Introduction

The goal of this project is to build and train a GAN model to generate imitated Monet paintings from regular photos

Link to source of data: https://www.kaggle.com/competitions/gan-getting-started/data

Github link: https://github.com/talexandervo/wk5gans

In [57]:
```
# import required libraries
import pathlib
import os
import sys
import concurrent.futures

import pandas as pd
from PIL import Image

import numpy as np
import random as rn

import tensorflow as tf
import tensorflow_datasets as tfds

import matplotlib.pyplot as plt
%matplotlib inline

from timeit import default_timer as timer
```

In [58]:
```
class Constants:
    QUICK_TEST = False
    MAX_FILES = sys.maxsize
    TARGET_SIZE = [256,256]
    RESIZE = [282,282]
    BATCH_SIZE = 4
    RETRAIN_MODEL = False
    EPOCHS = 20
    MAX_PHOTOS_FOR_TESTING = MAX_FILES
    AUTOTUNE = tf.data.experimental.AUTOTUNE
    OUTPUT_CHANNELS = 3
    LAMBDA = 10

class Config():
    def __init__(self):
        self.dataset_url = "https://www.kaggle.com/competitions/histopathologic-cancer-d
        self.data_root_dir =  "/kaggle/input/gan-getting-started/"
        self.working_dir = "/kaggle/working/"
        self.temp_dir = "/kaggle/working/temp/"

        if os.path.exists("/kaggle"):
            print("Working in kaggle notebook enviorment")
        else:
            print("Working locally")
            self.data_root_dir =  "./gan-getting-started/"
            self.working_dir = self.data_root_dir
            self.temp_dir = self.working_dir

        self.temp_train_dir = self.temp_dir + "monet_tfrec/"
        self.temp_test_dir = self.temp_dir + "photo_tfrec/"

        self.data_dir = self.data_root_dir
```

```
        self.train_csv =  self.data_dir + "train_labels.csv"
        self.test_csv = None #self.data_dir + "test.csv"

        self.origin_train_dir = self.temp_train_dir
        self.origin_test_dir = self.temp_test_dir

        self.train_dir = self.temp_train_dir #self.data_dir + "train/"
        self.test_dir = self.temp_test_dir #self.data_dir + "test/"

        self.dir_true = self.train_dir + "1/"
        self.dir_false = self.train_dir + "0/"

        self.origin_train_path = pathlib.Path(self.origin_train_dir).with_suffix('')
        self.origin_test_path = pathlib.Path(self.origin_test_dir).with_suffix('')

        self.train_path = pathlib.Path(self.train_dir).with_suffix('')
        self.test_path = pathlib.Path(self.test_dir).with_suffix('')



        # For GAN
        self.monet_jpg_dir = self.data_root_dir + "monet_jpg/"
        self.monet_tfrec_dir = self.data_root_dir + "monet_tfrec/" #Training dataset
        self.photo_jpg_dir = self.data_root_dir + "photo_jpg/"
        self.photo_tfrec_dir = self.data_root_dir + "photo_tfrec/" #Testing dataset

        self.train_path = pathlib.Path(self.monet_tfrec_dir).with_suffix('')
        self.test_path = pathlib.Path(self.photo_tfrec_dir).with_suffix('')

        self.output_dir = self.working_dir + "submission/"
        #End of GAN config

        #Try to get TPU, refer to the tutorial:https://www.kaggle.com/code/amyjang/monet
        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)

    def new_dir(directory):
        cmd = "mkdir " + directory
        os.system(cmd)

    def download_data(self):
        if not os.path.exists(self.data_dir):
            cmd = "pip install opendatasets"
            os.system(cmd)
            import opendatasets as od
            od.download(self.dataset_url)

            new_dir(data_dir)
            new_dir(train_dir)
            new_dir(test_dir)
            new_dir(dir_true)
            new_dir(dir_false)
```

1. Data preparing

Downloading data from the source, extract the compressed files to local disk.

In [59]:
```
config = Config()
config.download_data()
```

```
Working locally
Number of replicas: 1
```

Exploratory Data Analysis (EDA)

The dataset includes the real painting from Monet which is taken as training dataset in this notebook. The dataset also includes 7000+ photos to be taken as testing dataset. At the end of the notebook, the trained generator will be used to generate Monet painting from the testing dataset.

First, let's take a look at some Monet paintings to have some impression of how it looks like.

In [60]:
```
class TFRecordCoder:
    features = {
        "image_name": tf.io.FixedLenFeature([], tf.string),
        "image": tf.io.FixedLenFeature([], tf.string),
        "target": tf.io.FixedLenFeature([], tf.string)
    }

    def reader(item):
        entry = tf.io.parse_single_example(item, TFRecordCoder.features)
        image_name = entry['image_name']

        image = tf.image.decode_jpeg(entry['image'], channels=3)
        image = (tf.cast(image, tf.float32) / 127.5) - 1
        image = tf.reshape(image, [*Constants.TARGET_SIZE, 3])
        return image, image_name

    def decode(path):
        files = list(path.glob('*.tfrec'))
        print(files)
        ds = tf.data.TFRecordDataset(files)
        return ds.map(
            TFRecordCoder.reader,
            num_parallel_calls=tf.data.AUTOTUNE
        ).batch(batch_size=Constants.BATCH_SIZE)

    def recover(image):
        return (image +1)*127.5
```

In [61]:
```
class Dataset:
    def __init__(self, path, is_training = True, training_ratio = 0.8):
        self.path = path
        self.is_training = is_training
        self.training_ratio = training_ratio if training_ratio > 0 and  training_ratio <

        self.ds = None
        self.preprocessed_ds = None

#####

class DatasetBuilder:
    def build(path, is_training = True, training_ratio = 0.8):
        owner = Dataset(path, is_training, training_ratio)

        builder = DatasetBuilder(owner)
        builder.build_dataset()
        return builder
```

```
    def __init__(self,owner = None): # MUST set owner before using
        self._owner = owner


    def reset_owner(self, owner):
        self._owner = owner

    def owner(self):
        return self._owner


    def build_dataset(self):
        self.owner().ds = TFRecordCoder.decode(self.owner().path)

    def take_processed_for_show(ds, num):
        images = None
        try:
            i = 0
            it = iter(ds)
            return next(it)

            while True:
                batch = next(it)
                for k in range(len(batch)):
                    img = batch[k]
                    img = DatasetBuilder.rollback_one(img.numpy()).astype("uint8")
                    if images == None:
                        images = [img]
                    else:
                        images.append(img)
                    i += 1
                    if i >= num:
                        return  images

        except:
            print("To the end after {} loops.".format(i))

        return images

    def comparing_images(imgs0, imgs1,title=""):
        l0 = len(imgs0)
        l1 = len(imgs1)
        num = min(2,min(l0,l1))
        fig = plt.figure(figsize=(16,16))
        fig_width = 2
        fig_height = num
        ax = fig.subplots(fig_height,fig_width)
        plt.title(title, loc='center')
        for i in range(num):
            im0 = imgs0[i]
            im1 = imgs1[i]
            (iax0, iax1) = (ax[0],ax[1]) if num == 1 else (ax[i][0], ax[i][1])
            iax0.imshow(im0)
            iax1.imshow(im1)


        plt.show()
        return


    def show_processed_images(ds):
        num = 9
        fig = plt.figure(figsize=(16,16))
        fig_width = 3
        fig_height = num//3
```

```
        num = fig_height * fig_width
        ax = fig.subplots(fig_height,fig_width)
        try:
            i = 0
            it = iter(ds)
            while True:
                batch = next(it)
                for k in range(len(batch)):
                    img = batch[k]
                    #id = batch[1][k]

                    iax = ax[i//fig_height][i%fig_width]
                    iax.imshow(
                        DatasetBuilder.rollback_one(img.numpy()).astype("uint8")
                    )
                    #iax.set_title(id.numpy().decode())
                    i += 1
                    if i >= num:
                        return

        except:
            print("To the end after {} loops.".format(i))

    def rollback_one(image):
        return image*255

    def normalize(images):  # Convert from [-1,1] to [0,1]
        images = (images + 1)*0.5
        return images

    #Add random noise to the image
    def random_jitter(images):
        # resizing to 286 x 286 x 3 then randomly cropping
        shape = images.shape
        shape = [Constants.BATCH_SIZE,shape[1], shape[2], shape[3]]
        images = tf.image.resize(images, Constants.RESIZE, method=tf.image.ResizeMethod.

        images = tf.image.random_crop(images, size=shape, seed = 111)

        # random mirroring
        images = tf.image.random_flip_left_right(images)
        images = tf.image.random_flip_up_down(images)

        images = tf.image.random_saturation(images,90,100)
        return images

    #Add noise to enhace training accuracy
    def preprocess_image(self, images, ids):
        if self.owner().is_training:
            print("Training dataset")
            images = DatasetBuilder.random_jitter(images)
            images = DatasetBuilder.normalize(images)
            return images
        else: #For testing image, just needs to normalize to the same size range as trai
            print("Testing dataset")
            return DatasetBuilder.normalize(images)

    def preprocess(self):
        self.owner().preprocessed_ds = self.owner().ds.map(
            self.preprocess_image,
            num_parallel_calls=tf.data.AUTOTUNE
        )

        return self.owner().preprocessed_ds
        #print("Shape:", self.owner().preprocessed_ds, self.owner().ds)
```

```
    def show_images(self):
        num = 9
        fig = plt.figure(figsize=(16,16))
        fig_width = 3
        fig_height = num//3

        num = fig_height * fig_width
        ax = fig.subplots(fig_height,fig_width)
        try:
            i = 0
            it = iter(self.owner().ds)
            while True:
                batch = next(it)
                for k in range(len(batch[0])):
                    img = batch[0][k]
                    id = batch[1][k]

                    iax = ax[i//fig_height][i%fig_width]
                    iax.imshow( TFRecordCoder.recover(img.numpy()).astype("uint8"))
                    iax.set_title(id.numpy().decode())
                    i += 1
                    if i >= num:
                        return

        except:
            print("To the end after {} loops.".format(i))

    def statistics(self):
        print("Shape of the dataset:")
        for ids_batch in self.owner().id_ds:
            print(image_batch.shape)
            print(ids_batch.shape)
            break

        for image_batch in self.owner().image_ds:
            print(image_batch.shape)
            break
```

In [62]:
```
train_ds = DatasetBuilder.build(config.train_path, True).owner()
```

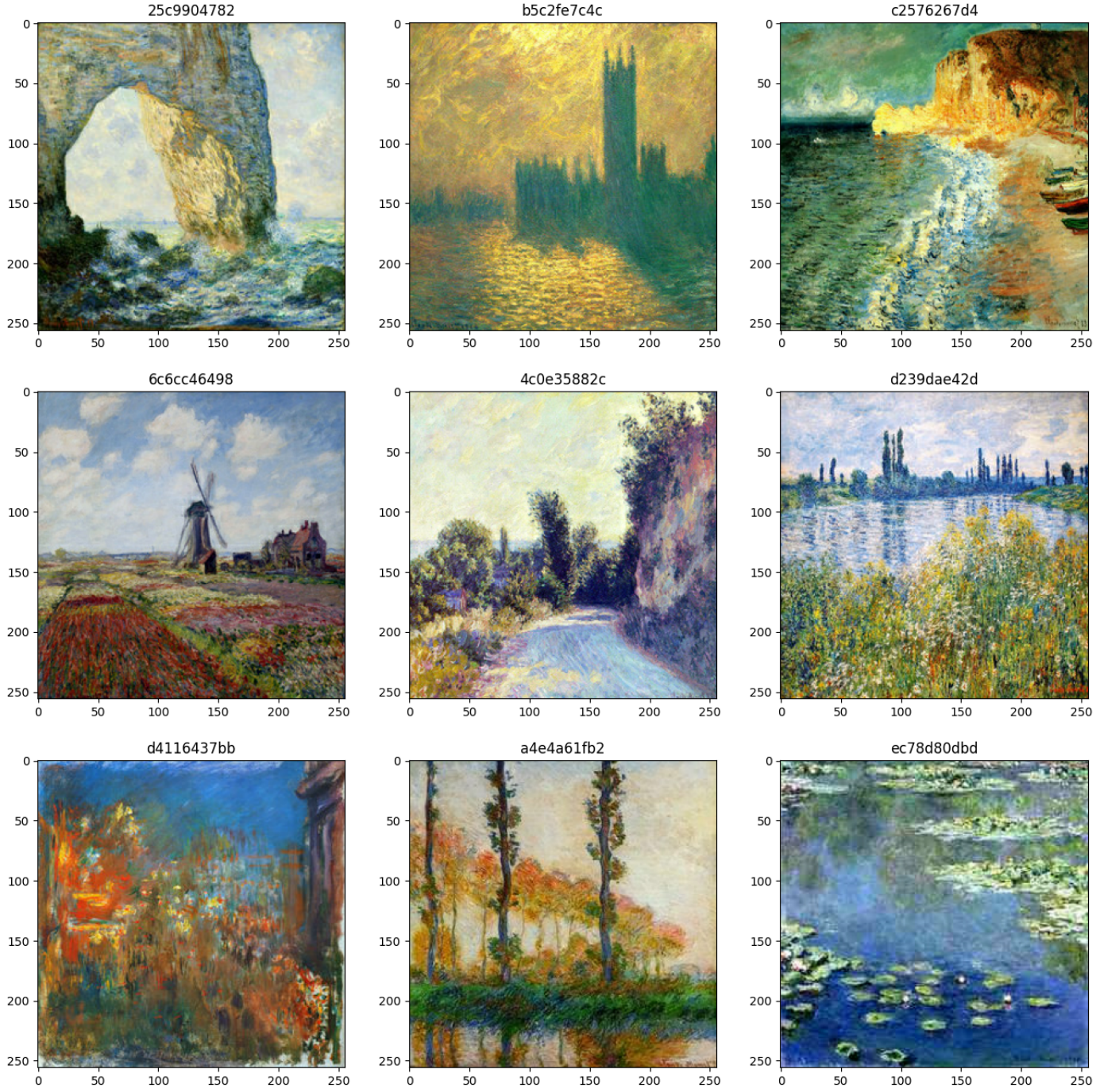
```
[PosixPath('gan-getting-started/monet_tfrec/monet00-60.tfrec'), PosixPath('gan-getting-s
tarted/monet_tfrec/monet16-60.tfrec'), PosixPath('gan-getting-started/monet_tfrec/monet0
8-60.tfrec'), PosixPath('gan-getting-started/monet_tfrec/monet04-60.tfrec'), PosixPath
('gan-getting-started/monet_tfrec/monet12-60.tfrec')]
```

In [63]:
```
print(train_ds.ds)
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name
=None), TensorSpec(shape=(None,), dtype=tf.string, name=None))>
```

In [64]:
```
train_builder = DatasetBuilder(train_ds)
```

In [65]:
```
train_builder.show_images()
```



In [66]:
```
test_ds = DatasetBuilder.build(config.test_path, False).owner()
```

```
[PosixPath('gan-getting-started/photo_tfrec/photo10-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo04-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo09-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo16-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo02-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo05-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo11-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo08-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo03-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo17-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo12-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo06-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo19-350.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo14-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo00-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo07-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo13-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo01-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo15-352.tfrec'), PosixPath('gan-getting-started/photo_tfrec/photo18-352.tfrec')]
```

In [67]:
```
print(test_ds.ds)
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name
```

=None), TensorSpec(shape=(None,), dtype=tf.string, name=None))>

In [68]:
```
test_builder = DatasetBuilder(test_ds)
```

Then, have another look at the regular photos to compare with the paintings.

In [69]:
```
test_builder.show_images()
```

The original dataset includes both jpeg format and compressed TFRecord format. To be sake of the performance, the tfrec format will be used for training.

Model Architecture and Training Methods

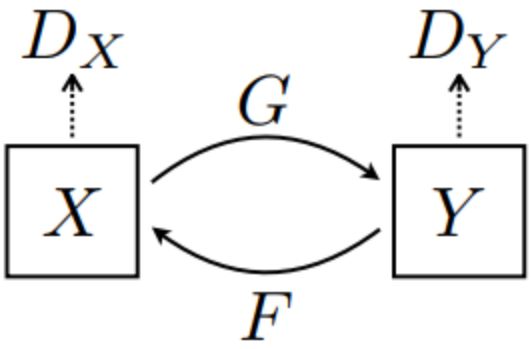
Refer to the tutorial https://www.tensorflow.org/tutorials/generative/cyclegan?authuser=8, this notenook builds and train the CycleGan model with unet, the main model and generator are imported and reused from the pix2pix models with adjustments on the optimizer and loss functions.

As explained in the tutorial:

"

There are 2 generators (G and F) and 2 discriminators (X and Y) being trained here.

Generator G learns to transform image X to image Y. Generator F learns to transform image Y to image X. Discriminator D_X learns to differentiate between image X and generated image X (F(Y)). Discriminator D_Y learns to differentiate between image Y and generated image Y (G(X)).



"

In this notebook, here is the mapping between the components of the task and the general concept:

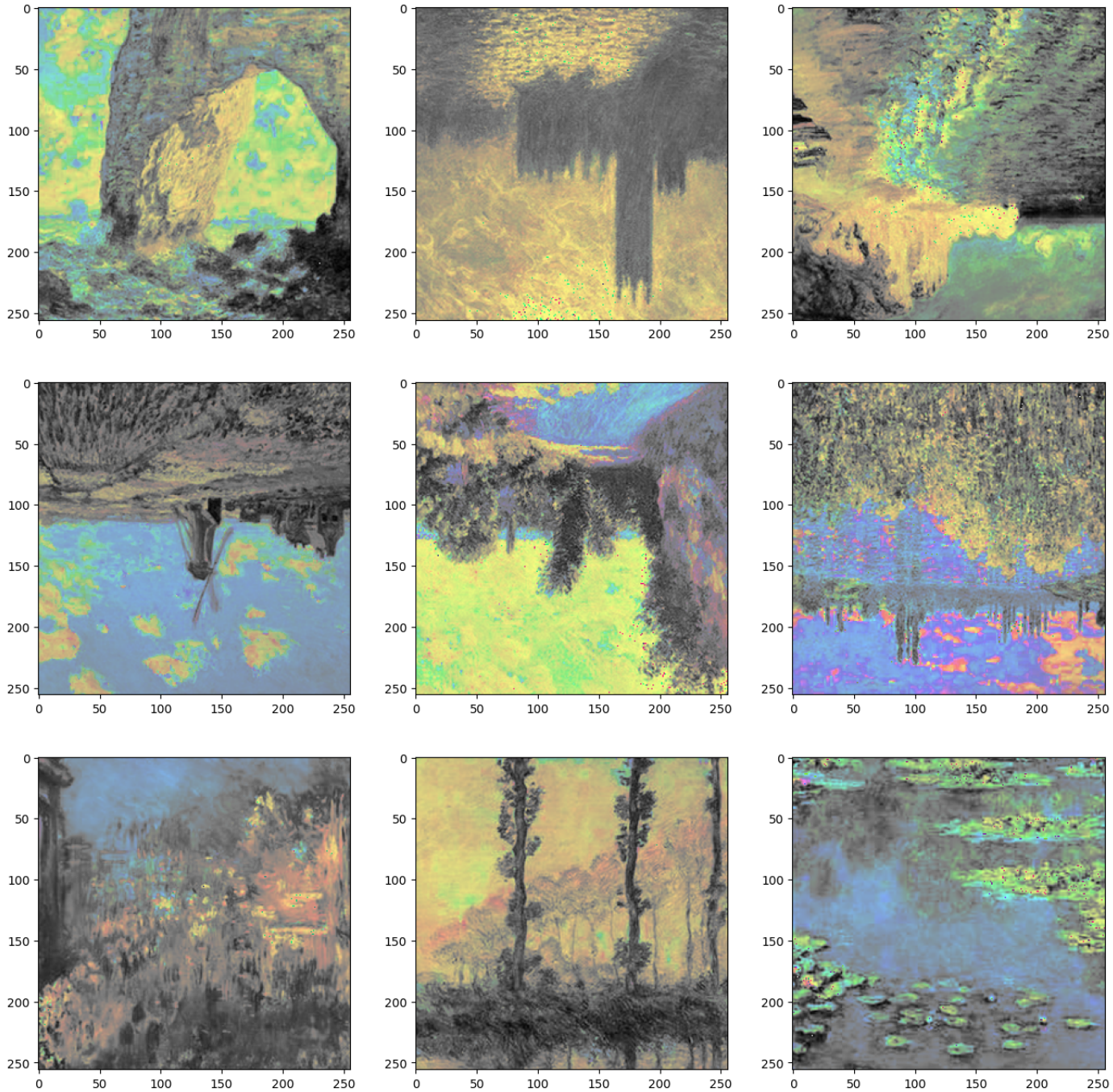
- X <> photo (discriminator)
- Y <> Monet (discriminator)
- G <> From photo to Monet (generator)
- F <> From Monet to photo (generator)

In [70]:
```
train_builder.preprocess()
```

Out[70]:
```
Training dataset
<ParallelMapDataset element_spec=TensorSpec(shape=(4, 256, 256, 3), dtype=tf.float32, na
me=None)>
```

Build the dataset and veirify.

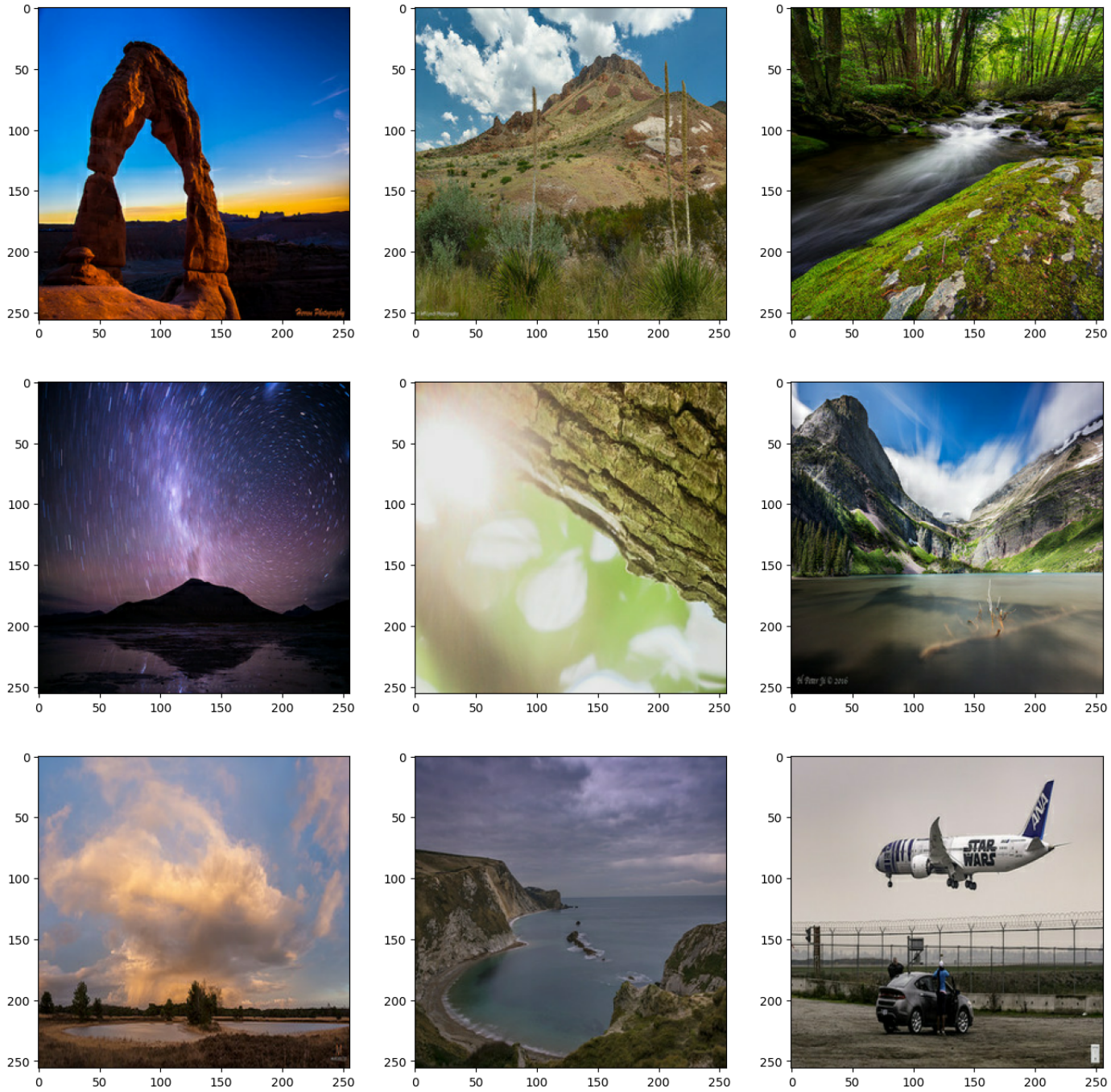
In [71]:
```
DatasetBuilder.show_processed_images(train_ds.preprocessed_ds)
```



In [72]:
```
test_builder.preprocess()
```

Out[72]:
```
Testing dataset
<ParallelMapDataset element_spec=TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32,
name=None)>
```

In [73]:
```
DatasetBuilder.show_processed_images(test_ds.preprocessed_ds)
```



In [74]:
```
!pip install git+https://github.com/tensorflow/examples.git
```

```
Collecting git+https://github.com/tensorflow/examples.git
  Cloning https://github.com/tensorflow/examples.git to /private/var/folders/qr/g0b69pq564xg82p0g6bvvwvh0000gn/T/pip-req-build-9wz5jn9l
  Running command git clone --filter=blob:none --quiet https://github.com/tensorflow/examples.git /private/var/folders/qr/g0b69pq564xg82p0g6bvvwvh0000gn/T/pip-req-build-9wz5jn9l
  Resolved https://github.com/tensorflow/examples.git to commit 37df0b9a2f7240831642cea4380e4f7ec33bee83
  Preparing metadata (setup.py) ... done
Requirement already satisfied: absl-py in /opt/homebrew/Caskroom/miniconda/base/envs/dl/lib/python3.9/site-packages (from tensorflow-examples===37df0b9a2f7240831642cea4380e4f7ec33bee83-) (1.4.0)
Requirement already satisfied: six in /opt/homebrew/Caskroom/miniconda/base/envs/dl/lib/python3.9/site-packages (from tensorflow-examples===37df0b9a2f7240831642cea4380e4f7ec33bee83-) (1.16.0)
```

Import and reuse the models from pix2pix

Here is the utility functions to build the GAN modle,refer to the tutorials

https://www.tensorflow.org/tutorials/generative/cyclegan?authuser=8.

The following function set are defined in the utility class:

1. generator: from Monet painting to photo and vice versa
2. discriminator: classification for Monet painting and photo respectively
3. loss and gradient differential calculation
4. optimizer

In [75]:
```
from tensorflow_examples.models.pix2pix import pix2pix as p2p
```

In [76]:
```
#Refer to the tutorial:https://www.tensorflow.org/tutorials/generative/cyclegan?authuser
class Generator:
    def __init__(self, train_ds, test_ds):
        self.train_ds = train_ds  #Monet painting, Y
        self.test_ds = test_ds    #Photo, X
        self.g_p2m_generator = p2p.unet_generator(Constants.OUTPUT_CHANNELS, norm_type='
        self.f_m2p_generator = p2p.unet_generator(Constants.OUTPUT_CHANNELS, norm_type='

        self.x_photo_discriminator = p2p.discriminator(norm_type='instancenorm', target=
        self.y_monet_discriminator = p2p.discriminator(norm_type='instancenorm', target=


    def test_generator(self):
        imgs0 = DatasetBuilder.take_processed_for_show(self.train_ds.preprocessed_ds,2)
        imgs1 = self.f_m2p_generator(imgs0)
        print()
        DatasetBuilder.comparing_images(imgs0,imgs1, "Painting VS Photo from Painting")


        imgs2 = self.g_p2m_generator(imgs1)
        print()
        DatasetBuilder.comparing_images(imgs0,imgs2,"Painting VS Painting from Photo(fro


        imgs0 = DatasetBuilder.take_processed_for_show(self.test_ds.preprocessed_ds,2)
        imgs1 = self.g_p2m_generator(imgs0)
        print()
        DatasetBuilder.comparing_images(imgs0,imgs1,"Photo VS Painting from Photo")


        imgs2 = self.f_m2p_generator(imgs1)
        print()
        DatasetBuilder.comparing_images(imgs0,imgs2,"Photo VS Photo from Painting(from t

    def test_monet_generator(self):
        imgs0 = DatasetBuilder.take_processed_for_show(self.test_ds.preprocessed_ds,1)
        imgs1 = self.g_p2m_generator(imgs0)
        print()
        DatasetBuilder.comparing_images(imgs0,imgs1,"Photo VS Painting from Photo")


        #imgs2 = self.f_m2p_generator(imgs1)
        #print()
        #DatasetBuilder.comparing_images(imgs0,imgs2,"Photo VS Photo from Painting(from


    def test_discriminator(self):
        imgs0 = DatasetBuilder.take_processed_for_show(self.train_ds.preprocessed_ds,1)
        accuracy_origin = self.y_monet_discriminator(imgs0)[0]
        imgs1 = self.g_p2m_generator(self.f_m2p_generator(imgs0))
        accuracy_regen = self.y_monet_discriminator(imgs1)[0]
```

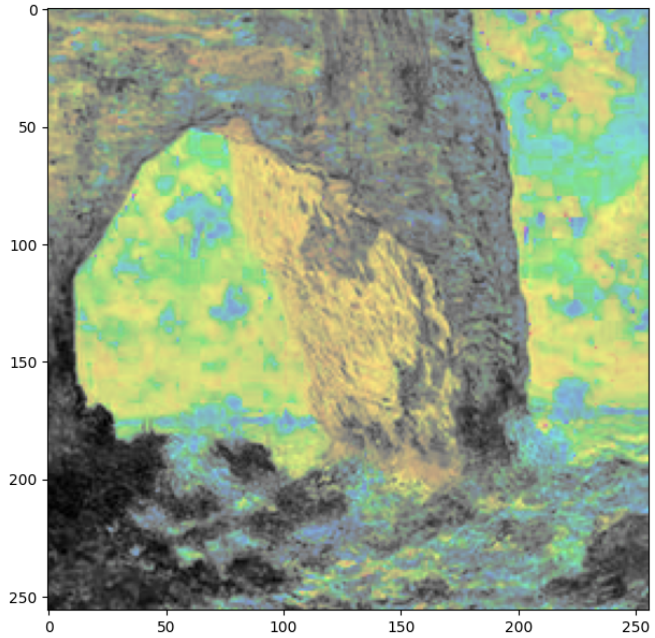
```
        plt.figure(figsize=(16, 16))
        plt.subplot(121)
        plt.title('Real painting')
        plt.imshow(accuracy_origin, cmap='RdBu_r')

        plt.subplot(122)
        plt.title('Regenerated painting')
        plt.imshow(accuracy_regen, cmap='RdBu_r')

        plt.show()

        imgs0 = DatasetBuilder.take_processed_for_show(self.test_ds.preprocessed_ds,1)
        accuracy_origin = self.x_photo_discriminator(imgs0)[0]
        imgs1 = self.f_m2p_generator(self.g_p2m_generator(imgs0))
        accuracy_regen = self.x_photo_discriminator(imgs1)[0]

        plt.figure(figsize=(16, 16))
        plt.subplot(121)
        plt.title('Real photo')
        plt.imshow(accuracy_origin, cmap='RdBu_r')

        plt.subplot(122)
        plt.title('Regenerated photo')
        plt.imshow(accuracy_regen, cmap='RdBu_r')

        plt.show()
```

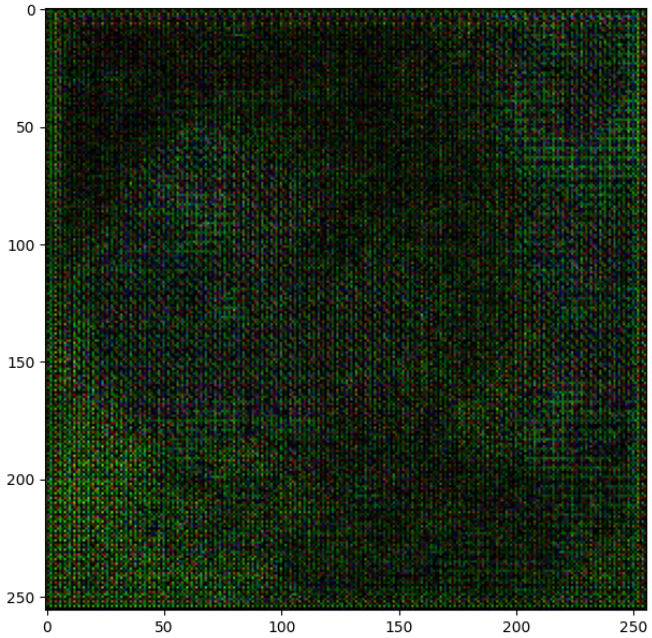
In [77]:
```
g = Generator(train_ds,test_ds)
```

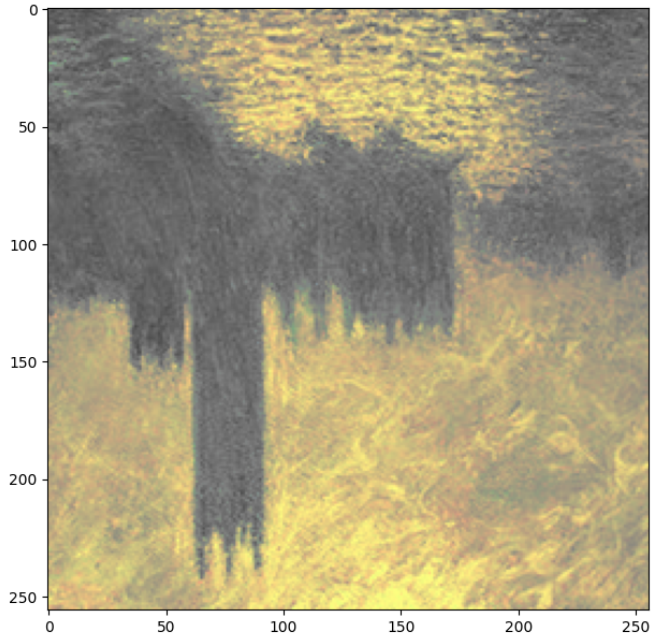
Before training, generate two sample images between photo and Monet painting along with watching the result of discriminators with respective generator and discriminators, to understand the use of the model.

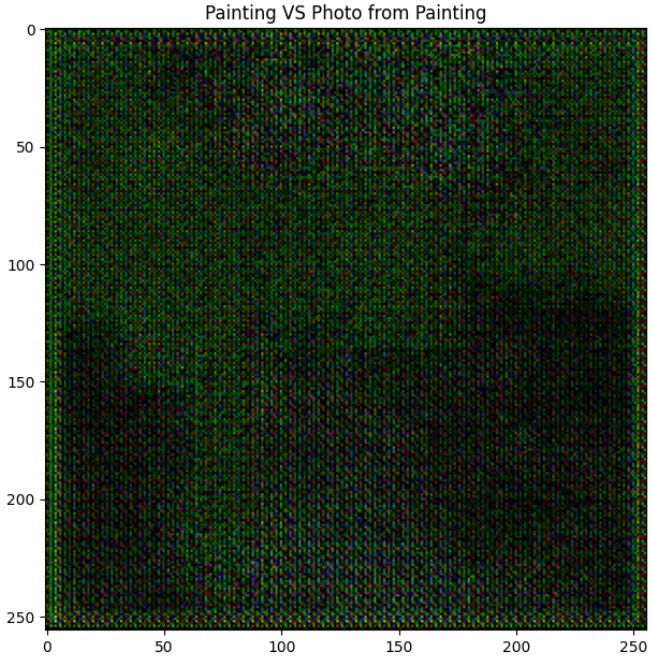
In [78]:
```
g.test_generator()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

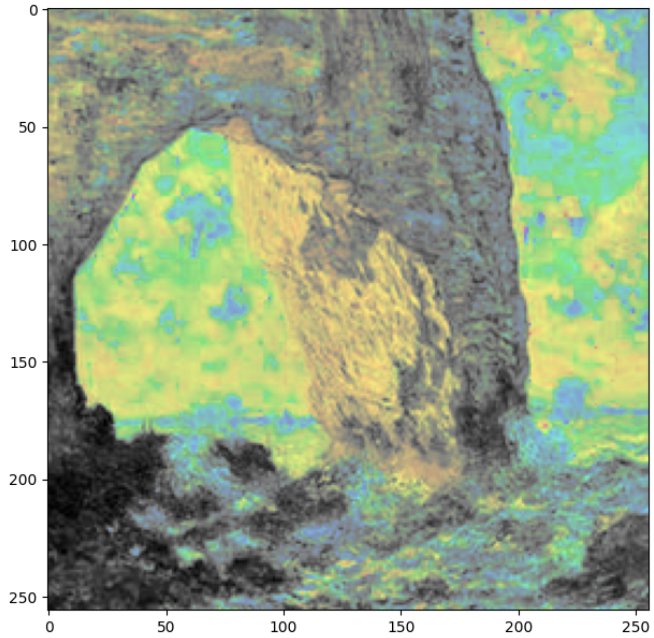


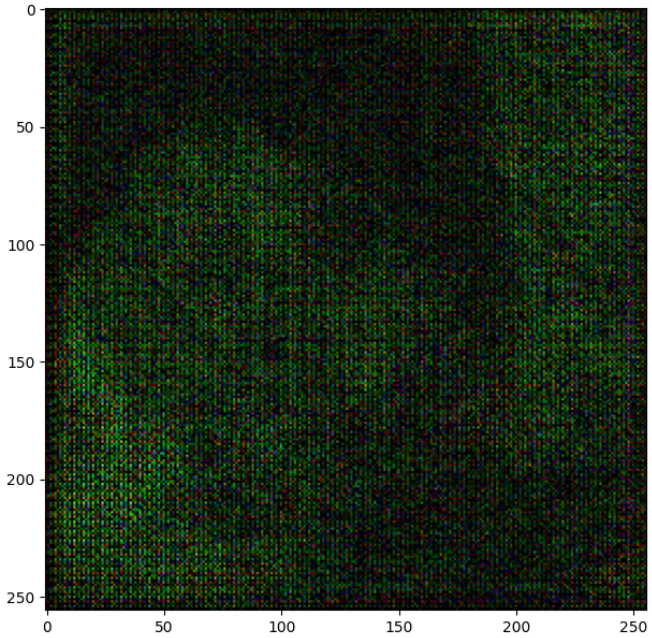


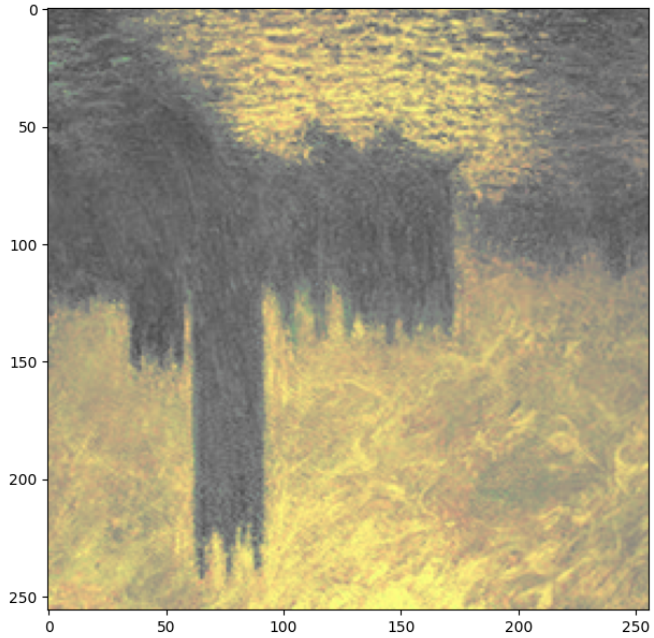


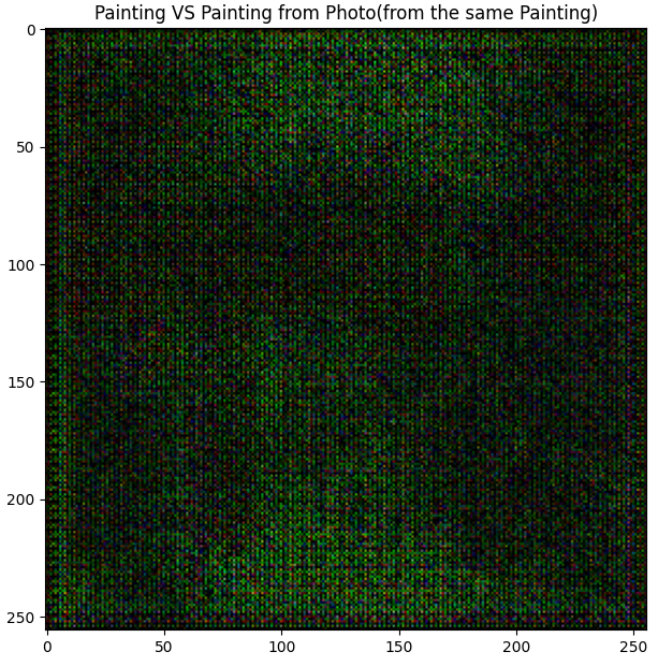



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

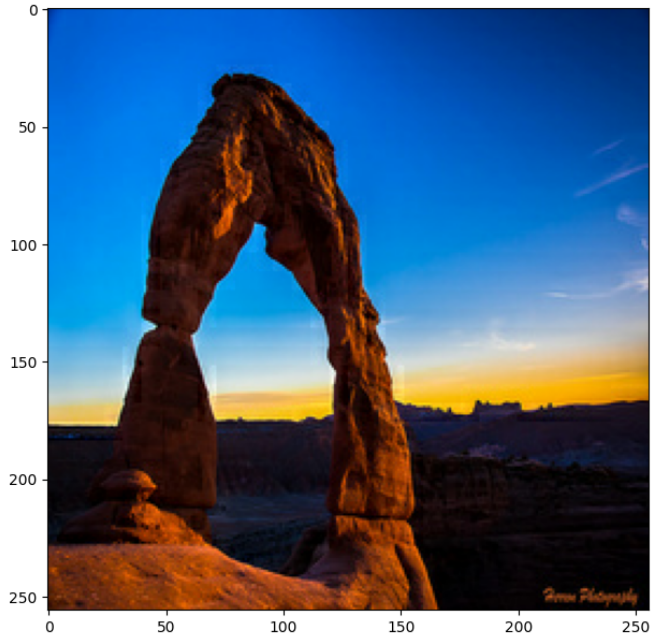


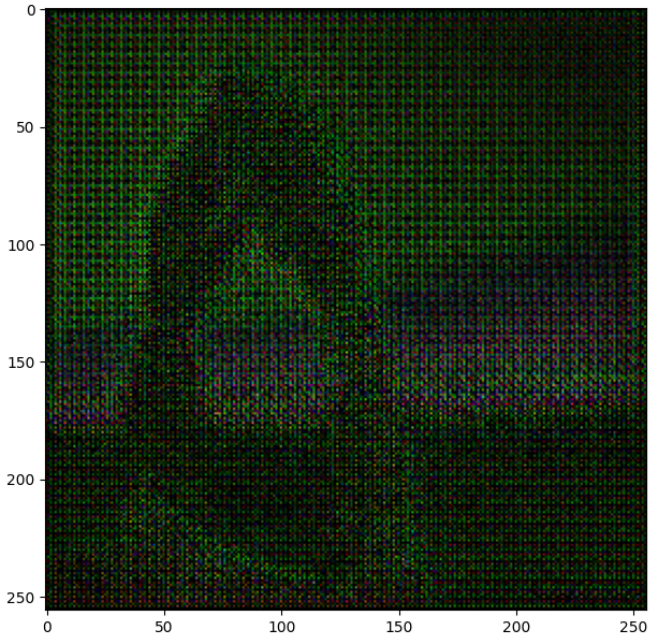


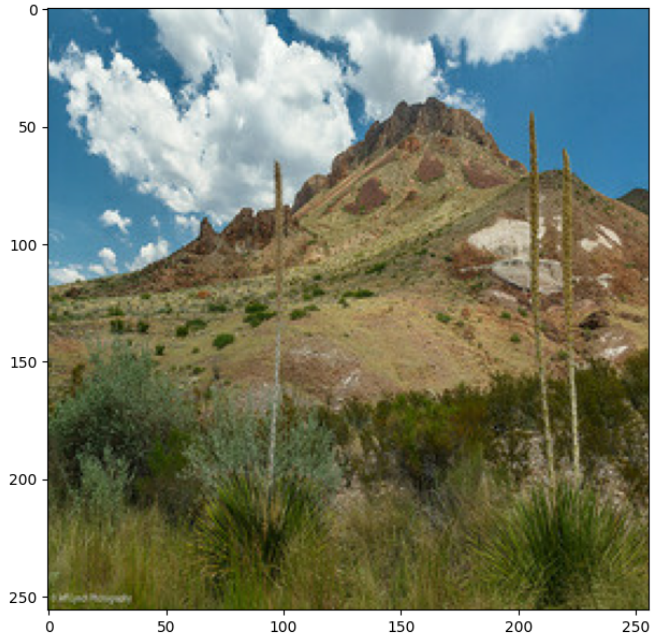


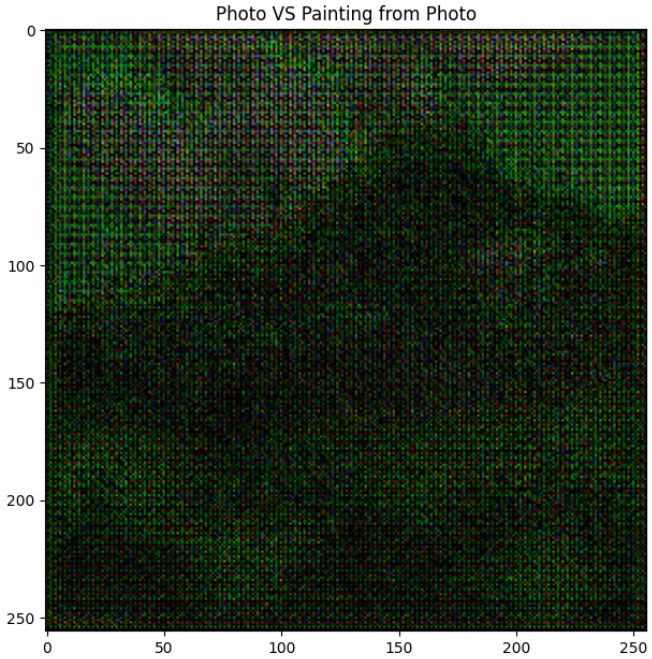



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

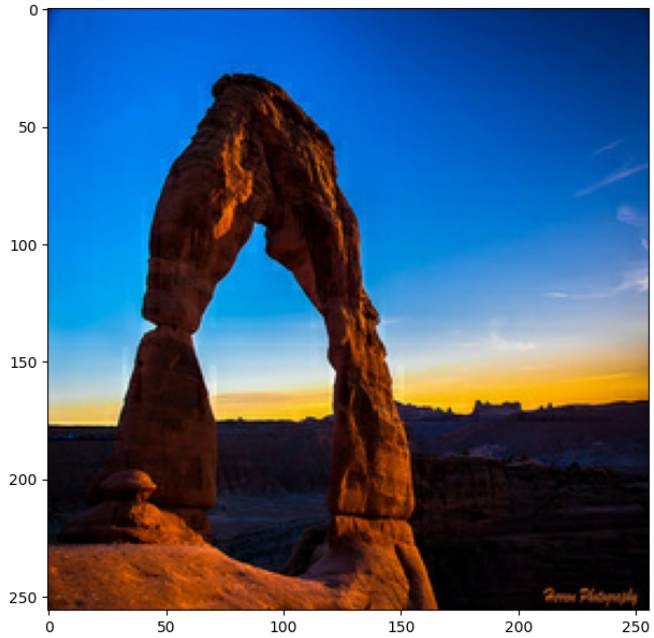


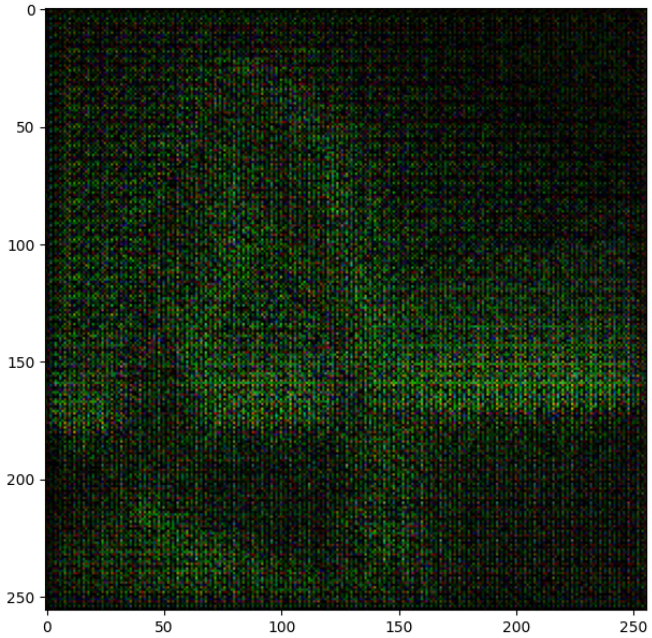


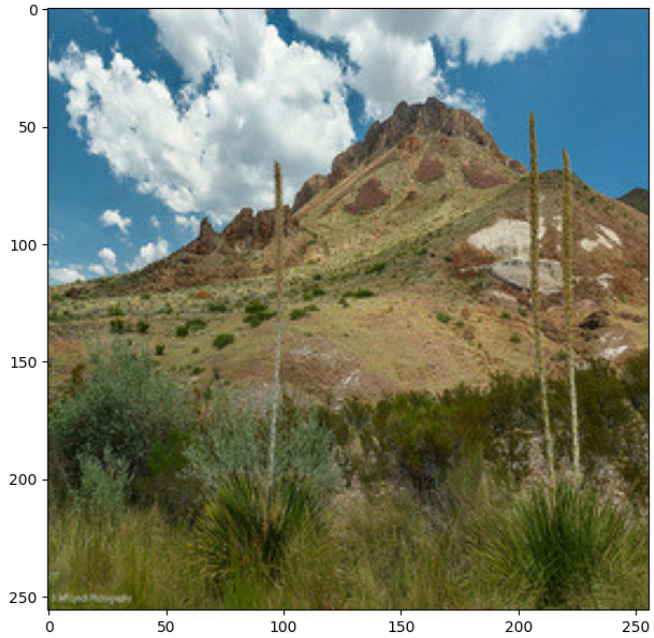


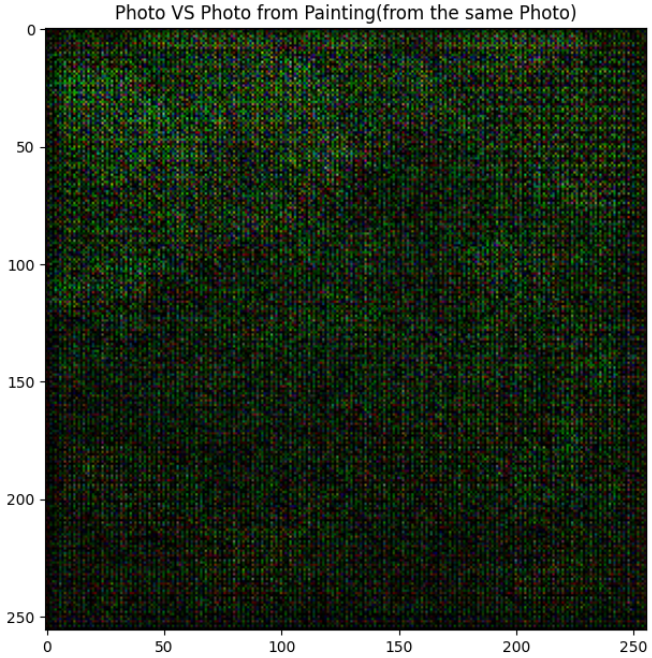



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



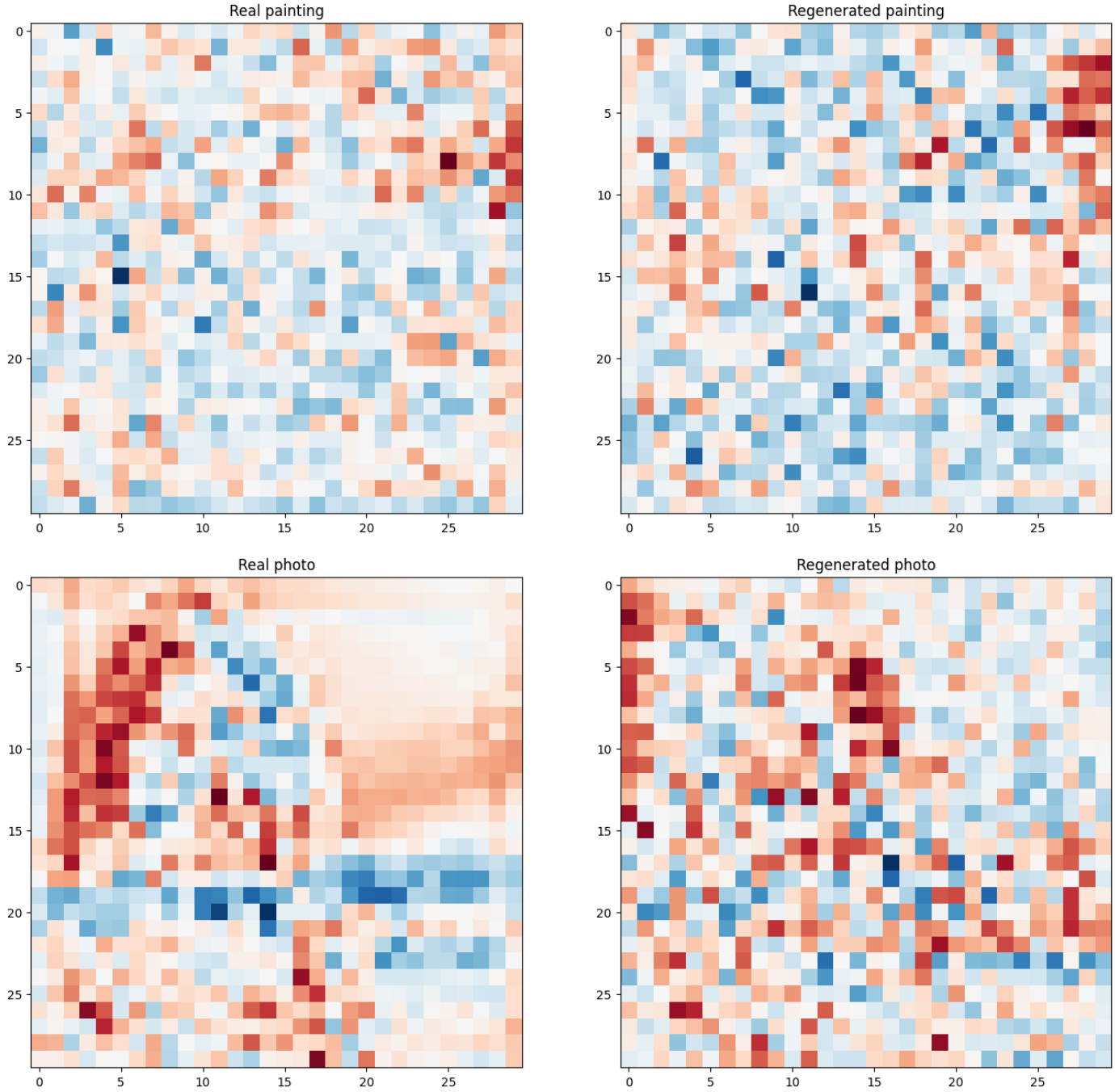






Check the output of the discriminator on the real image and the generated ones refer to the tutorial https://www.tensorflow.org/tutorials/generative/cyclegan?authuser=8.

In [79]:
```
g.test_discriminator()
```



Loss function and optimizer for cycle GAN in both directions. Refer to: https://www.tensorflow.org/tutorials/generative/cyclegan?authuser=8 following the paper: "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks"

In [80]:
```
class Loss:
    loss_func = tf.keras.losses.BinaryCrossentropy(from_logits=True)

    # Consistent loss
    def discriminator_loss(origin, generated):
        origin_loss = Loss.loss_func(tf.ones_like(origin), origin)
        generated_loss = Loss.loss_func(tf.zeros_like(generated), generated)
        total_disc_loss = origin_loss + generated_loss
        return total_disc_loss * 0.5

    # Consistent loss
    def generator_loss(disc_gen):
        return Loss.loss_func(tf.ones_like(disc_gen), disc_gen)

    #Run X->Y then Y->X with x, then x(origin) and x-gen(cycled) shall not have big diff
    def cycle_loss(origin_image, cycled_image):
```

```
        return Constants.LAMBDA *  tf.reduce_mean(tf.abs(origin_image - cycled_image))

    # Run X->Y with y as x, then y(origin) and y-gen(same) shall not have big difference
    def identity_loss(origin_image, same_image):
        return Constants.LAMBDA * 0.5 * tf.reduce_mean(tf.abs(origin_image - same_image))
```

In [81]:
```
from tensorflow.keras.optimizers.legacy import Adam
class Optimizer:
    optimizer_g_p2m_genarator = Adam(2e-4, beta_1=0.5)
    optimizer_f_m2p_genarator = Adam(2e-4, beta_1=0.5)
    optimizer_x_photo_discriminator = Adam(2e-4, beta_1=0.5)
    optimizer_y_monet_discriminator = Adam(2e-4, beta_1=0.5)
```

Build the training functions refer to the tutorial. Although the challenge is to generate Monet from regular photo, as mentioned in previous section, the CycleGAN will always perform training at both directions to adjust the parameters based on the self-regulated variances. In brief, the training includes the following steps:

- Prediction: Generate from instance of X to Y and then generate back from Y to X Same on the other cycle
- Loss calculation: Calculate the total loss at X and Y side
- Gradient calculation: Calculate the gradient using back-propagation with the calculated loss
- Optimize and adjust the parameters based on the gradient values.

In [82]:
```
class CycleGanModel:
    def __init__(self, train_ds, test_ds):
        self.generator = Generator(train_ds,test_ds)
        self.loss = Loss
        self.optimizer = Optimizer
        self.ckpt_manager = None

        self.init_check_point()

    def train_step(self, real_photo, real_monet):
        #The trainning function is from the tutorial: https://www.tensorflow.org/tutoria
        #To avoid too much code and change while keeping the semantic for the current pr
        #x==>photo, y==>monet
        #g==>photo to monet, f==>monet to photo
        real_x = real_photo
        real_y = real_monet


        # persistent is set to True because the tape is used more than
        # once to calculate the gradients.
        with tf.GradientTape(persistent=True) as tape:
            # Generator G translates X (photo)-> Y (monet)
            # Generator F translates Y -> X.

            fake_y = self.generator.g_p2m_generator(real_x, training=True)
            cycled_x = self.generator.f_m2p_generator(fake_y, training=True)

            fake_x = self.generator.f_m2p_generator(real_y, training=True)
            cycled_y = self.generator.g_p2m_generator(fake_x, training=True)

            # same_x and same_y are used for identity loss.
            # X as Y to generate X (Y->X where y=x)
            # Y as X to generate Y (X->Y where x=y)
            same_x = self.generator.f_m2p_generator(real_x, training=True)
            same_y = self.generator.g_p2m_generator(real_y, training=True)

            disc_real_x = self.generator.x_photo_discriminator(real_x, training=True)
```

```
            disc_real_y = self.generator.y_monet_discriminator(real_y, training=True)

            disc_fake_x = self.generator.x_photo_discriminator(fake_x, training=True)
            disc_fake_y = self.generator.y_monet_discriminator(fake_y, training=True)

            # calculate the loss
            gen_g_loss = self.loss.generator_loss(disc_fake_y)
            gen_f_loss = self.loss.generator_loss(disc_fake_x)

            total_cycle_loss = self.loss.cycle_loss(real_x, cycled_x) + self.loss.cycle_

            # Total generator loss = adversarial loss + cycle loss
            total_gen_g_loss = gen_g_loss + total_cycle_loss + self.loss.identity_loss(r
            total_gen_f_loss = gen_f_loss + total_cycle_loss + self.loss.identity_loss(r

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

            # Calculate the gradients for generator and discriminator
            g_p2m_generator_gradients = tape.gradient(
                total_gen_g_loss,
                self.generator.g_p2m_generator.trainable_variables
            )
            f_m2p_generator_gradients = tape.gradient(
                total_gen_f_loss,
                self.generator.f_m2p_generator.trainable_variables
            )

            x_photo_discriminator_gradients = tape.gradient(
                disc_x_loss,
                self.generator.x_photo_discriminator.trainable_variables
            )
            y_monet_discriminator_gradients = tape.gradient(
                disc_y_loss,
                self.generator.y_monet_discriminator.trainable_variables
            )

        # Apply the gradients to the optimizer on the related trainable_variables
        # Outside of with context
        self.optimizer.optimizer_g_p2m_genarator.apply_gradients(
            zip(g_p2m_generator_gradients,
                self.generator.g_p2m_generator.trainable_variables)
        )

        self.optimizer.optimizer_f_m2p_genarator.apply_gradients(
            zip(f_m2p_generator_gradients,
                self.generator.f_m2p_generator.trainable_variables)
        )

        self.optimizer.optimizer_x_photo_discriminator.apply_gradients(
            zip(x_photo_discriminator_gradients,
                self.generator.x_photo_discriminator.trainable_variables)
        )

        self.optimizer.optimizer_y_monet_discriminator.apply_gradients(
            zip(y_monet_discriminator_gradients,
                self.generator.y_monet_discriminator.trainable_variables)
        )

    def init_check_point(self):
        checkpoint_path = config.working_dir + "checkpoints/"
        if not os.path.exists(checkpoint_path):
            os.mkdir(checkpoint_path)


        #x==>photo, y==>monet
```

```
        #g==>photo to monet, f==>monet to photo
        ckpt = tf.train.Checkpoint(generator_g=self.generator.g_p2m_generator,
                                   generator_f=self.generator.f_m2p_generator,
                                   discriminator_x=self.generator.x_photo_discriminator,
                                   discriminator_y=self.generator.y_monet_discriminator,
                                   generator_g_optimizer=self.optimizer.optimizer_g_p2m_
                                   generator_f_optimizer=self.optimizer.optimizer_f_m2p_
                                   discriminator_x_optimizer=self.optimizer.optimizer_x_
                                   discriminator_y_optimizer=self.optimizer.optimizer_y_

        self.ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_kee

        # if a checkpoint exists, restore the latest checkpoint.
        if self.ckpt_manager.latest_checkpoint:
            ckpt.restore(self.ckpt_manager.latest_checkpoint)
            print ('Latest checkpoint restored!!')



    def check_result(self,epoch):
        print("Check result for epoch {}.".format(epoch))
        self.generator.test_monet_generator()

    def fit(self):
        if True: #not os.path.exists(saved_monet_generator):
            print("First time, start training.")
            try:
                os.makedirs(saved_model_path)
            except:
                pass

            train_start = timer()

            test_builder = DatasetBuilder(self.generator.test_ds)
            train_builder = DatasetBuilder(self.generator.train_ds)
            test_ds = test_builder.preprocess().shuffle(buffer_size = 100) # Preprocess
            train_ds = train_builder.preprocess().shuffle(buffer_size = 100)

            epochs = 0
            for epoch in range(Constants.EPOCHS):
                epoch_start = timer()

                n = 0
                for image_x, image_y in tf.data.Dataset.zip((test_ds, train_ds)):
                    self.train_step(image_x, image_y)
                    if n % 10 == 0:
                        print ('.', end='')
                    n += 1

                self.check_result(epoch+1)

                if (epoch + 1) % 5 == 0:
                    ckpt_save_path = self.ckpt_manager.save()
                    print ('Saving checkpoint for epoch {} at {}'.format(epoch+1, ckpt_s

                print ('Time taken for epoch {} is {} sec\n'.format(epoch+1, timer()-epo
                epochs += 1

            print ('Time taken for {} epochs is {} sec\n'.format(epochs, timer()-train_s

            #monet_generator.save(saved_monet_generator)
            #photo_generator.save(saved_photo_generator)
        else:
            monet_generator = tf.keras.models.load_model(saved_monet_generator)
            photo_generator = tf.keras.models.load_model(saved_photo_generator)
```

```
    def summary(self):
        self.generator.g_p2m_generator.summary()
        self.generator.f_m2p_generator.summary()
```

In [83]:
```
m = CycleGanModel(train_ds,test_ds)
```

Have a glance of the models before training.

In [84]:
```
m.summary()
```

```
Model: "model_20"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_11 (InputLayer)       [(None, None, None,  0           []
                              3)]

 sequential_180 (Sequential)  (None, None, None,   3072        ['input_11[0][0]']
                              64)

 sequential_181 (Sequential)  (None, None, None,   131328      ['sequential_180[0]
[0]']
                              128)

 sequential_182 (Sequential)  (None, None, None,   524800      ['sequential_181[0]
[0]']
                              256)

 sequential_183 (Sequential)  (None, None, None,   2098176     ['sequential_182[0]
[0]']
                              512)

 sequential_184 (Sequential)  (None, None, None,   4195328     ['sequential_183[0]
[0]']
                              512)

 sequential_185 (Sequential)  (None, None, None,   4195328     ['sequential_184[0]
[0]']
                              512)

 sequential_186 (Sequential)  (None, None, None,   4195328     ['sequential_185[0]
[0]']
                              512)
```

```
 sequential_187 (Sequential)    (None, None, None,   4195328     ['sequential_186[0]
[0]']
                                512)

 sequential_188 (Sequential)    (None, None, None,   4195328     ['sequential_187[0]
[0]']
                                512)

 concatenate_10 (Concatenate)   multiple             0           ['sequential_188[0]
[0]',
                                                                  'sequential_186[0]
[0]',
                                                                  'sequential_189[0]
[0]',
                                                                  'sequential_185[0]
[0]',
                                                                  'sequential_190[0]
[0]',
                                                                  'sequential_184[0]
[0]',
                                                                  'sequential_191[0]
[0]',
                                                                  'sequential_183[0]
[0]',
                                                                  'sequential_192[0]
[0]',
                                                                  'sequential_182[0]
[0]',
                                                                  'sequential_193[0]
[0]',
                                                                  'sequential_181[0]
[0]',
                                                                  'sequential_194[0]
[0]',
                                                                  'sequential_180[0]
[0]']

 sequential_189 (Sequential)    (None, None, None,   8389632     ['concatenate_10[0]
[0]']
                                512)

 sequential_190 (Sequential)    (None, None, None,   8389632     ['concatenate_10[1]
[0]']
                                512)

 sequential_191 (Sequential)    (None, None, None,   8389632     ['concatenate_10[2]
[0]']
                                512)

 sequential_192 (Sequential)    (None, None, None,   4194816     ['concatenate_10[3]
[0]']
                                256)
```

```
 sequential_193 (Sequential)    (None, None, None,   1048832     ['concatenate_10[4]
[0]']
                                128)

 sequential_194 (Sequential)    (None, None, None,   262272      ['concatenate_10[5]
[0]']
                                64)

 conv2d_transpose_87 (Conv2DTra  (None, None, None,   6147       ['concatenate_10[6]
[0]']
 nspose)                        3)


==================================================================================================
Total params: 54,414,979
Trainable params: 54,414,979
Non-trainable params: 0
__________________________________________________________________________________________________
Model: "model_21"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_12 (InputLayer)          [(None, None, None,  0           []
                                 3)]

 sequential_195 (Sequential)    (None, None, None,   3072        ['input_12[0][0]']
                                64)

 sequential_196 (Sequential)    (None, None, None,   131328      ['sequential_195[0]
[0]']
                                128)

 sequential_197 (Sequential)    (None, None, None,   524800      ['sequential_196[0]
[0]']
                                256)

 sequential_198 (Sequential)    (None, None, None,   2098176     ['sequential_197[0]
[0]']
                                512)

 sequential_199 (Sequential)    (None, None, None,   4195328     ['sequential_198[0]
[0]']
                                512)
```

```
 sequential_200 (Sequential)    (None, None, None,   4195328     ['sequential_199[0]
[0]']
                                512)


 sequential_201 (Sequential)    (None, None, None,   4195328     ['sequential_200[0]
[0]']
                                512)


 sequential_202 (Sequential)    (None, None, None,   4195328     ['sequential_201[0]
[0]']
                                512)


 sequential_203 (Sequential)    (None, None, None,   4195328     ['sequential_202[0]
[0]']
                                512)


 concatenate_11 (Concatenate)   multiple             0           ['sequential_203[0]
[0]',
                                                                  'sequential_201[0]
[0]',
                                                                  'sequential_204[0]
[0]',
                                                                  'sequential_200[0]
[0]',
                                                                  'sequential_205[0]
[0]',
                                                                  'sequential_199[0]
[0]',
                                                                  'sequential_206[0]
[0]',
                                                                  'sequential_198[0]
[0]',
                                                                  'sequential_207[0]
[0]',
                                                                  'sequential_197[0]
[0]',
                                                                  'sequential_208[0]
[0]',
                                                                  'sequential_196[0]
[0]',
                                                                  'sequential_209[0]
[0]',
                                                                  'sequential_195[0]
[0]']


 sequential_204 (Sequential)    (None, None, None,   8389632     ['concatenate_11[0]
[0]']
                                512)


 sequential_205 (Sequential)    (None, None, None,   8389632     ['concatenate_11[1]
[0]']
                                512)
```

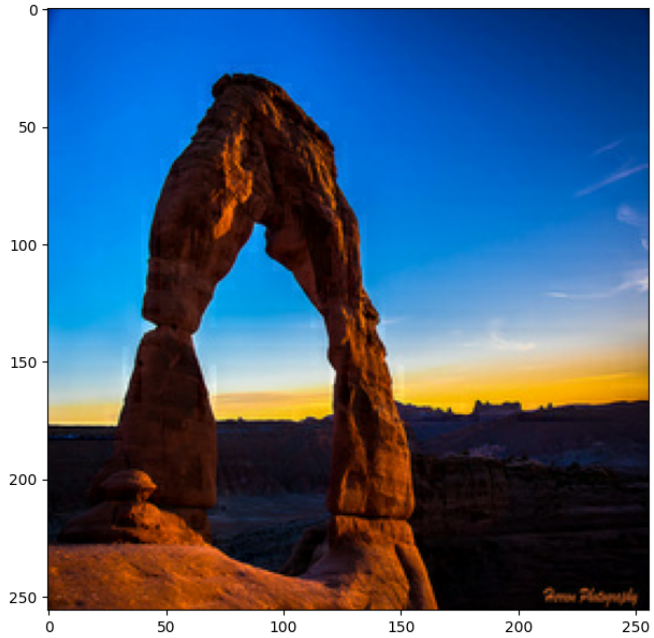
```
 sequential_206 (Sequential)    (None, None, None,   8389632     ['concatenate_11[2]
[0]']
                                512)

 sequential_207 (Sequential)    (None, None, None,   4194816     ['concatenate_11[3]
[0]']
                                256)

 sequential_208 (Sequential)    (None, None, None,   1048832     ['concatenate_11[4]
[0]']
                                128)

 sequential_209 (Sequential)    (None, None, None,   262272      ['concatenate_11[5]
[0]']
                                64)

 conv2d_transpose_95 (Conv2DTra  (None, None, None,   6147       ['concatenate_11[6]
[0]']
 nspose)                        3)


==================================================================================================
Total params: 54,414,979
Trainable params: 54,414,979
Non-trainable params: 0
__________________________________________________________________________________________________
```

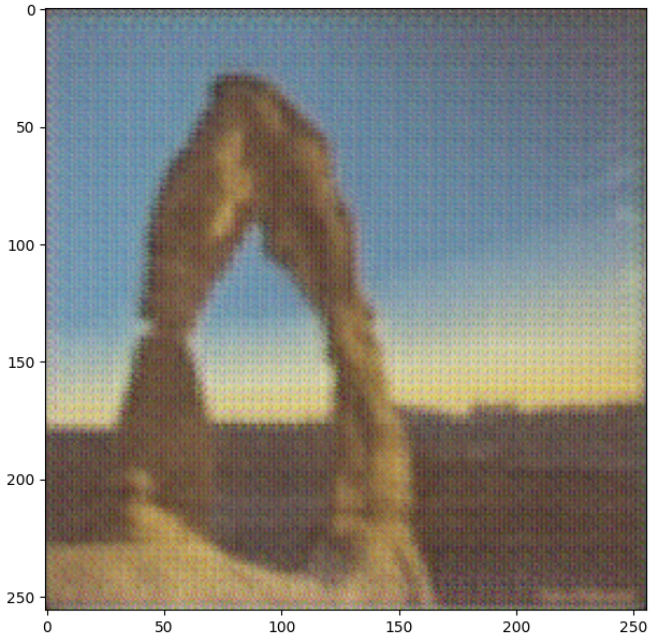
Training and checking.

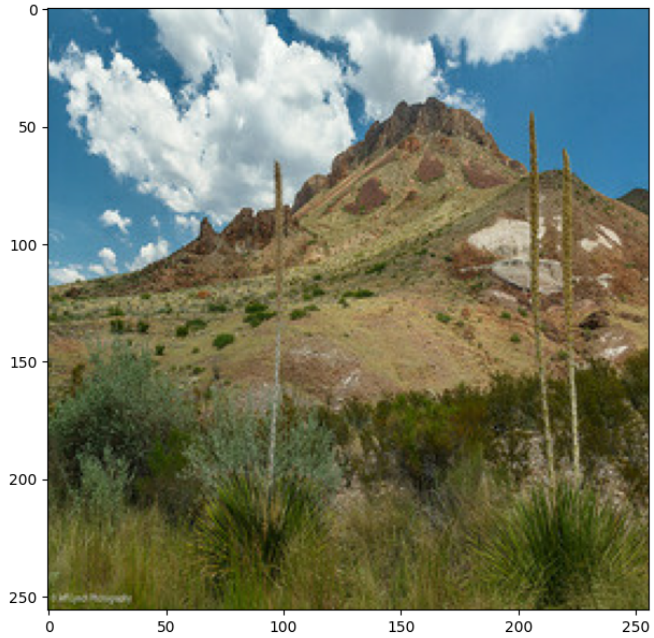
Although the model will calculate the loss and accuracy accordingly, to verify how the actual result looks like. After the model is trained, it is used to generate the Monet painting from the regular photos (testing dataset). Then 3 images are randomly selected to check the results. Based on the results, the model may be rerun with different epochs. According to multiple tries, one of the best result is when the epoch is 20.

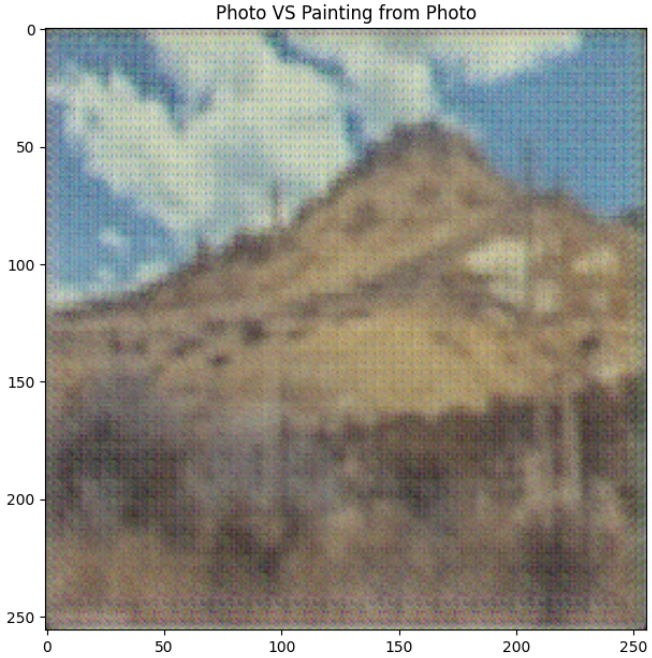
In [85]:
```
m.fit()
```

```
First time, start training.
Testing dataset
Training dataset
........Check result for epoch 1.
```

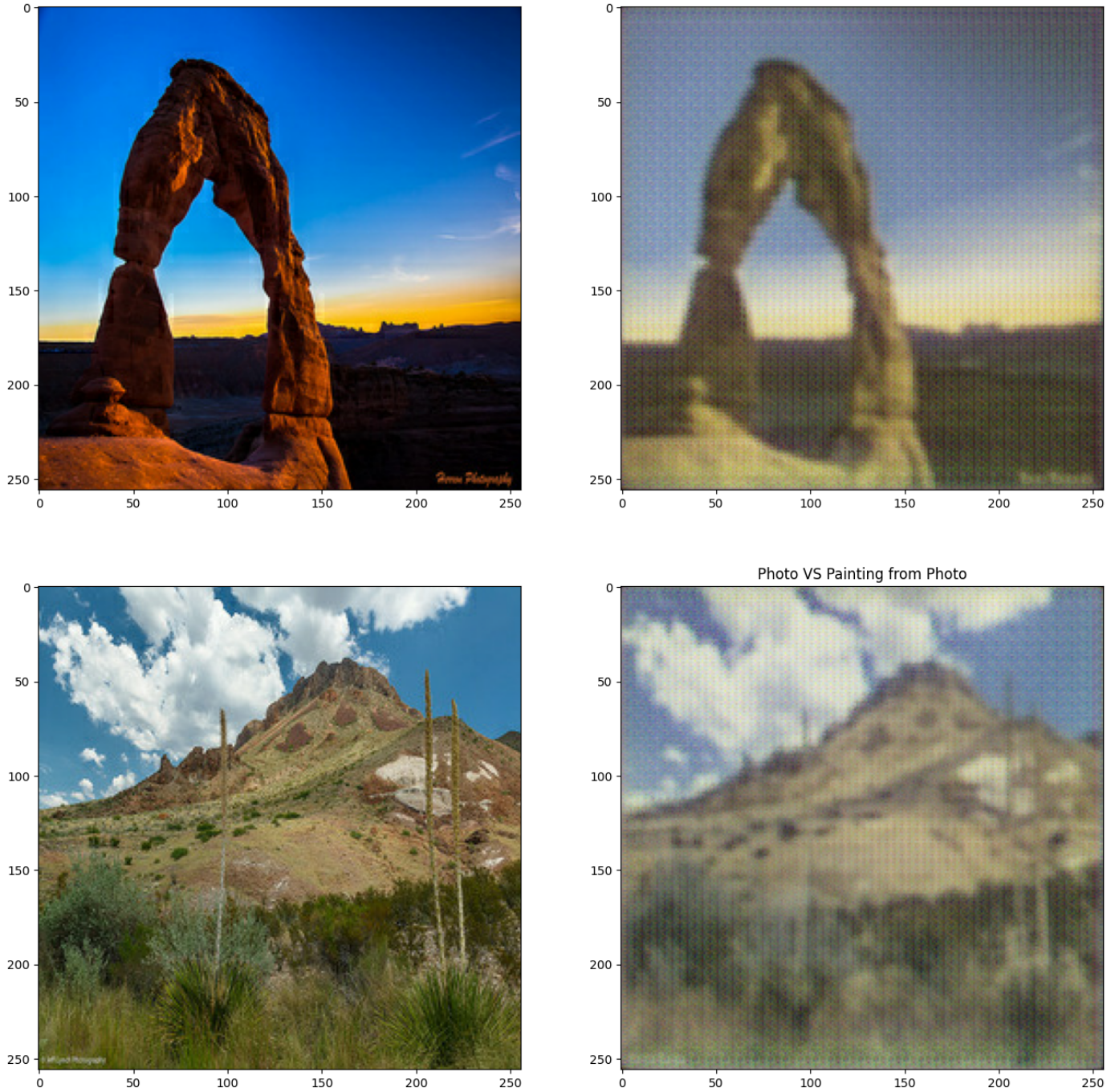






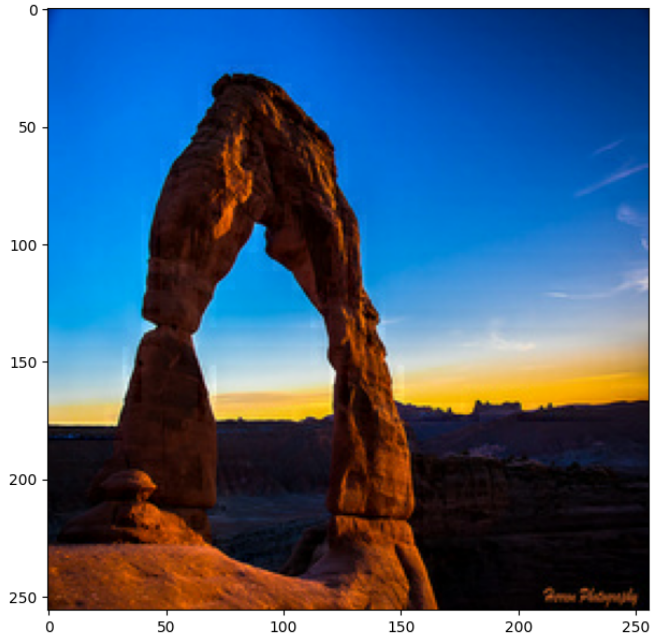
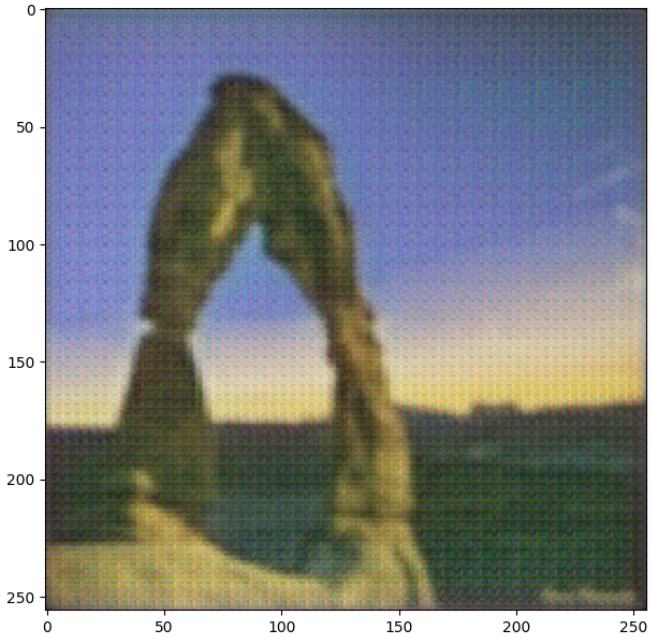
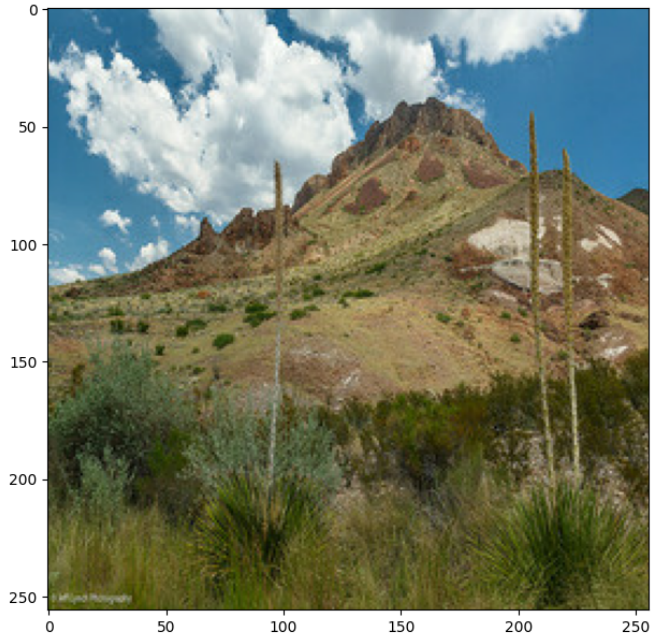
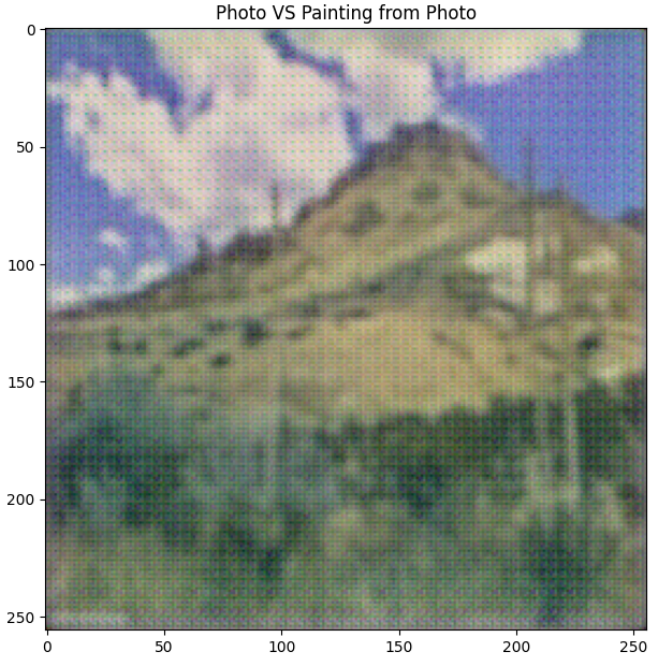



```
Time taken for epoch 1 is 949.783651958 sec

.........Check result for epoch 2.
```



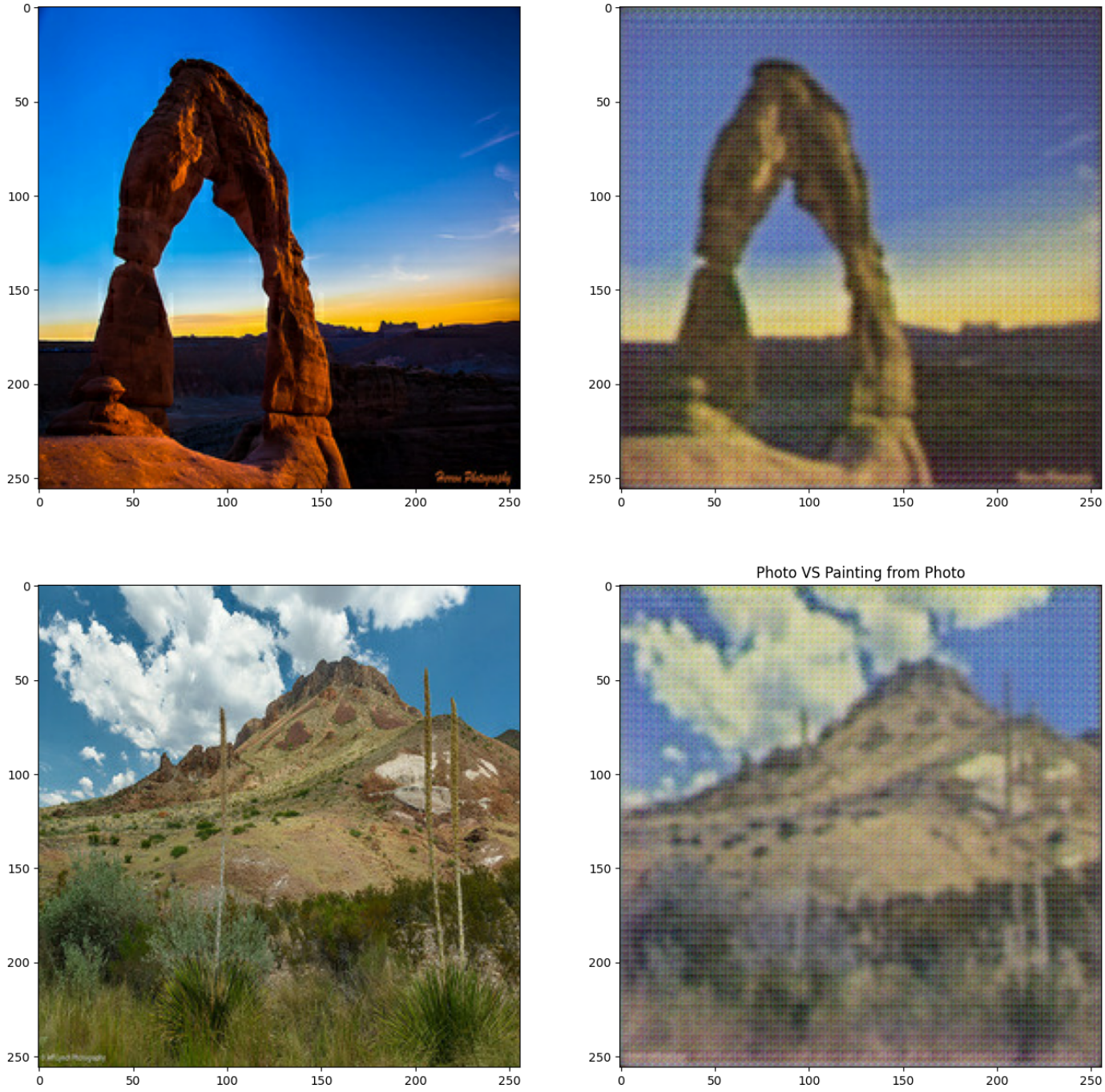

```
Time taken for epoch 2 is 942.1822972079999 sec

........Check result for epoch 3.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

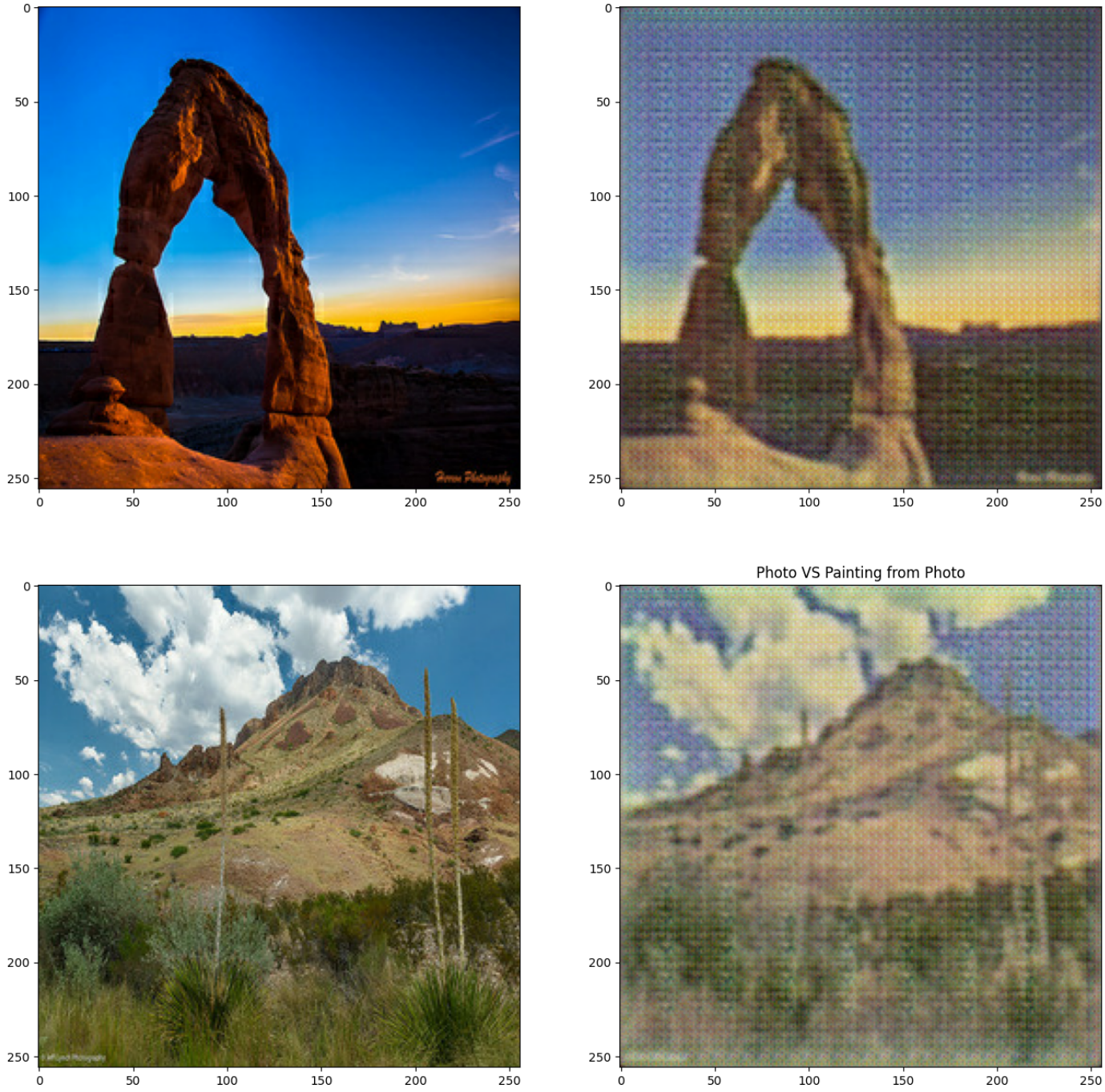

```
Time taken for epoch 3 is 1096.1143475000003 sec

........Check result for epoch 4.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



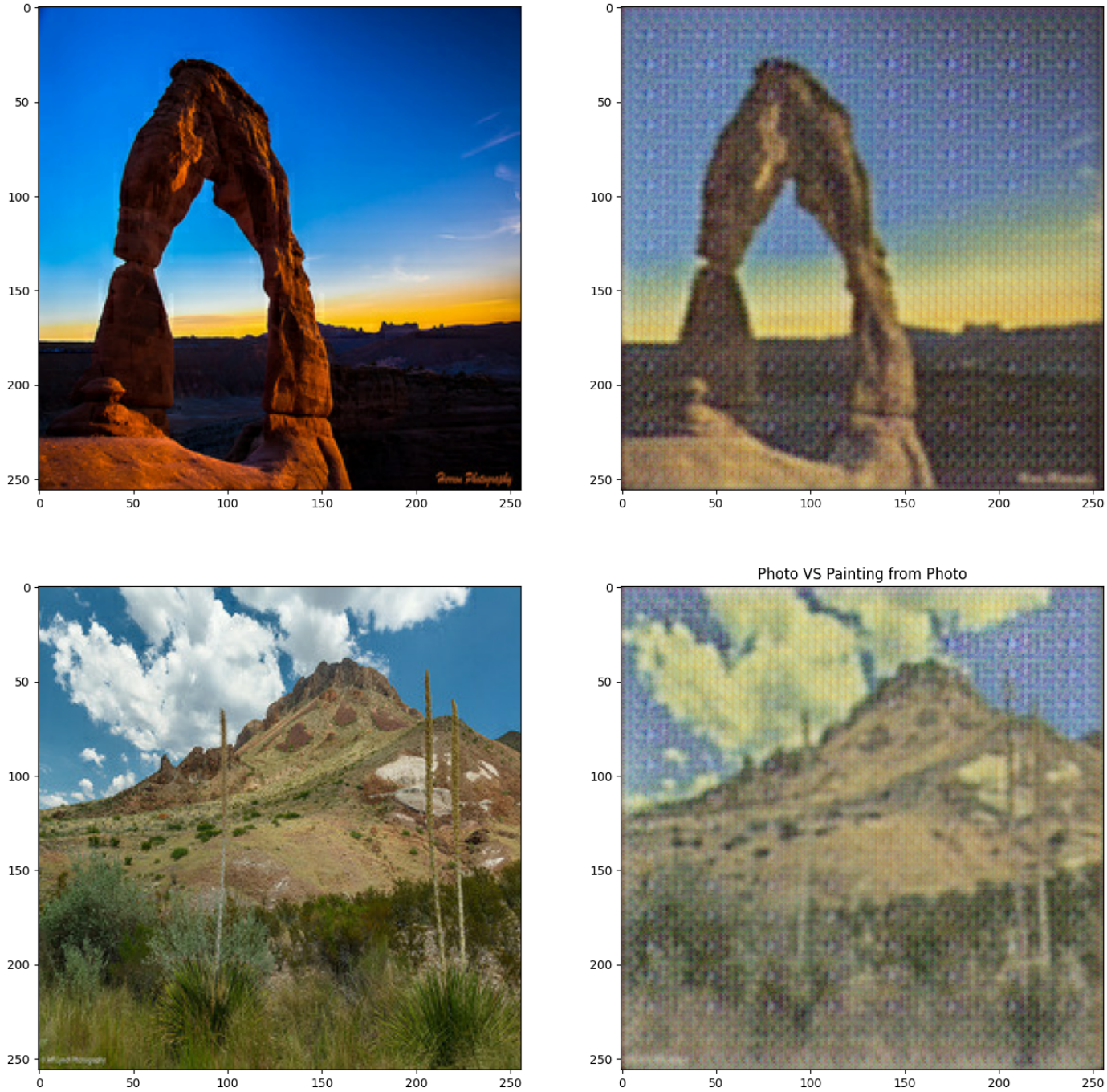

```
Time taken for epoch 4 is 996.1794810839997 sec

.........Check result for epoch 5.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



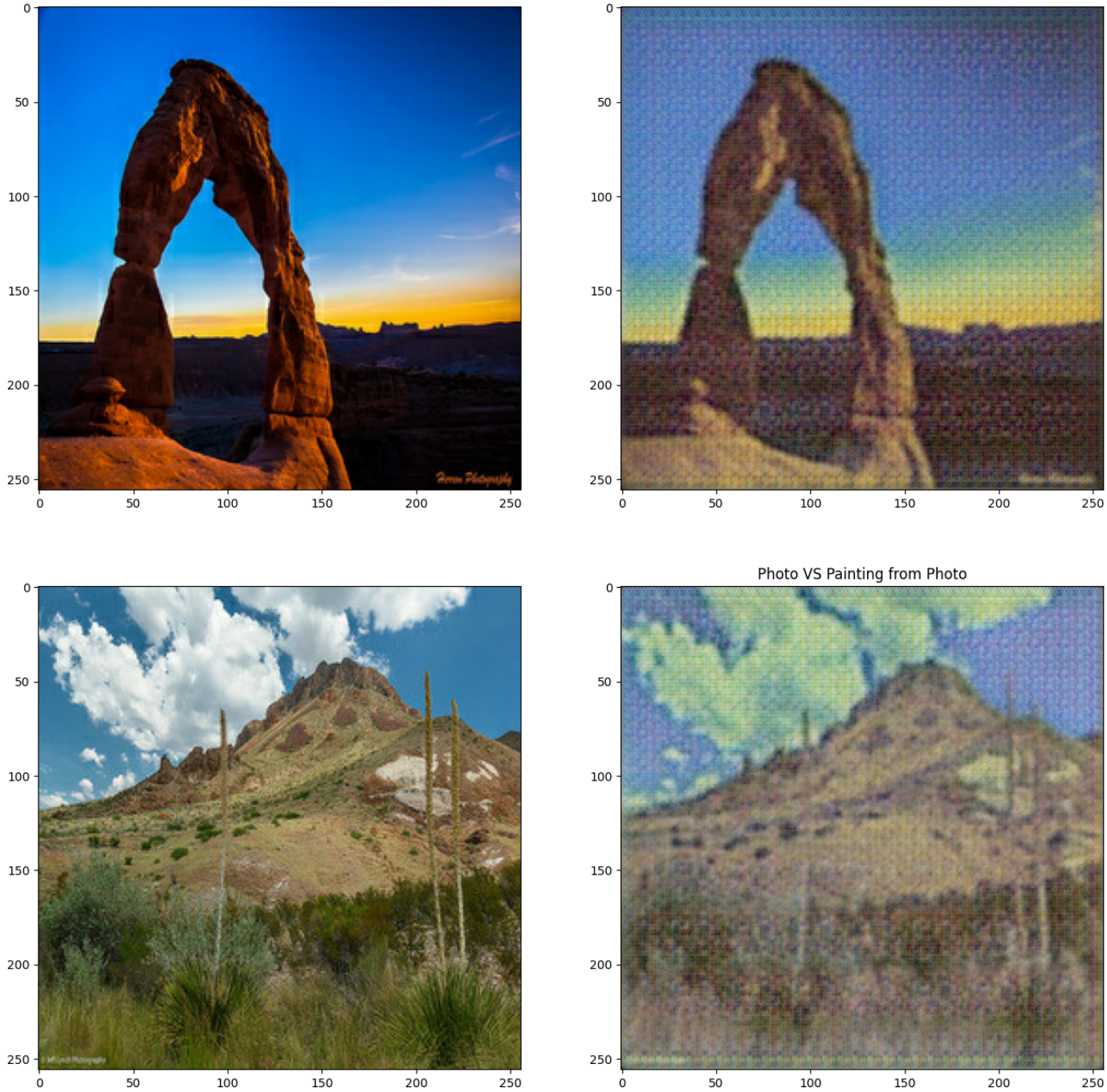

```
Saving checkpoint for epoch 5 at ./gan-getting-started/checkpoints/ckpt-1
Time taken for epoch 5 is 988.0513936670004 sec

.........Check result for epoch 6.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



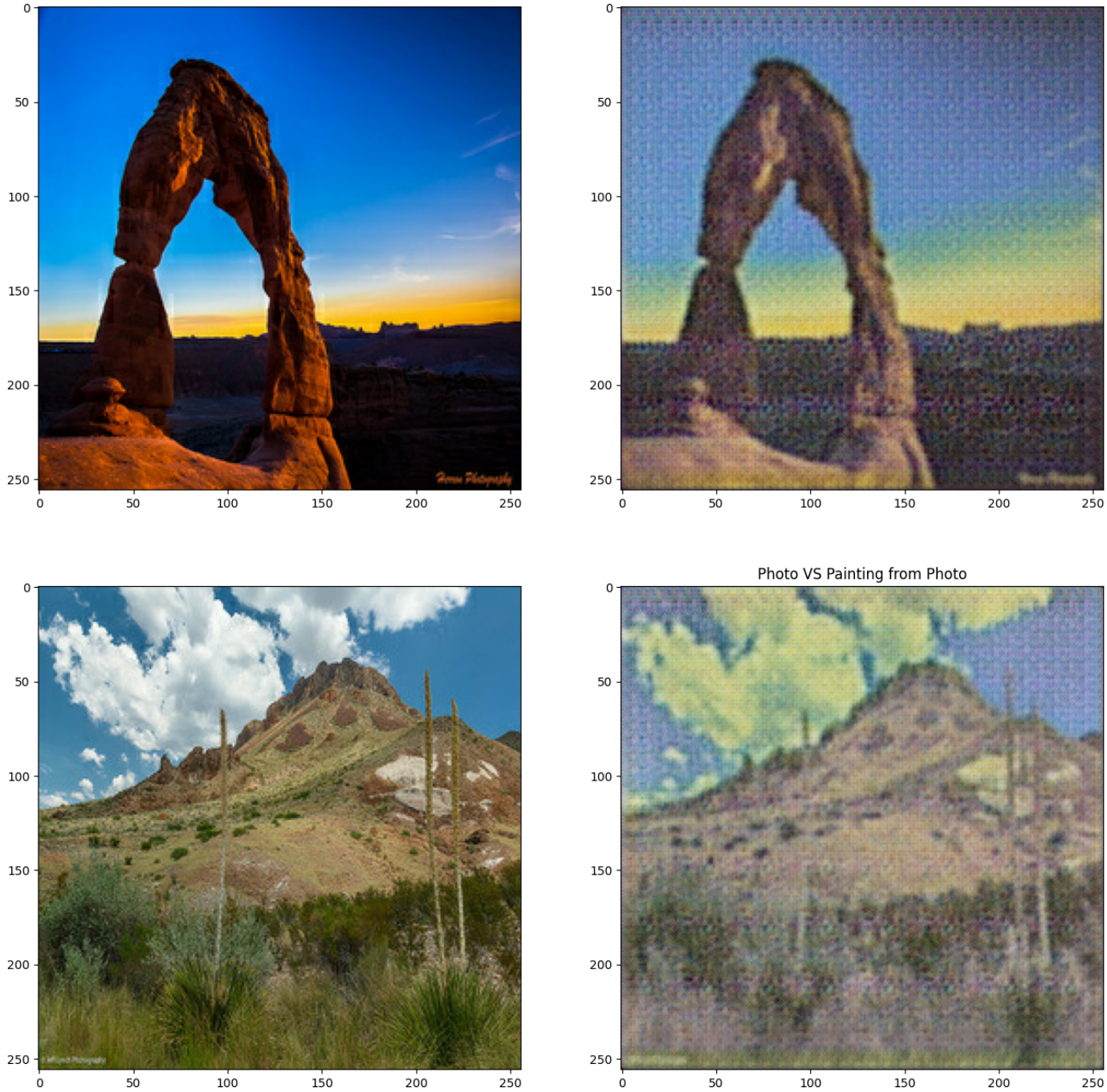

```
Time taken for epoch 6 is 919.4247882909995 sec

........Check result for epoch 7.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



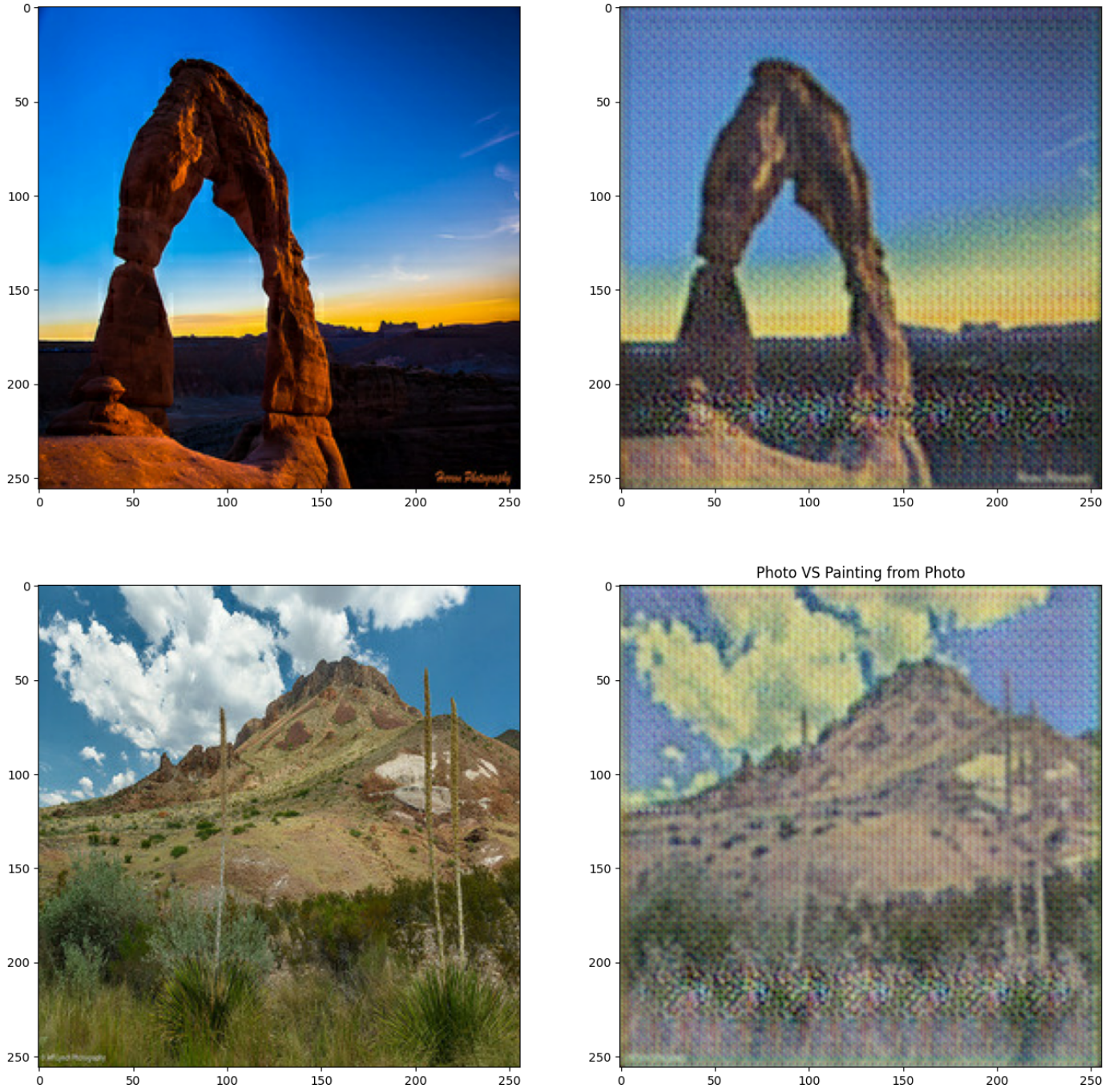

```
Time taken for epoch 7 is 1116.2013441669997 sec

........Check result for epoch 8.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



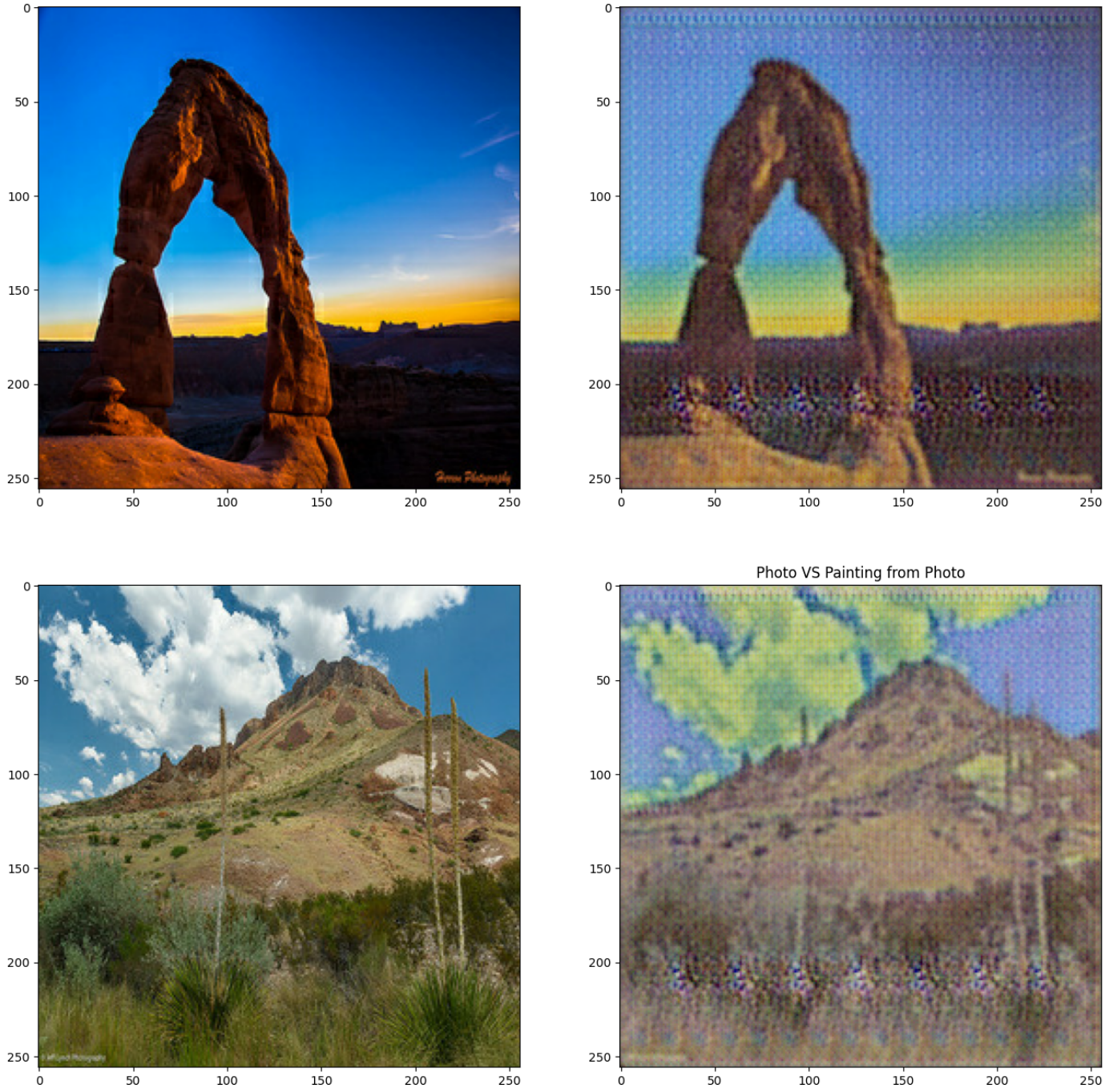

```
Time taken for epoch 8 is 1041.1298593749998 sec

........Check result for epoch 9.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



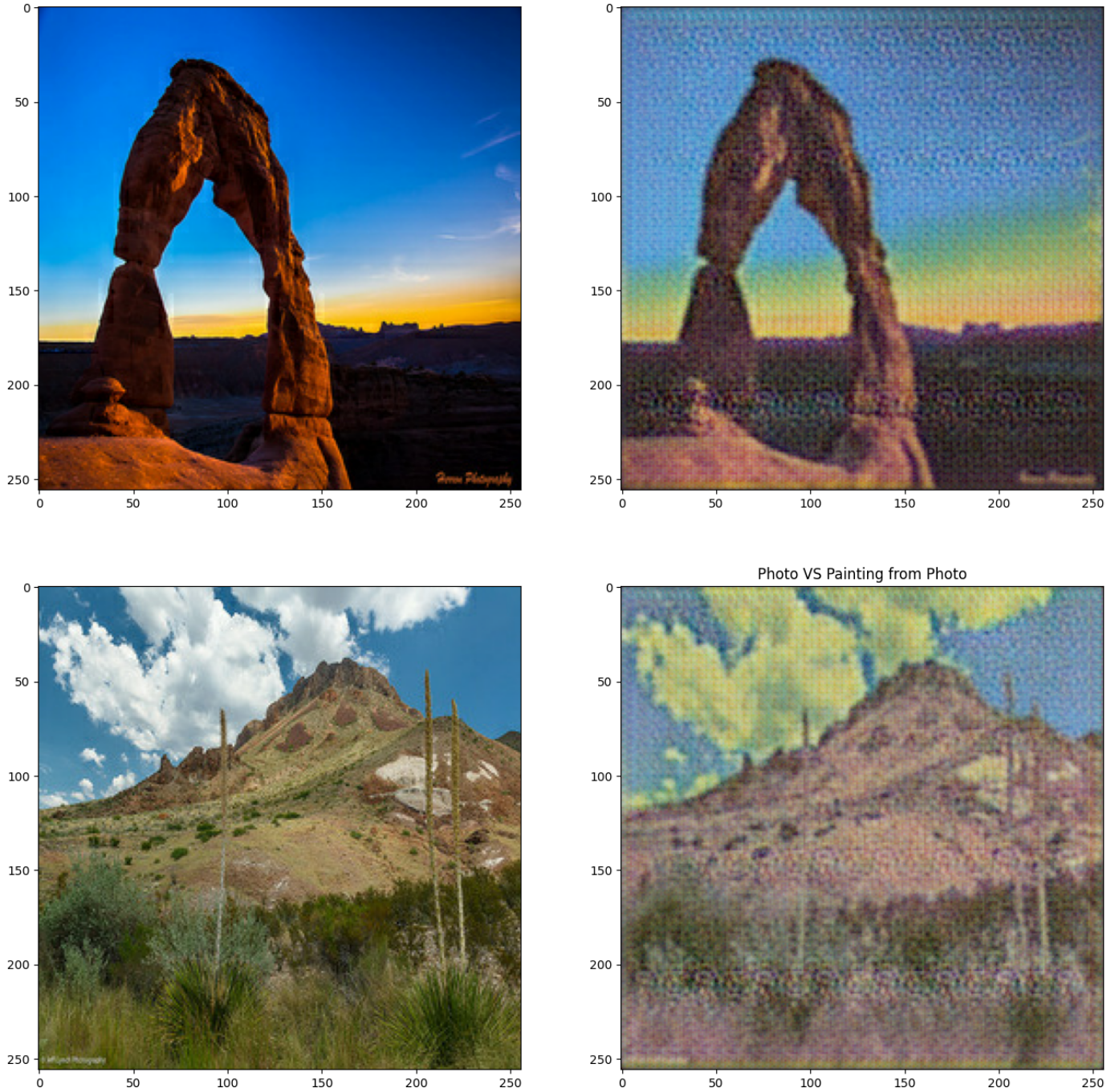

```
Time taken for epoch 9 is 974.8495983749999 sec

.........Check result for epoch 10.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

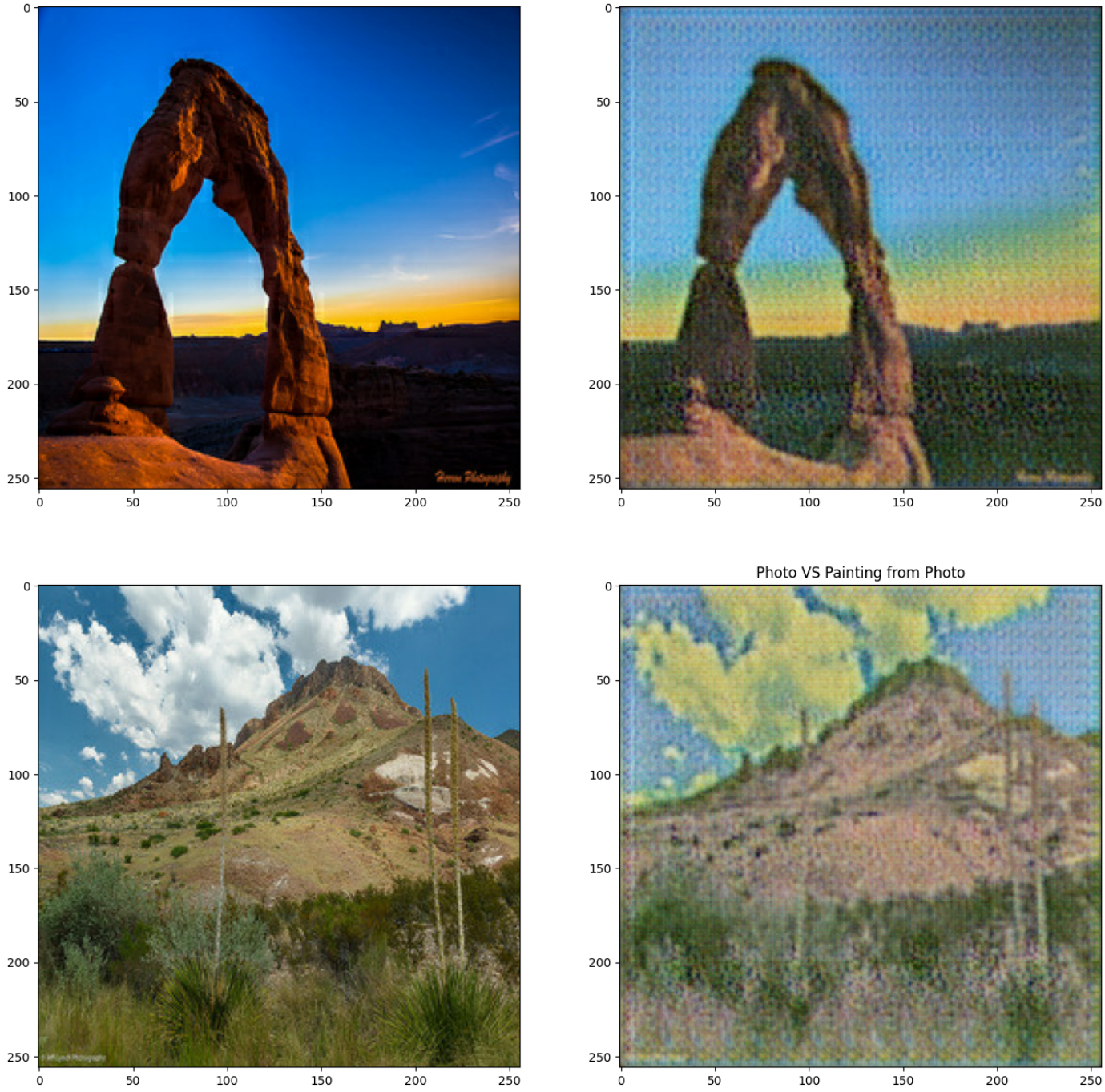



```
Saving checkpoint for epoch 10 at ./gan-getting-started/checkpoints/ckpt-2
Time taken for epoch 10 is 974.1098404159984 sec

.........Check result for epoch 11.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

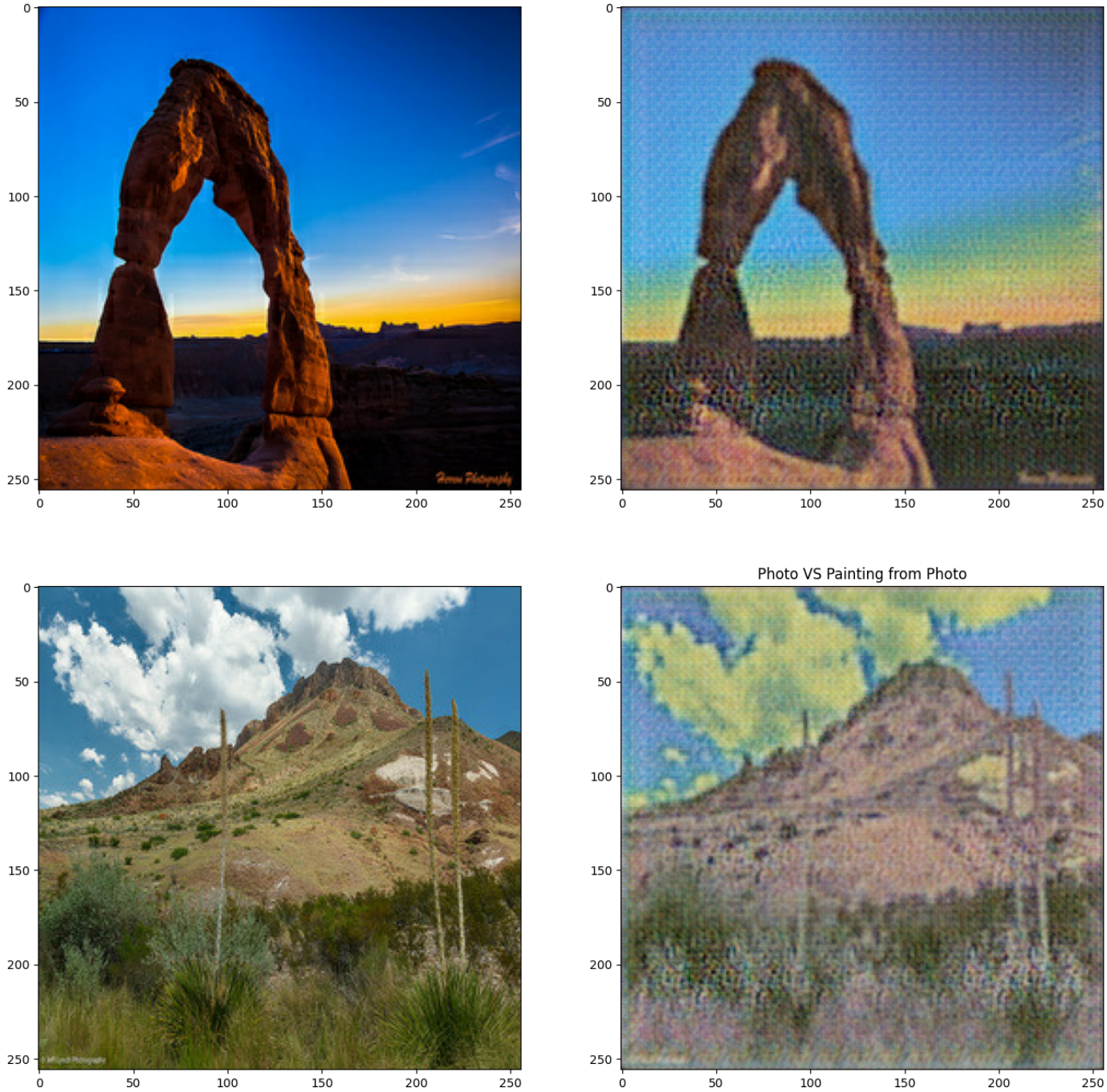



```
Time taken for epoch 11 is 1062.2120259160001 sec

........Check result for epoch 12.
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



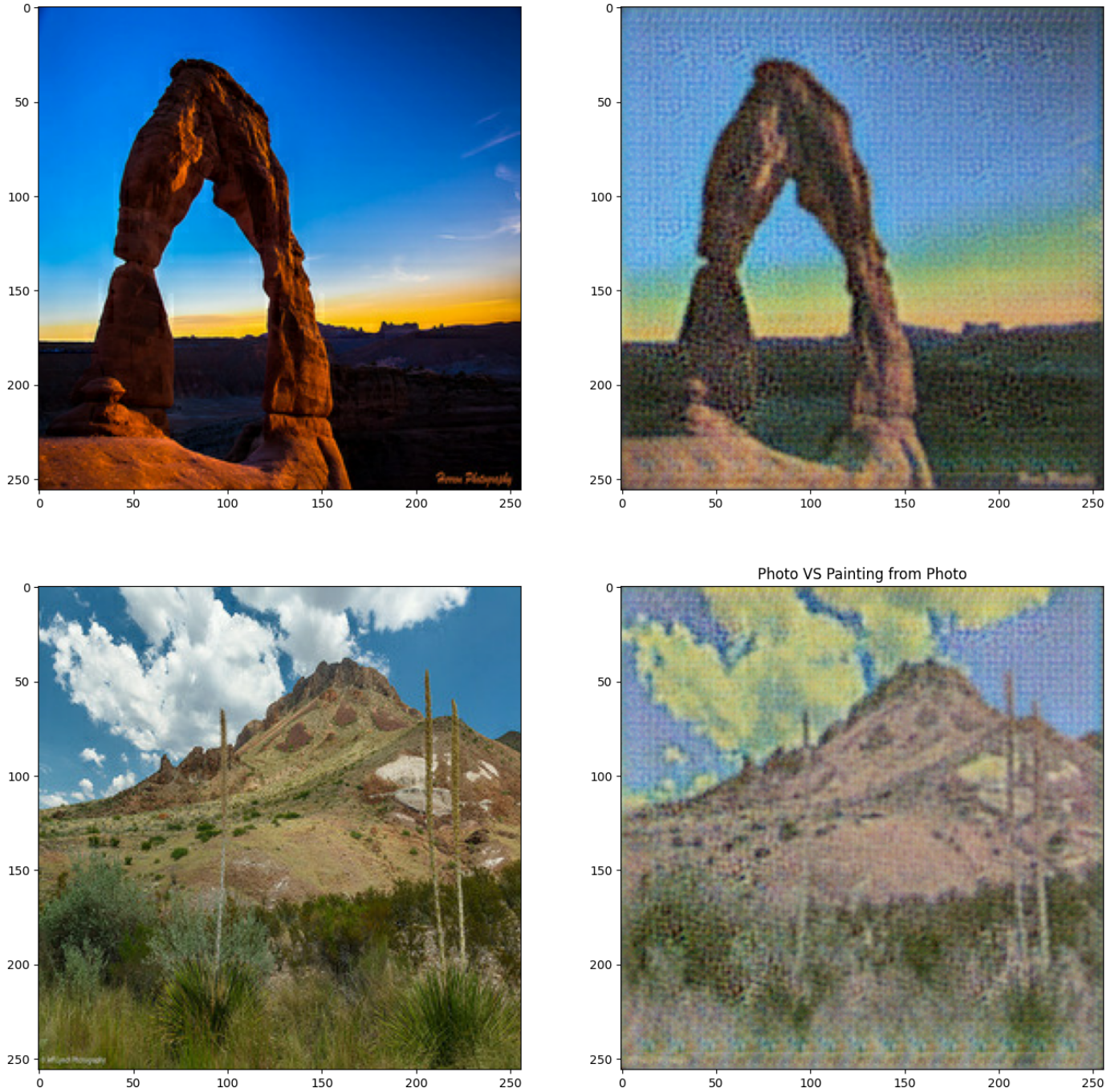

```
Time taken for epoch 12 is 1015.8588786250002 sec

........Check result for epoch 13.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

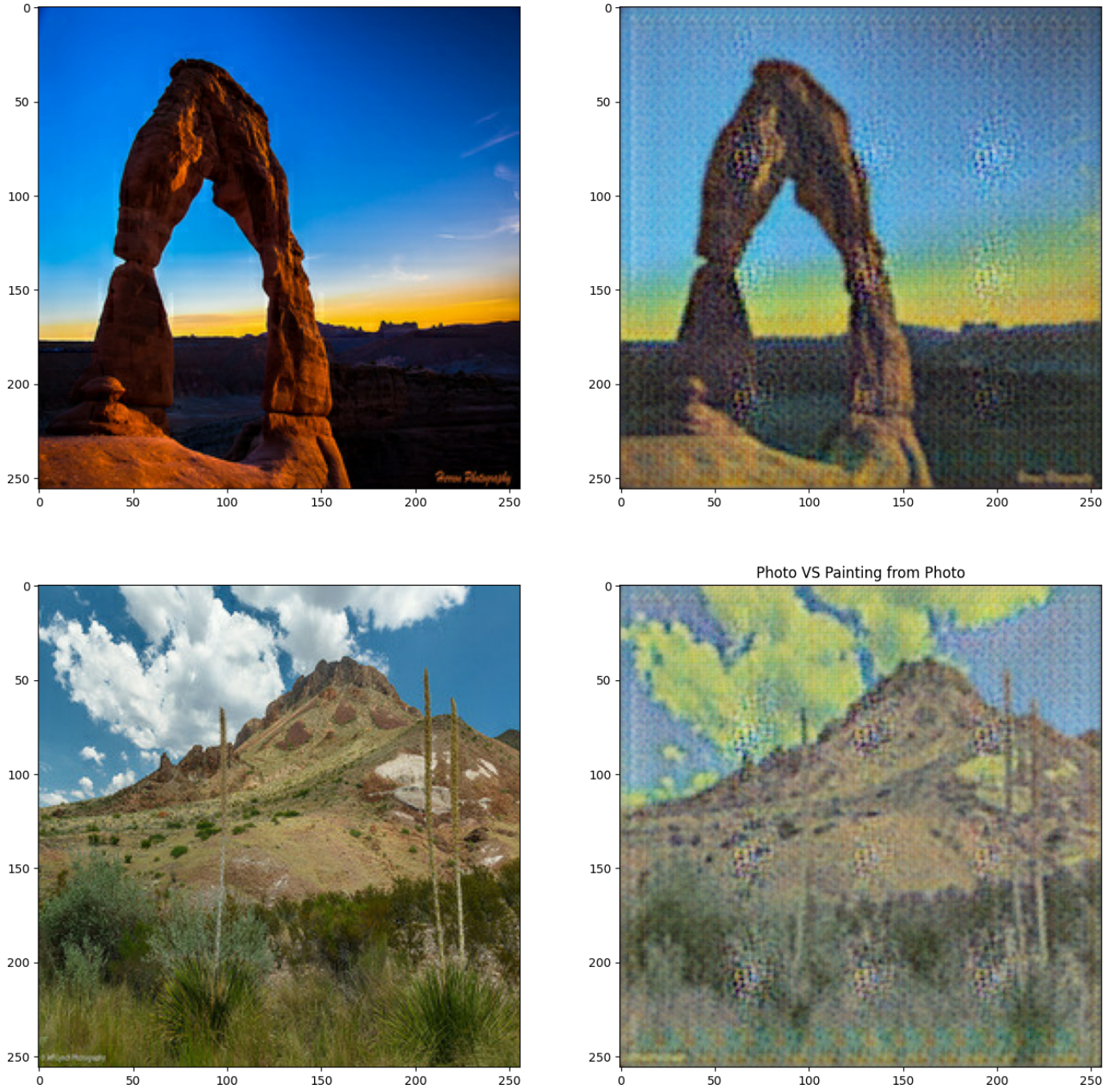



```
Time taken for epoch 13 is 1039.2908362079997 sec

........Check result for epoch 14.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

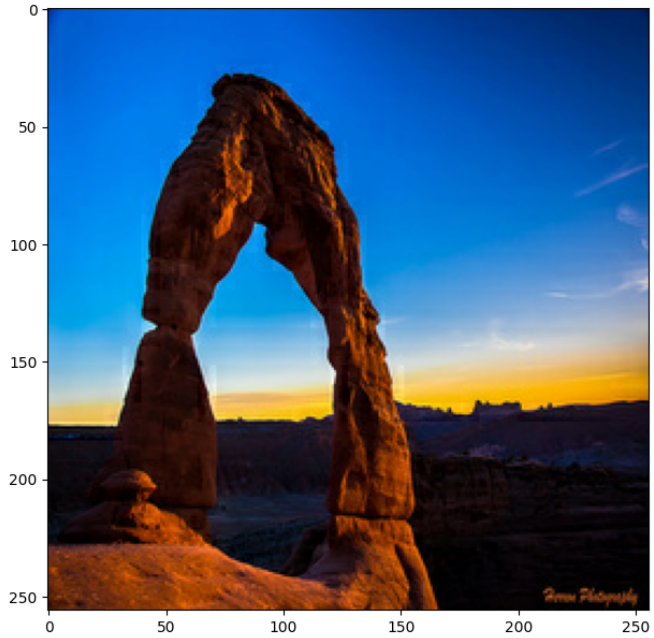
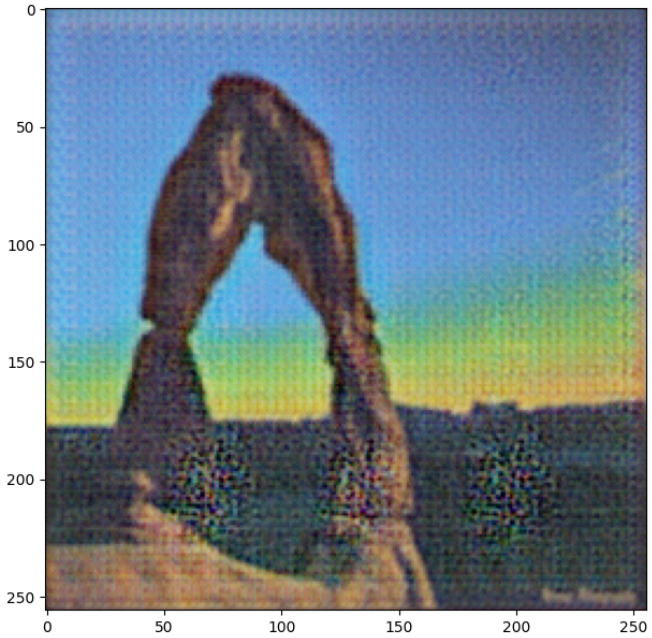
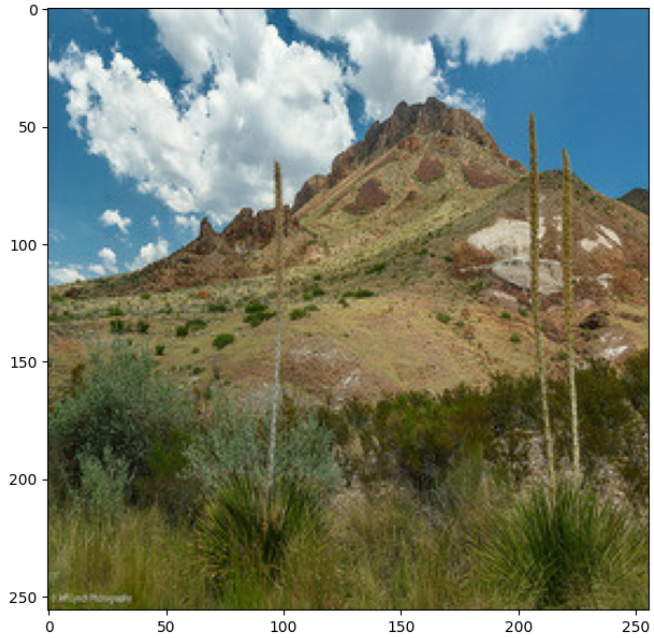
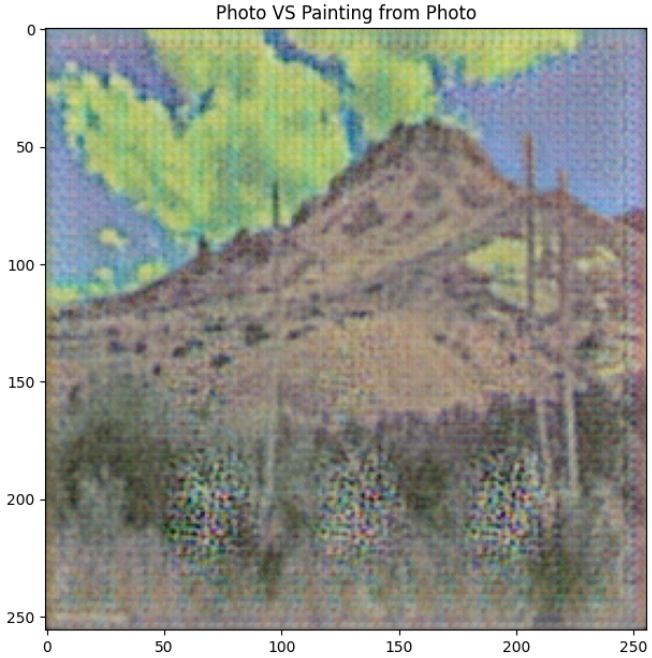



```
Time taken for epoch 14 is 999.1520964579995 sec

........Check result for epoch 15.
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```




```
Saving checkpoint for epoch 15 at ./gan-getting-started/checkpoints/ckpt-3
Time taken for epoch 15 is 1035.4343424170002 sec

.........Check result for epoch 16.
```

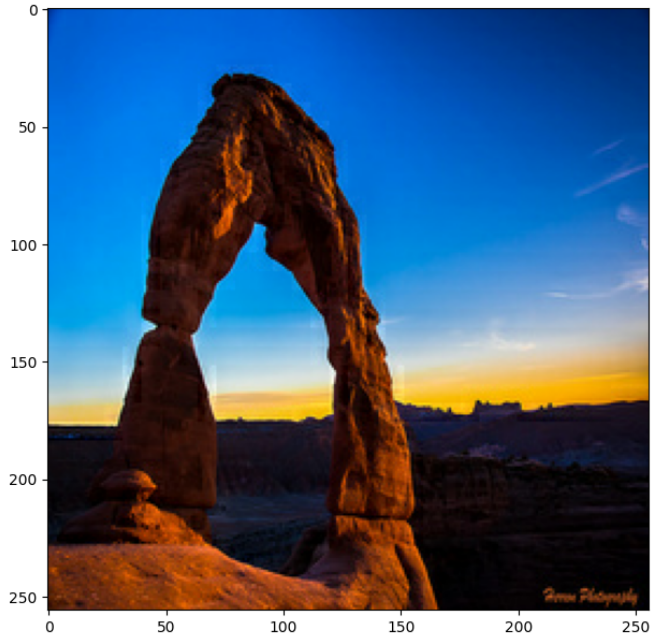
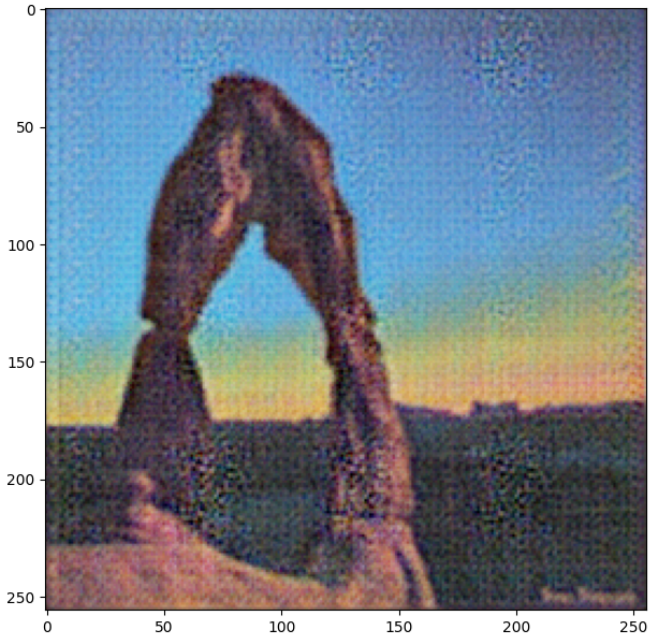
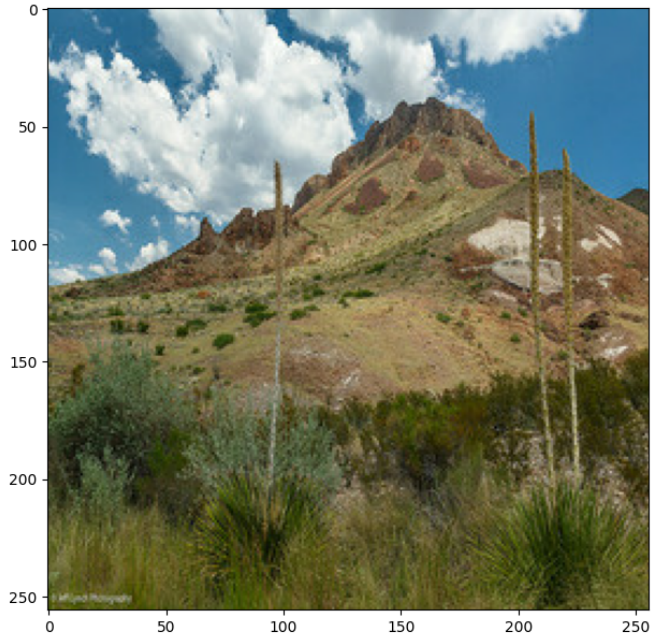
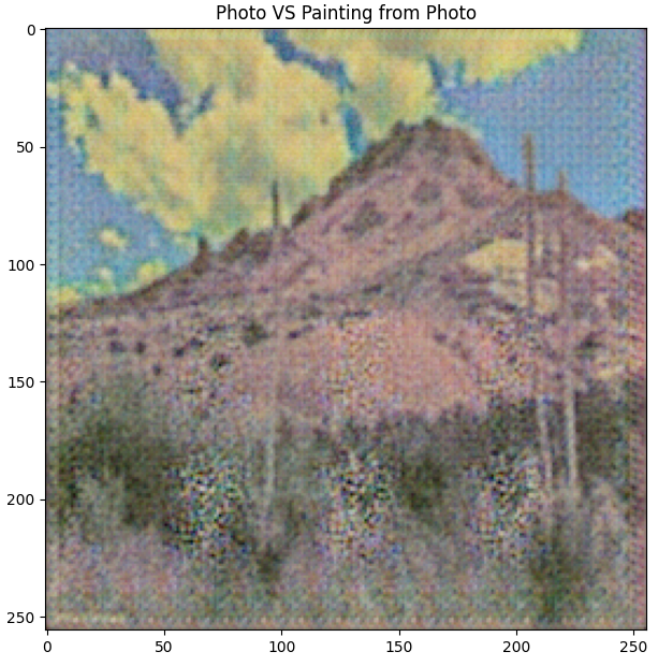
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```


```
Time taken for epoch 16 is 1098.8856094170023 sec

........Check result for epoch 17.
```

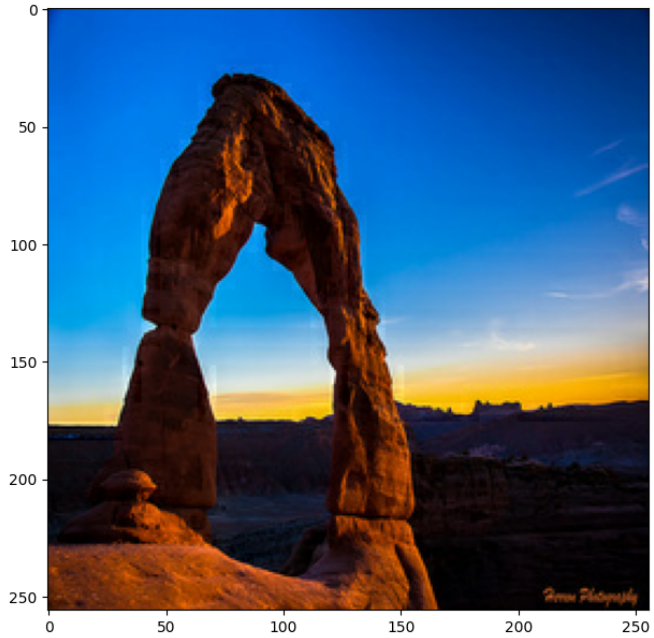
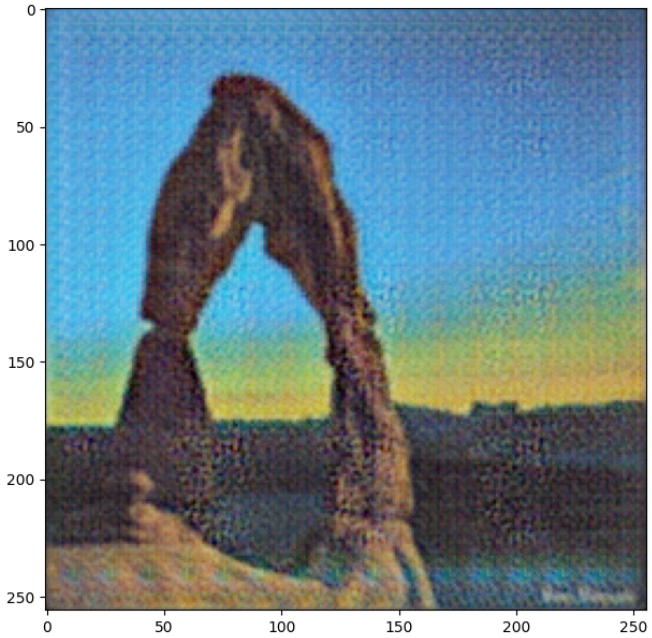
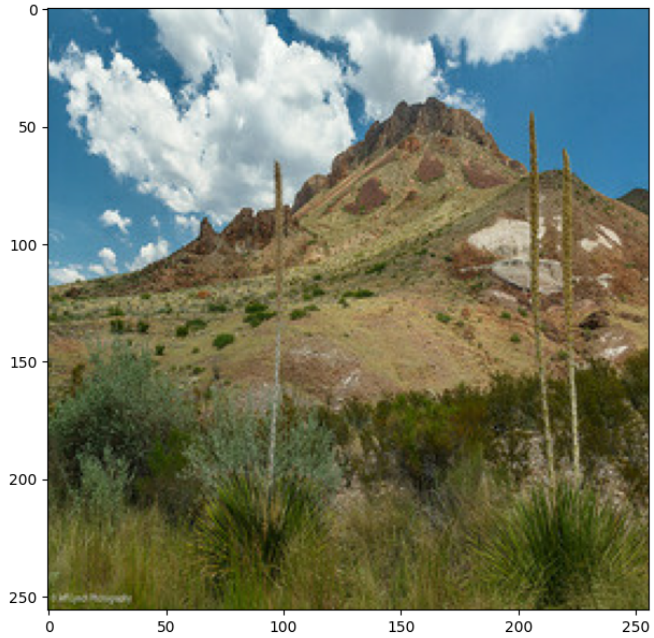
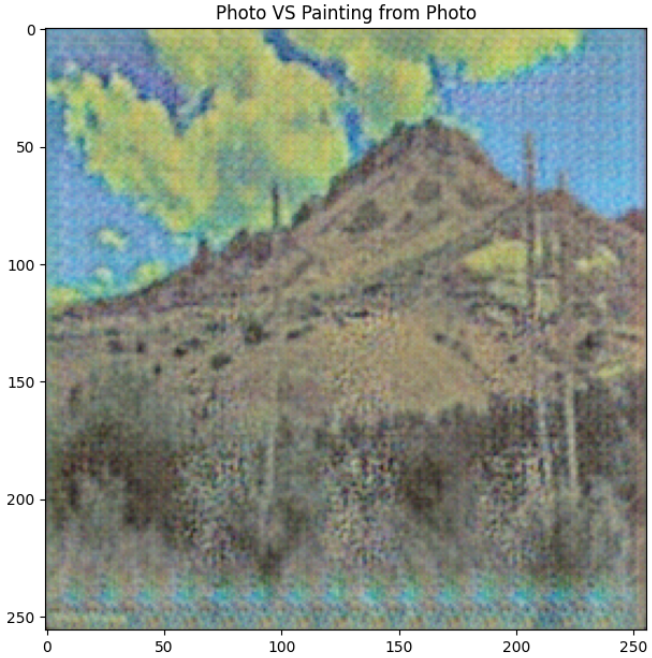
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

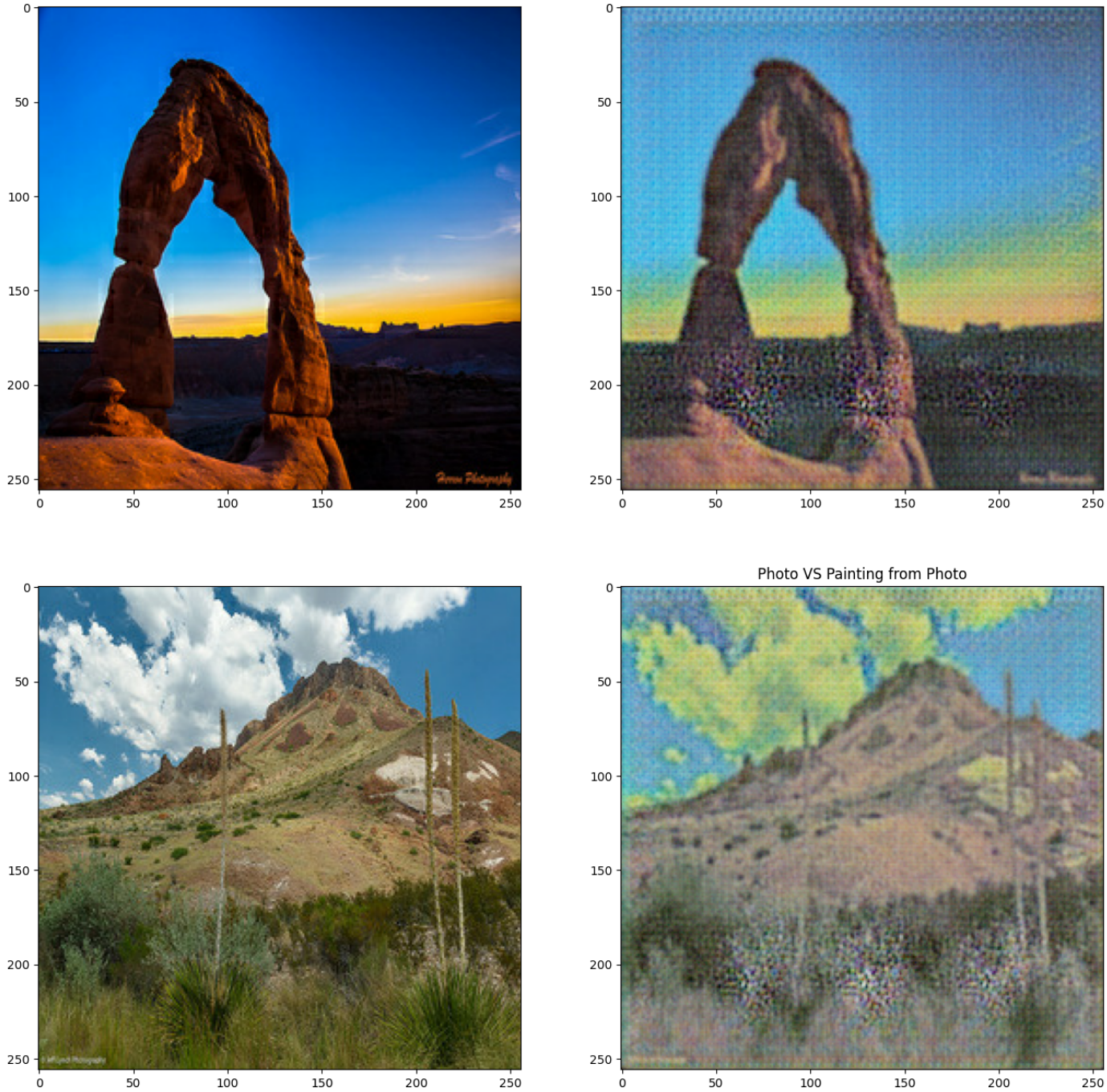






```
Time taken for epoch 17 is 1077.8233929590024 sec

........Check result for epoch 18.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

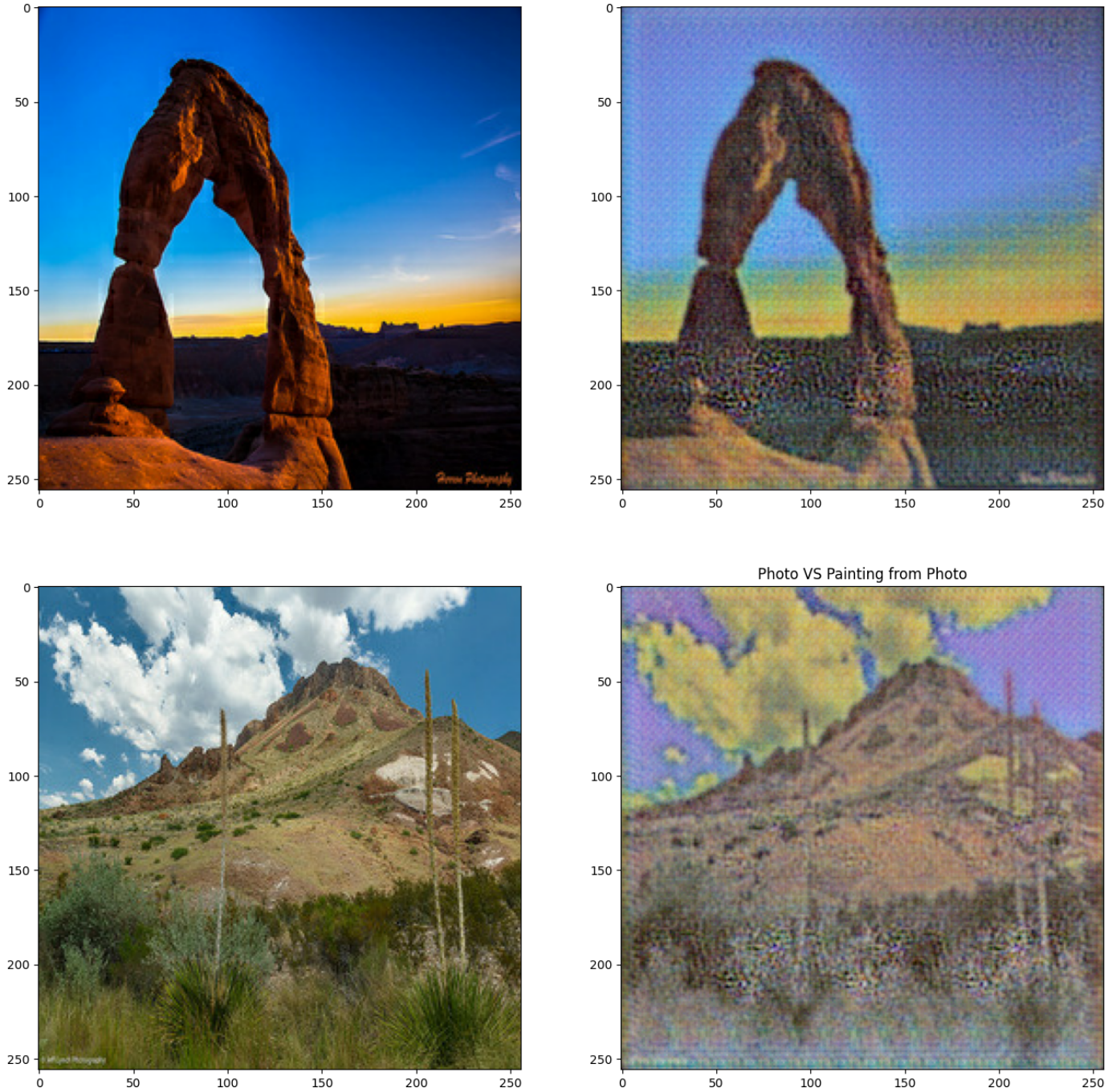






```
Time taken for epoch 18 is 1076.4342113750026 sec

........Check result for epoch 19.
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```




```
Time taken for epoch 19 is 1160.9295112920008 sec

........Check result for epoch 20.
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```




```
Saving checkpoint for epoch 20 at ./gan-getting-started/checkpoints/ckpt-4
Time taken for epoch 20 is 1089.0154120000007 sec

Time taken for 20 epochs is 20653.118702125 sec
```

Check the models once more after the training.

In [86]:
```
m.summary()
```

```
Model: "model_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (InputLayer)       [(None, None, None,  0           []
                              3)]
```

```
 sequential_180 (Sequential)    (None, None, None,   3072        ['input_11[0][0]']
                                64)

 sequential_181 (Sequential)    (None, None, None,   131328      ['sequential_180[0]
[0]']
                                128)

 sequential_182 (Sequential)    (None, None, None,   524800      ['sequential_181[0]
[0]']
                                256)

 sequential_183 (Sequential)    (None, None, None,   2098176     ['sequential_182[0]
[0]']
                                512)

 sequential_184 (Sequential)    (None, None, None,   4195328     ['sequential_183[0]
[0]']
                                512)

 sequential_185 (Sequential)    (None, None, None,   4195328     ['sequential_184[0]
[0]']
                                512)

 sequential_186 (Sequential)    (None, None, None,   4195328     ['sequential_185[0]
[0]']
                                512)

 sequential_187 (Sequential)    (None, None, None,   4195328     ['sequential_186[0]
[0]']
                                512)

 sequential_188 (Sequential)    (None, None, None,   4195328     ['sequential_187[0]
[0]']
                                512)

 concatenate_10 (Concatenate)   multiple             0           ['sequential_188[0]
[0]',
                                                                 'sequential_186[0]
[0]',
                                                                 'sequential_189[0]
[0]',
                                                                 'sequential_185[0]
[0]',
                                                                 'sequential_190[0]
[0]',
                                                                 'sequential_184[0]
```

```
[0]',
                                                                 'sequential_191[0]
[0]',
                                                                 'sequential_183[0]
[0]',
                                                                 'sequential_192[0]
[0]',
                                                                 'sequential_182[0]
[0]',
                                                                 'sequential_193[0]
[0]',
                                                                 'sequential_181[0]
[0]',
                                                                 'sequential_194[0]
[0]',
                                                                 'sequential_180[0]
[0]']

 sequential_189 (Sequential)    (None, None, None,   8389632     ['concatenate_10[0]
[0]']
                                512)

 sequential_190 (Sequential)    (None, None, None,   8389632     ['concatenate_10[1]
[0]']
                                512)

 sequential_191 (Sequential)    (None, None, None,   8389632     ['concatenate_10[2]
[0]']
                                512)

 sequential_192 (Sequential)    (None, None, None,   4194816     ['concatenate_10[3]
[0]']
                                256)

 sequential_193 (Sequential)    (None, None, None,   1048832     ['concatenate_10[4]
[0]']
                                128)

 sequential_194 (Sequential)    (None, None, None,   262272      ['concatenate_10[5]
[0]']
                                64)

 conv2d_transpose_87 (Conv2DTra  (None, None, None,   6147       ['concatenate_10[6]
[0]']
 nspose)                        3)


==================================================================================
==========
Total params: 54,414,979
Trainable params: 54,414,979
Non-trainable params: 0
```

```
__________
Model: "model_21"
__________
 Layer (type)                Output Shape              Param #     Connected to

==========
 input_12 (InputLayer)       [(None, None, None,  0           []

                              3)]


 sequential_195 (Sequential)  (None, None, None,   3072        ['input_12[0][0]']

                              64)


 sequential_196 (Sequential)  (None, None, None,   131328      ['sequential_195[0]
[0]']
                              128)


 sequential_197 (Sequential)  (None, None, None,   524800      ['sequential_196[0]
[0]']
                              256)


 sequential_198 (Sequential)  (None, None, None,   2098176     ['sequential_197[0]
[0]']
                              512)


 sequential_199 (Sequential)  (None, None, None,   4195328     ['sequential_198[0]
[0]']
                              512)


 sequential_200 (Sequential)  (None, None, None,   4195328     ['sequential_199[0]
[0]']
                              512)


 sequential_201 (Sequential)  (None, None, None,   4195328     ['sequential_200[0]
[0]']
                              512)


 sequential_202 (Sequential)  (None, None, None,   4195328     ['sequential_201[0]
[0]']
                              512)


 sequential_203 (Sequential)  (None, None, None,   4195328     ['sequential_202[0]
[0]']
                              512)
```

```
 concatenate_11 (Concatenate)   multiple             0           ['sequential_203[0]
[0]',
                                                                  'sequential_201[0]
[0]',
                                                                  'sequential_204[0]
[0]',
                                                                  'sequential_200[0]
[0]',
                                                                  'sequential_205[0]
[0]',
                                                                  'sequential_199[0]
[0]',
                                                                  'sequential_206[0]
[0]',
                                                                  'sequential_198[0]
[0]',
                                                                  'sequential_207[0]
[0]',
                                                                  'sequential_197[0]
[0]',
                                                                  'sequential_208[0]
[0]',
                                                                  'sequential_196[0]
[0]',
                                                                  'sequential_209[0]
[0]',
                                                                  'sequential_195[0]
[0]']

 sequential_204 (Sequential)    (None, None, None,   8389632     ['concatenate_11[0]
[0]']
                                512)

 sequential_205 (Sequential)    (None, None, None,   8389632     ['concatenate_11[1]
[0]']
                                512)

 sequential_206 (Sequential)    (None, None, None,   8389632     ['concatenate_11[2]
[0]']
                                512)

 sequential_207 (Sequential)    (None, None, None,   4194816     ['concatenate_11[3]
[0]']
                                256)

 sequential_208 (Sequential)    (None, None, None,   1048832     ['concatenate_11[4]
[0]']
                                128)

 sequential_209 (Sequential)    (None, None, None,   262272      ['concatenate_11[5]
[0]']
                                64)
```

```
 conv2d_transpose_95 (Conv2DTra  (None, None, None,   6147       ['concatenate_11[6]
[0]']
 nspose)                        3)


==================================================================================================
Total params: 54,414,979
Trainable params: 54,414,979
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [87]:

```python
class Predictor:
    def __init__(self, model:CycleGanModel):
        self.model = model

    def prepare(self):
        test_files = np.array(os.listdir(self.from_path))
        self.df = pd.DataFrame(test_files, columns=['id'])
        self.df = self.df.applymap(lambda x: os.path.splitext(x)[0])
        ImageLoader.build(self.df, self.from_path, self.to_path)

    def predict(self):
        if not os.path.exists(config.output_dir):
            os.makedirs(config.output_dir)

        num = 0
        test_ds = self.model.generator.test_ds.ds
        processed_test_ds = self.model.generator.test_ds.preprocessed_ds
        #ds_gen = self.model.generator.g_p2m_generator(processed_test_ds)

        #it_gen = iter(ds_gen)
        it_processed = iter(processed_test_ds)
        it_origin = iter(test_ds)
        try:
            while (True):
                photos = next(it_processed)
                fake_monet_paintings = self.model.generator.g_p2m_generator(photos)
                ids = next(it_origin)[1]
                img_num = len(photos)

                #print(photos.shape, fake_monet_paintings.shape, ids, img_num)
                for i in range(img_num):
                    #fake_monet = regulate_image(monet_generator(img, training=False))
                    #show_raw_image(None, fake_monet, None, None)

                    img = fake_monet_paintings[i]

                    img = DatasetBuilder.rollback_one(img.numpy()).astype("uint8")
                    id = ids[i]

                    #print("{} img.shape {}".format(img.shape, id))
                    im = Image.fromarray(np.array(img))

                    #print(img_id.numpy()[i].decode())
                    fullname = os.path.join(config.output_dir + id.numpy().decode() + ".
                    im.save(fullname)
                    num += 1

                if num >= Constants.MAX_PHOTOS_FOR_TESTING:
```

```
                break
        except Exception as e:
                print(f'caught {type(e)}: e')

        print("{} of images are converted to monet painting in the directory: {}".format
        self.verify()


    def verify(self):
        orig_photos = list(pathlib.Path(config.photo_jpg_dir).with_suffix('').glob('*.jp
        fake_monet_photos = list(pathlib.Path(config.output_dir).with_suffix('').glob('*
        assert((len(orig_photos) == len(fake_monet_photos)) or (len(fake_monet_photos)==

    def submit(self):
        import shutil
        shutil.make_archive(config.working_dir + "images", 'zip', config.output_dir)
```

Test

Generate Monet paintings with the trained model, store the generated images to the output directory.

In [88]:
```
predictor = Predictor(m)
```

In [89]:
```
predictor.predict()
```

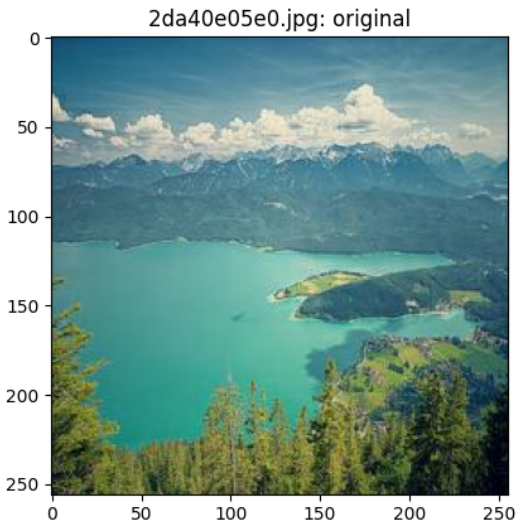
```
caught <class 'StopIteration'>: e
7038 of images are converted to monet painting in the directory: ./gan-getting-started/s
ubmission/
```

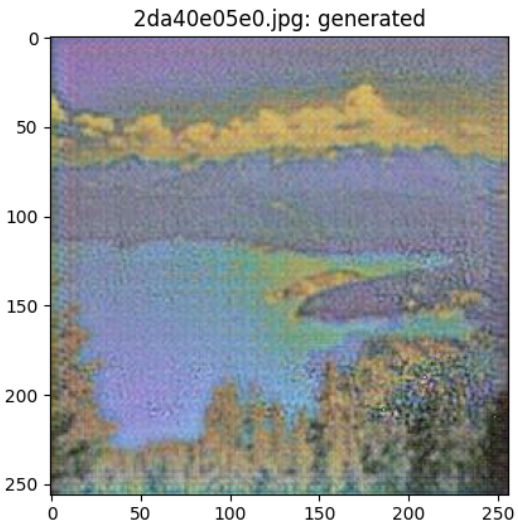
Visualize the result

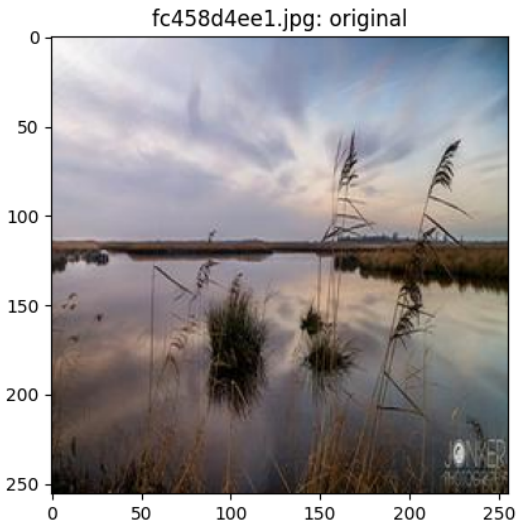
Randomly pick up 3 images to visualize the generated Monet paintings against the original photos.

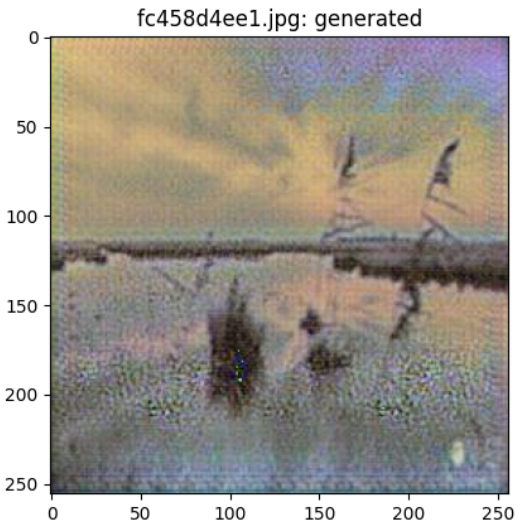
In [90]:
```
class PredictorDisplay:
    def __init__(self, owner):
        self.owner = owner


    def show_result(self):
        orig_photos = list(pathlib.Path(config.photo_jpg_dir).with_suffix('').glob('*.jp
        fake_monet_photos = list(pathlib.Path(config.output_dir).with_suffix('').glob('*
        import random as rn
        indice = rn.sample(range(0, len(fake_monet_photos)), 3)
        fig_width = 2
        fig_height = len(indice)
        fig = plt.figure(figsize=(16,16))
        ax = fig.subplots(fig_height,fig_width)

        for i in range(len(indice)):
            fake_monet_fullname = str(fake_monet_photos[indice[i]])
            fn = fake_monet_fullname.split(os.sep)[-1]
            orig_jpg_fullname = config.photo_jpg_dir + fn
            #print(orig_jpg_fullname, fake_monet_fullname)

            with Image.open(orig_jpg_fullname) as img:
                ax[i][0].imshow((img))
                ax[i][0].set_title(fn + ": original")
            with Image.open(fake_monet_fullname) as img:
                ax[i][1].imshow((img))
                ax[i][1].set_title(fn + ": generated")
```

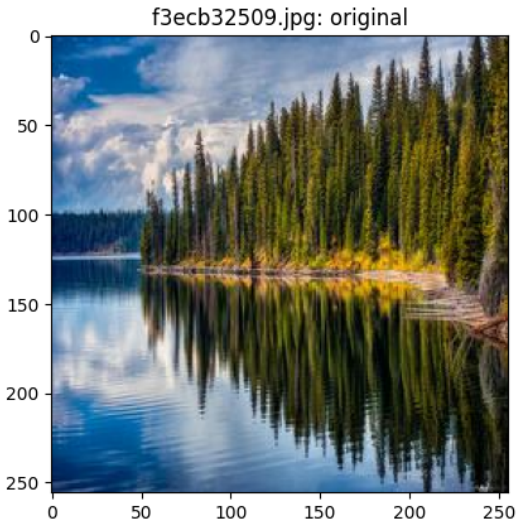
In [91]:
```
disp = PredictorDisplay(predictor)
disp.show_result()
```

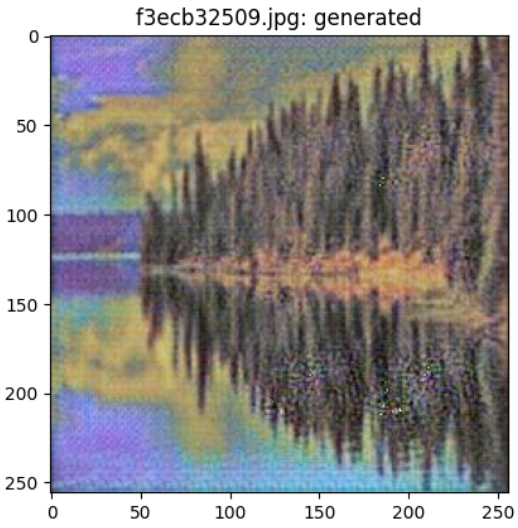












Compress the files for submitting.

In [92]:
```
predictor.submit()
```

Conclusion and Analysis

The overall results indicate satisfactory performance when dealing with complex photos. However, when applied to photos with fewer colors and simpler compositions, the generated paintings show minimal transformation or improvement.

To address this limitation, it is recommended to explore alternative models such as resNet, as suggested in the CycleGAN paper. The implementation of resNet may offer enhanced capabilities in transforming photos with simpler characteristics, potentially improving the quality and diversity of the generated paintings.

In [ ]: