CycleGAN for Monet Imitation

The purpose of this project is to create a Generative Adversarial Network to create imitation Monet paintings from photos, or in other words add Monet style to a photo. The GAN will consist of a generator and descriminator. The generator will create Monet Style photos and use the descriminator to learn. While the generator creates images the descriminator will determine if they are real or fake. The GAN need to create 7,000-10,000 images. The challenge is scored with Memorization-informed Fréchet Inception Distance(MiFID), which is a modification from Fréchet Inception Distance (FID). These are commonly used metrics for evaluating GANs, where a lower score is better.

This notebook utilizes a CycleGAN architecture to add Monet-style to photos. This is the kaggle competition page: https://www.kaggle.com/competitions/gan-getting-started

CycleGAN is a mothod that can capture the characteristics of one image domain and figure out how these characteristics could be translated into another image domain in the absence of paired training examples, using cycle-consistent adversarial networks. See TensorFlow and Keras CycleGAN documentation pages.

Github here: https://github.com/emhi6821/Week5_GAN/tree/main

In [22]:
```
import PIL
import shutil
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
import numpy as np
import glob
from keras.callbacks import CSVLogger


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)

AUTOTUNE = tf.data.experimental.AUTOTUNE

print(tf.__version__)
```

```
Number of replicas: 1
2.15.1
```

Load in the data & EDA

I start by loading in th photo dataset and Monet dataset fileames using glob for .tfrec files.

The competition says that the images are already sized to 256x256. Below I create a decoder for the images in the TFRecord files. These images are RGB images so the channel is set to 3. The images get scaled from -1 to 1. The GAN does not require labels so I get only return the images and get rid of the labels and ids.

In [6]:
```
top_path = r"/Users/emilyhill/Documents/WEEK5_GANS/gan-getting-started"
monet_tfrec = glob.glob(str(top_path + '/monet_tfrec/*.tfrec'))
print('Monet TFRecord Files:', len(monet_tfrec))

photo_tfrec = glob.glob(str(top_path + '/photo_tfrec/*.tfrec'))
print('Photo TFRecord Files:', len(photo_tfrec))
```

```
Monet TFRecord Files: 5
Photo TFRecord Files: 20
```

In [7]:
```
IMAGE_SIZE = [256, 256]

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
```

```
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
```

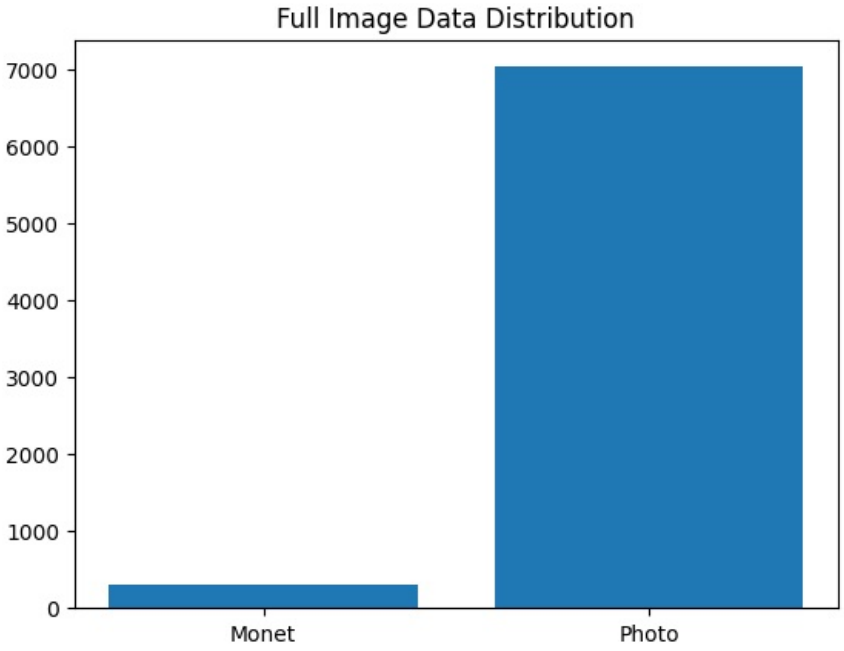
Next, I create a function to extract the images from the TFRecord files and load the monet and photo datasets into tensorflow datasets.

In [8]:
```
def load_dataset(filenames, labeled=True, ordered=False):
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
    return dataset
```

In [9]:
```
monet_ds = load_dataset(monet_tfrec, labeled=True).batch(1)
photo_ds = load_dataset(photo_tfrec, labeled=True).batch(1)

print('Monet Images: ', len(list(monet_ds)))
print('Photo Images: ', len(list(photo_ds)))
```

```
Monet Images:  300
Photo Images:  7038
```

In [10]:
```
plt.bar(['Monet', 'Photo'], [len(list(monet_ds)), len(list(photo_ds))])
plt.title('Full Image Data Distribution')
plt.show()
```



Let's visualize 5 photo and Monet examples. We can also confirm that the photos are 256x256

In [11]:
```
for example_monet, example_photo in zip(monet_ds.take(5), photo_ds.take(5)):
    #PLot Monet
    plt.subplot(122)
    plt.title('Monet')
    plt.imshow(example_monet[0] * 0.5 + 0.5)
    #Plot Photo
    plt.subplot(121)
    plt.title('Photo')
    plt.imshow(example_photo[0] * 0.5 + 0.5)
    plt.show()
```

