МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра комп'ютерної інженерії та інноваційних технологій

Пояснювальна записка

до кваліфікаційної роботи
другого (магістерського) рівня

на тему **ВИКОРИСТАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ**

Виконав: студент 2 курсу, групи ІКК-2.1
спеціальності
123 Комп'ютерна інженерія

Баришовець А.О.

Керівник Русу О.П.

Рецензент Григор'єва Т.І

Одеса – 2023

ДОВІДКА

кафедри КІ та ІТ про виконану магістерську роботу

студента 2 курсу, ФКПІ та КН групи ІКК-2.1

Баришовця Артема Олександровича

(прізвище, ім'я та по-батькові)

на тему: Використання згорткових нейронних мереж для розпізнавання зображень

Висновок нормоконтролера пояснювальна записка до кваліфікаційної роботи виконана з незначними порушеннями ДСТУ. Оформлено згідно вимог внутрішнього положення КПІ.

Нормоконтролер к.т.н., доцент каф. КІ та ІТ _____ В.В. Педаш

(науковий ступінь, вчене звання) (підпис, дата) (і. б. прізвище)

Висновок відповідального за наявність плагіату згідно з сертифікатом ID унікальність роботи підтверджено.

Відповідальна особа к.т.н., доц. каф. КІ та ІТ _____ В.В. Педаш

(науковий ступінь, вчене звання) (підпис, дата) (і. б. прізвище)

Попередній захист магістерської роботи

студ. Баришовця А.О. проведено «12» грудня 2023 р.

(прізвище і.б.)

Висновки Розділи МР відповідають завданню, усі пункти завдання виконано. В магістерській роботі проведено аналіз нейронних мереж для розпізнавання зображень виконано програмну реалізацію, котра включає в себе створення інтерфейсу і розробження згорткової нейронної мережі. Текстова та графічна робота виконані згідно вимог до її оформлення. МР до захисту спеціальності може бути рекомендована до захисту ЕК.

Члени комісії _____ к.т.н., доц. Ілона Л.Г.

(підпис) (науковий ступінь, вчене звання, прізвище і.б.)

_____ к.т.н., доц. Педаш В.В.

(підпис) (науковий ступінь, вчене звання, прізвище і.б.)

_____ вик. каф. КІ та ІТ Шевель О.В.

(підпис) (науковий ступінь, вчене звання, прізвище і.б.)

МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра комп'ютерної інженерії та інноваційних технологій
Освітній рівень магістр
Галузь знань 12 Інформаційні технології
Спеціальність 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
КІ а ІТ
к.т.н., доц.
Л.Г. Йона
«25» 09 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ
Баришовцю Артему Олександровичу

1. Тема роботи: Використання згорткових нейронних мереж для розпізнавання зображень
керівник роботи Русу О.П., к.т.н. доцент, кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «25» 09 2023 року № 1953

2. Строк подання студентом роботи 11.12.2023р.

3. Вихідні дані до роботи

1. Використовувати згорткові нейроні мережі
2. Розмір первинного зображення не менше 20x20 пікселів
3. Розпізнавати не менше 5 класів зображень

4. Зміст розрахунково-пояснювальної записки

Розділ 1: Види нейронних мереж
Розділ 2: Принципи роботи згорткових нейронних мереж
Розділ 3: Практична реалізація згорткової нейронної мережі для розпізнавання зображення

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень)
Слайд 2 – Основні характеристики роботи
Слайд 3 – Види нейронних мереж
Слайд 4 – Архітектура згорткової нейронної мережі
Слайд 5 – Вибір праткичної реалізації
Слайд 6 – Зовнішній вигляд програмного забезпечення
Слайд 7 – Датасет Fashion MNIST
Слайд 8 – Результати дослідження
Слайд 9 – Висновки та Рекомендації

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.09.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз видів нейронних мереж	25.09.2023-11.10.2023	Вик
2	Аналіз згорткових нейронних мереж	12.10.2023-30.10.2023	Вик
3	Практична реалізація та результати досліджень	31.10.2023-27.11.2023	Вик
4	Висновки та рекомендації	28.11.2023-30.11.2023	Вик
5	Перелік джерел посилання	01.12.2023-04.12.2023	Вик
6	Додатки А	05.12.2023-06.12.2023	Вик
7	Додаток Б	07.12.2023-08.12.2023	Вик

Студент _____ А.О. Баришовець

Керівник роботи _____ О.П. Русу

ВІДГУК КЕРІВНИКА

на кваліфікаційну роботу другого (магістерського) рівня
здобувача Баришовця Артема Олександровича
зі спеціальності 123 Комп'ютерна інженерія
на тему: «Використання згорткових нейронних мереж для розпізнавання зображень»

Кваліфікаційна робота Баришовця А.О. присвячена актуальним питанням розвитку систем комп'ютерного зору та обробки зображень. Результатом роботи є програмний модуль, в якому реалізована згорткова нейронна мережа, спроможна розпізнавати різноманітні зображення, за допомогою якої можна практично досліджувати засоби штучного інтелекту з метою визначення їх особливостей та обмежень.

Здобувач Баришовець А.О. самостійно провів усі необхідні дослідження, сформував тестову та навчальну вибірки зображень, розробив програмне забезпечення та з'ясував оптимальні налаштування, що забезпечили найбільшу імовірність розпізнавання зображень, відповідно до умов, зазначених у завданні на роботу. Під час роботи здобувач Баришовець А.О. дотримувався графіку консультації та календарного плану, а також враховував усі рекомендації, що надавалися йому протягом роботи. Поставлене завдання виконано у повному обсязі. Пояснювальна записка та демонстраційні аркуші виконано із дотриманням усіх необхідних вимог.

Під час виконання кваліфікаційної роботи здобувач Баришовець А.О. розібрався з усіма поставленими питаннями та показав уміння користуватись інформаційними джерелами, ставити та розв'язувати дослідницькі задачі.

Кваліфікаційна робота відповідає вимогам до кваліфікаційних робіт другого (магістерського) рівня та заслуговує оцінки «відмінно».

Здобувач Баришовець А.О. заслуговує присвоєння кваліфікації магістр з комп'ютерної інженерії за заявленою спеціальністю 123 «Комп'ютерна інженерія».

Керівник
доцент кафедри комп'ютерних наук,
к.т.н., О.П. Русу

РЕЦЕНЗІЯ

на кваліфікаційну роботу другого (магістерського) рівня
здобувача Баришовця Артема Олександровича
зі спеціальності 123 Комп'ютерна інженерія
на тему: «Використання згорткових нейронних мереж для розпізнавання зображень»

Кваліфікаційна робота здобувача Баришовця А.О. присвячена актуальним питанням розвитку систем для аналізу та розпізнавання зображень. У роботі створена та протестована штучна згорткова нейронна мережа, працездатність якої перевірена на тестовій вибірці зразків одягу та взуття. Особливістю даної роботи є можливість її практичного використання під час підготовки здобувачів освіти в галузі інформаційних технологій, оскільки розроблене програмне забезпечення реалізовано у вигляді програмного модуля для віртуальної лабораторії.

Під час обговорення роботи здобувач Баришовець А.О. показав достатню теоретичну підготовку та відповів на усі поставлені запитання. Кваліфікаційна робота відповідає отриманому завданню. В роботі використано усі вихідні дані. Текст роботи послідовний та зрозумілий, оформлення пояснювальної записки та демонстраційних аркушів якісне.

До недоліків роботи слід віднести:

– недостатню увагу приділено питанням вибору кількості згорткових шарів та шарів пулінгу;

– недостатню увагу приділено критеріям оцінки ступеню навчання нейронної мережі – за яких умов нейронна мережа вважається недонавченою, навченою або перенавченою.

Але названі недоліки не знижують цінності виконаної роботи.

Кваліфікаційна робота Баришовця А.О. відповідає вимогам до випускних кваліфікаційних робіт здобувачів другого (магістерського) рівня та заслуговує оцінки «відмінно».

Здобувач Баришовець А.О. заслуговує присвоєння кваліфікації магістр з комп'ютерної інженерії за заявленою спеціальністю 123 «Комп'ютерна інженерія».

Рецензент
Завідувач кафедри
інформаційних технологій
к.т.н., доцент Т.І. Григор'єва

UNICHECK
by Turnitin

Ім'я користувача:
Анна Серединко

Дата перевірки:
15.12.2023 17:49:29 MSK

Дата звіту:
17.12.2023 19:03:28 MSK

ID перевірки:
1016010073

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100001433

Назва документа: Баришовець

Кількість сторінок: 87 Кількість слів: 10770 Кількість символів: 90106 Розмір файлу: 2.35 MB ID файлу: 1015695766

28.3% Схожість

Найбільша схожість: 21.9% з джерелом з Бібліотеки (ID файлу: 1015695767)

7.71% Джерела з Інтернету	935	Сторінка 89
23.1% Джерела з Бібліотеки	37	Сторінка 94

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0% Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	2

РЕФЕРАТ

Текстова частина магістерської роботи містить 40 с., 21 рис., 12 табл., 41 джерело , 2 додатки.

Ключові слова: НЕЙРОННА МЕРЕЖА, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, ШАРИ, ДАТАСЕТ.

Об'єкт дослідження – нейронні мережі з прямим поширенням інформації для розпізнавання зображення.

Предмет дослідження – програмне забезпечення для розпізнавання зображень за допомогою згорткових нейронних мереж.

Мета дослідження – створення програмного модуля для розпізнавання зображень на основі згорткової нейронної мережі.

Методи дослідження – методи об'єктно-орієнтованого програмування, методи статистичної обробки.

В магістерській роботі проведено аналіз нейронних мереж для розпізнавання зображень. Проведено аналіз згорткових нейронних мереж для розпізнавання зображень. Виконано програмну реалізацію котра включає в себе створення інтерфейсу і розроблення згорткової нейронної мережі. Нейронна мережа виконана у вигляді готового програмного модуля, призначеного для використання в універсальній віртуальній лабораторії.

Проведено тестування працездатності розробленої нейронної мережі та оцінка ефективності її роботи.

ABSTRACT

The text part of the master's work contains 40 pages, 21 figures, 12 tables, 41 sources, 2 appendices.

Keywords: NEURAL NETWORK, CONVOLUTIONAL NEURAL NETWORK, IMAGE RECOGNITION, LAYERS, DATASET.

The object of research is neural networks for image recognition.

The subject of research is software for image recognition using convolutional neural networks.

The goal of the work is to develop a system that will recognize images using neural networks.

Research methods – analysis of neural networks, experimental choice of architecture.

In the master's thesis, an analysis of neural networks for image recognition was carried out. Convolutional neural networks for image recognition were analyzed. The software implementation of which includes the creation of an interface and the development of a convolutional neural network. The analysis of the results of the study of the effectiveness of the convolutional neural network for image recognition was carried out.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN	Згорткові нейронні мережі (Convolutional Neural Networks)
FNN	Прямого поширення (Feedforward Neural Networks)
BPNN	Зворотного поширення (Backpropagation Neural Networks)
RNN	Рекурентні нейронні мережі (Recurrent Neural Networks)
LSTM	Мережі довготривалої короткотривалої пам'яті (Long Short Term Memory Networks)

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN	Згорткові нейронні мережі (Convolutional Neural Networks)
FNN	Прямого поширення (Feedforward Neural Networks)
BPNN	Зворотнього поширення (Backpropagation Neural Networks)
RNN	Рекурентні нейронні мережі (Recurrent Neural Networks)
LSTM	Мережі довготривалої короткотривалої пам'яті (Long Short Term Memory Networks)

ВСТУП

У сучасному інформаційному суспільстві, в якому величезна кількість даних генерується щодня, важливо розвивати та вдосконалювати методи обробки інформації, зокрема, у сфері комп'ютерного зору та розпізнавання об'єктів на зображеннях. Нейронні мережі є потужний інструмент для вирішення завдань з обробки зображень, і вони стали ключовим елементом у багатьох сферах, включаючи медицину, автомобільну промисловість, безпеку та багато інших.

Магістерська робота присвячена вивченню та аналізу застосування згорткових нейронних мереж для розпізнавання зображень. Ця тема стає все більш актуальною в контексті стрімкого розвитку технологій та зростання обсягів великомасштабних даних. У роботі буде розглянуто теоретичні засади функціонування CNN, їхні переваги порівняно з іншими методами обробки зображень та конкретні приклади їхнього успішного застосування.

Основні завдання роботи включають вивчення архітектури CNN, їхніх основних принципів роботи та оптимізації. Детально буде розглянуто процес тренування CNN на великих наборах даних, а також підходи до покращення їхньої ефективності та точності. Крім того, в роботі буде проведено порівняльний аналіз різних архітектур CNN та їхніх застосувань у конкретних областях.

Результати цього дослідження можуть стати основою для розробки та вдосконалення систем розпізнавання зображень, що використовують CNN, та знайти практичне застосування у різноманітних галузях науки та техніки.

1 ВИДИ НЕЙРОННИХ МЕРЕЖ

1.1 Застосування нейронних мереж у вирішенні задач розпізнавання зображень

Нейронні мережі визнані як потужний інструмент у вирішенні складних завдань розпізнавання зображень. Їхній успіх базується на вдосконаленій здатності розуміти та інтерпретувати вміст зображень, що дозволяє вирішувати різноманітні завдання у сфері обробки зображень та комп'ютерного зору. Нижче представлено розгорнутий огляд застосувань згорткових нейронних мереж у вирішенні задач розпізнавання зображень:

- класифікація об'єктів: дозволяють ефективно вирішувати завдання класифікації об'єктів на зображеннях. Це застосування знаходить широкий застосування у сферах від розпізнавання облич до класифікації тварин, транспортних засобів та інших об'єктів у реальному часі;
- локалізація об'єктів: згорткові нейронні мережі дозволяють точно визначати положення та межі об'єктів на зображеннях. Це є важливим для завдань, де потрібно не лише визначити категорію об'єкта, але й точно вказати його місцезнаходження;
- сегментація зображень: використовуються для сегментації зображень, тобто розподілу зображення на окремі сегменти або області. Це важливо для виявлення та аналізу структур та зон на зображеннях, таких як органи на медичних зображеннях або об'єкти на сценах відеоспостереження;
- розпізнавання обличь: згорткові нейронні мережі виявилися дуже ефективними у завданнях розпізнавання обличь. Вони здатні автоматично вивчати унікальні особливості обличь та робити точні прогнози, що важливо у багатьох застосуваннях, включаючи системи безпеки та автентифікації;
- виявлення аномалій та артефактів: Застосування CNN дозволяє виявляти аномалії та артефакти на зображеннях, що може бути корисним у медичній діагностиці [13], виробничому контролі, а також у виявленні небезпечних ситуацій на відеоспостереженні;

- підвищення якості зображень: Згорткові нейронні мережі використовуються для завдань удосконалення якості зображень, включаючи зменшення шуму та усунення артефактів, що може бути важливим для покращення роботи систем в реальних умовах.

Загальною тенденцією є те, що застосування нейронних мереж у вирішенні задач розпізнавання зображень розширюється і поглиблюється в різних галузях, що робить цю тему дослідження надзвичайно актуальною.

1.2 Нейронні мережі прямого поширення

Нейронні мережі прямого поширення [1] представляє собою тип нейронні мережі, який входить до сімейства глибокого навчання. Вона характеризується архітектурою, в якій дані рухаються від входу до виходу без циклічних зв'язків, на рисунку 1.1 представлено приклад архітектури. Мережа складається з вхідного шару для прийому даних, прихованих шарів для внутрішнього представлення та вихідного шару для генерації прогнозів

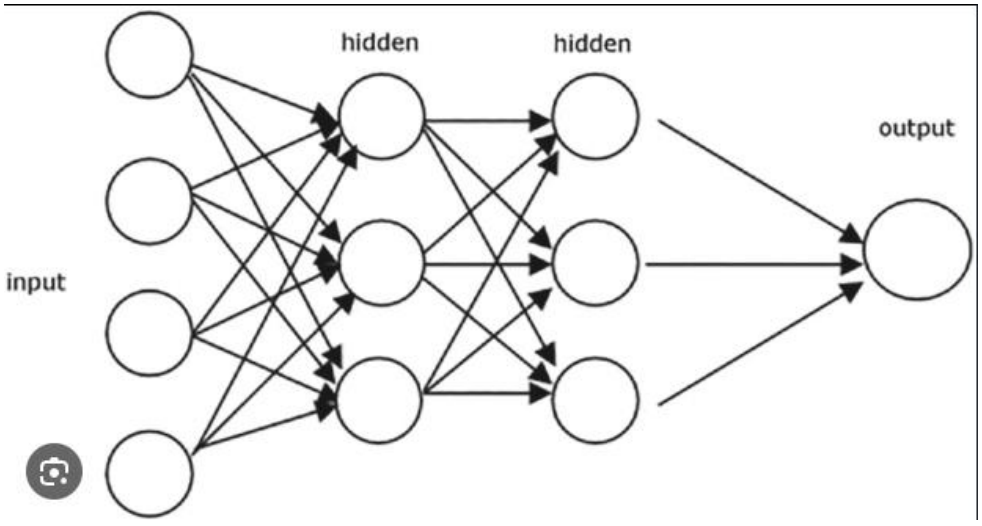


Рисунок 1.1 – Приклад архітектури нейронні мережі прямого поширення

У нейронів є ваги, які визначають їх вплив, і зсуви для керування цим впливом. Функції активації, такі як ReLU, Sigmoid і Tanh, вводять нелінійність для вивчення складних залежностей. Функція втрат визначає різницю між прогнозами

та фактичними значеннями, використовуючи, наприклад, середньоквадратичну помилку або крос-ентропію.

Для навчання мережі часто використовується алгоритм зворотного поширення помилки та градієнтний спуск для оновлення ваг і зсувів. Важливими перевагами FNN є простота реалізації та ефективність для різних задач. Однак вона може бути обмеженою для деяких складних завдань порівняно з більш складними архітектурами, такими як рекурентні нейронні мережі (RNN) або трансформери.

Переваги та недоліки:

- переваги: проста реалізація, ефективна для багатьох задач, здатна вивчати складні залежності;
- недоліки: може виявитися недостатньою для деяких складних завдань, особливо у порівнянні з більш складними архітектурами, такими як рекурентні нейронні мережі (RNN) або трансформери.

1.3 Нейронні мережі зворотного поширення

Мережі зворотного поширення [2] є підтип нейронних мереж прямого поширення і є ключовим компонентом у глибокому навчанні. Архітектура BPNN аналогічна до прямих поширення, з використанням вхідного, прихованого і вихідного шарів, на рисунку 1.2 зображено архітектура BPNN. Вони вирізняються застосуванням алгоритму зворотного поширення помилки для корекції параметрів мережі.

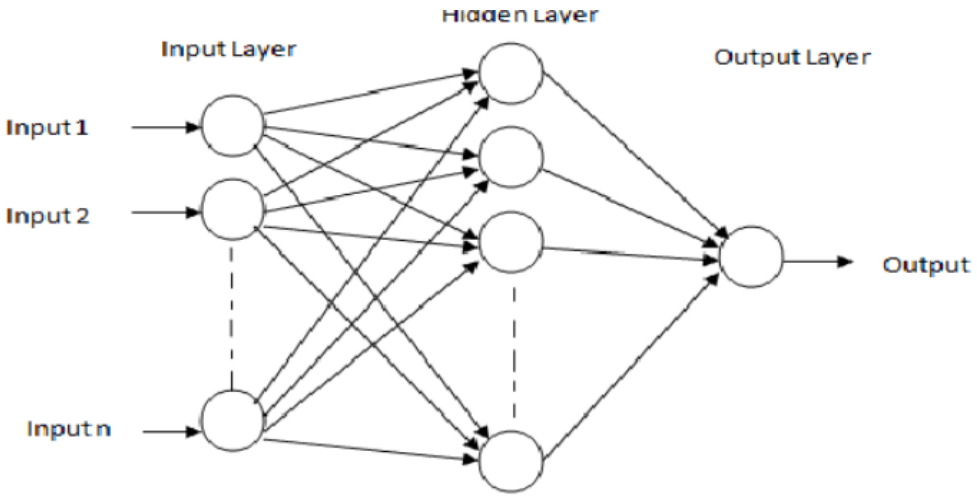


Рисунок 1.2 – Приклад архітектури нейронні мережі зворотнього поширення

Основним принципом BPNN є здатність обчислювати градієнти функції втрат відносно ваг і зсувів у зворотному напрямку. Ці градієнти використовуються для оновлення параметрів мережі, що дозволяє ефективно мінімізувати функцію втрат.

Так само, як і в прямих поширеннях, у BPNN використовуються функції активації для введення нелінійності в модель. Ці мережі застосовуються для різних задач, таких як класифікація та регресія, і здатні вивчати складні залежності завдяки механізму зворотного поширення.

Незважаючи на їх ефективність, BPNN можуть стикатися з труднощами вивчення довготривалих залежностей, порівняно з іншими архітектурами, такими як RNN. Проте, вони залишаються важливим елементом глибокого навчання та знаходять застосування у різних областях машинного навчання.

1.4 Рекурентні нейронні мережі

Рекурентні нейронні мережі [3] є класом нейронні мережі, які відзначаються здатністю моделювати послідовності та робити прогнози, враховуючи контекст з попередніх частин послідовності. Основною відмінністю RNN від прямого

поширення є використання циклічних зв'язків, що дозволяють передавати інформацію між часовими кроками.

Архітектура RNN включає вхідний шар, приховані шари та вихідний шар, архітектура представлена на рисунку 1.3. Ці мережі ідеально підходять для обробки послідовностей різної довжини, оскільки вони можуть адаптуватися до контексту в залежності від поданої вхідної інформації.

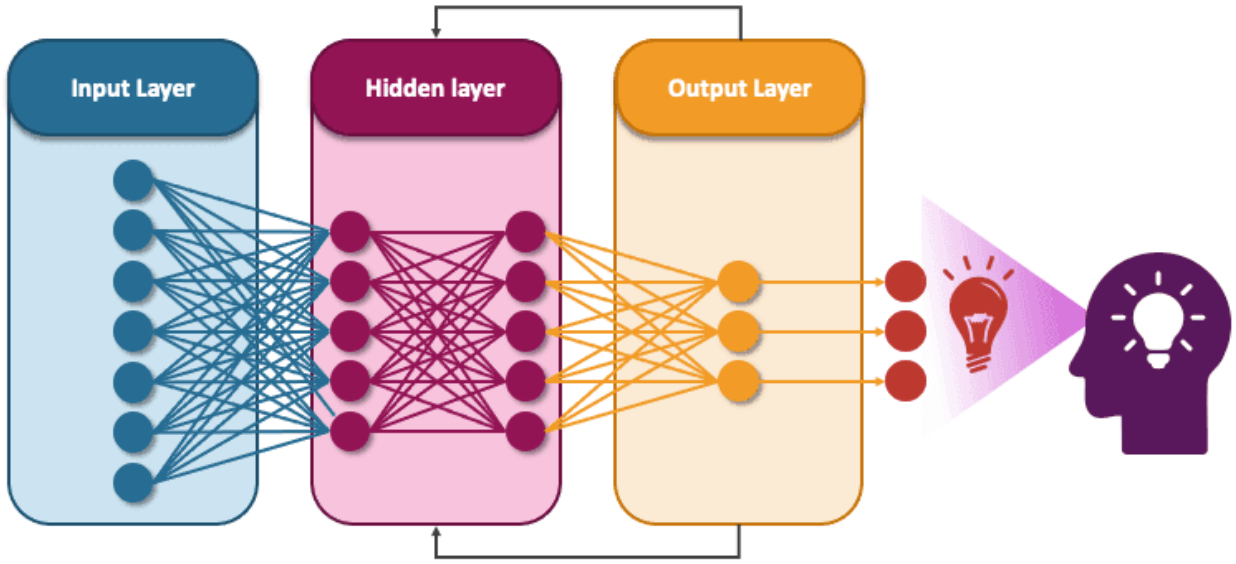


Рисунок 1.3 – Приклад архітектури рекурентної нейронної мережі

Основна властивість RNN полягає в тому, що кожен вихідний сигнал на кожному часовому кроці враховує попередні вихідні сигнали, що дозволяє зберігати пам'ять та враховувати контекст при обробці послідовностей. Проте, зворотній зв'язок в RNN може призводити до проблем вивчення довготривалих залежностей через проблему зниклих та вибухаючих градієнтів.

Додатковий розвиток RNN включає в себе варіації, такі як LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), які створені для подолання проблем пам'яті та градієнтів. Ці розширені архітектури використовуються в широкому спектрі завдань, таких як машинний переклад, розпізнавання мови та генерація тексту. RNN залишаються важливим інструментом в області обробки послідовностей в машинному навчанні.

1.5 Згорткові нейронні мережі

CNN [4] також є потужним класом архітектур глибокого навчання, які спеціально розроблені для обробки великих об'ємів даних з виокремленням просторових та ієрархічних особливостей, на рисунку 1.4 представлено архітектура згорткової нейронної мережі. Головною особливістю CNN є використання згорткових шарів для автоматизованого виявлення локальних патернів во вхідних даних.

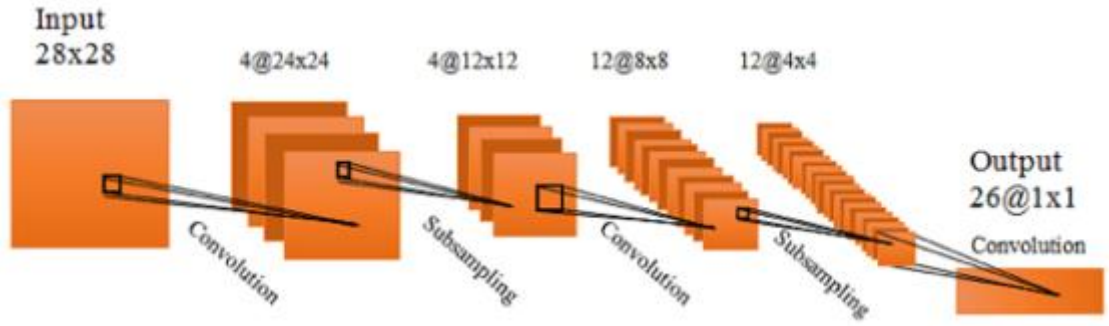


Рисунок 1.4 – Приклад архітектури згорткової нейронної мережі

Згорткові шари використовують фільтри, які сковзають по вхідних даних, виконуючи операцію згортки. Це дозволяє виявляти різні властивості, такі як границі, форми та текстури. Додатково, пулінгові шари використовуються для підвищення інваріантності до масштабу та варіантності об'єктів.

CNN показали свою ефективність у завданнях обробки зображень, таких як класифікація та виявлення об'єктів. Вони також застосовуються в інших областях, наприклад, в обробці природної мови. Для покращення навчання та зменшення кількості параметрів, можуть використовуватися додаткові шари, такі як повністю з'єднані шари або шари нормалізації.

Загальна ідея CNN полягає в тому, щоб автоматично вивчати та ієрархічно виокремлювати характеристики від вхідних даних, що робить їх відмінним інструментом для завдань, пов'язаних із зображеннями та обробкою відео.

Історія та етапи розвитку згорткових нейронних мереж відзначають захоплюючий етап в еволюції глибокого навчання та обробки зображень [17]. Ось більш детальний огляд цього шляху:

- початок розвитку: початки CNN відносяться до 1980-90-х років, коли дослідники спробували використовувати шари нейронів з локальними зв'язками для розпізнавання зображень. Однак технічні обмеження того часу та висока вартість обчислень обмежували їхнє використання;
- лейконграфова архітектура: згорткові нейронні мережі взяли свій початковий поштовх з робіт Яна Лейконграфа в 1998 році. Він вперше запропонував архітектуру CNN для класифікації рукописних цифр. Це була перша спроба використання згорткових шарів у нейронних мережах для обробки зображень;
- поширення використання: з появою потужніших та більш доступних обчислювальних ресурсів у 2010-х роках CNN стали більш популярними. Заснуванням глибокого навчання та з'явою потужних графічних процесорів, таких як GPU, дозволили великій кількості дослідників та практиків ефективно застосовувати CNN для обробки зображень;
- визнання на конкурсах: важливим моментом у розвитку CNN стало визнання їхньої ефективності на ряді важливих конкурсів з обробки зображень. Наприклад, архітектура CNN виграла конкурс ImageNet у 2012 році, позначивши початок ері CNN в сфері комп'ютерного зору;
- архітектурні інновації: у подальшому розвитку, дослідники працювали над поліпшенням архітектур CNN. Виникли інновації, такі як VGGNet, GoogLeNet (Inception), та ResNet, які додали глибину та ефективність у роботу CNN. Ці архітектури стали стандартом для багатьох задач обробки зображень;
- розширення застосувань: останнім етапом є розширення застосувань CNN за межі класифікації зображень. Вони вдало використовуються для сегментації зображень, виявлення об'єктів, аналізу відео та в багатьох інших областях, що робить їх важливим інструментом у широкому спектрі додатків.

Усі ці етапи відображають стійкий та стрімкий розвиток CNN від їхнього піднесення до популярності до сучасного використання в багатьох областях науки та технологій.

1.6 Мережі довготривалої короткотривалої пам'яті

Мережі довготривалої короткотривалої пам'яті (LSTM) [5] є розширенням RNN і призначені для вирішення проблем, пов'язаних із вивченням та зберіганням довготривалих залежностей в послідовних даних. Однією з основних особливостей LSTM є вміння зберігати та використовувати інформацію на тривалий період часу, запобігаючи проблемам зниклих та вибухаючих градієнтів, які часто виникають у звичайних RNN.

Механізм LSTMs включає в себе спеціальні структури, такі як "ворота" (gates), що дозволяють управляти потоком інформації. Вони включають в себе ворота забування (forget gate), ворота входу (input gate) та ворота виходу (output gate). Ці ворота регулюють потік інформації внутрішньою пам'яттю LSTMs, визначаючи, яка інформація зберігається, оновлюється чи видаляється.

LSTM має широкий спектр застосувань, включаючи обробку послідовностей у великому обсязі завдань. Вони ефективні в завданнях машинного перекладу, генерації тексту, розпізнавання мови та багатьох інших. Їхні унікальні властивості роблять їх потужним інструментом для моделювання складних залежностей в даних з тривалими взаємозв'язками, на рисунку 1.5 зображено приклад архітектури LSTM.

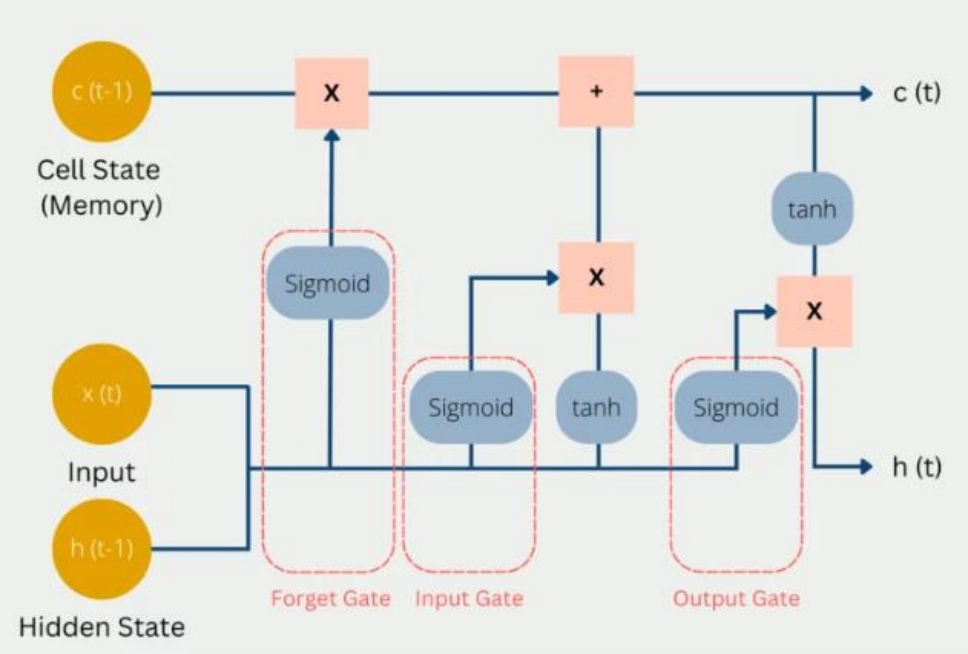


Рисунок 1.5 – Приклад архітектури мережі LSTM

1.7 Мережі глибокого навчання

Мережі глибокого навчання [6] — це клас алгоритмів машинного навчання, що використовують глибокі архітектури для автоматичного вивчення представлень даних. Основна ідея полягає в тому, щоб моделювати складні взаємозв'язки та властивості шляхом використання багатьох шарів або нейронів. Глибоке навчання відзначається здатністю ієрархічної абстракції, що дозволяє представляти дані на різних рівнях абстракції.

Ці мережі дозволяють автоматично вивчати корисні ознаки або представлення з великою кількістю даних, що робить їх ефективними для розв'язання складних завдань. Мережі глибокого навчання використовуються в різних областях, таких як комп'ютерне зору, обробка природної мови, аудіоаналіз та інше.

Однією з ключових характеристик глибокого навчання є можливість навчання на великій кількості даних без потреби вручну визначати ознаки чи шаблони. Це робить ці мережі потужними інструментами для завдань, де нюанси та складні взаємозв'язки не завжди очевидні.

Глибоке навчання включає в себе різноманітні архітектури, такі як CNN, RNN, LSTM та інші. Ці мережі сприяють значному прогресу в області штучного

інтелекту та розв'язанню завдань, які раніше вважалися складними для автоматизованого розуміння та обробки, на рисунку 1.6 зображено архітектуру мережі глибокого навчання.

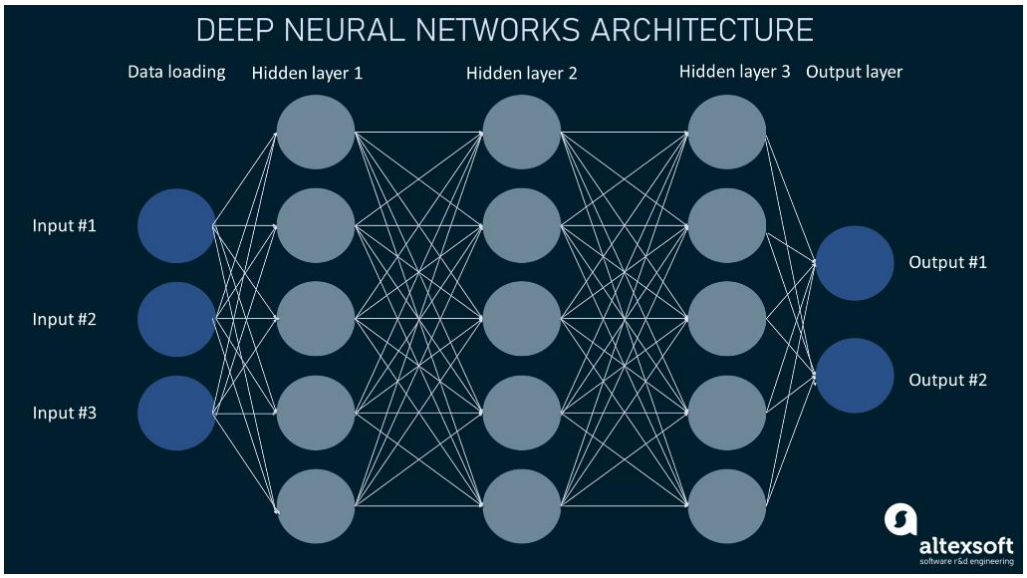


Рисунок 1.6 – Приклад архітектури мережі глибокого навчання

1.8 Рекурсивна нейронна мережа

Рекурсивна нейронна мережа [7] — це тип нейронної мережі, яка використовує рекурсивну структуру для обробки послідовностей або ієрархічних даних. Основна особливість полягає в здатності моделювати складні залежності, що виникають в ієрархічних або вкладених структурах даних.

Рекурсивні нейронні мережі взаємодіють із даними за допомогою рекурсивної структури, яка дозволяє їм ефективно враховувати контекст та взаємозв'язки між різними рівнями або частинами вхідних даних. Це може бути особливо корисно для завдань, де інформація має ієрархічний характер, таких як аналіз тексту або обробка деревоподібних структур, архітектура зображена на рисунку 1.7.

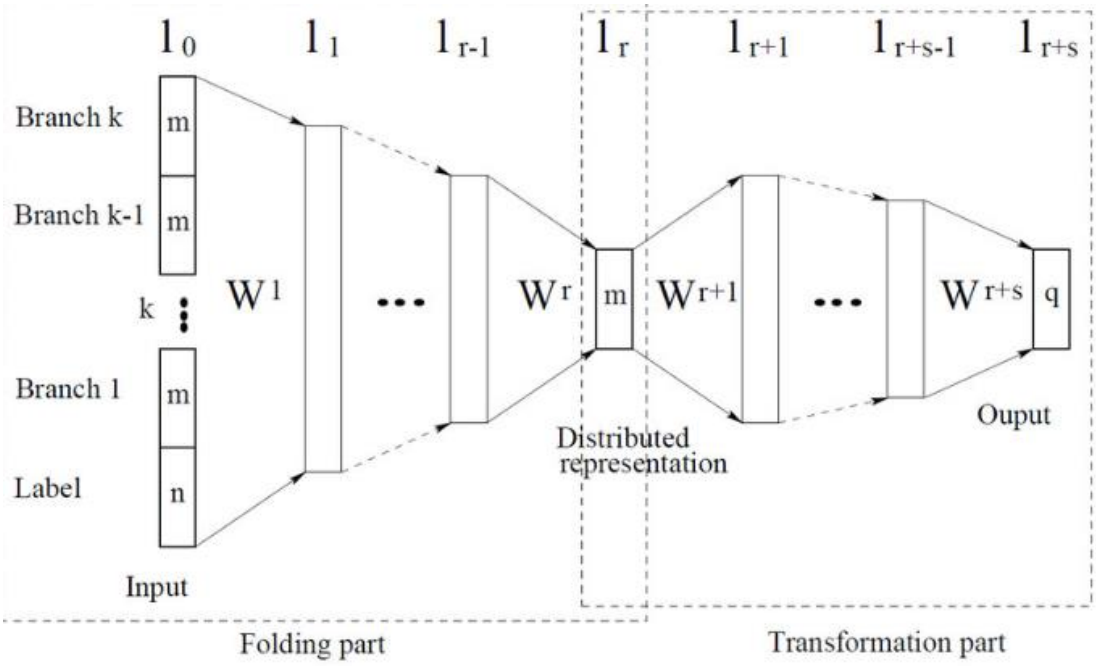


Рисунок 1.7 – Приклад архітектури рекурсивної нейронної мережі

Рекурсивні нейронні мережі можуть використовуватися в різних областях, таких як обробка природної мови, аналіз деревоподібних даних або структуровані представлення інформації. Завдяки їхній здатності ефективно враховувати ієрархічні структури, вони стають потужним інструментом для моделювання складних зв'язків в різноманітних типах даних.

1.9 Висновки до розділу

Проведено аналіз видів нейронних мереж для розпізнавання зображень з метою щоб обрати вид нейронної мережі для реалізації програмного забезпечення.

В результаті проведеного аналізу та розгляду характеристик різних типів нейронних мереж, було прийнято рішення вибрати згорткову нейронну мережу [8] для подальшої роботи. Вибір зумовлено її здатністю ефективно виявляти та розпізнавати візуальні ознаки та шаблони у зображеннях, що робить її відмінним вибором для завдань розпізнавання об'єктів.

2 ПРИНЦИПИ РОБОТИ ЗГОРТКОВИХ НЕЙРОНИХ МЕРЕЖ

Згорткові нейронні мережі є потужним інструментом у сфері розпізнавання зображень, і мають великий потенціал у майбутньому в контексті подальшого розвитку та застосування.

Одним з основних напрямків розвитку згорткових нейронні мережі є постійне покращення точності розпізнавання зображень. Завдяки постійному удосконаленню алгоритмів та збільшенню обсягу навчальних даних, майбутні згорткові нейронні мережі матимуть здатність розпізнавати зображення з ще більшою точністю та ефективністю.

Згорткові нейронні мережі [9] є нейронною мережевою моделлю, яка широко використовується для розпізнавання та класифікації зображень [10]. Вони є основним інструментом у сферах комп'ютерного зору та обробки зображень.

Згорткові нейронні мережі демонструють високу ефективність у завданнях розпізнавання зображень. Вони можуть автоматично вивчати [11] корисні ознаки зображень та робити прогнози з високою точністю. Їх використання знаходить широке застосування у сферах медицини, автоматичного водіння, відеоспостереження та багатьох інших [12].

2.1 Основні принципи роботи CNN

Згорткові нейронні мережі є архітектурою глибокого навчання, що відзначається використанням кількох ключових принципів для ефективної обробки та аналізу зображень. Одним із фундаментальних принципів є застосування локальних згорток, які проводять операції вибору інформації в обмежених областях вхідного зображення. Це дозволяє мережі виділяти локальні особливості, такі як краї та текстури.

Додатково, важливим аспектом є використання операцій пулінгу, які зменшують розмірність отриманого представлення, сприяючи узагальненню та

зменшенню кількості параметрів. Шари згорток та пулінгу часто стекуються, створюючи ієрархічну структуру для аналізу зображення.

Ще однією ключовою частиною є використання активаційних функцій, таких як ReLU, для введення нелінійності та навчання більш складних взаємодій. Вагові параметри та зв'язки мережі навчаються під час тренування, дозволяючи їй автоматично визначати розпізнавальні ознаки.

Згорткові нейронні мережі завершуються повністю зв'язаними шарами, які зв'язують згорткову інформацію з фінальними виходами мережі, такими як класифікація об'єктів. Ці принципи створюють компактні та потужні архітектури для ефективного розпізнавання зображень, де мережа може автоматично вивчати та використовувати узагальнені ознаки для ефективного вирішення завдань.

Важливо відзначити, що кожен шар CNN виконує абстракцію та ієрархічну розпізнавання, що дозволяє мережі визначати більш складні структури та патерни на зображенні. Такий поетапний підхід дозволяє мережі автоматично адаптуватися до різноманітних об'єктів та сценаріїв.

Однією з ключових особливостей CNN є їхня здатність до виявлення ієрархічних ознак. Починаючи з простих локальних патернів, таких як краї та форми, мережа поступово розпізнає більш складні концепції та об'єкти на вищому рівні абстракції. Це робить CNN ефективними для розпізнавання образів у великому спектрі завдань.

Крім того, розуміння того, як CNN навчаються вагові параметри та зв'язки, є критичним для адаптації до різноманітності зображень. Мережа вчиться реагувати на різні ознаки, визначаючи їхню вагомість у формуванні фінального вихідного сигналу.

Узагальнюючи, основні принципи роботи згорткових нейронних мереж є високорівневими стратегіями аналізу зображень, які поєднують у собі локальні фільтри, пулінг та активаційні функції для автоматичного вивчення та розуміння важливих ознак [14]. Це дозволяє CNN вирішувати складні задачі розпізнавання зображень, роблячи їх ефективними та витратно-ефективними інструментами в області машинного зору та штучного інтелекту.

2.2 Архітектурні особливості та шари CNN

Архітектурні особливості та шари згорткових нейронних мереж [15] складають комплексну структуру, що сприймає та аналізує вхідні зображення. Розглянемо основні аспекти цієї архітектури, на рисунку 2.1 представлено приклад архітектури згорткової нейронної мережі.

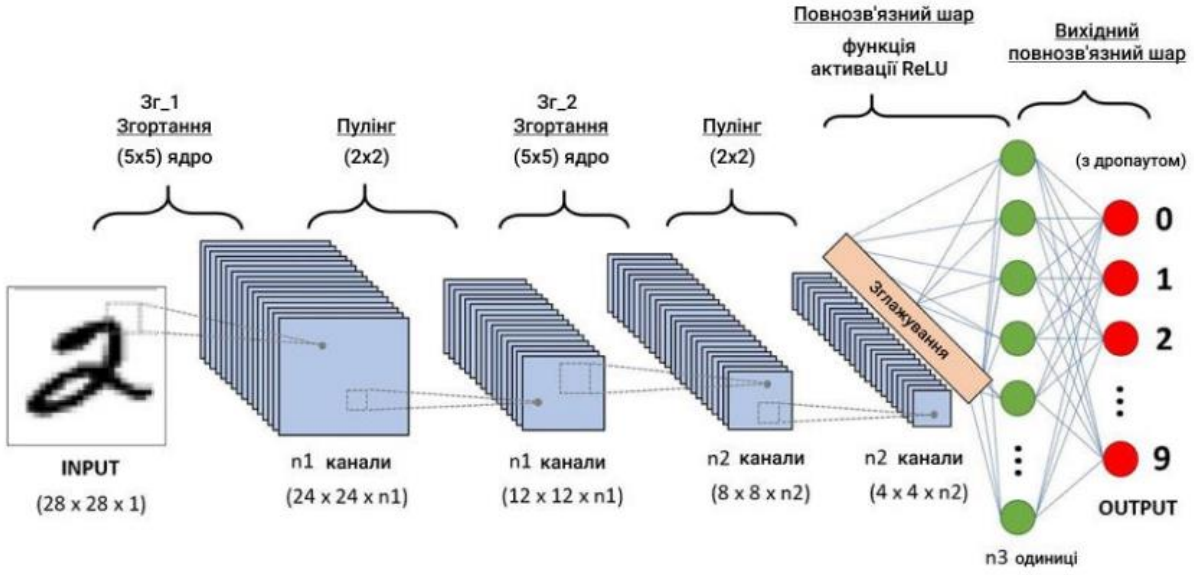


Рисунок 2.1 – Приклад архітектури згорткової нейронні мережі

Вхідний шар CNN функціонує як вхідна частина, де кожен піксель зображення взаємодіє з відповідними нейронами. Цей шар виступає важливою частиною, де дані нормалізуються та передаються для подальшої обробки.

Перший шар - це згортковий шар, де використовуються локальні згортки для виділення різноманітних особливостей. Кожна згортка є фільтром, що проводить операцію над невеликими областями зображення, розпізнаючи певні патерни або ознаки.

Після згорткового шару використовується шар пулінгу, який виконує пониження розмірності та об'єднання інформації. Це сприяє узагальненню та зменшенню кількості параметрів мережі, роблячи її більш ефективною.

Активаційні функції, такі як ReLU, введені для надання нелінійності та активування нейронів. Ці функції дозволяють мережі виявляти складніші зв'язки та підвищують її здатність до вивчення абстракцій.

Стековані згорткові та пулінгові шари утворюють ієрархічну структуру для аналізу зображення на різних рівнях складності. Кожен новий шар дозволяє мережі розпізнавати більш абстрактні концепції та структури, підвищуючи її здатність до розуміння складних взаємодій у зображеннях.

Після декількох згорткових та пулінгових шарів використовуються повністю зв'язані шари, які зв'язують вихідні дані з попередніми етапами з фінальним вихідним сигналом, таким як класифікація об'єктів.

Ці архітектурні особливості роблять CNN [16] потужним та універсальним інструментом для обробки та розпізнавання зображень у різних застосуваннях.

2.3 Роль CNN у вирішенні задач класифікації

Згорткові нейронні мережі відіграють ключову роль у вирішенні задач класифікації, яка є однією з найбільш поширених і важливих задач у галузі обробки зображень та машинного навчання.

Взаємодія із зображеннями: однією з основних ролей CNN є їхня здатність ефективно взаємодіяти із зображеннями. Замість розглядання кожного пікселя окремо, CNN визначають та аналізують локальні особливості та зв'язки між пікселями, що є ключовим для розпізнавання об'єктів на зображеннях.

Ефективна витяг ознак: згорткові шари CNN відповідають за витягання ознак із зображення. Кожен шар використовує згортки для визначення певних характеристик об'єктів, що допомагає у створенні високорівневих ознак та узагальнення для подальшого використання.

Градація концепцій: шари CNN розглядають образ на узагальнення. Починаючи з простих форм та текстур, мережа поступово будує більш складні концепції, що дозволяє класифікатору розрізняти об'єкти за їхніми характеристиками на різних рівнях складності.

Автоматизована ідентифікація: CNN дозволяють автоматизовано ідентифікувати та класифікувати об'єкти на зображеннях. Це важливо в різних сферах, включаючи розпізнавання обличь, класифікацію тварин чи визначення об'єктів на медичних зображеннях.

Навчання за зразками: CNN здатні навчатися за зразками, адаптуючи свої внутрішні вагові параметри для покращення точності класифікації. Це дозволяє їм ефективно пристосовуватися до різноманітних завдань класифікації зображень.

Розпізнавання контекстів: CNN допомагають розпізнавати об'єкти в різних контекстах та поза звичайними умовами, роблячи їх ефективними для задач класифікації в реальних умовах.

У цілому, CNN стали невід'ємною частиною багатьох систем класифікації зображень, дозволяючи автоматизувати та покращити точність розпізнавання об'єктів на зображеннях у різних галузях застосування.

2.4 Штучний нейрон

Кожна нейроморфна мережа ґрунтується на елементарних компонентах, які, в основному, є однотипними, імітуючи функцію нейронів мозку [18]. Кожен з цих нейронів характеризується своїм поточним станом, аналогічним до нервових клітин головного мозку.

Штучний нейрон, так само як його біологічний аналог, визначається своїм поточним станом і наявністю синапсів - односпрямованих вхідних зв'язків, які з'єднані з виходами інших нейронів. Крім того, у нейрона є аксон, що є вихідним зв'язком, з якого сигнал (збудження або гальмування) передається на синапси наступних нейронів, на рисунку 2.2 представлено штучний нейрон.

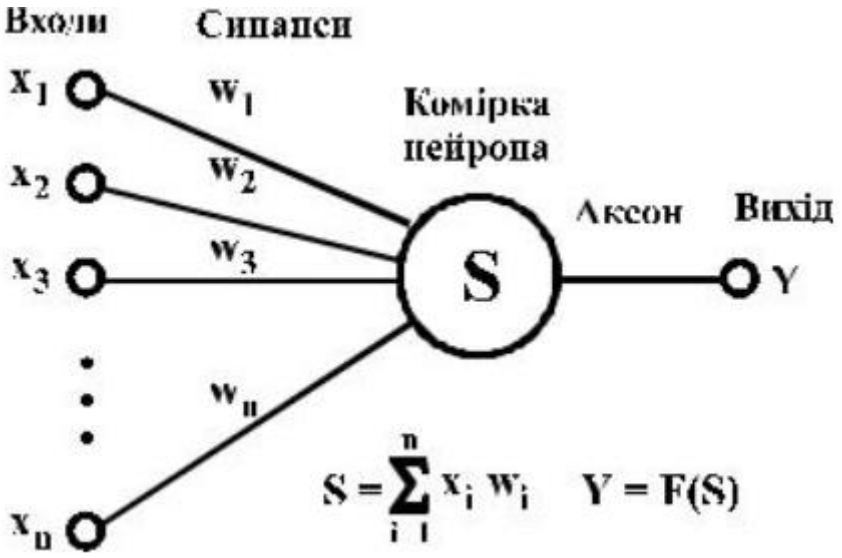


Рисунок 2.2 – Штучний нейрон

У кожному синапсі визначається параметр, відомий як синаптичний зв'язок або його вага (wi), який в фізичному сенсі еквівалентний електричній провідності.

Поточний стан нейрона визначається як зважена сума його входів:

$$s = \sum_{i=1}^{n} x_i \times w_i \qquad (2.1)$$

Вихід нейрона є функція його стану:

$$y = f(s) \qquad (2.2)$$

2.5 Згортковий шар

Шар згортки [19] є ключовим елементом згорткової нейронної мережі. Основний принцип роботи згорткового шару полягає в застосуванні згортки до вхідних даних за допомогою фільтра (ядро) для виявлення певних ознак, на рисунку 2.3 представлено прикад операції згортки. Під час згортки ядро фільтра рухається по вхідних даних, обчислюючи скалярний добуток між відповідними елементами ядра та частини вхідних даних. Цей процес допомагає відокремлювати важливі ознаки, такі як грані, текстури, чи форми, з вхідних даних.

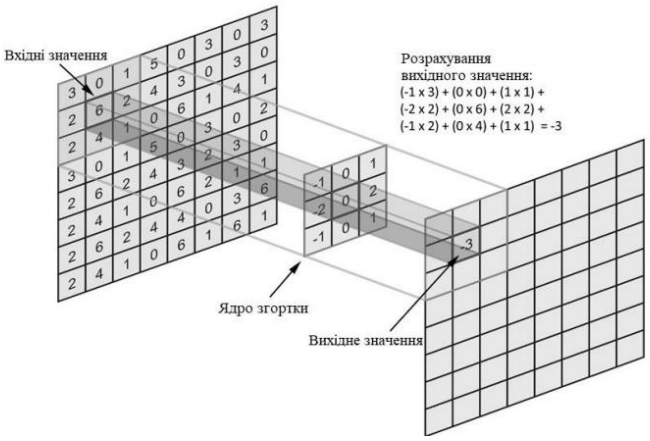


Рисунок 2.3 – Приклад операції згортки

Результатом згортки є карта ознак, що відображає просторовий розподіл активацій для кожного положення ядра вхідного зображення. Згортковий шар також використовує стрід (кількість пікселів, на які переміщується ядро) та заповнення для контролю розміру виходу.

Додавання неелементарних функцій активації, таких як ReLU, дозволяє введення нелінійності, а пулінг використовується для зменшення розміру карти ознак, зберігаючи при цьому важливі інформаційні ознаки. Згортковий шар може також використовувати кілька каналів для виявлення різних аспектів вхідних даних.

2.6 Шар пулінгу

Шар пулінг (або пулінг) є операцією, яка використовується для зменшення розмірності зображення (або вхідних даних) в згорткових нейронних мережах. Це допомагає зменшити кількість параметрів і обчислень в мережі, а також зробити модель менш чутливою до зміщень та змін масштабу.

Основні види шару пулінгу:

- максимальний пулінг (Max Pooling) [20]: для кожного регіону зображення обирається максимальне значення. Це забезпечує збереження найбільш важливих особливостей зображення, на рисунку 2.4 представлено max pooling;

- середній пулінг (Average Pooling) [21]: для кожного регіону обчислюється середнє значення. Це може допомогти згладжувати шум та зробити модель більш стійкою до змін;
- глобальний пулінг (Global Pooling) [22]: це спеціальний випадок, коли розмір пулінгового вікна дорівнює розміру вхідного зображення. Результатом є єдине число для кожного каналу, що представляє собою усереднене або максимальне значення по всьому зображенню.

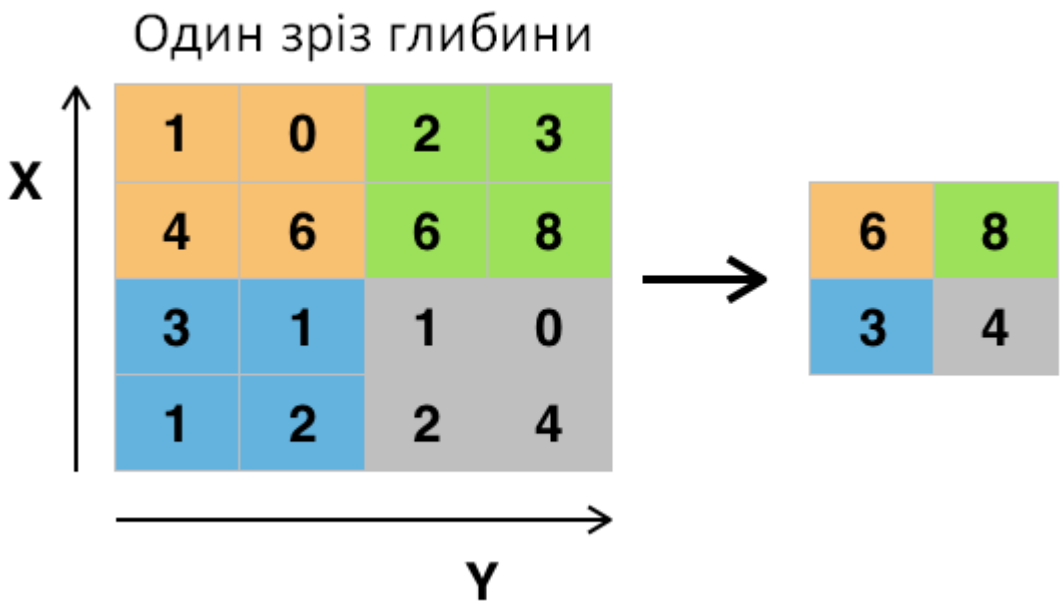


Рисунок 2.4 – Максимальний пулінг

Процес пулінгу [23] зазвичай виконується після згортки у згорткових шарах мережі. Вікно пулінгу переміщується по зображенню або фільтрується через вхідні дані з певним кроком. Це дозволяє зменшити просторові розміри карт ознак, зберігаючи при цьому ключові особливості.

Переваги шару пулінгу включають у себе зменшення обчислювального завантаження, зменшення кількості параметрів, що потрібно оптимізувати, та стійкість до невеликих трансформацій вхідних даних. Однак важливо враховувати, що при використанні пулінгу втрачається частина просторової інформації, і це може призвести до втрати деталей у вихідних картах ознак.

2.7 Обчислення використання пам'яті нейронною мережею

Згорткові нейронні мережі є потужним інструментом у сфері розпізнавання зображень. Однак вони вимагають значних обчислювальних ресурсів, зокрема пам'яті, для ефективної роботи. Питання використання пам'яті згортковою нейронною мережею [24] для розпізнавання зображень залишається актуальним у наукових та практичних кіл.

Згорткові нейронні мережі є типом штучних нейронних мереж, які спеціалізуються на обробці зображень. Вони складаються зі згорткових шарів, які використовують фільтри для виявлення різних ознак у вхідних зображеннях, пулінгових шарів для зменшення просторових розмірів оброблюваних зображень, а також повнозв'язаних шарів для класифікації зображень за отриманими ознаками. CNN демонструють вражаючі результати у розпізнаванні об'єктів на зображеннях, що робить їх популярними у багатьох областях, включаючи медицину, автоматичне водіння та багато інших.

Однак, використання пам'яті згортковою нейронною мережею є важливим аспектом, особливо при обробці великих обсягів даних. Загально визнано, що CNN потребують значних обчислювальних ресурсів, зокрема пам'яті, для ефективної обробки зображень. Питання використання пам'яті згортковою нейронною мережею виникає у зв'язку з обмеженими обчислювальними ресурсами та необхідністю оптимізації роботи мережі для практичних застосувань.

2.8 Планування архітектури згорткової нейронної мережі

Під час процесу проектування архітектури згорткової нейронної мережі [25], дослідники та інженери використовують різні підходи з метою досягнення оптимальної продуктивності та точності моделі. Вони можуть розглядати різноманітні аспекти, такі як розмір та глибина шарів, кількість фільтрів, розмір ядра, функції активації та зворотній зв'язок [26].

Одним із підходів є використання концепції глибоких архітектур [27], де збільшення кількості шарів дозволяє моделі вивчати більш абстрактні функції. Це може вимагати великої кількості параметрів, але забезпечує велику експресивність моделі [28].

Інший підхід - це використання пулінгу та стиснення інформації для зменшення просторового розміру областей, що дозволяє знизити обчислювальні витрати та контролювати перенавчання.

Також розглядають питання використання фільтрів різних розмірів для виявлення різних рівнів деталізації вхідних зображень, що допомагає моделі адаптуватися до різних аспектів вхідних даних.

Крім того, деякі підходи враховують оптимізацію функцій активації для покращення збіжності та швидкості навчання.

Загалом, успішне планування архітектури CNN [29] включає у себе балансування різних параметрів та врахування конкретних вимог завдання, для якого створюється модель.

2.9 Висновок до розділу

В розділу було проведено аналіз детальний аналіз в результаті чого було обрано кількість шарів згортки і пулінгу, а саме 5 шарів згортки і пулінгу. Розмір фільтрів згортки 5 а їх кількість рівняється 16 і розмір кроку пулінгу складає 2. Потім йде повнозв’язаний шар котрий складається з 4 шарів звязаних між собою 32 нейронів, а потім шар виходу в котрий включає 4 нейрони.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕННЯ

Практична реалізація була створена як програмний модуль до проекту універсальної віртуальної лабораторії Labs, на рисунку 3.1 представлено приклад програмного модуля.

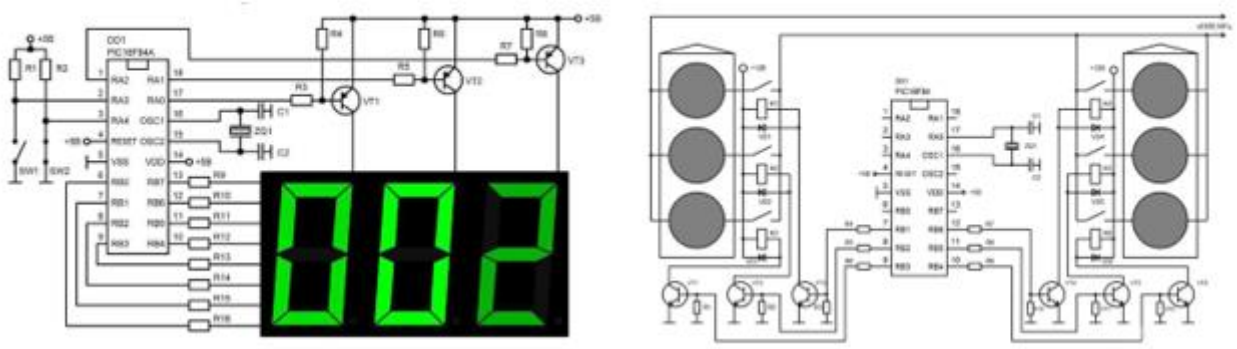


Рисунок 3.1 – Демонстація зовнішнього вигляду лабораторних макетів, спроектованих для програмування мікроконтролерів

Ядро програми включає в себе користувацький інтерфейс, який охоплює інтерактивну схему з можливістю масштабування, елементи управління процесом моделювання та параметрами аналізованої схеми, віртуальні вимірювальні пристрої і всі необхідні додаткові програмні модулі для здійснення досліджень.

Лабораторні макети економлять обчислювальні ресурси та автоматично адаптуються під параметри конкретного комп'ютера. Вони дозволяють індивідуально налаштовувати параметри елементів (щоб уникнути повторення результатів вимірів у різних груп студентів). Використання технології DLL дозволяє досягти високої швидкості обчислень, що у деяких випадках може перевищити швидкість обробки вивчених процесів у реальному часі. При цьому забезпечується максимальна відповідність форми та характеру поведінки вивчених сигналів.

3.1 Алгоритм навчання згорткової нейронної мережі

Алгоритм навчання згорткової нейронної мережі (CNN) [33] є важливим елементом в області глибокого навчання та обробки зображень. Ось кілька ключових аспектів, які відзначають важливість цього алгоритму [34]:

- здатність до розпізнавання патернів: згорткові шари в CNN використовують фільтри для виявлення локальних патернів в зображенні; це дозволяє нейронні мережі автоматично вчитися важливі ознаки, такі як границі, текстури та форми;
- локальність та інваріантність до зсуву: Згорткові операції дозволяють нейронні мережі реагувати на локальні ознаки, що робить їх інваріантними до зсуву; це означає, що вони можуть ефективно працювати з об'єктами, розташованими в різних частинах зображення;
- зменшення просторових розмірів: Max Pooling, який часто використовується після згорткових шарів, допомагає зменшити просторові розміри зображення, зберігаючи при цьому важливі ознаки; це сприяє зменшенню обчислювального завдання та робить мережу менш вразливою до перенавчання;
- можливість взаємодії з повністю з'єднаними шарами: Fully-Connected шари в кінці архітектури дозволяють моделі здійснювати класифікацію та приймати рішення на основі ознак, вивчених згортковими шарами;
- ефективність: CNN ефективні для обробки зображень та використовуються в широкому спектрі задач, включаючи розпізнавання облич, класифікацію об'єктів, сегментацію та багато інших;

Узагальнюючи, алгоритм навчання згорткової нейронної мережі відіграє ключову роль у досягненні вражаючих результатів у сфері комп'ютерного зору та обробки зображень.

Алгоритм навчання котрий був обраний для згорткової нейронні мережі включає в себе наступні кроки, на рисунку 3.2 представлено блок схему з алгортмом:

— Старт: Ініціалізація параметрів нейронні мережі.

— Input: Подача вхідного зображення.

— Convolution & Max Pooling: Згортка та використання Max Pooling для отримання нелінійних характеристик та зменшення просторових розмірів.

— X Shift & Y Shift: Зсув вхідного зображення вздовж осей X та Y.

— Resize: Зміна розмірів зображення до досягнення заданого розміру.

— Fully-Connected Layer: Перетворення ознак в одномірний вектор та подача на повністю з’єднаний шар нейронні мережі.

— Info Output: Вивід інформації про результати обробки.

— End: Завершення алгоритму навчання згорткової нейронні мережі.

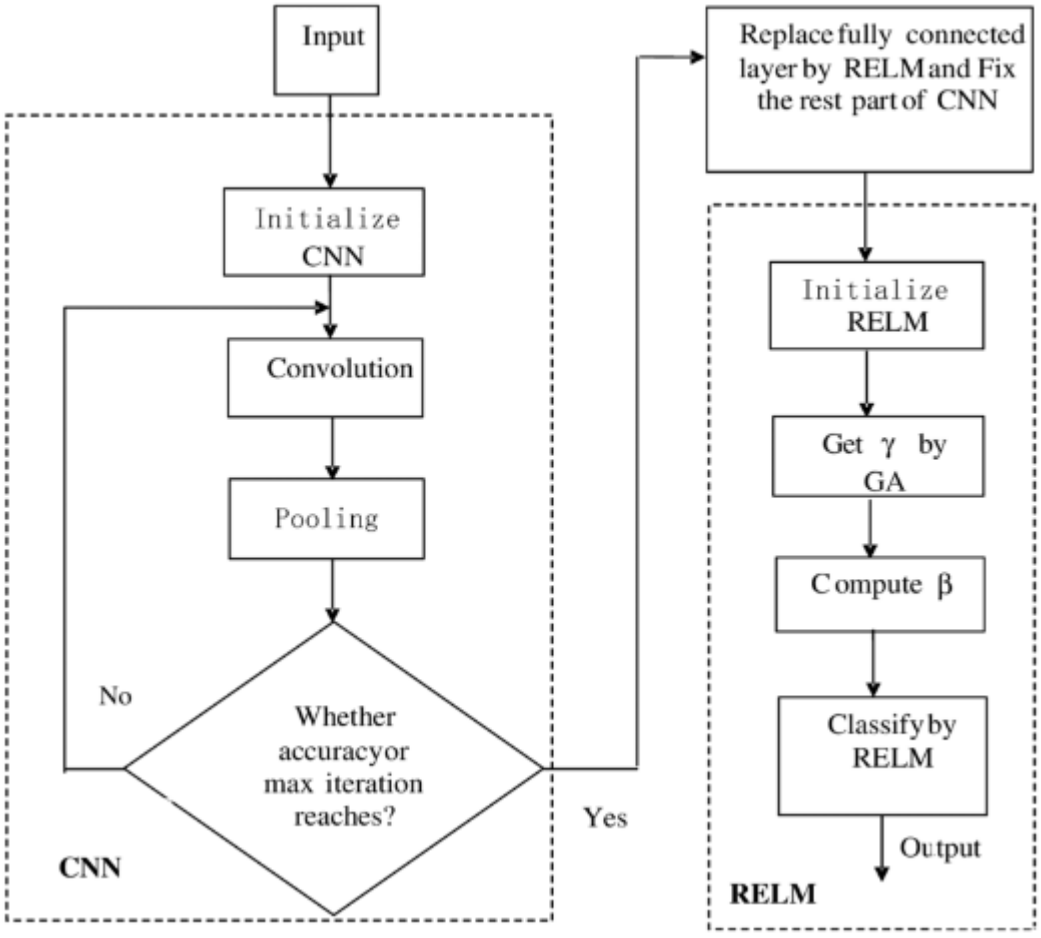


Рисунок 3.2 – Блок схема алгоритму навчання

3.2 Інтерфейс для згорткової нейронної мережі

У цьому розділі ми розглянемо розроблений інтерфейс для згорткової нейронної мережі, який надає користувачам зручний спосіб взаємодії з системою [35]. Інтерфейс включає в себе ключові функції, такі як завантаження зображень, відображення активності нейронів, а також можливості навчання, тренування та розпізнавання, на рисунку 3.3 зображено загальний вид інтерфейсу.

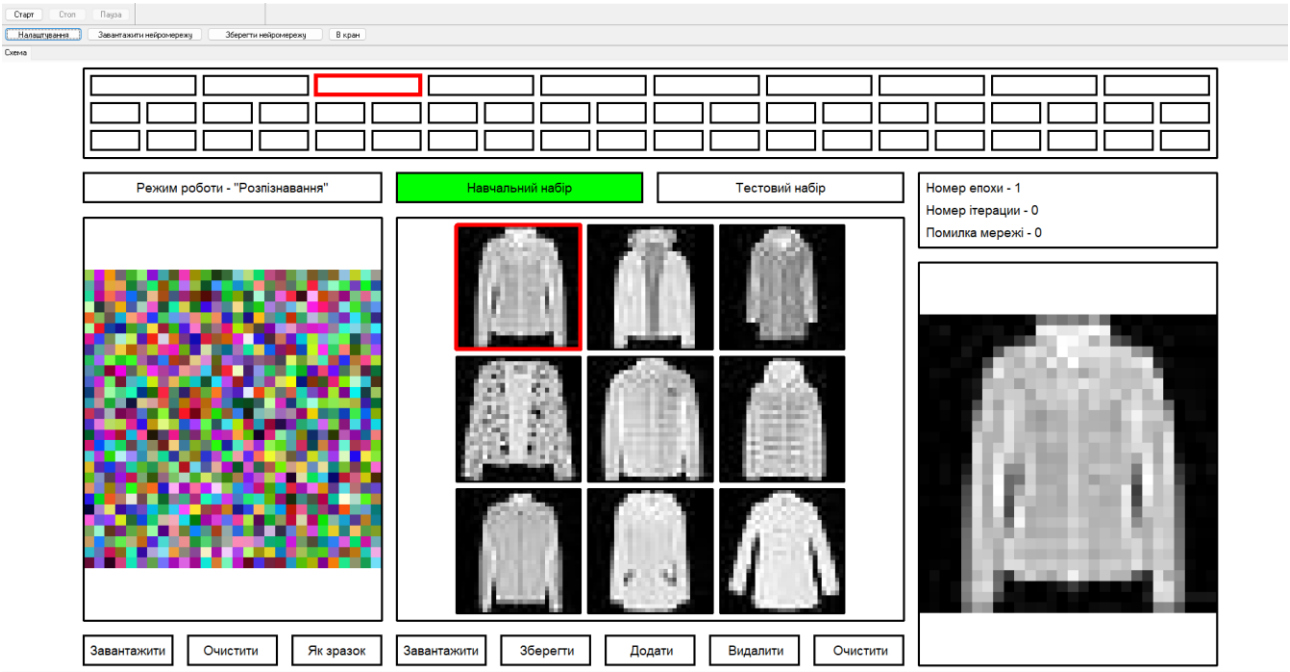


Рисунок 3.3 – Загальний вид інтерфейсу

Початкова стадія взаємодії з системою починається з можливості завантаження зображень, на рисунку 3.3 зображено кнопку і вікно куди завантажується зображення. Користувач може вибрати одне чи кілька зображень зі свого пристрою та імпортувати їх до системи. Це дозволяє використовувати реальні дані для навчання та тестування CNN.

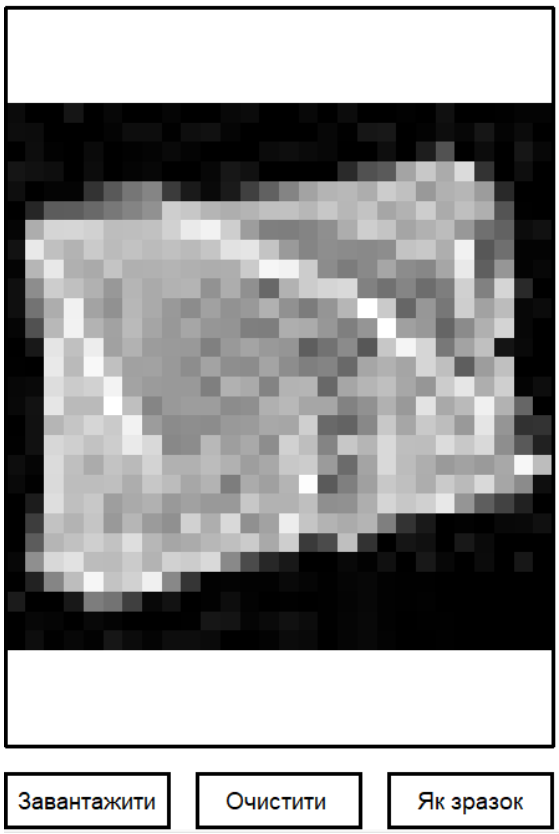


Рисунок 3.3 – Кнопка і вікно завантаження

Інтерфейс включає в себе візуалізацію активності нейронів у режимі реального часу. Користувач може спостерігати за тим, як нейрони реагують на різні частини зображення, що дозволяє отримати інсайди в роботу мережі та її здатність до виявлення особливостей, відстеження активності нейронів зображено на рисунку 3.4.

Рисунок 3.4 – Відображення активності нейронів

На рисунку 3.5 представлено вікно виборки для навчання та розпізнавання.

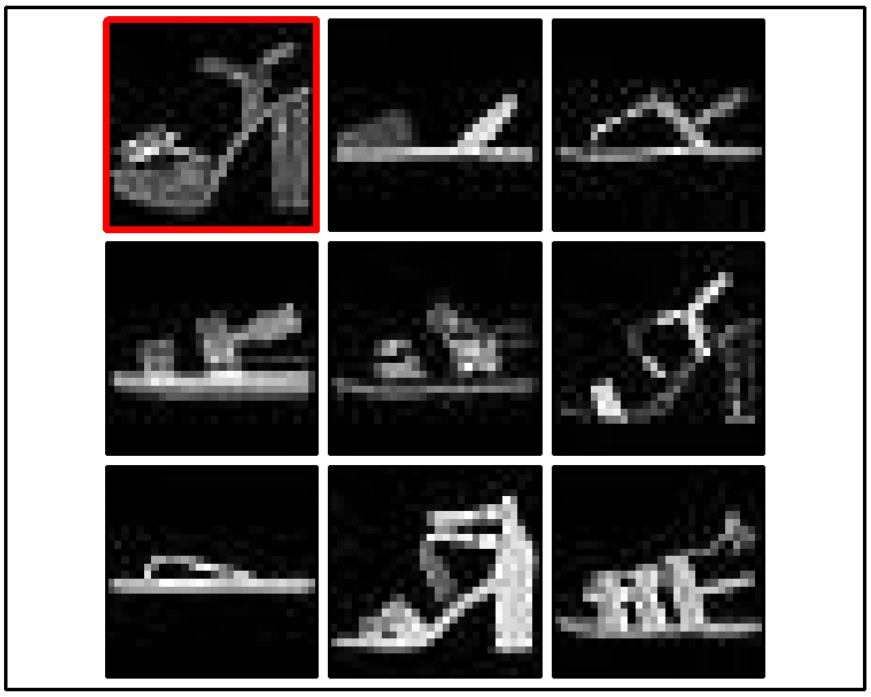


Рисунок 3.5 – Вікно виборки зображень

На рисунку 3.6 зображено відстеження епох, ітерацій і помилок мережі.

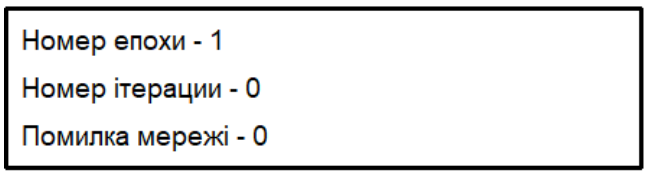


Рисунок 3.6 – Відстеження епох, ітерацій, помилок мережі

3.3 Опис класів і методів нейронної мережі

Для роботи з нейронною мережею створюємо глобальні змінні. Змінні створюємо в глобальному оточенні для їх доступності для будь якого класу або методу, в таблиці 3.1 представлено перелік змінних.

Таблиця 3.1 – глобальні змінні нейронної мережі

Назва	Тип	Значення за замовченням	Область видимості	Призначення
TNeuronInputs	array of PDouble	nil	global	вхідні дані нейрона
TNeuronWeights	array of PDouble	nil	global	ваги нейрона
TActivation FunctionParasm	array of Double	nil	global	параметри для функції активації нейрона

Створюємо клас TConvolutionLayer, він призначений для реалізацію двох операцій обробки зображень – згорткового шару (convolution layer) і шару пулінгу (pooling layer), які часто використовуються в глибокому навчанні та обробці зображень, в цьому класі є лише одна змінна FScreen методах ApplyConvolution та ApplyPooling вхідне зображення передається через FScreen. Кожна ітерація цих методів модифікує зображення, яке зберігається в FScreen. Наприклад, метод ApplyConvolution застосовує згортковий фільтр до пікселів вхідного зображення і зберігає результат у тому ж FScreen, в таблиці 3.2 представлено методи шару згортки.

Згортковий шар (Convolution Layer) [36]: згорткові шари використовують фільтри (або ядро), щоб виконати згортку зображення. Цей код застосовує згортковий фільтр до кожного пікселя вхідного зображення, використовуючи ваги фільтра для обчислення нового значення пікселя в результуючому зображенні. Згорткові шари допомагають виявити різні особливості та патерни в зображеннях.

Шар пулінгу (Pooling Layer) [37]: пулінг використовується для зменшення розміру зображення та виділення його важливих особливостей. У цьому коді використовується максимальний пулінг, де для кожного блоку 2x2 вхідного зображення вибирається максимальне значення, і це значення стає новим значенням пікселя в результуючому зображенні. Це допомагає зменшити розмір зображення, зберігаючи важливі особливості.

