

PEMBELAJARAN MESIN DAN PEMBELAJARAN MENDALAM

Modul 4

Unsupervised Learning

Yohanes Sigit Purnomo Wuryo Putro, ST., M.Kom., Ph.D.
Aloysius Gonzaga Pradnya Sidhawara, S.T., M.Eng.
Atanasius Surya Gunadharma
Maria

Tugas Modul Deep Learning

Dalam rangka meningkatkan kemampuan dalam klasifikasi gambar, tim pengembangan model di sini berupaya untuk mengembangkan sistem yang dapat memprediksi jenis pakaian berdasarkan gambar menggunakan pendekatan **Deep Learning**. Sistem ini akan dilatih untuk mengenali berbagai jenis pakaian, seperti t-shirt, celana jeans, sepatu, dan lainnya, menggunakan dataset **Fashion MNIST**.

Untuk mencapai tujuan ini, kita akan menggunakan model **Deep Learning** dengan **TensorFlow** dan **scikit-learn** untuk klasifikasi gambar. Dataset **Fashion MNIST** terdiri dari gambar-gambar pakaian dalam format 28x28 piksel yang telah diberi label sesuai dengan kategori pakaian yang dimiliki. Dengan pendekatan **Deep Learning**, model akan berupaya untuk mengelompokkan gambar ke dalam kategori-kategori yang sesuai tanpa menggunakan label langsung pada saat pelatihan.

Melalui penerapan model **Deep Learning** ini, diharapkan dapat meningkatkan akurasi dalam mengklasifikasikan jenis pakaian berdasarkan gambar, dan memberikan pemahaman yang lebih mendalam tentang bagaimana model deep learning dapat digunakan dalam aplikasi pengenalan gambar.

Menggunakan Library Tensorflow

Load Data

- Tahap pertama adalah import library yang dibutuhkan
- Load dataset menggunakan fungsi `fashion_mnist.load_data()` dari TensorFlow.

```
In [1]: import tensorflow as tf ##jika error, install tensorflow terlebih dahulu
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import logging
logging.getLogger('matplotlib.font_manager').disabled = True
import random

##di tensorflow menggunakan seed value bukan random state
seed_value = 42 #diisi dengan 2 digit npm terakhir praktikan
tf.random.set_seed(seed_value) # Mengatur seed untuk TensorFlow
np.random.seed(seed_value) # Mengatur seed untuk NumPy
random.seed(seed_value) # Mengatur seed untuk Python random
```

```
In [2]: fashion_mnist = tf.keras.datasets.fashion_mnist ## Cara untuk mengakses dataset Fashion MNIST yang sudah
        ## Tersedia secara Langsung di dalam TensorFlow.

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data() ## Memisahkan data pelatihan dan data pengujian
```

```
In [3]: ## Mendefinisikan nama-nama kelas untuk dataset Fashion MNIST.
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Eksplor Data

```
In [4]: ## Melihat jumlah data train

train_images.shape
## ada 60.000 gambar pada training set, dan setiap gambar memiliki ukuran 28x28 piksel
```

```
Out[4]: (60000, 28, 28)
```



Watermarkly

```
In [5]: train_labels ## Label dari gambar  
Out[5]: array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

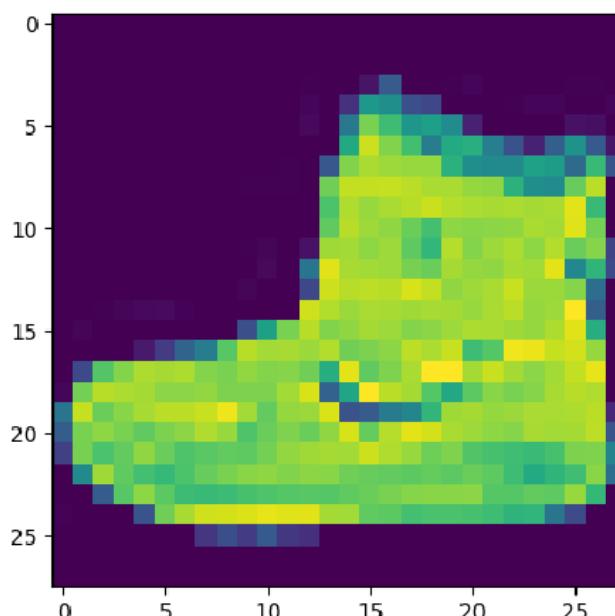
```
In [6]: test_images.shape  
## ada 10.000 gambar pada testing set, dan setiap gambar memiliki ukuran 28x28 piksel  
  
Out[6]: (10000, 28, 28)
```

```
In [7]: len(test_labels)  
## ada 10.000 gambar pada testing set  
  
Out[7]: 10000
```

Preprocessing Data

- Menampilkan gambar pertama dari dataset pelatihan Fashion MNIST agar kita dapat memvisualisasikan seperti apa data yang akan digunakan untuk melatih model

```
In [8]: plt.figure() ## Membuat sebuah figure baru untuk plot  
plt.imshow(train_images[0]) ## Menampilkan gambar pertama dari dataset pelatihan  
plt.colorbar() ## Menambahkan colorbar di samping gambar  
plt.grid(False) ## Mematikan grid pada plot  
plt.show() ## Menampilkan plot gambar
```



```
In [9]: # Reshape data jika model memerlukan data dalam bentuk matriks 2D  
train_images = train_images.reshape(-1, 28, 28, 1).astype('float32') / 255  
test_images = test_images.reshape(-1, 28, 28, 1).astype('float32') / 255
```

- Tampilkan 25 gambar pertama dari set pelatihan dan tampilkan nama kelas di bawah setiap gambar. Untuk memverifikasi bahwa data berada dalam format yang benar

```
In [10]: plt.figure(figsize=(10,10)) ## Mengatur ukuran figure menjadi 10x10 inch untuk menampilkan 25 gambar  
for i in range(25): ## Loop untuk menampilkan 25 gambar pertama dari dataset pelatihan  
    plt.subplot(5,5,i+1) ## Menempatkan gambar ke dalam subplot pada posisi baris ke-i dan kolom ke-i  
    plt.xticks([]) ## Menghapus tanda di sumbu x  
    plt.yticks([]) ## Menghapus tanda di sumbu y  
    plt.grid(False)  
    plt.imshow(train_images[i], cmap=plt.cm.binary) ## Menampilkan gambar ke-i dalam format biner (hitam-putih)  
    plt.xlabel(class_names[train_labels[i]]) ## Menambahkan Label kelas di bawah setiap gambar  
plt.show() ## Menampilkan semua gambar yang telah di-plot
```



Membangun Model

Menyiapkan Lapisan-lapisan

Watermarkly

Blok dasar dari sebuah jaringan saraf adalah lapisan. Lapisan-lapisan ini mengekstraksi representasi dari data yang diberikan kepadanya. Diharapkan, representasi tersebut bermakna untuk masalah yang sedang dihadapi.

Sebagian besar pembelajaran mendalam terdiri dari menghubungkan lapisan-lapisan sederhana. Sebagian besar lapisan, seperti `tf.keras.layers.Dense`, memiliki parameter-parameter yang dipelajari selama pelatihan.

Model ini terdiri dari

- Input Layer
- Flatten Layer
- Dense Layer 1
- Dense Layer 2
- Output Layer

```
In [11]: model_tf = tf.keras.Sequential([
    ## Membuat model Sequential untuk menyusun Lapisan-Lapisan jaringan saraf
    tf.keras.layers.Input(shape=(28, 28)), ## Menambahkan Lapisan input dengan bentuk (28, 28) untuk gambar 28x28 piksel
    tf.keras.layers.Flatten(), ## Mengubah gambar 2D (28x28) menjadi vektor 1D (784) untuk diproses oleh Lapisan Dense
    tf.keras.layers.Dense(128, activation='relu'), ## Lapisan Dense pertama dengan 128 unit dan fungsi aktivasi ReLU
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(300, activation='relu'), ## Lapisan Dense kedua dengan 300 unit dan fungsi aktivasi ReLU
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(10) ## Lapisan Dense output dengan 10 unit, satu untuk setiap kelas
])
```

Mengompilasi Model

Sebelum model siap dilatih, ada beberapa pengaturan yang perlu dilakukan pada langkah kompilasi:

1. Optimizer (Pengoptimal) — Menentukan bagaimana model diperbarui berdasarkan data dan fungsi kerugiannya.
2. Loss Function (Fungsi Kerugian) — Mengukur akurasi model selama pelatihan. Tujuannya adalah untuk meminimalkan fungsi ini agar model semakin akurat.
3. Metrics (Metode Pengukuran) — Digunakan untuk memonitor proses pelatihan dan pengujian. Contoh yang umum digunakan adalah akurasi, yaitu fraksi gambar yang terklasifikasi dengan benar.

```
In [12]: model_tf.compile(optimizer='adam', ## Menggunakan optimizer Adam untuk memperbarui bobot model selama pelatihan
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), ## Menggunakan fungsi kerugian Sparse Categorical Crossentropy
                      metrics=['accuracy']) ## Memilih akurasi sebagai metrik untuk memonitor kinerja model
```

Melatih Model

Dengan memanggil `model.fit()`, kita memberi tahu model untuk mulai belajar dari data yang telah disiapkan.

```
In [13]: from keras.callbacks import EarlyStopping  
  
# Konfigurasi EarlyStopping  
early_stopping = EarlyStopping(  
    monitor='val_accuracy', # Memantau akurasi validasi untuk menentukan perbaikan model  
    patience=5, # Hentikan pelatihan jika tidak ada perbaikan setelah 3 epoch  
    min_delta=0.0001, # Ambang batas perubahan minimum pada akurasi validasi agar dianggap sebagai perbaikan  
    restore_best_weights=True # Kembalikan ke bobot terbaik saat pelatihan dihentikan  
)  
  
# Melatih model dengan EarlyStopping  
history = model_tf.fit(  
    train_images, train_labels,  
    epochs=40,  
    validation_data=(test_images, test_labels),  
    callbacks=[early_stopping])
```

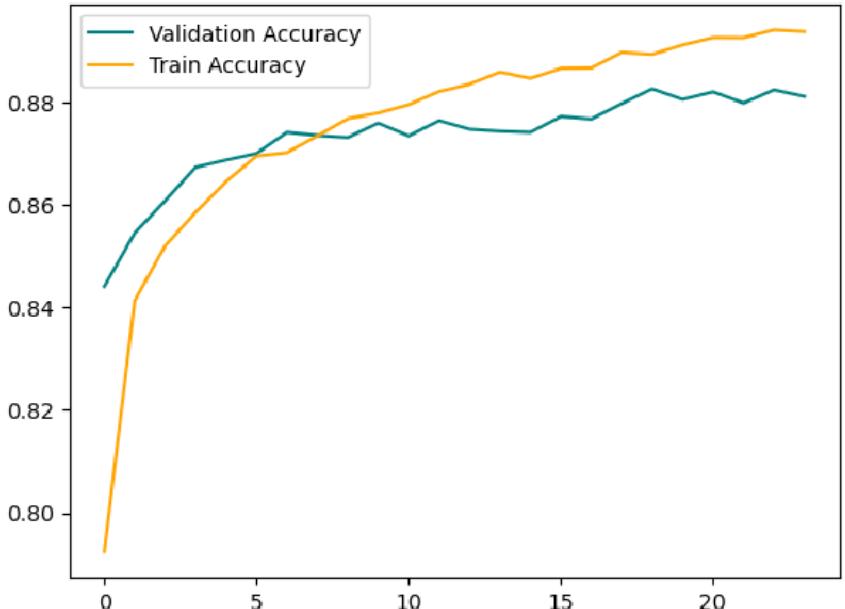
```
Epoch 1/40  
1875/1875 6s 2ms/step - accuracy: 0.7315 - loss: 0.7443 - val_accuracy: 0.8441 - val_loss: 0.4242  
Epoch 2/40  
1875/1875 4s 2ms/step - accuracy: 0.8383 - loss: 0.4489 - val_accuracy: 0.8545 - val_loss: 0.3966  
Epoch 3/40  
1875/1875 3s 2ms/step - accuracy: 0.8509 - loss: 0.4068 - val_accuracy: 0.8608 - val_loss: 0.3802  
Epoch 4/40  
1875/1875 3s 1ms/step - accuracy: 0.8574 - loss: 0.3907 - val_accuracy: 0.8673 - val_loss: 0.3687  
Epoch 5/40  
1875/1875 3s 1ms/step - accuracy: 0.8635 - loss: 0.3724 - val_accuracy: 0.8687 - val_loss: 0.3611  
Epoch 6/40  
1875/1875 3s 1ms/step - accuracy: 0.8684 - loss: 0.3578 - val_accuracy: 0.8700 - val_loss: 0.3573  
Epoch 7/40  
1875/1875 3s 1ms/step - accuracy: 0.8705 - loss: 0.3544 - val_accuracy: 0.8741 - val_loss: 0.3484  
Epoch 8/40  
1875/1875 3s 2ms/step - accuracy: 0.8727 - loss: 0.3469 - val_accuracy: 0.8735 - val_loss: 0.3490  
Epoch 9/40  
1875/1875 3s 2ms/step - accuracy: 0.8775 - loss: 0.3367 - val_accuracy: 0.8731 - val_loss: 0.3434  
Epoch 10/40  
1875/1875 3s 2ms/step - accuracy: 0.8772 - loss: 0.3328 - val_accuracy: 0.8759 - val_loss: 0.3513  
Epoch 11/40  
1875/1875 3s 1ms/step - accuracy: 0.8783 - loss: 0.3266 - val_accuracy: 0.8734 - val_loss: 0.3540  
Epoch 12/40  
1875/1875 3s 1ms/step - accuracy: 0.8807 - loss: 0.3225 - val_accuracy: 0.8764 - val_loss: 0.3429  
Epoch 13/40  
1875/1875 3s 1ms/step - accuracy: 0.8820 - loss: 0.3175 - val_accuracy: 0.8747 - val_loss: 0.3422  
Epoch 14/40  
1875/1875 3s 1ms/step - accuracy: 0.8846 - loss: 0.3113 - val_accuracy: 0.8743 - val_loss: 0.3441  
Epoch 15/40  
1875/1875 3s 1ms/step - accuracy: 0.8845 - loss: 0.3092 - val_accuracy: 0.8741 - val_loss: 0.3470  
Epoch 16/40  
1875/1875 3s 1ms/step - accuracy: 0.8857 - loss: 0.3073 - val_accuracy: 0.8772 - val_loss: 0.3474  
Epoch 17/40  
1875/1875 3s 1ms/step - accuracy: 0.8862 - loss: 0.3060 - val_accuracy: 0.8767 - val_loss: 0.3440  
Epoch 18/40  
1875/1875 3s 1ms/step - accuracy: 0.8897 - loss: 0.3002 - val_accuracy: 0.8797 - val_loss: 0.3402  
Epoch 19/40  
1875/1875 3s 1ms/step - accuracy: 0.8894 - loss: 0.2961 - val_accuracy: 0.8826 - val_loss: 0.3275  
Epoch 20/40  
1875/1875 3s 1ms/step - accuracy: 0.8910 - loss: 0.2988 - val_accuracy: 0.8805 - val_loss: 0.3450  
Epoch 21/40  
1875/1875 3s 1ms/step - accuracy: 0.8918 - loss: 0.2929 - val_accuracy: 0.8820 - val_loss: 0.3349  
Epoch 22/40  
1875/1875 3s 2ms/step - accuracy: 0.8924 - loss: 0.2913 - val_accuracy: 0.8798 - val_loss: 0.3440  
Epoch 23/40  
1875/1875 3s 1ms/step - accuracy: 0.8928 - loss: 0.2870 - val_accuracy: 0.8824 - val_loss: 0.3347  
Epoch 24/40  
1875/1875 3s 2ms/step - accuracy: 0.8923 - loss: 0.2880 - val_accuracy: 0.8811 - val_loss: 0.3361
```

Visualisasi (Plotting) Loss Curve untuk Training dan Validation

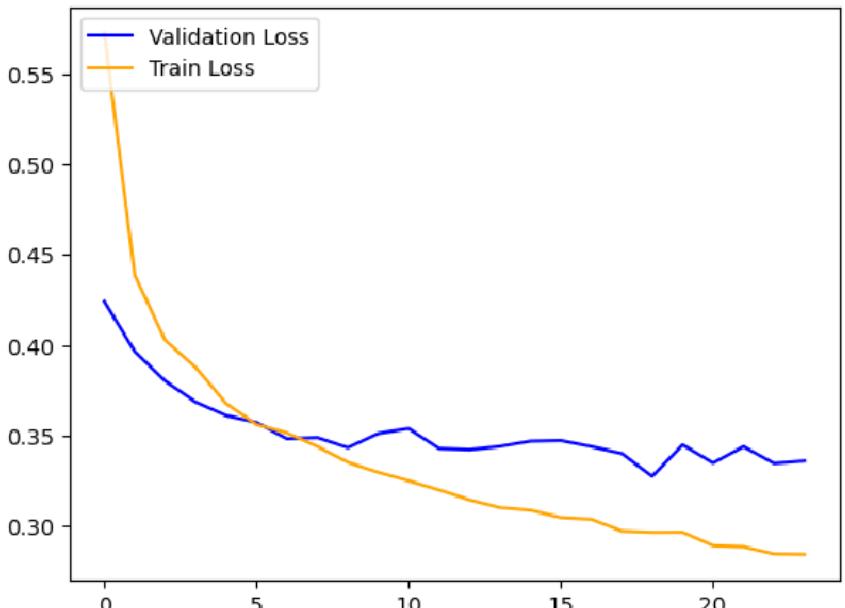
- digunakan untuk memantau performa model secara visual. Melalui plot ini, kita bisa melihat apakah model mengalami overfitting (ketika loss pada data pelatihan menurun tetapi loss pada data validasi meningkat) atau underfitting (ketika loss pada kedua data pelatihan dan validasi tinggi).



```
In [14]: fig = plt.figure()
plt.plot(history.history['val_accuracy'], color='teal', label='Validation Accuracy') # Plot akurasi validasi
plt.plot(history.history['accuracy'], color='orange', label='Train Accuracy') # Plot akurasi pelatihan
plt.legend(loc="upper left")
plt.show()
```



```
In [15]: fig = plt.figure()
plt.plot(history.history['val_loss'], color = 'blue', label = 'Validation Loss')
plt.plot(history.history['loss'], color = 'orange', label = 'Train Loss')
plt.legend(loc = "upper left")
plt.show()
```



Evaluasi Akurasi

- mengevaluasi kinerja model setelah pelatihan

```
In [16]: test_loss, test_acc_tf = model_tf.evaluate(test_images, test_labels, verbose=2) ## Mengevaluasi model TensorFlow pada data pengujian dan mengembalikan nilai loss dan akurasi

print('\nTest accuracy:', test_acc_tf) ## Menampilkan akurasi model pada data pengujian
```

313/313 - 0s - 678us/step - accuracy: 0.8826 - loss: 0.3275

Test accuracy: 0.8826000094413757

Membuat prediksi

```
In [17]: probability_model = tf.keras.Sequential([model_tf,
                                                tf.keras.layers.Softmax()])
## Membuat model baru yang menambahkan Layer Softmax di akhir model yang sudah dilatih (model_tf) untuk mengubah output Logits menjadi probabilitas
```

```
In [18]: predictions_tf = probability_model.predict(test_images)
## Membuat prediksi probabilitas untuk data pengujian (test_images) menggunakan model
```

313/313 ————— 0s 873us/step

```
In [19]: predictions_tf[0]
## Mengambil prediksi probabilitas untuk gambar pertama dari data pengujian.
```

```
Out[19]: array([2.0683486e-07, 2.3441070e-08, 1.1679006e-07, 1.4710794e-08,
   1.5451231e-07, 3.2621378e-03, 3.0379772e-07, 9.7031649e-03,
   3.5984647e-07, 9.8703361e-01], dtype=float32)
```

```
In [20]: np.argmax(predictions_tf[0])
## Mengambil indeks kelas dengan probabilitas tertinggi dari prediksi gambar pertama.
```

```
Out[20]: 9
```

```
In [21]: test_labels[0]
## Menampilkan Label asli atau kelas sebenarnya dari gambar pertama dalam dataset pengujian.
```

```
Out[21]: 9
```

Fungsi untuk menggrafikkan seluruh set prediksi untuk 10 kelas.



Watermarkly

```
In [22]: def plot_image(i, predictions_array, true_label, img): ## Fungsi plot_image menampilkan gambar dengan
    ## Label prediksi dan label asli, serta memberi warna berdasarkan kecocokan.
    true_label, img = true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                           100*np.max(predictions_array),
                                           class_names[true_label]),
               color=color)

def plot_value_array(i, predictions_array, true_label): ## Fungsi plot_value_array menampilkan grafik
    ## batang untuk probabilitas prediksi setiap kelas dan menandai warna merah untuk prediksi dan
    ## biru untuk label yang benar.

    true_label = true_label[i]
    plt.grid(False)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

Verifikasi Prediksi

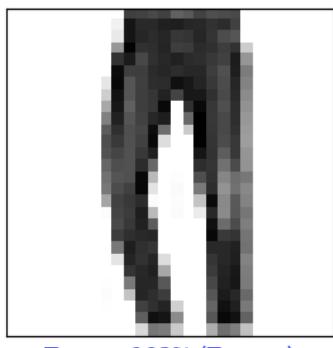
- prediksi yang benar dilabeli dengan biru
- prediksi yang salah dilabeli dengan merah



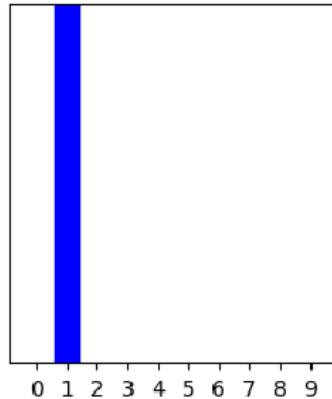
Watermarkly

In [23]:

```
i = 5 ## prediksi gambar ke 5
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions_tf[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions_tf[i], test_labels)
plt.show()
```



Trouser 100% (Trouser)



y

No Copy

No

- menampilkan 20 gambar pertama dari dataset pengujian (dalam grid 5x4), beserta prediksi model dan probabilitas untuk setiap gambar, serta memberikan warna border hijau untuk prediksi yang benar dan merah untuk prediksi yang salah.

No Copy

No Copy

No Copy

py No Copy

No Copy

No

No Copy

No Copy

Watermarkly

No Copy

No Copy

```
In [24]: import numpy as np
import matplotlib.pyplot as plt

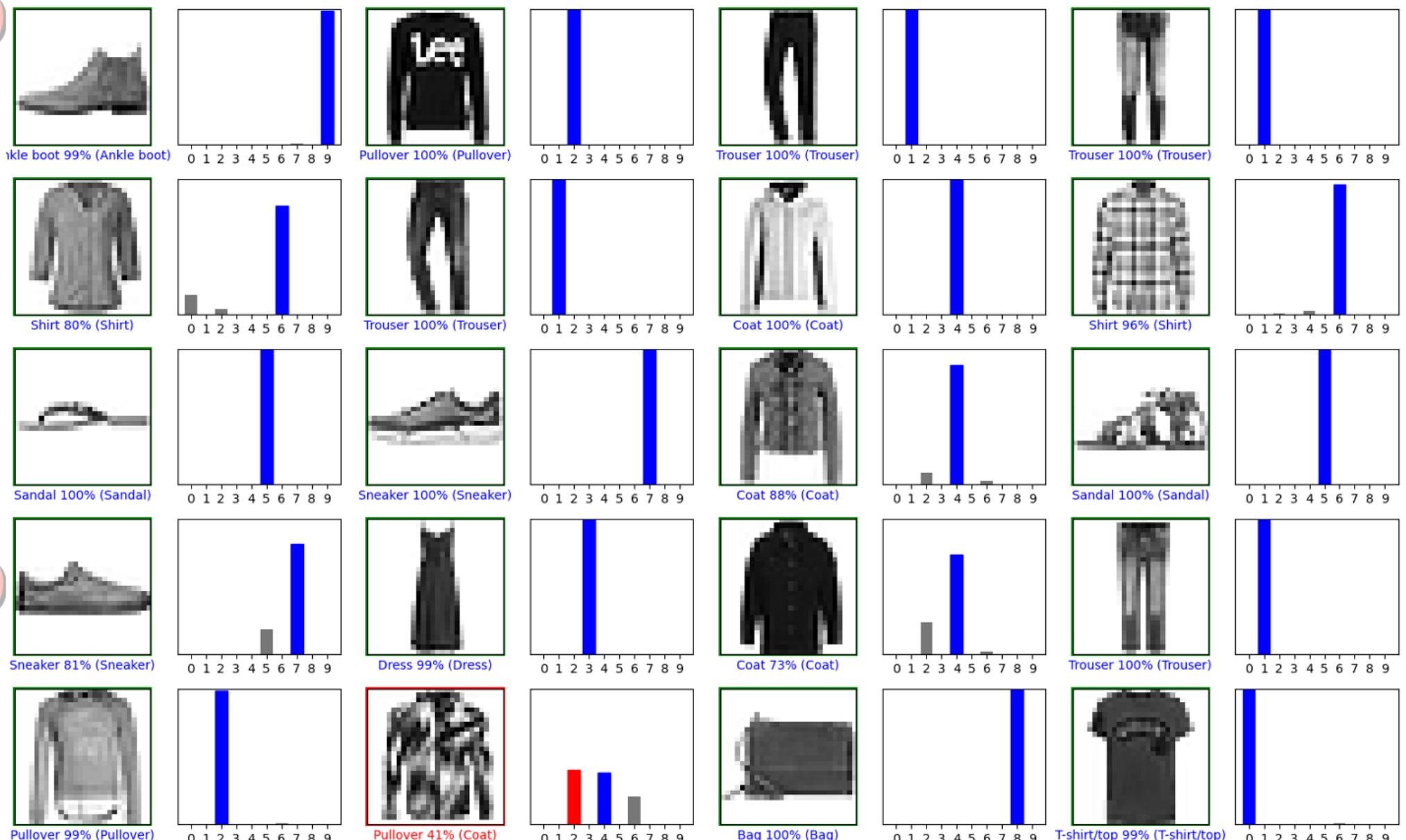
num_rows = 5
num_cols = 4
num_images = num_rows * num_cols

correct_predictions = [np.argmax(predictions_tf[i]) == test_labels[i] for i in range(num_images)]
average_accuracy = np.mean(correct_predictions) * 100

plt.figure(figsize=(2 * 2 * num_cols, 2 * num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
    plot_image(i, predictions_tf[i], test_labels, test_images)
    border_color = 'green' if correct_predictions[i] else 'red'
    plt.gca().patch.set_edgecolor(border_color)
    plt.gca().patch.set_linewidth(3)

    plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
    plot_value_array(i, predictions_tf[i], test_labels)

plt.suptitle(f'Akurasi Rata-rata: {average_accuracy:.2f}%')
plt.tight_layout()
plt.show()
```



- Gunakan model yang telah dilatih untuk membuat prediksi tentang sebuah gambar tunggal

```
In [25]: img = test_images[1] ## mencetak dimensi gambar kedua dari dataset pengujian
print(img.shape)

(28, 28, 1)
```

In [26]:

```
img = (np.expand_dims(img,0)) ## menambahkan dimensi batch pada gambar agar sesuai  
## dengan format input yang diharapkan oleh model,  
  
print(img.shape)  
  
(1, 28, 28, 1)
```

In [27]:

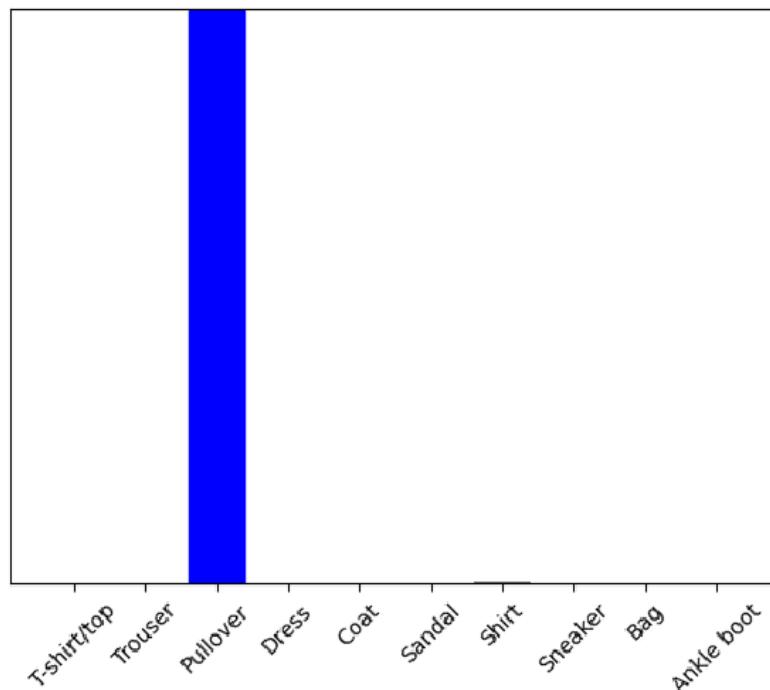
```
predictions_single = probability_model.predict(img) ## memprediksi probabilitas kelas dari gambar tunggal  
  
print(predictions_single) ## Mencetak hasil prediksi, yang berupa array probabilitas untuk setiap kelas
```

1/1 _____ 0s 16ms/step
[[1.9931425e-04 2.4238521e-13 9.9681997e-01 2.2356735e-08 4.6172753e-04
4.1765580e-19 2.5190182e-03 5.6723979e-21 1.5659167e-11 3.7146804e-18]]

- Menampilkan grafik batang yang menunjukkan probabilitas prediksi model untuk masing-masing kelas pada gambar yang diprediksi

In [28]:

```
plot_value_array(1, predictions_single[0], test_labels) ##Menampilkan grafik batang dengan probabilitas prediksi untuk gambar ke-1.  
_ = plt.xticks(range(10), class_names, rotation=45) ##Menambahkan Label kelas pada sumbu X dan memutar label tersebut sebesar 45 derajat agar lebih mudah dibaca.  
plt.show() ##Menampilkan grafik ke layar.
```



In [29]:

```
np.argmax(predictions_single[0]) ##menunjukkan kelas yang paling mungkin sesuai dengan gambar yang diprediksi
```

Out[29]:

```
2
```

- Membuat DataFrame dengan label asli, prediksi, dan probabilitas untuk setiap gambar, serta menggabungkan data piksel.
- Menampilkan 16 gambar sampel beserta prediksi, probabilitas, dan label asli.
- Menghitung dan menampilkan akurasi rata-rata, dengan warna border hijau untuk prediksi benar dan merah untuk prediksi salah.



Watermarkly

In [30]:

```
df_test = pd.DataFrame({  
    'Target': test_labels, # Label asli  
    'Predicted': np.argmax(predictions_tf, axis=1), # Label prediksi  
    'Proba_0': predictions_tf[:, 0], # Probabilitas kelas 0 (misal 'Other')  
    'Proba_1': predictions_tf[:, 1] # Probabilitas kelas 1 (misal 'T-shirt/top')  
})  
  
# Mengubah gambar menjadi array dua dimensi (banyak gambar x 784 piksel)  
pixels = test_images.reshape(len(test_images), -1)  
  
# Membuat DataFrame untuk kolom piksel  
df_pixels = pd.DataFrame(pixels, columns=[f'pixel{i+1}' for i in range(28*28)])  
  
# Menggabungkan kedua DataFrame dengan `pd.concat`  
df_test = pd.concat([df_test, df_pixels], axis=1)
```



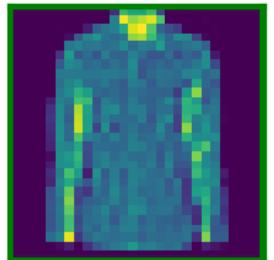
Watermarkly

```
In [31]: df_plot = df_test.sample(16, random_state=42).copy() # sesuaikan 2 digit npm terakhir  
# atau 1 digit npm terakhir  
fig, ax = plt.subplots(4, 4, figsize=(10, 10))  
  
# Hitung akurasi rata-rata  
rata2_accuracy = (df_test['Target'] == df_test['Predicted']).mean() * 100  
  
for i, axi in enumerate(ax.flat):  
    p0 = df_plot['Proba_0'].values[i] # probabilitas kelas 'Other'  
    p1 = df_plot['Proba_1'].values[i] # probabilitas kelas 'T-shirt/top'  
  
    # Prediksi dan probabilitas tertinggi  
    predicted_class = df_plot['Predicted'].values[i]  
    prediction_probability = max(p0, p1) * 100  
  
    # Label asli  
    true_class = df_plot['Target'].values[i]  
  
    # Ekstraksi data piksel dan tampilkan gambar  
    pixels = df_plot[[f'pixel{j}' for j in range(1, 28*28 + 1)]].values[i].reshape(28, 28)  
    axi.imshow(pixels, cmap='viridis')  
  
    # Tampilkan label, prediksi, dan probabilitas pada gambar  
    axi.set(xticks=[], yticks=[],  
            xlabel=f"True: {true_class}\nPred: {predicted_class}\nProb: {prediction_probability:.2f}%)")  
  
    # Warna bingkai hijau jika benar, merah jika salah  
    edge_color = 'green' if predicted_class == true_class else 'red'  
    axi.spines[:].set_edgecolor(edge_color)  
    axi.spines[:].set_linewidth(3)  
  
# Judul dengan akurasi rata-rata  
fig.suptitle(f'Contoh gambar yang diklasifikasikan dari set pengujian\n' f'Batas hijau: klasifikasi benar\nAkurasi Keseluruhan: {rata2_accuracy:.2f}%)')  
fig.tight_layout()  
plt.show()
```

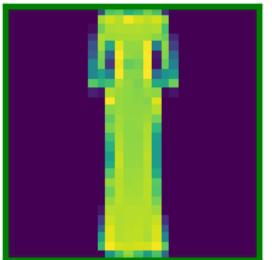


Watermarkly

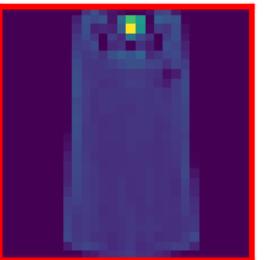
Contoh gambar yang diklasifikasikan dari set pengujian
Batas hijau: klasifikasi benar
Akurasi Keseluruhan: 88.26%



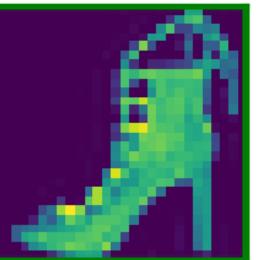
True: 6
Pred: 6
Prob: 0.02%



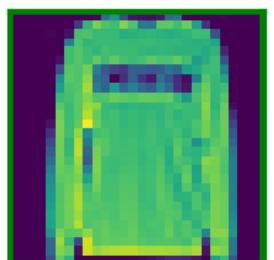
True: 3
Pred: 3
Prob: 6.43%



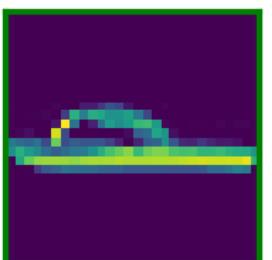
True: 0
Pred: 6
Prob: 37.13%



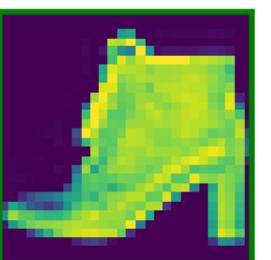
True: 5
Pred: 5
Prob: 0.00%



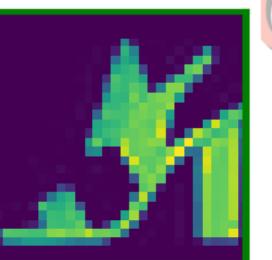
True: 2
Pred: 2
Prob: 2.54%



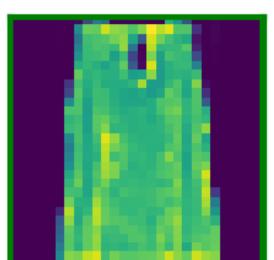
True: 5
Pred: 5
Prob: 0.00%



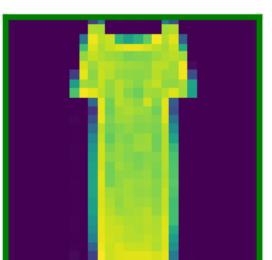
True: 9
Pred: 9
Prob: 0.00%



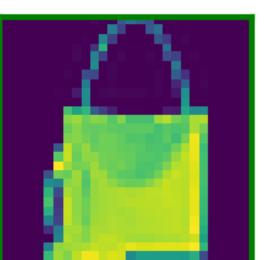
True: 5
Pred: 5
Prob: 0.00%



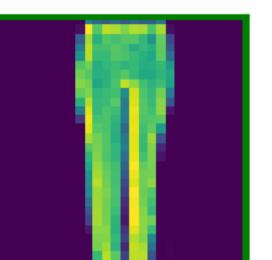
True: 6
Pred: 6
Prob: 25.37%



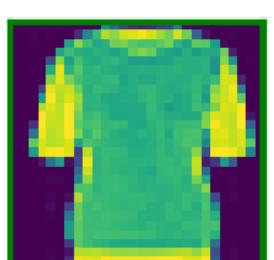
True: 3
Pred: 3
Prob: 3.18%



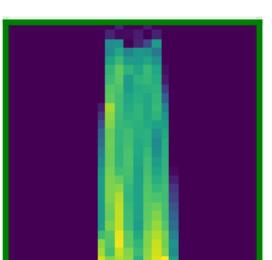
True: 8
Pred: 8
Prob: 0.00%



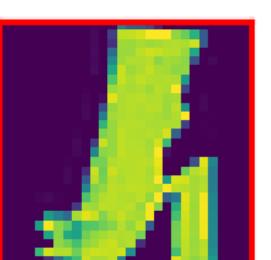
True: 1
Pred: 1
Prob: 100.00%



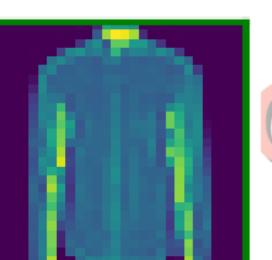
True: 0
Pred: 0
Prob: 89.51%



True: 3
Pred: 3
Prob: 0.44%



True: 9
Pred: 5
Prob: 0.42%



True: 6
Pred: 6
Prob: 0.00%

Menggunakan Library Scikit-Learn

Watermarkly

Load Data

- Import library yang dibutuhkan
- Mengambil dataset Fashion-MNIST dari OpenML

```
In [32]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier # Untuk model Multi-Layer Perceptron (MLP) Classifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import tensorflow as tf
```

```
In [33]: # Menonaktifkan peringatan
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter("ignore")
```

```
In [34]: # Mengambil dataset Fashion-MNIST dari OpenML
from sklearn.datasets import fetch_openml
fashion_mnist = fetch_openml('Fashion-MNIST', version=1)

# Membagi dataset Fashion MNIST menjadi set pelatihan dan pengujian
train_images = fashion_mnist.data[:60000] # train_images berisi 60.000 gambar pertama untuk pelatihan
train_labels = fashion_mnist.target[:60000].astype(int) # train_labels berisi 60.000 Label pertama untuk pelatihan (diubah menjadi integer)
test_images = fashion_mnist.data[60000:] # test_images berisi 10.000 gambar terakhir untuk pengujian
test_labels = fashion_mnist.target[60000:].astype(int) # test_labels berisi 10.000 Label terakhir untuk pengujian (diubah menjadi integer)
```

```
In [35]: # Normalisasi pixel gambar
train_images = train_images / 255.0
test_images = test_images / 255.0
```



Watermarkly

In [36]:

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, log_loss
from sklearn.model_selection import train_test_split
import numpy as np

# Split train_images menjadi train dan validation set
train_images_split, val_images, train_labels_split, val_labels = train_test_split(
    train_images, train_labels, test_size=0.2, random_state=42 # sesuaikan 2 digit npm terakhir
                                            # atau 1 digit npm terakhir
)

# Mendefinisikan model MLP dengan satu hidden Layer berukuran 128 neuron
model_sklearn = MLPClassifier(hidden_layer_sizes=(128, 300),
                               activation='relu', # Fungsi aktivasi ReLU untuk lapisan tersembunyi
                               alpha=0.0001, # Regularisasi L2 (penalty) dengan nilai 0.0001 untuk mencegah overfitting
                               learning_rate_init=0.001, # Kecepatan pembelajaran awal (learning rate) sebesar 0.001
                               early_stopping=True, # Menghentikan pelatihan jika tidak ada peningkatan pada validasi
                               validation_fraction=0.2, # Menyisihkan 20% data Latih untuk validasi saat early stopping
                               n_iter_no_change=10, # Menghentikan pelatihan jika tidak ada perubahan pada validasi selama 10 iterasi berturut-turut
                               random_state=42, # sesuaikan 2 digit npm terakhir
                                            # atau 1 digit npm terakhir
                               verbose=True) # Menampilkan informasi detail selama proses pelatihan

# Melatih model dan mendapatkan loss curves
model_sklearn.fit(train_images_split, train_labels_split)
training_loss = model_sklearn.loss_curve_
```

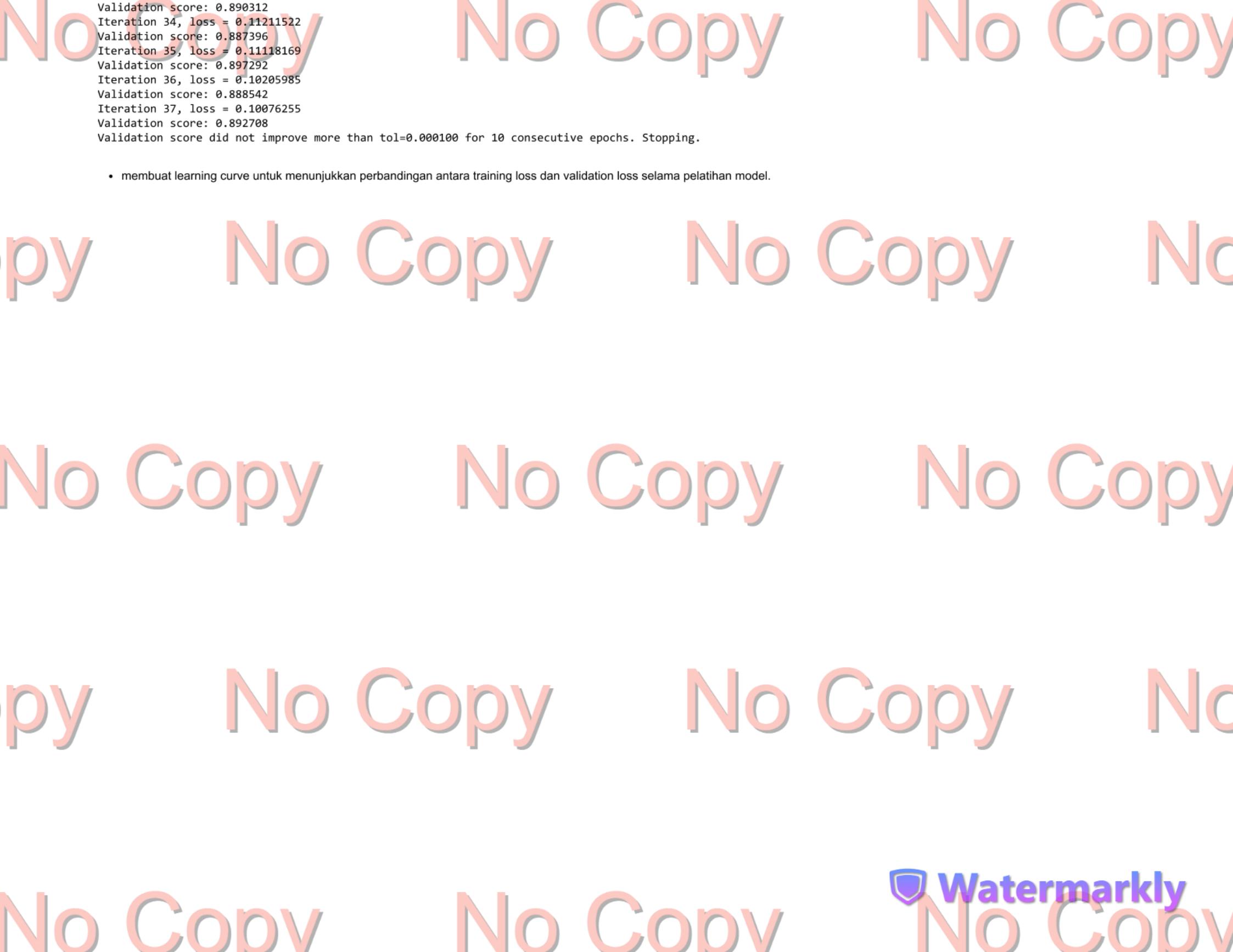


Watermarkly

Iteration 1, loss = 0.63552317
Validation score: 0.827292
Iteration 2, loss = 0.42263619
Validation score: 0.847500
Iteration 3, loss = 0.37126728
Validation score: 0.857604
Iteration 4, loss = 0.34177076
Validation score: 0.873958
Iteration 5, loss = 0.32000136
Validation score: 0.878750
Iteration 6, loss = 0.30229847
Validation score: 0.878125
Iteration 7, loss = 0.28603573
Validation score: 0.884687
Iteration 8, loss = 0.27602505
Validation score: 0.890521
Iteration 9, loss = 0.26272598
Validation score: 0.882396
Iteration 10, loss = 0.25466473
Validation score: 0.890521
Iteration 11, loss = 0.24281892
Validation score: 0.891250
Iteration 12, loss = 0.23238843
Validation score: 0.886771
Iteration 13, loss = 0.22470763
Validation score: 0.888958
Iteration 14, loss = 0.21646515
Validation score: 0.892708
Iteration 15, loss = 0.21138910
Validation score: 0.882812
Iteration 16, loss = 0.20453345
Validation score: 0.893646
Iteration 17, loss = 0.19352105
Validation score: 0.891250
Iteration 18, loss = 0.18908564
Validation score: 0.888646
Iteration 19, loss = 0.18192541
Validation score: 0.895208
Iteration 20, loss = 0.17369261
Validation score: 0.896250
Iteration 21, loss = 0.16764003
Validation score: 0.892813
Iteration 22, loss = 0.16727562
Validation score: 0.896250
Iteration 23, loss = 0.15872926
Validation score: 0.894479
Iteration 24, loss = 0.15161014
Validation score: 0.895104
Iteration 25, loss = 0.14656841
Validation score: 0.881250
Iteration 26, loss = 0.14482523
Validation score: 0.898750
Iteration 27, loss = 0.13445753
Validation score: 0.894896
Iteration 28, loss = 0.13452374
Validation score: 0.894271
Iteration 29, loss = 0.13340194
Validation score: 0.890104
Iteration 30, loss = 0.13184034
Validation score: 0.892396
Iteration 31, loss = 0.12445123
Validation score: 0.893021
Iteration 32, loss = 0.11653952
Validation score: 0.891875
Iteration 33, loss = 0.11663374

Validation score: 0.890312
Iteration 34, loss = 0.11211522
Validation score: 0.887396
Iteration 35, loss = 0.11118169
Validation score: 0.897292
Iteration 36, loss = 0.10205985
Validation score: 0.888542
Iteration 37, loss = 0.10076255
Validation score: 0.892708
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.

- membuat learning curve untuk menunjukkan perbandingan antara training loss dan validation loss selama pelatihan model.

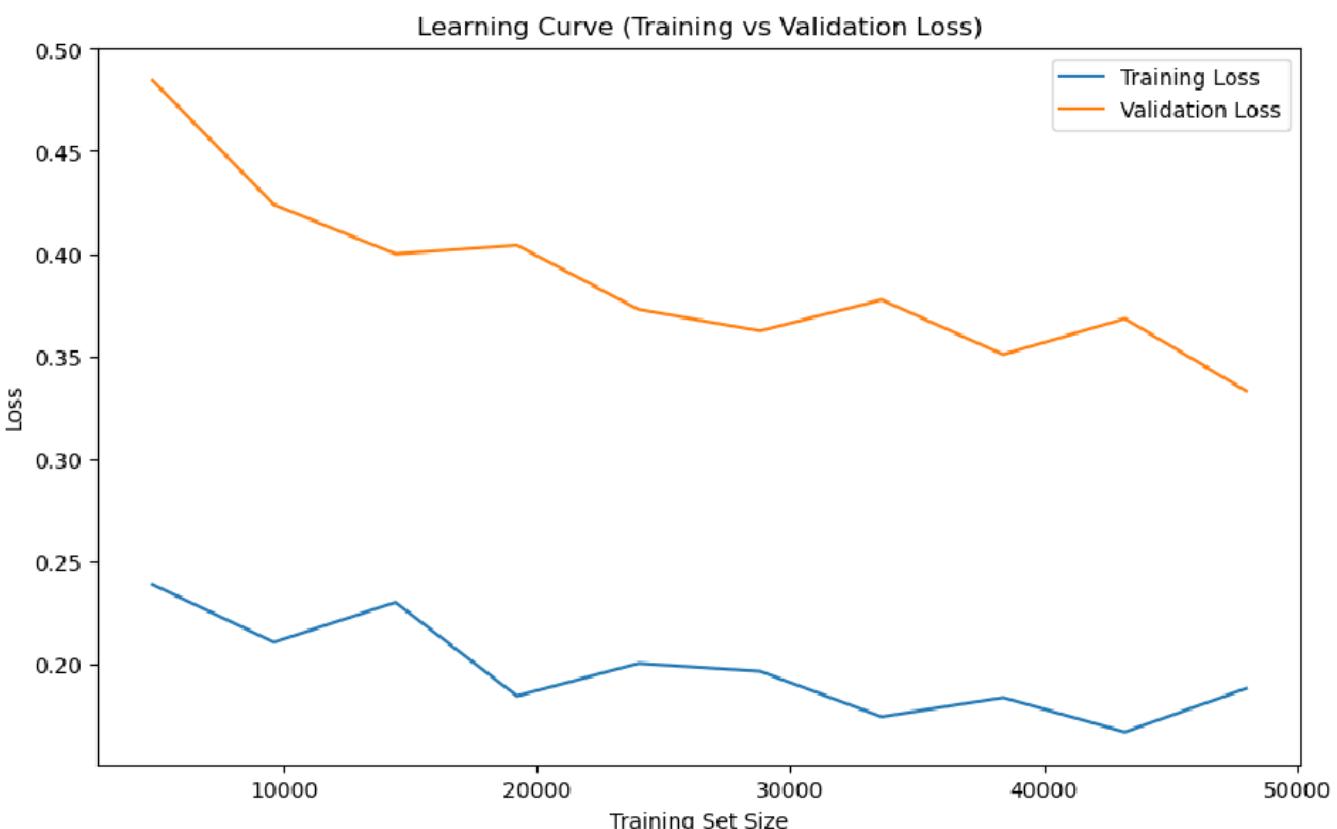


```
In [37]: from sklearn.model_selection import learning_curve

# Menggunakan Learning_curve untuk menghitung skor training dan validation loss pada berbagai ukuran data pelatihan
train_sizes, train_scores, val_scores = learning_curve(
    model_sklearn, train_images, train_labels, train_sizes=np.linspace(0.1, 1.0, 10), cv=5,
    scoring='neg_log_loss', random_state=42, n_jobs=-1) # sesuaikan 2 digit npm terakhir
                                                # atau 1 digit npm terakhir

# Menghitung rata-rata training dan validation Loss di setiap titik
train_loss_mean = -train_scores.mean(axis=1)
val_loss_mean = -val_scores.mean(axis=1)

# Membuat plot Learning curve dengan ukuran data pelatihan di sumbu X dan Loss di sumbu Y
plt.figure(figsize=(10, 6)) # Menentukan ukuran grafik
plt.plot(train_sizes, train_loss_mean, label='Training Loss') # Plot training Loss
plt.plot(train_sizes, val_loss_mean, label='Validation Loss') # Plot validation Loss
plt.xlabel('Training Set Size') # Label untuk sumbu X
plt.ylabel('Loss') # Label untuk sumbu Y
plt.title('Learning Curve (Training vs Validation Loss)')
plt.legend()
plt.show() # Menampilkan grafik
```



- Menghitung akurasi pada data testing



Watermarkly

```
In [38]: # Menggunakan model yang telah dilatih untuk memprediksi label pada data testing  
test_predictions = model_sklearn.predict(test_images)  
# Menghitung akurasi model pada data testing dengan membandingkan prediksi dan label asli  
test_acc_sklearn = accuracy_score(test_labels, test_predictions)  
# Menampilkan nilai akurasi pada data testing  
print('\nTest accuracy:', test_acc_sklearn)
```

Test accuracy: 0.8871

- Menampilkan laporan klasifikasi

```
In [39]: print("\nClassification Report:\n", classification_report(test_labels, test_predictions))  
predictions_proba = model_sklearn.predict_proba(test_images)
```

Classification Report:				
	precision	recall	f1-score	support
0	0.87	0.82	0.84	1000
1	0.99	0.97	0.98	1000
2	0.79	0.81	0.80	1000
3	0.86	0.92	0.89	1000
4	0.80	0.82	0.81	1000
5	0.97	0.96	0.97	1000
6	0.72	0.70	0.71	1000
7	0.93	0.97	0.95	1000
8	0.98	0.96	0.97	1000
9	0.97	0.95	0.96	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

- Mendapatkan probabilitas prediksi

```
In [40]: # Mendapatkan probabilitas prediksi  
predictions_sklearn = model_sklearn.predict_proba(test_images)  
print(predictions_sklearn[0]) # Menampilkan probabilitas prediksi untuk gambar tes pertama  
print("Predicted label for first image:", np.argmax(predictions_sklearn[0])) # Menampilkan Label yang diprediksi untuk gambar  
# tes pertama (kelas dengan probabilitas tertinggi)  
print("Actual label for first image:", test_labels.iloc[0]) # Menampilkan Label sebenarnya untuk gambar tes pertama  
[7.27031047e-11 1.58230648e-09 1.72957327e-08 5.06668988e-08  
1.61048229e-08 8.06031557e-07 1.87509016e-10 1.45934980e-04  
2.80917451e-11 9.99853173e-01]  
Predicted label for first image: 9  
Actual label for first image: 9
```

Fungsi untuk menggrafikkan seluruh set prediksi untuk 10 kelas.

```
In [41]: def plot_image(i, predictions_array, true_label, img):
    true_label, img = true_label.iloc[i], img.iloc[i].values.reshape(28, 28)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    color = 'blue' if predicted_label == true_label else 'red'

    plt.xlabel(f'{class_names[predicted_label]} {100*np.max(predictions_array):.2f}% ({class_names[true_label]})', color=color)

def plot_value_array(i, predictions_array, true_label):
    ## Fungsi plot_value_array menampilkan grafik
    ## batang untuk probabilitas prediksi setiap kelas dan menandai warna merah untuk prediksi dan
    ## biru untuk label yang benar.

    true_label = true_label.iloc[i]
    plt.grid(False)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    # Memberi warna merah pada bar yang menunjukkan prediksi, dan biru pada bar yang menunjukkan Label sebenarnya
    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

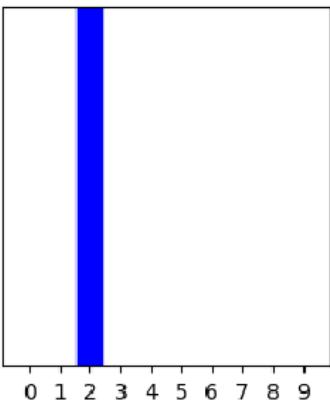
Verifikasi Prediksi

- prediksi yang benar dilabeli dengan biru
- prediksi yang salah dilabeli dengan merah

```
In [42]: i = 1 # Menentukan indeks gambar yang akan ditampilkan (gambar ke-1)
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1) # Membuat subplot pertama (di sebelah kiri) untuk menampilkan gambar beserta Label prediksi dan Label asli
plot_image(i, predictions_sklearn[i], test_labels, test_images)
plt.subplot(1, 2, 2) # Membuat subplot kedua (di sebelah kanan) untuk menampilkan grafik batang probabilitas prediksi setiap kelas
plot_value_array(i, predictions_sklearn[i], test_labels)
plt.show() # Menampilkan semua grafik yang telah dibuat
```



Pullover 99.90% (Pullover)



Watermarkly

No Copy

No Copy

No Copy

```
In [43]: import numpy as np
import matplotlib.pyplot as plt

# Number of rows and columns for the grid
num_rows = 5
num_cols = 4
num_images = num_rows * num_cols

# Calculate whether predictions are correct for the first `num_images` displayed images
correct_predictions = [np.argmax(predictions_sklearn[i]) == test_labels.iloc[i] for i in range(num_images)]
average_accuracy = np.mean(correct_predictions) * 100 # Average accuracy of displayed images

# Set up the figure and subplots
plt.figure(figsize=(2 * 2 * num_cols, 2 * num_rows))
for i in range(num_images):
    # Left subplot for the image with color border
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
    plot_image(i, predictions_sklearn[i], test_labels, test_images)
    border_color = 'green' if correct_predictions[i] else 'red'
    plt.gca().patch.set_edgecolor(border_color)
    plt.gca().patch.set linewidth(3)

    # Right subplot for the prediction probabilities
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
    plot_value_array(i, predictions_sklearn[i], test_labels)

plt.suptitle(f'Akurasi Rata-rata: {average_accuracy:.2f}%')
plt.tight_layout()
plt.show()
```

No Copy

No Copy

No Copy

No Copy

No Copy

No Copy

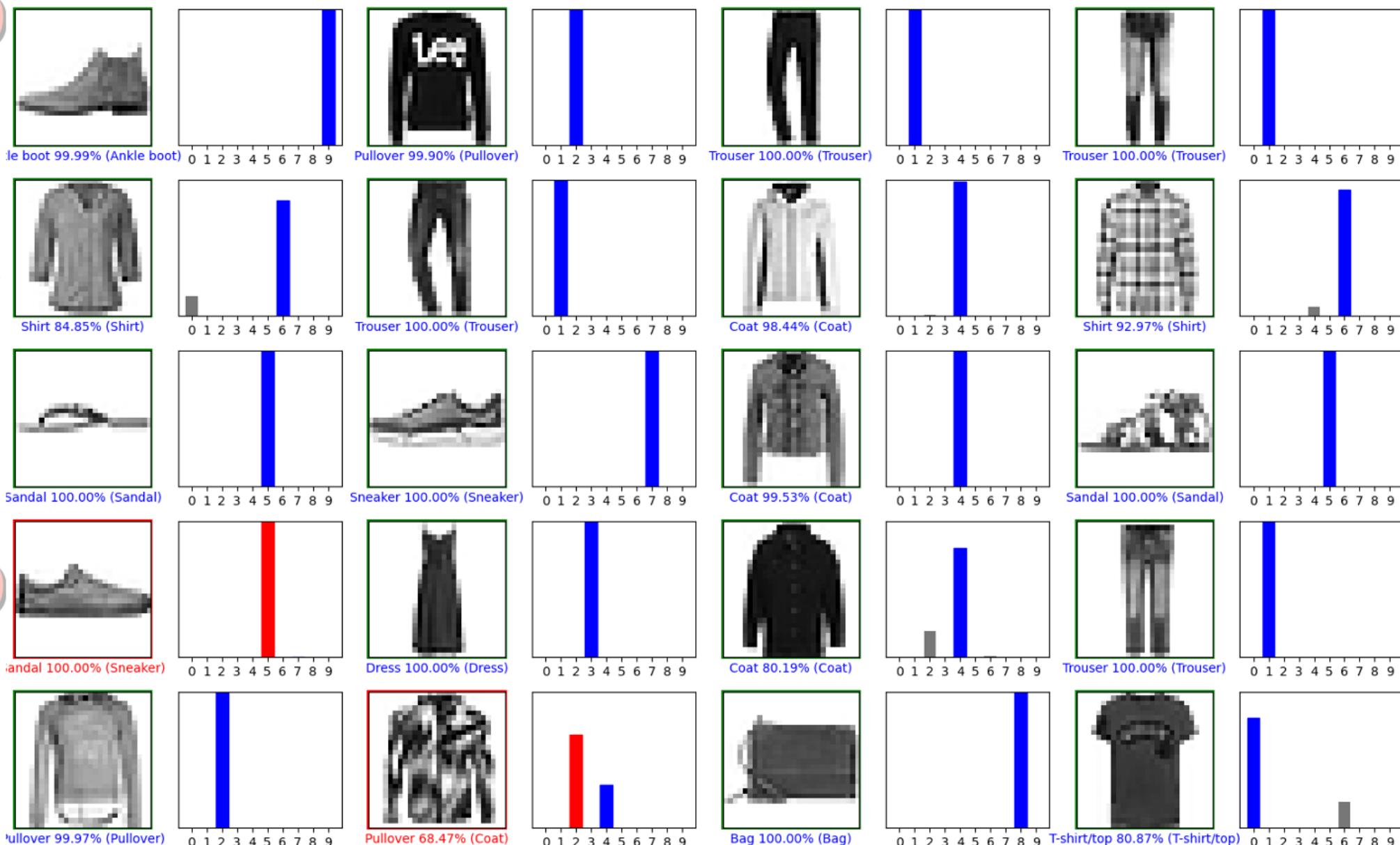
No

No Copy

No Copy

Watermarkly

No Copy



- menampilkan 16 gambar pertama dari dataset pengujian (dalam grid 4x4), beserta prediksi model dan probabilitas untuk setiap gambar, serta memberikan warna border hijau untuk prediksi yang benar dan merah untuk prediksi yang salah.



Watermarkly

No Copy

```
In [44]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Membuat contoh data untuk df_test
n_samples = 10000

df_test = pd.DataFrame({
    'Target': test_labels, # Label asli
    'Predicted': np.argmax(predictions_sklearn, axis=1), # Label prediksi
    'Proba_0': predictions_sklearn[:, 0], # Probabilitas kelas 0 (misal 'Other')
    'Proba_1': predictions_sklearn[:, 1] # Probabilitas kelas 1 (misal 'T-shirt/top')
})

# Normalisasi probabilitas agar Proba_0 + Proba_1 = 1 untuk setiap sampel
df_test['Proba_0'] = df_test['Proba_0'] / (df_test['Proba_0'] + df_test['Proba_1'])
df_test['Proba_1'] = 1 - df_test['Proba_0']

# Membuat data piksel sintetis (gambar 28x28 di-flatten menjadi 784 kolom)
for i in range(1, 28*28 + 1):
    df_test[f'pixel{i}'] = np.random.randint(0, 256, n_samples)

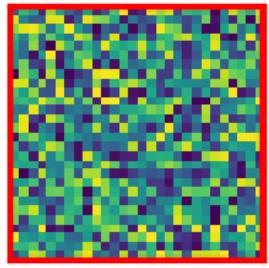
# Kode visualisasi yang diinginkan
df_plot = df_test.sample(16, random_state=42).copy()# sesuaikan 2 digit npm terakhir
                                                # atau 1 digit npm terakhir
fig, ax = plt.subplots(4, 4, figsize=(10, 10))
rata2_accuracy = (df_test['Target'] == df_test['Predicted']).mean() * 100

for i, axi in enumerate(ax.flat):
    p0 = df_plot['Proba_0'].values[i] # Mengambil probabilitas untuk kelas 0 dan 1 dari DataFrame
    p1 = df_plot['Proba_1'].values[i]
    predicted_class = df_plot['Predicted'].values[i] # Mengambil kelas yang diprediksi dan kelas yang benar dari DataFrame
    prediction_probability = max(p0, p1) * 100 # Menghitung probabilitas prediksi tertinggi (dalam persen)
    true_class = df_plot['Target'].values[i]
    pixels = df_plot[[f'pixel{j}' for j in range(1, 28*28 + 1)]].values[i].reshape(28, 28)
    axi.imshow(pixels, cmap='viridis') # Menampilkan gambar dengan colormap 'viridis' pada subplot
    axi.set(xticks=[], yticks=[],
            xlabel=f'Tru: {true_class}\nPred: {predicted_class}\nProb: {prediction_probability:.2f}%')
    edge_color = 'green' if predicted_class == true_class else 'red' # Menentukan warna border berdasarkan apakah prediksi benar (hijau) atau salah (merah)
    # Menetapkan warna dan ketebalan garis pada spines (border) untuk setiap subplot
    for spine in axi.spines.values():
        spine.set_edgecolor(edge_color)
        spine.set_linewidth(3)

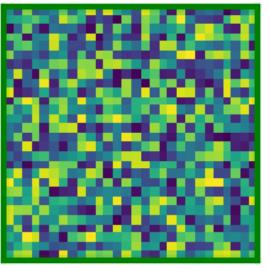
fig.suptitle(f'Contoh gambar yang diklasifikasikan dari set pengujian\n'
             f'Batas hijau: klasifikasi benar\nAkurasi Keseluruhan: {rata2_accuracy:.2f}%')
fig.tight_layout()
plt.show()
```

Watermarkly

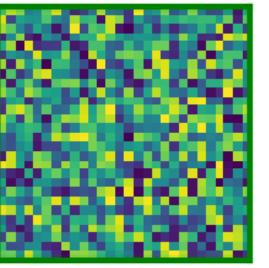
Contoh gambar yang diklasifikasikan dari set pengujian
Batas hijau: klasifikasi benar
Akurasi Keseluruhan: 88.71%



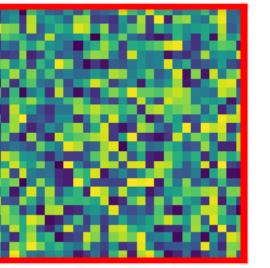
True: 6
Pred: 4
Prob: 99.94%



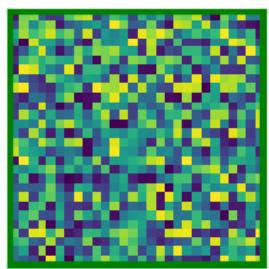
True: 3
Pred: 3
Prob: 100.00%



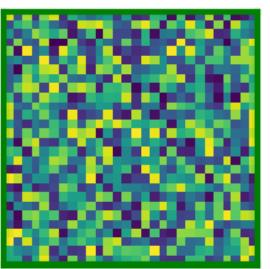
True: 0
Pred: 0
Prob: 99.76%



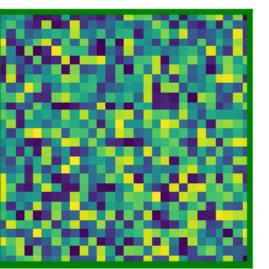
True: 5
Pred: 9
Prob: 100.00%



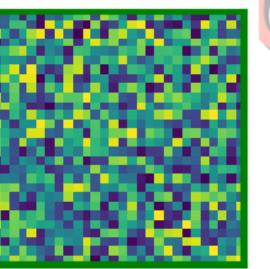
True: 2
Pred: 2
Prob: 99.99%



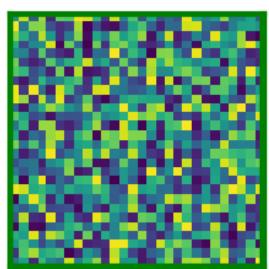
True: 5
Pred: 5
Prob: 100.00%



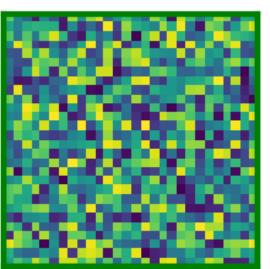
True: 9
Pred: 9
Prob: 94.39%



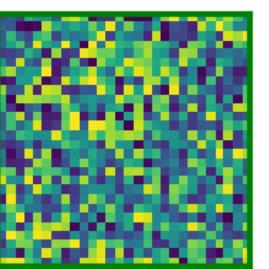
True: 5
Pred: 5
Prob: 100.00%



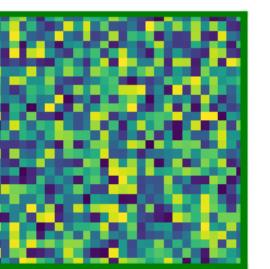
True: 6
Pred: 6
Prob: 99.99%



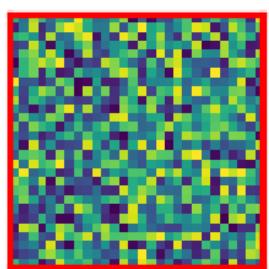
True: 3
Pred: 3
Prob: 99.97%



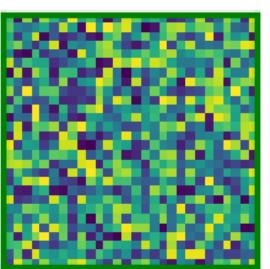
True: 8
Pred: 8
Prob: 99.87%



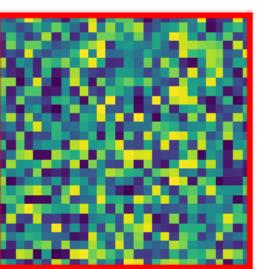
True: 1
Pred: 1
Prob: 100.00%



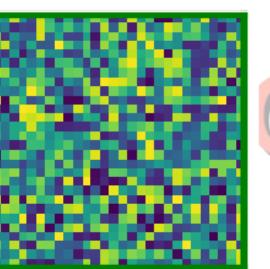
True: 0
Pred: 6
Prob: 100.00%



True: 3
Pred: 3
Prob: 85.31%



True: 9
Pred: 3
Prob: 87.95%



True: 6
Pred: 6
Prob: 100.00%

- menampilkan gambar prediksi beserta label prediksi dan probabilitasnya, serta grafik batang yang menunjukkan probabilitas prediksi untuk setiap kelas, dengan penandaan warna untuk perbandingan antara prediksi dan label yang benar.

In [45]:

```
import numpy as np
import matplotlib.pyplot as plt

def plot_image(i, predictions_array, true_label, img):
    # Ambil true_label dan gambar berdasarkan index `i`
    true_label, img = true_label.iloc[i], img.iloc[i].values.reshape(28, 28)

    # Menampilkan gambar
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img, cmap='gray') # Menggunakan 'gray' untuk gradasi yang lebih baik

    # Mendapatkan label prediksi
    predicted_label = np.argmax(predictions_array)
    color = 'blue' if predicted_label == true_label else 'red'

    # Memberi Label pada gambar
    plt.xlabel(f'{class_names[predicted_label]} {100*np.max(predictions_array):.2f}% ({class_names[true_label]})', color=color)

def plot_value_array(i, predictions_array, true_label):
    true_label = true_label.iloc[i]
    plt.grid(False)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    # Memberi warna pada bar
    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')

    # Menampilkan gambar prediksi pertama, prediksi label, dan array confidence
i = 19
plt.figure(figsize=(6, 3))

# Plot gambar prediksi
plt.subplot(1, 2, 1)
plot_image(i, predictions_sklearn[i], test_labels, test_images)

# Plot confidence array
plt.subplot(1, 2, 2)
plot_value_array(i, predictions_sklearn[i], test_labels)

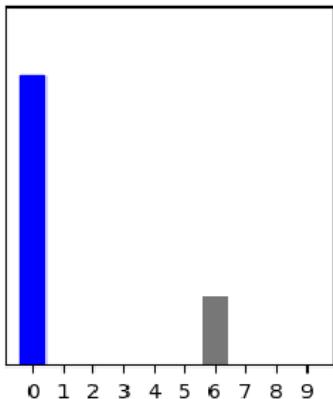
# Simpan dan tampilkan plot
# plt.savefig(f'image_{i}.png")
plt.show()
```



Watermarkly



T-shirt/top 80.87% (T-shirt/top)



- menyimpan gambar tertentu (tanpa label atau prediksi) ke dalam file JPEG dengan ukuran 28x28 piksel, menghilangkan sumbu dan grid pada gambar.

```
In [46]: import numpy as np
import matplotlib.pyplot as plt

# Fungsi untuk menyimpan gambar saja (tanpa prediksi dan Label)
def save_image_only(i, true_label, img):
    # Ambil gambar berdasarkan indeks `i` dan ubah menjadi ukuran 28x28
    img = img.iloc[i].values.reshape(28, 28)

    # Plot tanpa sumbu, label, dan grid
    plt.imshow(img, cmap='gray')
    plt.axis('off') # Menghilangkan sumbu
    plt.savefig(f"image_only_{i}.jpeg", bbox_inches='tight', pad_inches=0)
    plt.close()

# Contoh menyimpan gambar objek pertama
i = 19
save_image_only(i, test_labels, test_images)
```

- membandingkan kinerja dua model (TensorFlow dan Scikit-Learn) berdasarkan akurasi, arsitektur, dan penggunaan early stopping, serta menampilkan model terbaik berdasarkan akurasi tertinggi



Watermarkly

```
In [47]: import pandas as pd  
  
# Asumsikan `test_acc_tf` dan `test_acc_sklearn` sudah dihasilkan dari evaluasi model  
# Misalnya:  
# test_acc_tf = model_tf.evaluate(test_images, test_labels, verbose=2)[1]  
# test_acc_sklearn = accuracy_score(test_labels, test_predictions_sklearn)  
  
# Membuat dictionary hasil evaluasi  
data = {  
    'Model': ['TensorFlow', 'Scikit-Learn'],  
    'Accuracy': [test_acc_tf, test_acc_sklearn], # Menggunakan hasil evaluasi  
    'Architecture': ['Neural Network (2 hidden layers)', 'MLP (2 hidden layer)'],  
    'Early Stopping': ['Yes', 'Yes']  
}  
  
# Membuat DataFrame untuk menampilkan tabel perbandingan  
comparison_df = pd.DataFrame(data)  
  
# Menentukan model terbaik berdasarkan akurasi tertinggi  
best_model = comparison_df.loc[comparison_df['Accuracy'].idxmax()]  
  
# Menampilkan tabel perbandingan  
print("Comparison of Model Performance:")  
print(comparison_df)  
  
# Menampilkan model terbaik  
print("\nBest Model Based on Accuracy:")  
print(best_model)
```

```
Comparison of Model Performance:  
      Model  Accuracy          Architecture Early Stopping  
0   TensorFlow  0.8826  Neural Network (2 hidden layers)    Yes  
1  Scikit-Learn  0.8871        MLP (2 hidden layer)    Yes
```

```
Best Model Based on Accuracy:  
Model           Scikit-Learn  
Accuracy       0.8871  
Architecture    MLP (2 hidden layer)  
Early Stopping     Yes  
Name: 1, dtype: object
```

Dump Model

```
In [48]: #Cara dump pickle jika model sklearn yang terbaik  
import pickle  
  
# Misalnya `model_sklearn` adalah model terbaik Anda dari Scikit-Learn  
with open('best_model.pkl', 'wb') as file:  
    pickle.dump(model_sklearn, file)
```

```
In [49]: #Cara dump h5 jika model tensorflow yang terbaik  
import pickle  
import tensorflow as tf  
  
# Menyimpan model TensorFlow  
model_tf.save('best_model.h5')
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.



Watermarkly

Streamlit menggunakan hasil dump dari tensorflow

- Buat antarmuka Streamlit pada file yang berekstensi Python (.py)
- Buatlah sesuai dengan code berikut ini

```
In [50]: import streamlit as st
import tensorflow as tf
import numpy as np
from PIL import Image
import os

# Definisikan jalur model
model_path = r'D:\SURYA\UAJY\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul Deep Learning>Notebook\best_model_tf.h5'

# Muat model
if os.path.exists(model_path):
    try:
        # Mengurangi verbosity dari TensorFlow
        tf.get_logger().setLevel('ERROR')
        model = tf.keras.models.load_model(model_path, compile=False)

        # Nama kelas untuk Fashion MNIST
        class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                       'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

        # Fungsi untuk memproses gambar
        def preprocess_image(image):
            image = image.resize((28, 28)) # Ubah ukuran menjadi 28x28 piksel
            image = image.convert('L') # Ubah menjadi grayscale
            image_array = np.array(image) / 255.0 # Normalisasi
            image_array = image_array.reshape(1, 28, 28, 1) # Ubah bentuk menjadi 4D array
            return image_array

        # UI Streamlit
        st.title("Fashion MNIST Image Classifier")
        st.write("Unggah gambar item fashion (misalnya sepatu, tas, baju), dan model akan memprediksi kelasnya.")

        # File uploader untuk input gambar
        uploaded_file = st.file_uploader("Pilih gambar...", type=["jpg", "jpeg", "png"])

        # Tampilkan gambar yang diunggah dan tombol "Predict"
        if uploaded_file is not None:
            # Tampilkan gambar yang diunggah
            image = Image.open(uploaded_file)
            st.image(image, caption="Gambar yang Diunggah", use_column_width=True)

            # Tombol "Predict"
            if st.button("Predict"):
                # Proses gambar dan prediksi
                processed_image = preprocess_image(image)
                predictions = model.predict(processed_image)[0]

                # Mendapatkan kelas dan confidence dengan softmax
                predicted_class = np.argmax(predictions)
                confidence = predictions[predicted_class] * 100 # Pastikan confidence dalam persentase

                # Tampilkan hasil prediksi
                st.write("## Hasil Prediksi")
                st.write(f"Kelas Prediksi: **{class_names[predicted_class]}**")
                st.write(f"Confidence: **{confidence:.2f}**")

        except Exception as e:
            st.error(f"Error: {str(e)}")
    else:
        st.error("File model tidak ditemukan.")
```

2024-11-15 19:00:03.175

Warning: to view this Streamlit app on a browser, run it with the following command:

```
streamlit run c:\ProgramData\anaconda3\Lib\site-packages\ipykernel_launcher.py [ARGUMENTS]
```

Streamlit menggunakan hasil dump dari scikit-learn

- Buat antarmuka Streamlit pada file yang berekstensi Python (.py)
- Buatlah sesuai dengan code berikut ini

```
In [51]: import streamlit as st
import numpy as np
import pickle
from PIL import Image
import os

# Definisikan jalur model
model_directory = r'D:\SURYA\UAJY\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul Deep Learning>Notebook'
model_path = os.path.join(model_directory, r'best_model.pkl')

# Load the model
if os.path.exists(model_path):
    try:
        with open(model_path, 'rb') as model_file:
            model = pickle.load(model_file)

        # Nama kelas untuk Fashion MNIST
        class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                       'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

        # Fungsi untuk memproses gambar
        def preprocess_image(image):
            image = image.resize((28, 28)) # Ubah ukuran menjadi 28x28 piksel
            image = image.convert('L') # Ubah menjadi grayscale
            image_array = np.array(image) / 255.0 # Normalisasi
            image_array = image_array.reshape(1, -1) # Flatten ke bentuk 1D array
            return image_array

        # UI Streamlit
        st.title("Fashion MNIST Image Classifier")
        st.write("Unggah beberapa gambar item fashion (misalnya sepatu, tas, baju), dan model akan memprediksi kelas masing-masing.")

        # File uploader untuk input gambar (multiple files)
        uploaded_files = st.file_uploader("Pilih gambar...", type=["jpg", "jpeg", "png"], accept_multiple_files=True)

        # Sidebar dengan tombol "Predict" dan hasil prediksi
        with st.sidebar:
            st.write("## Navigator")
            predict_button = st.button("Predict") # Tombol di sidebar

            # Tampilkan hasil prediksi di bawah tombol "Predict"
            if uploaded_files and predict_button:
                st.write("### Hasil Prediksi")

                for uploaded_file in uploaded_files:
                    # Buka dan proses setiap gambar
                    image = Image.open(uploaded_file)
                    processed_image = preprocess_image(image)
                    predictions = model.predict_proba(processed_image)
                    predicted_class = np.argmax(predictions)
                    confidence = np.max(predictions) * 100

                    # Tampilkan nama file dan hasil prediksi
                    st.write(f"**Nama File:** {uploaded_file.name}")
                    st.write(f"Kelas Prediksi: **{class_names[predicted_class]}**")
                    st.write(f"Confidence: **{confidence:.2f}%**")
                    st.write("---") # Garis pemisah antara hasil prediksi

        # Tampilkan gambar yang diunggah di halaman utama
        if uploaded_files:
            for uploaded_file in uploaded_files:
                image = Image.open(uploaded_file)
                st.image(image, caption=f"Gambar: {uploaded_file.name}", use_column_width=True)

    except Exception as e:
```

```
    st.error(f"Error: {str(e)}")
else:
    st.error("File model tidak ditemukan.")
```