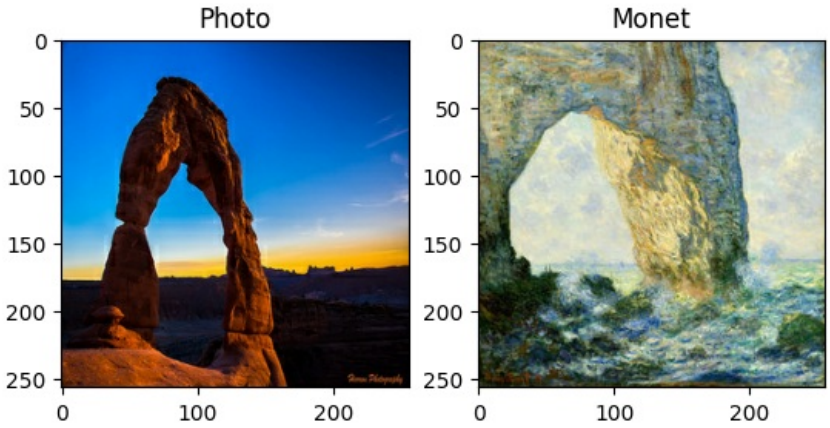


Photo
Monet
0
50
100
150
200
250
0
100
200

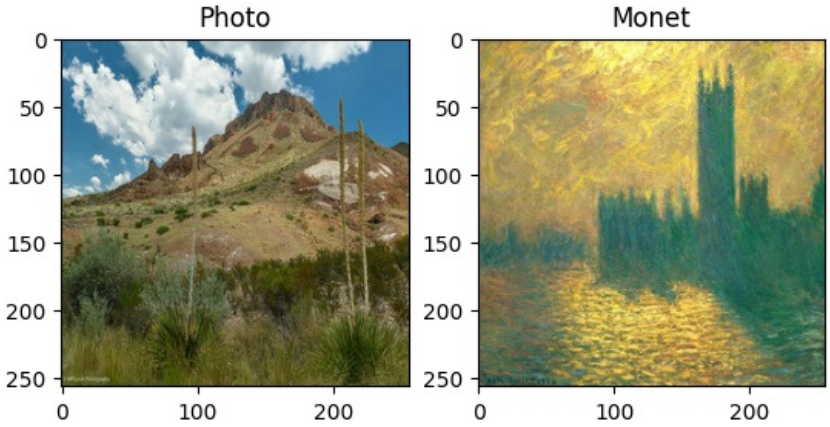


Photo
Monet
0
50
100
150
200
250
0
100
200

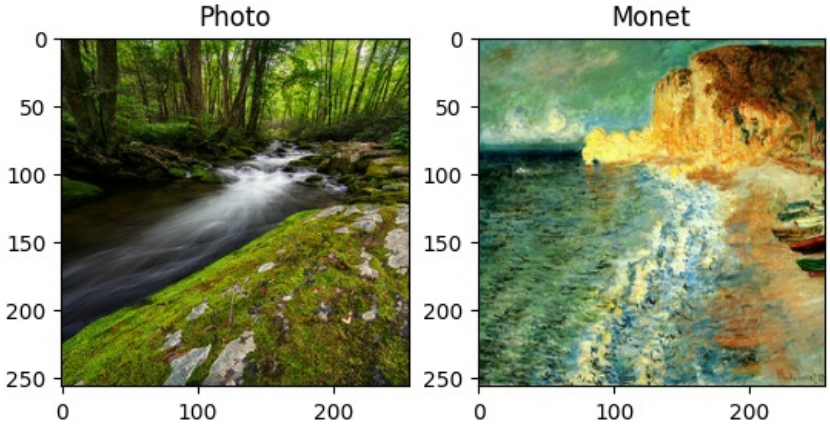


Photo
Monet
0
50
100
150
200
250
0
100
200

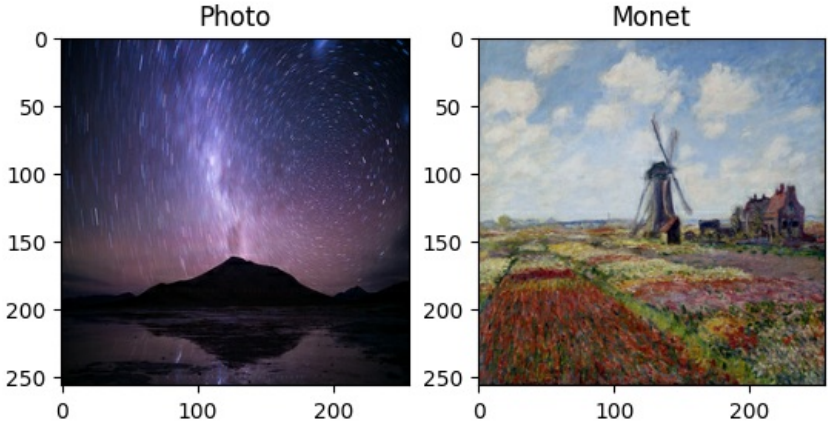
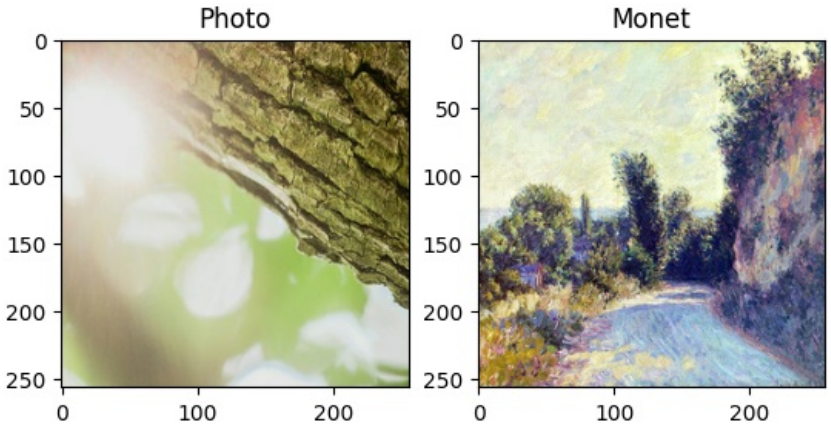


Photo
Monet
0
50
100
150
200
250
0
100
200



Generator

The CycleGAN below uses a UNET architecture. The generator is created with the downsample and upsample methods. The downsample function reduces the width and height of the image by the stride, or the step the filter takes. With a stride of 2, the filter is applied to every other pixel, which reduces the weight and height by 2. Upsample does the opposite of downsample and increases the height and width of the of the image by the stride. I use the tensorflow.keras layers Conv2D and Conv2DTranspose, which does basically the opposite of a Conv2D layer, for these functions.

Instance normalization, which normalizes all features of one channel, is used instead of batch normalization. The Groupsize is equal to the channel size. According to tensorflow its accuracy is more stable than batch normalization in a range of small batch sizes with an appropriate learning rate which is adjusted linearly with batch size.

In [12]:
```
# Weights initializer for the layers.
kernel_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)
# Gamma initializer for instance normalization.
gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)

class ReflectionPadding2D(layers.Layer):
    """Implements Reflection Padding as a layer.
    Args:
        padding(tuple): Amount of padding for the
        spatial dimensions.
    Returns:
        A padded tensor with the same type as the input tensor.
    """

    def __init__(self, padding=(1, 1), **kwargs):
        self.padding = tuple(padding)
        super().__init__(**kwargs)

    def call(self, input_tensor, mask=None):
        padding_width, padding_height = self.padding
        padding_tensor = [
            [0, 0],
            [padding_height, padding_height],
            [padding_width, padding_width],
            [0, 0],
        ]
        return tf.pad(input_tensor, padding_tensor, mode="REFLECT")


def residual_block(
    x,
    activation,
    kernel_initializer=kernel_init,
    kernel_size=(3, 3),
    strides=(1, 1),
    padding="valid",
    gamma_initializer=gamma_init,
    use_bias=False,
):
    dim = x.shape[-1]
    input_tensor = x

    x = ReflectionPadding2D()(input_tensor)
    x = layers.Conv2D(
        dim,
        kernel_size,
        strides=strides,
        kernel_initializer=kernel_initializer,
        padding=padding,
```

```
        use_bias=use_bias,
    )(x)
    x = tfa.layers.InstanceNormalization(gamma_initializer=gamma_initializer)(x)
    x = activation(x)

    x = ReflectionPadding2D()(x)
    x = layers.Conv2D(
        dim,
        kernel_size,
        strides=strides,
        kernel_initializer=kernel_initializer,
        padding=padding,
        use_bias=use_bias,
    )(x)
    x = tfa.layers.InstanceNormalization(gamma_initializer=gamma_initializer)(x)
    x = layers.add([input_tensor, x])
    return x


def downsample(
    x,
    filters,
    activation,
    kernel_initializer=kernel_init,
    kernel_size=(3, 3),
    strides=(2, 2),
    padding="same",
    gamma_initializer=gamma_init,
    use_bias=False,
):
    x = layers.Conv2D(
        filters,
        kernel_size,
        strides=strides,
        kernel_initializer=kernel_initializer,
        padding=padding,
        use_bias=use_bias,
    )(x)
    x = tfa.layers.InstanceNormalization(gamma_initializer=gamma_initializer)(x)
    if activation:
        x = activation(x)
    return x


def upsample(
    x,
    filters,
    activation,
    kernel_size=(3, 3),
    strides=(2, 2),
    padding="same",
    kernel_initializer=kernel_init,
    gamma_initializer=gamma_init,
    use_bias=False,
):
    x = layers.Conv2DTranspose(
        filters,
        kernel_size,
        strides=strides,
        padding=padding,
        kernel_initializer=kernel_initializer,
        use_bias=use_bias,
    )(x)
    x = tfa.layers.InstanceNormalization(gamma_initializer=gamma_initializer)(x)
    if activation:
        x = activation(x)
    return x
```

Combine Upsample & DownSample

The generator first downsamples the input image and then upsample while establishing long skip connections to help bypass the vanishing gradient problem. It concatenates the output of the downsample layer to the upsample layer. I define the input image size 256x256x3 because its an RGB image.

In [13]:
```
input_img_size = (256,256,3)
def get_resnet_generator(
    filters=64,
    num_downsampling_blocks=2,
    num_residual_blocks=9,
    num_upsample_blocks=2,
    gamma_initializer=gamma_init,
```

```
    name=None,
):
    img_input = layers.Input(shape=input_img_size, name=name + "_img_input")
    x = ReflectionPadding2D(padding=(3, 3))(img_input)
    x = layers.Conv2D(filters, (7, 7), kernel_initializer=kernel_init, use_bias=False)(
        x
    )
    x = tfa.layers.InstanceNormalization(gamma_initializer=gamma_initializer)(x)
    x = layers.Activation("relu")(x)

    # Downsampling
    for _ in range(num_downsampling_blocks):
        filters *= 2
        x = downsample(x, filters=filters, activation=layers.Activation("relu"))

    # Residual blocks
    for _ in range(num_residual_blocks):
        x = residual_block(x, activation=layers.Activation("relu"))

    # Upsampling
    for _ in range(num_upsample_blocks):
        filters //= 2
        x = upsample(x, filters, activation=layers.Activation("relu"))

    # Final block
    x = ReflectionPadding2D(padding=(3, 3))(x)
    x = layers.Conv2D(3, (7, 7), padding="valid")(x)
    x = layers.Activation("tanh")(x)

    model = keras.models.Model(img_input, x, name=name)
    return model
```

Build the discriminator

The discriminator takes in the input image and classifies it as real or fake (generated). Instead of outputing a single node, the discriminator outputs a smaller 2D image with higher pixel values indicating a real classification and lower values indicating a fake classification.

In [14]:
```
def get_discriminator(
    filters=64, kernel_initializer=kernel_init, num_downsampling=3, name=None
):
    img_input = layers.Input(shape=input_img_size, name=name + "_img_input")
    x = layers.Conv2D(
        filters,
        (4, 4),
        strides=(2, 2),
        padding="same",
        kernel_initializer=kernel_initializer,
    )(img_input)
    x = layers.LeakyReLU(0.2)(x)

    num_filters = filters
    for num_downsample_block in range(3):
        num_filters *= 2
        if num_downsample_block < 2:
            x = downsample(
                x,
                filters=num_filters,
                activation=layers.LeakyReLU(0.2),
                kernel_size=(4, 4),
                strides=(2, 2),
            )
        else:
            x = downsample(
                x,
                filters=num_filters,
                activation=layers.LeakyReLU(0.2),
                kernel_size=(4, 4),
                strides=(1, 1),
            )

    x = layers.Conv2D(
        1, (4, 4), strides=(1, 1), padding="same", kernel_initializer=kernel_initializer
    )(x)

    model = keras.models.Model(inputs=img_input, outputs=x, name=name)
    return model


# Get the generators
```

```
gen_G = get_resnet_generator(name="generator_G")
gen_F = get_resnet_generator(name="generator_F")

# Get the discriminators
disc_X = get_discriminator(name="discriminator_X")
disc_Y = get_discriminator(name="discriminator_Y")
```

```
/opt/anaconda3/envs/py39/lib/python3.9/site-packages/keras/src/initializers/initializers.py:120: UserWarning: Th
e initializer RandomNormal is unseeded and being called multiple times, which will return identical values each
time (even if the initializer is unseeded). Please update your code to provide a seed to the initializer, or avo
id using the same initializer instance more than once.
  warnings.warn(
```

Build the CycleGAN model

The CycleGAN class is a sublass of keras.Model so that fit() can be ran on it later to train the model. During the training step, the model transforms a photo to a Monet painting and then back to a photo. The difference between the original photo and the twice-transformed photo is the cycle-consistency loss. The goal is for the original photo and the twice-transformed photo to be similar to one another. The mean absolute error identity loss and cycle loss are used.

In [15]:
```
class CycleGan(keras.Model):
    def __init__(
        self,
        generator_G,
        generator_F,
        discriminator_X,
        discriminator_Y,
        lambda_cycle=10.0,
        lambda_identity=0.5,
    ):
        super().__init__()
        self.gen_G = generator_G
        self.gen_F = generator_F
        self.disc_X = discriminator_X
        self.disc_Y = discriminator_Y
        self.lambda_cycle = lambda_cycle
        self.lambda_identity = lambda_identity

    def call(self, inputs):
        return (
            self.disc_X(inputs),
            self.disc_Y(inputs),
            self.gen_G(inputs),
            self.gen_F(inputs),
        )

    def compile(
        self,
        gen_G_optimizer,
        gen_F_optimizer,
        disc_X_optimizer,
        disc_Y_optimizer,
        gen_loss_fn,
        disc_loss_fn,
    ):
        super().compile()
        self.gen_G_optimizer = gen_G_optimizer
        self.gen_F_optimizer = gen_F_optimizer
        self.disc_X_optimizer = disc_X_optimizer
        self.disc_Y_optimizer = disc_Y_optimizer
        self.generator_loss_fn = gen_loss_fn
        self.discriminator_loss_fn = disc_loss_fn
        self.cycle_loss_fn = keras.losses.MeanAbsoluteError()
        self.identity_loss_fn = keras.losses.MeanAbsoluteError()

    def train_step(self, batch_data):
        #Horse2Zebra
        #Photo2Monet
        # x is Photo and y is Monet
        real_x, real_y = batch_data

        # For CycleGAN, we need to calculate different
        # kinds of losses for the generators and discriminators.
        # We will perform the following steps here:
        #
        # 1. Pass real images through the generators and get the generated images
        # 2. Pass the generated images back to the generators to check if we
        #    can predict the original image from the generated image.
        # 3. Do an identity mapping of the real images using the generators.
        # 4. Pass the generated images in 1) to the corresponding discriminators.
        # 5. Calculate the generators total loss (adversarial + cycle + identity)
```

```
        # 6. Calculate the discriminators loss
        # 7. Update the weights of the generators
        # 8. Update the weights of the discriminators
        # 9. Return the losses in a dictionary

        with tf.GradientTape(persistent=True) as tape:
            # Photo to fake Monet
            fake_y = self.gen_G(real_x, training=True)
            # Monet to fake Photo -> y2x
            fake_x = self.gen_F(real_y, training=True)

            # Cycle (Photo to fake Monet to fake Photo): x -> y -> x
            cycled_x = self.gen_F(fake_y, training=True)
            # Cycle (Monet to fake Photo to fake Monet) y -> x -> y
            cycled_y = self.gen_G(fake_x, training=True)

            # Identity mapping
            same_x = self.gen_F(real_x, training=True)
            same_y = self.gen_G(real_y, training=True)

            # Discriminator output
            disc_real_x = self.disc_X(real_x, training=True)
            disc_fake_x = self.disc_X(fake_x, training=True)

            disc_real_y = self.disc_Y(real_y, training=True)
            disc_fake_y = self.disc_Y(fake_y, training=True)

            # Generator adversarial loss
            gen_G_loss = self.generator_loss_fn(disc_fake_y)
            gen_F_loss = self.generator_loss_fn(disc_fake_x)

            # Generator cycle loss
            cycle_loss_G = self.cycle_loss_fn(real_y, cycled_y) * self.lambda_cycle
            cycle_loss_F = self.cycle_loss_fn(real_x, cycled_x) * self.lambda_cycle

            # Generator identity loss
            id_loss_G = (
                self.identity_loss_fn(real_y, same_y)
                * self.lambda_cycle
                * self.lambda_identity
            )
            id_loss_F = (
                self.identity_loss_fn(real_x, same_x)
                * self.lambda_cycle
                * self.lambda_identity
            )

            # Total generator loss
            total_loss_G = gen_G_loss + cycle_loss_G + id_loss_G
            total_loss_F = gen_F_loss + cycle_loss_F + id_loss_F

            # Discriminator loss
            disc_X_loss = self.discriminator_loss_fn(disc_real_x, disc_fake_x)
            disc_Y_loss = self.discriminator_loss_fn(disc_real_y, disc_fake_y)

        # Get the gradients for the generators
        grads_G = tape.gradient(total_loss_G, self.gen_G.trainable_variables)
        grads_F = tape.gradient(total_loss_F, self.gen_F.trainable_variables)

        # Get the gradients for the discriminators
        disc_X_grads = tape.gradient(disc_X_loss, self.disc_X.trainable_variables)
        disc_Y_grads = tape.gradient(disc_Y_loss, self.disc_Y.trainable_variables)

        # Update the weights of the generators
        self.gen_G_optimizer.apply_gradients(
            zip(grads_G, self.gen_G.trainable_variables)
        )
        self.gen_F_optimizer.apply_gradients(
            zip(grads_F, self.gen_F.trainable_variables)
        )

        # Update the weights of the discriminators
        self.disc_X_optimizer.apply_gradients(
            zip(disc_X_grads, self.disc_X.trainable_variables)
        )
        self.disc_Y_optimizer.apply_gradients(
            zip(disc_Y_grads, self.disc_Y.trainable_variables)
        )

        return {
            "G_loss": total_loss_G,
            "F_loss": total_loss_F,
            "D_X_loss": disc_X_loss,
```

```
            "D_Y_loss": disc_Y_loss,
        }
```

Define loss functions

The generator wants to fool the discriminator into thinking the generated image is real. The discriminator loss function below compares real images to a matrix of 1s and fake images to a matrix of 0s. The perfect generator will have the discriminator output only 1s. The perfect discriminator will output all 1s for real images and all 0s for fake images. The discriminator loss outputs the average of the real and generated loss.

In [16]:
```
# Loss function for evaluating adversarial loss
adv_loss_fn = keras.losses.MeanSquaredError()

# loss function for the generators
def generator_loss_fn(fake):
    fake_loss = adv_loss_fn(tf.ones_like(fake), fake)
    return fake_loss


# loss function for the discriminators
def discriminator_loss_fn(real, fake):
    real_loss = adv_loss_fn(tf.ones_like(real), real)
    fake_loss = adv_loss_fn(tf.zeros_like(fake), fake)
    return (real_loss + fake_loss) * 0.5
```

This GANMonitor() function will be used later during training to visualize generated images after each epoch.

The identity loss compares the image with its generator (i.e. photo with photo generator). If given a photo as input, we want it to generate the same image as the image was originally a photo. The identity loss compares the input with the output of the generator.

In [17]:
```
class GANMonitor(keras.callbacks.Callback):
    """A callback to generate and save images after each epoch"""

    def __init__(self, num_img=4):
        self.num_img = num_img

    def on_epoch_end(self, epoch, logs=None):
        _, ax = plt.subplots(4, 2, figsize=(12, 12))
        for i, img in enumerate(photo_ds.take(self.num_img)):
            prediction = self.model.gen_G(img)[0].numpy()
            prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
            img = (img[0] * 127.5 + 127.5).numpy().astype(np.uint8)

            ax[i, 0].imshow(img)
            ax[i, 1].imshow(prediction)
            ax[i, 0].set_title("Input image")
            ax[i, 1].set_title("Translated image")
            ax[i, 0].axis("off")
            ax[i, 1].axis("off")

            prediction = keras.utils.array_to_img(prediction)
            prediction.save(
                "generated_img_{i}_{epoch}.png".format(i=i, epoch=epoch + 1)
            )
        plt.show()
        plt.close()
```

Below I compute the model adam optimizers and the two loss functions we defined for the discriminator and generator.

In [18]:
```
# Create cycle gan model
cycle_gan_model = CycleGan(
    generator_G=gen_G, generator_F=gen_F, discriminator_X=disc_X, discriminator_Y=disc_Y
)

# Compile the model
cycle_gan_model.compile(
    gen_G_optimizer=keras.optimizers.legacy.Adam(learning_rate=2e-4, beta_1=0.5),
    gen_F_optimizer=keras.optimizers.legacy.Adam(learning_rate=2e-4, beta_1=0.5),
    disc_X_optimizer=keras.optimizers.legacy.Adam(learning_rate=2e-4, beta_1=0.5),
    disc_Y_optimizer=keras.optimizers.legacy.Adam(learning_rate=2e-4, beta_1=0.5),
    gen_loss_fn=generator_loss_fn,
    disc_loss_fn=discriminator_loss_fn,
)
```

Below I define the callbacks and fit the model. I am using the GANMonitor callback which is a function created above to generate images after epoch to visualize results, a checkpoint callback to save a model at each epoch and a logger for the training history. Then I being training for 50 epochs.

In [19]:

```
callbacks = [GANMonitor(), keras.callbacks.ModelCheckpoint("save_at_{epoch}.keras"), CSVLogger('CycleGAN_traini
             keras.callbacks.ModelCheckpoint(filepath="./training_checkpoints/checkpoint_{epoch}.keras", save_w

cycle_gan_model.fit(
    tf.data.Dataset.zip((photo_ds, monet_ds)),
    epochs=50,
    callbacks=callbacks,)
```

```
Epoch 1/50
    300/Unknown - 3263s 11s/step - G_loss: 5.0576 - F_loss: 5.6748 - D_X_loss: 0.0986 - D_Y_loss: 0.1212
```

Input image

Translated image

Input image

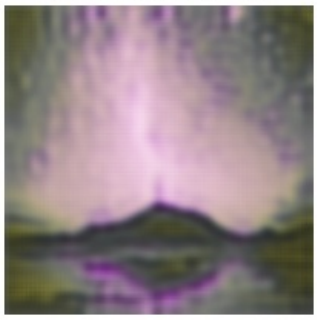
Translated image

Input image

Translated image

Input image

Translated image



```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
/opt/anaconda3/envs/py39/lib/python3.9/site-packages/keras/src/saving/saving_api.py:164: UserWarning: You are sa
ving a model that has not yet been built. It might not contain any weights yet. Consider building the model firs
t by calling it on some data.
  saving_lib.save_model(model, filepath)
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 3271s 11s/step - G_loss: 5.0512 - F_loss: 5.6777 - D_X_loss: 0.0983 -
D_Y_loss: 0.1209
Epoch 2/50
300/300 [==============================] - ETA: 0s - G_loss: 4.6119 - F_loss: 5.2049 - D_X_loss: 0.1346 - D_Y_lo
ss: 0.1828
```

Input image

Translated image

Input image

Translated image

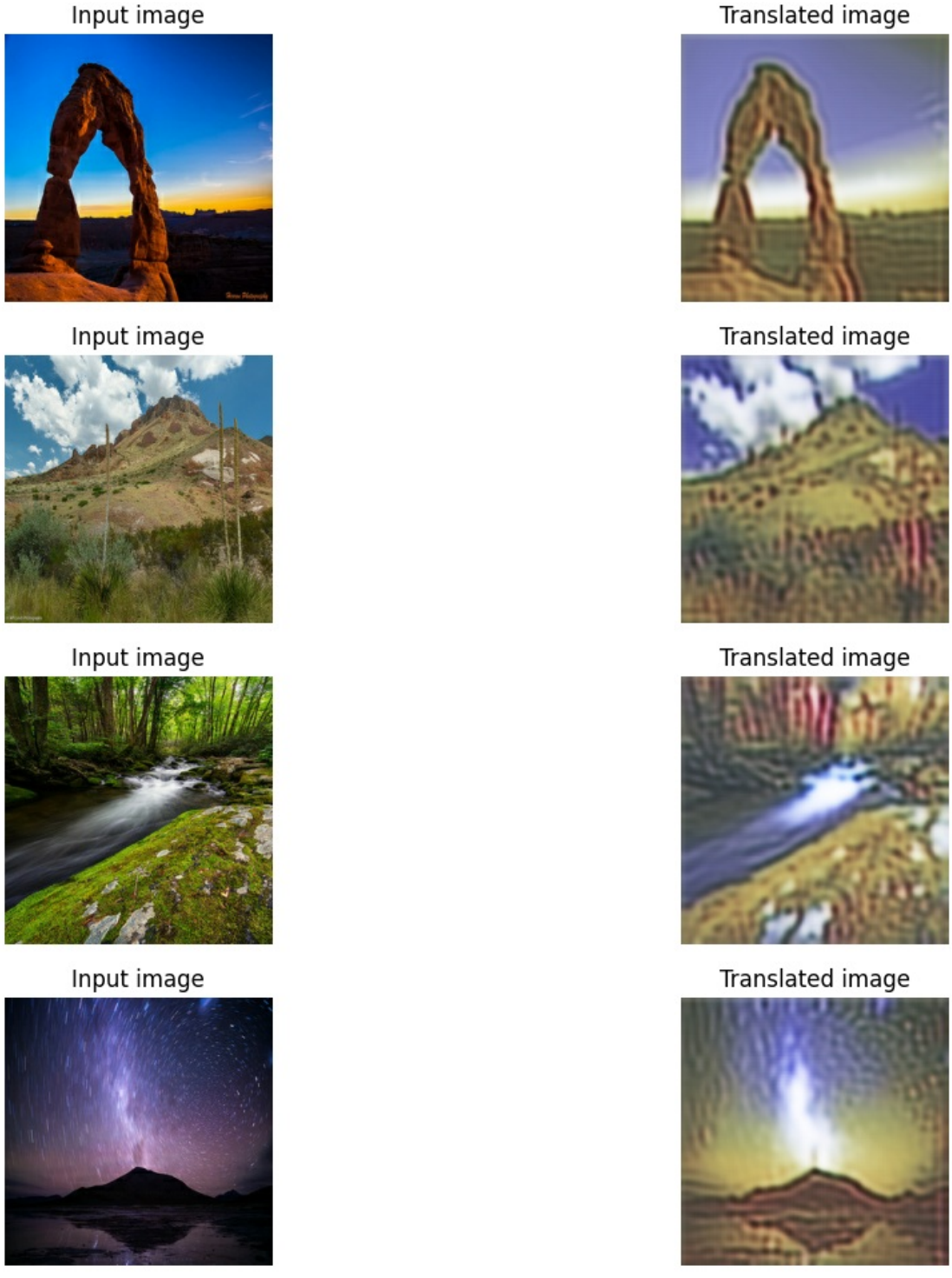
Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 3199s 11s/step - G_loss: 4.6048 - F_loss: 5.2048 - D_X_loss: 0.1344 -
D_Y_loss: 0.1830
Epoch 3/50
300/300 [==============================] - ETA: 0s - G_loss: 4.3367 - F_loss: 4.7622 - D_X_loss: 0.1623 - D_Y_lo
ss: 0.1826
```




```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2930s 10s/step - G_loss: 4.3297 - F_loss: 4.7598 - D_X_loss: 0.1619 -
D_Y_loss: 0.1828
Epoch 4/50
300/300 [==============================] - ETA: 0s - G_loss: 4.1971 - F_loss: 4.4956 - D_X_loss: 0.1808 - D_Y_lo
ss: 0.1591
```

Input image

Translated image

Input image

Translated image

Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2938s 10s/step - G_loss: 4.1931 - F_loss: 4.4907 - D_X_loss: 0.1813 -
D_Y_loss: 0.1593
Epoch 5/50
300/300 [==============================] - ETA: 0s - G_loss: 4.0898 - F_loss: 4.3408 - D_X_loss: 0.1675 - D_Y_lo
ss: 0.1449
```

Input image

Translated image

Input image

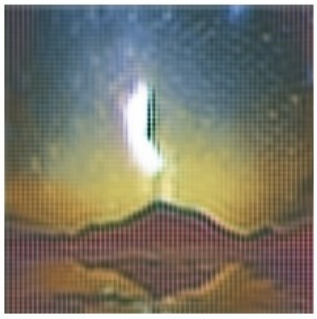
Translated image

Input image

Translated image

Input image

Translated image



```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2924s 10s/step - G_loss: 4.0850 - F_loss: 4.3379 - D_X_loss: 0.1673 -
D_Y_loss: 0.1458
Epoch 6/50
300/300 [==============================] - ETA: 0s - G_loss: 4.0719 - F_loss: 4.2226 - D_X_loss: 0.1594 - D_Y_lo
ss: 0.1415
```

Input image

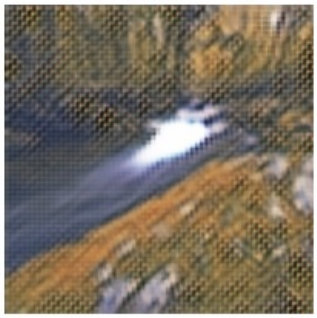
Translated image

Input image

Translated image

Input image

Translated image



Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2912s 10s/step - G_loss: 4.0679 - F_loss: 4.2174 - D_X_loss: 0.1598 -
D_Y_loss: 0.1420
Epoch 7/50
300/300 [==============================] - ETA: 0s - G_loss: 3.8708 - F_loss: 4.1403 - D_X_loss: 0.1494 - D_Y_lo
ss: 0.1795
```

Input image

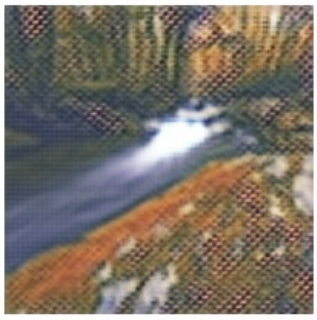
Translated image

Input image

Translated image

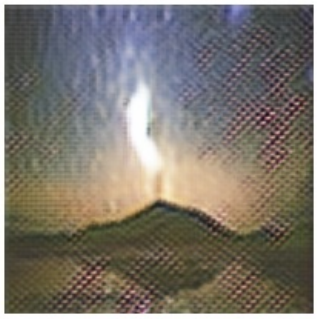
Input image

Translated image



Input image

Translated image



```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2902s 10s/step - G_loss: 3.8670 - F_loss: 4.1378 - D_X_loss: 0.1495 -
D_Y_loss: 0.1798
Epoch 8/50
300/300 [==============================] - ETA: 0s - G_loss: 3.7599 - F_loss: 4.0673 - D_X_loss: 0.1496 - D_Y_lo
ss: 0.1797
```

Input image

Translated image

Input image

Translated image

Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2905s 10s/step - G_loss: 3.7573 - F_loss: 4.0649 - D_X_loss: 0.1500 -
D_Y_loss: 0.1798
Epoch 9/50
300/300 [==============================] - ETA: 0s - G_loss: 3.6746 - F_loss: 4.0140 - D_X_loss: 0.1510 - D_Y_lo
ss: 0.1873
```


```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2900s 10s/step - G_loss: 3.6714 - F_loss: 4.0128 - D_X_loss: 0.1510 -
D_Y_loss: 0.1875
Epoch 10/50
300/300 [==============================] - ETA: 0s - G_loss: 3.5989 - F_loss: 3.8836 - D_X_loss: 0.1803 - D_Y_lo
ss: 0.1927
```


```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2892s 10s/step - G_loss: 3.5961 - F_loss: 3.8811 - D_X_loss: 0.1803 -
D_Y_loss: 0.1932
Epoch 11/50
300/300 [==============================] - ETA: 0s - G_loss: 3.5442 - F_loss: 3.8442 - D_X_loss: 0.1837 - D_Y_lo
ss: 0.1908
```

