МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики та моделювання складних систем

Допущено до захисту
Завідувач кафедри ПМ та МСС
_____ Коплик І.В.
(підпис)
«___» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
спеціальність 113 «Прикладна математика»
освітньо-професійна програма «Прикладна математика»

тема роботи: «**Побудова нейронної мережі для розпізнавання рукописних цифр**»

Виконавець
студент факультету ЕлІТ
Грибініченко Ярослав Андрійович _____
(підпис)

Науковий керівник
док. фіз.-мат. наук, професор
Лисенко Олександр Володимирович _____
(підпис)

Суми – 2023

РЕФЕРАТ

Кваліфікаційна робота: 83 с., 57 рисунків, 14 джерел.

Мета роботи: Метою дипломної роботи є вибір архітектури та побідова нейронної мережі, що дозволяє з мінімальними обчислювальними витратами розпізнавати області для розпізнавання рукописних цифр.

Об’єкт дослідження: методи та моделі нейронних мереж для класифікації рукописних цифр.

Предмет дослідження: порівняння ефективності різних архітектур нейромереж, зокрема багатошарового перцептрона і згорткової нейромережі, з метою визначення найоптимальнішої моделі для класифікації рукописних цифр

Методи навчання: зворотного поширення помилки, стохастичного градієнтного спуску, статистичні методи.

У роботі було проведено аналіз та порівняння ефективності різних моделей нейронних мереж у задачі класифікації рукописних цифр з використанням бази даних MNIST. Особлива увага була зосереджена на двох архітектурах нейромереж: багатошаровому перцептроні та згортковій нейромережі. Під час тренування обох нейромереж на тренувальному наборі даних було оцінено їхню точність на тестовому наборі. Виявлено, що згорткова нейромережа досягає середньої точності розпізнавання значення швидше, в порівнянні з багатошаровим перцептроном. Це означає, що згорткова нейромережа здатна ефективно виявляти та розпізнавати важливі ознаки у зображеннях рукописних цифр. Зазначимо, що згорткова нейромережа вимагає більш тривалого часу для навчання порівняно з багатошаровим перцептроном. Однак, враховуючи кращу точність розпізнавання, цей компроміс є виправданим. Запропоновано використовувати згорткові нейромережі для розпізнавання рукописних цифр, оскільки вони здатні виявляти ієрархічні структури та просторові залежності в даних, що сприяє покращенню точності класифікації. Такий підхід може бути корисним для широкого спектру завдань, пов'язаних з розпізнаванням образів та класифікацією зображень.

Ключові слова: ЗГОРТКОВА НЕЙРОНА МЕРЕЖА, РЕКУРЕНТНА НЕЙРОНА МЕРЕЖА, БАГАТОШАРОВИЙ ПАРЦЕПТРОН, ПАРЦЕПТРОН.

ЗМІСТ

Вступ

Машинне навчання, розпізнавання об'єктів та комп'ютерний зір (Computer Vision) є галузями штучного інтелекту, які займаються аналізом та обробкою візуальної інформації. В цьому тексті ми розглянемо основні концепції та методи, що використовуються у машинному навчанні для розпізнавання об'єктів та комп'ютерного зору.

Машинне навчання - це підгалузь штучного інтелекту, яка досліджує алгоритми та моделі, які можуть автоматично навчатися та вдосконалюватися на основі даних. Для розпізнавання об'єктів та комп'ютерного зору, машинне навчання використовується для навчання моделей на великій кількості зображень та визначення патернів, які допомагають ідентифікувати об'єкти або розуміти зображення.

У машинному навчанні для розпізнавання об'єктів та комп'ютерного зору використовуються різні методи та алгоритми. Деякі з них включають:

1. Нейронні мережі – зокрема згорткові нейронні мережі (Convolutional Neural Networks, CNN), є потужними архітектурами для розпізнавання об'єктів та комп'ютерного зору. Вони здатні виявляти та класифікувати об'єкти на зображеннях, використовуючи внутрішні зв'язки та шари згортки для витягування важливих ознак.

2. Згортка (Convolution) та пулінг (Pooling), які є операціями, які використовуються в згорткових нейронних мережах для обробки зображень. Згортка використовує фільтри для витягування локальних ознак зображення, таких як границі або кольорові плями. Пулінг зменшує розмір отриманого зображення, зберігаючи важливу інформацію.

3. Класифікація та локалізація – у розпізнаванні об'єктів важливим завданням є не тільки класифікація об'єктів, але і їхнє локалізація на зображенні. Це включає визначення положення та розміру об'єктів. Для цього використовуються методи, такі як архітектури R-CNN (Region-based

Convolutional Neural Networks), SSD (Single Shot MultiBox Detector) та YOLO (You Only Look Once).

Проблема обробки зображень включає в себе такі, що стали вже майже класичними, завдання, як сегментація, фільтрація перешкод і виділення зображень з фону, визначення меж об'єктів, розпізнавання образів. Центральне місце серед названих понять займає задача розпізнавання образів. Рішення її технічними засобами може бути здійснено шляхом моделювання операцій, виконуваних живими організмами в процесі комунікації і сприйняття навколишнього світу. Ці здібності досить добре вивчені на різних тварин (кити, дельфіни, кажани, деякі примати). Однак найбільш природно покласти в основу моделі розпізнавання здатності людини і його реакції на навколишню дійсність.

Розпізнавання образів як науковий напрям включає в себе велику кількість різних дисциплін і використовує методи, характерні для кожної з них. Поряд з всюдисущої інформатикою серед них можна виділити прикладну фізику, необхідну при розробці ефективних датчиків, а також різні розділи математики, що лежать в основі методів обробки даних. Розпізнавання образів є сукупність методів і засобів, що дозволяють, щонайменше, досягти, а якщо можливо, то й перевершити природні засоби сприйняття і аналізу навколишнього світу живими істотами.

Метою даної дипломної роботи є вибір архітектури нейронної мережі, що дозволяє з мінімальними обчислювальними витратами розпізнавати області для розпізнавання рукописних цифр, призначеного для користувача графічного інтерфейсу ПК.

Розділ 1
Нейроні мережі для розпізнавання рукописних цифр (огляд літератури)

1.1 Опис методів машинного навчання систем обробки зображень

Побудова моделей розпізнавання зображень з використанням нейронних мереж є складним завданням, яке вимагає ретельного планування, налагодження та оптимізації. У цьому тексті ми розглянемо основні принципи побудови таких моделей та важливі кроки, які допомагають досягти високої точності та ефективності.

Збір та підготовка даних:

Першим кроком у побудові моделі розпізнавання зображень є збір та підготовка даних. Це включає в себе створення набору даних, який містить зображення різних класів, а також мітки, що вказують на правильну класифікацію зображень. Для забезпечення точності моделі важливо мати достатньо рознообразний і збалансований набір даних.

Далі проводиться попередня обробка даних, така як зменшення розміру зображень, нормалізація значень пікселів, вирівнювання розмірів зображень тощо. Це допомагає знизити складність моделі та покращити швидкість навчання.

Вибір архітектури нейронної мережі:

Архітектура нейронної мережі визначає структуру та спосіб взаємодії її шарів. У випадку розпізнавання зображень, поширеним вибором є згорткові нейронні мережі (Convolutional Neural Networks, CNN). CNN мають спеціалізовані шари згортки, пулінгу та повнозв'язаних шарів, що дозволяють ефективно виявляти та аналізувати особливості зображень.

Оптимальний вибір архітектури залежить від конкретної задачі розпізнавання та доступних ресурсів. Зазвичай використовуються вже

налаштовані та популярні архітектури, такі як VGG, ResNet, Inception, або можна створити власну архітектуру згідно з вимогами задачі.

Навчання моделі:

Навчання моделі полягає у ваговому налаштуванні нейронної мережі з використанням набору даних. Зазвичай, навчання проводиться методом зворотнього поширення помилки (backpropagation), де ваги мережі оновлюються з урахуванням помилки прогнозування.

У процесі навчання важливо вибрати відповідні функції втрати (loss functions) та алгоритми оптимізації (optimization algorithms). Функція втрати визначає, наскільки точним є прогноз моделі порівняно зі звичайними мітками. Алгоритм оптимізації відповідає за оновлення ваг моделі, щоб мінімізувати функцію втрати.

Валідація та тестування моделі:

Після навчання моделі важливо перевірити її ефективність та загальну точність. Для цього використовуються окремі набори даних для валідації та тестування. Валідація допомагає підбирати оптимальні параметри моделі та контролювати перенавчання, а тестування оцінює загальну точність та продуктивність моделі.

Налагодження та оптимізація:

У процесі налагодження моделі важливо виявити й усунути можливі проблеми, такі як недостатня точність, перенавчання або недоношування (underfitting). Це може включати зміну гіперпараметрів моделі, додавання регуляризації, використання аугментації даних або виправлення помилок у даних.

Оптимізація моделі також включає зусилля щодо зменшення обчислювального завантаження та розміру моделі, особливо для застосувань у вбудованих системах чи мобільних пристроях.

Багатошаровий персептрон (МСП) використано в якості моделі для машинного навчання. Цей вид НС має дві основні переваги - простота в застосуванні і забезпечення необхідних узагальнюючих властивостей.

Початкове значення тришарового персептрона (рис. 1.1) обчислюється за виразом:

$$y = F_3(\sum_{j=1}^{n} v_j h_j - b_3) \quad (1.1.1)$$

де n - кількість нейронів в прихованому шарі;

v_j - вага синапсу нейрона j прихованого шару до вихідного нейрона;

h_j - початкове значення нейрона j прихованого шару;

b_3 - поріг вихідного нейрона;

F_3 - функція активації вихідного нейрона.

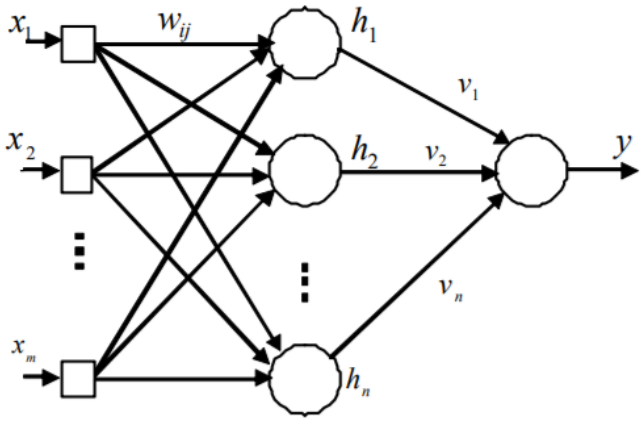

Рисунок 1.1 – Структура тришарового персептрона

Початкове значення нейрона j прихованого шару визначається за формулою:

$$h_j = F_2(\sum_{i=1}^{m} w_{ij} x_i - b_{2j}) \quad (1.1.2)$$

де w_{ij} - вага від і-го вхідного нейрона до j-му нейрону схованого шару; x_i - вхідні значення; b_{2j}- поріг j-го нейрона прихованого шару. Сигмоїдну функцію активації $F(x) = \frac{1}{1+e^x}(1.1.3)$ використано для нейронів прихованого шару і вихідного нейрона.

Ефективне значення помилка для навчальної ітерації t розраховується за формулою:

$$E^p(t) = \frac{1}{2}(y^p(t) - d^p(t))^2, \quad (1.1.4)$$

де для навчання вектора p, $y^p(t)$ - обчислене початкове значення багатошарового персептрону на ітерації t; $d^p(t)$ - бажане початкове значення багатошарового персептрону на навчальній ітерації t.

Під час навчання узагальнена помилка навчання для всіх навчальних векторів обчислюється за формулою:

$$E(t) = \sum_{p=1}^{P} E^p(t), \quad (1.1.5)$$

Помилка вихідного нейрона для навчання вектора p розраховується за формулою:

$$\gamma_3^p(t) = y^p(t) - d^p(t) \quad (1.1.6)$$

а помилка нейрона і прихованого шару за формулою:

$$\gamma_i^p(t) = \sum_{j=1}^{n} \gamma_3^p(t) \cdot v_i(t) \cdot h_j^p(t) \cdot (1 - h_j^p(t)), \quad (1.1.7)$$

Для навчання багатошарового персептрона використаний алгоритм зворотного поширення помилки, що складається з наступних кроків [2]:

1. Поставити крок навчання α (0 <α <1) і мінімальну середньоквадратичну помилку навчання багатошарового персептрона E_{min}, яку необхідно досягти в процесі навчання.

2. Ініціювати ваги і пороги нейронів випадковими величинами з діапазону (-0.5 ... 0.5).

3. Для навчального вектора p обчислити вихідне значення багатошарового персептрона y.

4. Обчислити помилку вихідного нейрона.

5. Модифікувати ваги і пороги вихідного нейрона за формулами:

$$v_j^p(t+1) = v_j^p(t) - \alpha \cdot \gamma_3^p(t) \cdot h_j^p(t) \cdot y^p(t) \cdot \left(1 - y^p(t)\right), \quad (1.1.8)$$

$$b_3^p(t+1) = b_3^p(t) + \alpha \cdot \gamma_3^p(t) \cdot y^p(t) \cdot (1 - y^p(t)). \quad (1.1.9)$$

6. Обчислити помилку нейронів прихованого шару $y_i^p(t)$.

7. Модифікувати ваги і пороги нейронів прихованого шару відповідно до формул:

$$w_{ij}^p(t+1) = w_{ij}^p(t) - \alpha \cdot \gamma_j^p(t) \cdot h_j^p(t) \cdot \left(1 - h_j^p(t)\right) \cdot x_i^p, \quad (1.1.10)$$

$$b_{2j}^p(t+1) = b_{2j}^p(t) + \alpha \cdot \gamma_j^p(t) \cdot h_j^p(t) \cdot \left(1 - h_j^p(t)\right). \quad (1.1.11)$$

8. Розрахувати середньоквадратичну помилку для тренувальної ітерації t.

9. Повторити кроки 3-8 для всіх векторів тренувальної вибірки.

10. Розрахувати узагальнену середньоквадратичну помилку $E(t)$ багатошарового персептрона.

11. Якщо $E(t)$ є все ще більше бажаної мінімальної помилки E_{min}, необхідно почати знову з кроку 3 або закінчити навчання в іншому випадку.

1.2 Моделі дослідження нейронних мереж SRCNN, SRResNet

Моделі дослідження нейронних мереж для зображень високої роздільної здатності (High-Resolution Image Super-Resolution Networks) є ключовими інструментами у сфері обробки та відновлення зображень. Серед таких моделей виділяються дві популярні архітектури: SRCNN (Super-Resolution Convolutional Neural Network) та SRResNet (Super-Resolution Residual Network).

SRCNN є однією з перших нейронних мереж, запропонованих для задачі збільшення роздільної здатності зображень. Вона складається з трьох основних шарів: згорткового шару (convolutional layer), нелінійної функції активації (activation function) та піксельного шару (pixel-shuffling layer). Згортковий шар відповідає за витягнення характеристик зображення, активаційна функція

допомагає зберігати нелінійні залежності, а піксельний шар відновлює зображення високої роздільної здатності шляхом зміни порядку пікселів.

SRCNN використовується для відновлення зображень, які мають низьку роздільну здатність, наприклад, зображення, що були стиснені або пошкоджені. Завдяки своїй простоті та ефективності, SRCNN став основою для подальшого розвитку моделей збільшення роздільної здатності.

SRResNet є ще більш потужною архітектурою, яка використовує глибоку нейронну мережу з використанням блоків з короткими з'єднаннями (residual blocks). Ця модель базується на принципі глибокого навчання і здатна досягти вражаючих результатів у відновленні високої роздільної здатності зображень.

SRResNet використовується для збільшення роздільної здатності зображень у медіа, медичній діагностиці, аналізі зображень та багатьох інших галузях. Вона має глибоку структуру з резидуальними блоками, що дозволяють впроваджувати глибоку резидуальну мережу, що надає здатність відновлювати зображення з високою деталізацією та чіткістю.

Обидві архітектури, SRCNN та SRResNet, відіграють важливу роль у сфері відновлення зображень високої роздільної здатності за допомогою нейронних мереж. Вони забезпечують значне поліпшення якості зображень та здатні забезпечувати багатоцільову функціональність, включаючи підвищення чіткості, видалення шуму та відновлення деталей. Застосування цих моделей відкриває нові можливості у багатьох галузях, де якість зображень є ключовим чинником, і допомагає вдосконалити інструменти для обробки та аналізу великих обсягів високоякісних зображень.

Одними із перших і базових моделей які досягли певного успіху в задачі збільшення роздільної здатності стали моделі SRCNN та її більш швидка модифікація FSRCNN (рисунок 1.2). В їх основі лежить отримання карт ознак з вхідного зображення і подальше підвищення дискретизації з цих карт до отримання зображення більшої роздільної здатності.

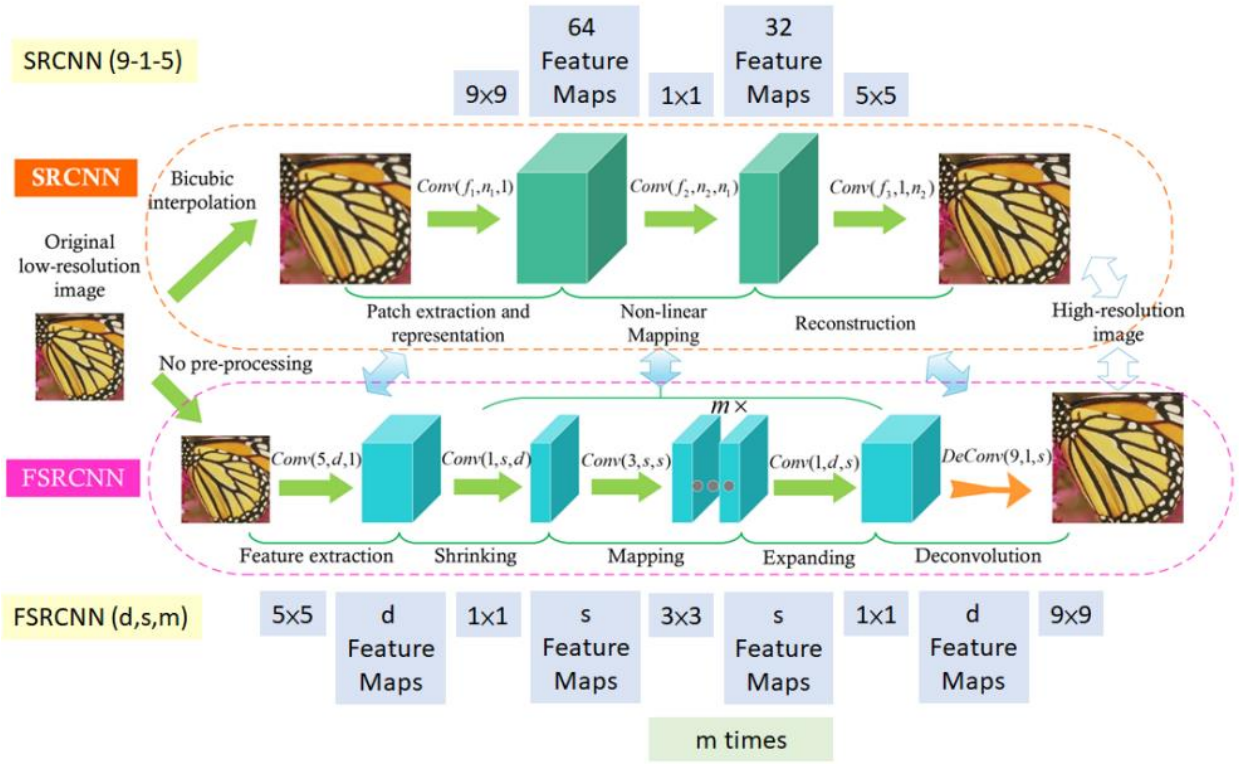

Рисунок 1.2 – Порівняння архітектур SRCNN і FSRCNN

Отже базовим підходом для застосування нейронних мереж в задачі SISR стали архітектури типу «кодувальник - декодувальник», в яких зображення спочатку стискається до певної карти ознак, і потім розтискається до збільшеного зображення. Вони показали кращу якість генерації, проте були набагато складніші в реалізації і обчислені, тому дослідники продовжили шукати альтернативні архітектури і підходи.

1.3 Згорткові нейронні мережі для вирішення задачі розпізнавання рукописних цифр

Для завдання масштабування зображень дослідники використовують різні види нейронних мереж. Найбільш ефективними показали себе такі моделі: згорткова нейронна мережа (Convolutional neural network, CNN) та генеративнозбірна мережа (Generative adversarial network, GAN) [5].

Згорткові нейронні мережі - це штучні нейронні мережі, що складаються з великої кількості перцептронів. Стандартна згорткова нейронна мережа складається з ланцюжка згорткових шарів, що використовуються для автоматичного виділення ознак, та кількох повнозв'язкових шарів, які використовуються для вирішення поставленого завдання.

Однією з найвідоміших моделей, розроблених для вирішення завдання масштабування зображень, є розроблена в 2014 нейромережа SRCNN (Super-Resolution Convolutional Neural Network) [5].

На вхід SRCNN подається зображення збільшене за допомогою методів інтерполяції, яке потім проходить процес поліпшення якості. Подібний метод застосовується у розробленому компанією Google методі RAISR. Існує безліч модифікацій даної моделі, однією з найшвидших за швидкістю та ефективних за якістю є FSRCNN (Fast Super-Resolution Convolutional Neural Network) [5], яка, на відміну від попередників, не використовує попередньої обробки зображення за допомогою інтерполяції.

Варіанти шарів нейронної мережі:

Повнозв'язковий шар:

Найпростіший і широко застосовуваний шар нейронної мережі – повнозв'язковий. Кожен нейрон у цьому шарі – персептрон із нелінійною функцією активації. Як функцію активації прийнято використовувати або логістичну функцію $f(x) = 1/(1 + e^{-x})$, або гіперболічний тангенс $f(x) = A\tanh(Bx)$. Обидва варіанти реалізовані в системі та доступні.

На вхід кожному з нейронів повнозв'язкового шару подаються виходи всіх нейронів попереднього шару. Нейрон вважає суму своїх входів, помножених на ваги, додає поріг і отримане значення пропускає через функцію активації. Результат подається на вихід нейрона.

$$Output = f\left(\sum_{i\,\in inputs} x_i \cdot w_i + bias\right), \qquad (1.3.1)$$

де x_i- вхід нейрона, w_i - вага, зіставлений цьому входу, $bias$ - адитивний поріг, що навчається.

Робота мережі відбувається «зліва направо», від вхідного шару до вихідного – так переміщається сигнал.

Для навчання повного шару використовується алгоритм зворотного поширення помилки.

Помилка поширюється у зворотний бік, від вихідного сигналу до вхідного. Спочатку кожному шарі вважається помилка для попереднього, потім коригуються ваги, які ведуть до нейронів шару від попереднього. Формула корекції ваги конкретного зв'язку виходить такою:

$$\Delta w = \eta \cdot \delta \cdot output \qquad (1.3.2)$$

Де w - вага зв'язку, η- швидкість навчання, δ- сума помилок, порахована даного нейрона шару, $output$ - вихід нейрона, що з нейроном поточного шару.

Шар згортки:

Шар згортки – спосіб зменшити кількість тренованих параметрів у шарі нейронної мережі. Шар складається з декількох карт, що являють собою матрицю нейронів, пов'язаних з попереднім шаром. Кожен із нейронів з'єднаний із заданою прямокутною областю. При створенні шару потрібно задати розміри прямокутної області, зсув області для сусідніх нейронів (зазвичай області розташовуються одна на одну), розміри карт, а також матрицю зв'язків карт, яка показує, які карти поточного шару пов'язані з картами попереднього.

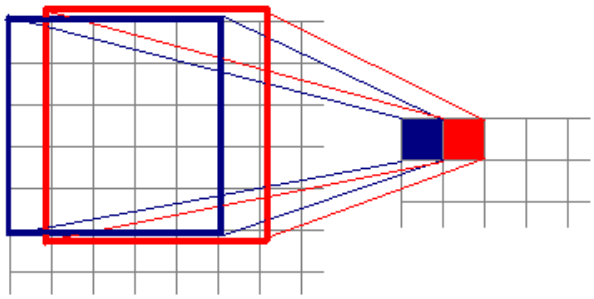

Рисунок 1.3 – Шар згортки

На відміну від повнозв'язного шару у тому, що з нейронів кожної карти загальні ваги. Виходить, якщо кожен нейрон з'єднаний з областю 5x5

попереднього шару, то у всієї карти всього 26 ваг, що навчаються (25 ваг - область і 1 вага - поріг). У різних картах набори ваг, що навчаються, різні і ніяк не залежать один від одного.

Для того, щоб не зберігати надмірну інформацію, система зберігає індекс ваги замість самої ваги. Прямий прохід у шарі згортки відбувається абсолютно так само, як і в повнозв'язному шарі – від вхідного шару до вихідного. При цьому потрібно враховувати, що ваги нейронів загальні.

У навчанні використовується модифікація алгоритму зворотного поширення помилки. Оскільки ваги загальні, те й корекцію потрібно проводити, використовуючи дані про помилки всіх нейронів, що користуються вагою. Для цього спочатку вважаємо суму помилок на всіх нейронах, а потім, використовуючи цю суму як оновлюємо значення ваги. Як у формулі модифікації ваги алгоритму зворотного поширення ми беремо всі вихідні значення нейронів, до яких ведуть зв'язки, що мають цю загальну вагу.$\delta \cdot output$

Згорткові шари нейронної мережі відповідають за вилучення особливостей вхідного зображення та шарів, що передують їм.

Шар субдискретизації:

Шар субдискретизації за своєю структурою нагадує шар згортки. У ньому так само, як і в шарі згортки, кожен нейрон карти з'єднаний із прямокутною областю на попередньому. Нейрони мають нелінійну функцію активації – логістичну чи гіперболічний тангенс. Тільки, на відміну шару згортки, області сусідніх нейронів не перекриваються. У шарі згортки, кожному нейрону області веде свій зв'язок, що має вагу. У шарі субдискретизації відбувається кожен нейрон усереднює виходи нейронів області, до якої він приєднаний. Виходить, що кожна карта має всього по дві ваги, що настроюються: мультиплікативний (вага усереднення нейронів) і адитивний (поріг).

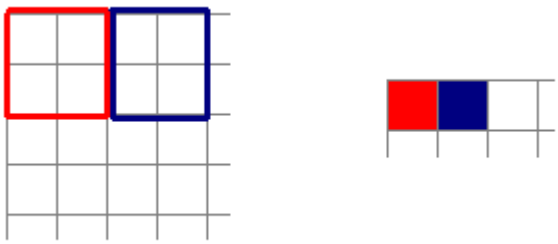

Рисунок 1.4 – Шар субдискретизації

Замість функції усереднення нейронів області може бути використана функція взяття максимуму або взяття значення медіани. Обидва ці варіанти також впроваджені в систему і можуть бути використані.

Найчастіше, при побудові мережі шари згортки і субдискретизації чергуються, щоб забезпечити витяг ознак при досить малій кількості параметрів, що тренуються.

Багатошаровий персептрон:

Багатошаровий персептрон є повнозв'язковою мережею прямого поширення. Це означає, що нейронні об'єднані у шари та сигнал від вхідного шару до вихідного шару, причому кожен нейрон одного шару пов'язаний з усіма нейронами попереднього шару. Нейрони одного шару не пов'язані один з одним. На рис. 1.5 представлена схема 2-хшарового персептрону.

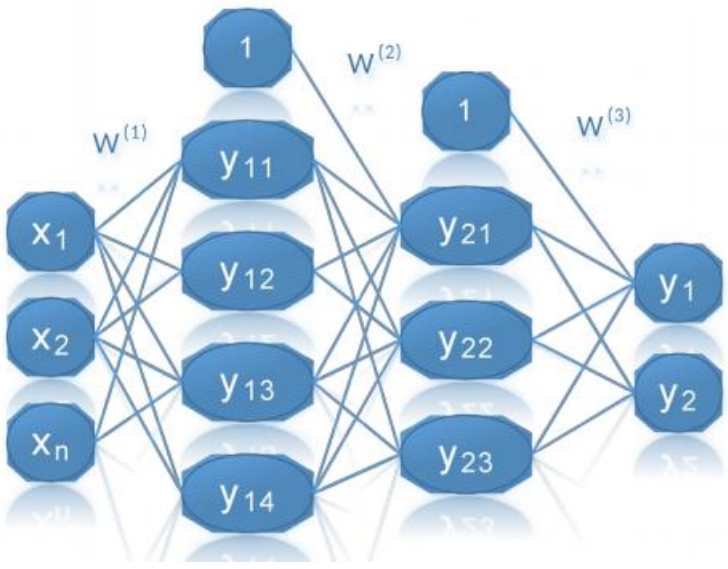

Рисунок 1.5 – Схема 2-шарового персептрона, де x - вектор вхідних сигналів, w(i) - матриця вагових коефіцієнтів і-го шару, yij - активаційне значення нейрона, yj - активаційне значення вихідного нейрона.

Як активаційну функцію в багатошаровому персептроні часто використовуються такі функції, як сигмоїда, гіперболічний тангенс. У роботі використовується функція ReLU (rectified linear unit) f(x)=max(0,x), реалізує пороговий перехід у нулі. Перевагою цієї функції і те, що з обчислення функції не потрібно трудомістких операцій і за великих значеннях аргументу немає «насичення» (градієнт дуже близький до нуля).

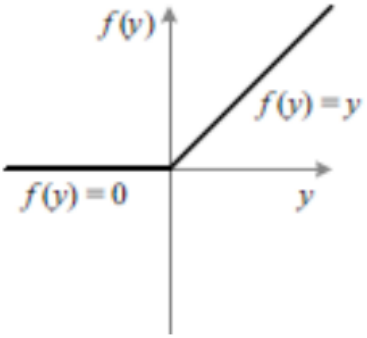

Рисунок 1.6 – Графік функції ReLU

Також як активаційну функцію нейронів вихідного шару для оцінки ймовірності класифікації використовується функція softmax.

Згорткові нейронні мережі:

Пакувальні нейронні мережі були запропоновані Яном Лекуном як спеціалізована архітектура для розпізнавання зображень [11]. Так як вхідне значення є зображенням, воно представляється у вигляді двомірної матриці, що дозволяє зберегти інформацію про сусідні точки. У багатошаровому перспептроні зображення представляється як одного вектора. Згорткові мережі характеризуюся наявністю згорткових і субдискретизуючих шарів, що чергуються. Згортковий шар є набір фільтрів (згорток) - матриць нейронів малого розміру. Схему застосування згортки можна побачити на рис. 1.7.

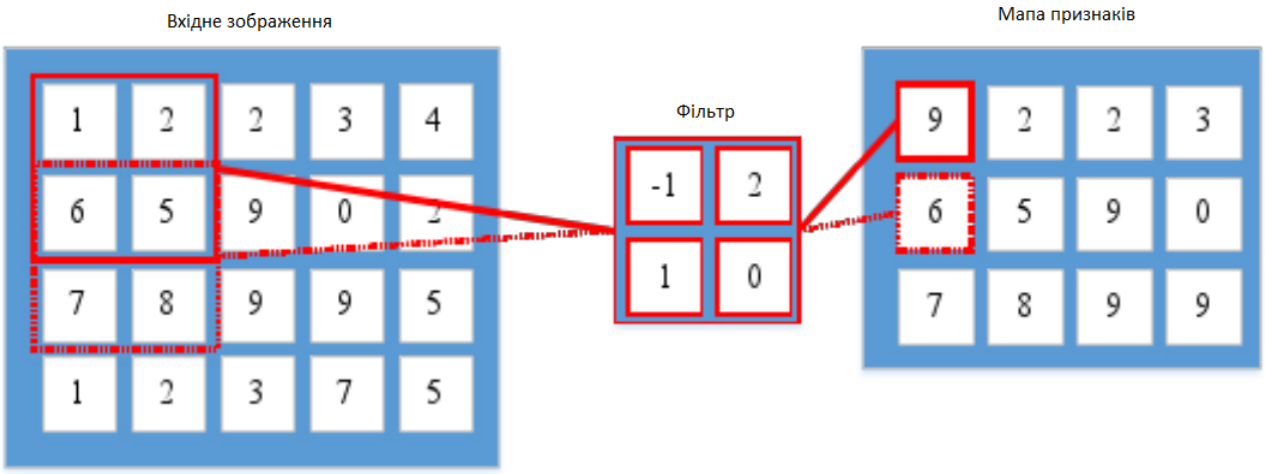

Рисунок 1.7 – Операція згортки

Виходом згорткового шару є карти ознак, елементи яких виходять в результаті скалярного добутку фільтра з ділянкою вхідного шару розміром із сам фільтр. Кожен фільтр видає одну карту ознак. У ході навчання параметри фільтрів налаштовуються так, що двомірні карти ознак, що отримуються, характеризують наявність у вхідному зображенні з певної ознаки. Із застосуванням до карт ознак додаткових згорткових шарів виявляються ознаки вищого порядку.

Далі, до карт ознак застосовуються шари субдискретизації (підвиборки). Шари субдискретизації застосовуються для просторового зменшення вхідного вектора, що веде до інваріантності щодо масштабу та обертання. Останній повнозв'язковий шар поєднує всі карти ознак та видає вектор виходів нейронної мережі, кожне значення якої характеризує близькість вхідного зображення до того чи іншого класу. На рис. 1.8 зображено схему згорткової нейронної мережі.

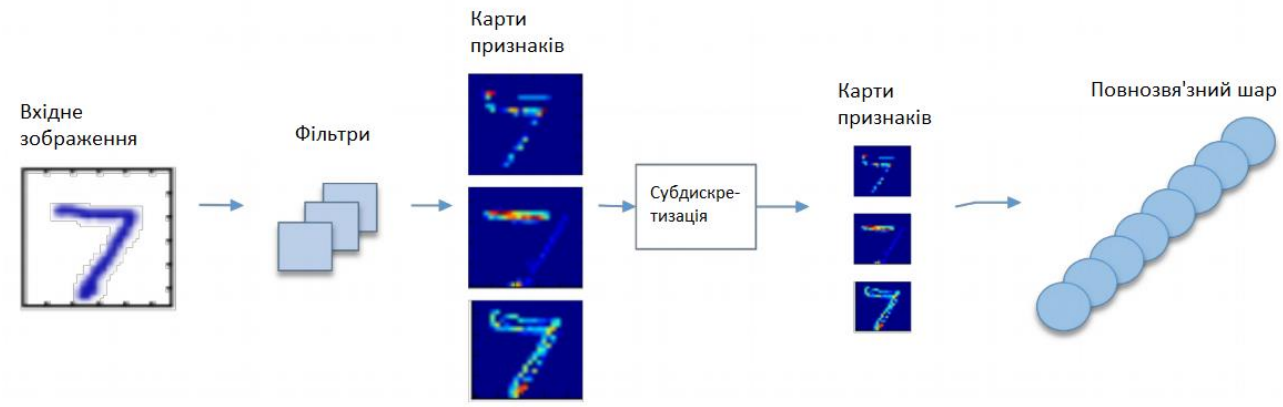

Рисунок 1.8 – Схема шарів згорткової нейронної мережі

Гіперапараметрами в мережі згортки будуть:

– розмір фільтра (stride) у згортковому шарі;

- крок, з яким переміщується фільтр у вхідному зображенні;

– крок усереднення у шарі субдискретизації;

- число згорткових шарів;

Згорткові нейронні мережі – це алгоритми глибокого навчання, які зазвичай використовуються для розпізнавання зображень та обробки природної мови. Їх архітектура подібна до організації нейронів зорової кори людини, що дозволяє їм дуже добре вловлювати закономірності з вхідних зображень.

Причина, за якою CNN краще при обробці зображень, пов'язана з кількістю параметрів, що використовуються. Якщо ми розглянемо вхідні дані з розмірами 1920x1080, у нас буде більше 2 мільйонів пікселів, зазвичай з трьома кольоровими каналами в кожному, в більшості глибоких нейронних мереж це вимагатиме величезного обсягу обробки, і результат буде не таким точним, оскільки дані будуть зведені в одновимірний масив , що призведе до втрати інформації з вихідного зображення.

Згорткові мережі суттєво зменшують вхідні зображення до меншої матриці, зберігаючи при цьому вихідні функції, щоб гарантувати найкращий кінцевий результат. Для цього вхідні дані подаються на Згортковий шар і Об'єднуючі шари і, нарешті, на Шар класифікації, також відомий як Повністю підключений. Шар. На відміну від більшості інших нейронних мереж, всі нейрони CNN мають однакову вагу і, як правило, не всі пов'язані між шарами.

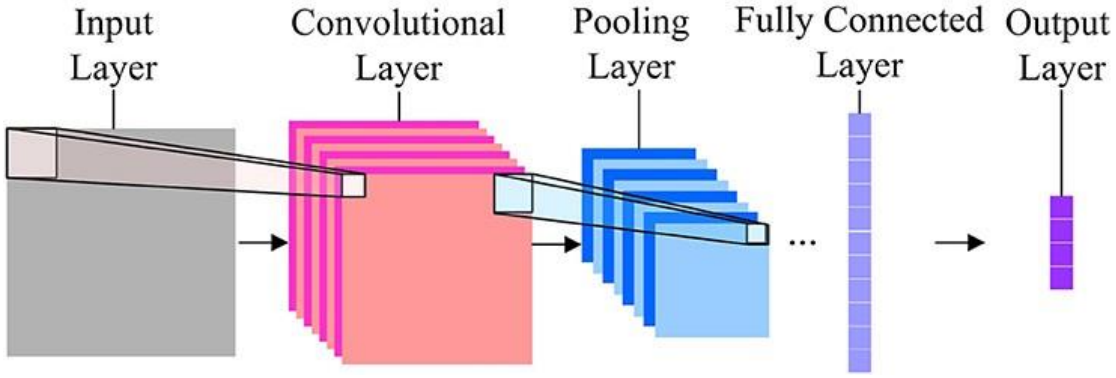

Рисунок 1.9 – Структура мережі

Щоб зменшити розмір вхідних даних, до зображення застосовуються фільтри, які називаються ядрами, зазвичай вони мають розмір 3x3 або 5x5 і отримують високорівневі функції, такі як краї, або застосовують перетворення, такі як розмиття. Ці операції можуть призвести до того, що згорнутий об'єкт збільшуватиме або зберігатиме ті самі розміри за допомогою Того ж заповнення або зменшуватиме з використанням Дійсного заповнення. Ядро переміщається по всьому зображенню за шаблоном, показаним нижче:

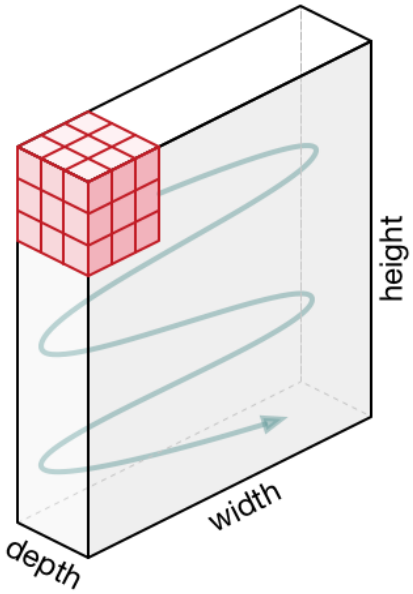

Рисунок 1.10 – Переміщення по шаблону

Згорткові нейронні мережі можуть використовувати набір фільтрів для виділення функцій із вхідного зображення, які дозволяють класифікувати.

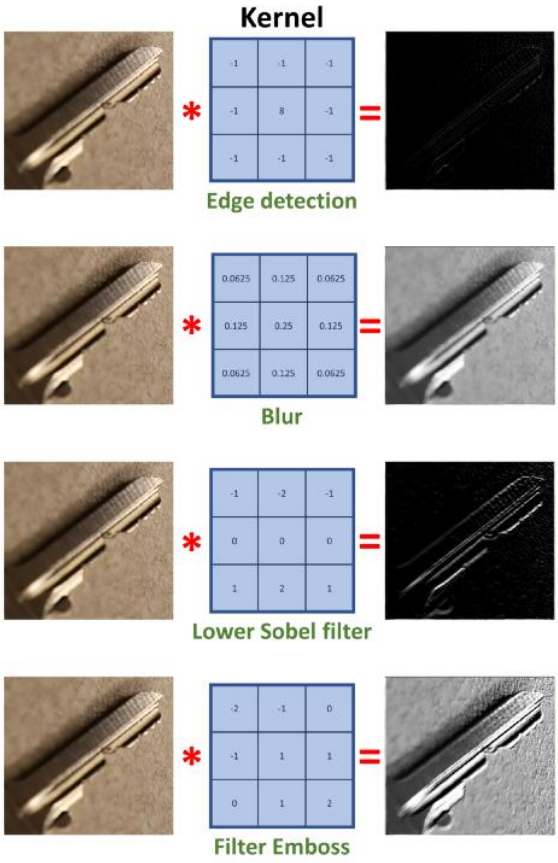

Рисунок 1.11 – Використання фільтрів

Перші шари охоплюють дрібніші і простіші функції, такі як виявлення країв. Коли ви комбінуєте кілька шарів та фільтрів, мережа може почати виявляти складніші об'єкти.

Кожне ядро виявляє певну функцію у вхідному зображенні. Карта ознак створюється нейронами з використанням одного і того ж ядра та змінюється під час навчання моделі з функцією мінімізації втрат. Ці операції передаються у функцію ReLU, де шари та карти об'єктів зливаються, а при зворотному поширенні значення у матрицях фільтрів оновлюються.

Шар пулу використовується для зменшення розмірів вихідних даних згорткового шару, зменшення обробки, необхідної для інтерпретації даних, він також корисний для отримання позиційно-інваріантних домінуючих функцій при збереженні процесу навчання моделі. Цей шар отримує вихідне зображення з попереднього шару та повертає максимальне сукупне значення або максимальне об'єднання в пул. Він також може мінімізувати або перетворити

його на іншу функцію, відому як середнє об'єднання в пул.Це знижує ймовірність перенавчання за рахунок визначення ймовірності найвпливовіших функцій.

1.4 Концептуальна модель реалізована на основі діаграми концептуальних класів

Розкладання на гармоніки, коли вихідні дані представлені дискретним набором точок {x} принципово неоднозначним [10]. Функції

$$f(t) = A \cdot \cos\left(\frac{2\pi tm}{T} + \varphi\right), \quad (1.4.1)$$

$$2f''(t) = A \cdot \cos\left(\frac{2\pi t(N-m)}{T} - \varphi\right), \quad (1.4.2)$$

$$f'(t) + f''(t) = \frac{A}{2} \cdot \cos\left(\frac{2\pi tm}{T} + \varphi\right) + \frac{A}{2} \cos\left(\frac{2\pi t(N-m)}{T} - \varphi\right), \quad (1.4.3)$$

$$f'(t) + f''(t) = A \cdot \cos\left(\frac{\pi tN}{T}\right) \cos\left(\frac{2\pi tm}{T} - \frac{\pi tN}{T} + \varphi\right) \quad (1.4.4)$$

дають після дискретизації одні й самі вихідні дані й самі результати ДПФ.

Дзеркальний ефект у переважній більшості випадків спотворює вихідну картину, оскільки насправді дуже рідко на вхід подається сума двох гармонійних сигналів f'(t) + f''(t) саме з таким співвідношенням частот: $m\vartheta$ і $(N-m)\vartheta$. У результаті вихідний спектр спотворюється, начебто відбиваючись у дзеркалі:

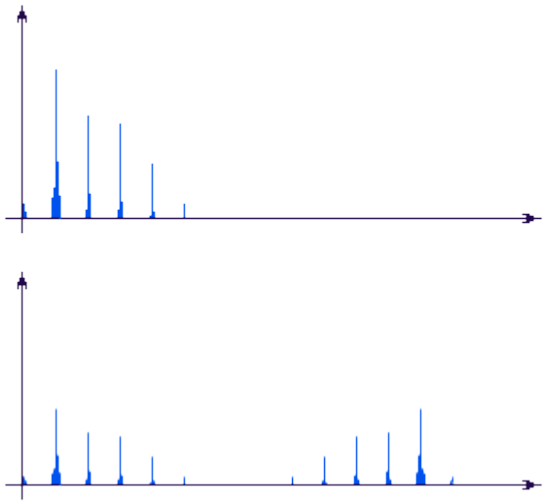

Рисунок 1.12 – Очікуваний спектр сигналу

На цьому рисунку зверху показаний очікуваний спектр сигналу, отриманий за допомогою безперервного перетворення Фур'є, а знизу - отриманий на комп'ютері дискретного перетворення Фур'є. Нижній спектр спотворений дзеркальним ефектом.

Нехай ми знайшли ненульовий коефіцієнт X_m. Приймаємо гіпотезу, що цей коефіцієнт відповідає вихідному гармонійному коливанню. Відновимо його амплітуду, фазу та частоту.

$$X_m = \left(\frac{A}{2}\right) N_e^{j\varphi}, \tag{1.4.5}$$

$$f(t) = A \cdot \cos\left(\frac{2\pi t m}{T} + \varphi\right) \tag{1.4.6}$$

Частота відновлюється найпростіше: $\frac{m}{T}$, де m - індекс знайденого ненульового елемента X_m. Амплітуда та фаза відновлюються за формулою:

$$A = \frac{2}{N}\sqrt{Re_m^2 + Im_m^2}\,, \tag{1.4.7}$$

$$\varphi = arctg\left(\frac{Im_k}{Re_k}\right) \tag{1.4.8}$$

Відома властивість перетворень Фур'є: вони лінійні. Тобто щоб отримати перетворення для суми функцій, можна зробити перетворення для кожної функції і потім їх скласти. Простіше кажучи, додавання та віднімання вихідної функції відповідає складання та віднімання результатів прямого ДПФ, і додавання та віднімання результатів зворотного ДПФ.

Рисунок 1.13 показує приклад поділу тимчасової області з використанням БПФ. У цьому прикладі 16-точковий сигнал розкладається на чотири окремі компоненти. Перший етап полягає у розбиття 16-ти точкового сигналу на два сигнали по 8 точок. На другому етапі відбувається розкладання даних сигналів на чотири сигнали по 4 точки. Ця процедура триває до того часу, поки сформовано N сигналів що з однієї точки. Черезрядкове розкладання використовується при поділі вибірки сигналу на дві вибірки. Сигнал поділяється

як при парному, і при непарному кількості відліків у вибірці. У системі проводиться 2N етапів у розкладанні, тобто. на 16-ти точковий сигнал (24) передбачає 4 етапи, 512 точки (27) вимагає 7 етапів, 4096 точки (212) передбачає 12 етапів і т.д.

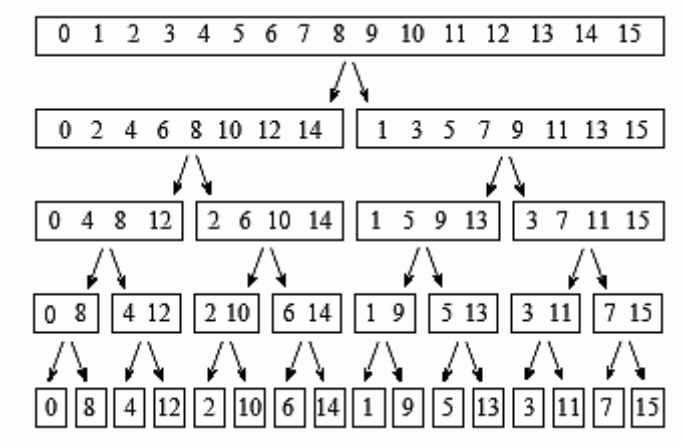

Рисунок 1.13 – Процедура розбиття сигналу

1.5 Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) є потужними архітектурами глибокого навчання, які здатні моделювати послідовності та враховувати контекстуальну залежність між елементами послідовності. У цьому тексті ми розглянемо основні принципи рекурентних нейронних мереж, їхню структуру та використання.

Основна ідея рекурентних нейронних мереж полягає в тому, що вони мають внутрішні зв'язки, що дозволяють передавати інформацію від попередніх кроків до наступних. Це дає можливість моделювати довгострокову залежність та контекст у послідовності.

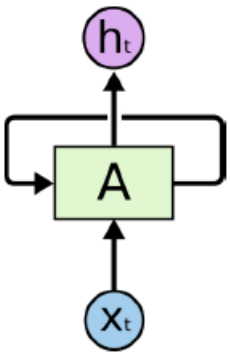

Рисунок 2.6 – Рекурентна мережа

Основною одиницею рекурентної нейронної мережі є рекурентний шар (Recurrent Layer), який приймає на вхід послідовність даних і генерує вихідну послідовність. У рекурентному шарі використовується рекурентний нейрон (Recurrent Neuron), який зберігає стан попереднього кроку та обчислює новий стан на основі вхідних даних та попереднього стану. Цей процес повторюється для кожного кроку послідовності.

У рекурентних нейронних мережах існує декілька типів рекурентних шарів, таких як прості RNN (Simple RNN), LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit). Кожен з цих типів має свої особливості, що дозволяють моделювати різні типи послідовностей та вирішувати різні завдання.

Прості RNN є найпростішою формою рекурентних нейронних мереж і мають обмежену здатність до моделювання довгострокових залежностей. LSTM та GRU шари були розроблені для подолання цього обмеження. Вони використовують спеціальні механізми, такі як ворота (gates) та комірки пам'яті, для контролю потоку інформації та запам'ятовування важливих даних на тривалий час. Це дозволяє їм ефективно моделювати довгострокові залежності та вирішувати завдання, такі як машинний переклад, генерація тексту та розпізнавання мови.

Існує багато різновидів, рішень та конструктивних елементів рекурентних нейронних мереж. Проблема рекурентної мережі полягає в тому, що якщо враховувати кожен крок часу, то стає необхідним для кожного кроку часу створювати свій шар нейронів, що викликає серйозні обчислювальні складності.

Крім того, багатошарові реалізації виявляються обчислювально нестійкими, тому що в них зазвичай зникають або зашкалюють ваги. Якщо обмежити розрахунок фіксованим тимчасовим вікном, то отримані моделі не відображатимуть довгострокових трендів. Різні підходи намагаються вдосконалити модель історичної пам'яті та механізм запам'ятовування та забування.

Рекурентні нейронні мережі не так сильно відрізняються від звичайних нейронних мереж. Їх можна уявити, як безліч копій однієї і тієї ж мережі, причому кожна копія передає повідомлення наступної копії.

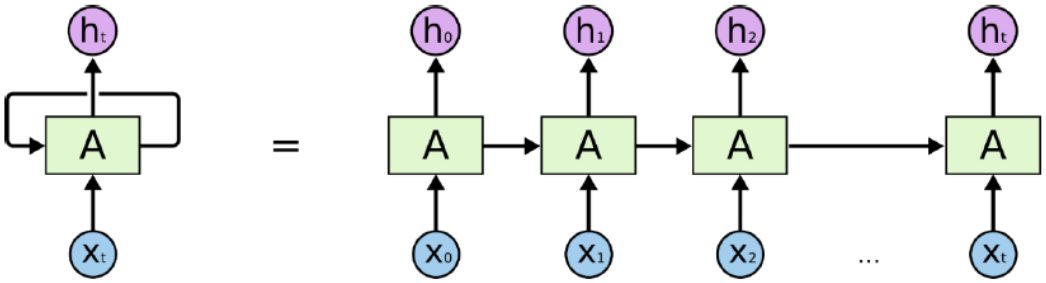

Рисунок 2.7 – Передача копій

Така “ланцюгова” сутність показує, що рекурентні нейронні мережі за своєю природою тісно пов'язані з послідовностями і списками.

Для навчання розпізнавача в загальному випадку необхідно подати на вхід зображення, вибрати архітектуру мережі, задати необхідні параметри та запустити процес навчання.

Для навчання нейронної мережі протягом однієї епохи можуть бути використані два різні підходи. У першому варіанті ми подаємо на вхід нейронної мережі приклади з навчальної вибірки один за одним. Після кожного прикладу ми оновлюємо ваги мережі. Такий підхід називається стохастичним (онлайн) навчанням. У другому варіанті ми подаємо на вхід нейронної мережі цілий пакет навчальних прикладів, після чого оновлюємо ваги мережі. Усередині пакета ми накопичуємо помилку на терезах мережі, щоб потім оновити їх.

Розглянемо стохастистичний режим трохи докладніше. При підрахунку помилки з прикладу навчальної вибірки необхідно обчислити градієнт функції

помилки кожному шарі нейронної мережі. Функція помилки, що використовується, - середньоквадратична.

$$E = \frac{1}{2}\sum_{j \in C} e_j^2, \qquad (2.2.1)$$

де безліч C включає всі нейрони вихідного шару мережі, - помилка для j-ого нейрона вихідного шару, різниця між виходом цього нейрона і бажаним значенням. Через те, що градієнт ми обчислюємо приблизно, до обчислень додасться деякий шум, який впливає на корекцію ваг. Але цей шум може бути корисним для навчання. Розглянемо переваги стохастичного навчання перед пакетним:e_j

- Стохастичне навчання в більшості випадків набагато швидше за пакетне.
- Стохастичне навчання часто призводить до найкращих розпізнавачів.
- Стохастичне навчання можна використовувати для відстеження змін

Уявімо випадків, у якому навчальна вибірка розміром 1000 складається з 10 ідентичних наборів по 100 прикладів. Усереднення градієнта по всій тисячі прикладів у пакетному навчанні дає такий самий результат, як і обчислення градієнта, засноване лише на першій сотні прикладів. Виходить, що пакетне навчання обчислює те саме значення 10 разів перед тим, як оновити ваги нейронної мережі. Стохастичне навчання, навпаки, представить цілу епоху як 10 ітерацій (епох) по навчальному набору довжини 100. На практиці приклади рідко зустрічаються більше одного разу в навчальній вибірці, але трапляються кластери дуже схожих один на одного прикладів.

Нелінійні мережі часто мають багато локальних мінімумів різної глибини. Завдання навчання полягає в попаданні мережі в один із мінімумів. Пакетне навчання призведе до мінімуму, на околицях якого спочатку розташовані ваги. У стохастичному навчанні шум, що з'являється при корекції ваг, призводить до

того, що мережа стрибає від одного локального мінімуму до іншого, можливо глибшого.

Розглянемо тепер переваги пакетного режиму навчання перед стохастичним:

- Умови збіжності добре вивчені
- Велика кількість технік прискорення навчання працює лише з пакетним режимом
- Теоретичний аналіз динаміки зміни ваг та швидкості збіжності простіше

Ці переваги випливають із того ж фактора шуму, який є у стохастичному навчанні. Цей шум можна прибирати різними способами. Незважаючи на деякі переваги пакетного режиму, стохастичний метод навчання використовується набагато частіше, особливо в тих завданнях, в яких велика вибірка. У системі реалізовано обидва методи навчання.

1.6 Тензорні операції

Рекурентні нейронні мережі (RNN) є багатим класом динамічних моделей, які використовуються для генерації послідовностей у таких різних областях, як музика, текст і дані захоплення руху. RNN можуть бути навчені для генерації послідовностей шляхом обробки послідовностей реальних даних по одному кроку за раз і прогнозування того, що буде далі. Припускаючи, що прогнози є ймовірнісними, нові послідовності можуть бути згенеровані з навченої мережі шляхом ітеративної вибірки вихідного розподілу мережі, а потім подачі вибірки як вхідні дані на наступному кроці. Хоча сама мережа є детермінованою, стохастичність, запроваджена під час відбору вибірок, викликає розподіл за послідовностями.

Рекурентні нейронні мережі «нечіткі» у тому плані, що вони не використовують точні шаблони з даних для навчання, щоб передбачити, а скоріше використовують багатовимірну інтерполяцію між прикладами навчання.

Рекурентні нейронні мережі, на відміну від шаблонних, складним шляхом синтезують та відновлюють навчальні дані та рідко генерують одні й ті самі речі двічі. Більше того, нечіткі передбачення не страждають від недоліків розмірності, і тому вони набагато краще підходять для моделювання реальних чи багатовимірних даних, ніж точні збіги (див. рис. 2.14).

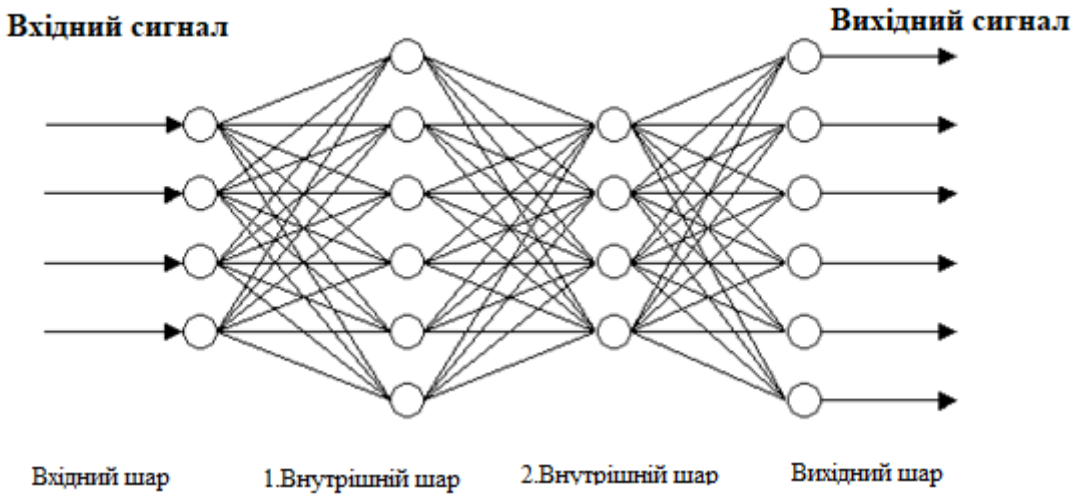

Рисунок 2.14 – Архітектура рекурентної нейронної мережі

Те, що рекурентні нейронні мережі дають оптимальні результати під час роботи з різними послідовностями зумовлено особливостями їхньої архітектури. Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) – це мережі, що містять зворотні зв'язки та дозволяють зберігати інформацію [2]. Рекурентну мережу можна розглядати як кілька копій однієї і тієї ж мережі, кожна з яких передає інформацію наступної копії. Якщо її розгорнути, ми побачимо, що RNN нагадують ланцюжок (див. рис. 2.15). Це і говорить про те, що вони тісно пов'язані із послідовностями та списками. Таким чином, RNN – найприродніша архітектура нейронних мереж для роботи з даними такого типу.

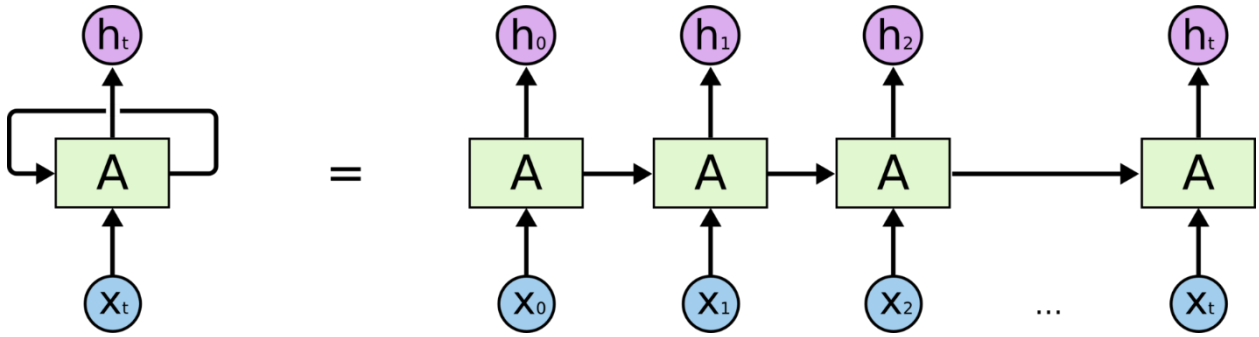

Рисунок 2.15 – Подання рекурентної нейронної мережі у вигляді послідовності

Довгострокова короткочасна пам'ять (LSTM – Long short-term memory) – це архітектура RNN, призначена для покращеного зберігання та доступу до інформації, ніж стандартні RNN, т.к. вона здатна до навчання довгострокових залежностей [9].

LSTM-модуль – це рекурентний модуль мережі, здатний запам'ятовувати значення як у короткі, і довгі проміжки часу. Ключем до цієї можливості є те, що LSTM-модуль (див. рис. 2.16) не використовує функції активації всередині своїх рекурентних компонентів. Таким чином, значення, що зберігається, не розмивається в часі, і градієнт або штраф не зникає при використанні методу зворотного поширення помилки в часі при тренуванні мережі.

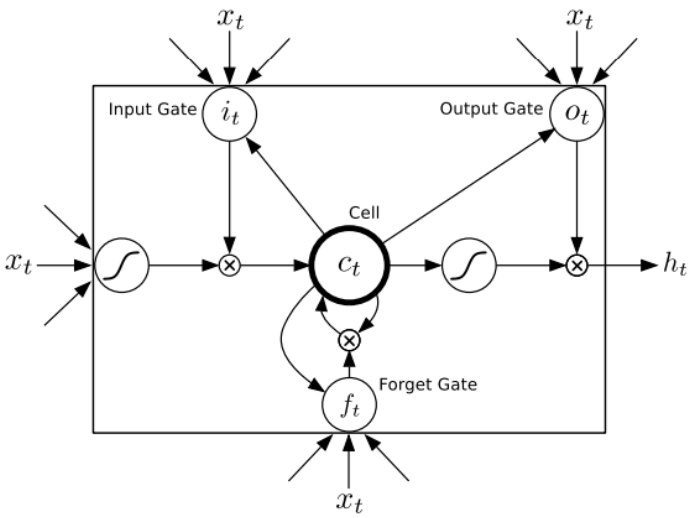

Рисунок 2.16 – Клітка довгої короткострокової пам'яті

Стан осередку схожий на конвеєрну стрічку (див. рис. 2.17), яка проходить безпосередньо через весь ланцюжок і бере участь лише в деяких лінійних перетвореннях. По ній інформація може текти без перешкод і не зазнавати жодних змін. Однак LSTM може видаляти інформацію зі стану комірки. Причому операція видалення регулюється спеціальними структурами, які називаються фільтрами (gates) (див. рис. 2.18).

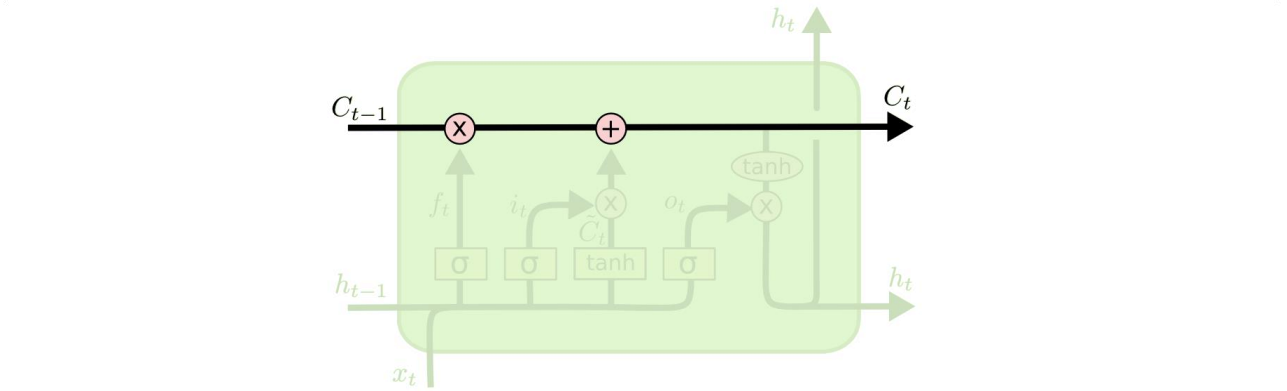

Рисунок 2.17 – Стан осередку – конвеєрна стрічка

Основою фільтрації є деякі умови, що дозволяють пропускати інформацію. Вони складаються із двох об'єктів. Ними є шар сигмоїдальної нейронної мережі та операція крапкового множення. Сигмоїдальний шар повертає числа від нуля до одиниці, які позначають, яку частку кожного блоку інформації слід пропустити далі через мережу. Нуль у разі означає “не пропускати нічого”, одиниця – “пропустити все”. У LSTM включає три фільтри, які дозволяють контролювати і захищати стан комірки.

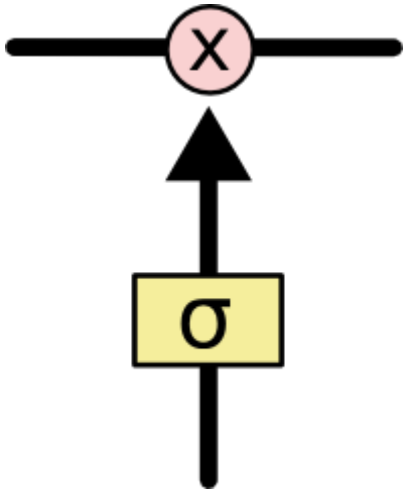

Рисунок 2.18 – Фільтр у LSTM

LSTM зараз дає передові результати в різних завданнях обробки послідовності, включаючи розпізнавання мови і почерку.

На рис. 2.19 зображено рекурентну нейронну мережу прогнозування. Вхідна векторна послідовність$x(1)$передається через зважені з'єднання в стек з N рекурентно пов'язаних прихованих шарів, щоб спочатку обчислити приховані векторні послідовності$h^n(2)$, а потім вихідну векторну послідовність (3).y

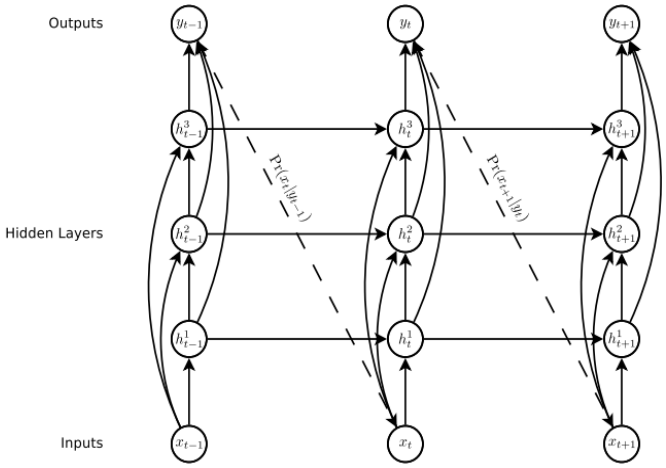

Рисунок 2.19 – Архітектура рекурентної нейронної мережі, де кола є шарами мережі, суцільні лінії – зважені зв'язки, а пунктирні – прогнози

$$x = (x_1, \dots, x_T) \quad (2.3.1)$$

$$h^n = (h_1^n, \dots, h_T^n) \quad (2.3.2)$$

$$y = (y_1, \dots, y_T) \quad (2.3.3)$$

Кожен вихідний векторy_tвикористовується для того, щоб параметризувати прогнозуючий розподіл по всіх можливих входах. Перший елемент кожної вхідної послідовності завжди є нульовим вектором, всі записи якого дорівнюють нулю; отже, мережа генерує прогноз першого реального входу без попередньої інформації. Мережа є «глибокою» як у просторі, і у часі, тому, що у кожен фрагмент інформації, що проходить або вертикально, чи горизонтально через

обчислювальний граф, впливатимуть численні послідовні вагові матриці і нелінійності.$\Pr(x_{t+1}|y_t)x_{t+1}x_1x_2$

Необхідно звертати увагу на «пропуск з'єднань» від входів до всіх прихованих шарів та від усіх прихованих шарів до виходів. Це значно полегшує процес навчання для глибоких мереж, зменшуючи кількість етапів обробки між дном мережі та верхом та, таким чином, пом'якшуючи проблему зникаючого градієнта. В особливому випадку, коли архітектура зводиться до звичайного однорівневого RNN (recurrent neural network$N = 1$- Рекурентної нейронної мережі) з прогнозуванням наступного кроку. Активації прихованого шару обчислюються шляхом ітерації наступних рівнянь від до і до :$t = 1Tn = 2N$

$$h_1^t = \mathcal{H}(W_{ih^1}x_t + W_{h^1h^1}h_{t-1}^1 + b_n^1) \quad (2.3.4)$$

$$h_t^n = \mathcal{H}(W_{ih^n}x_t + W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + b_n^1) \quad (2.3.5)$$

деW- вагові матриці (наприклад,W_{ih^n} - вагова матриця, що з'єднує входи зn-м прихованим шаром,$W_{h^1h^1}$- рекурентне з'єднання на першому прихованому шарі і т.д.),b – вектору зміщення (bias vector),$\mathcal{H}$ - Функція прихованого шару.

З урахуванням прихованих послідовностей, вихідні послідовності розраховуються так:

$$\hat{y}_t = b_y + \sum_{n=1}^{N} W_{h^ny}h_t^n \quad (2.3.6)$$

$$y_t = \mathcal{Y}(\hat{y}_t) \quad (2.3.7)$$

де$\mathcal{Y}$- Функція вихідного шару. Таким чином, вся мережа визначає функцію, параметризовану ваговими матрицями, від вхідних історій до вихідних векторів$x1: ty_t$.

Вихідні вектори використовуються для параметризації прогнозуючого розподілу для наступного входу. Форма має бути підібрана свідомо, спираючись на вхідні дані. Зокрема, знаходження відповідного прогнозуючого розподілу для багатовимірних реальних даних (зазвичай згадується як «моделювання щільності») може бути досить складним процесом. $y_t\Pr(x_{t+1}|y_t)\Pr(x_{t+1}|y_t)$

Імовірність, задана мережею для вхідної послідовності:x

$$\Pr(x) = \prod_{t=1}^{T} \Pr(x_{t+1}|y_t) \quad (2.3.8)$$

а втрати послідовності, що використовується для того, щоб навчити мережу, є негативним логарифмом від :$\mathcal{L}(x)$ $\Pr(x)$

$$\mathcal{L}(x) = -\sum_{t-1}^{T} \log \Pr(x_{t+1}|y_t) \quad (2.3.9)$$

Приватні похідні втрат за вагами мережі можуть бути ефективно розраховані зі зворотним розповсюдженням помилки у часі, застосованим до графа обчислень, і мережа потім може бути навчена за допомогою алгоритму градієнтного спуску.

У більшості RNN функція прихованого шару є поелементним застосуванням сигмоїдальної функції. Для версії LSTM реалізована за допомогою наступної складової функції:$\mathcal{HH}$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (2.3.10)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (2.3.11)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (2.3.12)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \quad (2.3.13)$$

$$h_t = o_t \tanh(c_t) \quad (2.3.14)$$

Де σ-функція логістичної сигмоїди, а й i, f, o c-відповідно, вхідні вектори активації для вхідного фільтра, що пропускає фільтра, тобто. фільтра, який здатний видаляти інформацію зі стану комірки, вихідного фільтра, а також комірки та входу в комірку (input gate, forget gate, output gate, cell and cell input), кожен з яких має той же розмір, що і прихований вектор .h

В оригінальному алгоритмі LSTM використовується спеціально розроблений розрахунок приблизного градієнта, який дає змогу оновлювати ваги після кожного тимчасового кроку [14]. Однак натомість повний градієнт може

бути розрахований зі зворотним поширенням помилки у часі, метод, який буде використовуватися далі.

Одна з труднощів при навчанні LSTM з повним градієнтом полягає в тому, що похідні іноді стають надмірно великими, що призводить до чисельних проблем. Щоб запобігти цьому, можна обрізати похідну втрат по відношенню до виходів мережі в шари LSTM (до застосування функцій сигмоїду та гіперболічного тангенсу), щоб вони знаходилися в заздалегідь визначеному діапазоні.

1.7 Матричні операції

Матричні операції є дуже важливими для моделей машинного навчання, таких як лінійна регресія, оскільки вони часто використовуються в них. TensorFlow підтримує всі найпоширеніші операції з матрицями, такі як множення, інверсія, обчислення визначника, рішення лінійних рівнянь та багато іншого.

Процес поширення сигналу в С-шарі представлено на рис. 2.20:

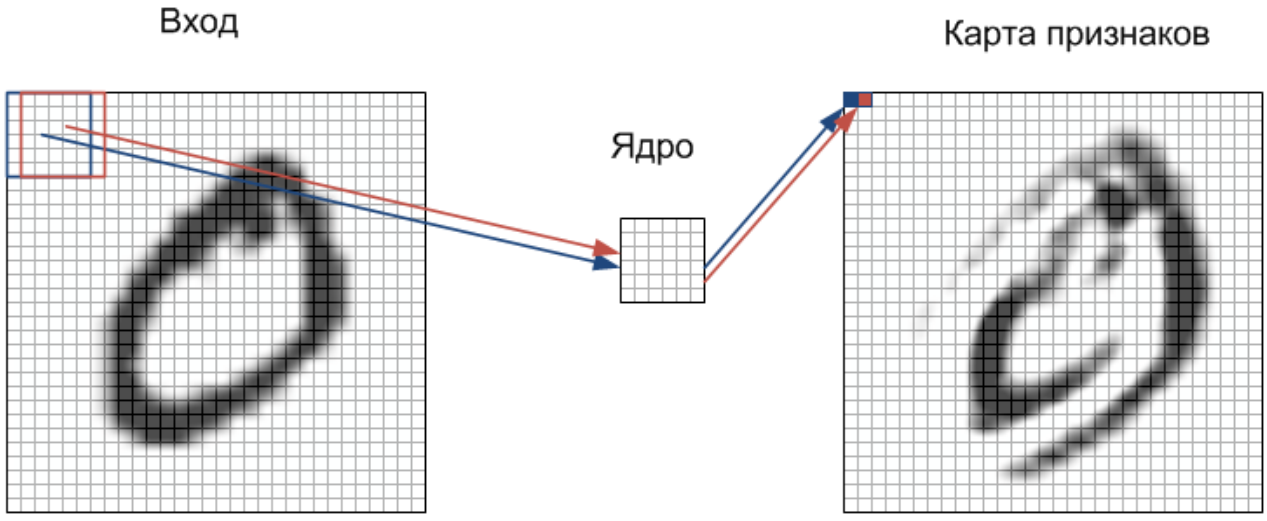

Рисунок 2.20 – Використання матричних операцій

Для того щоб можна було розпочати навчання нашої мережі, потрібно визначитися з тим, як вимірювати якість розпізнавання. В даном випадку для

цього будемо використовувати найпоширенішу в теорії нейронних мереж функцію середньоквадратичної помилки (СКО, MSE):

$$E^p = \frac{1}{2}(D^p - O(I^p, W))^2 \qquad (2.4.1)$$

У цій формулі Ep - це помилка розпізнавання для p-ої навчальної пари, Dp - бажаний вихід мережі, O(Ip,W) - вихід мережі, що залежить від p-го входу і вагових коефіцієнтів W, куди входять ядра згортки, зсуви, вагові коефіцієнти S- та F-шарів. Завдання навчання так налаштувати ваги W, щоб вони будь-якої навчальної пари (Ip,Dp) давали мінімальну помилку Ep. Щоб порахувати помилку для всієї навчальної вибірки, просто береться середнє арифметичне за помилками для всіх навчальних пар. Таку усереднену помилку позначимо як E.

Для мінімізації функції помилки Ep найефективнішими є градієнтні методи. Розглянемо суть градієнтних методів на прикладі найпростішого одновимірного випадку (тобто коли у нас лише одна вага). Якщо ми розкладемо в ряд Тейлора функцію помилки Ep, то отримаємо такий вираз:

$$E(W) = E(W_c) + (W - W_c)\frac{dE(W_c)}{dW} + \frac{1}{2}(W - W_c)^2\frac{d^2E(W_c)}{dW^2} + \cdots \qquad (2.4.2)$$

Тут E – та сама функція помилки, Wc – деяке початкове значення ваги. Для знаходження екстремуму функції необхідно взяти її похідну та прирівняти нулю. Так і вчинимо, візьмемо похідну функції помилки за вагами, відкинувши члени вище 2-го порядку:

$$\frac{dE(W)}{dW} = \frac{dE(W_c)}{dW} + (W - W_c)\frac{d^2E(W_c)}{dW^2} \qquad (2.4.3)$$

з цього виразу випливає, що вага, при якому значення функції помилки буде мінімальною, можна обчислити з наступного виразу:

$$W_{min} = W_c - (\frac{d^2E(W_c)}{dW^2})^{-1}\frac{dE(W_c)}{dW} \qquad (2.4.4)$$

Тобто. оптимальна вага обчислюється як поточний мінус похідна функції помилки за вагою, поділена на другу похідну функції помилки. Для багатовимірного випадку (тобто для матриці ваг) все одно, тільки перша похідна перетворюється на градієнт (вектор приватних похідних), а друга похідна перетворюється на Гессіан (матрицю других приватних похідних).

1.8 Сегментація за допомогою нейронних мереж

Розглянемо роботу фільтра №1 (виділеного на Рис. 3). Після того, як на вхід мережі подано зображення, фільтр переміщається по всьому об'єму, крок за кроком зсуваючись вниз або вправо, і для кожного положення обчислюється операція згортки. Тобто кожен елемент у ядрі множиться на значення відповідного пікселя зображення, потім отримані значення складаються та поділяються на кількість елементів. Таким чином, кожна відповідність елементів призводить до підсумкового значення 1. Аналогічно будь-яка невідповідність відображається результатом -1. У графічному вигляді це показано на рис.2.23.

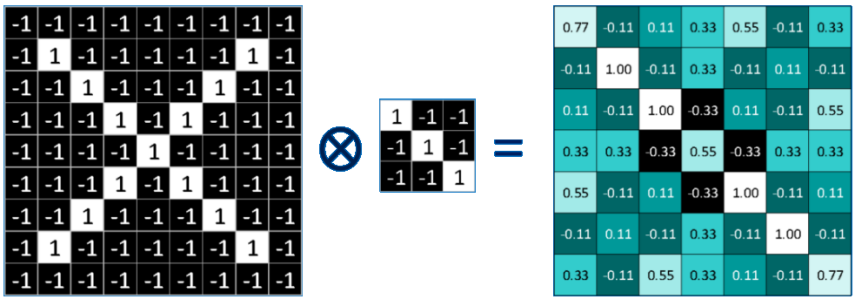

Рисунок 2.23 – Застосування фільтра №1 до зображення

Застосувавши аналогічно ядра №2 і №3, отримаємо три відфільтрованих варіанти вихідного зображення, які називаються картами ознак, Рис. 2.24.

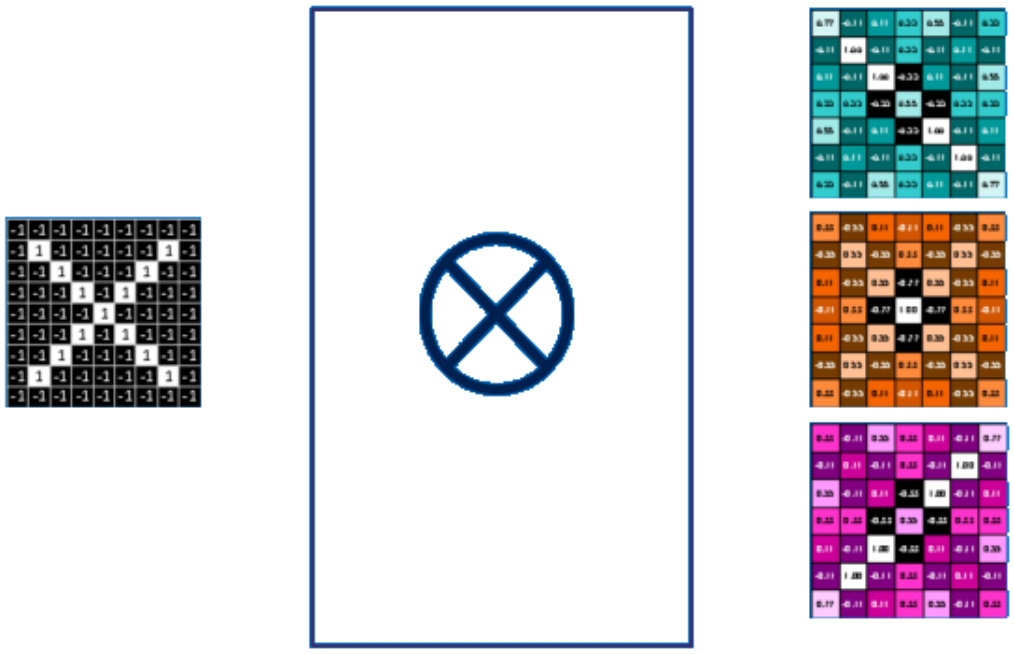

Рисунок 2.24 – Карти ознак вихідного зображення

Далі продемонструємо роботу ще одного інструменту згорткової мережі – шар субдискретизації. Він необхідний масштабування великих зображень (шарів) і, здійснюючи стиск, зберігає у своїй найзначнішу інформацію. Механізм його роботи простий: накласти невеликий фільтр на зображення та отримати максимальне значення із цієї області на кожному кроці. На практиці використовується ядро розміром 2 або 3 пікселя та кроком 2.

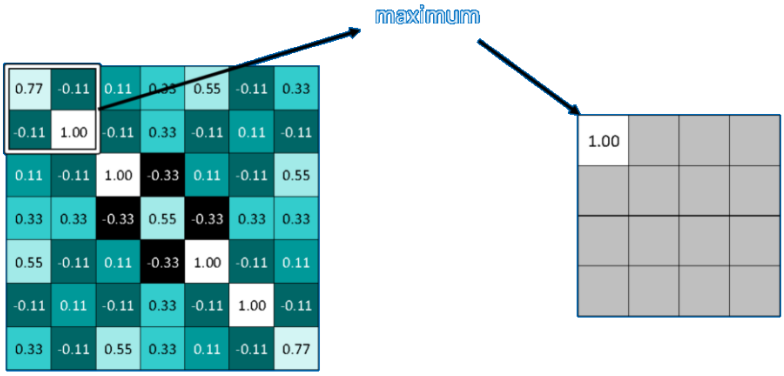

Рисунок 2.25 – Поодинока ітерація шару субдискретизації

Після вибору зображення має приблизно чверть пікселів, щодо вихідного. Результатом роботи цього шару є те, що мережі згортки можуть визначити, чи знаходиться потрібний фільтр у зображенні, незалежно від невеликих зрушень і спотворень. Після зазначеного шару субдискретизації, отримані на Рис. 2.25 карти ознак набудуть наступного вигляду (Рис. 2.26):

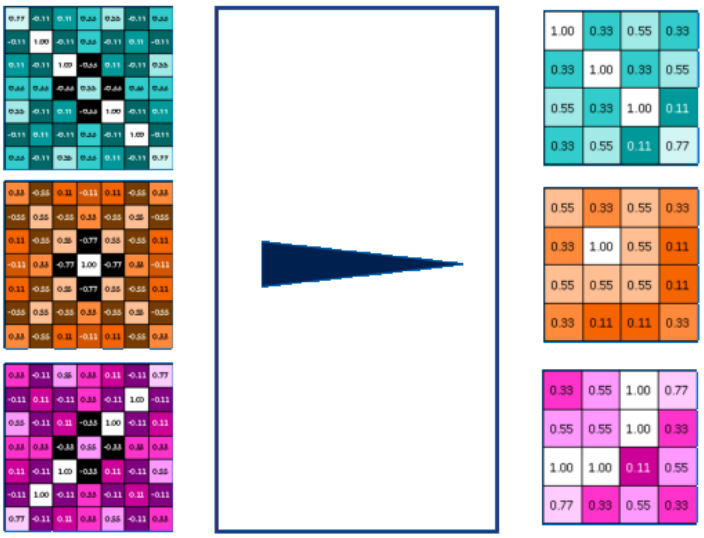

Рисунок 2.26 – Результат шару субдискретизації

Важливим допоміжним кроком цьому етапі є «Лінійна ретифікація», тобто, кожне негативне значення замінюється на нуль [13]. Це допомагає згорткової мережі від утримання вихідних значень від застрягання близько 0 або збільшення до нескінченності. Після проведення «ретифікації» вид карток ознак наведено на Рис. 2.27.

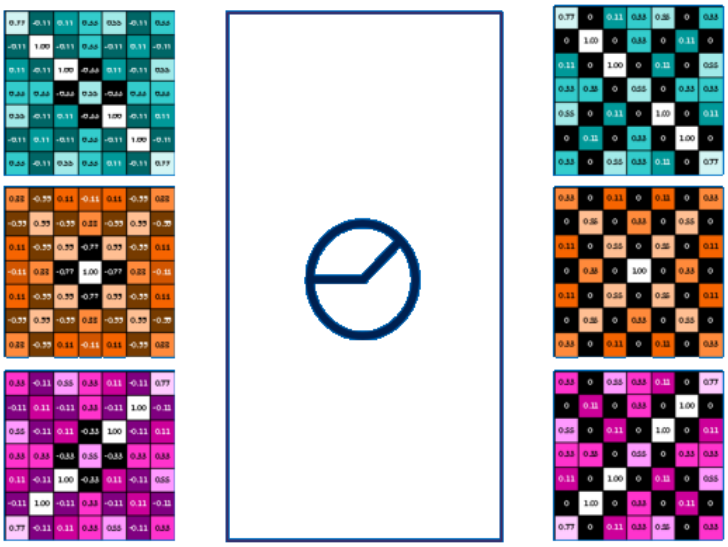

Рисунок 2.27 – Звільнення від негативних значень

Проводячи отримані карти ознак далі по згортковим шарам, ми маємо можливість обчислювати дедалі складніші функції, тобто розпізнавати високорівневі аспекти зображення, як-от форми і візерунки. На останньому шарі отриманий з вихідного зображення вектор подається на вхід повнозв'язного шару, нейрони якого визначають оцінку класу, у нашому прикладі ймовірність того, що представлене зображення X або O (Рис. 2.28).

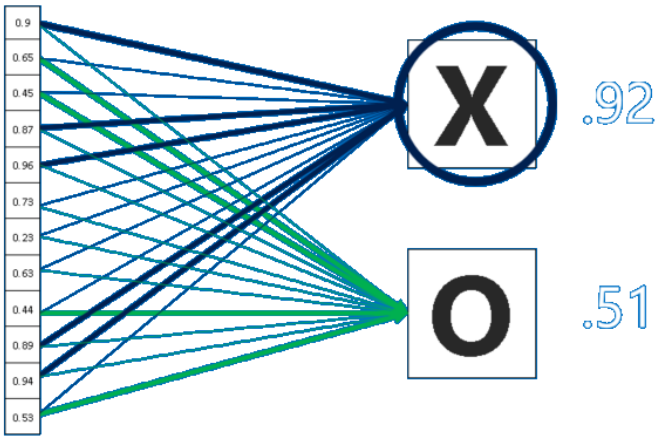

Рисунок 2.28 – робота повнозв'язного шару

1.9 Генерація вихідного коду для інформаційної частини програмного середовища

Нейронна мережа розпізнавання зображень складається з різних шарів, кожен з яких відповідає за виконання певних операцій і обчислень. Кожен шар має свої власні параметри, які визначають його поведінку та впливають на роботу всієї мережі. У цьому тексті ми розглянемо основні параметри, які використовуються в шарах нейронної мережі розпізнавання зображень.

Розмір ядра (Kernel Size): розмір ядра визначає просторовий розмір фільтра або згорткової матриці, який застосовується до вхідних даних. Це важливий параметр, який впливає на розмір та форму виходу шару. Наприклад, згортковий шар з ядром розміром 3x3 застосовує фільтр розміром 3x3 до вхідних даних для отримання вихідного зображення.

Кількість фільтрів (Number of Filters): кількість фільтрів визначає кількість незалежних фільтрів або ядер, які будуть застосовані до вхідних даних. Кожен фільтр відповідає за виявлення певних особливостей чи ознак у зображенні. Більша кількість фільтрів дозволяє моделі виявляти більш різноманітні особливості та складні залежності у зображеннях.

Крок зсуву (Stride): крок зсуву визначає кількість пікселів, на яку фільтр або ядро зсувається під час обробки зображення. Він впливає на розмір виходу шару та на чутливість моделі до деталей зображення. Більший крок зсуву призводить до зменшення розміру виходу, але може призвести до втрати деякої локальної інформації.

Заповнення (Padding): заповнення використовується для додавання додаткових пікселів навколо вхідних даних перед застосуванням фільтра або ядра. Це допомагає зберегти розмір зображення та уникнути зайвої зміни розміру під час обробки. Зазвичай використовуються два типи заповнення: 'valid', де заповнення не використовується, та 'same', де вхідні дані доповнюються таким чином, щоб розмір виходу співпадав з розміром вхідних даних.

Функція активації (Activation Function): функція активації визначає, які значення будуть передані з одного шару до наступного. Вона надає нелінійність та можливість моделі виражати складні залежності. Деякі поширені функції активації включають ReLU (Rectified Linear Unit), sigmoid та tanh. Вибір функції активації залежить від конкретної задачі та характеристик даних.

Складність шару (Layer Complexity): складність шару визначається кількістю параметрів, які потрібно навчити в процесі тренування. Більш складні шари можуть мати більшу потужність, але вимагають більше обчислювальних ресурсів та тренувального часу.

Параметри шарів нейронної мережі розпізнавання зображень грають важливу роль у визначенні поведінки та результатів моделі. Врахування цих параметрів під час побудови та налагодження моделі допомагає досягти кращої точності та ефективності у завданні розпізнавання зображень.

Просторовий розмір вихідного шару можна обчислити за формулою, яка включає розмір вхідного об'єму (), розмір рецептивного поля нейронів шару згортки (), крок, з яким вони застосовуються (), і розмір нульового відступу () на кордоні.$WFSP$

Формула має вигляд:

$$\frac{W-F+2P}{S}+1 \qquad (2.1.1)$$

Наприклад, для вхідного зображення розміром 7x7, фільтра 3x3 з кроком 1 і без нульового заповнення ми отримали вихід 5x5. З кроком 2 ми отримали вихід 3x3. Розглянемо ще одне графічний приклад на Рис. 2.1.

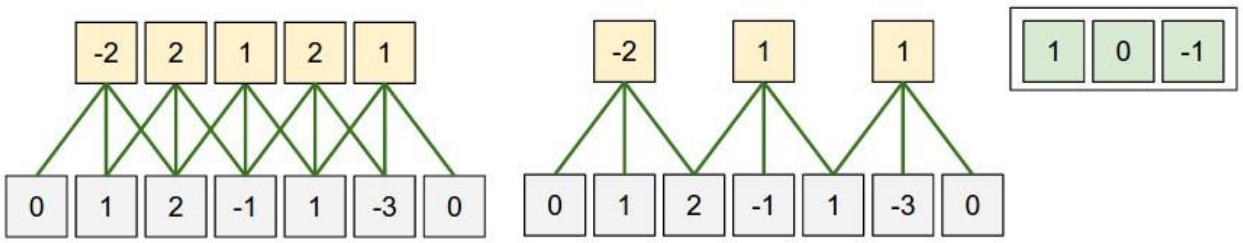

Рисунок 2.1 – Ілюстрація просторового розташування

У цьому прикладі вхідний вектор одномірний, розміру, розмір рецептивного поля та нульове заповнення. Зліва для кроку$W = 5F = 3, P = 1S = 1$, кількістьнейронів другого шару. Праворуч, відповідно розмір 2-го шару. Ваги нейронів у цьому прикладі [1, 0, -1] (показані праворуч), і зсув дорівнює нулю. Ці ваги загальні всім нейронів другого шару.$\frac{5-3+2}{1} + 1 = 5S = 2\frac{5-3+2}{2} + 1 = 3$

Зазначимо, що для випадку розмірність вхідного вектора збіглася з розмірністю вихідного. Якби не було нульового заповнення, то кількість нейронів другого шару дорівнювала 3, тому що саме стільки рецептивних полів було б «вписано» у вихідний вектор. Як правило, встановлення нульового заповнення при кроці $S = 1P = \frac{F-1}{2}S = 1$ гарантує, що вхідний та вихідний шари матимуть однаковий розмір у просторі. Це основне застосування нульового відступу, інші причини будуть згадані в розділі архітектури CHP.

Необхідно відзначити, що гіперпараметри просторового компонування накладають взаємні обмеження. Наприклад, коли вхідний вектор має розмір , не використовується нульовий відступ, а розмір фільтра тоді неможливо використовувати, оскільки, тобто не ціле число. Воно вказує на те, що вікна сканування перетинаються на вхідному зображенні, і таке поєднання параметрів вважається неприпустимим. Варто відзначити, що підібрати необхідні значення так, щоб усі виміри «працювали», може бути важко. Дещо полегшити завдання може використання нульового заповнення та деяких принципів проектування, які будуть згадані пізніше.$W = 10P = 0F = 3S = 2\frac{W-F+2P}{S} + 1 = \frac{10-3+0}{2} + 1 = 4.5$

У описі рис. 1 згадувалися загальні ваги згорткового шару. Такі ваги також називаються розділеними. Принцип розділених ваг використовується в згорткових шарах для контролю кількості параметрів, що настроюються. Якщо ваги розрізняються для кожного нейрона кожної карти ознак, підсумкова кількість параметрів, що настроюються дуже велика. Однак значне зменшення числа параметрів можна досягти, прийнявши таке припущення: якщо одна

ознака корисна для обчислення в деякій просторовій позиції, то вона також повинна бути корисною для обчислення в іншій позиції. Іншими словами, для кожного нейрона, що входить в окрему карту ознак будуть використовуватися однакові ваги та зміщення. На практиці під час навчання методом зворотного поширення помилки кожен нейрон у шарі обчислює градієнт для своїх ваг,$(x, y)(x_2, y_2)$

Зауважимо, що якщо всі нейрони одного шару однієї карти ознак використовують один і той же ваговий вектор, то пряме проходження вхідного образу по мережі в кожному шарі можна обчислити як згортання ваги нейронів з вхідними значеннями. Звідси випливає назва шару: згортковий. Саме тому прийнято називати набори ваги фільтром або ядром.

Однак припущення про спільне використання ваг не завжди має сенс. Це особливо помітно, коли вхідні зображення мають фіксовану структуру, наприклад, якщо на одній стороні зображення передбачається вивчати функції, що кардинально відрізняються, ніж на іншій. Одним із практичних прикладів є випадок, коли вхідними зображеннями є особи, які були центровані. Логічно припустити, що різні, специфічні, наприклад, для очей чи волосся особливості мають бути вивчені у різних просторових розташуваннях. У цьому випадку зазвичай ігнорується схема спільного використання ваг, а натомість просто викликаються локальні згорткові шари.

Кожен вхідний вектор складається із дійсної пари $x_t(x_1, x_2)$,яка визначає зсув пера від попереднього введення, поряд з двійковим має значення 1, якщо вектор завершує межу (тобто, якщо ручка була знята з дошки до того, як був записаний наступний вектор), і значення 0 в іншому випадку. Суміш двовимірних гауссіанів використовувалася для передбачення, тоді як розподіл Бернуллі використовувався для моделювання ймовірності кінця ходу. Отже, кожен вихідний вектор складається з ймовірності кінця ходу, а також набору середніх, стандартних відхилень, кореляцій та ваг суміші для $x_3x_1, x_2x_3y_te\mu_j\sigma_j\rho_j\pi_jM$ компонентів суміші. Тобто

$$x_t \in \mathbb{R} \times \mathbb{R} \times \{0,1\} \qquad (2.1.2)$$

$$y_t = (e_t, \{\pi_t^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_{j=1}^M) \quad (2.1.3)$$

Важливо, що середнє значення та стандартне відхилення є двовимірними векторами, тоді як вага компонента, кореляція та ймовірність кінця ходу є скалярними. Вектори отримані з виходів мережі, де $y_t \hat{y}_t$

$$\hat{y}_t = (\hat{e}_t, \{\hat{\pi}_t^j, \hat{\mu}_t^j, \hat{\sigma}_t^j, \hat{\rho}_t^j\}_{j=1}^M) = b_y + = \sum_{n=1}^N W_{h^n y} h_t^n \quad (2.1.4)$$

отримуємо:

$$e_t = \frac{1}{1+\exp(\hat{e}_t)} \Rightarrow e_t \in (0,1) \quad (2.1.5)$$

$$\pi_t^j = \frac{\exp(\hat{\pi}_t^j)}{\sum_{j'}^M \exp(\hat{\pi}_t^{j'})} \Rightarrow \pi_t^j \in \sum_j \pi_t^j = 1 \quad (2.1.6)$$

$$\mu_t^j = \hat{\mu}_t^j \quad \Rightarrow \quad \mu_t^j \in \mathbb{R}$$

$$\sigma_t^j = \exp(\hat{\sigma}_t^j) \quad \Rightarrow \quad \sigma_t^j > 0$$

$$\rho_t^j = \tanh(\hat{\rho}_t^j) \quad \Rightarrow \quad \rho_t^j \in (-1,1)$$

Щільність ймовірності наступного вхідного значення при заданому векторі визначається наступним чином:$\Pr(x_{t+1} \mid y_t)$ $x_{t+1} y_t$

$$\Pr(x_{t+1} \mid y_t) = \sum_j^M \pi_t^j \mathcal{N}(x_{t+1} \mid \mu_t^j, \sigma_t^j, \rho_t^j) \begin{cases} e_t, & \text{якщо } (x_{t+1})_3 = 1 \\ 1 - e_t, & \text{інакше} \end{cases} \quad (2.1.7)$$

де

$$\mathcal{N}(x \mid \mu, \sigma, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[\frac{-Z}{2(1-\rho^2)}\right] \quad (2.1.8)$$

$$Z = \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} + \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \quad (2.1.9)$$

Тепер визначити втрату послідовності (з точністю до константи, яка залежить лише від кількісного визначення даних та не впливає на навчання мережі):

$$\mathcal{L}(x) =$$

$$\sum_{t=1}^T -\log(\sum_j \pi_t^j \mathcal{N}(x_{t+1} \mid \mu_t^j, \sigma_t^j, \rho_t^j)) - \begin{cases} \log e_t, & \text{якщо } (x_{t+1})_3 = 1 \\ \log(1 - e_t), & \text{в інших випадках} \end{cases} \quad (2.1.10)$$

На рис. 2.2 представлено верхню теплову карту. Вона показує послідовність розподілів ймовірностей для передбачуваних положень пера під час написання слова «under». Щільності для послідовних передбачень підсумовуються, даючи високі значення там, де перекриваються розподіли.

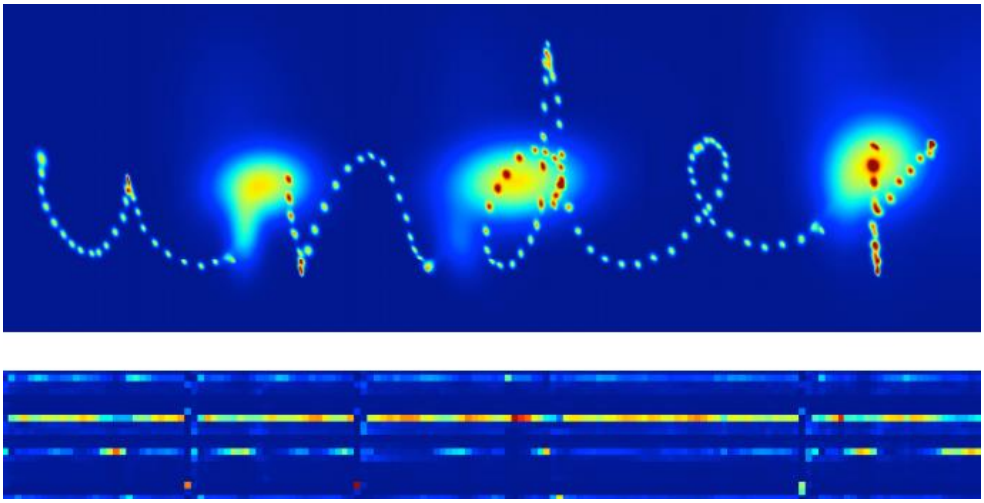

Рисунок 2.2 – «Теплова карта»

На карті щільності видно два типи прогнозу: маленькі краплі, які пишуть літери, є прогнозами під час написання штрихів, три великі краплі є прогнозами на кінцях штрихів для першої точки в наступному штриху. Прогнози кінця ходу мають набагато більшу дисперсію, тому що положення пера не було записано, коли воно знаходилося поза дошкою, і, отже, між кінцем однієї частини та початком наступної може бути велика відстань.

Алгоритм роботи системи можна розділити на 3 етапи:

1. Виділіть область розташування символів на зображенні.

2. Виділення окремих символів.

3. Розпізнавання символів.

Для аналізу вихідного зображення та пошуку на ньому об'єктів необхідно провести бінаризацію, яка є переведенням повнокольорового зображення в монохромне, де присутні лише два типи пікселів (темні та світлі). Бінаризація зображення ґрунтується на порівнянні інтенсивності кожного пікселя з граничним значенням інтенсивності. Якщо значення інтенсивності пікселя вище

значення інтенсивності порога, то цьому пікселю присвоюється значення 255, або 0.

Порогове значення T обчислюється за такою формулою:

$$T = \frac{I_{max}+I_{min}}{2}, \quad (2.1.11)$$

де I_{max} – максимальне значення інтенсивності зображення, I_{min} – мінімальне значення інтенсивності зображення.

Виділення області розташування символів на зображенні починається із сканування вхідного зображення вікном розміром 3×3:

P_0	P_1	P_2
P_7	$F(i,j)$	P_3
P_6	P_5	P_4

Усередині вікна обчислюється кількість точок з інтенсивністю 255, потім обчислюється відношення кількості даних точок і площі вікна, що сканує, і порівнюється з пороговим значенням T. Якщо дане відношення перевищує задане порогове значення, то дана область зображення визначається як область-кандидат. Експериментально було встановлено, що цей поріг дорівнює 0,3.

Вікно переміщається попіксельно по всьому зображенню і для точки, що опинилася в центрі вікна, обчислюється нове значення інтенсивності за формулою:

$$K_s(i,j) = \sqrt{X^2 + Y^2}\,, \quad (2.1.12)$$

$$X = [P_2 + 2P_3 + P_4] - [P_0 + 2P_7 + P_6], \quad (2.1.13)$$

$$Y = [P_0 + 2P_1 + P_2] - [P_6 + 2P_5 + P_4], \quad (2.1.14)$$

де $K_s(i,j)$ - нове значення інтенсивності точки з координатами (i, j); P_n - значення інтенсивності n-го пікселя.

Потім на зображенні проводиться дослідження областей, що мають найбільший контраст, та обчислюється кількість їх граней. Якщо кількість граней знаходиться в певному діапазоні, ця область відзначається як кандидат на

місцезнаходження номерного знака. Результатом цього підходу є перебування на зображенні всіх областей кандидатів, у яких може бути ознака цифри.

Основною перевагою даного методу є те, що він заснований на використанні інформації про символи та номерну пластину і не чутливий до параметрів контрастності та кольору зображення.

Недоліком цього методу є те, що результатом визначення меж є велика кількість ліній як горизонтальних, так і вертикальних, отже, велика кількість помилкових областей зображення може бути віднесена до кандидатів на зміст цифри. Тому пропонується використовувати виділення тільки вертикальних меж зображення, щоб уникнути надлишку кількості кордонів та неправдивих областей кандидатів.

Як вихідні дані для проведення процесу моделювання обробки були обрані бінарні зображення цифр, рисунок 4а. На рис. 2(б) показано бінарне зображення 456-точкового кордону цифри 3.

а) б)

Рисунок 2.4 – (а). Зображення цифр; Мал. 2(б). Зображення межі цифри 3

На рис. 2.4 показані відновлені зображення межі цифри 3. При описі цифри використовувалося відповідно 30, 25, 20, 15, 10 та 8 Хартлі-дескрипторів. Це становить 6,6%, 5,5%, 4,4%, 3,3%, 2,2% та 1,8% від усіх 456 дескрипторів.

Рисунок 2.5 – (а) - (е). Зображення відновлених меж цифри 3 із застосуванням Хартлі-дескрипторів

Розділ 2
Математична модель

Модель згорткової нейрогої мережі.

Спочатку у згортковому шарі використовуються фільтри (kernel) для виявлення різних ознак на зображенні. Кожен фільтр має свої ваги, які використовуються для обчислення активаційного значення в кожному пікселі карти активацій. Це робиться за допомогою операції згортки, яка множить кожен піксель вхідного зображення на відповідну вагу фільтра і сумує їх. Потім до цієї суми додається зсув (bias) і результат проходить через функцію активації. Отримуємо карту активацій, яка представляє собою відповідь фільтра на вхідне зображення.

Формула згортки:

$$C_{ij} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I_{(i+m)(j+n)} \cdot K_{mn} + B \qquad (2.5.1)$$

Тут C_{ij} - активація в позиції (i, j) на карті активацій, I - вхідне зображення, K - ваги фільтра (ядро), B - зсув (bias).

Після згорткового шару застосовується пулінг (підвибірка), що допомагає зменшити розмір карти активацій і зберегти важливі ознаки. У своїй моделі я використав максимальний пулінг, де обирається максимальне значення з певної області. Пулінг не має параметрів, а лише зменшує розмір карти активацій.

Формула максимального пулінгу:

$$P_{ij} = \max(I_{(2i)(2j)}, I_{(2i)(2j+1)}, I_{(2i+1)(2j)}, I_{(2i+1)(2j+1)}) \qquad (2.5.2)$$

У даній формулі P_{ij}- вихідне значення пулінгового шару, I - карта активацій з попереднього шару.

Після згорткових шарів і шарів пулінгу використовується повнозв'язаний (повнозв'язковий) шар, який приймає векторизовані значення з останнього шару пулінгу і виконує класифікацію об'єктів. Кожен нейрон повнозв'язаного шару

пов'язаний з кожним нейроном попереднього шару за допомогою вагів. Далі я застосовую функцію активації для отримання прогнозованої вірогідності належності до різних класів.

Формула активації повнозв'язаного шару:

$$A_j = \sigma(\sum_{i=0}^{N-1} W_{ij} \cdot H_i + B_j) \quad (2.5.3)$$

Де A_j - активація в нейроні j повнозв'язаного шару, W – ваги, H - вектор активацій попереднього шару, B - зсув (bias), σ - функція активації.

Далі для навчання мережі використовується функція втрат, яка порівнює прогнозовані вірогідності класів з правильними мітками. Категоріальна хрест-ентропія - одна з поширених функцій втрат для багатокласової класифікації. Ця функція штрафує за відхилення прогнозованих вірогідностей від очікуваних значень.

Формула категоріальної хрест-ентропії:

$$L = -\sum_{i=1}^{C} Y_i \log(P_i) \quad (2.5.4)$$

У даній формулі, L - значення функції втрат, Y - очікувані значення (one-hot encoding), P - прогнозовані значення.

Далі використовується метод оптимізації для зміни ваг і зсувів мережі, щоб мінімізувати значення функції втрат. Звичайною оптимізаційною процедурою для згорткових нейронних мереж є алгоритм зворотного поширення помилки (backpropagation), який використовує градієнти функції втрат по вагам для оновлення їх значень. У моїй роботі був використаний алгоритми оптимізації Adam.

Модель багатошарового перцептрону.

Вхідні дані x подаються на вхідний шар нейронної мережі. Кожен нейрон в прихованих шарах та вихідному шарі має свої ваги W^l(2.5.5) та зсув b^l(2.5.6). Зважений сумарний вхід z^l обчислюється шляхом взяття дотичного добутку вектору ваг W^l та активацій a^{l-1} попереднього шару, та додавання зсуву b^l.

Формула для обчислення зваженого сумарного входу z^l у шарі l виглядає наступним чином(2.5.7):

$$W^l = \begin{bmatrix} w_{11}^l & \cdots & w_{1m}^l \\ \vdots & \ddots & \vdots \\ w_{k1}^l & \cdots & w_{km}^l \end{bmatrix} \quad (2.5.5)$$

$$b^l = [b_1^l, b_2^l, \dots, b_k^l] \quad (2.5.6)$$

$$z^l = W^l \cdot a^{l-1} + b^l \quad (2.5.7)$$

де W^l - матриця ваг розміром $(k \times m)$, де k - кількість нейронів у шарі l, а m- кількість нейронів у попередньому шарі $(l-1), a^{l-1}$ - вектор активацій попереднього шару, b^l - зсув шару l.

Після обчислення зваженого сумарного входу, застосовується функція активації f^l, щоб отримати активацію шару a^l:

$$a^l = f^l(z^l) \quad (2.5.8)$$

Існує кілька різних функцій активації, які можуть бути використані в багатошаровому персептроні, кожна з яких має свої унікальні властивості:

- Сигмоїдна функція
- Функція ReLU
- Функція Tanh
- Функція softmax

У своїй моделі я використав функції ReLU(2.5.9) та softma(2.5.10):

$$f(x) = \max(0, z) \quad (2.5.9)$$

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad (2.5.10)$$

Процес передачі сигналу вперед повторюється для кожного прихованого шару персептрону (від шару 1 до шару $L-1$). Для кожного шару обчислюється зважений сумарний вхід z^l та активація a^l згідно з формулами, описаними 2.5.5-2.5.7.

Вихідний шар працює аналогічно прихованим шарам. Зважений сумарний вхід z^l та активація a^l обчислюються за формулами, описаними раніше. Оскільки багатошаровий персептрон використовується для класифікації, функцією активації останнього шару є функція softmax, що генерує ймовірності для кожного класу. Класифікація здійснюється шляхом вибору класу з найвищою ймовірністю.

Розділ 3
Програмний застосунок для розпізнавання рукописних цифр

3.1. Обґрунтування вибору засобів розробки

Мова Python стала однією з найпопулярніших мов програмування в галузі розробки програм нейронних мереж для розпізнавання зображень. Її простота, зручність використання та широкий спектр бібліотек і фреймворків зробили її оптимальним вибором для таких завдань. У цьому тексті ми розглянемо основні переваги Python для розробки програм нейронних мереж розпізнавання зображень.

Простота та зручність використання:

Python є високорівневою мовою програмування, що робить її легкою для вивчення та розуміння. Синтаксис мови Python простий і лаконічний, що дозволяє програмістам швидко розробляти програми нейронних мереж. Це особливо важливо в галузі розпізнавання зображень, де швидкість розробки і експериментування мають велике значення.

Широкий вибір бібліотек та фреймворків:

Python має велику кількість бібліотек і фреймворків, спеціалізованих на обробці зображень та розпізнаванні об'єктів. Одні з найпопулярніших бібліотек включають TensorFlow, Keras, PyTorch та OpenCV. Ці бібліотеки надають потужні інструменти для побудови та навчання нейронних мереж, а також для обробки та аналізу зображень.

Загальна популярність та активна спільнота:

Python є однією з найпопулярніших мов програмування, що має велику та активну спільноту розробників. Це означає, що знайти відповіді на питання та рішення проблем, з якими можна зіткнутися під час розробки програм нейронних мереж, стає набагато простіше. Існує безліч онлайн-ресурсів, форумів, блогів та

посібників, які допоможуть вам зрозуміти та використовувати Python для розробки програм розпізнавання зображень.

Екосистема інструментів:

Python має широку екосистему інструментів, що сприяють розробці програм нейронних мереж. Наприклад, Jupyter Notebook - інтерактивне середовище програмування, дозволяє зручно експериментувати з кодом та візуалізувати результати. Крім того, інтегровані середовища розробки (IDE) такі як PyCharm та Visual Studio Code забезпечують зручне редагування, налагодження та профілювання коду.

Інтеграція з іншими мовами та системами:

Python може легко інтегруватися з іншими мовами програмування, такими як C ++ або Java, що дозволяє використовувати швидкі алгоритми обробки зображень, реалізовані на інших мовах, разом з високорівневим Python-кодом. Крім того, Python має доступ до багатьох системних бібліотек, що дозволяє легко взаємодіяти з операційною системою, мережевими протоколами та іншими системними ресурсами.

У підсумку, мова Python є оптимальним вибором для розробки програм нейронних мереж розпізнавання зображень, завдяки своїй простоті, зручності використання, багатому вибору бібліотек та фреймворків, активній спільноті розробників та широкій екосистемі інструментів. Використовуючи Python для таких завдань, ви отримуєте швидку і продуктивну розробку програм, що володіють високою точністю та ефективністю при розпізнаванні зображень.

3.2 Архітектура нейронної мережі для розпізнавання рукописних цифр

Для прискорення роботи нейронної мережі використовується метод батчнормалізації (Batch Normalization), запропонований співробітниками компанії Google в 2015 році. Іноді зустрічається переклад цього терміна «пакетна нормалізація». Цей метод передбачає додавання додаткового шару, який дозволяє знизити спотворення даних шляхом їх нормалізації. Цей метод дозволяє скоротити час навчання за рахунок використання більш високого значення параметра швидкості навчання (learning rate), а також знизити перенавчання.

Архітектура генератора представлена на рисунку 3.1.

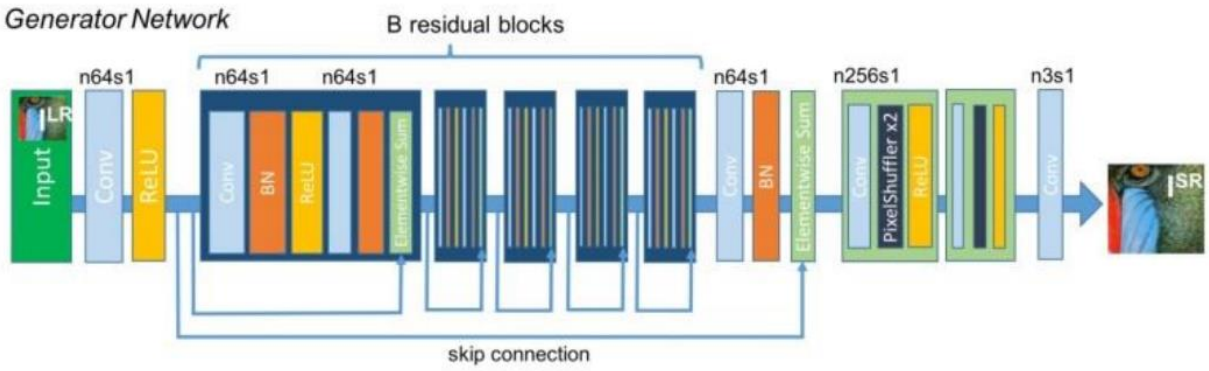

Рисунок 3.1 – Архітектура генератора

На рисунку 3.1 зображені наступні елементи:

- Input I LR – зображення низької роздільної здатності, яке подається на вхід генератора;

- Conv – згортковий шар з n картами ознак (feature maps) і кроком (stride), рівним s;

- ReLU – активаційна функція Parametric ReLU;

- BN – Batch Normalization, Батч-нормалізація;

- skip connection – додаткові зв'язку;

- B residual blocks – залишкові блоки;

- Elementwise sum – поелементно сума;

- PixelShuffler x2 – субпиксельной згорткові шари;

- ISR – зображення високої роздільної здатності, отримане в результаті роботи нейронної мережі.

Щоб відрізнити зображення з високою роздільною здатністю IHR від згенерованих зображень ISR, здійснюється навчання дискримінатора (D). На вхід дискримінатора в випадковому порядку подаються або зображення IHR і ISR. В результаті отримуємо 0 в разі, якщо зображення було розпізнано як ISR і 1 в разі, якщо розпізнано як IHR.

Архітектура дискримінатора зображена на рисунку 3.2.

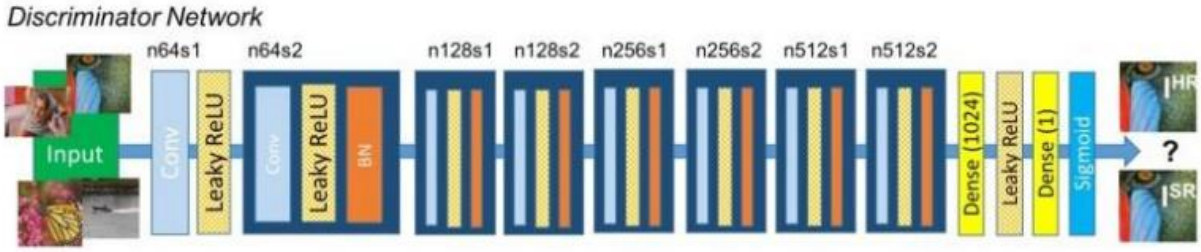

Рисунок 3.2 – Архітектура дискримінатора

На рисунку 3.2 зображені наступні елементи:

- Input – вхідні дані, зображення з високою роздільною здатністю і зображення, створені генератором;

- Conv – згортковий шар з n картами ознак (feature maps) і кроком (stride), рівним s;

- Leaky ReLU – активаційна функція Leaky ReLU;

- BN – Batch Normalization, Батчнормалізація;

- Dense – повнозв'язний шар;

- Sigmoid – активаційна функція сигмоїда.

IHR, ISR – зображення високої професійності і зображення низької роздільної здатності відповідно; певний дискримінатором клас зображення – справжнє зображення з високою роздільною здатністю або створене генератором.

Для коректного порівняння якості роботи алгоритмів і нейронних мереж необхідно використовувати об'єктивні метрики. Вхідними даними для кожного методу буде служити зменшене зображення. Отримане в результаті роботи методів зображення можна порівняти з оригіналом на предмет появи розбіжностей. Тобто чим більше результат схожий на оригінал, тим вище якість роботи алгоритму.

Одними з найбільш поширених методів порівняння двох зображень є:

- метод середньоквадратичної помилки моделі (Mean Squared Error, MSE);
- пікове відношення сигналу до шуму (Peak Signal-to-Noise Ratio, PSNR);
- індекс структурної подібності (Structural Similarity, SSIM).

Спроектуємо просту повнозв'язну багатошарову модель персептрона, яка досягає точності розпізнавання близько 90%.

Отримані результати будуть використані як основа для порівняння з більш складною архітектурою згорткових нейронних мереж.

Для завдання розпізнавання рукописних символів з навчальної вибірки MNIST за допомогою багатошарового персептрона елементи матриці вхідного зображення записуються по рядках, в результаті виходить вектор x довжини 282 = 784 ознаковий опис об'єкта.

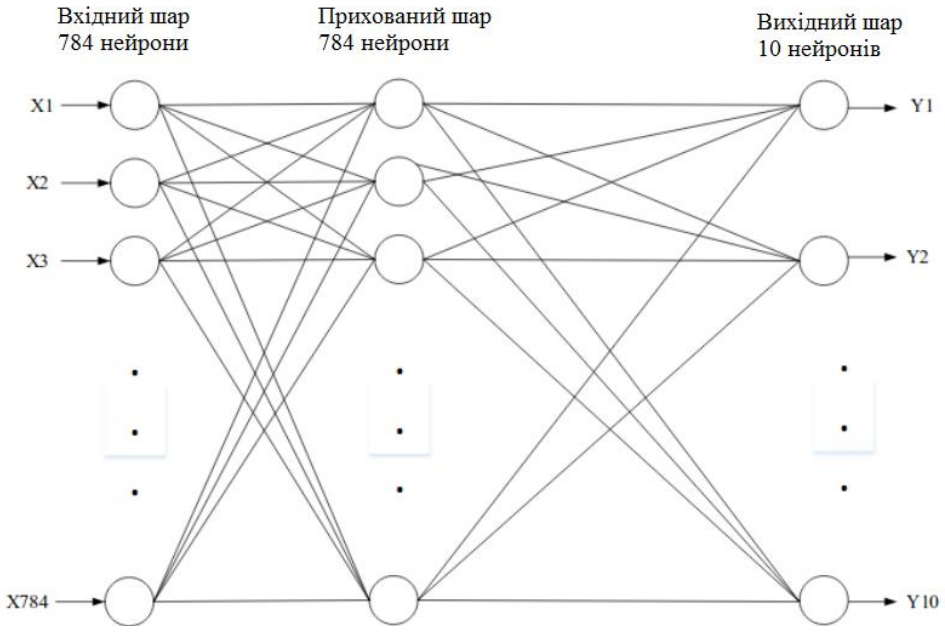

Рисунок 3.3 – Архітектура нейронної мережі для розпізнавання рукописних цифр

Згорткова нейромережа складається з декількох шарів (рис. 3.4), призначених для розпізнавання двовимірних форм з високою стійкістю до перенесення, масштабування та інших форм спотворення. Основні складові згорткової нейромережі наступні:

1. Вилучення ознак – кожен нейрон пов'язані з локальної рецептивної областю попереднього шару.

2. Набори особливостей (канали) – кожен шар згорткової нейромережі складається з кількох наборів особливостей, у яких нейрони мають однаковий набір терезів.

3. Сабсемплінг – кожен шар нейромережі виконує усереднення показників попереднього шару, зменшуючи розмірність шару нейромережі, тобто кількість параметрів.

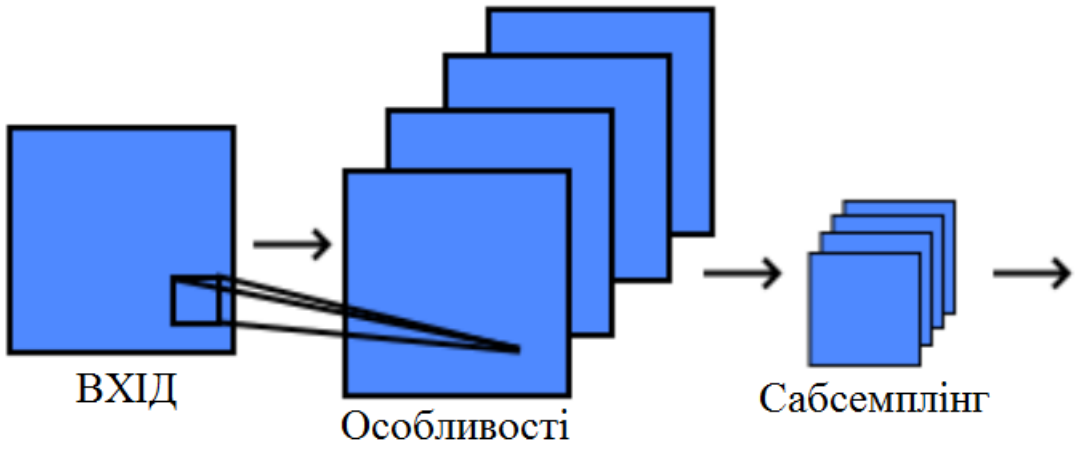

Рисунок 3.4 – Шари згорткової нейромережі

Перший шар виконує згортку. Кожен нейрон зосереджується на певній ділянці, яка надходить на вхід нейрона. Виходом кожного нейрона буде сума добутків ваг на значення вхідного сигналу із зазначеної області із застосуванням функції активації.

Нейрони одного каналу мають однакову матрицю ваги. Розмір вихідної області називається ядром згортки. Також важливими параметрами є крок згортки та набивання. Крок згортки визначає наскільки зрушуватиметься

область. Набивання збільшує розмірність вихідного зображення, розширюючи його нульовими значеннями з обох боків. На рис. 3.5 показаний процес застосування операції згортки до вихідного зображення розміром 5x5 пікселів для ядра згортки з розміром 3x3, доповненням 1 і кроком рівним 1.

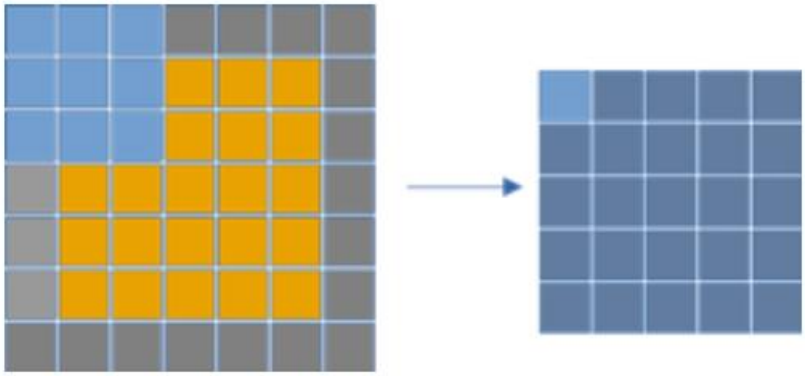

Рисунок 3.5 – Згортка для зображення 5x5 з ядром 3x3, набивкою – 1 та кроком – 1

Після процедури отримання ознак здійснюється Сабсемплінг. Початковий шар поділяється на невеликі області певного розміру. До кожної області застосовується функція пулінгу (pooling), яка скорочує розмір вихідної області. Найчастіше використовуються функції максимуму (max pooling) чи середнього (average pooling).Приклад застосування перетворення max pooling із розміром ядра 2x2 показаний на рис. 3.6.

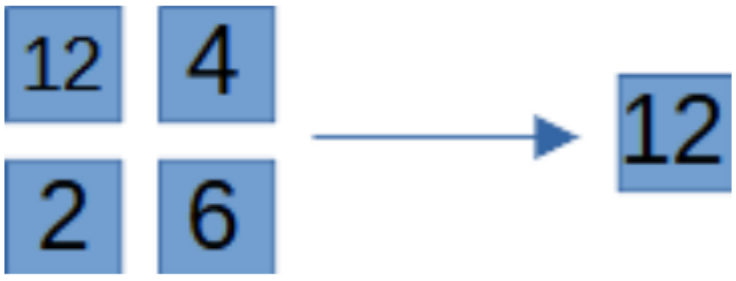

Рисунок 3.6 – Процедура Max Pooling із ядром 2x2

3.3 Тренування нейронної мережі

Нейронні мережі навчаються найшвидше на найбільш непередбачуваних прикладах з навчальної вибірки. Тому краще вибирати на кожній ітерації найменш знайомий системі приклад. Зауважимо, що це стосується лише стохастичного навчання, оскільки пакетного порядок, у якому пред'являються приклади, неважливий. Звичайно, немає простого способу зрозуміти, які навчальні приклади несуть у собі більшу кількість інформації, але існує можливість, яка використовує ту ж ідею - вибирати як наступний навчальний приклад той, що належить до іншого класу. Навчальні приклади з одного класу найчастіше містять схожу інформацію.

Персептрон змодельований наступним чином: вхідний шар являє собою набір з нейронів у кількості, що відповідає розмірності одновимірного вектора, отриманого з послідовності пікселів зображення. Перший прихований шар складається з 2048 нейронів, другий - з 1024 з функціями активації ReLU. Вихідний шар має 10 нейронів за кількістю класів зображення (цифри від 0 до 9). У останньому шарі використовується функція активації SoftMax.

Згорткова нейронна мережа (CHC) дозволяє виділити найважливіші як пікселі зображення, а й більш абстрактні структури, такі як лінії чи області і має складнішу структуру. У прикладі буде використано мережу з наступною архітектурою:

перший шар CHC – згортковий з фільтром 3 на 3 пікселя з додаванням порожнього рядка та стовпця, завдяки чому кінцевий розмір зображення співпадатиме з початковим.

Зображення розбивається на 32 карти ознак, далі відбувається операція MaxPooling, з матрицею 2 на 2 пікселя, яка дозволяє виявити найзначніші пікселі в кожній області розмірністю 2 на 2. Таким чином, зображення стискається в 2 рази. Наступним етапом наведені операції повторюються ще раз, але з використанням 64 фільтрів на згортковому шарі. У результаті отримане зображення і переведене в одномірний вектор, подається на вхід персептрону з 1

прихованим шаром з 128 нейронів і вихідний має також 10 нейронів - за кількістю класів зображень.

У всіх моделях використовувалася функція помилки "категоріальна кроссентропія", оптимізатор навчання Адама. Підготовка набору даних Для коректної обробки масиву даних необхідно нормалізувати. Набір даних MNIST містить 50 000 зображень у градаціях сірого розмірів 28 на 28 пікселів. Один канал кольору означає характеристику пікселя як інтенсивність кольору, 0 (чорний) до 255 (білий). Тому доречно розділити значення пікселя на 255 діапазон значень буде в межах від 0 до 1.

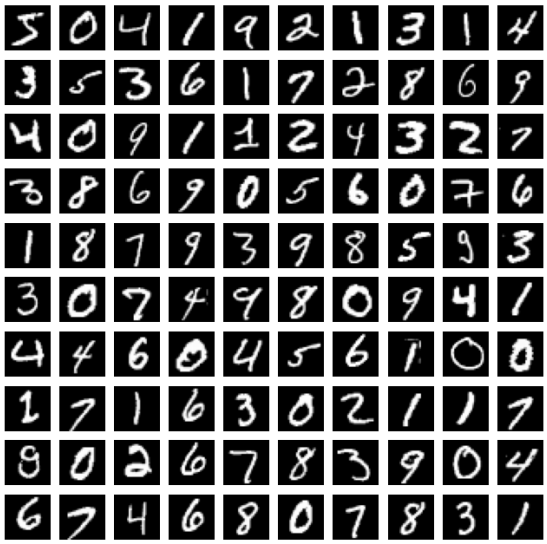

Рисунок 3.9 – Приклад рукописних цифр MNIST

Дані про правильні відповіді вибірки були нормалізовані за допомогою технології «One Hot Encoding» і перетворені на вектор виду [1,0,0,0,0,0,0,0,0,0], де порядковий номер одиниці означає клас зображення

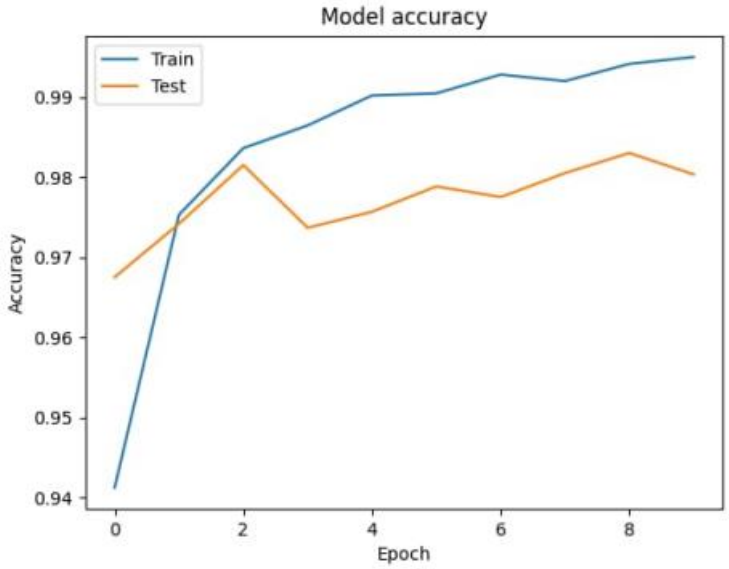

Рисунок 3.7 – Графік точності навчання персептрону

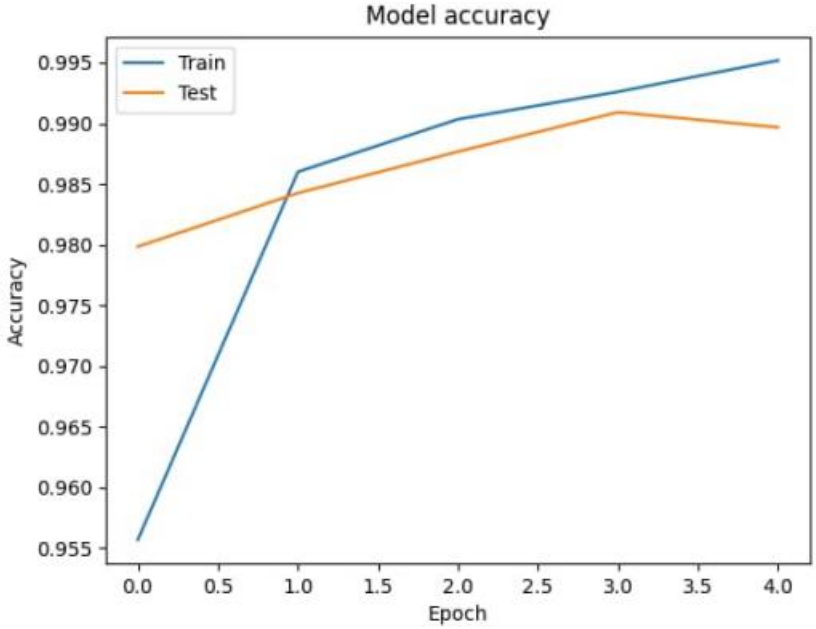

Рисунок 3.8 – Графік точності навчання СНР

Отримано порівняльну таблицю ефективності навчання нейронних мереж.

СР і завдання	Характеристика	Кількість епох	Точність навчальної\ тестової вибірки, %	Значення функції помилки в навчальній/ тестової вибірки	Час на навчання, с
ПНР MNIST		5	99,5\98	0,016\0.068	113
СНР MNIST		5	99,5 \ 98,9	0.015\0.034	121

Таблиця - Порівняння ефективності моделей СР

З таблиці, можна дійти висновку, що найкращою моделлю у цій задачі виявилася згорткова нейронна мережу, зі значенням точності на тестової вибірці 98,9. Мінусом цієї моделі виявляється час навчання, на 5 ітерацій було витрачено 121 секунда (значення округлені до цілих).

Найгіршою у задачі класифікації рукописних цифр виявилася модель персептрона з більшим часом навчання (113 секунд), що пов'язано з великою кількістю нейронів у прихованих шарах, та найгіршим значенням на тестовому наборі даних. PHC має найменший розкид у значеннях навчальної та тестової вибірки.

Для розпізнавання рукописних символів було створено дві нейромережі з різними архітектурами.

3.3.1 Модель багатошарового перцептрона

Дана модель складається з трьох повнозв'язних верств, архітектура якої представлена на рис. 3.9. На вхід нейромережі надходить зображення розміром 28×28. Далі йдуть три пов'язані шари з 784 нейронів і наприкінці нейромережа за допомогою функції SoftMax обчислює ймовірності приналежності символу, який зображений на вхідному зображенні, до одного з 10 класів.

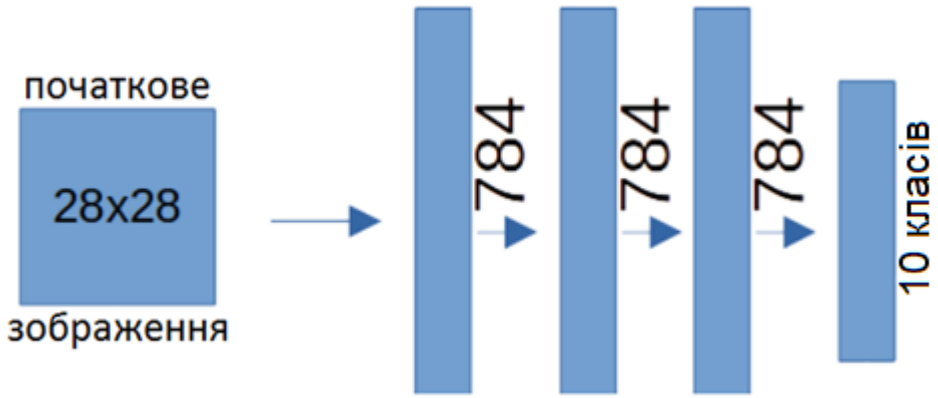

Рисунок 3.10 – Архітектура повної нейромережі для розпізнавання цифр

Для реалізації багатошарової перцептроної нейромережі був розроблено наступний код(Додаток А):

```
initializer = tf.keras.initializers.GlorotUniform(seed=42)
W1 = tf.Variable(initializer([input_size, hidden_size]))
b1 = tf.Variable(initializer([hidden_size]))
W2 = tf.Variable(initializer([hidden_size, hidden_size]))
b2 = tf.Variable(initializer([hidden_size]))
W3 = tf.Variable(initializer([hidden_size, output_size]))
b3 = tf.Variable(initializer([output_size]))
```

Рисунок 3.11 - Ініціалізація ваг та зсувів

```
for epoch in range(epochs):
    epoch_loss = 0
    num_batches = x_train.shape[0] // batch_size
    for batch in range(num_batches):
        x_batch = x_train[batch * batch_size: (batch + 1) * batch_size]
        y_batch = y_train[batch * batch_size: (batch + 1) * batch_size]
        with tf.GradientTape() as tape:
            hidden_output1 = relu(tf.matmul(x_batch, W1) + b1)
            hidden_output2 = relu(tf.matmul(hidden_output1, W2) + b2)
            logits = tf.matmul(hidden_output2, W3) + b3
            probs = softmax(logits)
            loss = tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_batch, probs))
        gradients = tape.gradient(loss, [W1, b1, W2, b2, W3, b3])
        optimizer.apply_gradients(zip(gradients, [W1, b1, W2, b2, W3, b3]))
        epoch_loss += loss
    epoch_loss /= num_batches
    print(f'Epoch {epoch + 1}/{epochs}, Loss: {epoch_loss}')
```

Рисунок 3.12 – Навчання моделі

3.3.2 Модель згорткової нейромережі

Згорткова нейромережа представлена трьома згортковими шарами з ядром згортки 5x5, кроком 1 і доповненням 2, а також трьох пов'язаних шарів. Архітектура згорткової нейромережі зображена на рис. 3.13.