Таблиця 3.2 – методи класу TConvolutionLayer

Назва	Тип	Параметри	Опис
ApplyConvolution	procedure	-	Застосовує згортковий фільтр до вхідного зображення.
ApplyPooling	procedure	-	Застосовує операцію пулінгу до вхідного зображення.

Надалі створюємо TFullyConnectedLayer котрий отримує зображення після пулінгу та застосовується операції повнозв’язаного шару до кожного пікселя, для кожного пікселя обчислюється вихідне значення яке буде використане для подальших обчислень, в даному класі також одна змінна FNeurons: TNeuronList; котра передає список нейроннів в шарі, а також в таблиці 3.3 представлено методи класу.

Таблиця 3.3 – методи класу TFullyConnectedLayer

Назва	Тип	Параметри	Опис
Create	constructor	-	Створює та ініціалізує об'єкт повнозв'язаного шару, включаючи створення та ініціалізацію нейронів.
Destroy	destructor	-	Звільняє ресурси, включаючи список нейронів (FNeurons), під час знищення об'єкта.
SaveToStream	procedure	F: TFileStream	Зберігає інформацію про нейрони шару у переданий потік за допомогою методу SaveToStream списку нейронів.
LoadFromStream	function	F: TFileStream: Boolean	Завантажує інформацію про нейрони шару з переданого потоку за допомогою методу LoadFromStream. Повертає True, якщо завантаження успішне, в іншому випадку - False.
ApplyFully ConnectedLayer	Procedure	PooledBitmap: TBitmap	Застосовує операції повнозв'язаного шару до кожного пікселя переданого зображення. Обчислює вихідне значення для кожного пікселя, використовуючи ваги нейронів та значення пікселів.

Потім створюємо клас TOutputLayer , який реалізує вихідний шар нейронної мережі. Вихідний шар в нейронних мережах відповідає за генерацію вихідних

значень, які часто інтерпретуються як результати роботи мережі. Основна його задача - обробка інформації, яка надходить від попередніх шарів та генерація вихідних сигналів. Клас має дві змінні, а саме FNeuronCount: Integer ця змінна визначає кількість нейронів у вихідному шарі. Кількість нейронів визначається при створенні об'єкта TOutputLayer і може бути налаштована користувачем в залежності від конкретного завдання чи вимог і наступна змінна TNeuronList, яка є списком нейронів у вихідному шарі, під час ініціалізації вихідного шару в конструкторі класу, для кожного нейрона створюється об'єкт класу TNeuron та його вага випадковим чином ініціалізується. Ці нейрони потім додаються до списку FNeurons.

Клас має два параметри дані про котрі представлено в таблиці 3.4.

Таблиця 3.4 – параметри класу TOutputLayer

Назва	Тип	Читання	Запис	Область видимості	Призначення
FNeuronCount	Integer	так	ні	private	Визначає кількість нейронів у вихідному шарі.
FNeurons	TNeuronList	так	ні	private	Містить список нейронів у вихідному шарі.

Огляд методів представлено в таблиці 3.5.

Таблиця 3.5 – методи класу TOutputLayer

Назва	Тип	Параметри	Опис
Create	constructor	AInputSize, ANeuronCount: Integer	Створює та ініціалізує об'єкт вихідного шару з вказаною кількістю нейронів та розміром входу. Викликає конструктор базового класу (inherited Create) для налаштування входу та кількості нейронів. Ініціалізує параметри нейронів виходного шару, їх ваги.
Destroy	destructor	-	Звільняє ресурси, пов'язані з кожним нейроном в вихідному шарі, а також звільняє список нейронів. Викликає деструктор базового класу (inherited) для звільнення його ресурсів.

Клас TNeuron представляє собою нейрон нейронної мережі, зміні котрої продемонстровані в таблиці 3.6, а методи в таблиці 3.7.

Таблиця 3.6 – змінні класу TNeuron

Назва	Тип	Опис
FColor	TColor	Колір, пов'язаний з виходом нейрона.
FInputs	TNeuronInputs	Масив вказівників на входи нейрона.
FActivationFunction	TActivationFunction	Функція активації для нейрона.
FActivationFunctionParams	TActivationFunctionParams	Параметри функції активації.
FFocused	Boolean	Прапорець, що вказує на активність нейрона.
FError	Double	Значення помилки нейрона.
Output	Double	Вихідне значення нейрона.

Таблиця 3.7 – методи класу TNeuron

Назва	Тип	Параметри	Опис
Calculate	procedure	aToLog: Boolean	Обчислює вихід нейрона, використовуючи вхідні значення та функцію активації.
Create	constructor	-	Конструктор класу, ініціалізує змінні та налаштовує деякі параметри за замовчуванням.
Destroy	destructor	-	Деструктор класу, звільняє ресурси, пов'язані з екземпляром класу.
GetActivation FunctionParamCount	function	Index: Integer	Повертає значення параметра функції активації за індексом.
GetColor	function	-	Повертає колір, пов'язаний з виходом нейрона.
GetInput	function	Index: Integer	Повертає вказівник на вхід нейрона за індексом.
GetInputCount	function	-	Повертає кількість входів нейрона.
SetActivation FunctionParamCount	procedure	Value: Integer	Задає кількість параметрів для функції активації.
SetActivation FunctionParams	procedure	Index: Integer, Value: Double	Задає значення параметра функції активації за індексом.
SetInput	procedure	Index: Integer, Value: PDouble	Задає вказівник на вхід нейрона за індексом.
SetInputCount	Procedure	Value: Integer	Value: Integer

Далі клас TNeuronNetwork котрий містить методи та властивості, пов'язані з управлінням шарами нейронні мережі, обробкою екранових виведень та збереженням/завантаженням стану мережі, в таблицях 3.8, 3.9 і 3.10 відповідно представлені змінні, властивості і методи.

Таблиця 3.8 – змінні класу TNeuronNetwork

Назва	Тип	Опис
BiasNeuroun	Double	Зсув для нейронів
ScreenOutputs	Масив Double	Масив для зберігання вихідних даних екрану
FLayers	TNeuronLayerList	Список шарів нейронні мережі
FScreen	TBitmap	Зображення екрану
FScreenImageFileName	String	Шлях до файлу зображення екрану
FFileName	String	Ім'я файлу для зберігання/завантаження

Таблиця 3.9 – властивості класу TNeuronNetwork

Назва	Тип	Читання	Запис	Область видимості	Опис
FileName	String	Так	Так	Захищена	Ім'я файлу для зберігання/завантаження
Screen	TBitmap	Так	-	Захищена	Зображення екрану

Таблиця 3.10 – методи класу TNeuronNetwork

Назва	Тип	Параметри	Опис
Create	constructor	-	Конструктор класу
Destroy	destructor	-	Деструктор класу
LoadScreenFromFile	procedure	-	Завантажує зображення екрану з файлу
ClearScreen	procedure	-	Заповнює екран випадковими кольорами
CopyToScreen	procedure	aSource: TBitmap	Копіює зображення на екран
SaveToFile	procedure	ShowDialog: Boolean	Зберігає нейронні мережіу у файл
SaveToStream	procedure	F: TFileStream	Зберігає нейронні мережіу у потік файлу
SetScreenDimentions	procedure	aWidth, aHeight: Integer	Встановлює розміри екрану

3.4 Результати дослідження

Попередня обробка датасету є критичним етапом в розробці системи розпізнавання одягу на основі згорткових нейронні мережі (CNN). Ефективна обробка даних допомагає підготувати датасет для оптимального навчання моделі, що в свою чергу позитивно впливає на якість та швидкість розпізнавання.

Першим кроком є збір відповідного датасету для задачі розпізнавання одягу. Це може бути набір фотографій одягу, які включають різні типи одягу та різні перспективи зйомки. Крім того, важливо визначити класи (категорії) одягу, які модель повинна розпізнавати, такі як футболки, штани, сукні тощо.

Для оцінки ефективності моделі важливо розділити датасет на дві частини: навчальний та валідаційний. Навчальний набір використовується для самого процесу навчання, тоді як валідаційний дозволяє визначити загальну ефективність моделі та виявити ознаки перенавчання.

Буде використаний датасет Fashion MNIST [38] фрагмент котрого представлений на рисунку 3.7.

Рисунок 3.7 – Фрагмент датасету Fashion MNIS

Під час тренування [39] згорткової нейронні мережі для розпізнавання одягу виникли деякі виклики, що обмежили її ефективність.

Модель була навчена на невеликому наборі з лише 9 екземплярів одного типу одягу. Це обмежене число прикладів може призвести до перенавчання та зниження загальної ефективності моделі.

У даному випадку, була вибрана архітектура, що не найкращим чином підходить для розпізнавання обраних типів одягу.

Хоча додаткове навчання було проведено до 100 епох, модель не досягла бажаного результату через недостатність та неадекватність даних, а також неправильно підібрану архітектуру. В майбутньому необхідно зосередитися на розширенні датасету, вдосконаленні архітектури та налаштуванні параметрів для досягнення кращих результатів в розпізнаванні одягу.

Отримані результати свідчать про деякі обмеження та проблеми у процесі тренування та тестування моделі для класифікації одягу. Зупинка тренування після 60 епох може бути недостатньою для досягнення оптимальних ваг моделі та уникнення недонавчання. Це може вплинути на здатність моделі до коректного класифікування різних типів одягу.

Точність моделі в межах від 30% до 50% також вказує на те, що її ефективність є відносно низькою, на рисунку 3.8 представлено графік ефективності нейронної мережі, а в таблиці 3.11 представлено точні дані по ефективності нейронної мережі. Тестування на розширеному датасеті, який містить 81 екземпляр одягу, дозволяє побачити певне поліпшення відсотка точності до 52-69%. Однак цей приріст є невеликим, що може вказувати на те, що обмежений обсяг даних може бути фактором, який гальмує ефективність моделі.

Таблиця 3.11 – Ефективність згорткової нейронної мережі

Епохи	Тренування	Тестування
10	10%	10%
20	12%	14%
30	22%	23%
40	26%	28%
50	34%	35%
60	48%	46%
70	48%	49%
80	48%	53%
90	48%	54%
100	48%	65%

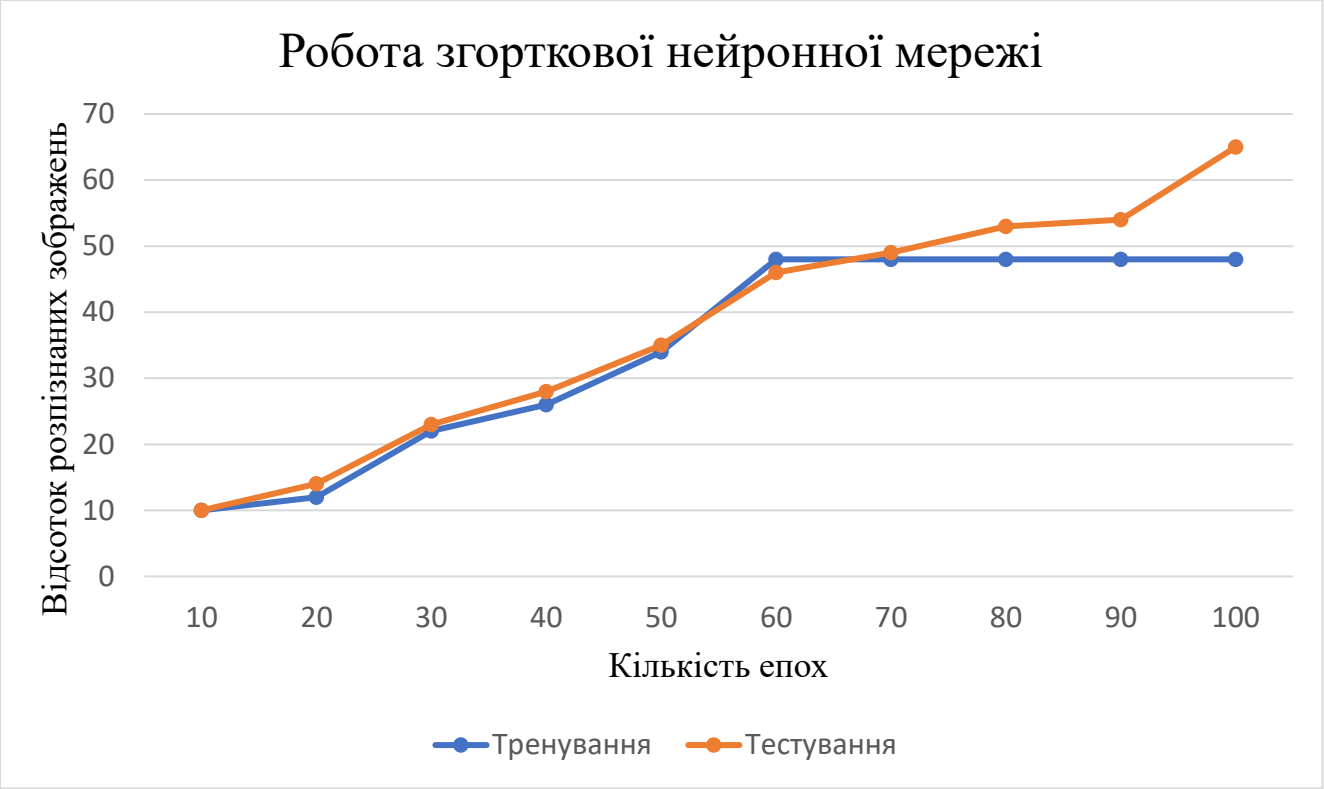


Рисунок 3.8 – Графік ефективності згорткової нейронної мережі

Також неправильно підібрана архітектура може впливати на результати. Це може включати в себе неоптимальний вибір глибини, ширини або типу шарів у нейронній мережі [40].

