МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Особливості побудови комп'ютерної системи розпізнавання рукописних текстів на зображеннях**

Виконав: студент 2 курсу, групи 8.1212-іпз
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
О.Ф. Крат
(ініціали та прізвище)
Керівник доцент, А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Тетяна Критська
"_01_" __вересня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

___Крат Олексій Федорович___
(прізвище, ім'я, по батькові)

1.Тема роботи __Особливості побудови комп'ютерної системи розпізнавання рукописних текстів на зображеннях_____

керівник роботи_____Безверхий Анатолій Ігорович, доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від ___09.10.2023 №1577-С_____
2. Строк подання студентом кваліфікаційної роботи_____30.11.2023_____
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми розпізнавання мов та розробка методів її вирішення;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
25 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.23	виконано
3	Аналіз існуючих методів рішення	13.09-14.09.23	виконано
4	Дослідження засобів виявлення об’єктів на зображеннях	15.09-20.09.23	виконано
5	Дослідження засобів виявлення контурів об’єктів та трансформації зображень	21.09-26.09.23	виконано
6	Узгодження подальших дій з науковим керівником	27.09-30.09.23	виконано
7	Вибір фреймворка та встановлення середовища розробки нейронної мережі	01.10-11.10.23	виконано
8	Збір тренувального набору для нейронної мережі з відкритих джерел та його попередня підготовка	12.10-25.10.23	виконано
9	Навчання моделі нейронної мережі за допомогою підготовленого тренувального набору	26.10-08.11.23	виконано
10	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	09.11-12.11.23	виконано
11	Експериментування з метою покращення точності нейронної мережі	13.11-20.11.23	виконано
12	Оформлення звіту	21.11-28.11.23	виконано

Студент _____ Крат О.Ф.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Безверхий А.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 70

Рисунків: 33

Таблиць: 1

Джерел: 23

Крат О.Ф. Особливості побудови комп’ютерної системи розпізнавання рукописних текстів на зображеннях: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник А. І. Безверхий. Запоріжжя : ЗНУ, 2023. 70 с.

Мета і завдання роботи полягають в дослідженні сучасних засобів розробки системи розпізнавання рукописних текстів на зображеннях та розробці системи, що буде ефективно вирішувати цю задачу.

У процесі дослідження була розглянута проблема розпізнавання рукописних текстів на зображеннях та підходи глибинного навчання до її вирішення. Як результат, була розроблена та навчена оптимальна модель глибокої згорткової нейронної мережі, яка може розпізнавати українські рукописні літери з точністю 97.63%.

Ключові слова: згорткова нейронна мережа, розпізнавання рукописних символів, набір даних (датасет), глибоке навчання, фреймворк, Tensorflow, Keras.

SUMMARY

Pages: 70

Figures: 33

Tables: 1

Sources: 23

Krat O.F. Peculiarities of building a computer system for recognizing handwritten texts on images: qualification work of the master of specialty 121 "Software Engineering" / Science head A.I. Bezverkhy. Zaporizhzhia : ZNU, 2023. 70 p.

The aim of the research is to study the modern means of development a system for handwritten character recognition, as well as to develop a computer system that will effectively solve this problem.

In the course of the research the problem of handwritten character recognition and deep learning approaches to its solution were considered. As a result, an optimal deep convolutional neural network model, which can recognize Ukrainian handwritten characters with 97.63% accuracy, has been developed and trained.

Keywords: convolutional neural network, handwritten character recognition, dataset, deep learning, framework, Tensorflow, Keras.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ 12
1.1 Огляд проблеми розпізнавання текстів на зображеннях 12
1.2 Алгоритми розпізнавання текстів на зображеннях 13
1.2.1 Шаблонні алгоритми 15
1.2.2 Ознакові алгоритми 15
1.2.3 Нейромережеві алгоритми 16
1.3 Огляд існуючих рішень для розпізнавання рукописних текстів на зображеннях 16
1.4 Аналіз програмного забезпечення для розпізнавання текстів на зображеннях 19
1.4.1 Бібліотека Tesseract 19
1.4.2 Програма ABBYY FineReader 20
1.4.3 Програма Google Cloud Vision 21
1.4.4 Програма Freemore OCR 22
1.5 Висновки з розділу 1 23
РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ 24
2.1 Поняття згорткових нейронних мереж 24
2.2 Огляд фреймворків та бібліотек глибокого навчання 27
2.2.1 Фреймворк Tensorflow 27
2.2.2 Фреймворк PyTorch 28
2.2.3 Фреймворк Apache MXNet 29
2.2.4 Фреймворк Microsoft Cognitive Toolkit 30
2.2.5 Бібліотека Keras 30
2.2.6 Фреймворк Core ML 32
2.2.7 Фреймворк ONNX 33

2.3 Датасет із зображеннями українських рукописних символів 34
2.4 Висновки з розділу 2 35
РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ 36
3.1 Встановлення та налаштування середовища для розробки системи розпізнавання рукописних текстів на зображеннях 36
3.2 Розробка і навчання згорткової нейронної мережі розпізнавати українські рукописні символи 38
3.3 Висновки з розділу 3 50
РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ 51
4.1 Експериментування для одержання найкращого результату навчання ЗНМ 51
4.1.1 Аналіз результатів навчання нейронної мережі 51
4.1.2 Способи підвищення точності розпізнавання 52
4.2 Висновки з розділу 4 65
ВИСНОВКИ 67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 68

ВСТУП

Актуальність теми

Розпізнавання тексту вважається одним із найперших завдань комп’ютерного зору, яким займалися дослідники. Вже понад століття з моменту його створення як галузі досліджень вчені не припиняли працювати над ним. Цьому можуть сприяти два важливі фактори.

По-перше, поширеність тексту та його важливість для нашого повсякденного життя як візуального кодування мови, яке широко використовується для спілкування та збереження всіх видів людської думки. По-друге, необхідність тексту для людей і його поширеність призвели до великих вимог до адекватності його доставки та прийому, що призвело до великої варіативності та постійного збільшення візуальних форм тексту.

Текст може бути надрукованим або рукописним із великою можливістю варіативності стилів почерку, шрифтів друку та параметрів форматування. Його можна знайти в документах у вигляді рядків, таблиць, форм або безлад у природних сценах. Текст може зазнавати незліченних типів і ступенів деградації, спотворень перегляду, оклюзій, фонів, інтервалів, нахилів і викривлень. Лише текст з розмовних мов (як важлива підмножина створеного людиною тексту) доступний десятками шрифтів, які відповідають тисячам мов [1].

Тому створення програмних засобів для розпізнавання тексту та дослідження цієї області є актуальним.

Тема роботи – особливості побудови комп’ютерної системи розпізнавання рукописних текстів на зображеннях.

Мета і завдання дослідження

Мета і завдання роботи полягають в дослідженні сучасних засобів розробки системи розпізнавання рукописних текстів на зображеннях та розробці системи, що буде ефективно вирішувати цю задачу.

Об’єкт дослідження

Об’єктом дослідження є процес розпізнавання тексту на зображенні за допомогою згорткової нейронної мережі.

Предмет дослідження

Предметом дослідження є система розпізнавання рукописних текстів на зображеннях.

Методи дослідження

Для вирішення поставлених задач використовуються наступні методи дослідження:

1. Аналіз існуючих рішень для проблеми розпізнавання рукописного тексту на зображенні.
2. Аналіз бібліотек глибокого навчання.
3. Синтез та узагальнення отриманих знань у процесі дослідження проблеми розпізнавання рукописного тексту на зображенні.
4. Експериментування зі зміною параметрів моделі нейронної мережі.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що для вирішення задачі розпізнавання рукописних текстів на зображеннях був вдосконалений сучасний підхід, а саме: була розроблена та навчена оптимальна модель глибокої згорткової нейронної мережі для розпізнавання українських рукописних літер.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, що була розроблена система, яка може розпізнавати українські рукописні літери з точністю 97.63%. Дана система може використовуватись в програмних продуктах для розпізнавання текстів як для звичайних користувачів, так і для дослідників.

Апробація одержаних результатів

Результати дослідження були представлені на XVI науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2023» [2], а також на III Всеукраїнській науково-практичній конференції за участю молодих науковців «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» [3].

Глосарій

Глибоке навчання (англ. Deep learning) — це сукупність методів машинного навчання (з учителем, з частковим залученням вчителя, без вчителя, з підкріпленням), що ґрунтуються на навчанні уявлень (англ. feature/representation learning), а не спеціалізованих алгоритмах під конкретні завдання.

Згорткова нейронна мережа, ЗНМ(англ. convolutional neural network, CNN) – спеціальна архітектура штучних нейронних мереж, запропонована Яном Лекуном у 1988 році та націлена на ефективне розпізнавання образів, входить до складу технологій глибокого навчання (англ. deep learning).

Набір даних, датасет (англ. Dataset) — це оброблена та структурована певним чином інформація, необхідна для навчання нейронної мережі.

Розпізнавання тексту із зображення або оптичне розпізнавання символів (optical character recognition, OCR) – механічний або електронний переклад зображень рукописного, машинописного або друкованого тексту в

текстові дані, що використовуються для представлення символів на комп'ютері (наприклад, у текстовому редакторі).

Штучні нейронні мережі (англ. Artificial neural network) – математична модель, а також її програмне чи апаратне втілення, побудована за принципом організації та функціонування біологічних нейронних мереж — мереж нервових клітин живого організму.

Штучний нейрон (англ. Artificial neuron) — це вузол штучної нейронної мережі, що є спрощеною моделлю природного нейрона, математично представлений як деяка нелінійна функція від лінійної комбінації його вхідних сигналів.

Python — це високорівнева мова програмування загального призначення з динамічною строгою типізацією та автоматичним управлінням пам'яттю, орієнтована на підвищення продуктивності розробника, читання коду та його якості, а також на забезпечення переносимості написаних на ньому програм.

TensorFlow — це відкрита програмна бібліотека для машинного навчання, розроблена Google для вирішення завдань побудови та тренування нейронної мережі з метою автоматичного знаходження та класифікації образів.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ

1.1 Огляд проблеми розпізнавання текстів на зображеннях

Через те, що вартість обробки зображень чи відео доволі висока та вимагає значних затрат з боку процесора, більшість додатків залежать від даних, інтерпретованих у цифровому вигляді. Дані повинні бути проструктуровані, щоб вони могли бути легко опрацьовані комп'ютером. Також важливо, щоб дані були релевантні та доступні для пошуку, наприклад, у текстовому форматі. Щоб автоматично чи інтерактивно отримувати такі дані, необхідно виконувати пошук за мультимедійними базами даних, використовуючи описові функції (особливості), які стосуються об'єкта дослідження: фотографій, відео тощо [4, 5].

Особливості, такі як колір, текстура, форма, рух, макет, можна отримати під час обробки на низькому рівні, використовуючи технології комп'ютерного бачення. Різновиди особливостей, які стосуються кольору, використовують як атомні блоки для пошуку зображень та відео, наприклад, домінуючий колір, регіональна чи глобальна гістограма, вектори усередненого кольору та власне зображення. Особливості текстури, такі як гістограми орієнтації краю, параметри випадкових полів Маркова, коефіцієнти перетворення вейвлет та Фур'є, локальні бінарні шаблони також широко використовуються для індексування природних образів [6].

Особливості форми переважно використовують в задачах пошуку та моделювання об'єктів. Типовими особливостями форми є область накладання, параметри перетворення Хафа, елементарні дескриптори, вимірювання зміни кривизни простору, дескриптори Фур'є та кути країв. Особливості руху під час обробки на низькому рівні містять величину руху та похідні. Особливостями макета можуть бути особливості просторових вимірювань у зображеннях або в кадрах відео.

Особливості низькорівневої обробки можна легко отримати, проте це не дасть чіткого уявлення про те, що власне присутнє на зображенні. Для отримання більш описових функцій особливості низькорівневої обробки переважно зливаються в більш інформативні об’єкти та події, наприклад, текст, людське обличчя, велосипед, будинок, небо, хмари, сонце, пляж тощо. Виявлення та розпізнавання об’єктів високого рівня привертає все більше уваги науковців. Наприклад, різні способи виявлення та розпізнавання обличчя були відомі протягом багатьох років і демонструють доволі вагомі результати.

Незважаючи на це, розпізнавання інших об’єктів, зокрема тексту, залишається актуальною проблемою і сьогодні [7-8].

Існує два види тексту на зображеннях: текст сцени та накладений текст. Текст сцени – це ті текстові рядки, які написані на деяких об’єктах у зображеннях. Вони, як правило, мають великі розміри, але можуть мати оклюзії, різні вирівнювання та рухи. Накладений текст не має оклюзій, як правило, є малого розміру та релевантний до відповідного зображення.

1.2 Алгоритми розпізнавання текстів на зображеннях

Розпізнавання тексту із зображення або оптичне розпізнавання символів [9] (optical character recognition, OCR) - це технологія для автоматизації вилучення даних з друкованого, письмового тексту, відсканованого документа, файлу зображення з метою подальшого перетворення тексту в машиночитану форму. Ця форма буде використана для роботи з даними, наприклад, редагування або пошук інформації.

Кожна система OCR складається з тих самих кроків алгоритму:

- передобробка;
- сегментація;
- виділення ознак;
- розпізнавання символів або класифікація;

- постобробка та виправлення помилок розпізнавання.

Ці алгоритмічні кроки виконуються послідовно, кожен результат кроку подається на вхід наступного кроку.

Крок передобробки. Перед тим як передати зображення на розпізнавання, необхідно його обробити та виділити необхідну інформацію, саме для цього використовується шар передобробки. На цьому етапі із зображенням можуть відбуватися операції очищення зображення від шумів, приведення до вигляду, що дозволяє виділити символи на фоні, фільтрація зображення, згладжування та збільшення контрастності. Якщо текст рукописний, то додатково застосовують підхід з випрямлення символів, оскільки багато хто пише символи з нахилом. В основному використовується бінаризація зображення, яка дозволяє точно виділити текст та прибрати фон.

Алгоритм сегментації. Сегментація зображення [9] - це виділення корисної інформації із зображення, з подальшою її обробкою. Сегментація в галузі розпізнавання тексту складається з кількох етапів:

- сегментація рядків – виділити на зображенні лініями фрагменти слів;
- сегментація слів – виділення слів, виділяємо окремі фрагменти зображень, де присутні слова;
- сегментація символів – розділяється розпізнане зображення слова на символи.

Крок класифікації зображення. Класифікація дозволяє розпізнати символ із зображення та перевести його в машиночитаний формат. Існують різні види алгоритмів розпізнавання, найпопулярнішими є:

- шаблонні алгоритми;
- ознакові алгоритми;
- нейромережеві алгоритми.

Алгоритм постобробки. У багатьох системах OCR результат, який отримується після класифікації, не вважається достатнім. Необхідно використовувати контекстну інформацію, яка дозволяє не лише знаходити помилки, а й виправляти їх. Існують різні методи здійснення постобробки,

наприклад, глобальні та локальні позиційні діаграми, триграми, n-грами, словники та різні поєднання всіх цих методів. Найпопулярнішим підходом є словник.

1.2.1 Шаблонні алгоритми

Суть методу полягає в тому, що йде порівняння кожного символу із шаблонами з бази. Найбільш підходящим шаблоном вважається той, який матиме найменшу кількість точок, відмінних від досліджуваного зображення. Шаблони для кожного символу зазвичай виходять усереднення зображень символів навчальної вибірки.

Даний алгоритм має високу точність розпізнавання тексту, а недоліком є те, що не можна розпізнати інший шрифт, який відрізняється від закладеного в систему. Цей метод повинен заздалегідь знати шрифт, який він розпізнає, саме цей момент обмежує універсальність шаблонних алгоритмів [9].

1.2.2 Ознакові алгоритми

Ознаковий метод полягає в тому, що зображення представляється як К-мірний вектор ознак. Розпізнавання полягає у порівняння його з набором еталонних векторів тієї ж розмірності. Ухвалення рішення про схожість образу до певного символу будується на підставі математичних рішень у рамках детерміністичного та ймовірнісного підходів. У системі розпізнавання даного методу використовується класифікація, заснована на підрахунку евклідової відстані між вектором ознак символу, що розпізнається, і векторами ознак еталонного опису. Кількість та тип ознак можуть визначити якість розпізнавання.

Головні переваги ознакових методів - це простота реалізації, хороша стійкість до змін форми символів, низька кількість помилок при розпізнаванні, висока швидкодія. Найголовніші недоліки даного алгоритму - нестійкість до різних дефектів зображення, наприклад шум, а також на етапі вилучення ознак

із символу відбувається втрата основної інформації, вилучення ведеться незалежно, через що розташування елементів символу втрачається [9].

1.2.3 Нейромережеві алгоритми

Існує багато моделей класифікаторів розпізнавання тексту, але завжди як базові архітектури використовуються згорткові нейронні мережі, а також функціонал збереження та накопичення результату розпізнавання, і рекурентна мережа для розпізнавання.

Вхідними даними для нейромережевого методу є зображення рядків та слів. Вихідними даними є символи, що йдуть по порядку, що формують машинний текст.

Основні недоліки – текст має бути у вертикальному положенні, складність підбору навчальної вибірки. Основні переваги – це висока швидкість та узагальненість. Саме тому цей метод зараз використовується у різних сучасних системах розпізнавання тексту[9].

1.3 Огляд існуючих рішень для розпізнавання рукописних текстів на зображеннях

У дослідженні Khandokar I. [10] ЗНМ реалізовано для розпізнавання символів із тестового набору даних. Основна мета роботи полягає в тому, щоб дослідити здатність ЗНМ розпізнавати символи з набору даних зображення та точність розпізнавання за допомогою навчання та тестування. ЗНМ розпізнає символи, розглядаючи форми та порівнюючи риси, які відрізняють їх. Дослідник експериментує з набором даних NIST, щоб отримати точність рукописних символів. Результати тесту свідчать про те, що точність 92,91% отримана на 200 зображеннях із навчальним набором із 1000 зображень від NIST.

Робота Deore S. та Pravin A. [11] присвячена підходу до тонкого налаштування та аналізу найсучаснішої глибокої згорткової нейронної мережі

(DCNN), розробленої для класифікації рукописних символів деванагарі. Набори даних складаються із 5800 ізольованих зображень 58 унікальних класів символів: 12 голосних, 36 приголосних і 10 цифр. На додаток до цієї бази даних реалізована двоетапна модель глибокого навчання VGG16 для розпізнавання цих символів за допомогою двох вдосконалених методів адаптивного градієнта. Двоетапний підхід глибокого навчання розроблено для підвищення загального успіху запропонованої системи розпізнавання. Перша модель досягає 94,84% точності тестування з втратою навчання 0,18 на новому наборі даних. Більше того, друга точно налаштована модель потребує дуже мало параметрів, які можна навчити, і особливо менше часу на навчання для досягнення найсучаснішої продуктивності на дуже малому наборі даних. Вона досягає 96,55% точності тестування з втратою тренування 0,12.

У дослідженні Shams M. [12] представлено алгоритм розпізнавання арабських літер і символів на основі використання нейронних мереж глибокої згортки (DCNN) і опорної векторної машини (SVM). Дослідник розглянув проблему розпізнавання арабських рукописних символів шляхом визначення подібності між шаблонами введення та попередньо збереженими шаблонами з використанням як повністю підключеної DCNN, так і вихідної SVM. Крім того, у статті визначено, що правильний рівень класифікації (CRR) залежить від точності виправлених класифікованих шаблонів розпізнаних рукописних арабських символів. Також визначено коефіцієнт класифікації помилок (ECR). Експериментальні результати цієї роботи свідчать про здатність запропонованого алгоритму розпізнавати, ідентифікувати та перевіряти введені рукописні арабські символи. Точність системи досягла 95,07% CRR з 4,93% ECR порівняно з сучасним рівнем техніки.

У статті M.B. Bora [13] системи розпізнавання рукописних символів були представлені за допомогою CNN-ECOC, який є комбінацією CNN і класифікатора ECOC. CNN використовується для виділення ознак, а ECOC для розпізнавання символів. Щоб знайти відповідний екстрактор функцій, було досліджено три популярні архітектури ЗНМ, а саме LeNet, AlexNet і ZfNet. З

результатів моделювання було помічено, що LeNet дає низький рівень точності. Тому його було модифіковано шляхом додавання шару вилучення та шару ReLu після першого повністю підключеного шару, що призвело до вищого рівня точності. Також було помічено, що точність класифікаторів ECOC вища порівняно з класифікатором softmax CNN. Серед реалізованих мереж AlexNet є найбільш придатною ЗНМ для поєднання з ECOC, щоб розпізнавати рукописні символи.

Стаття Liu H. [14] спрямована на дослідження переглянутої моделі на основі LeNet-5. Завдяки оптимізації та налаштуванню параметрів модель ЗНМ навчена набором даних EMNIST, і забезпечує точність 93,44%. Дослідження дає деяку інформацію про розпізнавання рукописних символів і пропонує деякі можливі рішення щодо збереження паперових матеріалів. Вивчаючи структуру LeNet-5, ця стаття доводить перевагу згорткових нейронних мереж. Проте існують певні обмеження щодо цього дослідження, які можуть надати пропозиції та рекомендації для майбутніх досліджень. LeNet-5 все ще обмежений у розпізнаванні літер і не може досягти надзвичайно високої точності. Деякі більш складні архітектури CNN можуть бути прийняті. У той же час набір даних не піддавався попередній обробці, і для підвищення точності можна використовувати такі операції, як вилучення ознак.

У статті N. Saqib [15] для вирішення систем HCR із багатокласовою класифікацією пропонується модель на основі CNN, яка досягла винятково хороших результатів із цією багатокласовою класифікацією. Моделі CNN були навчені за допомогою набору цифр MNIST, який складається з 60 000 навчальних і 10 000 тестових зображень. Їх також навчали зі значно більшим набором даних алфавіту Kaggle, який містить понад 297 000 навчальних зображень і тестовий набір, який формується на основі тестування понад 74 490 зображень. Для набору даних Kaggle загальна точність за допомогою оптимізатора «ADAM» становила 99,516%, 99,511% і 99,563% для швидкості навчання (LR) 0,001, LR 0,0001 і LR 0,00001 відповідно. Тим часом та сама модель з використанням «RMSprop» досягла точності 99,292%, 99,108% і

99,191% відповідно, на LR 0,001, LR 0,0001 і LR 0,00001. Для набору даних MNIST загальна точність за допомогою «RMSprop» становила 99,642%, 99,452% і 98,142% для LR 0,001, LR 0,0001 і LR 0,00001 відповідно. Тим часом та сама модель із використанням оптимізатора «ADAM» досягла точності 99,571%, 99,309% і 98,142% з LR 0,001, LR 0,0001 і LR 0,00001 відповідно. Можна легко зрозуміти, що для розпізнавання алфавіту точність зменшується зі збільшенням швидкості навчання (LR); навпаки, загальна точність пропорційно пов’язана з LR для розпізнавання цифр.

1.4 Аналіз програмного забезпечення для розпізнавання текстів на зображеннях

1.4.1 Бібліотека Tesseract

Бібліотека Tesseract [16] безкоштовна та проста у використанні. Вона несе в собі функціональність інструменту командного рядка, але є також оболонка для мови Python, яка називається pytesseract, а також додаток для комп'ютерів з графічним інтерфейсом gImageReader. Tesseract можна використовувати безпосередньо через командний рядок або (для програмістів) за допомогою API для отримання друкованого тексту із зображень. Він підтримує широкий спектр мов [17].

Бібліотека Tesseract OCR досить добре розпізнає відсканований текст, але коли справа доходить до рукописного тексту, відсоток розпізнаного тексту знижується і з'являються помилки. З розпізнаванням табличної інформації Tesseract OCR є труднощі, необхідно самостійно обробляти вихідні дані за допомогою додаткових технологій і бібліотек.

На даний момент останньою версією є Tesseract 5.0.

```
  @geo /tmp $ tesseract
Usage:
  tesseract imagename|stdin outputbase|stdout [options...] [configfile...]

OCR options:
  --tessdata-dir /path  specify location of tessdata path
  -l lang[+lang]        specify language(s) used for OCR
  -c configvar=value    set value for control parameter.
                        Multiple -c arguments are allowed.
  -psm pagesegmode      specify page segmentation mode.
These options must occur before any configfile.

pagesegmode values are:
  0 = Orientation and script detection (OSD) only.
  1 = Automatic page segmentation with OSD.
  2 = Automatic page segmentation, but no OSD, or OCR
  3 = Fully automatic page segmentation, but no OSD. (Default)
  4 = Assume a single column of text of variable sizes.
  5 = Assume a single uniform block of vertically aligned text.
  6 = Assume a single uniform block of text.
  7 = Treat the image as a single text line.
  8 = Treat the image as a single word.
  9 = Treat the image as a single word in a circle.
  10 = Treat the image as a single character.

Single options:
  -v --version: version info
  --list-langs: list available languages for tesseract engine. Can be used with
--tessdata-dir.
  --print-parameters: print tesseract parameters to the stdout.
  @geo /tmp $ tesseract -v
tesseract 3.03
 leptonica-1.71
  libgif 5.1.0 : libjpeg 8d : libpng 1.6.13 : libtiff 4.0.3 : zlib 1.2.8 : libwe
bp 0.4.1
```

Рисунок 1.1—Робота з бібліотекою Tesseract

1.4.2 Програма ABBYY FineReader

Продукт ABBYY FineReader [18] – розробка компанії ABBYY, яка входить до числа провідних компаній із розпізнавання тексту. Даний продукт є програмою з графічним інтерфейсом користувача, де можна завантажувати документи і отримувати результат у вигляді файлу. Також існує ABBYY Cloud OCR SDK API – це хмарний сервіс, який використовує двигун ABBYY FineReader OCR.

На відміну від Tesseract, ABBYY Cloud OCR є платним. ABBYY FineReader не має проблем із добре відсканованим текстом і непогано справляється з документами, які сфотографовані та, можливо, з якимись шумами та розворотами. Проте у рукописному документі він повністю не працює. Його головна перевага – можливість вилучення таблиці. Крім осередків він витягує такі дрібні деталі як шрифти.

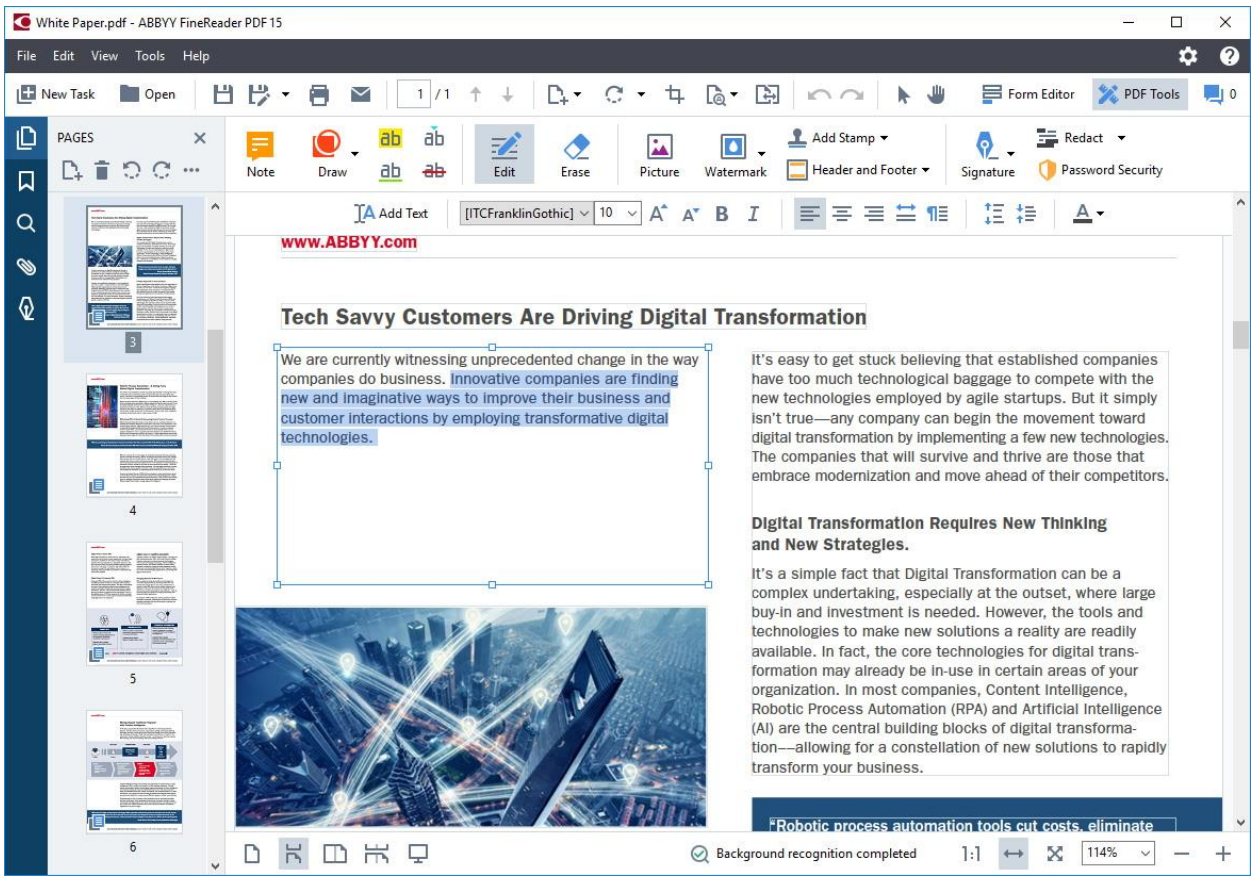

Рисунок 1.2 — Інтерфейс програми ABBYY FineReader

1.4.3 Програма Google Cloud Vision

Продукт Google Cloud Vision [19], являє собою хмарний сервіс розпізнавання текстів із зображень. Він також, як і продукт ABBYY, є платним. Google добре справляється з відсканованим текстом і розпізнає текст у документі, знятому на камеру, як і ABBYY. Однак він набагато кращий, ніж Tesseract або ABBYY у розпізнаванні почерку. Google Cloud Vision не дуже добре опрацьовує таблиці: він отримує текст, але це все. Фактично, результат роботи Cloud Vision є файлом JSON, що містить інформацію про позиції символів. Як і у випадку з Tesseract, на основі цієї інформації можна спробувати виявити таблиці, але ця функція не вбудована і необхідно використовувати додаткові ресурси і технології.

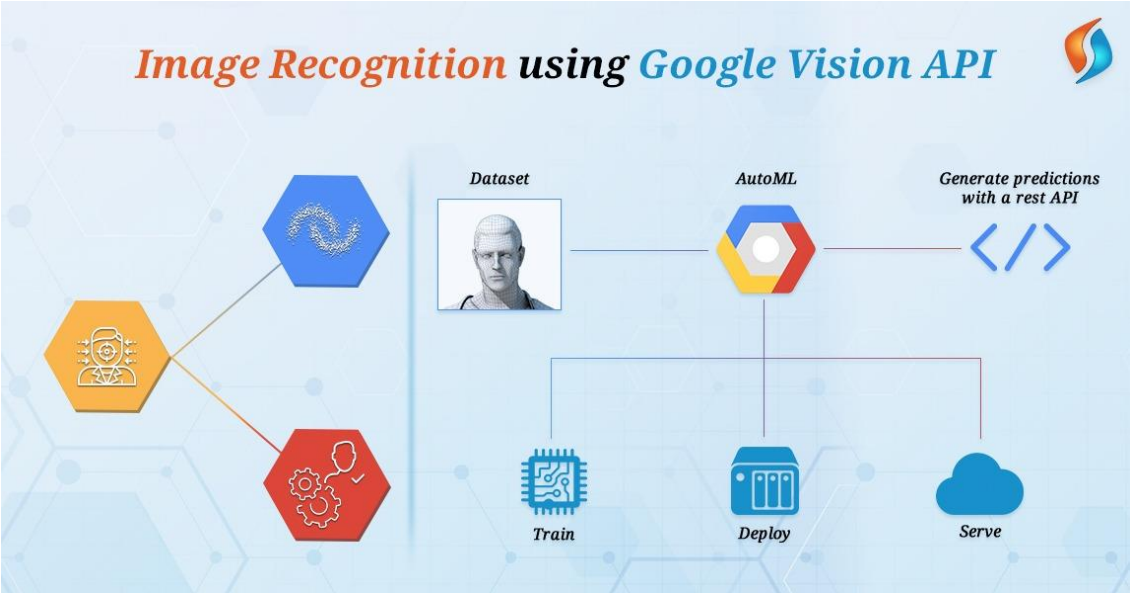

Рисунок 1.3 — Розпізнавання зображення в програмі Google Cloud Vision

1.4.4 Програма Freemore OCR

Freemore OCR призначена для вилучення тексту із зображення, створеного цифровими камерами та мобільними телефонами разом зі сканерами. Потім точний результат можна відредагувати та зберегти в TXT і Word [20].

Оскільки в наші дні так багато сканерів є частиною універсального принтера/сканера/копіювального апарату, уникати встановлення гігантських драйверів принтера, що автоматично запускаються та саморекламуються, — це те, до чого ми всі повинні прагнути. Якщо у вас є сканер і ви хочете уникнути повторного друку документів, Freemore OCR — чудова альтернатива. Якщо вам час від часу потрібно перетворювати відскановані зображення, факси, знімки екрана, PDF-документи та електронні книги в текст, Freemore OCR також підходить. Це безкоштовна програма оптичного розпізнавання символів, яка підтримує сканування з більшості сканерів Twain, а також може відкривати всі файли PDF і зображення. Ваше зображення або PDF-файл з’явиться в лівому вікні, а текст готовий для редагування або копіювання в правому.

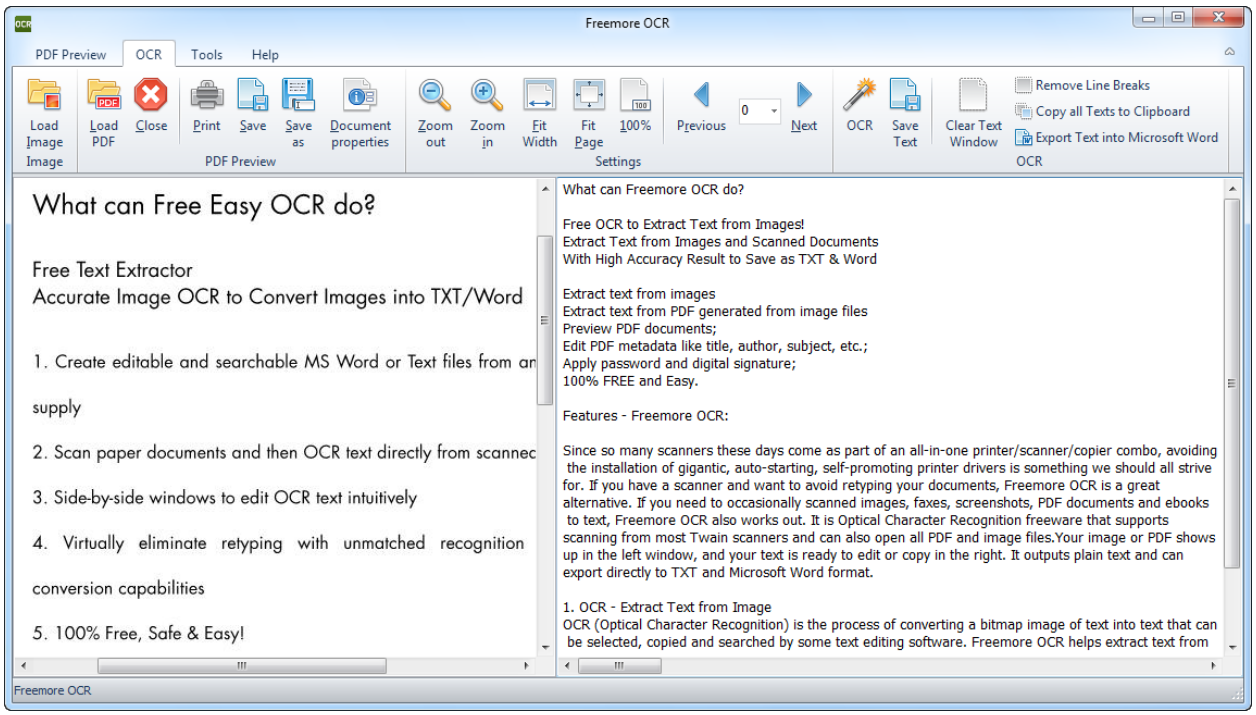

Рисунок 1.4 — Інтерфейс програми Freemore OCR

1.5 Висновки з розділу 1

1. Визначено проблему розпізнавання рукописних текстів на зображеннях.

2. Описано алгоритм розпізнавання текстів на зображеннях.

3. Розглянуто існуючі рішення проблеми розпізнавання рукописних текстів на зображеннях від різних дослідників, науковців.

4. В результаті аналізу декількох програм для розпізнавання текстів на зображеннях визначено, що більшість з них погано розпізнають українські рукописні тексти. Тому існує необхідність у створенні такої системи розпізнавання тексту на зображенні, яка буде значно краще розпізнавати українські рукописні символи.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ

2.1 Поняття згорткових нейронних мереж

Штучні нейронні мережі (ШНМ) – один із напрямків штучного інтелекту, основним завданням якої є моделювання роботи людського мозку. ШНМ складається з взаємопов'язаних між собою нейронів, які, у свою чергу, є простими процесорами, що працюють паралельно. Місце з'єднання двох нейронів називають синапсом. Кожен нейрон має внутрішній стан, який називається сигналом активації. Вихідні сигнали можуть бути надіслані іншим пристроям. Але, на відміну від природної нейронної мережі, ШНМ спрощена в тисячі та мільйони разів. Біологічні нейрони мають тривимірну структуру, коли як ШНМ лише двовимірну. Це зроблено тому, що комп'ютери просто не в змозі обробити такий потік інформації через брак ресурсів. Варто зазначити, що ШНМ мають функцію самонавчання. Нейронна мережа має два цикли: навчання та функціонування. Етап навчання нейронних мереж, своєю чергою, можна розділити на 2 види: навчання з учителем та навчання без вчителя [21].

В основі навчання нейронної мережі вчителем лежить надання їй навчальної вибірки, яка міститиме вхідні правильні дані. Нейронна мережа порівнюватиме отриманий результат з правильним і, виходячи з цього, змінювати ваги нейронів так, щоб відповідь мінімально відрізнялася від необхідного. Іншими словами, відбувається навчання нейронної мережі виконувати певну функцію, при цьому регулюючи значення (коефіцієнти ваги) між елементами. Мережа коригується на основі порівняння, необхідного та отриманого результатів.

Навчання без вчителя, на відміну від попереднього, полягає в тому, що нейронна мережа отримує лише вхідні дані. Далі мережа спостерігає і поступово починає класифікувати дані, що подаються. Внутрішньо

створюються власні групи, завдяки яким нейронна мережа починає впізнавати і відносити дані до тієї чи іншої власної вибірки.

Саме завдяки самонавчанню нейронної мережі стає можливим розпізнавати тексти різних складнощів. У разі розпізнавання тексту застосовуються згорткові нейронні мережі (ЗНМ). На даний момент згорткові нейронні мережі використовуються для розпізнавання рукописного введення, візуальних об'єктів, символів і так далі. На самому базовому рівні згорткова нейронна мережа є просто багаторівневою ієрархічною нейронною мережею.

Згорткова нейронна мережа (англ. convolutional neural network, CNN) – спеціальний вид нейронної мережі, націлений на ефективне розпізнавання образів, запропонований Яном Лекуном 1988 року [21]. На даний момент згорткова нейронна мережа входить до складу технологій глибокого навчання, а її модифікації вважаються найкращими за точністю та швидкістю алгоритмами знаходження об'єктів на сцені. Так, починаючи з 2012 року, згорткові нейромережі займають перші місця на міжнародному конкурсі з розпізнавання образів ImageNet Large Scale Visual Recognition Challenge, в рамках якого різноманітні програмні продукти щорічно змагаються у класифікації та розпізнаванні об'єктів та сцен у базі даних ImageNet [22].

Завдання класифікації – це завдання віднесення об'єкта до одного з попередньо визначених класів виходячи з низки ознак. Кожен з об'єктів у цій задачі представляється у вигляді вектора в N-мірному просторі, будь-який вимір у якому є описом однієї з ознак об'єкта.

У завданнях машинного навчання вхід – багатомірний масив даних (тензор), а ядро згортки – багатомірний масив параметрів, що потребує навчання. Якщо на вході є двовимірне зображення I та ядро K, то операція згортки виглядає наступним чином [21]:

$$s(i, j) = I * K > (i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

Важливо розуміти, що згортка – це лінійна операція, а зображення далекі за своєю природою від лінійності. Нелінійні шари часто розміщуються

безпосередньо після згорткового шару, щоб внести нелінійність до карти активації. Існує кілька типів нелінійних операцій, найпопулярніші з яких:

ReLu: $f(x) = \max(0, x)$

Сигмоїда: $f(x) = 1 / (1 + e^{-x})$

Гіперболічний тангенс: $f(x) = (e^{2x} - 1) / (e^{2x} + 1)$

Щоб знизити час навчання нейронної мережі, після згорткових шарів застосовують підвибірковий шар (пулінг), який зменшує розмірність вихідного зображення. Найпоширеніші види пулінгів – це об'єднання по максимуму (max pooling) і усереднення прямокутної області (average pooling).

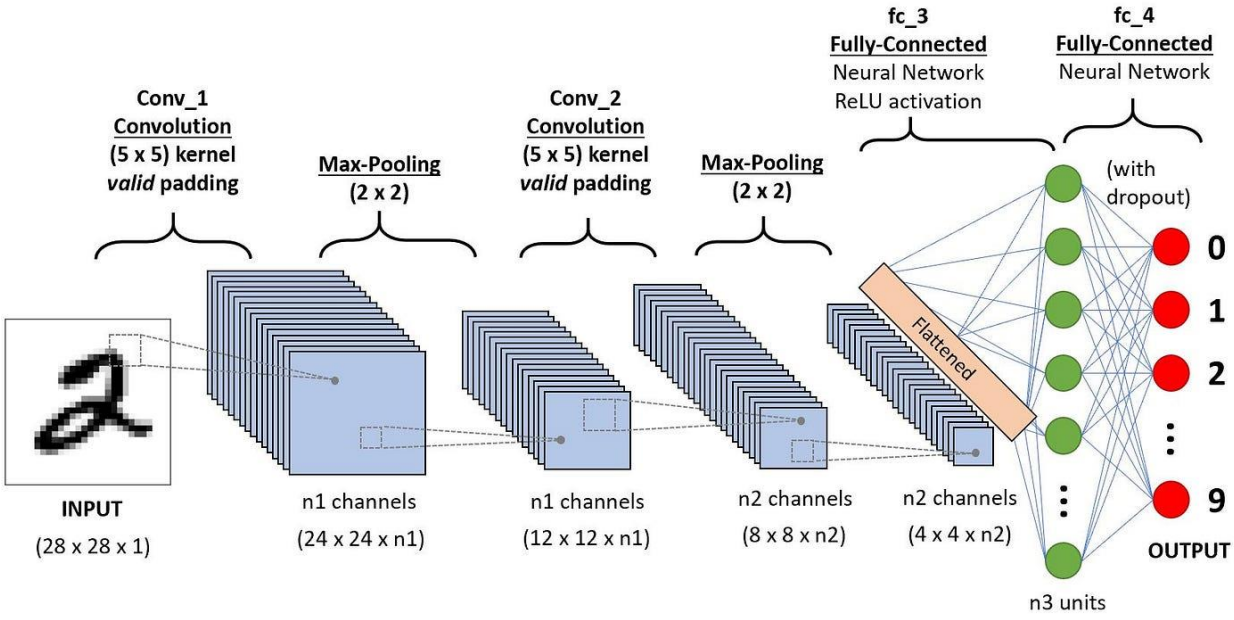

Рисунок 2.1 — Типова архітектура ЗНМ

Згорткові нейронні мережі користуються тим, що вхідні дані складаються з зображень, і вони обмежують побудову мережі розумнішим шляхом. Шари згорткових нейронних мереж складаються з нейронів, розташованих у 3-х вимірах (ширина, висота, глибина). На відміну від простих нейронних мереж, граничні ваги в мережі розподіляються між різними нейронами у прихованих шарах. Щоб розпочати розпізнавання тексту, попередньо необхідно обробити зображення, скориставшись відповідним модулем.

Попередня обробка пов'язана з підготовкою зображення для подальшого аналізу та використання. У деяких випадках може знадобитися покращення якості, усунення різних шумів. Для цього зображення спрощуються, покращуються, змінюються. Перший крок у попередній обробці полягає в тому, щоб перетворити кольорове зображення на чорно-біле. На наступному етапі застосовується фільтрація. Зрештою, попередня обробка зображення зводиться до виконання наступних операцій: зображення у чорно-білому форматі; фільтрація; виявлення даних; вилучення даних [21].

2.2 Огляд фреймворків та бібліотек глибокого навчання

2.2.1 Фреймворк Tensorflow

Говорячи про глибоке навчання, **TensorFlow** часто згадується в першу чергу. Ця дуже популярна платформа використовується не тільки Google – компанією, відповідальною за її створення, але й іншими компаніями, такими як Dropbox, eBay, Airbnb, Nvidia та багатьма іншими.

TensorFlow можна використовувати для розробки API високого та низького рівня, що дозволяє запускати програми практично на будь-якому пристрої. Хоча Python є його основною мовою, до інтерфейсу Tensoflow можна отримати доступ та керувати ним за допомогою інших мов програмування, таких як C++, Java, Julia та JavaScript.

Будучи відкритим вихідним кодом, TensorFlow дозволяє вам виконувати кілька інтеграцій з іншими API та отримувати швидку підтримку та оновлення від спільноти. Його залежність від статичних графіків для обчислень дозволяє вам виконувати негайні обчислення або зберігати операції для доступу в інший час. Ці причини, а також можливість «спостерігати» за розвитком вашої нейронної мережі через TensorBoard, роблять TensorFlow найпопулярнішим середовищем для глибокого навчання.

```python
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

```python
class MyModel(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable
optimizer.apply_gradients(zip(grads, model.traina
```

Рисунок 2.2 — Приклад використання TensorFlow

2.2.2 Фреймворк PyTorch

PyTorch це платформа, розроблена Facebook для підтримки роботи своїх сервісів. З того часу, як цей фреймворк став відкритим, він використовувався іншими компаніями, окрім Facebook, такими як Salesforce та Udacity.

Ця структура працює з динамічно оновлюваними графіками, що дозволяє вносити зміни в архітектуру вашого набору даних у міру його обробки. З PyTorch простіше розробити та навчити нейронну мережу навіть без досвіду глибокого навчання.

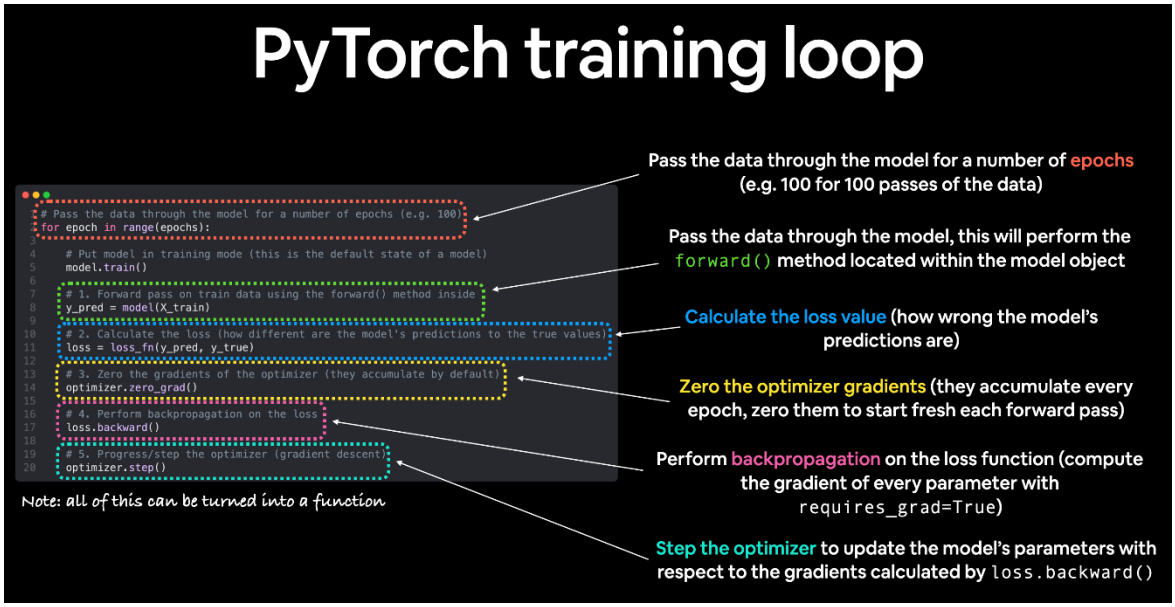

Рисунок 2.3 — Навчання моделі НМ в PyTorch

Будучи відкритим вихідним кодом та заснованим на Python, ви можете легко та швидко інтегруватися у PyTorch. Це також проста структура для вивчення, використання та налагодження. Якщо у вас є питання, ви можете розраховувати на відмінну підтримку та оновлення від обох спільнот – спільноти Python та спільноти PyTorch.

2.2.3 Фреймворк Apache MXNet

Завдяки високій масштабованості, високій продуктивності, швидкому усуненню неполадок та розширеній підтримці графічних процесорів ця платформа була створена Apache для використання у великих промислових проектах.

MXNet включає інтерфейс Gluon, який дозволяє розробникам всіх рівнів кваліфікації почати з глибокого навчання в хмарі, на периферійних пристроях і в мобільних додатках. Всього за кілька рядків коду Gluon ви можете побудувати лінійну регресію, згорткові мережі та рекурентні LSTM для виявлення об'єкта, розпізнавання мови, рекомендації та персоналізація.

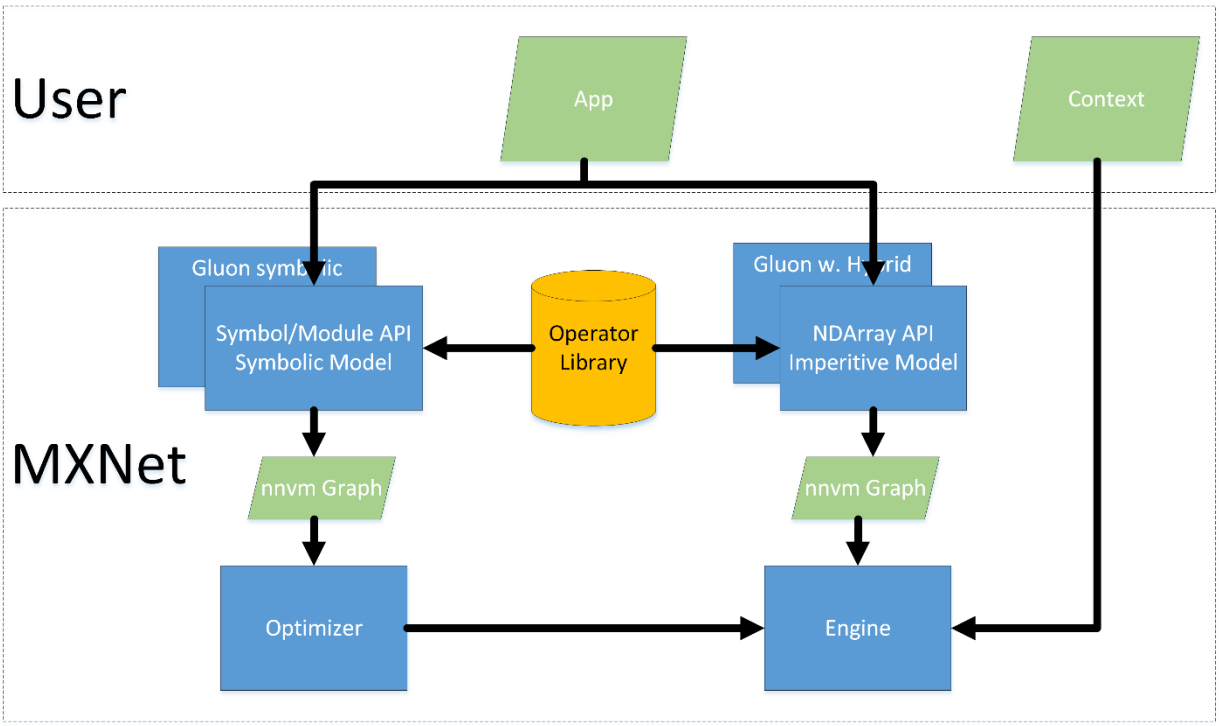

Рисунок 2.4 — Принцип роботи фреймворка MXNet

MXNet може використовуватися на різних пристроях і підтримується кількома мовами програмування, такі як Java, R, JavaScript, Scala та Go. Незважаючи на те, що кількість користувачів та учасників у спільноті невелика, MXNet має добре написану документацію та великий потенціал для зростання, особливо зараз, коли Amazon вибрала цю платформу як основний інструмент для машинного навчання на AWS.

2.2.4 Фреймворк Microsoft Cognitive Toolkit

Якщо ви плануєте розробляти програми або служби, що працюють в Azure (хмарні служби Microsoft), Microsoft Cognitive Toolkit – це платформа, яку слід вибрати для ваших проектів глибокого навчання. Це відкритий вихідний код, який підтримують такі мови програмування, як Python, C++, C#, Java та інші. Цей фреймворк призначений для того, щоб «думати як людський мозок», тому він може обробляти великі обсяги неструктурованих даних, пропонуючи при цьому швидке навчання та інтуїтивно зрозумілу архітектуру.

Обравши цей фреймворк - той же, що стоїть за Skype, Xbox і Cortana - ви отримаєте хорошу продуктивність ваших програм, масштабованість і просту інтеграцію з Azure. Однак у порівнянні з TensorFlow або PyTorch кількість членів спільноти та підтримки менша.

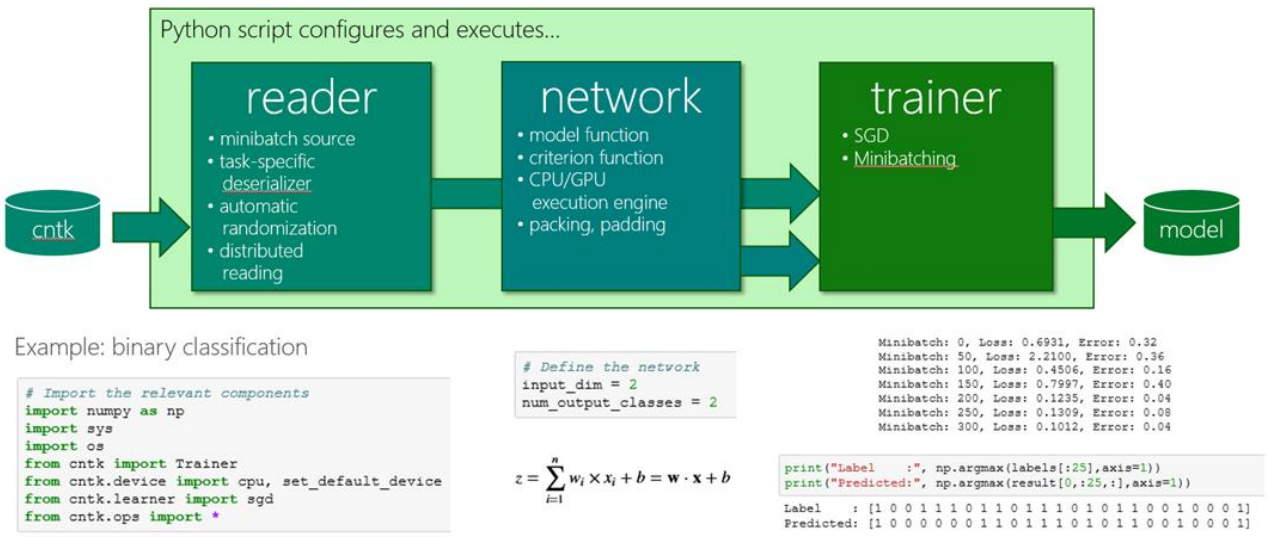

Рисунок 2.5 — Використання Microsoft Cognitive Toolkit

2.2.5 Бібліотека Keras

Як і PyTorch, **Keras** – це бібліотека на основі Python для проектів з інтенсивним використанням даних. API-інтерфейс keras працює на високому рівні та дозволяє інтегруватися з API-інтерфейсами низького рівня, такими як TensorFlow, Theano та Microsoft Cognitive Toolkit.

Деякими перевагами використання keras є його простота в освоєнні — це середовище, що рекомендується для початківців у галузі глибокого навчання; його швидкість розгортання; має велику підтримку з боку спільноти python та спільнот інших фреймворків, з якими він інтегрований.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Рисунок 2.6 — Створення моделі нейронної мережі в Keras

Keras містить різні реалізації будівельних блоків нейронних мереж, таких як шари, цільові функції, функції активації та математичні оптимізатори. Його код розміщено на GitHub, є форуми та канал підтримки Slack. Крім підтримки стандартних нейронні мережі, Keras пропонує підтримку згорткових нейронних мереж та рекурентних нейронних мереж.

Keras дозволяє створювати моделі глибокого навчання на смартфонах з iOS та Android, на віртуальній машині Java або в Інтернеті. Він також дозволяє використовувати розподілене навчання моделей глибокого навчання на кластерах графічних процесорів (GPU) та тензорних процесорів (TPU).

2.2.6 Фреймворк Core ML

Core ML був розроблений Apple для підтримки своєї екосистеми - IOS, Mac OS та iPad OS. Його API працює на низькому рівні, ефективно використовуючи ресурси ЦП і ДП, що дозволяє створеним моделям та додаткам продовжувати працювати навіть без підключення до Інтернету, що знижує «займану пам'ять» та енергоспоживання пристрою.

Спосіб, яким Core ML досягає цього, полягає не в тому, щоб створити ще одну бібліотеку машинного навчання, оптимізовану для роботи на iphone/ipad. Натомість Core ML більше схожий на компілятор, який бере специфікації моделі та навчені параметри, виражені за допомогою іншого програмного забезпечення для машинного навчання, і перетворює їх на файл, який стає ресурсом для програми iOS. Це перетворення в модель Core ML відбувається під час розробки програми, а не в режимі реального часу, коли використовується, і забезпечується бібліотекою Python coremltools.

Рисунок 2.7 — Розпізнавання об’єктів в Core ML

Core ML забезпечує високу продуктивність завдяки простій інтеграції навчання за допомогою машини моделі додатків. Він підтримує глибоке навчання з більш ніж 30 типами шарів, а також деревами рішень, машинами опорних векторів та методами лінійної регресії, і всі вони побудовані на основі низькорівневих технологій, таких як Metal та Accelerate.

2.2.7 Фреймворк ONNX

Цей фреймворк з'явився в результаті співпраці між Microsoft та Facebook з метою спростити процес перенесення та побудови моделей між різними фреймворками, інструментами, середовищами виконання та компіляторами.

ONNX визначає загальний тип файлу, який може працювати на декількох платформах, використовуючи при цьому переваги низькорівневих інтерфейсів API, таких як Microsoft Cognitive Toolkit, MXNet, Caffe і (з використанням конвертерів) Tensorflow і Core ML. Принцип ONNX полягає у навчанні моделі в стеку та її реалізації з використанням інших висновків та прогнозів.

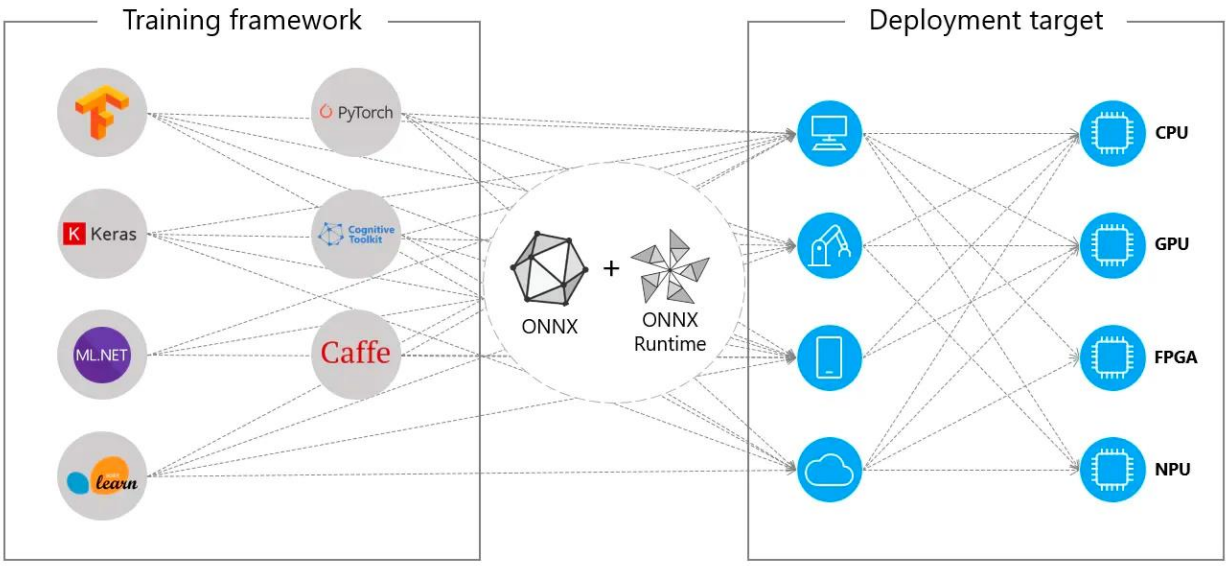

Рисунок 2.8 — Принцип роботи ONNX

LF AI Foundation, дочірня організація Linux Foundation, є організацією, що займається створенням екосистеми для підтримки з відкритим вихідним кодом інновації в галузі штучного інтелекту (ШІ), машинного навчання (МО) і глибокого навчання (ГО). 14 листопада 2019 року він додав ONNX як проект для випускників. Цей перехід ONNX під егіду LF AI Foundation був сприйнятий як важлива віха перетворення ONNX на незалежний від постачальників стандарт відкритого формату.

Зоопарк моделей ONNX – це колекція попередньо вивчених моделей глибокого навчання, доступних у форматі ONNX. Для кожної моделі є ноутбуки Jupyter для навчання моделі та виконання висновків з навченою

моделлю. Нотатки написані на Python і містять посилання на набір навчальних даних та посилання на оригінальний науковий документ, що описує архітектуру моделі.

У результаті порівняння фреймворків та бібліотек для розробки згорткової нейронної мережі було обрано високорівневу бібліотеку Keras, створену на основі TensorFlow.

2.3 Датасет із зображеннями українських рукописних символів

Cyrillic_ukr [23] – це датасет (набір даних) для навчання нейронної мережі, який містить до 15000 зображень українських рукописних символів в форматі PNG розміру 28*28 пікселів. Зображення буде розділено на навчальну та валідаційну вибірку під час навчання ЗНМ.

Рисунок 2.2 — Зображення в датасеті

Середня точність розпізнавання зростатиме зі збільшенням кількості навчальних зображень, оскільки більша кількість зображень у навчанні дає більш точну інформацію про параметри навчання, що згодом покращує точність класифікації під час фази тестування. Планується, що через певну кількість епох тренування точність буде досягати приблизно 97% для датасету

Cyrillic_ukr. Подальше збільшення кількості навчальних зображень продовжить підвищувати точність до певної межі, яка не може бути перевищена через числові помилки та обмеження на здатність ЗНМ розрізняти зображення для міток.

Завантажити датасет можна з офіційного сайту Kaggle.

2.4 Висновки з розділу 2

1. Визначено поняття згорткових нейронних мереж та описано їх структуру.

2. Розглянуто найпопулярніші фреймворки та бібліотеки глибокого навчання. Для розробки системи розпізнавання рукописних текстів на зображеннях було обрано високорівневу бібліотеку Keras, створену на основі TensorFlow.

3. У якості датасета для навчання нейронної мережі було обрано Cyrillic_ukr, що містить до 15000 зображень українських рукописних символів в форматі PNG розміру 28*28 пікселів

РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ

3.1 Встановлення та налаштування середовища для розробки системи розпізнавання рукописних текстів на зображеннях

Для розробки системи розпізнавання рукописного тексту з використанням бібліотеки Keras було обрано програму PyCharm, яку можна завантажити з сайту jetbrains.com. Для завантаження можна обрати пробну версію Professional або повністю безкоштовну версію Community.

Після встановлення та запуску програми відкриється вікно, у якому можна створити новий проект або відкрити наявний, налаштувати програму та встановити необхідні плагіни.

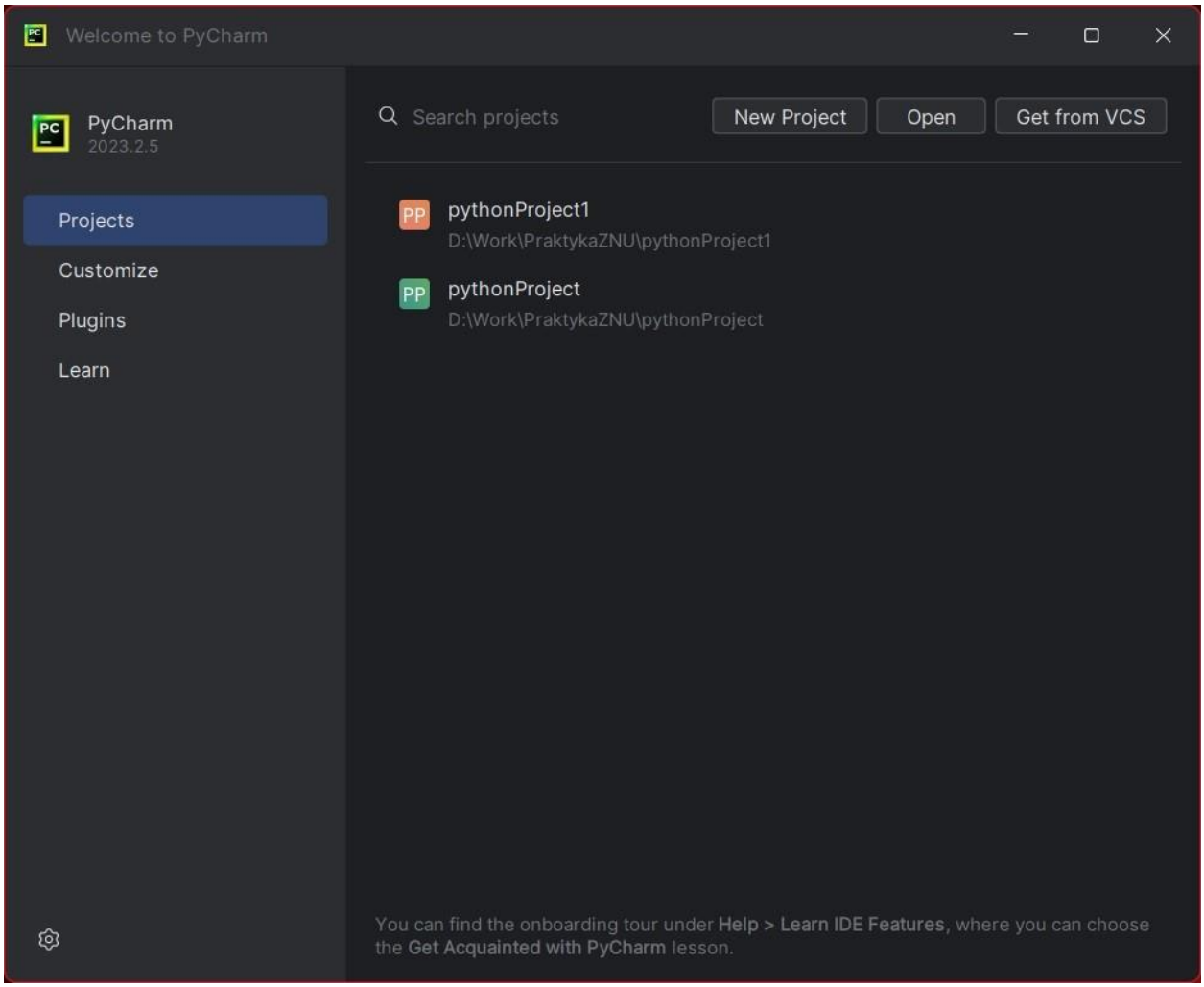

Рисунок 3.1 — Стартове вікно PyCharm

Далі натискаємо “New Project”, називаємо проект та обираємо місце збереження на диску. Потім створюємо віртуальне середовище (Virtualenv), для якого також потрібно задати місце збереження та обрати Python 3.7 як інтерпретатор.

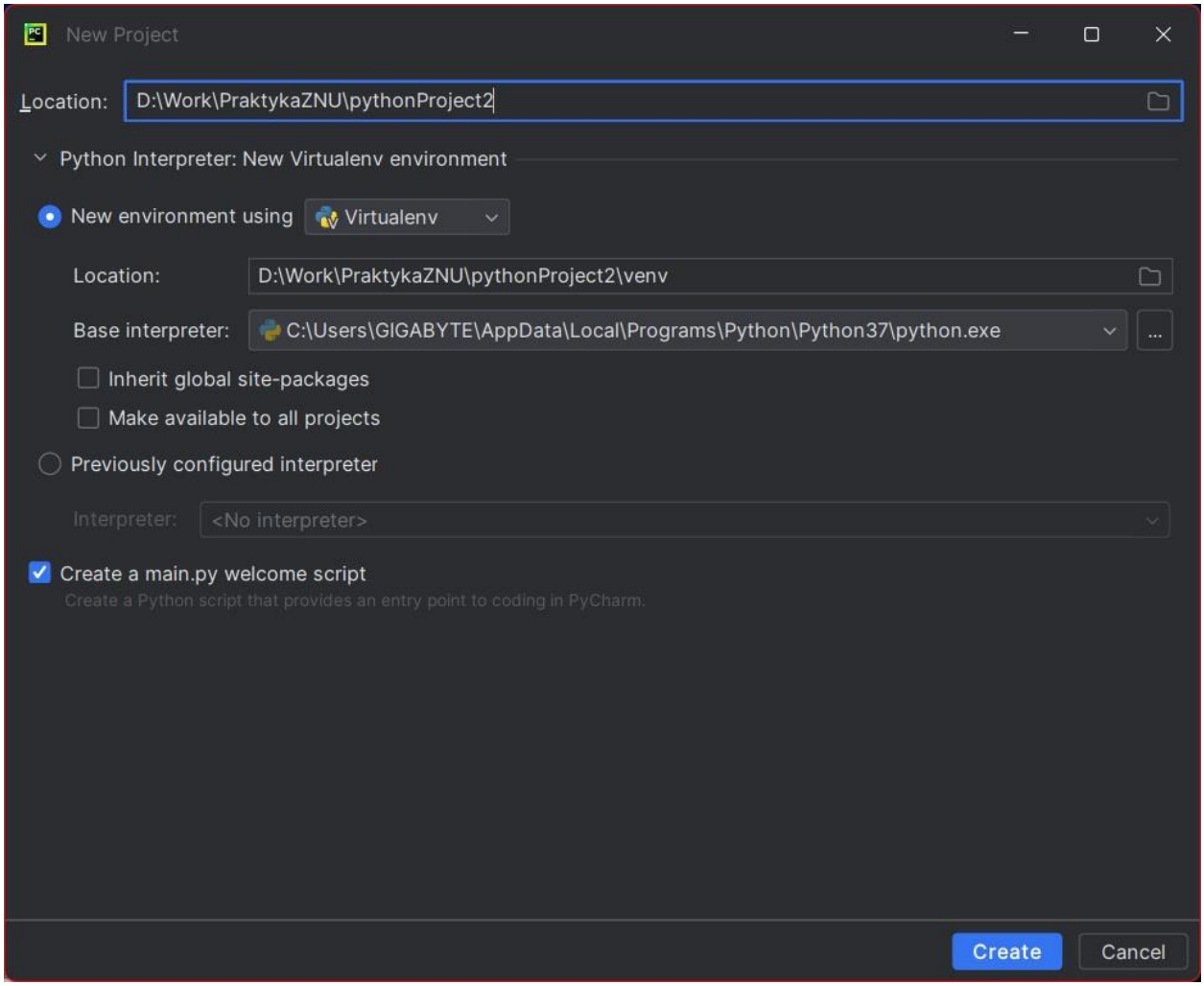

Рисунок 3.2 — Створення нового проекту

Після створення нового проекту додаємо нову папку і завантажуємо до неї датасет так, щоб структура проекту виглядала як на рис. 3.3.

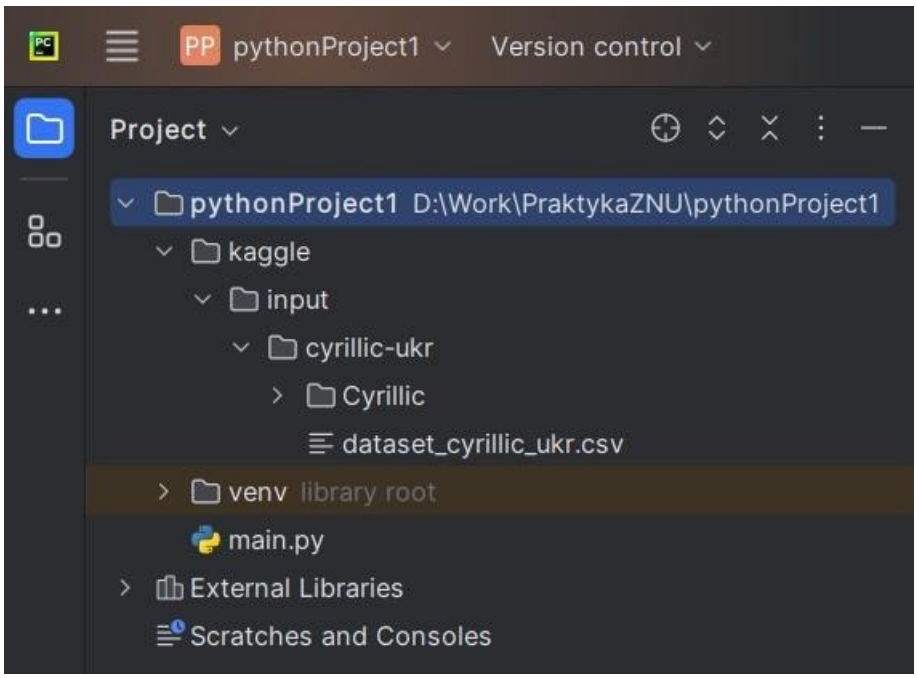

Рисунок 3.3 — Структура нового проекту

Тепер все готово для написання коду навчання згорткової нейронної мережі.

3.2 Розробка і навчання згорткової нейронної мережі розпізнавати українські рукописні символи

До початку навчання необхідно імпортувати такі пакети, як matplotlib, numpy, pandas, scikit-learn, tensorflow.

Лістинг 1 демонструє підготовку даних для навчання: оголошення стартових змінних, відкриття файлу dataset_cyrillic_ukr.csv, виділення правильних відповідей, нормалізація, розділ даних на 2 набори – для навчання і для перевірки.

Лістинг 1 – Підготовка даних для навчання

```
# Стартові змінні
x_train = pd.read_csv('kaggle/input/cyrillic-ukr/dataset_cyrillic_ukr.csv')
```

```
d = {'А': 0, 'Б': 1, 'В': 2, 'Г': 3, 'Ґ': 4, 'Д': 5, 'Е': 6, 'Є': 7, 'Ж': 8, 'З': 9, 'И': 10, 'І': 11, 'Ї': 12, 'Й': 13,
    'К': 14, 'Л': 15, 'М': 16, 'Н': 17, 'О': 18, 'П': 19, 'Р': 20, 'С': 21, 'Т': 22, 'У': 23, 'Ф': 24, 'Х': 25,
    'Ц': 26, 'Ч': 27, 'Ш': 28, 'Щ': 29, 'Ь': 30, 'Ю': 31, 'Я': 32}
input_shape = (28, 28, 1)

x_train.head()
# Виділення правильних відповідей
y_train = x_train['7']
del x_train['7']
y_train = utils.to_categorical(y_train)

# Розділ масиву на дві частини
x_train = numpy.array(x_train)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)  # Convert to 2D image format
x_train = x_train.astype(numpy.float32)
x_train /= 255.0

# Розділ даних на два набори - для навчання та перевірки
# test_size - об'єм набору для перевірки (10%)
# X_train - набір для навчання
# X_val - набір для перевірки
# Y_train - правильні відповіді для навчання
# Y_val - правильні відповіді для перевірки
random_seed = 2
```

```
X_train, X_val, Y_train, Y_val = train_test_split(x_train, y_train, test_size=0.1, random_state=random_seed)

# Створення генератора з поворотом, збільшенням та зсувом зображень
datagen = ImageDataGenerator(rotation_range=10, zoom_range=0.1, width_shift_range=0.1, height_shift_range=0.1)
```

Лістинг 2 демонструє створення нейронної мережі:

1. Побудова згорткової частини.
2. Побудова повнозв’язної частини.
3. Компіляція нейронної мережі.
4. Навчання ЗНМ.

Лістинг 2 – Навчання нейронної мережі

```
model = Sequential()
# Перший блок складається з двох згорткових шарів,
# на кожному по 32 карти ознак, розмір ядра згортки - 2 на 2
model.add(Conv2D(filters=64, kernel_size=(2, 2), padding='Same', activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(2, 2), padding='Same', activation='relu'))
# Шар підвибірки з вибором максимального значення в квадраті 2 на 2
model.add(MaxPooling2D(pool_size=(2, 2)))
# Шар, який вимикає нейрони з 25 відсотковою вірогідністю для позбавлення перенавчання
model.add(Dropout(0.25))
# Другий блок згортки, аналогічний першому
model.add(Conv2D(filters=64, kernel_size=(2, 2), padding='Same', activation='relu'))
```

```
model.add(Conv2D(filters=64, kernel_size=(2, 2), padding='Same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.25))

# Шар для перетворення двомірного виходу згорткової частини в одновимірний масив
model.add(Flatten())
# Повнозв'язний шар на 56 нейронів
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
# Вихід - повнозв'язний шар на 33 нейрони (кількість літер в алфавіті)
model.add(Dense(33, activation='softmax'))

# Активація функції втрат - категоріальна перехресна ентропія
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# Навчання НМ з використанням двох колбеків
# Перший колбек відповідає за збереження кращого варіанта мережі
checkpoint = ModelCheckpoint('mnist_cnn.hdf5', monitor='val_accuracy', save_best_only=True, verbose=1)
# Другий колбек - поділ швидкості навчання на 2 за умови, що за 3 ітерації не змінюється точність на валідаційному наборі
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience=3, verbose=1, factor=0.5, min_lr=0.00001)
# Розмір мінівибірки - 96 зображень
batch_size = 96
```

```
# Навчання НМ
history = model.fit(datagen.flow(X_train, Y_train, batch_size=batch_size), epochs=30,
                    validation_data=(X_val, Y_val), steps_per_epoch=X_train.shape[0] // batch_size, verbose=1,
                    callbacks=[checkpoint, learning_rate_reduction])
```

Процес навчання згорткової нейронної мережі відбувається наступним чином:

```
Epoch 1/30
138/138 [==============================] - ETA: 0s - loss: 2.3570 - accuracy: 0.3340
Epoch 1: val_accuracy improved from -inf to 0.71805, saving model to mnist_cnn.hdf5
138/138 [==============================] - 12s 84ms/step - loss: 2.3570 - accuracy: 0.3340 - val_loss: 0.8746 - val_accuracy: 0.7181 - lr: 0.0010
Epoch 2/30
138/138 [==============================] - ETA: 0s - loss: 1.1920 - accuracy: 0.6408
Epoch 2: val_accuracy improved from 0.71805 to 0.84855, saving model to mnist_cnn.hdf5
138/138 [==============================] - 12s 85ms/step - loss: 1.1920 - accuracy: 0.6408 - val_loss: 0.4885 - val_accuracy: 0.8485 - lr: 0.0010
Epoch 3/30
138/138 [==============================] - ETA: 0s - loss: 0.8591 - accuracy: 0.7325
```

Epoch 3: val_accuracy improved from 0.84855 to 0.89182, saving model to mnist_cnn.hdf5

138/138 [==============================] - 13s 93ms/step - loss: 0.8591 - accuracy: 0.7325 - val_loss: 0.3570 - val_accuracy: 0.8918 - lr: 0.0010

Epoch 4/30

138/138 [==============================] - ETA: 0s - loss: 0.7082 - accuracy: 0.7776

Epoch 4: val_accuracy improved from 0.89182 to 0.91954, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 105ms/step - loss: 0.7082 - accuracy: 0.7776 - val_loss: 0.2869 - val_accuracy: 0.9195 - lr: 0.0010

Epoch 5/30

138/138 [==============================] - ETA: 0s - loss: 0.6095 - accuracy: 0.8052

Epoch 5: val_accuracy improved from 0.91954 to 0.92495, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 105ms/step - loss: 0.6095 - accuracy: 0.8052 - val_loss: 0.2565 - val_accuracy: 0.9249 - lr: 0.0010

Epoch 6/30

138/138 [==============================] - ETA: 0s - loss: 0.5476 - accuracy: 0.8300

Epoch 6: val_accuracy improved from 0.92495 to 0.93306, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 105ms/step - loss: 0.5476 - accuracy: 0.8300 - val_loss: 0.2238 - val_accuracy: 0.9331 - lr: 0.0010

Epoch 7/30

138/138 [==============================] - ETA: 0s - loss: 0.4973 - accuracy: 0.8422

Epoch 7: val_accuracy improved from 0.93306 to 0.93847, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 105ms/step - loss: 0.4973 - accuracy: 0.8422 - val_loss: 0.2253 - val_accuracy: 0.9385 - lr: 0.0010

Epoch 8/30

138/138 [==============================] - ETA: 0s - loss: 0.4651 - accuracy: 0.8524

Epoch 8: val_accuracy improved from 0.93847 to 0.94388, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 107ms/step - loss: 0.4651 - accuracy: 0.8524 - val_loss: 0.1701 - val_accuracy: 0.9439 - lr: 0.0010

Epoch 9/30

138/138 [==============================] - ETA: 0s - loss: 0.4237 - accuracy: 0.8633

Epoch 9: val_accuracy did not improve from 0.94388

138/138 [==============================] - 14s 105ms/step - loss: 0.4237 - accuracy: 0.8633 - val_loss: 0.1855 - val_accuracy: 0.9425 - lr: 0.0010

Epoch 10/30

138/138 [==============================] - ETA: 0s - loss: 0.3955 - accuracy: 0.8724

Epoch 10: val_accuracy did not improve from 0.94388

138/138 [==============================] - 15s 106ms/step - loss: 0.3955 - accuracy: 0.8724 - val_loss: 0.1778 - val_accuracy: 0.9412 - lr: 0.0010

Epoch 11/30

138/138 [==============================] - ETA: 0s - loss: 0.3786 - accuracy: 0.8785

Epoch 11: val_accuracy improved from 0.94388 to 0.94726, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 107ms/step - loss: 0.3786 - accuracy: 0.8785 - val_loss: 0.1634 - val_accuracy: 0.9473 - lr: 0.0010

Epoch 12/30

138/138 [==============================] - ETA: 0s - loss: 0.3482 - accuracy: 0.8871

Epoch 12: val_accuracy did not improve from 0.94726

138/138 [==============================] - 15s 105ms/step - loss: 0.3482 - accuracy: 0.8871 - val_loss: 0.1578 - val_accuracy: 0.9466 - lr: 0.0010

Epoch 13/30

138/138 [==============================] - ETA: 0s - loss: 0.3410 - accuracy: 0.8888

Epoch 13: val_accuracy improved from 0.94726 to 0.95470, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 107ms/step - loss: 0.3410 - accuracy: 0.8888 - val_loss: 0.1508 - val_accuracy: 0.9547 - lr: 0.0010

Epoch 14/30

138/138 [==============================] - ETA: 0s - loss: 0.3139 - accuracy: 0.8977

Epoch 14: val_accuracy improved from 0.95470 to 0.95943, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 105ms/step - loss: 0.3139 - accuracy: 0.8977 - val_loss: 0.1394 - val_accuracy: 0.9594 - lr: 0.0010

Epoch 15/30

138/138 [==============================] - ETA: 0s - loss: 0.3146 - accuracy: 0.8996

Epoch 15: val_accuracy did not improve from 0.95943

138/138 [==============================] - 15s 106ms/step - loss: 0.3146 - accuracy: 0.8996 - val_loss: 0.1444 - val_accuracy: 0.9567 - lr: 0.0010

Epoch 16/30

138/138 [==============================] - ETA: 0s - loss: 0.2917 - accuracy: 0.9090

Epoch 16: val_accuracy did not improve from 0.95943

138/138 [==============================] - 15s 106ms/step - loss: 0.2917 - accuracy: 0.9090 - val_loss: 0.1338 - val_accuracy: 0.9594 - lr: 0.0010

Epoch 17/30

138/138 [==============================] - ETA: 0s - loss: 0.2889 - accuracy: 0.9070

Epoch 17: val_accuracy did not improve from 0.95943

Epoch 17: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

138/138 [==============================] - 15s 106ms/step - loss: 0.2889 - accuracy: 0.9070 - val_loss: 0.1328 - val_accuracy: 0.9561 - lr: 0.0010

Epoch 18/30

138/138 [==============================] - ETA: 0s - loss: 0.2532 - accuracy: 0.9172

Epoch 18: val_accuracy improved from 0.95943 to 0.96349, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 106ms/step - loss: 0.2532 - accuracy: 0.9172 - val_loss: 0.1202 - val_accuracy: 0.9635 - lr: 5.0000e-04

Epoch 19/30

138/138 [==============================] - ETA: 0s - loss: 0.2346 - accuracy: 0.9249

Epoch 19: val_accuracy did not improve from 0.96349

138/138 [==============================] - 15s 108ms/step - loss: 0.2346 - accuracy: 0.9249 - val_loss: 0.1237 - val_accuracy: 0.9635 - lr: 5.0000e-04

Epoch 20/30

138/138 [==============================] - ETA: 0s - loss: 0.2406 - accuracy: 0.9205

Epoch 20: val_accuracy improved from 0.96349 to 0.96416, saving model to mnist_cnn.hdf5

138/138 [==============================] - 16s 115ms/step - loss: 0.2406 - accuracy: 0.9205 - val_loss: 0.1128 - val_accuracy: 0.9642 - lr: 5.0000e-04

Epoch 21/30

138/138 [==============================] - ETA: 0s - loss: 0.2236 - accuracy: 0.9258

Epoch 21: val_accuracy did not improve from 0.96416

138/138 [==============================] - 16s 113ms/step - loss: 0.2236 - accuracy: 0.9258 - val_loss: 0.1160 - val_accuracy: 0.9628 - lr: 5.0000e-04

Epoch 22/30

138/138 [==============================] - ETA: 0s - loss: 0.2259 - accuracy: 0.9266

Epoch 22: val_accuracy improved from 0.96416 to 0.96687, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 106ms/step - loss: 0.2259 - accuracy: 0.9266 - val_loss: 0.1070 - val_accuracy: 0.9669 - lr: 5.0000e-04

Epoch 23/30

138/138 [==============================] - ETA: 0s - loss: 0.2213 - accuracy: 0.9284

Epoch 23: val_accuracy did not improve from 0.96687

138/138 [==============================] - 15s 106ms/step - loss: 0.2213 - accuracy: 0.9284 - val_loss: 0.1086 - val_accuracy: 0.9648 - lr: 5.0000e-04

Epoch 24/30

138/138 [==============================] - ETA: 0s - loss: 0.2022 - accuracy: 0.9339

Epoch 24: val_accuracy did not improve from 0.96687

138/138 [==============================] - 15s 105ms/step - loss: 0.2022 - accuracy: 0.9339 - val_loss: 0.1082 - val_accuracy: 0.9648 - lr: 5.0000e-04

Epoch 25/30

138/138 [==============================] - ETA: 0s - loss: 0.2143 - accuracy: 0.9289

Epoch 25: val_accuracy did not improve from 0.96687

Epoch 25: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

138/138 [==============================] - 15s 108ms/step - loss: 0.2143 - accuracy: 0.9289 - val_loss: 0.1094 - val_accuracy: 0.9648 - lr: 5.0000e-04

Epoch 26/30

138/138 [==============================] - ETA: 0s - loss: 0.1914 - accuracy: 0.9369

Epoch 26: val_accuracy improved from 0.96687 to 0.97093, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 105ms/step - loss: 0.1914 - accuracy: 0.9369 - val_loss: 0.0944 - val_accuracy: 0.9709 - lr: 2.5000e-04

Epoch 27/30

138/138 [==============================] - ETA: 0s - loss: 0.1854 - accuracy: 0.9410

Epoch 27: val_accuracy did not improve from 0.97093

138/138 [==============================] - 15s 106ms/step - loss: 0.1854 - accuracy: 0.9410 - val_loss: 0.1055 - val_accuracy: 0.9675 - lr: 2.5000e-04

Epoch 28/30

138/138 [==============================] - ETA: 0s - loss: 0.1830 - accuracy: 0.9432

Epoch 28: val_accuracy did not improve from 0.97093

```
138/138 [==============================] - 15s 106ms/step - loss: 0.1830 - accuracy: 0.9432 - val_loss: 0.0903 - val_accuracy: 0.9689 - lr: 2.5000e-04
Epoch 29/30
138/138 [==============================] - ETA: 0s - loss: 0.1885 - accuracy: 0.9383
Epoch 29: val_accuracy did not improve from 0.97093

Epoch 29: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
138/138 [==============================] - 15s 106ms/step - loss: 0.1885 - accuracy: 0.9383 - val_loss: 0.0955 - val_accuracy: 0.9689 - lr: 2.5000e-04
Epoch 30/30
138/138 [==============================] - ETA: 0s - loss: 0.1799 - accuracy: 0.9428
Epoch 30: val_accuracy improved from 0.97093 to 0.97228, saving model to mnist_cnn.hdf5
138/138 [==============================] - 15s 106ms/step - loss: 0.1799 - accuracy: 0.9428 - val_loss: 0.0921 - val_accuracy: 0.9723 - lr: 1.2500e-04
```

Отже, найвища точність розпізнавання на валідаційній вибірці становить 97.22%.

Лістинг 3 – Графік результату навчання

```
history_dict = history.history
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
epochs = range(1, len(acc_values) + 1)
plt.plot(epochs, acc_values, 'ro', label='Training acc')
plt.plot(epochs, val_acc_values, 'r', label='Validation acc')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Лістинг 4 – Збереження результату навчання

```
model.save('model_all.h5')
model_json = model.to_json()
with open("model.json", "w") as json_file:
   json_file.write(model_json)
model.save_weights("model_weights.h5")
print("Saved model to disk")
```

3.3 Висновки з розділу 3

1. Було налаштовано середовище для розробки системи розпізнавання рукописних текстів на зображеннях.

2. У налаштоване середовище було завантажено датасет Cyrillic_ukr.

3. Виконано навчання моделі згорткової нейронної мережі розпізнавати українські рукописні літери. Встановлено, що найвища точність розпізнавання на валідаційній вибірці досягає 97.22%.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ РОЗПІЗНАВАННЯ РУКОПИСНИХ ТЕКСТІВ НА ЗОБРАЖЕННЯХ

4.1 Експериментування для одержання найкращого результату навчання ЗНМ

4.1.1 Аналіз результатів навчання нейронної мережі

У процесі розробки системи розпізнавання рукописного тексту на зображенні була навчена згорткова нейронна мережа на датасеті Cyrillic_ukr. Після навчання фреймворк Tensorflow побудував графік залежності точності на навчальній та тестовій вибірці від кількості епох (рис. 4.1).

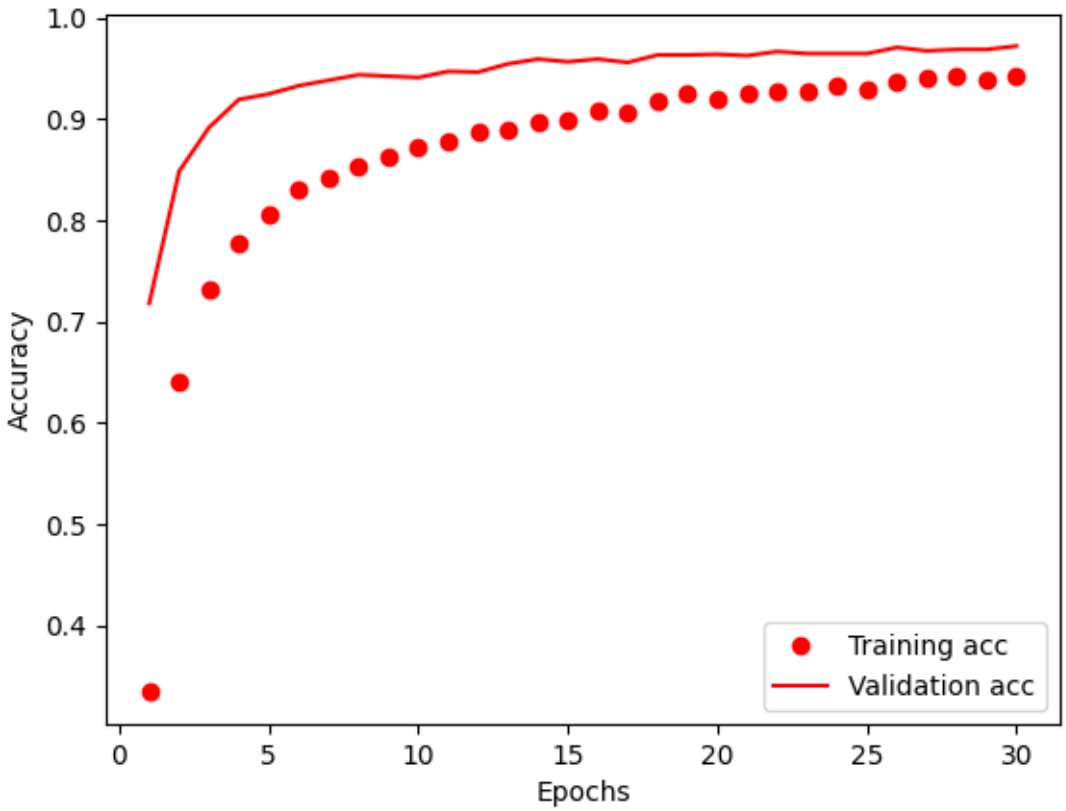

Рисунок 4.1 —Графік навчання нейронної мережі на датасеті Cyrillic_ukr

Як можна побачити на графіку, кінцева точність нейронної мережі на валідаційній вибірці становить 97.22%. Видно, що приблизно до 10 епохи точність швидко зростає, а починаючи з 19 епохи – набагато повільніше.

4.1.2 Способи підвищення точності розпізнавання

З метою підвищення точності нейронної мережі було проведено кілька експериментів, включаючи зміну кількості епох та відсотка валідаційної вибірки.

При збільшенні кількості епох до 40 графік навчання виглядає так:

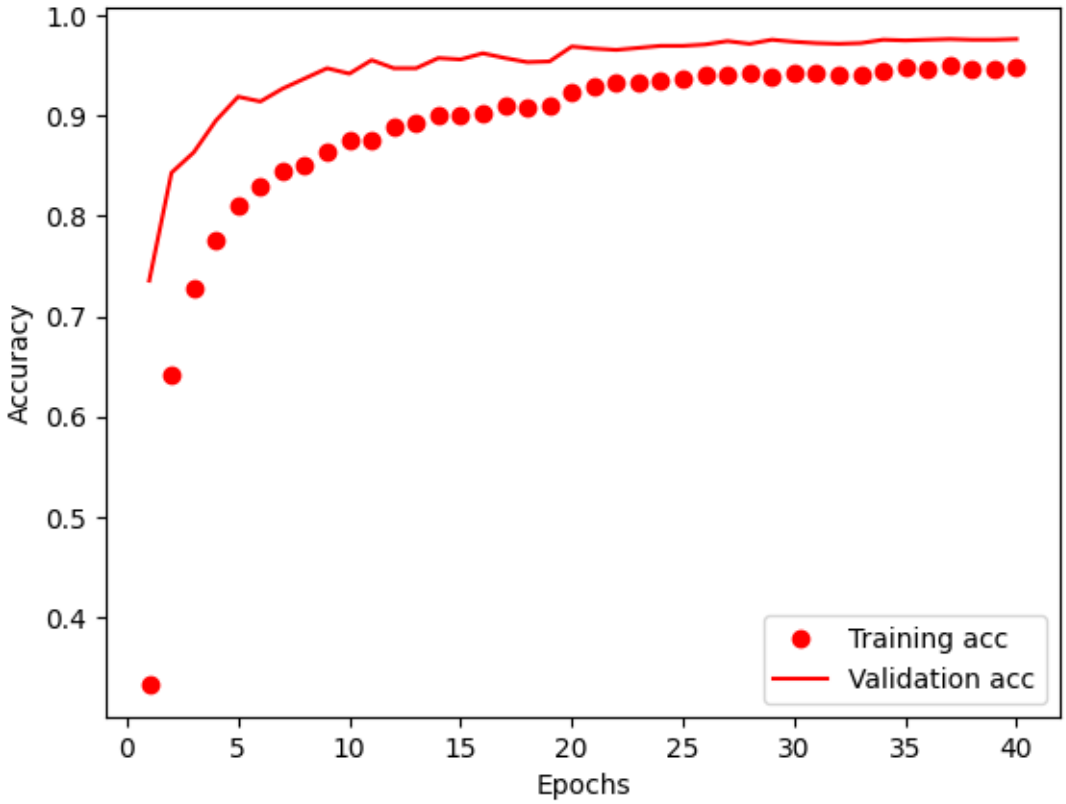

Рисунок 4.2 — Графік навчання нейронної мережі на 40 епох

На графіку видно, що з 16 епохи точність тимчасово спадає, тобто відбувається перенавчання, проте кінцева точність нейронної мережі на валідаційній вибірці покращується до 97.63%.

