ELE510
EDGE DETECTION USING IMAGE GRADIENTS(ref. Canny)

Yemisi Teju Olasoji, Daniel Fylling

Nov 12th, 2023

Contents

1 Introduction **4**

2 Theory **4**
- 2.1 Convert Image to Grayscale . . . 4
- 2.2 Gaussian Blur . . . 4
- 2.3 Intensity Gradient Calculation . . . 4
- 2.4 Non-Maximum Suppression . . . 4
- 2.5 Hysteresis Thresholding . . . 5
- 2.6 Edge Tracking . . . 5
- 2.7 Quantitative Edge Detection Metrics . . . 5

3 Implementation **6**

4 Experiments and Results **7**
- 4.1 Visualization of Intermediate Results . . . 7
- 4.2 Effect of Smoothing . . . 8
- 4.3 Effect of Noise . . . 9
- 4.4 Effect of Different Scales . . . 11
- 4.5 Comparison with Other Edge Detection Methods . . . 13

5 Conclusion **14**

A Appendix A **15**
- A.1 Visualization of Intermediate Results . . . 15
- A.2 Effect of Smoothing . . . 16
- A.3 Effect of Noise . . . 18
- A.4 Effect of Different Scales . . . 20
- A.5 Comparison with Other Edge Detection Methods . . . 22
- A.6 Dilate . . . 24

Abstract

Edge detection is a fundamental image processing technique used to identify boundaries within images. This project explores the theory and implementation of edge detection methods, with a focus on the Canny edge detection algorithm. Through experiments, we will demonstrate how this method functions for various types of edges,scales and noise levels comparing it with alternative detectors Sobel, Prewitt and Laplacian of Gaussian

1 Introduction

Edge detection is an essential image processing technique commonly employed in various computer vision applications, including data extraction, image segmentation, feature extraction, and pattern recognition. In more practical terms edge detection has been used in varying fields such as detecting cracks on the concrete surface of tunnels for health monitoring and maintenance of Chinese transport facilities [1], and segmentation of medical images [2]. This technique helps reduce the amount of noise and irrelevant details in an image while retaining its structural information. As a result, edge detection plays a crucial role in enhancing the accuracy and performance of computer vision algorithms.

The purpose of this project is to delve into the theory and practical aspects of edge detection using image gradients. We will primarily focus on the Canny edge detection method, which remains a popular algorithm due to its good performance [3]. The project aims to explain the core concepts of edge detection, discuss the theory behind the Canny algorithm, and showcase the results of experiments conducted to evaluate how the method works for different types of edges, scales and noise levels in comparison with other edge detectors

2 Theory

In this section, we discuss the implementation of the Canny edge detection algorithm. We explore the underlying theory and its practical application for detecting edges in images.

Canny edge detection is widely regarded as one of the most popular and effective methods for edge detection in Computer Vision. It employs a multi-stage algorithm to detect a wide range of edges in images. The Canny edge detector is a classic algorithm for detecting intensity edges in a grayscale image that relies on the gradient magnitude. The algorithm was developed by John F. Canny in 1986. It is a multi-stage algorithm that provides good and reliable detection. This algorithm can be broken down into six basic steps:

2.1 Convert Image to Grayscale

Processing a grayscale image is generally faster and requires less memory compared to processing colour images. The simplicity of grayscale images reduces the computational complexity and eases the implementation of edge detection algorithms. Moreover, structural features such as edges are often more evident in grayscale images because they highlight variations in luminance or intensity, which are less affected by colour changes.

2.2 Gaussian Blur

This step in Canny edge detection involves preprocessing the image to remove any noise or blur. This is usually achieved by applying a Gaussian filter which is an operator that helps in removing the noise in the input image which enables further processing to be smooth and flawless. The sigma value has to be set appropriately for better result.

2.3 Intensity Gradient Calculation

Sobel filter is to be used in this process. Sudden intensity change is the edge and infact the intensity change of the pixel is the edge. The resultant Sobel operated image is referred to as: Gradient Magnitude of the image. We preferred Sobel operator and it's the general approach. The resulting Gradient Approximation can be calculated with $|\mathbf{G}| = \sqrt{gx^2 + gy^2}$ where gx and gy are the components of the vector in the x and y directions, respectively.

The G will be compared with the threshold and with which one can understand the taken point is an edge or not. The Formula for finding the edge direction is: Theta = inv tan(ay/ax)

2.4 Non-Maximum Suppression

The gradient magnitude operators discussed previously normally obtain THICK EDGES but the final image is expected to have thin edges. Hence, this process enables us to derive THIN EDGES from thicker ones through the underlisted process: a) We related the identified edge direction to a direction that can be sketched in the image i.e. it is a prediction of how the movement of edges could happen e.g. If we take a 3x3 matrix as a reference, it is all about the colors.

Non maximum suppression is a process where suppression of the pixels to zero which cannot be considered as an edge is carried out. This enables the system to generate a THIN LINE in the output image. This result is obtained before Thresholding and smoothing.

2.5 Hysteresis Thresholding

As observed from the previous stage, Non -maximum thresholding has not provided excellent results as there is still some noise the output image even raises a threat in mind that some of the edges shown may not really be so and some edges could be missed in the process. Hence, here has to be a process to address this challenge which is THRESHOLDING We need to go through double thresholding and in this process, we have to set two thresholds. One, a High and another Low. Assume HIGH threshold to be value of 0.8. Any pixel with a value above 0.8 is seen as a STRONG EDGE. Another threshold, the lower one can be 0.2 and any pixel below this value is not an edge at all and hence we set them to 0. Now, what about the values in between 0.2 and 0.8? They may or may not be an edge. These are referred to as weak edges and there must be a process to determine which of the weak edges are actual edges so as not to miss them.

2.6 Edge Tracking

We can call the weak edges which are connected to stronger edges as Strong and actual edges and retain them while all weak edges which are not connected to stronger ones are to be removed. Now, the process is completed, and we get an output result.

2.7 Quantitative Edge Detection Metrics

In order to objectively evaluate the performance of edge detection algorithms, quantitative edge detection metrics are essential because they offer a way to gauge how well these algorithms perform in terms of identifying boundaries within images.The edge detection metrics that will be considered are : Ground truth,Precision,Recall and the F-measure. Researchers can optimize edge detection algorithms based on particular criteria and features of the images being examined as these measures allow for a methodical and impartial evaluation of edge detection methods.

Ground Truth Image

In image processing and computer vision, the ground truth image is essential because it provides an objective standard by which different algorithms' performance, like edge detection, may be evaluated.Ground truth image is an ideal or accurate representation of the required features within an image [4]. It contains accurate object boundary delineations that indicate expected edge locations when it comes to edge detection. The ground truth image serves as a benchmark for evaluating the performance of edge detection algorithms. Algorithms generate edges or boundaries within an image, and their results are compared against the ground truth to measure accuracy.

Precision

Precision is a metric used to measure how well an edge detection algorithm predicts the positive outcomes. It calculates the ratio of correctly predicted true positives to the total of correctly predicted false positives. It is calculated by dividing the true positive by the overall positives.

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

Recall

Recall is a a measure that assess the ability of an edge detection algorithm to capture all relevant edges present in the ground truth. It calculates the ratio of true positive predictions to the sum of true positives and false negatives. Recall is particularly important when ensuring that all actual edges are detected.

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

F1-Score

The F1-score is a composite metric that balances precision and recall, providing a single value that considers both false positives and false negatives. It is the harmonic mean of precision and recall, offering a comprehensive measure of overall performance.The result is a value between 0.01 for the worst and 1.0 for the best f-measure.The F1 Score is valuable when seeking a balance between precision and recall in edge detection applications.

$$\text{F-measure} = \frac{\text{true positive}}{2^{*}\text{true positive} + \text{false positive} + \text{false negative}}$$

3 Implementation

The coding approach or style is very much inspired by previous labs in the course. Aside from copying a function for non maximum suppression, templates for plotting and data frame generation from previous self-made works, all code has been developed from scratch for this project. ChatGPT helped speed up the process of trouble shooting faulty code tremendously and was used to help appropriately comment the code. It was initially considered to perform the entire experimental process for a second, fundamentally different image, but it was decided against, since it would make the report less focused and might prove insights more difficult to extract. In Table 1 we present the Libraries used in Python to produce the results in this project.

Library	**Explanation**
cv2 (OpenCV)	Used for image processing tasks.
numpy	Used for numerical operations.
matplotlib.pyplot	Used for plotting visualizations.
pandas	Used for creation of dataframe.
skimage.filters	Used for applying Sobel and Prewitt filters and generating noise.
sklearn.metrics	Used for calculating precision, recall, and F1 score for model evaluation.

Table 1: Python libraries and their uses in the code

Ground Truth Images

The images used for computation of evaluation metrics was produced by manually tracing the observable edges of the original image in paint.net software. To generate ground truth images for down scaled versions, the original ground truth was down scaled in the same software, levels adjusted manually and a threshold set to convert to 1-bit black/white output only.

Scaled Down Images

The down sized input images where produced in a similar manner as the down sized ground truth images, using resizing in external software (paint.net). This could have been done with coding, and can be seen as criticism to the method chosen. Down sizing the image using coding would give more control over the process.

Evaluation Metrics

While researching the scikit-learn webpage how to setup for our type of classification problem, it was found that the most appropriate weighting for precision, recall and F1-score be calculated based on a binary input. This makes sense since we just have two classes; edges and non-edges. However an input formatting error was produced while trying to run the sklearn functions with this setting, and we were unable to figure out how to format the input correctly for this to work. The average was instead set to 'weighted', and the following result were found meaningful, so this setting was used for the remainder of the project.

For each experiment in the next section a standalone code was created. This coding approach can be regarded as wasteful as a lot of code is repeated several times, like loading image, converting to grayscale and plotting. The upside of this is that the code should be easy to read and the results easily replicated.

4 Experiments and Results

After looking briefly into the intermediate results of the Canny algorithm this section focuses on the quantitative evaluation of edge detection results. We utilize metrics such as precision, recall, and F1-score to objectively assess the performance of the Canny edge detection algorithm. These metrics provide a comprehensive evaluation of its effectiveness.

Through this section we will use the same image for comparison purposes. The image was generated using OpenAI's image generator DALL·E 2. The image is digitally created, but in terms of gradients and edges it still presents enough complexity to give a foundation for discussion. The image resolution is 1024x1024 pixels. The input image is shown in Figure 1.

Figure 1: A rock climbing camel wearing a salsa outfit

4.1 Visualization of Intermediate Results

In this experiment, we focus on visualizing the intermediate steps of the Canny edge detection algorithm. We create visualizations of the gradient magnitude, non-maximum suppression, and edge tracking by hysteresis. The process was applied to the image without pre-smoothing. Pre-smoothing is commonly applied to images before edge detection, but we will get into that later. After processing, the result is shown in Figure 2. Code used to produce the results are found in Appendix A.1.

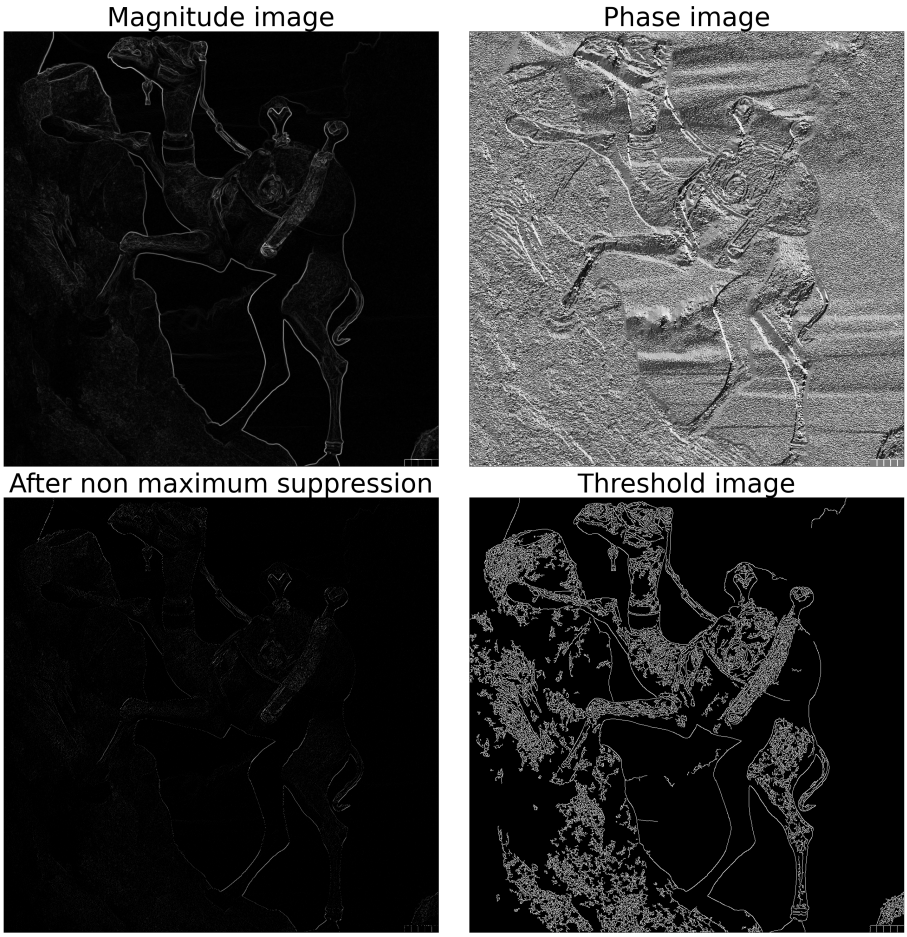

Figure 2: Intermediate stages for edge detection of Camel

Observations

- **Gradient Magnitude and Phase Images**: These images reflect the location and direction of edges, these images retained high-frequency details due to the absence of smoothing, resulting in pronounced noise.
- **Non-Maximum Suppression**: It is hard to make noticeable observations in the output image. Theoretically this step is less effective without prior smoothing, allowing minor variations in pixel values to stay as possible edges.
- **Canny Edge Detection**: Due to the lack of smoothing, final threshold image looks very "messy". This shows the necessity for noise reduction before edge detection, especially in a relatively complex image as we have presented.

4.2 Effect of Smoothing

Next we will see what happens if we apply different degrees of smoothing to the images, before applying edge detection. We compare the results with the initial approach to determine the effect of the smoothing. The result is shown in Figure 3. Code used to produce the result is found in Appendix A.2. As promised we will also compare the results numerically before commenting. Here all edge images will be compared against the Ground Truth image as mentioned in Implementation.

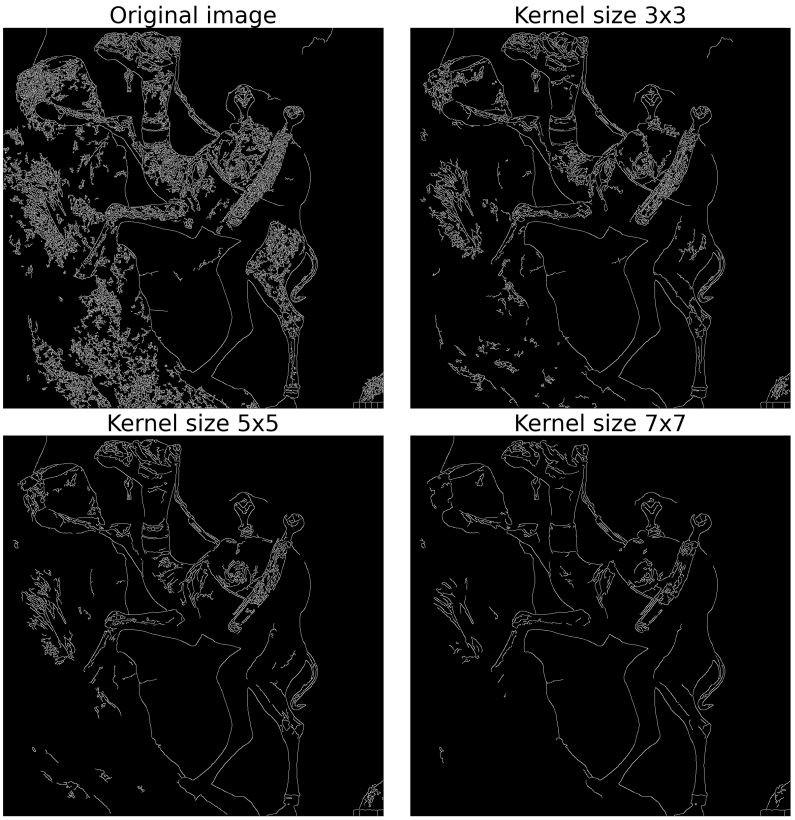

Figure 3: Intermediate stages for edge detection of Camel

Image	Precision	Recall	F1-score
Original	0.21	0.60	0.30
Kernel size 3x3	0.38	0.60	0.45
Kernel size 5x5	0.50	0.62	0.54
Kernel size 7x7	0.57	0.50	0.52

Table 2: Evaluation metrics for different degrees of smoothing

Observations

- The original image has the lowest precision, suggesting that it has many false positives. This makes sense compared to the visual results where we see that the original image clearly predicts the largest amount of edges.
- As the kernel size increases, the precision increases, which means the edges detected in the smoothed images more accurately represent the true edges.
- The recall is quite stable until it takes a sudden fall at the 7x7 kernel. This indicates that true edges are lost due to the amount of smoothing.
- The F1-score increases with the kernel size up to 5x5, then drops, indicating the best balance between precision and recall.

Based on these results we will carry the 5x5 kernel forward for further analysis.

