

Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій
Кафедра системного проектування

Звіт

Про виконання лабораторної роботи №5
З курсу «Системи машинного навчання»
«Нейронні мережі та PyTorch/Keras»

Виконала:

Студентка групи ФЕС-32

Філь Дарина

Перевірив:

Доцент Колич І. І.

Львів 2024

Мета: навчитися будувати та навчати прості нейронні мережі для задач класифікації та регресії.

Теоретичні відомості

Нейронні Мережі

Нейронні мережі — це обчислювальні моделі, які натхнені біологічними нейронними системами. Вони складаються з великої кількості взаємопов'язаних обчислювальних елементів, названих нейронами, які працюють разом для вирішення конкретних задач.

Основні Компоненти Нейронних Мереж

1. **Вхідний шар (Input Layer):** Приймає дані, які будуть оброблятися мережею.
2. **Приховані шари (Hidden Layers):** Шари, розташовані між вхідним і вихідним шарами, кожен з яких складається з певної кількості нейронів.
3. **Вихідний шар (Output Layer):** Видає результат обробки даних мережею.

Основні Параметри Нейронних Мереж

1. **Кількість шарів:** Впливає на здатність нейронної мережі навчатися складних патернів.
2. **Кількість нейронів у шарі:** Кожен шар може містити різну кількість нейронів, що впливає на обчислювальну потужність та здатність до узагальнення.
3. **Функції активації:** Визначають нелінійність, яку нейронна мережа може моделювати. Популярні функції активації включають ReLU, сигмоїдальну функцію та тангенс.

PyTorch та Keras

PyTorch — це відкрита бібліотека для машинного навчання, розроблена Facebook, яка є потужним інструментом для створення та навчання нейронних мереж.

Keras — це високорівнева бібліотека, яка може використовуватися з різними бекендами, включаючи TensorFlow та Theano. У даному прикладі ми будемо використовувати PyTorch як бекенд для Keras.

Хід роботи

Завдання

1. Підготовка середовища

1.1 Встановіть необхідні бібліотеки, якщо вони ще не встановлені: PyTorch, Keras, Matplotlib

1.2 Імпортуйте необхідні бібліотеки в Python.

2. Створення простої нейронної мережі, що відмінна від прикладу

2.1 Створіть та налаштуйте модель нейронної мережі за допомогою Keras (або PyTorch):

- Використовуйте послідовну модель (Sequential).
- Додайте шари до моделі (Dense). Кількість шарів має бути іншою, ніж у прикладі
- Вкажіть кількість нейронів у кожному шарі та функції активації.

2.2 Сконфігуруйте модель для навчання (в прикладі позначено):

- Виберіть алгоритм оптимізації (наприклад, sgd).
- Вкажіть функцію втрат (наприклад, categorical_crossentropy для задач класифікації).
- Вкажіть метрику для оцінки моделі (наприклад, accuracy).

3. Навчання нейронної мережі на наборі даних

3.1 Завантажте та підготуйте дані:

- Використайте набір даних, наприклад, Fashion-MNIST.
- Розділіть дані на тренувальну та тестову вибірки.
- Нормалізуйте дані, якщо необхідно.

3.2 Навчіть модель на тренувальних даних:

- Вкажіть кількість епох.
- Вкажіть розмір пакета (batch size).
- Збережіть історію навчання для подальшої візуалізації.

4. Оцінка та візуалізація результатів

4.1 Оцініть модель на тестових даних:

- Використайте метод оцінки (evaluate) для отримання метрик якості.

4.2 Візуалізуйте результати навчання та оцінки:

- Побудуйте графіки втрат та точності на тренувальних та тестових даних.
- Візуалізуйте помилково класифіковані приклади.

5. Оптимізація гіперпараметрів

5.1 Експериментуйте з різними параметрами моделі, щоб підвищити її точність:

- Змініть кількість шарів та нейронів.
- Використайте різні функції активації.
- Спробуйте різні алгоритми оптимізації.

5.2 Порівняйте результати:

- Запишіть результати експериментів.
- Проаналізуйте, як зміна параметрів впливає на якість моделі.

6. Оформити звіт

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.datasets import mnist
import os
import matplotlib.pyplot as plt
```

Рис.1 Завантаження бібліотек

```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 20)
        self.fc2 = nn.Linear(20, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
trainset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
testset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)
```

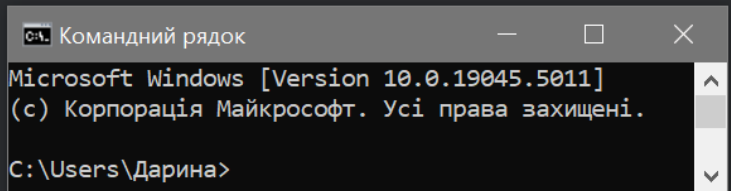


Рис.2 Розробка класу для простої нейронної мережі, завантаження набору даних Fashion-MNIST, його нормалізація та поділ на тренувальну і тестову вибірки

```
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

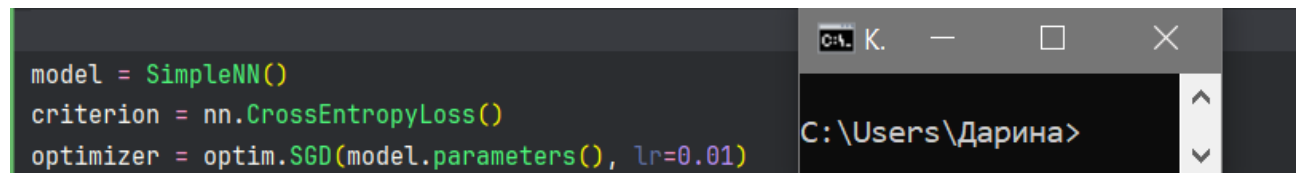


Рис.3 Вибір для моделі простої нейронної мережі, функції для класифікації та оптимізатора

```
n_epochs = 8
train_losses, test_losses = [], []
train_accuracies, test_accuracies = [], []

for epoch in range(n_epochs):
    running_loss = 0
    correct_train = 0
    total_train = 0

    model.train()
    for images, labels in trainloader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    _, predicted = torch.max(outputs.data, 1)
    total_train += labels.size(0)
    correct_train += (predicted == labels).sum().item()
```

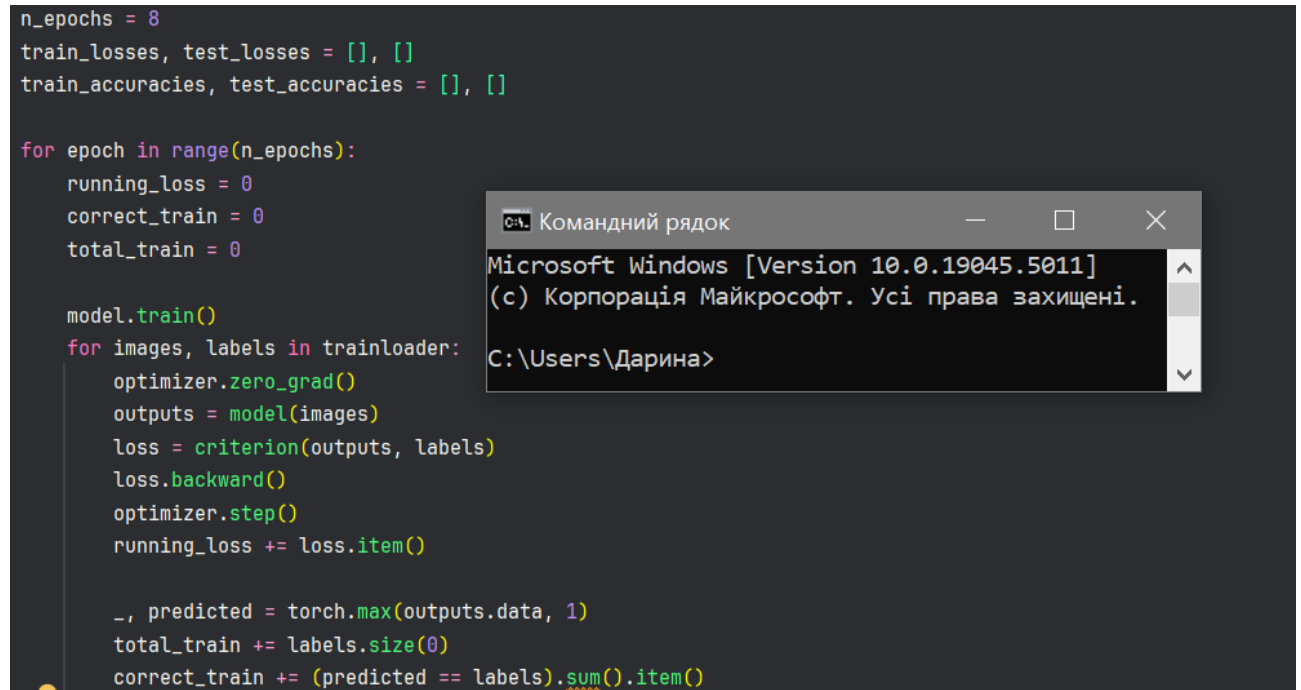


Рис. 4 Тренування моделі простої нейронної мережі, з кількістю епох = 8.

```
train_loss = running_loss / len(trainloader)
train_accuracy = 100 * correct_train / total_train
train_losses.append(train_loss)
train_accuracies.append(train_accuracy)

model.eval()
test_loss = 0
correct_test = 0
total_test = 0
with torch.no_grad():
    for images, labels in testloader:
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

    _, predicted = torch.max(outputs.data, 1)
    total_test += labels.size(0)
    correct_test += (predicted == labels).sum().item()

test_loss = test_loss / len(testloader)
test_accuracy = 100 * correct_test / total_test
test_losses.append(test_loss)
test_accuracies.append(test_accuracy)

print(f'Epoch {epoch + 1}/{n_epochs}, Train Loss: {train_loss}, Test Loss: {test_loss}, '
      f'Train Accuracy: {train_accuracy}%, Test Accuracy: {test_accuracy}%')
```

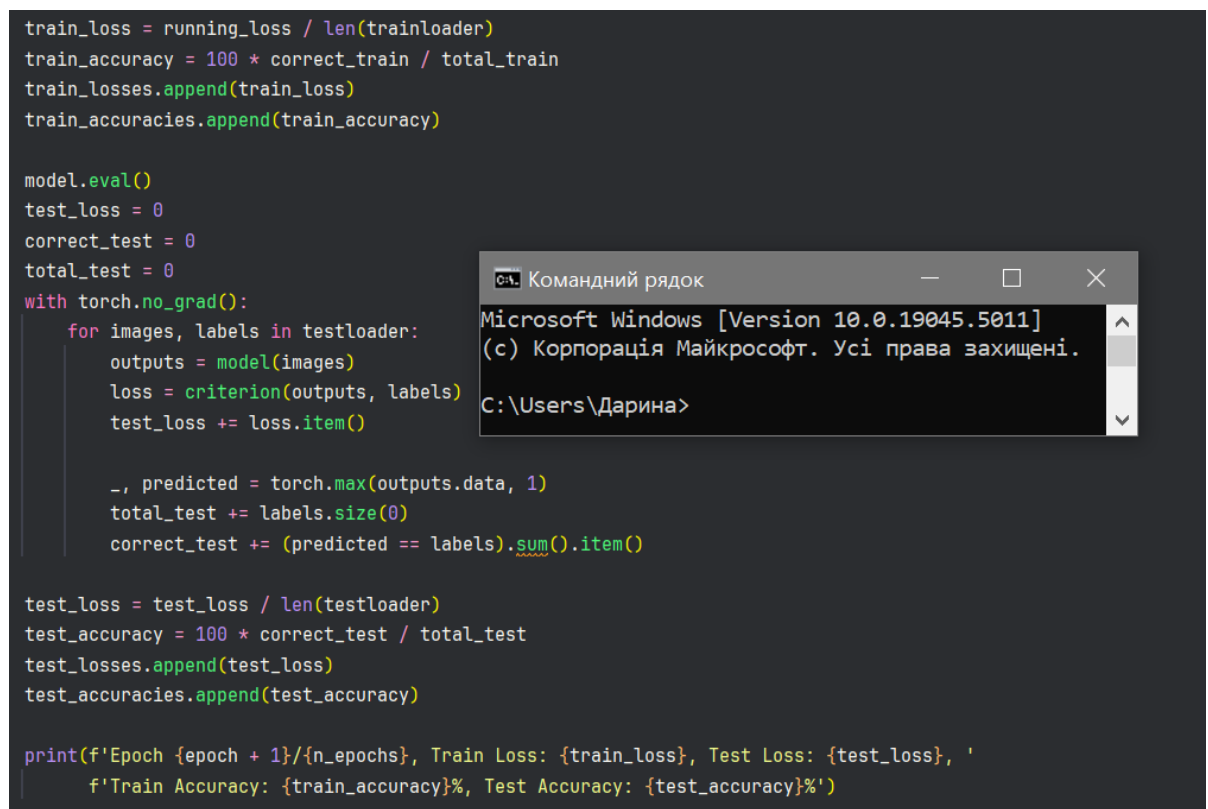


Рис.5 Код для оцінювання нашої моделі.

Epoch 1/8, Train Loss: 0.8217886887125369, Test Loss: 0.4105655503026239, Train Accuracy: 78.04166666666667%, Test Accuracy: 88.66%
Epoch 2/8, Train Loss: 0.3806043584892618, Test Loss: 0.3369900844635288, Train Accuracy: 89.14333333333333%, Test Accuracy: 90.48%
Epoch 3/8, Train Loss: 0.3356568569869502, Test Loss: 0.31884242254932216, Train Accuracy: 90.33833333333334%, Test Accuracy: 90.62%
Epoch 4/8, Train Loss: 0.3154600591325302, Test Loss: 0.2927339401475753, Train Accuracy: 90.855%, Test Accuracy: 91.7%
Epoch 5/8, Train Loss: 0.30140660169409283, Test Loss: 0.289911540331924, Train Accuracy: 91.29%, Test Accuracy: 91.71%
Epoch 6/8, Train Loss: 0.2916367744713196, Test Loss: 0.282259222511558, Train Accuracy: 91.59666666666666%, Test Accuracy: 91.83%
Epoch 7/8, Train Loss: 0.2829506052995541, Test Loss: 0.2717469290764943, Train Accuracy: 91.75833333333334%, Test Accuracy: 92.18%
Epoch 8/8, Train Loss: 0.27626361717769843, Test Loss: 0.2658057115366979, Train Accuracy: 91.91833333333334%, Test Accuracy: 92.33%

Рис.6 Результати навчання моделі за епохами показують, що з підвищенням кількості епох точність моделі збільшується. Однак видно, що з 7-ї до 8-ї епохи точність майже не змінилася

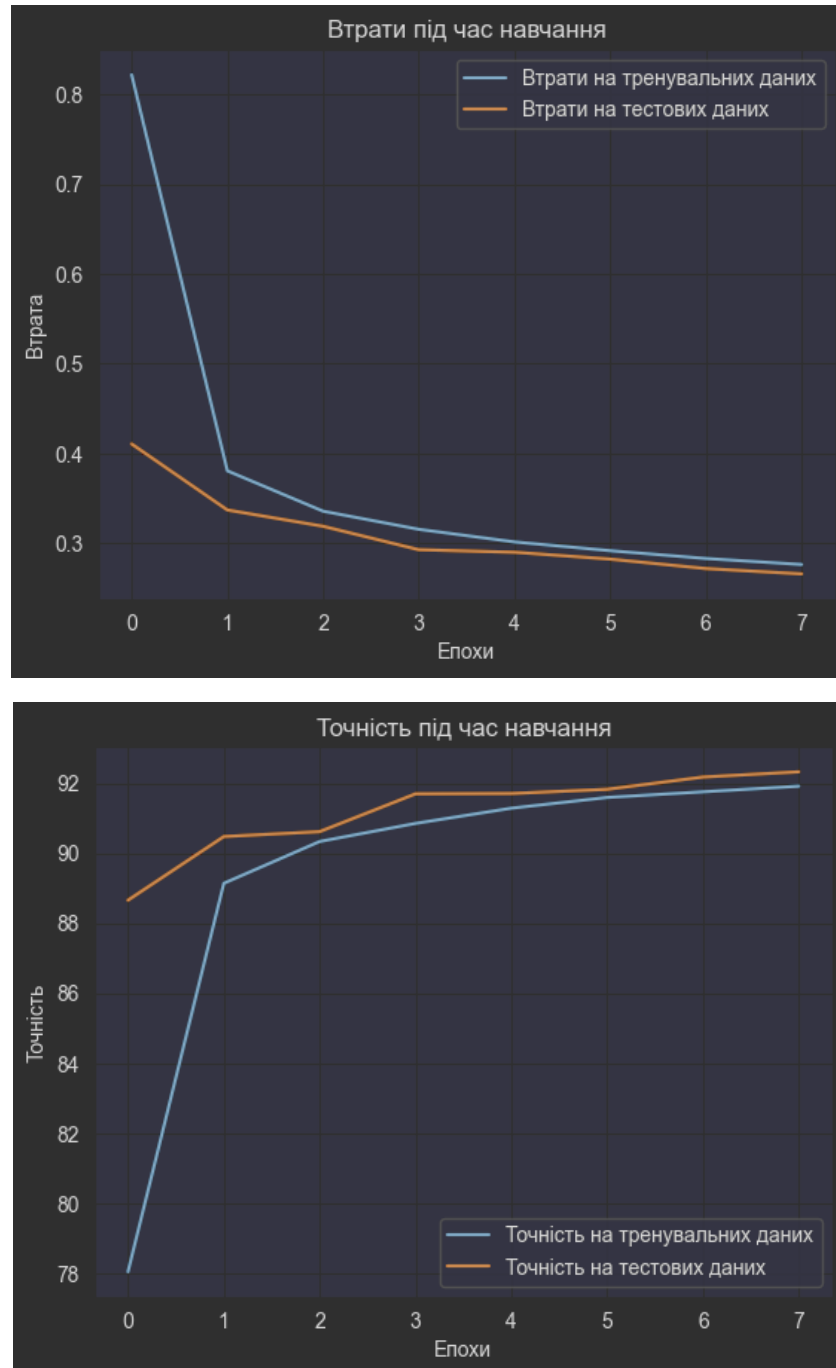


Рис.7-8 Графіки, що відображають точність та втрати відносно епохи тренування простої нейронної мережі.

```

os.environ["KERAS_BACKEND"] = "torch"
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images / 255.0
test_images = test_images / 255.0

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(5, activation='relu'),
    Dense(10, activation='softmax')
])

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

history = model.fit(train_images, train_labels, batch_size = 64, epochs=5, validation_split=0.2)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('\nTest accuracy:', test_acc)

```

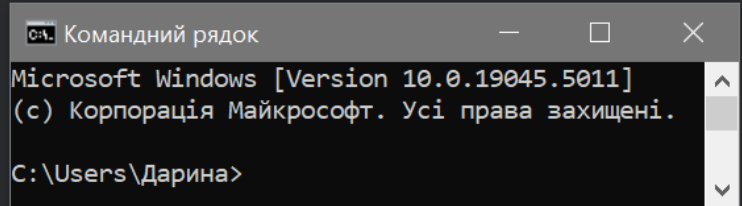


Рис.9 Створення та навчання нейронної мережі з послідовними шарами, де як бекенд обрано "torch", хоча це вже застаріла версія; наразі частіше використовується tensorflow. Для оптимізації застосовано оптимізатор adam, а кількість нейронів становить 5

```

Epoch 1/5
750/750 ————— 4s 3ms/step - accuracy: 0.4061 - loss: 1.6452 - val_accuracy: 0.7181 - val_loss: 0.8950
Epoch 2/5
750/750 ————— 2s 2ms/step - accuracy: 0.7409 - loss: 0.8498 - val_accuracy: 0.8169 - val_loss: 0.6817
Epoch 3/5
750/750 ————— 2s 2ms/step - accuracy: 0.8151 - loss: 0.6830 - val_accuracy: 0.8365 - val_loss: 0.6063
Epoch 4/5
750/750 ————— 2s 2ms/step - accuracy: 0.8362 - loss: 0.6108 - val_accuracy: 0.8454 - val_loss: 0.5702
Epoch 5/5
750/750 ————— 2s 2ms/step - accuracy: 0.8470 - loss: 0.5772 - val_accuracy: 0.8539 - val_loss: 0.5409
313/313 ————— 1s 2ms/step - accuracy: 0.8313 - loss: 0.6051

Test accuracy: 0.8482999801635742

```

Рис.10 Результат тренування моделі на 5 епохах

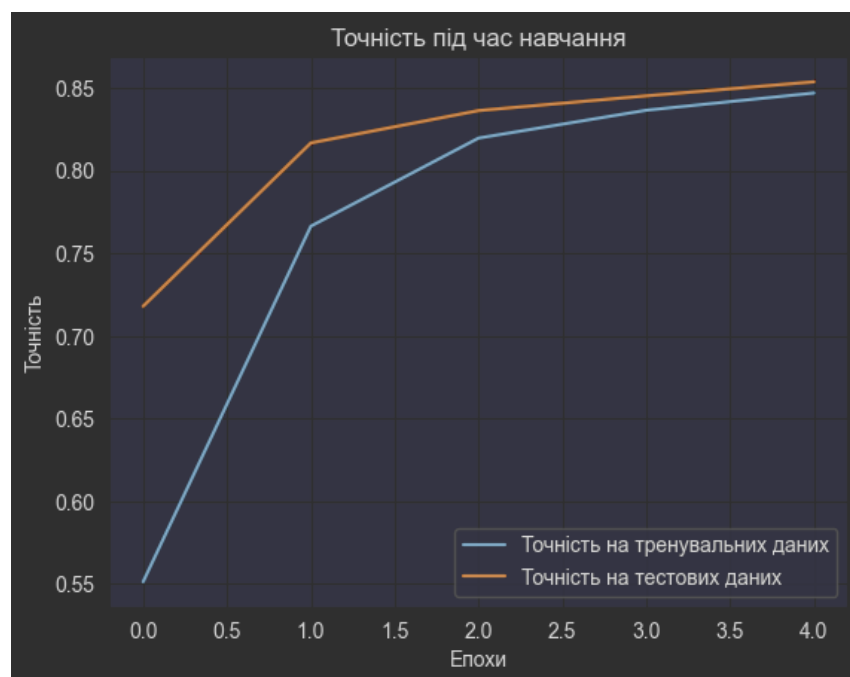
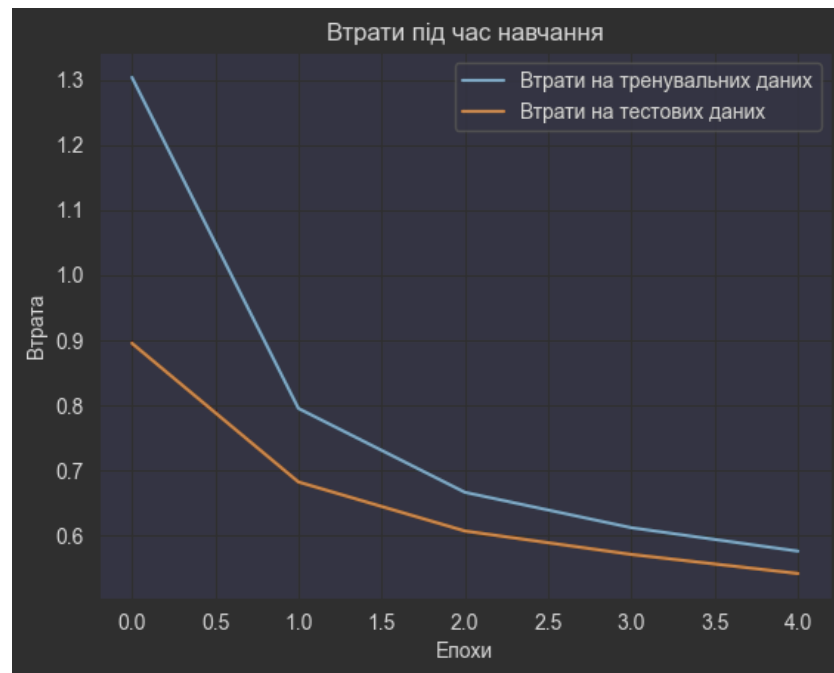


Рис.11-12 Графіки, що відображають точність та втрати відносно епохи тренування послідовної нейронної мережі.


```

os.environ["KERAS_BACKEND"] = "torch"
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images / 255.0
test_images = test_images / 255.0

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

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

history = model.fit(train_images, train_labels, batch_size = 32, epochs=8, validation_split=0.2)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('\nTest accuracy:', test_acc)

```

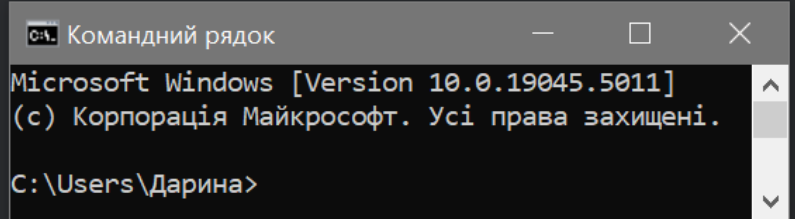


Рис.13 Збільшимо кількість нейронів, замінимо оптимізатор на rmsprop, зменшимо `batch_size` удвічі, додамо додаткові шари та застосуємо dropout з регуляризацією L2 для покращення узагальнювальної здатності моделі

```

Epoch 1/8
1500/1500 ————— 10s 5ms/step - accuracy: 0.7689 - loss: 0.7174 - val_accuracy: 0.9585 - val_loss: 0.1399
Epoch 2/8
1500/1500 ————— 7s 4ms/step - accuracy: 0.9473 - loss: 0.1862 - val_accuracy: 0.9692 - val_loss: 0.1099
Epoch 3/8
1500/1500 ————— 7s 4ms/step - accuracy: 0.9680 - loss: 0.1198 - val_accuracy: 0.9728 - val_loss: 0.0977
Epoch 4/8
1500/1500 ————— 6s 4ms/step - accuracy: 0.9743 - loss: 0.0900 - val_accuracy: 0.9703 - val_loss: 0.1088
Epoch 5/8
1500/1500 ————— 6s 4ms/step - accuracy: 0.9777 - loss: 0.0789 - val_accuracy: 0.9737 - val_loss: 0.1091
Epoch 6/8
1500/1500 ————— 10s 4ms/step - accuracy: 0.9823 - loss: 0.0626 - val_accuracy: 0.9733 - val_loss: 0.0994
Epoch 7/8
1500/1500 ————— 7s 4ms/step - accuracy: 0.9834 - loss: 0.0586 - val_accuracy: 0.9751 - val_loss: 0.0935
Epoch 8/8
1500/1500 ————— 7s 4ms/step - accuracy: 0.9862 - loss: 0.0475 - val_accuracy: 0.9743 - val_loss: 0.1074
313/313 ————— 1s 2ms/step - accuracy: 0.9739 - loss: 0.1115

Test accuracy: 0.9776999950408936

```

Рис.14 Результат тренування

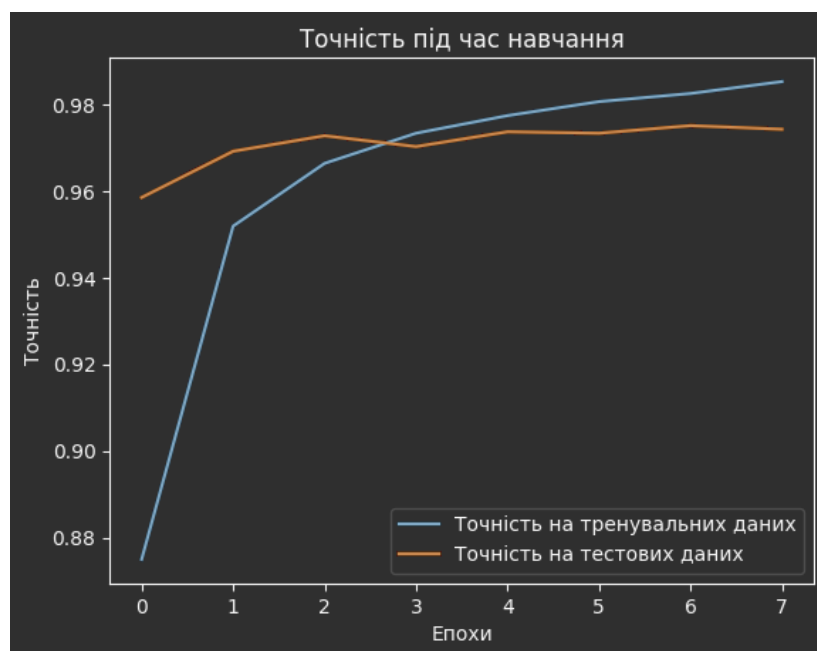
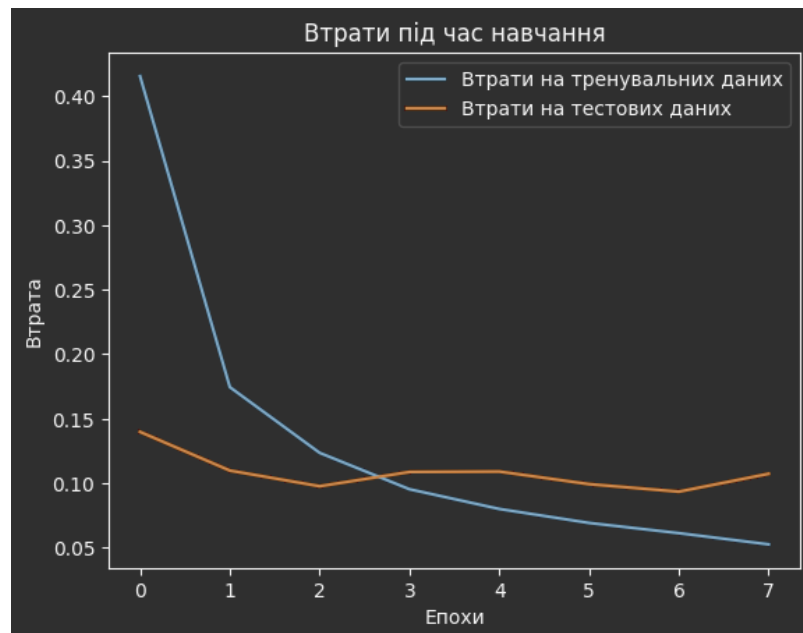


Рис.15-16 Графіки, що відображають точність та втрати відносно епохи тренування послідовної нейронної мережі.

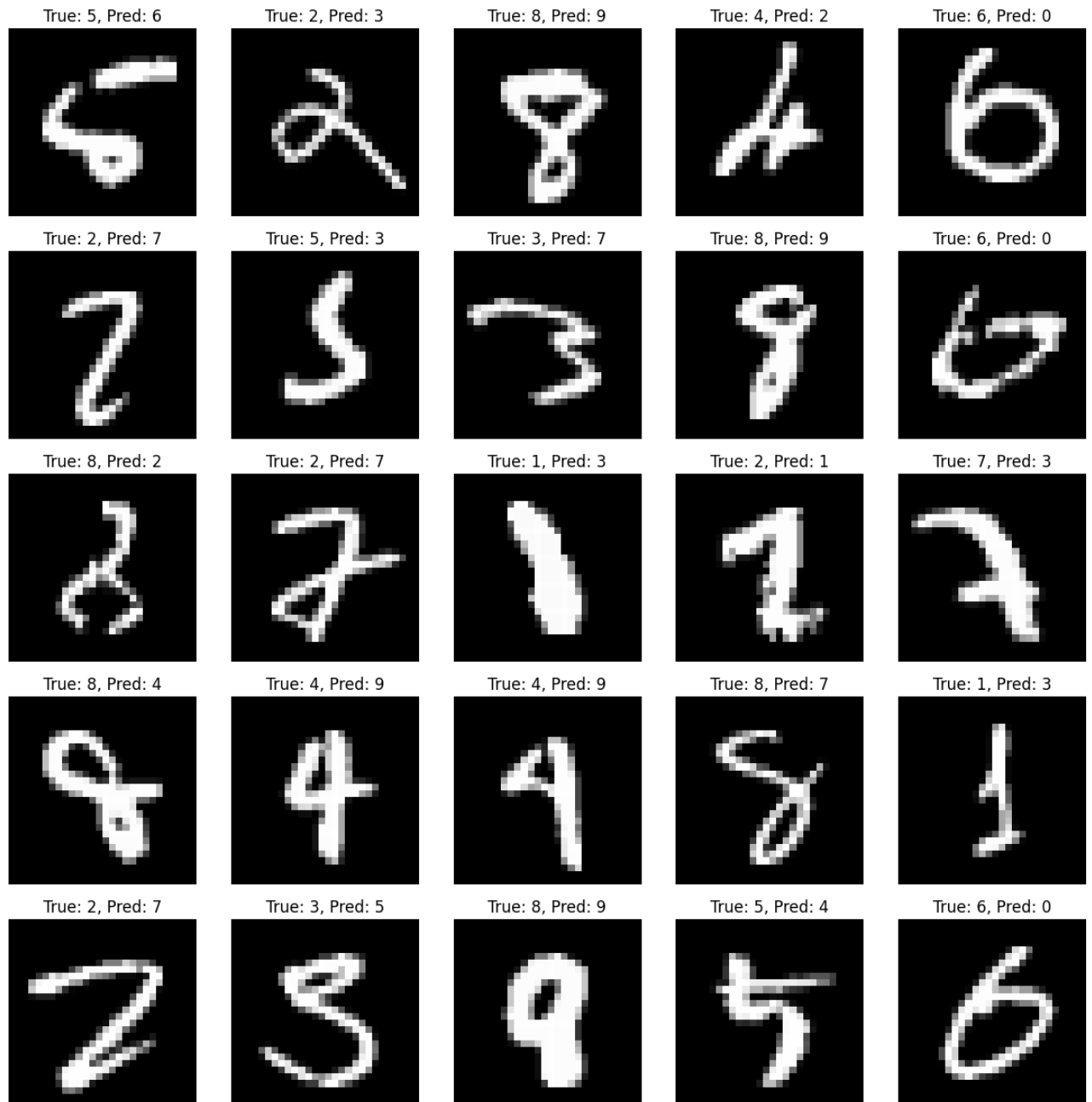


Рис.17 Неправильно класифіковані значення в моделі послідовних шарів

Висновок: у цій лабораторній роботі я навчилася тренувати прості та послідовні нейронні мережі й налаштовувати їх. Під час виконання роботи помітила, що точність моделі залежить від кількості епох (хоча при великій їх кількості може виникнути перенавчання), а також від обраного оптимізатора та кількості нейронів.