I’m Something of a Painter 🎨 Myself

> **“Every artist dips his brush in his own soul, and paints his own nature into his pictures.”**
> *- Henry Ward Beecher*

Use GANs to create art - will you be the next Monet?

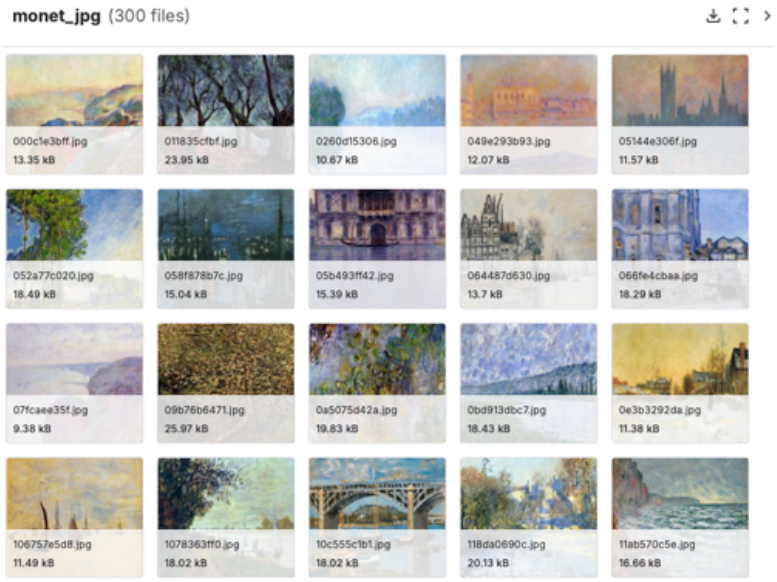
1. Project Overview

(1) Description of the project

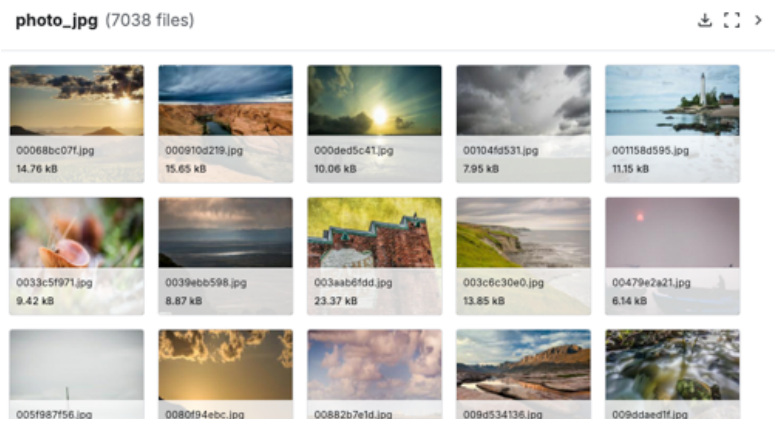
- In this project, generate images in the style of Monet from photo image like below. Monet-style art can be created from scratch using GAN architectures that generator is trained using a discriminator.

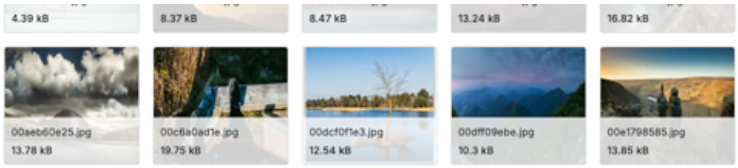
(2) Input Data

- monet_jpg - 300 Monet paintings sized 256x256 in JPEG format
- monet_tfrec - 300 Monet paintings sized 256x256 in TFRecord format



- photo_jpg - 7028 photos sized 256x256 in JPEG format
- photo_tfrec - 7028 photos sized 256x256 in TFRecord format



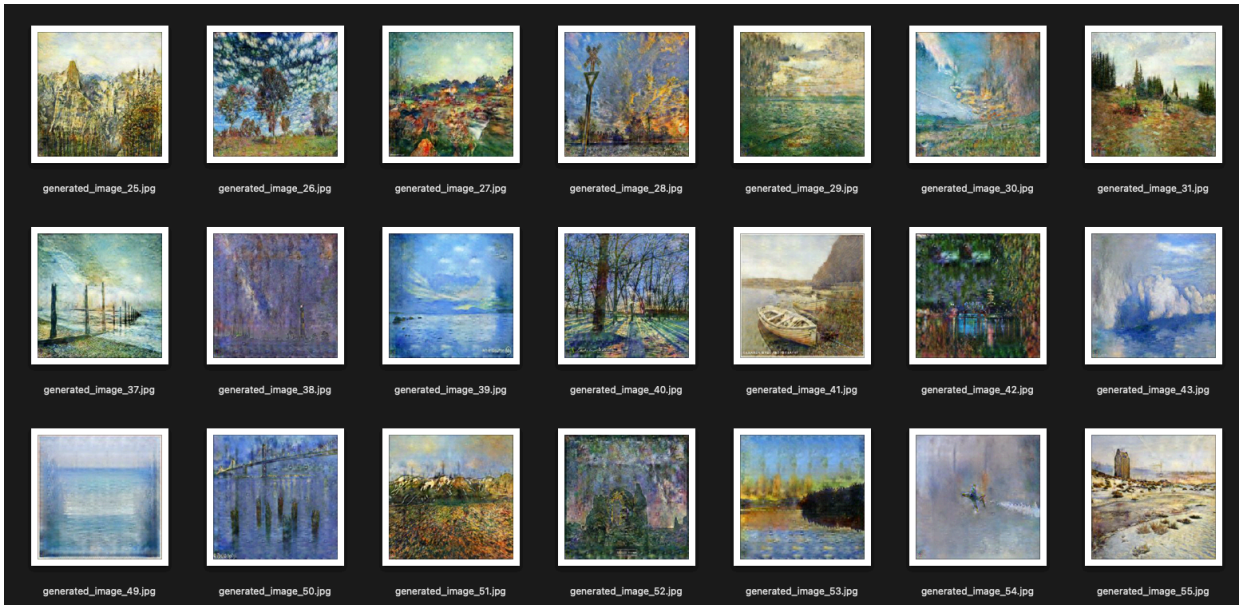


(3) Output Data format

- Output must be called images.zip and contain 7,000-10,000 images sized 256 x 256.

(4) Preview of this Project Output

- Below is a part of my ouput image.



2. Initial Approaches & Changing Architecture

- As I was looking for something exciting GAN experiment, initially considered **CycleGAN + Diffusion model**. This fusion seemed to promise to combine the strengths of both approaches, delivering fast generation, high-quality outputs, and enhanced style transfer capabilities.
- Even though I couldn't sure about the output, but definetely it might be interesting trial. For this advanture, I had to choose one among three types of hybrid models like below:

  1) Diffusion Initialization Using GAN (GAN Prior Diffusion): A GAN (e.g., StyleGAN) generates an initial image, which a Diffusion model enhances its details to accelerates generation while producing high-quality, refined images. 2) GAN Discriminator Using Diffusion (Diffusion-Guided GAN): The GAN's discriminator leverages a Diffusion model to provide richer, more detailed feedback not just simple "real vs. fake" judgments, acting as a contributor to image quality improvement. Enhances training stability and mitigates mode collapse—a common GAN limitation. 3) ✅ **Diffusion as a Noise Generator for GAN (Diffusion-GAN Hybrid)**: The Diffusion model generates structured noise (e.g., latent vectors) as

input for the GAN, replacing traditional Gaussian noise. Provide higher-quality outputs with greater diversity and realism. This is my My selection for the initial GAN trial.

💡 Fun Fact 💡💡💡

- Before training the **Diffusion-GAN Hybrid** model, curiosity led me to investigate whether similar ideas had been implemented before. I posed two questions:

  > 🤔 *“Has anyone else combined CycleGAN and Diffusion in this way?”*
  > 🤔 *“Could I be the first to propose this hybrid model (despite my inner skepticism) ?”*
  >
  > 🤓.....Of course not. Just found the paper with very similar concept in the below paper:

📚 [**A Latent Space of Stochastic Diffusion Models for Zero-Shot Image Editing and Guidance** (ICCV 2023)](**https://openaccess.thecvf.com/content/ICCV2023/papers/Wu_A_Latent_Space_of_Stochastic_Diffusion_Mo Shot_Image_ICCV_2023_paper.pdf)

> *Why these are similar?*
>
> ```
>    - Both projects utilize Diffusion models for GAN in image generati
> on and style transfer.
>    - The zero-shot editing in the paper aligns with my application of
> Monet-style transfer without paired data and utilizing diffusion mode
> l's prompt.
>    - Latent space manipulation for style editing mirrors my approach
> to generating and refining artistic outputs.
> ```

3. Lessons from Diffusion-GAN Hybrid

Building and training "Diffusion-GAN Hybrid" was a fun experiment, but FUN did not guarantee high performance.

(1) Using diffusion model just for generating noise and using text prompt required considerable time and GPU resources. And is it really worth of using expensive diffusion noise?

> *(my inner voice)* **"Adding noise itself is a great method to use in order to improve GAN output quality, but GAN and Diffusion Hybrid model is somehow expensive than the output quality."**

(2) The result MiFID (Memorization-informed FID) score and leader board ranking was not satisfied. How to improve output quality?

> *(Looking back I don't understand my obsession but...)* Definately I was so obsessed with **all kinds of 'NOISE' (Gaussian Noise, Diffusion Noise, Just Random Noise...)**, and reapeating hybrid model experiments, then suddenly I had a half way moment with new ideas...
>
> *(my inner voice)* The essnetial part of the GAN architecture is not "Noise itself".
>
> **Let's change my focus from Diffusion-GAN Hybrid approach to the process among Generator and Discriminator which are the core part of GAN".** And I started my new experiment with new objective function.

4. Finding New Objective Function for New Experiment

4.1 Cycle Consistency & Weight

- After the previous lessons, looking for new ideas. Wanted to make up for small knowledge, researched some papers, then found some interesting papers and objective function that seemed to be certainly worth a trial.
- Especially I liked the nobel approaches of **'cycle consistency on discriminator CNN feature level'** and **'weighting cycle consistency by the quality of generated images' in CycleGAN**.
- Theses concepts convinced it can maximize the performance of GAN. Later, actually it turned out that these allowed to evaluate more nuanced image quality, and to strengthen the consistency throughout the generation process, leading to more realistic and compelling outputs.

📚 **CycleGAN with Better Cycles** (arXiv:2408.15374) (https://arxiv.org/pdf/2408.15374)

> Better Cycle with combination of GAN loss and Cycle Consistency loss:
>
> - The overall objective function is optimized by combining the **GAN loss** and the **Cycle-Consistency loss**. This function encourages the two generators G and F to learn the transformation of each other's domain, X and Y.
> - 🎯 **Full Objective Function**:

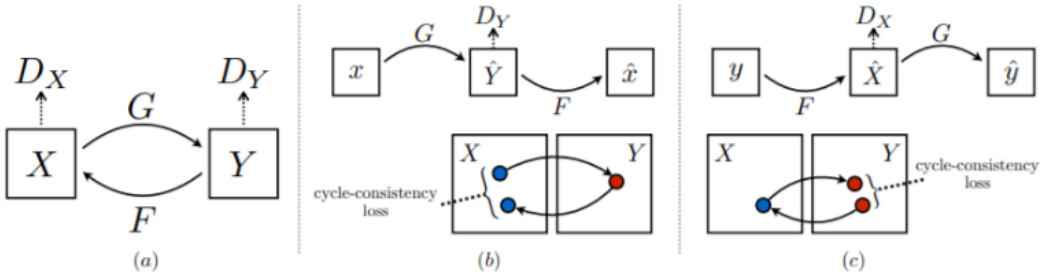
$$L(G; F; D_X; D_Y; t) =$$

$$L_{GAN}(G; D_Y; X; Y) + L_{GAN}(F; D_X; Y; X)$$

$$+\lambda_t \tilde{L}cyc(G; F; D_X; X; \gamma_t) + \lambda_t \tilde{L}cyc(F; G; D_Y; Y; \gamma_t)$$

> *Putting the above together, the full objective at epoch t is suggested that λ_t linearly decrease to a small value, and γ_t linearly increase to a value close to 1.*

- **Forward Cycle Consistency & Backward Cycle Consistency**:



- **Cycle consistency on discriminator CNN feature level**:
  - **Cycle consistency loss** is clever idea that encourages the generator to transform the input image and then restore it back to the original image. This ensures that the transformation between two domains is consistent. For example, an image from domain X transformed to domain Y and then transformed back to domain X should be similar to the original image.
    - Cycle consistency loss is also defined by combining **both "CNN feature-level" and "pixel-level" losses**, which allows the transformed image to maintain the structural features of the original image.
    - The GAN loss encourages the generator to generate realistic images to fool the discriminator, while the cycle-consistency loss ensures that the transformed images are consistent with the original images. By combining these two losses, the model can generate better quality images and learn to make the transformation between the two domains consistent.

$$L_{cyc}(G, F, X) = E_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1]$$

- **Weight decay & Weight cycle consistency by quality of generated image:**
  - Enforcing cycle consistency on cycles where generated images are not realistic actually hinders training. To solve this problems, added weight cycle consistency loss by the quality of generated images, which obtained using the discriminators' outputs.
  - Weight adjustment λ_t **is a weight** that is dynamically adjusted during the training process. It **starts with a low value initially and gradually increases** as training progresses. This helps the generator focus on generating realistic images in the early training stages, and later emphasizes the cycle consistency loss more, which helps increase the consistency

of the transformation.

$$\tilde{L}_{cyc}(G, F, D_X, X, \gamma) =$$

$$E_{x \sim p_{data}(x)} \left[D_X(x) \left(\gamma \| f_{D_X}(F(G(x))) - f_{D_X}(x) \|_1 + (1 - \gamma) \| F(G(x)) - x \|_1 \right) \right]$$

[Comparison]
||**Original CycleGAN** || **CycleGAN w/all modifications** || **CycleGAN w/all modifications except weighting cycle consistency**||

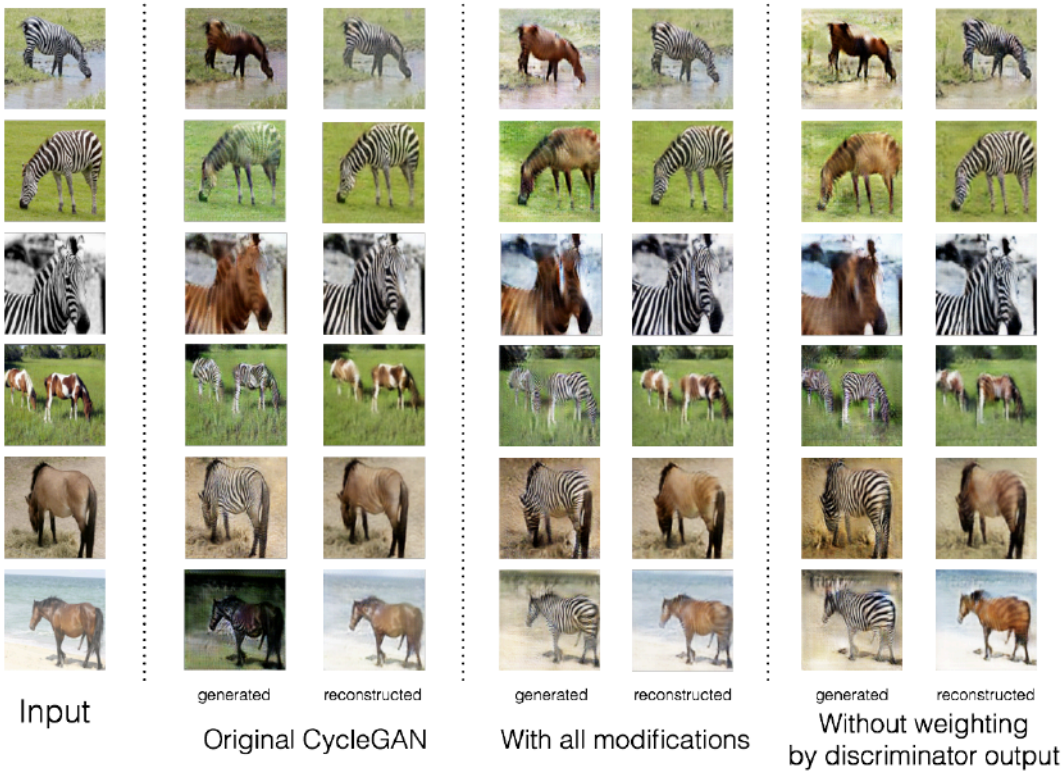


Figure 8: Comparison among original CycleGAN, CycleGAN with proposed modifications, and CycleGAN with proposed modifications except weighting cycle consistency by discriminator output on `horse2zebra` dataset. These images are hand picked from training set.

- **Adding Noise**:
  - However, in this paper, using random noise channel to create more diverse outputs cannot maintain model consistency in the process. Fortunately, I have just went though several noise experiment in the CycleGAN and Diffusion Hybrid architecher, and am pretty sure about adding noise can improve image quality and model consistency.

4.2 Differentiable Augmentation for GANs

- I didn't mention before but another lessons from my previous Diffusion-GAN Hybrid experiment was **necessity of data augumentation**. Due to unbalanced training dataset (Monet images:300, Photo images: 7038), most of the image from the generator had similar shades of colors and atmospheres.
- **Differentiable Augmentation** that can be found on the below paper which enabled efficient GAN training.
- Directly augmenting the training data manipulates the distribution of real images, yielding little benefit. However, **Differentiable Augmentation** for the generated samples effectively stabilizes training and leads to better convergence because we can more easily calculate gradients from differentiable augmentation.

📚 **Differentiable Augmentation for Data-Efficient GAN Training** (arXiv:2006.10738) (https://arxiv.org/pdf/2006.10738)

- The following highlights CIFAR-10 and CIFAR-100 results of FID calculation score.

Method	CIFAR-10			CIFAR-100		
	100% data	20% data	10% data	100% data	20% data	10% data
BigGAN [2]	9.59	21.58	39.78	12.87	33.11	66.71
+ DiffAugment	**8.70**	**14.04**	**22.40**	**12.00**	**22.14**	**33.70**
CR-BigGAN [50]	9.06	20.62	37.45	11.26	36.91	47.16
+ DiffAugment	**8.49**	**12.84**	**18.70**	**11.25**	**20.28**	**26.90**
StyleGAN2 [18]	11.07	23.08	36.02	16.54	32.30	45.87
+ DiffAugment	**9.89**	**12.15**	**14.50**	**15.22**	**16.65**	**20.75**

Table 4: **CIFAR-10 and CIFAR-100** results. We select the snapshot with the best FID for each method. Results are averaged over 5 evaluation runs; all standard deviations are less than 1% relatively. We use 10k samples and the validation set as the reference distribution for FID calculation, as done in prior work [50]. Concurrent works [14, 16] use a different protocol: 50k samples and the training set as the reference distribution. If we adopt this evaluation protocol, our BigGAN + DiffAugment achieves an FID of 4.61, CR-BigGAN + DiffAugment achieves an FID of 4.30, and StyleGAN2 + DiffAugment achieves an FID of 5.79.

Discriminator Loss Function L_D: Evaluates the discriminator's ability to distinguish between real and generated data by incorporating both types of samples into the loss calculation.

- $-D(T(x))$: This output reflects the discriminator's confidence in whether the data is real or fake. The negative sign indicates that the loss should decrease when the discriminator is confident that the augmented real data is real.
- $D(T(G(z)))$: This is the output of the discriminator when it processes the augmented fake data.

$$L_D = \mathbb{E}x \sim p_{\text{data}}(x)[f_D(-D(T(x)))] + \mathbb{E}_{z\sim p(z)}[f_D(D(T(G(z))))]$$

> **Generator Loss Function** L_G : The generator's loss, which evaluates how well the generator can produce data that the discriminator mistakenly identifies as real.
>
> - $D(T(G(z)))$: This is the output of the discriminator when it processes the augmented fake data. f_G: This is the loss function for the generator. It evaluates how well the generator is performing.

$$L_G = \mathbb{E}_{z \sim p(z)}[f_G(-D(T(G(z))))]$$

5. EDA (exploratory data analysis)

Environment Setting & Required Libraries

In [1]:

```
import tensorflow as tf
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver('local')
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
    print("TPU initialized successfully.")
except Exception as e:
    print("Error initializing TPU:", e)
```

```
WARNING: Logging before InitGoogle() is written to STDERR
E0000 00:00:1741470866.425740      10 common_lib.cc:612] Could not s
et metric server port: INVALID_ARGUMENT: Could not find SliceBuilder
port 8471 in any of the 0 ports provided in `tpu_process_addresses`
="local"
=== Source Location Trace: ===
learning/45eac/tfrc/runtime/common_lib.cc:230


INFO:tensorflow:Deallocate tpu buffers before initializing tpu syste
m.
INFO:tensorflow:Initializing the TPU system: local


WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1741470884.636581      10 service.cc:148] XLA service 0x
58e11db83260 initialized for platform TPU (this does not guarantee t
hat XLA will be used). Devices:
I0000 00:00:1741470884.636634      10 service.cc:156]   StreamExecut
or device (0): TPU, 2a886c8
I0000 00:00:1741470884.636638      10 service.cc:156]   StreamExecut
or device (1): TPU, 2a886c8
I0000 00:00:1741470884.636641      10 service.cc:156]   StreamExecut
or device (2): TPU, 2a886c8
I0000 00:00:1741470884.636644      10 service.cc:156]   StreamExecut
or device (3): TPU, 2a886c8
I0000 00:00:1741470884.636647      10 service.cc:156]   StreamExecut
or device (4): TPU, 2a886c8
I0000 00:00:1741470884.636649      10 service.cc:156]   StreamExecut
or device (5): TPU, 2a886c8
I0000 00:00:1741470884.636652      10 service.cc:156]   StreamExecut
or device (6): TPU, 2a886c8
I0000 00:00:1741470884.636654      10 service.cc:156]   StreamExecut
or device (7): TPU, 2a886c8


INFO:tensorflow:Finished initializing TPU system.


WARNING:absl:`tf.distribute.experimental.TPUStrategy` is deprecated,
please use the non-experimental symbol `tf.distribute.TPUStrategy` i
nstead.


INFO:tensorflow:Found TPU system:


INFO:tensorflow:Found TPU system:
```

```
INFO:tensorflow:*** Num TPU Cores: 8

INFO:tensorflow:*** Num TPU Cores: 8

INFO:tensorflow:*** Num TPU Workers: 1

INFO:tensorflow:*** Num TPU Workers: 1

INFO:tensorflow:*** Num TPU Cores Per Worker: 8

INFO:tensorflow:*** Num TPU Cores Per Worker: 8

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:CPU:0, CPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:CPU:0, CPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:0, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:0, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:1, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:1, TPU, 0, 0)



INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:2, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:2, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:3, TPU, 0, 0)
```

st/replica:0/task:0/device:TPU:3, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:3, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:4, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:4, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:5, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:5, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:6, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:6, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:7, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU:7, TPU, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU_SYSTEM:0, TPU_SYSTEM, 0, 0)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localho
st/replica:0/task:0/device:TPU_SYSTEM:0, TPU_SYSTEM, 0, 0)

TPU initialized successfully.

In [2]:

```
%matplotlib inline
import warnings
warnings.simplefilter(action='ignore', category=(RuntimeWarning, F
utureWarning, UserWarning))

import gc
import time
import os
import urllib.request
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import zipfile
from io import BytesIO
from matplotlib.pyplot import subplots
import tensorflow as tf
```

```
import keras
from keras import layers, Model
from keras.models import Sequential
from keras.layers import Layer, Dense, Activation, Dropout, Input, c
oncatenate, Average, Concatenate, Conv2D, Conv2DTranspose, BatchNorm
alization, ReLU, LeakyReLU, ZeroPadding2D, GaussianNoise
from keras.optimizers import Adam
from keras.utils import array_to_img
from keras.callbacks import Callback
from keras.initializers import RandomNormal
from keras.losses import BinaryCrossentropy

np.random.seed(0)
tf.random.set_seed(0)

AUTOTUNE = tf.data.AUTOTUNE

print("tf version: ", tf.__version__)
```

```
tf version:  2.18.0
```

In [3]:

```
url = 'https://raw.githubusercontent.com/mit-han-lab/data-efficient-
gans/master/DiffAugment-stylegan2/DiffAugment_tf.py'
filename = 'DiffAugment_tf.py'
urllib.request.urlretrieve(url, filename)
import DiffAugment_tf

function = [name for name in dir(DiffAugment_tf) if callable(getatt
r(DiffAugment_tf, name))]
print("DiffAugment functions:\n", function)
```

```
DiffAugment functions:
 ['DiffAugment', 'rand_brightness', 'rand_contrast', 'rand_cutout',
'rand_saturation', 'rand_translation']
```

In [4]:

```
competition = 'gan-getting-started'
photo_path_jpg = f"/kaggle/input/{competition}/photo_jpg"
monet_path_jpg = f"/kaggle/input/{competition}/monet_jpg"
photo_path_tfrec = f"/kaggle/input/{competition}/photo_tfrec"
monet_path_tfrec = f"/kaggle/input/{competition}/monet_tfrec"
```

```
photo_files_jpg = tf.io.gfile.glob(f"{photo_path_jpg}/*.jpg")
monet_files_jpg = tf.io.gfile.glob(f"{monet_path_jpg}/*.jpg")
photo_files_tfrec = sorted(tf.io.gfile.glob(f"{photo_path_tfrec}/*.t
frec"))
monet_files_tfrec = sorted(tf.io.gfile.glob(f"{monet_path_tfrec}/*.t
frec"))
```

In [5]:

```
photo_files_tfrds = tf.data.TFRecordDataset(photo_files_tfrec)
monet_files_tfrds = tf.data.TFRecordDataset(monet_files_tfrec)
```

In [6]:

```
def count_tfrecord_examples(tfrecords_dir: str,) -> int:

    count = 0
    for file_name in os.listdir(tfrecords_dir):
        tfrecord_path = os.path.join(tfrecords_dir, file_name)
        count += tf.data.TFRecordDataset(tfrecord_path).reduce(np.in
t64(0), lambda x, _: x + 1)
    return count.numpy()
```

Check Files

In [7]:

```
print('Photo .jpg files count:', len(photo_files_jpg))
print('Monet .jpg files count:', len(monet_files_jpg))

print('Photo .tfrec files count:', len(photo_files_tfrec))
print('Monet .tfrec files count:', len(monet_files_tfrec))
```

```
Photo .jpg files count: 7038
Monet .jpg files count: 300
Photo .tfrec files count: 20
Monet .tfrec files count: 5
```

Plot Images

In [8]:

```
def plot_img(image):
    plt.imshow(image)
    plt.axis('off')
    plt.show()

def open_plot_img(image_path):
    image = Image.open(image_path)
    plot_img(image)

def plot_tfrec(record, image_feature, extra_features=[]):
    example = tf.train.Example()
    example.ParseFromString(record.numpy())
    image = tf.image.decode_jpeg(
        example.features.feature[image_feature].bytes_list.value[0],
        channels=3
    )
    for feature in extra_features:
        print('Image type:\t', example.features.feature[feature].byt
es_list.value[0].decode('utf-8'))
    plot_img(image)
```

In [9]:

```
print('Photo:')
open_plot_img(os.path.join(photo_path_jpg, photo_files_jpg[996]))
```

Photo:

In [10]:

```
print('Monet:')
open_plot_img(os.path.join(monet_path_jpg, monet_files_jpg[198]))
```

Monet:

In [11]:

```
import os
import math
import random

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import albumentations as A
import tqdm as notebook_tqdm

def batch_visualization(path, n_images, is_random=True, figsize=(16,
16)):
    plt.figure(figsize=figsize)

    w = int(n_images ** .5)
    h = math.ceil(n_images / w)

    all_names = os.listdir(path)

    image_names = all_names[:n_images]
    if is_random:
        image_names = random.sample(all_names, n_images)

    for ind, image_name in enumerate(image_names):
        img = cv2.imread(os.path.join(path, image_name))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.subplot(h, w, ind + 1)
        plt.imshow(img)
        plt.axis("off")

    plt.show()
```

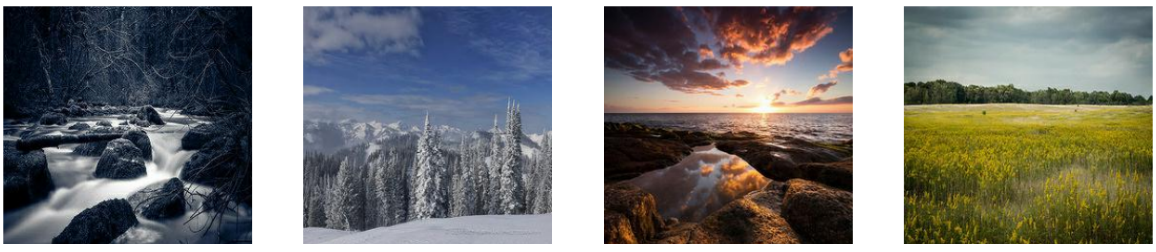
```
/usr/local/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWarnin
g: IProgress not found. Please update jupyter and ipywidgets. See ht
tps://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

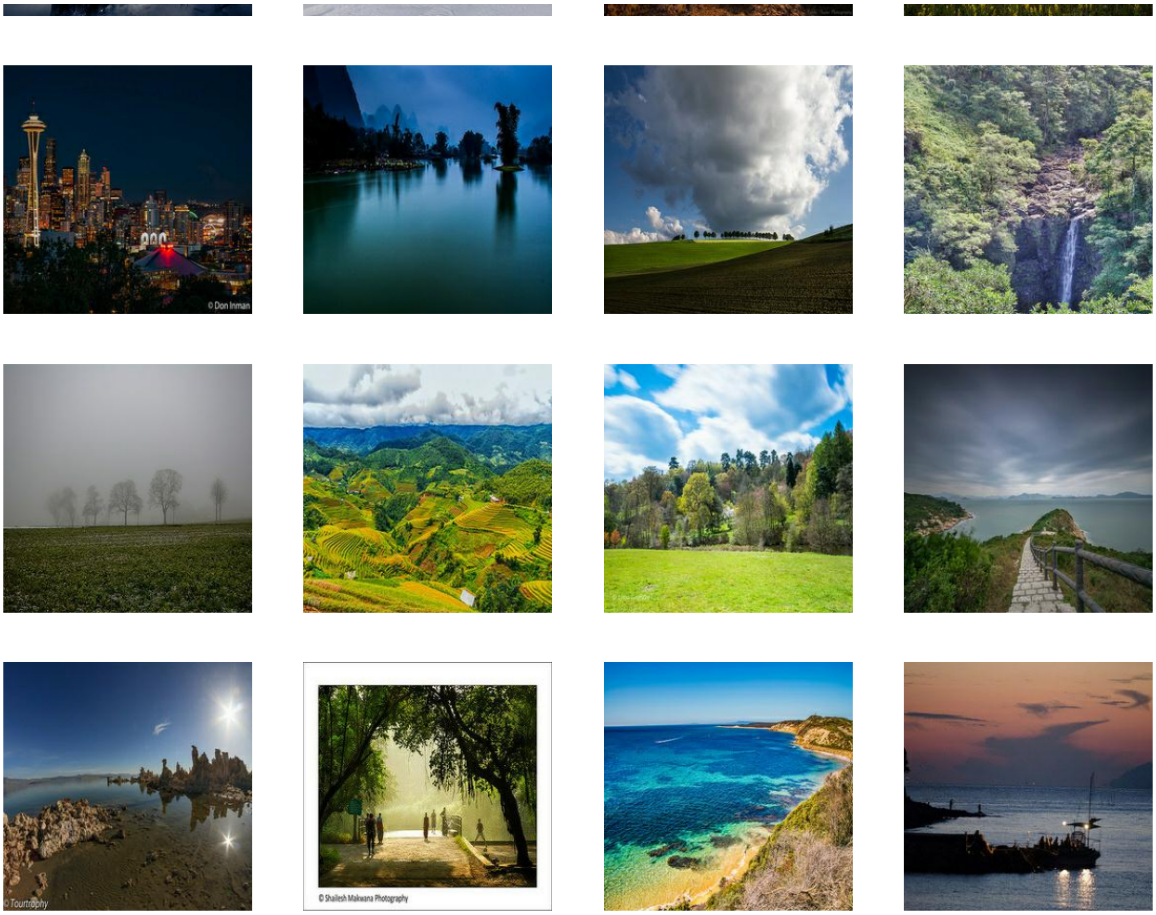
In [12]:

```
batch_visualization(monet_path_jpg, 16, is_random=True)
```

In [13]:

```
batch_visualization(photo_path_jpg, 16, is_random=True)
```





Color Channel Histograms

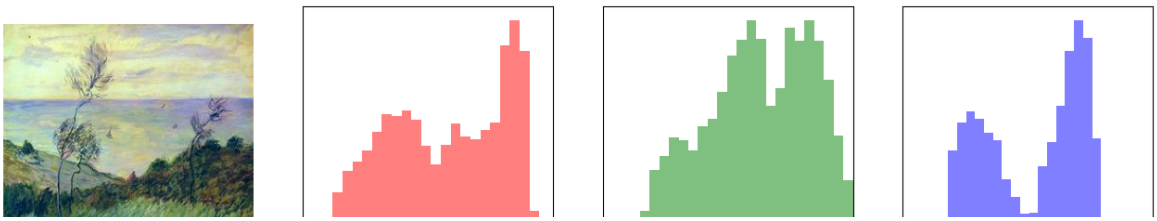
In [14]:

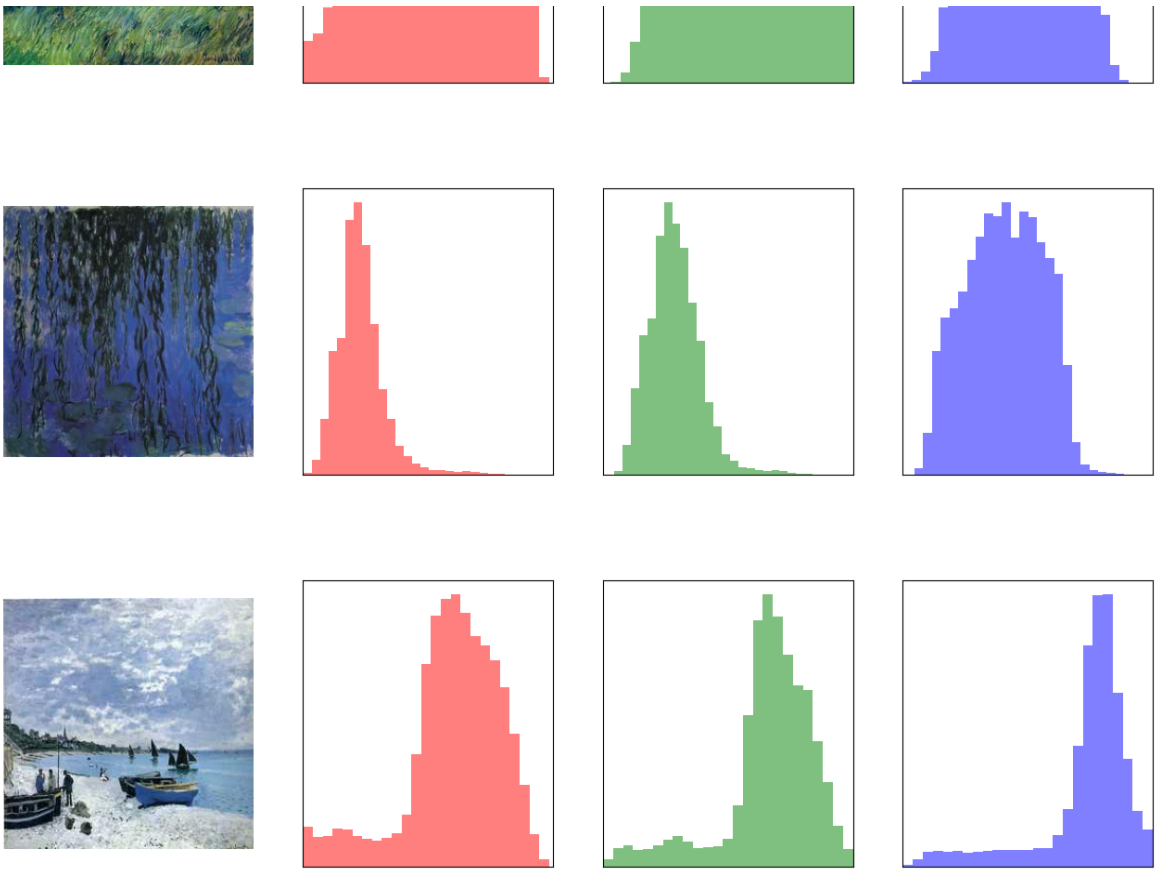
```
def color_hist_plot(image_path, figsize=(16, 4)):
    plt.figure(figsize=figsize)

    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(1, 4, 1)
    plt.imshow(img)
    plt.axis("off")

    colors = ["red", "green", "blue"]
    for i in range(len(colors)):
        plt.subplot(1, 4, i + 2)
        plt.hist(
```

```
            img[:, :, i].reshape(-1),
            bins=25,
            alpha=0.5,
            color=colors[i],
            density=True
        )
        plt.xlim(0, 255)
        plt.xticks([])
        plt.yticks([])


    plt.show()
```

In [15]:

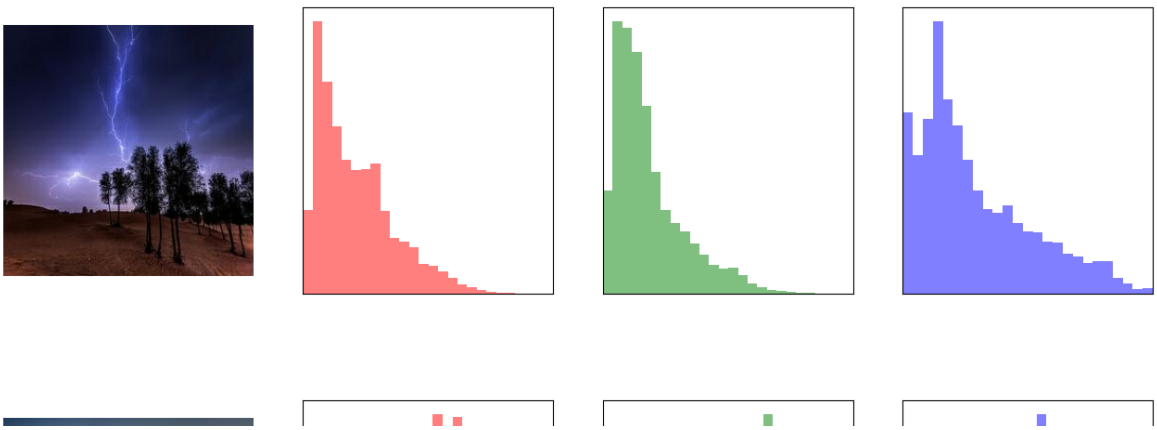
```
img_path = '/kaggle/input/gan-getting-started/monet_jpg/3d13fe022e.jpg'
color_hist_plot(img_path)

img_path = '/kaggle/input/gan-getting-started/monet_jpg/4bb4ca7b03.jpg'
color_hist_plot(img_path)

img_path = '/kaggle/input/gan-getting-started/monet_jpg/4f4de0bbba.jpg'
color_hist_plot(img_path)
```

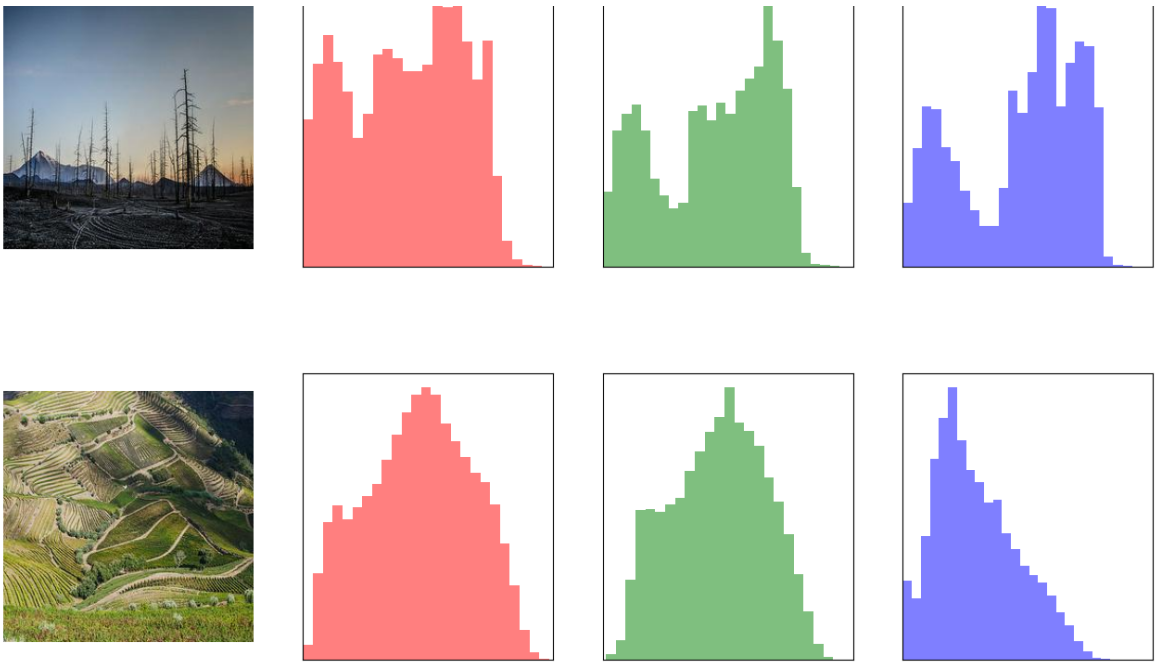




In [16]:

```
img_path = '/kaggle/input/gan-getting-started/photo_jpg/0341fa51d0.jpg'
color_hist_plot(img_path)

img_path = '/kaggle/input/gan-getting-started/photo_jpg/03bac83f64.jpg'
color_hist_plot(img_path)

img_path = '/kaggle/input/gan-getting-started/photo_jpg/03c9f2e3c2.jpg'
color_hist_plot(img_path)
```





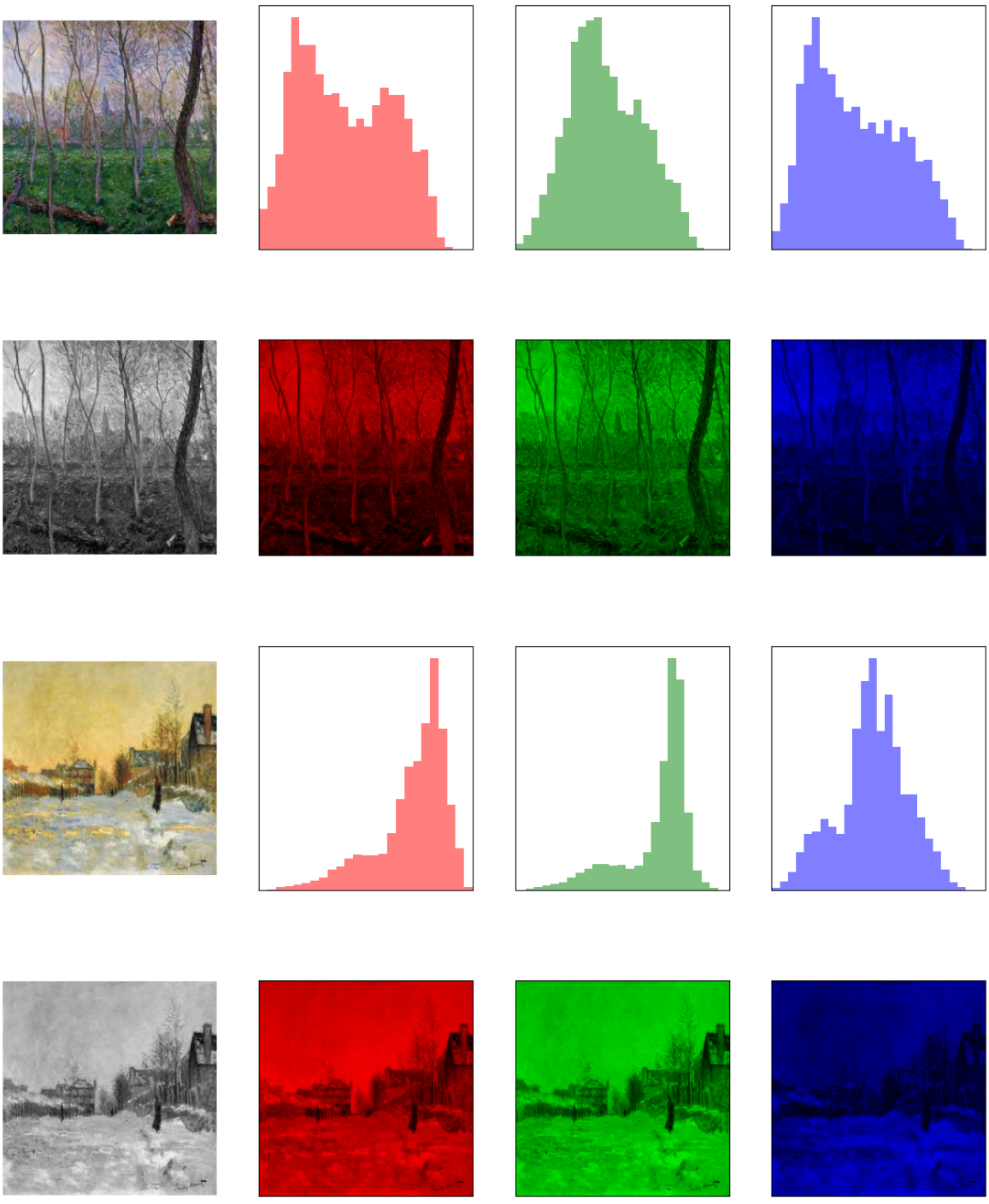
In [17]:

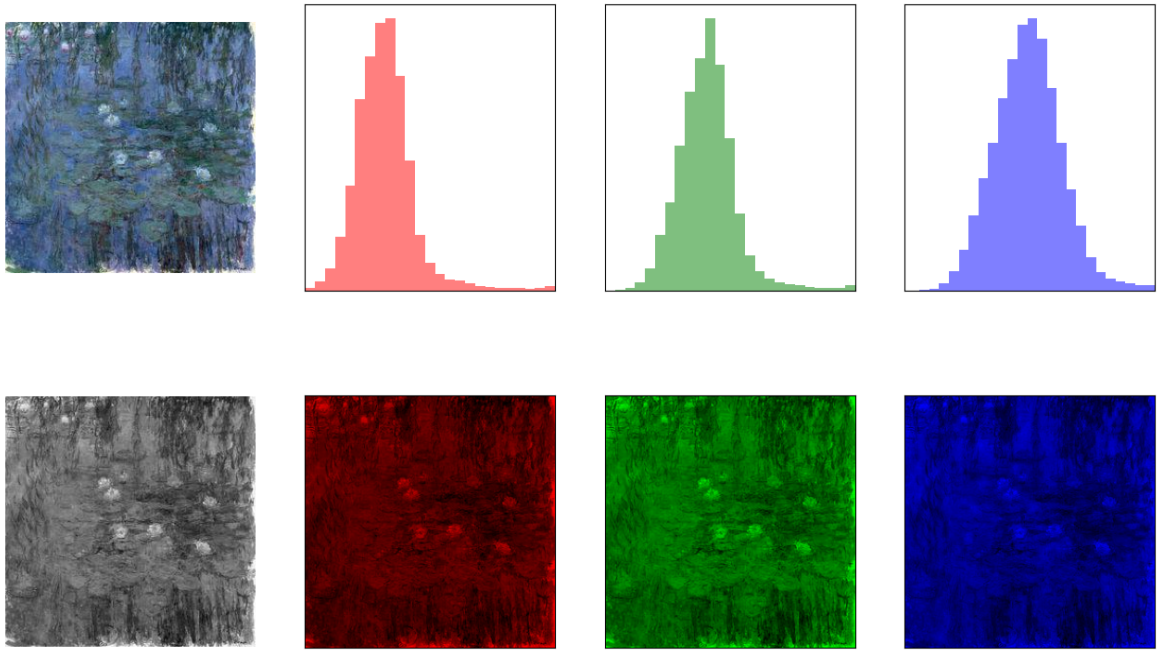
```
def channels_plot(image_path, figsize=(16, 4)):
    plt.figure(figsize=figsize)

    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(1, 4, 1)
    plt.imshow(np.mean(img, axis=2), cmap="gray")
    plt.axis('off')

    for i in range(3):
        plt.subplot(1, 4, i + 2)
        tmp_img = np.full_like(img, 0)
        tmp_img[:, :, i] = img[:, :, i]
        plt.imshow(tmp_img)
        plt.xlim(0, 255)
        plt.xticks([])
        plt.yticks([])


    plt.show()
```

In [18]:

```
img_path = '/kaggle/input/gan-getting-started/monet_jpg/281b73fb5e.jpg'
color_hist_plot(img_path)
channels_plot(img_path)
```

```
img_path = '/kaggle/input/gan-getting-started/monet_jpg/0e3b3292da.jpg'
color_hist_plot(img_path)
channels_plot(img_path)

img_path = '/kaggle/input/gan-getting-started/monet_jpg/295eb5c521.jpg'
color_hist_plot(img_path)
channels_plot(img_path)
```

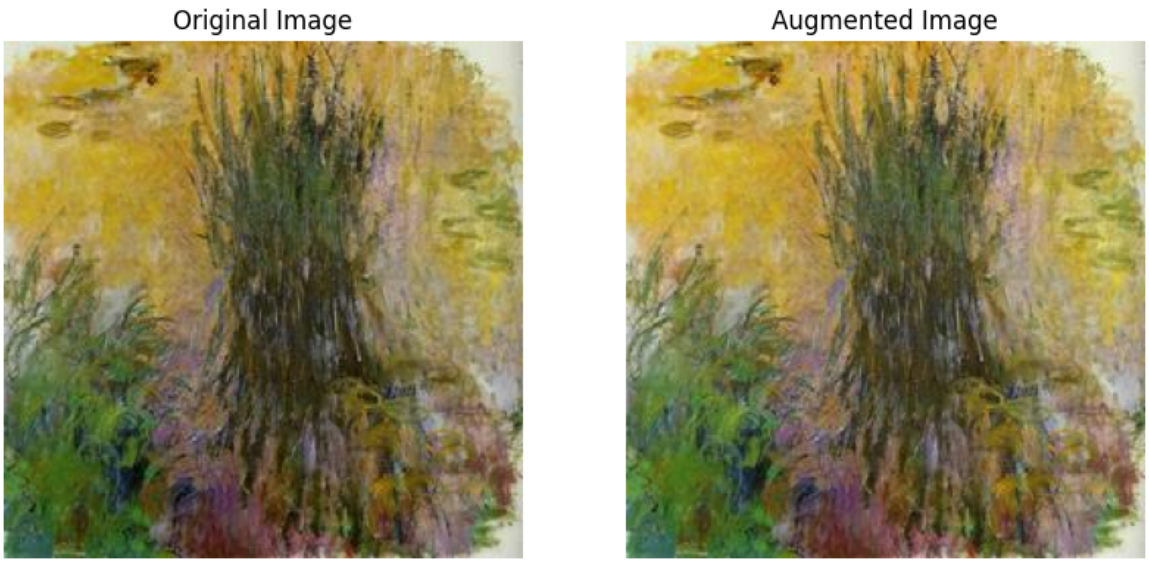




Plot Differentiable Augmentation

In [19]:

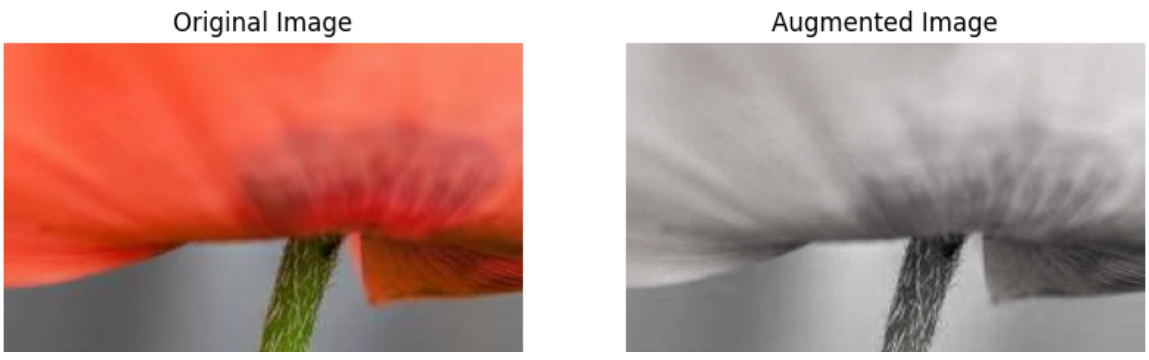
```
image_path = '/kaggle/input/gan-getting-started/monet_jpg/11be65b3e
9.jpg'  # Replace with your image path
image = Image.open(image_path).convert("RGB")
image_array = np.array(image)[np.newaxis, ...]  # Shape becomes (1, h
eight, width, channels)

augmented_image = DiffAugment_tf.DiffAugment(image_array, "color")
# Normalize the augmented image to the range [0, 1]
augmented_image = (augmented_image - np.min(augmented_image)) / (np.
max(augmented_image) - np.min(augmented_image))
augmented_image = augmented_image[0]

plt.figure(figsize=(10, 5))

# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('off')

# Augmented Image
plt.subplot(1, 2, 2)
plt.title("Augmented Image")
plt.imshow(augmented_image)
plt.axis('off')

plt.show()
```



In [20]:

```
image_path = '/kaggle/input/gan-getting-started/photo_jpg/0159685c51.jpg'  # Replace with your image path
image = Image.open(image_path).convert("RGB")
```

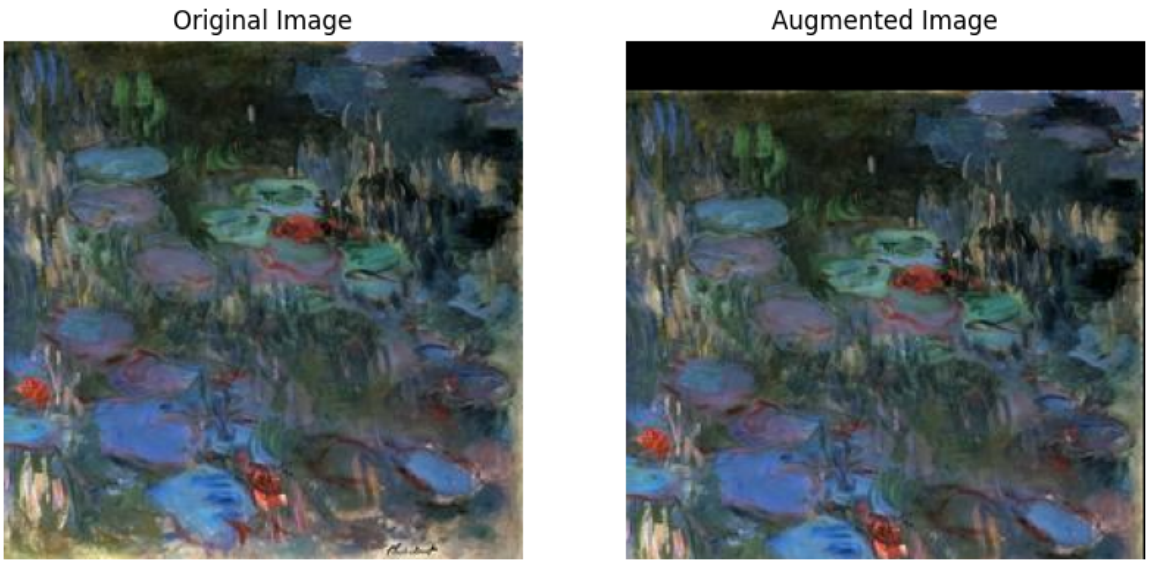
```
image_array = np.array(image)[np.newaxis, ...]  # Shape becomes (1, h
eight, width, channels)

augmented_image = DiffAugment_tf.DiffAugment(image_array, "color")
# Normalize the augmented image to the range [0, 1]
augmented_image = (augmented_image - np.min(augmented_image)) / (np.
max(augmented_image) - np.min(augmented_image))
augmented_image = augmented_image[0]

plt.figure(figsize=(10, 5))

# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('off')

# Augmented Image
plt.subplot(1, 2, 2)
plt.title("Augmented Image")
plt.imshow(augmented_image)
plt.axis('off')

plt.show()
```



In [21]:

```
image_path = '/kaggle/input/gan-getting-started/monet_jpg/1994b8d4a2.jpg'  # Replace with your image path
image = Image.open(image_path).convert("RGB")
image_array = np.array(image)[np.newaxis, ...]  # Shape becomes (1, height, width, channels)

augmented_image = DiffAugment_tf.DiffAugment(image_array, "translation")
augmented_image = augmented_image[0]

plt.figure(figsize=(10, 5))

# Original Image
```

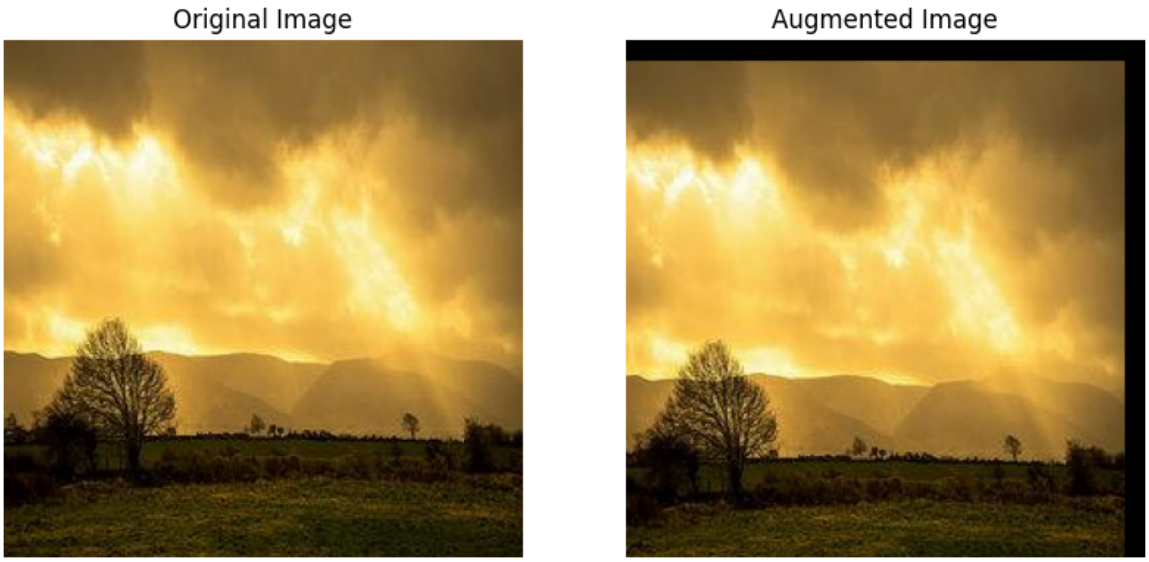
```
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('off')

# Augmented Image
plt.subplot(1, 2, 2)
plt.title("Augmented Image")
plt.imshow(augmented_image)
plt.axis('off')

plt.show()
```



In [22]:

```
image_path = '/kaggle/input/gan-getting-started/photo_jpg/018db0f253.jpg'  # Replace with your image path
image = Image.open(image_path).convert("RGB")
image_array = np.array(image)[np.newaxis, ...]  # Shape becomes (1, height, width, channels)

augmented_image = DiffAugment_tf.DiffAugment(image_array, "translation")
augmented_image = augmented_image[0]

plt.figure(figsize=(10, 5))

# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('off')
```

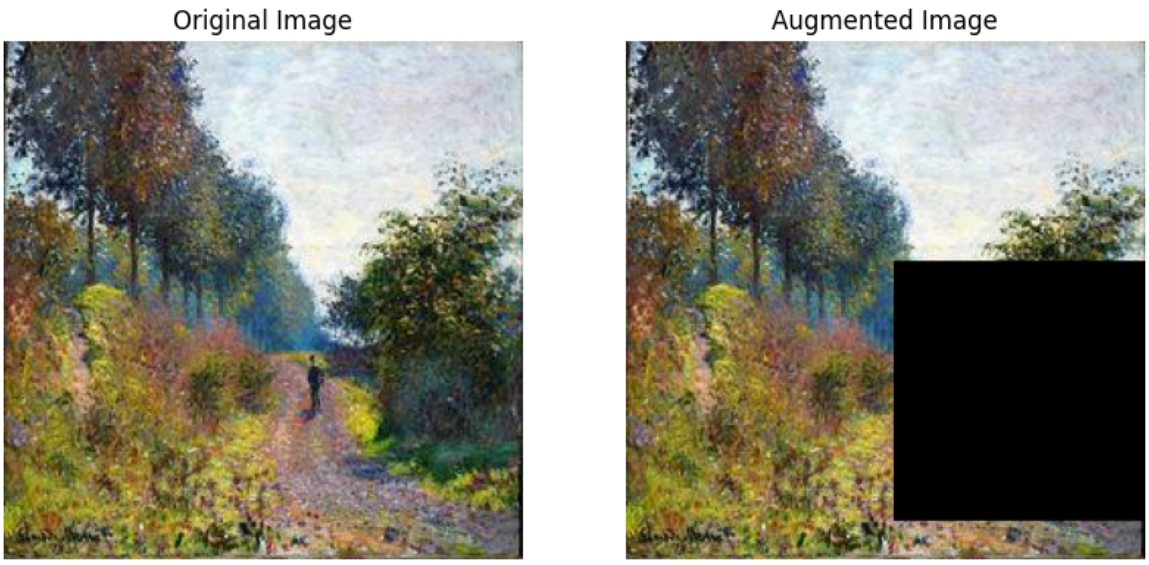
```
# Augmented Image
plt.subplot(1, 2, 2)
plt.title("Augmented Image")
plt.imshow(augmented_image)
plt.axis('off')

plt.show()
```



In [23]:

```
image_path = '/kaggle/input/gan-getting-started/monet_jpg/24af73333
4.jpg'  # Replace with your image path
image = Image.open(image_path).convert("RGB")
image_array = np.array(image)[np.newaxis, ...]  # Shape becomes (1, h
eight, width, channels)

augmented_image = DiffAugment_tf.DiffAugment(image_array, "cutout")
augmented_image = augmented_image[0]

# Normalize the augmented image to the range [0, 1] if it's a float
if augmented_image.dtype == np.float32 or augmented_image.dtype == n
p.float64:
    augmented_image = (augmented_image - np.min(augmented_image)) /
(np.max(augmented_image) - np.min(augmented_image))
# If it's an integer type, clip and convert to uint8
elif augmented_image.dtype == np.uint8:
    augmented_image = np.clip(augmented_image, 0, 255).astype(np.uin
t8)


plt.figure(figsize=(10, 5))
```

```
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('off')

# Augmented Image
plt.subplot(1, 2, 2)
plt.title("Augmented Image")
plt.imshow(augmented_image)
plt.axis('off')

plt.show()
```



In [24]:

```
image_path = '/kaggle/input/gan-getting-started/photo_jpg/01622039ef.jpg'  # Replace with your image path
image = Image.open(image_path).convert("RGB")
image_array = np.array(image)[np.newaxis, ...]  # Shape becomes (1, height, width, channels)

augmented_image = DiffAugment_tf.DiffAugment(image_array, "cutout")
augmented_image = augmented_image[0]

# Normalize the augmented image to the range [0, 1] if it's a float
if augmented_image.dtype == np.float32 or augmented_image.dtype == np.float64:
    augmented_image = (augmented_image - np.min(augmented_image)) / (np.max(augmented_image) - np.min(augmented_image))
# If it's an integer type, clip and convert to uint8
elif augmented_image.dtype == np.uint8:
    augmented_image = np.clip(augmented_image, 0, 255).astype(np.uint8)

plt.figure(figsize=(10, 5))

# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('off')

# Augmented Image
plt.subplot(1, 2, 2)
plt.title("Augmented Image")
plt.imshow(augmented_image)
```

```
plt.axis('off')

plt.show()
```

In [25]:

```
def tfrecord_structure(ds):

    record = next(iter(ds.take(1)))
    example = tf.train.Example()
    example.ParseFromString(record.numpy())

    print('TFRecord structure:')


    for feature_key in example.features.feature.keys():
        feature = example.features.feature[feature_key]


        if feature.HasField('bytes_list'):

            value = feature.bytes_list.value[0]
            print(f"{feature_key}: {value[:20]}... (length: {len(val
ue)})")
        elif feature.HasField('float_list'):
            value = feature.float_list.value
            print(f"{feature_key}: {value}")
        elif feature.HasField('int64_list'):
            value = feature.int64_list.value
            print(f"{feature_key}: {value}")
        else:
            print(f"{feature_key}: (Unknown type)")
```

Tfrecord Structure

In [26]:

```
tfrecord_structure(monet_files_tfrds)
```

```
TFRecord structure:
target: b'monet'... (length: 5)
image_name: b'25c9904782'... (length: 10)
image: b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\
x00'... (length: 36297)
```

In [27]:
```
tfrecord_structure(photo_files_tfrds)
```

```
TFRecord structure:
target: b'photo'... (length: 5)
image_name: b'0b91f359c5'... (length: 10)
image: b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\
x00'... (length: 20148)
```

In [28]:
```
plot_tfrec(
    next(iter(photo_files_tfrds.skip(np.random.randint(0, 84)))),
    'image', ['target', 'image_name']
)
```

```
Image type:      photo
Image type:      481456b5a6
```

In [29]:
```
plot_tfrec(
    next(iter(monet_files_tfrds.skip(np.random.randint(0, 96)))),
    'image', ['target', 'image_name']
```

```
)
```

```
Image type:      monet
Image type:      99d94af5dd
```

6. Preparing Datasets

In [30]:

```
if strategy.num_replicas_in_sync == 1:
    BATCH_SIZE = 2
    BUFFER_SIZE = 256
else:
    BATCH_SIZE_PER_REPLICA = 1
    BATCH_SIZE = BATCH_SIZE_PER_REPLICA * strategy.num_replicas_in_s
ync
    TEST_BATCH_SIZE_PER_REPLICA = 32
```

```
    TEST_BATCH_SIZE = TEST_BATCH_SIZE_PER_REPLICA * strategy.num_rep
licas_in_sync

print(BATCH_SIZE, TEST_BATCH_SIZE)
```

```
8 256
```

In [31]:

```
def decode_img_tensor(image):
    shape = [256, 256, 3]
    image = tf.image.decode_jpeg(image, channels=3)
    image = (tf.cast(image, tf.float32) / 127.5) - 1
    image = tf.reshape(image, shape)
    return image

def tfrecord_tensor(example):
    tfrecord_format = {
        "image": tf.io.FixedLenFeature([], tf.string),
    }
    example = tf.io.parse_single_example(example, tfrecord_format)
```

```
    image = decode_img_tensor(example['image'])
    return image

def load_dataset(filenames):
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.map(tfrecord_tensor, num_parallel_calls=AUTOTU
NE)
    return dataset

def augment_img(image):
    image = tf.image.resize(image, [286, 286])
    image = tf.image.random_crop(image, size=[BATCH_SIZE, 256, 256,
3])
    image = tf.image.random_flip_left_right(image)
    return image
```

In [32]:

```
def custom_gan_ds(photo_files, monet_files, augment_fn=None, repeat=
True, shuffle=True, cache=True, batch_size=1):
    monet_ds = load_dataset(monet_files)
    photo_ds = load_dataset(photo_files)
    if cache:
        monet_ds = monet_ds.cache()
        photo_ds = photo_ds.cache()
    if repeat:
        monet_ds = monet_ds.repeat()
        photo_ds = photo_ds.repeat()
    if shuffle:
        monet_ds_entries = monet_ds.reduce(0, lambda x,_: x+1).nump
y()
        photo_ds_entries = photo_ds.reduce(0, lambda x,_: x+1).nump
y()
        monet_ds = monet_ds.shuffle(monet_ds_entries)
        photo_ds = photo_ds.shuffle(photo_ds_entries)
```

```
    monet_ds = monet_ds.batch(batch_size)
    photo_ds = photo_ds.batch(batch_size)
    if augment_fn:
        monet_ds = monet_ds.map(augment_fn, num_parallel_calls=AUTOT
UNE)
        photo_ds = photo_ds.map(augment_fn, num_parallel_calls=AUTOT
UNE)
    monet_ds = monet_ds.prefetch(AUTOTUNE)
    photo_ds = photo_ds.prefetch(AUTOTUNE)
    gan_ds = tf.data.Dataset.zip((photo_ds, monet_ds))
    return gan_ds
```

In [33]:

```
train_ds = custom_gan_ds(
    photo_files_tfrec,
    monet_files_tfrec,
    augment_fn=augment_img,
    repeat=True,
    shuffle=False,
    cache=False,
    batch_size=BATCH_SIZE
)
test_photo_ds = load_dataset(photo_files_tfrec).batch(TEST_BATCH_SIZ
E).prefetch(TEST_BATCH_SIZE)
train_ds.element_spec, test_photo_ds.element_spec
```

Out[33]:

```
((TensorSpec(shape=(8, 256, 256, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(8, 256, 256, 3), dtype=tf.float32, name=None)),
 TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None))
```

In [34]:

```
random_photo_image, random_monet_image = next(iter(train_ds.take(
1)))
random_photo_image_test = next(iter(test_photo_ds.take(1)))
```

```
random_photo_image_test = next(iter(test_photo_ds.take(1)))
print(" --- Image Pixel Range ---")
print('Photo | min: {0:.2f}, max: {1:.2f}'.format(random_photo_image.numpy().min(), random_photo_image.numpy().max()))
print('Monet | min: {0:.2f}, max: {1:.2f}'.format(random_monet_image.numpy().min(), random_monet_image.numpy().max()))
print('Photo (test) | min: {0:.2f}, max: {1:.2f}'.format(random_photo_image_test.numpy().min(), random_photo_image_test.numpy().max()))
```

```
 --- Image Pixel Range ---
Photo | min: -1.00, max: 1.00
Monet | min: -1.00, max: 1.00
Photo (test) | min: -1.00, max: 1.00
```

In [35]:

```
plot_img((random_photo_image[0]+1)/2)
```

In [36]:
```
plot_img((random_photo_image[2]+1)/2)
```

In [37]:
```
len(random_photo_image)
```

Out[37]:
```
8
```

In [38]:
```
plot_img((random_photo_image[5]+1)/2)
```

In [39]:
```
del random_photo_image, random_monet_image, random_photo_image_test
gc.collect()
```

Out[39]:
```
119429
```

7. Cycle Consistent CycleGAN

I Augument Function

In [40]:
```
def diffaug_fn(image):
    return DiffAugment_tf.DiffAugment(image, "color,translation,cuto
ut")
```

I Convolutional Layers

In [41]:
```
class NormalizationLayer(tf.keras.layers.Layer):
    def __init__(self, epsilon=1e-5):
        super(NormalizationLayer, self).__init__()
        self.epsilon = epsilon

    def build(self, input_shape):
        self.scale = self.add_weight(
            name='scale',
            shape=input_shape[-1:],
            initializer=tf.random_normal_initializer(1., 0.02),
            trainable=True)
        self.offset = self.add_weight(
            name='offset',
            shape=input_shape[-1:],
            initializer='zeros',
            trainable=True)

    def call(self, x):
        mean, variance = tf.nn.moments(x, axes=[1, 2], keepdims=Tru
e)
        inv = tf.math.rsqrt(variance + self.epsilon)
        normalized = (x - mean) * inv
        return self.scale * normalized + self.offset
```

In [42]:

```
def down_conv(filters, size, apply_instancenorm=True, add_noise=Fals
e):
    initializer = tf.random_normal_initializer(0., 0.02)
    result = keras.Sequential()
    result.add(Conv2D(filters, size, strides=2, padding='same',
                      kernel_initializer=initializer, use_bias=Fals
e))
    if add_noise:
        result.add(GaussianNoise(0.2))
    if apply_instancenorm:
        result.add(NormalizationLayer())
    result.add(LeakyReLU())
    return result

def up_conv(filters, size, apply_dropout=False):
    initializer = tf.random_normal_initializer(0., 0.02)
    result = Sequential()
    result.add(Conv2DTranspose(filters, size, strides=2,
                               padding='same',
                               kernel_initializer=initializer,
                               use_bias=False))
    result.add(NormalizationLayer())
    if apply_dropout:
        result.add(Dropout(0.5))
    result.add(ReLU())
    return result
```

I Generator

In [43]:

```
def create_generator():
    inputs = Input(shape=[256,256,3])
    output_channels = 3

    down_stack = [
        down_conv(64, 4, apply_instancenorm=False),
        down_conv(128, 4),
        down_conv(256, 4),
        down_conv(512, 4),
        down_conv(512, 4),
        down_conv(512, 4),
        down_conv(512, 4)
```

```
        down_conv(512, 4),
    ]

    up_stack = [
        up_conv(512, 4, apply_dropout=True),
        up_conv(512, 4, apply_dropout=True),
        up_conv(512, 4, apply_dropout=True),
        up_conv(512, 4),
        up_conv(256, 4),
        up_conv(128, 4),
        up_conv(64, 4),
    ]

    initializer = tf.random_normal_initializer(0., 0.02)
    last = Conv2DTranspose(output_channels, 4,
                           strides=2,
                           padding='same',
                           kernel_initializer=initializer,
                           activation='tanh')

    x = inputs
    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)

    skips = reversed(skips[:-1])

    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = Concatenate()([x, skip])
    outputs = last(x)
    return keras.Model(inputs=inputs, outputs=outputs)
```

I Discriminator

In [44]:

```
def create_discriminator(add_noise=True):
    initializer = tf.random_normal_initializer(0., 0.02)
    inputs = Input(shape=[256, 256, 3], name='input_image')
```

```
    inputs = input(shape=[256, 256, 3], name='input_image')

    x = inputs
    x = down_conv(64, 4, False, add_noise=add_noise)(x)
    x = down_conv(128, 4, add_noise=add_noise)(x)
    x = down_conv(256, 4, add_noise=add_noise)(x)
    x = ZeroPadding2D()(x)
    x = Conv2D(512, 4, strides=1,
               kernel_initializer=initializer,
               use_bias=False)(x)
    if add_noise:
        x = GaussianNoise(0.2)(x)

    x = NormalizationLayer()(x)
    x = LeakyReLU()(x)
    x = ZeroPadding2D()(x)

    outputs = Conv2D(1, 4, strides=1,
                     kernel_initializer=initializer)(x)
    return tf.keras.Model(inputs=inputs, outputs=outputs)
```

8. Model

In [45]:

```
class TrainingGAN(Model):
    def __init__(self, generator_g, discriminator_x, generator_f, di
scriminator_y,
                 lambda_loss=10, gamma_loss=1e-4, lambda_id_loss=1e-
5, diffaug_fn=None, batch_size=32, **kwargs):
        super().__init__(**kwargs)
        self.generator_g = generator_g
        self.generator_f = generator_f
        self.discriminator_x = discriminator_x
        self.discriminator_y = discriminator_y
        self.lambda_loss = lambda_loss
        self.gamma_loss = gamma_loss
        self.lambda_id_loss = lambda_id_loss
        self.diffaug_fn = diffaug_fn
        disc_inputs = Input(shape=[None, None, 3], name='input_imag
e')
        self.discriminator_features_x = Model(inputs=self.discrimina
tor_x.input,
                                              outputs=self.discrimin
ator_x.layers[-2].output)
```

```
        self.discriminator_features_y = Model(inputs=self.discrimina
tor_y.input,
                                              outputs=self.discrimin
ator_y.layers[-2].output)

    def compile(self, generator_g_optimizer, discriminator_x_optimiz
er,
                generator_f_optimizer, discriminator_y_optimizer):
        super().compile()
        self.generator_g_optimizer = generator_g_optimizer
        self.discriminator_x_optimizer = discriminator_x_optimizer
        self.generator_f_optimizer = generator_f_optimizer
        self.discriminator_y_optimizer = discriminator_y_optimizer

    def _discriminator_bce_loss(self, real, generated):
        real_loss = BinaryCrossentropy(
            from_logits=True,
            reduction=tf.keras.losses.Reduction.NONE)(tf.ones_like(r
eal), real)
        real_loss = tf.reduce_mean(real_loss)
        generated_loss = BinaryCrossentropy(
            from_logits=True,
            reduction=tf.keras.losses.Reduction.NONE)(tf.zeros_like(
generated), generated)
        generated_loss = tf.reduce_mean(generated_loss)
        total_disc_loss = real_loss + generated_loss
        return total_disc_loss * 0.5

    def _generator_bce_loss(self, generated):
        loss = BinaryCrossentropy(
            from_logits=True,
            reduction=tf.keras.losses.Reduction.NONE)(tf.ones_like(g
enerated), generated)
        loss = tf.reduce_mean(loss)
        return loss

    def _cycle_loss(self, real_image, cycled_image):
        loss = tf.reduce_mean(tf.abs(real_image - cycled_image))
        return loss

    def _identity_loss(self, real_image, same_image):
        loss = tf.reduce_mean(tf.abs(real_image - same_image))
        return 0.5 * loss

    @tf.function
    def train_step(self, batch):
        (real_x, real_y) = batch
```

```
        batch_size = tf.shape(real_y)[0]
        with tf.GradientTape(persistent=True) as tape:
            fake_y = self.generator_g(real_x, training=True)
            cycled_x = self.generator_f(fake_y, training=True)
            fake_x = self.generator_f(real_y, training=True)
            cycled_y = self.generator_g(fake_x, training=True)
            same_x = self.generator_f(real_x, training=True)
            same_y = self.generator_g(real_y, training=True)

            if self.diffaug_fn:
                both_y = tf.concat([real_y, fake_y], axis=0)
                aug_y = self.diffaug_fn(both_y)
                aug_real_y = aug_y[:batch_size]
                aug_fake_y = aug_y[batch_size:]
                disc_real_y = self.discriminator_y(aug_real_y, train
ing=True)
                disc_fake_y = self.discriminator_y(aug_fake_y, train
ing=True)
            else:
                disc_real_y = self.discriminator_y(real_y, training=
True)
                disc_fake_y = self.discriminator_y(fake_y, training=
True)


            disc_real_x = self.discriminator_x(real_x, training=Tru
e)
            disc_fake_x = self.discriminator_x(fake_x, training=Tru
e)

            disc_feature_x = self.discriminator_features_x(real_x, t
raining=True)
            disc_feature_cycled_x = self.discriminator_features_x(cy
cled_x, training=True)
            disc_feature_y = self.discriminator_features_x(real_y, t
raining=True)
            disc_feature_cycled_y = self.discriminator_features_x(cy
cled_y, training=True)

            gen_g_loss = self._generator_bce_loss(disc_fake_y)
            gen_f_loss = self._generator_bce_loss(disc_fake_x)

            cycle_loss_x = ((1 - self.gamma_loss) * self._cycle_los
s(real_x, cycled_x) +
                            self.gamma_loss * self._cycle_loss(disc_
feature_x, disc_feature_cycled_x))
            cycle_loss_y = ((1 - self.gamma_loss) *  self._cycle_los
```

```
s(real_y, cycled_y) +
                                self.gamma_loss * self._cycle_loss(disc_
feature_y, disc_feature_cycled_y))

            total_cycle_loss = self.lambda_loss * (cycle_loss_x + cy
cle_loss_y)

            id_loss_y = self.lambda_id_loss * self._identity_loss(re
al_y, same_y)
            id_loss_x = self.lambda_id_loss * self._identity_loss(re
al_x, same_x)

            total_gen_g_loss = gen_g_loss + total_cycle_loss + id_lo
ss_y
            total_gen_f_loss = gen_f_loss + total_cycle_loss + id_lo
ss_x

            disc_x_loss = self._discriminator_bce_loss(disc_real_x,
disc_fake_x)
            disc_y_loss = self._discriminator_bce_loss(disc_real_y,
disc_fake_y)

        generator_g_gradients = tape.gradient(total_gen_g_loss, sel
f.generator_g.trainable_variables)
        generator_f_gradients = tape.gradient(total_gen_f_loss, sel
f.generator_f.trainable_variables)

        discriminator_x_gradients = tape.gradient(disc_x_loss, self.
discriminator_x.trainable_variables)
        discriminator_y_gradients = tape.gradient(disc_y_loss, self.
discriminator_y.trainable_variables)

        self.generator_g_optimizer.apply_gradients(zip(generator_g_g
radients, self.generator_g.trainable_variables))
        self.generator_f_optimizer.apply_gradients(zip(generator_f_g
radients, self.generator_f.trainable_variables))

        self.discriminator_x_optimizer.apply_gradients(zip(discrimin
ator_x_gradients, self.discriminator_x.trainable_variables))
        self.discriminator_y_optimizer.apply_gradients(zip(discrimin
ator_y_gradients, self.discriminator_y.trainable_variables))

        return {
            "disc_x_loss": disc_x_loss,
            "disc_y_loss": disc_y_loss,
            "total_gen_g_loss": total_gen_g_loss,
            "total_gen_f_loss": total_gen_f_loss,
```

```
            "gen_g_loss": cycle_loss_x,
            "gen_f_loss": cycle_loss_y,
            }
```

9. Callback

Plot Prediction Callback

In [46]:

```
class PlotPredCallback(keras.callbacks.Callback):
    def __init__(self, input_image, model_generator, epoch_interval=
None, nrows=1, figsize=(11, 11)):
        self.input_image = input_image
        self.model_generator = model_generator
        self.epoch_interval = epoch_interval
        self.nrows = nrows
        self.figsize=(11, 11)

    def _plot_pred(self):
        preds = self.model_generator.predict(self.input_image, verbo
se=0)
        (fig, axes) = subplots(nrows=self.nrows, ncols=2, figsize=se
lf.figsize)
        if self.nrows == 1:
            axes = [axes]
        for (ax, inp, pred) in zip(axes, self.input_image, preds):
            ax[0].imshow(array_to_img(inp))
            ax[0].set_title("Input Image")
            ax[0].set_axis_off()
            ax[1].imshow(array_to_img(pred))
            ax[1].set_title("Prediction")
            ax[1].set_axis_off()
        plt.show()

    def on_epoch_end(self, epoch, logs=None):
        if self.epoch_interval and epoch % self.epoch_interval == 0:
            self._plot_pred()

    def on_train_end(self, logs=None):
        self._plot_pred()
```

Learning Rate Callback

In [47]:

```
import numpy as np
from tensorflow.keras.callbacks import Callback

class LRCallback(Callback):
    def __init__(self, epochs_count, lr_start=2e-4, lr_end=5e-6):
        super().__init__()
        self.epoch_min = epochs_count // 2
        epochs_update_count = epochs_count - self.epoch_min
        self.learning_rate_values = np.linspace(lr_start, lr_end, ep
ochs_update_count)

    def _scheduler_fn(self, epoch, learning_rate):
        if epoch < self.epoch_min:
            return learning_rate
        else:
            return self.learning_rate_values[epoch - self.epoch_min]

    def on_epoch_begin(self, epoch, logs=None):
        new_lr = self._scheduler_fn(epoch, self.model.generator_g_op
timizer.learning_rate)
        self.model.generator_g_optimizer.learning_rate = new_lr
        self.model.discriminator_x_optimizer.learning_rate = new_lr
        self.model.generator_f_optimizer.learning_rate = new_lr
        self.model.discriminator_y_optimizer.learning_rate = new_lr
```

Lambda & Gamma Weights Callback

In [48]:

```
class LossWeightsCallback(Callback):
    def __init__(self, epochs, lambda_start=10, lambda_end=1e-4, gam
ma_start=1e-4, gamma_end=0.999):
        super().__init__()
        self.epochs = epochs
        self.lambda_start = lambda_start
        self.lambda_end = lambda_end
        self.gamma_start = gamma_start
        self.gamma_end = gamma_end

    def on_train_begin(self, logs=None):
        self.lambda_values = np.linspace(self.lambda_start, self.lam
bda_end, self.epochs)
        self.gamma_values = np.linspace(self.gamma_start, self.gamma
_end, self.epochs)

    def on_epoch_begin(self, epoch, logs=None):
        self.model.lambda_loss = self.lambda_values[epoch]
        self.model.gamma_loss = self.gamma_values[epoch]
```

In [49]:

```
EPOCHS = 43
STEPS_PER_EPOCH = 3000
LAMBDA_START = 3
LAMBDA_END = 1e-4
GAMMA_START = 1e-4
GAMMA_END = 0.999
LAMBDA_ID = 1e-4
EPOCH_INTERVAL_PLOT = 5
EPOCH_INTERVAL_FID = None
LR_START = 2e-4
LR_END = 5e-6
```

In [50]:

```
with strategy.scope():
    update_lr_cb = LRCallback(EPOCHS, lr_start=LR_START, lr_end=LR_E
ND)
```

In [51]:
```
with strategy.scope():
    generator_g = create_generator()
    generator_f = create_generator()
    discriminator_x = create_discriminator(add_noise=True)
    discriminator_y = create_discriminator(add_noise=True)
```

```
I0000 00:00:1741470923.906649      10 device_compiler.h:188] Compile
d cluster using XLA!  This line is logged at most once for the lifet
ime of the process.
```

In [52]:
```
generator_g.summary()
```

Model: "functional_15"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 256, 256, 3)	0	–
sequential (Sequential)	(None, 128, 128, 64)	3,072	input_layer[0
sequential_1 (Sequential)	(None, 64, 64, 128)	131,328	sequential[0]
sequential_2 (Sequential)	(None, 32, 32, 256)	524,800	sequential_1[
sequential_3 (Sequential)	(None, 16, 16, 512)	2,098,176	sequential_2[
sequential_4 (Sequential)	(None, 8, 8, 512)	4,195,328	sequential_3[
sequential_5 (Sequential)	(None, 4, 4, 512)	4,195,328	sequential_4[
sequential_6 (Sequential)	(None, 2, 2, 512)	4,195,328	sequential_5[
sequential_7 (Sequential)	(None, 1, 1, 512)	4,195,328	sequential_6[
sequential_8 (Sequential)	(None, 2, 2, 512)	4,195,328	sequential_7[
concatenate (Concatenate)	(None, 2, 2, 1024)	0	sequential_8[sequential_6[
sequential_9	(None, 4, 4, 512)	8,389,632	concatenate[0

(Sequential)			
concatenate_1 (Concatenate)	(None, 4, 4, 1024)	0	sequential_9[sequential_5[
sequential_10 (Sequential)	(None, 8, 8, 512)	8,389,632	concatenate_1
concatenate_2 (Concatenate)	(None, 8, 8, 1024)	0	sequential_10 sequential_4[
sequential_11 (Sequential)	(None, 16, 16, 512)	8,389,632	concatenate_2
concatenate_3 (Concatenate)	(None, 16, 16, 1024)	0	sequential_11 sequential_3[
sequential_12 (Sequential)	(None, 32, 32, 256)	4,194,816	concatenate_3
concatenate_4 (Concatenate)	(None, 32, 32, 512)	0	sequential_12 sequential_2[
sequential_13 (Sequential)	(None, 64, 64, 128)	1,048,832	concatenate_4
concatenate_5 (Concatenate)	(None, 64, 64, 256)	0	sequential_13 sequential_1[
sequential_14 (Sequential)	(None, 128, 128, 64)	262,272	concatenate_5
concatenate_6 (Concatenate)	(None, 128, 128, 128)	0	sequential_14 sequential[0]
conv2d_transpose_7 (Conv2DTranspose)	(None, 256, 256, 3)	6,147	concatenate_6

Total params: 54,414,979 (207.58 MB)

Trainable params: 54,414,979 (207.58 MB)

Non-trainable params: 0 (0.00 B)

In [53]:

```
discriminator_x.summary()
```

Model: "functional_35"

Layer (type)	Output Shape	Par
input_image (InputLayer)	(None, 256, 256, 3)	
sequential_30 (Sequential)	(None, 128, 128, 64)	3
sequential_31 (Sequential)	(None, 64, 64, 128)	131
sequential_32 (Sequential)	(None, 32, 32, 256)	524

zero_padding2d (ZeroPadding2D)	(None, 34, 34, 256)	
conv2d_19 (Conv2D)	(None, 31, 31, 512)	2,097
gaussian_noise_3 (GaussianNoise)	(None, 31, 31, 512)	
normalization_layer_30 (NormalizationLayer)	(None, 31, 31, 512)	1
leaky_re_lu_19 (LeakyReLU)	(None, 31, 31, 512)	
zero_padding2d_1 (ZeroPadding2D)	(None, 33, 33, 512)	
conv2d_20 (Conv2D)	(None, 30, 30, 1)	8

Total params: 2,765,569 (10.55 MB)

Trainable params: 2,765,569 (10.55 MB)

Non-trainable params: 0 (0.00 B)

10. Training

In [ ]:
```
plot_pred_photo = next(iter(test_photo_ds.take(1)))
plot_pred_photo = np.expand_dims(plot_pred_photo[0], axis=0)
with strategy.scope():
    plot_pred_cb = PlotPredCallback(
        input_image=plot_pred_photo,
        model_generator=generator_g,
        epoch_interval=EPOCH_INTERVAL_PLOT)
```

In [ ]:
```
with strategy.scope():
   weights_cb = LossWeightsCallback(
```

```
        EPOCHS,
        lambda_start=LAMBDA_START,
        lambda_end=LAMBDA_END,
        gamma_start=GAMMA_START,
        gamma_end=GAMMA_END)
```

In [ ]:

```
%%time

with strategy.scope():
    model_cycleGAN = TrainingGAN(
        generator_g=generator_g,
        generator_f=generator_f,
        discriminator_x=discriminator_x,
        discriminator_y=discriminator_y,
        lambda_loss=LAMBDA_START,
        lambda_id_loss=LAMBDA_ID,
        gamma_loss=GAMMA_START,
        diffaug_fn=diffaug_fn
    )

    generator_g_optimizer = Adam(learning_rate=LR_START, beta_1=0.5)
    discriminator_x_optimizer = Adam(learning_rate=LR_START, beta_1=
0.5)
    generator_f_optimizer = Adam(learning_rate=LR_START, beta_1=0.5)
    discriminator_y_optimizer = Adam(learning_rate=LR_START, beta_1=
0.5)
```

```
    model_cycleGAN.compile(
        generator_g_optimizer=generator_g_optimizer,
        discriminator_x_optimizer=discriminator_x_optimizer,
        generator_f_optimizer=generator_f_optimizer,
        discriminator_y_optimizer=discriminator_y_optimizer,
    )
```

In [ ]:

```
%%time
model_cycleGAN.fit(
        train_ds,
        epochs=EPOCHS,
        callbacks=[plot_pred_cb,weights_cb, update_lr_cb],
        steps_per_epoch=STEPS_PER_EPOCH
)
```

11. Prediction

In [ ]:

```
# Save model
model_cycleGAN.save('/kaggle/working/saved_model/my_trained_gan_model')
```

In [ ]:

```
def plot_preds(model, ds):
    ds_iter = iter(ds)
    for n_sample in range(8):
        example_sample = next(ds_iter)
        generated_sample = model.predict(example_sample, verbose=0)
        f = plt.figure(figsize=(10, 10))
        plt.subplot(121)
        plt.title('Input image')
        plt.imshow(example_sample[0] * 0.5 + 0.5)
        plt.axis('off')
        plt.subplot(122)
        plt.title('Generated image')
        plt.imshow(generated_sample[0] * 0.5 + 0.5)
        plt.axis('off')
        plt.show()
```

In [ ]:
```
plot_preds(generator_g, test_photo_ds.shuffle(888))
```

In [ ]:
```
def gen_and_save(generator, ds):
    count = 0
    if os.path.exists('images.zip'):
        os.remove('images.zip')
    with zipfile.ZipFile('images.zip', 'w') as zipf:
        for images_batch in ds:
            predictions = generator(images_batch, training=False)
            for pred in predictions:
                count += 1
                generated_image = array_to_img(pred)
                generated_image_bytes = BytesIO()
                generated_image.save(generated_image_bytes, format='
JPEG')
                generated_image_bytes.seek(0)
                zipf.writestr(f'generated_image_{count}.jpg', genera
ted_image_bytes.getvalue())
                if count % 1024 == 0:
                    print(f'Archived images: {count}')
```

In [ ]:
```
%%time
generate_and_save(generator_g, test_photo_ds)
```

In [ ]:

```
import os
import zipfile
from io import BytesIO
from tensorflow.keras.preprocessing.image import array_to_img

def gen_and_save(generator, ds):
    count = 0
    if os.path.exists('images.zip'):
        os.remove('images.zip')
    try:
        with zipfile.ZipFile('images.zip', 'w') as zipf:
            for images_batch in ds:
                predictions = generator(images_batch, training=Fals
e)
                for pred in predictions:
                    count += 1
                    generated_image = array_to_img(pred)


                    if generated_image.mode != 'RGB':
                        generated_image = generated_image.convert('R
GB')

                    generated_image_bytes = BytesIO()
                    generated_image.save(generated_image_bytes, form
at='JPEG')
                    generated_image_bytes.seek(0)
                    zipf.writestr(f'generated_image_{count}.jpg', ge
nerated_image_bytes.getvalue())
                    if count % 1024 == 0:
                        print(f'Archived images: {count}')
    except Exception as e:
        print(f"An error occurred: {e}")
```

In [ ]:

```
import os
import zipfile

os.mkdir("/kaggle/working")
zip_filename = "/kaggle/working/images.zip"

with zipfile.ZipFile(zip_filename, 'w', zipfile.ZIP_DEFLATED) as zip
f:
    processed = 0
    total_files = len(os.listdir("/kaggle/input/data-efficient-ga
n"))
    for file in os.listdir("/kaggle/input/data-efficient-gan"):
        if file.startswith("generated_image_") and file.endswith(".j
pg"):
            file_path = os.path.join("/kaggle/input/data-efficient-g
an", file)
            zipf.write(file_path, file)
            processed += 1
            if processed % 1000 == 0:
                print(f"Zipping... {processed}/{total_files} images
on TPU")
    print(f"Processed {processed} images for zipping")

print(f"Images zipped to {zip_filename}!")
```

In [ ]:

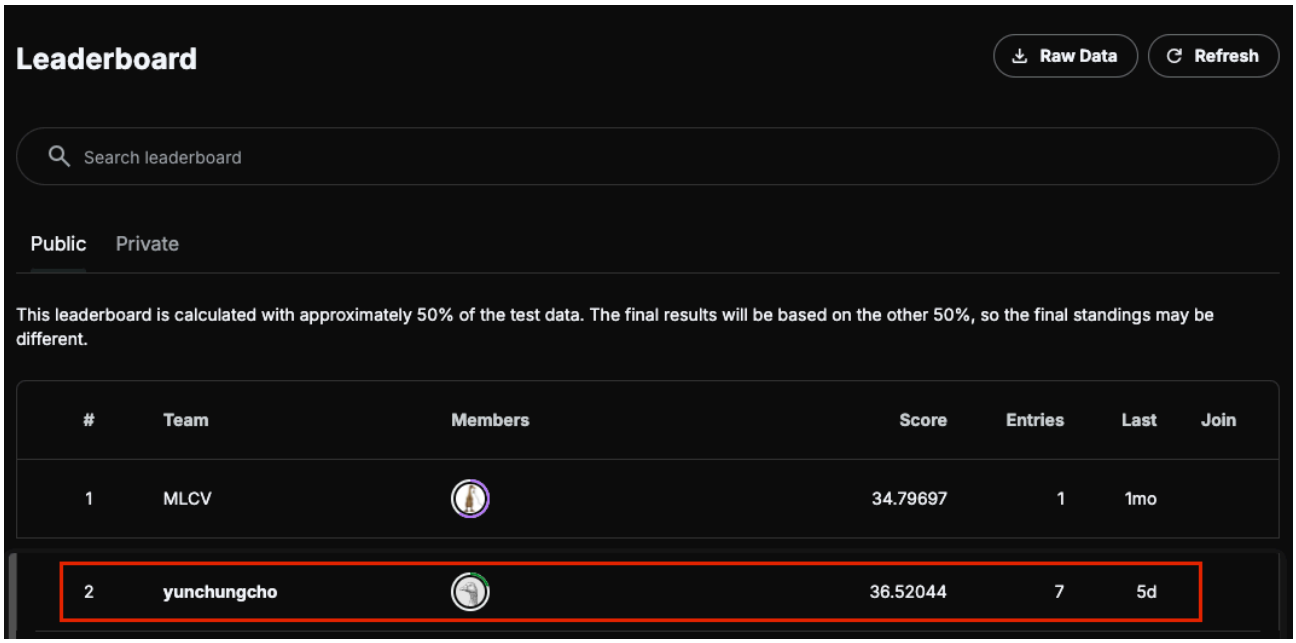
```
!ls
```

In [ ]:

```
import os
import shutil


os.makedirs("/root/.kaggle", exist_ok=True)
shutil.copy("/kaggle/input/kaggleapi/kaggle.json", "/root/.kaggle/ka
ggle.json")


!chmod 600 /root/.kaggle/kaggle.json

print("API key configured successfully!")
!kaggle competitions submit -c gan-getting-started   -f images.zip -
m "submit"
```

12. Submition Result

With $MiFID$ score 36.52044, ranked No. 2 on the leaderboard



GAN Matrix: FID and MiFID

> **FID (Fréchet Inception Distance)** is a metric that quantifies the similarity between two distributions of images, specifically comparing generated images to real images. It is primarily used to assess the quality of images produced by generative models, such as GANs (Generative Adversarial Networks). The formula for FID is:
>
> $$\mathrm{FID} = |\mu_r - \mu_g|^2 + \mathrm{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$
>
> *μ_r and Σ_r are the mean and covariance of the real images,*
> *μ_g and Σ_g are those of the generated images*

> **MiFID (Memorization-informed FID)** is an extension of the Fréchet Inception Distance (FID) that incorporates the concept of training sample memorization into the evaluation of generated images. It aims to assess not only the quality of generated images but also how well they generalize without memorizing training samples.
>
> MiFID helps to identify whether a generative model is simply memorizing training data rather than learning to generate new, diverse samples. This is particularly important in scenarios where overfitting can lead to poor generalization.
>
> Memorization Distance: The memorization distance is defined as the minimum cosine distance of all training samples in the feature space, averaged across all user-generated image samples. If this distance exceeds a predefined threshold (epsilon), it is assigned a value of 1.0. Mathematical Formulation:

Cosine Distance:

$$d_{ij} = 1 - \cos(f_{g_i}, f_{r_j}) = 1 - \frac{f_{g_i} \cdot f_{r_j}}{|f_{g_i}||f_{r_j}|}$$

f_g , f_r: Feature representations of generated and real images, respectively, defined in a pre-trained network.
f_{g_i} , f_{r_j}: The ith and jth vectors of f_g and f_r.

Minimum Distance:

$$d = \frac{1}{N} \sum_i \min_j d_{ij}$$

This defines the minimum distance of a certain generated image i across all real images j, then averaged across all generated images.

Thresholding: A threshold is applied to the memorization distance, where the weight is only considered when d is below a certain empirically determined threshold.

$$d_{thr} = \begin{cases} d, & \text{if } d < \epsilon \\ 1, & \text{otherwise} \end{cases}$$

Final MiFID Calculation:

$$\text{MiFID} = \text{FID} \cdot \frac{1}{d_{\text{thr}}}$$

d_{thr} is the threshold value that adjusts the FID based on the memorization distance.

MiFID enhances the traditional FID metric by accounting for the memorization of training samples, providing a more comprehensive evaluation of the generative model's performance. By incorporating memorization distance, MiFID helps to ensure that the generated images are not only high-quality but also diverse and representative of the underlying data distribution.

13. Training Log Analysis

Extract Loss Values

In [54]:
```
# Extracted loss values from the training log
```

