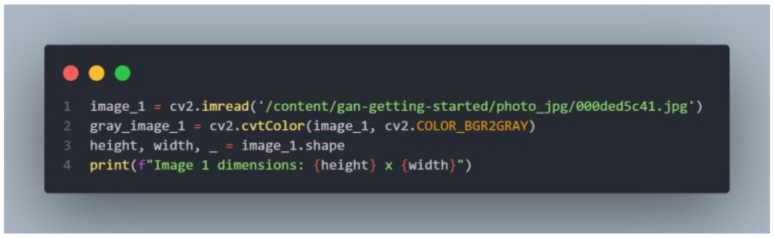
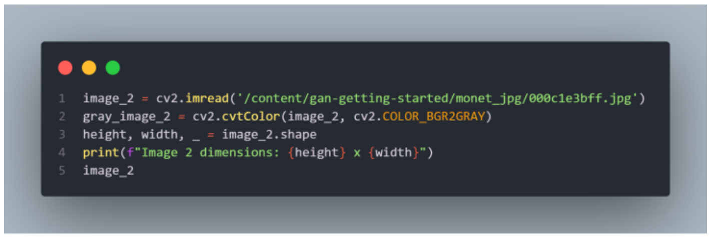
Question 1 Exploratory Data Analysis (EDA)

In this project, we aim to generate Monet-style images from photographs using Generative Adversarial Networks (GANs). To ensure the dataset is ready for training and the modeling process, an in-depth Exploratory Data Analysis (EDA) was conducted. The dataset consists of two main categories of images: Monet paintings and photographs, each of which requires careful analysis before building a machine learning model.

1. **Image Dimensions (Height x Width)**
   The first step in the analysis was to inspect the dimensions of the images. Consistency in image size is essential for training machine learning models, particularly GANs. The dimensions of the images were extracted for both Monet paintings and photos, confirming that the images are 256x256 pixels, as expected for this project.



```python
image_1 = cv2.imread('/content/gan-getting-started/photo_jpg/000ded5c41.jpg')
gray_image_1 = cv2.cvtColor(image_1, cv2.COLOR_BGR2GRAY)
height, width, _ = image_1.shape
print(f"Image 1 dimensions: {height} x {width}")
```

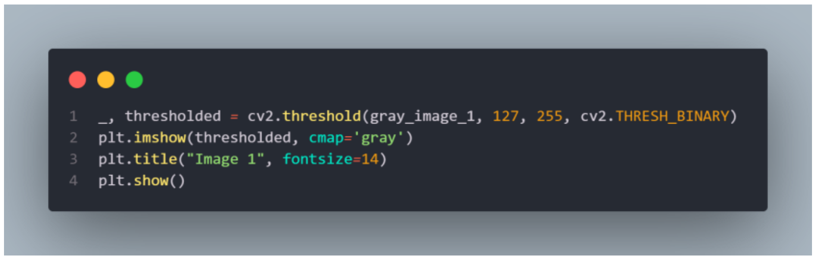


```python
image_2 = cv2.imread('/content/gan-getting-started/monet_jpg/000c1e3bff.jpg')
gray_image_2 = cv2.cvtColor(image_2, cv2.COLOR_BGR2GRAY)
height, width, _ = image_2.shape
print(f"Image 2 dimensions: {height} x {width}")
image_2
```

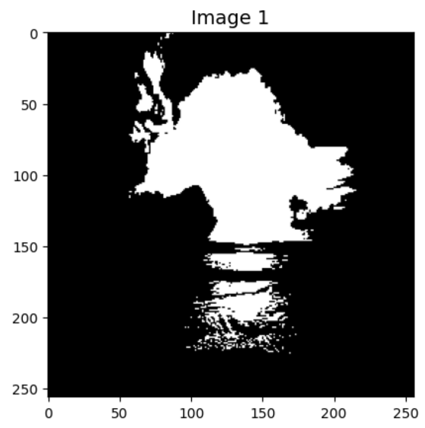
   Result: Both the Monet and photo images have the correct size of 256x256 pixels, ensuring they are ready for model training.

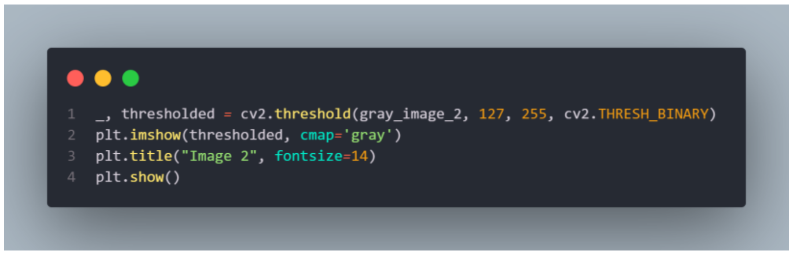
2. **Threshold Segmentation**
   The next analysis applied threshold segmentation to both images to isolate foreground elements from the background. By converting grayscale pixel values into binary form using a fixed threshold, we were able to simplify the visual structure of each image. This method helps to enhance object boundaries, making it easier to identify regions of interest. Threshold segmentation is particularly valuable in preprocessing steps for tasks

such as object detection, contour analysis, and image classification, especially when the contrast between objects and background is significant.

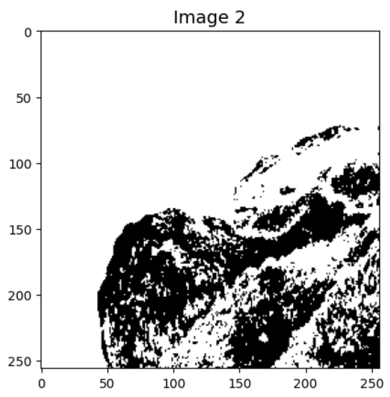


```python
_, thresholded = cv2.threshold(gray_image_1, 127, 255, cv2.THRESH_BINARY)
plt.imshow(thresholded, cmap='gray')
plt.title("Image 1", fontsize=14)
plt.show()
```





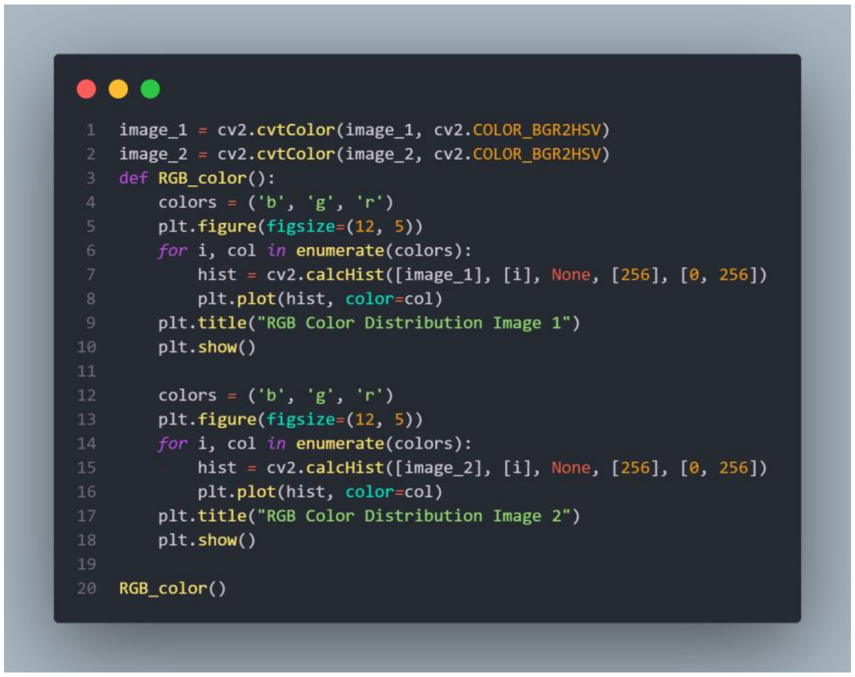
```python
_, thresholded = cv2.threshold(gray_image_2, 127, 255, cv2.THRESH_BINARY)
plt.imshow(thresholded, cmap='gray')
plt.title("Image 2", fontsize=14)
plt.show()
```



Result : shows a clear separation between objects and background based on pixel intensity. In Image 1, the dominant bright area in the center of the image indicates that many parts have high intensity values, while the dark parts indicate background or details with low intensity. In Image 2 shows a more diffuse and complex segmentation, depicting surface textures such as rocks or natural landscapes that have diverse light-dark distributions.

3. RGB Color Distribution

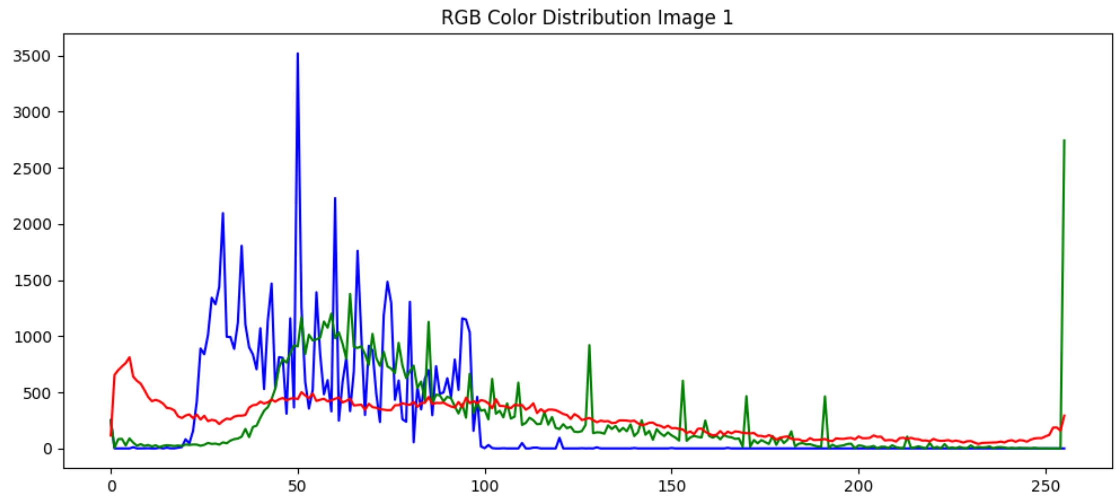
The next analysis focused on the RGB color distribution for the two images. By plotting histograms for each color channel (Red, Green, Blue), we were able to understand the pixel intensity distribution for both images. This step is crucial for assessing the balance of color channels and ensuring the images' color properties are suitable for style transfer tasks.

