МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА
Факультет економіки і підприємництва
Кафедра інформаційних технологій

Допущено до захисту
"_____" _____ 2023 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ОПТИЧНОГО РОЗПІЗНАВАННЯ СИМВОЛІВ З ВИКОРИСТАННЯМ НЕЙРОННОЇ МЕРЕЖІ

Виконав – студент 2 курсу, групи 21м-кн
освітньої програми "Комп'ютерні науки"
спеціальності 122 Комп'ютерні науки
Безусий Владислав Ігорович

Науковий керівник – к.т.н., доцент
Ліщук Роман Ігорович

Умань – 2023

Зміст

ВСТУП 3

РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ 5

1.1. Аналіз проблеми..... **Помилка! Закладку не визначено.**

1.2. Підходи до локалізації об’єктів інтересу на зображенні ... **Помилка! Закладку не визначено.**

1.3. Постановка задачі 7

РОЗДІЛ 2 ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ КЛАСИФІКАЦІЙНОГО АНАЛІЗУ ДЕРМОСКОПІЧНИХ ЗОБРАЖЕНЬ 15

2.1. Архітектура екстрактора ознак..... 15

2.2. Архітектура детектора..... 17

2.3. Метрики оцінки ефективності класифікатора ..... **Помилка! Закладку не визначено.**

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ..... 30

3.1. Формування вхідного математичного опису**Помилка! Закладку не визначено.**

3.2. Короткий опис програмної реалізації системи**Помилка! Закладку не визначено.**

3.3. Аналіз результатів моделювання**Помилка! Закладку не визначено.**

ВИСНОВКИ..... 42

СПИСОК ЛІТЕРАТУРИ..... 43

ДОДАТКИ..... 45

ВСТУП

У наш час компанії, які використовують системи на паперових носіях, прагнуть покращити продуктивність своїх систем шляхом оцифровування інформації. Цифрова система забезпечує безпечний і простий спосіб обміну та отримання інформації, що є однією з головних проблем у паперовій системі. Ця проблема стає тягарем, оскільки накопичується папір та зменшується продуктивність пов'язана з кількістю часу, витраченого на пошук відповідної інформації. Це означає, що коли папір накопичується, продуктивність падає. На щастя, оптичне розпізнавання символів, скорочено OCR (Optical Character Recognition), є ключем до вирішення цієї проблеми. OCR – це спосіб перетворення тексту, написаного на папері або у форматі PDF, у якому немає можливості пошуку, у формат, доступний для пошуку й редагування. Таким чином, компанії, які використовують цю технологію, можуть оцифрувати свої системи за неймовірно короткий час, умовно кажучи, порівняно з тим, щоб передруковувати все з нуля.

Розпізнавання символів привернуло багато уваги протягом останніх десятиліть через його важливість в обробці зображень, що дозволяє машині читати текст. Розпочато в 1957 році Френком Розенблатом, Чарльзом Вайтманом та іншими, де вони успішно розробили перший нейрокомп'ютер, який міг виявляти персонажів за допомогою нейронної мережі та те, що вони так називали «камерою високої роздільної здатності». Їхню нейронну мережу було навчено за допомогою техніки під назвою «Правило навчання перцептрона», яка здатна навчити лише одношарову мережу. У зв'язку з цим обмеженням тут використовується метод навчання – це алгоритм «Зворотного поширення помилки», здатний навчати багаторівневі мережі. Алгоритм зворотного поширення помилки є, по суті, найважливішою частиною штучних нейронних мереж. Їх основною метою є розробка алгоритму навчання для багатошарових нейронних мереж прямого зв'язку, що дозволяє мережам навчитися неявно отримувати відображення [1].

Його мета – оптимізувати вагові коефіцієнти, дозволяючи таким чином нейронній мережі навчитися правильно відображати довільні входи та виходи. Зворотне поширення помилки значно швидше, ніж інші алгоритми нейронної мережі. Можна розглядати зворотне поширення як передовий жадібний підхід. Це допомагає швидше отримати бажаний результат і навіть скорочує час навчання з місяців до годин. Його можна вважати основою нейронної мережі. Зворотне поширення має численні переваги, ось деякі з найважливіших [1]:

• Жодних параметрів налаштовувати не потрібно.

• Моделі не потрібно вивчати особливості функції.

• Зворотне поширення є гнучким методом, оскільки попередні знання мережі не потрібні.

• Це швидкий спосіб і його досить легко реалізувати.

• Цей підхід, як правило, працює досить добре в більшості ситуацій.

• Користувачеві не потрібно вивчати спеціальні функції.

РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Основні принципи організації технології оптичного розпізнавання символів

Оптичне розпізнавання символів (OCR) – це процес, який перетворює зображення тексту в текстовий формат, який можливо редагувати на комп'ютері. Наприклад, якщо відбувається сканування форми або квитанції, комп'ютер зберігає відсканований документ як файл зображення. Використання текстового редактору для редагування, пошуку або підрахунку слів у файлі зображення неможливо. Однак існує можливість використовувати оптичне розпізнавання символів, щоб перетворити зображення на текстовий документ із збереженням його вмісту як текстових даних.

Більшість бізнес-процесів передбачає отримання інформації з друкованих ЗМІ. Паперові форми, рахунки-фактури, відскановані юридичні документи та роздруковані контракти є частиною бізнес-процесів. Ці великі обсяги документів займають багато часу та місця для зберігання та керування ними. Безпаперовий документообіг – є одним із варіантів до вирішення проблеми, але сканування документа в зображення має певні труднощі. Процес вимагає ручного втручання і може бути виснажливим і повільним. Крім того, оцифрування вмісту цього документа створює файли зображень із прихованим текстом. Текст на зображеннях не може бути оброблений програмою обробки текстів так само, як текстові документи. Технологія OCR вирішує проблему, перетворюючи текстові зображення на текстові дані, які можна аналізувати за допомогою іншого бізнес-програмного забезпечення. Потім ви можете використовувати дані для проведення аналітики, оптимізації операцій, автоматизації процесів і підвищення продуктивності.

Механізм OCR або програмне забезпечення OCR працює за допомогою таких кроків:

• Отримання зображення - сканер зчитує документи та перетворює їх у двійкові дані. Програмне забезпечення OCR аналізує скановане зображення та класифікує світлі ділянки як фон, а темні – як текст.

• Попередня обробка - програмне забезпечення OCR спочатку очищає зображення та видаляє помилки, щоб підготувати його до читання. Це деякі з його методів очищення.

• Розпізнавання тексту - два основних типи алгоритмів OCR або програмних процесів, які програмне забезпечення OCR використовує для розпізнавання тексту, називаються зіставленням шаблону та вилученням ознак.

• Зіставлення шаблону - зіставлення за зразком працює шляхом виділення символьного зображення, яке називається гліфом, і порівняння його з аналогічним збереженим гліфом. Розпізнавання шаблонів працює, лише якщо збережений гліф має подібний шрифт і масштаб до вхідного гліфа. Цей метод добре працює зі сканованими зображеннями документів, набраних відомим шрифтом.

• Вилучення ознак - вилучення функцій розбиває або розкладає гліфи на такі ознаки, як лінії, замкнуті цикли, напрямок ліній і перетини ліній. Потім він використовує ці функції, щоб знайти найкращий відповідник або найближчого сусіда серед різноманітних збережених гліфів.

• Подальша обробка - після аналізу система перетворює витягнуті текстові дані в комп’ютеризований файл. Деякі системи оптичного розпізнавання символів можуть створювати анотовані PDF-файли, які містять як попередні, так і наступні версії відсканованого документа.

Науковці класифікують різні типи технологій OCR на основі їх використання та застосування. Простий механізм OCR працює, зберігаючи багато різних шаблонів шрифтів і текстових зображень як шаблони. Програмне забезпечення OCR використовує алгоритми зіставлення шаблонів для порівняння текстових зображень, символ за символом, із внутрішньою базою даних. Якщо система зіставляє текст слово за словом, це називається оптичним розпізнаванням слів. Це рішення має обмеження, оскільки існує практично необмежена кількість стилів шрифтів і рукописного тексту, і кожен окремий тип не може бути захоплений і збережений у базі даних.

Сучасні системи OCR використовують технологію інтелектуального розпізнавання символів (ICR), щоб читати текст так само, як це роблять люди. Вони використовують передові методи, які навчають машини поводитися як люди за допомогою програмного забезпечення для машинного навчання. Система машинного навчання під назвою нейронна мережа аналізує текст на багатьох рівнях, багаторазово обробляючи зображення. Він шукає різні атрибути зображення, такі як криві, лінії, перетини та петлі, і поєднує результати всіх цих різних рівнів аналізу, щоб отримати остаточний результат. Незважаючи на те, що ICR зазвичай обробляє зображення по одному символу за раз, процес є швидким, а результати отримуються за секунди.

Інтелектуальні системи розпізнавання слів працюють за тими ж принципами, що й ICR, але обробляють цілі зображення слів замість попередньої обробки зображень у символи. Оптичне розпізнавання позначок визначає логотипи, водяні знаки та інші текстові символи в документі.

Оптичне розпізнавання позначок визначає логотипи, водяні знаки та інші текстові символи в документі.

1.2. Порівняльна характеристика існуючих OCR-додатків

ABBYY FineReader

Точні механізми, які дозволяють людям розпізнавати об’єкти, ще не з’ясовані, але три основні принципи вже добре відомі вченим – цілісність, цілеспрямованість і адаптивність (IPA). Ці принципи є основою ABBYY FineReader OCR, що дозволяє відтворювати природне або людське розпізнавання.

Давайте подивимося, як FineReader OCR розпізнає текст. Спочатку програма аналізує структуру зображення документа. Він ділить сторінку на елементи, такі як блоки текстів, таблиці, зображення тощо. Рядки діляться на слова, а потім - на символи. Після виділення символів програма порівнює їх із набором шаблонних зображень. Він висуває численні гіпотези про те, що це за персонаж. На основі цих гіпотез програма аналізує різні варіанти розбиття

рядків на слова і слів на символи. Після обробки величезної кількості таких імовірнісних гіпотез програма врешті приймає рішення, представляючи вам розпізнаний текст (рис. 1.1) [3].

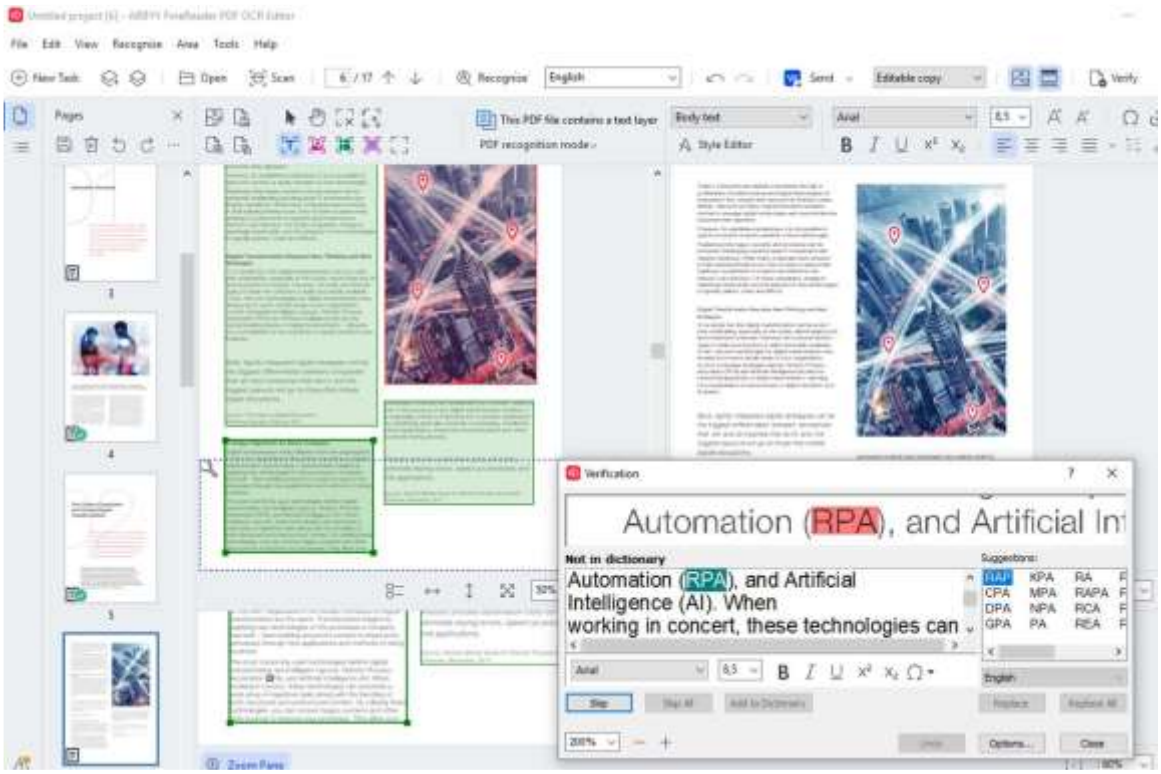


Рисунок 1.1 – Процес розпізнавання тексту в ABBYY FineReader

Крім того, ABBYY FineReader забезпечує підтримку словників для 48 мов. Це забезпечує вторинний аналіз елементів тексту на рівні слів. Завдяки підтримці словника програма забезпечує ще більш точний аналіз і розпізнавання документів і спрощує подальшу перевірку результатів розпізнавання.

Найдосконаліші системи оптичного розпізнавання символів, такі як технологія оптичного розпізнавання символів на основі штучного інтелекту ABBYY, орієнтовані на відтворення природного розпізнавання або розпізнавання «звірів». В основі цих систем лежать три фундаментальні принципи: цілісність, цілеспрямованість і адаптивність. Принцип цілісності говорить, що спостережуваний об'єкт завжди повинен розглядатися як «ціле», що складається з багатьох взаємопов'язаних частин. Принцип

цілеспрямованості передбачає, що будь-яка інтерпретація даних завжди повинна служити певній меті. А принцип адаптивності означає, що програма повинна бути здатною до самонавчання.

Не обов’язково бути фахівцем з OCR, щоб побачити переваги програми OCR, побудованої на принципах IPA. Ці принципи наділяють програму максимальною гнучкістю та інтелектом, максимально наближаючи її до розпізнавання людиною.

Після років досліджень компанія ABBYY змогла реалізувати описані вище принципи IPA у своїх технологіях OCR [3].

Kofax OmniPage

Програмне забезпечення OCR, дозволяє комп’ютерам читати документи так само, як і людина: розпізнаючи візерунки букв і виділяючи текст із зображення. У результаті, коли сканер зчитує документ, OmniPage використовує технологію оптичного розпізнавання тексту, щоб перетворити його безпосередньо в текстовий файл. OmniPage технологія оптичного розпізнавання символів є результатом десятиліть, витрачених на дослідження того, як покращити та застосувати цю функцію. Тепер, маючи у розпорядженні можливості, які були розблокувані, бізнес може цифрово архівувати, редагувати та шукати всі фізичні документи, створюючи першокласну організацію [7].

Хоча точність є найважливішим критерієм під час перетворення сканованих зображень у текстові документи, своєчасність також є важливою. Коли ви конвертуєте документ за допомогою OmniPage, команді не доведеться виконувати складний процес. Кількома клацаннями миші та за кілька секунд OmniPage створює текстовий документ, який можна редагувати, щоб можна приступити до роботи, що призвело до підвищення продуктивності та зменшення розчарування співробітників.

Однією з найбільш вражаючих особливостей даної технології OCR є те, що вона може навіть розпізнавати таблиці та інші діаграми серед тексту. Отриманий документ зберігає ці функції, заощаджуючи дорогоцінний час.

OmniPage точно транскрибує ваш документ, тож минули ті часи, коли фізичний документ потрібно було повторно вводити в текстовий процесор, щоб створити редаговану копію.

Особливості технології [7]:

• Редактор шаблонів форм OmniPage Forms (FTE) містить інтерфейс користувача для визначення шаблонів, а також виклики API, які легко витягують дані з таких документів, як рахунки-фактури, документи про іпотечну позику та заявки.

• OmniPage Document Classification (DC) використовує штучний інтелект, комп'ютерне зір і машинне навчання для автоматичного сортування документів. Включає інтерфейс користувача для розробки моделей, а також виклики API для програмної класифікації сторінок.

• Автоматично визначає знімки екрана та покращує роздільну здатність для кращої точності. Розпізнавання тексту для знімків екрану має вирішальне значення для запобігання втраті даних (DLP) або автоматизації робочих процесів робототехніки.

• Провідна на ринку попередня обробка зображень надає інструменти для досягнення найкращих результатів OCR для різноманітних сканерів, багатофункціональних пристроїв, телефонів і планшетів.

• Розпізнає документи понад 120 різними мовами. Включає автоматичне визначення мови, яке обробляє документи, що містять змішані мови.

Tesseract

Tesseract – це проєкт оптичного розпізнавання символів із відкритим вихідним кодом, який розроблявся у HP з 1984 і 1994 роками. Як надбудова, він з'явився нізвідки під час щорічного тестування точності OCR UNLV у 1995 році, яскраво засяяв своїми результатами, а потім знову зник той самий плащ секретності, під яким він був розроблений. Тепер вперше можна розкрити деталі архітектури та алгоритмів. Tesseract почав старт як докторський дослідницький проект у HP Labs, Брістоль, і зарекомендував себе як можливе програмне або апаратне доповнення для сканерів HP. Мотивація була надана

тими аргументами, що комерційні пропозиції оптичного розпізнавання символів (OCR) в той час були на етапі розвитку та з тріском зазнали поразки, оскільки якість друку була низькою. Після роботи над проєктом у сукупності з HP Labs Bristol, Tesseract мав значну перевагу в точності над комерційними механізмами, але не зайняв позицію лідера на ринку продаж. Потім розвиток Tesseract отримав в HP Labs Bristol у сфері дослідження оптичного розпізнавання символів для стиснення даних, отриманих в результаті сканування. Основною ціллю було зменшення відхилення, ніж на початковому рівні точності. Згодом робота на проєктом була повністю заморожена, а саме в кінці 1994 року. Проєкт був переданий до UNLV і взяв участь у щорічному конкурсі точності оптичного розпізнавання символів, де був одним з найкращих, навіть, серед новітніх комерційних продуктів [9].

Оскільки HP самостійно розробила технологію аналізу макета сторінки, яка використовувалася в її продуктах (і тому не була випущена для відкритого коду), Tesseract ніколи не потребував власного аналізу макета сторінки. Таким чином, Tesseract припускає, що його входом є бінарне зображення з необов’язковими визначеними полігональними областями.

Обробка відбувається за звичайним поетапним переміщенням, але деякі з етапів були незвичайними для того періоду. Першим кроком є аналіз підключених складових частин, у яких зберігаються контури компонентів. Це було обчислювально дороге дизайнерське рішення для тих років, але воно мало багато позитивних сторін: шляхом перевірки вкладеності контурів, а також кількості дочірніх контурів легко виявити інверсний текст і розпізнати його достатньо легко, як чорний - білий текст. Tesseract є одним з перших проєктів оптичного розпізнавання символів, здатний таким легким способом обробляти текст. На даному етапі контури збираються до купи, чисто вкладеним, у Blobs [9].

Blobs формулюються текстові рядки, а рядки та області піддаються перевірці на статичний крок. Рядки тексту діляться на слова не однаково, оскільки все залежить від міжсимвольного інтервалу. Текст із фіксованим

кроком одразу розбивається на клітинки символів. Пропорційний текст розбивається на слова за допомогою певних пробілів і нечітких пробілів.

Тоді розпізнавання відбувається як двоетапний процес. Під час першої ітерації відбувається спроба розпізнати кожне слово одне за одним. Відібрані слова надходять в адаптивний класифікатор як тестові дані. Адаптивний класифікатор має змогу збільшити точність розпізнавання символів в кінці сторінки.

Як результат адаптивний класифікатор отримав важливу інформацію невчасно, і не в змозі виконати зміни на початку сторінки, тоді відбувається друга ітерація, під час якої нерозпізнані слова підлягають повторному процесу ідентифікації. Остання ітерація вирішує питання неявних пропусків та перевіряє існуючі підходи для знаходження висоти, для пошуку малих символів (рис. 1.2) [8].



Рисунок 1.2 - Загальна робота Tesseract OCR

Розглянемо основні відмінності OCR додатків, їх оновлення, можливість взаємодії із проєктами, кількість доступних мов тощо (табл.1) [2, 4].

Таблиця 1 – Порівняльна характеристика OCR додатків

	ABBYY FineReader	**Kofax OmniPage**	**Tesseract**
Дата останньої стабільної версії	08/09/2020	18/03/2020	26/12/2019
Цінова політика	Платно, вартість індивідуальна	Платно, в залежності від збірки	Безкоштовно
Робота офлайн	Так	Так	Так
Кількість мов	210	125	100+
Підтримка української мови	Так	Так	Так
Мови програмування, фреймворки	C, C++, .NET, Delphi, Java, JS	C/C++, .NET Framework, .NET Core, Java	Python, JS, TypeScript, C, C++, C#, PHP, Java, Ruby, Go, Swif
Формат результату розпізнавання	TXT, PDF, CSV та 13 інших	DOC/DOCX, XLS/XLSX, PPTX, RTF, PDF, PDF з можливістю пошуку, HTML, TXT, XML, ePUB	Текст, PDF, TXT, HOCR, TSV
Інструменти передобробки	Так	Так	Ні
Можливість адаптації до специфічних сценаріїв	Так	Так	Так

1.3. Постановка задачі

Аналіз існуючих програмних додатків, які використовуються для оптичного розпізнавання символів показав, що ці сервіси в значній мірі відрізняються між собою швидкістю роботи, алгоритмами, які задіяні в їх реалізації, високою ціновою політикою, способом застосування, тобто мають свою специфіку та застосовуються для вирішення відповідного кола завдань.

На основі цього було вирішено розробити інформаційну технологію оптичного розпізнавання символів на основі багатошарової нейронної мережі, за допомогою якої буде розглянуто алгоритм зворотного поширення помилки як основний метод навчання нейронної мережі. Це дозволить досягнути вирішення задач оцифрування інформації, скорочення часу навчання мережі та отримання бажаного результату.

Метою даної роботи є розробка інформаційної технології оптичного розпізнавання символів, що мають фіксовану висоту та ширину, на основі проаналізованих уже на сьогодні методів рішення даної задачі та їх порівняльної характеристики. Цей інформаційний додаток може бути використаний для таких задач:

- оцінка основних додатків для реалізації оптичного розпізнавання символів;
- оптичне розпізнавання символів;
- вивчення основних принципів організації технології розпізнавання символів;
- оцінка алгоритму зворотного поширення помилки для навчання багатошарової нейронної мережі.
- використання об’єктно-орієнтованого програмування для розробки системи оптичного розпізнавання символів.

РОЗДІЛ 2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ ТА МОДЕЛЮВАННЯ СИСТЕМИ

2.1. Принципи навчання багатошарової нейронної мережі з використанням алгоритму зворотного поширення помилки

У даній роботі використовується алгоритм навчання багатошарової нейронної мережі з використанням алгоритму зворотного поширення. Щоб проілюструвати цей процес, використовується тришарова нейронна мережа з двома входами та одним виходом (рис. 2.1).

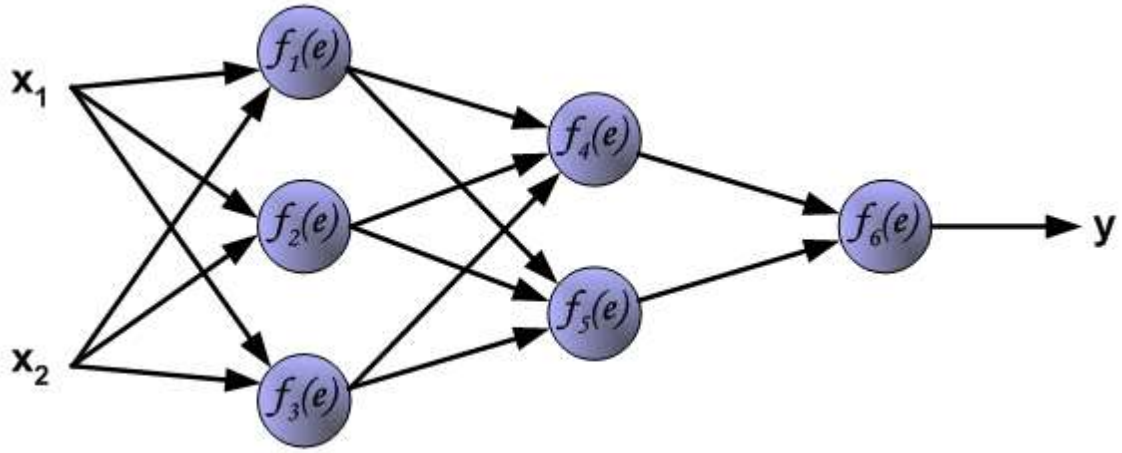


Рисунок 2.1 – Тришарова нейронна мережа

За будовою кожен нейрон складається з двох різних елементів, а саме перший з них – дендрити – відповідає за додавання добутків вагових коефіцієнтів до вхідних сигналів, відповідно другий з них реалізує нелінійну функцію, яка має назву функція активації нейронів. Сигнал e – загальна сума всіх початкових сигналів, а $y = f(e)$ – вихідний сигнал нелінійного елемента або нейрона (рис. 2.2) [6].

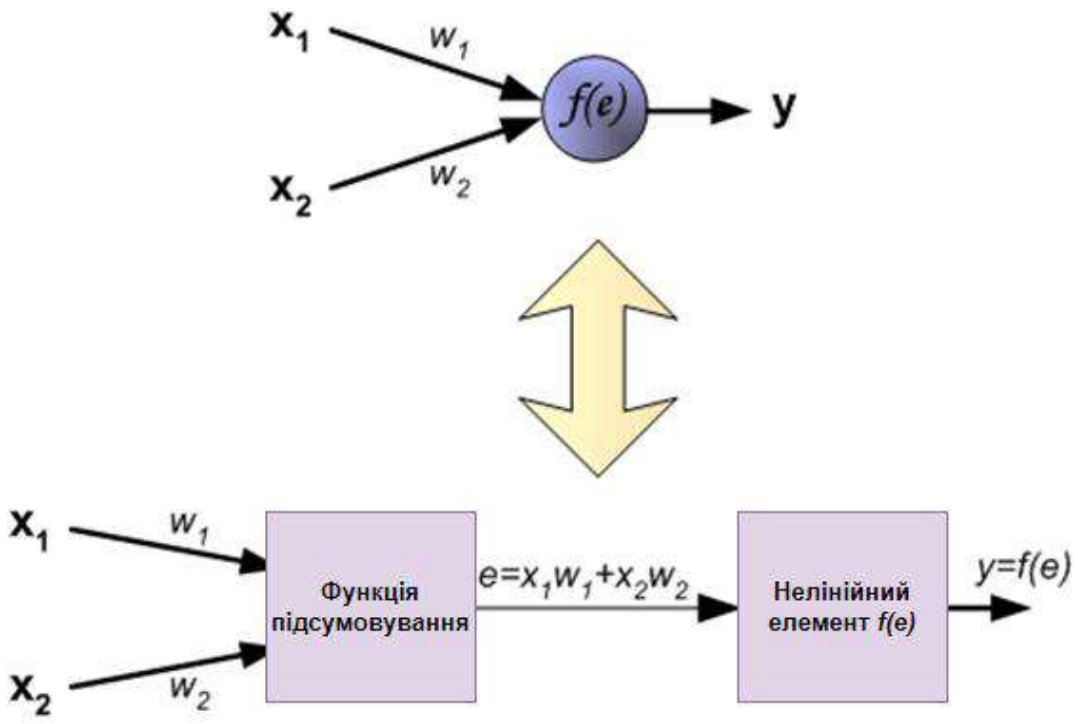


Рисунок 2.2 – Математичне відображення усіх складових нейрона

Для навчання нейронної мережі, потрібно сформувати навчальний набір даних. Навчальний набір даних складається з вхідних сигналів (x1 і x2), яким призначається відповідна ціль (бажаний вихід) k. Навчання мережі є ітеративним процесом. У кожній ітерації вагові коефіцієнти вузлів(нейронів) змінюються з використанням нових даних із навчального набору даних. Модифікація вагових коефіцієнтів розраховується за алгоритмом, описаним нижче (рис 2.3) [6].