Input image

Translated image

Input image

Translated image

Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2900s 10s/step - G_loss: 3.5411 - F_loss: 3.8443 - D_X_loss: 0.1841 -
D_Y_loss: 0.1913
Epoch 12/50
300/300 [==============================] - ETA: 0s - G_loss: 3.5354 - F_loss: 3.7594 - D_X_loss: 0.1827 - D_Y_lo
ss: 0.1869
```

Input image

Translated image

Input image

Translated image

Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2900s 10s/step - G_loss: 3.5327 - F_loss: 3.7561 - D_X_loss: 0.1826 -
D_Y_loss: 0.1873
Epoch 13/50
300/300 [==============================] - ETA: 0s - G_loss: 3.4824 - F_loss: 3.6683 - D_X_loss: 0.1873 - D_Y_lo
ss: 0.1879
```

Input image

Translated image

Input image

Translated image

Input image

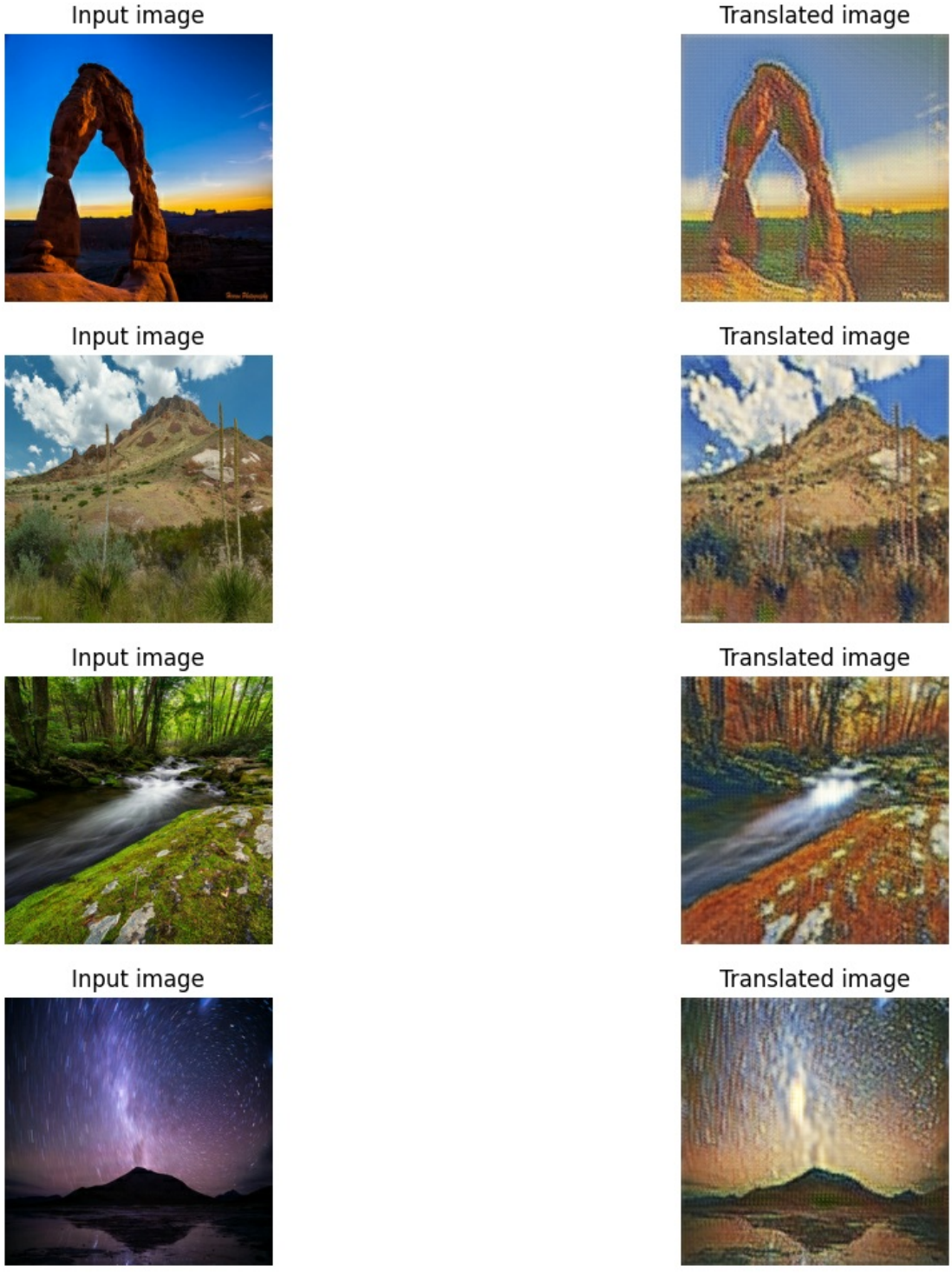
Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2913s 10s/step - G_loss: 3.4805 - F_loss: 3.6662 - D_X_loss: 0.1873 -
D_Y_loss: 0.1884
Epoch 14/50
300/300 [==============================] - ETA: 0s - G_loss: 3.4013 - F_loss: 3.6734 - D_X_loss: 0.1825 - D_Y_lo
ss: 0.1901
```


```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2883s 10s/step - G_loss: 3.3985 - F_loss: 3.6731 - D_X_loss: 0.1823 -
D_Y_loss: 0.1906
Epoch 15/50
300/300 [==============================] - ETA: 0s - G_loss: 3.3659 - F_loss: 3.6901 - D_X_loss: 0.1793 - D_Y_lo
ss: 0.1844
```




```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2885s 10s/step - G_loss: 3.3633 - F_loss: 3.6890 - D_X_loss: 0.1792 -
D_Y_loss: 0.1848
Epoch 16/50
300/300 [==============================] - ETA: 0s - G_loss: 3.3396 - F_loss: 3.6375 - D_X_loss: 0.1763 - D_Y_lo
ss: 0.1816
```

Input image

Translated image

Input image

Translated image

Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2898s 10s/step - G_loss: 3.3374 - F_loss: 3.6364 - D_X_loss: 0.1763 -
D_Y_loss: 0.1820
Epoch 17/50
300/300 [==============================] - ETA: 0s - G_loss: 3.3514 - F_loss: 3.6196 - D_X_loss: 0.1858 - D_Y_lo
ss: 0.1855
```

Input image

Translated image

Input image

Translated image

Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2898s 10s/step - G_loss: 3.3488 - F_loss: 3.6193 - D_X_loss: 0.1858 -
D_Y_loss: 0.1858
Epoch 18/50
300/300 [==============================] - ETA: 0s - G_loss: 3.3380 - F_loss: 3.6765 - D_X_loss: 0.1780 - D_Y_lo
ss: 0.1854
```


```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2902s 10s/step - G_loss: 3.3349 - F_loss: 3.6757 - D_X_loss: 0.1780 -
D_Y_loss: 0.1859
Epoch 19/50
300/300 [==============================] - ETA: 0s - G_loss: 3.2915 - F_loss: 3.6334 - D_X_loss: 0.1736 - D_Y_lo
ss: 0.1870
```

Input image

Translated image

Input image

Translated image

Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2908s 10s/step - G_loss: 3.2890 - F_loss: 3.6354 - D_X_loss: 0.1733 -
D_Y_loss: 0.1873
Epoch 20/50
300/300 [==============================] - ETA: 0s - G_loss: 3.2816 - F_loss: 3.6971 - D_X_loss: 0.1748 - D_Y_lo
ss: 0.1905
```

Input image

Translated image

Input image

Translated image

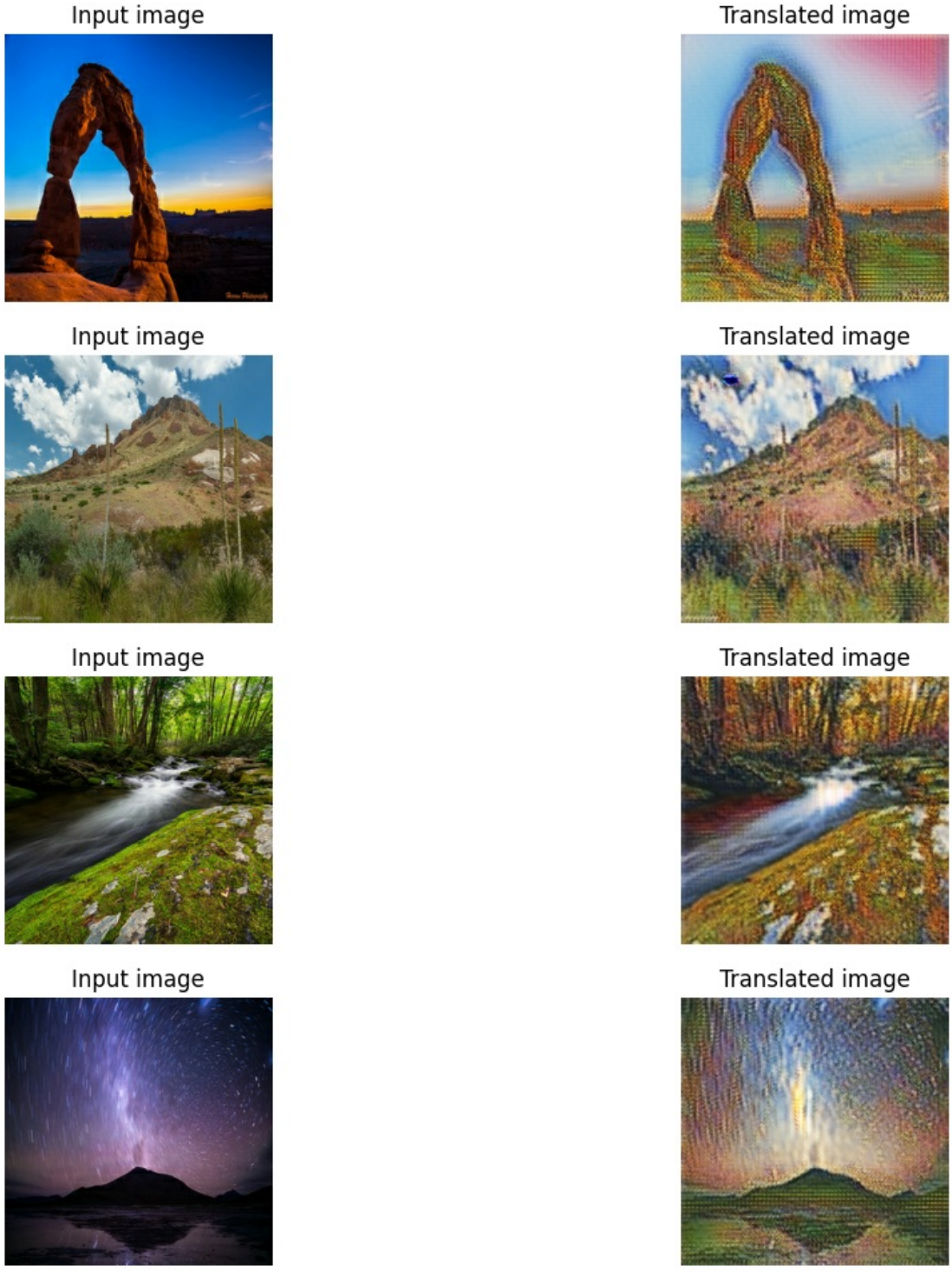
Input image

Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2917s 10s/step - G_loss: 3.2792 - F_loss: 3.7008 - D_X_loss: 0.1748 -
D_Y_loss: 0.1907
Epoch 21/50
300/300 [==============================] - ETA: 0s - G_loss: 3.2530 - F_loss: 3.6601 - D_X_loss: 0.1751 - D_Y_lo
ss: 0.1992
```




```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2918s 10s/step - G_loss: 3.2514 - F_loss: 3.6616 - D_X_loss: 0.1750 -
D_Y_loss: 0.1995
Epoch 22/50
300/300 [==============================] - ETA: 0s - G_loss: 3.2749 - F_loss: 3.7435 - D_X_loss: 0.1754 - D_Y_lo
ss: 0.1956
```

Input image

Translated image

Input image

Translated image

Input image

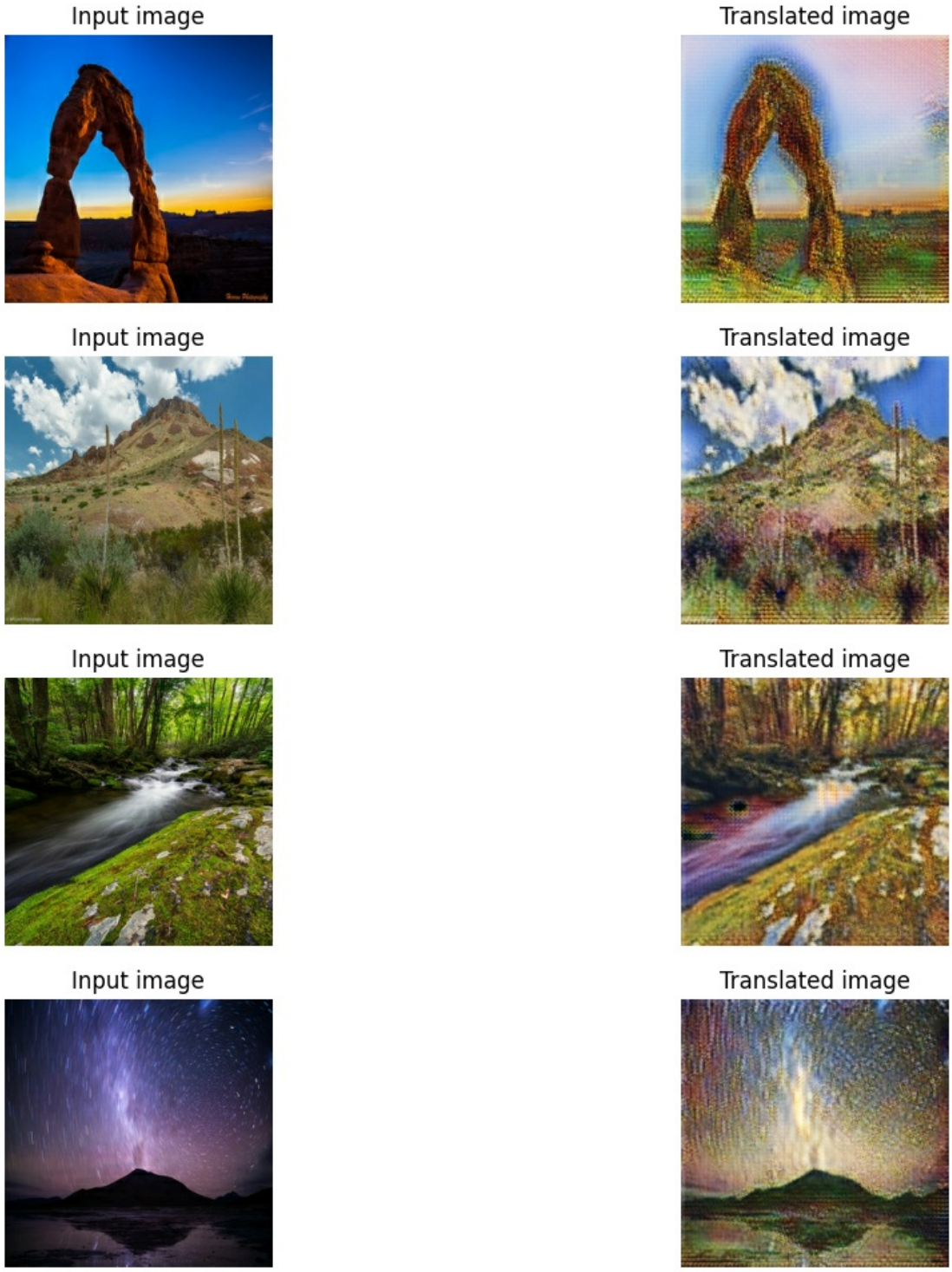
Translated image

Input image

Translated image

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2926s 10s/step - G_loss: 3.2723 - F_loss: 3.7481 - D_X_loss: 0.1755 -
D_Y_loss: 0.1960
Epoch 23/50
300/300 [==============================] - ETA: 0s - G_loss: 3.2288 - F_loss: 3.7591 - D_X_loss: 0.1722 - D_Y_lo
ss: 0.1985
```


```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 2940s 10s/step - G_loss: 3.2261 - F_loss: 3.7635 - D_X_loss: 0.1721 -
D_Y_loss: 0.1989
Epoch 24/50
300/300 [==============================] - ETA: 0s - G_loss: 3.2039 - F_loss: 3.7172 - D_X_loss: 0.1654 - D_Y_lo
ss: 0.1954
```




```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_conf
ig()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
300/300 [==============================] - 3116s 10s/step - G_loss: 3.2020 - F_loss: 3.7212 - D_X_loss: 0.1655 -
D_Y_loss: 0.1956
Epoch 25/50
 53/300 [====>.........................] - ETA: 42:42 - G_loss: 3.3745 - F_loss: 3.6248 - D_X_loss: 0.1806 - D_Y
_loss: 0.2002
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[19], line 4
      1 callbacks = [GANMonitor(), keras.callbacks.ModelCheckpoint("save_at_{epoch}.keras"), CSVLogger('CycleGAN
_training.csv'),
      2              keras.callbacks.ModelCheckpoint(filepath="./training_checkpoints/checkpoint_{epoch}.keras",
save_weights_only=True)]
----> 4 cycle_gan_model.fit(
      5     tf.data.Dataset.zip((photo_ds, monet_ds)),
      6     epochs=50,
      7     callbacks=callbacks,)

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/keras/src/utils/traceback_utils.py:65, in filter_trace
```

```
back.<locals>.error_handler(*args, **kwargs)
     63 filtered_tb = None
     64 try:
---> 65     return fn(*args, **kwargs)
     66 except Exception as e:
     67     filtered_tb = _process_traceback_frames(e.__traceback__)

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/keras/src/engine/training.py:1807, in Model.fit(self,
x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_w
eight, initial_epoch, steps_per_epoch, validation_steps, validation_batch_size, validation_freq, max_queue_size,
workers, use_multiprocessing)
   1799 with tf.profiler.experimental.Trace(
   1800     "train",
   1801     epoch_num=epoch,
   (...)
   1804     _r=1,
   1805 ):
   1806     callbacks.on_train_batch_begin(step)
-> 1807     tmp_logs = self.train_function(iterator)
   1808     if data_handler.should_sync:
   1809         context.async_wait()

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/util/traceback_utils.py:150, in filt
er_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/polymorphic_function/polymorph
ic_function.py:832, in Function.__call__(self, *args, **kwds)
    829 compiler = "xla" if self._jit_compile else "nonXla"
    831 with OptionalXlaContext(self._jit_compile):
--> 832   result = self._call(*args, **kwds)
    834 new_tracing_count = self.experimental_get_tracing_count()
    835 without_tracing = (tracing_count == new_tracing_count)

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/polymorphic_function/polymorph
ic_function.py:868, in Function._call(self, *args, **kwds)
    865   self._lock.release()
    866   # In this case we have created variables on the first call, so we run the
    867   # defunned version which is guaranteed to never create variables.
--> 868   return tracing_compilation.call_function(
    869       args, kwds, self._no_variable_creation_config
    870   )
    871 elif self._variable_creation_config is not None:
    872   # Release the lock early so that multiple threads can perform the call
    873   # in parallel.
    874   self._lock.release()

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/polymorphic_function/tracing_c
ompilation.py:139, in call_function(args, kwargs, tracing_options)
    137 bound_args = function.function_type.bind(*args, **kwargs)
    138 flat_inputs = function.function_type.unpack_inputs(bound_args)
--> 139 return function._call_flat(  # pylint: disable=protected-access
    140     flat_inputs, captured_inputs=function.captured_inputs
    141 )

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/polymorphic_function/concrete_
function.py:1323, in ConcreteFunction._call_flat(self, tensor_inputs, captured_inputs)
   1319 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(args)
   1320 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES_NONE
   1321     and executing_eagerly):
   1322   # No tape is watching; skip to running the function.
-> 1323   return self._inference_function.call_preflattened(args)
   1324 forward_backward = self._select_forward_and_backward_functions(
   1325     args,
   1326     possible_gradient_type,
   1327     executing_eagerly)
   1328 forward_function, args_with_tangents = forward_backward.forward()

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/polymorphic_function/atomic_fu
nction.py:216, in AtomicFunction.call_preflattened(self, args)
    214 def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215   """Calls with flattened tensor inputs and returns the structured output."""
--> 216   flat_outputs = self.call_flat(*args)
    217   return self.function_type.pack_output(flat_outputs)

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/polymorphic_function/atomic_fu
nction.py:251, in AtomicFunction.call_flat(self, *args)
    249 with record.stop_recording():
    250   if self._bound_context.executing_eagerly():
```

