# DTSA-5511 Week 5 - I'm Something of a Painter Myself

A Jupyter notebook with a description of the problem/data, exploratory data analysis (EDA) procedure, analysis (model building and training), result, and discussion/conclusion.

## Project description

A GAN consists of at least two neural networks: a generator model and a discriminator model. The generator is a neural network that creates the images. For our competition, you should generate images in the style of Monet. This generator is trained using a discriminator.

The two models will work against each other, with the generator trying to trick the discriminator, and the discriminator trying to accurately classify the real vs. generated images.

Your task is to build a GAN that generates 7,000 to 10,000 Monet-style images.

## Dataset Description

The dataset contains four directories: monet_tfrec, photo_tfrec, monet_jpg, and photo_jpg. The monet_tfrec and monet_jpg directories contain the same painting images, and the photo_tfrec and photo_jpg directories contain the same photos.

The monet directories contain Monet paintings.

The photo directories contain photos.

### Files

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

## 1. Exploratory Data Analysis (EDA)

### 1.1. Importing libraries

Loading libraries required for this mini-project.

In [1]:
```
#utilities
import numpy as np
import random
import re
import pandas as pd
```

```
import PIL

#plots
import matplotlib.pyplot as plt

# modeling
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa

from kaggle_datasets import KaggleDatasets

# Remove warnings
import warnings
warnings.filterwarnings('ignore')

# detect TPU device
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Device:', tpu.master())
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
except:
    strategy = tf.distribute.get_strategy()
print('Number of replicas:', strategy.num_replicas_in_sync)

# dynamically tune value at runtime
AUTOTUNE = tf.data.experimental.AUTOTUNE

# tensorflow version
print(tf.__version__)
```

```
2022-10-02 02:36:40.155299: W tensorflow/stream_executor/platform/default/dso_loader.cc:
60] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cann
ot open shared object file: No such file or directory; LD_LIBRARY_PATH: /opt/conda/lib
2022-10-02 02:36:40.155366: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore
above cudart dlerror if you do not have a GPU set up on your machine.
Device: grpc://10.0.0.2:8470
2022-10-02 02:36:41.942043: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating
XLA devices, tf_xla_enable_xla_devices not set
2022-10-02 02:36:41.942345: W tensorflow/stream_executor/platform/default/dso_loader.cc:
60] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open sh
ared object file: No such file or directory; LD_LIBRARY_PATH: /opt/conda/lib
2022-10-02 02:36:41.942367: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed
call to cuInit: UNKNOWN ERROR (303)
2022-10-02 02:36:41.942390: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] k
ernel driver does not appear to be running on this host (c97d3990c257): /proc/driver/nvi
dia/version does not exist
2022-10-02 02:36:41.943031: I tensorflow/core/platform/cpu_feature_guard.cc:142] This Te
nsorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler fla
gs.
2022-10-02 02:36:41.943461: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating
XLA devices, tf_xla_enable_xla_devices not set
2022-10-02 02:36:41.958258: I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:30
1] Initialize GrpcChannelCache for job worker -> {0 -> 10.0.0.2:8470}
```

```
2022-10-02 02:36:41.958322: I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:30
1] Initialize GrpcChannelCache for job localhost -> {0 -> localhost:30313}
2022-10-02 02:36:41.972947: I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:30
1] Initialize GrpcChannelCache for job worker -> {0 -> 10.0.0.2:8470}
2022-10-02 02:36:41.973002: I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:30
1] Initialize GrpcChannelCache for job localhost -> {0 -> localhost:30313}
2022-10-02 02:36:41.973542: I tensorflow/core/distributed_runtime/rpc/grpc_server_lib.c
c:411] Started server with target: grpc://localhost:30313
Number of replicas: 8
2.4.1
```

## 1.2. TFRecords

Reading the photo and Monet images on the TFRecords files and creating functions for loading those images.

In [2]:

```
IMAGE_SIZE = [256, 256]
BATCH_SIZE = 16

# Load the data
gcs_path = KaggleDatasets().get_gcs_path()

monet_files = tf.io.gfile.glob(str(gcs_path + '/monet_tfrec/*.tfrec'))
print('Monet TFRecord Files:', len(monet_files))

photo_files = tf.io.gfile.glob(str(gcs_path + '/photo_tfrec/*.tfrec'))
print('Photo TFRecord Files:', len(photo_files))
```

```
Monet TFRecord Files: 5
Photo TFRecord Files: 20
2022-10-02 02:37:09.354122: I tensorflow/core/platform/cloud/google_auth_provider.cc:18
0] Attempting an empty bearer token since no token was retrieved from files, and GCE met
adata check was skipped.
2022-10-02 02:37:09.423265: I tensorflow/core/platform/cloud/google_auth_provider.cc:18
0] Attempting an empty bearer token since no token was retrieved from files, and GCE met
adata check was skipped.
```

In [3]:

```
# Function to count the number of photo and Monet images
def count_data_items(filenames):
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1)) for filename in file
    return np.sum(n)

n_monet_samples = count_data_items(monet_files)
n_photo_samples = count_data_items(photo_files)
```

In [4]:

```
# Function to decode a JPEG-encoded image to an RGB channel
def decode_image(image):
    image = tf.image.decode_jpeg(image, channels=3)
    image = (tf.cast(image, tf.float32) / 127.5) - 1
    image = tf.reshape(image, [*IMAGE_SIZE, 3])
    return image

# Function to parse a simple example photo to decode the image
def read_tfrecord(example):
    tfrecord_format = {
        "image_name": tf.io.FixedLenFeature([], tf.string),
```

```
        "image": tf.io.FixedLenFeature([], tf.string),
        "target": tf.io.FixedLenFeature([], tf.string)
    }
    example = tf.io.parse_single_example(example, tfrecord_format)
    image = decode_image(example['image'])
    return image

# Function to extract image from files
def load_dataset(filenames, labeled=True, ordered=False):
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
    return dataset
```

## 1.3. Image visualization

Visualizing three samples of photo and Monet images.

In [5]:
```
# Loading the Monet images
monet_ds = load_dataset(monet_files, labeled=True).batch(10)
# Loading the photo images
photo_ds = load_dataset(photo_files, labeled=True).batch(10)
```

In [6]:
```
example_monet = next(iter(monet_ds))
example_photo = next(iter(photo_ds))
```

In [7]:
```
#number = random.randint(0, 9)
plt.figure(figsize=(10, 7))

plt.subplot(231)
plt.title('Photo 1')
plt.axis('off')
plt.imshow(example_photo[0] * 0.5 + 0.5)
plt.subplot(232)
plt.axis('off')
plt.title('Photo 2')
plt.imshow(example_photo[1] * 0.5 + 0.5)
plt.subplot(233)
plt.axis('off')
plt.title('Photo 3')
plt.imshow(example_photo[2] * 0.5 + 0.5)

plt.subplot(234)
plt.title('Monet 1')
plt.axis('off')
plt.imshow(example_monet[0] * 0.5 + 0.5)
plt.subplot(235)
plt.title('Monet 2')
plt.axis('off')
plt.imshow(example_monet[1] * 0.5 + 0.5)
plt.subplot(236)
plt.title('Monet 3')
plt.axis('off')
plt.imshow(example_monet[2] * 0.5 + 0.5)
```

```
<matplotlib.image.AxesImage at 0x7f501074b2d0>
```

Out[7]:


# 2. CycleGAN

Implementing CycleGAN because it is a very popular GAN architecture used to learn transformation between images of different styles.

## 2.1. Building RestNet architecture

This architecture introduces the concept of Residual Blocks to solve the problem of vanishing/exploring gradient.

In [8]:

```
OUTPUT_CHANNELS = 3

# Reducing the features of an array or an image.
def downsample(filters, size, apply_instancenorm=True):
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)

    result = keras.Sequential()
    result.add(layers.Conv2D(filters, size, strides=2, padding='same',
                             kernel_initializer=initializer, use_bias=False))

    if apply_instancenorm:
        result.add(tfa.layers.InstanceNormalization(gamma_initializer=gamma_init))

    result.add(layers.LeakyReLU())

    return result

# Inserting null-values between original values to increase the sampling rate
```

```
def upsample(filters, size, apply_dropout=False):
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)

    result = keras.Sequential()
    result.add(layers.Conv2DTranspose(filters, size, strides=2,
                                      padding='same',
                                      kernel_initializer=initializer,
                                      use_bias=False))

    result.add(tfa.layers.InstanceNormalization(gamma_initializer=gamma_init))

    if apply_dropout:
        result.add(layers.Dropout(0.5))

    result.add(layers.ReLU())

    return result

# Generating ResNet
def ResNetGenerator():
    inputs = layers.Input(shape=[256,256,3])

    # bs = batch size
    down_stack = [
        downsample(64, 4, apply_instancenorm=False), # (bs, 128, 128, 64)
        downsample(128, 4), # (bs, 64, 64, 128)
        downsample(256, 4), # (bs, 32, 32, 256)
        downsample(512, 4), # (bs, 16, 16, 512)
        downsample(512, 4), # (bs, 8, 8, 512)
        downsample(512, 4), # (bs, 4, 4, 512)
        downsample(512, 4), # (bs, 2, 2, 512)
        downsample(512, 4), # (bs, 1, 1, 512)
    ]

    up_stack = [
        upsample(512, 4, apply_dropout=True), # (bs, 2, 2, 1024)
        upsample(512, 4, apply_dropout=True), # (bs, 4, 4, 1024)
        upsample(512, 4, apply_dropout=True), # (bs, 8, 8, 1024)
        upsample(512, 4), # (bs, 16, 16, 1024)
        upsample(256, 4), # (bs, 32, 32, 512)
        upsample(128, 4), # (bs, 64, 64, 256)
        upsample(64, 4), # (bs, 128, 128, 128)
    ]

    initializer = tf.random_normal_initializer(0., 0.02)
    last = layers.Conv2DTranspose(OUTPUT_CHANNELS, 4,
                                  strides=2,
                                  padding='same',
                                  kernel_initializer=initializer,
                                  activation='tanh') # (bs, 256, 256, 3)

    x = inputs

    # Downsampling through the model
    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)
```

```
    skips = reversed(skips[:-1])

    # Upsampling and establishing the skip connections
    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = layers.Concatenate()([x, skip])

    x = last(x)

    return keras.Model(inputs=inputs, outputs=x)
```

In [9]:

```
#Building the ResNet and printing the model summary
generator = ResNetGenerator()
generator.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 256, 256, 3)      0
_________________________________________________________________
sequential (Sequential)      (None, 128, 128, 64)      3072
_________________________________________________________________
sequential_1 (Sequential)    (None, 64, 64, 128)       131328
_________________________________________________________________
sequential_2 (Sequential)    (None, 32, 32, 256)       524800
_________________________________________________________________
sequential_3 (Sequential)    (None, 16, 16, 512)       2098176
_________________________________________________________________
sequential_4 (Sequential)    (None, 8, 8, 512)         4195328
_________________________________________________________________
sequential_5 (Sequential)    (None, 4, 4, 512)         4195328
_________________________________________________________________
sequential_6 (Sequential)    (None, 2, 2, 512)         4195328
_________________________________________________________________
sequential_7 (Sequential)    (None, 1, 1, 512)         4195328
_________________________________________________________________
sequential_8 (Sequential)    (None, 2, 2, 512)         4195328
_________________________________________________________________
concatenate (Concatenate)    (None, 2, 2, 1024)        0
                                                       sequential_8[0][0]
                                                       sequential_6[0][0]
_________________________________________________________________
sequential_9 (Sequential)    (None, 4, 4, 512)         8389632
_________________________________________________________________
```

```
concatenate_1 (Concatenate)     (None, 4, 4, 1024)   0           sequential_9[0][0]
                                                                 sequential_5[0][0]
__________
sequential_10 (Sequential)      (None, 8, 8, 512)    8389632     concatenate_1[0][0]
__________
concatenate_2 (Concatenate)     (None, 8, 8, 1024)   0           sequential_10[0][0]
                                                                 sequential_4[0][0]
__________
sequential_11 (Sequential)      (None, 16, 16, 512)  8389632     concatenate_2[0][0]
__________
concatenate_3 (Concatenate)     (None, 16, 16, 1024) 0           sequential_11[0][0]
                                                                 sequential_3[0][0]
__________
sequential_12 (Sequential)      (None, 32, 32, 256)  4194816     concatenate_3[0][0]
__________
concatenate_4 (Concatenate)     (None, 32, 32, 512)  0           sequential_12[0][0]
                                                                 sequential_2[0][0]
__________
sequential_13 (Sequential)      (None, 64, 64, 128)  1048832     concatenate_4[0][0]
__________
concatenate_5 (Concatenate)     (None, 64, 64, 256)  0           sequential_13[0][0]
                                                                 sequential_1[0][0]
__________
sequential_14 (Sequential)      (None, 128, 128, 64) 262272      concatenate_5[0][0]
__________
concatenate_6 (Concatenate)     (None, 128, 128, 128 0           sequential_14[0][0]
                                                                 sequential[0][0]
__________
conv2d_transpose_7 (Conv2DTrans (None, 256, 256, 3)  6147        concatenate_6[0][0]
==================================================================================================
Total params: 54,414,979
Trainable params: 54,414,979
Non-trainable params: 0
__________________________________________________________________________________________________
```

## 2.2. Building the discriminator

The discriminator is used to distinguish real data from the data created by the generator.

