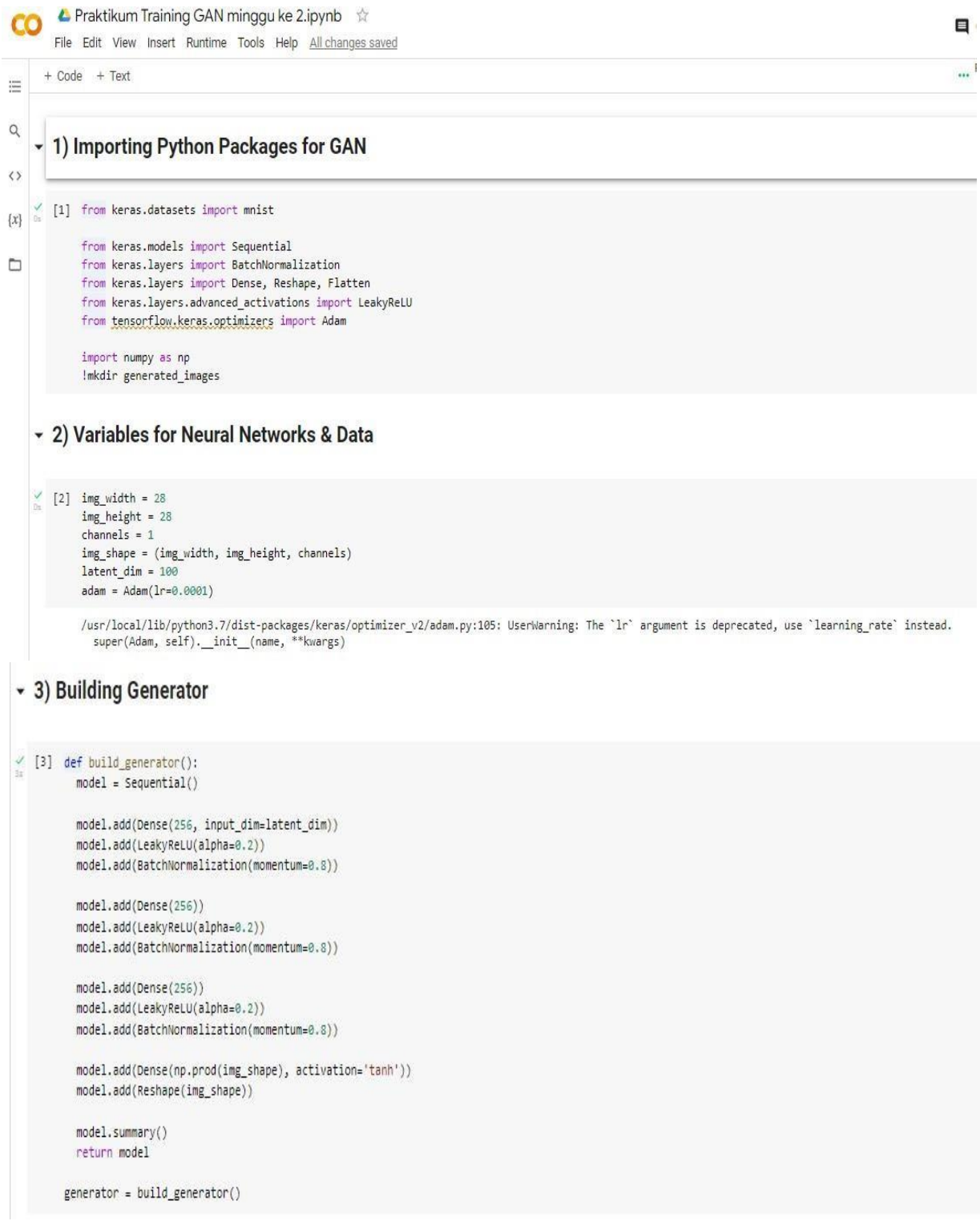
Minggu Ke 2


Praktikum Training GAN minggu ke 2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
1) Importing Python Packages for GAN
[1] from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import BatchNormalization
from keras.layers import Dense, Reshape, Flatten
from keras.layers.advanced_activations import LeakyReLU
from tensorflow.keras.optimizers import Adam
import numpy as np
!mkdir generated_images
2) Variables for Neural Networks & Data
[2] img_width = 28
img_height = 28
channels = 1
img_shape = (img_width, img_height, channels)
latent_dim = 100
adam = Adam(lr=0.0001)
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
super(Adam, self).__init__(name, **kwargs)
3) Building Generator
[3] def build_generator():
model = Sequential()
model.add(Dense(256, input_dim=latent_dim))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(np.prod(img_shape), activation='tanh'))
model.add(Reshape(img_shape))
model.summary()
return model
generator = build_generator()

