

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 Gradiently).
- 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

A screenshot of a code editor (VS Code) showing a Python script named `part1.py`. The script contains code for increasing posterization levels and a terminal window showing the execution of the script and the resulting posterized image.

```
130 # Increase posterization (make it more black and white)
131 key = cv2.waitKey(1) & 0xFF
132 if key == ord('q'):
133     break
134 elif key == ord('+'):
135     k = max(k - 1, 0)
136     print(f"k: {k} Intensity")
137 elif key == ord('-'):
138     k = min(k + 1, len(lvls))
139     print(f"k: {k} Intensity")
140
141 cap.release()
142 cv2.destroyAllWindows()
```

The terminal window shows the command `python .\part1.py` being run, followed by several lines of text indicating key presses and the resulting posterized image:

```
PS C:\Users\User\Documents\MyProjects\ImageProcessing> python .\part1.py
● (venv) PS C:\Users\User\Documents\MyProjects\ImageProcessing> python .\part1.py
● Press q to quit, + to increase posterize level, - to decrease posterize level
(venv) PS C:\Users\User\Documents\MyProjects\ImageProcessing> python .\part1.py
○ Press q to quit, + to increase posterize level, - to decrease posterize level
k: 0 Intensity levels: 4
```

The image preview in the code editor shows a stylized, high-contrast portrait of a man with glasses, representing the posterized output.

of intensity levels = 6

The screenshot shows a code editor interface with a sidebar and several tabs. The sidebar lists project files: README.md, part1.py, and README.md again. The main area has two tabs: README.md (marked with a 'M') and part1.py (marked with a '9+, M'). The part1.py tab contains the following code:

```
130 # Increase posterization (max)
131 key = cv2.waitKey() & 0xFF
132 if key == ord('q'):
133     break
134 elif key == ord('+'):
135     k = max(k - 1, 0)
136     print(f"K: {k} Intensity")
137 elif key == ord('-'):
138     k = min(k + 1, len(lvls))
139     print(f"K: {k} Intensity")
140
141 cap.release()
142 cv2.destroyAllWindows()
```

Below the code editor is a terminal window showing command-line interaction:

- (venv) PS C:\Users\User\Documents\MyProject
- Press q to quit, + to increase posterize level, - to decrease posterize level
- (venv) PS C:\Users\User\Documents\MyProjects\ImageProcessing> python .\part1.py
- Press q to quit, + to increase posterize level, - to decrease posterize level
- k: 0 Intensity levels: 4
- k: 1 Intensity levels: 6

To the right of the terminal is a small image window titled "Posterized Noir" displaying a posterized version of a portrait of a man with glasses.

of intensity levels = 8

A screenshot of a code editor (VS Code) showing a Python script named `part1.py`. The script uses OpenCV to capture a video from a camera and posterize the frames. A terminal window shows the execution of the script and the resulting posterized levels.

```
# Increase posterization (make image look like a poster)
key = cv2.waitKey(1) & 0xFF
if key == ord('q'):
    break
elif key == ord('+'):
    k = max(k - 1, 0)
    print(f"k: {k} Intensity")
elif key == ord('-'):
    k = min(k + 1, len(lvls))
    print(f"k: {k} Intensity")
cap.release()
cv2.destroyAllWindows()
```

Terminal output:

- Press q to quit, + to increase posterize level
- (venv) PS C:\Users\User\Documents\MyProjects\ImageProcessing> python .\part1.py
- Press q to quit, + to increase posterize level, - to decrease posterize level
- k: 0 Intensity levels: 4
- k: 1 Intensity levels: 6
- k: 2 Intensity levels: 8

of intensity levels = 10

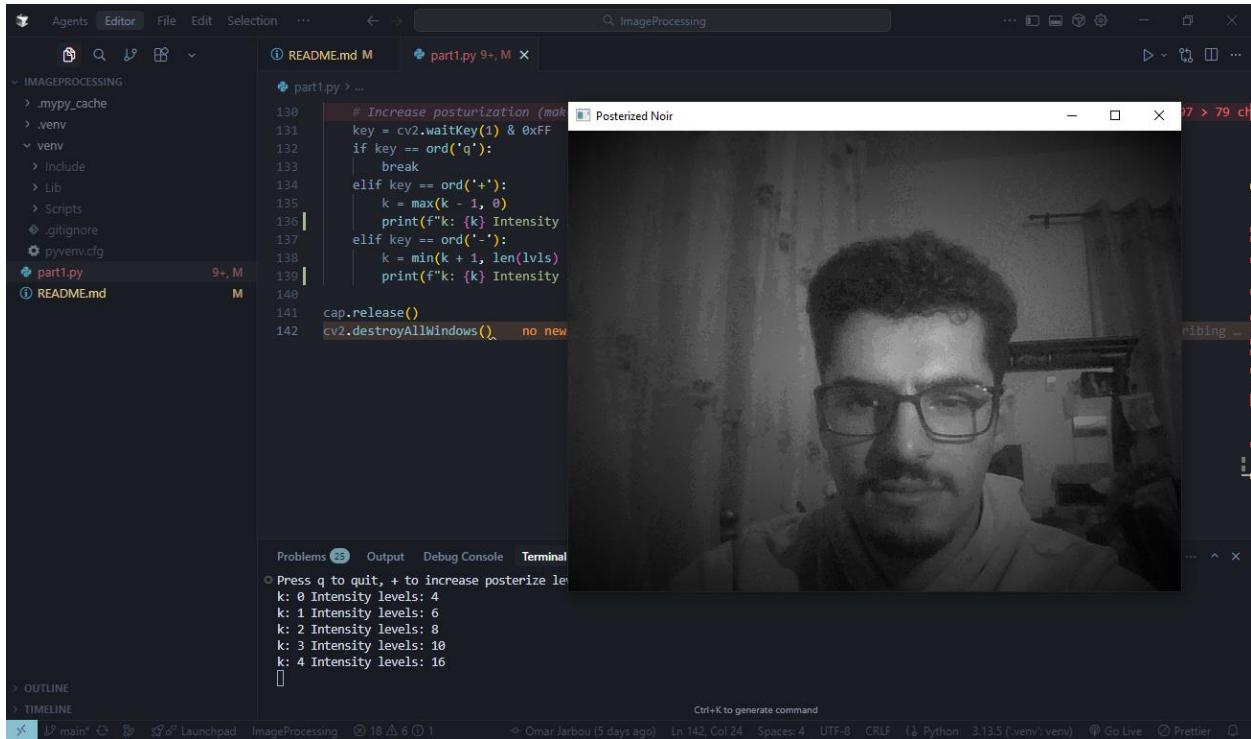
A screenshot of a code editor (VS Code) showing the same Python script (`part1.py`) as the previous screenshot. The script is identical, but the terminal output shows the intensity level has been increased to 10.

```
# Increase posterization (make image look like a poster)
key = cv2.waitKey(1) & 0xFF
if key == ord('q'):
    break
elif key == ord('+'):
    k = max(k - 1, 0)
    print(f"k: {k} Intensity")
elif key == ord('-'):
    k = min(k + 1, len(lvls))
    print(f"k: {k} Intensity")
cap.release()
cv2.destroyAllWindows()
```

Terminal output:

- Press q to quit, + to increase posterize level, - to decrease posterize level
- (venv) PS C:\Users\User\Documents\MyProjects>
- Press q to quit, + to increase posterize level, - to decrease posterize level
- k: 0 Intensity levels: 4
- k: 1 Intensity levels: 6
- k: 2 Intensity levels: 8
- k: 3 Intensity levels: 10

of intensity levels = 16



```
# Increase posterization (main)
key = cv2.waitKey(1) & 0xFF
if key == ord('q'):
    break
elif key == ord('+'):
    k = max(k - 1, 0)
    print(f"K: {k} Intensity")
elif key == ord('-'):
    k = min(k + 1, len(lvls))
    print(f"K: {k} Intensity")
cap.release()
cv2.destroyAllWindows()
```

Problems 25 Output Debug Console Terminal

Press q to quit, + to increase posterize level
k: 0 Intensity levels: 4
k: 1 Intensity levels: 6
k: 2 Intensity levels: 8
k: 3 Intensity levels: 10
k: 4 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.

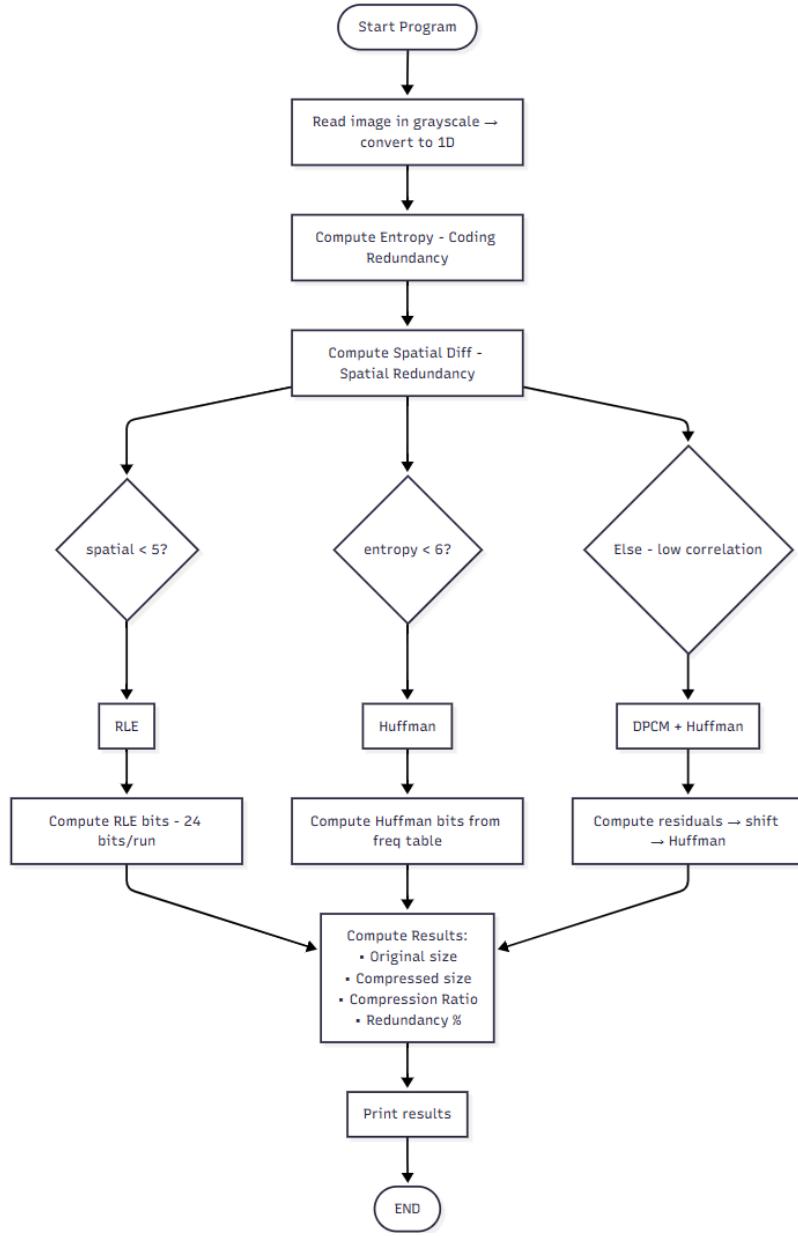
Implemented Compression Methods:

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

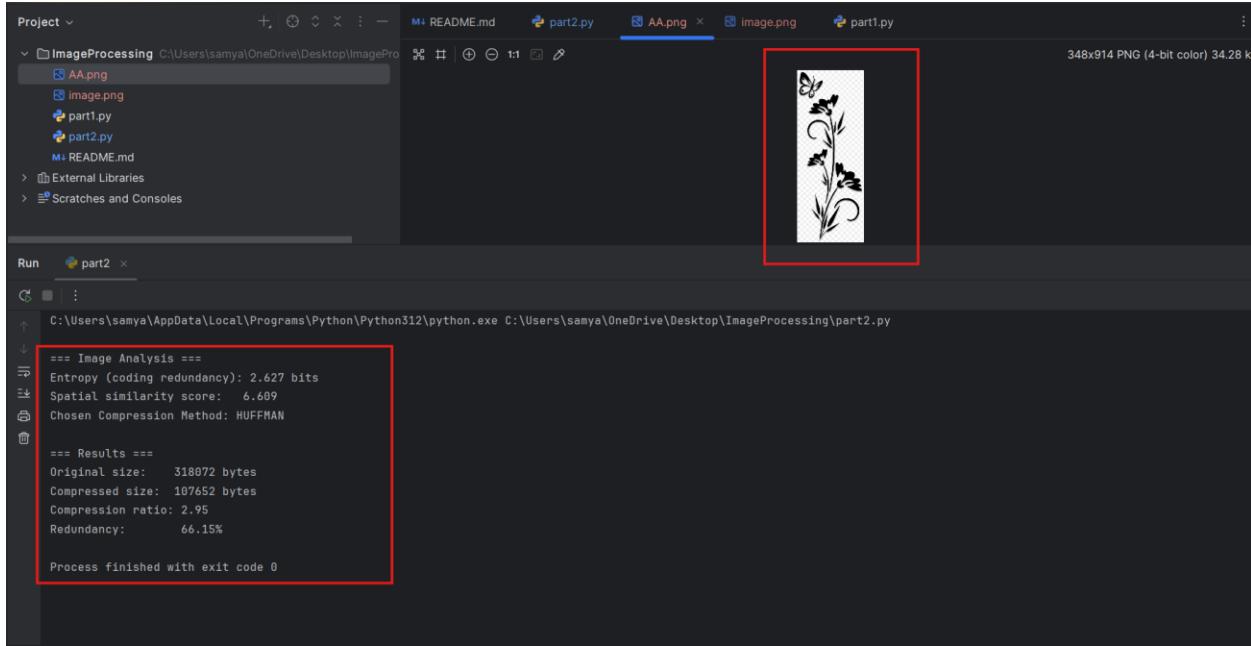
Redundancy Analysis Methods:

- 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 removed

```
==> Run part2 <==  
==== Image Analysis ====  
Entropy (coding redundancy): 6.759 bits  
Spatial similarity score: 5.667  
Chosen Compression Method: HUFFMAN  
  
==== Results ====  
Original size: 142800 bytes  
Compressed size: 121106 bytes  
Compression ratio: 1.18  
Redundancy: 15.19%  
Process finished with exit code 0
```

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>