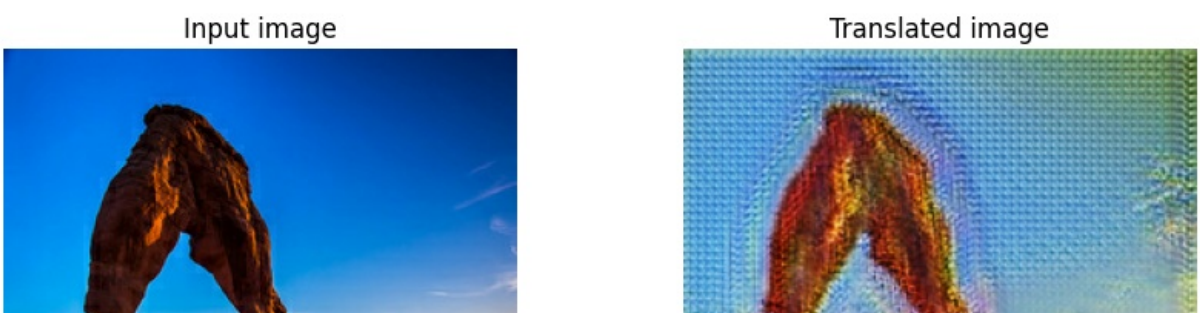
```
--> 251     outputs = self._bound_context.call_function(
    252         self.name,
    253         list(args),
    254         len(self.function_type.flat_outputs),
    255     )
    256   else:
    257     outputs = make_call_op_in_graph(
    258         self,
    259         list(args),
    260         self._bound_context.function_call_options.as_attrs(),
    261     )

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/context.py:1486, in Context.call_function(self, name, tensor_inputs, num_outputs)
   1484 cancellation_context = cancellation.context()
   1485 if cancellation_context is None:
-> 1486   outputs = execute.execute(
   1487       name.decode("utf-8"),
   1488       num_outputs=num_outputs,
   1489       inputs=tensor_inputs,
   1490       attrs=attrs,
   1491       ctx=self,
   1492   )
   1493 else:
   1494   outputs = execute.execute_with_cancellation(
   1495       name.decode("utf-8"),
   1496       num_outputs=num_outputs,
   (...)
   1500       cancellation_manager=cancellation_context,
   1501   )

File /opt/anaconda3/envs/py39/lib/python3.9/site-packages/tensorflow/python/eager/execute.py:53, in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51 try:
     52   ctx.ensure_initialized()
---> 53   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     54                                       inputs, attrs, num_outputs)
     55 except core._NotOkStatusException as e:
     56   if name is not None:

KeyboardInterrupt:
```

I stopped the model at 25/50 epochs for time.

Visualize Results

In [20]:

```
# Load the checkpoints
#weight_file = "./saved_checkpoints/cyclegan_checkpoints.090"
#cycle_gan_model.load_weights(weight_file).expect_partial()
#print("Weights loaded successfully")

_, ax = plt.subplots(4, 2, figsize=(10, 15))
for i, img in enumerate(photo_ds.take(4)):
    prediction = cycle_gan_model.gen_G(img, training=False)[0].numpy()
    prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
    img = (img[0] * 127.5 + 127.5).numpy().astype(np.uint8)

    ax[i, 0].imshow(img)
    ax[i, 1].imshow(prediction)
    ax[i, 0].set_title("Input image")
    ax[i, 0].set_title("Input image")
    ax[i, 1].set_title("Translated image")
    ax[i, 0].axis("off")
    ax[i, 1].axis("off")

    #prediction = keras.utils.array_to_img(prediction)
    #prediction.save("images/predicted_img_{i}.png".format(i=i))
plt.tight_layout()
plt.show()
```

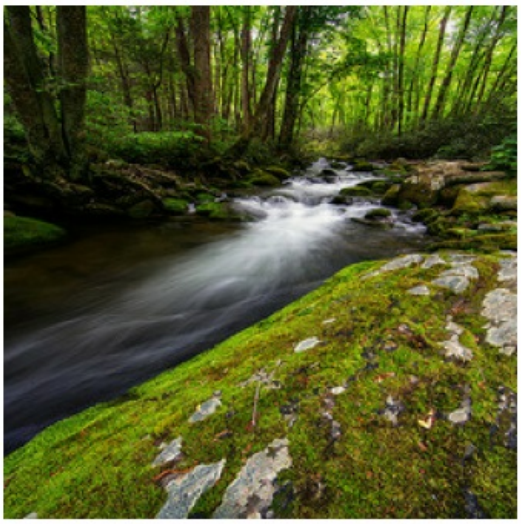


Input image

Translated image

Input image

Translated image



Input image

Translated image

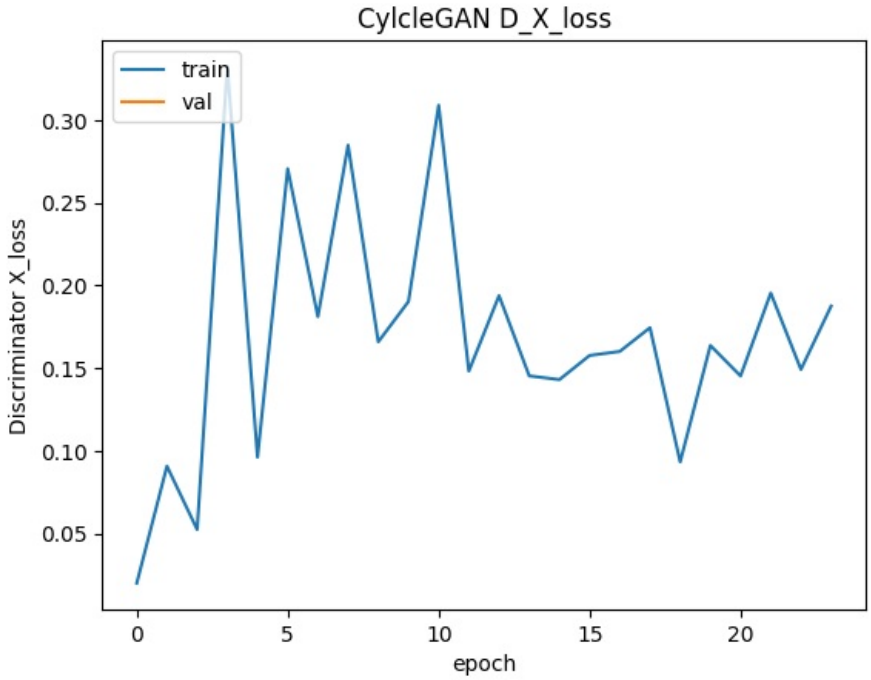
Below I try to plot the training and validation losses (Discriminator X & Y, Generator F & G losses) but my logging .csv seems to not have recorded the validation losses, only training.

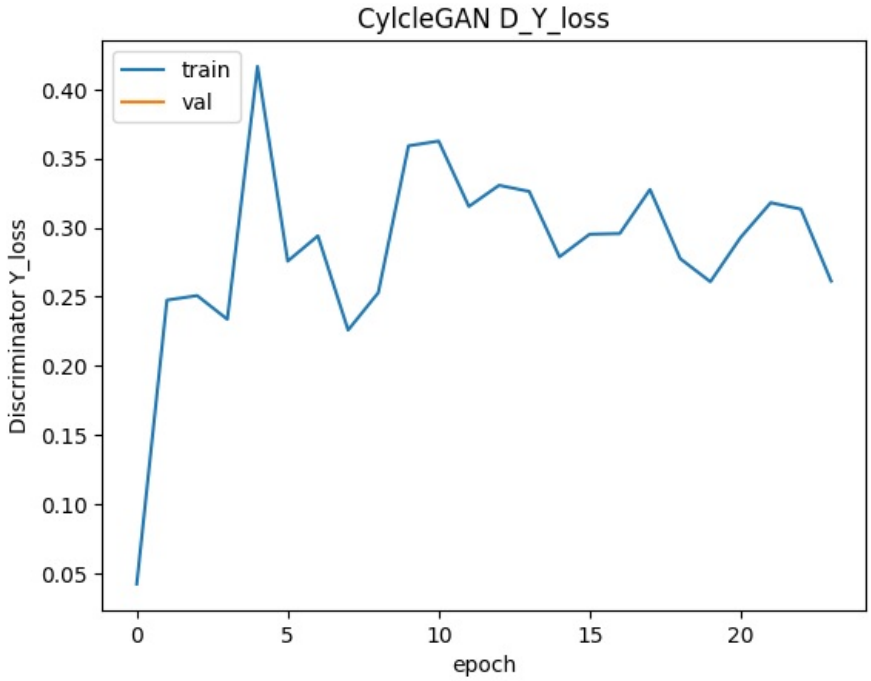
In [27]:
```
print(history)
history = pd.read_csv(r'CycleGAN_training.csv')
```

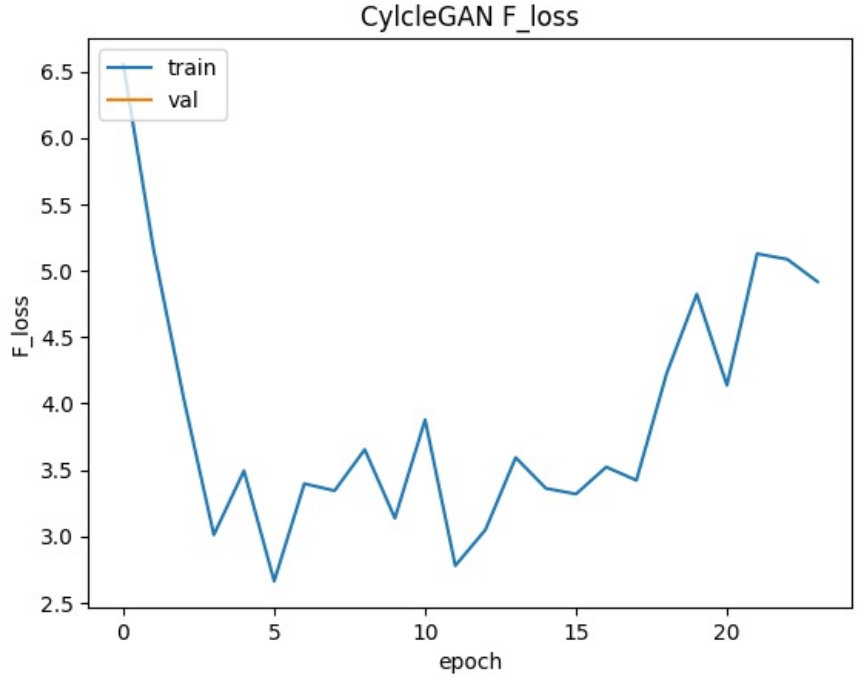
```
plt.plot(history['D_X_loss'])
plt.plot(history['val_D_X_loss'])
plt.title('CylcleGAN D_X_loss')
plt.ylabel('Discriminator X_loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history['D_Y_loss'])
plt.plot(history['val_D_Y_loss'])
plt.title('CylcleGAN D_Y_loss')
plt.ylabel('Discriminator Y_loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history['F_loss'])
plt.plot(history['val_F_loss'])
plt.title('CylcleGAN F_loss')
plt.ylabel('F_loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history['G_loss'])
plt.plot(history['val_G_loss'])
plt.title('CylcleGAN G_loss')
plt.ylabel('G_loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

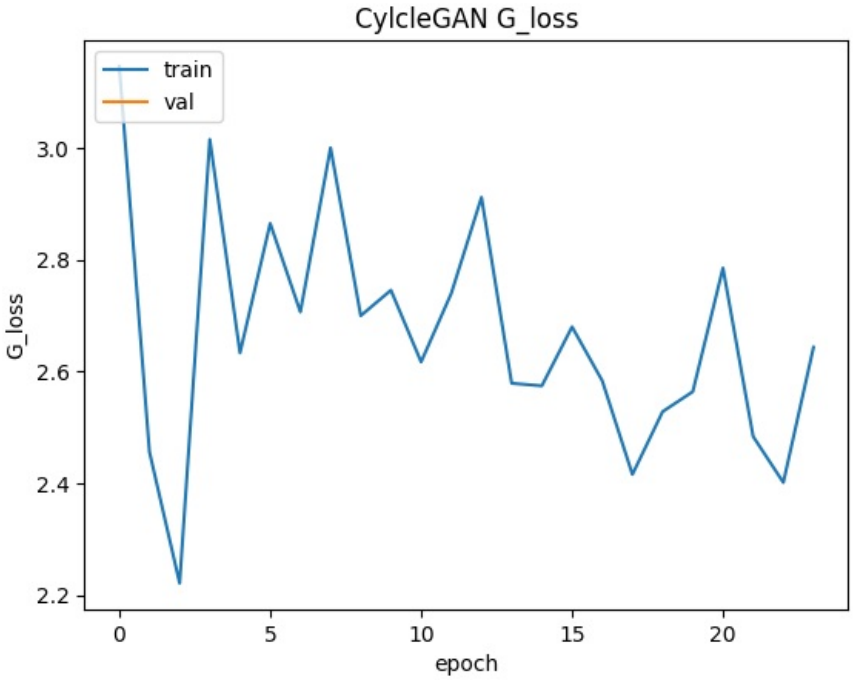
	epoch	D_X_loss	D_Y_loss	F_loss	G_loss	val_D_X_loss	val_D_Y_loss	\
0	0	0.019928	0.042167	6.553427	3.145375	NaN	NaN	
1	1	0.090707	0.247564	5.164968	2.456256	NaN	NaN	
2	2	0.052356	0.250806	4.040263	2.221504	NaN	NaN	
3	3	0.332519	0.233674	3.011059	3.014582	NaN	NaN	
4	4	0.096123	0.416639	3.493200	2.633110	NaN	NaN	
5	5	0.270557	0.275753	2.661965	2.864497	NaN	NaN	
6	6	0.181107	0.294041	3.396206	2.706265	NaN	NaN	
7	7	0.284792	0.225872	3.343287	2.999740	NaN	NaN	
8	8	0.165885	0.252900	3.653115	2.699522	NaN	NaN	
9	9	0.190263	0.359151	3.135803	2.745107	NaN	NaN	
10	10	0.308945	0.362603	3.877330	2.616914	NaN	NaN	
11	11	0.148184	0.315285	2.778791	2.739572	NaN	NaN	
12	12	0.193803	0.330606	3.049176	2.911160	NaN	NaN	
13	13	0.145287	0.326180	3.592870	2.578753	NaN	NaN	
14	14	0.143024	0.278828	3.360735	2.574259	NaN	NaN	
15	15	0.157641	0.295198	3.318253	2.679641	NaN	NaN	
16	16	0.160071	0.295735	3.521453	2.583370	NaN	NaN	
17	17	0.174429	0.327594	3.421333	2.415847	NaN	NaN	
18	18	0.093254	0.277560	4.225131	2.528076	NaN	NaN	
19	19	0.163672	0.260860	4.823249	2.563828	NaN	NaN	
20	20	0.145250	0.292853	4.136715	2.784875	NaN	NaN	
21	21	0.195389	0.318039	5.127097	2.483654	NaN	NaN	
22	22	0.149159	0.313347	5.085259	2.401499	NaN	NaN	
23	23	0.187503	0.261293	4.916391	2.643230	NaN	NaN	

	val_F_loss	val_G_loss
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	NaN
9	NaN	NaN
10	NaN	NaN
11	NaN	NaN
12	NaN	NaN
13	NaN	NaN
14	NaN	NaN
15	NaN	NaN
16	NaN	NaN
17	NaN	NaN
18	NaN	NaN
19	NaN	NaN
20	NaN	NaN
21	NaN	NaN
22	NaN	NaN
23	NaN	NaN


CylcleGAN D_X_loss
train
val
Discriminator X_loss
0.30
0.25
0.20
0.15
0.10
0.05
0
5
10
15
20
epoch


CylcleGAN D_Y_loss
train
val
Discriminator Y_loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
5
10
15
20
epoch


CylcleGAN F_loss
train
val
F_loss
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
0
5
10
15
20
epoch



Create submission file

In [ ]:
```
#! mkdir images
```

In [28]:
```
i = 1
for img in photo_ds:
    prediction = cycle_gan_model.gen_G(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
```

I had already trained my model in jupyter prior to finding out we have to submit a notebook through kaggle to submit our .zip of monet imitations. I saved my generated images to my local maching and then am reading them from the input of my kaggle notebook to be able to submit without having to retrain all over (took over 24 hours).

In [29]:
```
shutil.make_archive("images_zipped", 'zip', "images")
```

Out[29]:
```
'/Users/emilyhill/Documents/WEEK5_GANS/images_zipped.zip'
```

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

Conclusion

This is my first GAN to make and I relied heavily on keras documentation to help. The CycleGAN architecture aimed to map the translation between input image (photos) and output image (monet painting). The model iterates back and forth generating monet imitations from photos and trying to distinguish between real or fake monet paintings to train a discriminator and generator. If I had time to go back and redo this I would try to spend time understanding why my losses did not converge and tuning hyperparemeters. The model performed well with a 71.82 MiFID score. I would spend more time tuning hyperparemeters if I had the time.

In [ ]:
```
```