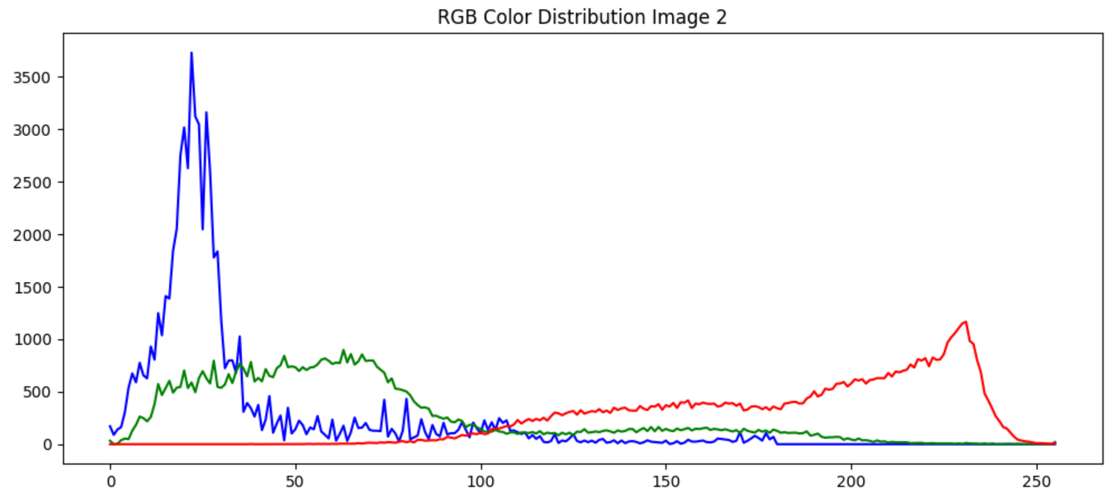


```python
image_1 = cv2.cvtColor(image_1, cv2.COLOR_BGR2HSV)
image_2 = cv2.cvtColor(image_2, cv2.COLOR_BGR2HSV)
def RGB_color():
    colors = ('b', 'g', 'r')
    plt.figure(figsize=(12, 5))
    for i, col in enumerate(colors):
        hist = cv2.calcHist([image_1], [i], None, [256], [0, 256])
        plt.plot(hist, color=col)
    plt.title("RGB Color Distribution Image 1")
    plt.show()

    colors = ('b', 'g', 'r')
    plt.figure(figsize=(12, 5))
    for i, col in enumerate(colors):
        hist = cv2.calcHist([image_2], [i], None, [256], [0, 256])
        plt.plot(hist, color=col)
    plt.title("RGB Color Distribution Image 2")
    plt.show()

RGB_color()
```

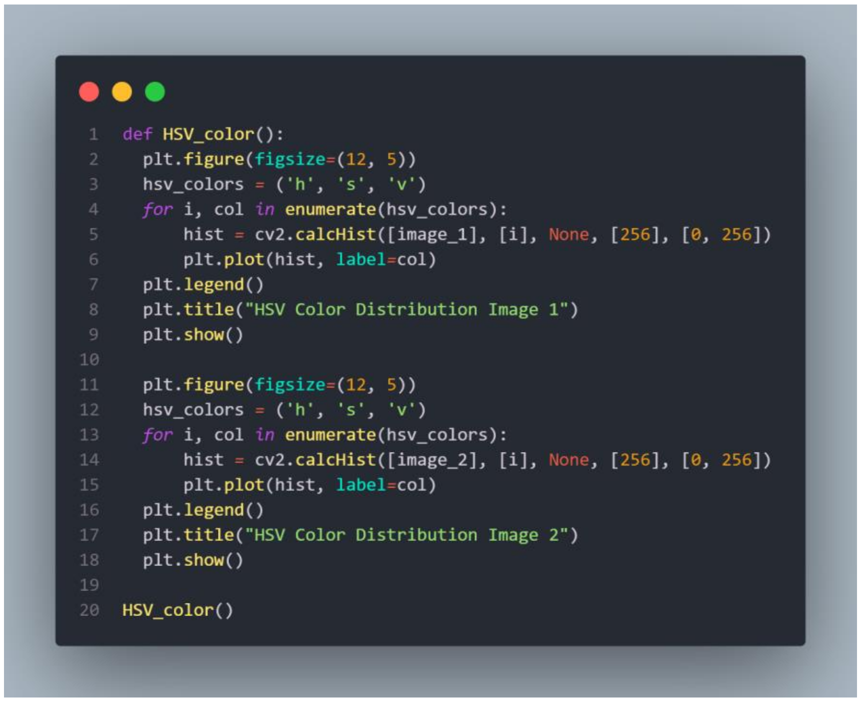




Result: The histograms for each of the color channels (Red, Green, Blue) were visualized. This analysis helps identify if any particular color channel is overrepresented or underrepresented in the images, providing insights into their overall color distribution.

4. **HSV Color Distribution**

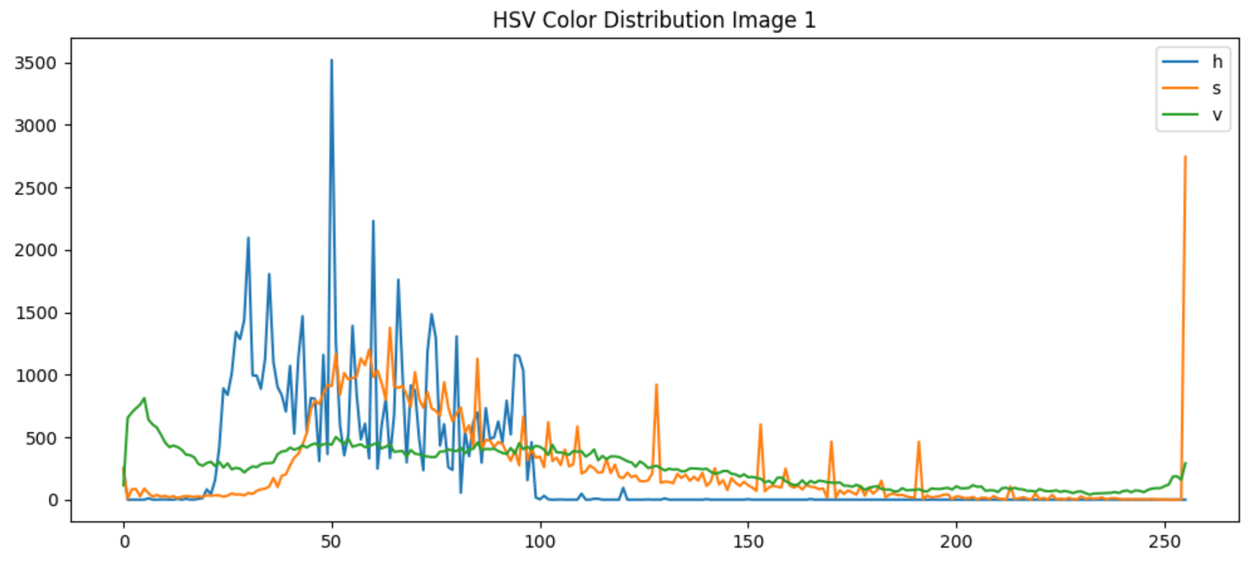
In addition to the RGB color distribution, we analyzed the HSV (Hue, Saturation, Value) color distribution. The HSV model is often more effective for understanding color relationships in images, and it is especially useful for style transfer tasks like this one.

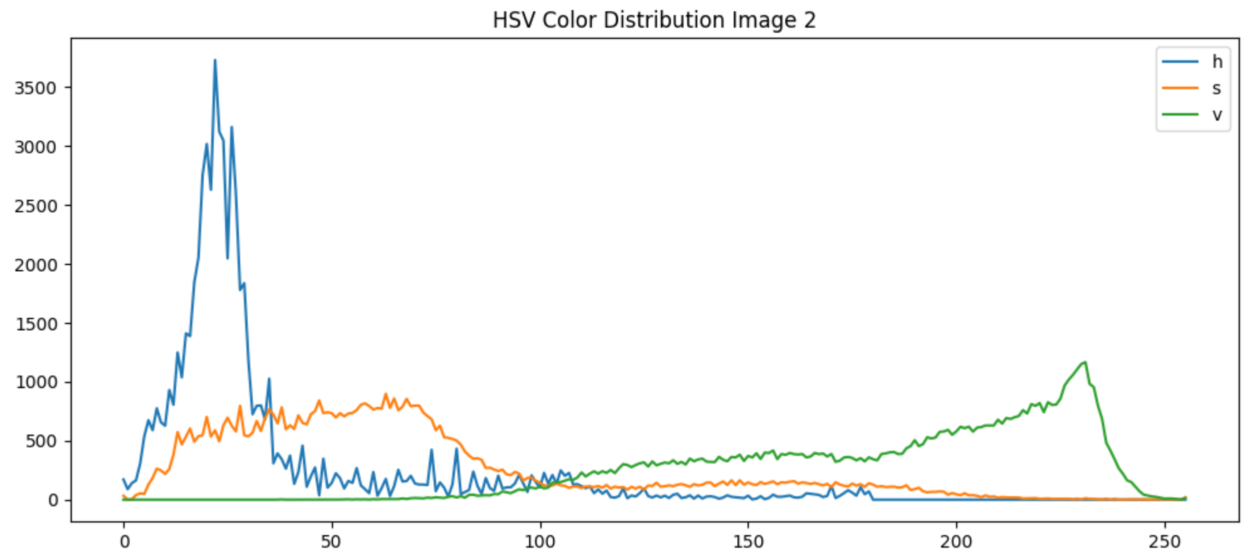


```python
def HSV_color():
  plt.figure(figsize=(12, 5))
  hsv_colors = ('h', 's', 'v')
  for i, col in enumerate(hsv_colors):
      hist = cv2.calcHist([image_1], [i], None, [256], [0, 256])
      plt.plot(hist, label=col)
  plt.legend()
  plt.title("HSV Color Distribution Image 1")
  plt.show()

  plt.figure(figsize=(12, 5))
  hsv_colors = ('h', 's', 'v')
  for i, col in enumerate(hsv_colors):
      hist = cv2.calcHist([image_2], [i], None, [256], [0, 256])
      plt.plot(hist, label=col)
  plt.legend()
  plt.title("HSV Color Distribution Image 2")
  plt.show()

HSV_color()
```

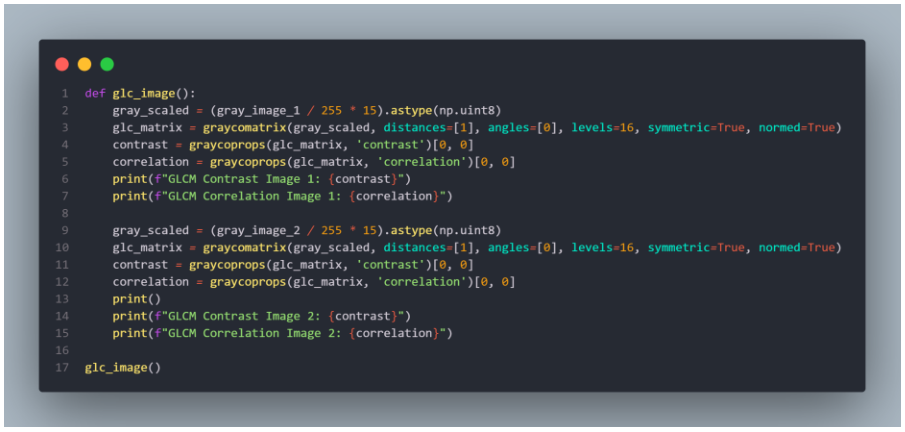




Result: The HSV color distributions were plotted for both images. These plots show the distribution of Hue, Saturation, and Value, which are essential when considering how the color will be transferred from photographs to Monet-style images.

Question 2 Statistical Analysis

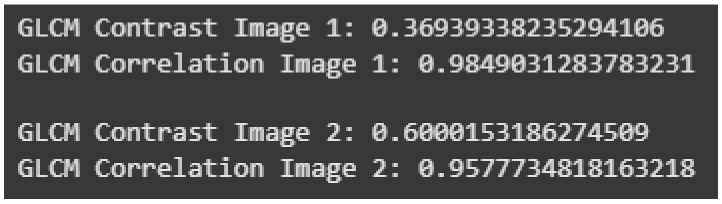
1. **GLCM (Gray Level Co-occurrence Matrix) - Contrast and Correlation**
   Next, we used the Gray Level Co-occurrence Matrix (GLCM) to compute contrast and correlation between pixels in the images. GLCM is a powerful technique for texture analysis, and it helps quantify how pixel intensities relate to each other within the image.



```python
def glc_image():
    gray_scaled = (gray_image_1 / 255 * 15).astype(np.uint8)
    glc_matrix = graycomatrix(gray_scaled, distances=[1], angles=[0], levels=16, symmetric=True, normed=True)
    contrast = graycoprops(glc_matrix, 'contrast')[0, 0]
    correlation = graycoprops(glc_matrix, 'correlation')[0, 0]
    print(f"GLCM Contrast Image 1: {contrast}")
    print(f"GLCM Correlation Image 1: {correlation}")

    gray_scaled = (gray_image_2 / 255 * 15).astype(np.uint8)
    glc_matrix = graycomatrix(gray_scaled, distances=[1], angles=[0], levels=16, symmetric=True, normed=True)
    contrast = graycoprops(glc_matrix, 'contrast')[0, 0]
    correlation = graycoprops(glc_matrix, 'correlation')[0, 0]
    print()
    print(f"GLCM Contrast Image 2: {contrast}")
    print(f"GLCM Correlation Image 2: {correlation}")

glc_image()
```

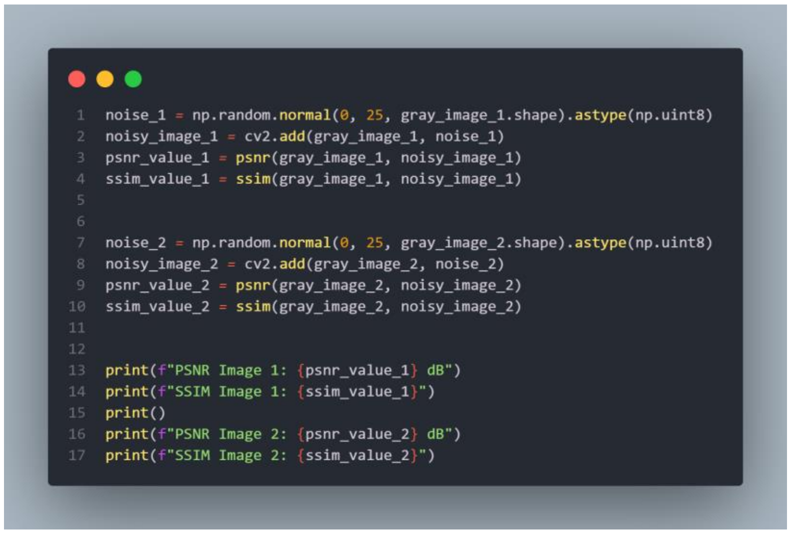


```
GLCM Contrast Image 1: 0.36939338235294106
GLCM Correlation Image 1: 0.9849031283783231

GLCM Contrast Image 2: 0.6000153186274509
GLCM Correlation Image 2: 0.9577734818163218
```

Result: The contrast and correlation values were computed for both images. These metrics provide insight into the textural properties of the images, which is important when understanding how GANs may generate or modify textures in the final Monet-style output

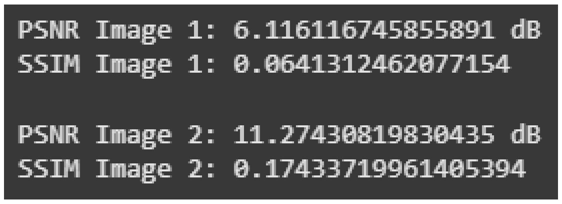
2. **PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index)**
   To assess the impact of noise on image quality, we added random noise to both images and calculated the Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM). These metrics are commonly used to evaluate image quality and similarity.



```python
noise_1 = np.random.normal(0, 25, gray_image_1.shape).astype(np.uint8)
noisy_image_1 = cv2.add(gray_image_1, noise_1)
psnr_value_1 = psnr(gray_image_1, noisy_image_1)
ssim_value_1 = ssim(gray_image_1, noisy_image_1)


noise_2 = np.random.normal(0, 25, gray_image_2.shape).astype(np.uint8)
noisy_image_2 = cv2.add(gray_image_2, noise_2)
psnr_value_2 = psnr(gray_image_2, noisy_image_2)
ssim_value_2 = ssim(gray_image_2, noisy_image_2)


print(f"PSNR Image 1: {psnr_value_1} dB")
print(f"SSIM Image 1: {ssim_value_1}")
print()
print(f"PSNR Image 2: {psnr_value_2} dB")
print(f"SSIM Image 2: {ssim_value_2}")
```



```
PSNR Image 1: 6.116116745855891 dB
SSIM Image 1: 0.0641312462077154

PSNR Image 2: 11.27430819830435 dB
SSIM Image 2: 0.17433719961405394
```

Result: The PSNR and SSIM values were calculated for both images, showing how well the images maintain quality despite added noise. PSNR values indicate the difference between the original and noisy image, with higher values suggesting better quality, while SSIM values provide a similarity score between the two images.

Question 3 Cycle-Consistent Generative Adversarial Networks (Cycle GANs)

CycleGAN (Cycle-Consistent Generative Adversarial Network) is a Generative Adversarial Network (GAN) architecture used to perform image-to-image translation without the need for annotated data pairs. This is very useful in cases where we want to transform images from domain A to domain B and vice versa, but do not have directly corresponding image pairs.CycleGAN consists of two sets of Generator and Discriminator, one each for domains A → B and B → A.

Components of Cycle GAN:

1. **Generator (G)**:
    - **Purpose**
        - **Generator A:** Changing an image from domain A (landscape photo) to domain B (Monet painting)
        - **Generator B:** Changing an image from domain B (Monet painting) to domain A (landscape photo)

    - **How It Works:** In CycleGAN, the generator receives input in the form of an image from a domain, such as a photo, instead of a noise vector as in classical GANs. This image is then processed through several convolutional layers, residual blocks, and transposed convolutions to generate a new image in the target domain. This process is designed so that the generator's output not only looks realistic but also has the visual style typical of the target domain, such as Monet's brushstrokes. In this way, the generator is able to perform end-to-end style transfer between domains without requiring parallel image pairs.

2. **Discriminator (D)**:
    - **Purpose:**
        - **Discriminator A :** Distinguish whether the image in domain B (Monet painting) is original or the result of Generator A.
        - **Discriminator B :** Distinguish the original domain A image (landscape photo) from the result of Generator B.

    - **How It Works:** Discriminator is a convolutional neural network that functions to evaluate the authenticity of an image by producing a certain score. When receiving an original image from the dataset, the discriminator is expected to give a high score (close to 1), while for images generated by the generator, the score given should be low (close to 0). During the training process, the discriminator continues to learn to distinguish between real and fake images by recognizing patterns, textures, and visual structures that are typical of each. The ultimate goal is for the discriminator to become more adept at distinguishing which images are authentic and which are just the result of generator engineering.

Training Process of Cycle GAN:

- **Data Preparation:** The training process begins by preparing two types of datasets, namely images from the Monet domain (paintings) and the Photo domain (real photos). All images are resized to 128×128 pixels and normalized into the range of [-1, 1] to suit the model's input needs. Next, these images are fed into the tf.data.Dataset object, shuffled, divided into batches, and paired between Monet images and Photo images to be trained simultaneously.

- **Transformation and Cycle Consistency:** At each training step, the model performs a back-and-forth transformation between domains. For example, a Photo image is first transformed into a Monet image by the Monet generator, then the result is converted back to Photo using the Photo generator. This process is called a forward cycle, and vice versa from Monet to Photo and back to Monet is called a backward cycle.

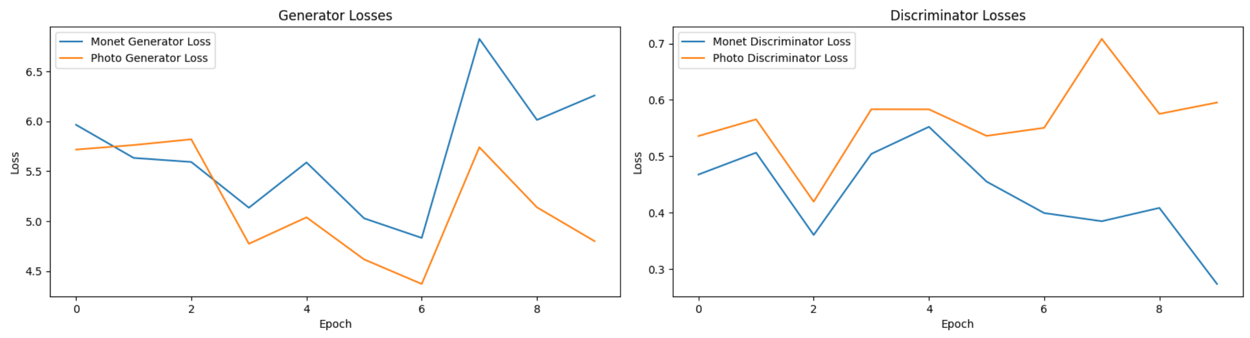
- **Identity Mapping:** To stabilize the training and maintain style consistency, an identity mapping mechanism is also applied, where images from a domain are fed into a generator from the same domain and are expected to remain unchanged. For example, a Monet image is fed into the Monet generator and the output is expected to remain the same.

- **Discriminator Role:** Two discriminators are trained simultaneously, namely for the Monet and Photo domains. Each is tasked with distinguishing the original image from the generator image. The Monet discriminator evaluates whether a Monet image is real or fake (generator output), as does the Photo discriminator. Meanwhile, the generator is trained to be able to deceive the discriminator, producing images that are as close as possible to the original image to be considered “real”.

- **Loss Calculation and Optimization:** After all predictions are made, the model calculates four main types of losses, namely generator loss (the ability to deceive the discriminator), discriminator loss (the ability to distinguish real from fake), cycle loss (to maintain content consistency in the transformation cycle), and identity loss (to maintain stability when the input does not need to be changed).

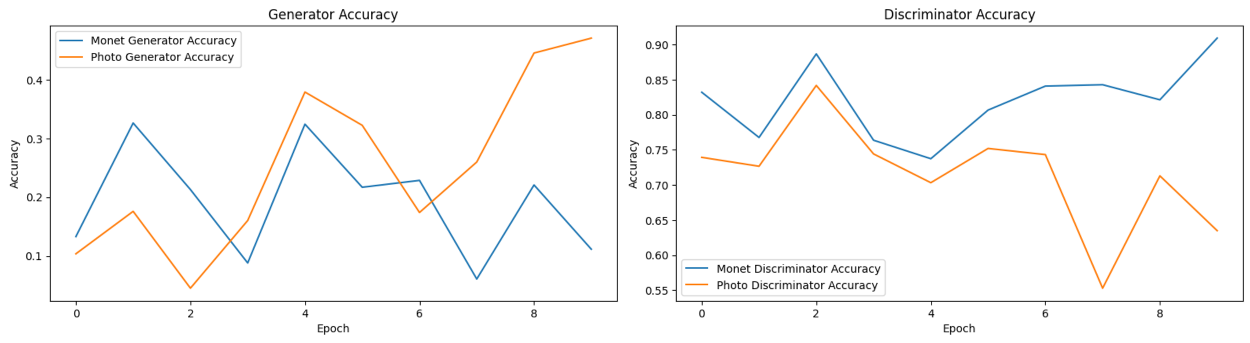
- **Performance Evaluation and Iterative Training:** Throughout the training process, the model records various performance metrics such as the accuracy of the generator in generating “convincing” images, and the accuracy of the discriminator in distinguishing real and fake images. This process is repeated for several epochs, and at each epoch, the model performs a number of training steps (steps_per_epoch) calculated based on the total number of images and the batch size.