При збільшенні кількості епох до 50 графік навчання виглядає таким чином:

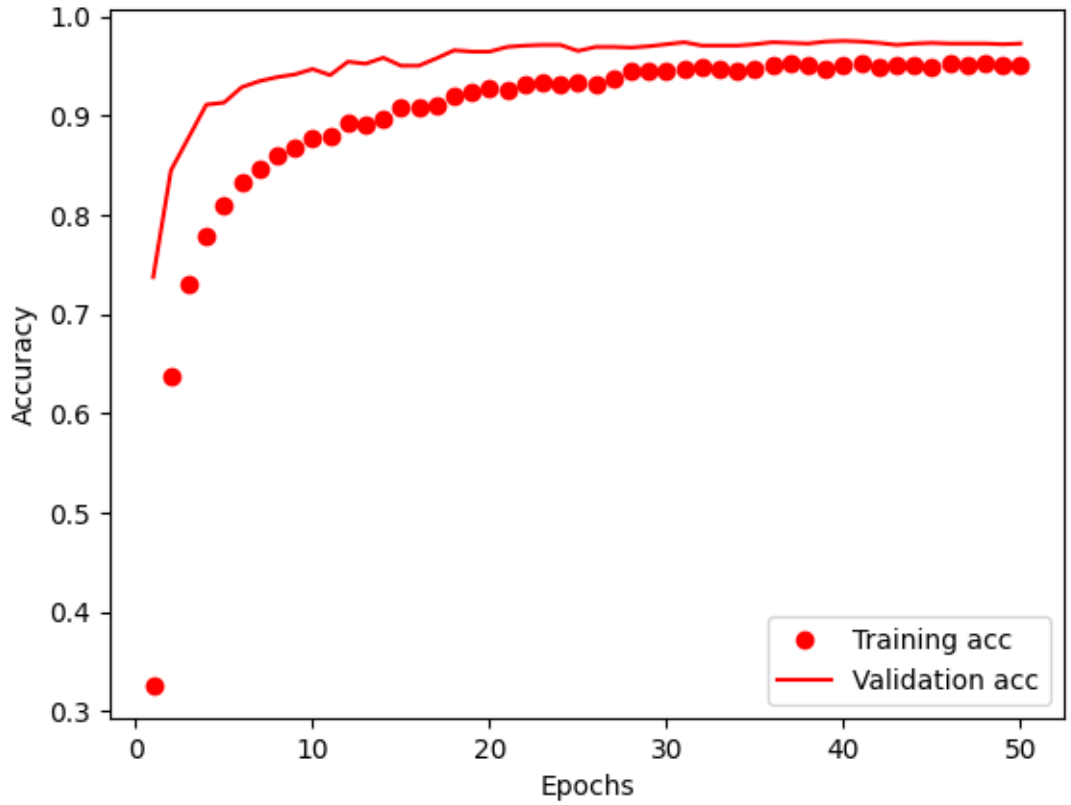

Рисунок 4.3 — Графік навчання нейронної мережі на 50 епох

Як можна побачити на графіку, кінцева точність нейронної мережі на валідаційній вибірці становить 97.56%. Отже, можна зробити висновок, що найкращої точності можна досягти на 40 епохах навчання.

Якщо в новій моделі НМ (на 40 епох) збільшити відсоток валідаційної вибірки з 10% до 15%, графік навчання виглядатиме так:

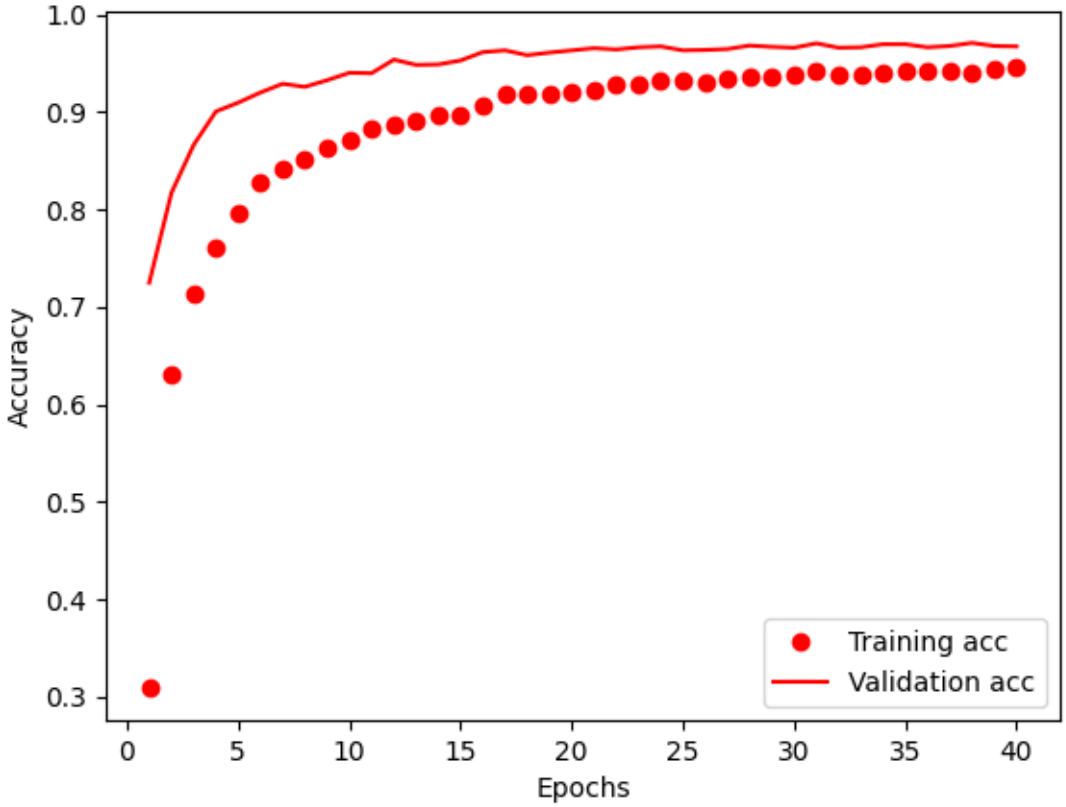

Рисунок 4.4 — Графік навчання НМ на 40 епох та з 15% валідаційної вибірки

На графіку видно, що кінцева точність нейронної мережі на валідаційній вибірці становить 97.11%. Тепер спробуємо зменшити відсоток валідаційної вибірки з 10% до 5% (рис. 4.5).

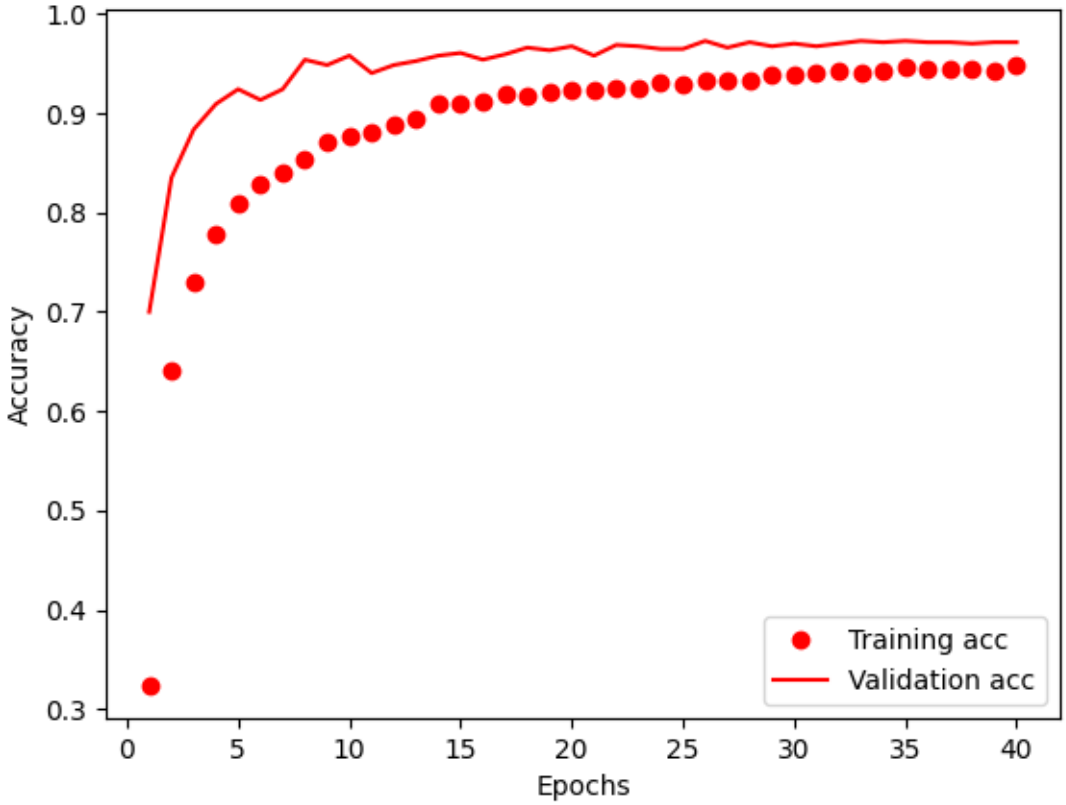

Рисунок 4.4 — Графік навчання НМ на 40 епох та з 5% валідаційної вибірки

Як можна побачити на графіку, кінцева точність нейронної мережі на валідаційній вибірці становить 97.29%. Отже, збільшення чи зменшення відсотка валідаційної вибірки не покращує точність розпізнавання.

Таблиця 4.1— Порівняння точності розпізнавання при зміні кількості епох навчання та відсотка валідаційної вибірки

Валідаційна вибірка	**Кількість епох навчання**		
	30	40	50
10%	97.22%	97.63%	97.56%
15%	-	97.11%	-
5%	-	97.29%	-

В результаті порівняння (табл. 4.1) можна зробити висновок, що нейронна мережа досягає найвищої точності при 40 епохах навчання та 10% валідаційної вибірки.

Наведемо процес навчання покращеної моделі ЗНМ:

Epoch 1/40

138/138 [==============================] - ETA: 0s - loss: 2.3565 - accuracy: 0.3326

Epoch 1: val_accuracy improved from -inf to 0.73563, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 98ms/step - loss: 2.3565 - accuracy: 0.3326 - val_loss: 0.8781 - val_accuracy: 0.7356 - lr: 0.0010

Epoch 2/40

138/138 [==============================] - ETA: 0s - loss: 1.1790 - accuracy: 0.6419

Epoch 2: val_accuracy improved from 0.73563 to 0.84314, saving model to mnist_cnn.hdf5

138/138 [==============================] - 12s 89ms/step - loss: 1.1790 - accuracy: 0.6419 - val_loss: 0.5168 - val_accuracy: 0.8431 - lr: 0.0010

Epoch 3/40

138/138 [==============================] - ETA: 0s - loss: 0.8768 - accuracy: 0.7287

Epoch 3: val_accuracy improved from 0.84314 to 0.86342, saving model to mnist_cnn.hdf5

138/138 [==============================] - 13s 96ms/step - loss: 0.8768 - accuracy: 0.7287 - val_loss: 0.4127 - val_accuracy: 0.8634 - lr: 0.0010

Epoch 4/40

138/138 [==============================] - ETA: 0s - loss: 0.7157 - accuracy: 0.7761

Epoch 4: val_accuracy improved from 0.86342 to 0.89520, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 107ms/step - loss: 0.7157 - accuracy: 0.7761 - val_loss: 0.3151 - val_accuracy: 0.8952 - lr: 0.0010

Epoch 5/40

138/138 [==============================] - ETA: 0s - loss: 0.6006 - accuracy: 0.8098

Epoch 5: val_accuracy improved from 0.89520 to 0.91886, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 108ms/step - loss: 0.6006 - accuracy: 0.8098 - val_loss: 0.2532 - val_accuracy: 0.9189 - lr: 0.0010

Epoch 6/40

138/138 [==============================] - ETA: 0s - loss: 0.5366 - accuracy: 0.8297

Epoch 6: val_accuracy did not improve from 0.91886

138/138 [==============================] - 14s 103ms/step - loss: 0.5366 - accuracy: 0.8297 - val_loss: 0.2816 - val_accuracy: 0.9141 - lr: 0.0010

Epoch 7/40

138/138 [==============================] - ETA: 0s - loss: 0.4928 - accuracy: 0.8454

Epoch 7: val_accuracy improved from 0.91886 to 0.92698, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 105ms/step - loss: 0.4928 - accuracy: 0.8454 - val_loss: 0.2089 - val_accuracy: 0.9270 - lr: 0.0010

Epoch 8/40

138/138 [==============================] - ETA: 0s - loss: 0.4578 - accuracy: 0.8508

Epoch 8: val_accuracy improved from 0.92698 to 0.93712, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 103ms/step - loss: 0.4578 - accuracy: 0.8508 - val_loss: 0.2055 - val_accuracy: 0.9371 - lr: 0.0010

Epoch 9/40

138/138 [==============================] - ETA: 0s - loss: 0.4216 - accuracy: 0.8648

Epoch 9: val_accuracy improved from 0.93712 to 0.94726, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 103ms/step - loss: 0.4216 - accuracy: 0.8648 - val_loss: 0.1661 - val_accuracy: 0.9473 - lr: 0.0010

Epoch 10/40

138/138 [==============================] - ETA: 0s - loss: 0.3851 - accuracy: 0.8752

Epoch 10: val_accuracy did not improve from 0.94726

138/138 [==============================] - 14s 103ms/step - loss: 0.3851 - accuracy: 0.8752 - val_loss: 0.1831 - val_accuracy: 0.9419 - lr: 0.0010

Epoch 11/40

138/138 [==============================] - ETA: 0s - loss: 0.3818 - accuracy: 0.8755

Epoch 11: val_accuracy improved from 0.94726 to 0.95538, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 105ms/step - loss: 0.3818 - accuracy: 0.8755 - val_loss: 0.1542 - val_accuracy: 0.9554 - lr: 0.0010

Epoch 12/40

138/138 [==============================] - ETA: 0s - loss: 0.3463 - accuracy: 0.8889

Epoch 12: val_accuracy did not improve from 0.95538

138/138 [==============================] - 14s 104ms/step - loss: 0.3463 - accuracy: 0.8889 - val_loss: 0.1708 - val_accuracy: 0.9473 - lr: 0.0010

Epoch 13/40

138/138 [==============================] - ETA: 0s - loss: 0.3290 - accuracy: 0.8926

Epoch 13: val_accuracy did not improve from 0.95538

138/138 [==============================] - 14s 102ms/step - loss: 0.3290 - accuracy: 0.8926 - val_loss: 0.1694 - val_accuracy: 0.9473 - lr: 0.0010

Epoch 14/40

138/138 [==============================] - ETA: 0s - loss: 0.3118 - accuracy: 0.9009

Epoch 14: val_accuracy improved from 0.95538 to 0.95740, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 103ms/step - loss: 0.3118 - accuracy: 0.9009 - val_loss: 0.1415 - val_accuracy: 0.9574 - lr: 0.0010

