week5-2

April 30, 2024

1 Week 5 GANs

Student, University of Colorado Boulder

1.1 Step 1 Brief description of the problem and data

The week 5 mini project is an unsupervised deep learning project.

GAN, Generative Adversarial Networkm is a concept in Deep Learning where to achieve an objective, a neural networks learns on its own.

A GAN consists of at least two neural networks: a generator model and a discriminator model. The generator, in our project, is a neural network that creates the images. For our competition, we should generate images in the style of Monet. This generator is trained using a discriminator. The discriminator — a neural network that discriminates between real/true data from fake/generated data, in out project, discriminates between images.

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

I will be building a GAN that generates 7,000 to 10,000 Monet-style images.

The details are available in the Kaggle competition https://www.kaggle.com/competitions/gan-getting-started

[1]:
```
import numpy as np
import pandas as pd
import os

import cv2
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import GroupNormalization
```

```
2024-04-29 16:59:05.902285: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
```

```
2024-04-29 16:59:05.902416: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-04-29 16:59:06.022842: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
```

1.2 Step 2 Exploratory Data Analysis (EDA) - Inspect, Visualize and Clean the data

Data Extraction User has to register on Kaggle to get access to the dataset. The data is expected to be present in /kaggle/input/gan-getting-started/ or a similar folder ./kaggle/input/gan-getting-started/ where the notebook is running

[2]:
```
data_path = '/kaggle'
if os.path.isdir('kaggle'):
    data_path = 'kaggle'

print(f'data path is {data_path}')


monet_jpg_path = os.path.join(data_path, 'input/gan-getting-started/monet_jpg/')
photo_jpg_path = os.path.join(data_path, 'input/gan-getting-started/photo_jpg/')

monet_tfrec_path = os.path.join(data_path, 'input/gan-getting-started/
 ↪monet_tfrec/')
photo_tfrec_path = os.path.join(data_path, 'input/gan-getting-started/
 ↪photo_tfrec/')

print(f'monet jpg path is {monet_jpg_path}, photo jpg path is {photo_jpg_path},␣
 ↪monet tfrec path is {monet_tfrec_path},  photo tfrec path is␣
 ↪{photo_tfrec_path}')
```

```
data path is /kaggle
monet jpg path is /kaggle/input/gan-getting-started/monet_jpg/, photo jpg path
is /kaggle/input/gan-getting-started/photo_jpg/, monet tfrec path is
/kaggle/input/gan-getting-started/monet_tfrec/,  photo tfrec path is
/kaggle/input/gan-getting-started/photo_tfrec/
```

[3]:
```
def get_files_in_path(path, count):
    paths = []
    c= 0
    for _, _, file_names in os.walk(path):
        for file_name in file_names:
            paths.append(os.path.join(path, file_name))
            c += 1
```

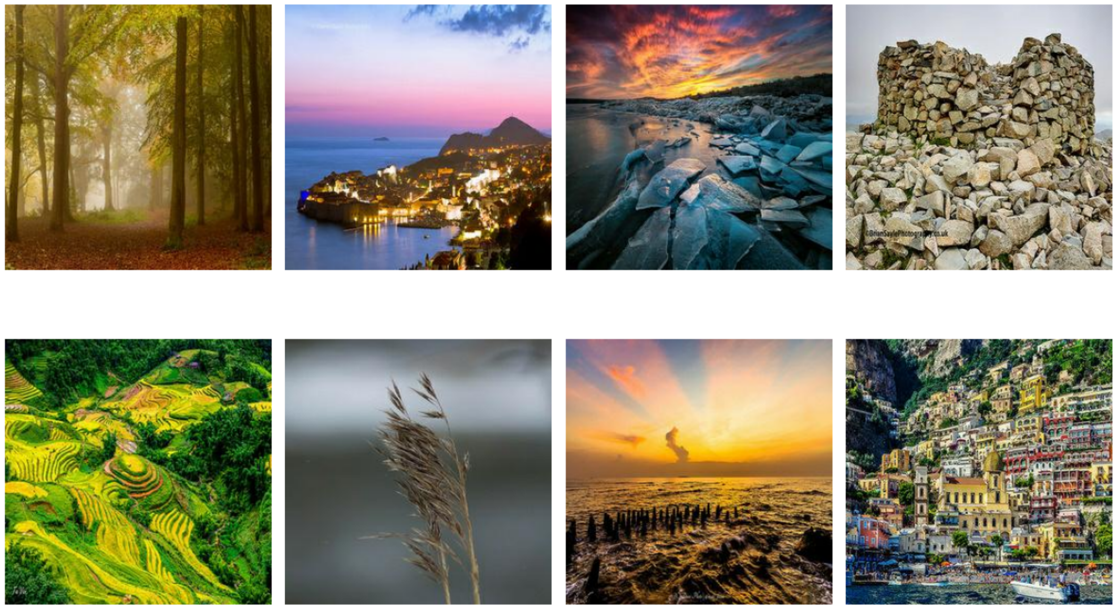
```
            if c == count:
                break
    return paths

def display_images(image_paths):
    figure, ax = plt.subplots(nrows=2,ncols=4,figsize=(12,8))
    for i, path in enumerate(image_paths):
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        try:
            ax.ravel()[i].imshow(image)
            ax.ravel()[i].set_axis_off()
        except Exception as ex:
            print(f'exception {ex} showing image {path}')
    plt.tight_layout()
    plt.show()
```

Let us look at some monet images

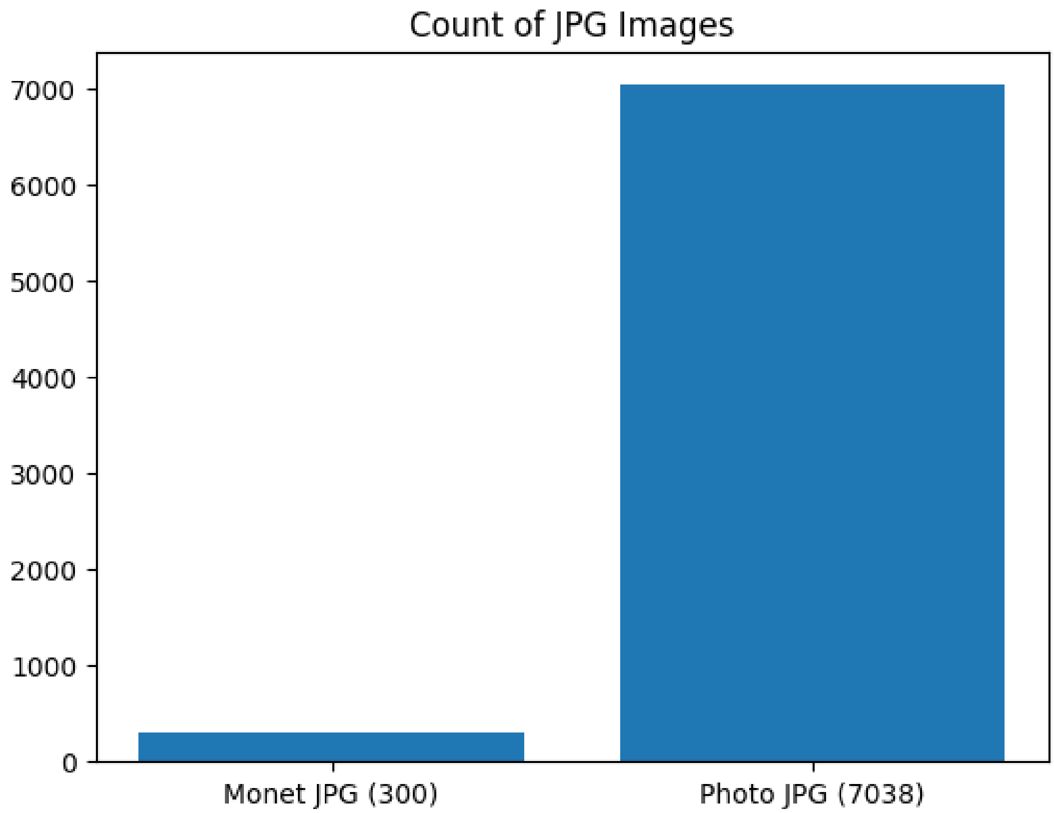
[4]:
```
display_images(get_files_in_path(monet_jpg_path, 8))
```

Let us look at some actual images

[5]:
```
display_images(get_files_in_path(photo_jpg_path, 8))
```



[6]:
```
monet_jpg_images = get_files_in_path(monet_jpg_path, 0)
photo_jpg_images = get_files_in_path(photo_jpg_path, 0)

monet_jpg_count = len(monet_jpg_images)
photo_jpg_count = len(photo_jpg_images)

x = [f'Monet JPG ({monet_jpg_count})', f'Photo JPG ({photo_jpg_count})']
y = [monet_jpg_count, photo_jpg_count]
plt.bar(x,y)
plt.title('Count of JPG Images')
plt.show()
```



Check for TPU

[7]:
```
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
2.15.0
```

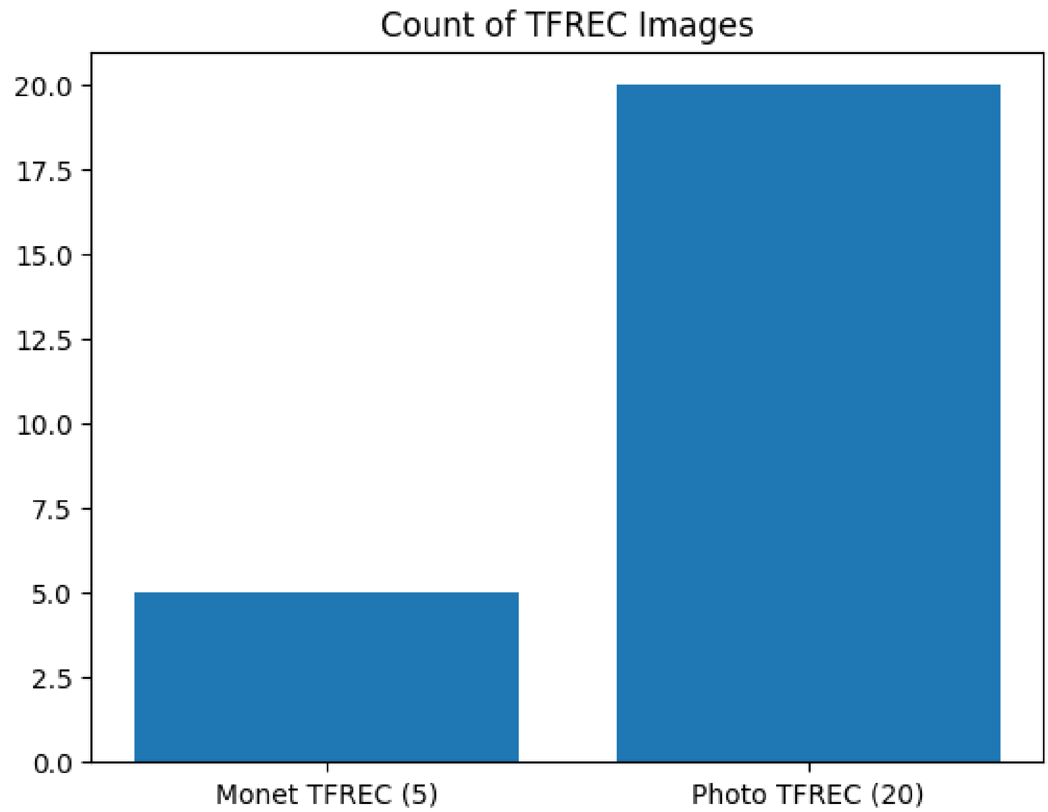
[8]:
```
IMAGE_SIZE = [256, 256]

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
```

[9]:
```
def load_dataset(filenames, labeled=True, ordered=False):
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
    return dataset

monet_tfrec_files = get_files_in_path(monet_tfrec_path, 0)
print('Monet TFRecord Files:', len(monet_tfrec_files))

photo_tfrec_files = get_files_in_path(photo_tfrec_path, 0)
print('Photo TFRecord Files:', len(photo_tfrec_files))

monet_ds = load_dataset(monet_tfrec_files, labeled=True).batch(1)
photo_ds = load_dataset(photo_tfrec_files, labeled=True).batch(1)
```

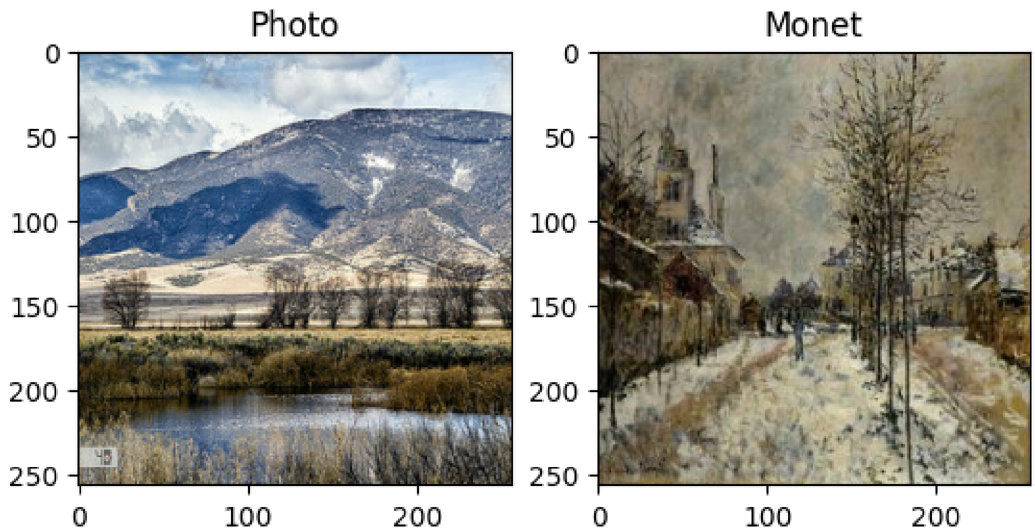
```
Monet TFRecord Files: 5
Photo TFRecord Files: 20
```

[10]:
```
monet_tfrec_count = len(monet_tfrec_files)
photo_tfrec_count = len(photo_tfrec_files)

x = [f'Monet TFREC ({monet_tfrec_count})', f'Photo TFREC ({photo_tfrec_count})']
y = [monet_tfrec_count, photo_tfrec_count]
plt.bar(x,y)
plt.title('Count of TFREC Images')
plt.show()
```



[11]:
```
example_monet = next(iter(monet_ds))
example_photo = next(iter(photo_ds))
plt.subplot(121)
plt.title('Photo')
plt.imshow(example_photo[0] * 0.5 + 0.5)

plt.subplot(122)
plt.title('Monet')
plt.imshow(example_monet[0] * 0.5 + 0.5)
```

[11]: <matplotlib.image.AxesImage at 0x790e21d463e0>



1.3 Model building

[12]:
```
OUTPUT_CHANNELS = 3
INPUT_SHAPE = [256,256,3]

class Sample:
    def __init__(self, filters, size, strides, padding, use_bias, upsample,␣
 ↪apply_instance_norm, apply_dropout):
        self.filters = filters
        self.size = size
        self.strides = strides
        self.padding = padding
        self.use_bias = use_bias
        self.upsample = upsample
        self.apply_instance_norm = apply_instance_norm
        self.apply_dropout = apply_dropout

    def get_initializer(self):
        return tf.random_normal_initializer(0., 0.02)

    def get_gamma_initializer(self):
        return keras.initializers.RandomNormal(mean=0.0, stddev=0.02)

    def get_sample(self):
        sample = keras.Sequential()
        if self.upsample:
            sample.add(layers.Conv2DTranspose(self.filters, self.size,␣
 ↪strides=self.strides, padding=self.padding,
```

```
                                kernel_initializer=self.get_initializer(),␣
↪use_bias=self.use_bias))
        else:
            sample.add(layers.Conv2D(self.filters, self.size, strides=self.
↪strides, padding=self.padding,
                                kernel_initializer=self.get_initializer(),␣
↪use_bias=self.use_bias))
        if self.apply_instance_norm:
            sample.add(GroupNormalization(groups=-1, gamma_initializer=self.
↪get_gamma_initializer()))

        if self.apply_dropout:
            sample.add(layers.Dropout(0.5))
        sample.add(layers.LeakyReLU())
        return sample
```

[13]:
```
def get_generator():
    inputs = layers.Input(shape=INPUT_SHAPE)

    down_stack = [
        Sample(filters=64, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=False, apply_dropout=False).get_sample(),
        Sample(filters=128, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=True, apply_dropout=False).get_sample(),
        Sample(filters=256, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=True, apply_dropout=False).get_sample(),
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=True, apply_dropout=False).get_sample(),
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=True, apply_dropout=False).get_sample(),
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=True, apply_dropout=False).get_sample(),
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=True, apply_dropout=False).get_sample(),
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=True, apply_dropout=False).get_sample(),
    ]

    up_stack = [
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=True, apply_instance_norm=False, apply_dropout=True).get_sample(),␣
↪# (bs, 2, 2, 1024)
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=True, apply_instance_norm=False, apply_dropout=True).get_sample(),␣
↪# (bs, 4, 4, 1024)
```

```
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=True, apply_instance_norm=False, apply_dropout=True).get_sample(),␣
↪# (bs, 8, 8, 1024)
        Sample(filters=512, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=True, apply_instance_norm=False, apply_dropout=False).get_sample(),␣
↪# (bs, 16, 16, 1024)
        Sample(filters=256, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=True, apply_instance_norm=False, apply_dropout=False).get_sample(),␣
↪# (bs, 32, 32, 512)
        Sample(filters=128, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=True, apply_instance_norm=False, apply_dropout=False).get_sample(),␣
↪# (bs, 64, 64, 256)
        Sample(filters=64, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=True, apply_instance_norm=False, apply_dropout=False).get_sample(),␣
↪# (bs, 128, 128, 128)
    ]

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

    x = inputs

    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)
    skips = reversed(skips[:-1])

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

    x = last(x)

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

[14]:
```
def get_discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)

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

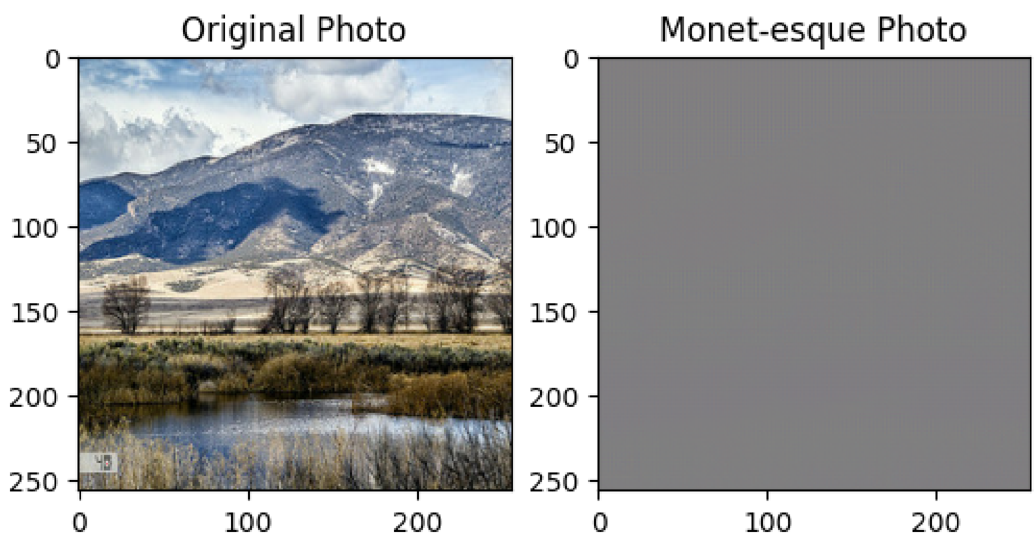
```
    x = inp
    s1 = Sample(filters=64, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=False, apply_dropout=False).get_sample()
    down1 = s1(x) # (bs, 128, 128, 64)

    s2 = Sample(filters=128, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=False, apply_dropout=False).get_sample()

    down2 = s2(down1) # (bs, 64, 64, 128)

    s3 = Sample(filters=256, size=4, strides=2, padding='same', use_bias=False,␣
↪upsample=False, apply_instance_norm=False, apply_dropout=False).get_sample()
    down3 = s3(down2) # (bs, 32, 32, 256)

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

    norm1 = GroupNormalization(groups=-1, gamma_initializer=gamma_init)(conv)

    leaky_relu = layers.LeakyReLU()(norm1)

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

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

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

[15]:
```
with strategy.scope():
    monet_generator = get_generator() # transforms photos to Monet-esque␣
↪paintings
    photo_generator = get_generator() # transforms Monet paintings to be more␣
↪like photos

    monet_discriminator = get_discriminator() # differentiates real Monet␣
↪paintings and generated Monet paintings
    photo_discriminator = get_discriminator() # differentiates real photos and␣
↪generated photos
```

Let us look at a monet without training the generator

[16]:
```
def generate_monet_and_show(example_photo):
    to_monet = monet_generator(example_photo)
    plt.subplot(1, 2, 1)
```

```
    plt.title("Original Photo")
    plt.imshow(example_photo[0] * 0.5 + 0.5)

    plt.subplot(1, 2, 2)
    plt.title("Monet-esque Photo")
    plt.imshow(to_monet[0] * 0.5 + 0.5)
    plt.show()

generate_monet_and_show(example_photo)
```



[17]:
```
class CycleGan(keras.Model):
    def __init__(
        self,
        monet_generator,
        photo_generator,
        monet_discriminator,
        photo_discriminator,
        lambda_cycle=10,
    ):
        super(CycleGan, self).__init__()
        self.m_gen = monet_generator
        self.p_gen = photo_generator
        self.m_disc = monet_discriminator
        self.p_disc = photo_discriminator
        self.lambda_cycle = lambda_cycle

    def compile(
        self,
```

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

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

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

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

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

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

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

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

```
            # evaluates total cycle consistency loss
            total_cycle_loss = self.cycle_loss_fn(real_monet, cycled_monet,␣
↪self.lambda_cycle) + self.cycle_loss_fn(real_photo, cycled_photo, self.
↪lambda_cycle)

            # evaluates total generator loss
            total_monet_gen_loss = monet_gen_loss + total_cycle_loss + self.
↪identity_loss_fn(real_monet, same_monet, self.lambda_cycle)
            total_photo_gen_loss = photo_gen_loss + total_cycle_loss + self.
↪identity_loss_fn(real_photo, same_photo, self.lambda_cycle)

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

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

        monet_discriminator_gradients = tape.gradient(monet_disc_loss,
                                                      self.m_disc.
↪trainable_variables)
        photo_discriminator_gradients = tape.gradient(photo_disc_loss,
                                                      self.p_disc.
↪trainable_variables)

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

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

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

        self.p_disc_optimizer.apply_gradients(zip(photo_discriminator_gradients,
```

```
                                                          self.p_disc.
↪trainable_variables))

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

[18]:
```
with strategy.scope():
    def discriminator_loss(real, generated):
        real_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True,␣
↪reduction=tf.keras.losses.Reduction.NONE)(tf.ones_like(real), real)

        generated_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True,␣
↪reduction=tf.keras.losses.Reduction.NONE)(tf.zeros_like(generated),␣
↪generated)

        total_disc_loss = real_loss + generated_loss

        return total_disc_loss * 0.5
```

[19]:
```
with strategy.scope():
    def generator_loss(generated):
        return tf.keras.losses.BinaryCrossentropy(from_logits=True,␣
↪reduction=tf.keras.losses.Reduction.NONE)(tf.ones_like(generated), generated)
```

[20]:
```
with strategy.scope():
    def calc_cycle_loss(real_image, cycled_image, LAMBDA):
        loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))

        return LAMBDA * loss1
```

[21]:
```
with strategy.scope():
    def identity_loss(real_image, same_image, LAMBDA):
        loss = tf.reduce_mean(tf.abs(real_image - same_image))
        return LAMBDA * 0.5 * loss
```

[22]:
```
with strategy.scope():
    monet_generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
    photo_generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

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

[23]:
```python
with strategy.scope():
    cycle_gan_model = CycleGan(
        monet_generator, photo_generator, monet_discriminator,␣
 ↪photo_discriminator
    )

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

[24]:
```python
history = cycle_gan_model.fit(
    tf.data.Dataset.zip((monet_ds, photo_ds)),
    epochs=25
)
```

```
Epoch 1/25

WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1714410065.497779      75 device_compiler.h:186] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

300/300            153s 210ms/step -
monet_disc_loss: 0.6479 - monet_gen_loss: 5.1362 - photo_disc_loss: 0.5927 -
photo_gen_loss: 5.2595 - loss: 0.0000e+00
Epoch 2/25

/opt/conda/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of
data; interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to use the
`.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)

300/300            66s 221ms/step -
monet_disc_loss: 0.6173 - monet_gen_loss: 3.8801 - photo_disc_loss: 0.5664 -
photo_gen_loss: 4.0591 - loss: 0.0000e+00
Epoch 3/25
300/300            66s 220ms/step -
monet_disc_loss: 0.5915 - monet_gen_loss: 3.9083 - photo_disc_loss: 0.6070 -
photo_gen_loss: 3.9389 - loss: 0.0000e+00
Epoch 4/25
300/300            66s 220ms/step -
monet_disc_loss: 0.6185 - monet_gen_loss: 3.7412 - photo_disc_loss: 0.5974 -
```

```
photo_gen_loss: 3.8024 - loss: 0.0000e+00
Epoch 5/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6147 - monet_gen_loss: 3.5571 - photo_disc_loss: 0.5971 -
photo_gen_loss: 3.6424 - loss: 0.0000e+00
Epoch 6/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6213 - monet_gen_loss: 3.4241 - photo_disc_loss: 0.5824 -
photo_gen_loss: 3.5556 - loss: 0.0000e+00
Epoch 7/25
300/300             66s 221ms/step -
monet_disc_loss: 0.6165 - monet_gen_loss: 3.3748 - photo_disc_loss: 0.5774 -
photo_gen_loss: 3.5381 - loss: 0.0000e+00
Epoch 8/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6234 - monet_gen_loss: 3.3034 - photo_disc_loss: 0.5781 -
photo_gen_loss: 3.4859 - loss: 0.0000e+00
Epoch 9/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6234 - monet_gen_loss: 3.2807 - photo_disc_loss: 0.5747 -
photo_gen_loss: 3.4741 - loss: 0.0000e+00
Epoch 10/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6223 - monet_gen_loss: 3.2167 - photo_disc_loss: 0.5708 -
photo_gen_loss: 3.4112 - loss: 0.0000e+00
Epoch 11/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6213 - monet_gen_loss: 3.1702 - photo_disc_loss: 0.5679 -
photo_gen_loss: 3.3817 - loss: 0.0000e+00
Epoch 12/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6251 - monet_gen_loss: 3.1251 - photo_disc_loss: 0.5698 -
photo_gen_loss: 3.3490 - loss: 0.0000e+00
Epoch 13/25
300/300             66s 221ms/step -
monet_disc_loss: 0.6171 - monet_gen_loss: 3.0988 - photo_disc_loss: 0.5670 -
photo_gen_loss: 3.3099 - loss: 0.0000e+00
Epoch 14/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6187 - monet_gen_loss: 3.0866 - photo_disc_loss: 0.5622 -
photo_gen_loss: 3.3129 - loss: 0.0000e+00
Epoch 15/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6192 - monet_gen_loss: 3.0396 - photo_disc_loss: 0.5551 -
photo_gen_loss: 3.2869 - loss: 0.0000e+00
Epoch 16/25
300/300             66s 220ms/step -
monet_disc_loss: 0.6192 - monet_gen_loss: 3.0319 - photo_disc_loss: 0.5648 -
```

```
photo_gen_loss: 3.2544 - loss: 0.0000e+00
Epoch 17/25
300/300              66s 221ms/step -
monet_disc_loss: 0.6139 - monet_gen_loss: 3.0338 - photo_disc_loss: 0.5543 -
photo_gen_loss: 3.2605 - loss: 0.0000e+00
Epoch 18/25
300/300              66s 221ms/step -
monet_disc_loss: 0.6098 - monet_gen_loss: 3.0221 - photo_disc_loss: 0.5686 -
photo_gen_loss: 3.2099 - loss: 0.0000e+00
Epoch 19/25
300/300              66s 220ms/step -
monet_disc_loss: 0.6219 - monet_gen_loss: 2.9877 - photo_disc_loss: 0.5711 -
photo_gen_loss: 3.1933 - loss: 0.0000e+00
Epoch 20/25
300/300              66s 220ms/step -
monet_disc_loss: 0.6300 - monet_gen_loss: 2.9164 - photo_disc_loss: 0.5655 -
photo_gen_loss: 3.1435 - loss: 0.0000e+00
Epoch 21/25
300/300              66s 220ms/step -
monet_disc_loss: 0.6213 - monet_gen_loss: 2.9100 - photo_disc_loss: 0.5661 -
photo_gen_loss: 3.1251 - loss: 0.0000e+00
Epoch 22/25
300/300              66s 220ms/step -
monet_disc_loss: 0.6182 - monet_gen_loss: 2.8748 - photo_disc_loss: 0.5574 -
photo_gen_loss: 3.1013 - loss: 0.0000e+00
Epoch 23/25
300/300              66s 220ms/step -
monet_disc_loss: 0.6010 - monet_gen_loss: 2.9186 - photo_disc_loss: 0.5546 -
photo_gen_loss: 3.1025 - loss: 0.0000e+00
Epoch 24/25
300/300              66s 220ms/step -
monet_disc_loss: 0.6139 - monet_gen_loss: 2.8522 - photo_disc_loss: 0.5591 -
photo_gen_loss: 3.0456 - loss: 0.0000e+00
Epoch 25/25
300/300              66s 220ms/step -
monet_disc_loss: 0.6211 - monet_gen_loss: 2.8102 - photo_disc_loss: 0.5539 -
photo_gen_loss: 3.0353 - loss: 0.0000e+00
```

[25]:

```python
_, ax = plt.subplots(5, 2, figsize=(12, 12))
for i, img in enumerate(photo_ds.take(5)):
    prediction = monet_generator(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 Photo")
```

```
    ax[i, 1].set_title("Monet-esque")
    ax[i, 0].axis("off")
    ax[i, 1].axis("off")
plt.show()
```

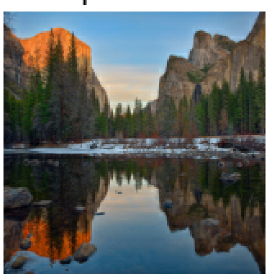
Input Photo

Monet-esque

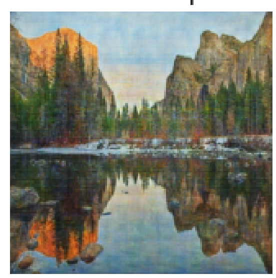
Input Photo

Monet-esque

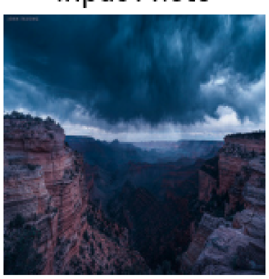
Input Photo



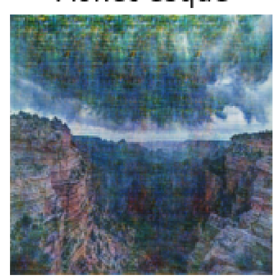
Monet-esque



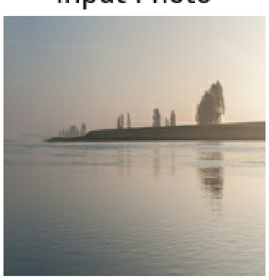
Input Photo



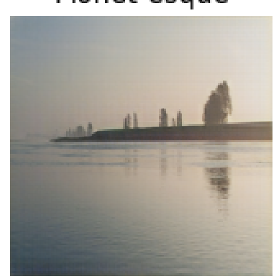
Monet-esque



Input Photo



Monet-esque



[26]:
```
import PIL
import shutil

! mkdir ../images


def generate_write_and_archive_images(directory_name):
    i = 1
    for img in photo_ds:
        prediction = monet_generator(img, training=False)[0].numpy()
        prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
        im = PIL.Image.fromarray(prediction)
        im.save("../images/" + str(i) + ".jpg")
        i += 1
        if i%100 == 0:
            print(f'generated {i} files')
    print(f'creating zip')
    shutil.make_archive("/kaggle/working/images", 'zip', "/kaggle/images")

generate_write_and_archive_images('images')
```

```
generated 100 files
generated 200 files
generated 300 files
generated 400 files
generated 500 files
generated 600 files
generated 700 files
generated 800 files
generated 900 files
generated 1000 files
generated 1100 files
generated 1200 files
generated 1300 files
generated 1400 files
generated 1500 files
generated 1600 files
generated 1700 files
generated 1800 files
generated 1900 files
generated 2000 files
generated 2100 files
generated 2200 files
generated 2300 files
generated 2400 files
generated 2500 files
generated 2600 files
generated 2700 files
```

```
generated 2800 files
generated 2900 files
generated 3000 files
generated 3100 files
generated 3200 files
generated 3300 files
generated 3400 files
generated 3500 files
generated 3600 files
generated 3700 files
generated 3800 files
generated 3900 files
generated 4000 files
generated 4100 files
generated 4200 files
generated 4300 files
generated 4400 files
generated 4500 files
generated 4600 files
generated 4700 files
generated 4800 files
generated 4900 files
generated 5000 files
generated 5100 files
generated 5200 files
generated 5300 files
generated 5400 files
generated 5500 files
generated 5600 files
generated 5700 files
generated 5800 files
generated 5900 files
generated 6000 files
generated 6100 files
generated 6200 files
generated 6300 files
generated 6400 files
generated 6500 files
generated 6600 files
generated 6700 files
generated 6800 files
generated 6900 files
generated 7000 files
creating zip
```

This model with Adam optimizer scored 49.41614 on Kaggle

1.3.1 Hyper parameter tuning

Instead of Adam optimizer, I will now try the RMSProp optimizer to see how it performs. I will use the same learning rate as Adam $2e^{-4}$ and leave the rest of the parameters to the default values. I will run the model for 25 epochs and generate monet images

[27]:
```
with strategy.scope():
    monet_generator_optimizer = tf.keras.optimizers.RMSprop(2e-4)
    photo_generator_optimizer = tf.keras.optimizers.RMSprop(2e-4)

    monet_discriminator_optimizer = tf.keras.optimizers.RMSprop(2e-4)
    photo_discriminator_optimizer = tf.keras.optimizers.RMSprop(2e-4)
```

[28]:
```
with strategy.scope():
    cycle_gan_model = CycleGan(
        monet_generator, photo_generator, monet_discriminator, 
 ↪photo_discriminator
    )

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

[29]:
```
history_rms = cycle_gan_model.fit(
    tf.data.Dataset.zip((monet_ds, photo_ds)),
    epochs=25
)
```

```
Epoch 1/25
300/300             138s 220ms/step -
monet_disc_loss: 0.6113 - monet_gen_loss: 2.8664 - photo_disc_loss: 0.5704 -
photo_gen_loss: 3.0427 - loss: 0.0000e+00
Epoch 2/25
300/300             66s 218ms/step -
monet_disc_loss: 0.6071 - monet_gen_loss: 2.8418 - photo_disc_loss: 0.5598 -
photo_gen_loss: 3.0107 - loss: 0.0000e+00
Epoch 3/25
300/300             65s 218ms/step -
monet_disc_loss: 0.6230 - monet_gen_loss: 2.7913 - photo_disc_loss: 0.5574 -
photo_gen_loss: 3.0068 - loss: 0.0000e+00
```

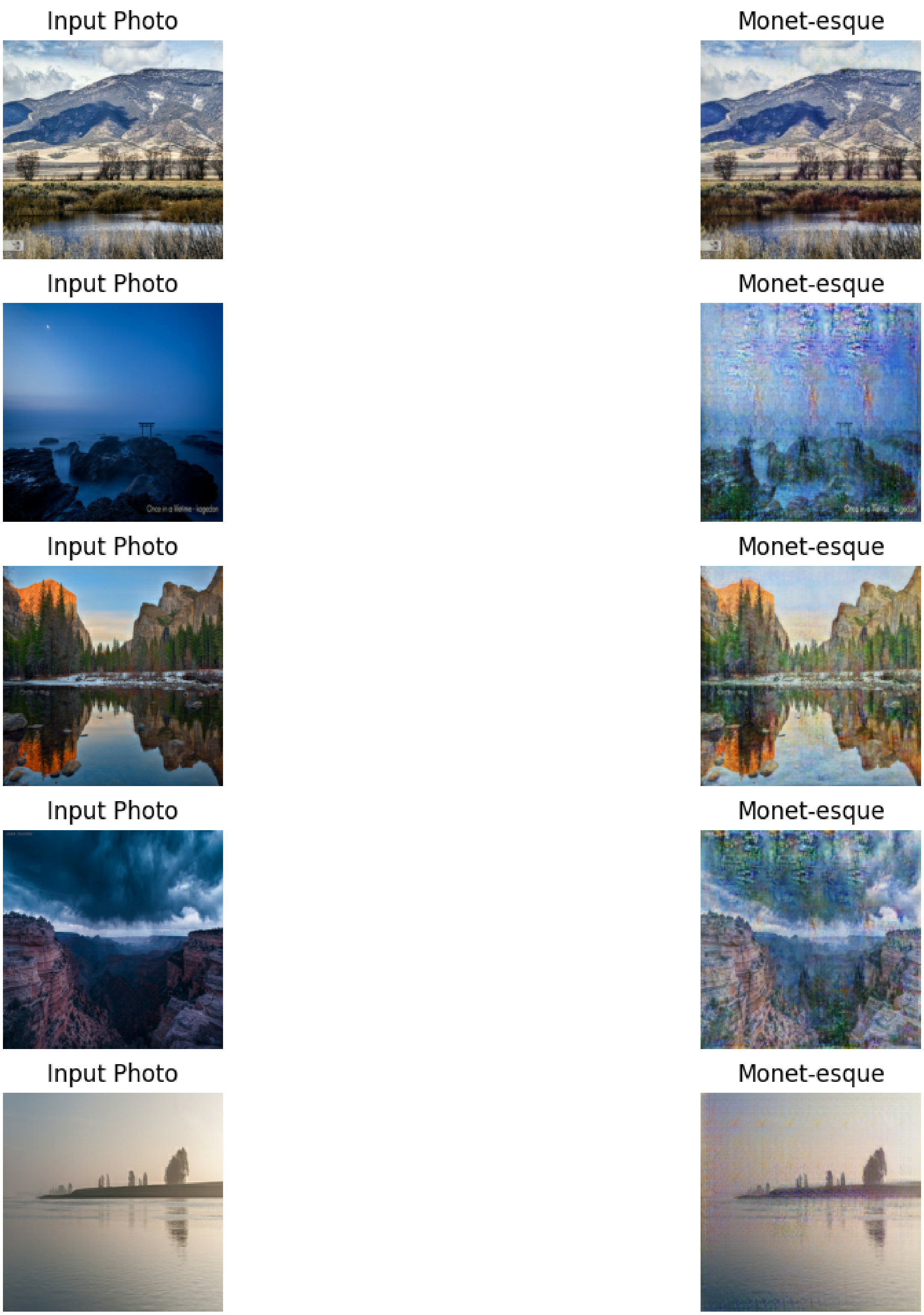
```
Epoch 4/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6214 - monet_gen_loss: 2.7908 - photo_disc_loss: 0.5606 -
photo_gen_loss: 3.0001 - loss: 0.0000e+00
Epoch 5/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6188 - monet_gen_loss: 2.7565 - photo_disc_loss: 0.5616 -
photo_gen_loss: 2.9596 - loss: 0.0000e+00
Epoch 6/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6129 - monet_gen_loss: 2.7633 - photo_disc_loss: 0.5573 -
photo_gen_loss: 2.9564 - loss: 0.0000e+00
Epoch 7/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6186 - monet_gen_loss: 2.7549 - photo_disc_loss: 0.5587 -
photo_gen_loss: 2.9728 - loss: 0.0000e+00
Epoch 8/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6221 - monet_gen_loss: 2.7313 - photo_disc_loss: 0.5538 -
photo_gen_loss: 2.9496 - loss: 0.0000e+00
Epoch 9/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6163 - monet_gen_loss: 2.7395 - photo_disc_loss: 0.5583 -
photo_gen_loss: 2.9438 - loss: 0.0000e+00
Epoch 10/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6222 - monet_gen_loss: 2.7278 - photo_disc_loss: 0.5522 -
photo_gen_loss: 2.9540 - loss: 0.0000e+00
Epoch 11/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6219 - monet_gen_loss: 2.7188 - photo_disc_loss: 0.5526 -
photo_gen_loss: 2.9513 - loss: 0.0000e+00
Epoch 12/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6182 - monet_gen_loss: 2.7055 - photo_disc_loss: 0.5519 -
photo_gen_loss: 2.9367 - loss: 0.0000e+00
Epoch 13/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6172 - monet_gen_loss: 2.6870 - photo_disc_loss: 0.5485 -
photo_gen_loss: 2.9182 - loss: 0.0000e+00
Epoch 14/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6171 - monet_gen_loss: 2.6957 - photo_disc_loss: 0.5495 -
photo_gen_loss: 2.9349 - loss: 0.0000e+00
Epoch 15/25
300/300              65s 218ms/step -
monet_disc_loss: 0.6150 - monet_gen_loss: 2.6912 - photo_disc_loss: 0.5411 -
photo_gen_loss: 2.9327 - loss: 0.0000e+00
```

```
Epoch 16/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6167 - monet_gen_loss: 2.6778 - photo_disc_loss: 0.5432 -
photo_gen_loss: 2.9171 - loss: 0.0000e+00
Epoch 17/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6124 - monet_gen_loss: 2.6768 - photo_disc_loss: 0.5445 -
photo_gen_loss: 2.9152 - loss: 0.0000e+00
Epoch 18/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6131 - monet_gen_loss: 2.6716 - photo_disc_loss: 0.5396 -
photo_gen_loss: 2.9083 - loss: 0.0000e+00
Epoch 19/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6112 - monet_gen_loss: 2.6775 - photo_disc_loss: 0.5431 -
photo_gen_loss: 2.9167 - loss: 0.0000e+00
Epoch 20/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6112 - monet_gen_loss: 2.6701 - photo_disc_loss: 0.5413 -
photo_gen_loss: 2.9015 - loss: 0.0000e+00
Epoch 21/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6093 - monet_gen_loss: 2.6967 - photo_disc_loss: 0.5465 -
photo_gen_loss: 2.9217 - loss: 0.0000e+00
Epoch 22/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6070 - monet_gen_loss: 2.6639 - photo_disc_loss: 0.5420 -
photo_gen_loss: 2.9007 - loss: 0.0000e+00
Epoch 23/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6109 - monet_gen_loss: 2.6700 - photo_disc_loss: 0.5421 -
photo_gen_loss: 2.8909 - loss: 0.0000e+00
Epoch 24/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6119 - monet_gen_loss: 2.6693 - photo_disc_loss: 0.5460 -
photo_gen_loss: 2.8978 - loss: 0.0000e+00
Epoch 25/25
300/300                65s 218ms/step -
monet_disc_loss: 0.6116 - monet_gen_loss: 2.6776 - photo_disc_loss: 0.5417 -
photo_gen_loss: 2.9024 - loss: 0.0000e+00
```

[30]:

```python
_, ax = plt.subplots(5, 2, figsize=(12, 12))
for i, img in enumerate(photo_ds.take(5)):
    prediction = monet_generator(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)
```

```python
    ax[i, 0].imshow(img)
    ax[i, 1].imshow(prediction)
    ax[i, 0].set_title("Input Photo")
    ax[i, 1].set_title("Monet-esque")
    ax[i, 0].axis("off")
    ax[i, 1].axis("off")
plt.show()
```


Input Photo
Monet-esque
Input Photo
Monet-esque
Input Photo
Monet-esque
Input Photo
Monet-esque
Input Photo
Monet-esque

[31]:
```python
import PIL
import shutil

! mkdir ../images


def generate_write_and_archive_images(directory_name):
    i = 1
    for img in photo_ds:
        prediction = monet_generator(img, training=False)[0].numpy()
        prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
        im = PIL.Image.fromarray(prediction)
        im.save("../images/" + str(i) + ".jpg")
        i += 1
        if i%100 == 0:
            print(f'generated {i} files')
    print(f'creating zip')
    shutil.make_archive("/kaggle/working/images", 'zip', "/kaggle/images")

generate_write_and_archive_images('images')
```

```
generated 100 files
generated 200 files
generated 300 files
generated 400 files
generated 500 files
generated 600 files
generated 700 files
generated 800 files
generated 900 files
generated 1000 files
generated 1100 files
generated 1200 files
generated 1300 files
generated 1400 files
generated 1500 files
generated 1600 files
generated 1700 files
generated 1800 files
generated 1900 files
generated 2000 files
generated 2100 files
generated 2200 files
generated 2300 files
generated 2400 files
generated 2500 files
generated 2600 files
generated 2700 files
```

```
generated 2800 files
generated 2900 files
generated 3000 files
generated 3100 files
generated 3200 files
generated 3300 files
generated 3400 files
generated 3500 files
generated 3600 files
generated 3700 files
generated 3800 files
generated 3900 files
generated 4000 files
generated 4100 files
generated 4200 files
generated 4300 files
generated 4400 files
generated 4500 files
generated 4600 files
generated 4700 files
generated 4800 files
generated 4900 files
generated 5000 files
generated 5100 files
generated 5200 files
generated 5300 files
generated 5400 files
generated 5500 files
generated 5600 files
generated 5700 files
generated 5800 files
generated 5900 files
generated 6000 files
generated 6100 files
generated 6200 files
generated 6300 files
generated 6400 files
generated 6500 files
generated 6600 files
generated 6700 files
generated 6800 files
generated 6900 files
generated 7000 files
creating zip
```

RMSProp scored better than the Adam optimizer but not by a huge margin.

Pix2Pix I have also tried to use pre-built model called Pix2Pix but to keep this notebook short, I am not including the code here. It will be available on github

This model didnt perform as good as the other finetuned model. But the score was decent and it was fairly simple to use

1.4 Results Comparision

Though all the 3 models explored performed well, the custom built model with RMSProp optimizer performed better than the other models even without much hyperparameter tuning for that optimizer

A brief comparision of the three models

	Optimizer	GPU	Kaggle score	Implementation complexity
Custom built	Adam	Yes	49.41614	Complex
Custom built	RMSProp	Yes	48.25865	Complex
Pre-trained Pix2Pix	Adam	Yes	69.97726	Simple

1.5 Conclusion

- I have tried 3 different models to generate monet images from photos. The results seem promising and finetuning can be explored further to get better scores
- Without GPU/TPU it is hard/almost impossible to do deep learning. Jupyter kernel kept restarting on my Mac and I spent a lot of time on it by the time I realised I can use the GPU on Kaggle for free (with some limits)
- GANs are an area that I would like to explore more to get more depth in understanding

1.6 References

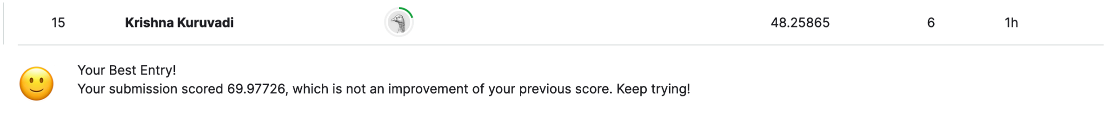
- https://www.kaggle.com/competitions/gan-getting-started/overview
- https://www.kaggle.com/code/amyjang/monet-cyclegan-tutorial
- https://www.tensorflow.org/tutorials/generative/cyclegan

1.7 Deliverables

Github: https://github.com/krishnakuruvadi/week5

Report: https://github.com/krishnakuruvadi/week5

Other notebooks: https://github.com/krishnakuruvadi/week5

Kaggle: 

15	**Krishna Kuruvadi**		48.25865	6	1h

Your Best Entry!
Your submission scored 69.97726, which is not an improvement of your previous score. Keep trying!