Result Model



Generator Losses : In the Generator Losses graph, it can be seen that the Monet Generator (blue line) has a higher and fluctuating loss value throughout the training process. This indicates that generating Monet-style images is a more complex task, so the model has difficulty achieving stable convergence. In contrast, the Photo Generator (orange line) shows a lower and more stable loss, indicating that the model has an easier time learning the conversion from Monet to photo style.

Discriminator Losses : The discriminator loss graph shows an increase in loss in the middle of the epoch, likely due to the generator's ability to produce convincing fake images. The Monet Discriminator shows better progress by gradually decreasing its loss at the end of training, indicating an increase in its ability to distinguish between original images and generator results. The Photo Discriminator, although showing some stability, does not experience a significant decrease in loss like the Monet Discriminator.

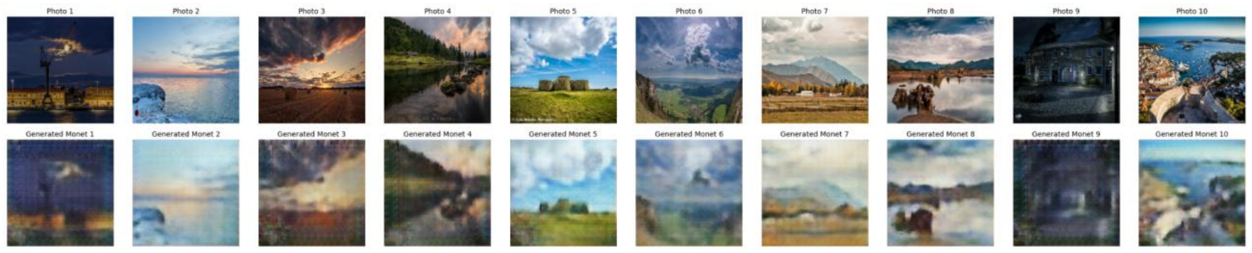


Generator Accuracy : The Photo Generator experiences a significant increase in accuracy throughout the training process. This indicates that the generator is increasingly able to produce images that are difficult for the Photo Discriminator to distinguish, indicating its effectiveness in mimicking the photo domain. In contrast, the Monet Generator shows sharp fluctuations in accuracy and does not show a steady trend of improvement. This condition strengthens the hypothesis that the conversion to the Monet style is a greater challenge for the model, making its learning process more inconsistent.

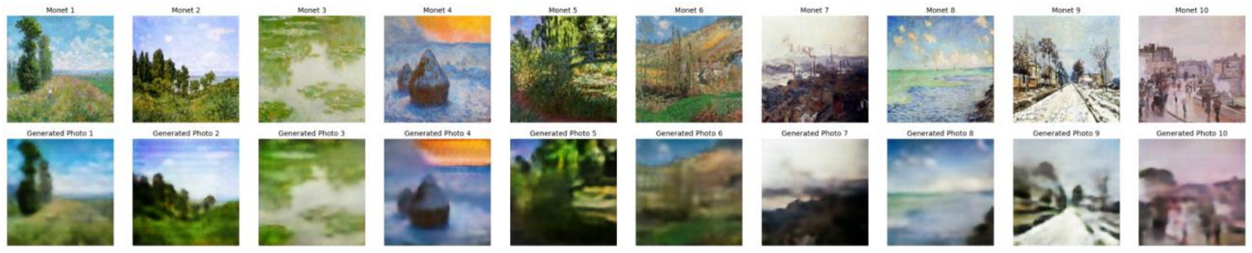
Discriminator Accuracy : Discriminator Accuracy graph shows that both discriminators have high performance with accuracy between 0.7 to 0.8. The Photo Discriminator accuracy decreases towards the end of training, likely because the generator starts generating more realistic images. In contrast, the Monet Discriminator maintains stable accuracy, reflecting its consistency in recognizing fake images. This is also consistent with the fluctuation of loss in the Monet Generator indicating a greater challenge in fooling the Monet Discriminator.

The image below shows the results of the style transformation from a landscape photo to a Monet-style painting using the CycleGAN model.

Photo → Monet Painting



Monet Painting → Photo


Monet 1
Monet 2
Monet 3
Monet 4
Monet 5
Monet 6
Monet 7
Monet 8
Monet 9
Monet 10
Generated Photo 1
Generated Photo 2
Generated Photo 3
Generated Photo 4
Generated Photo 5
Generated Photo 6
Generated Photo 7
Generated Photo 8
Generated Photo 9
Generated Photo 10