GAN Example

GAN Example

Recap

In our previous discussion, we delved into the fascinating world of generative adversarial networks (GANs). These powerful models consist of two neural networks working in tandem: the generator and the discriminator. The generator's primary function is to create synthetic outputs, while the discriminator's role is to distinguish between the generated (fake) outputs and the real ones. This adversarial relationship is at the heart of GANs, where the generator aims to maximize the discriminator's loss by generating increasingly convincing outputs, and the discriminator strives to minimize its loss by accurately identifying real and fake samples.

We also explored the mechanics behind GANs, including the calculation of loss for each network and the process of updating their parameters after each training iteration (epoch). However, as we discussed, training these models can be a challenging endeavor, often requiring careful hyperparameter tuning and a deep understanding of the underlying principles.

In this presentation, we will dive into the practical aspects of building a simple GAN using Python.

GAN Example

Example

We know that training GANs can become increasingly complex when attempting to generate high-quality images, especially when dealing with large and diverse datasets. In this example, however, we will focus on the fundamental concepts and principles of GANs. To avoid unnecessary complications, we will use a simple dataset consisting of 28x28 grayscale images of apples.
By breaking down the process into some key components, we can systematically build our GAN model and gain a solid understanding of how each part contributes to the overall performance and functionality of the system.

Our GAN model will consist of the following steps:

1. Loading and preprocessing the apple image dataset

2. Defining the generator network architecture

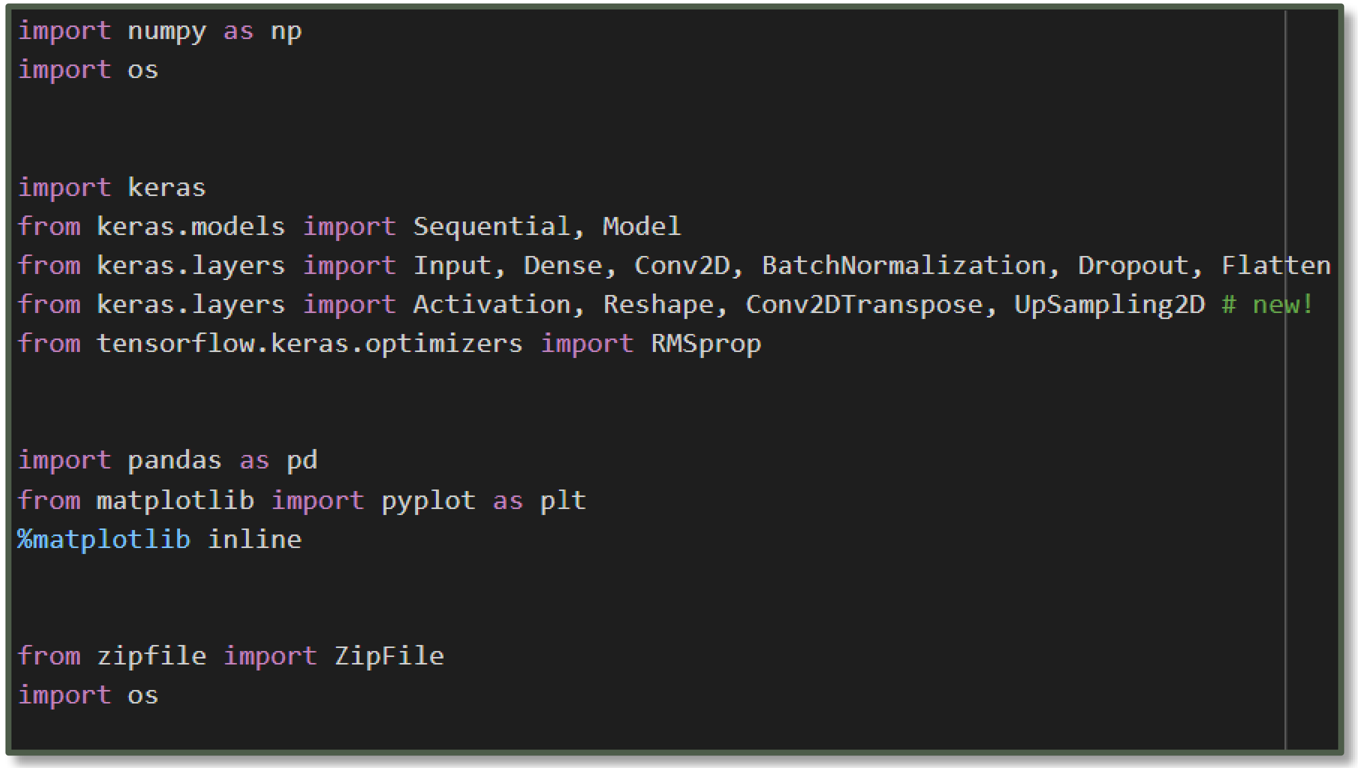
3. Defining the discriminator network architecture

