TF & KERAS. РОБОТА із ЗОБРАЖЕННЯМИ

Файл: TF_KERAS_Image_03_001

Створення простої нейронної мережі (MLP) з використанням набору даних MNIST (або fashion MNIST)

[MNIST]

Вбудована база данных зображень рукописних цифр

Вбудована база данных зображень одягу (fashon)

Імпорт необхідних бібліотек

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

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

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

y_train = y_train.astype(np.int32)
y_test = y_test.astype(np.int32)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
•[1m11490434/11490434•[0m •[32m----------------------•[0m•[37m•[0m •[1m1s•[0m
0us/step
```

```
y_train[0]
```

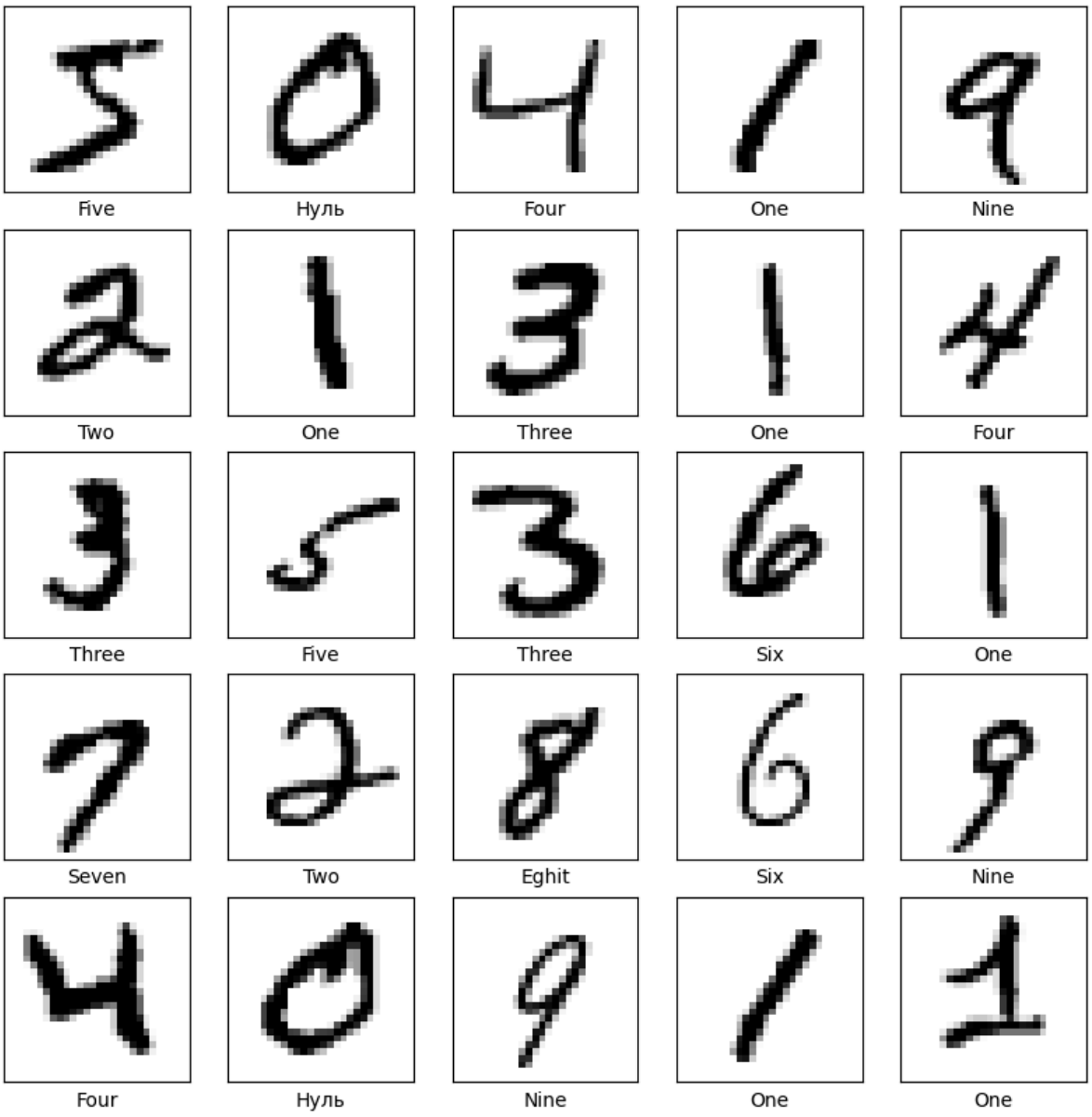
```
5
```

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']
```

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



Нормалізування значення пікселів до діапазону [0, 1]

```
x_train = x_train.astype(np.float32) / 255.0
x_test = x_test.astype(np.float32) / 255.0
```

Визначення гіперпараметрів нейронної мережі

```
n_inputs = 28 * 28 # Кількість вхідних нейронів (28x28 пікселів)
n_hidden1 = 256    # Кількість нейронів у першому прихованому шарі
n_hidden2 = 64    # Кількість нейронів у другому прихованому шарі
n_outputs = 10     # Кількість вихідних нейронів (10 класів)
n_epochs = 10      # Кількість епох
batch_size = 32    # Розмір батчу
n_batches = len(x_train) // batch_size # Кількість батчів
```

Ініціалізація ваг та зміщення для кожного шару

```
weights1 = tf.Variable(tf.compat.v1.initializers.variance_scaling()(shape=
(n_inputs, n_hidden1)), dtype=tf.float32)
biases1 = tf.Variable(tf.zeros(shape=(n_hidden1)), dtype=tf.float32)
weights2 = tf.Variable(tf.compat.v1.initializers.variance_scaling()(shape=
(n_hidden1, n_hidden2)), dtype=tf.float32)
biases2 = tf.Variable(tf.zeros(shape=(n_hidden2)), dtype=tf.float32)
weights3 = tf.Variable(tf.compat.v1.initializers.variance_scaling()(shape=
(n_hidden2, n_outputs)), dtype=tf.float32)
biases3 = tf.Variable(tf.zeros(shape=(n_outputs)), dtype=tf.float32)
```

Визначення моделі нейронної мережі

```
@tf.function
def neural_net(X):
    # Вирівняння вхідного зображення
    X = tf.reshape(X, shape=(-1, n_inputs))
    # Обчислення виходу першого прихованого шару
    hidden1 = tf.nn.relu(tf.matmul(X, weights1) + biases1)
    # Обчислення виходу другого прихованого шару
    hidden2 = tf.nn.relu(tf.matmul(hidden1, weights2) + biases2)
    # Обчислюємо вихід вихідного шару
    logits = tf.matmul(hidden2, weights3) + biases3
    return logits
```

Визначення функції втрат (cross-entropy)

```
@tf.function
def loss_fn(logits, labels):
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=labels,
logits=logits)
    return tf.reduce_mean(cross_entropy)
```

Визначення оптимізатора

```
optimizer = tf.optimizers.Adam()
```

Визначення функції навчання

```
@tf.function
def train_step(X, y):
    with tf.GradientTape() as tape:
        # Визначення logits для поточного батча
        logits = neural_net(X)
        # Розрахунок помилки
        loss = loss_fn(logits, y)
    # Обчислення градієнтів втрат з урахуванням ваг та зсувів
    grads = tape.gradient(loss, [weights1, biases1, weights2, biases2, weights3,
biases3])
    # Оновлення ваг та зміщення за допомогою оптимізатора
    optimizer.apply_gradients(zip(grads, [weights1, biases1, weights2, biases2,
weights3, biases3]))
    return loss
```

Тренування нейронної мережі

```
for epoch in range(n_epochs):
    for batch in range(n_batches):
        # Вибір випадкової групи зображень та labels
        X_batch = x_train[batch*batch_size:(batch+1)*batch_size]
        y_batch = y_train[batch*batch_size:(batch+1)*batch_size]
        # Навчання моделі на поточному батчі
        loss = train_step(X_batch, y_batch)
    # Оцінка моделі на тестовому наборі даних
    logits = neural_net(x_test)
    predictions = tf.argmax(logits, axis=1, output_type=tf.int32)
    accuracy = tf.reduce_mean(tf.cast(tf.equal(predictions, y_test), tf.float32))
    print("Epoch {}/{} - loss: {:.4f} - accuracy: {:.4f}".format(epoch+1,
n_epochs, loss, accuracy))
```

```
Epoch 1/10 - loss: 0.0490 - accuracy: 0.9557
Epoch 2/10 - loss: 0.0641 - accuracy: 0.9691
Epoch 3/10 - loss: 0.0208 - accuracy: 0.9720
Epoch 4/10 - loss: 0.0025 - accuracy: 0.9769
Epoch 5/10 - loss: 0.0019 - accuracy: 0.9738
Epoch 6/10 - loss: 0.0005 - accuracy: 0.9789
Epoch 7/10 - loss: 0.0013 - accuracy: 0.9734
Epoch 8/10 - loss: 0.0004 - accuracy: 0.9756
Epoch 9/10 - loss: 0.0002 - accuracy: 0.9672
Epoch 10/10 - loss: 0.0008 - accuracy: 0.9766
```

Виведення остаточних ваг та зміщення моделі (не обов'язково)

```
# print("Weights of layer 1:\n", weights1)
# print("Biases of layer 1:\n", biases1)
# print("Weights of layer 2:\n", weights2)
# print("Biases of layer 2:\n", biases2)
print("Weights of layer 3:\n", weights3)
# print("Biases of layer 3:\n", biases3)
```

```
Weights of layer 3:
 <tf.Variable 'Variable:0' shape=(64, 10) dtype=float32, numpy=
array([[ 0.02538086, -0.3870372 , -0.27484465,  0.28272575, -0.59118515,
         0.3525255 ,  0.18905467, -0.38679397,  0.11948424,  0.01913715],
       [-0.16285905, -0.0189609 , -0.06563882,  0.30048686, -0.06226109,
        -0.00079243, -0.5100669 , -0.00929168,  0.00195026,  0.04419762],
       [ 0.07921629, -0.12922414, -0.02042313, -0.3686254 ,  0.27851492,
        -0.04895547,  0.28949386, -0.23570983, -0.4184421 ,  0.18538521],
       [-0.05558435,  0.08827802,  0.04662464, -0.12437236, -0.08475275,
        -0.25525212, -0.21195902, -0.1647973 , -0.32253712,  0.03891188],
       [-0.24172346,  0.33653244, -0.4772238 ,  0.01351405, -0.09096777,
         0.3359343 , -0.41561958, -0.08178926, -0.01050005, -0.07430203],
       [-0.4620073 ,  0.24973395, -0.15176052, -0.16927902, -0.24208257,
         0.48717058, -0.18655996, -0.249866  , -0.30621126, -0.08875612],
       [ 0.21794088,  0.33052468, -0.21440646, -0.7757801 ,  0.07027098,
        -0.3082369 , -0.24692173,  0.03484223, -0.09964778, -0.12333496],
       [-0.3711217 ,  0.36855078,  0.0916075 , -0.20683268,  0.0283139 ,
         0.23213129, -0.1108862 ,  0.27069277, -0.19946587, -0.26642856],
       [-0.00590906,  0.05544207, -0.03155496, -0.18677507, -0.03279214,
        -0.00499272,  0.43765357, -0.49312848,  0.00307865, -0.18101273],
       [ 0.21520364, -0.04557542, -0.37489164, -0.39983433,  0.10830559,
        -0.37852424,  0.01321649,  0.14609256, -0.0912885 ,  0.03846175],
       [-0.39794213, -0.12169448, -0.3799234 , -0.05136108,  0.20475802,
        -0.6184796 ,  0.1294753 ,  0.39213896, -0.08224169,  0.3066275 ],
       [ 0.3210874 , -0.51103926,  0.2018095 , -0.51473695, -0.07205856,
        -0.38315824, -0.22445378, -0.26225153,  0.05894298,  0.13077042],
       [ 0.02104329,  0.2135649 ,  0.08896656,  0.09770665, -0.03158835,
        -0.20786496,  0.2334741 , -0.09687078,  0.14869343, -0.5804225 ],
       [ 0.26273316, -0.32648182,  0.38897404, -0.05722869, -0.28322336,
        -0.31265292, -0.15178724, -0.0701827 ,  0.3299272 ,  0.01884578],
       [ 0.00781714, -0.085851  , -0.11851903, -0.08862623,  0.02764354,
        -0.08448804,  0.03796908,  0.10601319, -0.18745802,  0.06897281],
       [-0.16940685,  0.421233  ,  0.24511924, -0.07403769, -0.18654205,
         0.3083529 ,  0.01897901, -0.08421364, -0.27193862, -0.2198607 ],
       [ 0.15623783,  0.09385733, -0.02565708, -0.52100176,  0.08740838,
        -0.2416919 , -0.01935178, -0.34069481, -0.01035196, -0.27588764],
       [ 0.16602992, -0.25964537,  0.05188641, -0.22148223,  0.22184701,
        -0.05304268, -0.24192536, -0.18189092, -0.12884636,  0.03759852],
       [ 0.03747614, -0.01591522,  0.04326222,  0.06506875, -0.44963506,
         0.22396338,  0.16850531, -0.38307208,  0.14571649, -0.1851827 ],
       [-0.41879836, -0.12749161,  0.00888029,  0.14057548,  0.07433573,
         0.20040324, -0.17351425,  0.08571186,  0.00473336, -0.41730708],
       [ 0.30045092, -0.29780847, -0.17588113,  0.02624344, -0.37488234,
```

```
  0.16510324,  0.11889464,  0.10174973,  0.09404342,  0.06152857],
[-0.2504567 , -0.15839264, -0.23276442,  0.17874397,  0.03410573,
  0.08730368, -0.51531667, -0.29201758, -0.05538187,  0.08126342],
[-0.06708298, -0.28380948, -0.01218134,  0.11431748,  0.05660716,
  0.17265156,  0.03735461,  0.23148839, -0.06068728, -0.02099478],
[-0.10926887, -0.09318828,  0.05543288,  0.29760998,  0.26593345,
  0.05264048, -0.409001  ,  0.16841756, -0.05266045, -0.30198568],
[ 0.31301543, -0.00201786,  0.42254484, -0.20620371, -0.13161668,
  0.49953923, -0.03841307, -0.23334755, -0.28741625, -0.49036807],
[-0.23279965, -0.02936219,  0.14840652,  0.13501444, -0.27200896,
 -0.28573534,  0.12066971,  0.1694753 ,  0.14520386, -0.71624506],
[-0.2994354 , -0.1152624 ,  0.2206532 ,  0.17587097, -0.1153072 ,
 -0.40598983, -0.21938998,  0.3283239 , -0.2451817 ,  0.01364075],
[-0.30977252,  0.29851496, -0.2820221 , -0.39515644,  0.43985865,
 -0.04882917,  0.15023012, -0.29899102, -0.03915768,  0.04712303],
[-0.21956211, -0.05949951, -0.08725773,  0.02970232, -0.02195176,
  0.01938933, -0.48544312, -0.15017383,  0.48372868,  0.13080068],
[-0.4754644 , -0.17738485,  0.23570003, -0.13321823,  0.15102905,
  0.09649929, -0.24131325,  0.31319904, -0.7007969 ,  0.20999025],
[-0.2635476 , -0.3443643 , -0.00174015, -0.23477179,  0.09653432,
  0.17239408, -0.11108696,  0.19136044,  0.13993025, -0.10992362],
[-0.30555025,  0.13183725, -0.32245818,  0.14924686, -0.07639816,
  0.08800802,  0.00464615, -0.30109048,  0.18914539,  0.08024232],
[ 0.05181944, -0.40860802, -0.14675325, -0.131835  , -0.5500343 ,
  0.12188559,  0.25306374, -0.31334433,  0.01032229,  0.15026596],
[ 0.06947041,  0.00665463,  0.14567116,  0.00294146, -0.57847273,
 -0.3485153 , -0.39854658,  0.05695349,  0.04845081,  0.17389338],
[-0.15280384, -0.0355292 ,  0.10103316, -0.05934033, -0.06058548,
 -0.09954613, -0.06835191, -0.04522338,  0.08928202, -0.25446516],
[ 0.01195681, -0.13565584,  0.33328262,  0.1200643 , -0.04799574,
 -0.1153515 , -0.10119721, -0.25762096, -0.4077774 ,  0.04651308],
[ 0.2713013 , -0.41794327, -0.34437293,  0.19820188,  0.01799292,
 -0.4611283 ,  0.34824717, -0.53200394, -0.35230005,  0.20937961],
[ 0.49974456, -0.21839377, -0.12699485, -0.15503225, -0.1839252 ,
  0.18303704,  0.02562047,  0.16688256, -0.448458  , -0.16076559],
[-0.35506105,  0.3135149 ,  0.3227605 , -0.44542572, -0.12841268,
 -0.01396444,  0.3256075 , -0.4006888 , -0.15214781, -0.2229156 ],
[ 0.14550011, -0.4890034 , -0.41769108, -0.14995322, -0.07398363,
  0.37140623,  0.01735474,  0.20861448,  0.11090719,  0.10294583],
[-0.01198592, -0.37752545,  0.09462616, -0.1732268 , -0.463428  ,
  0.16514449, -0.5201868 ,  0.23043157,  0.08081083,  0.2272479 ],
[-0.18590814, -0.32561454, -0.2765209 ,  0.07852453,  0.12873308,
 -0.22208345, -0.3824703 , -0.35128745,  0.24387188,  0.24557702],
[-0.6939837 , -0.09564773, -0.01145206,  0.17860952, -0.07093691,
  0.1802089 ,  0.074704  , -0.26847807,  0.11374757,  0.15203618],
[-0.14005782,  0.07316439, -0.13570525, -0.34629506,  0.3405324 ,
 -0.29480714, -0.10265061,  0.09497361,  0.29355836,  0.05618447],
[ 0.11040208, -0.12769628, -0.09027414, -0.29169425,  0.18198831,
  0.14713031,  0.36929688, -0.24871734, -0.314386  , -0.11401875],
[-0.07669695,  0.36065048, -0.614946  ,  0.24119285, -0.01269707,
  0.10443467, -0.25528392,  0.17073727, -0.3313338 , -0.07621253],
[ 0.24835175, -0.14845996,  0.13050856,  0.11956771,  0.10475033,
 -0.38946766, -0.54229015,  0.02187714, -0.24252063,  0.24094348],
[-0.00226099, -0.38192555,  0.20796052, -0.2062102 , -0.14739999,
 -0.22047809,  0.25849792,  0.22576538, -0.03330587,  0.02002374],
[-0.13594891,  0.10287376,  0.18335646,  0.10626531, -0.23521636,
```

```
         0.02819986, -0.2242717 , -0.32154238,  0.20063135, -0.522412  ],
       [-0.42389232, -0.51899344, -0.23442893,  0.20880309,  0.08972305,
        -0.05473632, -0.42601025, -0.02358953, -0.03087702,  0.08421383],
       [-0.36896175,  0.08540787, -0.02941645, -0.16380931, -0.1677457 ,
        -0.15062225, -0.18990196,  0.169508  ,  0.14655861, -0.33601612],
       [ 0.12837057,  0.17311433,  0.09145027,  0.19623192, -0.07930063,
        -0.31512007,  0.0591361 , -0.08312506, -0.16458741, -0.3030473 ],
       [-0.25401795,  0.1767948 , -0.21446681, -0.16404271,  0.02414917,
         0.21645163,  0.11724631, -0.37711552,  0.37277022,  0.11598916],
       [ 0.05531086, -0.38620684,  0.22438811,  0.03970169,  0.36998054,
         0.07587218, -0.28769132, -0.5728786 , -0.2314096 ,  0.25451675],
       [ 0.33629438,  0.32736158,  0.04918087, -0.26859805, -0.4492767 ,
        -0.3312768 ,  0.11668689,  0.11732461, -0.11654183, -0.56614953],
       [-0.34286317, -0.10497176, -0.20514032,  0.04676041,  0.26067385,
         0.07221457,  0.03078266, -0.20208973,  0.27690014,  0.01376585],
       [ 0.04330146, -0.4421283 ,  0.04992338,  0.1373343 ,  0.3201741 ,
        -0.02795971,  0.06297982,  0.07668146, -0.19286202, -0.49560654],
       [-0.26556537, -0.51893216, -0.30437347, -0.1586592 , -0.1276    ,
        -0.05112486,  0.2189792 ,  0.15148222, -0.263661  ,  0.27034256],
       [-0.12438679, -0.31759915, -0.01945985, -0.5137253 ,  0.17196104,
         0.20169114,  0.07350325, -0.32785025,  0.18468638,  0.21387015],
       [-0.01092746,  0.01589375, -0.5904527 ,  0.07620912, -0.00215037,
        -0.13656396, -0.23566854, -0.19687092,  0.27172044,  0.33870202],
       [ 0.08398019, -0.27145386, -0.07460296, -0.3719656 ,  0.14783446,
         0.12346844,  0.1507765 ,  0.383111  , -0.40151066, -0.09048247],
       [ 0.26701918,  0.10503407, -0.3193323 , -0.15040378,  0.30935854,
        -0.1813772 ,  0.20093027, -0.27158663,  0.23025748,  0.04487828],
       [-0.2646265 ,  0.18212636, -0.13054906,  0.18966381, -0.19763677,
         0.336325  , -0.08553725, -0.18410097, -0.5693924 ,  0.30099884],
       [-0.20641236,  0.10942201, -0.00293798,  0.02802257,  0.05666351,
        -0.46468076, -0.4650558 ,  0.20593856,  0.00634399,  0.05443659]],
      dtype=float32)>
```

Функція для прогнозування зображень

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

Назви класів

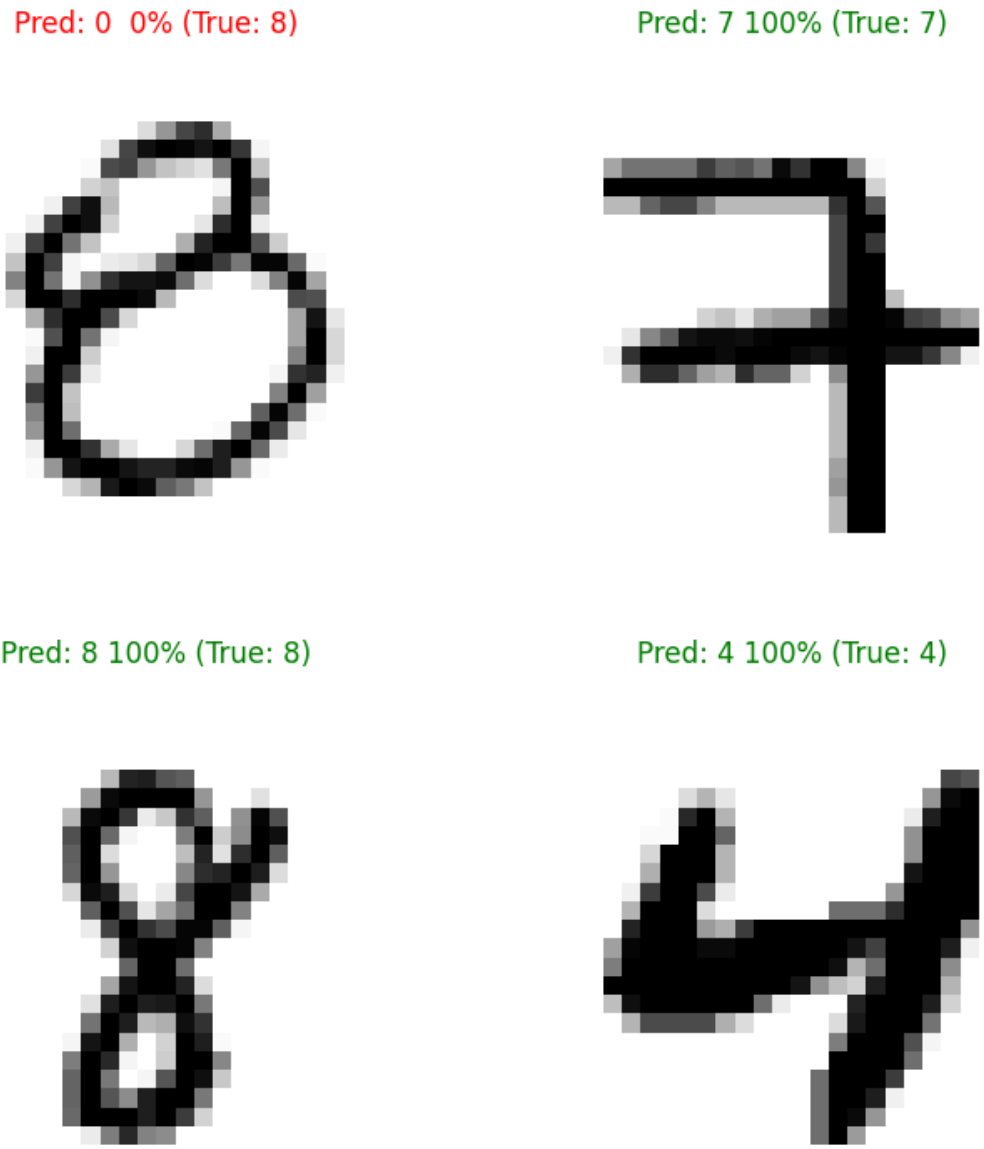
```
# class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat','Sandal',
'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
class_names = ['0', '1', '2', '3', '4',
               '5', '6', '7', '8', '9']
```

Створення функції для побудови випадкового зображення разом з його прогнозом

```
accuracy_metric = tf.metrics.SparseCategoricalAccuracy()
def plot_random_image(images, true_labels, classes):
  plt.figure(figsize=(9, 9))
  random_number = random.randint(1000, 2000)
  for i in range(4):
    ax = plt.subplot(2, 2, i + 1)
    target_image = images[i+random_number]
    y_pred = predict(target_image.reshape(28, 28, 1))
    pred_label = classes[y_pred.numpy()[0]]
    true_label = classes[true_labels[i+random_number]]
    accuracy = accuracy_metric(y_pred, y_test)
    if(accuracy*1000 > 100):
      accuracy=100
    else:
      accuracy*=1000
    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(pred_label, accuracy,
true_label), color=color)
    plt.axis(False)
```

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

```
plot_random_image(x_test, y_test, class_names)
```



Методи оцінки класифікації

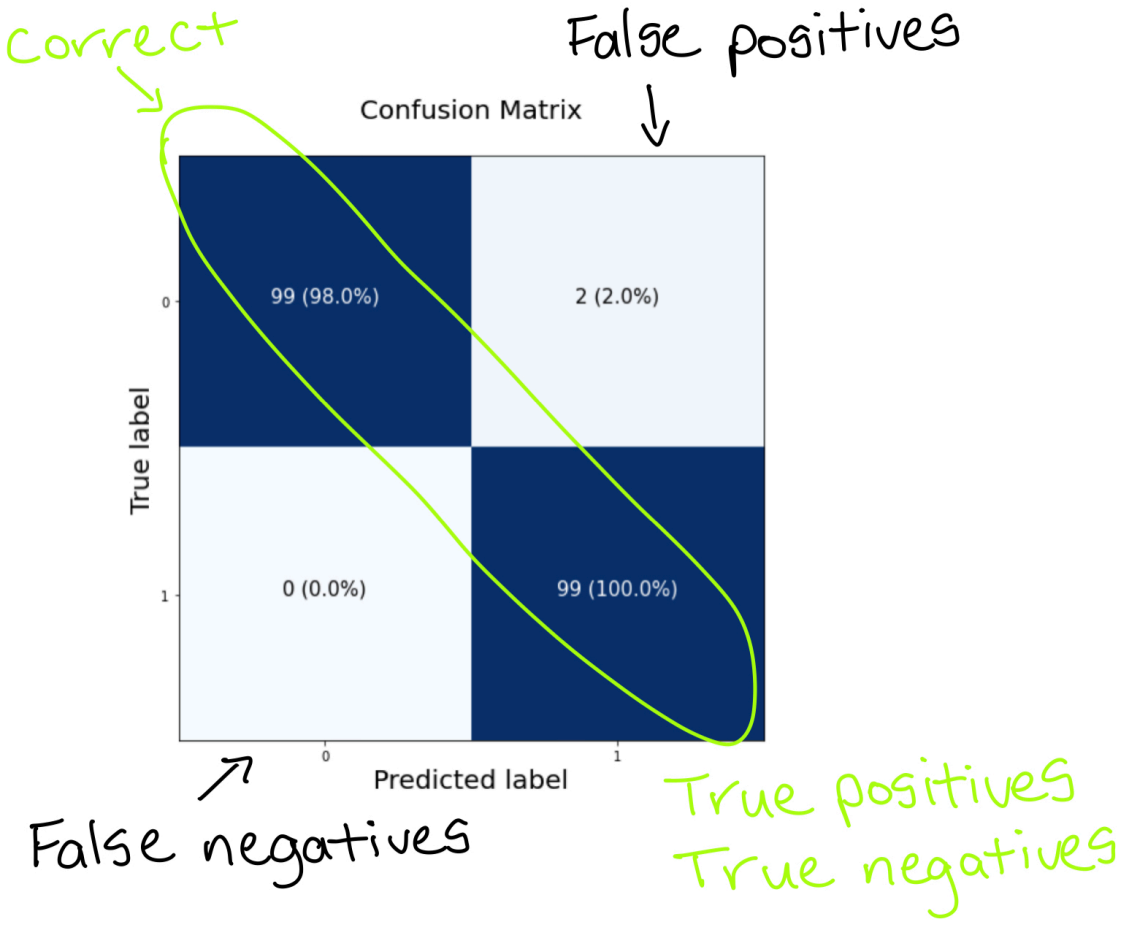
Існує ряд різних оціночних метрик, які ми можемо використовувати для оцінки наших моделей класифікації. Але для почтаку розглянемо саме Confusion Matrix.

Confusion Matrix - це таблиця, яка представляє собою підсумок результатів прогнозування для задачі класифікації. Нижче ми бачимо матрицю помилок. 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.

Для того, щоб повністю зрозуміти confusion matrix, спочатку потрібно ознайомитися з наступними термінами:

1. ***True Positive*** (**TP**-Істинно позитивний) означає, що зразок, який належить до позитивного класу, класифіковано правильно.
2. ***True Negative*** (**TN**-Вірогідно негативний) - це зразок, що належить до негативного класу, який класифіковано правильно.
3. ***False Positive*** (**FP**-Хибнопозитивний) - зразок, що належить до негативного класу, але помилково класифікований як такий, що належить до позитивного класу.
4. ***False Negative*** (**FN**-Хибнонегативний) - зразок, що належить до позитивного класу, але помилково класифікований як такий, що належить до негативного класу.

Розглянемо приклад наведений нижче, тут ми маємо анатомію confusion matrix. Правильні прогнози з'являються по першій діагоналі (зверху вниз зліва направо). Неправильні з'являються по другій діагоналі (зверху вниз справа наліво). По осі y ми маємо "true label" - тобто істинну мітку (або правильну), а по осі x ми маємо "predictid label" - тобто прогнозовані мітки (нейронною мережею).



Ми можемо створити **confusion matrix** за допомогою методу confusion_matrix від Scikit-Learn.

Оцінка ефективності

З **confusion matrix** ми також можемо обчислити **accuracy, precision** та **F1 score**. Ці показники допомагають зрозуміти продуктивність моделі. Ці показники можна обчислити за нижче неаведеними формулами.

Accuracy. Кількість правильно класифікованих зразків з усіх зразків, присутніх у тестовому наборі.

$$Accuracy = (TP + TN)/(TP + TN + FP + FN)$$

Precision (для позитивного класу). Кількість зразків, які дійсно належать до позитивного класу з усіх зразків, що були передбачені моделлю як такі, що належать до позитивного класу.

$$Precision = TP/(TP + FP)$$

F1-score (для позитивного класу). Середнє гармонійне значення оцінок точності та пригадування, отриманих для позитивного класу.

$$F1score = 2 * (precision * recall)/(precision + recall)$$

де **recall** - це частка істинних позитивних прогнозів від усіх фактичних позитивних прикладів, і розраховується як:

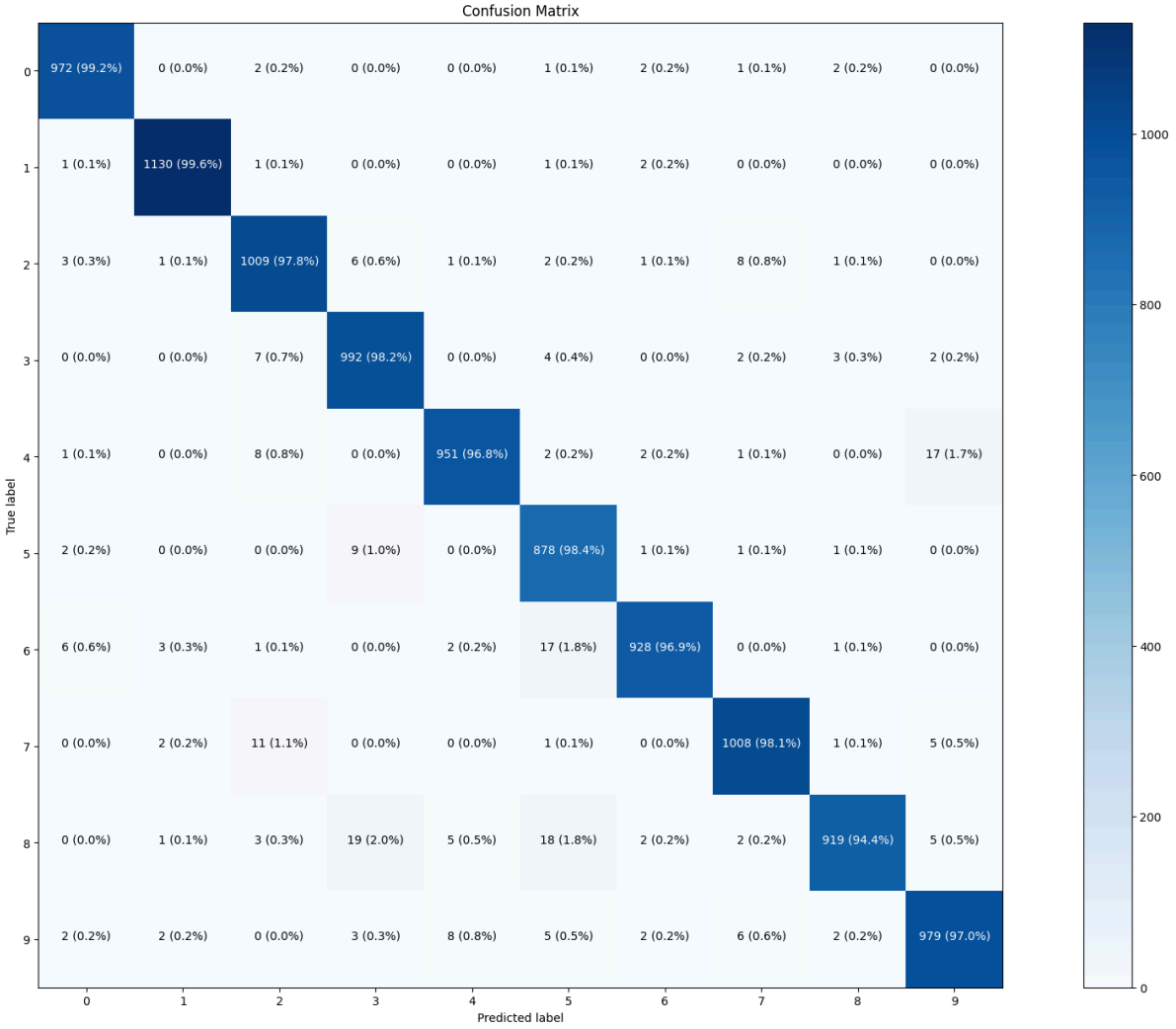
$$Recall = TP/(TP + FN)$$

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()
```

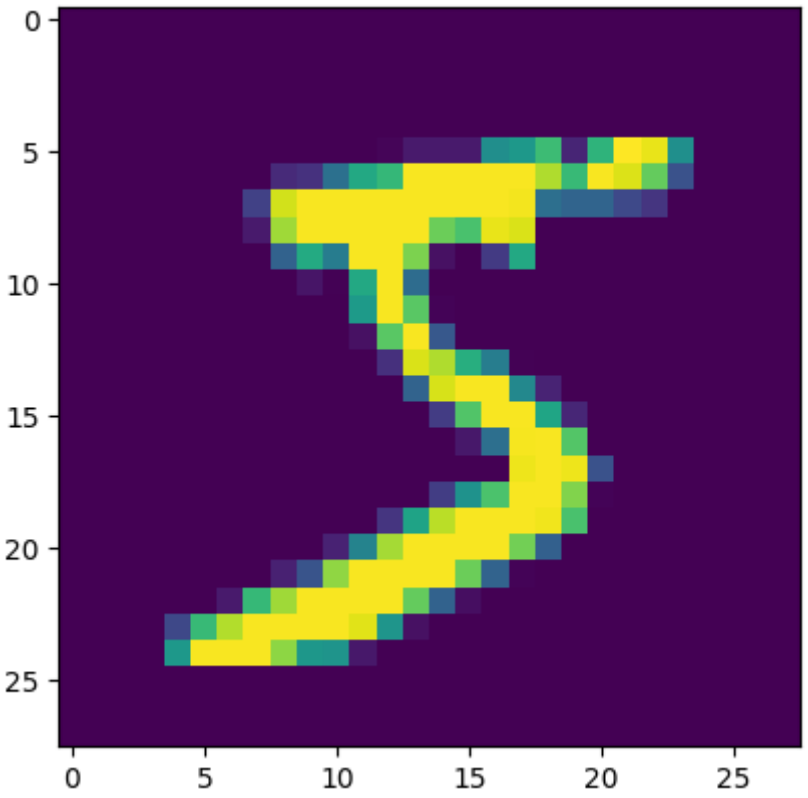
```
  # Встановіть поріг для різних кольорів
  threshold = (cm.max() + cm.min()) / 2.

  # 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 = neural_net(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)
```




```
# https://scikit-image.org/
import skimage as ski
```

```
plt.imshow(x_train[0])
plt.show()
```




```
print(type(x_train[1]), x_train[1].shape)
predict(x_train[1])
```

```
<class 'numpy.ndarray'> (28, 28)
```

```
<tf.Tensor: shape=(1,), dtype=int32, numpy=array([0], dtype=int32)>
```

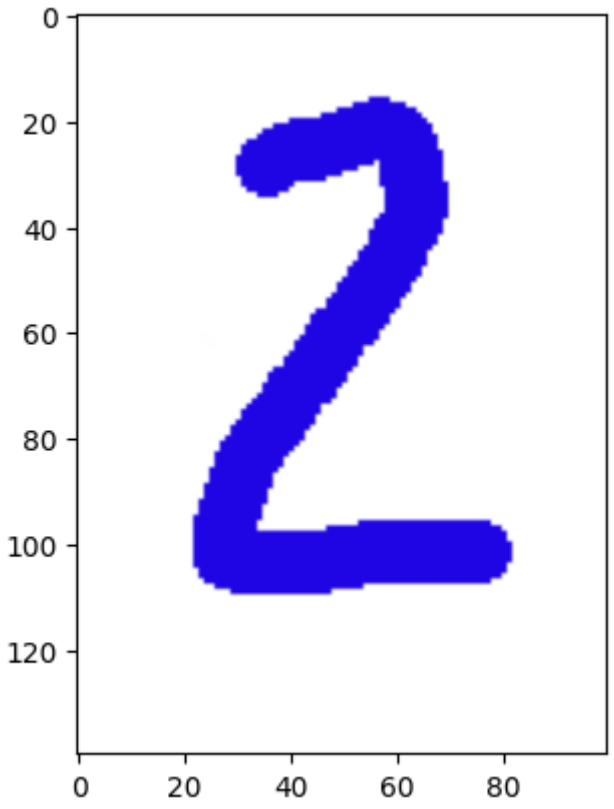
```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Mounted at /content/gdrive
```

```
path = '/content/gdrive/MyDrive/Colab_Notebooks/VIP_2024_LEC/_Digits_/'
image_name = '02_TWO_01.png'
# image_name = '01_ONE_01.png'
image_path = path + image_name
print(image_path)
```

```
/content/gdrive/MyDrive/Colab_Notebooks/VIP_2024_LEC/_Digits_/02_TWO_01.png
```

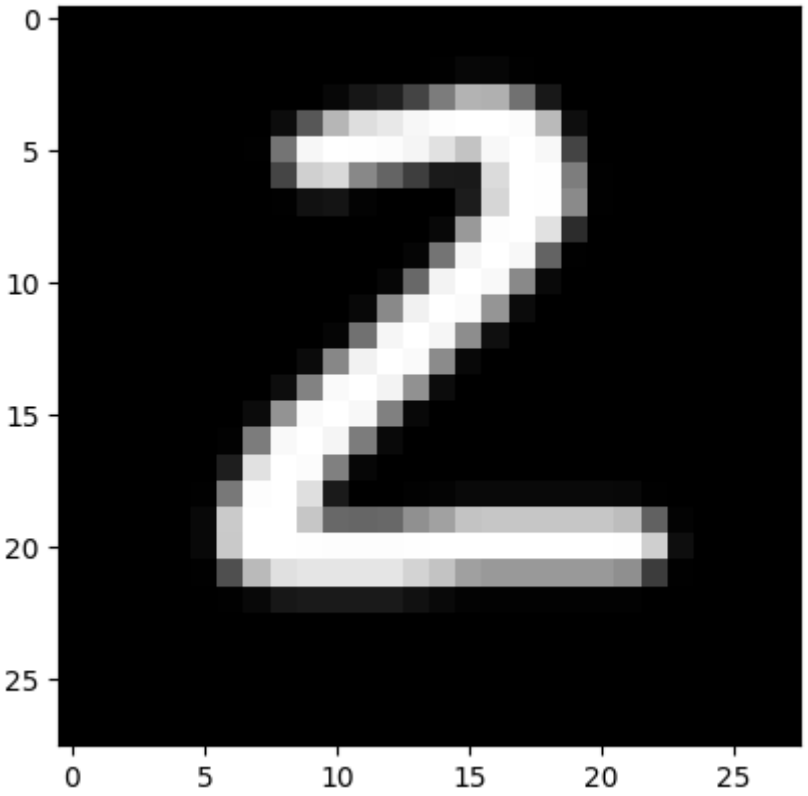
```
# Просто вывод этого же изображения в формате 224 на 224
img = ski.io.imread(image_path)
plt.imshow(img)
plt.show()
```




```
img_grayscale = ski.color.rgb2gray(img)
img_pred =  ski.transform.resize(img_grayscale, (28, 28))
```

```
img_pred_ = np.float32(1.0-img_pred)
print(type(img_pred_), img_pred_.shape)
plt.imshow(img_pred_, cmap='gray')
plt.show()
```

```
<class 'numpy.ndarray'> (28, 28)
```




```
predict(img_pred_)
```

```
<tf.Tensor: shape=(1,), dtype=int32, numpy=array([2], dtype=int32)>
```

```

```