Storing the values in a new array "quantize" '''
        for i in range(0, rowSize, 1):
                for j in range(0, columnSize, 1):
                        quantize[i][j] = angle[i][j]

                        '''
                Changing the range of values from [0,359] to [0,180]
                this doesnot affect the sectors assigned and
                reduces the number of comparisions by a factor of 2.
            '''
            if quantize[i][j] > 180.0:
                quantize[i][j] = 180.0 - quantize[i][j]

    '''
        So at this point the array has values between 0 and 180 (included)
        We assign value:
        0 for 0-22.5 and 157.5-180,
        1 for 22.5-67.5,
        2 for 67.5-112.5 and
        3 for 112.5-157.5
    '''
    print('Progress: Quantizing')
    for i in range(rowSize):
        for j in range(columnSize):
            if quantize[i][j] <= 22.5 or quantize[i][j] > 157.5:
                quantize[i][j] = 0
            elif quantize[i][j] > 22.5 and quantize[i][j] <= 67.5:
                quantize[i][j] = 1
            elif quantize[i][j] > 67.5 and quantize[i][j] <= 112.5:
                quantize[i][j] = 2
            elif quantize[i][j] > 112.5 and quantize[i][j] <= 157.5:
                quantize[i][j] = 3

    ''' Converting the float values to int '''
    quantize = quantize.astype(int)

    supression = np.zeros((rowSize, columnSize))
    ''' Comparing relevant pixel locations in Gradient Magnitude '''
    for i in range(rowSize):
        for j in range(columnSize):
            if magnitude[i][j] > 0:
                supression[i][j] = checkValues(magnitude, quantize[i][j], i, j)

    ''' Normalizing the suppressed image '''
    supression = normalize(supression)

    printSupressed = supression.astype(int)

    ''' Saving the suppressed image to the file system '''
    cv2.imwrite('Supressed.jpg', printSupressed)
    supression = supression.astype(int)
    makeHistogram(supression)


''' Method to compare location (i, j) with relevant neighbours '''
def checkValues(magnitude, a, i, j):

    # Check values to the left and the right
    if a == 0:
        if magnitude[i][j] > magnitude[i][j + 1] and magnitude[i][j] >= magnitude[i][j - 1]:
```

```
				return magnitude[i][j]
			else:
				return 0

		# Check values top-right and bottom-left
		elif a == 1:
			if magnitude[i][j] > magnitude[i - 1][j + 1] and magnitude[i][j] >= magnitude[i + 1][j
- 1]:
				return magnitude[i][j]
			else:
				return 0

		# Check values to the top and bottom
		elif a == 2:
			if magnitude[i][j] >= magnitude[i - 1][j] and magnitude[i][j] >= magnitude[i + 1][j]:
				return magnitude[i][j]
			else:
				return 0

		# Check values to the top-left and bottom-right
		elif a == 3:
			if magnitude[i][j] > magnitude[i - 1][j - 1] and magnitude[i][j] >= magnitude[i + 1][j
+ 1]:
				return magnitude[i][j]
			else:
				return 0


'''
	This method is used to normalize the array [0, x] -> [0, 255]
	given that the minimum value in the array is 0
	which is always true due to the gaussian filtering performed in the first step
'''


def normalize(img):
	oldMax = 0
	newMax = 255
	rowSize = len(img)
	columnSize = len(img[1])

	''' Calculating the maximum value in the array '''
	for i in range(rowSize):
		for j in range(columnSize):
			if img[i][j] > oldMax:
				oldMax = img[i][j]

	ratio = newMax / oldMax

	''' Performing normalization '''
	for i in range(rowSize):
		for j in range(columnSize):
			img[i][j] = img[i][j] * ratio
```

```
        return img


'''
        Method to create Histogram:

        We create a list of elements (histogram)
        each index corresponding to a grey level value
        and the value at the index representing the
        number of pixels with that grey level value.
        Eg: histogram[3] = 40 means there are 40 pixels with value 3
        Histogram can be made once as it doesn't
        change with the threshold.

'''
def makeHistogram(img):
        print('Progress: Creating Histogram')
        rowSize = len(img)
        columnSize = len(img[1])

        totalPixels = rowSize * columnSize

        ''' converting the image into a 1D array '''
        image1D = []
        for i in range(rowSize):
                for j in range(columnSize):
                        image1D.append(img[i][j])

        histogram = [0 for i in range(256)]
        loc = 0

        ''' counting the number of times a value occurs and storing it '''
        for i in range(totalPixels):
                loc = int(image1D[i])
                histogram[loc] = histogram[loc] + 1

        pTileThresholding(img, histogram, 10, "Threshold 10.jpg")
        pTileThresholding(img, histogram, 30, "Threshold 30.jpg")
        pTileThresholding(img, histogram, 50, "Threshold 50.jpg")


''' Method for performing P-Tile Thresholding '''
def pTileThresholding(img, histogram, percentage, name):

        print('Progress: Performing Thresholding')
        print('P-value: ', percentage)

        rowSize = len(img)
        columnSize = len(img[1])

        thresholdImage = np.zeros((rowSize, columnSize))
        ''' Saving the value of image to a new array '''
        for i in range(rowSize):
                for j in range(columnSize):
```

```
                thresholdImage[i][j] = img[i][j]

    '''
        I am not considering the pixels with value 0
        while caluculating the threshold as they
        were giving a white image for 30% and 50%
        due to the majority of pixels being 0
    '''
    totalPixels = rowSize * columnSize - histogram[0]

    ''' calculating the number of pixels in x% of the image '''
    topPPixels = int((totalPixels * percentage) / 100)
    threshold = 255
    sum = 0

    '''
        Finding the threshold i.e. for which grey level value
        do we cross the mark of top x% pixels
    '''
    for i in range(255, 0, -1):
        sum = sum + histogram[i]
        if sum >= topPPixels:
            threshold = i
            break

    print('Threshold: ', threshold)

    ''' performing thresholding '''
    for i in range(rowSize):
        for j in range(columnSize):
            if thresholdImage[i][j] >= threshold:
                thresholdImage[i][j] = 255
            else:
                thresholdImage[i][j] = 0

    thresholdImage = thresholdImage.astype(int)

    ''' counting the number of edge pixels '''
    edgePixel = 0
    for i in range(rowSize):
        for j in range(columnSize):
            if thresholdImage[i][j] != 0:
                edgePixel = edgePixel + 1

    print("Total edge pixels (white pixels): ", edgePixel)

    ''' Save the threshold image as a file '''
    cv2.imwrite(name, thresholdImage)
    return


''' Driver Function '''
def cannyEdgeDetector():
    print('Progress: Starting for Image 1')
```

```
        gaussianFiltering(img1, gaussian)

        print('Progress: Starting for Image 2')
        gaussianFiltering(img2, gaussian)

        # print('Progress: Starting for Image 3')
        # gaussianFiltering(img3, gaussian)

        return


cannyEdgeDetector()
```