ART & ML Project 2

Creating Anime Faces using DC GAN.

Link to output folder:
https://drive.google.com/drive/folders/1quHOSHkQfyA3saFIVFS0CMRteNBDVjhR?usp=sharing

Link to Dataset:
https://www.kaggle.com/soumikrakshit/anime-faces

CODE:

We used DC GAN to create anime faces. To write code we went through DC GAN Paper and few YouTube tutorials to understand how to write using tensor flow. We wanted to try input shape of 512 x 512 but GPU was crashing so we had to do it with 128 x 128.

Output is generated but we hope to get better output hence I have submitted link to the folder where output will be saved.

We've used Tensorflow because GCP has dedicated TPU to run Tensorflow.

```
# -*- coding: utf-8 -*-
"""ART & ML_Project-2

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/
1MYu_SiEPwisL_odPAbD0IAXSUg4yXEAN

# Pulling Data set from KAGGLE
"""

! pip install kaggle

! mkdir ~/.kaggle

cp kaggle.json ~/.kaggle/

! chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets download soumikrakshit/anime-faces

"""Unzipping downloaded dataset

"""

! unzip anime-faces.zip

"""# MAIN

Import required Libraries (We are using DCGAN in Tensorflow)
"""

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

```
import numpy as np
import matplotlib.pyplot as plt
import os
import gdown
from zipfile import ZipFile

"""We will set memeory growth due to this the memory of a graphics card
will be fully allocated to that process only!"""

physical_devices = tf.config.list_physical_devices("GPU")
tf.config.experimental.set_memory_growth(physical_devices[0], True)

"""We are here setting our directory. we have specified image size and
batch size also we are normalisisng it using lambda."""

dataset = keras.preprocessing.image_dataset_from_directory(
    "/content/data", label_mode=None, image_size=(128, 128),
batch_size=32
)
dataset = dataset.map(lambda x: x / 255.0)

"""Discriminator"""

discriminator = keras.Sequential(
    [
        keras.Input(shape=(128,128, 3)),
        layers.Conv2D(120, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(0.2),
        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(0.2),
        layers.Flatten(),
        layers.Dropout(0.2),
        layers.Dense(1, activation="sigmoid"),
    ],
    name="discriminator",
)
print(discriminator.summary())

"""Generator"""

latent_dim = 128

generator = keras.Sequential(
    [
        keras.Input(shape=(latent_dim,)),
        layers.Dense(8 * 8 * 128),
        layers.Reshape((8, 8, 128)),
        layers.Conv2DTranspose(128, kernel_size=4, strides=2,
padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(256, kernel_size=4, strides=2,
padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(512, kernel_size=4, strides=2,
padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(512, kernel_size=4, strides=2,
padding="same"),
```

```
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(3, kernel_size=5, padding="same",
activation="sigmoid"),
    ],
    name="generator",
)
generator.summary()

"""DCGAN MODEL"""

class GAN(keras.Model):
    def __init__(self, discriminator, generator, latent_dim):
        super(GAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.latent_dim = latent_dim

    def compile(self, d_optimizer, g_optimizer, loss_fn):
        super(GAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.loss_fn = loss_fn
        self.d_loss_metric = keras.metrics.Mean(name="d_loss")
        self.g_loss_metric = keras.metrics.Mean(name="g_loss")

    @property
    def metrics(self):
        return [self.d_loss_metric, self.g_loss_metric]

    def train_step(self, real_images):

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


        generated_images = self.generator(random_latent_vectors)


        combined_images = tf.concat([generated_images, real_images],
axis=0)


        labels = tf.concat(
            [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))], axis=0
        )

        labels += 0.05 * tf.random.uniform(tf.shape(labels))


        with tf.GradientTape() as tape:
            predictions = self.discriminator(combined_images)
            d_loss = self.loss_fn(labels, predictions)
        grads = tape.gradient(d_loss,
self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights)
        )
```

```
        random_latent_vectors = tf.random.normal(shape=(batch_size,
self.latent_dim))


        misleading_labels = tf.zeros((batch_size, 1))


        with tf.GradientTape() as tape:
            predictions =
self.discriminator(self.generator(random_latent_vectors))
            g_loss = self.loss_fn(misleading_labels, predictions)
        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(zip(grads,
self.generator.trainable_weights))


        self.d_loss_metric.update_state(d_loss)
        self.g_loss_metric.update_state(g_loss)
        return {
            "d_loss": self.d_loss_metric.result(),
            "g_loss": self.g_loss_metric.result(),
        }

"""Class to save output of Model"""

class GANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=3, latent_dim=128):
        self.num_img = num_img
        self.latent_dim = latent_dim

    def on_epoch_end(self, epoch, logs=None):
        random_latent_vectors = tf.random.normal(shape=(self.num_img,
self.latent_dim))
        generated_images = self.model.generator(random_latent_vectors)
        generated_images *= 255
        generated_images.numpy()
        for i in range(self.num_img):
            img =
keras.preprocessing.image.array_to_img(generated_images[i])
            img.save("/content/sample_data/Output Folder/
anime_faces_img_%03d_%d.png" % (epoch, i))

"""This is just regular training. We have set it to 100 epohcs I m
thinking of increasing it but I don't have acesss to GPU right now, I
have mailed them but they's haven got back to me."""

epochs = 100  # In practice, use ~100 epochs

gan = GAN(discriminator=discriminator, generator=generator,
latent_dim=latent_dim)
gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss_fn=keras.losses.BinaryCrossentropy(),
)

gan.fit(
    dataset, epochs=epochs, callbacks=[GANMonitor(num_img=10,
latent_dim=latent_dim)]
)
```

References

Dataset: https://www.kaggle.com/soumikrakshit/anime-faces

Tutorial : https://www.tensorflow.org/tutorials/generative/dcgan

DC GAN : https://arxiv.org/abs/1511.06434

Tutorial :