UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS791 - Image Processing

Assignment 1 - Edge Detection

Reuben Jooste
Student number: u21457060
Email: u21457060@tuks.co.za

1 Background Information

In this assignment we had to implement a program capable of optimizing an edge detection pipeline (i.e. an optimizer) such that we can use it for detecting edges within an image. We were given 11 images where each image has a different contrast to the next image. *Figure 1* below shows an example of one of the images in the dataset provided. An edge detection pipeline usually involves the following steps:

- **Step 1 - Noise reduction**

  During noise reduction, the goal is to reduce the noise within the image such that it does not impact the quality of our edge detection. If there are too much noise then our pipeline might detect edges that are not really seen as edges. However, reducing too much noise might lead to the loss of important details within the image. Our GA will choose from three filters: *Gaussian, Median, and Bilateral*

- **Step 2 - Gradient Calculation**

  This step is important as we want to calculate how fast the pixel intensity values in both the horizontal (X-axis) and vertical (Y-axis) directions. Again, our GA will choose from three methods: *Sobel, Prewitt, and Laplacian*

- **Step 3 - Edge Thinning**

  During this step we will apply the *Non-maximum Suppression* method such that we refine the edges detected in the previous step to a single pixel width.

- **Step 4 - Thresholding**

  This step is important as we want to separate the detected edges from the existing noise within our image. This allows us to sort of segment the image into a foreground and a background. Our GA will choose from the following three methods: *Hysteresis and Binary.* It should be noted that the Hysteresis method is often used an edge linking method.

- **Step 5 - Linking the edges**

  Finally, we apply edge linking methods such ensure that the detected edges are all linked together to create a more accurate representation of the detected edges. In simple terms, it helps to connect the edge segments that are discontinuous. Our GA will choose from the following list of methods: *Hough Transform, Gradient direction analysis, and Region Growing.*

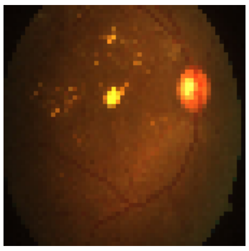


Figure 1: Img1 from the dataset

2 Pipeline Configuration and Experimental Setup

2.1 Dataset Transformation

The input image were all coloured (i.e. having three dimensions). Due to this being an edge detection task, we converted all the images to a gray-scale format, reducing the dimensions to only 1. These gray images were then passed into the Canny Edge Detector to generate the target/reference output images. Now that we have gray scale images as well as reference images, we move on to explaining our edge detection approach in more detail.

2.2 Population Generation and Representation

As mentioned in the previous section, we will be implementing a Genetic Algorithm as the optimizer for building an optimized edge detection pipeline. Our GA will start by randomly generating a predefined number of pipelines (chromosomes). Each chromosome will be represented using a similar structure as the following example representation:

```
chromosome = {
    "noise_reduction": {
        "method": "gaussian",
        "params": {
            "ksize": (5, 5),
            "sigma": 1.4
        },
    },
    "gradient": {
        "method": "sobel",
        "params": {
            "ksize": 5
        },
    },
    "thinning": True,
    "thresholding": {
        "method": "binary",
        "params": {
            "low_thresh": 15,
        },
    },
    "linking": {
        "method": "region_growing",
        "params": {
            "threshold": 10
        }
    }
}
```

2.3 Fitness Function and Population Evaluation

Once the GA has generated a number of chromosome i.e. the population, it will then evaluate each individual in the population. The evaluation process takes place at the start of each generation. To evaluate an individual the algorithm takes the individual (pipeline) and applies it to the image such that it can detect edges. The algorithm then returns the detected edges and compares it with the target image. We were required to use the following metrics during the comparison:

- **Mean Squared Error:** Measures the average squared distance between the observed pixel values and the target pixel values.
- **SIFT feature descriptor:** Matches the extracted features from both input images (generated image and reference image) to identify the K nearest neighbours for each feature.

- **Peak-Signal-Noise-Ratio (PSNR):** Measures the ratio between the power of noise (that which affects the quality of the image's representation) and the power of a signal (maximum possible value).
- **Structural Similarity:** Rather than solely relaying on pixel-value differences, this metric compares the similarity in terms of the changes in structural information between two images.

From these four metrics, we consider MSE as the main evaluation metric due to it being a simple metric to compute as well as allowing us to directly evaluate the difference between the two images in terms of their pixel values. Our **fitness function** is defined as the following:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

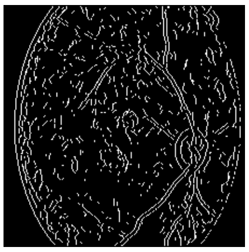
where n represents the number of observations, y_i represents the true pixel value, and $\hat{y}_i$ represents the observed pixel value (i.e. the value from the generated image). Based on our fitness function we then arrive at our ***objective function***: we want to minimize the MSE such that the generated image is closely related to the target image.

2.4 Why we chose our objective function

We feel that the chosen objective function is a suitable choice due to the following reasons:

- Pixel-wise accuracy is critical thus making it suitable for our image generation task.
- MSE is simple and straightforward to calculate making it easy and efficient to compare the differences in pixel intensity values.
- MSE is also computationally efficient.
- The minimization of MSE encourages the preservation of the finer details (e.g. tiny edges) within the generated image.
- Minimizing the MSE can lead to increased scores for other metrics such as the PSNR.

Figure 28 shows a target image generated using the Canny edge detector as well as an image which was generated when we applied a particular pipeline configuration to it.


(a) Pipeline Generated Image

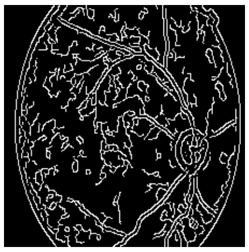

(b) Canny Edge Detector

Figure 2: Pipeline Image vs Canny Image

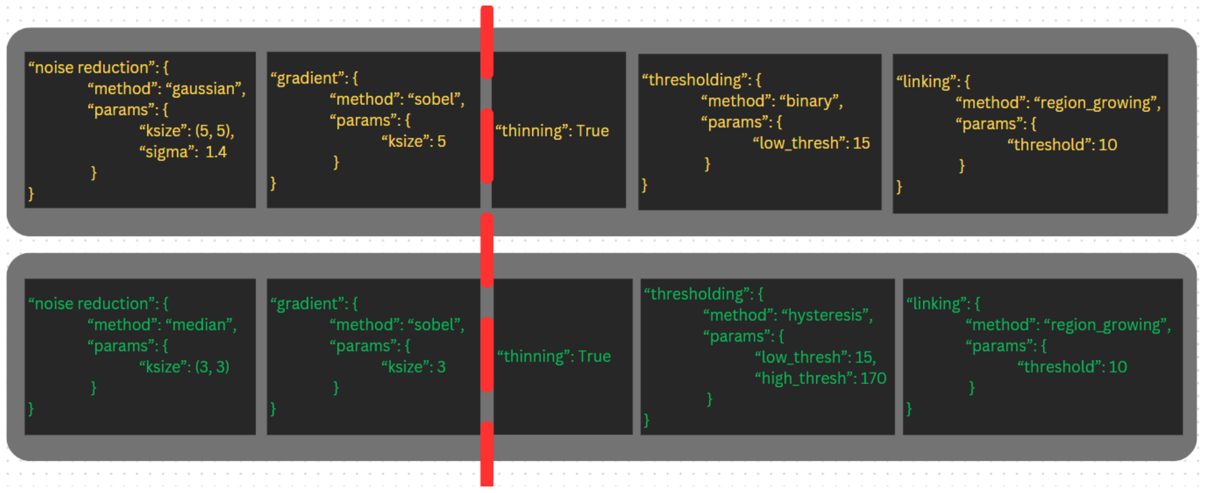
2.5 Selection

During the evolutionary process i.e. creating a new population by altering the existing population, we use the well-known **Tournament selection** method. Based on the predefined tournament size, we select a certain number of individuals from the population. From this tournament pool we select the fittest individual to act as the parent. We then apply the tournament selection twice such that we have two parent chromosomes to work with when needing to generate two new offspring.

