Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ДОСЛІДЖЕННЯ ТА ПОРІВНЯННЯ НЕЙРОННИХ МЕРЕЖ НА ПРИКЛАДІ РЕАЛІЗАЦІЇ ЗАДАЧІ РОЗПІЗНАВАННЯ РУКОПИСНИХ ЦИФР

Текстова частина до курсової роботи

за спеціальністю „Комп'ютерні науки" 122

Керівник курсової роботи

доцент

Ющенко Ю. О.

_____

"____" _____ 2021 р.

Виконав студент КН-3

Бельковець В. Ю.

"____" _____ 2021 р.

Київ 2021

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
доцент, к.ф.-м.н.
_____С. С. Гороховський
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту 3 року навчання БП «Комп'ютерні науки»

Бельковцю Владиславу Юрійовичу

на тему:

Дослідження та порівняння нейронних мереж на прикладі реалізації задачі розпізнавання рукописних цифр

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Анотація

Вступ

Опис предметної області

Реалізація моделей нейронних мереж

Багатошаровий персептрон

Згорткова нейронна мережа

Visual Geometry Group 16

Висновки

Список літератури

Дата видачі „___” _____ 2021 р.

Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітки
1.	Отримання завдання на курсову роботу.	11.10.2020	
2.	Огляд технічної літератури за темою роботи.	12.01.2021	
3.	Аналіз актуальності теми	15.01.2021	
3.	Дослідження теоретичної частини	29.02.2021	
4.	Написання практичної частини	29.02.2021	
5.	Попередня демонстрація проекту науковому керівнику	15.03.2021	
6.	Кінцеві правки практичної роботи	20.04.2021	
7.	Написання теоретичної частини	30.04.2021	
8.	Корегування роботи згідно з результатами перевірки роботи науковим керівником.	05.05.2021	
8.	Створення презентації	09.05.2021	
10.	Подання роботи на кафедру для перевірки на плагіат	17.05.2021	
11.	Захист курсової роботи	18 – 29.05.2020	

ЗМІСТ

Перелік умовних позначень

CNN — Згорткові нейронні мережі (Convolutional Neural Networks)

VGG16 — Visual Geometry Group 16

ReLU — випрямлений лінійний вузол (Rectified Linear unit)

MNIST — об'ємна база даних зразків рукописного написання цифр (Mixed National Institute of Standards and Technology)

RGB — адаптивна колірна модель (Red Green Blue)

GPU — графічний процесор (Graphics Processing Unit)

CPU — центральний процесор (Central Processing Unit)

ВСТУП

На початку 60-х р В 1943 році В. Маккалох та В.Піттс створили першу обчислювальну модель для нейронної мережі, яка має назву “порогова логіка”. Це стало початком розвитку нейронних мереж. Вже в кінці 1940-х Дональд Гебб створив таку гіпотезу навчання, яка потім стала відома як Геббове навчання. Вона описувала пристосування нейронів мозку під час процесу навчання. Вже через декілька років Розенблат створив перцептрон, що слугував для розпізнавання образів.

Тоді люди тільки починали чути про штучний інтелект. В той час під керівництвом О. Г. Івахненка була зібрана та випробувана перша версія перцептрона – машина «Альфа». Один із колег Івахненка, Михайло Шлезінгер, тоді займався симуляцією нейронних мереж на цифровий електронній машині «Київ». Це все і стало початком поступового розвитку штучного інтелекту.

По справжньому широку популярність нейронні мережі отримали дещо пізніше. Пов'язано це з появленням обчислювальних машин більшої потужності, котрих було достатньо для роботи з штучними нейронними мережами. Через збільшення потужності обчислювальних машин кіл-сть задач для нейронних мереж все збільшувалася.

На сьогодні нейронні мережі широко використовується в багатьох сферах людської діяльності. Воно використовується у воєнній сфері, навчальній, медичні та, наприклад, у розпізнаванні лінгвістичних образів та символів. Для таких та подібних задач існує велика кількість підходів до побудови архітектури, але досі не існує такої моделі, котра була б найкращої по всім аспектам, що підтверджує актуальність даної роботи.

Ця робота присвячена аналізу основних понять нейронних мереж і глибинного навчання. Необхідно дослідити принципи розпізнавання рукописних цифр, що пов'язані з певною архітектурою нейронної мережі, та є максимально актуальним у наш час технологічного прогресу через велику кількість нових компаній, котрі починають використовувати нейронні мережі у свої продуктах. Виконати побудову, навчання і тестування моделей, після чого порівняти їх по точності розпізнавання задля оптимізації їх використання.

Основна мета цієї роботи - вивчити основи роботи з нейронними

мережами і бібліотекою Keras для створення, навчання і тестування моделей мереж за допомогою мови програмування Python та аналіз і порівняння різних моделей навчання на прикладі розпізнавання рукописних чисел.

Робота складається з двох розділів. В першому - дослідження трьох найпоширеніших типів нейронних мереж задля розуміння задачі та подальшої роботи.

В другому - реалізація та аналіз нейронних мереж на прикладі розпізнавання рукописних чисел. Після чого їх порівняння, базуючись на аналізі різних аспектів навчання.

1. Опис предметної області

1.1. Машинне навчання

Машинне навчання - це напрямок штучного інтелекту, що пов’язаний з побудовою аналітичних моделей для виявлення певних закономірностей. Ключова ланка машинного навчання – певний набор даних, завдяки якому і відбувається побудова певних передбачень стосовно нового вхідного набору даних.

Зазвичай, виділяють два види машинного навчання:

Навчання з вчителем

В такому випадку алгоритм працює з даними, що містять не тільки змінні, а й значення, яке має видавати модель після навчання. Помилкою навчання в цьому випадку є різниця між цільовим і фактичним результатом моделі. Ця помилка мінімізується в процесі навчання та слугує «вчителем». Зазвичай використовується для задач регресії та класифікації.

Навчання без вчителя

В цьому випадку замість помилки навчання використовується інформація про поточний стан параметрів нашої моделі та прикладів вхідної множини. Тобто нам потрібно шукати залежності між об'єктами вхідних даних. Зазвичай використовується для задач кластеризації.

1.2. Штучна нейронна мережа

Штучна нейронна мережа - це обчислювальна модель, яка навчається, комбінуючи штучні нейрони для розуміння вхідних даних, і прогнозує очікуємий результат. Однією з головних особливостей нейронних мереж є – вміння самостійно навчатися та адаптуватися під прецеденти, тобто під минулий досвід, неухильно та поступово зменшуючи кількість та вагомість помилок.

Концепція нейронних мереж моделює принципи взаємодії нейронів головного мозку, імітує структуру нервової системи людини. Вона являє собою певну кількість нейронів, що є обчислювальним елементом, які відносяться до певного шару мережі. Після цього вхідні дані проходять обробку через всі шари цієї мережі. Під час цієї обробки параметри обчислювальних елементів можуть змінюватися залежно від результатів, які були отримані на минулих вхідних даних.

Кожен нейрон пов'язаний один з одним і зовнішнім середовищем за допомогою зв'язків, кожен з яких має певний коефіцієнт, на який множиться значення вхідних даних, які проходять через нього.

Нейронні мережі являють собою моделі, які засновані на машинному навчанні, тобто вони набувають всі властивості в процесі навчання.

Навчання має під собою знаходження певних коефіцієнтів зв’зяків між нейронами. Нейронна мережа здатна виявляти складні залежності між вхідними даними, які можливо ще невідомі людям, та певними результуючими данимищо значить, що вона здатна повернути правильний результат на вхідних даних, котрі раніше були відсутні.

Різні архітектури нейронних мереж, такі як багатошаровий персептрон (англ. Multilayer Perceptron, MLP) і згорткова нейронна мережа (англ. Convolutional Neural Network, CNN), а також відома архітектура багатошарової згорткової нейронної мережі Visual Geometry Group 16, використовуючи добре відому базу даних цифрових зображень рукописних цифр MNIST, допоможуть знайти різницю в точності методів розпізнавання, що лежать в їх основі.

1.3. **Багатошаровий персептрон**

Багатошаровий персептрон - це архітектура штучної нейронної мережі з прямим зв'язком (Рис. 1). Він відображає вхідні дані у відповідний вихідний набір. Кожна функція активації шару зазвичай використовується для створення нелінійного відображення між входом і виходом в діапазоні від 0 до 1. Традиційно багатошаровий персептрон складається з двох-трьох шарів, розташованих один за одним.

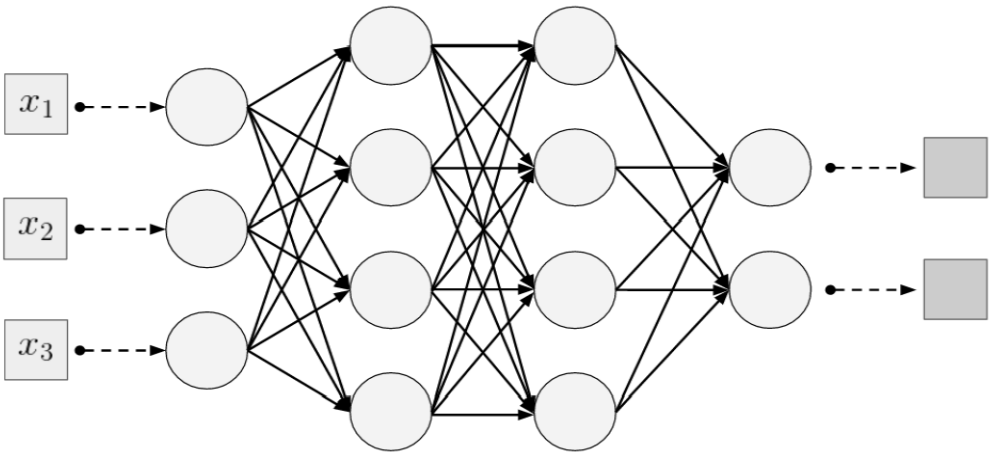

Рис. 1 - Приклад нейромережі з прямим зв'язком

В основному процес навчання складається з трьох кроків. Перший крок - це прямий прохід, на якому вхідні дані відправляються далі до подальших шарів, за допомогою множення значень входів на ваги нейронів, підсумовування, додавання величини зсуву. На другому етапі після отримання прогнозованого результату він порівнюється з фактичним результатом, а потім розраховується величина помилки (втрати). На останньому етапі виконується так зване зворотне поширення помилки по шарам. Воно передбачає два проходи по всім шарам мережі: прямий і зворотній. При прямому проході вхідні дані подаються на вхідний прошарок нейронної мережі, після чого поширюється по мережі від шару до шару. Потім формуються результати, котрі і є реакцією на певний образ. Зворотній прохід - це поширення сигналів помилки від виходів мережі до її входів, в напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи. Після закінчення роботи цього методу відбувається оновлення ваг. У багатошаровому персептрона вихід вузла визначається певною функцією активації. Це функція, аргументом якої є виважена сума входів штучного нейрона, а значенням - вихід нейрона.

В багатошаровому персептроні вихід визначається наступними функціями:

Сигмоїдальна функція

$$A = \frac{1}{1 + e^{-x}}$$

Вона прагне привести значення до однієї зі сторін кривої. Така поведінка дозволяє знаходити чіткі межі при прогнозі. Графік надано на Рис. 2

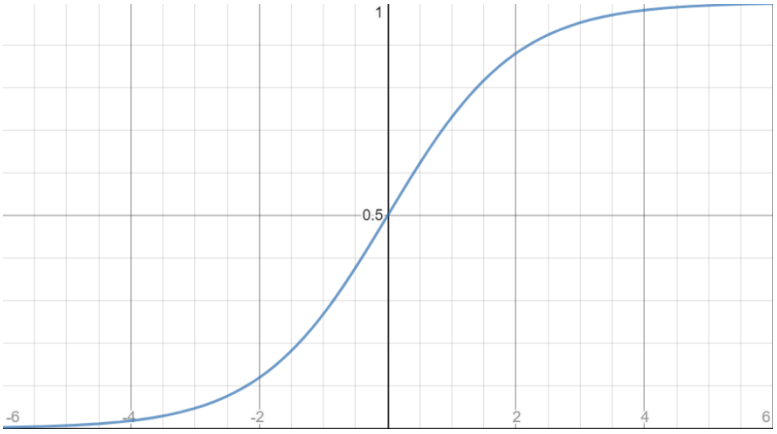

Рис. 2 – Графік сигмоїдальної функції

Гіперболічний тангенс

$$A = \frac{2}{1 + e^{-2x}} - 1$$

Гіперболічний тангенс дуже схожий на сігмоїду. Він нелінійний, добре підходить для комбінації шарів. Діапазон значень функції (- 1, 1). Однак варто зазначити, що градієнт тангенсіальної функції більше, ніж у сігмоїдальної (похідна крутіше). Графік надано на Рис. 3

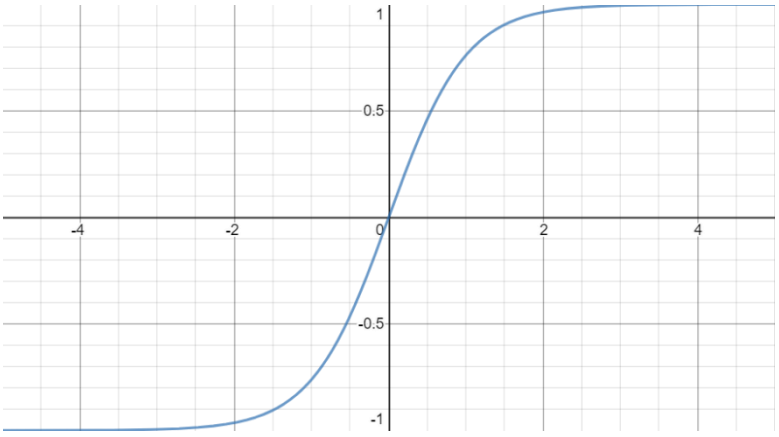

Рис. 3 – Графік тангенсіальної функції

ReLu (англ. Rectified Linear)

$A = \max(0, x)$

ReLu повертає значення x, якщо x позитивно, і 0 в іншому випадку. ReLu менш вимогливо до обчислювальних ресурсів, ніж гіперболічний тангенс або сигмоїда, так як виробляє більш прості математичні операції. Тому має сенс використовувати ReLu при створенні глибоких нейронних мереж.

Графік надано на Рис. 4

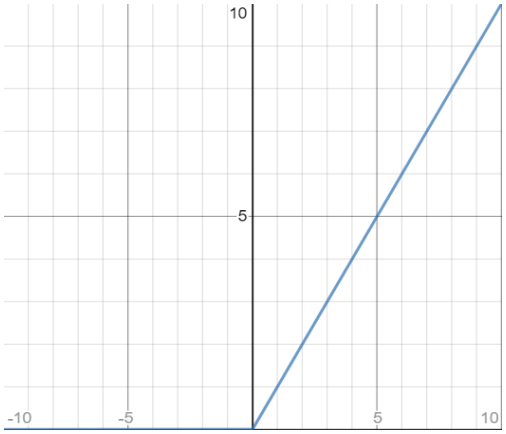

Рис. 4 – Графік ReLu функції

Багатошаровий персептрон як і раніше є гарним рішенням для більшості завдань, які не можна достатньо прийнятно вирішити нейронною мережею з маленькою кількістю шарів. Більш сучасні методи глибокого навчання значно поліпшили стан справ в області розпізнавання мовлення, візуального розпізнавання і виявлення об'єктів, а також у багатьох інших областях. Термін «глибинне навчання» відноситься до нейронних мереж з більш ніж одним прихованим шаром.

Глибина нейронної мережі дозволяє їй будувати ієрархію ознак зі зростаючою абстракцією, при цьому кожен наступний шар діє як фільтр для все більш і більш складних ознак, які об'єднують ознаки попереднього рівня. Ця ієрархія ознак і фільтрів, які моделюють значення даних, створюються автоматично, коли глибинні мережі навчаються відновлювати невідомі залежності. Оскільки вони можуть працювати з невивченими даними, які формують більшість даних в світі, глибинні мережі можуть виявитися більш

точними, ніж традиційні алгоритми машинного навчання, які не здатні обробляти такі дані.

1.4. **Згорткові нейронні мережі**

Згорткові нейронні мережі (CNN) - це клас штучних нейронних мереж, які є більш вдосконаленими моделями в порівнянні з багатошаровим персептроном. CNN об'єднує згорткові шари і так звані шари пулінгу, котрі використовуються для зменшення розмірності зображення, з повнозв'язними шарами, використовуваними в багатошаровому персептроні. Шари згортки і пулінгу необхідні для отримання ознак з вхідних даних, а повнозв'язні шари використовуються для перетворення витягнутих і оброблених ознак у вихідні сигнали мережі. Згортковий шар виконує математичну операцію, звану згорткою. В даному наборі даних MNIST всі цифрові зображення зберігаються в двовимірному форматі. Двовимірна матриця меншого розміру, звана в термінах згорткових нейромереж ядром, або фільтром, використовується для виконання операції згортки з кожним нейроном згорткового шару. По суті, згортка є не що інше, як скалярний добуток між ядром і вхідним тензором. Доповнюючи нулі в масивах зображень іноді необхідні для здійснення згортки, щоб зберегти пропорції в розмірах фільтра і тензора. Шари пулінгу дозволяють зменшити розмірність зображення і в той же час використовувати меншу кількість параметрів.

Так само, як і в архітектурі багатошарового персептрону, в CNN використовується алгоритм зворотного поширення помилки для вивчення просторової ієрархії ознак і коригування ваг нейронів. Як правило, для навчання моделі CNN потрібне використання графічних процесорів через велику кількість параметрів, що налаштовуються.

Згорткові нейронні мережі завжди були основою глибинного навчання. Перші CNN використовувалися ще в 20 ст. Тепер глибинні згорткові нейронні мережі вважаються одним з найбільших досягнень в задачах класифікації зображень. На рис 5 наведено простий приклад архітектури згорткових

нейронних мереж, призначеної для завдання класифікації цифр на зображеннях. Для цього набору даних є десять класів, таким чином, загальне число вихідних сигналів мережі дорівнює десяти.

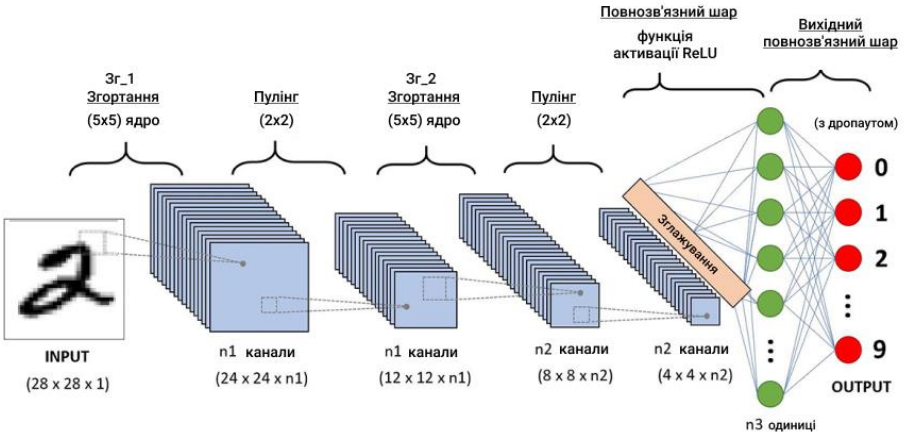

Рис. 5 - Приклад архітектури згорткової нейронної мережі

У мережі з Рис. 5 використовуються череговані пари шарів згортки і пулінгу, за якими слідує повнозв'язний шар з функцією активації ReLU і вихідний повнозв'язний шар. Використання функції ReLU дозволяє знизити обчислювальне навантаження при активації нейронів, і виконати згладжування шару (знизити розмірність багатовимірного масиву) з більш якісним наближенням.

При подачі на вихідний шар мережі, що виконує передбачення, використовується згладжування шару і його подальше зменшення(редукція) до десяти вихідних сигналів. Для зниження ймовірності перенавчання мережі використовується метод регуляризації, відомий як дропаут. Суть методу полягає в отриманні підмереж з вихідної мережі, і оновленні всіх ваг мережі в рамках кожної з виділених підмереж. Всім нейронам присвоюється ймовірність включення в ту чи іншу підмережу, яка називається коефіцієнтом дропаутів. Грубо кажучи, метод дропаутів усереднює результати підмережі.

1.5. **Архітектура Visual Geometry Group 16**

Архітектура Visual Geometry Group 16 (VGG16) - це архітектура згорткової нейронної мережі, названа на честь створившої її Оксфордської групи - Visual Geometry Group. Це дуже глибока згорткова нейронна мережа, актуальна навіть сьогодні, незважаючи на те що недавні досягнення, пов'язані з появою таких нових архітектур, як Inception і ResNet, дозволяють перевершити отримані нею результати для більшості задач.

Вхідний шар мережі являє собою зображення RGB фіксованого розміру 224 на 224 пікселів. Зображення проходить через послідовність згортальних шарів, в яких використовуються фільтри розміру 3 на 3 (в одній з конфігурацій мережі також використовуються фільтри згортки 1 на 1), які можна розглядати як лінійне перетворення вхідних каналів (з подальшою нелінійністю). Крок згортки дорівнює одиниці; пулінг здійснюється п'ятьма послідовними шарами з максимізацією, які слідують за деякими згортковими шарами (не для всіх згортальних шарів використовується пулінг). Пулінг виконується в сітці з кроком 2; фільтр має розмір 2 на 2.

Три повнозв'язних рівня слідують за стеком згорткових шарів (які мають різну глибину в різних архітектурах): перші два шари мають 4096 нейронів на кожному, третій виконує класифікацію і містить 1000 каналів. Останній шар - це шар Softmax (функція активації, яка узагальнює логістичну функцію). Конфігурація повнозв'язних шарів однакова у всіх мережах.

Всі приховані шари працюють з функцією активації ReLU.

1.6. **Постановка задачі**

В рамках роботи необхідно навчити і протестувати класифікатори на основі розглянутих раніше архітектур нейронних мереж, призначені для ідентифікації рукописних цифр на зображення. Набір даних містить набір MNIST. Він містить написані від руки зображення арабських цифр (0 - 9) розміром 28 на 28 пікселів. Кожна цифра на зображенні розташовується в центрі квадрата і має розмір приблизно 20 на 20 пікселів. Набір MNIST був отриманий з бази даних NIST; зображення піддалися нормалізації,

згладжуванню і іншим методам попередньої обробки, в тому числі приведення до напівтонової режиму (відтінки сірого).

Всього в наборі даних 70000 зображень. З них 60000 складають навчальну вибірку, а 10000 - тестову. Всі цифри наборі представлені з однаковою частотою. Приклади використовуваних зображень з набору даних наведені на Рис. 6. Можна використовувати як, багатошаровий персептрон , так і CNN для класифікації зображень. Багатошаровий персептрон приймає вектор в якості одиниці вхідних даних, а CNN - тензор (багатовимірний масив). Паралельна робота по створенню моделі багатошарового персептрону і двох моделей CNN дозволить правильно налаштувати різні параметри для обох алгоритмів, що лежать в їх основі. Зміна точності для обох алгоритмів може бути продемонстровано з використанням різних оптимізаторів і значень епох (ітерацій).

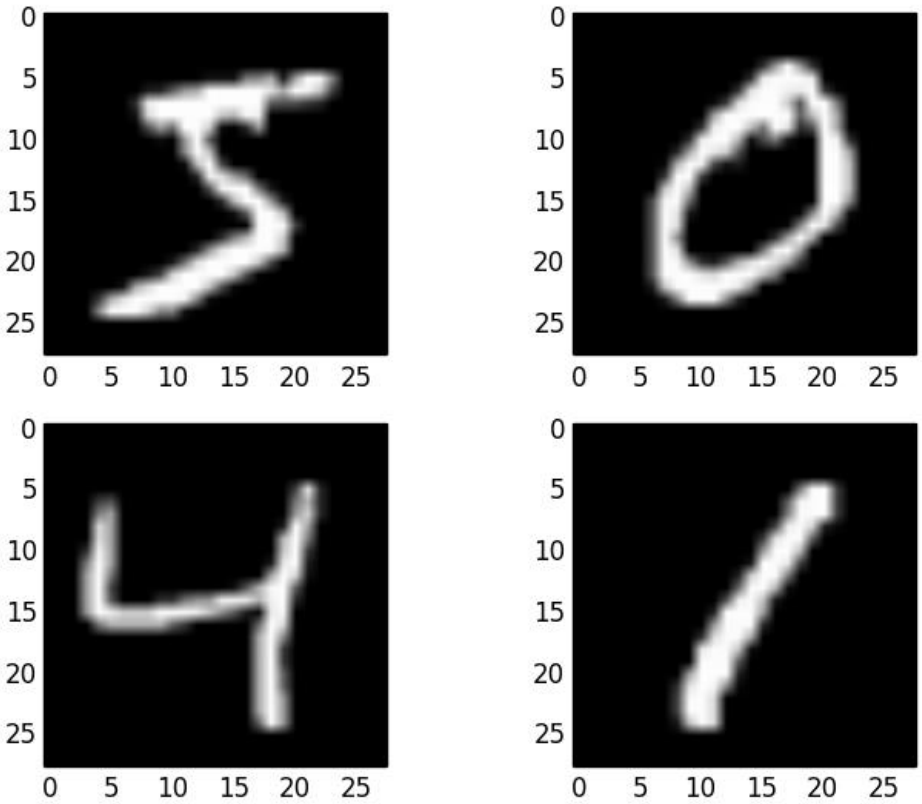

Рис. 6 - Приклади зображень цифр з набору даних MNIST

Програмний код для вирішення даного завдання зручно реалізувати в форматі * .ipynb-файлу на мові програмування Python 3 (в середовищі Jupyter Notebook).

Код, призначений для вирішення завдання, повинен виконати попередню обробку даних, побудувати, скомпілювати і навчити різні моделі нейронних мереж на навчальній вибірці (навчальні дані зображень), а також застосувати побудовану модель до тестової вибірки (тестовий набір даних).

2. Реалізація моделей нейронних мереж

2.1. Бібліотеки та інструменти

Мовою для реалізації моделей було обрано Python, оскільки вона є найпопулярнішою мовою для подібних задач та має велику кількість бібліотек, котрі прискорюють розробку.

Для реалізації моделей нейронних мереж використовується популярна бібліотека Tensorflow з фреймворком Keras, котрий є найшвидшим фреймворком для даних задач та має велику кількість документації, що прискорює написання коду (також призначеним для завантаження набору даних і попередньо побудованої моделі VGG16).

TensorFlow - це програмна бібліотека з відкритим вихідним кодом для чисельних розрахунків з використанням графів потоків даних. TensorFlow був створений і до сих пір підтримується командою Google Brain в дослідницькій організації Google Machine Intelligence для машинного і глибокого навчання. В даний час він випущений під ліцензією Apache 2.0 з відкритим вихідним кодом.

Keras - це бібліотека-оболонка Python, яка надає доступ до інших інструментів глибокого навчання, таким як TensorFlow, CNTK, Theano і т.д. Вона була розроблена з метою забезпечення швидкого маніпулювання інструментами глибокого навчання і випущена під ліцензією MIT. Keras поставляється як пакет для Python версій 2.7 - 3.9 і може безперешкодно працювати на GPU і CPU з урахуванням базових структур.

Для наявного набору даних з метою зниження навантаження на графічний процесор і скорочення часу навчання було взято всього 1024 випадкових примірника з навчальної вибірки і 256 примірника для тестового набору даних. Всі наведені бібліотеки показані на Рис. 4

Імпорт необхідних бібліотек

```
from keras import datasets, utils, layers, models, optimizers
import numpy as np
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
import keras
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras import models,layers , optimizers , datasets , utils
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

Рис. 4 – Код імпорту бібліотек

2.2. **Багатошаровий персептрон**

Повнозв'язні шари і кількість нейронів в кожному шарі є гіперпараметрами (задаються до початку навчання і не змінюються в процесі). Для отримання оптимальної точності класифікації в рамках експерименту змінювалося як кількість повнозв'язних шарів, так і кількість нейронів в кожному шарі. В якості можливих рішень також можна реалізувати семантичний пошук по нейронам і верствам з допомогою сітки. Зміни в точності також можна виявити, змінивши оптимізатор для моделі, який використовується для визначення різниці в результатах і справжніх значеннях на навчальній вибірці, після чого можна переходити до перебудови ваг в

нейронах мережі.

```
# На вхідному шарі моделі - 784 нейрона
inputs = layers.Input(shape=(784,))
# Модель багатошарового персептрона складається з набору декількох повнозв'язних шарів
x = layers.Dense(512, activation='relu')(inputs)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dense(64, activation='relu')(x)
x = layers.Dense(32, activation='relu')(x)
# На вихідному шарі - 10 нейронів (по числу цифр, тобто вихідних міток класів)
outputs = layers.Dense(10, activation='softmax')(x)

# Побудувати модель в keras і виконати її компіляцію
model = models.Model(inputs=inputs, outputs=outputs)


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

# Після компіляції модель повинна бути навчена розпізнавати рукописні номери:

''' Збереження моделі у файлі h5 (вага мережі зберігається у форматі HDF5) '''
from keras.callbacks import ModelCheckpoint
checkpointer = ModelCheckpoint(filepath="mlp.h5", verbose=1, save_best_only=True, save_weights_only=True)

'''
 Число епох (проходів по усіх екземплярах повчальної вибірки)  = 10","
 Розмір підвибірки (кількість зразків, узятих впродовж епохи)  = 32
'''
history = model.fit( xtraindata , ytraindata , batch_size = 32, epochs = 10, validation_data =( xtestdata , ytestdata ), callbac

# Оцінити навчену модель з точки зору продуктивності
score = model.evaluate ( xtestdata , ytestdata )
```

Рис. 7 – Код побудови, компіляції та навчання моделі

Використовуючи код, наведений на Рис. 7, можна досягти результату в 91% точності на тестовій вибірці (Рис. 8). Аналогічна точність була підтверджена в процесі проведення експерименту при багаторазовому прогоні даних. Кожен раз передбачувані ваги в першому шарі змінюються, тому спостерігаються деякі зміни в точності.

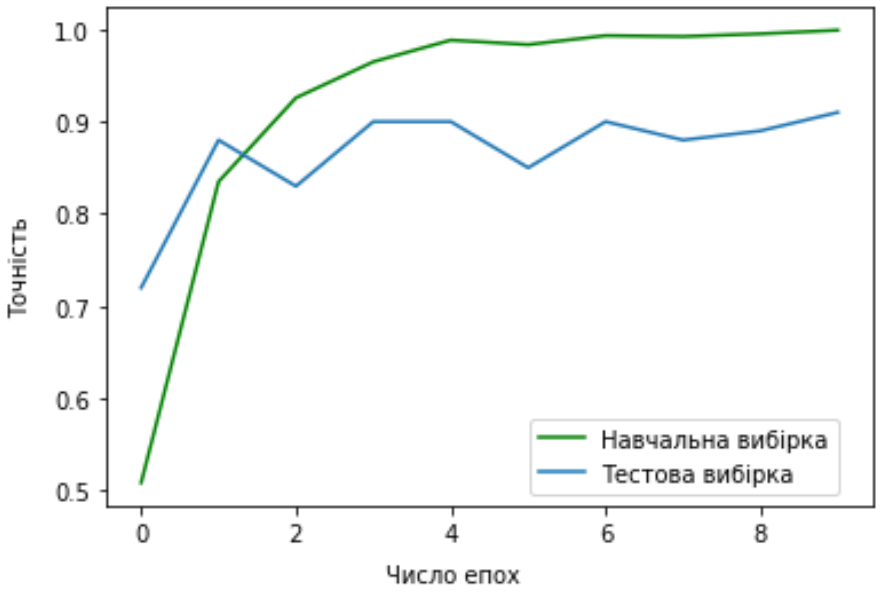

Рис. 8 - Графік точності розпізнавання цифр для моделі багатошарового персептрона

Спочатку було взято п'ять повнозв'язних шарів по 128 нейронів у кожному і використаний оптимізатор Nadam з функцією активації Softmax на виході, отриманий результат склав близько 88%. Було відзначено зміну точності при зміні кількості повнозв'язних прихованих шарів. Спочатку було передбачено шість повнозв'язних шарів, але в цій моделі за рахунок перенавчання точність знизилася. Також було виявлено, що модель дає точність 69% для оптимізатора Sgd. Що стосується кращих результатів експерименту, то найменша точність в 51% була отримана з оптимізатором Adadelta.

Краща точність склала 90,9% в порівнянні з іншими моделями. У цьому випадку було взято п'ять повнозв'язних шарів з 512, 256, 128, 64, 32 нейронами відповідно і використаний оптимізатор Nadam.

2.3. **Згорткова нейронна мережа**

Спочатку була побудована нейронна мережа, що використовує два згортальних шари з розміром ядра 3, одним шаром максимізації пулінгу з розміром фільтра 2 і двома повнозв'язними шарами. Використовуючи одні й ті ж дані, ті ж приховані шари, але різні оптимізатори, можна отримати різну точність. При навчанні з оптимізатором Adam точність на навчальній вибірці починається з 50% і для 12 епох закінчується майже на 100%. Точність навіть на тестових даних починається з 80%, а через 12 епох закінчується майже на 100%. При навчанні з оптимізатором Adadelta точність обох наборів даних починається приблизно з 1% і закінчується в межах 15 - 25% для 12 епох. Це набагато менше, ніж з оптимізатором Adam, але не означає, що Adadelta не є хорошим оптимізатором.

З ростом числа епох вдається домогтися 100% точності на навчальній вибірці. Можна також збільшити розмір набору навчальних даних, щоб досягти приблизно 100% точності. При використанні оптимізатора Adam більше шансів отримати як перенавчання, так і недонавчанні в порівнянні з

оптимізатором Adadelta. Так що у кожного оптимізатора є свої переваги і недоліки.

Налаштування гіперпараметрів - один з важливих аспектів навчання моделі. Гіперпараметри, які можуть бути розглянуті, - це епохи, підвибірки, додавання згортальних шарів, шари з максимізацією пулінгу, повнозв'язні шари. При додаванні повнозв'язну шару з 256 нейронами з функцією активації ReLU і коефіцієнтом дропаутів, що дорівнює 0,5, було виявлено, що точність падає. Також був протестований варіант налаштування, який починається з більшої кількості нейронів (256) в першому згортковому шарі, а потім зменшується до половини (128) у другому шарі згортки.

В результаті було обрано саме цей підхід, враховуючи ідею організації згорткової нейронної мережі, в якій кожен шар змішує елементи з попереднього шару і зменшує їх удвічі. Але точність знизилася в порівнянні з попередньою версією, де нейрони на кожному шарі були подвоєні. Це може бути пов'язано з тим, що вхідний сигнал низький у порівнянні з першим шаром.

Побудова моделі

```
# Вхідне зображення має розмір 28 на 28 пікселів; підрахувати число входів (кол.-під зображень)
inputs = layers.Input(shape=(28,28,1,))
# Додати вхідний згортковий шар з функцією активації ReLU і 32 фільтрами згортки розміру 3x3
cnn = Conv2D(32 , kernel_size=(3,3) , activation='relu', padding='valid',input_shape=(28,28,1))(inputs)
# Додати другий згортковий шар з 64 фільтрами
cnn = Conv2D(64 , kernel_size=(3,3) , activation='relu')(cnn)
# Додати шар ПУЛІНГ з фільтром 2x2
cnn = MaxPooling2D(pool_size=(2,2))(cnn)
# Додати шар виключення (дропаутів) для запобігання перенавчання
cnn = Dropout(0.25)(cnn)
# Додати шар, що зменшує розмірність (це буде необхідно для класифікації)
cnn = Flatten()(cnn)
# Додати повнозв'язний шар і ще один шар дропаутів
cnn = Dense(132,activation='relu')(cnn)
cnn = Dropout(0.5)(cnn)
outputs = Dense(10,activation='softmax')(cnn)
model = models.Model ( inputs = inputs ,outputs = outputs )
```

Компіляція и навчання моделі

```
# Скомпілювати і навчити модель - 12 епох, розмір підвибірки - 32
model.compile(loss='categorical_crossentropy', optimizer='nadam',metrics=['accuracy'])

''' Виконати збереження моделі в h5-файл (ваги мережі зберігаються в форматі HDF5) '''
from keras.callbacks import ModelCheckpoint
checkpointer = ModelCheckpoint(filepath="cnn.h5", verbose=1, save_best_only=True, save_weights_only=True)

history = model.fit(x_train,y_train,batch_size=32,epochs=12,verbose=1,validation_data=(x_test, y_test), callbacks=[checkpointer])

# Оцінити модель на навчальній і на тестовій вибірці
accuracy = model.evaluate(x_test,y_test,verbose=1,batch_size=32)
accuracy1 = model.evaluate(x_train,y_train,verbose=1,batch_size=32)
```

Рис 9 – Побудова, компіляція та навчання моделі CNN

Після проведення ряду експериментів була отримана та реалізована (Рис. 9) оптимальна структура мережі. Виявлено різницю в точності класифікації,

яка забезпечується багатошаровим персептроном і CNN для набору даних MNIST, який був навчений на 1024 примірниках навчальної вибірки і протестований на 100 зображеннях тестової вибірки. Що стосується експерименту, було виявлено, що CNN забезпечує меншу точність в 88% (Рис.10) в порівнянні з 91% для MLP.

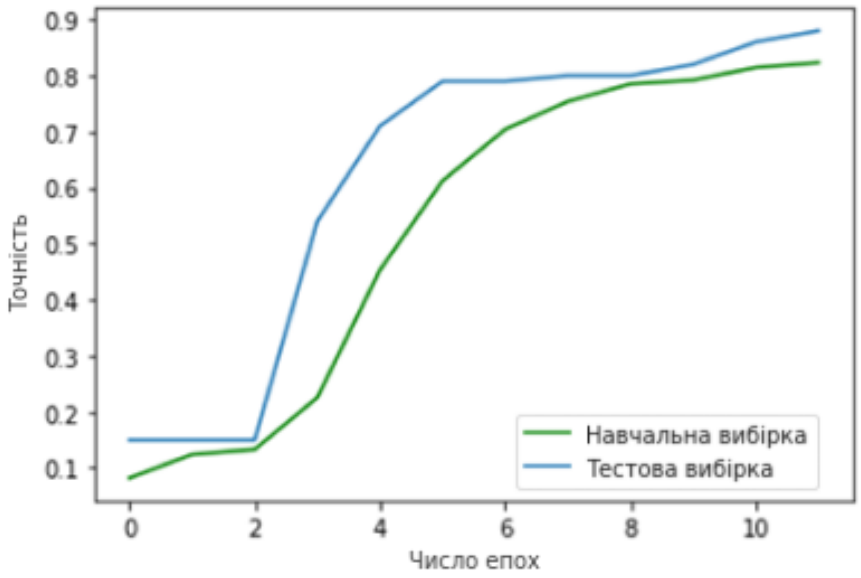

Рис 10 - Графік точності розпізнавання цифр для моделі згорткової нейронної мережі

2.4. Модифікована архітектура Visual Geometry Group 16

Для Visual Geometry Group 16 (VGG16) розмір зображень був змінений до необхідного розміру 224 на 224 пікселів, а самі зображення оброблені так, щоб мати три канали (RGB-формат, відповідно до вимог, що пред'являються до моделі VGG16). Спочатку був видалений шар для передбачення (з числом нейронів, рівним 1000) і були «заморожені» всі шари, крім повнозв'язних, для етапу навчання. «Замороження» означає, що стан шару не буде змінюватися в процесі навчання (ні при навчанні за допомогою методу fit (), ні при навчанні з будь-яким іншим методом, який покладається на зміну ваг із застосуванням градієнтного спуску). Потім був доданий останній повнозв'язний шар розміром в 10 нейронів, тобто на 10 цифр. Модель була скомпільована з оптимізатором Adam і категоріальної крос-ентропією в якості опції втрат.

В процесі експерименту використано 20 епох на етапі навчання мережі. Було відзначено, що з кожним кроком епохи точність моделі зростала, наближаючись до стовідсоткової в міру того, як кроки наближалися до завершення, код реалізації на Рис. 11.

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

Завантаження і попередня обробка даних

```
from keras.applications.vgg16 import preprocess_input
from keras.utils import to_categorical
from keras.preprocessing.image import img_to_array, array_to_img
import tensorflow as tf
from scipy import ndimage
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train[:1024, :, :]
y_train = y_train[:1024]
x_test = x_train[:256, :, :]
y_test = y_train[:256]
x_train = np.dstack([x_train] * 3)
x_test = np.dstack([x_test] * 3)
x_train = x_train.reshape(-1, 28,28,3)
x_test = x_test.reshape (-1, 28,28,3)
x_train = np.asarray([img_to_array(array_to_img(im, scale=False).resize((224,224))) for im in x_train])
x_test = np.asarray([img_to_array(array_to_img(im, scale=False).resize((224,224))) for im in x_test])
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
x_train_RGB = preprocess_input(x_train)
x_test_RGB = preprocess_input(x_test)
x_train_RGB.shape, x_test.shape, y_train.shape, y_test.shape
```

Навчання моделі

```
r = model.fit(x_train_RGB, y_train, validation_data=(x_test_RGB, y_test), epochs=20)
```

Рис. 11 – Завантаження та навчання моделі VGG16

У підсумку модель досягла точності в 100% як на навчальній, так і на тестовій вибірці (Рис. 12).

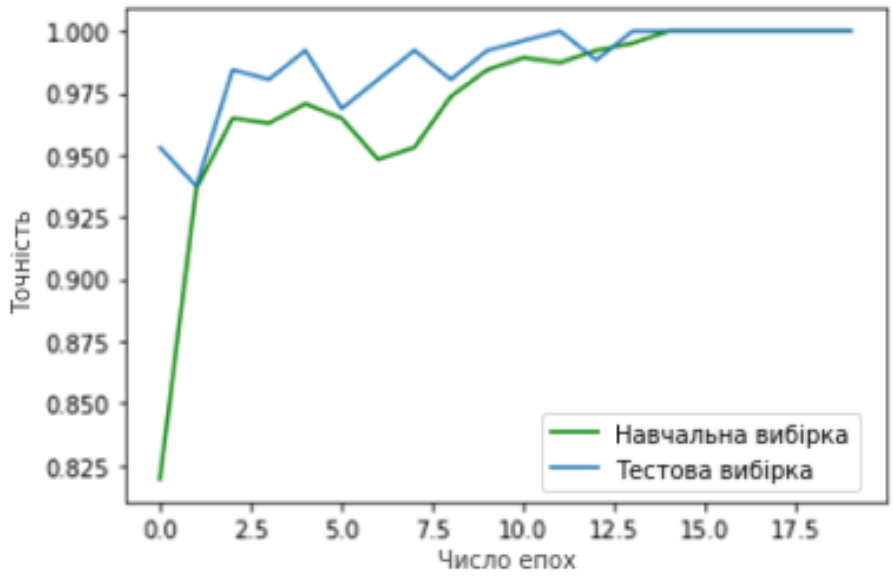

Рис. 12 - Графік точності розпізнавання цифр для моделі VGG16

2.5 Аналіз і перевірка отриманих результатів

Можна зробити висновок, що модифікована відповідно до вимог, поставлених для вирішення завдання, модель VGG16 має кращу точність і кращу криву навчання в порівнянні з CNN і багатошаровим персептроном, які були створені з нуля. В основному це пов'язано з навчанням тільки повнозв'язних шарів VGG16. Таким чином, трансферне навчання - дуже корисне явище, при якому можна повторно використовувати попередньо навчені моделі і налаштовувати їх для роботи з новими наборами даних без необхідності створювати нову модель або повторно навчати всі шари.

Корисність проведеного дослідження полягає в тому, що аналіз результатів, які показали беззаперечні переваги використання VGG16 та високу точність для задач розпізнавання рукописних чисел, можуть бути корисними для широкого класу задач таких як розпізнавання літер та інших геометричних фігур.

ВИСНОВОК

В ході виконання курсової роботи були досліджені теоретичні основи розпізнавання об'єктів на зображеннях, а також організація і принцип роботи нейронних мереж, які в останні роки домоглися великих успіхів при розпізнаванні об'єктів на зображеннях, особливо символів тексту.

Штучні нейронні мережі - це потужна технологія, яка успішно використовується в галузі задач так званого «комп'ютерного зору». В рамках даної роботи набір даних MNIST, що містить зображення рукописних цифр, був використаний для створення, навчання і тестування точності розпізнавання цифр на зображеннях таких моделей архітектури нейронних мереж, як багатошаровий персептрон, згорткова нейронна мережа і модифікована VGG16, різновид багатошарової згорткової нейронної мережі. Для моделі архітектури VGG16 необхідні вхідні зображення більшого розміру, тому наявні зображення збільшуються за допомогою інтерполяції. Результат показує, що VGG16 має точність вище, ніж багатошаровий персептрон і згорткова нейронна мережа, яка, в свою чергу, також має вищу точність, ніж багатошаровий персептрон. Збільшення навчальних даних з 1024 до 60000 зображень призводить до збільшення точності як багатошарового персептрона, так і згорткової нейронної мережі. ніж багатошаровий персептрон і згорткова нейронна мережа, яка, в свою чергу, також має вище точність, ніж багатошаровий персептрон.

Нейронні мережі та методи глибокого навчання перетворили додатки, які раніше вимагали експертних знань в області комп'ютерного та машинного зору, в інженерні завдання. Глубинне навчання дозволило перенести навантаження від творців додатків, які розробляють і пишуть сценарії алгоритмів, заснованих на правилах, на інженерів, навчальні системи. Воно також відкрило нові можливості для рішень завдань, які ніколи не виконувалися без людини. Таким чином, глубинне навчання дало можливість істотно полегшити роботу в області машинного зору, розширюючи межі

можливостей комп'ютера і камери, тобто того, що вони можуть точним чином вивчити. У зв'язку з цим стає актуальним вивчення основ глибокого навчання (безпосередньо включають в себе базові відомості про штучний інтелект, машинного навчання і нейронних мереж), а також знайомство з бібліотеками, що дозволяють реалізувати моделі глибокого навчання для вирішення прикладних завдань.

Ця робота є корисною особливо у час дистанційної освіти, коли багато процесів навчання може бути оптимізовано завдяки нейронним мережам. Результати роботи можуть бути використані у розробці нових додатків, спрямованих на розпізнавання геометричних образів, наприклад, формалізація надписів на електронній дошці у виді документу. Коли у всьому світі документи переводять у електронний формат, розуміння певних моделей нейронних мереж, котрі застосовуються для вирішення цієї проблеми – є необхідністю для побудови гарного додатку, наприклад, для допомоги викладачам, котрі переводять у електронний формат фото-скани робіт студентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андреас М. Введення в машинне навчання за допомогою Python. Керівництво для фахівців по роботі з даними / Мюллер Андреас // М .: Альфа-книга, 2017. - 487 с.

2. Бенгфорт Б. Прикладний аналіз текстових даних на Python. Машинне навчання та створення додатків обробки природної мови / Б. Бенгфорт // СПб .: Питер, 2019. - 368 с.

3. Каллан, Р. Нейронні мережі: Короткий довідник / Р. Каллан. - М .: Вільямс І.Д., 2017. - 288 с.

4. Ніколенко С. Глибоке навчання / С. Ніколенко, А. Кадурін, Е. Архангельська // СПб .: Питер, 2018. - 480 с.

5. Плас Д. Python для складних завдань. Наука про дані і машинне навчання. Керівництво / Плас Джейк Вандер // М .: Питер, 2018. - 759 с.

6. Рашка С. Python і машинне навчання / С. Рашка // ДМК Пресс, 2017. - 418 с.

7. Редько В.Г. Еволюція, нейронні мережі, інтелект: Моделі і концепції еволюційної кібернетики / В.Г. Редько // М .: Ленанд, 2019. - 224 с.

8. Хайкін С. Нейронні мережі: повний курс / С. Хайкін // М .: Діалектика, 2019. - 1104 с.

9. Шолль Ф. Глибоке навчання на Python / Ф. Шолль // Пітер, 2018. - 400 с.

10. How to Digitize Texts with Open-Source Command-Line Optical Character Recognition (OCR) Software [Електронний ресурс] / Режим доступу: https://hdw.artsci.wustl.edu/articles/154

11. Karpathy A. et al. Convolutional Neural Networks for Visual Recognition [Електронний ресурс]. Режим доступу: http://www.cs231n.stanford.edu

12. The MNIST DATABASE of handwritten digits [Електронний ресурс]. Режим доступу : http://yann.lecun.com/exdb/mnist/

13. Kay A. Tesseract: Open-Source Optical Character Recognition Engine [Електронний ресурс] / Режим доступу: http://www.linuxjournal.com/article/9676

14. Nielsen MA Neural Networks and Deep Learning [Електронний ресурс]. Режим доступу: http://www.neuralnetworksanddeeplearning.com

15. OCRopy: Python-based tools for document analysis and OCR [Електронний ресурс] / Режим доступу: https://github.com/tmbdev/ocropy

16. Theodoridis S., Koutroumbas K., Pattern Recognition. New York: Elsevier Science, 2008 [Електронний ресурс]. Режим доступу: https://books.google.com/books?id=QgD-3Tcj8DkC

Додаток

Повний вихідний код

Лістинг А1. Вихідний код ноутбука Code.ipynb для порівняння моделей

```
# - * - coding: utf-8 - * -
"" "
# Порівняння різних архітектур нейронних мереж в задачі розпізнавання рукописних символів (MNIST)
## Імпорт необхідних бібліотек
"" "

from keras import datasets, utils, layers, models, optimizers
import numpy as np
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
import keras
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras import models, layers, optimizers, datasets, utils
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D

"" "## Реалізація, навчання і тестування моделі багатошарового персептрона (MLP)

### Завантаження і попередня обробка даних

Набір даних MNIST містить 60,000 навчальних і 10,000 тестових зображень цифр 0-9 (10 класів) розміром 28 х 28 пікселів. В рамках завдання навчальна вибірка буде складатися з 1024 зображень; тестова - з 100.
"" "

# Завантаження набору даних з Keras
(Xtrain, ytrain), (xtest, ytest) = datasets.mnist.load_data ()

# Зміна розмірності і нормалізація даних:
# Число вхідних нейронів - 784;
# У базі для кожного пікселя зберігається значення яскравості від 0 до 255.
xtrain = xtrain.reshape (60000, 784) / 255
xtest = xtest.reshape (10000, 784) / 255
```

```
# Отримання і висновок розмірності навчальної вибірки
xtraindata = xtrain [0 1024]
print (xtraindata.shape)

# Отримання тестової вибірки
xtestdata = xtest [0: 100]

# Зробити мітки класів категоріальним
ytrain = utils.to_categorical (ytrain, 10)
ytraindata = ytrain [0 1024]
ytest = utils.to_categorical (ytest, 10)
ytestdata = ytest [0: 100]

# Вивести мітки для навчальної вибірки
print (ytrain)

"" "### Побудова, компіляція і навчання моделі" ""

# На вхідному шарі моделі - 784 нейрона
inputs = layers.Input (shape = (784,))
# Модель багатошарового персептрона складається з набору декількох повнозв'язних шарів
x = layers.Dense (512, activation = 'relu') (inputs)
x = layers.Dense (256, activation = 'relu') (x)
x = layers.Dense (128, activation = 'relu') (x)
x = layers.Dense (64, activation = 'relu') (x)
x = layers.Dense (32, activation = 'relu') (x)
# На вихідному шарі - 10 нейронів (по числу цифр, тобто вихідних міток класів)
outputs = layers.Dense (10, activation = 'softmax') (x)

# Побудувати модель в keras і виконати її компіляцію
model = models.Model (inputs = inputs, outputs = outputs)

'' '
В якості опції втрати використовується категоріальна крос-ентропія,
оскільки вирішується проблема являє собою задачу многоклассовой класифікації.
Для оптимізації моделі використовується Nesterov Adam оптимізатор.
Так само, як оптимізатор Adam, по суті, є RMSprop з імпульсом,
так і Nadam - це RMSprop з імпульсом Нестерова. '' '

model.compile (loss = 'categorical_crossentropy', optimizer
```

```
= 'nadam', metrics = [ 'accuracy'])

    # Після компіляції модель повинна бути навчена для того, щоб виконати розпізнавання рукописних цифр:

    '' 'Виконати збереження моделі в h5-файл (ваги мережі зберігаються в форматі HDF5)' ''
    from keras.callbacks import ModelCheckpoint
    checkpointer = ModelCheckpoint (filepath = "mlp.h5", verbose = 1, save_best_only = True, save_weights_only = True)

    '' '
     Число епох (проходів за всіма примірниками навчальної вибірки) = 10
     Розмір підвибірки (кількість зразків, узятих протягом доби) = 32
    '' '
    history = model.fit (xtraindata, ytraindata, batch_size = 32, epochs = 10, validation_data = (xtestdata, ytestdata), callbacks = [checkpointer])

    # Оцінити навчену модель з точки зору продуктивності
    score = model.evaluate (xtestdata, ytestdata)

    # Вивести отримані результати
    print ( 'Величина помилки на тестовій вибірці:', score [0], 'Точність на тестовій вибірці:', score [1])

    # Зберегти модель в файл
    model.save ( 'mlp.h5')

    "" "### Візуалізація побудованої моделі" ""

    # Використовується функція plot_model з keras, модель виводиться у вигляді png-файла
    SVG (model_to_dot (model, show_shapes = True) .create (prog = 'dot', format = 'svg'))
    from tensorflow.keras.utils import plot_model
    plot_model (model, show_shapes = True, to_file = 'mlp.png')

    "" "### Графік залежності точності від числа епох" ""

    import matplotlib.pyplot as plt
    print (history.history.keys ())
    print (history.epoch)
    plt.plot (history.epoch, history.history [ 'accuracy'], 'g')
```

```
     plt.plot (history.epoch, history.history [ 'val_accuracy'])
     plt.ylabel ( 'Точність')
     plt.xlabel ( 'Число епох')
     plt. legend ([ 'Навчальна вибірка', 'Тестова вибірка'], loc
= 'lower right')
     plt.show ()

     "" "## Реалізація, навчання і тестування моделі сверточное
нейронної мережі (CNN)

     ### Завантаження і попередня обробка даних
     "" "

     (X_train, y_train), (x_test, y_test) = mnist.load_data ()
     x_train = x_train [1024]
     y_train = y_train [1024]
     x_test = x_test [: 100]
     y_test = y_test [: 100]
     x_train = x_train.reshape (1024,28,28,1) / 255
     x_test = x_test.reshape (100,28,28,1) / 255
     x_train = x_train.reshape (1024,28,28,1) / 255
     x_test = x_test.reshape (100,28,28,1) / 255
     y_train = utils.to_categorical (y_train, 10)
     y_test = utils.to_categorical (y_test, 10)

     "" "### Побудова моделі" ""

     # Вхідне зображення має розмір 28 на 28 пікселів; підрахувати
число входів (кол.-під зображень)
     inputs = layers.Input (shape = (28,28,1,))
     # Додати вхідний сверточних шар з функцією активації ReLU і
32 фільтрами згортки розміру 3x3
     cnn = Conv2D (32, kernel_size = (3,3), activation = 'relu',
padding = 'valid', input_shape = (28,28,1)) (inputs)
     # Додати другий сверточних шар з 64 фільтрами
     cnn = Conv2D (64, kernel_size = (3,3), activation = 'relu')
(cnn)
     # Додати шар ПУЛІНГ з фільтром 2x2
     cnn = MaxPooling2D (pool_size = (2,2)) (cnn)
     # Додати шар виключення (дропаутов) для запобігання
перенавчання
     cnn = Dropout (0.25) (cnn)
     # Додати шар, що зменшує розмірність (це буде необхідно для
класифікації)
     cnn = Flatten () (cnn)
     # Додати повнозв'язну шар і ще один шар дропаутов
```

```
    cnn = Dense (132, activation = 'relu') (cnn)
    cnn = Dropout (0.5) (cnn)
    # Додати на вихід oftmax-шар з 10 нейронами (за кількістю класів)
    outputs = Dense (10, activation = 'softmax') (cnn)
    model = models.Model (inputs = inputs, outputs = outputs)

    "" "### Компіляція і навчання моделі" ""

    # Скомпілювати і навчити модель - 12 епох, розмір підвибірки - 32
    model.compile (loss = 'categorical_crossentropy', optimizer = 'nadam', metrics = [ 'accuracy'])

    '' 'Виконати збереження моделі в h5-файл (ваги мережі зберігаються в форматі HDF5)' ''
    from keras.callbacks import ModelCheckpoint
    checkpointer = ModelCheckpoint (filepath = "cnn.h5", verbose = 1, save_best_only = True, save_weights_only = True)

    history = model.fit (x_train, y_train, batch_size = 32, epochs = 12, verbose = 1, validation_data = (x_test, y_test), callbacks = [checkpointer])

    # Оцінити модель на навчальній і на тестовій вибірці
    accuracy = model.evaluate (x_test, y_test, verbose = 1, batch_size = 32)
    accuracy1 = model.evaluate (x_train, y_train, verbose = 1, batch_size = 32)

    # Вивести отримані результати
    print ( 'Величина помилки на навчальній вибірці:', accuracy1 [0], 'Точність на навчальній вибірці:', accuracy1 [1])
    print ( 'Величина помилки на тестовій вибірці:', accuracy [0], 'Точність на тестовій вибірці:', accuracy [1])

    # Зберегти модель в файл
    model.save ( 'cnn.h5')

    "" "### Візуалізація побудованої моделі" ""

    SVG (model_to_dot (model, show_shapes = True) .create (prog = 'dot', format = 'svg'))
    from tensorflow.keras.utils import plot_model
    plot_model (model, show_shapes = True, to_file = 'cnn.png')
```

```
    "" "### Графік залежності точності від числа епох" ""

    plt.plot (history.epoch, history.history [ "accuracy"], 'g',
label = "Навчальна вибірка")
    plt.plot (history.epoch, history.history [ "val_accuracy"],
label = "Тестова вибірка")
    plt.legend ()
    plt.xlabel ( "Число епох")
    plt.ylabel ( "Точність")

    "" "## Завантаження, модифікація, навчання і тестування
моделі VGG16

    ### Завантаження моделі з keras і висновок інформації про неї
    "" "

    vgg16 = keras.applications.vgg16.VGG16 (weights = 'imagenet',
include_top = True)
    vgg16.summary ()

    "" "### Модифікація моделі для вирішення завдання MNIST" ""

    # Як показує інформація про моделі, останні три шари -
повнозв'язні

    # Створити нову послідовну модель
    model = keras.models.Sequential ()

    # Додати в неї всі шари, крім останнього вихідного
повнозв'язну шару від 1000 нейронів
    for layer in vgg16.layers [: - 1]:
     model.add (layer)

    # Додати останній вихідний шар на 10 нейронів для вирішення
задачі класифікації цифр
    model.add (keras.layers.Dense (10, activation = 'softmax'))

    # Заморозити все шари, крім останніх повнозв'язних

    '' 'Стан замороженого шару не буде змінюватися в процесі
навчання
     (Ні при навчанні за допомогою fit (), ні при навчанні з будь-
яким іншим методом,
     який покладається на зміну ваг із застосуванням градієнтного
спуску).
    '' '
```

```
for layer in model.layers [: - 3]:
 layer.trainable = False

# Вивести інформацію про нову модель
model.summary ()

"" "### Компіляція моделі" ""

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

"" "### Завантаження і попередня обробка даних" ""

from keras.applications.vgg16 import preprocess_input
from keras.utils import to_categorical
from keras.preprocessing.image import img_to_array, array_to_img
import tensorflow as tf
from scipy import ndimage
mnist = keras.datasets.mnist
(X_train, y_train), (x_test, y_test) = mnist.load_data ()
x_train = x_train [: 1024,:,:]
y_train = y_train [1024]
x_test = x_train [: 256,:,:]
y_test = y_train [: 256]
x_train = np.dstack ([x_train] * 3)
x_test = np.dstack ([x_test] * 3)
x_train = x_train.reshape (-1, 28,28,3)
x_test = x_test.reshape (-1, 28,28,3)
x_train = np.asarray ([img_to_array (array_to_img (im, scale = False) .resize ((224,224))) for im in x_train])
x_test = np.asarray ([img_to_array (array_to_img (im, scale = False) .resize ((224,224))) for im in x_test])
x_train = x_train.astype ( 'float32')
x_test = x_test.astype ( 'float32')
y_train = to_categorical (y_train, 10)
y_test = to_categorical (y_test, 10)
x_train_RGB = preprocess_input (x_train)
x_test_RGB = preprocess_input (x_test)
x_train_RGB.shape, x_test.shape, y_train.shape, y_test.shape

"" "### Навчання моделі" ""

r = model.fit (x_train_RGB, y_train, validation_data = (x_test_RGB, y_test), epochs = 20)
```

```
    # Оцінити модель на навчальній і на тестовій вибірці
    scoreVGG = model.evaluate (x_test_RGB, y_test)
    print ( 'Величина помилки на тестовій вибірці:', scoreVGG
[0], 'Точність на тестовій вибірці:', scoreVGG [1])

    # Зберегти модель в файл
    model.save ( 'vgg16.h5')

    "" "### Графік залежності точності від числа епох" ""

    import matplotlib.pyplot as plt
    print (r.history [ 'accuracy'])
    print (r.epoch)
    plt.plot (r.epoch, r.history [ 'accuracy'], 'g')
    plt.plot (r.epoch, r.history [ 'val_accuracy'])
    plt.ylabel ( 'Точність')
    plt.xlabel ( 'Число епох')
    plt. legend ([ 'Навчальна вибірка', 'Тестова вибірка'], loc
= 'lower right')
    plt.show ()

    "" "### Візуалізація побудованої моделі" ""

    SVG (model_to_dot (model, show_shapes = True) .create (prog
= 'dot', format = 'svg'))
    from tensorflow.keras.utils import plot_model
    plot_model (model, show_shapes = True, to_file = 'vgg16.png')
```