Image_Generation

October 20, 2022

[ ]:
```
import tensorflow as tf
from tensorflow import keras
from keras import datasets, layers, models, losses, Model
from keras.models import load_model
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.axis import Axis
import numpy as np
import random
import PIL
from PIL import Image
```

[ ]:
```
# Load data, we only need the training images.
from keras.datasets.mnist import load_data
(X_train, _), (_, _) = load_data()
print('Train Shape: ', X_train.shape)
print(X_train[0].shape)
```

[ ]:
```
# Reshape the images for the CNN.
X_train = tf.expand_dims(X_train, axis=3)
print('Train Shape: ', X_train.shape)
print(X_train[0].shape)
```

[ ]:
```
# Standardize the pixel values
X_train = X_train.numpy().astype('float32') / 255.0
```

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

conv1 = layers.Conv2D(16, (7,7), (2,2), padding='same',
                              activation=layers.LeakyReLU(alpha=0.2))(input_img)
drop1 = layers.Dropout(rate=0.30)(conv1)

conv2 = layers.Conv2D(32, (5,5), (2,2), padding='same',
                              activation=layers.LeakyReLU(alpha=0.2))(drop1)
drop2 = layers.Dropout(rate=0.35)(conv2)

conv3 = layers.Conv2D(64, (5,5), (2,2), padding='same',
```

```
                                 activation=layers.LeakyReLU(alpha=0.2))(drop2)
drop3 = layers.Dropout(rate=0.40)(conv3)

conv4 = layers.Conv2D(128, (5,5), (2,2), padding='same',
                                 activation=layers.LeakyReLU(alpha=0.2))(drop3)
drop4 = layers.Dropout(rate=0.45)(conv4)

flat1 = layers.Flatten()(drop4)

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

D = keras.Model(inputs=input_img, outputs=score)

D.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0004),
                                 loss='binary_crossentropy', metrics=['accuracy'])
```

[ ]:
```
D.summary()
```

[ ]:
```
# Construct the generator.
random_input = keras.Input(shape=200)

Dense1 = layers.Dense(units = 32*32*32)(random_input)

B_Norm1 = layers.BatchNormalization()(Dense1)

Relu1 = layers.Activation('relu')(B_Norm1)

Reshape1 = layers.Reshape((32,32,32))(Relu1)

Drop1 = layers.Dropout(0.35)(Reshape1)

Up1 = layers.UpSampling2D((2, 2))(Drop1)

DeConv1 = layers.Conv2DTranspose(filters=32, kernel_size=(7, 7), strides=1,␣
 ↪padding='same')(Up1)

B_Norm2 = layers.BatchNormalization()(DeConv1)

Relu2 = layers.Activation('relu')(B_Norm2)

Up2 = layers.UpSampling2D((2, 2))(Relu2)

DeConv2 = layers.Conv2DTranspose(filters=16, kernel_size=(7, 7), strides=1,␣
 ↪padding='same')(Up2)

B_Norm3 = layers.BatchNormalization()(DeConv2)
```

```
Relu3 = layers.Activation('relu')(B_Norm3)

Up3 = layers.UpSampling2D((2, 2))(Relu3)

DeConv3 = layers.Conv2DTranspose(filters=8, kernel_size=(7, 7), strides=1,␣
  ↪padding='same')(Up3)

B_Norm4 = layers.BatchNormalization()(DeConv3)

Relu4 = layers.Activation('relu')(B_Norm4)

output_img = layers.Conv2DTranspose(filters=1, kernel_size=(5, 5), strides=1,␣
  ↪padding='same',
                                                                     ␣
  ↪activation='sigmoid')(Relu4)

G = keras.Model(inputs=random_input, outputs=output_img)
```

[ ]:
```
G.summary()
```

[ ]:
```
# Join the discriminator and generator, forming the final GAN model.
latent_input = keras.Input(shape=200)

gen = G

disc = D

img = gen(latent_input)

disc.trainable = False

score = disc(img)

GAN = keras.Model(inputs = latent_input, outputs=score)

GAN.compile(loss='binary_crossentropy', optimizer=keras.optimizers.
  ↪Adam(learning_rate=0.0004), metrics=['accuracy'])
```

[ ]:
```
GAN.summary()
```

[ ]:
```
# Function retrieves a random sample of real images from the training data.
def getRealSamples(batchSizeIn):
    indices = random.sample(range(X_train.shape[0]), k=int(batchSizeIn))
    X_real = X_train[indices]
    y_real = np.ones((int(batchSizeIn), 1))
    return X_real, y_real
```

```
[ ]: # Function returns a sample of latent points from a normal distribution.
def getLatentSamples(num_samples):
    latent = np.random.normal(size=(200*int(num_samples)))
    latent = np.reshape(latent, (int(num_samples), 200))
    labels = np.ones((int(num_samples), 1))
    return latent, labels
```

```
[ ]: # Function imputs latent data into the generator and returns the fake image␣
 ↪samples outputted
# by the generator.
def getFakeSamples(num_samples, Gin):
    num_samples = int(num_samples)
    latent_input, _ = getLatentSamples(num_samples)
    output_img = Gin.predict(latent_input)
    output_labels = np.zeros((num_samples, 1))
    return output_img, output_labels
```

```
[ ]: # Function plots a sample of 16 'predicted' images from the trained generator.
def pred_plot(Gin):
    x, _ = getLatentSamples(1)
    gen_images = Gin.predict(x)
    plt.figure(figsize=(6, 6))
    for i in range(1):
        ax = plt.subplot()
        plt.imshow(gen_images.reshape((256,256,1)), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    plt.show()
```

```
[ ]: # Function plots the networks loss values with respect to epoch.
def loss_plot(dLossIn, gLossIn):
    fig, ax = plt.subplots(figsize=(12, 8))
    ax.plot(dLossIn, label='Discriminator Loss')
    ax.plot(gLossIn, label='GAN (generator) Loss')
    ax.legend()
    ax.set_xlim(xmin=-5, xmax=305)
    ax.set_ylim(ymin=-0.2, ymax=1.2)
    plt.xlabel('Epoch')
    plt.ylabel('Loss value')
    plt.title("Loss value vs Epoch for Generator and Discriminator")
    plt.show()
```

```
[ ]: # Lists to store the models loss values at the end of each epoch
dLoss = list()
gLoss = list()
img = Image.open("C:\\Users\\ltnoo\\Desktop\\final.jpg")
img.load()
```

```
img = img.convert(mode='L')
img = np.asarray(img).reshape(256,256,1)
img = img.astype("float32") / 255.0
# Function trains the GAN (generator) model.
def trainGAN(G, D, GAN, num_epochs, batchSize):
    d_loss = float()
    g_loss = float()
    batches_per_epoch = 128
    for epoch in range(num_epochs):
        for batch in range(batches_per_epoch):
            #X_real, y_real = getRealSamples(batchSize / 2)
            X_real, y_real = [img] * int(batchSize / 2), np.ones((int(batchSize␣
↪/ 2), 1))

            X_fake, y_fake = getFakeSamples(batchSize / 2, G)
            X_combined, y_combined = np.vstack((X_real, X_fake)), np.
↪vstack((y_real, y_fake))

            d_loss, _ = D.train_on_batch(X_combined, y_combined)
            X_latent, y_latent = getLatentSamples(batchSize)
            g_loss, _ = GAN.train_on_batch(X_latent, y_latent)

            if (batch % 64 == 0) | (batch == 0):
                print('Epoch %d: , %d/%d, d_loss = %.3f, g_loss = %.3f' %␣
↪(epoch+1, batch+1,
                                                                batches_per_epoch,␣
↪d_loss, g_loss))
        #dLoss.append(d_loss)
        #gLoss.append(g_loss)

        if (epoch % 2 == 0) | (epoch == 0):
            pred_plot(G)
            #filename = 'generator_model_%03d.h5' % (epoch + 1)
            #GAN.save(filename)
            G.save('Generator.h5')
```

[ ]:
```
trainGAN(load_model('Generator.h5'), D, GAN, 13, 128)
```

[ ]:
```
g = load_model('Generator.h5')
g.compile()
```

[ ]:
```
x, _ = getLatentSamples(1)
pred = g.predict(x)
pred = pred.reshape((256,256,1))
```

```
1/1 [==============================] - 6s 6s/step
```

[ ]: (256, 256, 1)

[ ]:
```
original_img = Image.open("train_img.JPG")
converted = original_img.convert(mode="1")
fig, ax = plt.subplots(1,2,figsize=(7,7))
ax[0].imshow(pred, cmap='gray')
ax[1].imshow(converted, cmap='gray')
plt.show()
```