Міністерство освіти і науки України

Національний технічний університет "ХПІ"

кафедра "Інформатика та інтелектуальна власність"

**ЗВІТ**

до лабораторної роботи № 6

Тема : «Згорткова нейронна мережа (CNN)»

Варіант номер 18

з дисципліни "Основи штучного інтелекту"

Виконав:

студент групи КН-321В

Хома Д.М.

Перевірив:

Паржин Ю. В.

Харків 2023

**1 Код програми**

Код програми наведено на рис. 1.1.

from keras.datasets import mnist  
from keras.models import Sequential  
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D  
from keras.utils import to\_categorical  
from keras.models import load\_model  
import matplotlib  
matplotlib.use('TkAgg')  
import matplotlib.pyplot as plt  
  
# Завантаження даних  
(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  
x\_train = x\_train.reshape(60000,28,28,1) # Перетворення форми даних тренувального набору  
x\_test = x\_test.reshape(10000,28,28,1) # Перетворення форми даних тестового набору  
y\_train = to\_categorical(y\_train) # Конвертація міток класів тренувального набору  
y\_test = to\_categorical(y\_test) # Конвертація міток класів тестового набору  
  
  
model = Sequential() # Створення моделі Sequential для послідовного надбудовання шарів  
model.add(Conv2D(64, kernel\_size=3, activation='relu', input\_shape=(28,28,1))) # Додавання згорткового шару  
model.add(MaxPooling2D(pool\_size=(2, 2))) # Додавання шару MaxPooling для зменшення розмірності зображення  
  
model.add(Conv2D(32, kernel\_size=3, activation='relu')) # Додавання другого згорткового шару  
model.add(MaxPooling2D(pool\_size=(2, 2))) # Додавання другого шару MaxPooling для зменшення розмірності зображення  
model.add(Flatten()) # Створює шар, який перетворює 2D-дані у вектор.  
model.add(Dense(10, activation='softmax')) # Створює одношаровий перцептрон  
  
# Компіляція моделі  
model.compile(optimizer='sgd', loss='mean\_squared\_error', metrics=['accuracy'])   
  
# Навчання моделі на тренувальних даних  
history = model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=10) # Тренування моделі  
  
# Візуалізація результатів  
plt.plot(history.history['accuracy'])   
plt.plot(history.history['val\_accuracy'])   
plt.title('Model accuracy')   
plt.ylabel('Accuracy')   
plt.xlabel('Epoch')   
plt.legend(['Train', 'Test'], loc='upper left')   
plt.show()   
  
plt.plot(history.history['loss'])   
plt.plot(history.history['val\_loss'])   
plt.title('Model loss')   
plt.ylabel('Loss')   
plt.xlabel('Epoch')   
plt.legend(['Train', 'Test'], loc='upper left')   
plt.show()

Рисунок 1.1 – Код програми

Після кожного згорткового шару Conv2D додається шар MaxPooling2D, який допомагає зменшити розмірність отриманих зображень на рис. 1.2.

model.add(MaxPooling2D(pool\_size=(2, 2))) # Додавання шару MaxPooling для зменшення розмірності зображення

Рисунок 1.2 – Шар MaxPooling2D

Змінено оптимізатор на SGT та функцію помилок на mean\_squared\_error на рис. 1.3.

# Компіляція моделі  
model.compile(optimizer='sgd', loss='mean\_squared\_error', metrics=['accuracy'])

Рисунок 1.3 - Оптимізатор SGT

**2 Тестування на датасеті MNIST**

У завдані було прописано протестувати на 10, 50, 100 епохах навчання, але я обрав 10,15,20 епох тому, що час очікування навчання був дуже великим і по підрахункам 100 навчання зайняло би більше години при постійному піковому навантажені на CPU. Також я намагався навчати нейромережу на відеокарті, але я дізнався, що в нових версіях TensorFlow не підтримується використання GPU на Windows, але якщо відкатити TensorFlow з 2.14 до 2.10 версії – тоді ще можна спробувати, завантажив CUDA, cuDNN, завантажив нові драйвери, перевірив сумісності, додав шляхи CUDA та cuDNN до змінних системного середовища, але TensorFlow не бачить мої графічні процесори. Все таки це можливо зробити для Anaconda або під лінукс, в моєму випадку я навчав нейромережу на процесорі. Отже результати навчання наведені на рис. 2.1 – 2.9