```
epochs = list(range(1, 44))  # 43 epochs

# Discriminator losses
disc_x_loss = [
    0.6992, 0.6903, 0.6887, 0.6882, 0.6876, 0.6867,
    0.6861, 0.6854, 0.6849, 0.6842, 0.6845, 0.6846,
    0.6842, 0.6841, 0.6841, 0.6833, 0.6833, 0.6826,
    0.6823, 0.6824, 0.6819, 0.6815, 0.6811, 0.6809,
    0.6806, 0.6801, 0.6799, 0.6799, 0.6796, 0.6793,
    0.6798, 0.6798, 0.6798, 0.6795, 0.6795, 0.6794,
    0.6797, 0.6794, 0.6798, 0.6796, 0.6794, 0.6791,
    0.6788
]

disc_y_loss = [
    00.7003, 0.6941, 0.6871, 0.6875, 0.6879, 0.6852,
    0.6854, 0.6850, 0.6847, 0.6843, 0.6831, 0.6841,
    0.6832, 0.6825, 0.6828, 0.6825, 0.6802, 0.6816,
    0.6798, 0.6790, 0.6773, 0.6755, 0.6750, 0.6743,
    0.6722, 0.6708, 0.6709, 0.6702, 0.6686, 0.6683,
    0.6680, 0.6678, 0.6667, 0.6648, 0.6652, 0.6647,
    0.6643, 0.6640, 0.6633, 0.6627, 0.6629, 0.6617,
    0.6618
]

# Generator losses
gen_f_loss = [
    0.0689, 0.0449, 0.0443, 0.0426, 0.0411, 0.0410,
    0.0405, 0.0404, 0.0403, 0.0408, 0.0410, 0.0405,
    0.0404, 0.0404, 0.0403, 0.0408, 0.0412, 0.0412,
    0.0414, 0.0417, 0.0420, 0.0422, 0.0425, 0.0425,
    0.0425, 0.0427, 0.0432, 0.0430, 0.0431, 0.0430,
    0.0428, 0.0427, 0.0428, 0.0428, 0.0429, 0.0427,
    0.0425, 0.0423, 0.0422, 0.0421, 0.0420, 0.0419,
    0.0417
]

gen_g_loss = [
    0.0713, 0.0424, 0.0414, 0.0402, 0.0379, 0.0375,
    0.0366, 0.0360, 0.0353, 0.0352, 0.0349, 0.0343,
    0.0339, 0.0335, 0.0331, 0.0329, 0.0327, 0.0325,
    0.0322, 0.0323, 0.0322, 0.0323, 0.0321, 0.0318,
    0.0319, 0.0323, 0.0321, 0.0319, 0.0317, 0.0316,
    0.0313, 0.0310, 0.0306, 0.0303, 0.0301, 0.0297,
    0.0295, 0.0292, 0.0289, 0.0287, 0.0285, 0.0283,
    0.0283
]
```

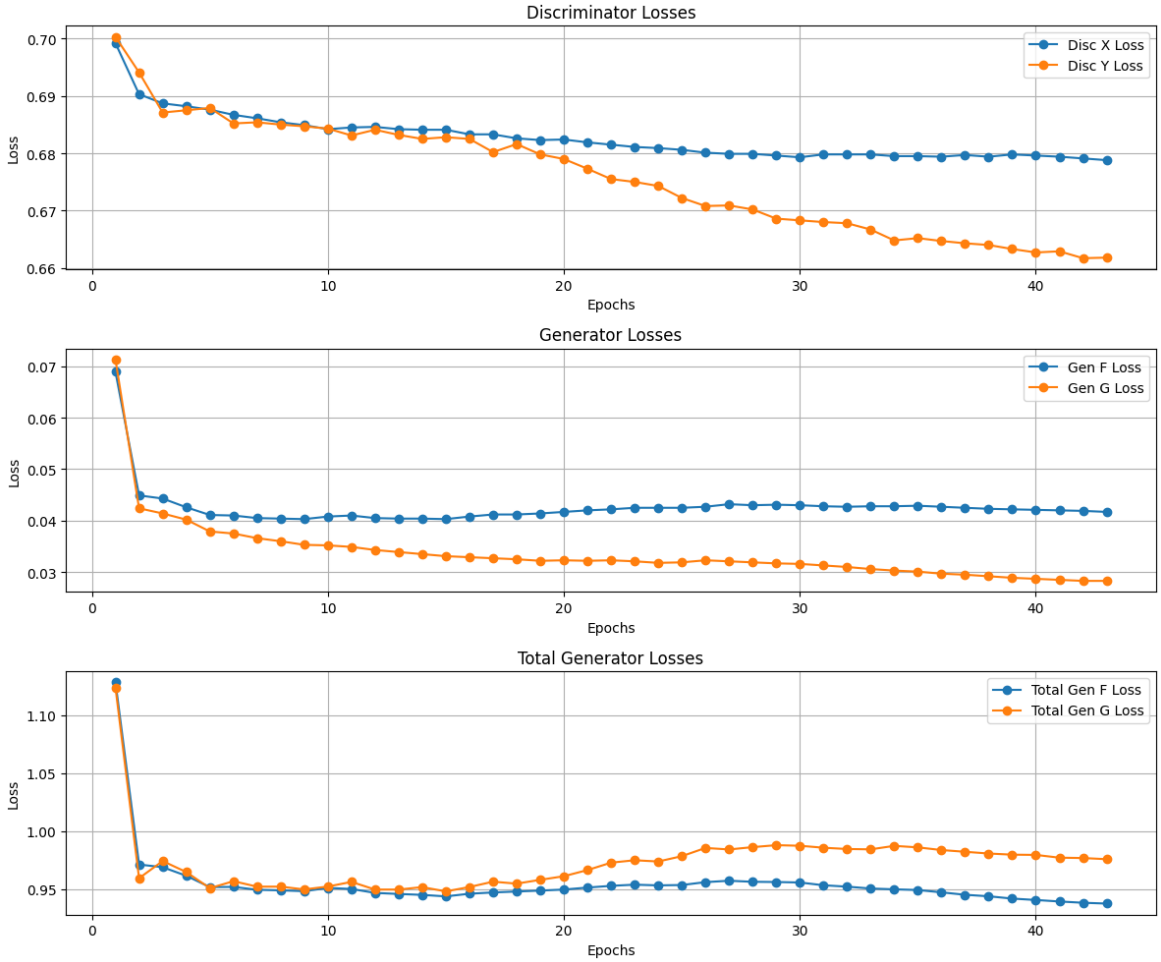
```
# Total generator losses
total_gen_f_loss = [
    1.1286, 0.9713, 0.9690, 0.9616, 0.9518, 0.9520,
    0.9495, 0.9490, 0.9483, 0.9511, 0.9502, 0.9468,
    0.9459, 0.9451, 0.9438, 0.9461, 0.9472, 0.9480,
    0.9488, 0.9497, 0.9514, 0.9529, 0.9539, 0.9533,
    0.9536, 0.9561, 0.9573, 0.9564, 0.9562, 0.9558,
    0.9534, 0.9522, 0.9506, 0.9499, 0.9493, 0.9473,
    0.9452, 0.9439, 0.9419, 0.9407, 0.9394, 0.9383,
    0.9376
]

total_gen_g_loss = [
    1.1240, 0.9595, 0.9743, 0.9648, 0.9509, 0.9569,
    0.9522, 0.9522, 0.9498, 0.9523, 0.9563, 0.9497,
    0.9497, 0.9518, 0.9481, 0.9519, 0.9565, 0.9548,
    0.9581, 0.9612, 0.9665, 0.9729,
    0.9750, 0.9739, 0.9785, 0.9856, 0.9843, 0.9863,
    0.9880, 0.9875, 0.9858, 0.9847, 0.9844, 0.9874,
    0.9861, 0.9838, 0.9823, 0.9808, 0.9798, 0.9796,
    0.9772, 0.9769, 0.9759
]
```

Discriminator, Generator, Total Generator Losses

In [55]:

```
# Now we can plot the losses
plt.figure(figsize=(12, 10))

# Discriminator Losses
plt.subplot(3, 1, 1)
plt.plot(epochs, disc_x_loss, label='Disc X Loss', marker='o')
plt.plot(epochs, disc_y_loss, label='Disc Y Loss', marker='o')
plt.title('Discriminator Losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()

# Generator Losses
plt.subplot(3, 1, 2)
plt.plot(epochs, gen_f_loss, label='Gen F Loss', marker='o')
plt.plot(epochs, gen_g_loss, label='Gen G Loss', marker='o')
plt.title('Generator Losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()

# Total Generator Losses
plt.subplot(3, 1, 3)
plt.plot(epochs, total_gen_f_loss, label='Total Gen F Loss', marker
```

```
='o')
plt.plot(epochs, total_gen_g_loss, label='Total Gen G Loss', marker
='o')
plt.title('Total Generator Losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```



Insights from Trainig Log

1. Loss Decrease Trend:

- **total_gen_f_loss** and **total_gen_g_los**s represent the loss values of the generator and discriminator. Both loss values show a decreasing trend over time. This indicates that the model is getting better and better as it trains. The lower the loss value, the closer the model's predictions are to the actual data.

2. Stability:

- The values of **total_gen_f_loss** and **total_gen_g_loss** remain relatively stable. This suggests that the model is not overfitting and is maintaining stable performance during the training process.

3. Relative Performance:

- In GANs, the **generator loss (Gen G loss)** can be higher than the **discriminator loss (Gen F loss)** due to the inherent competition between the two networks. The generator aims to create realistic data to fool the discriminator, and if the discriminator is effective, it will classify many generated samples as fake, leading to a higher generator loss. This dynamic reflects the ongoing struggle for improvement between the generator and discriminator during training.

4. Discriminator Losses:

- **disc_x_loss** and **disc_y_loss** show a consistent downward trend, indicating that both discriminators are becoming more effective at distinguishing real images from generated ones. The losses stabilize around **0.67 to 0.68**, suggesting that the discriminators are learning well but may also indicate that they are becoming too strong relative to the generators.

5. Generator Losses:

- **gen_f_loss** and **gen_g_loss** both show a decreasing trend, indicating that the generators are improving in their ability to produce realistic images. However, the generator losses are relatively low compared to the discriminator losses, which may suggest that the generators are not being challenged enough by the discriminators.

6. Total Generator Losses:

- **total_gen_f_loss** and **total_gen_g_loss** also show a decreasing trend, but the values are higher than the individual generator losses. This indicates that the total loss is influenced by both the adversarial loss and the cycle consistency loss, which is expected in a CycleGAN setup.

Insights from Training Setup & Log

1. Generators' and Discriminators' Balance:

- The relatively low generator losses compared to the discriminator losses suggest that the discriminators are becoming too strong. This aligns with the training setup where the learning rate for the discriminators is the same as for the generators. If the discriminators are updated more frequently or learn faster, they may overpower the generators.
- Improvement Suggestion: Consider adjusting the training frequency of the discriminators or implementing techniques like label smoothing to prevent them from becoming too strong.

2. Cycle Consistency Loss:

- The gradual decrease in the cycle consistency loss weight (from 3 to 1e-4) may lead to a situation where the generators focus more on fooling the discriminators rather than preserving the content of the images. This is reflected in the generator losses, which are low but may not be producing high-quality images.
- Improvement Suggestion: Maintain a higher weight for the cycle consistency loss for a longer period to ensure that the generators learn to preserve content effectively.

3. Learning Rate Dynamics:

- The learning rate schedule shows a gradual decay, which is beneficial for stabilizing training. However, the initial learning rate of **2e-4** may be too high, especially if the discriminators are learning faster.
- Improvement Suggestion: Experiment with a lower starting learning rate for both generators and discriminators to prevent overshooting and to allow for more stable convergence.

4. Monitoring Metrics:

- The absence of **FID** evaluation intervals in the training setup may hinder the ability to assess the quality of generated images. This is crucial for understanding how well the generators are performing in terms of producing realistic images.
- Improvement Suggestion: Implement regular FID evaluations to monitor the quality of generated images over time.

5. Epochs and Steps:

6. Epochs and Steps:

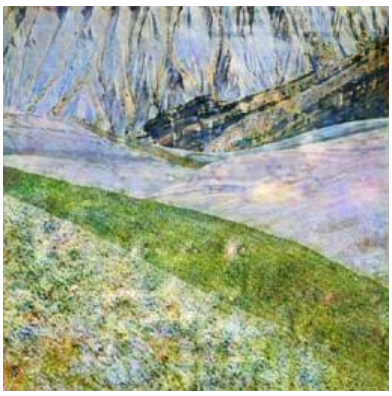
- With **43 epochs** and **3000 steps** per epoch, the model is exposed to a significant amount of training data. However, if the generators are not improving, it may be beneficial to extend the training duration or implement early stopping based on validation metrics.
- Improvement Suggestion: Monitor the training process closely and consider extending the number of epochs if the model shows signs of improvement.

14. Conclusion

- The training logs indicate that while the discriminators are learning effectively, the generators may not be keeping pace, leading to a potential imbalance in the training dynamics. By addressing the insights and suggestions outlined before, can enhance the performance of this CycleGAN model with better Cycle, ensuring that both generators and discriminators learn effectively and produce high-quality results in image-to-image translation tasks.
- Continuous monitoring and adjustments based on the training logs and adding additional monitoring matrix such as FID or MiFID will be key to achieving optimal performance.

> Randomly Selected Output Images:




>

> **"What I cannot create, I do not understand."**
>
> *- Richard Feynman*
> *The theoretical physicist who received the Nobel prize in 1965 for his work developing quantum electrodynamics*

In [ ]: