

import requests

from IPython.core.display import HTML

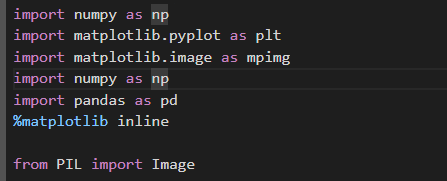
Ini adalah pernyataan impor pada Python. Library **requests** digunakan untuk membuat permintaan HTTP, dan **HTML** adalah kelas yang disediakan oleh IPython untuk menampilkan konten HTML di Jupyter Notebook.

styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/master/content/styles/cs109.css").text

Baris ini menggunakan fungsi **requests.get** untuk membuat permintaan HTTP GET ke URL yang ditentukan. Dalam hal ini, ini mengambil file CSS (Cascading Style Sheets) dari repositori GitHub Harvard-IACS. Konten yang diambil kemudian disimpan dalam variabel **styles**.

HTML(styles)

terakhir, kode ini membuat objek **HTML** dengan konten file CSS yang diambil (**styles**) dan menampilkannya. Ini pada dasarnya menerapkan gaya yang didefinisikan dalam file CSS ke output dari sel-sel di Jupyter Notebook.



Tampaknya Anda telah mengimpor beberapa pustaka umum dalam ilmu data dan pengolahan gambar. Berikut adalah penjelasan singkat untuk setiap pernyataan impor:

1. \*\*NumPy (`import numpy as np`):\*\*

NumPy adalah pustaka kuat untuk operasi numerik di Python. Ini menyediakan dukungan untuk array dan matriks besar, serta fungsi matematika untuk operasi pada elemen-elemen tersebut.

2. \*\*Matplotlib (`import matplotlib.pyplot as plt` dan `import matplotlib.image as mpimg`):\*\*

Matplotlib adalah pustaka plotting yang banyak digunakan di Python. `matplotlib.pyplot` menyediakan kerangka kerja plotting mirip dengan MATLAB, dan `matplotlib.image` khusus menangani fungsionalitas terkait gambar.

3. \*\*Pandas (`import pandas as pd`,)

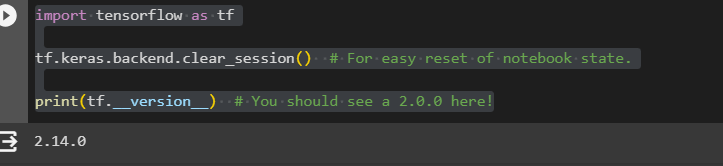
Pandas adalah pustaka manipulasi data untuk Python. Ini menyediakan struktur data seperti DataFrame untuk penanganan dan analisis data yang efisien.

4. \*\*`%matplotlib inline`:\*\*

Ini adalah perintah magis Jupyter yang memungkinkan plot Matplotlib ditampilkan langsung di notebook. Berguna untuk plotting langsung tanpa perlu memanggil `plt.show()`.

5. \*\*PIL (Python Imaging Library) (`from PIL import Image`):\*\*

PIL adalah pustaka untuk membuka, memanipulasi, dan menyimpan berbagai format file gambar. Itu berguna untuk bekerja dengan gambar dalam berbagai proyek, seperti pengolahan gambar atau pengolahan data citra.



1. import tensorflow as tf

Ini mengimpor pustaka TensorFlow ke dalam proyek Anda. TensorFlow adalah pustaka open-source yang sangat populer untuk pembelajaran mesin dan pengembangan model kecerdasan buatan. Dengan mengimpornya, Anda dapat menggunakan semua fitur dan fungsi yang disediakan oleh TensorFlow.

1. **Clear TensorFlow Session:**

pythonCopy code

tf.keras.backend.clear\_session()

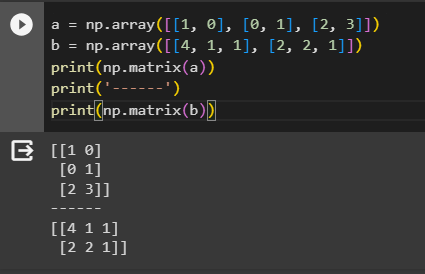
Ini adalah fungsi untuk membersihkan atau mengatur ulang sesi TensorFlow. Sesuai dengan nama fungsi, **clear\_session()** membantu menghapus semua opsional dari sesi TensorFlow, memungkinkan Anda untuk memulai dengan sesi yang bersih dan mencegah konflik dengan sesi sebelumnya. Ini umumnya berguna saat Anda mengembangkan model secara iteratif dalam Jupyter Notebook atau lingkungan serupa.

1. **Print TensorFlow Version:**

pythonCopy code

print(tf.\_\_version\_\_)

Ini mencetak versi TensorFlow yang saat ini diinstal. Hal ini bermanfaat untuk memastikan bahwa versi yang diinginkan atau yang diperlukan oleh suatu kode atau proyek tertentu telah diinstal. Dalam kasus ini, kode mencetak versi TensorFlow yang diharapkan, yaitu "2.0.0".

****

**1.Membuat Array NumPy a:**

a = np.array([[1, 0], [0, 1], [2, 3]])

Ini membuat array dua dimensi (**a**) dengan elemen-elemen yang ditentukan.

**2.Membuat Array NumPy b:**

b = np.array([[4, 1, 1], [2, 2, 1]])

Ini membuat array dua dimensi (**b**) dengan elemen-elemen yang ditentukan.

**3.Mencetak Matriks a:**

print(np.matrix(a))

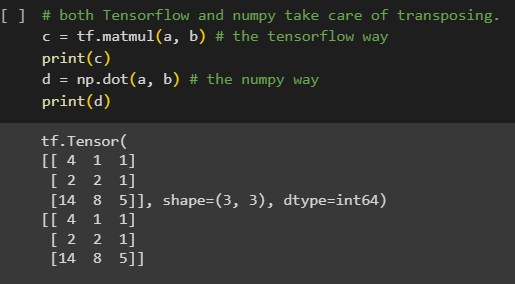
Ini mencetak matriks yang dihasilkan dari array ‘**a’** menggunakan fungsi **np.matrix**. Meskipun **np.matrix** tidak disarankan untuk digunakan dalam pengembangan baru, namun di sini digunakan untuk keperluan cetakan.

**4.Mencetak Matriks b:**

print(np.matrix(b))

Ini mencetak matriks yang dihasilkan dari array **b** menggunakan fungsi **np.matrix**.

jadi, keseluruhan kode menciptakan dua array NumPy dan mencetaknya sebagai matriks menggunakan **np.matrix**. Mencetak array sebagai matriks memudahkan untuk memvisualisasikan dan membaca struktur data dua dimensi.



**TensorFlow (tf):**

c = tf.matmul(a, b)

print("Hasil TensorFlow:")

print(c)

* **tf.matmul(a, b)**: Fungsi ini dari TensorFlow digunakan untuk melakukan perkalian matriks antara matriks **a** dan **b**.
* **print("Hasil TensorFlow:")**: Ini hanya mencetak teks untuk memberi tahu bahwa hasil yang dicetak setelah ini berasal dari TensorFlow.
* **print(c)**: Ini mencetak hasil perkalian matriks dari TensorFlow.

**NumPy (np):**

a\_np = np.array(a)

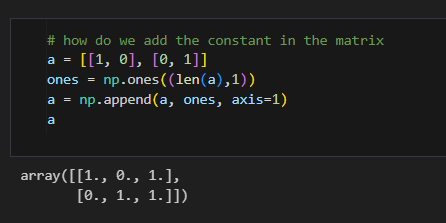
b\_np = np.array(b)

d = np.dot(a\_np, b\_np)

print("Hasil NumPy:")

print(d)

* **np.array(a)**: Mengonversi matriks TensorFlow **a** menjadi matriks NumPy **a\_np**.
* **np.array(b)**: Mengonversi matriks TensorFlow **b** menjadi matriks NumPy **b\_np**.
* **np.dot(a\_np, b\_np)**: Fungsi ini dari NumPy digunakan untuk melakukan perkalian matriks antara **a\_np** dan **b\_np**.
* **print("Hasil NumPy:")**: Ini hanya mencetak teks untuk memberi tahu bahwa hasil yang dicetak setelah ini berasal dari NumPy.
* **print(d)**: Ini mencetak hasil perkalian matriks dari NumPy.



**Inisialisasi Matriks:**

a = [[1, 0], [0, 1]]

Baris ini menginisialisasi matriks **a** berukuran 2x2 dengan nilai **[[1, 0], [0, 1]]**.

**Membuat Kolom Ones:**

ones = np.ones((len(a), 1))

Baris ini membuat vektor kolom dari ones dengan panjang yang sama dengan jumlah baris dalam matriks **a**. Variabel **ones** akan terlihat seperti:

[[1.]

[1.]]

**Menambahkan Kolom Ones ke Matriks:**

a = np.append(a, ones, axis=1)

Baris ini menambahkan vektor kolom ones ke matriks asli **a** sepanjang sumbu kedua (**axis=1**), efektif menambahkan kolom ones di sebelah kanan matriks. Setelah operasi ini, matriks **a** akan terlihat seperti:

[[1. 0. 1.]

[0. 1. 1.]]

Setiap baris sekarang memiliki elemen tambahan (1.0) di akhir.



import numpy as np

import matplotlib.pyplot as plt

def sigmoid(x, c, a):

z = ((x - c) / a)

return 1.0 / (1.0 + np.exp(-z))

x = np.linspace(-5.0, 5.0, 500) # Menghasilkan 500 titik secara merata dalam rentang dari -5 hingga 5

c = 1.0 # Parameter yang memindahkan fungsi sigmoid sepanjang sumbu x

a = 0.5 # Parameter yang mengontrol "ketajaman" fungsi sigmoid

# Plot fungsi sigmoid

plt.plot(x, sigmoid(x, c, a), label='sigmoid')

# Plot fungsi tangen hiperbolik (tanh) untuk perbandingan

plt.plot(x, np.tanh(x), label='tanh')

# Menambahkan garis grid dan legenda pada plot

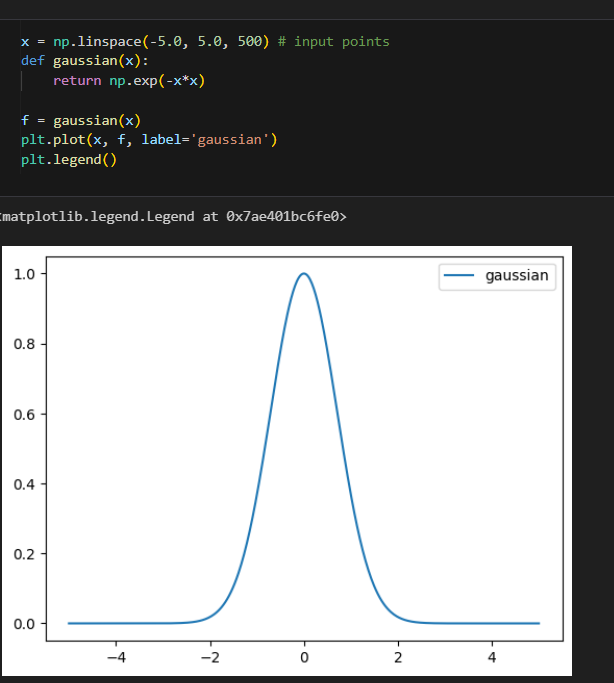
plt.grid()

plt.legend()

# Menampilkan plot

plt.show()

1. Kode tersebut mendefinisikan fungsi sigmoid dengan parameter **x**, **c**, dan **a**. Fungsi sigmoid adalah fungsi matematika yang memetakan nilai real apa pun ke nilai antara 0 dan 1, yang berguna dalam pembelajaran mesin dan jaringan saraf.
2. **np.linspace(-5.0, 5.0, 500)** menghasilkan 500 titik secara merata dalam rentang dari -5 hingga 5, yang akan digunakan sebagai nilai input (**x**) untuk fungsi sigmoid.
3. Kode kemudian menetapkan nilai untuk parameter **c** dan **a**. **c** adalah parameter yang menggeser fungsi sigmoid sepanjang sumbu x, dan **a** mengontrol "ketajaman" kurva sigmoid.
4. Fungsi sigmoid dipetakan menggunakan **plt.plot(x, sigmoid(x, c, a), label='sigmoid')**.
5. Selain itu, fungsi tangen hiperbolik (tanh) dipetakan untuk perbandingan menggunakan **plt.plot(x, np.tanh(x), label='tanh')**.



import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(-5.0, 5.0, 500) # Menghasilkan 500 titik secara merata dalam rentang dari -5 hingga 5

def gaussian(x):

return np.exp(-x\*x)

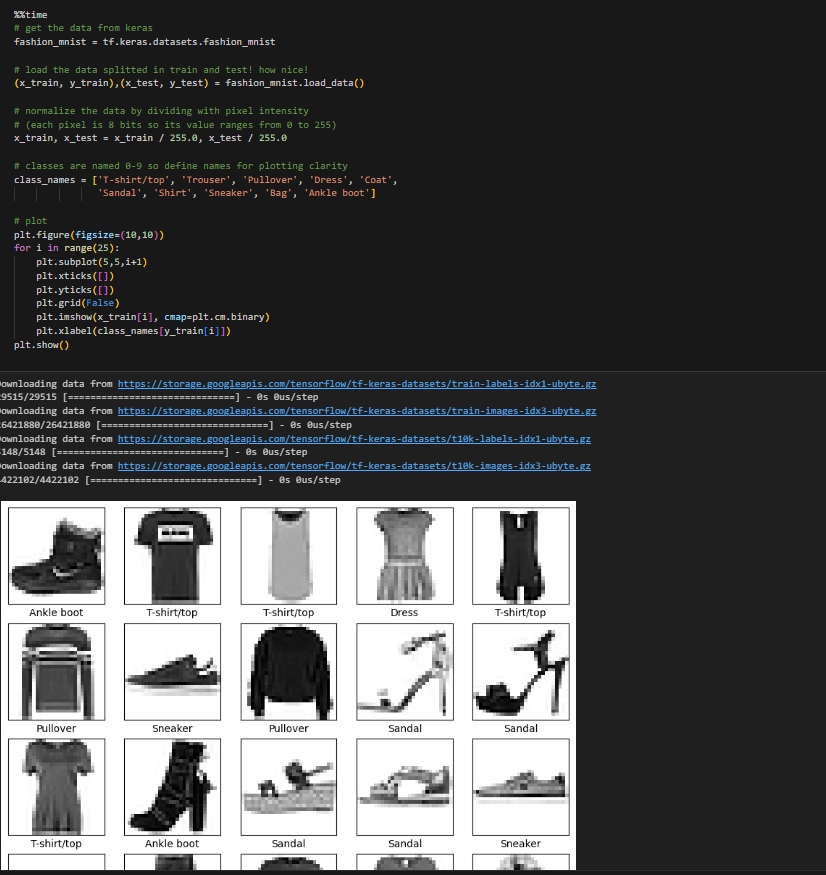
f = gaussian(x) # Menghitung nilai fungsi Gaussian untuk setiap nilai x

plt.plot(x, f, label='gaussian') # Membuat plot fungsi Gaussian dengan label 'gaussian'

plt.legend() # Menambahkan legenda pada plot

plt.show() # Menampilkan plot

1. **x = np.linspace(-5.0, 5.0, 500)**: Membuat array **x** yang berisi 500 nilai terdistribusi merata dalam rentang dari -5 hingga 5. Nilai-nilai ini akan digunakan sebagai input untuk fungsi Gaussian.
2. **def gaussian(x):**: Mendefinisikan fungsi Gaussian. Fungsi ini mengambil input **x** dan mengembalikan nilai dari fungsi Gaussian, yang didefinisikan sebagai **np.exp(-x\*x)**.
3. **f = gaussian(x)**: Menghitung nilai fungsi Gaussian untuk setiap nilai **x** yang telah dihasilkan.
4. **plt.plot(x, f, label='gaussian')**: Membuat plot fungsi Gaussian dengan menggunakan nilai **x** sebagai sumbu x dan nilai fungsi Gaussian sebagai sumbu y. Label 'gaussian' ditambahkan untuk keperluan legenda.
5. **plt.legend()**: Menambahkan legenda pada plot untuk memberikan informasi tentang elemen-elemen yang diplot.
6. **plt.show()**: Menampilkan plot secara keseluruhan.



%%time

# Mengukur waktu eksekusi sel ini

# Impor pustaka TensorFlow

import tensorflow as tf

Baris pertama menggunakan perintah khusus Jupyter Notebook (**%%time**) untuk mengukur waktu eksekusi sel ini.

Baris kedua mengimpor pustaka TensorFlow, yang merupakan sebuah kerangka kerja pembelajaran mesin yang populer.

# Dapatkan dataset Fashion MNIST dari Keras

\fashion\_mnist = tf.keras.datasets.fashion\_mnist

Baris ini menggunakan antarmuka Keras pada TensorFlow untuk mendapatkan dataset Fashion MNIST. Dataset ini berisi gambar-gambar skala abu-abu dari 10 jenis pakaian yang berbeda.

# Muat data yang sudah dibagi menjadi set pelatihan dan pengujian

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

Data Fashion MNIST dimuat ke dalam set pelatihan dan pengujian. Variabel **x\_train** berisi gambar-gambar pelatihan, dan **y\_train** berisi label-label yang sesuai. Hal yang sama berlaku untuk set pengujian.

# Normalisasi data dengan membagi intensitas piksel

# (setiap piksel memiliki nilai 8 bit sehingga nilainya berkisar antara 0 hingga 255)

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

Data dinormalisasi dengan membagi setiap nilai piksel dengan 255.0 sehingga intensitas piksel berada dalam rentang antara 0 dan 1.

# Kelas-kelas dinamai 0-9, jadi tentukan nama-nama untuk kejelasan plotting

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Deklarasi kelas-kelas yang merepresentasikan jenis-jenis pakaian dalam dataset.

pythonCopy code

# Plot

plt.figure(figsize=(10, 10))

for i in range(25):

plt.subplot(5, 5, i+1)

plt.xticks([])

plt.yticks([])

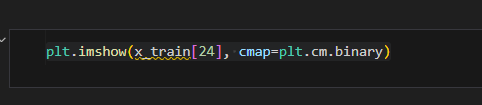
plt.grid(False)

plt.imshow(x\_train[i], cmap=plt.cm.binary)

plt.xlabel(class\_names[y\_train[i]])

plt.show()

Membuat plot untuk menampilkan 25 gambar pertama dari set pelatihan. Setiap gambar diberi label sesuai dengan kelas pakaian yang sesuai.



import matplotlib.pyplot as plt

# Menampilkan gambar dari x\_train pada indeks ke-24

plt.imshow(x\_train[24], cmap=plt.cm.binary)

# Menambahkan label sumbu x

plt.xlabel('Label X')

# Menambahkan label sumbu y

plt.ylabel('Label Y')

# Menambahkan judul gambar

plt.title('Judul Gambar')

# Menampilkan warna bar untuk skala

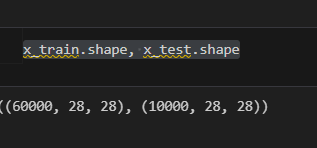
plt.colorbar()

# Menampilkan gambar

plt.show()

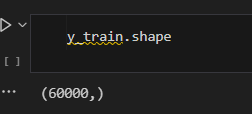
Penjelasan singkat:

* **plt.imshow()**: Fungsi ini digunakan untuk menampilkan gambar. **cmap=plt.cm.binary** menunjukkan penggunaan peta warna biner.
* **plt.xlabel()**, **plt.ylabel()**, dan **plt.title()**: Digunakan untuk menambahkan label sumbu x, sumbu y, dan judul gambar.
* **plt.colorbar()**: Menambahkan warna bar untuk skala warna pada peta warna.
* **plt.show()**: Menampilkan gambar.



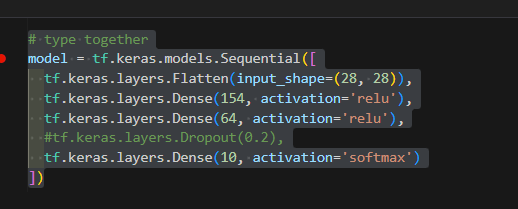
1. **x\_train.shape**: Ini mengambil bentuk dari larik atau tensor **x\_train**. Dalam machine learning, **x\_train** sering kali mewakili fitur input dari dataset pelatihan. Bentuk ini umumnya mencakup informasi tentang jumlah sampel (baris) dan jumlah fitur (kolom) dalam dataset.
2. **x\_test.shape**: Begitu pula, ini mengambil bentuk dari larik atau tensor **x\_test**, yang biasanya mewakili fitur input dari dataset pengujian. Bentuk ini memberikan informasi tentang jumlah sampel dan fitur dalam dataset pengujian.

Sebagai contoh, jika Anda bekerja dengan larik NumPy, bentuknya bisa terlihat seperti **(jumlah\_sampel, jumlah\_fitur)**.



**y\_train.shape** adalah kode yang digunakan untuk mendapatkan bentuk (shape) dari array atau tensor **y\_train**. Dalam konteks machine learning, **y\_train** biasanya merupakan array atau tensor yang berisi label atau target yang sesuai dengan setiap sampel pada dataset pelatihan.

Dengan menggunakan **y\_train.shape**, Anda dapat mengetahui jumlah dimensi dan ukuran dari array atau tensor tersebut. Informasi ini penting karena membantu Anda memahami struktur dari data target yang digunakan untuk melatih model machine learning.



# Importing TensorFlow library

import tensorflow as tf

# Defining the neural network model

model = tf.keras.models.Sequential([

# Flatten layer to convert 2D input (28x28) to 1D array (28\*28 = 784)

tf.keras.layers.Flatten(input\_shape=(28, 28)),

# Dense layer with 154 neurons and ReLU activation function

tf.keras.layers.Dense(154, activation='relu'),

# Dense layer with 64 neurons and ReLU activation function

tf.keras.layers.Dense(64, activation='relu'),

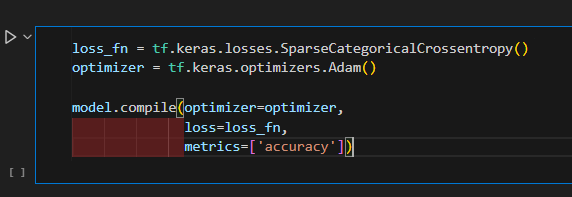
# Dense layer with 10 neurons (for 10 classes) and softmax activation function

tf.keras.layers.Dense(10, activation='softmax')

])

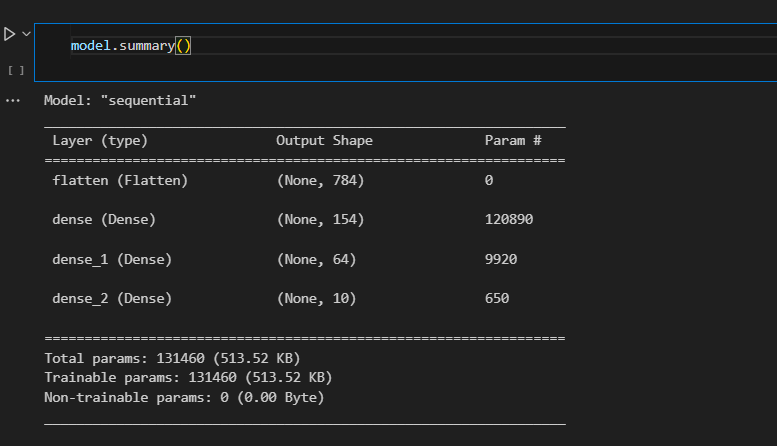
ode tersebut mendefinisikan sebuah model jaringan saraf menggunakan TensorFlow's Keras API. Mari kita bahas tujuan dari setiap bagian:

1. **Flatten Layer:**
   * **Input Shape:** (28, 28)
   * Flatten layer adalah lapisan input. Digunakan untuk meratakan data input 2D (28x28) menjadi larik 1D berukuran 784. Hal ini diperlukan sebelum meneruskan data ke lapisan fully connected (Dense).
2. **Dense Layer (Hidden Layer):**
   * **Jumlah Neuron:** 154
   * **Activation Function:** ReLU (Rectified Linear Unit)
   * Ini adalah lapisan tersembunyi pertama dengan 154 neuron dan fungsi aktivasi ReLU. Fungsi aktivasi ReLU memperkenalkan non-linearitas ke dalam model, memungkinkannya untuk mempelajari pola yang kompleks dalam data.
3. **Dense Layer (Hidden Layer):**
   * **Jumlah Neuron:** 64
   * **Activation Function:** ReLU
   * Ini adalah lapisan tersembunyi kedua dengan 64 neuron dan fungsi aktivasi ReLU. Sekali lagi, ini menambahkan non-linearitas ke dalam model.
4. **Dense Layer (Output Layer):**
   * **Jumlah Neuron:** 10
   * **Activation Function:** Softmax
   * Ini adalah lapisan output dengan 10 neuron, masing-masing mewakili kelas dalam tugas klasifikasi. Fungsi aktivasi softmax digunakan untuk mendapatkan probabilitas untuk setiap kelas. Ini umumnya digunakan dalam masalah klasifikasi multi-kelas.



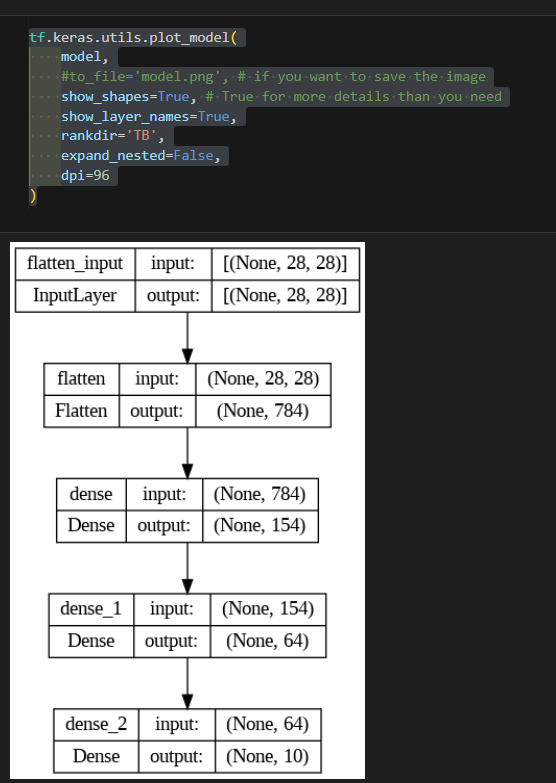
1. **Sparse Categorical Crossentropy Loss Function (tf.keras.losses.SparseCategoricalCrossentropy()):**
   * **Fungsi:** Ini adalah fungsi kerugian yang digunakan dalam pelatihan model klasifikasi multikelas ketika label target adalah bilangan bulat (seperti 0, 1, 2, dst.).
   * **Penjelasan Lebih Lanjut:** Fungsi ini membandingkan distribusi probabilitas yang dihasilkan oleh model dengan label kelas yang sebenarnya. Itu menghitung kerugian berdasarkan perbedaan antara distribusi probabilitas prediksi dan distribusi one-hot encoded dari label target. Dengan menggunakan **SparseCategoricalCrossentropy**, Anda tidak perlu mengonversi label target menjadi representasi one-hot encoded.
2. **Adam Optimizer (tf.keras.optimizers.Adam()):**
   * **Fungsi:** Ini adalah algoritma optimisasi yang populer dalam pelatihan model machine learning. Adam (Adaptive Moment Estimation) adalah metode optimisasi yang menggabungkan konsep dari metode-metode lain seperti Momentum dan RMSprop.
   * **Penjelasan Lebih Lanjut:** Adam mengadaptasi laju pembelajaran (learning rate) untuk setiap parameter model berdasarkan momentum dari gradien pertama (mean) dan gradien kedua (uncentered variance). Ini membantu mempercepat pelatihan dan dapat menangani masalah laju pembelajaran yang berbeda-beda untuk setiap parameter.
3. **Compile Model:**
   * **Fungsi:** Saat Anda memanggil **compile** pada model Keras, Anda mengkonfigurasi proses pelatihan model dengan menentukan optimizer, fungsi kerugian, dan metrik evaluasi.
   * **Penjelasan Lebih Lanjut:** Dalam contoh kode Anda, model menggunakan Adam sebagai optimizer, Sparse Categorical Crossentropy sebagai fungsi kerugian, dan akan dievaluasi berdasarkan akurasi (accuracy) selama pelatihan.

Dengan mengonfigurasi model seperti ini, Anda siap untuk melatihnya menggunakan data pelatihan yang sesuai. Selanjutnya, Anda bisa menggunakan metode **fit** untuk melatih model dengan data tersebut.



Fungsi dari **model.summary()** adalah untuk memberikan ringkasan yang komprehensif tentang arsitektur model. Output ringkasan ini sangat berguna untuk memahami struktur model, memeriksa jumlah parameter yang dapat di-train, dan melihat ukuran output dari setiap layer. Dengan mengeksekusi **model.summary()**, Anda akan mendapatkan informasi berikut:

1. **Layer Names:** Daftar layer dalam model beserta nama masing-masing layer.
2. **Layer Type:** Jenis layer, seperti Dense (fully connected), Conv2D (convolutional), Activation, dll.
3. **Output Shape:** Ukuran output dari setiap layer. Ini memberikan wawasan tentang bagaimana dimensi data berubah saat melewati setiap layer.
4. **Param #:** Jumlah parameter yang dapat di-train untuk setiap layer. Ini mencakup bobot (weights) dan bias dalam layer tersebut.
5. **Trainable:** Menunjukkan apakah layer tersebut dapat di-train atau tidak. Beberapa layer, seperti layer aktivasi, mungkin tidak memiliki parameter yang dapat di-train.



Fungsi dari `tf.keras.utils.plot\_model` adalah untuk membuat visualisasi grafis dari arsitektur model. Dengan memanggil fungsi ini, Anda dapat menghasilkan diagram yang menunjukkan bagaimana layer-layer dalam model terhubung satu sama lain. Berikut adalah penjelasan dari parameter yang Anda set:

- `model`: Model yang akan divisualisasikan.

- `show\_shapes=True`: Menunjukkan bentuk (shape) output dari setiap layer.

- `show\_layer\_names=True`: Menunjukkan nama-nama layer pada diagram.

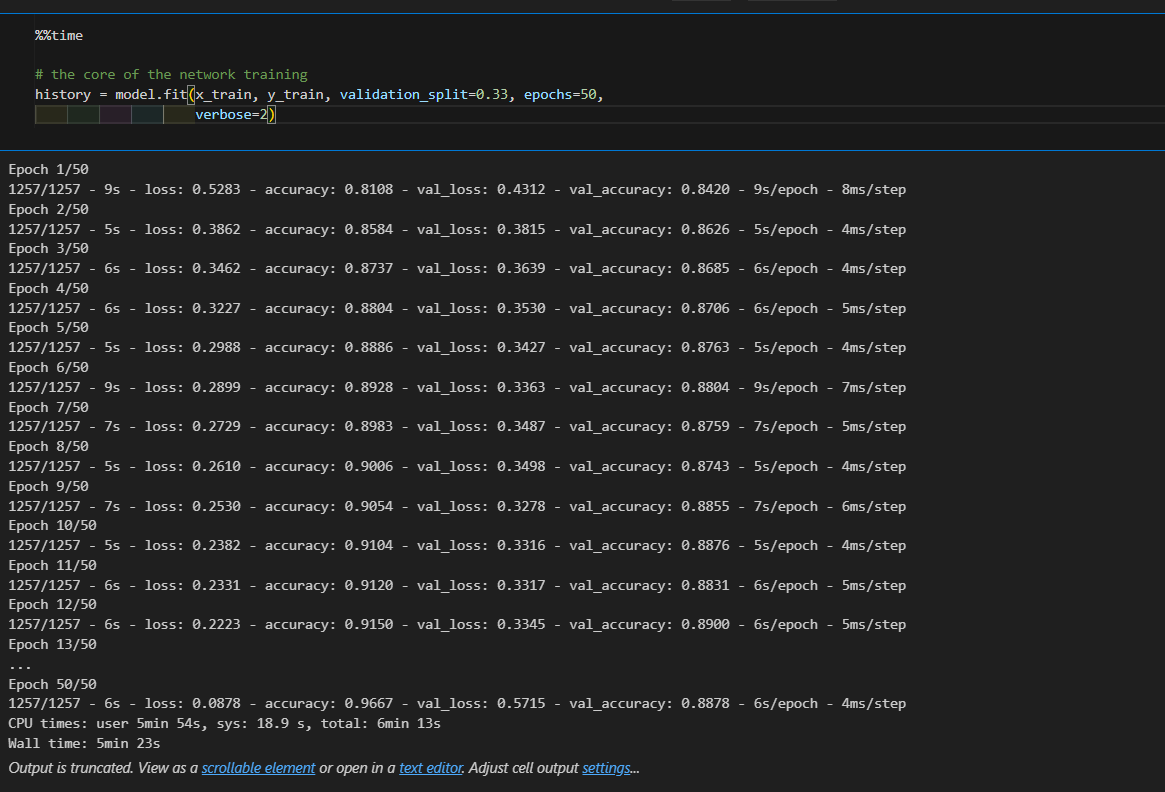
- `rankdir='TB'`: Menentukan arah penempatan layer. 'TB' berarti dari atas ke bawah (top to bottom).

- `expand\_nested=False`: Menunjukkan apakah layer-layer yang bersarang (nested) harus di-expand dalam diagram atau tidak.

- `dpi=96`: Menentukan resolusi gambar dalam dot per inch (DPI).

Anda juga memiliki opsi untuk menyimpan diagram ke file gambar dengan menggunakan parameter `to\_file` dan memberikan nama file serta format (misalnya, 'model.png'). Namun, pada contoh kode yang Anda berikan, opsi ini di-comment sehingga diagram tidak akan disimpan sebagai file gambar.

Dengan memanfaatkan fungsi ini, Anda dapat dengan mudah memvisualisasikan arsitektur model Anda untuk memahami strukturnya dan memastikan bahwa model telah dibangun sesuai dengan yang Anda harapkan.



**%%time** adalah magic command di Jupyter Notebook yang digunakan untuk mengukur waktu eksekusi sel di notebook. Dengan menambahkan **%%time** di awal sel, Anda dapat melihat waktu yang dibutuhkan untuk mengeksekusi sel tersebut.

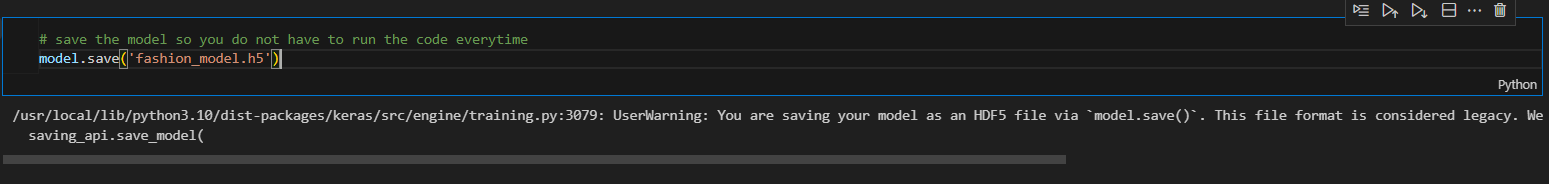
# the core of the network training

history = model.fit(x\_train, y\_train, validation\_split=0.33, epochs=50, verbose=2)

Kode di atas adalah inti dari proses pelatihan model. Mari jelaskan beberapa parameter yang digunakan:

* **x\_train** dan **y\_train**: Data latih, di mana **x\_train** adalah fitur (input) dan **y\_train** adalah label (output).
* **validation\_split=0.33**: Parameter ini menentukan seberapa banyak data yang akan digunakan sebagai data validasi selama pelatihan. Dalam hal ini, 33% dari data latih (**x\_train** dan **y\_train**) akan diambil sebagai data validasi.
* **epochs=50**: Jumlah epoch atau iterasi penuh melalui seluruh data latih. Model akan melihat seluruh dataset sebanyak 50 kali selama proses pelatihan.
* **verbose=2**: Ini mengatur tingkat detail output selama pelatihan. **verbose=2** berarti akan menampilkan detail bar progress untuk setiap epoch.

Hasil dari pelatihan model akan disimpan dalam variabel **history**, yang biasanya berisi informasi seperti nilai loss dan metrik lainnya untuk setiap epoch, sehingga Anda dapat menganalisis bagaimana performa model Anda berkembang selama pelatihan.



Kode yang Anda berikan menunjukkan penggunaan fungsi **save** dari Keras untuk menyimpan model machine learning ke dalam file dengan format H5 (HDF5), yang umumnya digunakan untuk menyimpan model Keras.

Mari kita bahas lebih lanjut:

1. **Model:**
   * **model**: Merupakan objek model machine learning yang telah Anda definisikan dan latih menggunakan Keras.
2. **Fungsi save:**
   * **save** adalah metode yang disediakan oleh Keras untuk menyimpan model ke dalam file.
   * Format file yang digunakan di sini adalah H5 (HDF5), format yang umumnya digunakan untuk menyimpan data besar dan kompleks.
3. **Nama File:**
   * **'fashion\_model.h5'**: Ini adalah nama file yang akan digunakan untuk menyimpan model. Nama file dapat disesuaikan sesuai keinginan Anda.
4. **Tujuan:**
   * Dengan menyimpan model ke dalam file, Anda dapat menghindari kebutuhan untuk menjalankan kode pelatihan setiap kali Anda ingin menggunakan model tersebut.
   * File H5 menyimpan arsitektur model, konfigurasi pelatihan, dan bobot model sehingga dapat digunakan kembali tanpa harus melatih ulang.

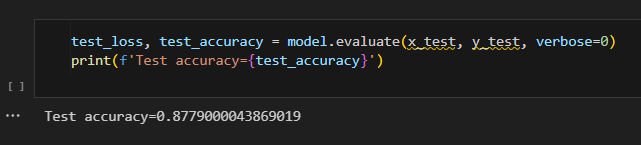
Jika Anda ingin menggunakan kembali model yang telah Anda simpan, Anda dapat melakukannya dengan mengimpor model dari file yang telah Anda simpan, seperti ini:

from keras.models import load\_model

# Memuat model dari file

loaded\_model = load\_model('fashion\_model.h5')

Setelah memuat model, Anda dapat menggunakannya untuk membuat prediksi atau melanjutkan pelatihan jika diperlukan.

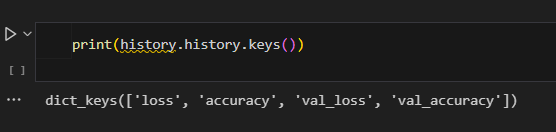


test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test, verbose=0)

print(f'Akurasi Pengujian={test\_accuracy}')

1. **model.evaluate(x\_test, y\_test, verbose=0):**
   * Baris ini mengevaluasi model pembelajaran mesin yang telah di-training menggunakan dataset pengujian.
   * **x\_test**: Mewakili fitur-fitur input dari dataset pengujian.
   * **y\_test**: Mewakili label atau nilai sebenarnya dari dataset pengujian.
   * **verbose=0**: Mengontrol tingkat kejelasan selama proses evaluasi. Tingkat kejelasan 0 berarti tanpa keluaran selama evaluasi. Jika Anda atur **verbose=1**, maka akan muncul bar kemajuan selama evaluasi.
2. **test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test, verbose=0):**
   * Metode **evaluate** mengembalikan nilai kerugian (loss) dan akurasi dari pengujian, yang kemudian disimpan dalam variabel **test\_loss** dan **test\_accuracy** secara berturut-turut.
3. **print(f'Akurasi Pengujian={test\_accuracy}'):**
   * Baris ini mencetak nilai akurasi pengujian ke konsol. Akurasi adalah metrik umum yang digunakan untuk mengevaluasi kinerja model klasifikasi. Ini mencerminkan rasio prediksi yang benar terhadap total instance dalam dataset pengujian.

Secara ringkas, potongan kode ini digunakan untuk menilai seberapa baik model telah melakukan pada dataset pengujian.



Ketika Anda menggunakan objek **history** setelah melatih model dengan Keras, **history** tersebut berisi informasi tentang metrik dan kerugian (loss) pada setiap epoch selama proses pelatihan. Pemanggilan **keys()** pada objek **history.history** memberikan daftar kunci yang menyediakan akses ke informasi tersebut.

Berikut adalah beberapa kunci umum yang mungkin muncul setelah melakukan pelatihan model:

* **'loss'**: Nilai kerugian pada setiap epoch selama pelatihan pada data pelatihan.
* **'accuracy'**: Akurasi pada setiap epoch selama pelatihan pada data pelatihan.
* **'val\_loss'**: Nilai kerugian pada setiap epoch selama pelatihan pada data validasi (jika menggunakan data validasi).
* **'val\_accuracy'**: Akurasi pada setiap epoch selama pelatihan pada data validasi (jika menggunakan data validasi).

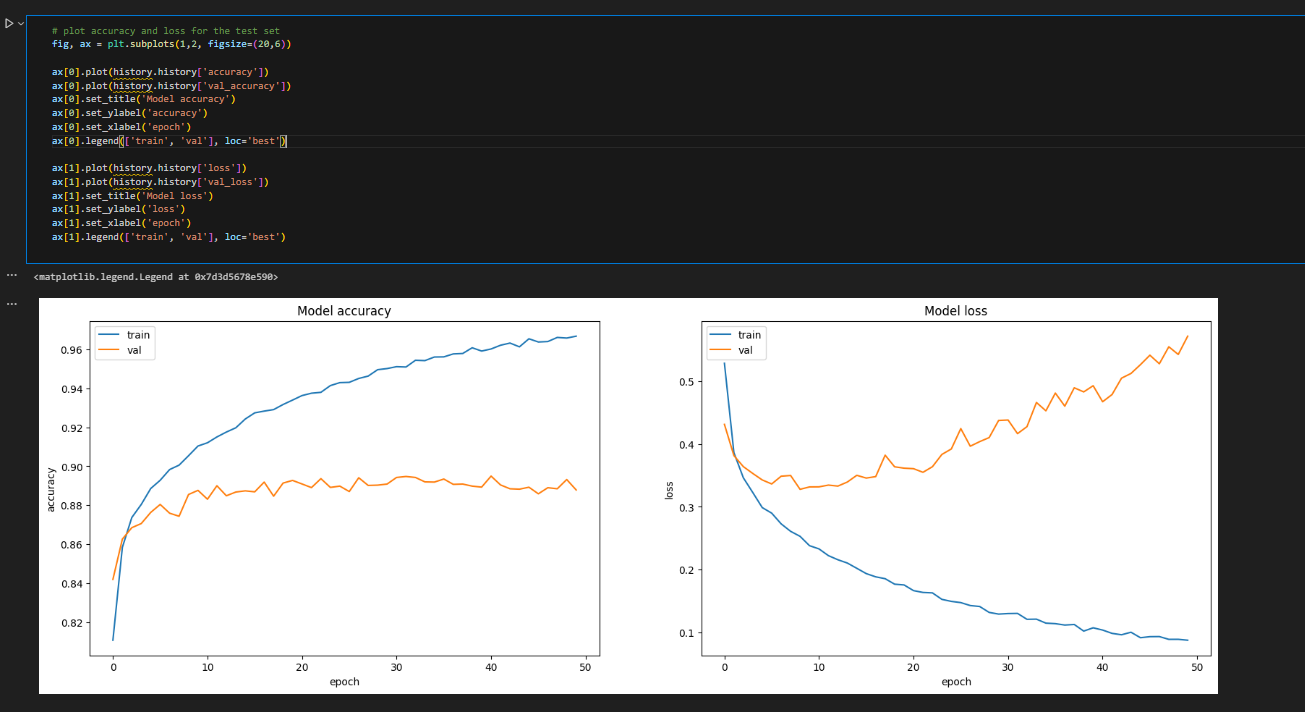
Sebagai contoh, jika Anda ingin mengetahui perkembangan nilai kerugian selama pelatihan, Anda dapat menggunakan:

print(history.history['loss'])

Demikian pula, jika Anda ingin mengetahui akurasi pada setiap epoch, Anda dapat menggunakan:

print(history.history['accuracy'])

nformasi ini membantu Anda memahami bagaimana model Anda belajar dari data pelatihan dan seberapa baik model tersebut generalis pada data yang tidak terlihat selama pelatihan (data validasi).



Tentu, mari kita bahas fungsi-fungsi utama dari kode tersebut:

1. \*\*Grafik Akurasi Model (`ax[0]`):\*\*

- `ax[0].plot(history.history['accuracy'])`: Menambahkan garis plot untuk akurasi pada set pelatihan pada sumbu pertama (indeks 0) dari subplot.

- `ax[0].plot(history.history['val\_accuracy'])`: Menambahkan garis plot untuk akurasi pada set validasi pada sumbu pertama dari subplot.

- `ax[0].set\_title('Model accuracy')`: Menetapkan judul untuk subplot pertama sebagai 'Model accuracy'.

- `ax[0].set\_ylabel('accuracy')`: Menetapkan label sumbu y sebagai 'accuracy'.

- `ax[0].set\_xlabel('epoch')`: Menetapkan label sumbu x sebagai 'epoch'.

- `ax[0].legend(['train', 'val'], loc='best')`: Menambahkan legenda yang menjelaskan bahwa garis pertama adalah data pelatihan ('train') dan garis kedua adalah data validasi ('val'), dan menempatkannya di lokasi terbaik (`loc='best'`).

2. \*\*Grafik Kerugian Model (`ax[1]`):\*\*

- `ax[1].plot(history.history['loss'])`: Menambahkan garis plot untuk kerugian pada set pelatihan pada sumbu kedua (indeks 1) dari subplot.

- `ax[1].plot(history.history['val\_loss'])`: Menambahkan garis plot untuk kerugian pada set validasi pada sumbu kedua dari subplot.

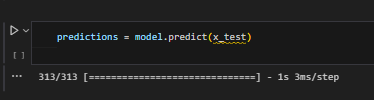
- `ax[1].set\_title('Model loss')`: Menetapkan judul untuk subplot kedua sebagai 'Model loss'.

- `ax[1].set\_ylabel('loss')`: Menetapkan label sumbu y sebagai 'loss'.

- `ax[1].set\_xlabel('epoch')`: Menetapkan label sumbu x sebagai 'epoch'.

- `ax[1].legend(['train', 'val'], loc='best')`: Menambahkan legenda yang menjelaskan bahwa garis pertama adalah data pelatihan dan garis kedua adalah data validasi, dan menempatkannya di lokasi terbaik.

Fungsi-fungsi ini digunakan untuk membuat visualisasi yang dapat membantu dalam memahami sejauh mana model belajar selama pelatihan (akurasi) dan seberapa baik model tersebut dapat mengurangi kesalahan (kerugian/loss) pada kedua set data tersebut. Dengan memonitor kedua metrik ini selama beberapa epoch, kita dapat mengevaluasi dan memahami performa model kita.



Baris kode `predictions = model.predict(x\_test)` digunakan untuk menghasilkan prediksi dari model pembelajaran mesin yang telah dilatih sebelumnya. Mari kita bahas fungsi dari baris kode ini:

- \*\*`model`\*\*: Ini adalah model pembelajaran mesin yang telah diinisialisasi dan dilatih sebelumnya. Model ini dapat menjadi model jaringan saraf (neural network), pohon keputusan, mesin vektor dukungan (support vector machine), atau jenis model pembelajaran mesin lainnya, tergantung pada implementasi Anda.

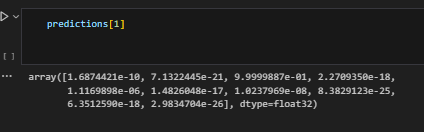
- \*\*`predict`\*\*: Metode `predict` digunakan untuk membuat prediksi berdasarkan input yang diberikan. Dalam konteks ini, `model.predict(x\_test)` akan menghasilkan prediksi untuk data uji (`x\_test`).

- \*\*`x\_test`\*\*: Variabel ini umumnya digunakan untuk menyimpan data uji, yaitu data yang tidak digunakan selama proses pelatihan model dan akan digunakan untuk menguji seberapa baik model dapat melakukan prediksi pada data yang belum pernah dilihat sebelumnya.

- \*\*`predictions`\*\*: Setelah baris kode dieksekusi, variabel `predictions` akan berisi hasil prediksi yang dihasilkan oleh model untuk setiap contoh dalam data uji.

Sebagai contoh, jika Anda memiliki model klasifikasi yang dilatih untuk mengenali gambar angka tulisan tangan (seperti pada dataset MNIST), `predictions` akan berisi probabilitas atau kelas prediksi untuk setiap gambar di `x\_test`.

Contoh penggunaan selanjutnya mungkin termasuk evaluasi performa model dengan membandingkan prediksi dengan label sebenarnya pada data uji atau melakukan visualisasi untuk memahami sejauh mana model berhasil melakukan prediksi.



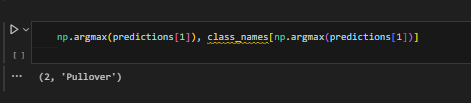
Dengan menulis `predictions[1]`, Anda mengakses hasil prediksi dari model untuk contoh kedua dalam dataset uji. Mari kita jelaskan fungsinya:

- \*\*`predictions`\*\*: Ini adalah variabel yang telah Anda tetapkan sebelumnya dan berisi hasil prediksi dari model untuk seluruh dataset uji.

- \*\*`[1]`\*\*: Ini adalah indeks. Dalam konteks ini, `predictions[1]` berarti Anda ingin mengakses hasil prediksi untuk contoh kedua dalam dataset uji. Perlu diingat bahwa indeks array dimulai dari 0, jadi `predictions[1]` merujuk pada elemen kedua dalam array (indeks 1).

Sebagai contoh, jika kita anggap `predictions` adalah array numpy dan model kita melakukan klasifikasi dengan output dalam bentuk probabilitas (misalnya, untuk klasifikasi biner), maka `predictions[1]` dapat berisi nilai probabilitas atau skor prediksi untuk contoh kedua.

Contoh penggunaan lebih lanjut bisa mencakup membandingkan nilai prediksi ini dengan label yang sebenarnya (ground truth) untuk mengukur seberapa baik model Anda berperforma pada contoh tersebut.



Mari kita bahas fungsi dari potongan kode `np.argmax(predictions[1])`:

- \*\*`predictions[1]`\*\*: Ini merujuk pada hasil prediksi untuk contoh kedua dalam dataset uji, seperti yang telah dijelaskan sebelumnya.

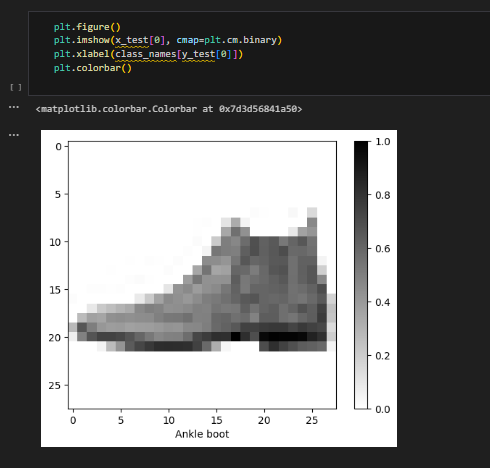
- \*\*`np.argmax()`\*\*: Ini adalah fungsi dari pustaka NumPy yang digunakan untuk mengembalikan indeks dari nilai terbesar dalam suatu array.

Dengan menggabungkan keduanya, `np.argmax(predictions[1])` mengembalikan indeks di mana nilai prediksi mencapai nilai maksimum untuk contoh kedua. Ini sering digunakan dalam konteks klasifikasi, di mana output model mungkin berupa distribusi probabilitas untuk setiap kelas.

- \*\*`class\_names[np.argmax(predictions[1])]`\*\*: Jika `class\_names` adalah suatu daftar atau array yang berisi label kelas yang sesuai dengan indeks di output model, potongan ini akan mengambil label kelas yang memiliki probabilitas tertinggi atau skor prediksi tertinggi untuk contoh kedua.

Sebagai contoh, jika `np.argmax(predictions[1])` menghasilkan nilai 3 dan `class\_names` adalah daftar yang berisi label kelas, maka `class\_names[np.argmax(predictions[1])]` akan memberikan label kelas yang sesuai dengan prediksi tertinggi untuk contoh kedua.

Contoh penggunaan lebih lanjut bisa termasuk mengevaluasi apakah prediksi model sesuai dengan label sebenarnya untuk contoh tersebut atau menghasilkan output yang lebih mudah dibaca, seperti label kelas yang dapat ditampilkan.



Mari kita bahas fungsi dari potongan kode tersebut:

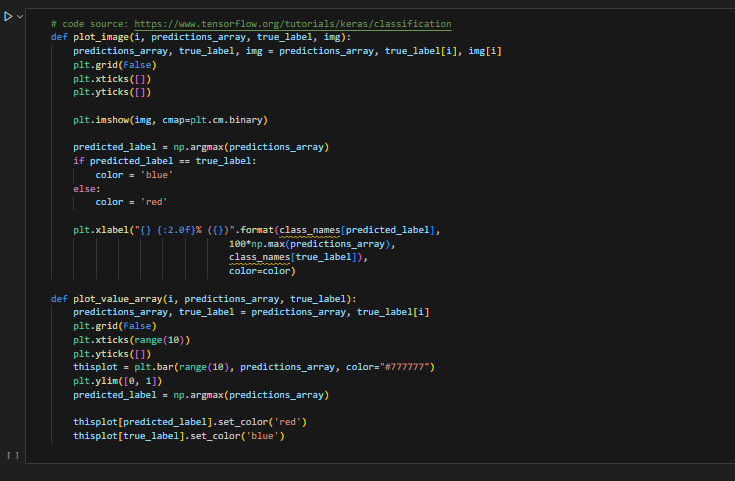
1. \*\*`plt.figure()`\*\*: Membuat gambar (figure) baru untuk menampung plot atau visualisasi.

2. \*\*`plt.imshow(x\_test[0], cmap=plt.cm.binary)`\*\*: Menampilkan gambar pertama dari dataset uji (`x\_test[0]`). Fungsi `imshow` digunakan untuk menampilkan gambar, dan parameter `cmap=plt.cm.binary` menunjukkan bahwa gambar tersebut akan ditampilkan dengan skala warna biner (hitam-putih). `x\_test[0]` adalah piksel nilai intensitas dari gambar pertama.

3. \*\*`plt.xlabel(class\_names[y\_test[0]])`\*\*: Menambahkan label sumbu x pada plot. Labelnya diambil dari `class\_names` dengan indeks yang sesuai dengan label kelas pada `y\_test[0]`. Ini memberikan informasi tentang kelas yang seharusnya diwakili oleh gambar pertama dalam dataset uji.

4. \*\*`plt.colorbar()`\*\*: Menambahkan batang warna (colorbar) ke plot. Colorbar ini memberikan informasi tentang nilai intensitas piksel pada gambar. Dalam konteks ini, karena gambar ditampilkan dalam skala warna biner, colorbar mungkin memberikan informasi tentang nilai hitam (0) dan nilai putih (1) pada gambar.

Secara keseluruhan, potongan kode ini digunakan untuk secara visual mengeksplorasi dan memahami gambar pertama dalam dataset uji, serta label kelas yang sesuai dan nilai intensitas pikselnya. Colorbar dapat memberikan konteks tambahan tentang nilai piksel dalam gambar hitam-putih tersebut.



Fungsi `plot\_image` ini memiliki beberapa tujuan untuk membantu visualisasi hasil prediksi dari model klasifikasi. Berikut penjelasan fungsi-fungsi utamanya:

- \*\*Input Parameters:\*\*

- `i`: Indeks contoh gambar yang akan divisualisasikan.

- `predictions\_array`: Array prediksi model untuk contoh gambar tersebut.

- `true\_label`: Label sebenarnya dari contoh gambar tersebut.

- `img`: Data piksel gambar tersebut.

- \*\*Set-Up Plot:\*\*

- `plt.grid(False)`: Menghilangkan garis-garis grid pada plot.

- `plt.xticks([])` dan `plt.yticks([])`: Menghilangkan penanda sumbu x dan y.

- \*\*Menampilkan Gambar:\*\*

- `plt.imshow(img, cmap=plt.cm.binary)`: Menampilkan gambar dalam skala warna biner (hitam-putih).

- \*\*Label dan Warna:\*\*

- `predicted\_label = np.argmax(predictions\_array)`: Mencari indeks kelas dengan nilai prediksi tertinggi.

- `if predicted\_label == true\_label:`: Jika prediksi model sesuai dengan label sebenarnya.

- `color = 'blue'`: Warna label teks ditetapkan menjadi biru.

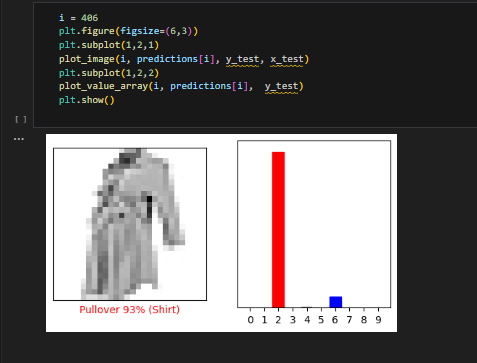
- `else:`: Jika prediksi model tidak sesuai dengan label sebenarnya.

- `color = 'red'`: Warna label teks ditetapkan menjadi merah.

- \*\*Menambahkan Label pada Plot:\*\*

- `plt.xlabel("{} {:2.0f}% ({})".format(class\_names[predicted\_label], 100\*np.max(predictions\_array), class\_names[true\_label]), color=color)`: Menambahkan teks label pada sumbu x. Label ini mencakup kelas prediksi, nilai probabilitas tertinggi, dan label sebenarnya. Warna teks disesuaikan berdasarkan kesesuaian prediksi.

Fungsi ini dirancang untuk memvisualisasikan gambar, prediksi, dan label sebenarnya dalam suatu plot untuk membantu pemahaman hasil model klasifikasi pada contoh tertentu.



Kode tersebut menggunakan pustaka Matplotlib di Python untuk membuat gambar dengan dua panel (subplot). Mari kita bahas setiap bagian dari kode tersebut:

i = 406

plt.figure(figsize=(6,3))

* **i = 406**: Menginisialisasi variabel **i** dengan nilai 406. Nilai ini kemungkinan merupakan indeks yang akan digunakan nanti.
* **plt.figure(figsize=(6,3))**: Membuat objek gambar (figure) dengan lebar 6 inci dan tinggi 3 inci. Ini mengatur ukuran gambar yang akan dibuat.

python  
plt.subplot(1,2,1)

plot\_image(i, predictions[i], y\_test, x\_test)

* **plt.subplot(1,2,1)**: Membuat subplot dengan 1 baris dan 2 kolom, dan memilih subplot pertama. Ini berarti kita akan memiliki dua subplot sejajar, dan yang pertama akan digunakan untuk menampilkan gambar.
* **plot\_image(i, predictions[i], y\_test, x\_test)**: Ini mungkin adalah sebuah fungsi yang tidak terlihat dalam potongan kode Anda, tetapi kita bisa menduga bahwa ini digunakan untuk menampilkan gambar dan prediksi terkait pada subplot pertama.

plt.subplot(1,2,2)

plot\_value\_array(i, predictions[i], y\_test)

* **plt.subplot(1,2,2)**: Memilih subplot kedua dari subplot yang sama. Subplot kedua digunakan untuk menampilkan sebuah array nilai (value array).
* **plot\_value\_array(i, predictions[i], y\_test)**: Sama seperti sebelumnya, ini mungkin adalah sebuah fungsi yang tidak terlihat dalam potongan kode Anda, tetapi kita bisa menduga bahwa ini digunakan untuk menampilkan array nilai pada subplot kedua.

plt.show()

* **plt.show()**: Menampilkan gambar yang telah dibuat dengan subplot-subplotnya.

Jadi, secara keseluruhan, potongan kode ini digunakan untuk membuat gambar dengan dua subplot, di mana subplot pertama menampilkan gambar dan prediksi terkait, sedangkan subplot kedua menampilkan array nilai. Fungsi-fungsi **plot\_image** dan **plot\_value\_array** yang digunakan untuk menampilkan informasi pada subplot mungkin harus didefinisikan di tempat lain dalam kode Anda.