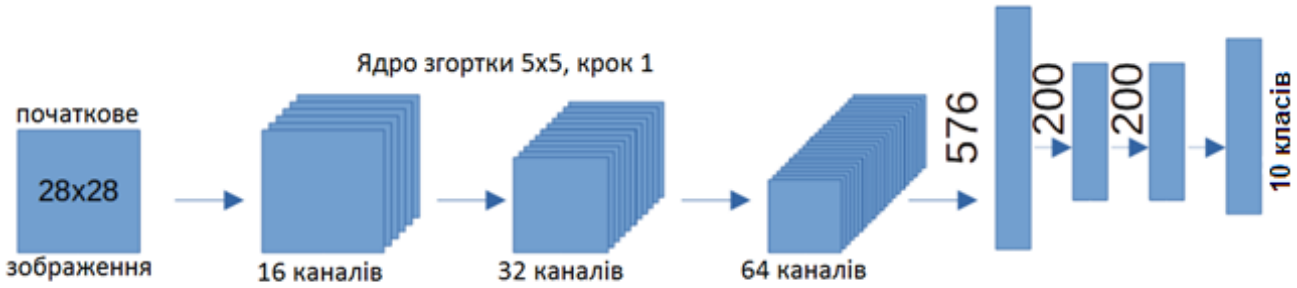

Рисунок 3.13 – Архітектура згорткової нейромережі для розпізнавання цифр

На вхід згорткової нейромережі надходить зображення розміром 28×28. Далі йдуть три згорткові шари розміром 28x28, 14x14, 7x7 нейронів які складаються з 16, 32, 64 каналів відповідно. Усі нейрони каналу мають однакові ваги. Після обробки сигналу на останньому згортковому шарі за допомогою MaxPooling виходить 64 канали нейромережі з областями розміром 3x3. Потім йдуть три повнозв'язних шари з 576, 200 і 200 нейронами. Результатом обчислень є значення ймовірності приналежності кожного з 10 класів цифр, отримані з допомогою функції Softmax.

Для реалізації згорткової нейромережі з трьома згортковими та трьома повнозв'язними шарами було розроблено наступний код(Додаток Б):

```
initializer = tf.initializers.GlorotUniform(seed=42)
W1 = tf.Variable(initializer(shape=(5, 5, 1, 16)))
b1 = tf.Variable(initializer(shape=(16,)))
W2 = tf.Variable(initializer(shape=(5, 5, 16, 32)))
b2 = tf.Variable(initializer(shape=(32,)))
W3 = tf.Variable(initializer(shape=(5, 5, 32, 64)))
b3 = tf.Variable(initializer(shape=(64,)))
W4 = tf.Variable(initializer(shape=(4 * 4 * 64, 200)))
b4 = tf.Variable(initializer(shape=(200,)))
W5 = tf.Variable(initializer(shape=(200, 200)))
b5 = tf.Variable(initializer(shape=(200,)))
W6 = tf.Variable(initializer(shape=(200, num_classes)))
b6 = tf.Variable(initializer(shape=(num_classes,)))
```

Рисунок 3.13 - Ініціалізація ваг та зсувів

```python
for epoch in range(epochs):
    epoch_loss = 0.0
    for batch in range(num_batches):
        x_batch = x_train[batch * batch_size: (batch + 1) * batch_size]
        y_batch = y_train[batch * batch_size: (batch + 1) * batch_size]
        with tf.GradientTape() as tape:
            probs = forward_pass(x_batch, training=True)
            loss = tf.reduce_mean(tf.keras.backend.categorical_crossentropy(y_batch, probs))
        gradients = tape.gradient(loss, [W1, b1, W2, b2, W3, b3, W4, b4, W5, b5, W6, b6])
        optimizer.apply_gradients(zip(gradients, [W1, b1, W2, b2, W3, b3, W4, b4, W5, b5, W6, b6]))
        epoch_loss += loss
    epoch_loss /= num_batches
    print(f'Epoch {epoch + 1}/{epochs}, Loss: {epoch_loss}')
```

Рисунок 3.14 – Навчання моделі

Середнє значення точності розпізнавання обчислювалася як сума збігів відповідно до правильних класів символів, поділена на довжину тестового набору даних. На рис. 3.15 і 3.16 зображені графіки навчання та залежності точності розпізнавання нейромереж з архітектурами загорткової нейронної мережі та багатошарового персептрону.

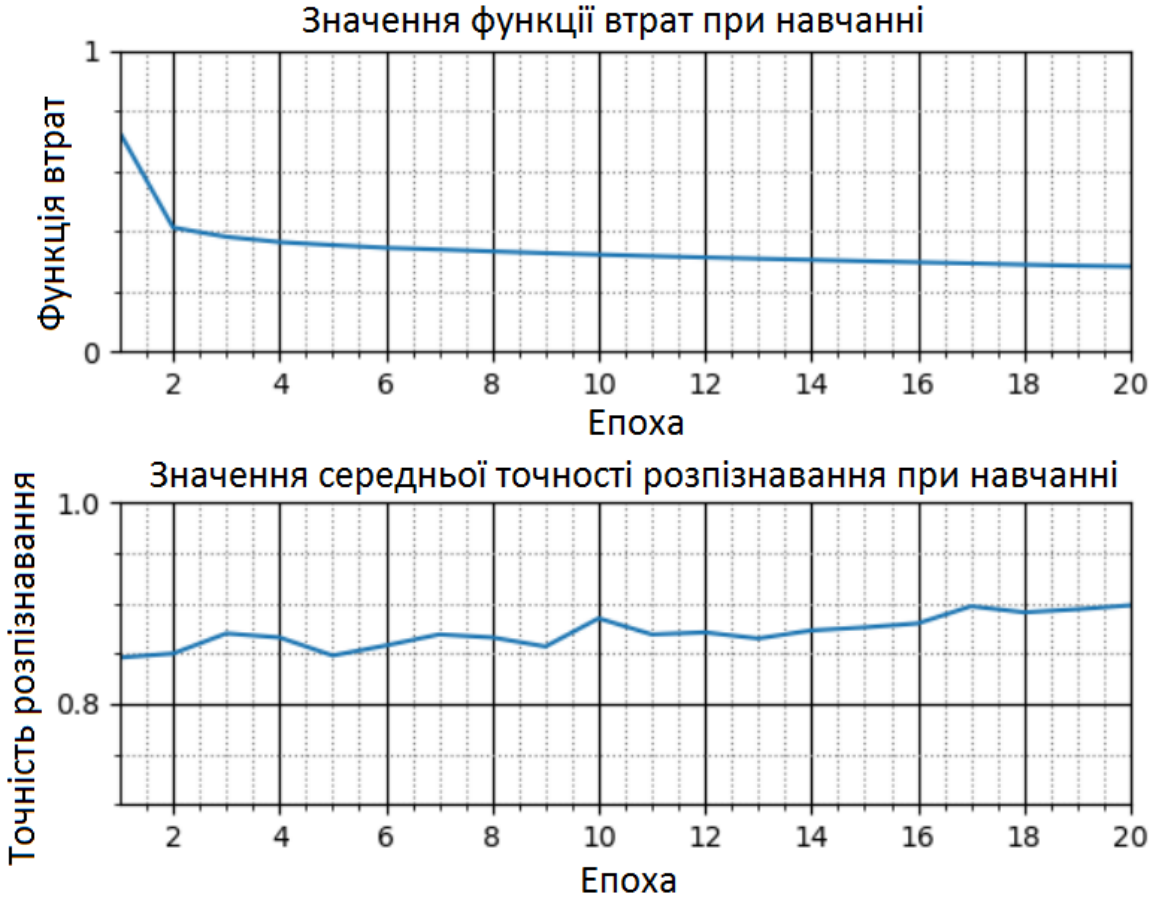

Рисунок 3.15 – Графік навчання згорткової нейромережі

У першому графіку відображено залежність функції втрат від епохи, а в другому точності розпізнавання від епохи. Тестування проводилося до 20-ти епох.

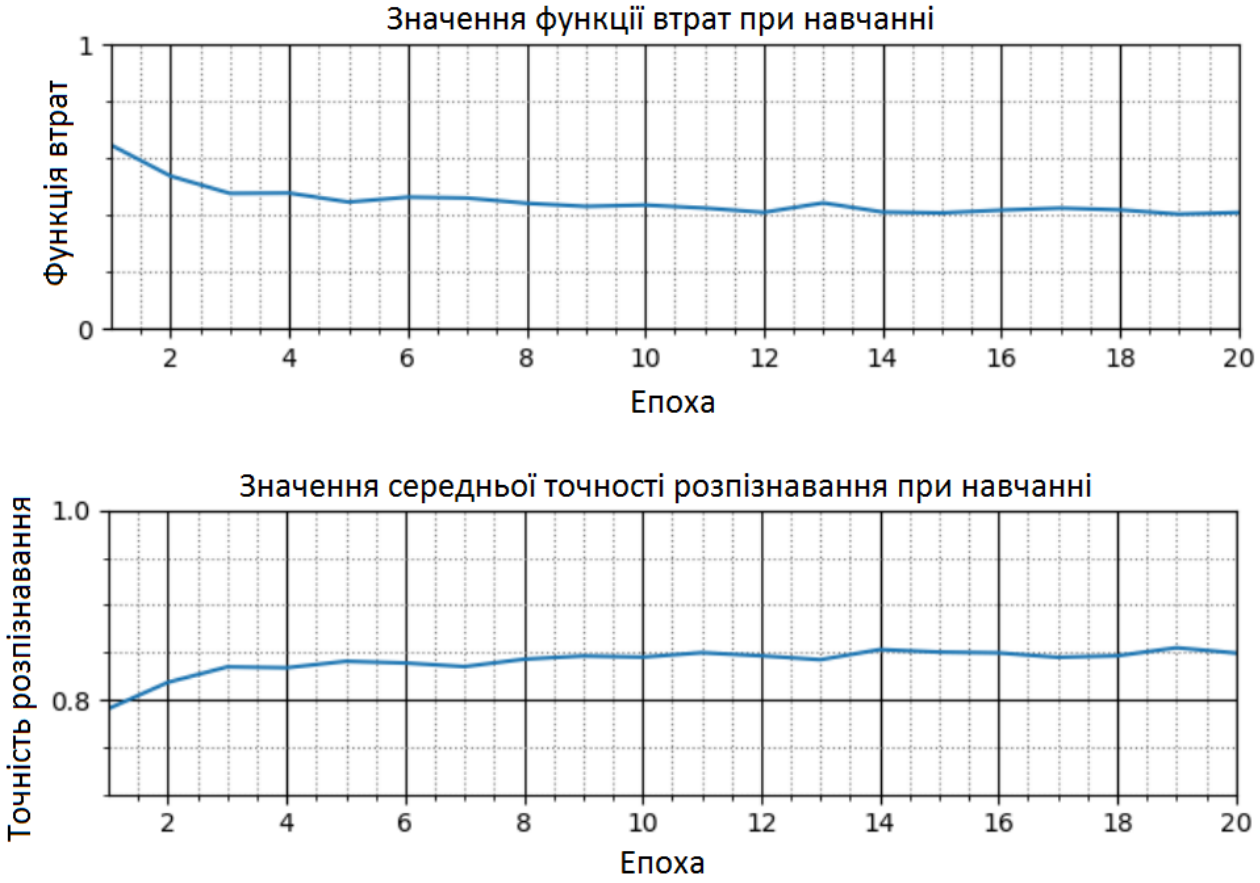

Рисунок 3.16 – Графік навчання тришарової повнозв'язної нейромережі

З отриманих графіків випливає, що багатошарова згортка нейромережа показує більш стабільні результати, а також швидше досягає непоганого значення середньої точності розпізнавання - 89,8% на навчальних прикладах. Нейромережа, що складається з трьох повнозв'язних шарів (персептрон), працює менш стабільно і досягає середньої точності - 84,9% на навчальному наборі даних.

Таким чином вибір згорткової нейромережі є кращим за точністю розпізнавання.

3.4 Тестування

Головне вікно програми розпізнавання цифр, яка використовує багатошаровий парсептрон, складається з поля для формування зовнішнього вигляду символу, результату розпізнавання цифри та точності розпізнавання:

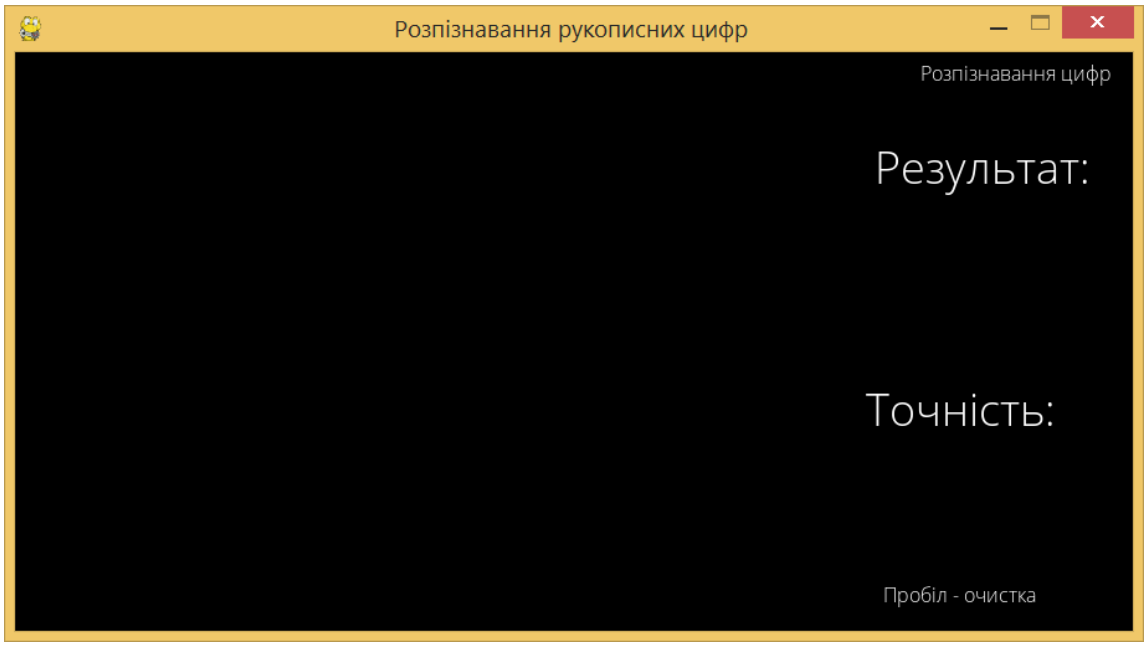

Рисунок 3.20 – Стартовий вид вікна інтерфейсного додатку

Для формування символу слід натиснути на ліву кнопку миші та сформувати образ цифри, статус якої зміниться з чорного кольору на білий, для того щоб очистити полотно використовується клавіша пробіл:

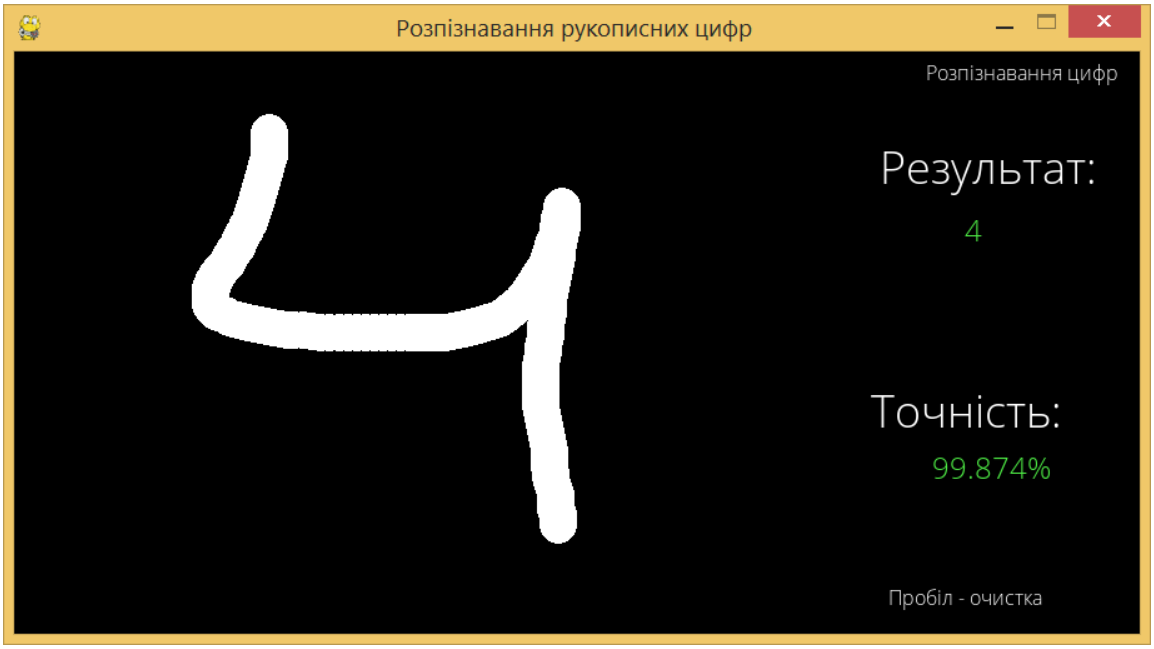

Рисунок 3.21 – Завантаження образу рукописної цифри

Після формування символу активується підсистема розпізнавання, результат якого виводиться в текстове поле:

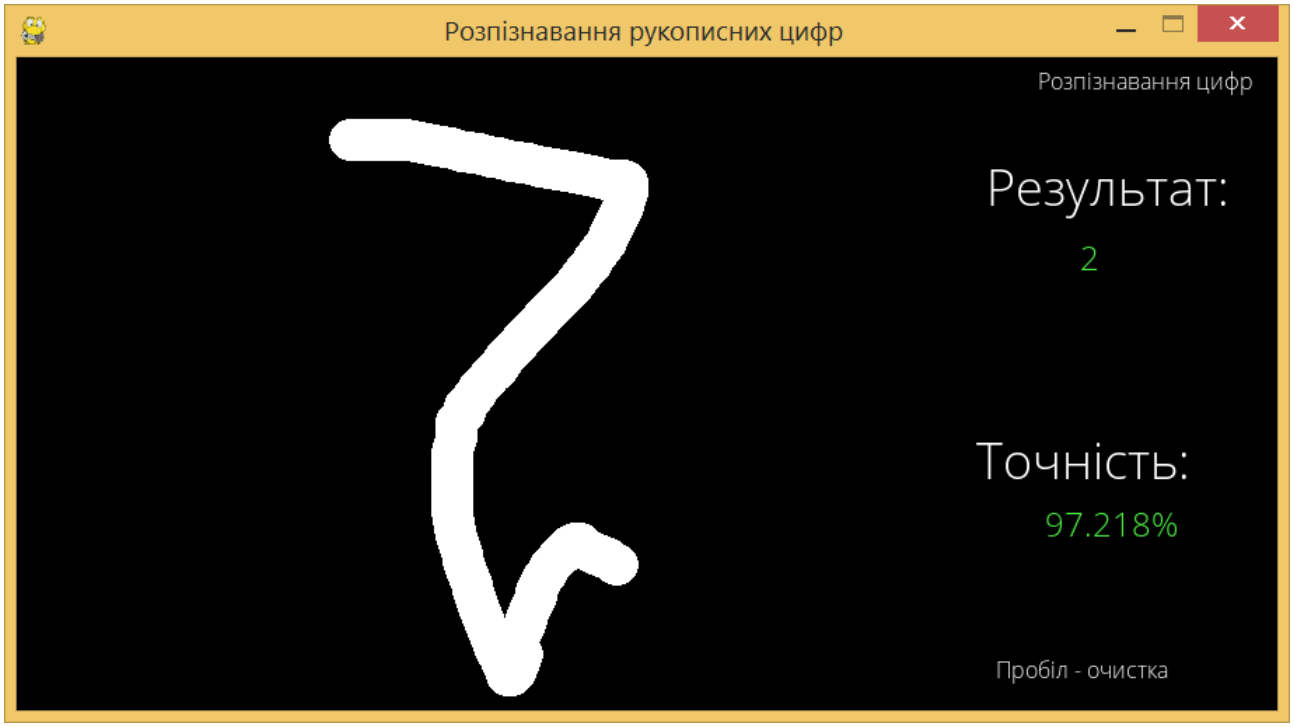

Рисунок 3.23 – Результат розпізнавання цифри 2

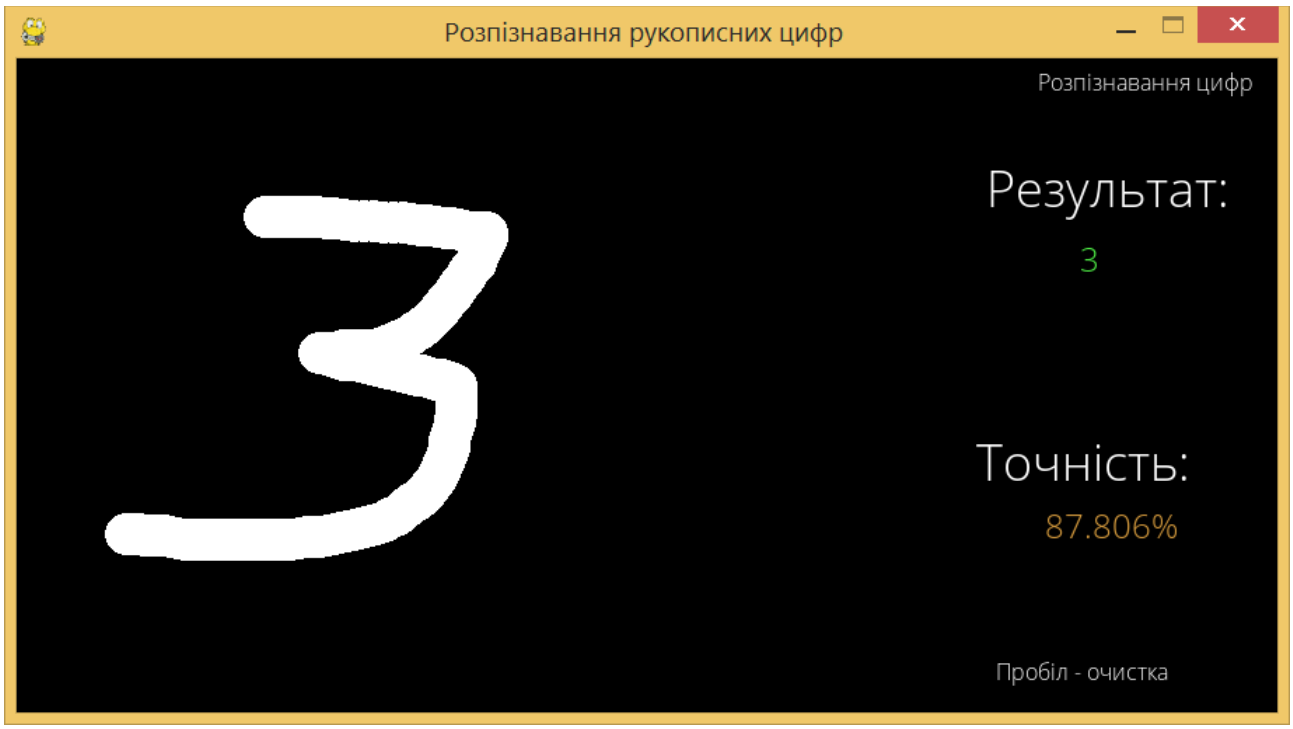

Рисунок 3.24 – Результат розпізнавання цифри 3 після навчання

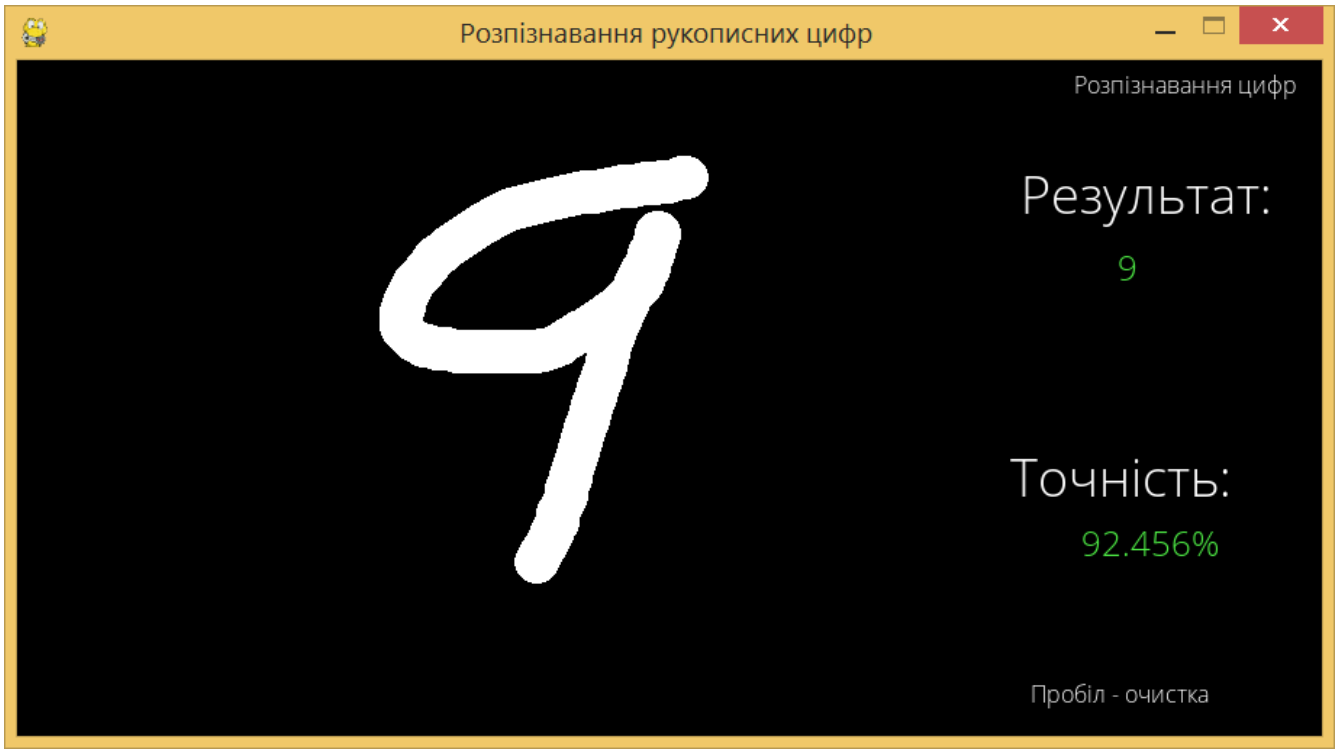

Рисунок 3.25 – Результат розпізнавання символу 9 після навчання

Головне вікно програми розпізнавання цифр, яка використовує згорткону нейрону мережу, складається з поля для формування зовнішнього вигляду символу, результату розпізнавання цифри, точності розпізнавання, клавішей для початку розпізнавання цифр та очистки полотна та повзунка для вибору