4.3 Effect of Noise

Here we will look into the effect of noise on edge detection using the Canny algorithm. We keep the blurred 5x5 image as a reference and add varying levels of Gaussian noise to it, then analyze how noise affects the algorithm's performance. The performance of the edge detection is visually presented in Figure 4. The quantitative evaluation against the ground truth image is shown in Table 3.

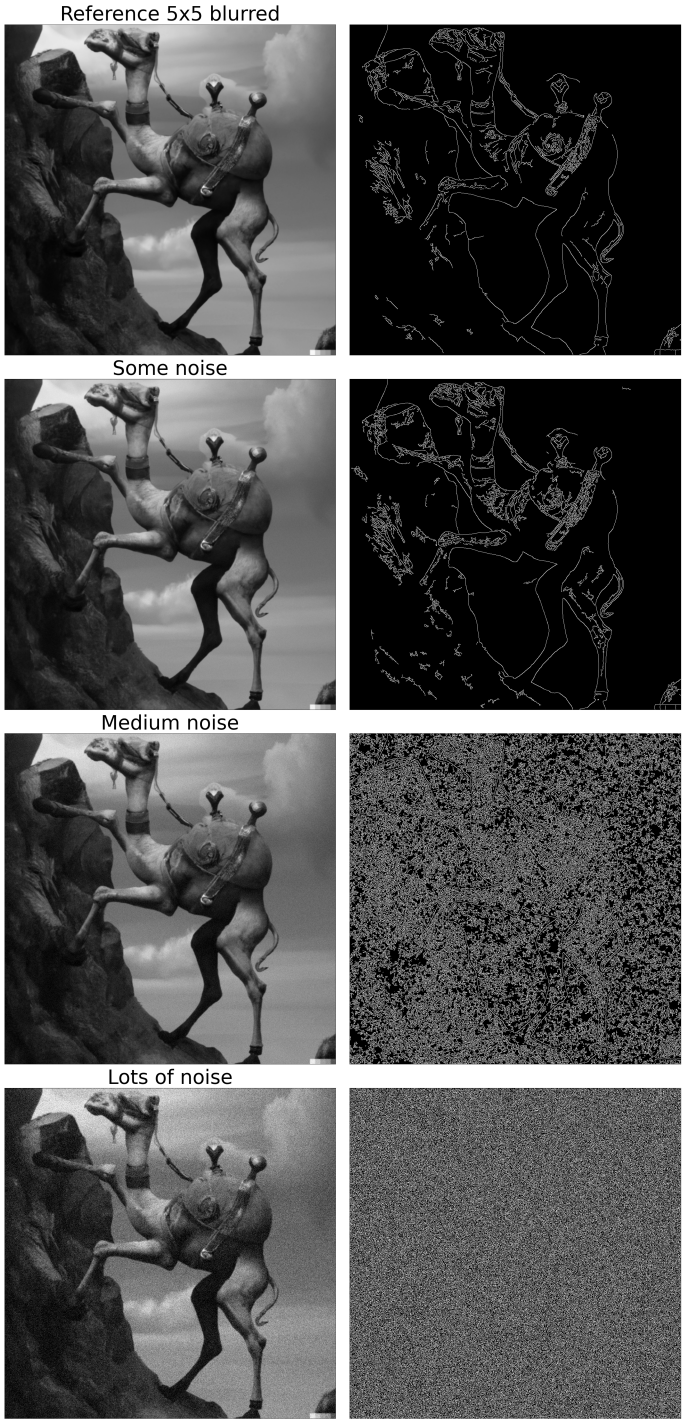

Figure 4: Edge detection at varying degrees of noise.

Image	Precision	Recall	F1-score
Reference 5x5	0.50	0.62	0.54
Some noise	0.37	0.54	0.43
Medium noise	0.05	0.52	0.10
Lots of noise	0.03	0.44	0.06

Table 3: Quantitative evaluation after applying Gaussian noise.

Observations

- The reference image *obviously* achieved the highest scores across all three metrics, as the slight blur was already proven to help in edge detection in the previous section.

- As the noise levels increased, there was a significant decline in precision, indicating more false positives in edge detection.
- Recall remained relatively high even with the added noise, suggesting that the majority of true edges were still being detected. This is a good reason for not relying on a single parameter as evaluation metric. We can remind ourselves that we would expect a recall of 0.5 for a comparison to pure white noise, so the image with *Lots of noise* is actually doing worse than blind guessing somehow.
- The F1-score, which balances precision and recall, decreased dramatically as noise intensified, indicating that edge detection is getting gradually worse. Here we can clearly see the relationship between the output images and the evaluation metrics.

Perhaps the most striking feature of this experiment is that to a human it is still quite clear where the major edges are, even for the image with the most added noise. The experiment clearly demonstrates that the addition of Gaussian noise adversely affects the performance of the edge detection algorithm. The results underscore the importance of preprocessing steps as already indicated in the previous experiment.

4.4 Effect of Different Scales

As previously mentioned, the highest resolution ground truth image was manually traced, ensuring that it accurately represents the true edges in the image. The ground truth images for lower resolutions were created by downscaling the high-resolution ground truth image. This downscaling caused some edges to thicken, increasing the likelihood of the Canny algorithm detecting them.

Figure 5: Edge detection at varying resolutions.

Image	Precision	Recall	F1-score
1024p	0.50	0.62	0.54
512p	0.91	0.04	0.09
256p	0.93	0.07	0.14
128p	0.93	0.13	0.22

Table 4: Evaluation metrics for edge detection at different image resolutions.

Observations

- As the resolution decreases, the 'thickening' of edges in the ground truth images leads to an artificially high precision. This is because even if the Canny algorithm detects parts of a thickened edge, it counts as a 'hit'.

- The Canny algorithm, by its design, detects edges as one-pixel-wide lines. It simply isn't designed to recognize thick edges as single edges, which leads to the low recall values observed.

In this experiment, we learned a valuable lesson regarding the importance of the accuracy of the Ground Truth images while evaluating the edge detection output. In our case, the scaled-down truth images were not attentively created, and the results are very unreliable as a result. It is important to understand the inner workings of the algorithm so that we don't accept poor output as anything other than the result of poor input.

4.5 Comparison with Other Edge Detection Methods

Here, we compare the Canny edge detection results with other common edge detection methods, namely the Sobel operator, Prewitt operator, and Laplacian of Gaussian. Since the other edge detection methods operate in a completely different way, it is not expected that they will perform well for the single pixel ground truth image. Therefore a new dilated version (A.6) of the ground truth image has been prepared and we will look at the performance of all methods against both the slim and chunky truth images. The thresholds for the new algorithms were manually adjusted to give as good results as we could get.

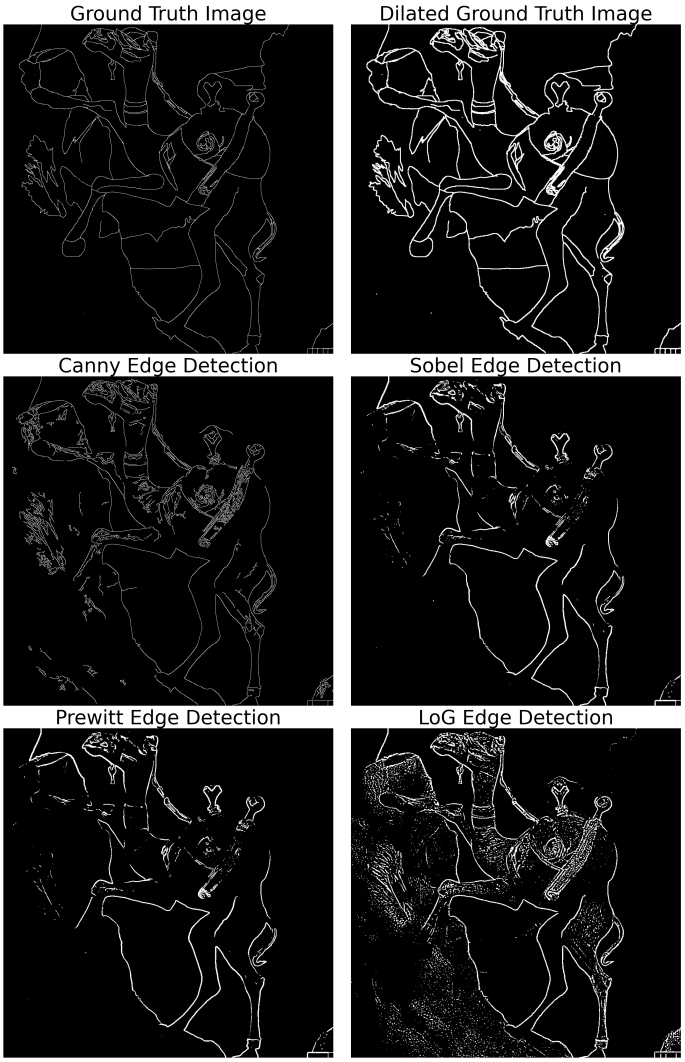

Figure 6: Edge detection between various methods.

Method	Precision	Recall	F1-score
Canny	0.50	0.62	0.54
Sobel	0.36	0.47	0.39
Prewitt	0.36	0.46	0.38
LoG	0.09	0.24	0.13

Table 5: Performance of various methods on single pixel ground truth

Method	Precision	Recall	F1-score
Canny	0.60	0.24	0.33
Sobel	0.86	0.37	0.50
Prewitt	0.86	0.37	0.50
LoG	0.36	0.31	0.32

Table 6: Performance of various methods on dilated pixel ground truth

Observations

- Visually we observe the resemblance between the single pixel ground truth image and the Canny detector output, and similar likeness between the dilated truth image and the other methods. This sets the expectations for what we will see from analysing the quantitative results.
- As expected, Canny performs well against the single pixel ground truth, where it shows the highest F1-score. However, when compared to the dilated ground truth, its recall drops because Canny fails to cover the entire width of the dilated edges.
- Both Sobel and Prewitt methods generate edges that are inherently thicker than those produced by the Canny method. This makes them more compatible with the dilated ground truth, as we can see from the improved precision and F1-score.
- The LoG method also results in broader edge responses, but is more sensitive to noise and textures, leading to a far busier edge map. This is reflected in the still lower-than-others precision.

It is clear that the fundamental differences in how these algorithms work impacts their performance in a specific task in a big way. Canny is less suitable for detecting wider edges, as its design is producing thin lines. In contrast, Sobel and Prewitt's thicker edges align better with the dilated ground truth, leading to higher precision scores. The LoG's sensitivity to broader features makes it the least precise in this context, but it still benefits somewhat from the dilation.

5 Conclusion

This comparative analysis shows that it's crucial to match the edge detection method to the characteristics of the edges in the images being processed. There isn't a one-size-fits-all solution, and what works best for one type of image or ground truth may not be suitable for another. Through the different experiments an increased understanding of the inner workings of the Canny algorithm in particular and edge detection algorithms in general was gained. The results of each individual experiment were not of the highest quality in every instance, this served to enhance the understanding even further. In a project like this, where the journey is the true goal, new light was certainly shed on the ignorance of the authors. What more could we ask for?

References

[1] C. Li, P. Xu, L. Niu, Y. Chen, L. Sheng, and M. Liu, "Tunnel crack detection using coarse-to-fine region localization and edge detection," *WIREs Data Mining and Knowledge Discovery*, vol. 9, no. 5, p. e1308, 2019.

[2] Z. Zhang, H. Fu, H. Dai, J. Shen, Y. Pang, and L. Shao, "Et-net: A generic edge-attention guidance network for medical image segmentation," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019* (D. Shen, T. Liu, T. M. Peters, L. H. Staib, C. Essert, S. Zhou, P.-T. Yap, and A. Khan, eds.), Springer International Publishing, 2019.

[3] S. Birchfield, *Image processing and Analysis*. Cengage Learning, 2018.

[4] O. dictionary, "ground truth."

