GAN_BETA

December 4, 2022

[ ]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pathlib
import os
import cv2
from sklearn.model_selection import train_test_split
```

[ ]:
```
def Create_Generator():
    gen = tf.keras.Sequential()
    gen.add(tf.keras.layers.Dense(16*16*384, use_bias = False,␣
 ↪input_shape=(100,)))
    gen.add(tf.keras.layers.BatchNormalization())
    gen.add(tf.keras.layers.LeakyReLU(alpha=0.2))

    gen.add(tf.keras.layers.Reshape((16,16,384)))
    gen.add(tf.keras.layers.Conv2DTranspose(192, (5,5), strides=(1,1),␣
 ↪padding='same', use_bias = False))
    gen.add(tf.keras.layers.BatchNormalization())
    gen.add(tf.keras.layers.LeakyReLU(alpha=0.2))

    gen.add(tf.keras.layers.Conv2DTranspose(96, (5,5), strides=(2,2),␣
 ↪padding='same', use_bias = False))
    gen.add(tf.keras.layers.BatchNormalization())
    gen.add(tf.keras.layers.LeakyReLU(alpha=0.2))

    gen.add(tf.keras.layers.Conv2DTranspose(48, (5,5), strides=(2,2),␣
 ↪padding='same', use_bias = False))
    gen.add(tf.keras.layers.BatchNormalization())
    gen.add(tf.keras.layers.LeakyReLU(alpha=0.2))

    gen.add(tf.keras.layers.Conv2DTranspose(3, (5,5), strides=(2,2),␣
 ↪padding='same', use_bias = False, activation='tanh'))
    return gen
```

[ ]:
```
def Create_Discriminator():
    dis = tf.keras.Sequential()
```

```
    dis.add(tf.keras.layers.Conv2D(96, (5,5), strides=(2,2), padding='same'))
    dis.add(tf.keras.layers.LeakyReLU(alpha=0.2))
    dis.add(tf.keras.layers.Dropout(0.3))

    dis.add(tf.keras.layers.Conv2D(192, (5,5), strides=(2,2), padding='same'))
    dis.add(tf.keras.layers.LeakyReLU(alpha=0.2))
    dis.add(tf.keras.layers.Dropout(0.3))

    dis.add(tf.keras.layers.Flatten())
    dis.add(tf.keras.layers.Dense(1))
    return dis
```

[ ]:
```
def make_trainable(dis, flag):
    dis.trainable = flag
    for l in dis.layers:
        l.trainable = flag
    return dis
```

[ ]:
```
def make_noise(n, z):
    return np.random.normal(0, 1, size=(n,z))
```

[ ]:
```
def plot_sample(n,z,Generator, index):
    print("Samples:")
    samples = Generator.predict(make_noise(n,z))
    samples = (samples + 1.0) / 2.0
    plt.figure(figsize=(15,6))
    for i in range(n):
        plt.subplot(1,n, (i+1))
        plt.imshow(samples[i].reshape(128,128,3))
        plt.axis('off')
    plt.savefig("GAN_" + str(index) + "_Epochs_Samples",
                bbox_inches='tight',
                pad_inches = 0.5,
                transparent = False,
                dpi =400)
    plt.show()
```

[ ]:
```
def LoadAndDecodeImage(imagePath):
    rawImageData = tf.io.read_file(imagePath)
    imageData = tf.io.decode_jpeg(rawImageData, channels = 3)
    imageData = tf.image.convert_image_dtype(imageData, tf.float32)
    imageData = tf.image.resize(imageData, (128, 128))
    return imageData
```

[ ]:
```
def processPath(imagePath):
    return LoadAndDecodeImage(imagePath)
```

[ ]:
```
def generator_loss(generated_output):
    return tf.nn.sigmoid_cross_entropy_with_logits(labels = tf.
 ↪ones_like(generated_output), logits = generated_output)
```

[ ]:
```
def discriminator_loss(real_output, generated_output):
    real = tf.nn.sigmoid_cross_entropy_with_logits(labels = tf.
 ↪ones_like(real_output), logits = real_output)
    generated = tf.nn.sigmoid_cross_entropy_with_logits(labels = tf.
 ↪zeros_like(generated_output), logits = generated_output)
    total_loss = real + generated
    return total_loss
```

[ ]:
```
def main():

    path = os.getcwd() + "/Samples/img_align_celeba/"
    dir_list = os.listdir(path)
    for i in range(len(dir_list)):
        dir_list[i] = path + dir_list[i]

    ds_file_path = np.array(dir_list)
    trainFiles, testFiles = train_test_split(ds_file_path, train_size=0.8,␣
 ↪random_state=42)
    DS = tf.data.Dataset.from_tensor_slices(trainFiles)
    VDS = tf.data.Dataset.from_tensor_slices(testFiles)

    AUTOTUNE = tf.data.experimental.AUTOTUNE
    train_ds = DS.map(LoadAndDecodeImage, num_parallel_calls=AUTOTUNE)
    test_ds = DS.map(LoadAndDecodeImage, num_parallel_calls=AUTOTUNE)

    train_np = tfds.as_numpy(train_ds)
    test_np = tfds.as_numpy(test_ds)

    train_list = []
    test_list = []
    for images in train_np:
        train_list.append(images)

    for images in test_np:
        test_list.append(images)

    X_train = np.array(train_list)
    X_test = np.array(test_list)

    epochs = 300
    batch_size = 64
    input_dim = 100
```

```
  Generator = Create_Generator()
  Discriminator = Create_Discriminator()

  generator_optimizer = tf.optimizers.Adam(1e-4)
  discriminator_optimizer = tf.optimizers.Adam(1e-4)
  batch_no = int(len(X_train)/batch_size)

  np.random.seed(42)

  for i in range(0, epochs):
      for j in range(batch_no):

          rand_sample = np.random.randint(0, len(X_train), size=batch_size)

          image_batch = X_test[rand_sample]

          input_noise = make_noise(batch_size, input_dim)

          with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:

              generated_images = Generator(input_noise, training=True)
              real_output = Discriminator(image_batch, training=True)
              generated_output = Discriminator(generated_images,␣
↪training=True)

              Generator_Loss = generator_loss(generated_output)
              Discriminator_Loss = discriminator_loss(real_output,␣
↪generated_output)


          gradient_of_generator = gen_tape.gradient(Generator_Loss, Generator.
↪trainable_variables)
          generator_optimizer.apply_gradients(zip(gradient_of_generator,␣
↪Generator.trainable_variables))

          gradient_of_discriminator = disc_tape.gradient(Discriminator_Loss,␣
↪Discriminator.trainable_variables)
          discriminator_optimizer.
↪apply_gradients(zip(gradient_of_discriminator, Discriminator.
↪trainable_variables))

      if i % 10 == 0:
          print("Epoch %i" % i)
          plot_sample(5, input_dim, Generator, i)
```

```
    baseline_model_accuracy = Discriminator.evaluate(generated_images,␣
↪image_batch, verbose=0)

    print('Baseline test accuracy:', baseline_model_accuracy)
    tf.keras.models.save_model(Generator, '.h5', include_optimizer=True)
```

[ ]:

[ ]:
```
if __name__ == '__main__':
    main()
```

[ ]: