Deeploy CV Project

Assignment : 1

Name: Vaibhav Itauriya

Roll Number: 231115

ID : 223

Branch: ME

Date of submission: 17/12/2024

GitHub Repository of this Project:

Answer 1

In this code, I have learned about using the matlplotlib library to print (plot) image in the Jupyter Notebook environment.
The code involves reading an image file and plotting it on a subplot using the 'imshow' method to visualize the image.

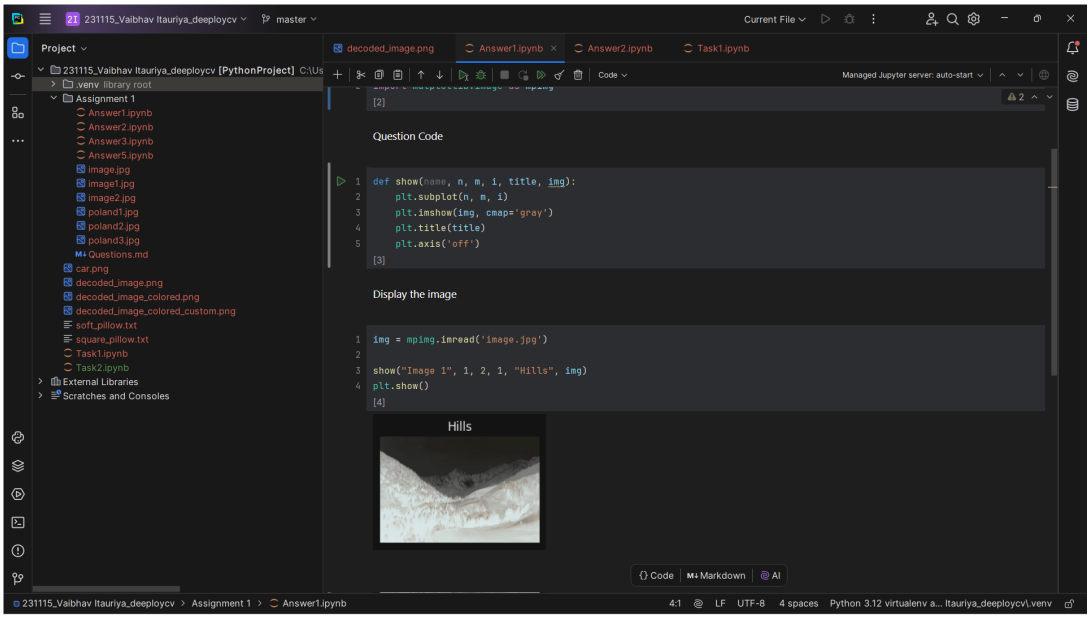

Figure 1: Output in Pycharm.

Solution Description

The function 'show()' generates a plot for the image, with customisable parameters for the subplot's position, title, and layout. The variables n, m and i are used for the following:

- **n**: Specifies the number of rows in the subplot grid.

- **m**: Specifies the number of columns in the subplot grid.
- **i**: Specifies the index of the current subplot where the image will be displayed.

The 'cmap = 'gray' ' argument displays the image in grayscale. The 'axis('off')' command hides the axes for a cleaner image presentation.

Link to Github Code: Click here.

Answer 2

In this code, I have worked with different image processing techniques using the OpenCV library. These operations allow me to manipulate and analyse images in various ways, like turning them into grayscale, detecting edges, applying filters, and converting between colour formats.

By the end of the code, the image goes through several steps, each applying a specific transformation. These steps help us understand how images can be modified for edge detection, enhancing details, or changing colours. The code demonstrates a practical application of different image processing techniques you might have learned theoretically earlier.

As I have analysed, I have found that my computer generates RGB images and that both Sobel and Canny are good, but Canny is better when plotting edges of straight lines or uniform shapes.