In [10]:

```
# Function to create the discriminator
def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)
```

```
    inp = layers.Input(shape=[256, 256, 3], name='input_image')

    x = inp

    down1 = downsample(64, 4, False)(x) # (bs, 128, 128, 64)
    down2 = downsample(128, 4)(down1) # (bs, 64, 64, 128)
    down3 = downsample(256, 4)(down2) # (bs, 32, 32, 256)

    zero_pad1 = layers.ZeroPadding2D()(down3) # (bs, 34, 34, 256)
    conv = layers.Conv2D(512, 4, strides=1,
                         kernel_initializer=initializer,
                         use_bias=False)(zero_pad1) # (bs, 31, 31, 512)

    norm1 = tfa.layers.InstanceNormalization(gamma_initializer=gamma_init)(conv)

    leaky_relu = layers.LeakyReLU()(norm1)

    zero_pad2 = layers.ZeroPadding2D()(leaky_relu) # (bs, 33, 33, 512)

    last = layers.Conv2D(1, 4, strides=1,
                         kernel_initializer=initializer)(zero_pad2) # (bs, 30, 30, 1)

    return tf.keras.Model(inputs=inp, outputs=last)
```

In [11]:

```
# Generating the discriminator
discriminator = Discriminator()
discriminator.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_image (InputLayer)     [(None, 256, 256, 3)]     0
_________________________________________________________________
sequential_15 (Sequential)   (None, 128, 128, 64)      3072
_________________________________________________________________
sequential_16 (Sequential)   (None, 64, 64, 128)       131328
_________________________________________________________________
sequential_17 (Sequential)   (None, 32, 32, 256)       524800
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 34, 34, 256)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 31, 31, 512)       2097152
_________________________________________________________________
instance_normalization_16 (I (None, 31, 31, 512)       1024
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 31, 31, 512)       0
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 33, 33, 512)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 30, 30, 1)         8193
=================================================================
Total params: 2,765,569
Trainable params: 2,765,569
Non-trainable params: 0
_________________________________________________________________
```

In [12]:

```
with strategy.scope():
```

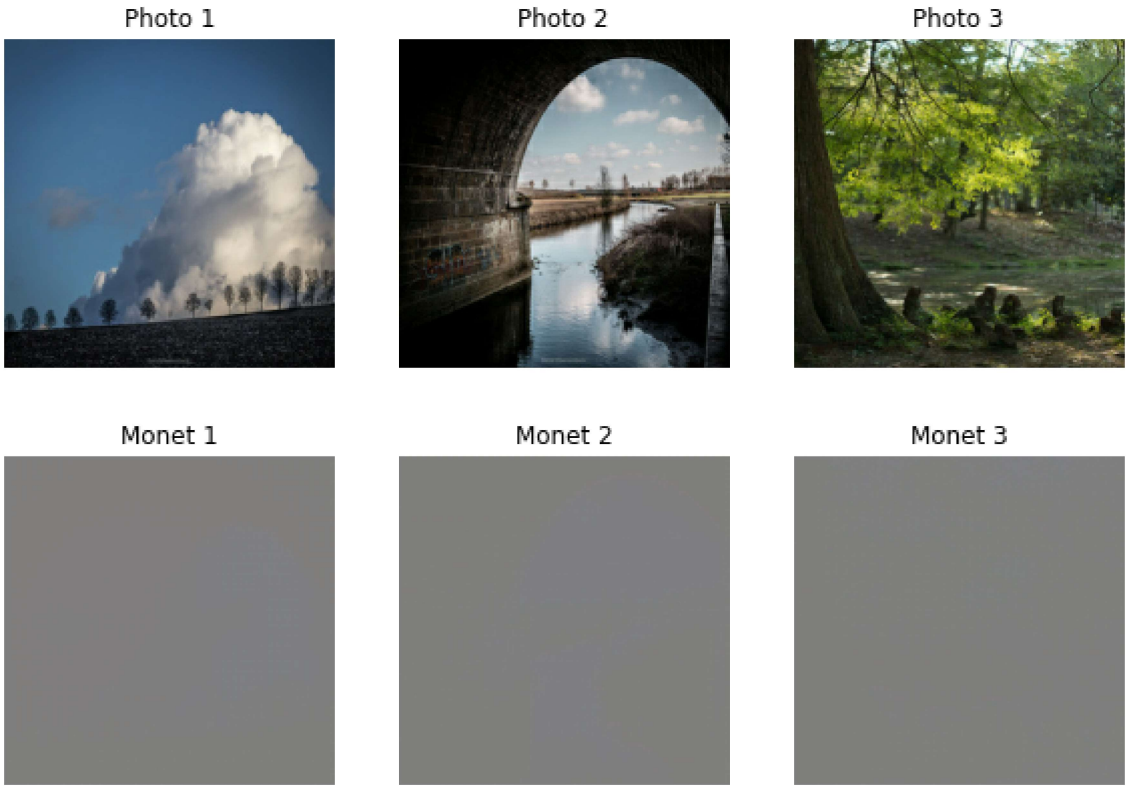
```
    monet_generator = ResNetGenerator() # transforms photos to Monet-esque paintings
    photo_generator = ResNetGenerator() # transforms Monet paintings to be more like ph

    monet_discriminator = Discriminator() # differentiates real Monet paintings and gen
    photo_discriminator = Discriminator() # differentiates real photos and generated ph
```

In [13]:

```
# Visualization
plt.figure(figsize=(10, 7))
to_monet = monet_generator(example_photo)

plt.subplot(231)
plt.title('Photo 1')
plt.axis('off')
plt.imshow(example_photo[0] * 0.5 + 0.5)
plt.subplot(232)
plt.axis('off')
plt.title('Photo 2')
plt.imshow(example_photo[1] * 0.5 + 0.5)
plt.subplot(233)
plt.axis('off')
plt.title('Photo 3')
plt.imshow(example_photo[2] * 0.5 + 0.5)

plt.subplot(234)
plt.title('Monet 1')
plt.axis('off')
plt.imshow(to_monet[0] * 0.5 + 0.5)
plt.subplot(235)
plt.title('Monet 2')
plt.axis('off')
plt.imshow(to_monet[1] * 0.5 + 0.5)
plt.subplot(236)
plt.title('Monet 3')
plt.axis('off')
plt.imshow(to_monet[2] * 0.5 + 0.5)
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x7f4fec44d650>
```




Since our generators are not trained yet, the generated Monet-esque photo does not show what is expected at this point.

## 2.3. Building the cGAN model

Overriding the train_step() method of the Model class for training via fit().

In [14]:

```
# CycleGan class
class CycleGan(keras.Model):
    def __init__(
        self,
        monet_generator,
        photo_generator,
        monet_discriminator,
        photo_discriminator,
        lambda_cycle=10,
    ):
        super(CycleGan, self).__init__()
        self.m_gen = monet_generator
        self.p_gen = photo_generator
        self.m_disc = monet_discriminator
        self.p_disc = photo_discriminator
        self.lambda_cycle = lambda_cycle

    def compile(
        self,
        m_gen_optimizer,
        p_gen_optimizer,
        m_disc_optimizer,
        p_disc_optimizer,
        gen_loss_fn,
        disc_loss_fn,
```

```
        cycle_loss_fn,
        identity_loss_fn
    ):
        super(CycleGan, self).compile()
        self.m_gen_optimizer = m_gen_optimizer
        self.p_gen_optimizer = p_gen_optimizer
        self.m_disc_optimizer = m_disc_optimizer
        self.p_disc_optimizer = p_disc_optimizer
        self.gen_loss_fn = gen_loss_fn
        self.disc_loss_fn = disc_loss_fn
        self.cycle_loss_fn = cycle_loss_fn
        self.identity_loss_fn = identity_loss_fn

    def train_step(self, batch_data):
        real_monet, real_photo = batch_data

        with tf.GradientTape(persistent=True) as tape:
            # photo to monet back to photo
            fake_monet = self.m_gen(real_photo, training=True)
            cycled_photo = self.p_gen(fake_monet, training=True)

            # monet to photo back to monet
            fake_photo = self.p_gen(real_monet, training=True)
            cycled_monet = self.m_gen(fake_photo, training=True)

            # generating itself
            same_monet = self.m_gen(real_monet, training=True)
            same_photo = self.p_gen(real_photo, training=True)

            # discriminator used to check, inputing real images
            disc_real_monet = self.m_disc(real_monet, training=True)
            disc_real_photo = self.p_disc(real_photo, training=True)

            # discriminator used to check, inputing fake images
            disc_fake_monet = self.m_disc(fake_monet, training=True)
            disc_fake_photo = self.p_disc(fake_photo, training=True)

            # evaluates generator loss
            monet_gen_loss = self.gen_loss_fn(disc_fake_monet)
            photo_gen_loss = self.gen_loss_fn(disc_fake_photo)

            # evaluates total cycle consistency loss
            total_cycle_loss = self.cycle_loss_fn(real_monet, cycled_monet, self.lambda

            # evaluates total generator loss
            total_monet_gen_loss = monet_gen_loss + total_cycle_loss + self.identity_lo
            total_photo_gen_loss = photo_gen_loss + total_cycle_loss + self.identity_lo

            # evaluates discriminator loss
            monet_disc_loss = self.disc_loss_fn(disc_real_monet, disc_fake_monet)
            photo_disc_loss = self.disc_loss_fn(disc_real_photo, disc_fake_photo)

        # Calculate the gradients for generator and discriminator
        monet_generator_gradients = tape.gradient(total_monet_gen_loss,
                                                  self.m_gen.trainable_variables)
        photo_generator_gradients = tape.gradient(total_photo_gen_loss,
                                                  self.p_gen.trainable_variables)

        monet_discriminator_gradients = tape.gradient(monet_disc_loss,
                                                      self.m_disc.trainable_variables)
```

```
        photo_discriminator_gradients = tape.gradient(photo_disc_loss,
                                                      self.p_disc.trainable_variables)

        # Apply the gradients to the optimizer
        self.m_gen_optimizer.apply_gradients(zip(monet_generator_gradients,
                                                 self.m_gen.trainable_variables))

        self.p_gen_optimizer.apply_gradients(zip(photo_generator_gradients,
                                                 self.p_gen.trainable_variables))

        self.m_disc_optimizer.apply_gradients(zip(monet_discriminator_gradients,
                                                  self.m_disc.trainable_variables))

        self.p_disc_optimizer.apply_gradients(zip(photo_discriminator_gradients,
                                                  self.p_disc.trainable_variables))

        return {
            "monet_gen_loss": total_monet_gen_loss,
            "photo_gen_loss": total_photo_gen_loss,
            "monet_disc_loss": monet_disc_loss,
            "photo_disc_loss": photo_disc_loss
        }
```

# 3. Training

Fitting the best weights and biases to CGAN model to minimize the loss function over prediction range

In [15]:

```
with strategy.scope():
    # Returns the gradients of the loss with respect to the learnable parameters in the
    def discriminator_loss(real, generated):
        real_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True, reduction=tf.k

        generated_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True, reduction

        total_disc_loss = real_loss + generated_loss

        return total_disc_loss * 0.5

with strategy.scope():
    # Goes through the discriminator and gets classified as either “Real” or “Fake” bas
    def generator_loss(generated):
        return tf.keras.losses.BinaryCrossentropy(from_logits=True, reduction=tf.keras.

with strategy.scope():
    # Calculate the cycle consistency loss be finding the average of their difference.
    def calc_cycle_loss(real_image, cycled_image, LAMBDA):
        loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))

        return LAMBDA * loss1

with strategy.scope():
    # Compares the input with the output of the generator.
    def identity_loss(real_image, same_image, LAMBDA):
        loss = tf.reduce_mean(tf.abs(real_image - same_image))
        return LAMBDA * 0.5 * loss
```

In [16]:
```
with strategy.scope():
    monet_generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
    photo_generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

    monet_discriminator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
    photo_discriminator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
```

In [17]:
```
with strategy.scope():
    cycle_gan_model = CycleGan(
        monet_generator, photo_generator, monet_discriminator, photo_discriminator
    )

    cycle_gan_model.compile(
        m_gen_optimizer = monet_generator_optimizer,
        p_gen_optimizer = photo_generator_optimizer,
        m_disc_optimizer = monet_discriminator_optimizer,
        p_disc_optimizer = photo_discriminator_optimizer,
        gen_loss_fn = generator_loss,
        disc_loss_fn = discriminator_loss,
        cycle_loss_fn = calc_cycle_loss,
        identity_loss_fn = identity_loss
    )
```

# 4. Prediction and Submission

Generating the Monet images and submitting the images.

In [18]:
```
# Train the model for 50 epochs.
history1 = cycle_gan_model.fit(
    tf.data.Dataset.zip((monet_ds, photo_ds)),
    epochs=100
).history
```

```
Epoch 1/100
30/30 [==============================] - 110s 127ms/step - monet_gen_loss: 10.2396 - pho
to_gen_loss: 10.7708 - monet_disc_loss: 0.6603 - photo_disc_loss: 0.6552
Epoch 2/100
 1/30 [>.............................] - ETA: 4s - monet_gen_loss: 6.5794 - photo_gen_lo
ss: 6.5716 - monet_disc_loss: 0.6514 - photo_disc_loss: 0.6085
2022-10-02 02:40:10.686212: W ./tensorflow/core/distributed_runtime/eager/destroy_tensor
_handle_node.h:57] Ignoring an error encountered when deleting remote tensors handles: I
nvalid argument: Unable to find the relevant tensor remote_handle: Op ID: 11295, Output
num: 0
Additional GRPC error information from remote target /job:worker/replica:0/task:0:
:{"created":"@1664678410.682832454","description":"Error received from peer ipv4:10.0.0.
2:8470","file":"external/com_github_grpc_grpc/src/core/lib/surface/call.cc","file_line":
1056,"grpc_message":"Unable to find the relevant tensor remote_handle: Op ID: 11295, Out
put num: 0","grpc_status":3}
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 5.4800 - photo_
gen_loss: 5.7299 - monet_disc_loss: 0.5964 - photo_disc_loss: 0.5785
Epoch 3/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 5.2647 - photo_
gen_loss: 5.5166 - monet_disc_loss: 0.5418 - photo_disc_loss: 0.5032
```

```
Epoch 4/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.9640 - photo_
gen_loss: 5.2243 - monet_disc_loss: 0.6414 - photo_disc_loss: 0.5540
Epoch 5/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.7816 - photo_
gen_loss: 4.9205 - monet_disc_loss: 0.5443 - photo_disc_loss: 0.5323
Epoch 6/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.5648 - photo_
gen_loss: 4.8819 - monet_disc_loss: 0.5883 - photo_disc_loss: 0.4048
Epoch 7/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.2725 - photo_
gen_loss: 4.7553 - monet_disc_loss: 0.6147 - photo_disc_loss: 0.3448
Epoch 8/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.1104 - photo_
gen_loss: 4.3476 - monet_disc_loss: 0.5558 - photo_disc_loss: 0.5051
Epoch 9/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.0830 - photo_
gen_loss: 4.0021 - monet_disc_loss: 0.4862 - photo_disc_loss: 0.6551
Epoch 10/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.1479 - photo_
gen_loss: 4.0923 - monet_disc_loss: 0.4060 - photo_disc_loss: 0.5363
Epoch 11/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.2753 - photo_
gen_loss: 4.3584 - monet_disc_loss: 0.3562 - photo_disc_loss: 0.4076
Epoch 12/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.4193 - photo_
gen_loss: 4.5006 - monet_disc_loss: 0.3761 - photo_disc_loss: 0.3299
Epoch 13/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.4483 - photo_
gen_loss: 4.6060 - monet_disc_loss: 0.4645 - photo_disc_loss: 0.3058
Epoch 14/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.4844 - photo_
gen_loss: 4.6179 - monet_disc_loss: 0.3535 - photo_disc_loss: 0.3476
Epoch 15/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.4514 - photo_
gen_loss: 4.5803 - monet_disc_loss: 0.3735 - photo_disc_loss: 0.3493
Epoch 16/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.5034 - photo_
gen_loss: 4.5021 - monet_disc_loss: 0.4844 - photo_disc_loss: 0.4175
Epoch 17/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.6273 - photo_
gen_loss: 4.5863 - monet_disc_loss: 0.3936 - photo_disc_loss: 0.4312
Epoch 18/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.4044 - photo_
gen_loss: 4.4199 - monet_disc_loss: 0.5390 - photo_disc_loss: 0.4941
Epoch 19/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.2448 - photo_
gen_loss: 4.4235 - monet_disc_loss: 0.5435 - photo_disc_loss: 0.4851
Epoch 20/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.1183 - photo_
gen_loss: 4.4222 - monet_disc_loss: 0.5003 - photo_disc_loss: 0.4191
Epoch 21/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.4390 - photo_
gen_loss: 4.5313 - monet_disc_loss: 0.4753 - photo_disc_loss: 0.4309
Epoch 22/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.2273 - photo_
gen_loss: 4.5711 - monet_disc_loss: 0.5876 - photo_disc_loss: 0.3821
Epoch 23/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.3185 - photo_
gen_loss: 4.3250 - monet_disc_loss: 0.4302 - photo_disc_loss: 0.5171
```

```
Epoch 24/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.4183 - photo_
gen_loss: 4.3664 - monet_disc_loss: 0.3975 - photo_disc_loss: 0.4856
Epoch 25/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.3264 - photo_
gen_loss: 4.2333 - monet_disc_loss: 0.4063 - photo_disc_loss: 0.4330
Epoch 26/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.4340 - photo_
gen_loss: 4.2715 - monet_disc_loss: 0.3783 - photo_disc_loss: 0.4585
Epoch 27/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.2510 - photo_
gen_loss: 4.3467 - monet_disc_loss: 0.5118 - photo_disc_loss: 0.3772
Epoch 28/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.2039 - photo_
gen_loss: 4.3752 - monet_disc_loss: 0.5018 - photo_disc_loss: 0.4660
Epoch 29/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.2400 - photo_
gen_loss: 4.4833 - monet_disc_loss: 0.5268 - photo_disc_loss: 0.4090
Epoch 30/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.0980 - photo_
gen_loss: 4.4374 - monet_disc_loss: 0.5615 - photo_disc_loss: 0.3893
Epoch 31/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.0014 - photo_
gen_loss: 4.4424 - monet_disc_loss: 0.6032 - photo_disc_loss: 0.4129
Epoch 32/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.9578 - photo_
gen_loss: 4.3751 - monet_disc_loss: 0.5786 - photo_disc_loss: 0.3901
Epoch 33/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.0356 - photo_
gen_loss: 4.6684 - monet_disc_loss: 0.5785 - photo_disc_loss: 0.3739
Epoch 34/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 4.0379 - photo_
gen_loss: 4.4705 - monet_disc_loss: 0.5671 - photo_disc_loss: 0.3729
Epoch 35/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 4.0237 - photo_
gen_loss: 4.4614 - monet_disc_loss: 0.5621 - photo_disc_loss: 0.4444
Epoch 36/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.8837 - photo_
gen_loss: 4.3235 - monet_disc_loss: 0.5772 - photo_disc_loss: 0.3932
Epoch 37/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.9236 - photo_
gen_loss: 4.2497 - monet_disc_loss: 0.5809 - photo_disc_loss: 0.5426
Epoch 38/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.8922 - photo_
gen_loss: 4.1922 - monet_disc_loss: 0.5572 - photo_disc_loss: 0.5055
Epoch 39/100
30/30 [==============================] - 4s 128ms/step - monet_gen_loss: 3.8873 - photo_
gen_loss: 4.0957 - monet_disc_loss: 0.5318 - photo_disc_loss: 0.4482
Epoch 40/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.8740 - photo_
gen_loss: 4.0233 - monet_disc_loss: 0.5404 - photo_disc_loss: 0.4803
Epoch 41/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.8501 - photo_
gen_loss: 4.0673 - monet_disc_loss: 0.5540 - photo_disc_loss: 0.4424
Epoch 42/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.8718 - photo_
gen_loss: 4.1206 - monet_disc_loss: 0.5520 - photo_disc_loss: 0.4364
Epoch 43/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.9342 - photo_
gen_loss: 4.2717 - monet_disc_loss: 0.5783 - photo_disc_loss: 0.4045
```

```
Epoch 44/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.8884 - photo_
gen_loss: 4.1717 - monet_disc_loss: 0.5661 - photo_disc_loss: 0.4365
Epoch 45/100
30/30 [==============================] - 4s 128ms/step - monet_gen_loss: 3.8645 - photo_
gen_loss: 4.0944 - monet_disc_loss: 0.5398 - photo_disc_loss: 0.5554
Epoch 46/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.8396 - photo_
gen_loss: 4.0297 - monet_disc_loss: 0.5683 - photo_disc_loss: 0.4895
Epoch 47/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.8305 - photo_
gen_loss: 4.0820 - monet_disc_loss: 0.5516 - photo_disc_loss: 0.5222
Epoch 48/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.7425 - photo_
gen_loss: 3.9013 - monet_disc_loss: 0.5636 - photo_disc_loss: 0.4788
Epoch 49/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.6943 - photo_
gen_loss: 3.8157 - monet_disc_loss: 0.5704 - photo_disc_loss: 0.5426
Epoch 50/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.6746 - photo_
gen_loss: 3.9741 - monet_disc_loss: 0.5902 - photo_disc_loss: 0.4393
Epoch 51/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.7174 - photo_
gen_loss: 4.0434 - monet_disc_loss: 0.5703 - photo_disc_loss: 0.4601
Epoch 52/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.6243 - photo_
gen_loss: 3.9023 - monet_disc_loss: 0.6269 - photo_disc_loss: 0.4989
Epoch 53/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.5623 - photo_
gen_loss: 3.8427 - monet_disc_loss: 0.6101 - photo_disc_loss: 0.4776
Epoch 54/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.5264 - photo_
gen_loss: 4.0212 - monet_disc_loss: 0.6371 - photo_disc_loss: 0.5179
Epoch 55/100
30/30 [==============================] - 4s 128ms/step - monet_gen_loss: 3.4732 - photo_
gen_loss: 3.8365 - monet_disc_loss: 0.6288 - photo_disc_loss: 0.4917
Epoch 56/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.4189 - photo_
gen_loss: 3.7464 - monet_disc_loss: 0.6276 - photo_disc_loss: 0.4940
Epoch 57/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.4239 - photo_
gen_loss: 3.7717 - monet_disc_loss: 0.6088 - photo_disc_loss: 0.4902
Epoch 58/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.4429 - photo_
gen_loss: 3.7405 - monet_disc_loss: 0.5993 - photo_disc_loss: 0.5262
Epoch 59/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.4038 - photo_
gen_loss: 3.7178 - monet_disc_loss: 0.6180 - photo_disc_loss: 0.4973
Epoch 60/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.3849 - photo_
gen_loss: 3.7063 - monet_disc_loss: 0.6122 - photo_disc_loss: 0.4657
Epoch 61/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.3585 - photo_
gen_loss: 3.7079 - monet_disc_loss: 0.6093 - photo_disc_loss: 0.5167
Epoch 62/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.4340 - photo_
gen_loss: 3.7587 - monet_disc_loss: 0.6010 - photo_disc_loss: 0.4671
Epoch 63/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.3666 - photo_
gen_loss: 3.7070 - monet_disc_loss: 0.6170 - photo_disc_loss: 0.4572
```

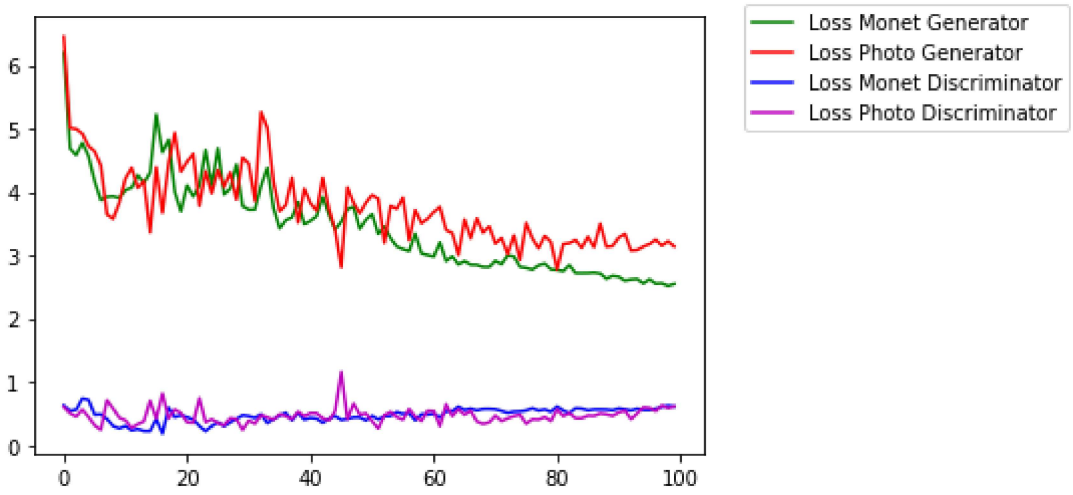
```
Epoch 64/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.3599 - photo_
gen_loss: 3.6940 - monet_disc_loss: 0.6140 - photo_disc_loss: 0.5179
Epoch 65/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.3231 - photo_
gen_loss: 3.5934 - monet_disc_loss: 0.6344 - photo_disc_loss: 0.5162
Epoch 66/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.3155 - photo_
gen_loss: 3.6705 - monet_disc_loss: 0.6292 - photo_disc_loss: 0.4703
Epoch 67/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.3093 - photo_
gen_loss: 3.6389 - monet_disc_loss: 0.6239 - photo_disc_loss: 0.4843
Epoch 68/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.3002 - photo_
gen_loss: 3.6864 - monet_disc_loss: 0.6340 - photo_disc_loss: 0.5132
Epoch 69/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.2420 - photo_
gen_loss: 3.6860 - monet_disc_loss: 0.6356 - photo_disc_loss: 0.4369
Epoch 70/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.2397 - photo_
gen_loss: 3.6514 - monet_disc_loss: 0.6282 - photo_disc_loss: 0.4839
Epoch 71/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.2326 - photo_
gen_loss: 3.5205 - monet_disc_loss: 0.6216 - photo_disc_loss: 0.5013
Epoch 72/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.2358 - photo_
gen_loss: 3.5480 - monet_disc_loss: 0.6203 - photo_disc_loss: 0.4641
Epoch 73/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.2596 - photo_
gen_loss: 3.5563 - monet_disc_loss: 0.6094 - photo_disc_loss: 0.4897
Epoch 74/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.2204 - photo_
gen_loss: 3.6532 - monet_disc_loss: 0.6220 - photo_disc_loss: 0.5601
Epoch 75/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.2378 - photo_
gen_loss: 3.4804 - monet_disc_loss: 0.6209 - photo_disc_loss: 0.5197
Epoch 76/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.2231 - photo_
gen_loss: 3.4991 - monet_disc_loss: 0.6206 - photo_disc_loss: 0.5078
Epoch 77/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.2429 - photo_
gen_loss: 3.5526 - monet_disc_loss: 0.6182 - photo_disc_loss: 0.4854
Epoch 78/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.2282 - photo_
gen_loss: 3.4827 - monet_disc_loss: 0.6161 - photo_disc_loss: 0.4960
Epoch 79/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.2302 - photo_
gen_loss: 3.4811 - monet_disc_loss: 0.6136 - photo_disc_loss: 0.5292
Epoch 80/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.1995 - photo_
gen_loss: 3.4747 - monet_disc_loss: 0.6115 - photo_disc_loss: 0.4999
Epoch 81/100
30/30 [==============================] - 4s 128ms/step - monet_gen_loss: 3.1870 - photo_
gen_loss: 3.4260 - monet_disc_loss: 0.6181 - photo_disc_loss: 0.5298
Epoch 82/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.2062 - photo_
gen_loss: 3.4613 - monet_disc_loss: 0.6118 - photo_disc_loss: 0.5528
Epoch 83/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.1870 - photo_
gen_loss: 3.3970 - monet_disc_loss: 0.6147 - photo_disc_loss: 0.5455
```

```
Epoch 84/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.1533 - photo_
gen_loss: 3.3994 - monet_disc_loss: 0.6194 - photo_disc_loss: 0.5287
Epoch 85/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.1653 - photo_
gen_loss: 3.3708 - monet_disc_loss: 0.6123 - photo_disc_loss: 0.5336
Epoch 86/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.1524 - photo_
gen_loss: 3.3777 - monet_disc_loss: 0.6101 - photo_disc_loss: 0.5367
Epoch 87/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.1309 - photo_
gen_loss: 3.3484 - monet_disc_loss: 0.6165 - photo_disc_loss: 0.5436
Epoch 88/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.1352 - photo_
gen_loss: 3.4127 - monet_disc_loss: 0.6195 - photo_disc_loss: 0.5509
Epoch 89/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0834 - photo_
gen_loss: 3.3117 - monet_disc_loss: 0.6139 - photo_disc_loss: 0.5372
Epoch 90/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0840 - photo_
gen_loss: 3.3030 - monet_disc_loss: 0.6197 - photo_disc_loss: 0.5385
Epoch 91/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0839 - photo_
gen_loss: 3.3020 - monet_disc_loss: 0.6147 - photo_disc_loss: 0.5422
Epoch 92/100
30/30 [==============================] - 4s 130ms/step - monet_gen_loss: 3.0649 - photo_
gen_loss: 3.2945 - monet_disc_loss: 0.6130 - photo_disc_loss: 0.5526
Epoch 93/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.0510 - photo_
gen_loss: 3.2740 - monet_disc_loss: 0.6156 - photo_disc_loss: 0.5434
Epoch 94/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0628 - photo_
gen_loss: 3.2618 - monet_disc_loss: 0.6196 - photo_disc_loss: 0.5621
Epoch 95/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0334 - photo_
gen_loss: 3.2355 - monet_disc_loss: 0.6204 - photo_disc_loss: 0.5581
Epoch 96/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0212 - photo_
gen_loss: 3.2268 - monet_disc_loss: 0.6147 - photo_disc_loss: 0.5521
Epoch 97/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0180 - photo_
gen_loss: 3.2244 - monet_disc_loss: 0.6147 - photo_disc_loss: 0.5572
Epoch 98/100
30/30 [==============================] - 4s 126ms/step - monet_gen_loss: 3.0034 - photo_
gen_loss: 3.2204 - monet_disc_loss: 0.6167 - photo_disc_loss: 0.5714
Epoch 99/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0157 - photo_
gen_loss: 3.2150 - monet_disc_loss: 0.6176 - photo_disc_loss: 0.5565
Epoch 100/100
30/30 [==============================] - 4s 127ms/step - monet_gen_loss: 3.0110 - photo_
gen_loss: 3.2230 - monet_disc_loss: 0.6201 - photo_disc_loss: 0.5562
```

In [20]:

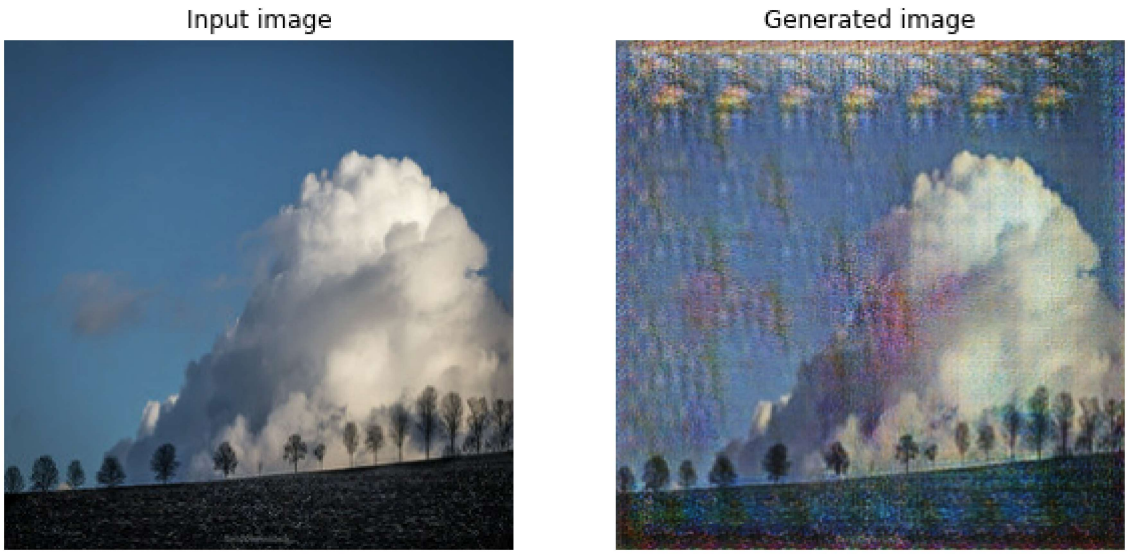
```
# Chart to display the loss generator and loss discriminator for photo and Monet images
loss_results_df = pd.DataFrame(history1)
loss_results_df = loss_results_df.applymap(np.mean)

plt.plot(loss_results_df.index, loss_results_df['monet_gen_loss'], color='g', label='Lo
plt.plot(loss_results_df.index, loss_results_df['photo_gen_loss'], color='r', label='Lo
plt.plot(loss_results_df.index, loss_results_df['monet_disc_loss'], color='b', label='L
```

```
plt.plot(loss_results_df.index, loss_results_df['photo_disc_loss'], color='m', label='L
plt.legend(loc='upper center', bbox_to_anchor=(1.3, 1.05))
plt.show()
```



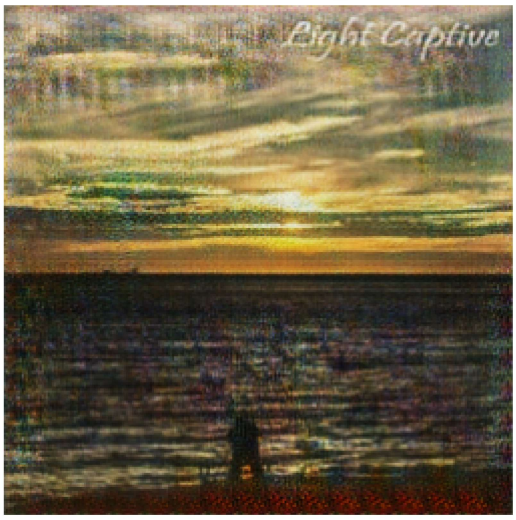

In [24]:

```
ds_iter = iter(photo_ds)
for n_sample in range(4):
        example_sample = next(ds_iter)
        generated_sample = monet_generator(example_sample)

        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()
```



Input image


Generated image




Input image

Generated image

Input image

Generated image

In [25]:
```
! mkdir ../images
```

In [26]:
```
i = 1
for img in photo_ds:
```

```
    prediction = monet_generator(img, training=False)[0].numpy()
    prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
    im = PIL.Image.fromarray(prediction)
    im.save("../images/" + str(i) + ".jpg")
    i += 1
```

```
2022-10-02 02:55:03.756071: W ./tensorflow/core/distributed_runtime/eager/destroy_tensor
_handle_node.h:57] Ignoring an error encountered when deleting remote tensors handles: I
nvalid argument: Unable to find the relevant tensor remote_handle: Op ID: 526912, Output
num: 0
Additional GRPC error information from remote target /job:worker/replica:0/task:0:
:{"created":"@1664679303.755997879","description":"Error received from peer ipv4:10.0.0.
2:8470","file":"external/com_github_grpc_grpc/src/core/lib/surface/call.cc","file_line":
1056,"grpc_message":"Unable to find the relevant tensor remote_handle: Op ID: 526912, Ou
tput num: 0","grpc_status":3}
```

In [27]:
```
import shutil
shutil.make_archive("/kaggle/working/images", 'zip', "/kaggle/images")
```

Out[27]:
```
'/kaggle/working/images.zip'
```

# 5. Discussion

## 5.1. Conclusion

By increasing the epochs numbers during the training, the generation of images looked more like Monet images. Thus, I selected 100 from values between 30 to 100.

## 5.2. Further improvement

For further improvement, I would like to add Differentiable Augmentation (DiffAugment). It enables the gradients to be propagated through the augmentation back to the generator, regularizes the discriminator without manipulating the target distribution, and maintains the balance of training dynamics. Three choices of transformation I would like to experiments which are translation, cutOut, and color.