img_gen

November 18, 2022

[ ]:

```
import tensorflow as tf
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    tf.config.set_logical_device_configuration(gpus[0], [tf.config.
 ↪LogicalDeviceConfiguration(memory_limit=5292)])

import keras
import matplotlib.pyplot as plt
import numpy as np
from keras import layers, optimizers, losses, metrics, Model
from PIL import Image
from sklearn import preprocessing as pre
```

[ ]:

```
# Construct the discriminator.
def create_D():
    input_img = keras.Input(shape=(256,256, 1))

    conv1 = layers.Conv2D(8, (2,2), (2,2), padding='valid',
                            activation=layers.ReLU())(input_img)
    drop1 = layers.Dropout(rate=0.5)(conv1)

    down1 = layers.MaxPool2D(pool_size=(2,2), strides=(2,2),␣
 ↪padding="valid")(drop1)

    flat1 = layers.Flatten()(down1)

    score = layers.Dense(1, activation='sigmoid')(flat1)

    D: Model = Model(inputs=input_img, outputs=score)
    D.compile(optimizer=optimizers.Nadam(learning_rate=0.002), loss=losses.
 ↪BinaryCrossentropy(), metrics=metrics.MeanAbsoluteError())
    return D
```

[ ]:

```
# Construct the generator.
def create_G():
    random_input = keras.Input(shape=(100, ))
    Dense1 = layers.Dense(units = 16*16)(random_input)

    B_Norm1 = layers.BatchNormalization()(Dense1)

    Relu1 = layers.LeakyReLU()(B_Norm1)

    reshape1 = layers.Reshape(target_shape=(16, 16, 1))(Relu1)

    DeConv1 = layers.Conv2DTranspose(filters=4, kernel_size=(2, 2),␣
 ↪strides=(1,1), padding='same')(reshape1)

    B_Norm2 = layers.BatchNormalization()(DeConv1)

    Relu2 = layers.LeakyReLU()(B_Norm2)

    DeConv2 = layers.Conv2DTranspose(filters=16, kernel_size=(3,3),␣
 ↪strides=(1,1), padding='same')(Relu2)

    re = layers.Reshape((16,16,16,1))(DeConv2)

    maxi0 = layers.MaxPool3D(pool_size=(4,4,2))(re)

    B_Norm3 = layers.BatchNormalization()(maxi0)

    Relu3 = layers.LeakyReLU()(B_Norm3)

    flat0 = layers.Flatten()(Relu3)

    dense00 = layers.Dense(units=256*256, activation='tanh')(flat0)

    output_img = layers.Reshape((256, 256, 1))(dense00)

    G = Model(inputs=random_input, outputs=output_img)
    return G
```

[ ]:

```
# Join the discriminator and generator, forming the final GAN model.
def create_GAN():
    D = create_D()
    G = create_G()
    latent_input = keras.Input(shape=(100, ))
    img = G(latent_input)
    D.trainable = False
    score = D(img)
    GAN = Model(inputs = latent_input, outputs=score)
    GAN.compile(loss=losses.BinaryCrossentropy(),
            optimizer=optimizers.Nadam(learning_rate=0.002), metrics=metrics.
 ↪MeanAbsoluteError())
```

```
    return D, G, GAN
```

[ ]:

```
# Function returns a sample of latent points from a normal distribution.
def getLatentSamples(num_samples) -> np.ndarray:
    latent = np.random.normal(size=100*int(num_samples))
    latent = np.reshape(a=latent, newshape=(int(num_samples), 100))
    return latent


# Function imputs latent data into the generator and returns the fake image␣
 ↪samples outputted
# by the generator.
def getRealSamples(batch_size: int, train_img: np.ndarray):
    X_real, y_real = np.array(train_img.tolist() * batch_size, dtype=np.
 ↪float32).reshape((batch_size, train_img.shape[0], train_img.shape[1])), np.
 ↪ones(shape=batch_size, dtype=np.int8)
    assert X_real.shape == (batch_size, train_img.shape[0], train_img.shape[1])
    assert y_real.shape == (batch_size, )
    return X_real, y_real

# Function imputs latent data into the generator and returns the fake image␣
 ↪samples outputted
# by the generator.
def getFakeSamples(G:Model, batch_size:int):
    latent = getLatentSamples(batch_size)
    assert latent.shape == (batch_size, 100)
    fake_imgs = G(latent)
    fake_imgs = np.asarray(fake_imgs, dtype=np.float32)
    assert fake_imgs.shape[0] == batch_size
    fake_imgs = np.squeeze(fake_imgs)
    output_labels = np.zeros(shape=batch_size, dtype=np.int8)
    assert output_labels.shape == (batch_size, )
    return fake_imgs, output_labels
```

[ ]:

```
# Function plots a sample of 16 'predicted' images from the trained generator.
def pred_plot(G: Model, epoch:int, train_img_shape: tuple[int, int]):
    shape = train_img_shape
    latent = getLatentSamples(1)
    gen_image = G(latent)
    gen_image = np.asarray(gen_image, dtype=np.float32)
    gen_image = np.squeeze(gen_image)
    assert gen_image.shape == shape
    filename = str('images/gen3-%d.png' %epoch)
    plt.imsave(filename, gen_image, cmap='gray')
    if epoch % 100 == 0:
        im=Image.open(filename)
        im.load()
        im.convert('L').show()
```

[ ]:

```
# Function trains the GAN (generator) model.
def trainGAN(D:Model, G:Model, GAN:Model, num_epochs: int, batchSize: int, train_img: np.ndarray, train_img_shape: tuple[int, int]):
    d_loss, g_loss = [], []
    d_mae, g_mae = [], []
    pred_plot(G, 0, train_img_shape)
    for epoch in range(1, num_epochs+1):
        X_real, y_real = getRealSamples(int(batchSize/2), train_img)
        X_fake, y_fake = getFakeSamples(G, int(batchSize / 2))
        assert X_real.shape == X_fake.shape
        assert X_real.shape[0] == int(batchSize / 2)
        assert y_real.shape == y_fake.shape
        assert y_real.shape[0] == int(batchSize / 2)
        D.trainable = True
        d_loss_fake, mae_fake = D.train_on_batch(X_fake, y_fake)
        d_loss_real, mae_real = D.train_on_batch(X_real, y_real)
        D.trainable = False
        X_latent, y_latent = getLatentSamples(batchSize), np.ones(shape=batchSize, dtype=np.int8)
        loss, mae = GAN.train_on_batch(X_latent, y_latent)

        d_loss_res = float((d_loss_fake + d_loss_real) / 2)
        d_mae_res = float((mae_fake + mae_real) / 2)
        d_loss.append(d_loss_res)
        d_mae.append(d_mae_res)
        g_loss.append(loss)
        g_mae.append(mae)
        if (epoch % 50 == 0) and epoch != 0:
            pred_plot(G, epoch, train_img_shape)
            models = (D, G, GAN)
            filepaths = ('D_final.h5', 'G_final.h5', 'GAN_final.h5')
            for model, name in zip(models, filepaths):
                model.save(name)
            print('Epoch: %d\nDiscriminator Loss: %.3f, Generator Loss: %.3f\nDiscriminator MAE: %.3f, Generator MAE: %.3f' % (epoch, d_loss[-1], g_loss[-1], d_mae[-1], g_mae[-1]))

    fig, ax = plt.subplots(2,2, figsize=(11,11))
    ax[0][0].plot(list(range(len(d_loss))), d_loss, label='D_loss', color='red', linestyle='dashed', linewidth=1)
    ax[0][1].plot(list(range(len(g_loss))), g_loss, label='G_loss', color='red', linestyle='solid', linewidth=1)
    ax[1][0].plot(list(range(len(d_mae))), d_mae, label='D_mae', color='blue', linestyle='dashed', linewidth=1)
    ax[1][1].plot(list(range(len(g_mae))), g_mae, label='G_mae', color='blue',␣
 ↪linestyle='solid', linewidth=1)
    fig.legend()
    fig.tight_layout()
    plt.savefig("loss_plots")
    plt.show()
```

[ ]:

```
img = Image.open("orig_img_color.png")
img.load()
img = img.convert(mode='RGB').convert(mode='L')
img = np.asarray(img, dtype=np.float32)
try:
    assert img.shape == (256, 256)
except:
    img = np.reshape(a=img, newshape=(256, 256))
training_img = pre.minmax_scale(img.flatten(), (-1,1))
training_img = training_img.reshape(img.shape)

D, G, GAN = create_GAN()

def show_summary(D, G, GAN):
    D.summary()
    G.summary()
    GAN.summary()

verbose = input('Enter T/F for verbose output: ')
if verbose == 'T':
    show_summary(D, G, GAN)

trainGAN(D, G, GAN, 10000, 16, training_img, training_img.shape)
```