**Міністерство освіти і науки України**

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

**Кафедра САПР**



**Лабораторна робота №1**

з дисципліни: “Застосування систем штучного інтелекту у технологічних рішеннях”

**на тему:**

“ Огляд інструментів для роботи з AI: налаштування середовища (Python, TensorFlow, PyTorch, Google Colab). Створення базового класифікатора рукописних цифр MNIST для прикладу в навчанні”

**Варіант №3**

**Виконав**:

ст. групи ПП-44

Верещак Б. О.

**Прийняв**:

доц. Левкович М.В.

**Львів - 2025**

**Мета роботи**

Налаштувати робоче середовище для розробки моделей ШІ (Python + бібліотеки, Google Colab) та створити базовий класифікатор рукописних цифр MNIST як референтний приклад, що демонструє повний цикл: підготовка даних → побудова простої нейромережі (MLP або CNN) у TensorFlow/Keras (або PyTorch) → навчання й оцінювання якості (accuracy, confusion matrix) → збереження моделі та відтворюваний запуск.

**Індивідуальне завдання**

Для виконання роботи використовуйте надані приклади коду. Порівняйте отримане значення точності для моделі CNN з точністю простої повнозв’язної мережі. Залежно від вашого номера у списку групи, виконайте додаткові індивідуальні вимоги згідно з таблицею:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варіанта | Batch size | Кількість епох | Оптимізатор | Додаткове завдання |
| 3 | 128 | 15 | RMSprop | Побудувати confusion matrix |

**Хід роботи**

**1. Налаштування середовища та завантаження даних**

Перед початком виконання лабораторної роботи потрібно підготувати середовище розробки, я обрав google collab, і імпортувати потрібні модулі та датасети, в нашому випадку MNIST для класифікації рукописних цифр.

import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

import random

mnist = tf.keras.datasets.mnist

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

train\_images = (np.expand\_dims(train\_images, axis= 1)/255.).astype(np.float)

train\_labels = (train\_labels).astype(np.int64)

test\_images = (np.expand\_dims(test\_images, axis=-1)/255.).astype(np.float32)

test\_labels = (test\_labels).astype(np.int64)

**2. Побудова повнозв’язної нейронної мережі**

На цьому кроці нам потрібно створити саму повнозв’язну нейронну мережу. Спочатку ініціалізуємо саму модель

def build\_fc\_model():

  fc\_model = tf.keras.Sequential([

    tf.keras.layers.Flatten(input\_shape=(28, 28, 1)),

    tf.keras.layers.Dense(128, activation="relu"),

    tf.keras.layers.Dense(10, activation="softmax")

  ])

  return fc\_model

model = build\_fc\_model()

Тепер потрібно скомпілювати модель та надати її дані для навчання. Згідно варіанту використовуємо оптимізатор RMSprop разом з розміром пакета (batch\_size) 128 та кількістю епох (epochs) 15.

model.compile(

    optimizer=tf.keras.optimizers.RMSprop(learning\_rate=0.1),

    loss="sparse\_categorical\_crossentropy",

    metrics=["accuracy"]

)

BATCH\_SIZE = 128

EPOCHS = 15

history = model.fit(

 train\_images,train\_labels,

 batch\_size=BATCH\_SIZE,

 epochs=EPOCHS

)

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

print("Точність на навчальних даних:", history.history["accuracy"][-1], "\n Значення втрати", history.history['loss'][-1])

print("Точність на тестових даних:", test\_acc, "\n Значення втрати", test\_loss)

В результаті ми отримуємо точність і похибку моделі. З результатів видно що великої різниці між точністю на навчальних, чи тестових даних становить близько 1.1%, але видно що значення втрати більше з тестовими даними.

Точність на навчальних даних: 0.9927833080291748

Значення втрати: 0.03181539848446846

Точність на тестових даних: 0.979200005531311

Значення втрати: 0.20555131137371063

**3. Побудова згорткової нейронної мережі**

З результатів минулої мережі видно, що повнозв’язна мережа не ідеально справляються з розпізнаванням цифр, тому переважно використовують згорткові нейронні мереж, які довели свою ефективність у розв’язанні завдань комп’ютерного зору. Спочатку ініціалізуємо архітектуру

def build\_cnn\_model():

  cnn\_model = tf.keras.Sequential([

    tf.keras.layers.Conv2D(filters=24, kernel\_size=(3,3), activation=tf.nn.relu),

    tf.keras.layers.MaxPool2D(pool\_size=(2,2)),

    tf.keras.layers.Conv2D(filters=36, kernel\_size=(3,3), activation=tf.nn.relu),

    tf.keras.layers.MaxPool2D(pool\_size=(2,2)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation=tf.nn.relu),

    tf.keras.layers.Dense(10, activation=tf.nn.softmax)

  ])

  return cnn\_model

cnn\_model = build\_cnn\_model()

Тепер давайте скомпілюємо модель за нашим варіантом

cnn\_model.compile(

  optimizer=tf.keras.optimizers.RMSprop(learning\_rate=1e-3),

  loss="sparse\_categorical\_crossentropy",

  metrics=["accuracy"]

)

history\_cnn = cnn\_model.fit(

  train\_images,

  train\_labels,

  batch\_size=BATCH\_SIZE,

  epochs=EPOCHS

)

test\_loss, test\_acc = cnn\_model.evaluate(test\_images, test\_labels, verbose=2)

print("Точність на навчальних даних:", history\_cnn.history["accuracy"][-1], "\nЗначення втрати:", history\_cnn.history['loss'][-1])

print("Точність на тестових даних:", test\_acc, "\nЗначення втрати:", test\_loss)

Тут результати на навчальних даних практично ідеальні, так як модель розроблена спеціально під такий тип розпізнавання

Точність на навчальних даних: 0.9990166425704956

Значення втрати: 0.003010542131960392

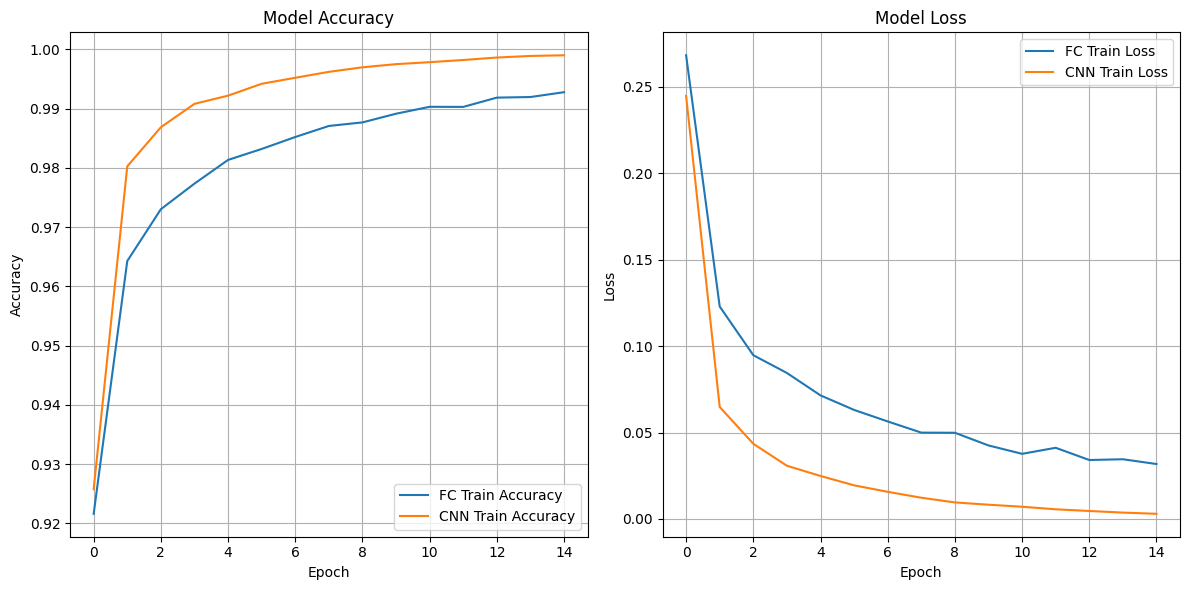
Точність на тестових даних: 0.9902999997138977

Значення втрати: 0.043695155531167984

**4. Порівняння моделей**

Для кращого порівняння використаних мереж, давайте спробуємо порівняти результати між ними

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Модель | Точність навч. | Втрата навч. | Точність тест. | Втрата тест. | Параметри навчання |
| FCN | 99.28% | 3.18% | 97.92% | 20.56% | batch\_size=128, epochs=15, RMSprop |
| CNN | 99.90% | 0.30% | 99.03% | 4.37% | batch\_size=128, epochs=15, RMSprop |
| Результати | +0.62% | -2.88% | +1.05 | -16.19 | CNN ефективніша |



Графіки навчання моделей показують: CNN швидше досягає високої точності, має менше перенавчання та згорткові шари краще виділяють просторові ознаки.

**5. Індивідуальне завдання. Confusion matrix**

Для виконання цього завдання я використовував готову бібліотеку з sklearn, щоб згенерувати та відобразити матрицю змішування. З результатів видно, що вертикально розташовані правильні мітки, а по горизонталі, ті які модель передбачила. Тому ті результати, що на діагоналі показують кількість правильно вгаданих цифр.

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

predictions\_fcnn = model.predict(test\_images)

predictions\_cnn = cnn\_model.predict(test\_images)

predicted\_labels\_fcnn = np.argmax(predictions\_fcnn, axis=1)

predicted\_labels\_cnn = np.argmax(predictions\_cnn, axis=1)

cm\_fcnn = confusion\_matrix(test\_labels, predicted\_labels\_fcnn)

cm\_cnn = confusion\_matrix(test\_labels, predicted\_labels\_cnn)

disp\_fcnn = ConfusionMatrixDisplay(confusion\_matrix=cm\_fcnn, display\_labels=range(10))

disp\_fcnn.plot(cmap=plt.cm.Blues, values\_format='d')

plt.title("Confusion Matrix for FCNN")

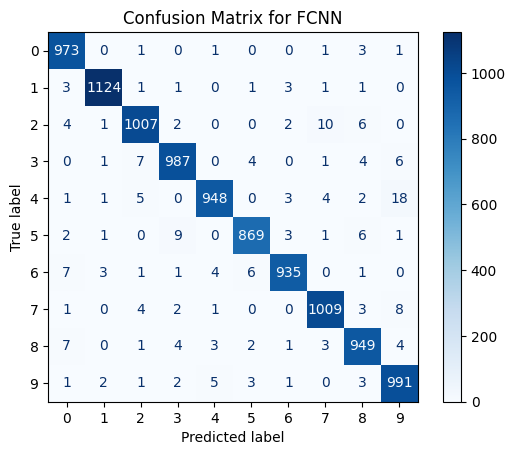
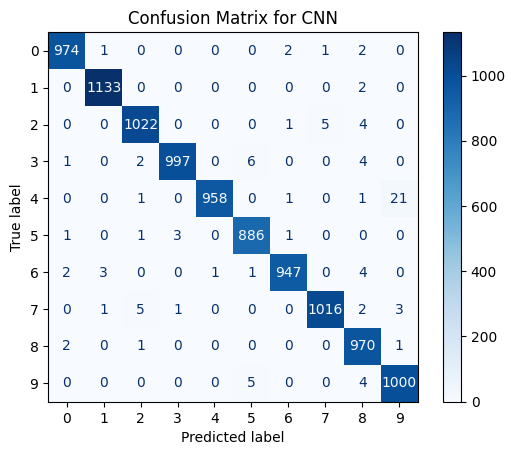
plt.show()

disp\_cnn = ConfusionMatrixDisplay(confusion\_matrix=cm\_cnn, display\_labels=range(10))

disp\_cnn.plot(cmap=plt.cm.Blues, values\_format='d')

plt.title("Confusion Matrix for CNN")

plt.show()

**Висновки**

У ході виконання лабораторної роботи було налаштовано робоче середовище TensorFlow/Keras та проведено порівняльний аналіз нейронних мереж: повнозв’язна мережа показала точність на тестових даних у 97.92%, тоді як згорткова досягла 99.02%, що демонструє її перевагу у 0.62% для задач комп’ютерного зору. Для варіанту 3 з параметрами batch size=128, epochs=15 та оптимізатором RMSprop було досягнуто ефективного навчання без значного перенавчання, а індивідуальне завдання з confusion matrix візуалізацією підтвердило, що основні помилки класифікації виникають між візуально подібними цифрами (4 і 9, 3 і 5), що є типовим для набору даних MNIST.