Untitled

August 25, 2023

1 I'm Something of a Painter Myself

1.1 Introduction

1.2 Setup

[22]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
import tensorflow_datasets as tfds

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
```

```
Number of replicas: 1
```

[2]:
```
path = "./gan-getting-started/"

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

photo_filenames = tf.io.gfile.glob(str(path + '/photo_tfrec/*.tfrec'))
```

```
print('Photo TFRecord Files:', len(photo_filenames))
```

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

1.3 EDA & Preprocessing

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

# normalize function
def normalize(image):
    return (tf.cast(image, tf.float32) / 127.5) - 1

# decode image
def decode_image(image):
    image = tf.image.decode_jpeg(image, channels=3)
    image = normalize(image)
    image = tf.reshape(image, [*image_size, 3])
    return image

# augmentation functions to increase sample size
def jitter(image):
    image = tf.image.resize(image, [int(256*1.3), int(256*1.3)],
                            method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
    image = tf.image.random_crop(image, size=[256, 256, 3])
    return image

def flip(image):
    return tf.image.flip_left_right(image)

# read TFRrecord
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

AUTOTUNE = tf.data.AUTOTUNE

# function to load in datasets
def load_dataset(filenames, labeled=True, ordered=False):
    dataset = tf.data.TFRecordDataset(filenames)
```

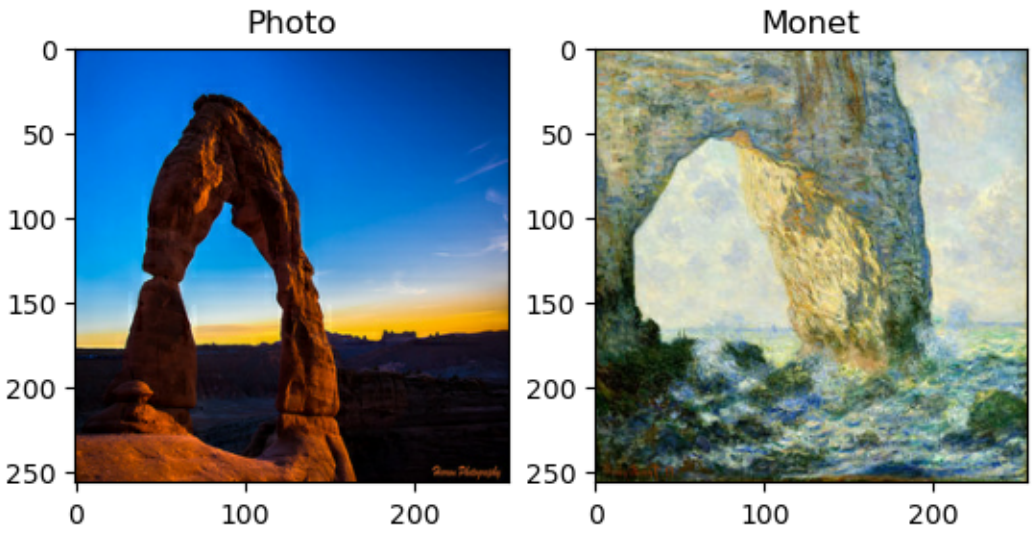
```
    dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
    return dataset
```

[9]:
```
monet_ds = load_dataset(monet_filenames, labeled=True).batch(1)
photo_ds = load_dataset(photo_filenames, labeled=True).batch(1)
```

[10]:
```
plt.subplot(121)
plt.title('Photo')
plt.imshow(next(iter(photo_ds))[0] * 0.5 + 0.5)

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

[10]:
```
<matplotlib.image.AxesImage at 0x296963e50>
```



1.4 Generator & Discriminator

[11]:
```
def Generator(LATENT_DIM=128, OUTPUT_CHANNELS=3):
    inputs = layers.Input(shape=[LATENT_DIM,])

    q = 4  # Change this value to control initial spatial dimensions
    x = layers.Dense(q * q * LATENT_DIM)(inputs)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Reshape((q, q, LATENT_DIM))(x)

    # Upsampling layers
```

```
    x = layers.Conv2DTranspose(LATENT_DIM, 4, strides=2, padding='same',␣
↪use_bias=False)(x)
    x = tfa.layers.InstanceNormalization()(x)
    x = layers.LeakyReLU(alpha=0.2)(x)

    x = layers.Conv2DTranspose(LATENT_DIM, 4, strides=2, padding='same',␣
↪use_bias=False)(x)
    x = tfa.layers.InstanceNormalization()(x)
    x = layers.LeakyReLU(alpha=0.2)(x)

    x = layers.Conv2DTranspose(LATENT_DIM // 2, 4, strides=2, padding='same',␣
↪use_bias=False)(x)
    x = tfa.layers.InstanceNormalization()(x)
    x = layers.LeakyReLU(alpha=0.2)(x)

    x = layers.Conv2DTranspose(LATENT_DIM // 4, 4, strides=2, padding='same',␣
↪use_bias=False)(x)
    x = tfa.layers.InstanceNormalization()(x)
    x = layers.LeakyReLU(alpha=0.2)(x)

    x = layers.Conv2DTranspose(LATENT_DIM // 8, 4, strides=2, padding='same',␣
↪use_bias=False)(x)
    x = tfa.layers.InstanceNormalization()(x)
    x = layers.LeakyReLU(alpha=0.2)(x)

    # Output layer
    initializer = tf.random_normal_initializer(0., 0.02)
    last = layers.Conv2DTranspose(OUTPUT_CHANNELS, 4, strides=2,␣
↪padding='same', kernel_initializer=initializer, activation='tanh')(x)

    return keras.Model(inputs=inputs, outputs=last, name="generator")
```

[29]:
```
with strategy.scope():
    def discriminator_loss(predictions_real, predictions_gen, labels_real):
        gen_loss = tf.reduce_mean(tf.square(predictions_gen - tf.
↪reduce_mean(predictions_real) + labels_real))
        real_loss = tf.reduce_mean(tf.square(predictions_real - tf.
↪reduce_mean(predictions_gen) - labels_real))
        return (gen_loss + real_loss) / 2

    def generator_loss(predictions_real, predictions_gen, labels_real):
        gen_loss = tf.reduce_mean(tf.square(predictions_gen - tf.
↪reduce_mean(predictions_real) - labels_real))
        real_loss = tf.reduce_mean(tf.square(predictions_real - tf.
↪reduce_mean(predictions_gen) + labels_real))
        return (gen_loss + real_loss) / 2
```

[12]:
```
def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)
    inp = layers.Input(shape=[256, 256, 3], name='input_image')
    x = inp

    # First convolutional layer
    x = layers.Conv2D(64, 4, strides=2, padding='same',␣
 ↪kernel_initializer=initializer, use_bias=False)(x)
    x = layers.LeakyReLU()(x)

    # Second convolutional layer
    x = layers.Conv2D(128, 4, strides=2, padding='same',␣
 ↪kernel_initializer=initializer, use_bias=False)(x)
    x = layers.LeakyReLU()(x)

    # Third convolutional layer
    x = layers.Conv2D(256, 4, strides=2, padding='same',␣
 ↪kernel_initializer=initializer, use_bias=False)(x)
    x = layers.LeakyReLU()(x)

    # Fourth convolutional layer
    x = layers.Conv2D(512, 4, strides=1, padding='valid',␣
 ↪kernel_initializer=initializer, use_bias=False)(x)
    x = tfa.layers.InstanceNormalization(gamma_initializer=gamma_init)(x)
    x = layers.LeakyReLU()(x)

    # Final convolutional layer
    x = layers.Conv2D(1, 4, strides=1, padding='valid',␣
 ↪kernel_initializer=initializer)(x)
    x = layers.LeakyReLU(alpha=0.2)(x)

    # Flatten and output
    x = layers.Flatten()(x)
    output = layers.Dense(1, activation='sigmoid')(x)

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

[24]:
```
with strategy.scope():
    monet_generator = Generator()
    monet_discriminator = Discriminator()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_image (InputLayer)    [(None, 256, 256, 3)]     0
```

```
 conv2d (Conv2D)              (None, 128, 128, 64)      3072

 leaky_re_lu_6 (LeakyReLU)    (None, 128, 128, 64)      0

 conv2d_1 (Conv2D)            (None, 64, 64, 128)       131072

 leaky_re_lu_7 (LeakyReLU)    (None, 64, 64, 128)       0

 conv2d_2 (Conv2D)            (None, 32, 32, 256)       524288

 leaky_re_lu_8 (LeakyReLU)    (None, 32, 32, 256)       0

 conv2d_3 (Conv2D)            (None, 29, 29, 512)       2097152

 instance_normalization_5 (   (None, 29, 29, 512)       1024
 InstanceNormalization)

 leaky_re_lu_9 (LeakyReLU)    (None, 29, 29, 512)       0

 conv2d_4 (Conv2D)            (None, 26, 26, 1)         8193

 leaky_re_lu_10 (LeakyReLU)   (None, 26, 26, 1)         0

 flatten (Flatten)            (None, 676)               0

 dense_1 (Dense)              (None, 1)                 677

=================================================================
Total params: 2765478 (10.55 MB)
Trainable params: 2765478 (10.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```python
with strategy.scope():
    def discriminator_loss(predictions_real, predictions_gen, labels_real):
        gen_loss = tf.reduce_mean(tf.square(predictions_gen - tf.
 ↪reduce_mean(predictions_real) + labels_real))
        real_loss = tf.reduce_mean(tf.square(predictions_real - tf.
 ↪reduce_mean(predictions_gen) - labels_real))
        return (gen_loss + real_loss) / 2

    def generator_loss(predictions_real, predictions_gen, labels_real):
        gen_loss = tf.reduce_mean(tf.square(predictions_gen - tf.
 ↪reduce_mean(predictions_real) - labels_real))
        real_loss = tf.reduce_mean(tf.square(predictions_real - tf.
 ↪reduce_mean(predictions_gen) + labels_real))
```

```
        return (gen_loss + real_loss) / 2
```

1.5 Model Development & Training

[32]:
```
class CustomMonetGan(keras.Model):
    def __init__(self, custom_generator, custom_discriminator, latent_dim,␣
 ↪real_label=0.5, fake_label=0):
        super(CustomMonetGan, self).__init__()
        self.generator = custom_generator
        self.discriminator = custom_discriminator
        self.latent_dim = latent_dim
        self.real_label = real_label
        self.fake_label = fake_label

    def compile(self, d_optimizer, g_optimizer, d_loss_function,␣
 ↪g_loss_function):
        super(CustomMonetGan, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.d_loss_function = d_loss_function
        self.g_loss_function = g_loss_function

    def train_step(self, real_images):
        if isinstance(real_images, tuple):
            real_images = real_images[0]

        batch_size = tf.shape(real_images)[0]
        random_latent_vectors = tf.random.normal(shape=(batch_size, self.
 ↪latent_dim))

        labels_gen  = tf.zeros((batch_size, 1)) + self.fake_label
        labels_real = tf.zeros((batch_size, 1)) + self.real_label

        labels_gen  += 0.05 * tf.random.uniform(tf.shape(labels_gen))
        labels_real += 0.05 * tf.random.uniform(tf.shape(labels_real))

        with tf.GradientTape() as d_tape:
            generated_images = self.generator(random_latent_vectors,␣
 ↪training=False)

            predictions_real = self.discriminator(real_images, training=True)
            predictions_gen  = self.discriminator(generated_images,␣
 ↪training=True)
```

```
            d_loss = self.d_loss_function(predictions_real, predictions_gen,␣
↪labels_real)

        d_gradients = d_tape.gradient(d_loss, self.discriminator.
↪trainable_weights)
        self.d_optimizer.apply_gradients(zip(d_gradients, self.discriminator.
↪trainable_weights))

        with tf.GradientTape() as g_tape:
            generated_images = self.generator(random_latent_vectors,␣
↪training=True)

            predictions_real = self.discriminator(real_images, training=False)
            predictions_gen  = self.discriminator(generated_images,␣
↪training=False)

            g_loss = self.g_loss_function(predictions_real, predictions_gen,␣
↪labels_real)

        g_gradients = g_tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(zip(g_gradients, self.generator.
↪trainable_weights))

        return {"discriminator_loss": d_loss, "generator_loss": g_loss}
```

[34]:
```
EPOCHS = 50

LR_G = 0.001
LR_D = 0.0005
beta_1 = 0.5

real_label = 0.66
fake_label = 0

with strategy.scope():
    custom_monet_gan = CustomMonetGan(
        custom_discriminator=monet_discriminator,
        custom_generator=monet_generator,
        latent_dim=LATENT_DIM,
        real_label=real_label,
        fake_label=fake_label
    )

    custom_monet_gan.compile(
        d_optimizer=tf.keras.optimizers.Adam(learning_rate=LR_D, beta_1=beta_1),
        g_optimizer=tf.keras.optimizers.Adam(learning_rate=LR_G, beta_1=beta_1),
```

```
        d_loss_function=discriminator_loss,
        g_loss_function=generator_loss
    )
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
```

[36]:
```
custom_monet_gan.fit(
    monet_ds,
    epochs=EPOCHS,
)
```

```
Epoch 1/50
300/300 [==============================] - 48s 154ms/step - discriminator_loss:
0.4725 - generator_loss: 0.4953
Epoch 2/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4675 - generator_loss: 0.4673
Epoch 3/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4698 - generator_loss: 0.4698
Epoch 4/50
300/300 [==============================] - 49s 162ms/step - discriminator_loss:
0.4674 - generator_loss: 0.4674
Epoch 5/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
0.4706 - generator_loss: 0.4706
Epoch 6/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
0.4703 - generator_loss: 0.4703
Epoch 7/50
300/300 [==============================] - 49s 165ms/step - discriminator_loss:
0.4676 - generator_loss: 0.4676
Epoch 8/50
300/300 [==============================] - 49s 162ms/step - discriminator_loss:
0.4697 - generator_loss: 0.4697
Epoch 9/50
300/300 [==============================] - 50s 165ms/step - discriminator_loss:
0.4710 - generator_loss: 0.4710
Epoch 10/50
300/300 [==============================] - 50s 167ms/step - discriminator_loss:
0.4706 - generator_loss: 0.4706
Epoch 11/50
300/300 [==============================] - 49s 165ms/step - discriminator_loss:
```

```
0.4694 - generator_loss: 0.4694
Epoch 12/50
300/300 [==============================] - 51s 169ms/step - discriminator_loss:
0.4685 - generator_loss: 0.4685
Epoch 13/50
300/300 [==============================] - 48s 161ms/step - discriminator_loss:
0.4691 - generator_loss: 0.4691
Epoch 14/50
300/300 [==============================] - 50s 166ms/step - discriminator_loss:
0.4698 - generator_loss: 0.4698
Epoch 15/50
300/300 [==============================] - 50s 167ms/step - discriminator_loss:
0.4689 - generator_loss: 0.4689
Epoch 16/50
300/300 [==============================] - 50s 166ms/step - discriminator_loss:
0.4686 - generator_loss: 0.4686
Epoch 17/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4673 - generator_loss: 0.4673
Epoch 18/50
300/300 [==============================] - 50s 168ms/step - discriminator_loss:
0.4680 - generator_loss: 0.4680
Epoch 19/50
300/300 [==============================] - 48s 160ms/step - discriminator_loss:
0.4687 - generator_loss: 0.4687
Epoch 20/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4732 - generator_loss: 0.4732
Epoch 21/50
300/300 [==============================] - 49s 162ms/step - discriminator_loss:
0.4705 - generator_loss: 0.4705
Epoch 22/50
300/300 [==============================] - 49s 165ms/step - discriminator_loss:
0.4696 - generator_loss: 0.4696
Epoch 23/50
300/300 [==============================] - 50s 166ms/step - discriminator_loss:
0.4700 - generator_loss: 0.4700
Epoch 24/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4661 - generator_loss: 0.4661
Epoch 25/50
300/300 [==============================] - 48s 162ms/step - discriminator_loss:
0.4692 - generator_loss: 0.4692
Epoch 26/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
0.4673 - generator_loss: 0.4673
Epoch 27/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
```

```
0.4699 - generator_loss: 0.4699
Epoch 28/50
300/300 [==============================] - 49s 162ms/step - discriminator_loss:
0.4691 - generator_loss: 0.4691
Epoch 29/50
300/300 [==============================] - 48s 162ms/step - discriminator_loss:
0.4678 - generator_loss: 0.4678
Epoch 30/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
0.4690 - generator_loss: 0.4690
Epoch 31/50
300/300 [==============================] - 49s 165ms/step - discriminator_loss:
0.4717 - generator_loss: 0.4717
Epoch 32/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4682 - generator_loss: 0.4682
Epoch 33/50
300/300 [==============================] - 50s 167ms/step - discriminator_loss:
0.4699 - generator_loss: 0.4699
Epoch 34/50
300/300 [==============================] - 50s 166ms/step - discriminator_loss:
0.4697 - generator_loss: 0.4697
Epoch 35/50
300/300 [==============================] - 49s 165ms/step - discriminator_loss:
0.4702 - generator_loss: 0.4702
Epoch 36/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4716 - generator_loss: 0.4716
Epoch 37/50
300/300 [==============================] - 50s 168ms/step - discriminator_loss:
0.4668 - generator_loss: 0.4668
Epoch 38/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4683 - generator_loss: 0.4683
Epoch 39/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4706 - generator_loss: 0.4706
Epoch 40/50
300/300 [==============================] - 634s 2s/step - discriminator_loss:
0.4687 - generator_loss: 0.4687
Epoch 41/50
300/300 [==============================] - 52s 174ms/step - discriminator_loss:
0.4701 - generator_loss: 0.4701
Epoch 42/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4685 - generator_loss: 0.4685
Epoch 43/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
```

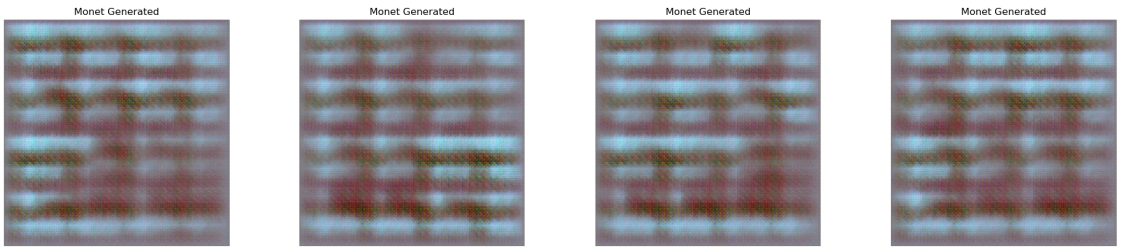
```
0.4671 - generator_loss: 0.4671
Epoch 44/50
300/300 [==============================] - 49s 165ms/step - discriminator_loss:
0.4707 - generator_loss: 0.4707
Epoch 45/50
300/300 [==============================] - 50s 165ms/step - discriminator_loss:
0.4702 - generator_loss: 0.4702
Epoch 46/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
0.4695 - generator_loss: 0.4695
Epoch 47/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
0.4696 - generator_loss: 0.4696
Epoch 48/50
300/300 [==============================] - 49s 163ms/step - discriminator_loss:
0.4690 - generator_loss: 0.4690
Epoch 49/50
300/300 [==============================] - 49s 165ms/step - discriminator_loss:
0.4691 - generator_loss: 0.4691
Epoch 50/50
300/300 [==============================] - 49s 164ms/step - discriminator_loss:
0.4709 - generator_loss: 0.4709
```

[36]: `<keras.src.callbacks.History at 0x298b0b090>`

1.6 Model Performance

[38]:
```
fig, ax = plt.subplots(1, 4, figsize=(25, 5))
for i in range(4):
    prediction = monet_generator(np.random.randn(1, LATENT_DIM),␣
 ↪training=False)[0].numpy()
    prediction = (prediction * 127.5 + 127.5).astype(np.uint8)

    ax[i].imshow(prediction)
    ax[i].set_title("Monet Generated")
    ax[i].axis("off")
plt.show()
```



1.7 Submission

[39]:
```
import PIL
! mkdir ../images

for i in range(10000):
    prediction = monet_generator(np.random.randn(1, LATENT_DIM),␣
 ↪training=False)[0].numpy()
    prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
    im = PIL.Image.fromarray(prediction)
    im.save(f"../images/{i}.jpg")

import shutil
shutil.make_archive("/kaggle/working/images", 'zip', "/kaggle/images")
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[39], line 11
      8     im.save(f"../images/{i}.jpg")
     10 import shutil
---> 11 shutil.make_archive("/kaggle/working/images", 'zip', "/kaggle/images")

File ~/anaconda3/lib/python3.11/shutil.py:1133, in make_archive(base_name,␣
 ↪format, root_dir, base_dir, verbose, dry_run, owner, group, logger)
   1130             os.chdir(root_dir)
   1132 try:
-> 1133     filename = func(base_name, base_dir, **kwargs)
   1134 finally:
   1135     if save_cwd is not None:

File ~/anaconda3/lib/python3.11/shutil.py:985, in _make_zipfile(base_name,␣
 ↪base_dir, verbose, dry_run, logger, owner, group, root_dir)
    983         logger.info("creating %s", archive_dir)
    984     if not dry_run:
--> 985         os.makedirs(archive_dir)
    987 if logger is not None:
    988     logger.info("creating '%s' and adding '%s' to it",
    989                 zip_filename, base_dir)

File <frozen os>:215, in makedirs(name, mode, exist_ok)

File <frozen os>:225, in makedirs(name, mode, exist_ok)
```

```
OSError: [Errno 30] Read-only file system: '/kaggle'
```

1.8 Conclusion

[ ]: