This is a description of the edge detection algorithm developed to detect the edges in the given Mario jpeg image.

Various different methods have been employed to detect edges in the given image. They are:

1. Sobel Operator (Both kernels)
2. Scharr Operator
3. Robert's Operator
4. Prewitt's Operator

STRUCTURE AND CONTENTS

The structure and contents of each file in this repository is as follows:

1. week1.ipynb – This contains rough work / experiments used in the development phase.
2. week1_final.ipynb – This file contains the FINAL code for the project with all the results and proper plots for each operation.
3. Copy of file(2) has also been included in this repo, it also contains rough work and experiments.
4. README – Overview of the code.

PRE-PROCESSING

- First few lines of the code are self-explanatory, they involve preprocessing of the image. I have used the library open-cv to load the image into the interactive development environment and then convert them into grayscale.
- Converting the image into grayscale helps simplify our task as we are able to assign a single numerical value between 0-255 to each cell and in further computations, detecting the change in intensity becomes easier. For instance, had it been still in RGB, our algorithms to detect the change in intensity would have become much lengthier and more complicated. In RGB, algorithms would need to account for variations across all three channels.
- To extract the grayscale value for the image, I have divided it into a **160x300 matrix** where each cell contains a single integral value ranging from 0-255 with 0 indicating a totally black cell and 255 representing a white cell. This has been called the **grid value matrix**.
  Experiment #1: Use of the grayscale formula v/s the average value calculation for the image:
  Initial versions of the code relied on the average grayscale value to construct the grid value matrix. Although insignificant for this code, using the grayscale formula is a better practice as it helps to construct the grid value matrix according to the perceived intensity of each channel. Example. Maximum intensity of green is perceived to be brighter than that of blue and hence respective weights are assigned to account for these tiny details.
  IMPACT: Due to the simplicity of the image and absence of sharp features, it makes little difference which method we use here. However, for images with more complex details, the grayscale formula provides a more accurate representation of intensity.
- In essence, to extract features from the image and achieve our goal of detecting edges, we need to perform a convolution. A function, **convolve,** with inputs as the grid value matrix, the kernel, stride and padding is written out. This function has been used throughout the code to perform convolutions.
- In pre-processing, another function, the **scaler function** has been constructed. This function is responsible for scaling all the values in its input matrix to the grayscale range. This has been done to avoid discrepancies caused by varying value ranges in the input data, ensuring uniformity and consistency during computations and further analysis. By scaling values to the

standard grayscale range (0-255), the function helps maintain compatibility with image processing algorithms.

ALGORITHM IMPLEMENTATION

There are primarily two broad approaches in the edge detection problem – the first approach involves using gradient descent and the other one is the gaussian approach. In this code, I have only used the gradient descent based approaches.

The gradient descent based algorithms involve analysing changes in intensity to identify edges, by calculating gradients in the image. This code contains the implementation of the four aforementioned gradient descent algorithms.

- **Sobel-Feldman Operator**
  The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. It is named after Irwin Sobel and Gary M. Feldman, colleagues at the Standford Artificial Intelligence Laboratory (SAIL).
  This operator is designed in such a manner that it can predict edges using variations in intensity of various cells in the grid value matrix. A Sobel operator consists of two matrices – the Sobel X and Sobel Y, which measure the variation in intensity in the X and Y (i.e., horizontal and vertical) directions respectively.

- **Sobel-X:** The matrix is as shown below. Its working can be understood as weighted subtractions. When this matrix is convolved over the grid value matrix, it subtracts left values from the right and the middle row has been cleverly left blank so that during a convolution, it does not affect the result as in essence, it ignores changes in the vertical direction. Once this is done, the resulting matrix obtained is nothing but the gradient of the intensity in the horizontal (x) direction.
  Another thing to note in this matrix is the use of “2” and “-2” in the middle row in contrast to “1” and “-1” in the top and bottom rows, this has been done to emphasise on the intensity changes in direct neighbourhood of the concerned cell. Obviously, the weights have been so chosen that they approximate the gradient (or partial derivative) of the intensity function in the horizontal direction. This design helps enhance the detection of horizontal edges while reducing the influence of diagonal or vertical intensity variations.

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

- **Sobel-Y:** The matrix is shown below and behaves the same as Sobel-X but in the horizontal direction, i.e., calculates the derivative of the intensity function in the Y (horizontal) direction.

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- **Combining Both:**
  (A) Square-Root Approach: After performing the convolutions and obtaining the X and Y matrices, we can compute the overall change in intensity with a formula analogous to the Pythagorean theorem.

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

  The resultant, **G**, indicates the overall change in intensity (in other words, the gradient magnitude) and the matrix constructed is the output matrix for the operator. This matrix highlights edges in the image, as areas with high gradient magnitudes correspond to regions of rapid intensity change.
  Achieved an F1 score of 0.85, Precision of 0.78, Recall of 0.94

  (B) Modulus Approach: This approach just uses a different formula for calculating G, i.e. With this approach, I achieved an **F1 score of 0.86, Precision of 0.81 and Recall of 0.91.**

$$G = |G_x| + |G_y|$$

  No major differences observed right away but as is evident from the evaluation metrics, the modulus approach does give us a better precision and recall, suggesting that it could get a hold of TP (True Positives) much better than the square root operator as is indicated by the higher precision score. A slightly lower recall could indicate that higher FN (False Negatives) being identified by this model.

This section contains a report of various modifications/comparisons/experiments which I did to measure the performance of my model.

- **Experiment #2:** Use of Gaussian Blur
  In this project, a Gaussian blur was applied to the input image to test its effect on edge detection. However, because the image had clear and well-defined intensity transitions, the blur did not result in any significant changes. The edges were already prominent, and the minor noise reduction from the blur had little effect on the outcome. Hence, this was NOT added in the final code. All evaluation metrics remained unchanged till the first two decimal places.

- **Experiment #3:** Adding a threshold value
  A threshold was added in the Sobel function to see how it changed the output image. The logic was such that if the threshold was exceeded, we would get a white response and if it was not exceeded then black would be returned. This was also not of any significant use as the algorithm failed to detect certain edges near the characters hand. Hence this was NOT added in the final code.

- **Experiment #4:** Comparison with Prewitt's operator
  Prewitt's operator, similar to the Sobel-Feldman operator is another edge detection operator which is implemented in the same way as the Sobel operator. It is structurally identical to the

$$\mathbf{G_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix}$$

Sobel Operator, i.e., it is also a 3x3 kernel with negative and positive values and emphasizes on the derivative of the intensity function to detect edges in the image. The matrix for this operator is shown below

According to the evaulation metrics, this operator lagged behind the Sobel-Feldman operator on all counts, i.e., F1 Score, Precision and Recall. (This behaviour is expected as Prewitt's operator lacks the emphasis on immediate neighbours, which Sobel-Feldman accounts for).