Кожен етап навчання починається з примусової обробки обох вхідних сигналів із навчального набору. Після цього етапу можна визначити значення вихідних сигналів для кожного нейрона у кожному шарі мережі. На рис. 2.3 нижче продемонстровано, як сигнал розповсюджується мережею. У вхідному шарі символи $W(Xm)n$ відображають вагові коефіцієнти з’єднань між входом Xm мережі й нейроном n. Символи yn відображають вихідний сигнал нейрона n (рис. 2.3) [6].

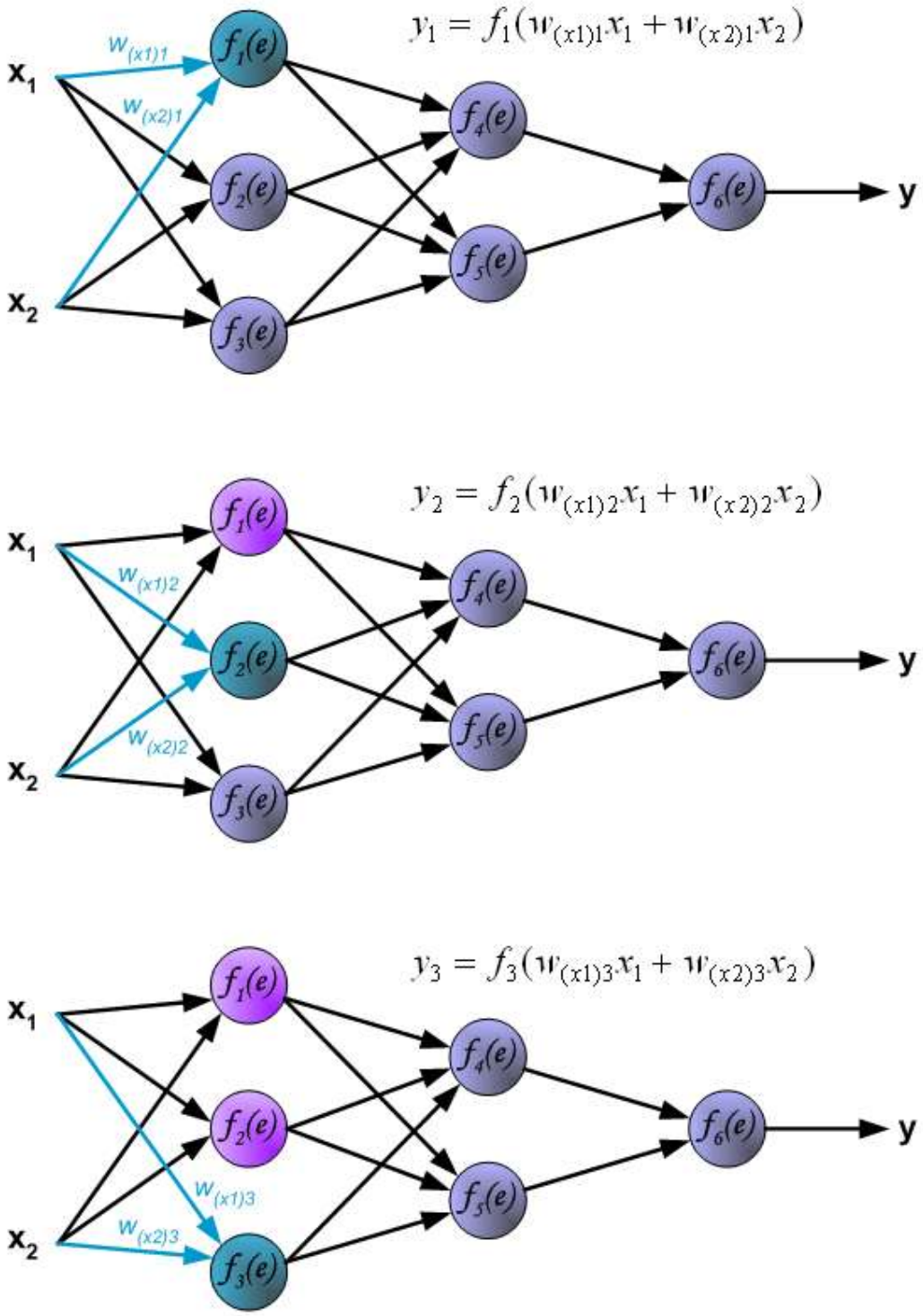


Рисунок 2.3 – Мережа розповсюдження сигналу на вхідному шарі

Символи W_{mn} відповідають за зберігання вагових коефіцієнтів з'єднань між виходом нейрона m і входом нейрона n на наступному шарі, так відбувається поширення сигналів через прихований шар (рис. 2.4).

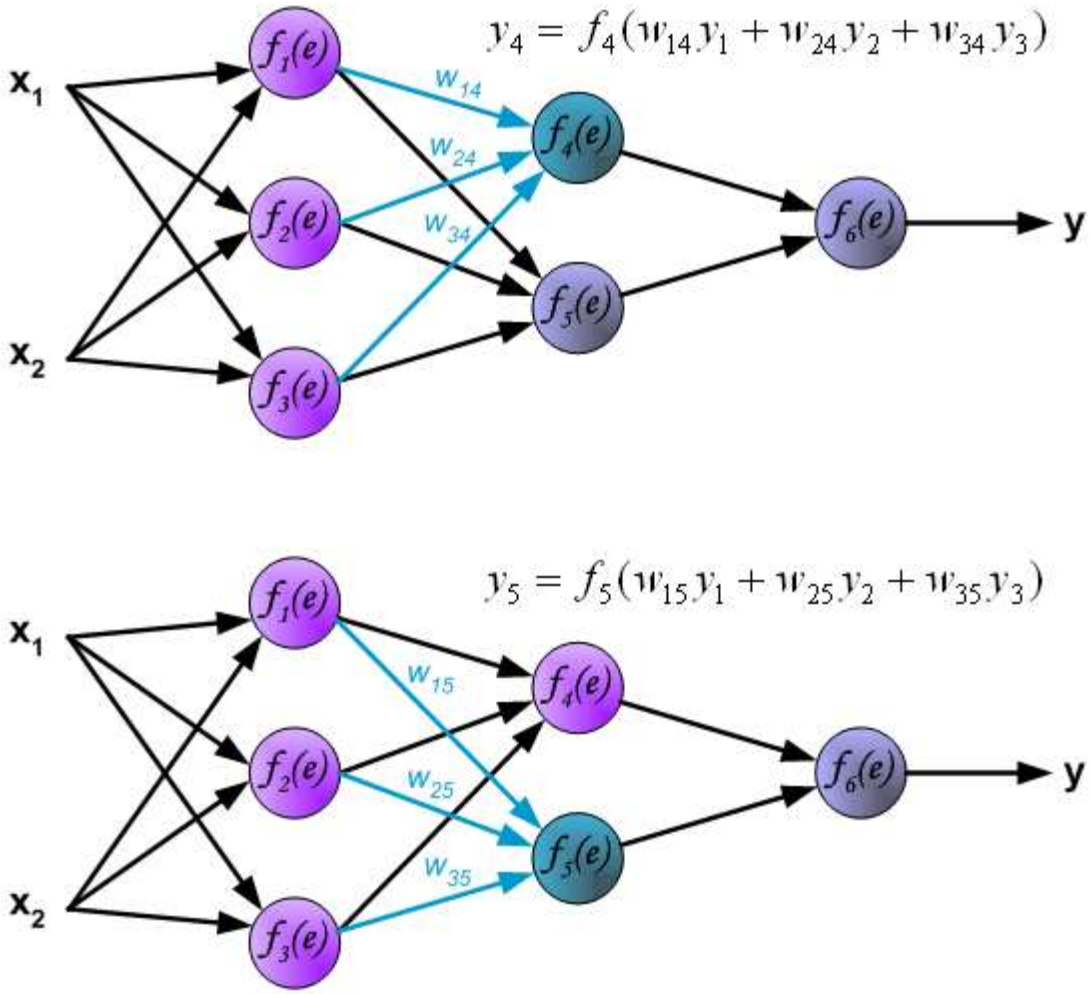


Рисунок 2.4 – Мережа розповсюдження сигналу на прихованому шарі

Розповсюдження сигналів мережею через вихідний шар подано на рис. 2.5.

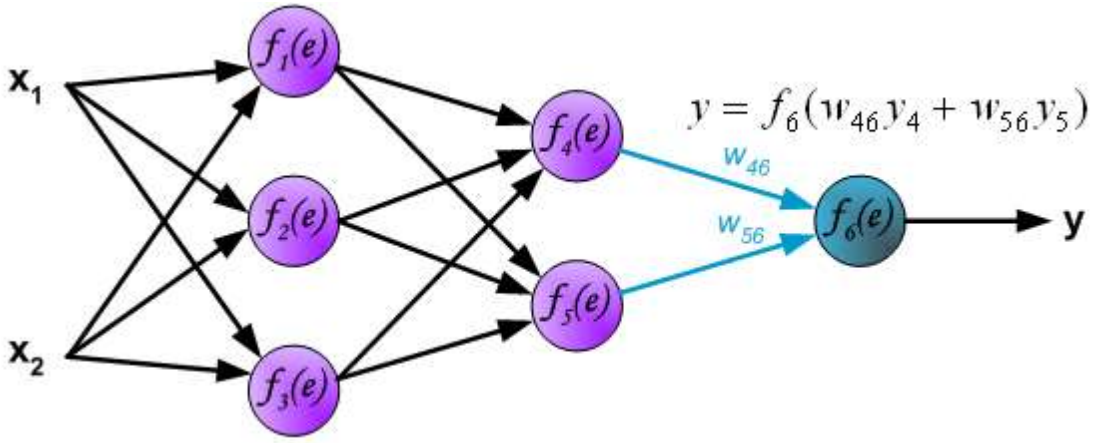


Рисунок 2.5 – Мережа розповсюдження сигналу у вихідному шарі

На наступному етапі алгоритму вихідний сигнал мережі y підлягає порівнянню з цільовим вихідним значенням k, яке знаходиться в наборі навчальних даних. Різниця між цими сигналами, y та k відповідно, називається сигналом помилки δ нейрона вихідного шару (рис. 2.6).

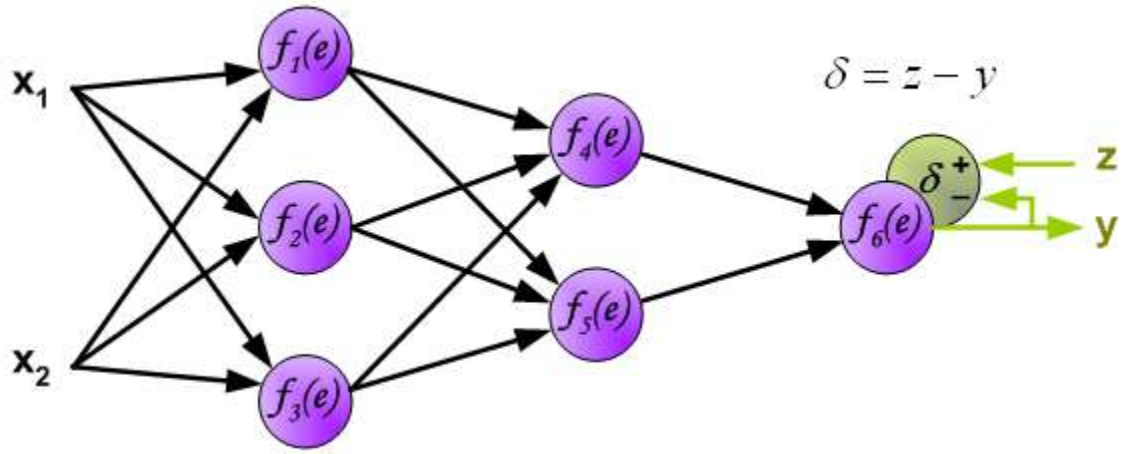


Рисунок 2.6 – Помилка нейрона вихідного шару

Не можна здійснити обчислення сигналу помилки для внутрішніх нейронів, оскільки вихідні значення цих нейронів невідомі. Доволі довгий час метод навчання багатошарових мереж, який приводить до потрібних результатів був невідомий. Тільки у середині 80-х років XX століття був розроблений алгоритм зворотного поширення помилки. Основна мета полягала в тому, щоб поширити сигнал помилки δ (обчислений за один крок навчання) назад до всіх нейронів, вихідні сигнали яких були вхідними для заключного нейрона (рис. 2.7) [6].

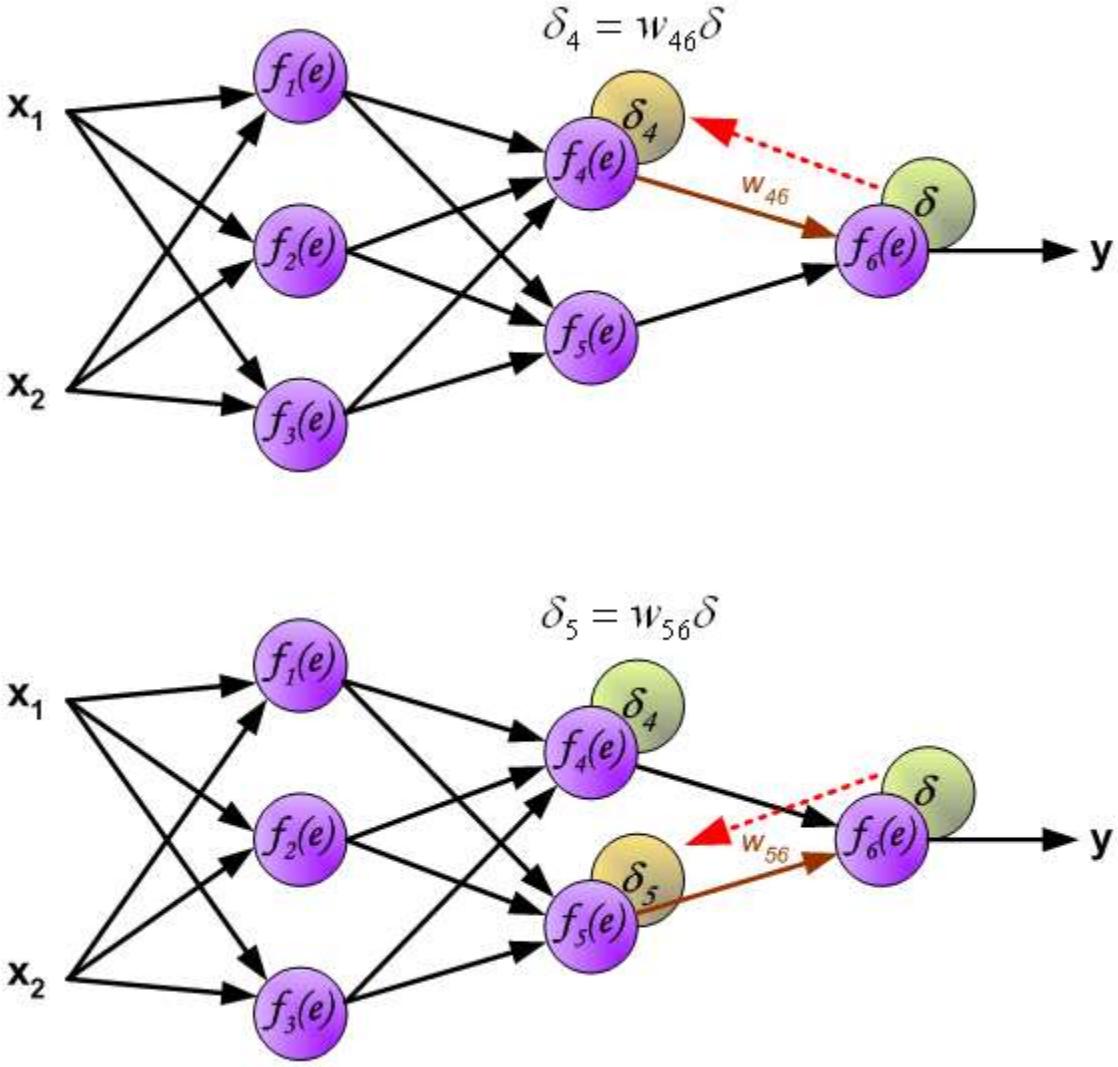


Рисунок 2.7 – Зворотне поширення помилки

Коефіцієнти W_{mn}, які використовуються для зворотного поширення помилок, дорівнюють коефіцієнтам, які використовуються під час обчислення вихідного сигналу. Змінюється лише напрямок розповсюдження даних (сигнали поширюються від виходу до входу один за одним). Ця техніка використовується на всіх шарах мережі. У випадку коли помилки надходять від кількох нейронів, вони додаються (рис. 2.8).

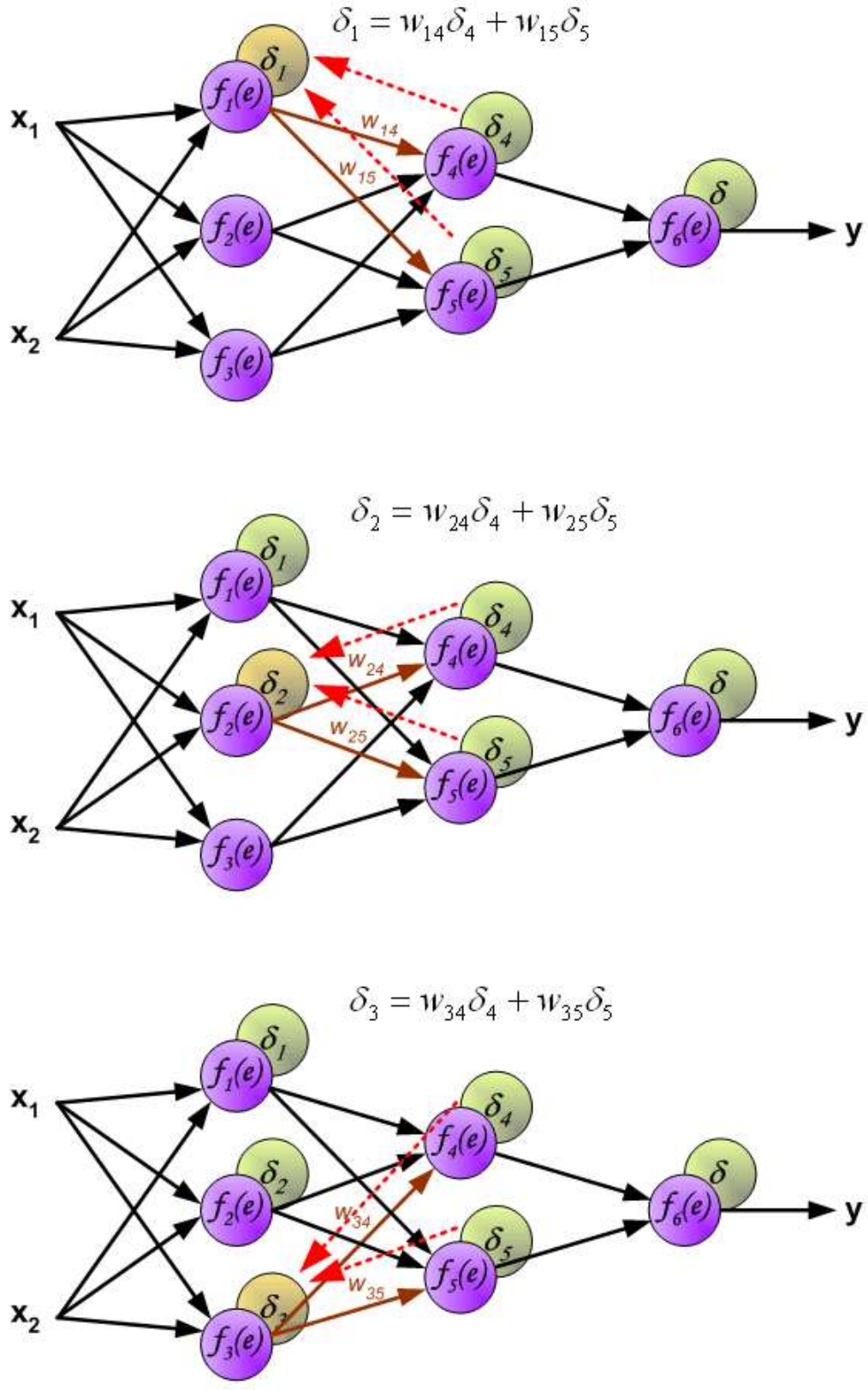


Рисунок 2.8 – Зворотне поширення помилки для прихованого шару

Коли відбувається обчислення величини похибки для кожного нейрона, вагові коефіцієнти кожного вхідного вузла (дендрита) нейрона можуть бути змінені. У наведених нижче формулах $df(e)/de$ подає похідну функції від активації нейрона, ваги якої можуть змінюватися (рис. 2.9).

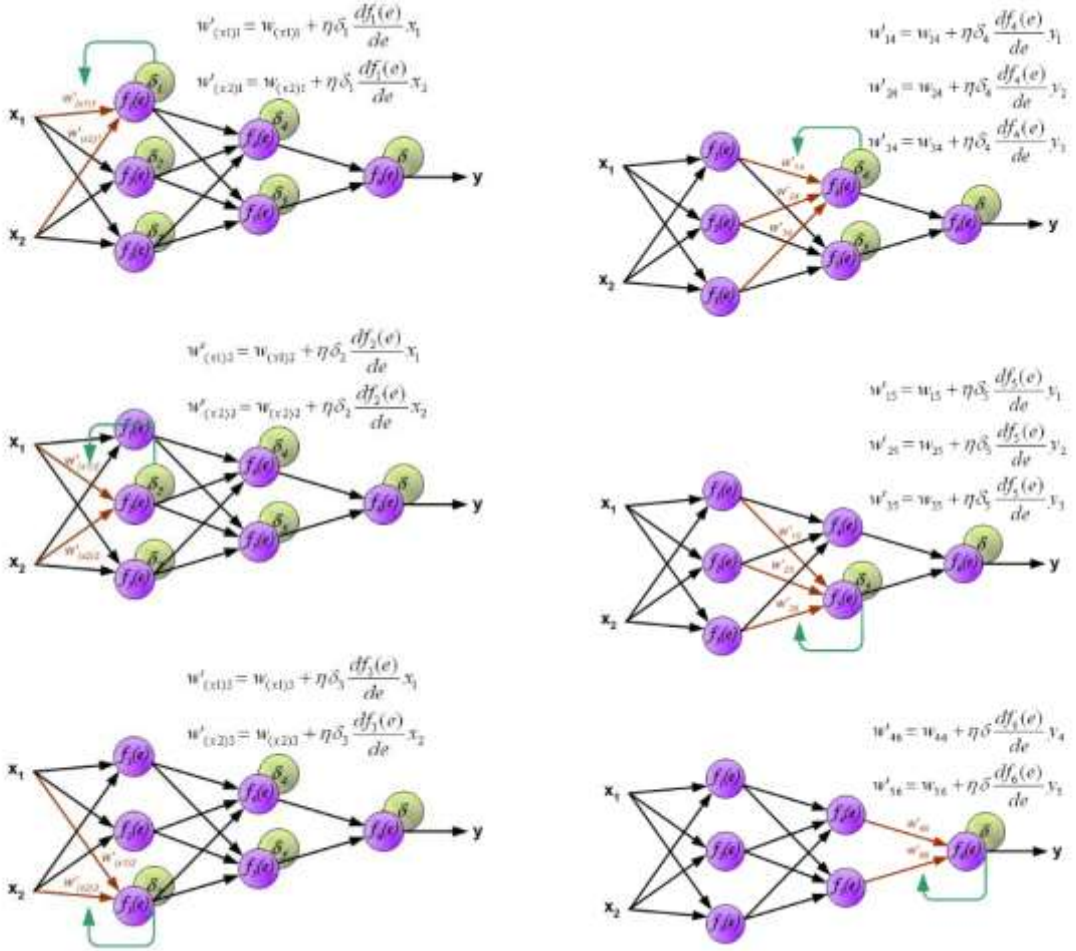


Рисунок 2.9 – Метод градієнтного спуску при зворотному поширенні помилки

Коефіцієнт η впливає на швидкість навчання мережі. Існує відповідний перелік способів для обрання значень цього параметра. Перший спосіб полягає в тому, щоб почати процес навчання з великого значення параметра η. Поки відбувається встановлення вагових коефіцієнтів, показник поступово знижується. Другий, більш складний метод, починає навчання з малого значення параметра η. Під час процесу навчання параметр збільшується, коли навчання просувається, а потім знову зменшується на завершальному етапі. Використання останнього методу дає перевагу, оскільки за допомогою нього можна визначити знаки вагових коефіцієнтів [6].

2.2. Вибір мови програмування для реалізації оптичного розпізнавання символів

Мова програмування Python дуже зручна для початківців і відносно проста у вивченні. Не дивно, що більшість університетів викладають Python як початкову мову (рис. 2.10). Завдяки своєму спрощеному синтаксису, подібному до англійської, він дозволяє новачкам зосередитися на основах програмування, концепціях і практиках кодування, а не на тонкощах структури мови. Як наслідок, студенти швидко сприймають специфічне мислення розробника.

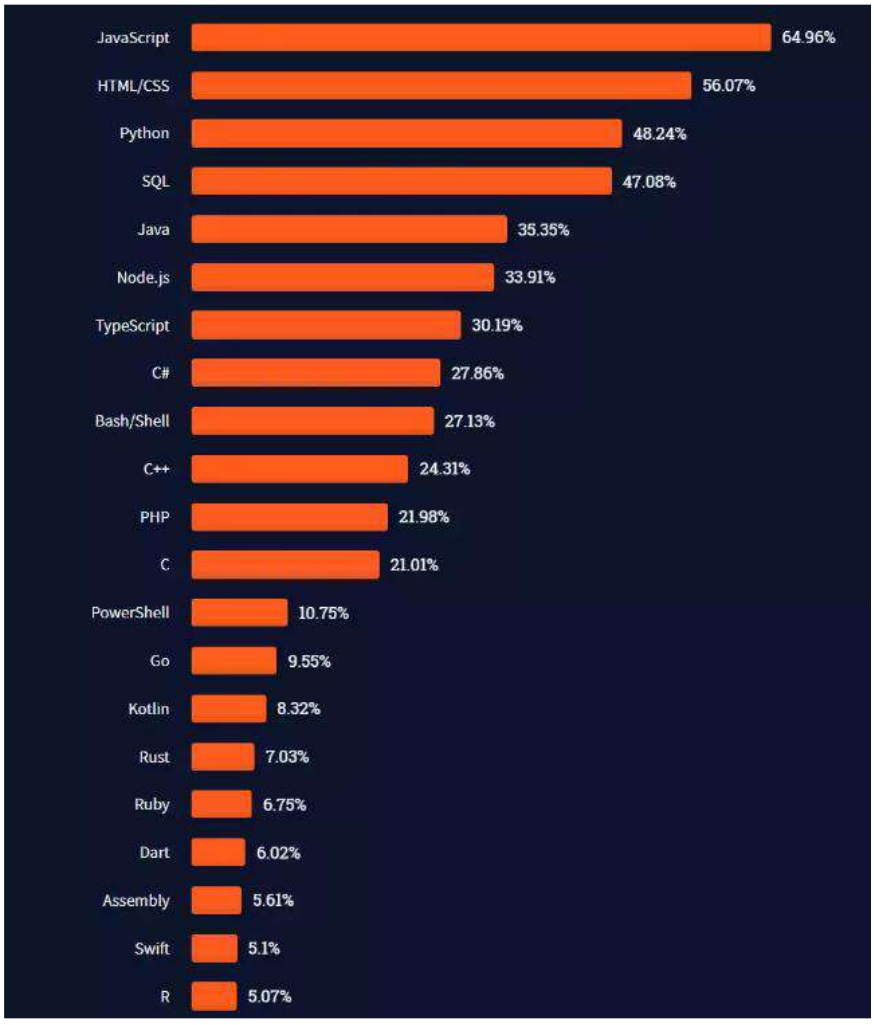


Рисунок 2.10 – Найпопулярніші технології програмування

Чіткий, лаконічний синтаксис спрощує та прискорює не лише вивчення Python, але й створення програмного забезпечення з його допомогою. Крім того, його стандартна бібліотека надає багато готових функцій, які дозволяють програмістам працювати з Інтернет-протоколами, керувати операційними

системами, маніпулювати даними або інтегрувати веб-сервіси з меншими зусиллями.

Як наслідок, та сама програма, швидше за все, займатиме менше рядків коду на Python, ніж на більш докладній Java або C++. Це робить її затребуваною технологією для створення прототипів програмного забезпечення.

Продуктивність розробки підвищується ще більше завдяки доступним структурам Python, які надають попередньо закодовані компоненти. Замість того, щоб створювати все з нуля, програмісти можуть використовувати готові до використання будівельні блоки.

Серед найпопулярніших фреймворків Python для швидкої розробки веб-додатків є Django, Flask, Falcon.

Python не залежить від платформи: можна запускати той самий вихідний код у різних операційних системах, будь-то macOS, Windows або Linux. Переносимість досягається за рахунок байт-коду та віртуальної машини Python (PVM), які служать посередниками між розробником і фактичним ЦП, що виконує програму.

Крім того, Python легко об’єднується з іншими мовами за допомогою таких розширень, як Cython для C, Gython для Go, Jython для Java та IronPython для .Net. Вони дозволяють розробникам змішувати мови, запозичувати функціональність, якої не вистачає їхній основній технології, і запускати “чужий” код у своїх програмах.

Вчені, інженери, фінансові аналітики, математики та інші експерти, які займаються дослідженнями на основі даних, є ключовими користувачами екосистеми SciPy на основі Python. Він містить такі потужні інструменти для аналізу та візуалізації даних, як pandas, NumPy, scikit-learn і matplotlib.

Для програмної реалізації інформаційної технології оптичного розпізнавання символів будуть використовуватися такі інструменти як NumPy, Pyplot та Image.

2.3. Принцип роботи та архітектура інформаційної технології

Метод навчання нейронної мережі

Спочатку для зручності розуміння продемонстровано метод навчання та принцип роботи мережі, а потім її архітектура.

Нейронна мережа містить три шари: вхідний (вхідними даними для нього є пікселі вхідного зображення), прихований і вихідний. Кожен нейрон на вихідному шарі класифікує вхідний набір на класифікований вихід. Це означає, що для навчання символу “a” лише нейрон, який класифікує символ “a”, запускає значення, близьке до 1, решта запускає значення, близьке до 0. Точність нейронної мережі визначається як близькість фактичного до цільового результату, який покращується шляхом зменшення помилки, визначеної за цією формулою (2.1):

$$Загальна\ помилка = 1/2\Sigma(ak - bk)^2 \quad (2.1)$$

де:

• a – вихідне значення;

• b – цільове значення;

• k – вихідний нейрон.

Цей підхід до зменшення середньої квадратичної помилки називається градієнтним спуском і використовується в поєднанні з навчальним алгоритмом зворотного поширення помилки, регулюючи вагові коефіцієнти мережі таким чином, щоб загальна помилка наближалася до глобального мінімуму. Ваги та зміщення ініціалізуються випадковими числами від -0,5 до 0,5. Якщо ваги були ініціалізовані таким чином, що вони лежать між 0 і 1, нейронна мережа ніколи не зійдеться. Це пов’язано з тим, що вхідні дані для всіх прихованих нейронів будуть вищими за активний вхідний діапазон сигмоїдної функції, в результаті чого вихідні дані завжди дорівнюють 1.

Мережа розроблена, а потім навчена на наборі входів «A» так, що для кожного елемента «B» в наборі «A» лише нейрон «K», який класифікує елемент «B», запускає найвище значення серед усіх інших вихідних нейронів. Цей алгоритм навчання описано в псевдокоді нижче:

```
Initialize weights and biases ()
while Total Error > Target Error
   for each element B in set A
      target output = {1 for neuron K classifying B; 0 otherwise}
      feed forward through network ()
      calculate total error ()
      back propagate through net and adjust weights ()
loop until criteria is met
```

Після навчання мережі на всіх елементах набору «А» її можна використовувати для розпізнавання. Кількість вихідних нейронів залежить від кількості символів, які потрібно класифікувати. Таким чином, якщо потрібно класифікувати символи у кількості X, то X нейронів будуть присутні. Що стосується кількості входів, то мережа має M*N входів, де M – ширина, а N – висота зображення символу. Прихований шар містить 100 нейронів. Оскільки дана нейронна мережа може класифікувати вхідні дані лише тоді, якщо їхня кількість фіксована, зображення має пройти етап попередньої обробки, щоб забезпечити таку однорідність вхідних даних.

Процес попередньої обробки

Як було описано в попередньому пункті, кількість входів до нейронної мережі має бути фіксованою. Таким чином, щоб забезпечити таку однаковість, розмір зображення змінюється до фіксованої ширини та висоти. Цього можна досягти, виконавши відповідні операції [10]:

• бінаризація, метод перетворення зображення в чорно-біле (тобто чорний = 1, білий = 0);

• обрізання, визначення меж символів на зображенні (тобто ліва, права, верхня та нижня межі);

• нормалізація, зміна розміру до потрібної ширини та висоти.

Вище наведені кроки продемонстровано на прикладі літери А (рис. 2.11).

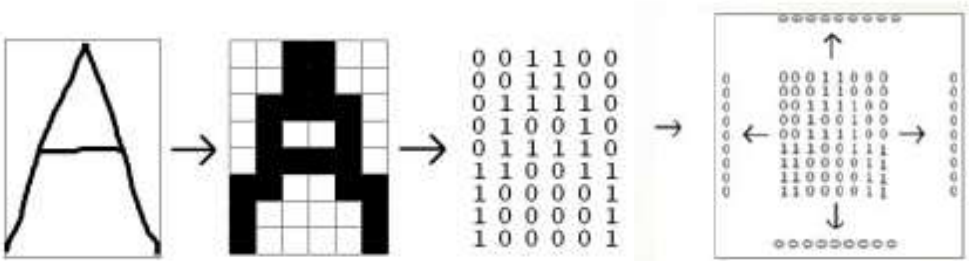


Рисунок 2.11 – Операції над зображенням

На рис. 2.11 показано двійкові вхідні дані, для яких визначаються межі, перетворюються на матричний формат і потім змінюються розміри. Потім вихідні дані етапу попередньої обробки вводяться в нейронну мережу.

Архітектура нейронної мережі

Загальна архітектура нейронної мережі продемонстрована на рис. 2.12. Вхідні дані цієї системи – I, блок P представляє фазу попередньої обробки, яка виводить PI (попередньо оброблені вхідні дані). PI служить вхідним сигналом для мережі, цей вхід є матрицею 18×16, яка помножена на ваги, що з’єднують входи з нейронами в прихованому шарі. Таким чином, вагова матриця між вхідним і прихованим шаром має розмір 18×16×100, де 100 – це кількість нейронів у прихованому шарі. Функція активації, яка використовується як для нейронів прихованого, так і для вихідного шару, – це логістична сигмоїдна функція. Це означає, що їхні виходи знаходяться в діапазоні від 0 до 1. Виходи прихованих нейронів подаються як вхідні дані на вихідний рівень. Потім ці входи множаться на ваги, що з’єднують приховані нейрони з вихідними нейронами. Вихідний нейрон із максимальним вихідним значенням означає найбільшу ймовірність. Таким чином, усі нейрони запускатимуть значення близьке до 0, що вказує на низьку ймовірність, за винятком того, що класифікує вхідний символ, буде запускати значення близьке до 1 (рис. 2.12).

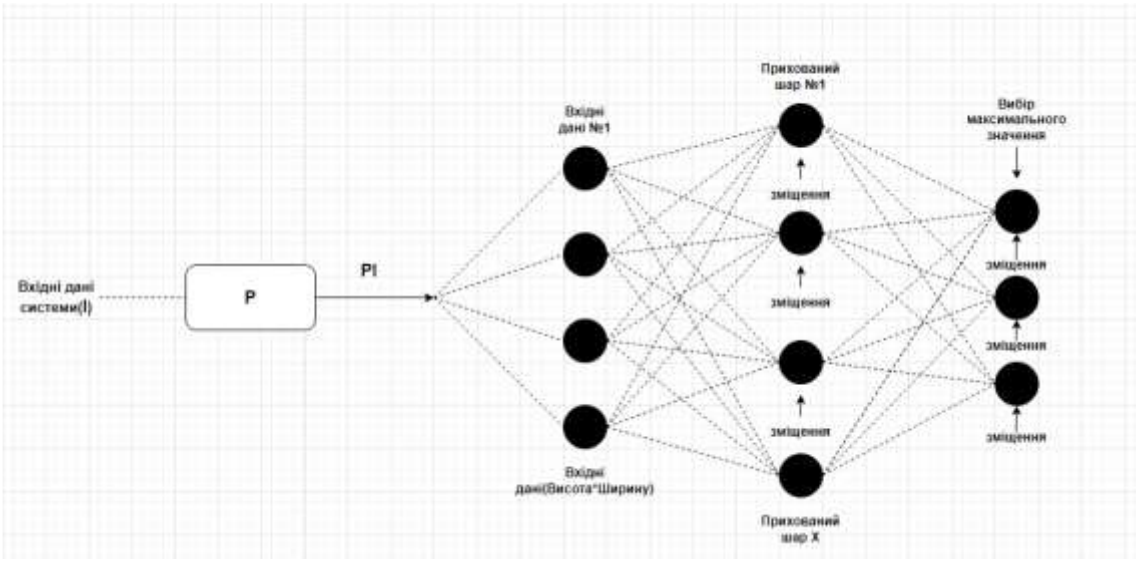


Рисунок 2.12 – Архітектура мережі

Процес розпізнавання символів та сканування документів

У документі, який потрібно оцифровувати, спочатку шукають абзаци, потім рядки з абзаців, слова з рядків і, зрештою, символи зі слів. Ці символи є вхідними даними для нейронної мережі. Мережа виведе цифровий символ, який відповідає символу на зображенні. Абзац визначається як комбінація чорних пікселів із білим проміжком розміром кілька десятків пікселів після нього (залежить від розміру шрифту). Після того, як абзац знайдено, рядок знаходить спочатку виявленням горизонтального чорного пікселя, який визначає верхню частину, а потім останній знайдений чорний горизонтальний піксель визначає нижню частину. Після того, як рядок знайдено, символи знаходять спочатку виявленням вертикального чорного пікселя, який визначає лівий, а потім останній знайдений чорний вертикальний піксель, що визначає правий. Це повторюється доти, доки не буде знайдено жодного вертикального чорного пікселя (тобто жодних символів у рядку). Коли всі символи знайдено, слово визначається як комбінація символів, за якою йде білий проміжок розміром 20 пікселів (рис. 2.13).

Після сканування документа на наявність символів кожен символ проходить етап попередньої обробки, а потім вводиться в нейронну мережу для розпізнавання.

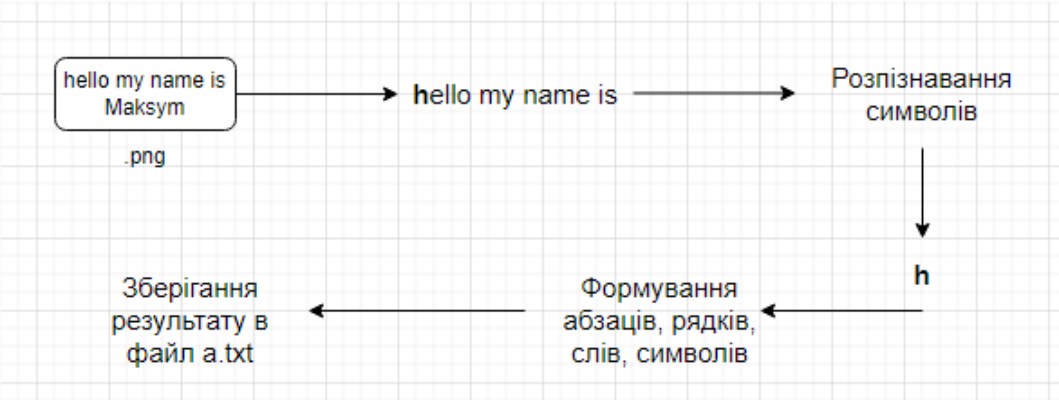


Рисунок 2.13 – Процес розпізнавання символів

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОПТИЧНОГО РОЗПІЗНАВАННЯ СИМВОЛІВ

3.1. Реалізація модуля розпізнавання символів

Для реалізації даного модуля спочатку необхідно підключити бібліотеки numpy, pyplot та image, а також імпортувати раніше створені модулі попередньої обробки та модуля вилучення символів та рядків, які знадобляться в реалізації ряду функцій (рис. 3.1). Matplotlib.pyplot – це набір функцій, завдяки яким matplotlib працює як MATLAB. NumPy – це бібліотека Python, яка має великий обсяг функцій для роботи з масивами даних. Модуль Image надає клас із такою ж назвою, який використовується для представлення зображення PIL. Модуль також надає низку стандартних функцій, включаючи функції для завантаження зображень із файлів і створення нових зображень.

```
#підключення бібліотек
from PIL import Image as im
import numpy as np
import matplotlib.pyplot as plt

# підключення файлів
import CropAndNormalize as cAn
import LineExtraction as lN
```

Рисунок 3.1 – Підключення бібліотек та модулів

Ініціалізація алфавіту для якого буде відбуватися оптичне розпізнавання символів (рис 3.2).

```
#Алфавіт для навчання
alphabet = ['a','b','c','d','e','f','g','h','i',
            'j','k','l','m','n','o','p','q','r',
            's','t','u','v','w','x','y','z']
```

Рисунок 3.2 – Ініціалізація алфавіту

Для реалізації функції ініціалізації вагових коефіцієнтів та зміщення використовується функція numpy.random.random для повернення випадкового плаваючого значення у визначеному діапазоні, у нашому випадку від -0.5 до 0.5 (рис. 3.3).

```
#Ініціалізація вагових коефіцієнтів та зміщення
def initWeights(wth,hght,countOfHiddenNeurons):
    #ваги з віхдного до прихованого шару
    Wi_h = np.random.random(size=(countOfHiddenNeurons,hght,wth))-0.5
    #ваги з прихованого до вхідного шару
    Wh_o = np.random.random(size=(26,countOfHiddenNeurons))-0.5
    # зміщення прихованого шару
    Bh = np.random.random(countOfHiddenNeurons) - 0.5
    # зміщення вихідного шару
    Bo = np.random.random(26) - 0.5

    return Wi_h, Wh_o, Bh, Bo
```

Рисунок 3.3 – Розрахунок вагових коефіцієнтів та зміщення

Для активації нейронів прихованого та вихідного шару реалізовано функцію активації – логістична сигмоїдна функція (рис. 3.4).

```
#Сигмоїдна функція активації нейронів
def logActivation(sum):
        out = 1 / (1 + np.exp(-sum))
        return out
```

Рисунок 3.4 – Функція активації

Для обчислення значень вихідних сигналів для всіх нейронів з вхідного до прихованого шару відбувається за допомогою вкладених циклів, а саме кількість прихованих шарів та кількість входів, потім розраховується добуток вагових коефіцієнтів до вхідних сигналів і до загальної суми всіх дендритів додається зміщення. Після чого, за допомогою логістичної функції відбувається обчислення вихідного сигналу прихованого нейрона та його зберігання до масиву outputOfHiddNeurons (рис. 3.5).

```python
#Розповсюдження сигналу через мережу
def feedForward(normalized_data, Wi_h, Wh_o, Bh, Bo):

    n_h = 0
    [countOfHiddNeurons,hght,wth] = Wi_h.shape
    #Розповсюдження сигналу з вхідного до прихованого шару
    outOfHiddNeurons = []
    netInForHiddNeurons = []

    for hiddNeuron in range (0,countOfHiddNeurons):

        for i in range (0,hght):
            for j in range (0,wth):
                WxP = Wi_h[hiddNeuron,i,j] * normalized_data[i,j]
                n_h = n_h + WxP

        n_h = n_h + Bh[hiddNeuron]
        outOfHiddNeurons.append(logActivation(n_h))
        netInForHiddNeurons.append(n_h)
        n_h = 0
```

Рисунок 3.5 – Розповсюдження сигналу з вхідного до прихованого шару

Отримані вихідні сигнали з прихованого шару outOfHiddNeurons множимо на вагові коефіцієнти з прихованого до вихідного шару. Знаходимо вхід для k-го вихідного нейрона, додаючи зміщення, та за допомогою логістичної функції обчислюємо вихід для вихідних нейронів та зберігаємо в масив outputOfOutNeurons (рис. 3.6).

```python
#Розповсюдження сигналу з прихованого шару до вихідного
outHiddXweightsH_O = outOfHiddNeurons * Wh_o
netInForOutNeurons = np.sum(outHiddXweightsH_O, axis= 1)
outputOfOutNeurons = []

for outputNeuron in range(0,26):

    totalInForNeuron = netInForOutNeurons[outputNeuron] + Bo[outputNeuron]
    outputOfOutNeurons.append(logActivation(totalInForNeuron))

return outputOfOutNeurons, outOfHiddNeurons
```

Рисунок 3.6 – Розповсюдження сигналу з прихованого до вихідного шару

Для кожного виходу вихідного нейрона обчислюємо похибку від цільового виходу (рис. 3.7).

```python
#Обрахунок похибки для кожного виходу вихідного нейрона
def calculateMistakeAtOutput(outputOfOutNeurons, targetOutput):
    outMistake =[]
    for outputNeuron in range(0,26):
        outputNeuronError = outputOfOutNeurons[outputNeuron] - targetOutput[outputNeuron]
        outMistake.append(outputNeuronError)
    return outMistake
```

Рисунок 3.7 – Обчислення похибки

Для реалізації коригування вагових коефіцієнтів реалізовано функцію з алгоритмом зворотного поширення. Спочатку відбувається збереження попередніх вагових коефіцієнтів. Зворотній пошук відбувається з вихідного до прихованого шару. Обчислення коригування відбувається, як добуток швидкості навчання, помилкою на поточному виході нейрона, сигмоїдною похідною та виходом поточного прихованого нейрона (рис. 3.8). Нові вагові коефіцієнти знаходяться за формулою (3.1):

$$Wnew = імпульс * Wold - коригування \quad (3.1)$$

```python
#Зворотній пошук та коригування вагових коефіцієнтів
def backPropagate(Wi_h, Wh_o, Bh, Bo, normalized_data, outMistake, outputOfOutNeurons, outOfHiddNeurons, learningRate, momentum):
    #збереження попередніх вагових коефіцієнтів
    prevWh_o = np.array(Wh_o[:,:])
    prevWi_h = np.array(Wi_h[:,:])

    [countOfHiddNeurons,hght,wth] = Wi_h.shape
    #Зворотній пошук від вихідного до прихованого шару
    #коригування вагових коефіцієнтів
    for outNeuron in range(0,26):
        for hiddNeuron in range(0,countOfHiddNeurons):
            adj = (learningRate * outMistake[outNeuron] * outputOfOutNeurons[outNeuron] * (1 - outputOfOutNeurons[outNeuron]) * ou
            Wh_o[outNeuron, hiddNeuron] = (momentum * Wh_o[outNeuron, hiddNeuron]) - adj
```

Рисунок 3.8 – Обчислення коригування та нових вагових коефіцієнтів

Наступним етапом відбувається зворотній пошук з прихованого до вхідного шару. Відбувається обчислення дельта-похибки на кожному вихідному нейроні щодо поточного прихованого нейрона. За допомогою циклу по всім вхідним ваговим коефіцієнтам, що з’єднуються до поточного прихованого нейрона відбувається обчислення дельта загальної помилки відносно ваги, яку потрібно скоригувати. Ці вагові коефіцієнти з’єднують вхідний та поточний прихований шар (рис. 3.9).

```
    #Зворотній пошук від прихованого до вхідного шару
    #коригування вагових коефіцієнтів
    for hiddNeuron in range(0,countOfHiddNeurons):
        delTotalMistake_hiddNeuron = 0
        for outNeuron in range(0,26):
            #Обчислення дельта похибки
            delMistakeOutNeuron_hiddNeuron = outMistake[outNeuron] *  outputOfOutNeurons[outNeuron] * (1-outputOfOutNeurons[outNe
            delTotalMistake_hiddNeuron = delTotalMistake_hiddNeuron + delMistakeOutNeuron_hiddNeuron

        for i in range (0,hght):
            for j in range (0,wth):
                delTotalMistake_inTohiddNeuronWeight = delTotalMistake_hiddNeuron * outOfHiddNeurons[hiddNeuron] *(1 - outOfHiddN
                Wi_h[hiddNeuron,i,j] = (momentum * Wi_h[hiddNeuron,i,j]) - (learningRate * delTotalMistake_inTohiddNeuronWeight)

    return Wi_h, Wh_o
```

Рисунок 3.9 – Обчислення коригування та нових вагових коефіцієнтів

За навчання мережі відповідає функція trainNetwork. За допомогою вкладених циклів, а саме досягнення цільової помилки, символів які потрібно навчити та кількість навчальних зразків відбувається процес попередньої обробки, обчислення похибки для вихідних нейронів вихідного шару, регулювання вагових коефіцієнтів та обчислення середньо квадратичної помилки (рис. 3.10).

```
# Навчання мережі
def trainNetwork(Wi_h, Wh_o, Bh, Bo,hght,wth,countOfTraining,learningRate,momentum,targetError):
    count = 0
    factualError = 1
    mistakeList = []
    y_axis = []

    while factualError > targetError:
        for characterToTrain in range(0,26):
            targetOut = np.zeros(26)
            targetOut[characterToTrain] = 1
            # кількість навчальних зразків
            for n in range (0,countOfTraining):
                #бінаризація та нормалізація
                training_S = 'samples/%s%d.png' %(alphabet[characterToTrain],n)
                char_in = im.open(training_S)
                bAndW = cAn.convertToBW(char_in)
                reverseBW = cAn.toggleOnesAndZeros(bAndW)
                cutBW = cAn.crop(reverseBW)
                normalized_data = cAn.normalize(cutBW,wth,hght)
                #завершення попередньої обробки

                outputOfOutNeurons, outputOfHiddNeurons = feedForward(normalized_data, Wi_h, Wh_o, Bh, Bo)
                outMistake = calculateMistakeAtOutput(outputOfOutNeurons, targetOut)
                Wi_h, Wh_o = backPropagate(Wi_h, Wh_o, Bh, Bo, normalized_data, outMistake, outputOfOutNeurons, outputOfHiddNeurons, learningRate,momentum)

        #Обчислення середньоквадратичної помилки
        factualError = 0
        for x in range(0,26):
            rootMeanSquare = 0.5 * outMistake[x]**2
            factualError = factualError + rootMeanSquare
```

Рисунок 3.10 – Навчання нейронної мережі

Для повернення розпізнаних символів реалізовано функцію recognizeChar (рис. 3.11). Розпізнаним символом є нейрон із найвищим результатом.

```
#Розпізнавання зображення
def recognizeChar(inNormalizedData,Wi_h,Wh_o,Bh,Bo):
    outputOfOutNeurons, outputOfHiddNeurons = feedForward(inNormalizedData, Wi_h, Wh_o, Bh, Bo)
    #розпізнаним символом є нейрон із найвищим результатом
    prefOut = np.argmax(outputOfOutNeurons)
    return alphabet[prefOut]
```

Рисунок 3.11 – Функція повернення розпізнаних символів

3.2. Попередня обробка зображення

Для реалізації даного модуля знадобляться бібліотеки numpy та image. Функції convertToBW та toggleOnesAndZeros відповідають за бінаризацію, а саме чорному кольору буде відповідати 1, а білому 0 (рис. 3.12).

```
def convertToBW(imageIn):
    blackAndWhite = imageIn.convert('1')
    blackAndWhite = np.array(blackAndWhite)*1
    return blackAndWhite


def toggleOnesAndZeros(blackAndWhite):
    return (blackAndWhite ^1)
```

Рисунок 3.12 – Функції бінаризації

Наступним кроком в процесі попередньої обробки зображення є функція визначення меж зображення, а саме лівої, правої, верхньої та нижньої меж (рис. 3.13, 3.14).

```
def crop(blackAndWhiteToggled):

    [numberOfRowPixels , numberOfColumnPixels] = blackAndWhiteToggled.shape


    verticalSumOfBlackPixels = np.sum(blackAndWhiteToggled,axis=0)
    leftDetected = False
    for i in range(0,numberOfColumnPixels):
        if verticalSumOfBlackPixels[i] > 0 and leftDetected == False:
            leftDetected = True
            left = i
        elif verticalSumOfBlackPixels[i] > 0 and leftDetected == True:
            right = i
```

Рисунок 3.13 – Визначення лівої та правої меж

```
horizontalSumOfBlackPixels = np.sum(blackAndWhiteToggled,axis=1)
topDetected = False
for i in range(0,numberOfRowPixels):
    if horizontalSumOfBlackPixels[i] > 0 and topDetected == False:
        topDetected = True
        top = i
    elif horizontalSumOfBlackPixels[i] > 0 and topDetected == True:
        bottom = i
```

Рисунок 3.14 – Визначення верхньої та нижньої меж

Останнім етапом в процесі попередньої обробки є нормалізація, за якої відбувається зміна зображення до бажаного розміру. Для зміни розміру зображення використовується функція resize в бібліотеці image. Одним із аргументів даної функції є resample – це фільтр, який потрібно використовувати для повторної вибірки. У нашому випадку використовується HAMMING, оскільки за допомогою нього створюється більш чітке зображення, ніж з BILINEAR, і не має дислокацій на локальному рівні, як з BOX (рис. 3.15).

```
def normalize(character_in,width,height):

    normalized = character_in.resize((width,height),im.HAMMING)
    NormalizedArray = np.array(normalized)
    return NormalizedArray
```

Рисунок 3.15 – Функція нормалізації

3.3. Реалізація модуля вилучення символів та рядків

Для реалізації даного модуля застосовано бібліотеки numpy та image, а також необхідно імпортувати раніше створений модуль попередньої обробки. Спочатку відбувається розпізнавання абзаців. Для того, щоб вважатися новим абзацом, пробіли мають бути більше за визначену кількість пікселів (рис. 3.16).

```python
def cropParagraphs(numberOfLines,topOfLines,bottomOfLines):
    locationOfNewLines = []
    for i in range(1,numberOfLines):
        whiteSpaceDistance = topOfLines[i] - bottomOfLines[i-1]
        if whiteSpaceDistance > 60:
            locationOfNewLines.append(i-1)
    return locationOfNewLines
```

Рисунок 3.16 – Розпізнавання абзаців

Наступним кроком відбувається розпізнавання рядків та символів. Відбувається виявлення перших та останніх чорних пікселів в рядку за рахунок чого відбувається створення списку, який містить усі виявлені рядки (рис. 3.17).

```python
def cropLines(blackAndWhite):                                                    #

    [numberRowPixels , numberColumnPixels] = blackAndWhite.shape
    h_firstBlackPixelDetected = False
    firstBlackPixelRow  = 0
    lastBlackPixelRow   = 0
    topOfLines = []
    bottomOfLines = []

    for i in range (0,numberRowPixels):
        sumOfAllPixelsInRow_i = sum(blackAndWhite[i,:])
        if sumOfAllPixelsInRow_i >= 1 and h_firstBlackPixelDetected == False:
            h_firstBlackPixelDetected = True
            firstBlackPixelRow = i
            lastBlackPixelRow  = i

        elif sumOfAllPixelsInRow_i >= 1 and h_firstBlackPixelDetected == True:
            lastBlackPixelRow  = i


        elif sumOfAllPixelsInRow_i < 1 and h_firstBlackPixelDetected == True:
            h_firstBlackPixelDetected = False
            topOfLines.append(firstBlackPixelRow)
            bottomOfLines.append(lastBlackPixelRow)


    numberOfLines = len(topOfLines)
    croppedLinesList = []
    for i in range(0,numberOfLines):
        croppedLine = blackAndWhite[(range(topOfLines[i],bottomOfLines[i])),:]
        croppedLinesList.append(croppedLine)

    return croppedLinesList,numberOfLines,topOfLines,bottomOfLines
```

Рисунок 3.17 – Розпізнавання рядків

3.3. Тестування роботи інформаційної системи

Для тестування роботи програмного коду було створено зображення з символами (рис. 3.18).

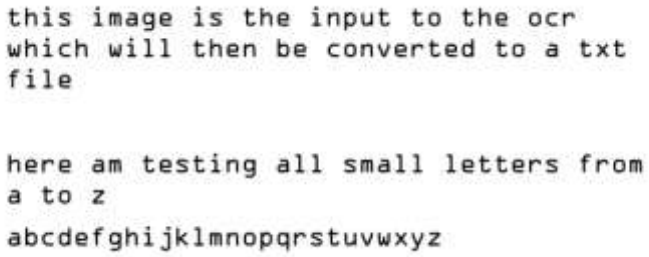


Рисунок 3.18 – Тестове зображення

Після запуску та виконання програми можна побачити як змінювалася загальна похибка відносно кількості проведених ітерацій в консольному (рис. 3.19) та графічному відображенні (рис. 3.20). Також отримуємо текстовий файл із розпізнаними символами (рис. 3.21).

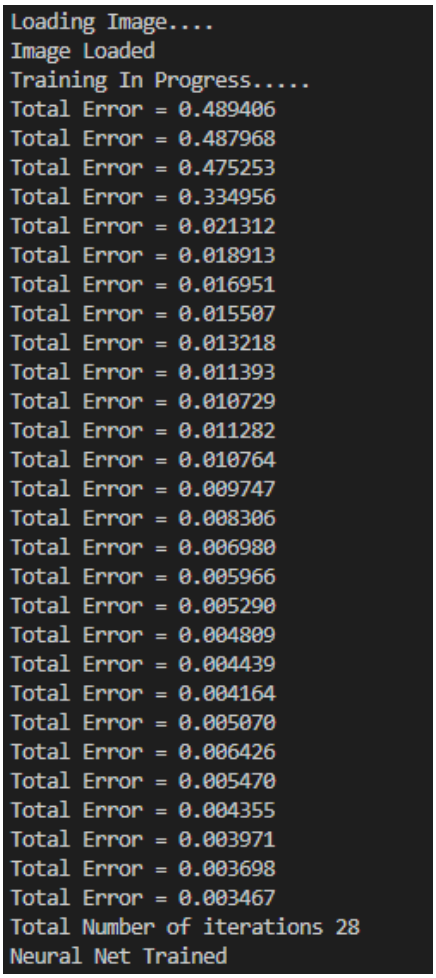


Рисунок 3.19 – Консольне відображення навчання мережі

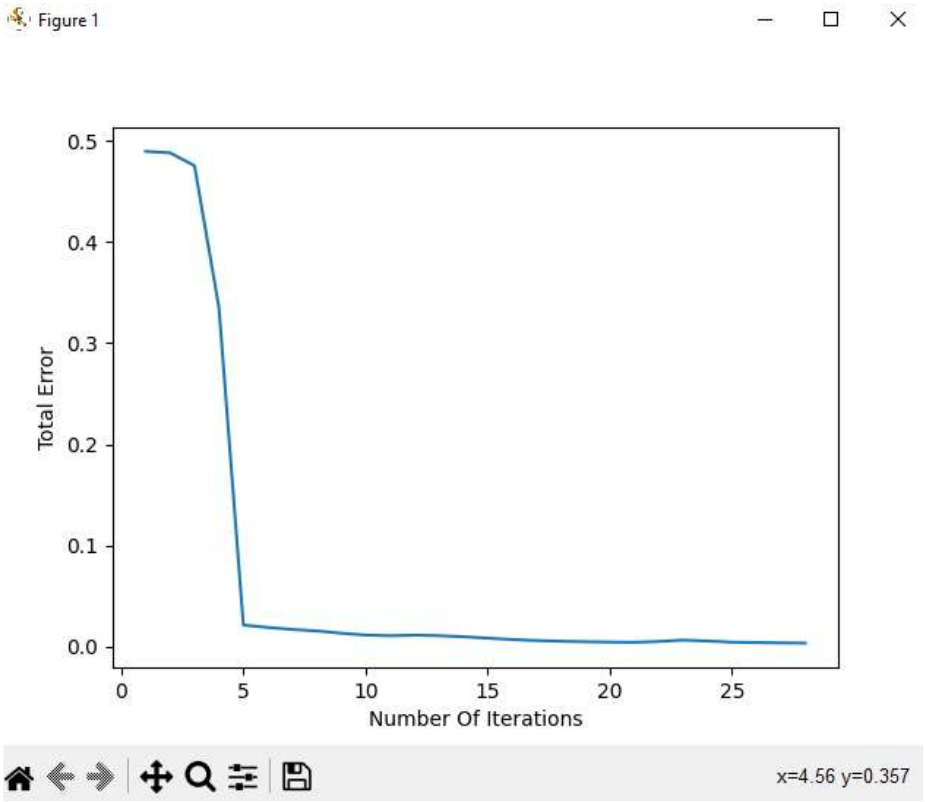


Рисунок 3.20 – Графічне відображення навчання мережі

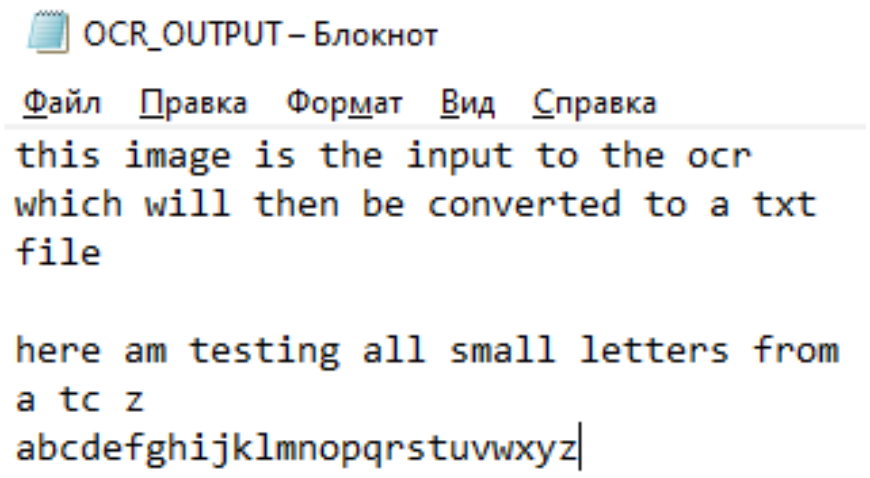


Рисунок 3.21 – Текстовий файл із розпізнаними символами

Для тестування продуктивності нейронної мережі змінимо швидкість навчання на 0.1, 0.5 та 1 (рис. 3.22, 3.23). Для активації використовується сигмоїдна функція, кількість прихованих нейронів становить – 55 та цільова середньоквадратична похибка дорівнює 0.0035.

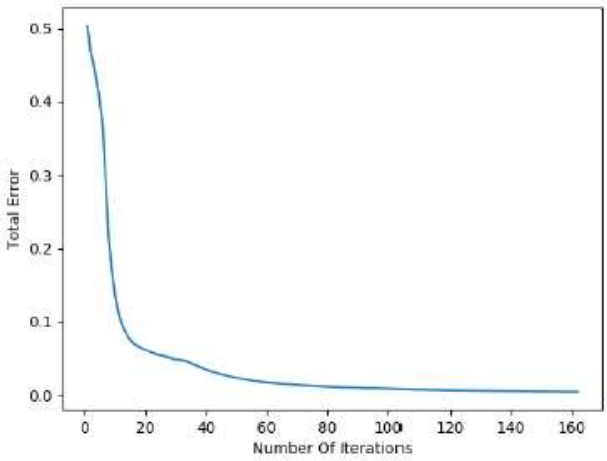


Рисунок 3.22 – Результат навчання зі швидкістю 0.1

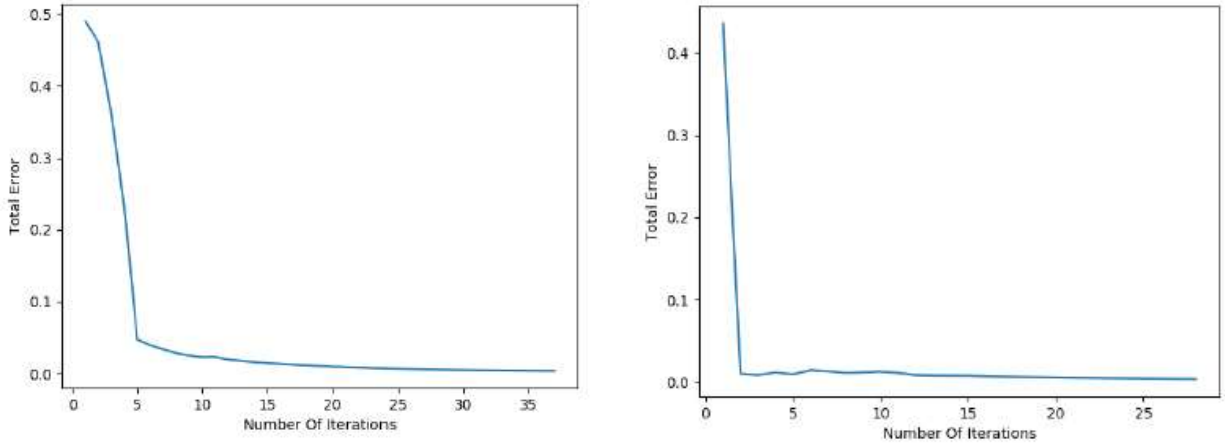


Рисунок 3.23 - Результат навчання зі швидкістю 0.5 та 1

Можемо зробити висновок, що чим більша швидкість навчання, тим швидше наближення до глобального мінімуму.

Для наступного тестового сценарію змінимо кількість прихованих нейронів, швидкість навчання 0.5, а цільова похибка буде 0.0035 та відстежимо продуктивність розпізнавання символів.