**10 епох**

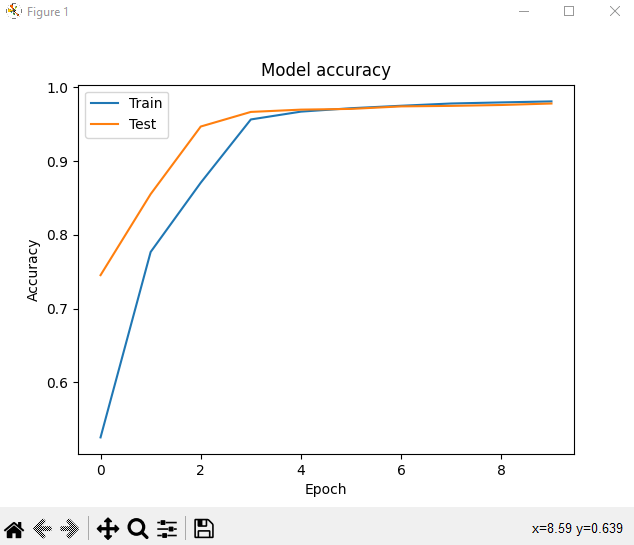


Рисунок 2.1 - Графік точності навчання 10 епох

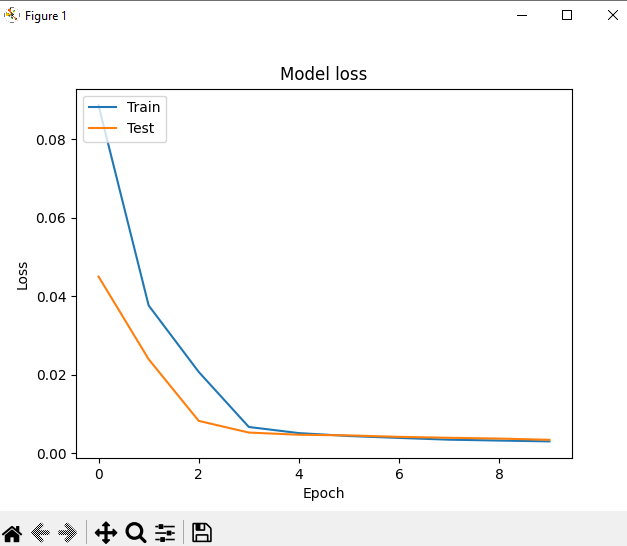


Рисунок 2.2 - Графік кількості втрат 10 епох

Результати роботи нейромережі на рис. 2.3.



Рисунок 2.3 – Результати роботи

Загальна точніть 98.1 %, втрати – 0.003.

**15 епох**

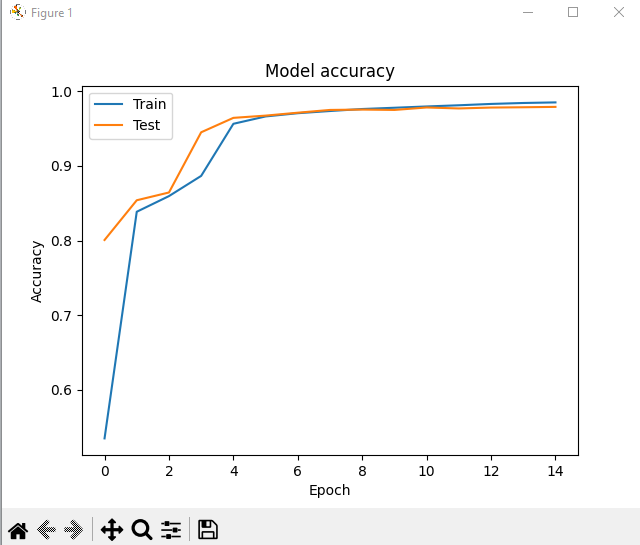


Рисунок 2.4 - Графік точності навчання 15 епох

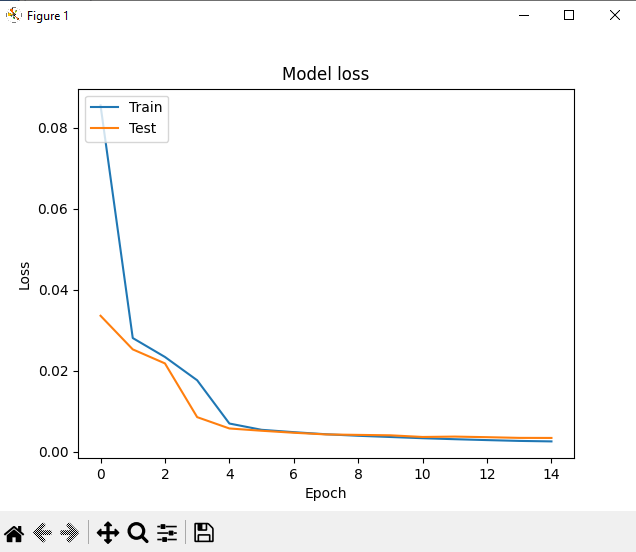


Рисунок 2.5 - Графік кількості втрат 15 епох

Результати роботи нейромережі на рис. 2.6.



Рисунок 2.6 – Результати роботи

Загальна точніть 98.49 %, втрати – 0.0025.

**20 епох**

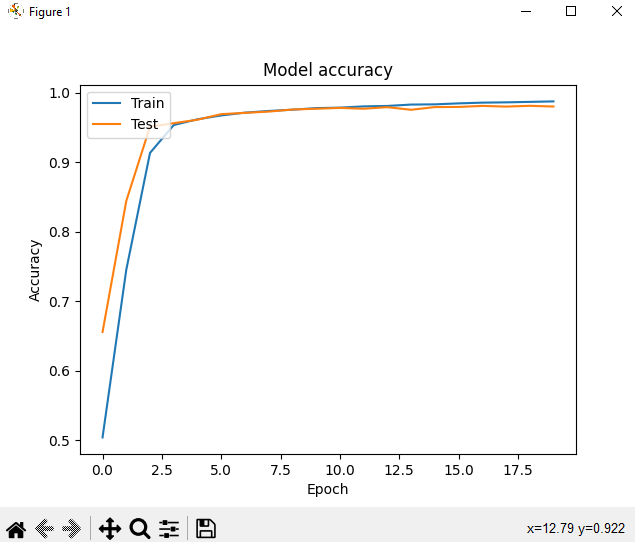


Рисунок 2.7 - Графік точності навчання 20 епох

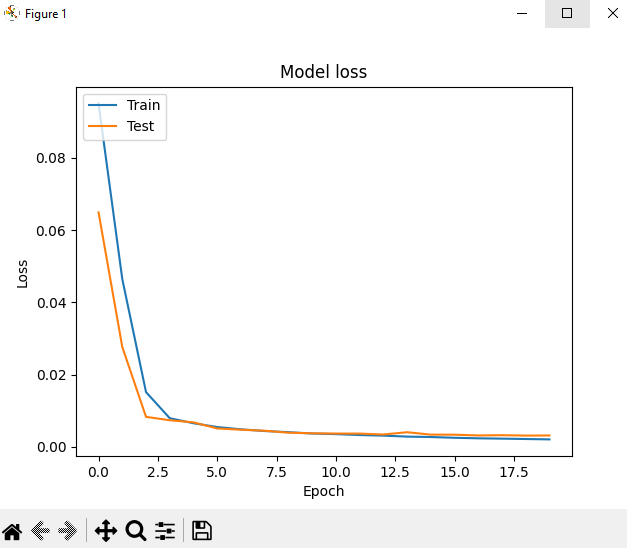


Рисунок 2.8 - Графік кількості втрат 20 епох

Результати роботи нейромережі на рис. 2.9.



Рисунок 2.9 – Результати роботи

Загальна точніть 98.79 %, втрати – 0.0021.

Отже зі збільшенням кількості епох точність збільшується а втрати зменльшуються на табл. 2.10.

Таблиця 2.10

|  |  |  |
| --- | --- | --- |
| Кількість епох | Точніть | Втрати |
| 10 епох | 98.1% | 0.003 |
| 15 епох | 98.49% | 0.0025 |
| 20 епох | 98.79% | 0.0021 |

Якщо порівнювати нейромережу без шару Макспулінгу, оптимізатор Adam, функція втрат: categorical\_crossentropy категоріальна перехрестна ентропія, отримаємо такі результати на рис. 2.11.

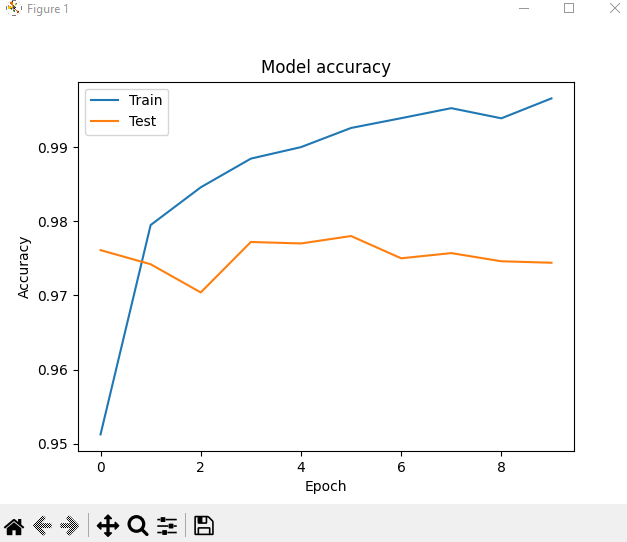


Рисунок 2.11 - Графік точності навчання 10 епох

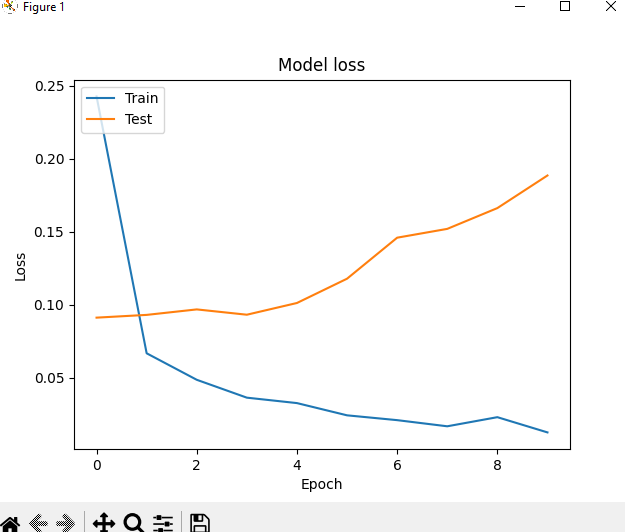


Рисунок 2.12 - Графік кількості втрат 10 епох

Результати роботи нейромережі на рис. 2.13.



Рисунок 2.13 – Результати навчання

Загальна точніть 99.66 %, втрати – 0.0128. Час навчання набагато більше ніж у нейромережі з Макспулінгом, оптимізатором SGT, та функцією втрат: середньоквадратична помилка Точність вище, але функція витрат є також важливим поняттям у навчанні, оскільки вона є мірою того, наскільки далеким є певний розв'язок від оптимального розв'язку задачі, яку потрібно розв'язати.

Var\_loss = 0.188. Величина дисперсії функції втрат (Var\_loss) вказує на розкид або різноманітність значень функції втрат під час навчання. Чим менше ця величина, тим більш стабільним було навчання моделі, наприклад як у переробленій нейромережі за завданням, див. табл. 2.10. Висока дисперсія може вказувати на те, що модель нестабільна під час навчання і може погіршувати якість передбачень на нових даних.

**6. Висновки**

Було додано шари пулінгу. Ця операція зменшує розмірність сформованих карт ознак. У даній архітектурі мережі вважається, що інформація про факт наявності шуканої ознаки важливіша за точне знання її координат, тому з кількох сусідніх нейронів карти ознак вибирається максимальний (з максимальною ознакою) і приймається за один нейрон ущільненої карти ознак меншої розмірності. За рахунок цієї операціїмережа стає інваріантнішою до масштабу вхідного зображення. Додавання шарів MaxPooling2D, може призвести до швидшого зменшення втрат і покращення швидкості збіжності моделі.

Змінено оптимізатор з Adam на SGD. Тривають довгі суперечки хто кращий, прочитавши декілька статей дійшов до висновку, що для різних задач підходять різні оптимізатори.

Використання стохастичного градієнтного спуску та середньоквадратичної помилки може призвести до іншого руху в просторі параметрів моделі під час навчання порівняно з оптимізатором 'adam' та функцією втрат 'categorical\_crossentropy'.

Зміна функції помилок може вплинути на швидкість збіжності моделі та її здатність до узагальнення.

Збільшення кількості епох може допомогти підвищити точність моделі. Проте, велика кількість епох може також призвести до перенавчання.