Практична робота №5

Класифікація зображень засобами нейронних мереж

Мета роботи: набути практичних навичок розробки, навчання та оптимізації згорткових нейронних мереж (CNN) для класифікації зображень. Опанувати повний інженерний цикл: від завантаження та аналізу даних до навчання, валідації, тестування на незалежних даних та збереження моделі у промислових форматах.

Стек технологій:

TensorFlow 2.x - *основна бібліотека глибокого навчання, tensorflow.org*

Keras Sequential API - *побудова моделей шар за шаром, keras.io/models/sequential/*

Google Colaboratory - *хмарне середовище з безкоштовним GPU/TPU*

Scikit-learn - *метрики якості: confusion matrix, report, scikit-learn.org*

Seaborn - *візуалізація матриці плутанини*

TF Hub - *попередньо навчені моделі (Transfer Learning), tfhub.dev*

Теоретичний матеріал

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) - клас нейронних мереж, оптимізованих для обробки даних із просторовою структурою (зображення, відео, часові ряди). На відміну від багатошарового перцептрону (MLP), CNN автоматично виокремлюють просторові ознаки через операцію дискретної згортки.

Основна математична операція - дискретна 2D-згортка:

$$(I * K)[i, j] = \Sigma_m \Sigma_n I[i+m, j+n] \cdot K[m, n]$$

де I - вхідна карта ознак (feature map), K - фільтр (ядро), m, n — індекси ядра. Застосування кількох фільтрів до одного шару дозволяє виявляти різні просторові ознаки: краї, текстури, форми.

Вихідний шар класифікатора використовує функцію Softmax для перетворення логітів у ймовірнісний розподіл:

$$P(y = k \mid x) = \exp(z_k) / \Sigma_j \exp(z_j), \quad k \in \{0, ..., 9\}$$

де z_k — логіт k-го класу. Для навчання використовується функція втрат Sparse Categorical Cross-Entropy:

$$L = -(1/N)\,\Sigma_i \log P(y = y_i \mid x_i)$$

Датасет MNIST

MNIST (Modified National Institute of Standards and Technology) — еталонний датасет для класифікації рукописних цифр. Складається з 70 000 зображень розміром 28×28 пікселів у відтінках сірого. Кожне значення пікселя знаходиться в діапазоні 0 (білий) — 255 (чорний).

Таблиця 1

Характеристики датасету MNIST

Характеристика	**Значення**
Навчальна вибірка	60 000 зображень (x_train, y_train)
Тестова вибірка	10 000 зображень (x_test, y_test)
Розмір зображення	28 × 28 пікселів, 1 канал (grayscale)
Діапазон значень	uint8: 0–255; float32 після нормалізації
Кількість класів	10 (цифри 0–9), рівномірний розподіл
Типовий розмір класу	~6 000 прикладів на клас у train
Формат зберігання	IDX binary format, стиснутий (.gz)

Зміст роботи

Завдання 1. *Базова класифікація.*

Реалізуйте повний пайплайн CNN для класифікації MNIST.

1. Завантажте датасет MNIST, виведіть форми вибірок та статистику пікселів.
2. Реалізуйте функцію *preprocess_image* з *per-image standardization*.
3. Сформуйте датасети *tf.data: train (80%), validation (20%)* з *shuffle* та *prefetch*.
4. Побудуйте та навчіть Модель 1 (базова CNN, навчання - 5 епох). Виведіть *model.summary()*.
5. Побудуйте графіки *loss/accuracy*. Зафіксуйте ознаки *overfitting* у звіті, якщо такі є.
6. Додайте Dropout (Модель 2, навчання - 10 епох). Порівняйте графіки з Моделлю 1.

7. Реалізуйте Модель 3 (глибока CNN з BatchNorm, навчання 20+ епох з *callbacks*).

8. Розрахуйте точність на тестовій вибірці. Побудуйте матрицю плутанини.

9. Визначте топ-3 класових пар, які модель плутає найчастіше. Поясніть причини.

Завдання 2. *Оптимізація та збереження*

Проведіть систематичні експерименти для досягнення точності ≥ 99.0% та збережіть модель.

1. Реалізуйте *EarlyStopping* та *ReduceLROnPlateau callbacks*.

2. Порівняйте *learning rate*: 0.001, 0.0005, 0.0001 - зафіксуйте вплив на збіжність.

3. Порівняйте batch size: 32, 64, 128 - зафіксуйте вплив на точність і час епохи.

4. Додайте аугментацію (*RandomRotation*, *RandomZoom*). Зробіть висновок про ефект.

5. Збережіть найкращу модель у форматах *.keras* та *.tflite*.

6. Порівняйте розміри файлів: *.keras* та *.tflite*. Розрахуйте коефіцієнт стиснення.

7. Сформуйте порівняльну таблицю усіх архітектур: параметри, точність, час навчання.

Завдання 3. *Тестування на незалежних даних*

Протестуйте модель на зображеннях, що не входять до MNIST.

1. Намалюйте від руки (у графічному редакторі) щонайменше 20 цифр (2 екземпляри кожної).

2. Реалізуйте *preprocess_custom_image* з автоматичним визначенням та корекцією інверсії.

3. Запустіть *batch_evaluate_custom* та розрахуйте точність на власних даних.

4. Для кожної помилки проаналізуйте причину (чи стиль написання, розмір, яскравість).

5. Порівняйте точність на *MNIST*-тесті та на власних зображеннях. Поясніть різницю.

Завдання 4. *Розширене (для оцінки «відмінно»)*

Виберіть одне із трьох завдань:

1. *Transfer Learning*. Завантажте *MobileNetV2* з *TF Hub*, адаптуйте для *MNIST* (28×28×1 → 96×96×3 через *resize*). Порівняйте з моделлю з нуля за точністю та часом навчання.

2. Навчіть 3 різні CNN-архітектури, реалізуйте *Voting Ensemble*. Чи перевищує ансамбль найкращу одиночну модель?

3. Реалізуйте *Gradient-weighted Class Activation Mapping* для 10 прикладів кожного класу. Визначте, яку частину цифри модель використовує для класифікації.

Методичні рекомендації

Етапи роботи:

1. Завантаження та первинна перевірка:

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

В бібліотеці Keras реалізовані методи для завантаження популярних наборів даних для алгоритмів машинного навчання *mnist.load_data()* - повертає два набора даних для навчання і для тестів:

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

Перевірка форм та типів

```
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test  shape:', x_test.shape)
```

```
print('Pixel range  :', x_train.min(), '–', x_train.max())
print('Dtype        :', x_train.dtype)
```

2. ***Проводимо візуалізацію завантаженого набору даних.*** Перш ніж будувати модель, важливо дослідити дані: переглянути приклади та перевірити баланс класів. Нерівномірний розподіл (class imbalance) потребує додаткових технік.

```
def visualize_dataset_item(x, y):
  print("Цифра:", y)
  plt.imshow(x, cmap=plt.cm.gray)
  plt.axis('off')

item_index = 0
visualize_dataset_item(x_train[item_index], y_train[item_index])
```

Далі потрібно провести аналіз розподілу класів за допомогою гістограми:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.hist(y_train, bins=10, color='steelblue', edgecolor='black')
ax1.set_title('Train: розподіл по класах')
ax2.hist(y_test,  bins=10, color='coral',     edgecolor='black')
ax2.set_title('Test:  розподіл по класах')
for ax in (ax1, ax2):
    ax.set_xlabel('Клас (цифра 0–9)')
    ax.set_ylabel('Кількість прикладів')
plt.tight_layout(); plt.show()
```

3. Передобробка та формування пайплайну tf.data

Якісна передобробка (preprocessing) безпосередньо впливає на швидкість навчання та фінальну точність. Per-image standardization гарантує нульове середнє та одиничне стандартне відхилення для кожного зображення незалежно:

$$x_{norm} = (x - \mu_i) / \sigma_i,$$

де μ_i та σ_i - середнє та стандартне відхилення конкретного зображення (не всього датасету). Це прискорює збіжність у порівнянні зі звичайним діленням на 255.

Проводимо стандартизацію зображень:

```
def preprocess_image(image):
  image = tf.reshape(image, [28, 28, 1])
```

```
  image = tf.image.per_image_standardization(image)

  return image
```

3.1 Пайплайн tf.data з розбивкою 80/20

tf.data реалізує ефективний пайплайн із паралельним завантаженням, перемішуванням та попередньою буферизацією. Параметр prefetch(tf.data.AUTOTUNE) дозволяє CPU готувати наступний батч поки GPU обробляє поточний:

```
total_count = len(y_train)

images_dataset = tf.data.Dataset.from_tensor_slices(x_train)
images_dataset = images_dataset.map(preprocess_image,
                                    num_parallel_calls=tf.data.AUTOTUNE)

labels_dataset = tf.data.Dataset.from_tensor_slices(y_train)
dataset = tf.data.Dataset.zip((images_dataset, labels_dataset))

# Перемішування перед розбивкою (seed=42 для відтворюваності)
dataset = dataset.shuffle(buffer_size=10000, seed=42,
                          reshuffle_each_iteration=True)

# Розбивка 80% train / 20% validation
train_size = round(total_count * 0.8)
train_dataset = dataset.take(train_size).batch(128, drop_remainder=False)
train_dataset = train_dataset.prefetch(tf.data.AUTOTUNE)

val_dataset   = dataset.skip(train_size).batch(256, drop_remainder=False)
val_dataset   = val_dataset.prefetch(tf.data.AUTOTUNE)

print(f'Train: {train_size} зображень ({train_size//128+1} батчів по
128)')
print(f'Val  : {total_count - train_size} зображень')
```

3.2. Аугментація даних (для поглибленого виконання)

Аугментація штучно збільшує різноманітність навчальних даних через випадкові трансформації. Застосовується ЛИШЕ до навчальної вибірки. Для MNIST доцільні: невеликі повороти (±10°), масштабування (±10%) та зміщення (±10%). Аугментація є особливо важливою при малих датасетах.

```
# Шар аугментації (застосовується лише під час training=True)
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomRotation(0.1),          # ±10% обертання
    tf.keras.layers.RandomZoom(0.1),               # ±10% масштаб
```

```
    tf.keras.layers.RandomTranslation(0.1, 0.1),  # ±10% зміщення XY
], name='data_augmentation')

# Включення до train_dataset:
train_dataset = train_dataset.map(
    lambda x, y: (data_augmentation(x, training=True), y),
    num_parallel_calls=tf.data.AUTOTUNE
)
```

4. Архітектура нейронної мережі CNN

4.1. Базова архітектура - Модель 1

Мінімальна CNN для ознайомлення з принципами побудови. Використовується як відправна точка для порівняльного аналізу. Очікувана точність на тесті: 97-98%.

```
model_v1 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3),
                           input_shape=(28, 28, 1),
                           activation='relu',
                           name='conv1'),
    tf.keras.layers.MaxPool2D(name='pool1'),
    tf.keras.layers.Conv2D(64, (2, 2),
                           activation='relu',
                           name='conv2'),
    tf.keras.layers.Flatten(name='flatten'),
    tf.keras.layers.Dense(128, activation='relu', name='fc1'),
    tf.keras.layers.Dense(10,  activation='softmax', name='output'),
], name='CNN_v1_baseline')

model_v1.summary()
```

4.2. Проведення навчання

```
model_v1.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['sparse_categorical_accuracy'],)
```

В коді використано міра помилки – *sparse categorical crossentropy*, метрика оптимізації *adam* (метод адаптивної інерції), який поєднує в собі і ідею накопичення руху і ідею слабшого поновлення ваги для типових ознак.

```
history = model_v1.fit(train_dataset, epochs = 5, validation_data = val_dataset)
```

Побудова графіків історії навчання та діагностика моделі

```
def show_history_charts(history):
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Графік функції втрат (Loss)
    axes[0].plot(history.history['loss'],     label='Train',      color='s
teelblue')
    axes[0].plot(history.history['val_loss'], label='Validation',
color='coral')
    axes[0].set_title('Функція втрат (Loss)', fontsize=13)
    axes[0].set_xlabel('Epoch'); axes[0].set_ylabel('Loss')
    axes[0].legend(); axes[0].grid(alpha=0.3)

    # Графік точності (Accuracy)
    key = 'sparse_categorical_accuracy'
    axes[1].plot(history.history[key],           label='Train',      color
='steelblue')
    axes[1].plot(history.history['val_' + key],  label='Validation',
color='coral')
    axes[1].set_title('Точність (Accuracy)', fontsize=13)
    axes[1].set_xlabel('Epoch'); axes[1].set_ylabel('Accuracy')
    axes[1].set_ylim(0.9, 1.0)
    axes[1].legend(); axes[1].grid(alpha=0.3)

    plt.suptitle('Історія навчання', fontsize=14)
    plt.tight_layout()
    plt.savefig('training_history.png', dpi=150, bbox_inches='tight')
    plt.show()

show_history_charts(history)
```

Таблиця 2

Діагностика стану моделі за графіками навчання

Ситуація	Ознаки на графіку	Рекомендоване рішення
Overfitting (перенавчання)	Train loss ↓, Val loss ↑ (розходяться)	Збільшити Dropout; аугментація; L2-регуляризація
Underfitting (недонавчання)	Обидва loss залишаються великими	Глибша архітектура; більше фільтрів; більше епох
Нестабільне навчання	Різкі осциляції loss по епохах	Зменшити learning rate; збільшити batch size
Рання конвергенція	Loss стабілізується вже після 3–5 епох	Перевірити preprocessing; зменшити LR
Оптимальний стан	Обидва loss знижуються та сходяться	Модель готова; зберегти та перейти до тестування

Висновки. Отримані значення свідчать про *overfitting*. Потрібно змінити модель додавши додатковий шар Dropout і змінити кількість ітерацій навчання.

4.4. Модель з Dropout - Модель 2

Шар Dropout під час кожного кроку навчання випадково «вимикає» частку neurons з імовірністю rate. Це змушує архітектуру генерувати універсальні ознаки, підвищуючи стабільність класифікації та запобігаючи надмірній адаптації до навчальної вибірки. Цільовий показник точності (accuracy) очікується в межах 98–98.5%

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), input_shape=(28, 28, 1), activation='
relu'),
  tf.keras.layers.MaxPool2D(),
  tf.keras.layers.Conv2D(64, (2, 2), activation='relu'),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.35),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

4.5. Глибока архітектура — Модель 3

Подвоєні Conv2D-блоки з BatchNormalization та збільшеною кількістю фільтрів. BatchNormalization нормалізує вихід кожного шару, що прискорює навчання та дозволяє використовувати вищий learning rate. Очікувана точність: > 99%.

```
model_v3 = tf.keras.models.Sequential([
    # Блок 1: локальні деталі (32 фільтри, ядро 5×5)
    tf.keras.layers.Conv2D(32, (5,5), padding='same',
                           input_shape=(28,28,1), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(32, (5,5), padding='same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.25),

    # Блок 2: абстрактні ознаки (64 фільтри, ядро 3×3)
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation='relu'),
```

```
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.25),

    # Класифікатор
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.35),
    tf.keras.layers.Dense(10, activation='softmax'),
], name='CNN_v3_deep')

print('Параметрів:', model_v3.count_params())
```

5. *Callbacks - автоматизація процесу навчання*

Callbacks - об'єкти, що реагують на події під час навчання. Три найважливіші: *EarlyStopping* зупиняє навчання при припиненні покращення, *ModelCheckpoint* зберігає найкращі ваги, *ReduceLROnPlateau* динамічно зменшує learning rate при стагнації:

EarlyStopping - зупинка якщо *val_loss* не покращується 5 епох, *restore_best_weights=True* - автоматично повертає найкращі ваги

```
callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True,
        verbose=1
    ),
```

ModelCheckpoint зберігає модель лише якщо *val_accuracy* зросла

```
    tf.keras.callbacks.ModelCheckpoint(
        filepath='best_mnist_model.keras',
        monitor='val_sparse_categorical_accuracy',
        save_best_only=True,
        verbose=1
    ),
```

*ReduceLROnPlateau: LR *= 0.5* якщо *val_loss* не падає 3 епохи

```
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,       # новий LR = старий LR * 0.5
```

```
        patience=3,
        min_lr=1e-6,
        verbose=1
    ),
]

history = model_v3.fit(
    train_dataset,
    epochs=3,
    validation_data=val_dataset,
    callbacks=callbacks,
    verbose=1
)

best_acc = max(history.history['val_sparse_categorical_accuracy'])
print(f'Найкраща val accuracy: {best_acc:.4f}')
```

6. *Тестування та оцінка якості моделі*

6.1. Тестування на стандартній тестовій вибірці

Фінальний етап машинного навчання - валідація моделі на незалежних даних. Процес складається з трьох ключових моментів:

– підготовки даних через конвеєр tf.data,

```
test_images_ds = tf.data.Dataset.from_tensor_slices(x_test)
test_images_ds = test_images_ds.map(preprocess_image,
                                    num_parallel_calls=tf.data.AUTOTUNE)
test_labels_ds = tf.data.Dataset.from_tensor_slices(y_test)
test_dataset = tf.data.Dataset.zip((test_images_ds, test_labels_ds))
test_dataset = test_dataset.batch(128).prefetch(tf.data.AUTOTUNE)
```

– отримання прогнозів

```
y_pred_probs = model_v3.predict(test_dataset, verbose=1)
y_pred       = np.argmax(y_pred_probs, axis=1)
```

– статистичного аналізу помилок.

```
correct   = np.sum(y_pred == y_test)
incorrect = np.sum(y_pred != y_test)
accuracy  = correct / len(y_test) * 100

print(f'Test Accuracy  : {accuracy:.2f}%')
print(f'Правильних     : {correct} / {len(y_test)}')
print(f'Неправильних   : {incorrect}')

# Збереження індексів помилкових передбачень
invalid_prediction_indexes = np.where(y_pred != y_test)[0]
```

6.2. Матриця плутанини та детальний звіт по класах

Матриця плутанини (confusion matrix) - таблиця N×N, де рядки відповідають вхідним класам, а стовпці - передбаченим. Позадіагональні елементи показують, які саме класи модель плутає між собою. Для MNIST типовими помилками є: 4↔9, 3↔5, 7↔1.

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=range(10), yticklabels=range(10))
plt.title('Матриця плутанини (Confusion Matrix)', fontsize=14)
plt.ylabel('Справжній клас'); plt.xlabel('Передбачений клас')
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=150)
plt.show()
```

Детальний звіт по кожному класу

```
print(classification_report(
    y_test, y_pred,
    target_names=[f'Цифра {i}' for i in range(10)]
))
```

Виводить *precision, recall, f1-score, support* для кожного класу

6.3. Візуалізація та аналіз помилкових передбачень

Можна переглянути зображення, які мережа класифікує не коректно:

```
invalid_prediction_indexes = []
for i in range(len(y_test)):
  if np.argmax(y_pred[i]) != y_test[i]:
    invalid_prediction_indexes.append(i)

index = 0

print("Predicted:", np.argmax(y_pred[invalid_prediction_indexes[index]]))
visualize_dataset_item(x_test[invalid_prediction_indexes[index]], y_test[i
nvalid_prediction_indexes[index]])
```

Оцінимо якість навчання мережі на тестових даних:

```
print("Test accuracy:", str(100 -
 round(len(invalid_prediction_indexes) / len(y_pred) * 100, 2)) + "%")
```

7. Збереження та завантаження моделі

TensorFlow підтримує кілька форматів збереження з різними характеристиками та призначенням.

Таблиця 3

Формати збереження моделі TensorFlow

Формат	**Розширення**	**Переваги**	**Застосування**
Keras Native	.keras	Зручний, підтримує довільні об'єкти Python	Розробка, продовження навчання
HDF5 (legacy)	.h5	Широка сумісність зі старими системами	Спільний доступ, архіви
TFLite	.tflite	Малий розмір після квантизації, швидкий	Android, iOS, Edge/IoT пристрої
ONNX (через tf2onnx)	.onnx	Фреймворк-незалежний, PyTorch-сумісний	Мультифреймворкові системи

Keras Native рекомендований для розробки

```
model_v3.save('mnist_cnn_v3.keras')
```

TFLite зі стандартною квантизацією (float32 -> int8, менший розмір)

```
converter = tf.lite.TFLiteConverter.from_keras_model(model_v3)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()
with open('mnist_model.tflite', 'wb') as f:
    f.write(tflite_model)

import os
keras_size  = os.path.getsize('mnist_cnn_v3.keras')  / 1024
tflite_size = os.path.getsize('mnist_model.tflite')  / 1024
print(f'.keras  розмір: {keras_size:.1f}  KB')
print(f'.tflite розмір: {tflite_size:.1f} KB  (зменшення: {100*(1-
tflite_size/keras_size):.0f}%)')
```

Завантаження та перевірка моделі

```
loaded_model = tf.keras.models.load_model('mnist_cnn_v3.keras')
```

Верифікація, передбачення на одному зображенні

```
sample = tf.expand_dims(preprocess_image(x_test[0]), axis=0)  # shape:
(1,28,28,1)
prob   = loaded_model.predict(sample, verbose=0)[0]
print(f'Передбачення: {np.argmax(prob)} | Правильна відповідь:
{y_test[0]}')
```

```
print(f'Впевненість:  {prob.max()*100:.2f}%')
```

8. Тестування на незалежних даних

8.1. Обробка та передбачення для зовнішнього зображення

Тестування на зображеннях, що не входять до датасету MNIST, - ключовий крок оцінки реальної якості моделі. Основна складність: MNIST має білі цифри на чорному фоні, а наприклад зображення з Paint - навпаки. Необхідна інверсія.

```
from PIL import Image
import requests
from io import BytesIO

def preprocess_custom_image(image_path):
    """
    Завантажує зображення, конвертує у формат MNIST (28x28, grayscale,
    білі цифри на чорному фоні) та виконує per_image_standardization.
    """
    img = Image.open(image_path).convert('L')  # конвертація у grayscale
    img = img.resize((28, 28), Image.LANCZOS)  # масштабування до 28x28
    img_array = np.array(img, dtype=np.float32)

    # Інверсія: якщо фон білий (>128 середнє), інвертуємо
    if img_array.mean() > 128:
        img_array = 255.0 - img_array

    tensor = tf.reshape(img_array, [28, 28, 1])
    tensor = tf.cast(tensor, tf.float32)
    tensor = tf.image.per_image_standardization(tensor)
    return tensor[tf.newaxis, ...], img_array  # (1,28,28,1) + raw


def predict_digit(image_path, model):
    """Повна діагностика: відображення + бар-чарт ймовірностей."""
    tensor, raw = preprocess_custom_image(image_path)
    probs  = model.predict(tensor, verbose=0)[0]
    pred   = int(np.argmax(probs))
    conf   = float(probs[pred]) * 100

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
    ax1.imshow(raw, cmap='gray')
    ax1.set_title(f'Передбачення: {pred} ({conf:.1f}%)', fontsize=13)
    ax1.axis('off')
    colors = ['crimson' if i == pred else 'steelblue' for i in range(10)]
    ax2.bar(range(10), probs, color=colors)
    ax2.set_xticks(range(10))
    ax2.set_title('Розподіл ймовірностей по класах')
```

```
    ax2.set_xlabel('Клас'); ax2.set_ylabel('Ймовірність')
    ax2.set_ylim(0, 1)
    plt.tight_layout(); plt.show()

    return pred, conf
```

Приклад використання:

```
pred, conf = predict_digit('my_digit_7.png', loaded_model)
print(f'Результат: цифра {pred}, впевненість: {conf:.1f}%')
```

Контрольні запитання

1. Назвіть сфери застосування нейронних мереж.
2. Назвіть класифікацію нейронних мереж за топологією. Чим відрізняються рекурентні (RNN/LSTM) від згорткових (CNN) мереж?
3. До яких мереж належать мережі без зворотних зв'язків (feedforward)? Чому CNN-класифікатор є feedforward мережею, а не рекурентною?
4. На які види поділяють нейронні мережі за принципом структури нейронів? Поясніть відмінність функцій активації: Sigmoid, ReLU, LeakyReLU, Softmax.
5. Поясніть поняття «Перцептрон». Чому класичний перцептрон не розв'язує задачу XOR?
6. З яких типів елементів складається перцептрон? Опишіть роль вхідного, прихованого та вихідного шарів. Яку функцію виконують ваги (weights) та зміщення (bias)?
7. Який перцептрон називається багатошаровим (MLP)? У чому принципова відмінність MLP від CNN при роботі з зображеннями?
8. Які шари використовуються в CNN? Опишіть послідовність Conv2D → BatchNorm → ReLU → MaxPool → Dropout. Навіщо кожен шар і яку проблему він вирішує?
9. Назвіть та охарактеризуйте етапи побудови нейромережевої моделі: від збору даних до деплою. Який етап є найбільш критичним з практичної точки зору та чому?

10. Яким чином перевіряється правильність роботи навченої мережі? Поясніть відмінність між accuracy, precision, recall та F1-score. Коли кожна з них є пріоритетною?

11. Охарактеризуйте навчання «з учителем» (supervised learning).

12. Охарактеризуйте навчання «без учителя» (unsupervised learning).

13. Назвіть недоліки алгоритму зворотного розповсюдження помилки. Що таке «зникаючий градієнт» (vanishing gradient) і як BatchNormalization, ResNet (skip connections) вирішують цю проблему?

14. Охарактеризуйте датасет MNIST. Чому точність моделі на власноруч написаних цифрах зазвичай нижча, ніж на MNIST-тесті?

15. Що таке overfitting та underfitting? Назвіть і поясніть 5 технік боротьби з overfitting: Dropout, L1/L2 регуляризація, BatchNorm, аугментація, EarlyStopping.

16. Що таке Transfer Learning? Коли Transfer Learning є критичним, а коли навчання з нуля є кращим вибором?

17. Порівняйте оптимізатори SGD, Adam, RMSprop, AdamW та LAMB. Який оптимізатор є найбільш доцільним для CNN на зображеннях і чому? Що таке weight decay?