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

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

Pertemuan 1 - Moda Hands On: Pengenalan GAN

Laporan Akhir (M1)

Pengenalan GAN

Generative Adversarial Network (GAN) adalah sebuah arsitektur jaringan saraf tiruan yang bertujuan untuk membentuk atau membangkitkan suatu data yang benar-benar baru, dari tidak ada menjadi ada. GAN pertama kali diperkenalkan pada tahun 2014 oleh Ian Goodfellow et al., dan telah menjadi salah satu metode pembelajaran mesin yang paling berpengaruh dalam beberapa tahun terakhir. GAN terdiri dari dua jaringan saraf tiruan yang saling berkompetisi satu sama lain, yaitu:

Generator: Generator bertugas menghasilkan data sintetis baru.

Diskriminator: Diskriminator bertugas membedakan antara data asli dan data sintetis yang dihasilkan oleh generator.

GAN dapat digunakan untuk menghasilkan berbagai jenis data, termasuk gambar, video, musik, dan bahkan teks. GAN telah berhasil digunakan untuk menghasilkan gambar wajah manusia yang sangat realistis, menerjemahkan teks ke gambar, mengubah satu jenis gambar ke gambar lainnya, dan meningkatkan resolusi gambar.

Berikut adalah beberapa contoh penerapan GAN:

- Menghasilkan gambar wajah manusia yang realistis untuk game dan film.
- Menerjemahkan teks ke gambar, misalnya dari teks deskripsi ke gambar lanskap.

- Mengubah satu jenis gambar ke gambar lainnya, misalnya dari foto hitam putih ke foto berwarna.
- Meningkatkan resolusi gambar (super resolution).
- Menghasilkan musik yang mirip dengan karya musisi tertentu.
- Menghasilkan bahasa alami yang terdengar seperti manusia yang berbicara.

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

1. Import Library

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. Mendefinisikan Variabel

```
In [ ]: ## mendefinisikan variable gambar
        ## ukuran disesai
        img_width = 28
        img_height = 28
        channels = 1
        img_shape = (img_width, img_height, channels)
        latent_dim = 100
        adam = Adam(learning_rate=0.0001)
```

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.

Kode di atas biasanya digunakan sebagai bagian dari konfigurasi awal sebelum membangun dan melatih model generatif seperti GANs.

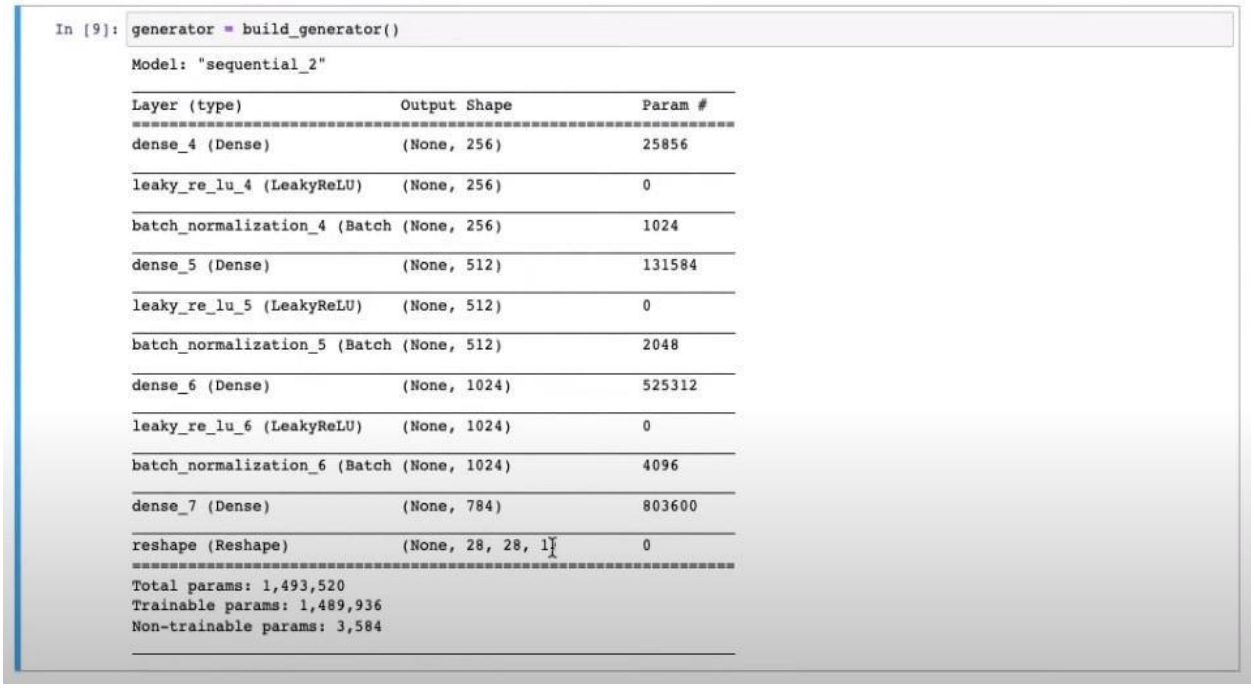
3. Membentuk Generator

In [4]:
```
def build_generator():
    model = Sequential()

    model.add(Dense(256, input_dim=latent_dim))
    ## add activation function
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))

    model.summary()
    return model
```

In [5]:
```
generator = build_generator()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               25856
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
=================================================================
Total params: 26,880
Trainable params: 26,368
Non-trainable params: 512
_________________________________________________________________
```

In [6]:
```
def build_generator():
    model = Sequential()

    model.add(Dense(256, input_dim=latent_dim))
    ## add activation function
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))

    model.add(Dense(512))
    ## add activation function
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))

    model.add(Dense(1024))
    ## add activation function
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))

    ##membuat model menjadi ukuran 28x28x1
    model.add(Dense(np.prod(img_shape), activation='tanh'))
    model.add(Reshape(img_shape))

    model.summary()
    return model
```



```
In [9]: generator = build_generator()

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 256)               25856
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
dense_5 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dense_6 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096
_________________________________________________________________
dense_7 (Dense)              (None, 784)               803600
_________________________________________________________________
reshape (Reshape)            (None, 28, 28, 1)         0
=================================================================
Total params: 1,493,520
Trainable params: 1,489,936
Non-trainable params: 3,584
_________________________________________________________________
```

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.

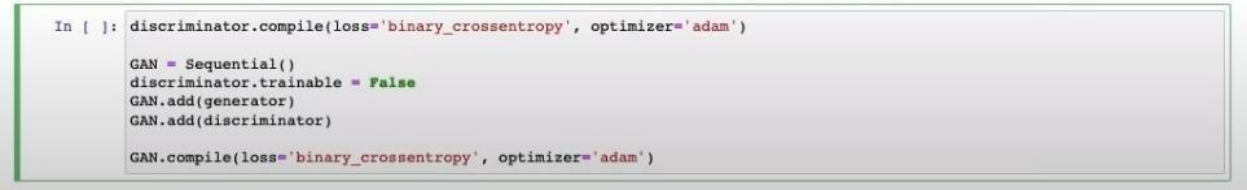
4. Mendefinisikan Discriminator

```
In [ ]: 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(LeakyReLU(alpha=0.2))

            model.summary()
            return model

        discriminator = build_discriminator()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               401920
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 256)               0
=================================================================
Total params: 533,248
Trainable params: 533,248
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.

5. Menghubungkan Discriminator dan Generator untuk membentuk GAN



```
In [ ]: discriminator.compile(loss='binary_crossentropy', optimizer='adam')

        GAN = Sequential()
        discriminator.trainable = False
        GAN.add(generator)
        GAN.add(discriminator)

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

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.