A Appendix A

A.1 Visualization of Intermediate Results

```
#%%
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('./images/DALLE-camel.png', cv2.IMREAD_GRAYSCALE
    )

# Compute gradient magnitude and phase
gradient_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
gradient_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
G_mag = np.sqrt(gradient_x ** 2 + gradient_y ** 2)
G_phase = np.arctan2(gradient_y, gradient_x)

# Non-maximum suppression
def nonMaxSuppression(G_mag, G_phase):
    G_localmax = np.zeros((G_mag.shape))
    X, Y = G_mag.shape

    # For each pixel, adjust the phase to ensure that -pi/8 <=
        theta < 7*pi/8
    for x in range(X-2):
        x=x+1
        for y in range(Y-2):
            y=y+1
            theta = G_phase[x,y]
            if theta >= np.pi*0.875:
                theta = theta - np.pi
            if theta < -np.pi*0.125:
                theta = theta + np.pi
            if -np.pi*0.125 <= theta < np.pi*0.125:
                n1 = G_mag[x-1, y]
                n2 = G_mag[x+1, y]
            if np.pi*0.125 <= theta < np.pi*0.375:
                n1 = G_mag[x-1, y-1]
                n2 = G_mag[x+1, y+1]
            if np.pi*0.375 <= theta < np.pi*0.625:
                n1 = G_mag[x, y-1]
                n2 = G_mag[x, y+1]
            if np.pi*0.625 <= theta < np.pi*0.875:
                n1 = G_mag[x, y-1]
                n2 = G_mag[x, y+1]
            if (G_mag[x,y] >= n1) & (G_mag[x,y] >= n2):
                G_localmax[x,y] = G_mag[x,y]

    return G_localmax
G_localmax = nonMaxSuppression(G_mag, G_phase)
```

```

# Edge tracking by hysteresis
I_edges = cv2.Canny(image, threshold1=30, threshold2=100)

# Create a figure with subplots to display the intermediate results
plt.figure(figsize=(30, 30))
fontsize = 40

plt.subplot(221)
plt.imshow(G_mag, cmap='gray')
plt.title('Magnitude image', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(222)
plt.imshow(G_phase, cmap='gray')
plt.title('Phase image', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(223)
plt.imshow(G_localmax, cmap='gray')
plt.title('After non maximum suppression', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(224)
plt.imshow(I_edges, cmap='gray')
plt.title('Threshold image', fontsize=fontsize)
plt.xticks([]), plt.yticks([])
plt.tight_layout()

plt.show()

#%%
```

A.2 Effect of Smoothing

```
#%%
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import precision_score, recall_score, f1_score

# Load the image
image = cv2.imread('./images/DALLE-camel.png', cv2.IMREAD_GRAYSCALE
    )

# Apply degrees of Gaussian smoothing
blurred_image_3 = cv2.GaussianBlur(image, (3, 3), 0)
blurred_image_5 = cv2.GaussianBlur(image, (5, 5), 0)
blurred_image_7 = cv2.GaussianBlur(image, (7, 7), 0)

# Edge detection
t1 = 30
t2 = 100
I_edges = cv2.Canny(image, threshold1=t1, threshold2=t2)
Ib3_edges = cv2.Canny(blurred_image_3, threshold1=t1, threshold2=t2
    )
Ib5_edges = cv2.Canny(blurred_image_5, threshold1=t1, threshold2=t2
    )
```

```
Ib7_edges = cv2.Canny(blurred_image_7, threshold1=t1, threshold2=t2)

# Load ground_truth_image
ground_truth_image = cv2.imread('./images/DALLE-camel-truth.png', cv2.IMREAD_GRAYSCALE)
ground_truth_binary = (ground_truth_image > 0).astype(np.uint8) * 255

# Create a DataFrame to store the evaluation metrics
data = {
    "Image": ["Original", "Kernel size 3x3", "Kernel size 5x5", "Kernel size 7x7"],
    "Precision": [],
    "Recall": [],
    "F1-score": []}


# Calculate evaluation metrics for each result and add to the DataFrame
average = 'weighted'
for edges, label in zip([I_edges, Ib3_edges, Ib5_edges, Ib7_edges], data["Image"]):
    edges_binary = (edges > 0).astype(np.uint8) * 255
    precision = precision_score(ground_truth_binary, edges_binary, average=average)
    recall = recall_score(ground_truth_binary, edges_binary, average=average)
    f1 = f1_score(ground_truth_binary, edges_binary, average=average)

    data["Precision"].append(precision)
    data["Recall"].append(recall)
    data["F1-score"].append(f1)

# Create a pandas DataFrame
df = pd.DataFrame(data)

print(df)

# Create a figure with subplots to display the intermediate results
plt.figure(figsize=(30, 30))
fontsize = 40

plt.subplot(221)
plt.imshow(I_edges, cmap='gray')
plt.title('Original image', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(222)
plt.imshow(Ib3_edges, cmap='gray')
plt.title('Kernel size 3x3', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(223)
plt.imshow(Ib5_edges, cmap='gray')
plt.title('Kernel size 5x5', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(224)
plt.imshow(Ib7_edges, cmap='gray')
```

```
plt.title('Kernel size 7x7', fontsize=fontsize)
plt.xticks([]), plt.yticks([])
plt.tight_layout()

plt.show()
#%%
```

A.3 Effect of Noise

```
#%%
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import precision_score, recall_score, f1_score
from skimage.util import random_noise


# Load the image
image = cv2.imread('./images/DALLE-camel.png', cv2.IMREAD_GRAYSCALE
    )

# Apply Gaussian smoothing
blurred_image_5 = cv2.GaussianBlur(image, (5, 5), 0)

# Apply degrees of Gaussian white noise
def add_gauss_noise(image, mode='gaussian',mean=0, var=0.1, amount
    =0.05):
     if mode == 'gaussian':
         noisy_image = random_noise(image,  mode=mode, mean=mean,
             var=var)
     if mode == 's&p':
         noisy_image = random_noise(image,  mode=mode, amount =
             amount)
     noisy_image = np.clip(noisy_image, 0, 255) * 255
     return noisy_image.astype(np.uint8)

noisy_image_1 = add_gauss_noise(blurred_image_5, mode='gaussian',
    mean=0, var=0.0001, amount=0.05)
noisy_image_2 = add_gauss_noise(blurred_image_5, mode='gaussian',
    mean=0, var=0.001, amount=0.05)
noisy_image_3 = add_gauss_noise(blurred_image_5, mode='gaussian',
    mean=0, var=0.01, amount=0.05)

# Edge detection
t1 = 30
t2 = 100
Im_1 = cv2.Canny(blurred_image_5, threshold1=t1, threshold2=t2)
Im_2 = cv2.Canny(noisy_image_1, threshold1=t1, threshold2=t2)
Im_3 = cv2.Canny(noisy_image_2, threshold1=t1, threshold2=t2)
Im_4 = cv2.Canny(noisy_image_3, threshold1=t1, threshold2=t2)

# Load ground_truth_image
ground_truth_image = cv2.imread('./images/DALLE-camel-truth.png',
    cv2.IMREAD_GRAYSCALE)
ground_truth_binary = (ground_truth_image > 0).astype(np.uint8) *
    255

# Create a DataFrame to store the evaluation metrics
data = {
```

```
    "Image": ["Recerence 5x5", "Some noise", "Medium noise", "Lots of noise"],
    "Precision": [],
    "Recall": [],
    "F1-score": []}


# Calculate evaluation metrics for each result and add to the DataFrame
average = 'weighted'
for edges, label in zip([Im_1, Im_2, Im_3, Im_4], data["Image"]):
    edges_binary = (edges > 0).astype(np.uint8) * 255
    precision = precision_score(ground_truth_binary, edges_binary, average=average)
    recall = recall_score(ground_truth_binary, edges_binary, average=average)
    f1 = f1_score(ground_truth_binary, edges_binary, average=average)

    data["Precision"].append(precision)
    data["Recall"].append(recall)
    data["F1-score"].append(f1)

# Create a pandas DataFrame
df = pd.DataFrame(data)

print(df)

plt.figure(figsize=(30, 60))
fontsize = 40

plt.subplot(421)
plt.imshow(blurred_image_5, cmap='gray')
plt.title('Reference 5x5 blurred', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(422)
plt.imshow(Im_1, cmap='gray')
plt.xticks([]), plt.yticks([])

plt.subplot(423)
plt.imshow(noisy_image_1, cmap='gray')
plt.title('Some noise', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(424)
plt.imshow(Im_2, cmap='gray')
plt.xticks([]), plt.yticks([])
plt.tight_layout()

plt.subplot(425)
plt.imshow(noisy_image_2, cmap='gray')
plt.title('Medium noise', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(426)
plt.imshow(Im_3, cmap='gray')
plt.xticks([]), plt.yticks([])

plt.subplot(427)
plt.imshow(noisy_image_3, cmap='gray')
```

```
plt.title('Lots␣of␣noise', fontsize=fontsize)
plt.xticks([]), plt.yticks([])

plt.subplot(428)
plt.imshow(Im_4, cmap='gray')
plt.xticks([]), plt.yticks([])
plt.tight_layout()

plt.show()
#%%
```

A.4 Effect of Different Scales

```
#%%
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import precision_score, recall_score, f1_score

# Load the images
image_full = cv2.imread('./images/DALLE-camel.png', cv2.
    IMREAD_GRAYSCALE)
image_half = cv2.imread('./images/DALLE-camel-half.png', cv2.
    IMREAD_GRAYSCALE)
image_quart = cv2.imread('./images/DALLE-camel-quart.png', cv2.
    IMREAD_GRAYSCALE)
image_eighth = cv2.imread('./images/DALLE-camel-eighth.png', cv2.
    IMREAD_GRAYSCALE)

# Apply Gaussian smoothing
blurred_full = cv2.GaussianBlur(image_full, (5, 5), 0)
blurred_half = cv2.GaussianBlur(image_half, (5, 5), 0)
blurred_quart = cv2.GaussianBlur(image_quart, (5, 5), 0)
blurred_eighth = cv2.GaussianBlur(image_eighth, (5, 5), 0)

# Edge detection
t1 = 30
t2 = 100
Im_1 = cv2.Canny(blurred_full, threshold1=t1, threshold2=t2)
Im_2 = cv2.Canny(blurred_half, threshold1=t1, threshold2=t2)
Im_3 = cv2.Canny(blurred_quart, threshold1=t1, threshold2=t2)
Im_4 = cv2.Canny(blurred_eighth, threshold1=t1, threshold2=t2)

# Load ground_truth_images
Im_1_gt = cv2.imread('./images/DALLE-camel-truth.png', cv2.
    IMREAD_GRAYSCALE)
Im_2_gt = cv2.imread('./images/DALLE-camel-truth-half.png', cv2.
    IMREAD_GRAYSCALE)
Im_3_gt = cv2.imread('./images/DALLE-camel-truth-quart.png', cv2.
    IMREAD_GRAYSCALE)
Im_4_gt = cv2.imread('./images/DALLE-camel-truth-eighth.png', cv2.
    IMREAD_GRAYSCALE)

# Create a DataFrame to store the evaluation metrics
data = {
    "Image": ["1024p", "512p", "256p", "128p"],
    "Precision": [],
    "Recall": [],
    "F1-score": []}
```

```
40
41
42  # Calculate evaluation metrics for each result and add to the
        DataFrame
43  average = 'weighted'
44  for edges, truth, label in zip([Im_1, Im_2, Im_3, Im_4], [Im_1_gt,
        Im_2_gt, Im_3_gt, Im_4_gt], data["Image"]):
45      edges_binary = (edges > 0).astype(np.uint8) * 255
46      truth_binary = (truth > 0).astype(np.uint8) * 255
47      precision = precision_score(truth_binary, edges_binary, average
            =average)
48      recall = recall_score(truth_binary, edges_binary, average=
            average)
49      f1 = f1_score(truth_binary, edges_binary, average=average)
50
51      data["Precision"].append(precision)
52      data["Recall"].append(recall)
53      data["F1-score"].append(f1)
54
55  # Create a pandas DataFrame
56  df = pd.DataFrame(data)
57
58  print(df)
59
60  plt.figure(figsize=(30, 60))
61  fontsize = 60
62
63  plt.subplot(421)
64  plt.imshow(Im_1_gt, cmap='gray')
65  plt.title('Reference␣1024p␣image', fontsize=fontsize)
66  plt.xticks([]), plt.yticks([])
67
68  plt.subplot(422)
69  plt.imshow(Im_1, cmap='gray')
70  plt.xticks([]), plt.yticks([])
71
72  plt.subplot(423)
73  plt.imshow(Im_2_gt, cmap='gray')
74  plt.title('512p␣image', fontsize=fontsize)
75  plt.xticks([]), plt.yticks([])
76
77  plt.subplot(424)
78  plt.imshow(Im_2, cmap='gray')
79  plt.xticks([]), plt.yticks([])
80  plt.tight_layout()
81
82  plt.subplot(425)
83  plt.imshow(Im_3_gt, cmap='gray')
84  plt.title('256p␣image', fontsize=fontsize)
85  plt.xticks([]), plt.yticks([])
86
87  plt.subplot(426)
88  plt.imshow(Im_3, cmap='gray')
89  plt.xticks([]), plt.yticks([])
90
91  plt.subplot(427)
92  plt.imshow(Im_4_gt, cmap='gray')
93  plt.title('128p␣image', fontsize=fontsize)
94  plt.xticks([]), plt.yticks([])
95
96  plt.subplot(428)
```

```
plt.imshow(Im_4, cmap='gray')
plt.xticks([]), plt.yticks([])
plt.tight_layout()

plt.show()

#%%
```