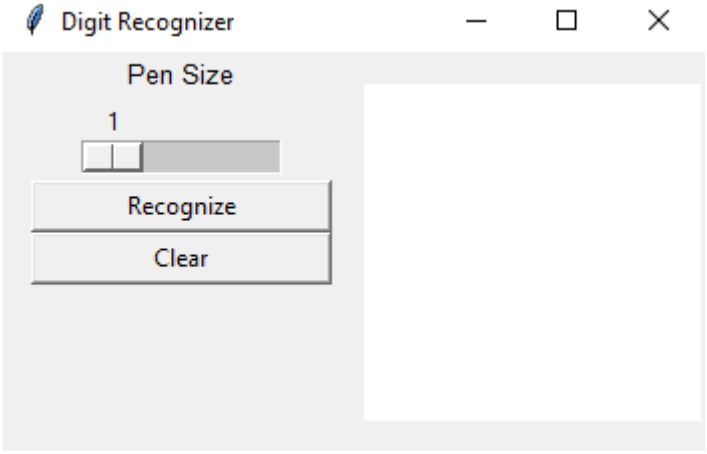

Рисунок 3.26 – Стартовий вид вікна інтерфейсного додатку

Для формування символу слід натиснути на ліву кнопку миші та сформувати образ цифри:

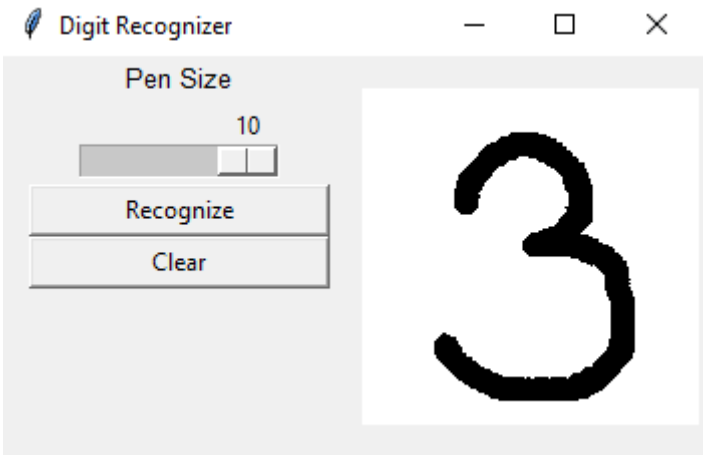

Рисунок 3.27 – Завантаження образу рукописної цифри

Після формування зображення необхідно натиснути на кнопку «Recognize» для розпізнавання написаного символу, якщо потрібно очистити зображення, використовується кнопка «Clear»:

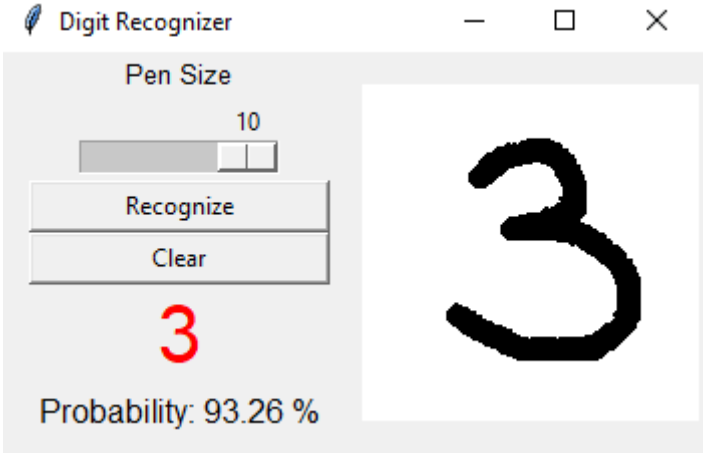

Рисунок 3.28 – Результат розпізнавання цифри 3 після навчання

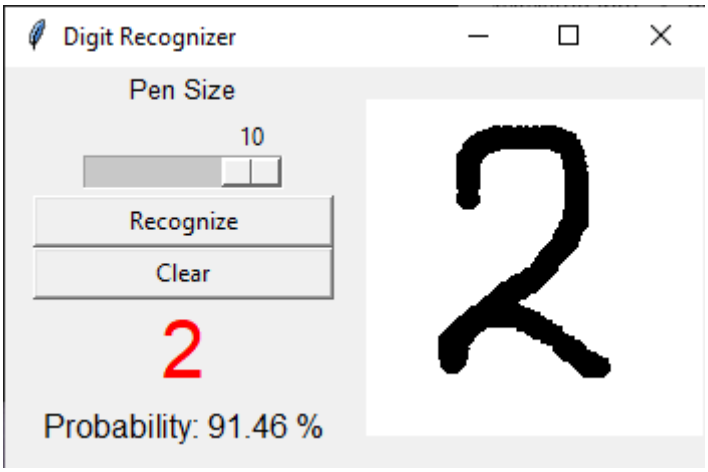

Рисунок 3.29 – Результат розпізнавання цифри 2

Висновок

На основі проведених експериментів і аналізу результатів, можна зробити висновок, що згорткова нейромережа є кращим варіантом порівняно з багатошаровим перцептроном для задачі класифікації рукописних цифр.. Вона демонструє більш стабільні результати і досягає вищої середньої точності розпізнавання 98.9% на навчальному наборі даних. Згорткова нейромережа також має високу точність 89.8% на тестовій вибірці даних, що підтверджує її ефективність.

З іншого боку, персептрон, який складається з трьох повнозв'язних шарів, показує менш стабільні результати 98% і досягає нижчої середньої точності 84.9% розпізнавання на навчальному наборі даних. Час навчання такої моделі складає 113 що є нищим ніж згорткова, що може бути доцільним у деяких випадках.

Отже, висновок полягає в тому, що згорткова нейромережа є оптимальним варіантом для класифікації рукописних цифр з точки зору стабільності, точності і ефективності.

Список використаної літератури

1. Boser, B., Guyon I., Vapnik, V. A training algorithm for optimal margin classifiers. In D. Haussler, editor, proceedings of the 5th Annual ACM Workshop on COLT, pp. 144-152, Pittsburgh, 1992.
2. Breiman, L. Bagging predictors. Machine Learning, Vol. 24 (2), pp. 123- 140, 1996.
3. Shoghian, Sh., Kouzehgar, M. A Comparison among Wolf Pack Search and Four other Optimization Algorithms. World Academy of Science, Engineering & Technology, Vol. 6, Issue 72, pp. 418-423, 2012.
4. Zahadat, P., Schmickl, T. Wolfpack-inspired evolutionary algorithm and a reaction-diffusion-based controller are used for pattern formation. In proceedings of the 2014 Conference on Genetic and Evolutionary Computation (GECCO‘2014), pp. 241-248, 2014.
5. Lia, C.-M., Duc, Y.-C., Wua, J.-X., Lind, C.-H., Hoe, Y.-R., Lina, Y., Chen, T. Synchronizing chaotification with support vector machine and wolf pack search algorithm for estimation of peripheral vascular occlusion in diabetes mellitus. Biomedical Signal Processing and Control, Vol. 9, pp. 45-55, 2014.
6. Антоненко В. М. Сучасні інформаційні системи і технології: управління знаннями : навч. Посібник / В. М. Антоненко, С. Д. Мамченко, Ю. В. Рогушина. – Ірпінь : Нац. університет ДПС України, 2016. – 212 с.
7. Андреас М. Введення в машинне навчання за допомогою Python. Керівництво для фахівців по роботі з даними / Мюллер Андреас // М .: Альфакнига, 2017. - 487 с.
8. Бенгфорт Б. Прикладний аналіз текстових даних на Python. Машинне навчання та створення додатків обробки природної мови / Б. Бенгфорт // СПб .: Питер, 2019. - 368 с.
9. Годун В.М. Інформаційні системи і технології в статистиці: навч. посіб. / В.М. Годун, Н.С. Орленко, М. А. Сендзюк; за ред. В.Ф. Ситника. – К.: КНЕУ, 2003. – 267 с.

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

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

12. Плас Д. Python для складних завдань. Наука про дані і машинне навчання. Керівництво / Плас Джейк Вандер // К .: ВМС, 2018. - 759 с.

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

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

Додаток А

```
import numpy as np
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 28 * 28).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28 * 28).astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

input_size = 28 * 28
hidden_size = 512
output_size = 10
learning_rate = 0.001
epochs = 20
batch_size = 128

initializer = tf.keras.initializers.GlorotUniform(seed=42)
W1 = tf.Variable(initializer([input_size, hidden_size]))
b1 = tf.Variable(initializer([hidden_size]))
W2 = tf.Variable(initializer([hidden_size, hidden_size]))
b2 = tf.Variable(initializer([hidden_size]))
W3 = tf.Variable(initializer([hidden_size, output_size]))
b3 = tf.Variable(initializer([output_size]))
```

```
def relu(x):
    return tf.maximum(0, x)

def softmax(x):
    exps = tf.exp(x - tf.reduce_max(x, axis=1, keepdims=True))
    return exps / tf.reduce_sum(exps, axis=1, keepdims=True)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(hidden_size, activation='relu', input_shape=(input_size,)),
    tf.keras.layers.Dense(hidden_size, activation='relu'),
    tf.keras.layers.Dense(output_size, activation='softmax')
])

optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer,
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

```
for epoch in range(epochs):
    epoch_loss = 0
    num_batches = x_train.shape[0] // batch_size

    for batch in range(num_batches):
        x_batch = x_train[batch * batch_size: (batch + 1) * batch_size]
        y_batch = y_train[batch * batch_size: (batch + 1) * batch_size]

        with tf.GradientTape() as tape:
            hidden_output1 = relu(tf.matmul(x_batch, W1) + b1)
            hidden_output2 = relu(tf.matmul(hidden_output1, W2) + b2)
            logits = tf.matmul(hidden_output2, W3) + b3
            probs = softmax(logits)

            loss = tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_batch, probs))

        gradients = tape.gradient(loss, [W1, b1, W2, b2, W3, b3])
        optimizer.apply_gradients(zip(gradients, [W1, b1, W2, b2, W3, b3]))

        epoch_loss += loss

    epoch_loss /= num_batches
    print(f'Epoch {epoch + 1}/{epochs}, Loss: {epoch_loss}')
```

```
hidden_output1 = relu(tf.matmul(x_test, W1) + b1)
hidden_output2 = relu(tf.matmul(hidden_output1, W2) + b2)
logits = tf.matmul(hidden_output2, W3) + b3
probs = softmax(logits)
predictions = np.argmax(probs, axis=1)
accuracy = np.mean(np.equal(predictions, np.argmax(y_test, axis=1)))
print('Test accuracy:', accuracy)

model.save('parseptron')
```

```
import pygame
import pygame.freetype
import numpy as np
from tensorflow import keras
from PIL import Image
import os

WHITE = (255, 255, 255)
BLACK = (  0,   0,   0)
LIME  = ( 79, 235,  70)
ORANGE = (209, 152, 61)
RED =   ( 219, 64,  64)

pygame.freetype.init()
FONT_BIG = pygame.freetype.Font("Resources/OpenSans-Light.ttf", 36)
FONT_SMALL = pygame.freetype.Font("Resources/OpenSans-Light.ttf", 24)
FONT_SUPA_SMALL = pygame.freetype.Font("Resources/OpenSans-Light.ttf", 16)

model = keras.models.load_model("parseptron")
print("[+] Loaded model -> parseptron")

pygame.init()

screen = pygame.display.set_mode((900, 466))
screen.fill(BLACK)

pygame.display.set_caption("Розпізнавання рукописних цифр ")
```

```
if os.path.isdir("img_tmp") == False:
    os.mkdir("img_tmp")

def determine_color(acc):
    acc = int(acc)

    if acc >= 90:
        return LIME
    elif acc < 90 and acc >= 80:
        return ORANGE
    else:
        return RED

def process_data():
    rect = pygame.Rect((0,0), (634, 466))
    sub = screen.subsurface(rect)
    pygame.image.save(sub, "img_tmp/screenshot.jpg")

    img = Image.open("img_tmp/screenshot.jpg")
    img = img.resize((28,28))

    array = np.asarray(img)
    array = array / 255.0
    array = array[:, :, 0]
    array = (np.expand_dims(array, 0))

    img.save("img_tmp/postprocess.jpg")
    return array
```

```
mainloop = True
continous_circle = False

FONT_SUPA_SMALL.render_to(screen, (730, 10), "Розпізнавання цифр", WHITE)
FONT_BIG.render_to(screen, (695, 80), "Результат:", WHITE)
FONT_BIG.render_to(screen, (685, 275), "Точність:", WHITE)
FONT_SUPA_SMALL.render_to(screen, (700, 430), "Пробіл - очистка", WHITE)

while mainloop:
    mouse_x, mouse_y = pygame.mouse.get_pos()

    if continous_circle == True:
        if (mouse_x >= 10 and mouse_x <= 634) and (mouse_y >= 10 and mouse_y <= 466):
            pygame.draw.circle(screen, WHITE, (mouse_x, mouse_y), 15)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            mainloop = False
        elif event.type == pygame.MOUSEBUTTONDOWN:

            if (mouse_x >= 10 and mouse_x <= 634) and (mouse_y >= 10 and mouse_y <= 466):
                pygame.draw.circle(screen, WHITE, (mouse_x, mouse_y), 15)
                continous_circle = True
```

```
        elif event.type == pygame.MOUSEBUTTONUP:
            continous_circle = False

            predictions = model.predict(process_data())
            result = str(np.argmax(predictions[0]))
            accuracy = np.max(predictions[0])
            accuracy = round(float(accuracy), 5)
            accuracy = accuracy * 100

            pygame.draw.rect(screen, BLACK, ((760, 135), (100, 50)))
            pygame.draw.rect(screen, BLACK, ((735, 325), (200, 50)))

            FONT_SMALL.render_to(screen, (760, 135), result, LIME)
            FONT_SMALL.render_to(screen, (735, 325), str(accuracy) + "%", determine_color(accuracy))

        elif event.type == pygame.KEYDOWN:

            if event.key == pygame.K_ESCAPE:
                mainloop = False

            if event.key == pygame.K_SPACE:
                pygame.draw.rect(screen, BLACK, ((0, 0), (655, 466)))

    pygame.display.update()

pygame.quit()
```

Додаток Б

```
import numpy as np
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

input_shape = (28, 28, 1)
num_classes = 10
learning_rate = 0.001
epochs = 20
batch_size = 128

initializer = tf.initializers.GlorotUniform(seed=42)
W1 = tf.Variable(initializer(shape=(5, 5, 1, 16)))
b1 = tf.Variable(initializer(shape=(16,)))
W2 = tf.Variable(initializer(shape=(5, 5, 16, 32)))
b2 = tf.Variable(initializer(shape=(32,)))
W3 = tf.Variable(initializer(shape=(5, 5, 32, 64)))
b3 = tf.Variable(initializer(shape=(64,)))
W4 = tf.Variable(initializer(shape=(4 * 4 * 64, 200)))
b4 = tf.Variable(initializer(shape=(200,)))
W5 = tf.Variable(initializer(shape=(200, 200)))
b5 = tf.Variable(initializer(shape=(200,)))
W6 = tf.Variable(initializer(shape=(200, num_classes)))
b6 = tf.Variable(initializer(shape=(num_classes,)))
```

```
def relu(x):
    return tf.maximum(0, x)

def forward_pass(x, training=False):
    conv1 = tf.nn.conv2d(x, W1, strides=(1, 1), padding='VALID') + b1
    conv1 = relu(conv1)
    conv2 = tf.nn.conv2d(conv1, W2, strides=(1, 1), padding='VALID') + b2
    conv2 = relu(conv2)
    conv3 = tf.nn.conv2d(conv2, W3, strides=(1, 1), padding='VALID') + b3
    conv3 = relu(conv3)
    maxpool = tf.nn.max_pool2d(conv3, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1), padding='VALID')
    flatten = tf.reshape(maxpool, shape=(-1, 4 * 4 * 64))
    fc1 = tf.matmul(flatten, W4) + b4
    fc1 = relu(fc1)
    fc2 = tf.matmul(fc1, W5) + b5
    fc2 = relu(fc2)
    logits = tf.matmul(fc2, W6) + b6
    if training:
        probs = tf.nn.softmax(logits)
    else:
        probs = logits
    return probs

optimizer = tf.keras.optimizers.Adam(learning_rate)

num_batches = len(x_train) // batch_size
```

```
for epoch in range(epochs):
    epoch_loss = 0.0

    for batch in range(num_batches):
        x_batch = x_train[batch * batch_size: (batch + 1) * batch_size]
        y_batch = y_train[batch * batch_size: (batch + 1) * batch_size]

        with tf.GradientTape() as tape:
            probs = forward_pass(x_batch, training=True)
            print("y_batch shape:", y_batch.shape)
            print("probs shape:", probs.shape)
            loss = tf.reduce_mean(tf.keras.backend.categorical_crossentropy(y_batch, probs))

        gradients = tape.gradient(loss, [W1, b1, W2, b2, W3, b3, W4, b4, W5, b5, W6, b6])
        optimizer.apply_gradients(zip(gradients, [W1, b1, W2, b2, W3, b3, W4, b4, W5, b5, W6, b6]))

        epoch_loss += loss

    epoch_loss /= num_batches
    print(f'Epoch {epoch + 1}/{epochs}, Loss: {epoch_loss}')

probs_test = forward_pass(x_test)
predictions = np.argmax(probs_test, axis=1)
accuracy = np.mean(np.equal(predictions, np.argmax(y_test, axis=1)))
print('Test accuracy:', accuracy)
```

```
from PIL import ImageTk, Image, ImageDraw, ImageOps
import PIL
import numpy as np
from tkinter import *

width = 168
height = 168
white = (255, 255, 255)
```

```
def recognize():
    inverted_img = ImageOps.invert(image)
    grayscaled_img= inverted_img.convert('L')
    resized_img=grayscaled_img.resize((28,28),PIL.Image.ANTIALIAS)
    data = np.asarray(resized_img)
    inputs=np.reshape(data,(1,784)).T

    with open('weights/fst_w.csv','r') as f:
        fst_w= np.loadtxt(f, delimiter=",")
    with open('weights/snd_w.csv','r') as f:
        snd_w= np.loadtxt(f, delimiter=",")

    x1=np.dot(fst_w, inputs)
    y1=1/(1+np.exp(-x1))

    x2=np.dot(snd_w, y1)
    y2=1/(1+np.exp(-x2))

    probability=round( ((np.max(y2)/np.sum(y2))*100), 2)

    print(np.argmax(y2),' ',probability,'%')

    lbl1['text']=np.argmax(y2),
    lbl2['text']='Probability:',probability,'%'
```

```
def paint(event):
    x1, y1 = (event.x - 1), (event.y - 1)
    x2, y2 = (event.x + 1), (event.y + 1)
    cv.create_oval(x1, y1, x2, y2, fill="black",width=penSize_slider.get())
    draw.line([x1, y1, x2, y2],fill="black",width=penSize_slider.get())

def clear():
    cv.delete("all")
    draw.rectangle((0, 0, 168, 168), fill=(255, 255, 255, 255))

root = Tk()
root.title("Digit Recognizer")

cv = Canvas(root, width=width, height=height, bg='white')
cv.pack()

image = PIL.Image.new("RGB", (width, height), white)
draw = ImageDraw.Draw(image)

cv.pack(side=RIGHT)
cv.bind("<B1-Motion>", paint)
```

```
button=Button(text="Recognize",command=recognize,width=20)
button2=Button(text="Clear",command=clear, width = 20)
lbl0=Label(text="Pen Size",font="Arial 10",width=15)
lbl1=Label(text=" ",font="Arial 30",fg="red")
lbl2=Label(text=" ",font="Arial 12",width=15)

lbl0.pack()
penSize_slider = Scale(from_= 1, to = 10,orient=HORIZONTAL)
penSize_slider.pack()

button.pack()
button2.pack()

lbl1.pack()
lbl2.pack()

root.minsize(350, 200)
root.maxsize(350, 200)

root.mainloop()
```