Digital Image Processing – Project

Student1:Samia Hamed
Student Number : 12113019

Student2: Omar Jarbou
Student Number: 12112067

Part 1: Real-Time Processing Pipeline (Image Stylization Using Posterization and Vignette Mask After Converting the Image to Grayscale and Sharpening it)

- Introduction

Overview

The goal of this code is to apply two different image-processing effects to a sharped grayscale image in order to produce a stylized, cinematic look:

- A vignette effect – darkens the edges of the image while keeping the center bright (Transition is Gradientlly).
- A posterization effect – reduces the number of brightness levels (K levels), creating a cartoon/noir appearance.

These two techniques can be applied separately or combined to produce artistic variations.

How the Code Works (High-Level Idea)

a) Preparing the Vignette Mask

The function create_vignette_mask(width, height) generates a mask (with the same size as the original image) based on the distance of each pixel from the center of the image.

Pixels near the center get values close to 1 (no darkening).

Pixels near the corners get values close to 0 (strong darkening).

When this mask is multiplied with the actual image, the edges become darker → producing a vignette effect.

b) Posterizing the Image

The posterize(img, levels) function reduces the total number of gray levels in the image. Instead of using 256 possible values (0–255), the image is quantized to K discrete intensity bands/levels, such as 0, 51, 102, 153, 204, 255 if K = 6.

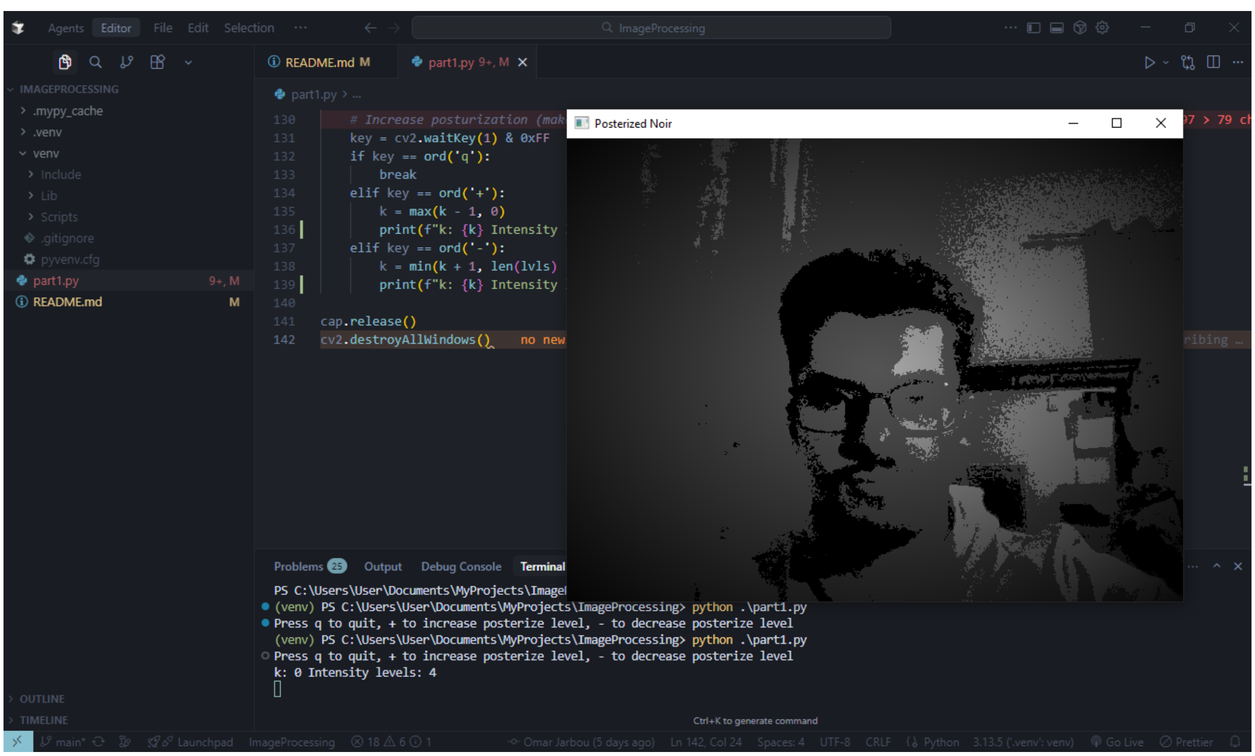
This removes smooth gradients and gives the image a stylized/cartooned/noir appearance.

c) Final Workflow

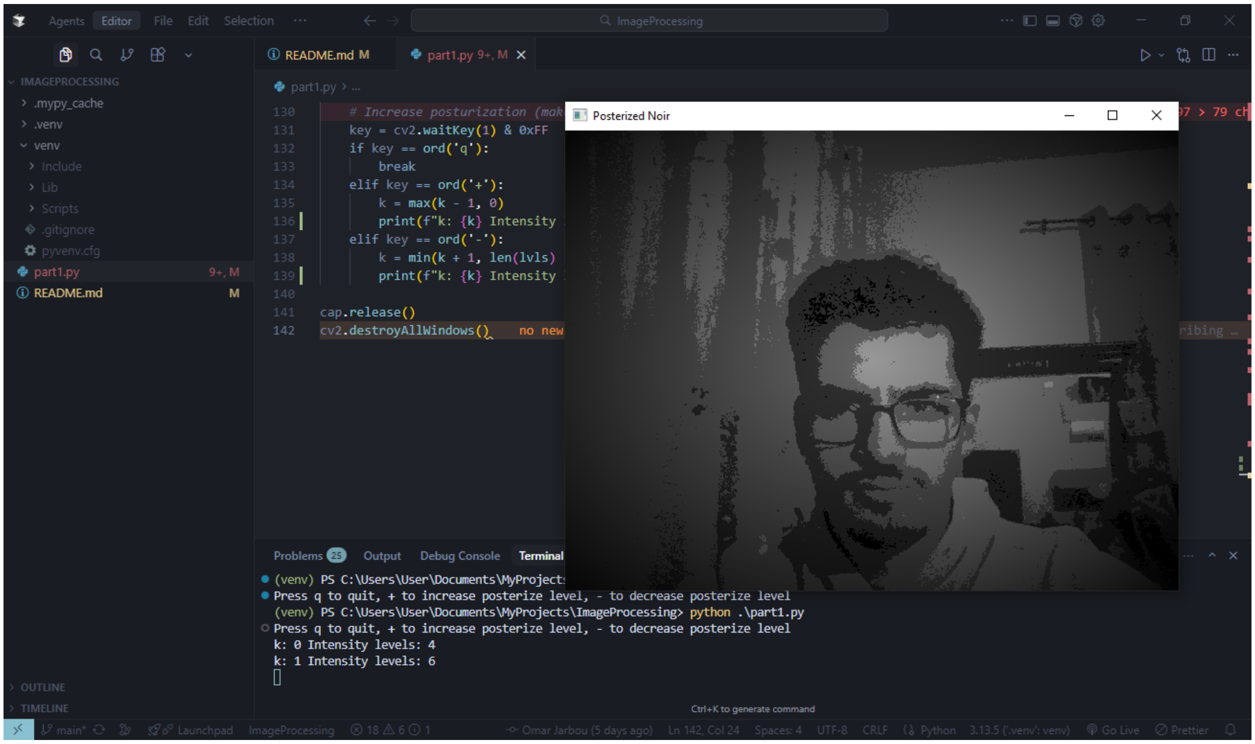
- Create a vignette mask matching the image size.
- Load a grayscale image using cv2.cvtColor.
- Sharpening the image using OpenCV's Unsharp masking (cv2.addWeighted), which requires a blurred version of the image before it can do it work (sharp = original + amount*(original - blurred)). We get that blurred image using cv2.GaussianBlur.
- Apply posterization on the sharped image with the K (# of levels) value entered by the user.
- Multiply the image by the vignette mask to darken the edges.
- Display or save the results.