2.6 Genetic Operators

In our GA we employ the following three genetic operators:

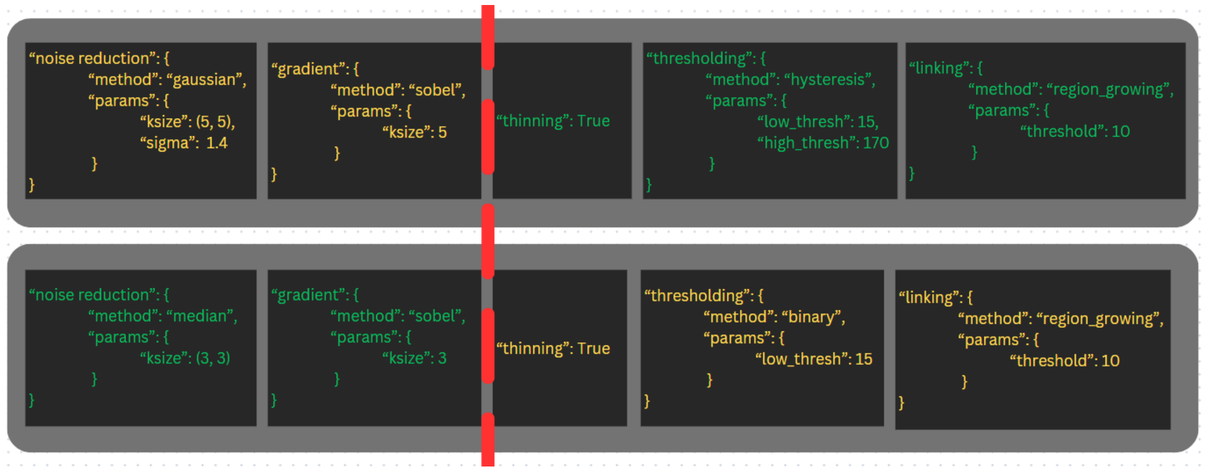
- **Crossover:** We employ *single-point crossover* which means that we select a random component between ”noise reduction” and ”Edge linking” and swap the two parts of the two parents to create two new offspring. This operator allows the GA to explore new regions of the solution space and helps maintain diversity in the population. Figure 3 and Figure 4 illustrates an example of this operation.



```
"noise reduction": {
        "method": "gaussian",
        "params": {
                "ksize": (5, 5),
                "sigma": 1.4
        }
}
"gradient": {
        "method": "sobel",
        "params": {
                "ksize": 5
        }
}
"thinning": True
"thresholding": {
        "method": "binary",
        "params": {
                "low_thresh": 15
        }
}
"linking": {
        "method": "region_growing",
        "params": {
                "threshold": 10
        }
}
```

```
"noise reduction": {
        "method": "median",
        "params": {
                "ksize": (3, 3)
        }
}
"gradient": {
        "method": "sobel",
        "params": {
                "ksize": 3
        }
}
"thinning": True
"thresholding": {
        "method": "hysteresis",
        "params": {
                "low_thresh": 15,
                "high_thresh": 170
        }
}
"linking": {
        "method": "region_growing",
        "params": {
                "threshold": 10
        }
}
```

Figure 3: Before Crossover



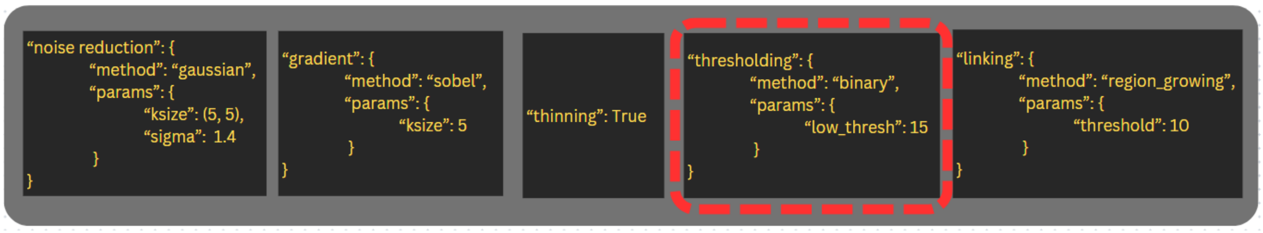
```
"noise reduction": {
        "method": "gaussian",
        "params": {
                "ksize": (5, 5),
                "sigma": 1.4
        }
}
"gradient": {
        "method": "sobel",
        "params": {
                "ksize": 5
        }
}
"thinning": True
"thresholding": {
        "method": "hysteresis",
        "params": {
                "low_thresh": 15,
                "high_thresh": 170
        }
}
"linking": {
        "method": "region_growing",
        "params": {
                "threshold": 10
        }
}
```

```
"noise reduction": {
        "method": "median",
        "params": {
                "ksize": (3, 3)
        }
}
"gradient": {
        "method": "sobel",
        "params": {
                "ksize": 3
        }
}
"thinning": True
"thresholding": {
        "method": "binary",
        "params": {
                "low_thresh": 15
        }
}
"linking": {
        "method": "region_growing",
        "params": {
                "threshold": 10
        }
}
```

Figure 4: After Crossover

- **Reproduction:** We simply make a copy of the current best individual and add it into the new population. By doing this we ensure a good quality solution is preserved which then helps to guide the search towards more promising regions of the solution space.

- **Mutation:** Finally, we also employ mutation on a selected individual's components. The operator randomly selects a component (mutation gene) and then also randomly decides to mutate the method (this also changes the parameters as different methods have different parameters) or to mutate just the parameters. By introducing small changes to the population, we ensure that diversity is maintained as well as prevent the algorithm from getting stuck at local optima. Figure 5 and Figure 6 shows an example of mutating a component's method.



```
"noise reduction": {
        "method": "gaussian",
        "params": {
                "ksize": (5, 5),
                "sigma": 1.4
        }
}
```

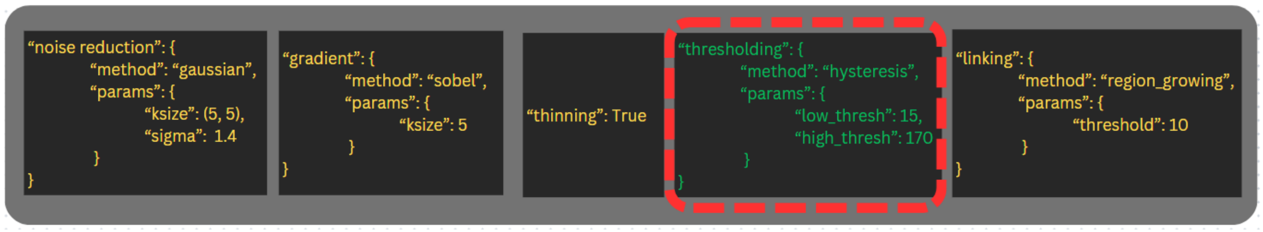
```
"gradient": {
        "method": "sobel",
        "params": {
                "ksize": 5
        }
}
```

```
"thinning": True
```

```
"thresholding": {
        "method": "binary",
        "params": {
                "low_thresh": 15
        }
}
```

```
"linking": {
        "method": "region_growing",
        "params": {
                "threshold": 10
        }
}
```

Figure 5: Before Mutation



```
"noise reduction": {
        "method": "gaussian",
        "params": {
                "ksize": (5, 5),
                "sigma": 1.4
        }
}
```

```
"gradient": {
        "method": "sobel",
        "params": {
                "ksize": 5
        }
}
```

```
"thinning": True
```

```
"thresholding": {
        "method": "hysteresis",
        "params": {
                "low_thresh": 15,
                "high_thresh": 170
        }
}
```

```
"linking": {
        "method": "region_growing",
        "params": {
                "threshold": 10
        }
}
```

Figure 6: After Mutation

2.7 Stopping Criteria

For our GA we proposed two stopping criteria:

- **Number of generations:** This is a predefined hyperparameter indicating the number of iterations it should take for the GA to converge at an optimal solution. If the GA is allowed to continuously run then it might carry on for many generations while it might have already converged i.e. it makes very little or no increase in fitness. Therefore, we define a number of iterations for which the GA must run and we assume the algorithm would converge within this number of generations. We will experiment with the range 20 to 50 for this hyperparameter.

- **Minimum increase threshold:** We terminate the GA algorithms and return the current best found solution (pipeline) when there has not been a significant increase in the average performance after 10 iterations (generations). We use a minimum threshold of 1 meaning that the following generation must have an improvement of at least 1 in terms of fitness. If not then the GA algorithm will converge otherwise it will continue running checking the condition again after the next, upcoming generations.

2.8 Hyperparameters

The following are hyperparameters which we need to fine tune for the GA such that the it can arrive at optimal solutions:

- **Population size:** This determines the number of chromosomes used during the evolutionary process. We use larger values such that it will help increase the diversity in the search space. By doing this, the GA can find more promising solutions during the evolutionary process.

- **Number of Generations:** As mentioned earlier, we use this parameter as one of our stopping criteria. We will use relatively large numbers such that we can allow for more explorations and refinement in the search space until it converges.

- **Crossover Rate:** This determines the rate at which the crossover operator must be performed. We will experiment with values in the range 0.8 to 0.9 such that the GA can strike a balance between exploration as well as exploitation. The rate is high such that diversity within the population is maintained over the generations.
- **Mutation Rate:** This determines the rate at which the GA will make a small random change to one of the individuals in the population. We will experiment with small values (0.1 to 0.2) since we do not want to make changes too frequently as this will lead to randomness. However, by using a small change the GA is able to escape local optima if it gets stuck there. It is important to note that using too small values could lead to stagnation.
- **Tournament Size:** As mentioned, we use the tournament selection method thus we are required to use a hyperparameter which defines the size of the selected pool of individuals. We will experiment with values within the range 3 to 7 which are small sizes compared to the population sizes. This will introduce more diversity. If we use values that are too large it will promote *selection pressure* which affects the GA causing it to converge too quickly.

We now present the hyperparameters for the different components within the edge detection pipeline. These values are in a way fine tunes by the GA itself:

- **Noise Reduction:**
  - ⋆ **Method:** The type of method to apply in order to reduce the noise within the image such that real edges can be detected.
  - ⋆ **Kernel Size(ksize):** Defines the size of the matrix (window) used by the Gaussian and Median filters. By using a larger size the filter will consider more neighbouring pixels when trying to smooth the image. However, too large sizes could lead to blurring important edge details.
  - ⋆ **Sigma:** The Gaussian filter uses this to control the extent of the smoothing effect. Large sizes will lead to neighbouring pixels having more influence and thus causing a higher smoothing effect.
  - ⋆ **Diameter (d):** Used by the bilateral filter, this parameter defines the area of the neighbouring pixels used to help reduce noise while preserving edge details.
  - ⋆ **Sigma Color:** Used by the bilateral filtering, this parameter defines which pixels to consider based on their intensity differences. Larger values will lead to more aggressive noise reduction.
  - ⋆ **Sigma Space:** Used by bilateral filtering, this parameter controls the distance between pixels that are considered during noise reduction.
- **Gradient Calculation:**
  - ⋆ **Kernel Size (ksize):** Typically used by the Sobel filter. It determines the window size of for which derivatives in the image are calculated. Larger sizes helps lead to more accurate calculations.
- **Thresholding:**
  - ⋆ **Low Threshold:** Used by the Binary and Hysteresis thresholding methods. It defines the minimum requirement for pixel values. If a pixel has a lower intensity value then it is considered as a non-edge pixel.
  - ⋆ **High Threshold:** In contrast with the low threshold, this parameter defines the minumum requirement for pixels to be consider strong edge pixels.
- **Edge Linking:**
  - ⋆ **Gradient Direction Threshold:** This parameter is used by the Gradient Direction Analysis method and it determines how much influence the gradient directions have on edge linking. We will experiment with low values (7 to 25).
  - ⋆ **Hough Transform Threshold:** This parameter defines how much evidence is required to detect edges. Using a smaller value will ensure that more lines are detected and thus linked together.

- ⋆ **Hough Transform Min. Line Length:** Lines detected on the image much have a length greater than the parameter value otherwise it is ignored. Due to having rounded edges in our images, we use smaller values.
- ⋆ **Hough Transform Max Line Gap:** This parameter controls the maximum distance between two line segments to be considered part of the same line. Again, our image has tight spaces between different edges thus we will experiment with small values.
- ⋆ **Region Growing Threshold:** This parameter controls the extent of which pixels to include in the edge linking process based on their pixel intensity values. Our method takes in an image in binary (0 and 1) format. Thus our threshold value will be between 0 and 1.

Belowe we present a table with all hyperparameters and the respective experiment values:

🟩 Global Hyperparameters
🟪 Noise Reduction (NR)
🟦 Gradient Calculation (G)
🟨 Thresholding (TH)
🟧 Edge Linking (EL)

Table 1: Legend

Hyperparameter	**Experiment Values**
Population Size	[100, 125, 150, 200, 225, 250]
Generations	[30, 40, 50]
Tournament Size	[3, 4, 5, 6, 7]
Crossover Rate	[0.75, 0.8, 0.82, 0.85, 0.9]
Mutation Rate	[0.05, 0.1, 0.125, 0.15, 0.2]
(NR) Method	['gaussian', 'median', 'bilateral]
(NR) ksize	[3, 5, 7, 9, 11]
(NR) sigma	$X \sim \text{Uniform}(1, 100)$
(NR) d	[5, 9, 13, 15, 21]
(NR) sigma color	[75, 100, 125]
(NR) sigma space	[75, 100, 125]
(G) Method	['sobel', 'prewitt', 'laplacian']
(G) ksize	[3, 5]
(TH) Method	['binary', 'hysteresis']
(TH) binary threshold	$X \sim \text{Uniform}(7, 15)$
(TH) hysteresis low threshold	$X \sim \text{Uniform}(40, 60)$
(TH) hysteresis high threshold	$X \sim \text{Uniform}(200, 255)$
(EL) Method	['gradient direction analysis', 'region growing', 'hough']
(EL) gda threshold	[5, 9, 11, 15, 16, 18, 20]
(EL) Hough Transform Low Threshold	[1, 3, 5, 8, 9, 10]
(EL) Hough Transform Min. Line Length	$X \sim \text{Uniform}(0.1, 1)$
(EL) Hough Transform Max Line Gap	[1, 2, 3]
(EL) Region Growing Threshold	$X \sim \text{Uniform}(0.005, 0.015)$

Table 2: GA Hyperparameters

3 Results

In this section we present our findings. We first talk about our implementation of the Canny Edge Detector more so about the parameters used. In the following section we talk about the optimal general pipeline configuration that was trained on one image and applied to the other images. In the following subsection we talk about each individual pipeline generated on each individual image. We will also compare each of the output images with that generated by our Canny edge detector.

3.1 Canny Edge Detector

We used the well-known **Open CV** Python Libary for our Canny Edge Detector implementation. Below we show an example of how the library was used:

```
    import cv2
    edges = cv2.Canny('path/to/inputImg.png', lowT, highT, ksize, L2
       Gradient=True)
```

We note that the library takes in some arguments. We used the following values for these arguments:

- **lowT (lower threshold):** This specifies the lower threshold used by the Canny algorithm. We used a value of 100.
- **highT (high threshold):** This specifies the high threshold used by the Canny algorithm. We used the value 200.
- **ksize (Kernel Size):** This specifies the window size used by the Sobel and Gaussian filter within the Canny algorithm. Here we used a size of 5.

3.2 General Pipeline

For our general pipeline we had to select an image at random while keeping in mind factors such as noise, contrast, etc. We then selected the following image to build a general pipeline:

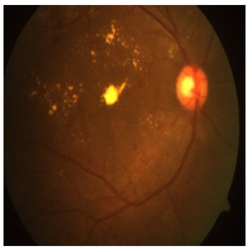


Figure 7: Img0.png

After experimenting with the available hyperparameter options, we fine tuned our GA and arrived at the following general pipeline configuration and parameters:

- **Seed value:** 8279
- **Population Size:** 125
- **Generations:** 30
- **Crossover Rate:** 0.75
- **Mutation Rate:** 0.1
- **Tournament Size:** 3

```
    chromosome = {
        "noise_reduction": {
            "method": "gaussian",
            "params": {
                "ksize": (3, 3),
                "sigma": 15.628919134150035
            }
        },
        "gradient": {
            "method": "sobel",
            "params": {
                "ksize": 3,
                "ddepth": cv2.CV_64F
            }
        },
        "thinning": True,



        "thresholding": {
            "method": "binary",
            "params": {
                "bin_thresh": 10.917010955943017
            }
        },
        "linking": {
            "method": "gradient_direction_analysis",
            "params": {
                "gda_threshold": 8
            }
        }
    }
```

Our GA used this pipeline as a general pipeline for all other images. We now present the generated edge detected images with the respective Canny generated images and then we will present the fitness scores for each image in a table.

- **Img0.png:**

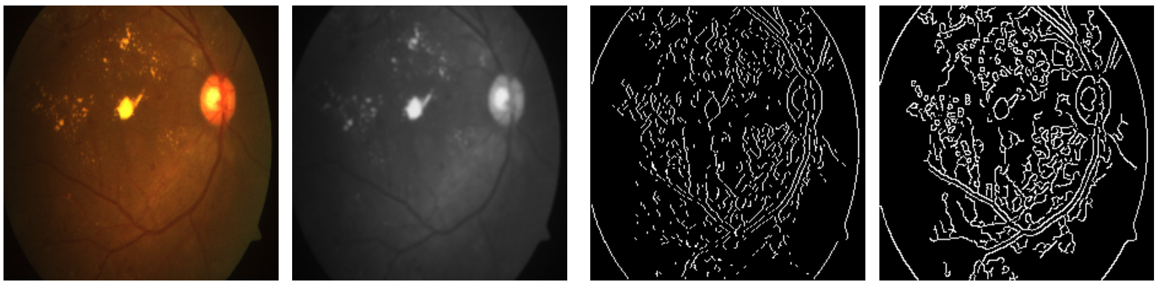
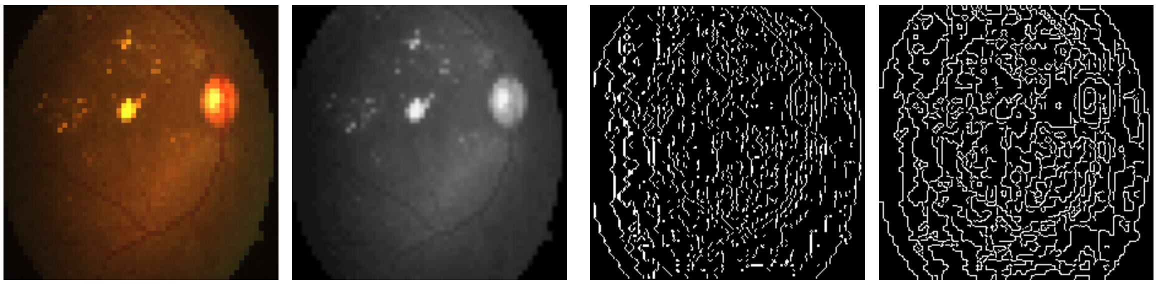

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 8: Pipeline Image vs Canny Image

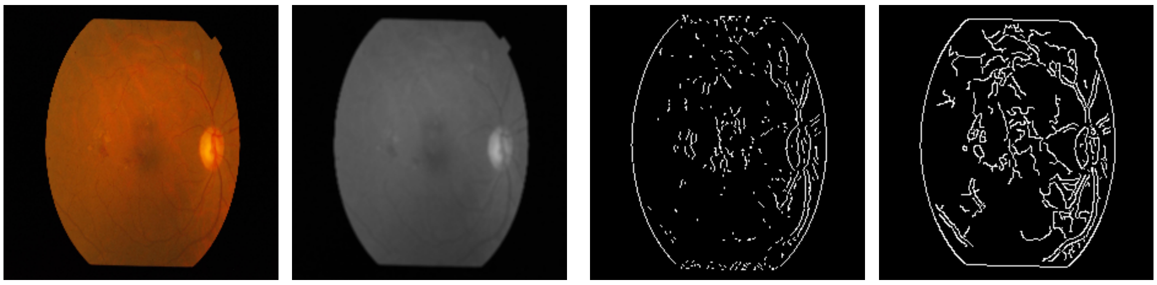
- **Img1.png - General Pipeline Applied:**



(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 9: Img1 Pipeline Image vs Canny Image

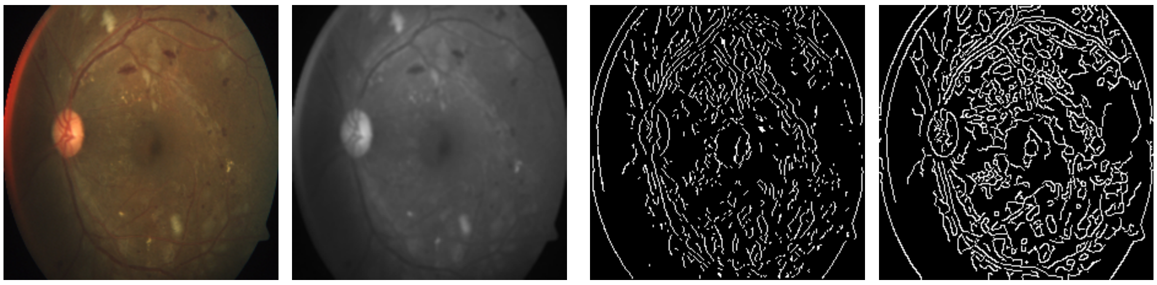
- **Img2.png - General Pipeline Applied:**



(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 10: Img2 Pipeline Image vs Canny Image

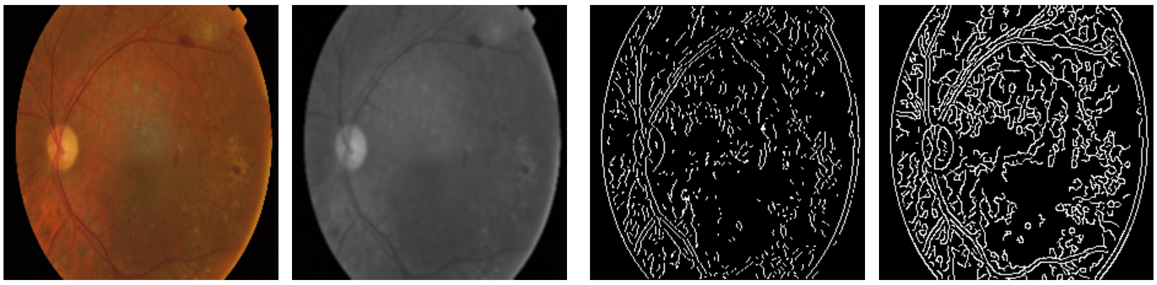
- **Img3.png - General Pipeline Applied:**



(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 11: Img3 Pipeline Image vs Canny Image

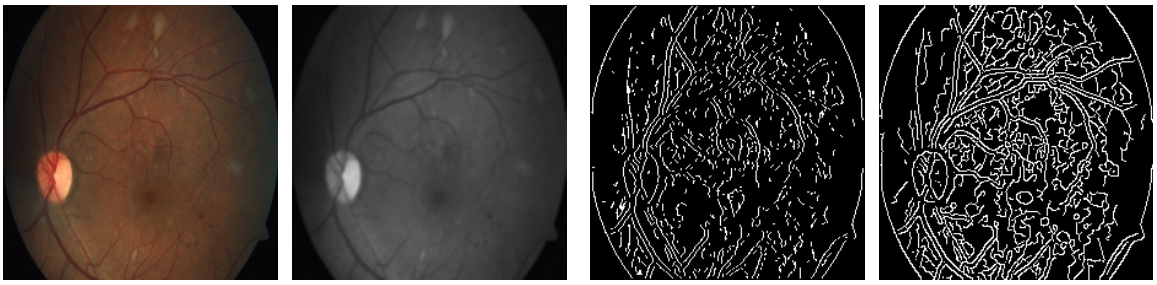
- **Img4.png - General Pipeline Applied:**



(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 12: Img4 Pipeline Image vs Canny Image

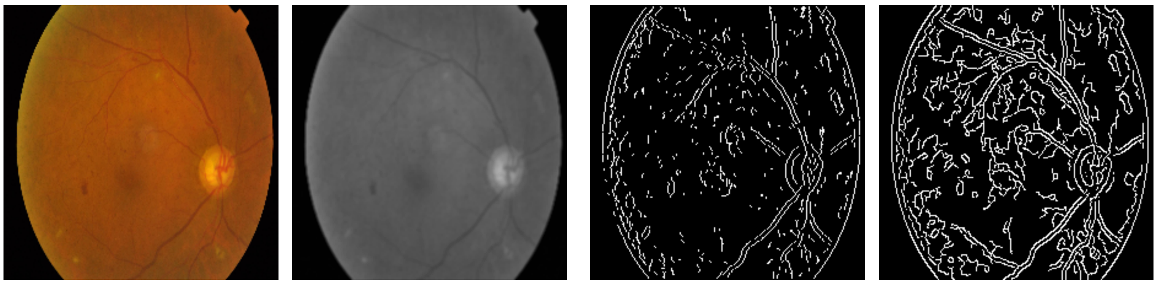
- **Img5.png - General Pipeline Applied:**



(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 13: Img5 Pipeline Image vs Canny Image

- **Img6.png - General Pipeline Applied:**



(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 14: Img6 Pipeline Image vs Canny Image

- **Img7.png - General Pipeline Applied:**

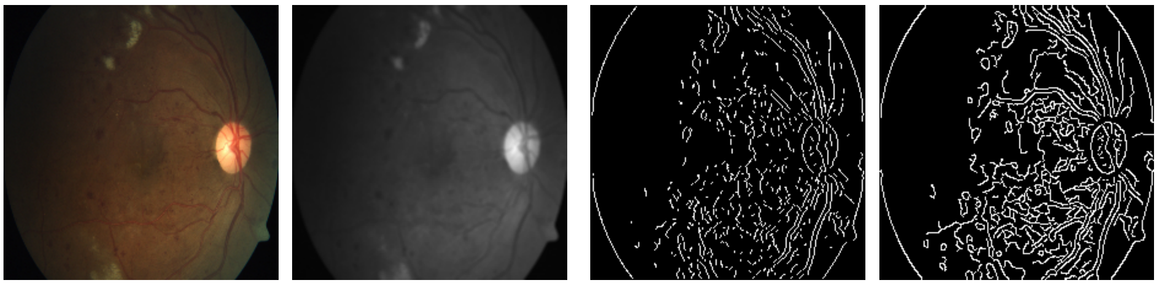

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 15: Img7 Pipeline Image vs Canny Image

- **Img8.png - General Pipeline Applied:**

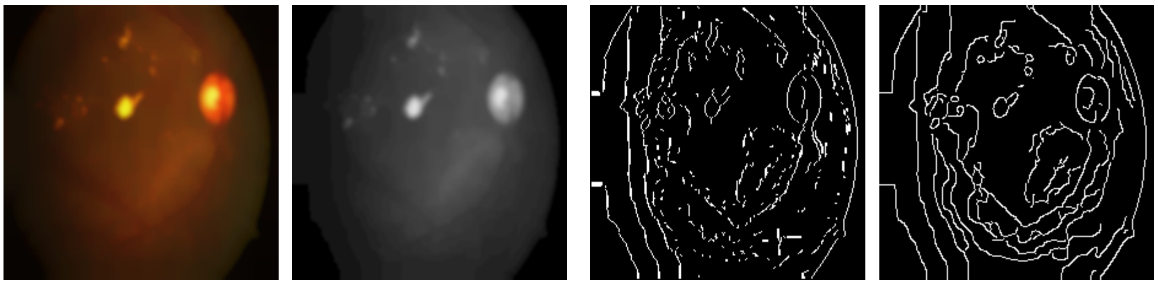

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 16: Img8 Pipeline Image vs Canny Image

- **Img9.png - General Pipeline Applied:**

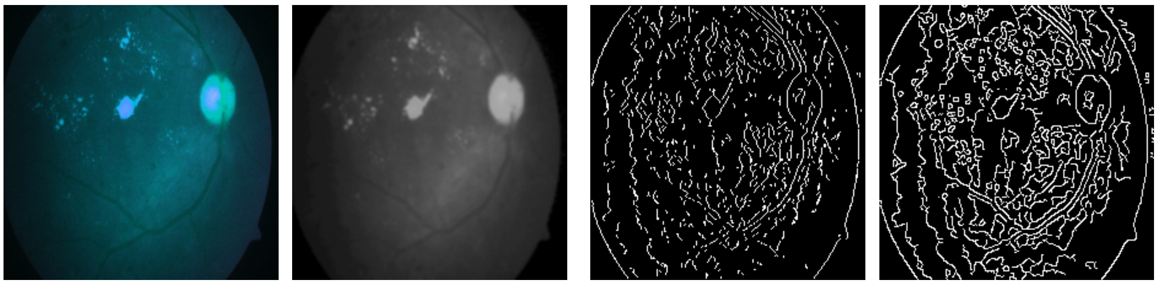

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 17: Img9 Pipeline Image vs Canny Image

- **Img10.png - General Pipeline Applied:**

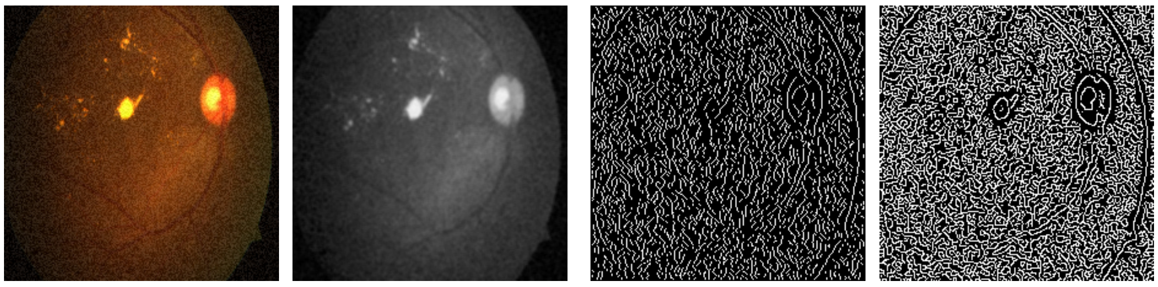

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 18: Img10 Pipeline Image vs Canny Image

We now tabulate the metric results for each of these images:

Image	MSE	Structural Sim.	PSNR	SIFT
Img0.png	**5475.3393**	**0.5758**	**10.7467**	**0.4087**
Img1.png	**8405.4558**	**0.4176**	**8.8852**	**0.4033**
Img2.png	**3347.4086**	**0.6965**	**12.8837**	**0.3884**
Img3.png	**7237.8154**	**0.4987**	**9.5347**	**0.4444**
Img4.png	**7310.3879**	**0.4853**	**9.4914**	**0.4074**
Img5.png	**7507.3706**	**0.48**	**9.3759**	**0.389**
Img6.png	**6145.3394**	**0.496**	**10.2453**	**0.4494**
Img7.png	**5417.0221**	**0.5959**	**10.7932**	**0.3787**
Img8.png	**3920.2133**	**0.6321**	**12.1977**	**0.4048**
Img9.png	**7014.914**	**0.4716**	**9.6706**	**0.4622**
Img10.png	**14734.8184**	**0.3707**	**6.4474**	**0.3219**
Averages	**6686.1665**	**0.5111**	**9.8981**	**0.4086**

Table 3: General Pipeline Application results

3.2.1 Critical Analysis based on the results:

When we compare the output images of all the generated images with that of their respective Canny's output, it seemed that our general pipeline was performing relatively well since the images seemed similar to that of the Canny's. However, now looking at the results for each image we note that the metrics scores are relatively high especially the MSE of *Img10.png*. With the MSE being relatively high across all images indicates that there is a considerable pixel-wise difference. This is due to many factors such as intensity values, misaligned detected edges, etc. This then indicates that our general pipeline produces results which diverge from the results of the Canny Edge Detector (CED).

This claim is supported by the other metrics such as Peak-Signal-Noise-Ratio (PSNR), reflecting the noisiness within the images. Despite these negative claims, we note that the general pipeline achieves roughly a 50% average Structural Similarity (SSIM) accross all the images. This highlights that even though there are substantial differences in terms of detected edges, there are still some common structural features between the two images. Furthermore, the SIFT score (used to detect key features) is roughly a 40% average across all images. This highlights that our general pipeline does find some alignment in key features but also highlights that it fails to capture critical features.

Overall, our general pipeline seems to output images which look similar to the CED's output but when diving deeper into the comparison (in terms of metrics) it is clear that our pipeline produces results which diverges in both the intensity and locations of detected edges, indicating areas where improvements can be made in terms of the accuracy, alignment, and intensity of the detected edges.

3.3 Individual Pipelines

We will now present the results for each individual image. This time a unique pipeline was generated for each image to optimize the edge detection results. We will also later on, compare these individual results with that of the general pipeline to analyse whether our individually optimized pipelines performed better or worse than the general pipeline configuration. Since the general pipeline was optimized on *Img0.png* we will not present its results again.

- **Img1.png:**
  - ⋆ **Seed value:** 1167
  - ⋆ **Population Size:** 250
  - ⋆ **Generations:** 30
  - ⋆ **Crossover Rate:** 0.85
  - ⋆ **Mutation Rate:** 0.05
  - ⋆ **Tournament Size:** 4

```
    chromosome = {
        "noise_reduction": {
            "method": "gaussian",
            "params": {
                "ksize": (3, 3),
                "sigma": 91.56240590788063
            }
        },
        "gradient": {
            "method": "sobel",
            "params": {
                "ksize": 3,
                "ddepth": cv2.CV_64F
            }
        },
        "thinning": True,
        "thresholding": {
            "method": "binary",
            "params": {
                "bin_thresh": 9.057617799918967
            }
        },
        "linking": {
            "method": "gradient_direction_analysis",
            "params": {
                "gda_threshold": 10
            }
        }
    }
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

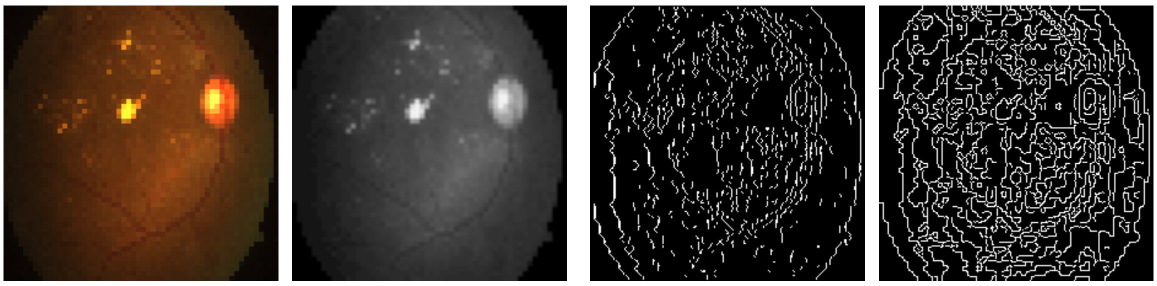

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 19: Img1 Pipeline Image vs Canny Image

We note that out generated image detected a lot less edges (which could be noise). This might be due to the pipeline smoothing the input image too much i.e. blurring the image too much leading to important details being lost. This is supported by the relatively high Sigma value for our Gaussian filter method which is approximately **91.5624**.

- **Img2.png:**
  - ⋆ **Seed value:** 3305
  - ⋆ **Population Size:** 225
  - ⋆ **Generations:** 30
  - ⋆ **Crossover Rate:** 0.8
  - ⋆ Mutation Rate: 0.125
  - ⋆ **Tournament Size:** 7

```
    chromosome = {
        "noise_reduction": {
            "method": "median",
            "params": {
                "median_ksize": 9
            }
        },
        "gradient": {
            "method": "laplacian",
            "params": {
                "ksize": 5,
                "ddepth": cv2.CV_64F
            }
        },
        "thinning": True,
        "thresholding": {
            "method": "hysteresis",
            "params": {
                "hyst_low": 51.10531144447759,
                "hyst_high": 221.7624077943912
            }
        },
        "linking": {
            "method": "gradient_direction_analysis",
            "params": {
                "gda_threshold": 6
            }
        }}
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

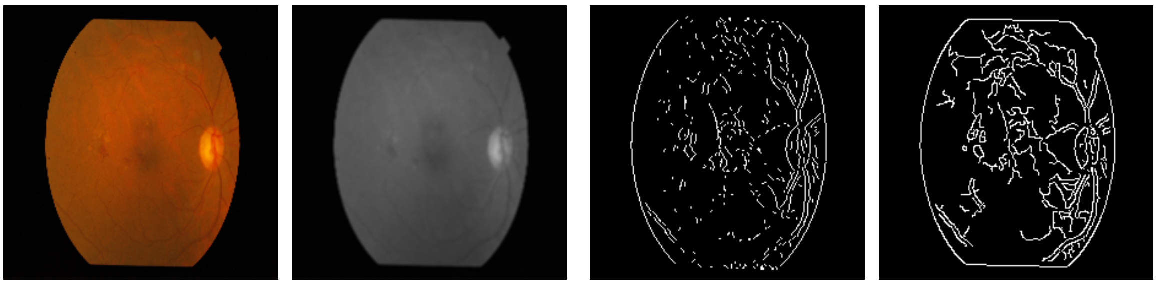

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 20: Img2 Pipeline Image vs Canny Image

Our individual pipeline seems to have detected some of the important edges on the middle right-hand side of the image. However, the edges to the left of the images were only partially detected. This could be due to the blurring kernel size being too large. We note that our pipeline used a **window size of 9** which is quite large. Decreasing this could potentially improve the output results. Furthermore, the linking method used a **threshold of 6** which is relatively small. Increasing this could also potentially improve our results. Despite this, our image still achieved a better result than that of the general pipeline.

- **Img3.png:**
  - ⋆ **Seed value:** 9463
  - ⋆ **Population Size:** 100
  - ⋆ **Generations:** 30
  - ⋆ **Crossover Rate:** 0.9
  - ⋆ **Mutation Rate:** 0.15
  - ⋆ **Tournament Size:** 3

```
    chromosome = {
        "noise_reduction": {
            "method": "median",
            "params": {
                "median_ksize": 5
            }},
        "gradient": {
            "method": "sobel",
            "params": {
                "ksize": 3,
                "ddepth": cv2.CV_64F
            }},
        "thinning": True,
        "thresholding": {
            "method": "binary",
            "params": {
                "bin_thresh": 11.655210948843493
            }},
        "linking": {
            "method": "gradient_direction_analysis",
            "params": {
                "gda_threshold": 12
            }}
    }
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

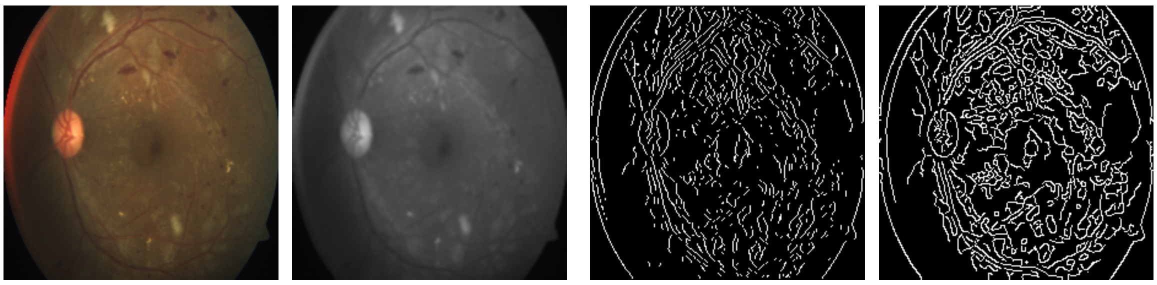

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 21: Img3 Pipeline Image vs Canny Image

Our pipeline seems to have detected most of the regions containing edges. However, it failed to detect finer details which could be due to factors such as **thresholding or noise reduction**. We further note that most of the detected edges (lines) are discontinuous (i.e. not linked properly). This is again, likely due to the small **threshold of 12**.

- **Img4.png:**
  - ⋆ **Seed value:** 9782
  - ⋆ **Population Size:** 200
  - ⋆ **Generations:** 50
  - ⋆ **Crossover Rate:** 0.85
  - ⋆ **Mutation Rate:** 0.2
  - ⋆ **Tournament Size:** 6

```
    chromosome = {
        "noise_reduction": {
            "method": "median",
            "params": {
                "median_ksize": 3
            }},
        "gradient": {
            "method": "sobel",
            "params": {
                "ksize": 3,
                "ddepth": cv2.CV_64F
            }},
        "thinning": True,
        "thresholding": {
            "method": "binary",
            "params": {
                "bin_thresh": 11.785918262418416
            }},
        "linking": {
            "method": "hough",
            "params": {
                "hough_thresh": 10,
                "hough_min_line_length": 0.9172857419242878,
                "hough_max_line_gap": 2
            }}
    }
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

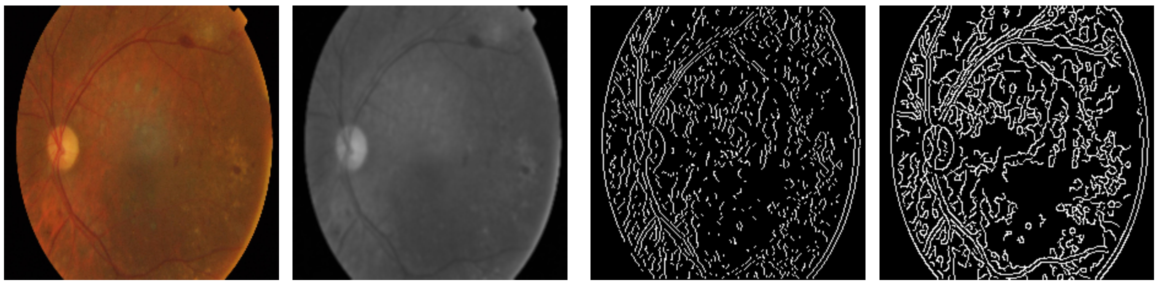

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 22: Img4 Pipeline Image vs Canny Image

The individual pipeline seems to have correctly identified the regions containing the edges. However, it failed to capture the finer details. This is likely due to the **Hough Transform** linking method having a **max line gap of 2** or due to the **minimum line length of approximately 0.9173**. By decreasing the minimum line length the pipeline should be able to identify shorter edges (lines) and link them with surrounding edges. Despite this, our individual pipeline seemed to have performed better than the general pipeline.

- **Img5.png:**
  - ⋆ **Seed value:** 9498
  - ⋆ **Population Size:** 250
  - ⋆ **Generations:** 40
  - ⋆ **Crossover Rate:** 0.85
  - ⋆ **Mutation Rate:** 0.2
  - ⋆ **Tournament Size:** 3

```
    chromosome = {
        "noise_reduction": {
            "method": "median",
            "params": {
                "median_ksize": 3
            }},
        "gradient": {
            "method": "sobel",
            "params": {
                "ksize": 3,
                "ddepth": cv2.CV_64F
            }},
        "thinning": True,
        "thresholding": {
            "method": "hysteresis",
            "params": {
                "hyst_low": 48.7225154287756,
                "hyst_high": 218.3610912550052
            }},
        "linking": {
            "method": "gradient_direction_analysis",
            "params": {
                "gda_threshold": 6
            }}
    }
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

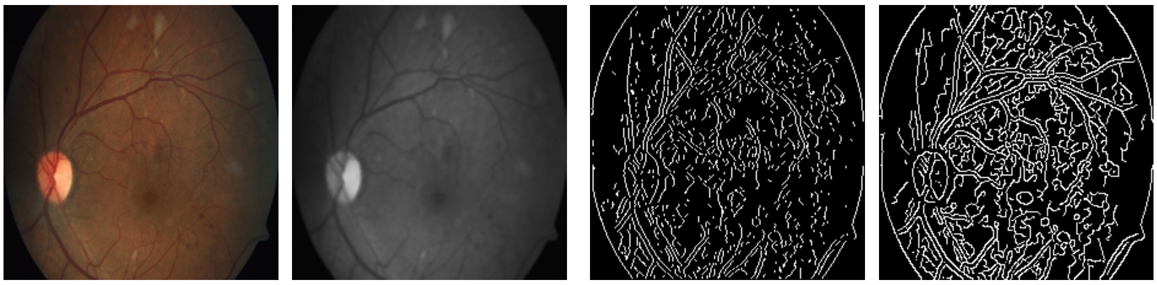

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 23: Img5 Pipeline Image vs Canny Image

The individual pipeline seems to detect some of the edge regions correctly. However, there are a lot of discontinuous lines causing the image to have more *dots* than lines. This is likely due to the linking method having a **threshold of 6**. We further note that the pipeline failed to detect finer details. This is likely due to the **Hysteresis thresholding** method. The **lower threshold of approximately 48.7225** is relatively high which could lead to small edges not accurately being detected.

- **Img6.png:**
  - ⋆ **Seed value:** 9872
  - ⋆ **Population Size:** 125
  - ⋆ **Generations:** 30
  - ⋆ **Crossover Rate:** 0.75
  - ⋆ **Mutation Rate:** 0.15
  - ⋆ **Tournament Size:** 3

```
    chromosome = {
        "noise_reduction": {
            "method": "gaussian",
            "params": {
                "ksize": (7, 7),
                "sigma": 45.28473703389396
        }},
        "gradient": {
            "method": "sobel",
            "params": {
                "ksize": 5,
                "ddepth": cv2.CV_64F
        }},
        "thinning": True,
        "thresholding": {
            "method": "binary",
            "params": {
                "bin_thresh": 11.17565387290022
        }},
        "linking": {
            "method": "gradient_direction_analysis",
            "params": {
                "gda_threshold": 11
        }}
    }
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

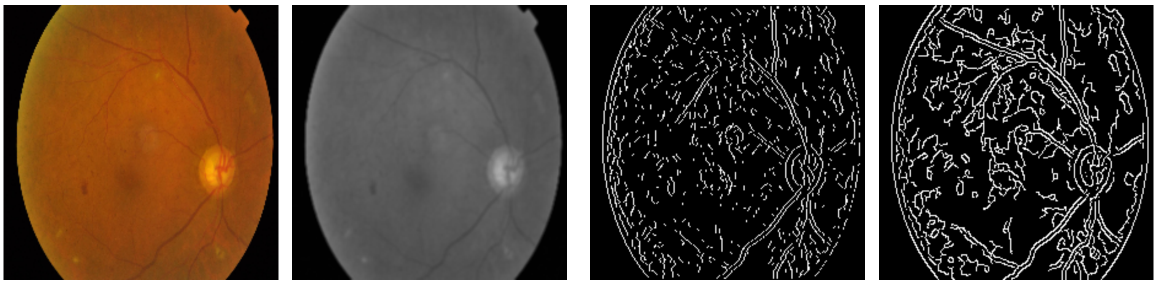

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 24: Img6 Pipeline Image vs Canny Image

Looking at the output image which the individual pipeline generated, we notice that the two images seem similar. Our pipeline seemed to have correctly identified the major edges within the input image (e.g. the circular feature with the darker lines connected to it was correctly detected). However, the smaller, finer details was not detected well enough. The pipeline seems to identify the regions for the edges correctly but failed to connect them. This is, again, likely due to the **gradient directional analysis** edge linking method having a **low threshold of 11**. If for example we used a large threshold (20) then the gradient directions could have had a larger influence on the linking process which in turn could have lead to better linked edges.

- **Img7.png:**
  - ⋆ **Seed value:** 5963
  - ⋆ **Population Size:** 250
  - ⋆ **Generations:** 30
  - ⋆ **Crossover Rate:** 0.8
  - ⋆ **Mutation Rate:** 0.125
  - ⋆ **Tournament Size:** 5

```
    chromosome = {
        "noise_reduction": {
            "method": "gaussian",
            "params": {
                "ksize": (5, 5),
                "sigma": 23.83385249615137
        }},
        "gradient": {
            "method": "laplacian",
            "params": {
                "ksize": 5,
                "ddepth": cv2.CV_64F
        }},
        "thinning": True,
        "thresholding": {
            "method": "binary",
            "params": {
                "bin_thresh": 10.157274781369228
        }},
        "linking": {
            "method": "gradient_direction_analysis",
            "params": {
                "gda_threshold": 8
        }}}
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

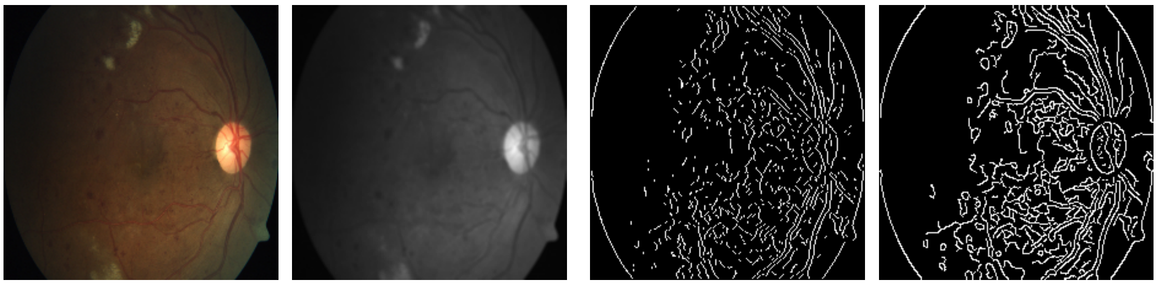

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 25: Img7 Pipeline Image vs Canny Image

Our pipeline seems to have correctly identified the regions in which most of the edges are found. However, once again the edge linking method's **threshold of 8** is too low causing the pipeline to not link the detected edges properly. There are also some finer details not detected by our individual pipeline which could be due to factors such as the **sigma value** of the Gaussian filter being **too high (≈23.8339)**, the **kernel sizes** for both the **Laplacian and Gaussian filter** being **too high (5)** which leads to too much smoothing, loosing finer details. Overall, our pipeline showed good results compared to that when we applied the general pipeline.

- **Img8.png:**
  - ⋆ **Seed value:** 5963
  - ⋆ **Population Size:** 150
  - ⋆ **Generations:** 30
  - ⋆ **Crossover Rate:** 0.9
  - ⋆ **Mutation Rate:** 0.15
  - ⋆ **Tournament Size:** 3

```
    chromosome = {
        'noise_reduction': {
            'method': 'median',
            'params': {
                'median_ksize': 9
        }},
        'gradient': {
            'method': 'sobel',
            'params': {
                'ksize': 3,
                'ddepth': cv2.CV_64F
        }},
        'thinning': True,
        'thresholding': {
            'method': 'binary',
            'params': {
                'bin_thresh': 12.078256915690261
        }},
        'linking': {
            'method': 'gradient_direction_analysis',
            'params': {
                'gda_threshold': 13
        }}}
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

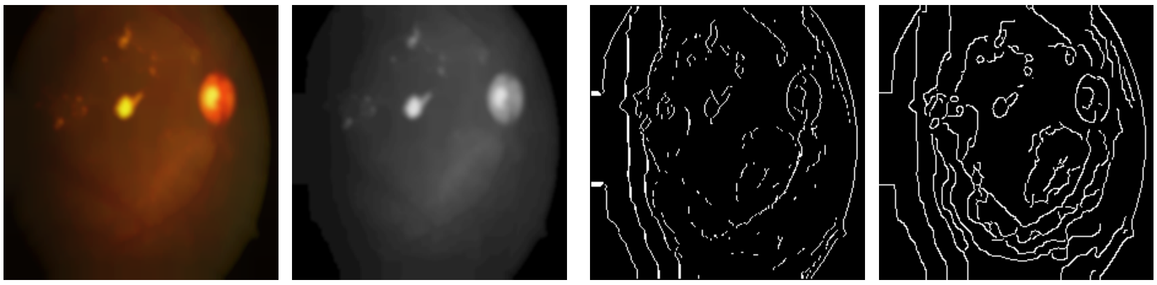

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 26: Img8 Pipeline Image vs Canny Image

Our pipeline seem to have correctly detected most of the edges. The generated output image looks similar to that of the CED. However, the edges found in the lower middle regions of the image were not correctly identified as our pipeline failed to detected the finer details. This is likely due to factors such as the **kernel size** of the **Median filter** having a relatively **high value of 9**. This likely caused too much blurring/smoothing leading to the loss of the finer details. Another factor is the **low threshold value of 13** for the **Gradient Directional Analysis** edge linking method. If we have used a higher threshold then maybe the pipeline could have linked the edges more accurately. Despite these factors, our pipeline performed relatively well.

- **Img9.png:**
  - ⋆ **Seed value:** 5963
  - ⋆ **Population Size:** 225
  - ⋆ **Generations:** 50
  - ⋆ **Crossover Rate:** 0.9
  - ⋆ **Mutation Rate:** 0.05
  - ⋆ **Tournament Size:** 7

```
    chromosome = {
        'noise_reduction': {
            'method': 'median',
            'params': {
                'median_ksize': 9}},
        'gradient': {
            'method': 'sobel',
            'params': {
                'ksize': 3,
                'ddepth': cv2.CV_64F}},
        'thinning': True,
        'thresholding': {
            'method': 'hysteresis',
            'params': {
                'hyst_low': 58.09045218793383,
                'hyst_high': 243.46758242448038}},
        'linking': {
            'method': 'hough',
            'params': {
                'hough_thresh': 2,
                'hough_min_line_length': 0.5588437610319986,
                'hough_max_line_gap': 1}}
    }
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

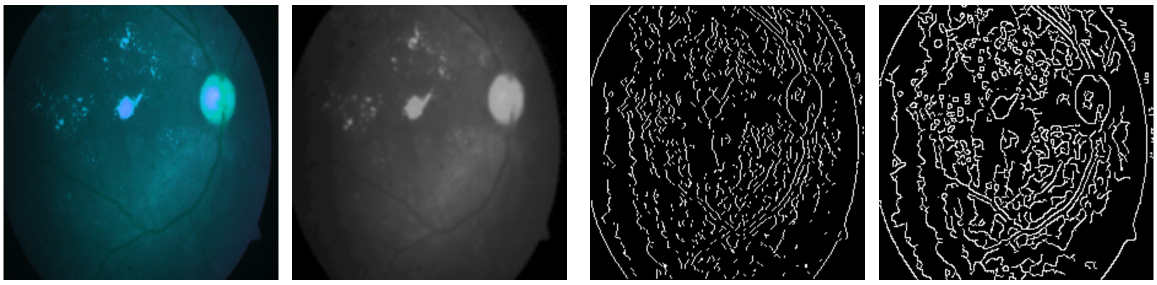

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 27: Img9 Pipeline Image vs Canny Image

The individual pipeline seems to have highlighted/detected the region containing the edges. We see this by comparing the output image of the individual pipeline side by side with that of the CED. However, it is evident that our pipeline did not accurately link the edges. This could be due to finer details not detected which is likely caused by the following factors: a relatively **high kernel size (9)** used by the **Median filter**, the **low threshold** of the **Hysteresis** thresholding method being **too low (≈58.0904)**, and the **low threshold value of 2** for the **Hough Transform** linking method. By, for example, increasing the threshold value for the linking method could potentially lead to better, more accurate linked edges.

- **Img10.png:**
  - ⋆ **Seed value:** 8217
  - ⋆ **Pop Size:** 150
  - ⋆ **Generations:** 50
  - ⋆ **Crossover Rate:** 0.9
  - ⋆ **Mutation Rate:** 0.1
  - ⋆ **Tournament Size:** 4

```
    chromosome = {
        'noise_reduction': {
            'method': 'gaussian',
            'params': {
                'ksize': (3, 3),
                'sigma': 23.91815659542971
        }},
        'gradient': {
            'method': 'sobel',
            'params': {
                'ksize': 3,
                'ddepth': cv2.CV_64F
        }},
        'thinning': True,
        'thresholding': {
            'method': 'binary',
            'params': {
                'bin_thresh': 8.695434927435723
        }},
        'linking': {
            'method': 'gradient_direction_analysis',
            'params': {
                'gda_threshold': 12
        }}}
```

Our GA then used this parameters and pipeline configuration and applied it to the input, gray scale image. The following were the output images compared side by side with the CED.

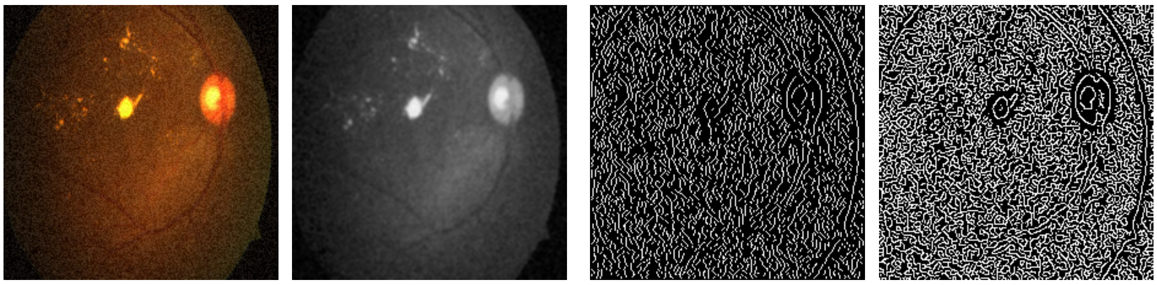

(a) Original Image (b) Gray Scale (c) Pipeline Image (d) Canny Image

Figure 28: Img10 Pipeline Image vs Canny Image

By comparing the two output images side by side we can see that our pipeline generates a similar image to that of the CED just with less fine edges detected. When looking at the input image we note that it contains a lot of noise which could suggest that our implementation of the Canny did not reduce the noise enough and thus leading to a much noisier output image. However, as mentioned our pipeline failed to detect the finer details which is likely due to our **sigma value of ≈24** of the **Gaussian Noise reduction filter** being slightly too high which can cause too much smoothing and thus leading to important details being lost within the image. Edge linking also seemed to produce worse results than that of the thresholding step e.g. Figure 29 shows the comparison between our output image and the output image if we were to stop the pipeline process before the edge linking step. From the comparison we note that the linking step fails to link some important edges which could be due to the **low threshold value of 12** that was used. We also note the difference in performance scores in Table 4.

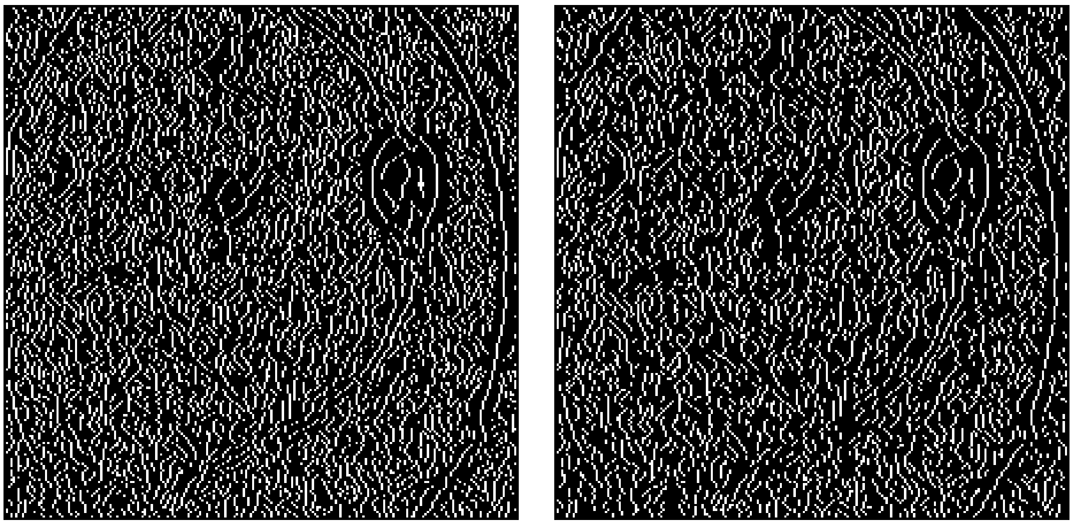

(a) Img10.png before linking (b) Img10.png after linking

Figure 29

3.3.1 Critical Analysis on Individual Pipelines

We obtained the following metric scores when we compared the two generated images for each individual pipeline (i.e. we compared images **c) and d)**):

Image	MSE	Structural Sim.	PSNR	SIFT
Img0.png	**5475.3393**	**0.5758**	**10.7467**	**0.4087**
Img1.png	**8242.1676**	**0.3651**	**8.9704**	**0.523**
Img2.png	**3153.0179**	**0.6989**	**13.1435**	**0.3636**
Img3.png	**6846.442**	**0.5119**	**9.7762**	**0.3881**
Img4.png	**6876.2486**	**0.5157**	**9.7573**	**0.3804**
Img5.png	**7237.8154**	**0.4927**	**9.5347**	**0.3766**
Img6.png	**5851.1614**	**0.4958**	**10.4584**	**0.4062**
Img7.png	**5213.5598**	**0.5915**	**10.9595**	**0.4264**
Img8.png	**3492.5537**	**0.659**	**12.6994**	**0.449**
Img9.png	**6582.0706**	**0.4951**	**9.9472**	**0.3991**
Img10.png	**14644.1028**	**0.3852**	**6.4742**	**0.2804**
***Img10.png (after thresh.)**	**13607.3521**	**0.4481**	**6.7931**	**0.2642**
Averages (without *)	**6921.8479**	**0.5297**	**10.2508**	**0.4005**

Table 4: Individual Pipeline Application results

Overall, the generated output images looked similar to that of the Canny Edge Detector (CED). However, when doing a deeper analysis in terms of how these individual pipeline performed (in terms of the metric scores) we note that the results do not seem as good results. For instance, the MSE metric on average is **6921.8479** with a few outliers such as *Img10.png* and *Img2.png* having scores of ≈ 3153 and ≈ 14644 respectively. Based on these results it suggests that there are major pixel-wise differences between the pipelines' generated images and the CED's images. On the other hand the Peak-Signal-Noise-Ratio (PSNR) scores averaged on **10.2508** which reflects the noisiness resulting in the generated images.

Furthermore, we note that the individual pipelines achieved ≈**53%** structural similarity (SSIM) on average. This then highlights that even though the pipelines did not detect the edges accurately there are still some common features among the output images. Additionally, the SIFT score was used to help indicate the percentage of key features detected by our individual pipelines. We note that the individual pipelines achieved an average score of approximately 40%.

Therefore, even though our individual pipelines did not detect the edges accurately in terms of the location or intensity values of the pixels, the pipelines still managed to maintain the structural details of the image. This then supports our claim of why the output images look similar to that of the CED. This then highlights areas where improvements can be made to ensure that the individual pipelines are able to detect edges accurately.

Finally, when comparing the individual pipeline scores with that of the general pipeline we observe that every individual pipeline achieved a better fitness score than the general pipeline. This is due to the GA optimizing the pipeline for each respective image rather than optimizing it for a single image and applying it for all images.