Практична робота 4

Виконали: Безкоровайний Назарій та Браткова Поліна КА-23

Ми обрали два набори даних, де один - з кольоровими зображеннями, а інший - з чорно-білими:

- **Shoe vs Sandal vs Boot Dataset** - кольорові зображення (обраний додатково)
- **Chinese MNIST** - чорно-білі зображення (наш варіант)

1. Завантажити два набори зображень згідно з варіантом: чорно-білі (дивіться попередній практикум) та кольорові. Якщо набір великого розміру, можна обрати частину.

ImageDataGenerator() - створюється генератор для попередньої обробки зображень перед подачею в модель. Зображення будуть змінюватися (обертатися, зсуватися, масштабуватися, дзеркалитися), щоб допомогти моделі бути стійкішою до різних варіацій.

```
if mode == 0:
    base_dir = './Shoe vs Sandal vs Boot Dataset'
    train_dir = base_dir

    # Аугментація та розділення на train/val
    train_datagen = ImageDataGenerator(
        rescale=1.0 / 255,
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
        validation_split=0.2
    )

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical',
        subset='training'
```

```
    )

    val_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical',
        subset='validation',
        shuffle=False
    )

    num_classes = train_generator.num_classes

elif mode == 1:
    csv_path = './Chinese MNIST/chinese_mnist.csv'
    image_dir = './Chinese MNIST/data'

    # Завантаження CSV
    df = pd.read_csv(csv_path)

    df['filename'] = df.apply(lambda row:
f"data/input_{row['suite_id']}_{row['sample_id']}_{row['code']}.jpg", axis=1)
    df['filepath'] = df['filename'].apply(lambda f: os.path.join(image_dir, f))
    df = df[df['filepath'].apply(os.path.exists)]

    df['label'] = (df['value'] - 1).astype(str)

    # Ділимо на train/val
    from sklearn.model_selection import train_test_split
    train_df, val_df = train_test_split(df, test_size=0.2, random_state=42,
stratify=df['label'])

    # Генератор зображень
    datagen = ImageDataGenerator(rescale=1./255)


    train_generator = datagen.flow_from_dataframe(
        train_df,
        x_col='filepath',
        y_col='label',
        target_size=(64, 64),
        class_mode='categorical',
        batch_size=32,
        shuffle=True
    )
```

```
  val_generator = datagen.flow_from_dataframe(
      val_df,
      x_col='filepath',
      y_col='label',
      target_size=(64, 64),
      class_mode='categorical',
      batch_size=32,
      shuffle=False
  )

  num_classes = len(train_generator.class_indices)
```

2. Підготувати дані для навчання - за необхідності.

Зроблено в п.1

3. Розділити дані на навчальну, перевірочну і тестову підмножини.

Зроблено в п.1

4. Побудувати і навчити базову модель з одним згортковим шаром. Оцінити правильність (accuracy) та точність моделі на тренувальній і перевірочній множинах.

```
model = Sequential(
  [
      Conv2D(
          32,
          kernel_size=(3, 3), # (5, 5)
          strides=(1, 1), # (2, 2)
          padding="valid", # same
          activation="relu",
          input_shape=(64, 64, 3),
      ),
      MaxPooling2D(pool_size=(2, 2)),
      Flatten(),
      Dense(128, activation="relu"),
      Dropout(0.5),
      Dense(num_classes, activation="softmax"),
  ]
)
```

Validation Accuracy: 0.8577

CNN Classification Report:

	precision	recall	f1-score	support
0	0.94	0.94	0.94	1000
1	0.85	0.77	0.81	1000
2	0.79	0.86	0.82	1000
accuracy			0.86	3000
macro avg	0.86	0.86	0.86	3000
weighted avg	0.86	0.86	0.86	3000

CNN AUC: 0.9582

Модель досить точна навіть і з одним шаром.

5. Дослідити різні значення параметрів padding і strides згорткового шару базової моделі, а також розмірність ядра (kernel) згортки та їх вплив на точність моделі. На перевірочній множині обрати значення вказаних параметрів.

```
model = Sequential(
    [
        Conv2D(
            32,
            kernel_size=(5, 5), # (3, 3)
            strides=(2, 2), # (1, 1)
            padding="same", # valid
            activation="relu",
            input_shape=(64, 64, 3),
        ),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(128, activation="relu"),
        Dropout(0.5),
        Dense(num_classes, activation="softmax"),
    ]
)
```

Validation Accuracy: 0.8843

CNN Classification Report:

	precision	recall	f1-score	support
0	0.93	0.95	0.94	1000
1	0.85	0.86	0.85	1000
2	0.86	0.83	0.84	1000
accuracy			0.88	3000
macro avg	0.88	0.88	0.88	3000
weighted avg	0.88	0.88	0.88	3000

CNN AUC: 0.9705

6. Дослідити кілька альтернативних архітектур згорткових моделей, які включають: кілька згорткових шарів, шар/шари нормалізації за міні-батчами, шар/шари дропауту. На перевірочній множині оцінити якість побудованих моделей і обрати найкращу архітектуру. Використати показники якості: accuracy, precision, recall, f1-score, AUC.

Будуємо згорткову нейронну мережу з 3 згортковими шарами. Очікується вхідне зображення розміром 64×64 пікселів з 3 каналами (кольорове RGB зображення). **Conv2D(16, (3, 3)):** 16 фільтрів розміром 3×3, виявляє локальні шаблони (краї, текстури тощо). **MaxPooling2D((2, 2)):** зменшує розмірність зображення, виділяючи найважливіші ознаки, зменшуючи обчислювальні витрати та переобучення. **Flatten():** перетворює 3D-вихід із згорткових шарів у 1D-вектор, який можна подати у повнозв’язні (Dense) шари. **Dense(256):** 256 нейронів, класична повнозв'язна шарова структура для прийняття рішення на основі виділених ознак. **Dropout(0.5):** вимикає випадково 50% нейронів під час тренування для зменшення переобучення. **Dense(num_classes, activation='softmax'):** вихідний шар, де к-сть класів = к-сть нейронів, softmax перетворює виходи в імовірності — значення в межах [0,1], які сумуються до 1, для класифікації.

Методика така: фільтруємо - зменшуємо - сильніше фільтруємо - зменшуємо - … Таким чином оптимізуємо результати.

```
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D((2, 2)),

    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(num_classes, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Примітка. Досліджувалось в роботі більше архітектур. Проілюстрована найвдаліша.

```
# Навчання моделі
early_stop = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=10,
    validation_data=val_generator,
    validation_steps=len(val_generator),
    callbacks=[early_stop]
)

# Оцінка моделі
loss, accuracy = model.evaluate(val_generator)
print(f"Validation Accuracy: {accuracy:.4f}")
```

```
# Передбачення для валідаційної множини
val_preds = model.predict(val_generator)
val_pred_labels = np.argmax(val_preds, axis=1)
true_labels = val_generator.classes

# classification report
print("\nCNN Classification Report:\n")
print(classification_report(true_labels, val_pred_labels))

# AUC (One-vs-Rest)
y_true_bin = label_binarize(true_labels, classes=np.arange(num_classes))
auc = roc_auc_score(y_true_bin, val_preds, average='macro', multi_class='ovr')
print(f"CNN AUC: {auc:.4f}")
```

Примітка. Результати позитивні. Будуть показані в п.9.

7. Чи впливає на правильність (accuracy) моделі додавання регуляризації: дропаут, рання зупинка навчання, та використання різних методів ініціалізації ваг?

Дропаут - не дає моделі перенавчитись. Рання зупинка навчання - оптимізує алгоритм (не змінюється loss протягом кількох епох - завершуємо). Тому так, це може іноді допомогти.

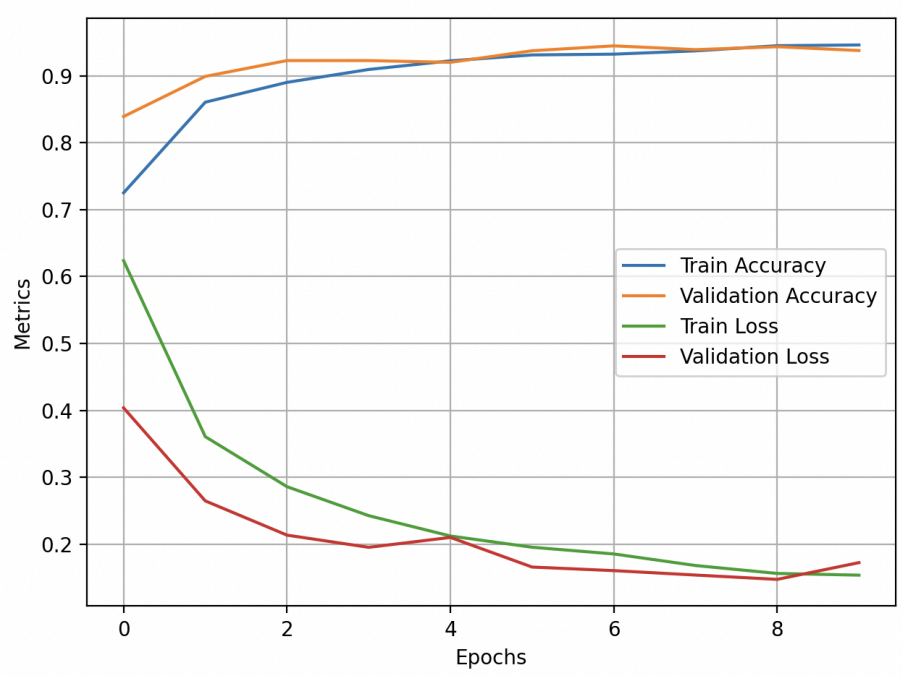
8. Відобразити графіки, які ілюструють оцінки якості навчання мереж на навчальній та перевірочній множинах

```
# Візуалізація навчання
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Metrics')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

9. Розрахувати на тестовій множині оцінки якості обраної найкращої моделі.

Додатково було реалізовано MLP для порівняння (повний лістинг буде прикладений як додатковий файл типу .py)

Shoe vs Sandal vs Boot Dataset



Validation Accuracy: 0.9450

CNN Classification Report:

	precision	recall	f1-score	support
0	0.98	0.94	0.96	1000
1	0.95	0.91	0.93	1000
2	0.88	0.95	0.91	1000
accuracy			0.93	3000
macro avg	0.93	0.93	0.93	3000
weighted avg	0.93	0.93	0.93	3000

CNN AUC: 0.9924

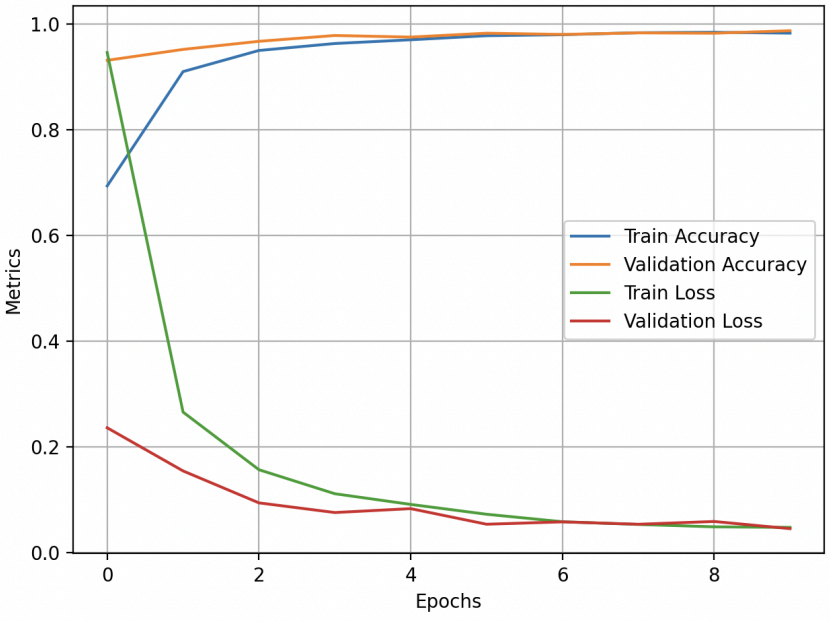
MLP Classification Report:

	precision	recall	f1-score	support
1	0.89	0.95	0.92	1000
2	0.94	0.78	0.85	1000
3	0.84	0.92	0.87	1000
accuracy			0.88	3000
macro avg	0.89	0.88	0.88	3000
weighted avg	0.89	0.88	0.88	3000

MLP AUC: 0.9692

MLP Accuracy: 0.8833
CNN Accuracy: 0.9450

Chinese MNIST



Validation Accuracy: 0.9877

CNN Classification Report:

	precision	recall	f1-score	support

0	1.00	1.00	1.00	200
1	1.00	0.99	1.00	200
2	0.99	0.97	0.98	200
3	0.97	1.00	0.99	200
4	0.98	0.98	0.98	200
5	1.00	0.99	0.99	200
6	0.99	1.00	1.00	200
7	0.99	0.99	0.99	200
8	1.00	1.00	1.00	200
9	0.99	0.97	0.98	200
10	0.97	0.97	0.97	200
11	0.98	0.97	0.98	200
12	0.98	0.97	0.98	200
13	0.98	1.00	0.99	200
14	0.98	0.99	0.99	200
accuracy			0.99	3000
macro avg	0.99	0.99	0.99	3000
weighted avg	0.99	0.99	0.99	3000

CNN AUC: 0.9999

MLP Classification Report:

	precision	recall	f1-score	support
-1	0.92	0.86	0.89	200
0	0.85	0.81	0.83	200
1	0.40	0.86	0.55	200
2	0.22	0.06	0.09	200
3	0.89	0.70	0.79	200
4	0.71	0.62	0.67	200
5	0.72	0.72	0.72	200
6	0.81	0.62	0.70	200
7	0.93	0.90	0.91	200
8	0.61	0.64	0.62	200

9	0.71	0.78	0.74	200
99	0.64	0.68	0.66	200
999	0.72	0.73	0.73	200
9999	0.65	0.70	0.68	200
99999999	0.80	0.82	0.81	200
accuracy			0.70	3000
macro avg	0.71	0.70	0.69	3000
weighted avg	0.71	0.70	0.69	3000

MLP AUC: 0.9491

MLP Accuracy: 0.7010
CNN Accuracy: 0.9877

Примітка. В даній роботі використовували MLP з бібліотеки. В минулій роботі ми реалізовували MLP вручну. Для набору Chinese MNIST, поставивши однакові налаштування як і в минулій роботі, отримали однакову точність як і в минулій роботі - 0.7010. Для набору Shoe vs Sandal vs Boot Dataset використали кращий MLP з краще підібраними параметрами, аби побачити найвищу точність.

10. Завантажити зображення тестової множини і розпізнати його навченими моделями.

Переконуємося, що CNN розпізнає дійсно правильно. Завантажимо, до прикладу, перше зображення з boot. Отримаємо 0-й клас - клас призначений для boot.

```
if mode == 0:
    image_path = './Shoe vs Sandal vs Boot Dataset/boot/boot (1).jpg'

    img = load_img(image_path, target_size=(64, 64))
    img_array = img_to_array(img) / 255.0

    img_cnn = np.expand_dims(img_array, axis=0)  # (1, 64, 64, 3)

    cnn_prediction = model.predict(img_cnn)
```

```
cnn_class = np.argmax(cnn_prediction)

print(f"\nCNN передбачений клас: {cnn_class}\n")
```

CNN передбачений клас: 0

11. Порівняти побудовані згорткові моделі та багатошаровий персептрон в задачі класифікації чорно-білих та кольорових зображень.

Примітка. Набір Shoe vs Sandal vs Boot Dataset має краще MLP Accuracy, ніж Chinese MNIST, через те, що параметри підібрані вдало з потужним оптимізатором. В наборі Chinese MNIST використовується MLP зі звичайним градієнтним спуском без моментів.

Shoe vs Sandal vs Boot Dataset	**Chinese MNIST**
MLP Accuracy: 0.8833 CNN Accuracy: 0.9450	MLP Accuracy: 0.7010 CNN Accuracy: 0.9877

Згорткова нейронна мережа явно краще працює з зображеннями, ніж багатошаровий персептрон.

12. Зробити висновки щодо якості класифікації на основі побудованих моделей.

CNN явно краще працює з зображеннями, ніж MLP. Наведемо важливі тези, які отримали у висновку з даної роботи.

- CNN зберігає структуру зображення: у зображеннях важливо, які пікселі знаходяться поруч. MLP руйнує просторову інформацію, бо розгортає зображення в один довгий вектор (flatten()), і не розрізняє, де був який піксель.
- CNN використовує фільтри (ядра), які вивчають локальні шаблони — краї, текстури, форми тощо. MLP не має поняття "локальності", тобто він не бачить маленьких областей зображення, а лише всю картину одразу.
- CNN має менше параметрів, бо фільтри повторно використовуються по всьому зображенню (weight sharing). MLP має дуже багато ваг —

наприклад, для 64×64×3 зображення це вже 12,288 входів, і кожен підключений до кожного нейрона наступного шару — це дуже багато параметрів.

- CNN вчиться бути нечутливим до дрібних зсувів і поворотів об’єктів (через згортки + пулінг). MLP — ні, і часто просто запам’ятовує конкретні позиції об’єктів.
- CNN масштабуються значно краще завдяки своїй архітектурі. MLP дуже швидко стає неефективним, якщо розмір зображення збільшується (наприклад, 128×128 або 256×256).

Отже, CNN набагато краще працює з зображеннями ніж MLP через достатньо велику кількість причин.