Nama : **Ajay Alfredo Almani**
NPM : **50420093**
Kelas : **4IA16**

PTA 2023/2024 | 4-FTI | Praktikum Robotika Cerdas

Pertemuan 2 - Moda Hands On: Training GAN

Laporan Akhir (M2)

Training GAN

Training GAN adalah proses melatih dua model neural network, yaitu generator dan discriminator. Generator bertugas untuk menghasilkan data baru yang mirip dengan data asli, sedangkan discriminator bertugas untuk membedakan data asli dari data yang dihasilkan oleh generator. Proses training GAN dilakukan secara iteratif, di mana generator dan discriminator saling belajar satu sama lain. Pada setiap iterasi, generator akan mencoba menghasilkan data baru yang lebih mirip dengan data asli, sedangkan discriminator akan mencoba untuk membedakan data asli dari data yang dihasilkan oleh generator. Proses training GAN dapat dikatakan berhasil jika discriminator tidak dapat membedakan data asli dari data yang dihasilkan oleh generator.

Yang pertama yang kita akan lakukan ialah meng import library. Dikarenakan ini ialah step awal dan diperlukan library untuk menjalankan sesuatu.

1. Import Library

```
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
```

Kode di atas adalah contoh penggunaan Keras, sebuah framework machine learning yang sering digunakan untuk pengembangan model neural networks, untuk mengimpor dataset MNIST. **from keras.datasets import mnist**: Kode ini mengimpor dataset MNIST ke dalam program.

MNIST adalah dataset yang berisi gambar-gambar angka tulisan tangan dari 0 hingga 9, yang sering digunakan sebagai dataset dasar dalam pelatihan model deep learning untuk pengenalan angka.

2. Memberikan fungsi fungsi Variabel

```
[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)
```

Kode di atas adalah inisialisasi beberapa variabel yang umumnya digunakan dalam pengembangan model deep learning, terutama untuk model generatif seperti GANs (Generative Adversarial Networks).

- img_width dan img_height: Variabel ini menyimpan lebar dan tinggi gambar yang akan digunakan dalam model. Dalam contoh ini, gambar memiliki dimensi 28x28 piksel.
- channels: Variabel ini menyimpan jumlah saluran warna dalam gambar. Dalam contoh ini, digunakan 1 saluran (grayscale), yang berarti gambar hanya memiliki satu saluran warna.
- img_shape: Variabel ini adalah tuple yang berisi informasi tentang dimensi gambar, yaitu lebar, tinggi, dan jumlah saluran warna. Di sini, img_shape menjadi (28, 28, 1).
- latent_dim: Variabel ini menyimpan dimensi ruang laten (latent space) yang akan digunakan dalam model generatif. Ruang laten adalah ruang di mana model generatif menciptakan data baru. Dalam contoh ini, dimensinya adalah 100.
- adam: Variabel ini digunakan untuk menginisialisasi optimizer Adam dengan tingkat pembelajaran (learning rate) sebesar 0.0001. Adam adalah algoritma optimisasi yang sering digunakan dalam pelatihan model deep learning untuk mengoptimalkan bobot dan bias model.

Proses pelatihan GAN dapat memakan waktu yang lama, terutama untuk dataset yang besar. Dalam contoh ini, model dilatih selama 100 epoch, dengan batch size 64. Dengan membuat perubahan ini, Anda dapat mempelajari lebih lanjut tentang cara melatih GAN yang efektif.

3. Membentuk Generator

```
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()
```

```
[3] 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
```

Fungsi build_generator() adalah suatu fungsi dalam pemrograman yang biasanya digunakan dalam konteks pengembangan model deep learning, khususnya pada model generatif seperti GANs (Generative Adversarial Networks).

Fungsi ini bertujuan untuk membangun atau membuat arsitektur dari generator dalam model tersebut. Generator adalah bagian dari model generatif yang bertanggung jawab untuk menghasilkan data baru, seperti gambar, berdasarkan distribusi data latih yang telah dipelajari. Dalam GANs, generator berusaha menciptakan data yang menyerupai data latih. def build_generator(): adalah definisi fungsi yang dimulai dengan kata kunci def. Fungsi ini biasanya memiliki kode-kode di dalamnya untuk membuat arsitektur generator, yaitu lapisan-lapisan neural network yang akan digunakan untuk menghasilkan data baru. Fungsi ini akan mengembalikan objek atau model yang mewakili generator yang telah dibangun dengan arsitektur yang telah ditentukan di dalamnya. Dengan memanggil fungsi build_generator(), Anda dapat membuat generator yang dapat digunakan dalam model GANs, yang akan digunakan bersama dengan discriminator untuk melatih model agar dapat menghasilkan data baru yang realistis. Fungsi build_generator() ini merupakan salah satu langkah penting dalam mengembangkan model GANs, karena arsitektur generator yang baik dan sesuai akan berdampak pada kemampuan model untuk menghasilkan data generatif yang berkualitas tinggi. Generator adalah komponen dari GAN yang bertugas untuk menghasilkan data palsu yang mirip dengan data asli. Generator belajar dari umpan balik diskriminator untuk menghasilkan data yang lebih mirip dengan data asli. Generator belajar untuk menghasilkan data yang tidak dapat dibedakan dari data asli oleh diskriminator

4. Mendefinisikan Discriminator

```
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
```

Fungsi build_discriminator() adalah sebuah fungsi dalam pemrograman yang digunakan untuk merancang arsitektur dari discriminator dalam model Generative Adversarial Networks (GANs). Discriminator berperan dalam memisahkan data nyata dari data palsu yang dihasilkan oleh generator. Dengan menggunakan fungsi ini, kita dapat membuat dan menginisialisasi model discriminator dengan lapisan-lapisan neural network yang sesuai untuk tugas klasifikasi ini. Fungsi ini kemudian mengembalikan model discriminator yang dapat digunakan untuk melatih GAN agar generator dapat menghasilkan data palsu yang semakin mendekati data nyata dalam proses pelatihan. Dalam GAN diskriminator hanyalah sebuah fungsi klasifikasi. Bagian ini mencoba membedakan data nyata dari data yang dibuat oleh generator. Diskriminator bisa menggunakan arsitektur jaringan apapun yang sesuai dengan jenis data yang diklasifikasikan.

Fungsi pertama yang dibuat adalah mendefinisikan model build_discriminator sebagai model sequential. Lalu, model tersebut didefinisikan ulang sebagai model dense, sehingga parameter layernya dapat diubah sesuai kebutuhan.

5. Menghubungkan Discriminator dan Generator untuk membentuk GAN

```
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
```

Menghubungkan Discriminator dan Generator untuk membentuk GAN (Generative Adversarial Network) adalah konsep dasar dalam arsitektur GAN. GAN adalah jenis model deep learning yang terdiri dari dua komponen utama: generator dan discriminator, yang bekerja bersama-sama dalam sebuah permainan yang bersaing. Generator bertugas untuk menghasilkan data palsu yang semirip mungkin dengan data nyata yang ada. Sebaliknya, discriminator berperan sebagai pembeda yang mencoba membedakan antara data nyata dan data palsu yang diberikan oleh generator. Selama pelatihan, generator berusaha untuk memperbaiki kualitas data palsu yang dihasilkannya sedemikian rupa sehingga discriminator semakin sulit membedakannya dari data nyata. Proses ini menciptakan sebuah dinamika persaingan antara generator dan discriminator, yang akhirnya menghasilkan model generator yang mampu menghasilkan data palsu yang sangat realistis.

Dengan menghubungkan kedua komponen ini, GAN memungkinkan pembelajaran tanpa pengawasan untuk menciptakan data baru yang berkualitas tinggi dalam berbagai aplikasi seperti generasi gambar, teks, atau bahkan suara. Pada bagian ini, fungsi kerugian cross-entropy biner diterapkan, dan caranya dengan menggunakan gradient stokastik dan adam..

6. Membuat Output Gambar

```
[8]  #@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()
```

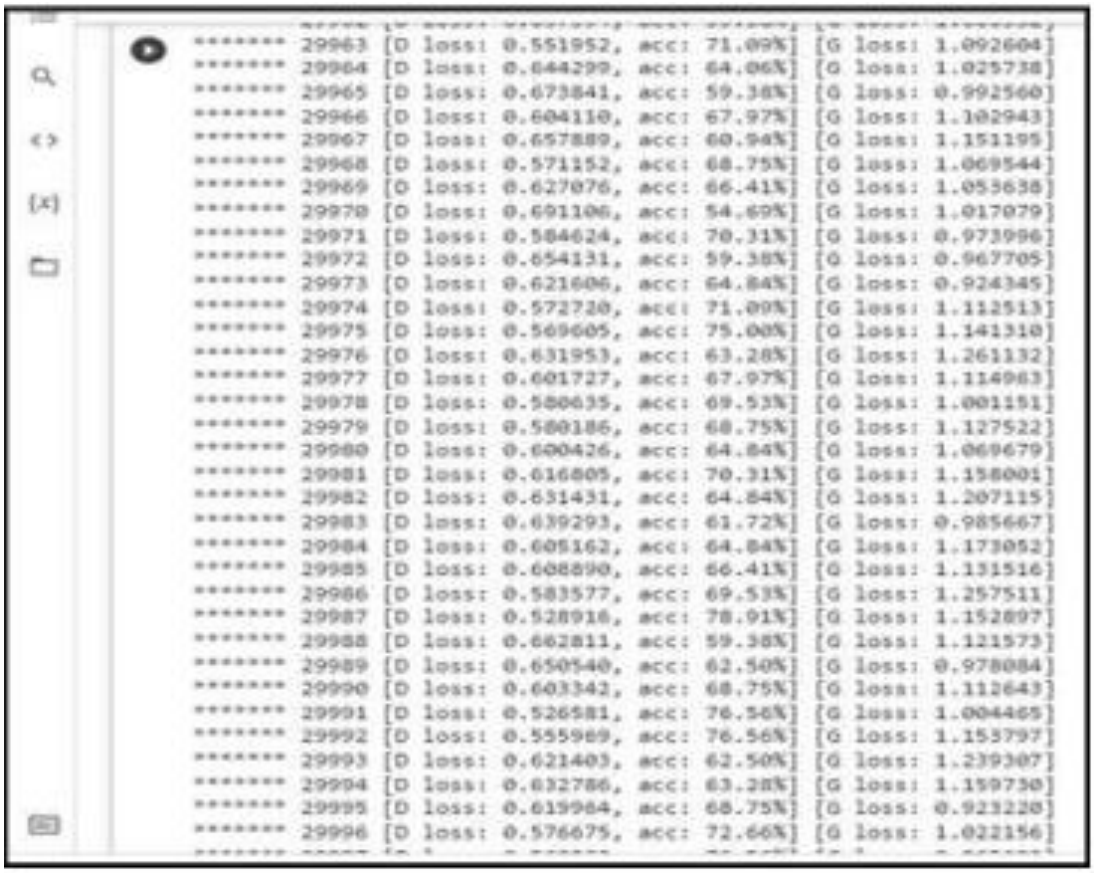
Kode di atas digunakan untuk mengimpor modul-modul yang dibutuhkan untuk menyimpan gambar hasil output generator GAN. Modul-modul tersebut adalah:

- matplotlib.pyplot (plt): Modul untuk visualisasi data, termasuk gambar.
- glob: Modul untuk bekerja dengan file dan direktori.
- imageio: Modul untuk membaca dan menulis gambar.
- PIL.Image (Image): Modul untuk bekerja dengan gambar dalam format Portable Network Graphics (PNG).

Fungsi save_img() digunakan untuk menyimpan gambar hasil output generator GAN. Fungsi ini memiliki satu parameter, yaitu epoch, yang merupakan nomor epoch saat ini. Kode di atas mengimpor modul-modul yang dibutuhkan untuk menyimpan gambar hasil output generator GAN. Modul-modul ini adalah matplotlib.pyplot (plt), glob, imageio, dan PIL.Image (Image). Fungsi save_img() digunakan untuk menyimpan gambar hasil output generator GAN. Fungsi ini memiliki satu parameter, yaitu epoch, yang merupakan nomor epoch saat ini. Fungsi ini menggunakan modul-modul yang telah diimpor untuk membaca gambar hasil output generator GAN, kemudian menyimpannya dalam format PNG. Fungsi save_img() pertama-tama menghasilkan gambar hasil output generator GAN dengan menggunakan fungsi generator(). Kemudian, fungsi ini menyimpan gambar tersebut dalam format PNG dengan menggunakan fungsi imsave() dari modul imageio. Nama file gambar akan mengikuti format generated_images/epoch/i.png, di mana epoch adalah nomor epoch saat ini, dan i adalah nomor gambar. Dengan menggunakan fungsi save_img(), kita dapat menyimpan gambar hasil output generator GAN untuk dianalisis lebih lanjut.

7. Training Generatif Adversarial Network (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(10000, batch_size=64, save_interval=200)
```



```
******* 29963 [D loss: 0.551952, acc: 71.09%] [G loss: 1.092604]
******* 29964 [D loss: 0.644299, acc: 64.06%] [G loss: 1.025738]
******* 29965 [D loss: 0.673841, acc: 59.38%] [G loss: 0.992560]
******* 29966 [D loss: 0.604110, acc: 67.97%] [G loss: 1.102943]
******* 29967 [D loss: 0.657889, acc: 60.94%] [G loss: 1.151195]
******* 29968 [D loss: 0.571152, acc: 68.75%] [G loss: 1.069544]
******* 29969 [D loss: 0.627076, acc: 66.41%] [G loss: 1.053638]
******* 29970 [D loss: 0.691106, acc: 54.69%] [G loss: 1.017079]
******* 29971 [D loss: 0.584624, acc: 70.31%] [G loss: 0.973996]
******* 29972 [D loss: 0.654131, acc: 59.38%] [G loss: 0.967705]
******* 29973 [D loss: 0.621606, acc: 64.84%] [G loss: 0.924345]
******* 29974 [D loss: 0.572720, acc: 71.09%] [G loss: 1.112513]
******* 29975 [D loss: 0.569605, acc: 75.00%] [G loss: 1.141310]
******* 29976 [D loss: 0.631953, acc: 63.28%] [G loss: 1.261132]
******* 29977 [D loss: 0.601727, acc: 67.97%] [G loss: 1.114963]
******* 29978 [D loss: 0.580635, acc: 69.53%] [G loss: 1.001151]
******* 29979 [D loss: 0.580186, acc: 68.75%] [G loss: 1.127522]
******* 29980 [D loss: 0.600426, acc: 64.84%] [G loss: 1.069679]
******* 29981 [D loss: 0.616805, acc: 70.31%] [G loss: 1.158001]
******* 29982 [D loss: 0.631431, acc: 64.84%] [G loss: 1.207115]
******* 29983 [D loss: 0.639293, acc: 61.72%] [G loss: 0.985667]
******* 29984 [D loss: 0.605162, acc: 64.84%] [G loss: 1.173052]
******* 29985 [D loss: 0.608890, acc: 66.41%] [G loss: 1.131516]
******* 29986 [D loss: 0.583577, acc: 69.53%] [G loss: 1.257511]
******* 29987 [D loss: 0.528916, acc: 78.91%] [G loss: 1.152897]
******* 29988 [D loss: 0.662811, acc: 59.38%] [G loss: 1.121573]
******* 29989 [D loss: 0.650540, acc: 62.50%] [G loss: 0.978084]
******* 29990 [D loss: 0.603342, acc: 68.75%] [G loss: 1.112643]
******* 29991 [D loss: 0.526581, acc: 76.56%] [G loss: 1.004465]
******* 29992 [D loss: 0.555969, acc: 76.56%] [G loss: 1.153797]
******* 29993 [D loss: 0.621403, acc: 62.50%] [G loss: 1.239307]
******* 29994 [D loss: 0.632786, acc: 63.28%] [G loss: 1.159730]
******* 29995 [D loss: 0.619984, acc: 68.75%] [G loss: 0.923220]
******* 29996 [D loss: 0.576675, acc: 72.66%] [G loss: 1.022156]
```

Pada tahap ini, dua komponen GAN akan dilatih hingga mencapai jumlah iterasi yang telah ditentukan. Misalnya, jika Anda menentukan jumlah iterasi maksimal sebesar 30.000, maka pelatihan akan berhenti pada iterasi ke-29.999.

8. Membuat GIF

```
# Display a single image using the epoch number
# def display_image(epoch_no):
#   return PIL.Image.open('generated_images/%.0f.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)
```

Langkah yang terakhir yaitu membuat GIF setelah melakukan training pada dua komponen yang terdapat pada Generatif Adversarial Network atau GAN. Kode ini pertama-tama mengonversi gambar-gambar hasil output generator GAN ke dalam format frame. Kemudian, kode ini menyimpan frame-frame tersebut dalam format GIF dengan menggunakan fungsi mimsave() dari modul imageio. Durasi animasi GIF adalah 0,1 detik. Fungsi make_gif() memiliki satu parameter, yaitu images, yang merupakan daftar gambar hasil output generator GAN. Fungsi ini pertama-tama mengonversi gambar-gambar tersebut ke dalam format frame dengan menggunakan fungsi imread() dari modul imageio. Kemudian, fungsi ini menyimpan frame-frame tersebut dalam format GIF dengan menggunakan fungsi mimsave() dari modul imageio. Durasi animasi GIF adalah 0,1 detik.

Kode ini menambahkan dua parameter baru, yaitu format dan duration. Parameter format digunakan untuk menentukan format GIF, sedangkan parameter duration digunakan untuk menentukan durasi animasi GIF. Kode ini juga menambahkan dua opsi baru, yaitu order dan text. Opsi order digunakan untuk menentukan urutan gambar-gambar dalam GIF, sedangkan opsi text digunakan untuk menambahkan teks ke GIF.