Epoch 15/40

138/138 [==============================] - ETA: 0s - loss: 0.3159 - accuracy: 0.8995

Epoch 15: val_accuracy did not improve from 0.95740

138/138 [==============================] - 14s 103ms/step - loss: 0.3159 - accuracy: 0.8995 - val_loss: 0.1427 - val_accuracy: 0.9561 - lr: 0.0010

Epoch 16/40

138/138 [==============================] - ETA: 0s - loss: 0.3049 - accuracy: 0.9023

Epoch 16: val_accuracy improved from 0.95740 to 0.96214, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 102ms/step - loss: 0.3049 - accuracy: 0.9023 - val_loss: 0.1224 - val_accuracy: 0.9621 - lr: 0.0010

Epoch 17/40

138/138 [==============================] - ETA: 0s - loss: 0.2813 - accuracy: 0.9103

Epoch 17: val_accuracy did not improve from 0.96214

138/138 [==============================] - 14s 103ms/step - loss: 0.2813 - accuracy: 0.9103 - val_loss: 0.1293 - val_accuracy: 0.9574 - lr: 0.0010

Epoch 18/40

138/138 [==============================] - ETA: 0s - loss: 0.2806 - accuracy: 0.9079

Epoch 18: val_accuracy did not improve from 0.96214

138/138 [==============================] - 14s 104ms/step - loss: 0.2806 - accuracy: 0.9079 - val_loss: 0.1350 - val_accuracy: 0.9533 - lr: 0.0010

Epoch 19/40

138/138 [==============================] - ETA: 0s - loss: 0.2673 - accuracy: 0.9095

Epoch 19: val_accuracy did not improve from 0.96214

Epoch 19: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

138/138 [==============================] - 14s 104ms/step - loss: 0.2673 - accuracy: 0.9095 - val_loss: 0.1276 - val_accuracy: 0.9540 - lr: 0.0010

Epoch 20/40

138/138 [==============================] - ETA: 0s - loss: 0.2370 - accuracy: 0.9229

Epoch 20: val_accuracy improved from 0.96214 to 0.96890, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 104ms/step - loss: 0.2370 - accuracy: 0.9229 - val_loss: 0.1123 - val_accuracy: 0.9689 - lr: 5.0000e-04

Epoch 21/40

138/138 [==============================] - ETA: 0s - loss: 0.2222 - accuracy: 0.9291

Epoch 21: val_accuracy did not improve from 0.96890

138/138 [==============================] - 14s 103ms/step - loss: 0.2222 - accuracy: 0.9291 - val_loss: 0.1147 - val_accuracy: 0.9669 - lr: 5.0000e-04

Epoch 22/40

138/138 [==============================] - ETA: 0s - loss: 0.2128 - accuracy: 0.9328

Epoch 22: val_accuracy did not improve from 0.96890

138/138 [==============================] - 14s 103ms/step - loss: 0.2128 - accuracy: 0.9328 - val_loss: 0.1118 - val_accuracy: 0.9655 - lr: 5.0000e-04

Epoch 23/40

138/138 [==============================] - ETA: 0s - loss: 0.2049 - accuracy: 0.9334

Epoch 23: val_accuracy did not improve from 0.96890

Epoch 23: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

138/138 [==============================] - 14s 102ms/step - loss: 0.2049 - accuracy: 0.9334 - val_loss: 0.1040 - val_accuracy: 0.9675 - lr: 5.0000e-04

Epoch 24/40

138/138 [==============================] - ETA: 0s - loss: 0.1965 - accuracy: 0.9359

Epoch 24: val_accuracy improved from 0.96890 to 0.96957, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 104ms/step - loss: 0.1965 - accuracy: 0.9359 - val_loss: 0.0996 - val_accuracy: 0.9696 - lr: 2.5000e-04

Epoch 25/40

138/138 [==============================] - ETA: 0s - loss: 0.1897 - accuracy: 0.9364

Epoch 25: val_accuracy did not improve from 0.96957

138/138 [==============================] - 14s 104ms/step - loss: 0.1897 - accuracy: 0.9364 - val_loss: 0.0979 - val_accuracy: 0.9696 - lr: 2.5000e-04

Epoch 26/40

138/138 [==============================] - ETA: 0s - loss: 0.1880 - accuracy: 0.9403

Epoch 26: val_accuracy improved from 0.96957 to 0.97093, saving model to mnist_cnn.hdf5

138/138 [==============================] - 15s 106ms/step - loss: 0.1880 - accuracy: 0.9403 - val_loss: 0.0965 - val_accuracy: 0.9709 - lr: 2.5000e-04

Epoch 27/40

138/138 [==============================] - ETA: 0s - loss: 0.1822 - accuracy: 0.9409

Epoch 27: val_accuracy improved from 0.97093 to 0.97431, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 103ms/step - loss: 0.1822 - accuracy: 0.9409 - val_loss: 0.0881 - val_accuracy: 0.9743 - lr: 2.5000e-04

Epoch 28/40

138/138 [==============================] - ETA: 0s - loss: 0.1780 - accuracy: 0.9417

Epoch 28: val_accuracy did not improve from 0.97431

138/138 [==============================] - 14s 105ms/step - loss: 0.1780 - accuracy: 0.9417 - val_loss: 0.0903 - val_accuracy: 0.9716 - lr: 2.5000e-04

Epoch 29/40

138/138 [==============================] - ETA: 0s - loss: 0.1908 - accuracy: 0.9391

Epoch 29: val_accuracy improved from 0.97431 to 0.97566, saving model to mnist_cnn.hdf5

138/138 [==============================] - 14s 105ms/step - loss: 0.1908 - accuracy: 0.9391 - val_loss: 0.0873 - val_accuracy: 0.9757 - lr: 2.5000e-04

Epoch 30/40

138/138 [==============================] - ETA: 0s - loss: 0.1780 - accuracy: 0.9423

Epoch 30: val_accuracy did not improve from 0.97566

138/138 [==============================] - 14s 103ms/step - loss: 0.1780 - accuracy: 0.9423 - val_loss: 0.0843 - val_accuracy: 0.9736 - lr: 2.5000e-04

Epoch 31/40

138/138 [==============================] - ETA: 0s - loss: 0.1759 - accuracy: 0.9429

Epoch 31: val_accuracy did not improve from 0.97566

138/138 [==============================] - 14s 103ms/step - loss: 0.1759 - accuracy: 0.9429 - val_loss: 0.0916 - val_accuracy: 0.9723 - lr: 2.5000e-04

Epoch 32/40

138/138 [==============================] - ETA: 0s - loss: 0.1762 - accuracy: 0.9412

Epoch 32: val_accuracy did not improve from 0.97566

Epoch 32: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.

138/138 [==============================] - 14s 104ms/step - loss: 0.1762 - accuracy: 0.9412 - val_loss: 0.0886 - val_accuracy: 0.9716 - lr: 2.5000e-04

Epoch 33/40

138/138 [==============================] - ETA: 0s - loss: 0.1756 - accuracy: 0.9412

Epoch 33: val_accuracy did not improve from 0.97566

138/138 [==============================] - 14s 103ms/step - loss: 0.1756 - accuracy: 0.9412 - val_loss: 0.0789 - val_accuracy: 0.9723 - lr: 1.2500e-04

Epoch 34/40

138/138 [==============================] - ETA: 0s - loss: 0.1664 - accuracy: 0.9438

Epoch 34: val_accuracy did not improve from 0.97566

138/138 [==============================] - 14s 104ms/step - loss: 0.1664 - accuracy: 0.9438 - val_loss: 0.0821 - val_accuracy: 0.9757 - lr: 1.2500e-04

Epoch 35/40

138/138 [==============================] - ETA: 0s - loss: 0.1636 - accuracy: 0.9475

Epoch 35: val_accuracy did not improve from 0.97566

Epoch 35: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.

138/138 [==============================] - 14s 103ms/step - loss: 0.1636 - accuracy: 0.9475 - val_loss: 0.0821 - val_accuracy: 0.9750 - lr: 1.2500e-04

Epoch 36/40

138/138 [==============================] - ETA: 0s - loss: 0.1585 - accuracy: 0.9474

Epoch 36: val_accuracy did not improve from 0.97566

138/138 [==============================] - 14s 103ms/step - loss: 0.1585 - accuracy: 0.9474 - val_loss: 0.0824 - val_accuracy: 0.9757 - lr: 6.2500e-05

Epoch 37/40

138/138 [==============================] - ETA: 0s - loss: 0.1580 - accuracy: 0.9494

Epoch 37: val_accuracy improved from 0.97566 to 0.97634, saving model to mnist_cnn.hdf5

```
138/138 [==============================] - 15s 106ms/step - loss: 0.1580 -
accuracy: 0.9494 - val_loss: 0.0796 - val_accuracy: 0.9763 - lr: 6.2500e-05
Epoch 38/40
138/138 [==============================] - ETA: 0s - loss: 0.1582 - accuracy:
0.9472
Epoch 38: val_accuracy did not improve from 0.97634
138/138 [==============================] - 14s 102ms/step - loss: 0.1582 -
accuracy: 0.9472 - val_loss: 0.0807 - val_accuracy: 0.9757 - lr: 6.2500e-05
Epoch 39/40
138/138 [==============================] - ETA: 0s - loss: 0.1680 - accuracy:
0.9461
Epoch 39: val_accuracy did not improve from 0.97634
138/138 [==============================] - 14s 105ms/step - loss: 0.1680 -
accuracy: 0.9461 - val_loss: 0.0820 - val_accuracy: 0.9757 - lr: 6.2500e-05
Epoch 40/40
138/138 [==============================] - ETA: 0s - loss: 0.1551 - accuracy:
0.9478
Epoch 40: val_accuracy did not improve from 0.97634

Epoch 40: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
138/138 [==============================] - 14s 104ms/step - loss: 0.1551 -
accuracy: 0.9478 - val_loss: 0.0801 - val_accuracy: 0.9763 - lr: 6.2500e-05
Saved model to disk
```

4.2 Висновки з розділу 4

1. Проаналізовано результат навчання моделі ЗНМ.

2. Експериментальним методом було визначено, що точність навчання зростає з 97.22% до 97.63% при збільшенні кількості епох з 30 до 40, а зменшення чи збільшення об’єму валідаційної вибірки не покращує результат навчання.

ВИСНОВКИ

Відповідно до мети і завдань дослідження було виконано наступне:

1. Проаналізовані наукові дослідження з проблеми розпізнавання тексту на зображенні та розглянуто використання описових функцій (особливостей), які стосуються об’єкта дослідження.
2. Розглянутий алгоритм розпізнавання тексту на зображенні, визначено етапи алгоритму та описано види алгоритмів розпізнавання.
3. Проаналізоване наявне програмне забезпечення для розпізнавання тексту на зображенні та визначено, що існує необхідність у створенні більш ефективної системи розпізнавання українського рукописного тексту на зображенні.
4. Розглянутий фреймворки та бібліотеки глибокого навчання, у результаті їх порівняння було обрано бібліотеку Keras.
5. Обрано датасет Cyrillic_ukr, який використано для навчання моделі нейронної мережі.
6. Виконане навчання моделі згорткової нейронної мережі розпізнавати українські рукописні літери. Встановлено, що найвища точність розпізнавання на валідаційній вибірці досягає 97.22%.
7. Експериментальним методом було визначено, що точність навчання зростає з 97.22% до 97.63% при збільшенні кількості епох з 30 до 40, а зменшення чи збільшення об’єму валідаційної вибірки не покращує результат навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yousef, Mohamed, Khaled F. Hussain, and Usama S. Mohammed. "Accurate, data-efficient, unconstrained text recognition with convolutional neural networks." Pattern Recognition, 2020, 108: 107482.

2. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «Молода наука-2023»: у 5 т. / Запорізький національний університет. – Запоріжжя : ЗНУ, 2023. Т.5. с. 94-96.

3. Збірник наукових праць III Всеукраїнської науково-практичної конференції за участю молодих науковців «АКТУАЛЬНІ ПИТАННЯ СТАЛОГО НАУКОВО-ТЕХНІЧНОГО ТА СОЦІАЛЬНО-ЕКОНОМІЧНОГО РОЗВИТКУ РЕГІОНІВ УКРАЇНИ». Запоріжжя : Запорізький національний університет, 2023. С. 110–111.

4. Datong Chen, Jean-Marc Odobez, Herve Bourlard. Text detection and recognition in images and video frames // Pattern Recognition journal, 2003. – P. 595-608.

5. Convolutional Neural Network [Electronic resource]/ Stanford. – Access mode: http://ufldl.stanford.edu/tutorial /supervised/ConvolutionalNeuralNetwork/ (date of application: 28.11.2023).

6. Парамуд, Я.С.; Яркун, В.І. Метод розпізнавання символів на зображенні на основі згорткової нейронної мережі. Вісник Національного університету Львівська політехніка. Комп’ютерні системи та мережі, 2018, 905: с. 96-105.

7. Convolutional networks [Electronic resource]/ Stanford. – Access mode: http://cs231n.github.io/convolutionalnetworks/#overview.

8. Coreml [Electronic resource]/ Apple. – Access mode: https://developer.apple.com/documentation/coreml (date of application: 28.11.2023).

9. Bobrov, K.A.; Shulman, V.D.; Vlasov, K.P. Analysis of text recognition technologies from image. International Journal of Humanities and Natural Sciences, 2022, vol. 3-2: c. 124-128.

10. Khandokar, I., et al. "Handwritten character recognition using convolutional neural network." Journal of Physics: Conference Series. Vol. 1918. No. 4. IOP Publishing, 2021.

11. Deore S. P., Pravin A. Devanagari Handwritten Character Recognition using fine-tuned Deep Convolutional Neural Network on trivial dataset. Sādhanā. 2020. Vol. 45, no. 1.

12. Shams, Mahmoud, Amira Elsonbaty, and Wael ElSawy. "Arabic handwritten character recognition based on convolution neural networks and support vector machine." arXiv preprint arXiv: 2009.13450, 2020.

13. Handwritten Character Recognition from Images using CNN-ECOC / M. B. Bora et al. Procedia Computer Science. 2020. Vol. 167. P. 2403–2409.

14. Liu H. Handwritten English character recognition using convolutional neural network. Applied and Computational Engineering. 2023. Vol. 4, no. 1. P. 199–204.

15. Convolutional-Neural-Network-Based handwritten character recognition: an approach with massive multisource data / N. Saqib et al. Algorithms. 2022. Vol. 15, no. 4. P. 129.

16. Tesseract OCR. – [Electronic resource]. – Access mode: https://github.com/tesseract-ocr/tesseract (date of application: 28.11.2023).

17. Tesseract documentation. Tesseract OCR. – [Electronic resource]. – Access mode: https://tesseract-ocr.github.io/ (date of access: 06.12.2023).

18. ABBYY FineReader. – [Electronic resource]. – Access mode: https://pdf.abbyy.com (date of application: 28.11.2023).

19. Cloud Vision API. – [Electronic resource]. – Access mode: https://cloud.google.com/vision/ (date of application: 28.11.2023).

20. FreeMoreSoft – Freemore OCR – Extract Text from Image for Free!. FreeMoreSoft - Best Free Audio, Video, Disc, Image and Document Software. –

[Electronic resource]. – Access mode: https://www.freemoresoft.com/freeocr/index.php (date of access: 06.12.2023).

21. Bengio Y., Courville A., Goodfellow I. Deep Learning. MIT Press, 2016. 800 p.

22. Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) [Electronic resource]. – Access mode: https://image-net.org/challenges/LSVRC/2012/results.html#t1 (date of application: 28.11.2023).

23. Cyrillic_ukr. Kaggle: Your Machine Learning and Data Science Community. – [Electronic resource]. – Access mode: https://www.kaggle.com/datasets/oleksiichornyi/cyrillic-ukr (date of application: 28.11.2023).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Крат Олексій Федорович, студент _2_ курсу форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти se22m-09@stu.zsea.edu.ua

- підтверджую, що написана мною кваліфікаційна робота на тему **«Особливості побудови комп'ютерної системи розпізнавання рукописних текстів на зображеннях»** відповідає нормам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2023 Підпис _____ Крат Олексій Федорович
(студент)

Дата 30.11.2023 Підпис _____ Безверхий Анатолій Ігорович
(науковий керівник)