TF & KERAS. РОБОТА із ЗОБРАЖЕННЯМИ

Файл: TF_KERAS_Image_03_0021

Створення простої нейронної мережі з використанням набору даних MNIST KERAS

MNIST

Вбудована база данных зображень рукописних цифр

Вбудована база данных зображень одягу (fashon)

Імпортування бібліотек

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

```
print(tf.executing_eagerly())
```

```
True
```

Завантаження набору даних fashion MNIST

Завантаження та підготовка дадасету MNIST dataset. Значення кожного пиксела (інтенсівність) змінюється від 0 до 255. Масштабування інтесивності до діапазону від 0 до 1 (ділення на `255.0`).

Автоматичне привдення INTEGER дані до FLOAT

```
(train_data, train_labels), (test_data, test_labels) =
tf.keras.datasets.mnist.load_data()
# (train_data, train_labels), (test_data, test_labels) =
tf.keras.datasets.fashion_mnist.load_data()
```

```
print(train_data.shape)
print(train_labels.shape)
print(test_data.shape)
print(test_labels.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

Прописування назви класів зображень

```
class_names = ['Нуль', 'One', 'Two', 'Three', 'Four', 'Five',
               'Six', 'Seven', 'Eghit', 'Nine']
#class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
#               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Нормалізація даних

Нормалізація даних є важливим кроком у підготовці даних для глибокого навчання. Вона допомагає поліпшити продуктивність моделі та прискорити процес навчання.

Нормалізація даних означає масштабування значень ознак вхідних даних таким чином, щоб вони мали порівнянний масштаб. Це може бути зроблено шляхом віднімання середнього значення ознаки і ділення на стандартне відхилення ознаки. Також можливі інші методи нормалізації, наприклад, масштабування в діапазон [0,1] або [-1,1].

Нормалізація допомагає зменшити розкид значень ознак, що може допомогти поліпшити стабільність навчання, прискорити збіжність алгоритму градієнтного спуску та зменшити ймовірність вибуху градієнта. Це також може зменшити залежність моделі від початкових значень ваг і зробити її більш стійкою до шуму в даних.

Таким чином, нормалізація даних є важливим кроком у підготовці даних для глибокого навчання, який допомагає поліпшити продуктивність моделі та прискорити процес навчання.

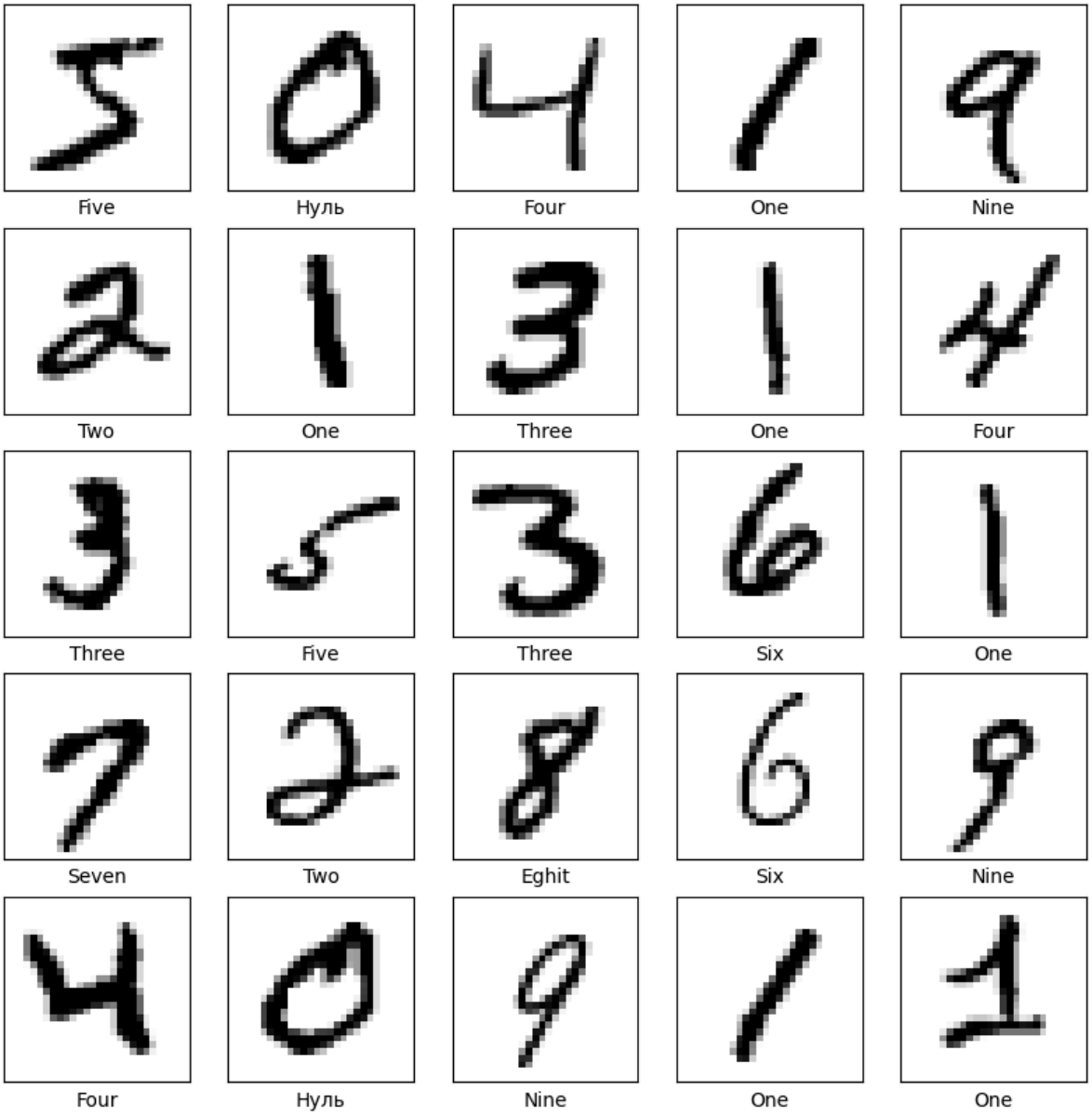
```
train_data = train_data / 255.0
test_data = test_data / 255.0
```

```
print(len(train_data))
print(len(test_data))
```

```
60000
10000
```

Для перевірки: відображаємо перші 25 зображень з навчального набору та під кожним зображенням надаємо назву класу .

```
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(train_data[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



Побудова та складання моделі

Послідовна модель Keras з чотирьма різними шарами. Модель являє собою нейронну мережу прямого поширення (MLP -> передає значення зліва направо).

Шар 1: Вхідний шар, і він складатиметься з 784 нейронів. Використовуємо шар **flatten** з формою вхідних даних (28,28) для позначення того, що вхідні дані повинні надходити саме в такій формі. **flatten** означає, що шар перетворить масив форми (28,28) у вектор з 784 нейронів так, щоб кожен піксель був пов'язаний з одним нейроном.

Шар 2: Перший прихований шар. Шар типу **Dense** визначає, що цей шар буде повністю пов'язаним і кожен нейрон з попереднього шару з'єднується з кожним нейроном цього шару. Він має 256 нейронів і використовує функцію активації **RELU**.

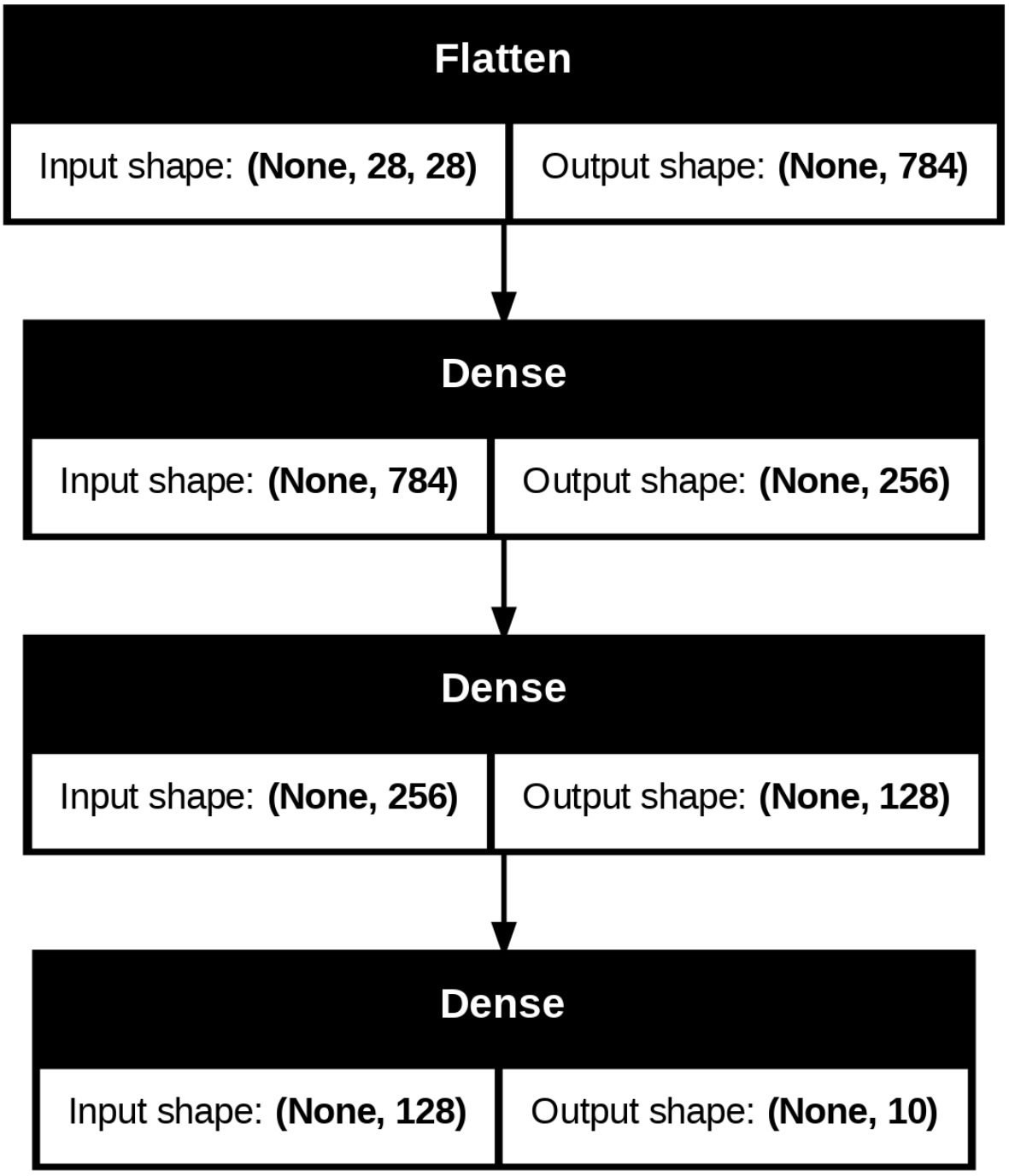
Шар 3: Другий прихований шар. Шар типу **Dense**. Має 128 нейронів і використовує функцію **RELU**.

Шар 4: Вихідний шар. Шар типу **Dense**. Має 10 нейронів, вихід яких використовується для визначення вихідних даних моделі. Кожен нейрон представляє ймовірність того, що дане зображення належить до одного з 10 різних класів. Функція активації **softmax** використовується на цьому шарі для обчислення розподілу ймовірностей для кожного класу. Це визначає, що значення будь-якого нейрона в цьому шарі буде між 0 і 1, де 1 означає високу ймовірність того, що зображення належить до цього класу.

```
#tf.random.set_seed(42) # задання константи для фіксування результатів
model = tf.keras.Sequential() # створення послідовної моделі keras
#model.add(tf.keras.layers.Input(shape=((28, 28, 1 )))) # Вхід
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(256, activation="relu")) # 1-й Прихований шар)
model.add(tf.keras.layers.Dense(128, activation="relu")) # 2-й Прихований шар
model.add(tf.keras.layers.Dense(10, activation="softmax")) # Вихідний шар
```

Відображення моделі

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



Компіляція моделі. Визначення функції втрат (**Crossentropy**), оптимізатора (**Adam**), метрики (**accuracy**).

```
# Компіляція моделі
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.002),
              metrics=["accuracy"])
```

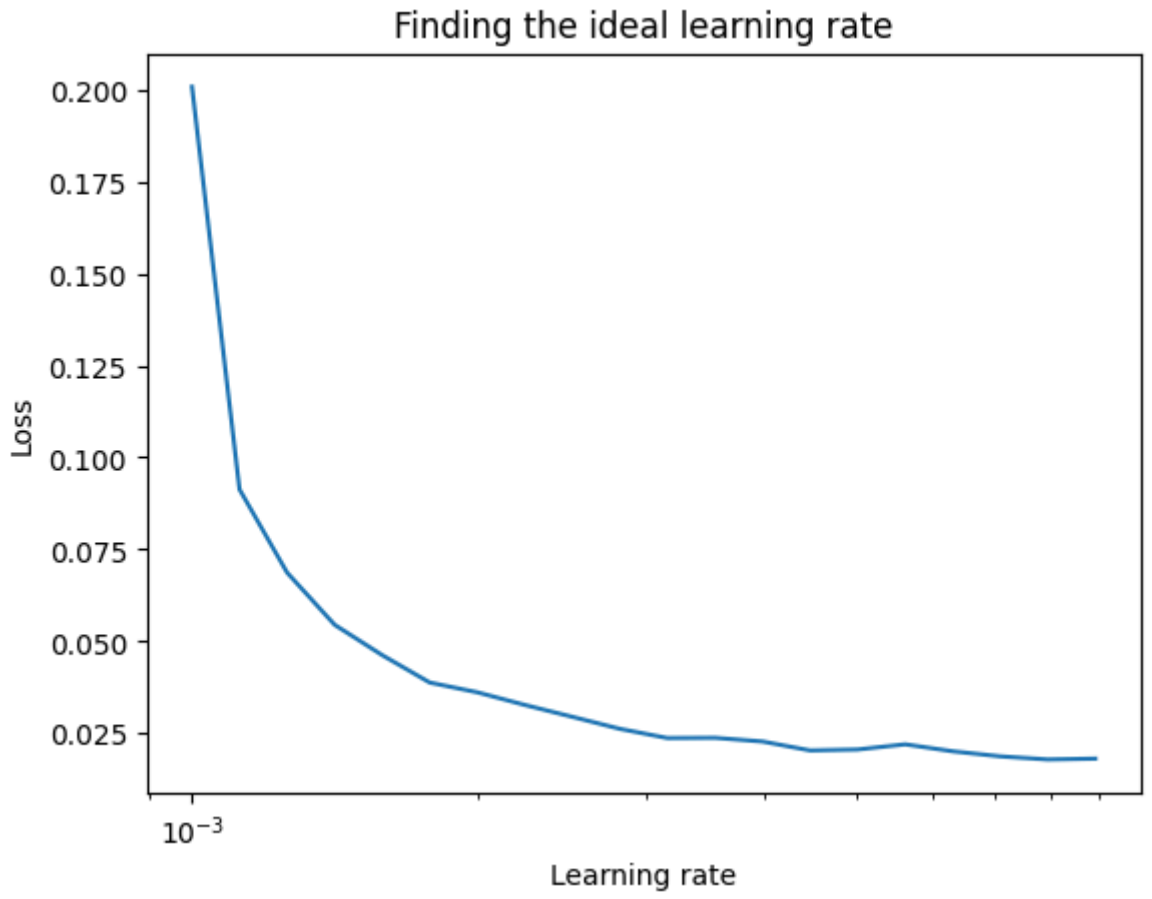
Навчання моделі. Зберігання метрик для відсліжування

```
epoch_num = 20
#lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epochs: 1e-3 *
10**(epochs/20))
history = model.fit(train_data, # навчання моделі
                                train_labels,
                                epochs=epoch_num,
                                validation_data=(test_data, test_labels))
```

```
Epoch 1/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m7s•[0m 2ms/step -
accuracy: 0.8965 - loss: 0.3311 - val_accuracy: 0.9641 - val_loss: 0.1191
Epoch 2/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9725 - loss: 0.0895 - val_accuracy: 0.9746 - val_loss: 0.0831
Epoch 3/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9802 - loss: 0.0625 - val_accuracy: 0.9778 - val_loss: 0.0783
Epoch 4/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9847 - loss: 0.0498 - val_accuracy: 0.9773 - val_loss: 0.0835
Epoch 5/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9858 - loss: 0.0427 - val_accuracy: 0.9763 - val_loss: 0.0875
Epoch 6/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9895 - loss: 0.0317 - val_accuracy: 0.9759 - val_loss: 0.0969
Epoch 7/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9898 - loss: 0.0323 - val_accuracy: 0.9799 - val_loss: 0.0913
Epoch 8/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9913 - loss: 0.0272 - val_accuracy: 0.9805 - val_loss: 0.0905
Epoch 9/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9924 - loss: 0.0242 - val_accuracy: 0.9804 - val_loss: 0.0929
Epoch 10/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m6s•[0m 2ms/step -
accuracy: 0.9928 - loss: 0.0216 - val_accuracy: 0.9792 - val_loss: 0.1038
Epoch 11/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9949 - loss: 0.0180 - val_accuracy: 0.9773 - val_loss: 0.1067
Epoch 12/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9936 - loss: 0.0200 - val_accuracy: 0.9774 - val_loss: 0.1233
Epoch 13/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m6s•[0m 2ms/step -
accuracy: 0.9936 - loss: 0.0209 - val_accuracy: 0.9773 - val_loss: 0.1258
Epoch 14/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9942 - loss: 0.0192 - val_accuracy: 0.9771 - val_loss: 0.1428
Epoch 15/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9941 - loss: 0.0185 - val_accuracy: 0.9795 - val_loss: 0.1521
Epoch 16/20
```

```
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9951 - loss: 0.0169 - val_accuracy: 0.9775 - val_loss: 0.1531
Epoch 17/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m4s•[0m 2ms/step -
accuracy: 0.9954 - loss: 0.0174 - val_accuracy: 0.9778 - val_loss: 0.1466
Epoch 18/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9952 - loss: 0.0172 - val_accuracy: 0.9815 - val_loss: 0.1309
Epoch 19/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m6s•[0m 2ms/step -
accuracy: 0.9962 - loss: 0.0146 - val_accuracy: 0.9773 - val_loss: 0.1525
Epoch 20/20
•[1m1875/1875•[0m •[32m--------------------•[0m•[37m•[0m •[1m5s•[0m 2ms/step -
accuracy: 0.9956 - loss: 0.0134 - val_accuracy: 0.9807 - val_loss: 0.1778
```

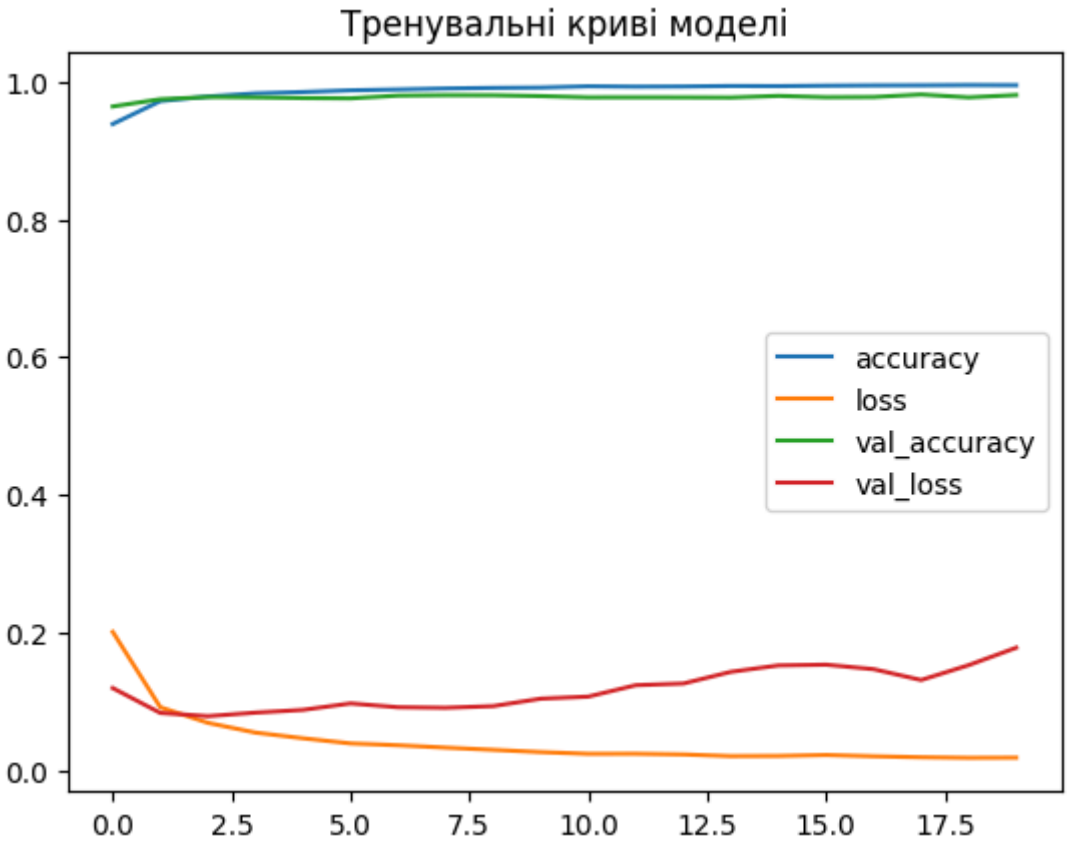
```python
# Plot the learning rate decay curve
import numpy as np
import matplotlib.pyplot as plt
lrs = 1e-3 * (10**(np.arange(epoch_num)/20))
plt.semilogx(lrs, history.history["loss"]) # want the x-axis to be log-scale
plt.xlabel("Learning rate")
plt.ylabel("Loss")
plt.title("Finding the ideal learning rate");
```



Графік для перевірки прогресу навчання

```
pd.DataFrame(history.history).plot(title="Тренувальні криві моделі")
```

```
<Axes: title={'center': 'Тренувальні криві моделі'}>
```




```
# 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):
    im_num = i + 20
    ax = plt.subplot(2, 2, i + 1)
    target_image = images[im_num]
    pred_probs = model.predict(target_image.reshape(1, 28, 28))
    pred_label = classes[pred_probs.argmax()]
    true_label = classes[true_labels[im_num]]
    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[im_num]], pred_label,
100*tf.reduce_max(pred_probs), true_label), color=color)
```

```
    plt.axis(False)
```

```
plot_random_image(model, test_data, test_labels, class_names)
```

```
•[1m1/1•[0m •[32m--------------------•[0m•[37m•[0m •[1m0s•[0m 16ms/step
•[1m1/1•[0m •[32m--------------------•[0m•[37m•[0m •[1m0s•[0m 15ms/step
•[1m1/1•[0m •[32m--------------------•[0m•[37m•[0m •[1m0s•[0m 15ms/step
•[1m1/1•[0m •[32m--------------------•[0m•[37m•[0m •[1m0s•[0m 22ms/step
```

Оцінка класифікації. Confusion Matrix.

Confusion Matrix - це таблиця, яка представляє собою підсумок результатів прогнозування для задачі класифікації. Розташування прогнозованих і фактичних значень змінює положення хибнонегативних (FN) і хибнопозитивних (FP), але істиннопозитивні (TP) і істиннонегативні (TN) залишаються на тому ж місці в матриці, розміщені по діагоналі один до одного. Але через це ситуація стає заплутаною.

З confusion matrix ми також можемо обчислити accuracy, precision та F1 score. Ці показники допомагають зрозуміти продуктивність моделі.

```
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)

  # Список класів присутен?
  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)

  # Мітки осі x з'являлися внизу
  ax.xaxis.set_label_position("bottom")
  ax.xaxis.tick_bottom()

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

  # Будуємо 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 = model(test_data)

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