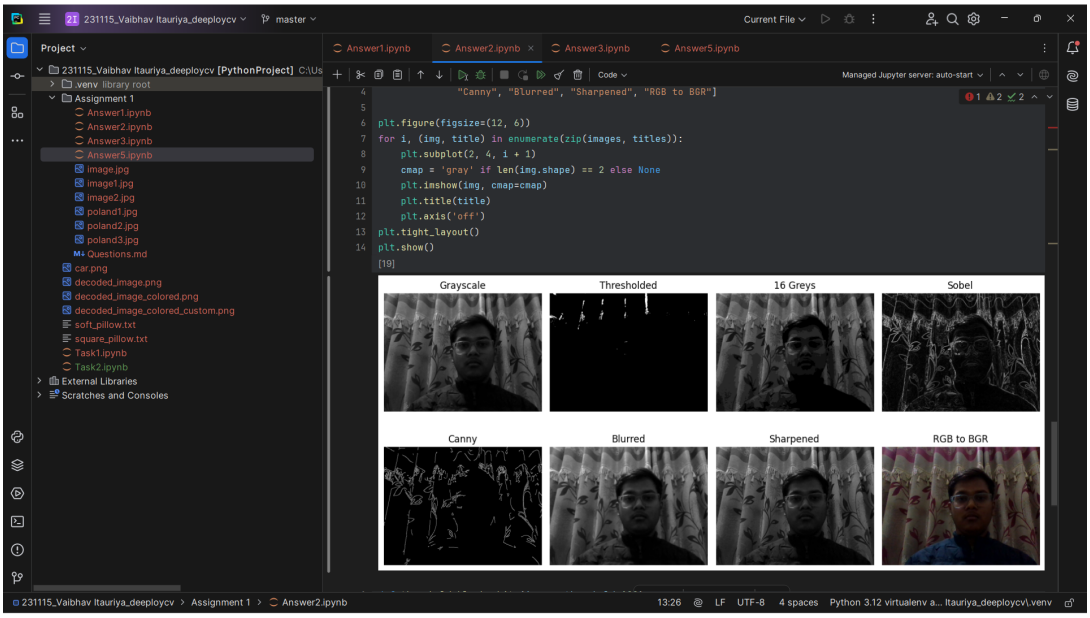

Figure 2: Output in Pycharm.

Solution Description

In this code, we perform several operations on images using OpenCV. The following image processing techniques are demonstrated:

- **Capture Image:** The `capture_image()` function uses the webcam (or another camera) to capture an image. It's done using `cv2.VideoCapture`, and the captured frame is returned for further processing.
- **Convert to Grayscale:** The `to_grayscale()` function converts the captured image to grayscale. This simplifies the image and makes it easier to process for other operations.
- **Thresholding:** The `threshold_black_white()` function changes the grayscale image into a black-and-white (binary) image. Pixels with values above a certain threshold become white, and others become black. This operation helps highlight certain features in an image by turning everything into just two colours (black and white).
- **Reduce to 16 Shades of Grey:** The `reduce_to_16_greys()` function reduces the grayscale image to just 16 shades of gray. This simplifies the original grayscale values, helping to create a less detailed picture. It's often used in cases where extreme detail isn't necessary.
- **Sobel Filter:** The `sobel_filter()` function applies the Sobel filter to detect edges in the image. It calculates the gradient of the image in both horizontal and vertical directions. The result is a new image that highlights the edges of objects.
- **Canny Edge Detection:** The `canny_edge()` function detects edges in the image using the Canny edge detection algorithm. This method is popular for detecting the boundaries of objects in an image.
- **Gaussian Blur:** The `gaussian_blur()` function applies a Gaussian blur to the image. This helps smooth the image and reduce noise. It's commonly used to prepare an image before applying edge detection or other operations.
- **Sharpen Image:** The `sharpen_image()` function enhances the edges and details of the image. The sharpening filter makes the image clearer by increasing the contrast along the edges.
- **RGB to BGR Conversion:** The `rgb_to_bgr()` function converts an image from the RGB colour format to the BGR colour format. This conversion is needed because OpenCV uses BGR (Blue-Green-Red) instead of RGB (Red-Green-Blue).

Displaying the Images: At the end of the code, all the images generated in these steps are displayed in a grid using `matplotlib`. Each image is shown with its respective title, and the axes are turned off for a cleaner view. The `tight_layout()` function ensures the images fit nicely in the window.

Link to Github Code: Click here.

Answer 3

This code aims to create a hybrid image by combining two images using high-pass and low-pass filters.

- **High-Pass Filter:** The high-pass filter is applied to the first image (`img1`). It highlights high-frequency components, such as edges and fine details, while suppressing low-frequency components.
- **Low-Pass Filter:** The low-pass filter is applied to the second image (`img2`). It smoothens the image using a **Gaussian blur**, removing high-frequency details and retaining low-frequency components like large smooth areas.
- **Combining Images:** After applying the high-pass and low-pass filters, the images are combined using a **weighted sum** with a ratio of 30% from the high-pass and 70% from the low-pass filtered images. This allows the smooth background from the second image to dominate while the fine details from the first image remain visible. I have used this by trying out different ratios on my own.

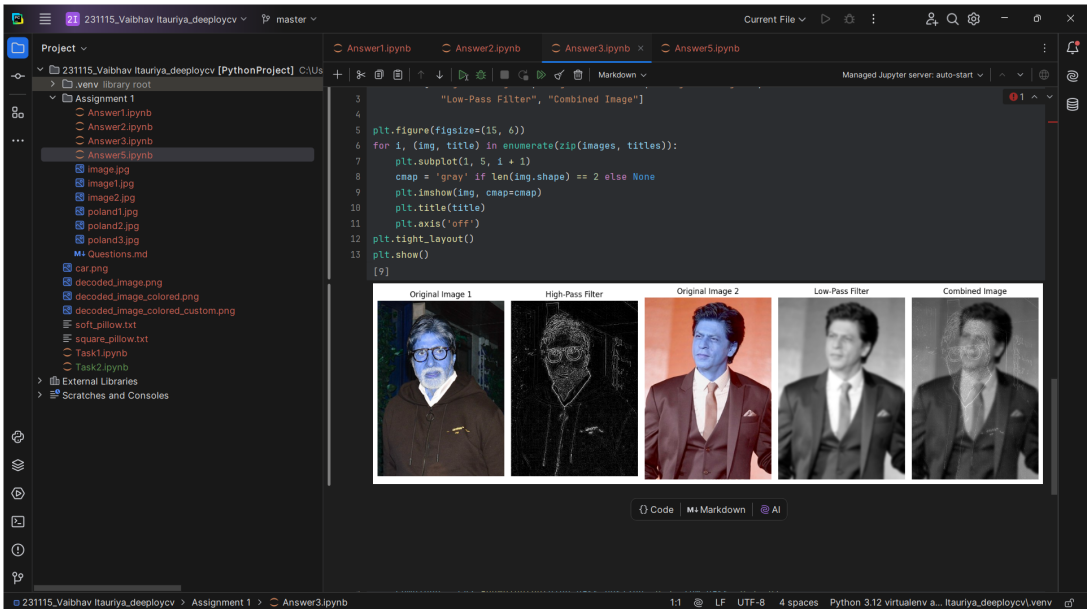

Figure 3: Sample Image of Output in Pycharm.

Solution Description

- `cv2.imread()`: Read two images (`img1` and `img2`).
- `cv2.cvtColor()`: Convert both images to grayscale.
- **High-pass filter** is applied to `img1` to extract fine details.
- **Low-pass filter** is applied to `img2` to extract smooth components.

- The two filtered images are combined using `cv2.addWeighted()` with the 30% high-pass and 70% low-pass ratio.
- The final hybrid image, along with the original and filtered images, is displayed using `matplotlib`.

Hybrid Image Effect:

- **Squinting Eyes:** When squinting, high-frequency details (edges) become less visible, making the low-pass filtered image (`img2`) more prominent. In this case, squinting will reveal **Sharukh Khan** because the smoother features of `img2` dominate.
- **Close-Up View:** When looking closely at the image, the high-frequency components (edges) from the high-pass filter (`img1`) become more evident. This will reveal **Amitabh Bachhan** when viewed up close due to the fine details retained from `img1`.

Link to Github Code: Click here.

Answer 4

As such, I have not found any such difficulties in this Python, Numpy and matplotlib colab file, and it helped me brush up on my Python knowledge and also helped me recall some concepts that I forgot after now using them for a longer time.

Answer 5

The task is to identify whether a given image represents the flag of Indonesia or Poland, using basic image processing techniques in Python with `NumPy` and `Pillow`. Both flags consist of two horizontal stripes, but with different color arrangements: Indonesia has a red top stripe and Poland has a white top stripe.

When tasked with detecting the flag region in an image, I initially tried a straightforward approach based on threshold and essential edge detection. However, I quickly ran into a challenge. This method wasn't compelling enough, particularly in images where the edges were not clearly defined or when the flag's colours blended with the background. Realising the limitations of this initial approach, I spent considerable time researching and experimenting with different techniques to improve accuracy. The process involved not only refining my edge detection method but also fine-tuning the algorithm to reliably detect the flag's boundaries and differentiate between similar-looking flags like Poland and Indonesia.

- I first implemented a basic thresholding technique to convert the image to binary form and then used simple edge detection to find the flag's location. However, the results were inconsistent, and many flags were not detected correctly.
- I then tried to improve edge detection, eventually learning that a Sobel filter could be the key to detecting more refined edges.

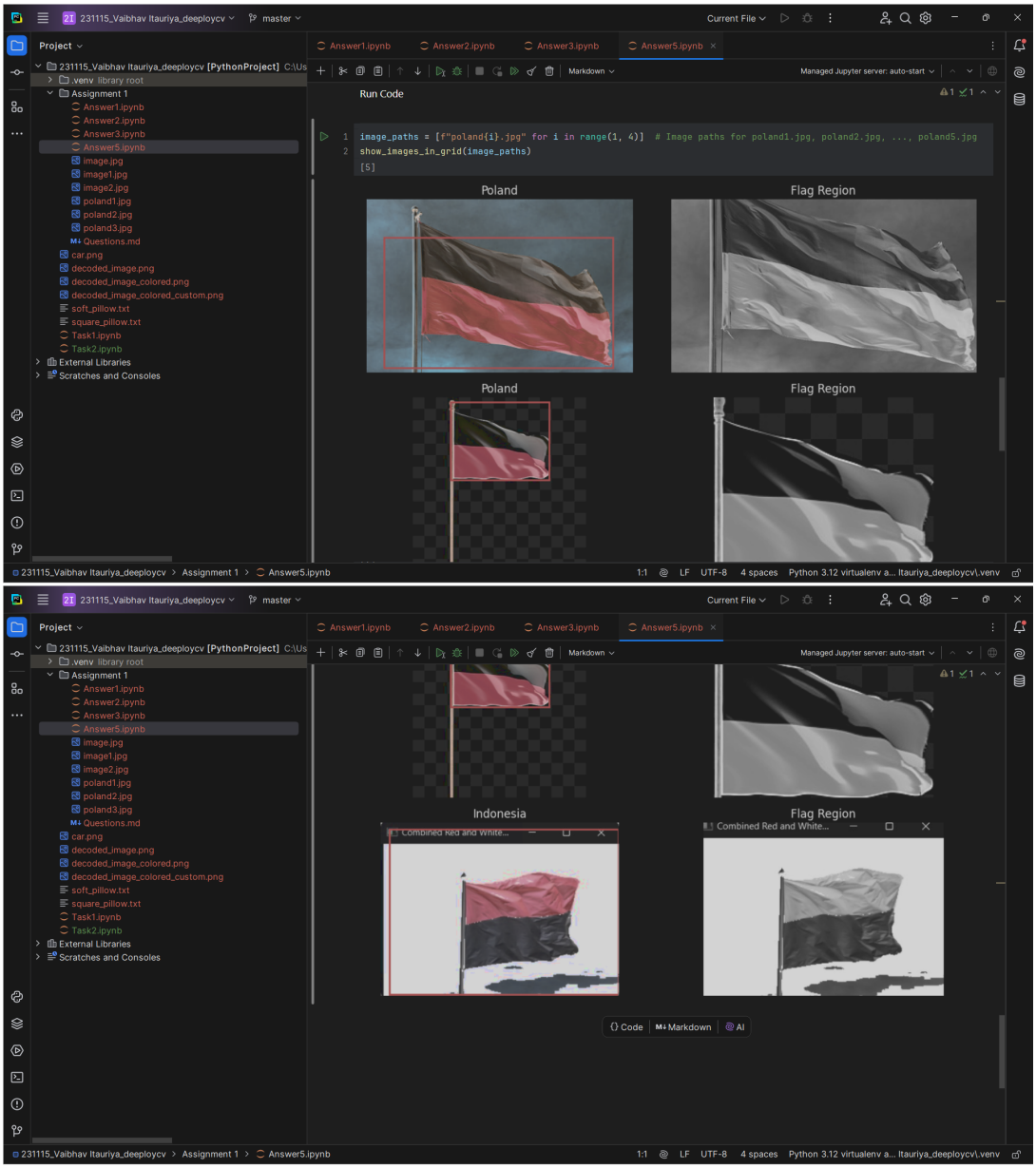

Figure 4: Sample Image of Output in Pycharm.

Solution Description

- **Thresholding and Edge Detection:** Initially, the image was converted to grayscale and thresholding was applied to create a binary image. This allowed me to distinguish the flag from the background. I then applied a difference-based edge detection

approach to identify vertical and horizontal edges, but this approach was not precise enough for detecting flags in varied scenarios.

- **Sobel Filter:** The breakthrough came with the use of the Sobel filter, a more advanced edge detection technique that calculates the gradient of the image. This helped identify sharper transitions (edges), making it easier to detect flag boundaries, especially when the colors were similar to the background or the flag was not centrally located.
- **Region Extraction and Flag Classification:** Once the edges were detected, I focused on extracting the flag region by identifying the areas with the most significant edge response. The cropped flag region was resized to a standard size (100x60 pixels) and divided into top and bottom halves. By comparing the average intensity of each half, the flag was classified:
  - If the top is lighter and the bottom is darker, the flag belongs to **Poland**.
  - If the top is darker and the bottom is lighter, the flag belongs to **Indonesia**.
- **Image Display and Visualization:** The results were displayed using Matplotlib, showing both the original image with the detected regions and the cropped flag regions. This helped to visually confirm the flag classification.

Link to Github Code: Click here.