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               25856

 leaky_re_lu (LeakyReLU)     (None, 256)               0

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dense_1 (Dense)             (None, 256)               65792

 leaky_re_lu_1 (LeakyReLU)   (None, 256)               0

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_2 (Dense)             (None, 256)               65792

 leaky_re_lu_2 (LeakyReLU)   (None, 256)               0

 batch_normalization_2 (Batc  (None, 256)              1024
 hNormalization)

 dense_3 (Dense)             (None, 784)               201488

 reshape (Reshape)           (None, 28, 28, 1)         0

=================================================================
Total params: 362,000
Trainable params: 360,464
Non-trainable params: 1,536
_________________________________________________________________
```

4) Building Discriminator

```
[4]  def build_discriminator():
       model = Sequential()

       model.add(Flatten(input_shape=img_shape))
       model.add(Dense(512))
       model.add(LeakyReLU(alpha=0.2))
       model.add(Dense(256))
       model.add(Dense(1, activation='sigmoid'))

       model.summary()
       return model

     discriminator = build_discriminator()
     discriminator.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense_4 (Dense)             (None, 512)               401920

 leaky_re_lu_3 (LeakyReLU)   (None, 512)               0

 dense_5 (Dense)             (None, 256)               131328

 dense_6 (Dense)             (None, 1)                 257

=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
_________________________________________________________________
```

5) Connecting Neural Networks to build GAN

```
[5] GAN = Sequential()
    discriminator.trainable = False
    GAN.add(generator)
    GAN.add(discriminator)

    GAN.compile(loss='binary_crossentropy', optimizer=adam)
    GAN.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 28, 28, 1)         362000

 sequential_1 (Sequential)   (None, 1)                 533505

=================================================================
Total params: 895,505
Trainable params: 360,464
Non-trainable params: 535,041
_________________________________________________________________
```

6) Outputting Images

```
[6]  #@title
     ## **7) Outputting Images**
     import matplotlib.pyplot as plt
     import glob
     import imageio
     import PIL

     save_name = 0.00000000

     def save_imgs(epoch):
         r, c = 5, 5
         noise = np.random.normal(0, 1, (r * c, latent_dim))
         gen_imgs = generator.predict(noise)
         global save_name
         save_name += 0.00000001
         print("%.8f" % save_name)

         # Rescale images 0 - 1
         gen_imgs = 0.5 * gen_imgs + 0.5

         fig, axs = plt.subplots(r, c)
         cnt = 0
         for i in range(r):
             for j in range(c):
                 axs[i,j].imshow(gen_imgs[cnt, :,:,0], cmap='gray')
                 # axs[i,j].imshow(gen_imgs[cnt])
                 axs[i,j].axis('off')
                 cnt += 1
         fig.savefig("generated_images/%.8f.png" % save_name)
         print('saved')
         plt.close()
```

7) Training GAN

```
def train(epochs, batch_size=64, save_interval=200):
  (X_train, _), (_, _) = mnist.load_data()

  # print(X_train.shape)
  #Rescale data between -1 and 1
  X_train = X_train / 127.5 -1.
  # X_train = np.expand_dims(X_train, axis=3)
  # print(X_train.shape)

  #Create our Y for our Neural Networks
  valid = np.ones((batch_size, 1))
  fakes = np.zeros((batch_size, 1))

  for epoch in range(epochs):
    #Get Random Batch
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    #Generate Fake Images
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    #Train discriminator
    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fakes)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))

    #inverse y label
    g_loss = GAN.train_on_batch(noise, valid)

    print("******* %d [D loss: %f, acc: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100* d_loss[1], g_loss))

    if(epoch % save_interval) == 0:
      save_imgs(epoch)