4. Implementing the training process, including updating the parameters of both networks

GAN Example

Import the Modules

To begin our project, we first need to import the necessary modules that we will use throughout the implementation. We will utilize familiar libraries such as NumPy and Keras, which we have employed in most of our previous examples. Additionally, in this example, we will introduce two new modules.
The first new module is Conv2DTranspose, which generates a transposed convolutional layer. This layer will play a crucial role in our generator network.
Since our dataset is zipped, we will need to unzip it before we can use it. To accomplish this task, we will utilize the ZipFile module.



```
import numpy as np
import os


import keras
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Conv2D, BatchNormalization, Dropout, Flatten
from keras.layers import Activation, Reshape, Conv2DTranspose, UpSampling2D # new!
from tensorflow.keras.optimizers import RMSprop


import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline


from zipfile import ZipFile
import os
```

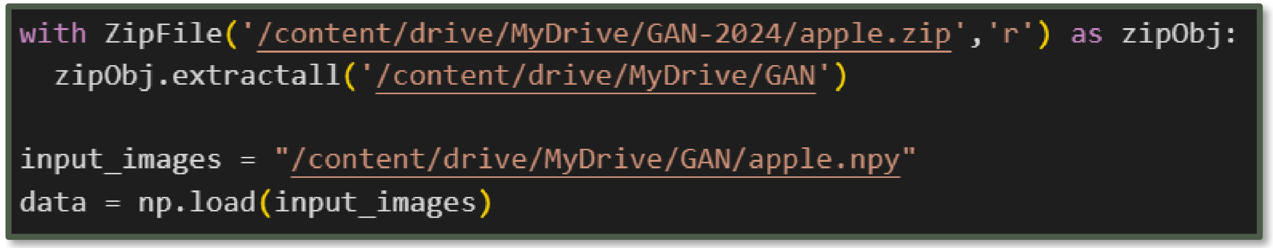
GAN Example

Load the Dataset

To load our dataset, we first need to unzip the compressed file. We will utilize the ZipFile module from the Python standard library for this purpose.

After extracting the dataset, we can define the path to the GAN folder and load the data using NumPy.

By following these steps, we have successfully unzipped the dataset and loaded the apple images into memory, ready for further preprocessing and training our GAN model.

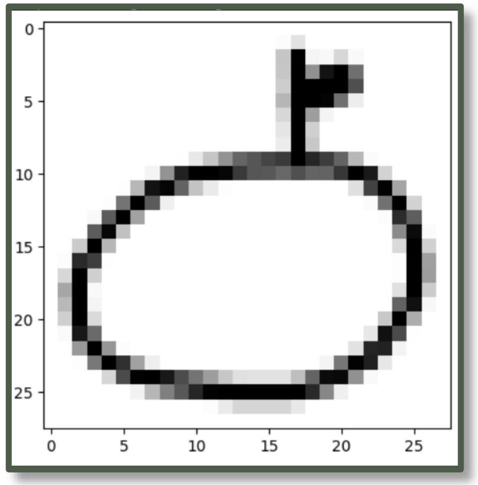


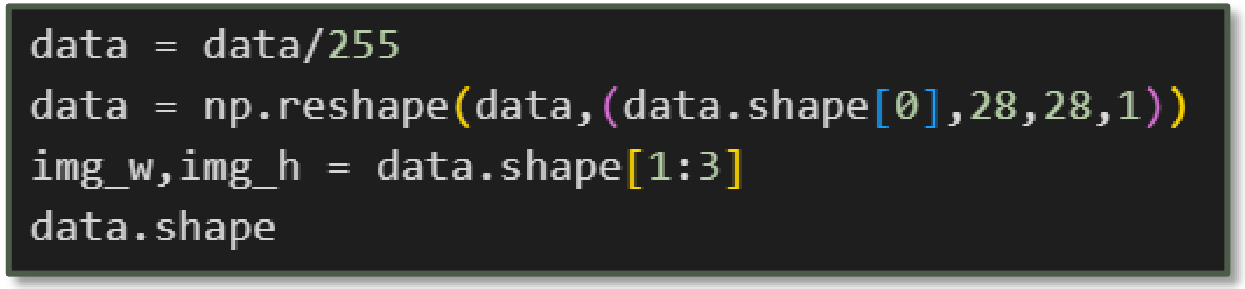
```
with ZipFile('/content/drive/MyDrive/GAN-2024/apple.zip','r') as zipObj:
  zipObj.extractall('/content/drive/MyDrive/GAN')

input_images = "/content/drive/MyDrive/GAN/apple.npy"
data = np.load(input_images)
```

GAN Example

Preprocess the Data

Before building an training the model we will preprocess our data. We will divide each pixel's value by 255 to scale the numbers, ensuring that every pixel has a value between 0 and 1. Next, we will reshape the data into the correct format so that our discriminator can process it effectively. Finally, to ensure everything has gone smoothly, we will print the shape of the dataset and plot one sample from it.



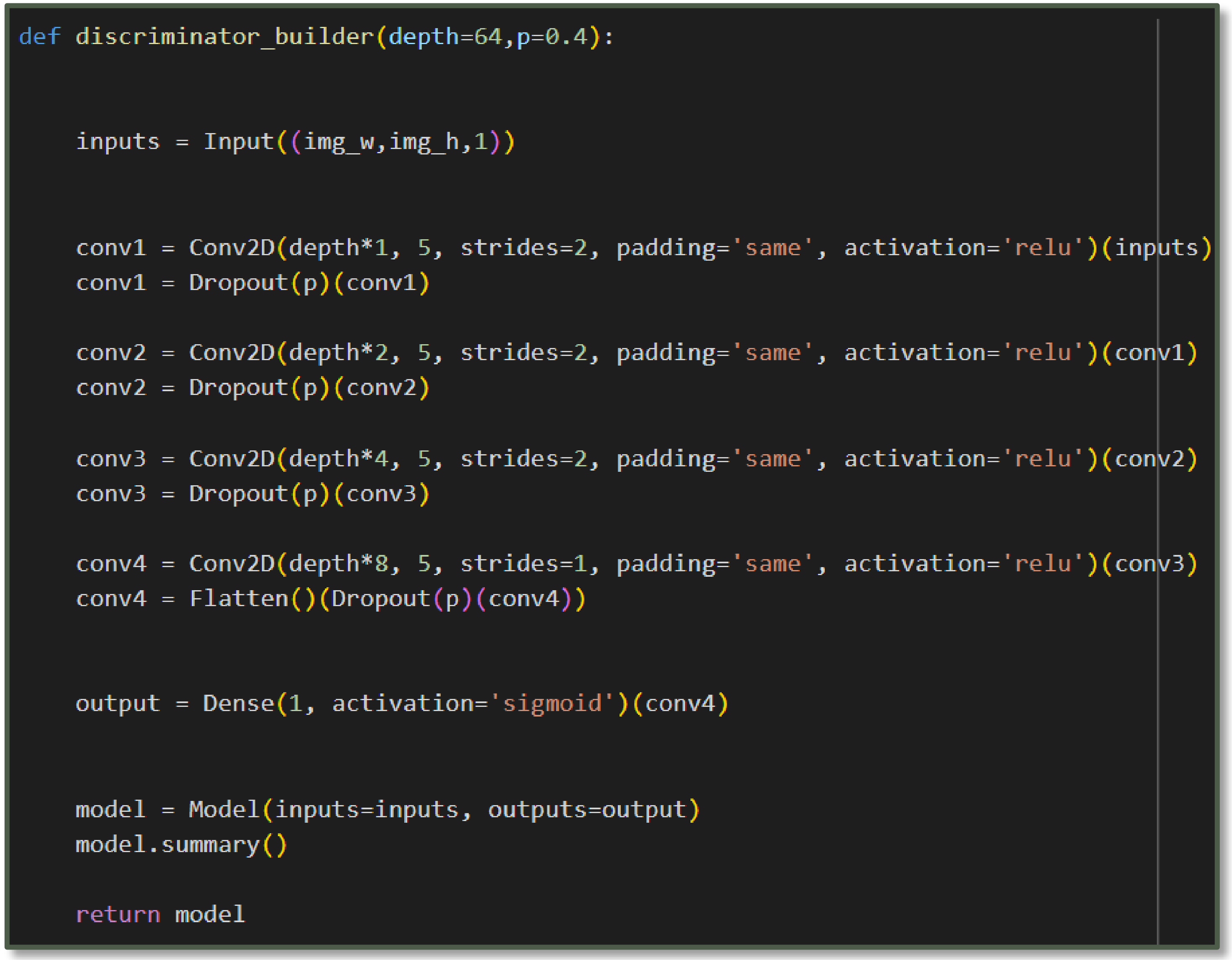


```
data = data/255
data = np.reshape(data,(data.shape[0],28,28,1))
img_w,img_h = data.shape[1:3]
data.shape
```

GAN Example

Build the Discriminator

We will first define the discriminator network by creating a function that returns the model. This network will be a simple Convolutional Neural Network (CNN), and we have previously covered how to define CNNs using the Conv2D and MaxPooling layers.
In our example, we will build a discriminator with four convolutional layers and one fully connected layer. The first three convolutional layers will have a stride of 2, while the last convolutional layer will have a stride of 1. The fully connected layer will consist of one neuron with a sigmoid activation function to generate the output for this binary classification problem.
Finally, we will compile the model and return it.