A.5 Comparison with Other Edge Detection Methods

```
#%%
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from skimage import filters
from sklearn.metrics import precision_score, recall_score, f1_score

# Load the original image
image = cv2.imread('./images/DALLE-camel.png', cv2.IMREAD_GRAYSCALE
    )

# Apply Gaussian smoothing
blurred = cv2.GaussianBlur(image, (5, 5), 0)

# Edge detection using different methods
edges_canny = cv2.Canny(blurred, threshold1=30, threshold2=100)
edges_sobel = filters.sobel(blurred)
edges_prewitt = filters.prewitt(blurred)
edges_log = cv2.Laplacian(blurred, cv2.CV_64F, ksize=5)  #
    Laplacian of Gaussian

# Normalize the LoG edges
log_1 = edges_log - np.min(edges_log)
log_norm =  (log_1 / np.max(log_1))

# Threshold edges from Sobel, Prewitt, and Laplacian of Gaussian
thresh = 0.07
edges_sobel_binary = (edges_sobel > thresh).astype(np.uint8) * 255
edges_prewitt_binary = (edges_prewitt > thresh).astype(np.uint8) *
    255
thresh = 0.61
edges_log_binary = (log_norm > thresh).astype(np.uint8) * 255

# Load ground truth image
ground_truth = cv2.imread('./images/DALLE-camel-truth.png', cv2.
    IMREAD_GRAYSCALE)
ground_truth_dilated = cv2.imread('./images/DALLE-camel-truth-
    dilated.png', cv2.IMREAD_GRAYSCALE)
ground_truth_binary = (ground_truth > 0).astype(np.uint8) * 255

# Create a DataFrame to store the evaluation metrics
data = {
    "Method": ["Canny", "Sobel", "Prewitt", "LoG"],
    "Precision": [],
    "Recall": [],
    "F1-score": []
}

```

```
# Calculate evaluation metrics for each method and add to the DataFrame
average = 'weighted'
for edges_binary, method in zip([edges_canny, edges_sobel_binary, edges_prewitt_binary, edges_log_binary], data["Method"]):
    precision = precision_score(ground_truth_binary, edges_binary, average=average)
    recall = recall_score(ground_truth_binary, edges_binary, average=average)
    f1 = f1_score(ground_truth_binary, edges_binary, average=average)

    data["Precision"].append(precision)
    data["Recall"].append(recall)
    data["F1-score"].append(f1)

# Create a pandas DataFrame
df = pd.DataFrame(data)

print(df)

plt.figure(figsize=(30, 45))
fontsize = 40

plt.subplot(321)
plt.imshow(ground_truth, cmap='gray')
plt.title('Ground Truth Image', fontsize=fontsize)
plt.axis('off')

plt.subplot(322)
plt.imshow(ground_truth_dilated, cmap='gray')
plt.title('Dilated Ground Truth Image', fontsize=fontsize)
plt.axis('off')

plt.subplot(323)
plt.imshow(edges_canny, cmap='gray')
plt.title('Canny Edge Detection', fontsize=fontsize)
plt.axis('off')

plt.subplot(324)
plt.imshow(edges_sobel_binary, cmap='gray')
plt.title('Sobel Edge Detection', fontsize=fontsize)
plt.axis('off')

plt.subplot(325)
plt.imshow(edges_prewitt_binary, cmap='gray')
plt.title('Prewitt Edge Detection', fontsize=fontsize)
plt.axis('off')

plt.subplot(326)
plt.imshow(edges_log_binary, cmap='gray')
plt.title('Laplacian of Gaussian Edge Detection', fontsize=fontsize)
plt.axis('off')

plt.tight_layout()
plt.show()


#%%
```

A.6 Dilate

```
#%%
import cv2
import numpy as np

# Read the ground truth image
ground_truth = cv2.imread('./images/DALLE-camel-truth.png', cv2.
    IMREAD_GRAYSCALE)

# Convert ground truth to binary
_, ground_truth_binary = cv2.threshold(ground_truth, 127, 255, cv2.
    THRESH_BINARY)

# Define a 3x3 kernel
kernel = np.ones((3, 3), np.uint8)

# Dilate
dilated_ground_truth = cv2.dilate(ground_truth_binary, kernel,
    iterations=1)

# Save
cv2.imwrite('./images/DALLE-camel-truth-dilated.png',
    dilated_ground_truth)
#%%
```