  # print(valid)


train(30000, batch_size=64, save_interval=200)
```

```
******* 29951 [D loss: 0.589171, acc: 67.97%] [G loss: 1.395642]
******* 29952 [D loss: 0.558636, acc: 71.09%] [G loss: 1.352116]
******* 29953 [D loss: 0.509654, acc: 68.75%] [G loss: 1.188756]
******* 29954 [D loss: 0.520957, acc: 71.88%] [G loss: 1.620432]
******* 29955 [D loss: 0.602423, acc: 71.09%] [G loss: 1.591233]
******* 29956 [D loss: 0.588075, acc: 70.31%] [G loss: 1.427646]
******* 29957 [D loss: 0.600280, acc: 66.41%] [G loss: 1.279619]
******* 29958 [D loss: 0.543135, acc: 70.31%] [G loss: 1.429614]
******* 29959 [D loss: 0.517202, acc: 76.56%] [G loss: 1.734145]
******* 29960 [D loss: 0.631524, acc: 64.84%] [G loss: 1.572028]
******* 29961 [D loss: 0.664978, acc: 65.62%] [G loss: 1.397924]
******* 29962 [D loss: 0.627157, acc: 68.75%] [G loss: 1.363812]
******* 29963 [D loss: 0.566750, acc: 69.53%] [G loss: 1.481601]
******* 29964 [D loss: 0.547315, acc: 77.34%] [G loss: 1.559325]
******* 29965 [D loss: 0.536611, acc: 78.12%] [G loss: 1.421046]
******* 29966 [D loss: 0.550907, acc: 72.66%] [G loss: 1.263606]
******* 29967 [D loss: 0.660345, acc: 61.72%] [G loss: 1.153933]
******* 29968 [D loss: 0.538504, acc: 75.00%] [G loss: 1.307243]
******* 29969 [D loss: 0.532854, acc: 78.12%] [G loss: 1.593028]
******* 29970 [D loss: 0.462812, acc: 75.00%] [G loss: 1.746032]
******* 29971 [D loss: 0.456108, acc: 80.47%] [G loss: 1.841790]
******* 29972 [D loss: 0.583858, acc: 66.41%] [G loss: 1.461316]
******* 29973 [D loss: 0.619556, acc: 73.44%] [G loss: 1.292511]
******* 29974 [D loss: 0.539022, acc: 70.31%] [G loss: 1.429295]
******* 29975 [D loss: 0.501318, acc: 72.66%] [G loss: 1.671717]
******* 29976 [D loss: 0.398879, acc: 85.16%] [G loss: 1.906677]
******* 29977 [D loss: 0.647697, acc: 65.62%] [G loss: 1.524808]
******* 29978 [D loss: 0.634908, acc: 71.09%] [G loss: 1.205254]
******* 29979 [D loss: 0.640701, acc: 63.28%] [G loss: 1.311170]
******* 29980 [D loss: 0.584872, acc: 71.09%] [G loss: 1.338706]
******* 29981 [D loss: 0.582139, acc: 71.88%] [G loss: 1.609013]
******* 29982 [D loss: 0.496531, acc: 75.78%] [G loss: 1.774616]
******* 29983 [D loss: 0.573115, acc: 70.31%] [G loss: 1.567858]
******* 29984 [D loss: 0.488099, acc: 72.66%] [G loss: 1.230682]
******* 29985 [D loss: 0.571470, acc: 72.66%] [G loss: 1.097119]
******* 29986 [D loss: 0.610216, acc: 63.28%] [G loss: 1.478118]
******* 29987 [D loss: 0.513266, acc: 75.78%] [G loss: 1.715492]
******* 29988 [D loss: 0.448614, acc: 79.69%] [G loss: 1.665279]
******* 29989 [D loss: 0.480862, acc: 75.00%] [G loss: 1.635394]
******* 29990 [D loss: 0.593129, acc: 66.41%] [G loss: 1.359214]
******* 29991 [D loss: 0.499047, acc: 71.88%] [G loss: 1.414220]
******* 29992 [D loss: 0.526219, acc: 71.88%] [G loss: 1.494169]
******* 29993 [D loss: 0.515515, acc: 69.53%] [G loss: 1.756819]
******* 29994 [D loss: 0.524061, acc: 73.44%] [G loss: 1.669944]
******* 29995 [D loss: 0.545207, acc: 68.75%] [G loss: 1.393118]
******* 29996 [D loss: 0.628788, acc: 64.06%] [G loss: 1.135735]
******* 29997 [D loss: 0.598283, acc: 69.53%] [G loss: 1.451259]
******* 29998 [D loss: 0.558456, acc: 67.97%] [G loss: 1.557040]
******* 29999 [D loss: 0.514513, acc: 74.22%] [G loss: 1.758997]
```

8) Making GIF

```
[12] # Display a single image using the epoch number
     # def display_image(epoch_no):
     #   return PIL.Image.open('generated_images/%.8f.png'.format(epoch_no))

     anim_file = 'dcgan.gif'

     with imageio.get_writer(anim_file, mode='I') as writer:
       filenames = glob.glob('generated_images/*.png')
       filenames = sorted(filenames)
       for filename in filenames:
         image = imageio.imread(filename)
         writer.append_data(image)
       image = imageio.imread(filename)
       writer.append_data(image)
```