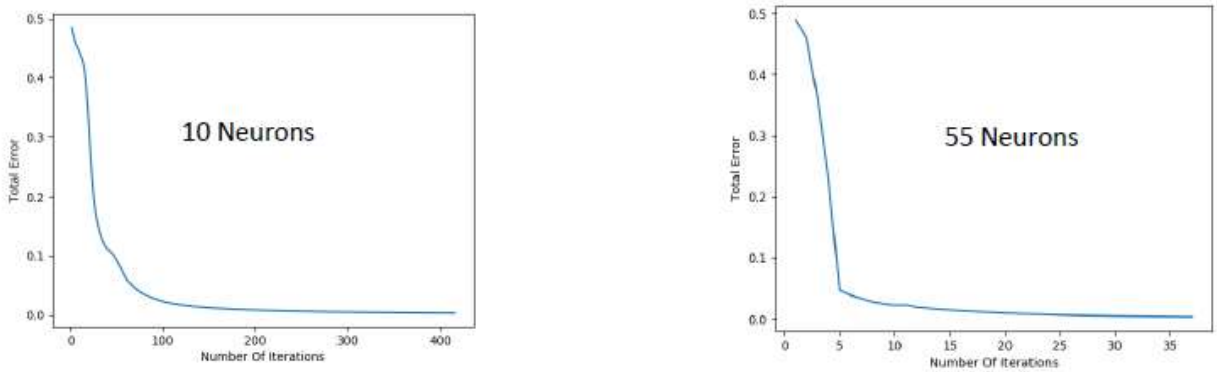


Рисунок 3.24 – Результат навчання з 10 та 55 нейронами у прихованому шарі

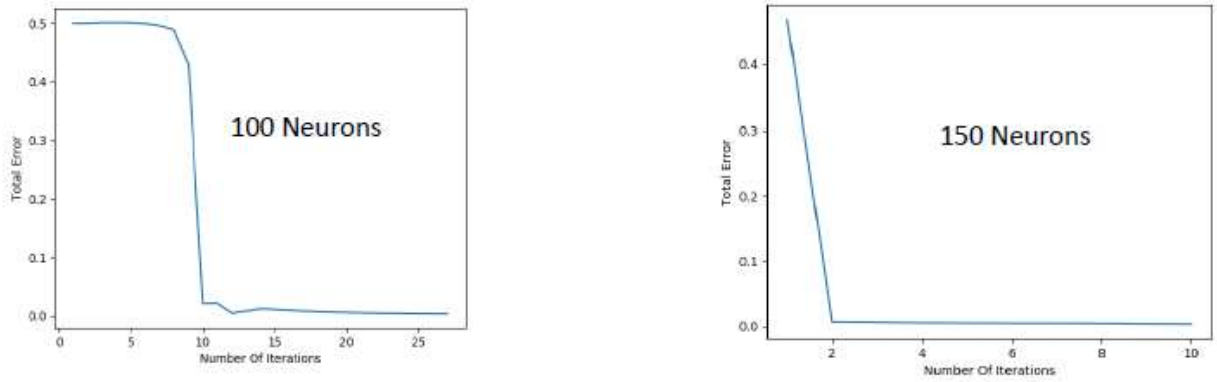


Рисунок 3.25 – Результат навчання зі 100 та 150 нейронами у прихованому шарі

Кількість ітерацій навчання становить 416 для 10 нейронів порівняно з 38 для 55 нейронів, 27 для 100 і 10 для 150 (рис. 3.24, 3.25). Ефективність розпізнавання для 150 прихованих нейронів була нижчою, ніж у випадку з 55. Це означає, що зі збільшенням кількості прихованих нейронів результати є незадовільними.

ВИСНОВКИ

У даній роботі було проведено інформаційний огляд основних принципів організації оптичного розпізнавання символів. Було проаналізовано існуючі OCR-додатки на ринку програмного забезпечення та за допомогою порівняльної характеристики продемонстровано переваги та недоліки кожного з програмних рішень, а саме ABBYY FineReader, Kofax OmniPage, Tesseract. Для реалізація інформаційної технології на основі багатошарової нейронної мережі були обрано алгоритм зворотного поширення помилки та для розроблення – мова програмування Python. Розроблено принцип роботи та архітектура інформаційної технології, а саме покроково було описано метод навчання нейронної мережі, процес попередньої обробки, який складається з бінаризації, обрізання зображення та нормалізації.

На практичному прикладі було реалізовано інформаційну технологію оптичного розпізнавання символів на основі багатошарової нейронної мережі, яка включає три основні модулі: попередньої обробки, вилучення символів та рядків, розпізнавання символів. Було проведено ряд тестів з різними значеннями параметрів для визначення ефективності навчання та розпізнавання. Дана інформаційна технологія може бути використана для оцінки основних додатків для реалізації оптичного розпізнавання символів, вивчення основних принципів організації даної технології, оцінки алгоритму зворотного поширення помилки для навчання багатошарової нейронної мережі та для оптичного розпізнавання символів.

СПИСОК ЛІТЕРАТУРИ

1. Backpropagation. URL: https://www.engati.com/glossary/back-propagation.

2. Порівняння програмного забезпечення для оптичного розпізнавання символів. URL: https://uk.wikipedia.org/wiki/Порівняння_програмного_забезпечення_для_оптичного_розпізнавання_символів.

3. ABBYY® FineReader PDF 15 User's Guide. URL: https://pdf.abbyy.com/media/1676/users_guide.pdf.

4. 7 Best OCR Software of 2022 (Free and PAID). URL: https://theecmconsultant.com/best-ocr-software/#nanonets.

5. How Does Backpropagation in a Neural Network Work? URL: https://builtin.com/machine-learning/backpropagation-neural-network.

6. Principles of training multi-layer neural network using backpropagation.URL:http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

7. Optical Character Recognition Software. URL: https://www.kofax.com/products/omnipage/ocr-developer-portal.

8. How Kofax OmniPage Puts OCR Technology to Work for Your Business. URL: https://www.kofax.com/eg/omnipage/learning-how-kofax-omnipage-puts-ocr-technology-to-work-for-your-business.

9. Tesseract OCR with Java with Examples. URL: https://www.geeksforgeeks.org/tesseract-ocr-with-java-with-examples/.

10. An Overview of the Tesseract OCR Engine. URL: https://static.googleusercontent.com/media/research.google.com/uk//pubs/archive/33418.pdf.

11. Shashank Araokar, Visual Character Recognition using Artificial Neural Networks, MGM's College of Engineering and Technology, University of Mumbai, India.

12. Neural Network Design 2nd Edtion. URL: https://hagan.okstate.edu/NNDesign.pdf.

13. Md Fazlul Kader and Kaushik Deb. Neural network-based english alphanumeric character recognition. International Journal of Computer Science. 2012. №4.

14. Backpropagation Step by Step. URL: https://hmkcode.com/ai/backpropagation-step-by-step/.

15. Python AI: How to Build a Neural Network & Make Predictions. URL: https://realpython.com/python-ai-neural-network/.

16.Shalkoff R. J. Digital image processing and computer vision / R. J. Shalkoff. – New York-Chichester-Brisbane-Toronto-Singapore: John Wiley & Sons, 1989. – 489 p.

17. Duda R. O. Pattern Classification, second ed. / R. O. Duda, P. E. Hart, D. G. Stork. – John Wiley & Sons, New York, 2001. 738 p.

ДОДАТКИ

```
#підключення бібліотек
from PIL import Image as im
import numpy as np
import matplotlib.pyplot as plt
# підключення файлів
import CropAndNormalize as pb
import LineExtraction as ln
#Алфавіт для навчання
alphabet = ['a','b','c','d','e','f','g','h','i',
'j','k','l','m','n','o','p','q','r',
's','t','u','v','w','x','y','z']
#Ініціалізація вагових коефіцієнтів та зміщення
def initWeights(wth,hght,countOfHiddenNeurons):
#ваги з віхдного до прихованого шару
Wi_h = np.random.random(size=(countOfHiddenNeurons,hght,wth))-0.5
#ваги з прихованого до вхідного шару
Wh_o = np.random.random(size=(26,countOfHiddenNeurons))-0.5
# зміщення прихованого шару
Bh = np.random.random(countOfHiddenNeurons) - 0.5
# зміщення вихідного шару
Bo = np.random.random(26) - 0.5
return Wi_h, Wh_o, Bh, Bo
#Сигмоїдна функція активації нейронів
def logActivation(sum):
out = 1 / (1 + np.exp(-sum))
return out
#Розповсюдження сигналу через мережу
def feedForward(normalized_data, Wi_h, Wh_o, Bh, Bo):
n_h = 0
```

```
[countOfHiddNeurons,hght,wth] = Wi_h.shape
#Розповсюдження сигналу з вхідного до прихованого шару
outOfHiddNeurons = []
netInForHiddNeurons = []
for hiddNeuron in range (0,countOfHiddNeurons):
for i in range (0,hght):
for j in range (0,wth):
WxP = Wi_h[hiddNeuron,i,j] * normalized_data[i,j]
n_h = n_h + WxP
n_h = n_h + Bh[hiddNeuron]
outOfHiddNeurons.append(logActivation(n_h))
netInForHiddNeurons.append(n_h)
n_h = 0
#Розповсюдження сигналу з прихованого шару до вихідного
outHiddXweightsH_O = outOfHiddNeurons * Wh_o
netInForOutNeurons = np.sum(outHiddXweightsH_O, axis= 1)
outputOfOutNeurons = []
for outputNeuron in range(0,26):
totalInForNeuron = netInForOutNeurons[outputNeuron] + Bo[outputNeuron]
outputOfOutNeurons.append(logActivation(totalInForNeuron))
return outputOfOutNeurons, outOfHiddNeurons
#Обрахунок похибки для кожного виходу вихідного нейрона
def calculateMistakeAtOutput(outputOfOutNeurons, targetOutput):
outMistake =[]
for outputNeuron in range(0,26):
outputNeuronError = outputOfOutNeurons[outputNeuron] -
targetOutput[outputNeuron]
outMistake.append(outputNeuronError)
return outMistake
#Зворотній пошук та коригування вагових коефіцієнтів
```

```
def backPropagate(Wi_h, Wh_o, Bh, Bo, normalized_data, outMistake, outputOfOutNeurons, outOfHiddNeurons, learningRate, momentum):
#збереження попередніх вагових коефіцієнтів
prevWh_o = np.array(Wh_o[:,:])
prevWi_h = np.array(Wi_h[:,:])
[countOfHiddNeurons,hght,wth] = Wi_h.shape
#Зворотній пошук від вихідного до прихованого шару
#коригування вагових коефіцієнтів
for outNeuron in range(0,26):
for hiddNeuron in range(0,countOfHiddNeurons):
adj = (learningRate * outMistake[outNeuron] * outputOfOutNeurons[outNeuron] * (1 - outputOfOutNeurons[outNeuron]) * outOfHiddNeurons[hiddNeuron])
Wh_o[outNeuron, hiddNeuron] = (momentum * Wh_o[outNeuron, hiddNeuron]) - adj
#Зворотній пошук від прихованого до вхідного шару
#коригування вагових коефіцієнтів
for hiddNeuron in range(0,countOfHiddNeurons):
delTotalMistake_hiddNeuron = 0
for outNeuron in range(0,26):
#Обчислення дельта похибки
delMistakeOutNeuron_hiddNeuron = outMistake[outNeuron] * outputOfOutNeurons[outNeuron] * (1-outputOfOutNeurons[outNeuron]) * prevWh_o[outNeuron,hiddNeuron]
delTotalMistake_hiddNeuron = delTotalMistake_hiddNeuron + delMistakeOutNeuron_hiddNeuron
for i in range (0,hght):
for j in range (0,wth):
```

```
delTotalMistake_inTohiddNeuronWeight = delTotalMistake_hiddNeuron * outOfHiddNeurons[hiddNeuron] *(1 - outOfHiddNeurons[hiddNeuron]) * normalized_data[i,j]
Wi_h[hiddNeuron,i,j] = (momentum * Wi_h[hiddNeuron,i,j]) - (learningRate * delTotalMistake_inTohiddNeuronWeight)
return Wi_h, Wh_o
# Навчання мережі
def trainNetwork(Wi_h, Wh_o, Bh, Bo,hght,wth,countOfTraining,learningRate,momentum,targetError):
count = 0
factualError = 1
mistakeList = []
y_axis = []
while factualError > targetError:
for characterToTrain in range(0,26):
targetOut = np.zeros(26)
targetOut[characterToTrain] = 1
# кількість навчальних зразків
for n in range (0,countOfTraining):
#Бінаризація та нормалізація
training_S = 'samples/%s%d.png' %(alphabet[characterToTrain],n)
char_in = im.open(training_S)
bAndW = pb.convertToBW(char_in)
reverseBW = pb.toggleOnesAndZeros(bAndW)
cutBW = pb.crop(reverseBW)
normalized_data = pb.normalize(cutBW,wth,hght)
#завершення попередньої обробки
outputOfOutNeurons, outputOfHiddNeurons = feedForward(normalized_data, Wi_h, Wh_o, Bh, Bo)
outMistake = calculateMistakeAtOutput(outputOfOutNeurons, targetOut)
```

```
Wi_h, Wh_o = backPropagate(Wi_h, Wh_o, Bh, Bo, normalized_data, outMistake, outputOfOutNeurons, outputOfHiddNeurons, learningRate,momentum)
#Обчислення середньоквадратичної помилки
factualError = 0
for x in range(0,26):
rootMeanSquare = 0.5 * outMistake[x]**2
factualError = factualError + rootMeanSquare
print('Total Error = %f' %factualError)
count = count + 1
mistakeList.append(factualError)
y_axis.append(count)
#Графічне відображення загальної помилки від ітерацій
print('Total Number of iterations %d' %count)
plt.plot(y_axis, mistakeList)
plt.ylabel('Total Error')
plt.xlabel('Number Of Iterations')
plt.show()
return (Wi_h, Wh_o, Bh, Bo)
#Розпізнавання зображення
def recognizeChar(inNormalizedData,Wi_h,Wh_o,Bh,Bo):
outputOfOutNeurons, outputOfHiddNeurons = feedForward(inNormalizedData, Wi_h, Wh_o, Bh, Bo)
#розпізнаним символом є нейрон із найвищим результатом
prefOut = np.argmax(outputOfOutNeurons)
return alphabet[prefOut]
```