TF & KERAS. РОБОТА із ЗОБРАЖЕННЯМИ

Файл: TF_KERAS_Image_04_002

Створення простої CNN мережі з використанням набору даних MNIST або fashion MNIST

Example from [A simple 2D CNN for MNIST digit recognition]

Імпорт необхідних бібліотек

```
import tensorflow as tf
import keras
import numpy as np
import matplotlib.pyplot as plt
import random
```

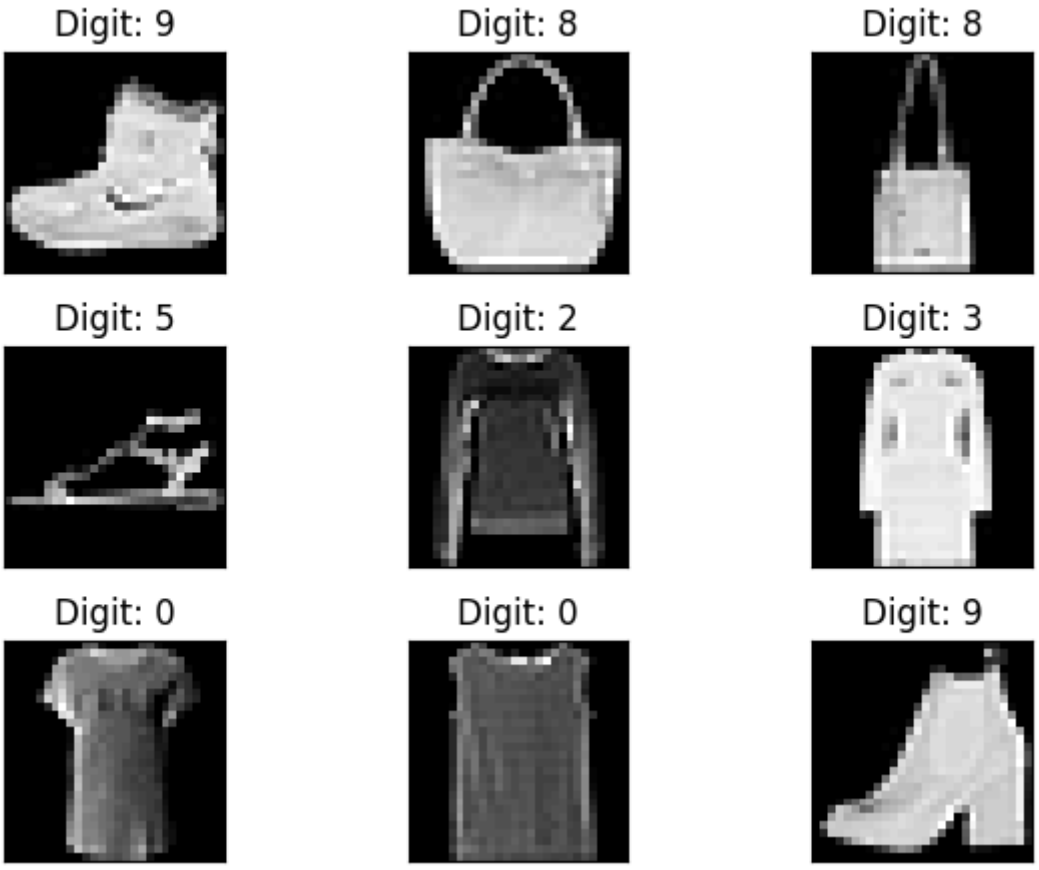
Завантаження набору даних з MNIST

```
# (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
(X_train, y_train), (X_test, y_test) =
tf.keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

Візуалізація тренувального набору

```
fig = plt.figure()
for i in range(9):
  plt.subplot(3,3,i+1)
  plt.tight_layout()
  plt.imshow(X_train[100*i], cmap='gray', interpolation='none')
  plt.title("Digit: {}".format(y_train[100*i]))
  plt.xticks([])
  plt.yticks([])
# fig
```




```
'''
We are working with "channels_last"
'''


#reshaping
#this assumes our data format
#For 3D data, "channels_last" assumes (conv_dim1, conv_dim2, conv_dim3, channels)
while
#"channels_first" assumes (channels, conv_dim1, conv_dim2, conv_dim3).

'''
if k.image_data_format() == 'channels_first':
    #X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    #X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    #input_shape = (1, img_rows, img_cols)
else:
    #X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
    #X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    #input_shape = (img_rows, img_cols, 1)
'''
```

Нормалізування значення пікселів до діапазону [0, 1]

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
y_train = y_train.astype(np.int32)
y_test = y_test.astype(np.int32)
Y_test = y_test.copy()

print('X_train shape:', X_train.shape) #X_train shape: (60000, 28, 28, 1)
print('y_train shape:', y_train.shape) #Y_train shape: (60000, )

# Визначення формату вхыдних даних
img_rows = X_train.shape[1]
img_cols = X_train.shape[2]
input_shape = (img_rows, img_cols, 1)
print('Input shape:', input_shape)
```

```
X_train shape: (60000, 28, 28)
y_train shape: (60000,)
Input shape: (28, 28, 1)
```

```
print(y_test[0], Y_test[0])
```

```
9 9
```

```
# Встановлюємо кількість категорій
num_category = 10
# Перетворюємо вектор класів до двійкової мтриці класів
y_train = keras.utils.to_categorical(y_train, num_category)
y_test = keras.utils.to_categorical(y_test, num_category)
print('Y_train shape:', y_train.shape) #Y_train shape: (60000, 10)
print('Y_test shape:', y_test.shape) #Y_train shape: (60000, 10)
```

```
print(y_test[0],Y_test[0])
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.] 9
```

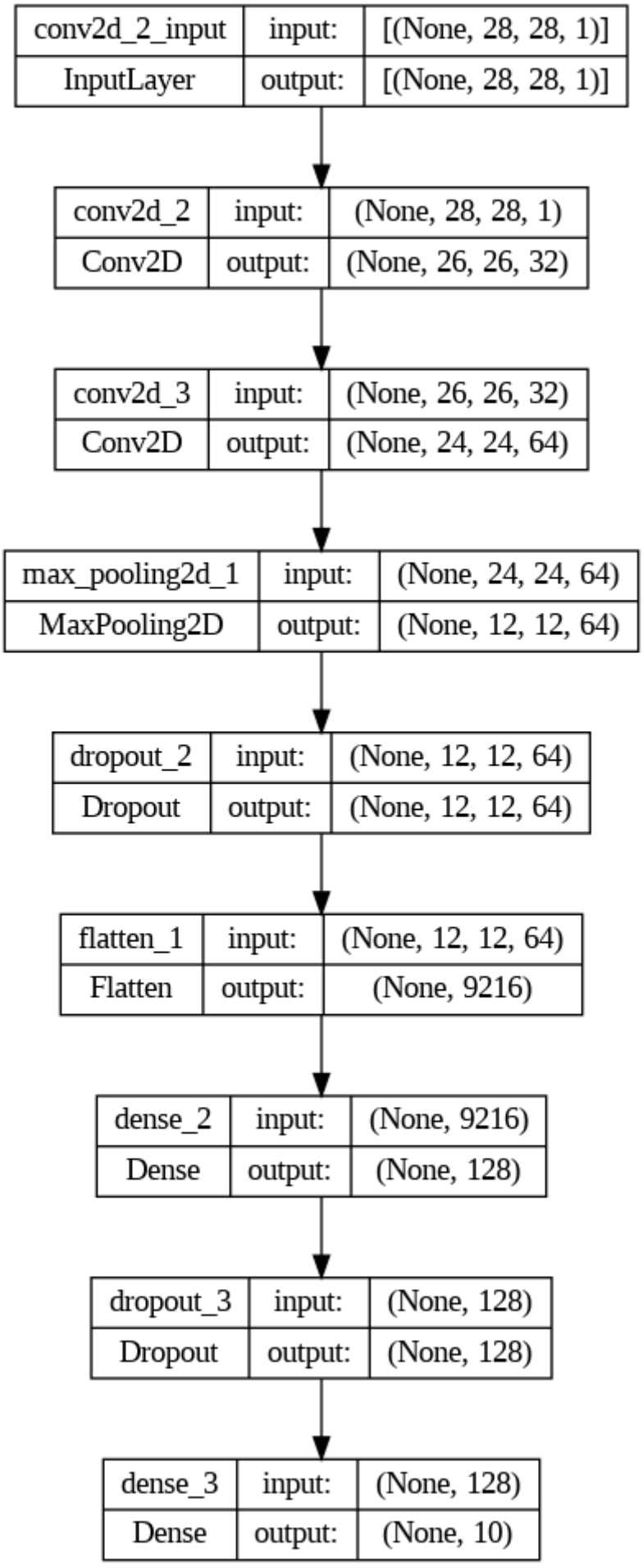
Створення моделі

```
## Побудова моделі
cnnmodel = keras.Sequential()
#1-й згортковий шар, активація ReLU
#використовуємо 32 згоркових ядра, кожний розміром 3x3
cnnmodel.add(keras.layers.Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
```

```
                 input_shape=input_shape))

#2-й згортковий шар, активація ReLU
#використовуємо 64 згоркових ядра, кожний розміром 3x3
cnnmodel.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))

# Шар пулінгу - підвибірка по максімуму ядра 2*2
cnnmodel.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))

# Випадково включаємо (виключаємо) нейрони для поліпшення збіжності
cnnmodel.add(keras.layers.Dropout(0.25))

# Шар згладжування (перевorення виходу до плоского вектору )
cnnmodel.add(keras.layers.Flatten())

#Повносвящний шар 128 нейронів
cnnmodel.add(keras.layers.Dense(128, activation='relu'))

# Ще раз випадково включаємо (виключаємо) нейрони для поліпшення збіжності
cnnmodel.add(keras.layers.Dropout(0.5))

# Вихідний шар 10 нейронів, активація oftmax
cnnmodel.add(keras.layers.Dense(num_category, activation='softmax'))
```

Відображення моделі

```
tf.keras.utils.plot_model(cnnmodel, 'multi_input_and_output_model.png',
show_shapes=True)
```


conv2d_2_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d_2 | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 32)
conv2d_3 | input: | (None, 26, 26, 32)
Conv2D | output: | (None, 24, 24, 64)
max_pooling2d_1 | input: | (None, 24, 24, 64)
MaxPooling2D | output: | (None, 12, 12, 64)
dropout_2 | input: | (None, 12, 12, 64)
Dropout | output: | (None, 12, 12, 64)
flatten_1 | input: | (None, 12, 12, 64)
Flatten | output: | (None, 9216)
dense_2 | input: | (None, 9216)
Dense | output: | (None, 128)
dropout_3 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_3 | input: | (None, 128)
Dense | output: | (None, 10)

Компіляція моделі

```
# Оптимізація Adaptive learning rate (adaDelta)
# Метрика accuracy
cnnmodel.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Визначення гіперпараметрів нейронної мережі

```
batch_size = 128 # Розмір батчу
num_epoch = 20   # Кількість епох
```

Тренування моделі

```
#model training
model_log = cnnmodel.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=num_epoch,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Epoch 1/20
469/469 [==============================] - 5s 10ms/step - loss: 0.5049 - accuracy:
0.8150 - val_loss: 0.4319 - val_accuracy: 0.8438
Epoch 2/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4898 - accuracy:
0.8206 - val_loss: 0.4168 - val_accuracy: 0.8520
Epoch 3/20
469/469 [==============================] - 4s 10ms/step - loss: 0.4703 - accuracy:
0.8288 - val_loss: 0.3944 - val_accuracy: 0.8595
Epoch 4/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4633 - accuracy:
0.8304 - val_loss: 0.3852 - val_accuracy: 0.8605
Epoch 5/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4513 - accuracy:
0.8357 - val_loss: 0.3739 - val_accuracy: 0.8638
Epoch 6/20
469/469 [==============================] - 5s 10ms/step - loss: 0.4364 - accuracy:
0.8398 - val_loss: 0.3741 - val_accuracy: 0.8681
Epoch 7/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4258 - accuracy:
0.8450 - val_loss: 0.3581 - val_accuracy: 0.8716
Epoch 8/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4164 - accuracy:
0.8471 - val_loss: 0.3457 - val_accuracy: 0.8766
Epoch 9/20
469/469 [==============================] - 7s 14ms/step - loss: 0.4051 - accuracy:
0.8533 - val_loss: 0.3369 - val_accuracy: 0.8783
```

```
Epoch 10/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4003 - accuracy:
0.8532 - val_loss: 0.3367 - val_accuracy: 0.8832
Epoch 11/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3907 - accuracy:
0.8571 - val_loss: 0.3299 - val_accuracy: 0.8832
Epoch 12/20
469/469 [==============================] - 6s 13ms/step - loss: 0.3806 - accuracy:
0.8603 - val_loss: 0.3222 - val_accuracy: 0.8850
Epoch 13/20
469/469 [==============================] - 5s 10ms/step - loss: 0.3779 - accuracy:
0.8612 - val_loss: 0.3168 - val_accuracy: 0.8880
Epoch 14/20
469/469 [==============================] - 5s 10ms/step - loss: 0.3737 - accuracy:
0.8608 - val_loss: 0.3211 - val_accuracy: 0.8871
Epoch 15/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3703 - accuracy:
0.8642 - val_loss: 0.3078 - val_accuracy: 0.8902
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3655 - accuracy:
0.8668 - val_loss: 0.3132 - val_accuracy: 0.8903
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3627 - accuracy:
0.8667 - val_loss: 0.3073 - val_accuracy: 0.8912
Epoch 18/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3600 - accuracy:
0.8671 - val_loss: 0.3021 - val_accuracy: 0.8917
Epoch 19/20
469/469 [==============================] - 4s 10ms/step - loss: 0.3562 - accuracy:
0.8695 - val_loss: 0.2980 - val_accuracy: 0.8956
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3505 - accuracy:
0.8719 - val_loss: 0.3012 - val_accuracy: 0.8922
```

Функція для прогнозування зображень

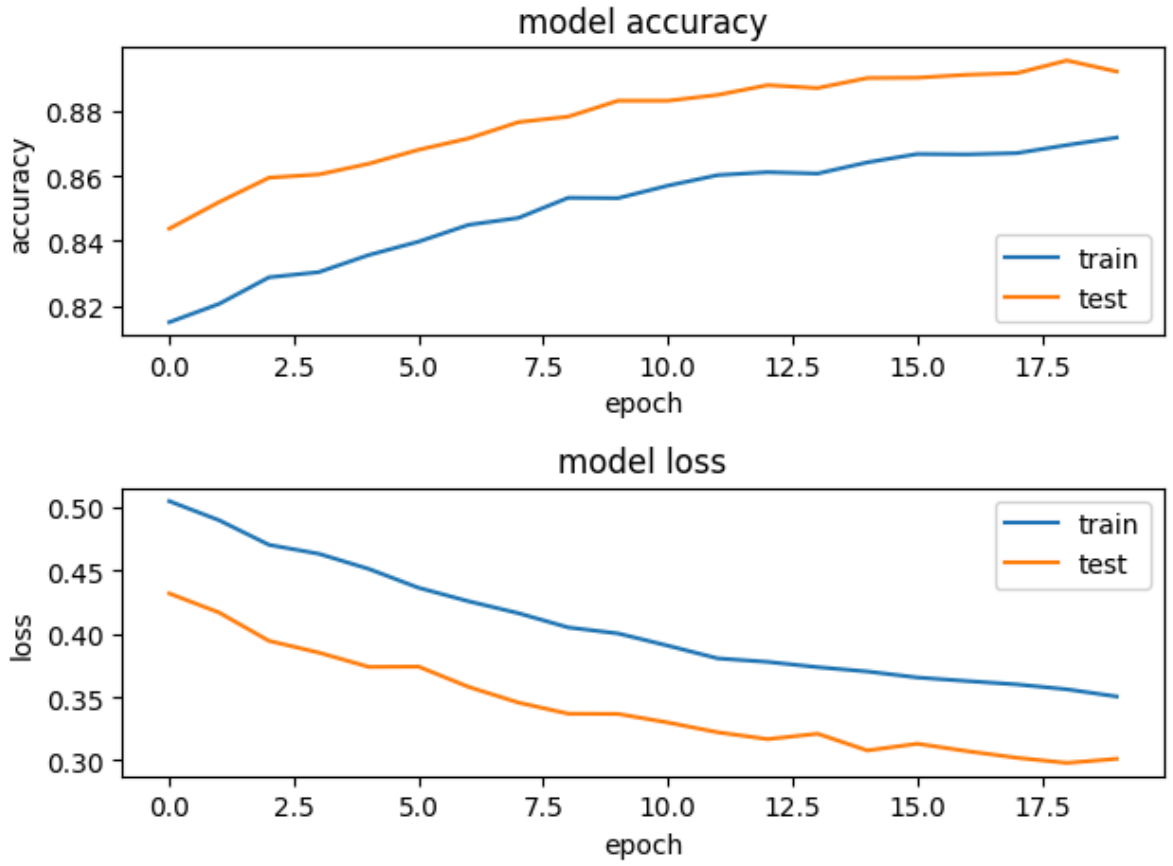
```
score = cnnmodel.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.3012251853942871
Test accuracy: 0.8921999931335449
```

Графік для перевірки прогресу навчання

```
# plotting the metrics
fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(model_log.history['accuracy'])
plt.plot(model_log.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
plt.legend(['train', 'test'], loc='lower right')
plt.subplot(2,1,2)
plt.plot(model_log.history['loss'])
plt.plot(model_log.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.tight_layout()
# fig
```



Назви класів

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat','Sandal',
'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
# class_names = ['0', '1', '2', '3', '4',
#                '5', '6', '7', '8', '9']
```

Створення функції для побудови випадкового зображення разом з його прогнозом

```
def predict(X):
    logits = cnnmodel(X)
    return tf.argmax(logits, axis=1, output_type=tf.int32)
```

```
# Створити функцію для побудови випадкового зображення разом з його прогнозом
def plot_random_image(model, images, true_labels, classes):
  plt.figure(figsize=(10, 10))
  for i in range(4):
    ax = plt.subplot(2, 2, i + 1)
    target_image = images[i]
    pred_probs = model.predict(target_image.reshape(1, 28, 28))
    pred_label = classes[pred_probs.argmax()]
    true_label = classes[true_labels[i]]
    plt.imshow(target_image, cmap=plt.cm.binary)
    if pred_label == true_label:
      color = "green"
    else:
      color = "red"
    plt.title("{} Pred: {} {:2.0f}% (True:
{})".format(class_names[true_labels[i]], pred_label,
100*tf.reduce_max(pred_probs), true_label), color=color)
    plt.axis(False)
```

Тестування роботи написанної та навченної нейронної мережі на випадкових зображень з тестового набору

```
plot_random_image(cnnmodel, X_test, Y_test, class_names)
```

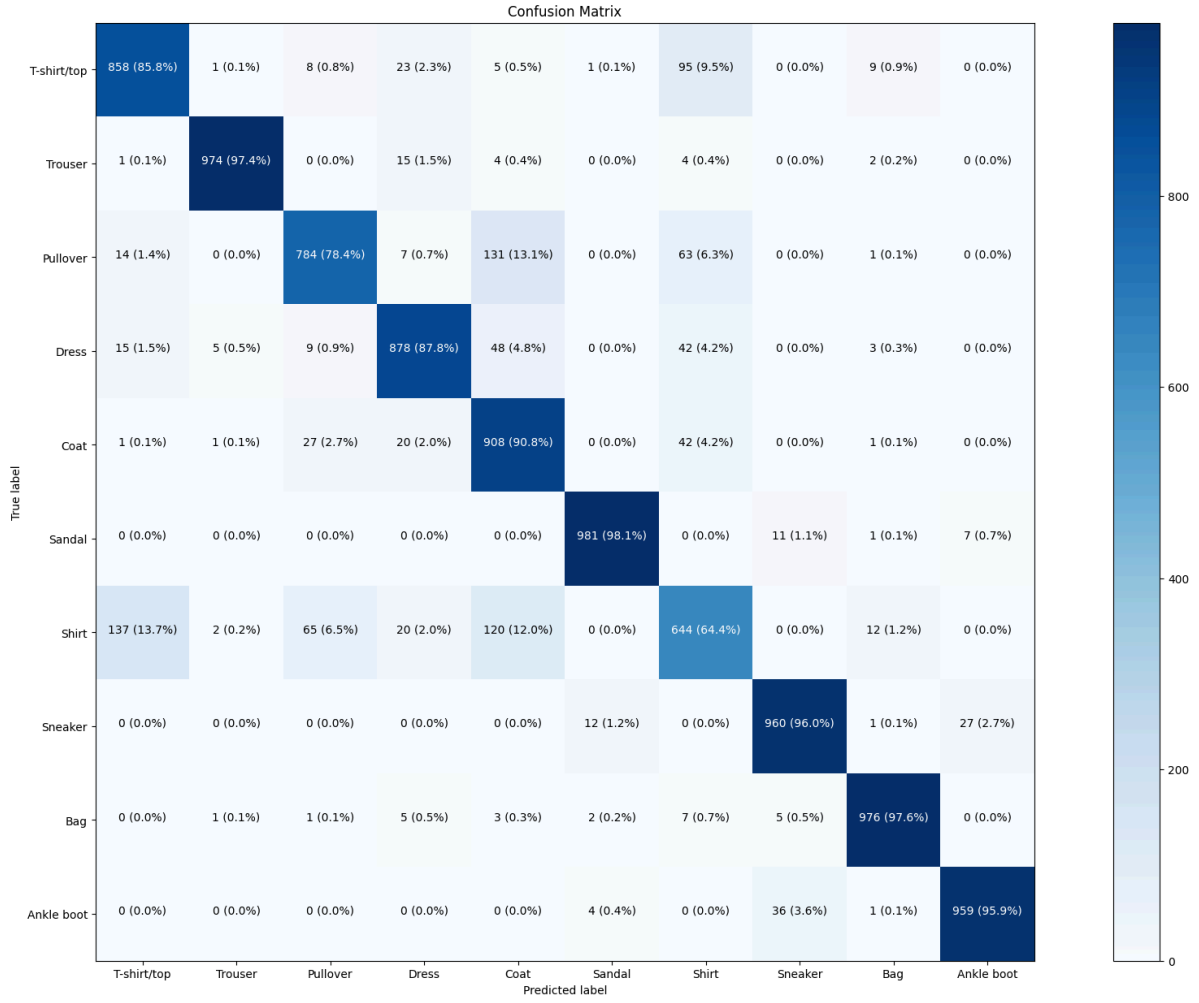
```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
```

Створення confusion matrix для моделі нейронної мережі

```
import itertools
from sklearn.metrics import confusion_matrix

def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(8, 8),
text_size=15):

  # Створення confusion matrix
  cm = confusion_matrix(y_true, y_pred)
  cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] # нормалізація
  n_classes = cm.shape[0] # знаходимо кількість класів, з якими ми маємо справу

  # Побудуємо фігуру
  fig, ax = plt.subplots(figsize=figsize)
  cax = ax.matshow(cm, cmap=plt.cm.Blues) # кольори показуватимуть, наскільки
"правильним" є клас, темніший == кращий
  fig.colorbar(cax)

  # Чи є список класів?
```

```
  if classes:
    labels = classes
  else:
    labels = np.arange(cm.shape[0])

  # Позначення осей
  ax.set(title="Confusion Matrix",
         xlabel="Predicted label",
         ylabel="True label",
         xticks=np.arange(n_classes), # створити достатню кількість слотів осей для кожного класу
         yticks=np.arange(n_classes),
         xticklabels=labels, # осі будуть позначені іменами класів (якщо вони існують) або індексами
         yticklabels=labels)

  # Зробимо так, щоб мітки осі x з'являлися внизу
  ax.xaxis.set_label_position("bottom")
  ax.xaxis.tick_bottom()

  # Встановіть поріг для різних кольорів
  threshold = (cm.max() + cm.min()) / 2.

  # Побудуйте текст у кожній комірці
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, f"{cm[i, j]} ({cm_norm[i, j]*100:.1f}%)",
             horizontalalignment="center",
             color="white" if cm[i, j] > threshold else "black",
             size=text_size)
```

```
logits = cnnmodel(X_test)
predictions = tf.argmax(logits, axis=1, output_type=tf.int32)
make_confusion_matrix(y_true=Y_test,
                      y_pred=predictions,
                      classes=class_names,
                      figsize=(25, 15),
                      text_size=10)
```


Confusion Matrix
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
True label
Predicted label
858 (85.8%) 1 (0.1%) 8 (0.8%) 23 (2.3%) 5 (0.5%) 1 (0.1%) 95 (9.5%) 0 (0.0%) 9 (0.9%) 0 (0.0%)
1 (0.1%) 974 (97.4%) 0 (0.0%) 15 (1.5%) 4 (0.4%) 0 (0.0%) 4 (0.4%) 0 (0.0%) 2 (0.2%) 0 (0.0%)
14 (1.4%) 0 (0.0%) 784 (78.4%) 7 (0.7%) 131 (13.1%) 0 (0.0%) 63 (6.3%) 0 (0.0%) 1 (0.1%) 0 (0.0%)
15 (1.5%) 5 (0.5%) 9 (0.9%) 878 (87.8%) 48 (4.8%) 0 (0.0%) 42 (4.2%) 0 (0.0%) 3 (0.3%) 0 (0.0%)
1 (0.1%) 1 (0.1%) 27 (2.7%) 20 (2.0%) 908 (90.8%) 0 (0.0%) 42 (4.2%) 0 (0.0%) 1 (0.1%) 0 (0.0%)
0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 981 (98.1%) 0 (0.0%) 11 (1.1%) 1 (0.1%) 7 (0.7%)
137 (13.7%) 2 (0.2%) 65 (6.5%) 20 (2.0%) 120 (12.0%) 0 (0.0%) 644 (64.4%) 0 (0.0%) 12 (1.2%) 0 (0.0%)
0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 12 (1.2%) 0 (0.0%) 960 (96.0%) 1 (0.1%) 27 (2.7%)
0 (0.0%) 1 (0.1%) 1 (0.1%) 5 (0.5%) 3 (0.3%) 2 (0.2%) 7 (0.7%) 5 (0.5%) 976 (97.6%) 0 (0.0%)
0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 4 (0.4%) 0 (0.0%) 36 (3.6%) 1 (0.1%) 959 (95.9%)
800
600
400
200
0