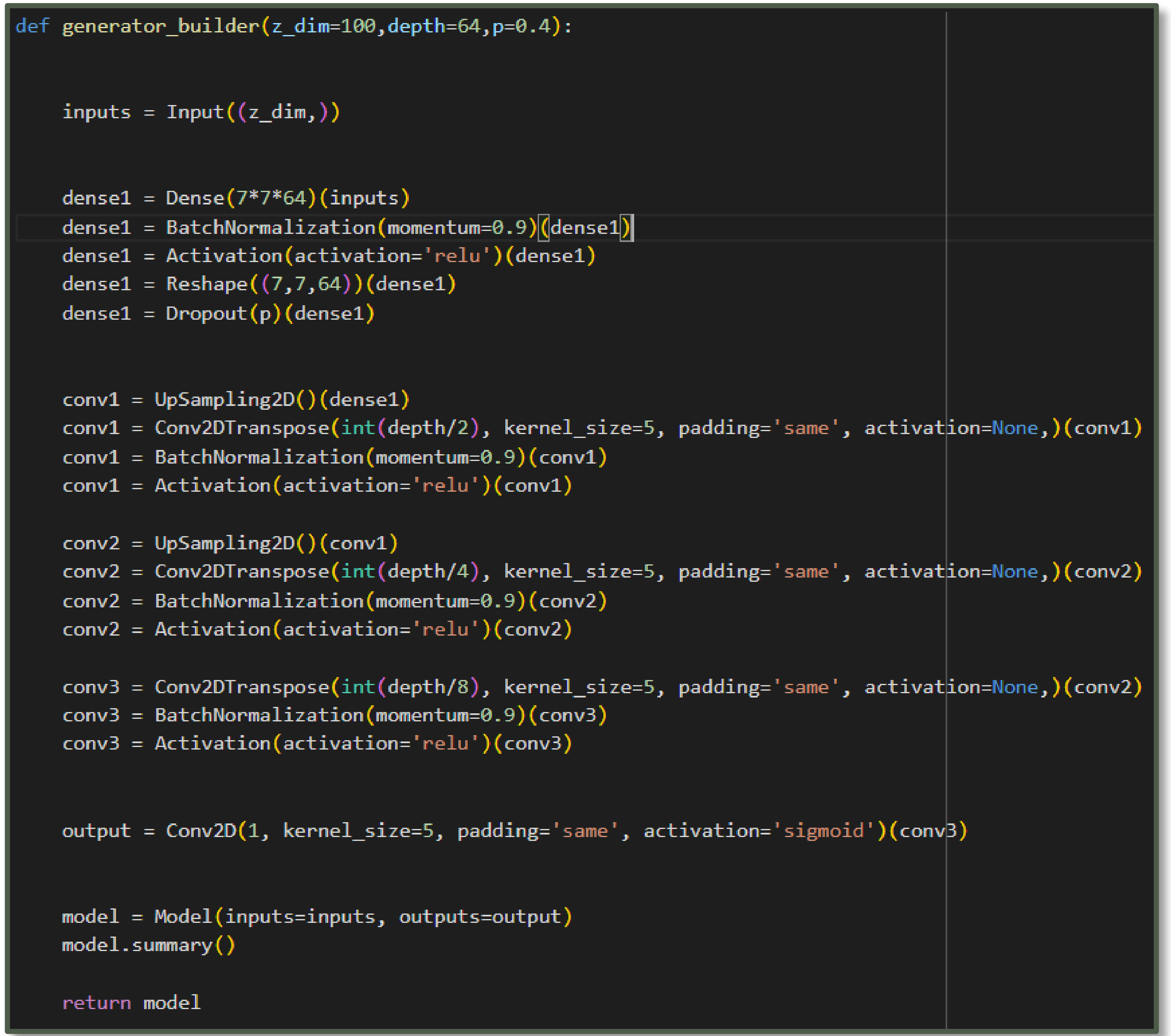


```
def discriminator_builder(depth=64,p=0.4):


    inputs = Input((img_w,img_h,1))


    conv1 = Conv2D(depth*1, 5, strides=2, padding='same', activation='relu')(inputs)
    conv1 = Dropout(p)(conv1)

    conv2 = Conv2D(depth*2, 5, strides=2, padding='same', activation='relu')(conv1)
    conv2 = Dropout(p)(conv2)

    conv3 = Conv2D(depth*4, 5, strides=2, padding='same', activation='relu')(conv2)
    conv3 = Dropout(p)(conv3)

    conv4 = Conv2D(depth*8, 5, strides=1, padding='same', activation='relu')(conv3)
    conv4 = Flatten()(Dropout(p)(conv4))


    output = Dense(1, activation='sigmoid')(conv4)


    model = Model(inputs=inputs, outputs=output)
    model.summary()

    return model
```

GAN Example

Build the Generator

Now, we will define the generator. This network takes a vector of random numbers as its input and generates a 28x28 image from it. In this example, we will use a network architecture that includes one dense layer, three transposed convolutional layers, and one convolutional layer. We define the transposed convolutional layers using the Conv2DTranspose layer in Keras.
We will use the ReLU activation function for all layers except for the final layer, where we will use the sigmoid activation function to generate the final pixel values. Additionally, we will apply batch normalization as a regularization method to improve training stability.
Finally, we will compile the model and return it

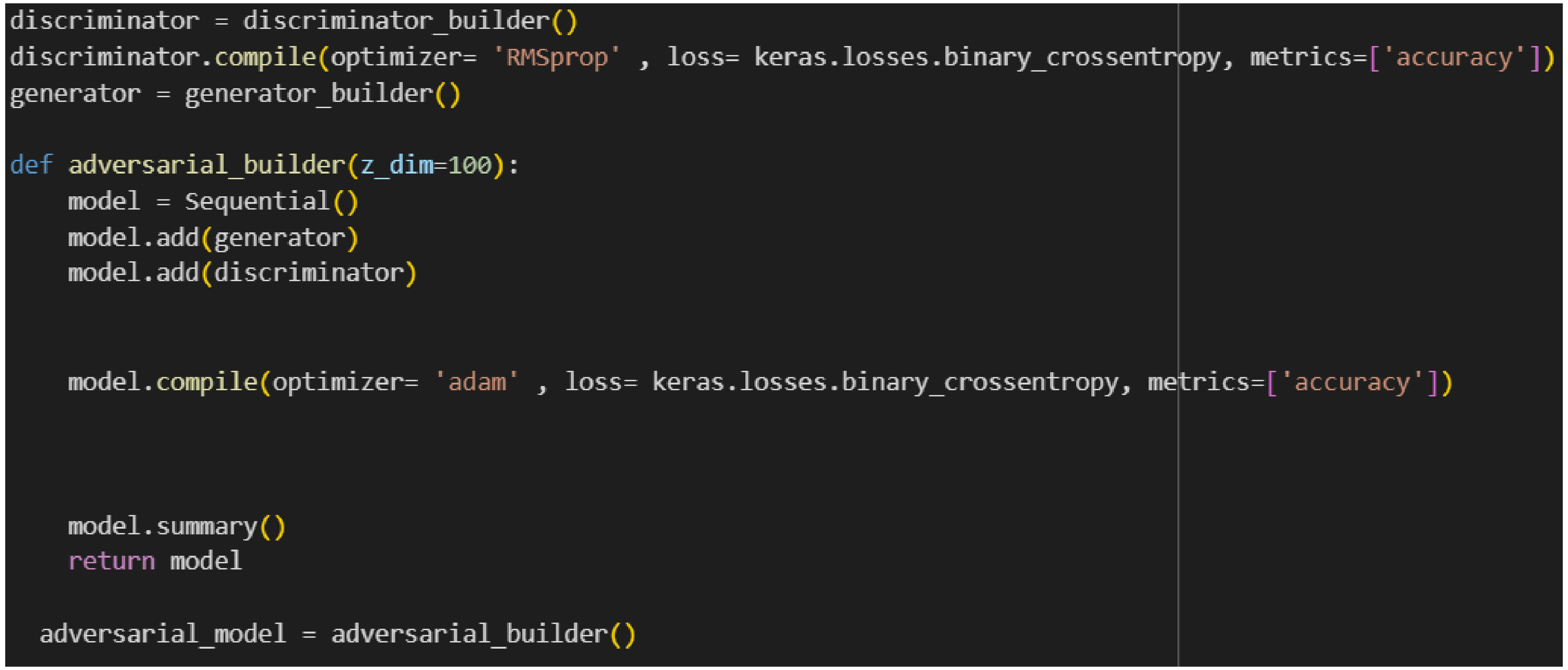


```
def generator_builder(z_dim=100,depth=64,p=0.4):


    inputs = Input((z_dim,))


    dense1 = Dense(7*7*64)(inputs)
    dense1 = BatchNormalization(momentum=0.9)(dense1)
    dense1 = Activation(activation='relu')(dense1)
    dense1 = Reshape((7,7,64))(dense1)
    dense1 = Dropout(p)(dense1)


    conv1 = UpSampling2D()(dense1)
    conv1 = Conv2DTranspose(int(depth/2), kernel_size=5, padding='same', activation=None,)(conv1)
    conv1 = BatchNormalization(momentum=0.9)(conv1)
    conv1 = Activation(activation='relu')(conv1)

    conv2 = UpSampling2D()(conv1)
    conv2 = Conv2DTranspose(int(depth/4), kernel_size=5, padding='same', activation=None,)(conv2)
    conv2 = BatchNormalization(momentum=0.9)(conv2)
    conv2 = Activation(activation='relu')(conv2)

    conv3 = Conv2DTranspose(int(depth/8), kernel_size=5, padding='same', activation=None,)(conv2)
    conv3 = BatchNormalization(momentum=0.9)(conv3)
    conv3 = Activation(activation='relu')(conv3)


    output = Conv2D(1, kernel_size=5, padding='same', activation='sigmoid')(conv3)


    model = Model(inputs=inputs, outputs=output)
    model.summary()

    return model
```

GAN Example

Build the GAN

Now that we have defined the generator and discriminator networks, we can combine them to form the final adversarial network. First, we will create instances of the previously defined models. Then, we will define a new sequential model. We will add the generator and the discriminator to this new model and compile it. Finally, we will return this new model.
Next, we will create an instantiation of the adversarial network using the function we defined.

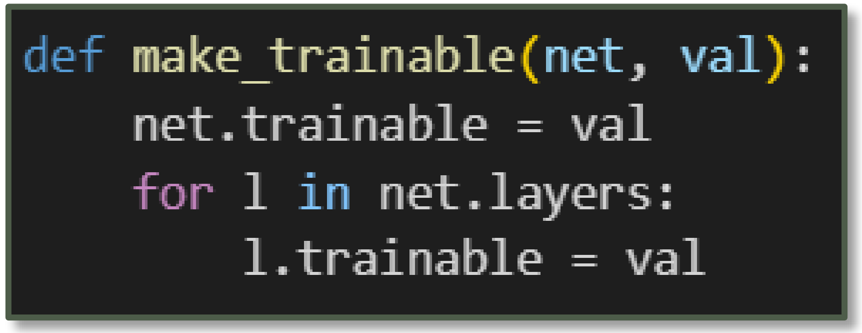


```
discriminator = discriminator_builder()
discriminator.compile(optimizer= 'RMSprop' , loss= keras.losses.binary_crossentropy, metrics=['accuracy'])
generator = generator_builder()

def adversarial_builder(z_dim=100):
    model = Sequential()
    model.add(generator)
    model.add(discriminator)


    model.compile(optimizer= 'adam' , loss= keras.losses.binary_crossentropy, metrics=['accuracy'])



    model.summary()
    return model

  adversarial_model = adversarial_builder()
```

GAN Example

Freeze layers

When we want to train our adversarial network, we must train both the generator and the discriminator. As discussed earlier, the training processes of the networks are independent of each other. This means that when we are training our generator network, the parameters of the discriminator should not be updated.
To manage this, we will define a function that takes a network and a boolean value that determines whether that network should be trainable. Then, we will change the trainable property of all the layers in that network to the desired value.

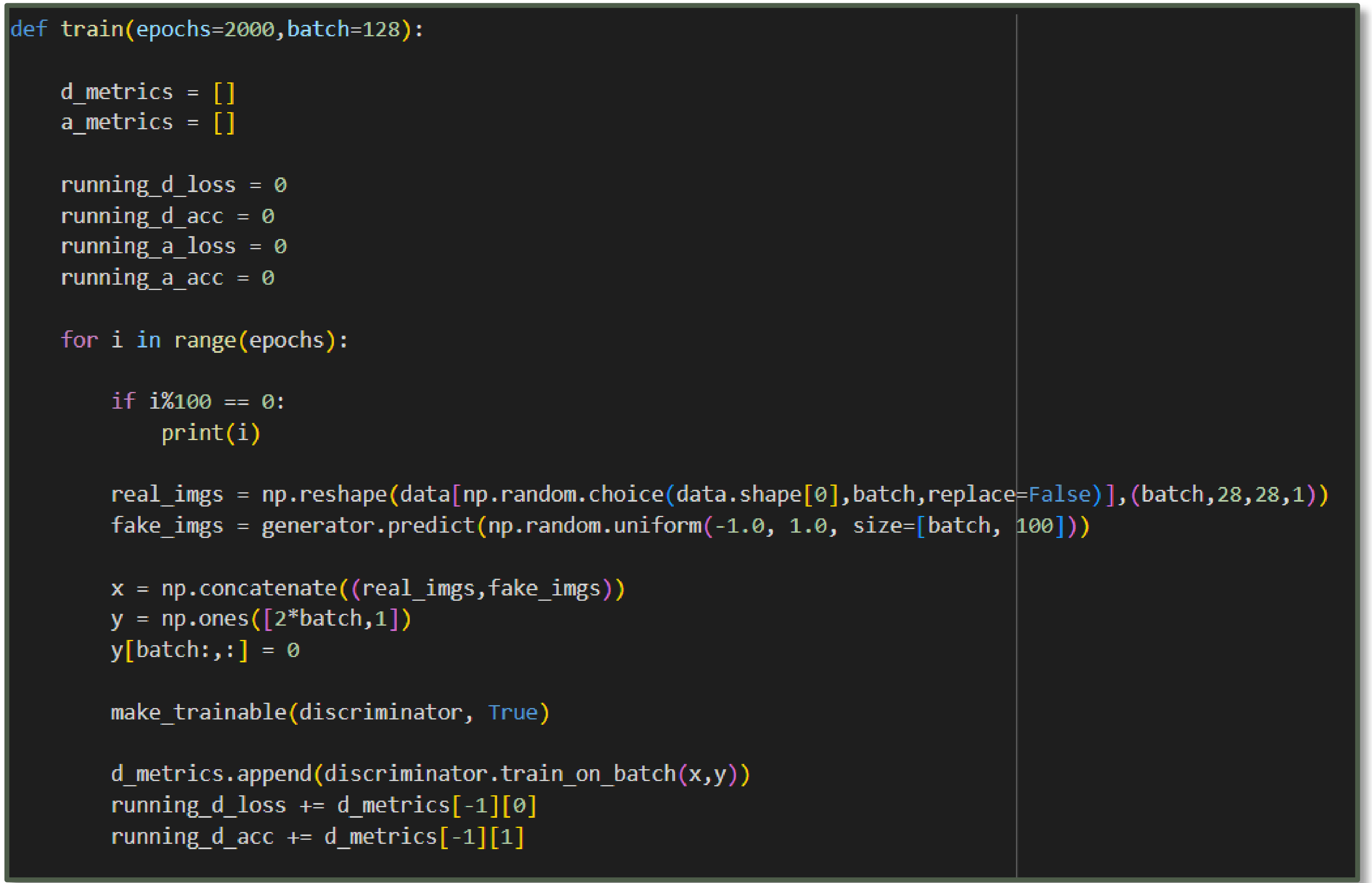


```
def make_trainable(net, val):
    net.trainable = val
    for l in net.layers:
        l.trainable = val
```

GAN Example

Train the Discriminator

Training the discriminator is similar to training any other Convolutional Neural Network (CNN). First, we create a batch of real and fake images. To generate fake images, we ask the generator to produce images from random seeds. Next, we create a vector of corresponding labels, where we use 1 to represent real images and 0 to represent fake images. We then make all the layers of the discriminator trainable using the function defined earlier. Finally, we train the discriminator on the batch we created and store the loss and accuracy in a list for later analysis.

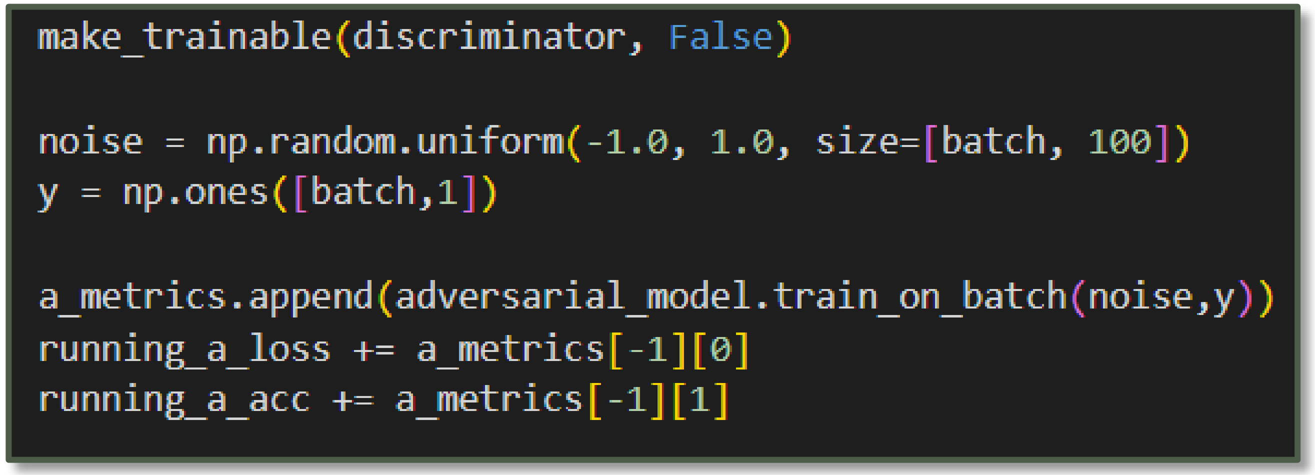


```python
def train(epochs=2000,batch=128):

    d_metrics = []
    a_metrics = []

    running_d_loss = 0
    running_d_acc = 0
    running_a_loss = 0
    running_a_acc = 0

    for i in range(epochs):

        if i%100 == 0:
            print(i)

        real_imgs = np.reshape(data[np.random.choice(data.shape[0],batch,replace=False)],(batch,28,28,1))
        fake_imgs = generator.predict(np.random.uniform(-1.0, 1.0, size=[batch, 100]))

        x = np.concatenate((real_imgs,fake_imgs))
        y = np.ones([2*batch,1])
        y[batch:,:] = 0

        make_trainable(discriminator, True)

        d_metrics.append(discriminator.train_on_batch(x,y))
        running_d_loss += d_metrics[-1][0]
        running_d_acc += d_metrics[-1][1]
```

GAN Example

Train the Generator

To train the generator, we first freeze all the layers of the discriminator network. Next, we create a batch of random noise to serve as the seeds for the generator network. The goal of the generator is to fool the discriminator into thinking that the generated images are real. This means that the desired output for the generator network is 1, so we will label all of our samples as 1. We then train the adversarial network using the inputs and the labels. Since the parameters of the discriminator are frozen, we will only update the parameters of the generator network. Finally, we store the loss and accuracy of the model for further evaluation.



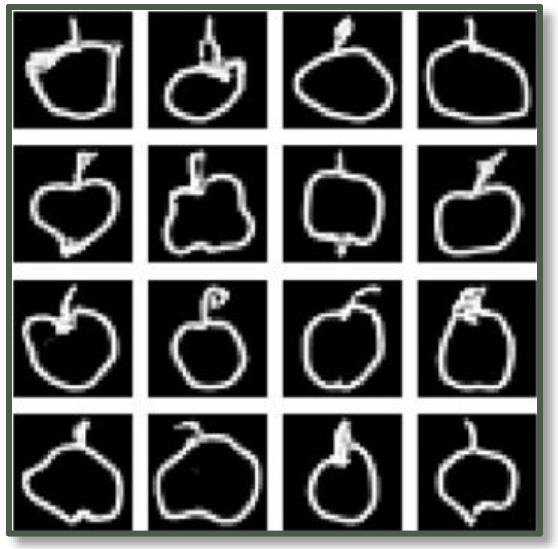
```
make_trainable(discriminator, False)

noise = np.random.uniform(-1.0, 1.0, size=[batch, 100])
y = np.ones([batch,1])

a_metrics.append(adversarial_model.train_on_batch(noise,y))
running_a_loss += a_metrics[-1][0]
running_a_acc += a_metrics[-1][1]
```

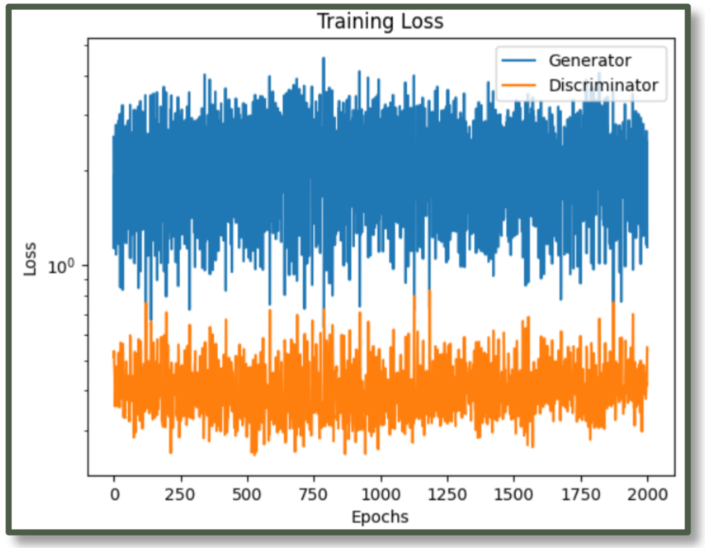
GAN Example

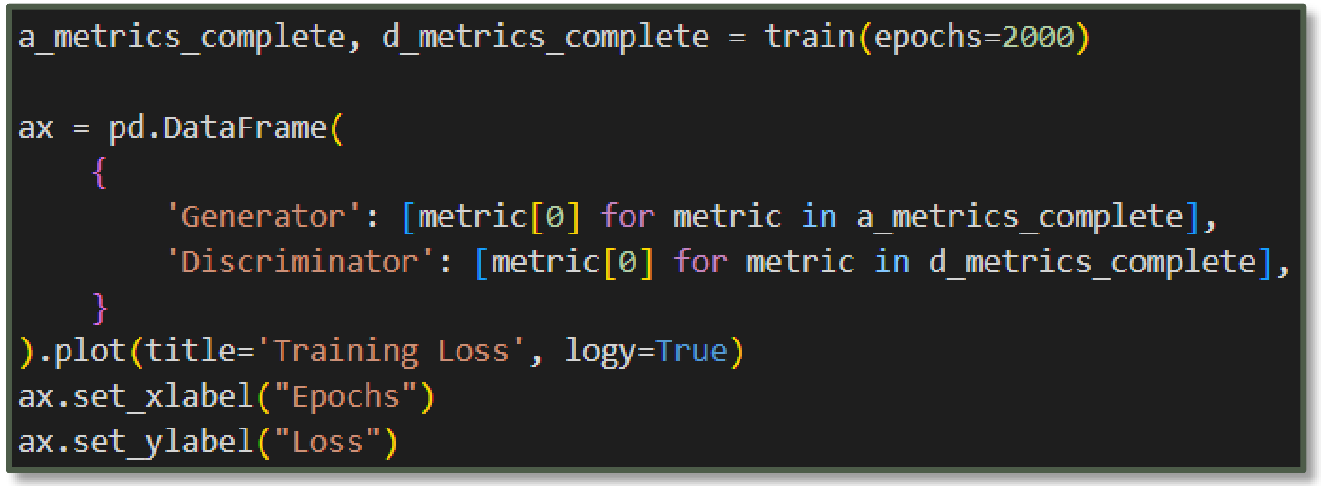
Train and Evaluate the Model

Now, we will use the training function defined earlier to train the model for 2,000 epochs. Training Generative Adversarial Networks (GANs) is quite challenging, and they often require many epochs to learn how to generate acceptable images effectively.

At the end of the training process, we will plot the loss of both the generator and the discriminator networks throughout the training. By examining the results, we can clearly see that convergence in these models is quite tricky and can take a significant amount of time.







```
a_metrics_complete, d_metrics_complete = train(epochs=2000)

ax = pd.DataFrame(
    {
        'Generator': [metric[0] for metric in a_metrics_complete],
        'Discriminator': [metric[0] for metric in d_metrics_complete],
    }
).plot(title='Training Loss', logy=True)
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
```