MEMBANGUN CONVOLUTIONAL NEURAL NETWORK (CNN) DI 'KERAS' MENGGUNAKAN **DATASET MNIST**

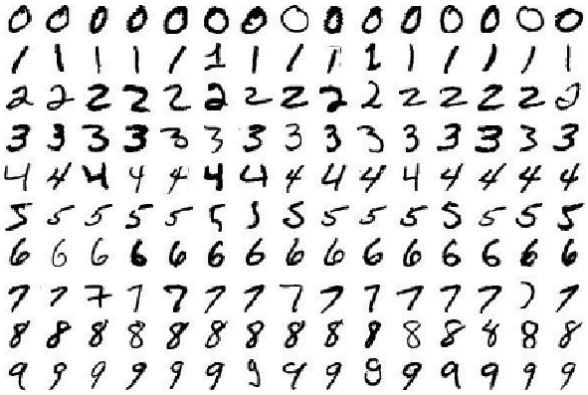
PERSIAPAN

1) INSTALL PYTHON : https://www.python.org/downloads/
2) INSTALL TENSORFLOW: https://www.tensorflow.org/install/pip
3) INSTALL KERAS
4) INSTALL JUGA LIBARY YANG LAIN SEPERTI: NUMPY, MATPLOTLIB, PANDAS DLL

STEP BY STEP

1. **Loading dataset**

   Dataset mnist disediakan dengan mudah kepada kami sebagai bagian dari library Keras, sehingga kami dapat dengan mudah memuat dataset. Dataset ini terdiri dari 70.000 citra dari tulisan tangan dengan angka 0-9. Dari 70.000 gambar yang disediakan dalam kumpulan data, 60.000 diberikan untuk pelatihan dan 10.000 diberikan untuk pengujian. Berikut contoh dataset MNIST:



Gamba1. Contoh dataset MNIST

Saat kita memuat dataset, kemudian dibagi menjadi 2 grup yaitu: X_train dan X_test akan berisi citra, dan y_train dan y_test akan berisi angka yang menunjukkan kelas citra tersebut.

```
from keras.dataset import mnist

#download dataset MNIST dan membagi kedalam kelompok train
dan test

(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

2. Data pre-processing

Selanjutnya, kita perlu membentuk kembali input dataset kita (X_train dan X_test) ke bentuk yang diharapkan model kita saat kita melatih model. Angka pertama adalah jumlah gambar (60.000 untuk X_train dan 10.000 untuk X_test). Kemudian muncul bentuk setiap gambar (28x28). Angka terakhir adalah 1, yang menandakan bahwa citra berwarna abu-abu (grayscale).

```
#reshape ukuran dataset
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```

Kita perlu 'one-hot-encode' variabel target kita. Ini berarti bahwa kolom akan dibuat untuk setiap kategori *output* dan variabel biner dimasukkan untuk setiap kategori. Misalnya, kita melihat bahwa citra pertama dalam kumpulan data adalah **5**. Ini berarti bahwa **angka keenam** dalam larik kita akan memiliki **1** dan sisa larik akan diisi dengan 0.

```
from tensorflow.keras.utils import to_categorical
#one-hot encode target
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Note: jika `from tensorflow.keras.utils import to_categorical` error coba dengan `from keras.utils import to_categorical`

3. CNN Arsitektur

Membangun CNN dengan code berikut:

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten

model = Sequential()

model.add(Conv2D(64, kernel_size=3, activation='relu',
input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add =(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

Tipe model yang akan kita gunakan adalah Sequential. Sequential adalah cara termudah untuk membangun model di Keras. Ini memungkinkan Anda untuk membangun model layer by layer.

Kita menggunakan fungsi 'add()' untuk menambahkan layer ke model. Terdapat 2 layer konvolusi yang dinyatakan dengan Conv2D. Ini adalah layer konvolusi yang akan menangani citra input, yang dilihat sebagai matriks 2 dimensi. 64 di lapisan pertama dan 32 di lapisan kedua adalah jumlah node di setiap layer. Angka ini dapat disesuaikan menjadi lebih tinggi atau lebih rendah, tergantung pada ukuran dataset.

Ukuran kernel adalah ukuran matriks filter untuk konvolusi. Jadi ukuran kernel 3 berarti kita akan memiliki matriks filter 3x3.

Aktivasi adalah fungsi aktivasi untuk layer. Fungsi aktivasi yang akan kita gunakan untuk 2 layer pertama kita adalah ReLU, atau Rectified Linear Activation. Fungsi aktivasi ini telah terbukti bekerja dengan baik di jaringan saraf.

Lapisan pertama kami juga mengambil bentuk input. Ini adalah bentuk setiap gambar input, 28,28,1 seperti yang terlihat sebelumnya, dengan 1 menandakan bahwa gambar tersebut abu-abu.

Di antara layer Conv2D dan layer Dense, ada layer 'Flatten'. Flatten berfungsi sebagai penghubung antara konvolusi dan layer Dense.

'Dense' adalah jenis layer yang akan kita gunakan untuk layer output kita. Dense adalah jenis layer standar yang digunakan dalam banyak kasus untuk jaringan saraf.

Kami akan memiliki 10 node di lapisan output kami, satu untuk setiap kemungkinan hasil (0–9).

Aktivasinya adalah 'softmax'. Softmax membuat jumlah output menjadi 1 sehingga output dapat diartikan sebagai probabilitas. Model kemudian akan membuat prediksinya berdasarkan opsi mana yang memiliki probabilitas tertinggi.

4. **Compiling Model**

Selanjutnya, kita perlu mengkompilasi model kita. Penyusunan model membutuhkan tiga parameter: pengoptimal, kerugian, dan metrik.

Pengoptimal mengontrol kecepatan pembelajaran. Kita akan menggunakan '***adam***' sebagai ***optimizer***. Adam umumnya adalah pengoptimal yang baik untuk digunakan dalam banyak kasus. Pengoptimal adam menyesuaikan tingkat pembelajaran selama pelatihan.

Tingkat pembelajaran menentukan seberapa cepat bobot optimal untuk model dihitung. Tingkat pembelajaran yang lebih kecil dapat menghasilkan bobot yang lebih akurat (hingga titik tertentu), tetapi waktu yang diperlukan untuk menghitung bobot akan lebih lama.

Kita akan menggunakan '***categorical_crossentropy***' untuk ***loss function***. Ini adalah pilihan paling umum untuk klasifikasi. Skor yang lebih rendah menunjukkan bahwa model tersebut berkinerja lebih baik.

Untuk mempermudah interpretasi, kita akan menggunakan metrik 'akurasi' untuk melihat skor akurasi pada set validasi saat kami melatih model.

```
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
```

5. **Training Model**

Sekarang kita akan melatih model. Untuk melatih, kita akan menggunakan fungsi 'fit()' pada model dengan parameter berikut: **data pelatihan (train_X), data target (train_y), data validasi, dan jumlah epoch.**

Untuk data validasi kami, kita akan menggunakan set pengujian yang diberikan di dataset, yang telah kitabagi menjadi X_test dan y_test.

Jumlah epoch adalah berapa kali model akan menggilir data. Semakin banyak epoch yang kita jalankan, semakin model akan meningkat, hingga titik tertentu. Setelah titik itu, model akan berhenti berkembang selama setiap epoch. Untuk model kita, kita akan mengatur jumlah epoch menjadi 3.

```
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=3)
```

FULL CODE

```
from keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D

#PRE-PROCESSING DATA
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

#CNN ARSITEKTUR
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu',
input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

#COMPILING MODEL
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

#TRAINING MODEL
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)

#EVALUATE MODEL
evaluasi = model.evaluate(X_test, y_test, verbose=0)
print('Loss:', evaluasi[0])
print('Acc:', evaluasi[1])
```