- Results

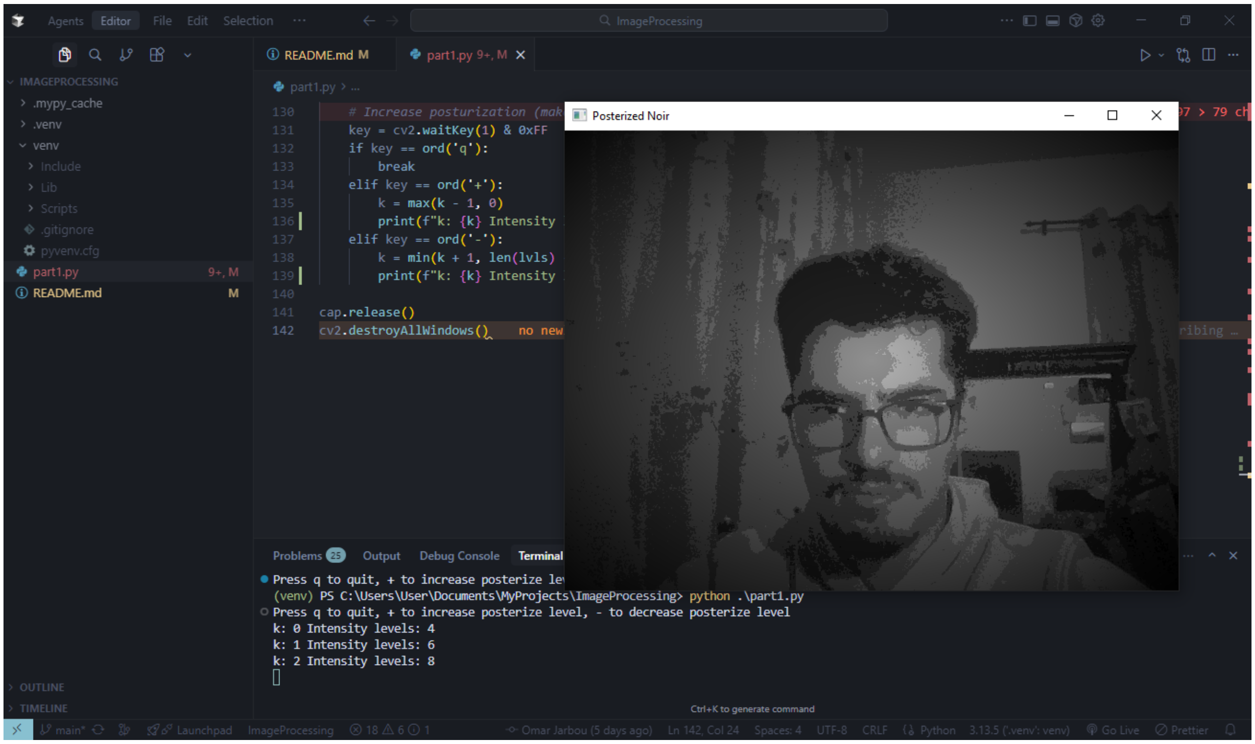
of intensity levels = 4



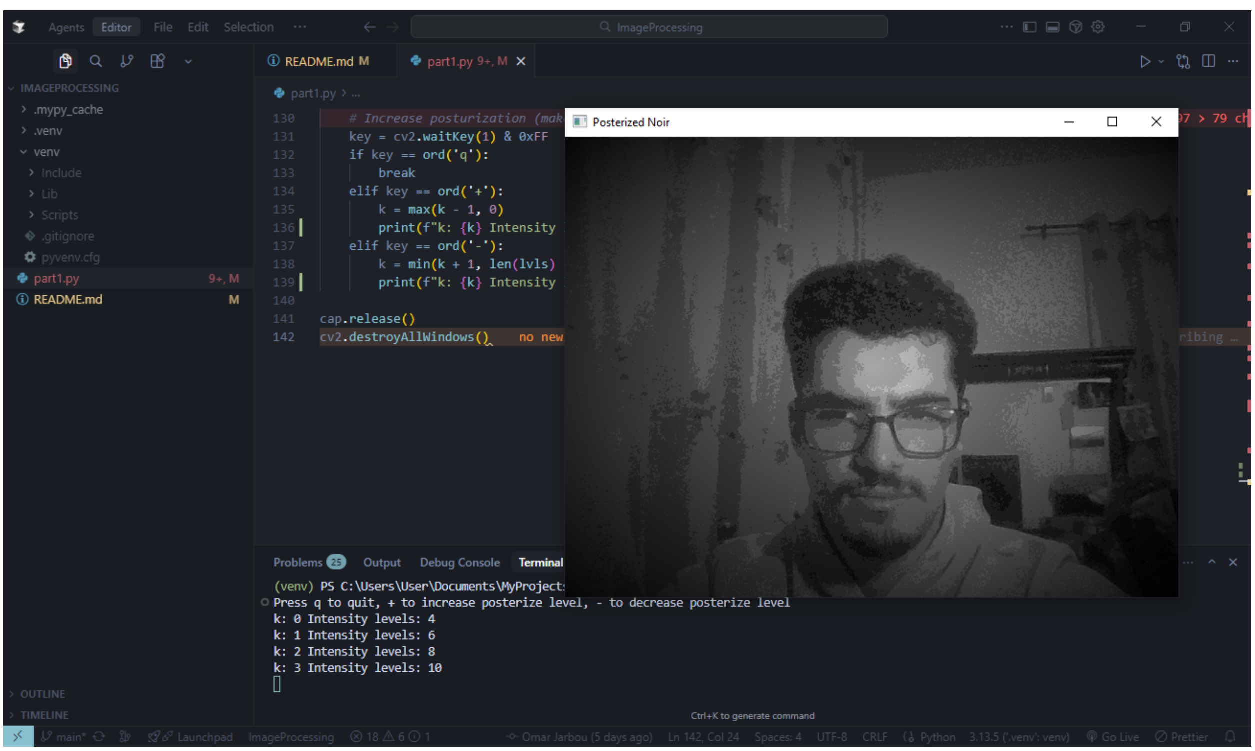
of intensity levels = 6



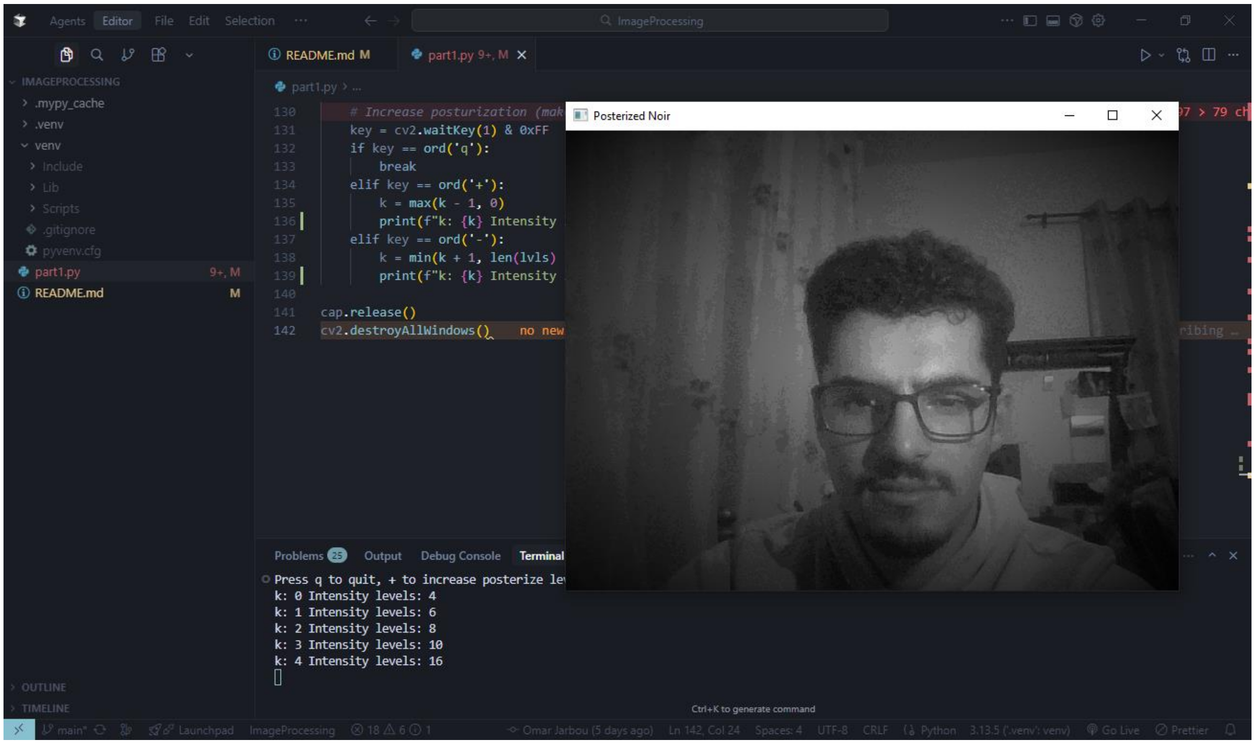
of intensity levels = 8



of intensity levels = 10



of intensity levels = 16



AND SO ON…

Part2: Image Compression and Redundancy

- Introduction

Digital images contain a large amount of data, and many of these data are redundant.
Image compression techniques aim to reduce the number of bits needed to represent the image while preserving as much information as possible.
This project implements a simple program that analyzes an input image, identifies the type of redundancy it contains, and applies an appropriate compression algorithm learned in the course.

The program displays:

- Original image size
- Compressed size
- Compression ratio
- Redundancy percentage

Types of Redundancy :

- Coding Redundancy : Occurs when symbols (pixel values) are represented by inefficient codewords.
- Spatial (Interpixel) Redundancy : Occurs when neighboring pixels have similar values.
- Psychovisual Redundancy **:** Not used in this project because it requires *lossy* compression (like JPEG), which is beyond the scope of Part 2.

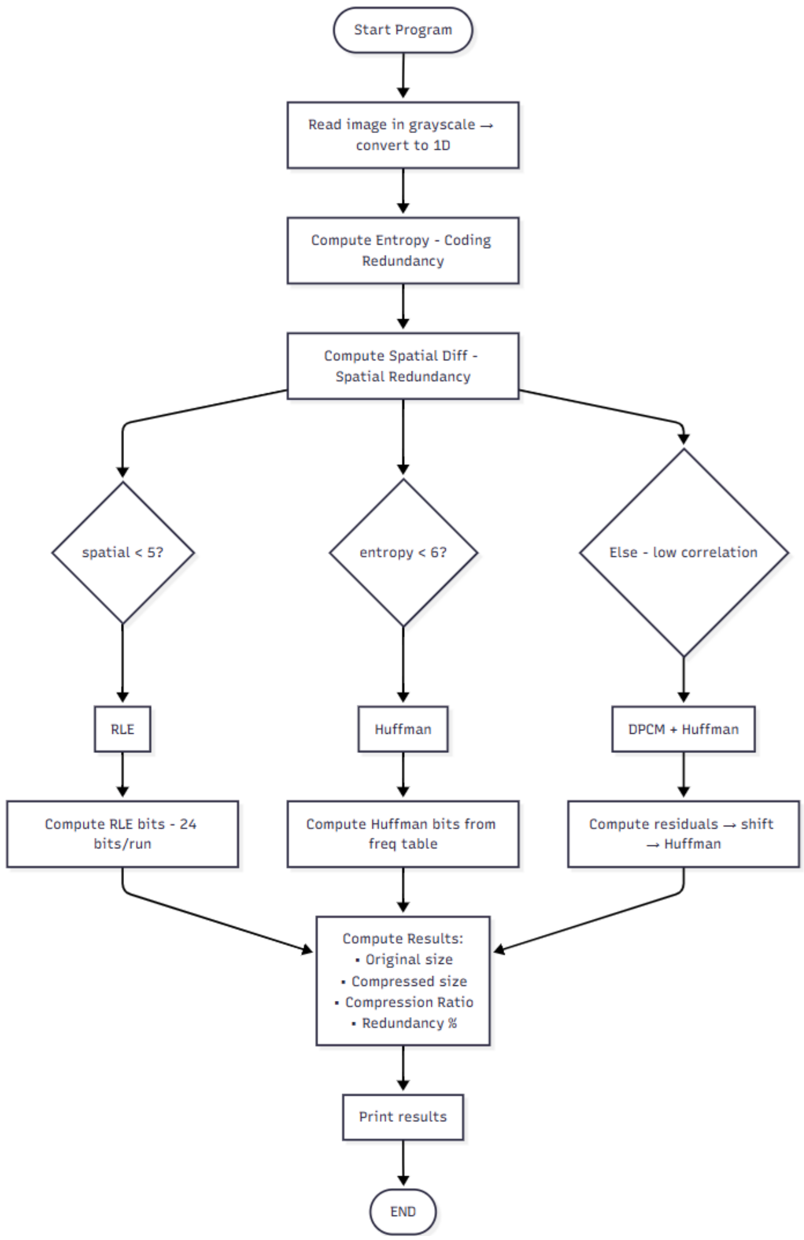
Implemented Compression Methods:

- Run-Length Encoding (RLE)
- Huffman Coding
- DPCM + Huffman

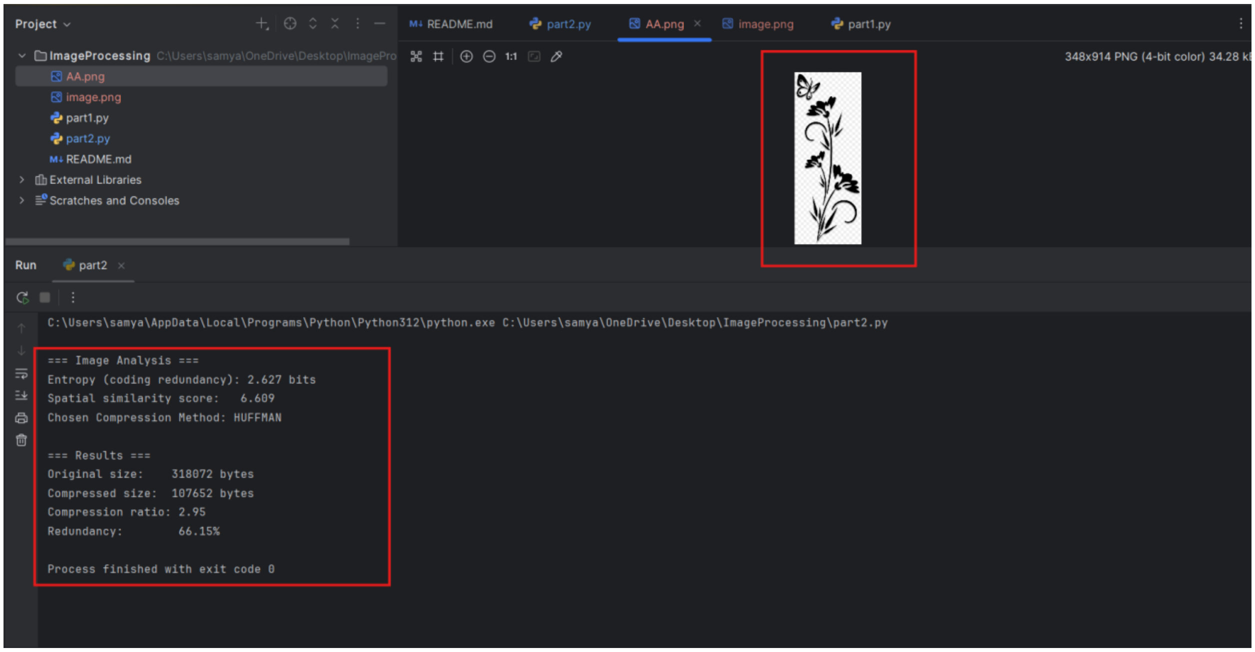
Redundancy Analysis Methods:

- Entropy Calculation
- Spatial Similarity Score
- Method Selection Criteria

- Code Flow



- Output and result

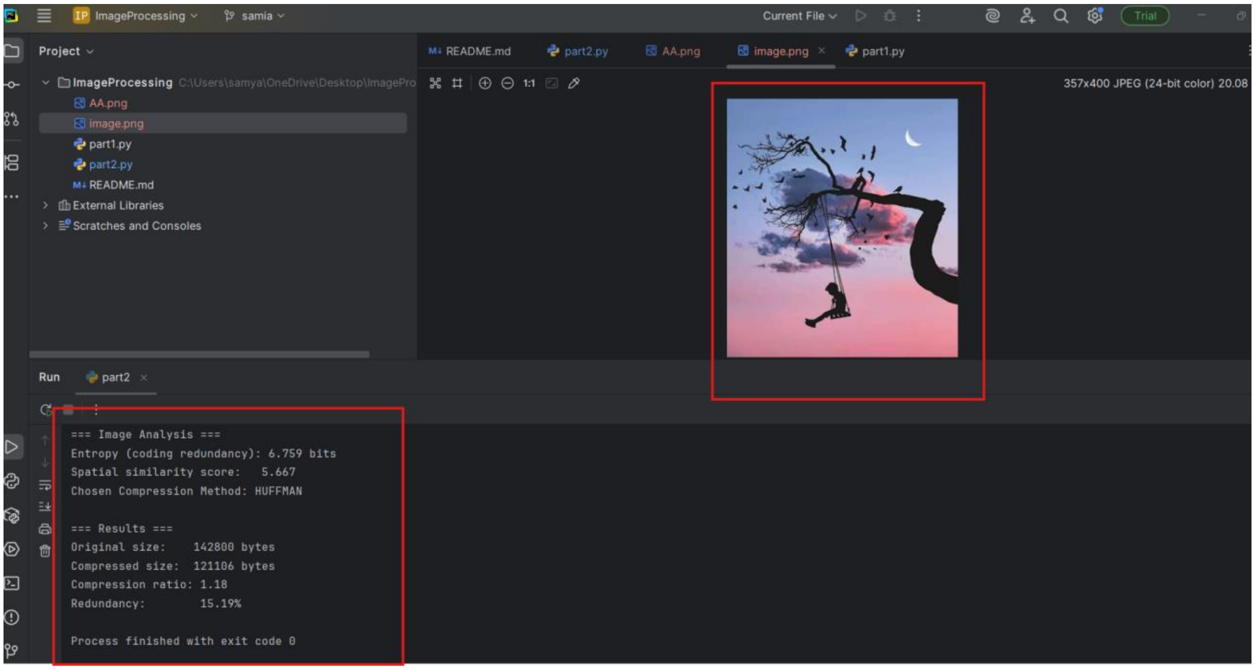


Analysis:

- **Entropy = 2.62 bits** → The image has low information content with many repeated pixel values, making it highly compressible using Huffman coding.
- **Spatial similarity = 6.6** → The image is not smooth enough for RLE, so Huffman is the most suitable method.

Results:

- **Compression ratio = 2.95** → The image size was reduced to almost one-third, indicating very effective compression.
- **Redundancy = 66%** → About two-thirds of the data was redundant and successfully remove



Analysis:

- **Entropy = 6.75 bits** → The image contains many colors and high complexity, meaning its data is almost non-repetitive and difficult to compress.
- **Spatial similarity = 5.66** → Some local similarity exists, but not enough for RLE; Huffman is still the most appropriate option.

Results:

- **Compression ratio = 1.18** → Compression was weak because the image has very little redundancy.
- **Redundancy = 15%** → Only a small portion of the data was compressible, and 85% is essential information.

Repository link:

https://github.com/OmarJarbou/ImageProcessing.git