Резюмуючи, для покращення ефективності моделі рекомендується розглянути продовження тренування для досягнення оптимальних ваг, розширення обсягу тренувального датасету та перегляд архітектури моделі [41].

3.5 Висновок до розділу

Після програмної реалізації були проаналізовані результати дослідження згорткової нейронної мережі і було отримано ефективність згорткової нейронної мережі в розпізнаванні зображення. За даними аналізу модель є неефективною, оскільки точність у межах від 30% до 50% є невисокою. Проведення тестування на розширеному наборі даних, який включає 81 зразок одягу, призводить до помітного покращення відсотка точності в межах 52-69%. Проте цей зріст є обмеженим, що

може свідчити про те, що обмежений обсяг даних може бути чинником, який стримує ефективність моделі.

Це лише один приклад поганої реалізації, важливо враховувати, що ця тема має багато аспектів для подальшого розвитку і удосконалення.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

Головними результатами роботи є наступні пункти:

1. Проведено аналіз видів нейронних мереж для розпізнавання зображень з метою щоб обрати вид нейронної мережі для реалізації програмного забезпечення. За результатами проведеного аналізу та розгляду характеристик різних типів нейронних мереж, було прийнято рішення вибрати згорткову нейронну мережу.
2. В розділу було проведено аналіз детальний аналіз в результаті чого було обрано кількість шарів згортки і пулінгу, а саме 5 шарів згортки і пулінгу. Розмір фільтрів згортки 5 а їх кількість рівняється 16 і розмір кроку пулінгу складає 2. Потім йде повнозв'язаний шар котрий складається з 4 шарів звязаних між собою 32 нейронів, а потім шар виходу в котрий включає 4 нейрони.
3. За даними аналізу дана модель є неефективною, оскільки точність у межах від 30% до 50% є невисокою. Проведення тестування на розширеному наборі даних, який включає 81 зразок одягу, призводить до помітного покращення відсотка точності в межах 52-69%. Проте цей зріст є обмеженим, що може свідчити про те, що обмежений обсяг даних може бути чинником, який стримує ефективність моделі.

Результати даної роботи доцільно рекомендувати для удосконалення і практичного застосування у наступних напрямках.

1. При створенні програмного забезпечення функція котрого полягає в розпізнаванні об'єктів завчасно продумуючи сверу використання нейронної мережі.
2. Під час підготовки фахівців в галузі інформаційних технологій, інженерії програмного забезпечення, комп'ютерних науках, комп'ютерній інженерії і кібербезпеки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Terrence L. Fine. Feedforward Neural Network Methodology, 1999, pp. 129-234.
2. Nazri Mohd. Nawi, Abdullah Khan & Mohammad Zubair Rehman. A New Back-Propagation Neural Network Optimized with Cuckoo Search Algorithm, 2013, pp. 413-426.
3. Filippo Maria Bianchi , Enrico Maiorino , Michael C. Kampffmeyer , Antonello Rizzi , Robert Jenssen. Recurrent Neural Networks for Short-Term Load Forecasting, 2017, pp. 23-39.
4. Mohit Sewak , Md. Rezaul Karim , Pradeep Pujari. Practical Convolutional Neural Networks, 2018, pp. 109-180.
5. Emil Hvitfeldt, Julia Silge. Long short-term memory (LSTM) networks, 2021, pp. 10-30.
6. Seth Weidman. Deep Learning from Scratch: Building with Python from First Principles 1st Edition, 2019, pp. 50-103.
7. Shiuan Wan, Mei-Ling Yeh, Hong-Lin Ma, Tein-Yin Chou. The Robust Study of Deep Learning Recursive Neural Network for Predicting of Turbidity of Water, 2022, https://doi.org/10.3390/w14050761.
8. Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, Kaori Togashi. Convolutional neural networks: an overview and application in radiology, 2018, https://doi.org/10.1007/s13244-018-0639-9.
9. Purwono Purwono, Alfian Ma'arif, Wahyu Rahmaniar, Haris Imam, Karim Fathurrahman, Zatu Kusuma Frisky, Qazi Mazhar Ul Haq. International Journal of Robotics and Control Systems. Understanding of Convolutional Neural Network (CNN): A Review, 2023, DOI:10.31763/ijrcs.v2i4.888.
10. Ihsan Uluocak, Mehmet Bilgili. Acta Geophysica. Daily air temperature forecasting using LSTM-CNN and GRU-CNN models, 2023, DOI:10.1007/s11600-023-01241-y.

11. Xinyi Xu, Shi Tu, Yunfan Xue, Lin Chai. Visualizing CNN: An Experimental Comparative Study, 2013, DOI:10.1007/978-3-031-47665-5_17, pp. 199-212.

12. Xiu Zhang, Xin Zhang, Wei Wang. Convolutional Neural Network, 2023, DOI:10.1007/978-981-99-6449-9_2, pp. 39-71.

13. Venkata vara Prasad, Lokeswari Y Venkataramana. Journal Of Advanced Zoology. Optimization of Deep CNN Techniques to Classify Breast Cancer and Predict Relapse, 2023, DOI:10.17762/jaz.v44i4.2182.

14. Pranab Dey. Artificial Neural Network in Pathology: Basic Principles and Applications, 2023, DOI:10.1007/978-981-19-6616-3_25, pp. 267-275.

15. Khalid Elghazi, IsmailHassan, Tawfik Masrour. Genetic Algorithm for CNN Architecture Optimization, 2023, DOI:10.1007/978-3-031-43520-1_8, pp. 86-97.

16. Yingjian Yang, Nanrong Zeng, Ziran Chen, Wei Li, Yingwei Guo, Shicong Wang, Wenxin Duan, Yang Liu, Rongchang Chen, Yan Kang. ournal of Healthcare Engineering. Multi-Layer Perceptron Classifier with the Proposed Combined Feature Vector of 3D CNN Features and Lung Radiomics Features for COPD Stage Classification, 2023, DOI:10.1155/2023/3715603.

17. Rachel Draelos, MD, PhD. The History of Convolutional Neural Networks. Retrieved from https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/. 2019.

18. Sashmita Panda. Comparative study of single biological neuron with an artificial neuron, 2023, DOI:10.54646/bjbnt.2023.02.

19. Takumi Ando. Save GPU RAM Usage in Convolutional Layers to Load Huge Images, 2023, DOI:10.1101/2023.09.19.558533.

20. Naila Murray, Florent Perronnin. Generalized Max Pooling, 2014, DOI:10.1109/CVPR.2014.317.

21. He Li, Peng Weiwen, Sidum Adumene, Mohammad Yazdi. Intelligent Reliability and Maintainability of Energy Infrastructure Assets, 2023, DOI:10.1007/978-3-031-29962-9_5, pp. 73-91.

22. J. Sun, X. He, W. Tan, X. Wu, Jifeng Shen, H. Lu. Recognition of crop seedling and weed recognition based on dilated convolution and global pooling in CNN. Nongye

Gongcheng Xuebao/Transactions of the Chinese Society of Agricultural Engineering, 2018, DOI:10.11975/j.issn.1002-6819.2018.11.020.

23. Dingjun Yu, Hanli Wang, Peiqiu Chen & Zhihua Wei. International Conference on Rough Sets and Knowledge Technology, 2014, pp 364–375.

24. CNN memory consumption. Retrieved from https://datascience.stackexchange.com/questions/17286/cnn-memory-consumption.

25. Introduction to Convolution Neural Network. Retrieved from https://www.geeksforgeeks.org/introduction-convolution-neural-network/.

26. Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network. Retrieved from https://www.upgrad.com/blog/basic-cnn-architecture/.

27. Convolutional Neural Networks (CNN) — Architecture Explained. Retrieved from https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243.

28. Mohammad Taye. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions, 2023, DOI:10.3390/computation11030052.

29. Umberto Michelucci. Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection 1st ed. Edition, 2019, pp. 102-140.

30. What Is a Convolutional Neural Network?. Retrieved from https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html.

31. What is Delphi IDE and Pascal Programming Language? A Beginner’s Guide. Retrieved from https://www.devart.com/delphi-programming-language/.

32. Primoz Gabrijelcic. Delphi High Performance: Master the art of concurrency, parallel programming, and memory management to build fast Delphi apps, 2nd Edition, 2023, pp 203-235.

33. CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization. Retrieved from https://arxiv.org/abs/2004.15004.

34. The Best Optimization Algorithm for Your Neural Network. Retrieved from https://towardsdatascience.com/the-best-optimization-algorithm-for-your-neural-network-d16d87ef15cb.

35. Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, Duen Horng Chau, 2020, pp. 1-7, CNN 101: Interactive Visual Learning for Convolutional Neural Networks. Retrieved from https://doi.org/10.1145/3334480.3382899.

36. Convolutional Layer. Retrieved from https://www.sciencedirect.com/topics/engineering/convolutional-layer.

37. CNN | Introduction to Pooling Layer. Retrieved from https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/.

38. Fashion MNIST. Retrieved from https://www.kaggle.com/datasets/zalando-research/fashionmnist.

39. Training, Testing & Deployment of a Classification Model Using Convolutional Neural Networks and Machine Learning Classifiers. Retrieved from https://www.turing.com/kb/training-testing-deployment-of-classification-model-using-convolutional-neural-networks-and-machine-learning-classifiers.

40. How do you interpret the output of a neural network?. Retrieved from https://www.quora.com/How-do-you-interpret-the-output-of-a-neural-network.

41. Convolutional Neural Network: Benefits, Types, and Applications. Retrieved from https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network/#.

ДОДАТОК А
ПЕРЕЛІК КОПІЙ ДЕМОНСТРАЦІЙНОГО МАТЕРІАЛУ

Презентація магістерської роботи

На тему: «Використання згорткових нейронних мереж для розпізнавання зображення»

Студента групи ІКК 2.1
Баришовця Артема Олександровича

ВСТУП

В сучасному світі непередбачуваного росту технологій та зв'язку, кожною секундою генерується надзвичайно велика кількість даних.

З урахуванням цього потоку даних стає критично важливим розвивати та вдосконалювати методи обробки інформації.

Однією з перспективних галузей розвитку є сфера комп'ютерного зору та розпізнавання об'єктів на зображеннях. Завдяки великій кількості доступної візуальної інформації, виникає необхідність у високоефективних методах автоматизованого розпізнавання та обробки зображень.

Види нейромереж

— **Прямого поширення (Feedforward Neural Networks, FNN) -** Інформація переміщується лише в одному напрямку, від вхідних вузлів до вихідних.

— **Зворотнього поширення (Backpropagation Neural Networks, BPNN) -** Метод зворотнього поширення допомагає мережі коригувати ваги відповідно до очікуваних вихідних значень.

— **Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) -** Має здатність зберігати інформацію в попередніх входах, використовуючи зв'язки між нейронами.

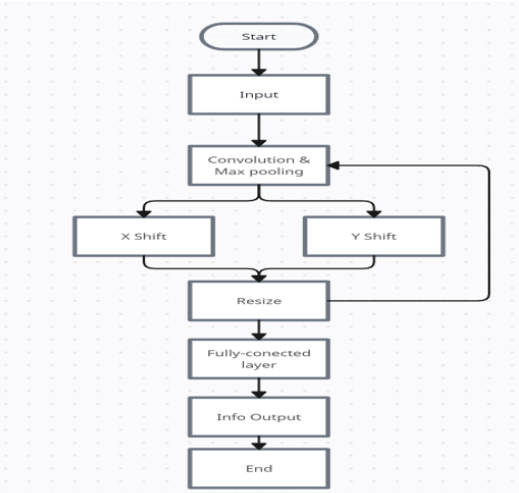
— **Згорткові нейронні мережі (Convolutional Neural Networks, CNN) -** Використовують згортку для зменшення кількості параметрів та спільного використання ваг.

— **Мережі довготривалої короткотривалої пам'яті (Long Short-Term Memory Networks, LSTM) -** Вирішують проблему зниклих градієнтів у рекурентних мережах і дозволяють враховувати довгострокові залежності у даних.

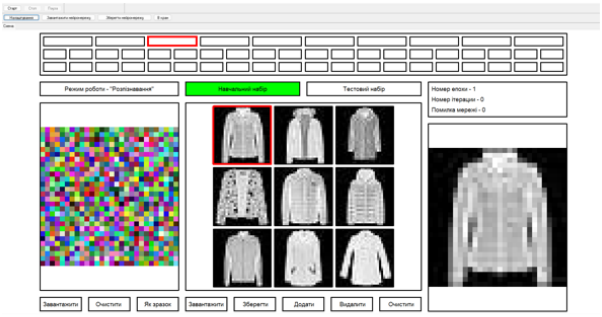
— **Рекурсивна нейронна мережа (Recursive Neural Network, Recursive NN) -** Використовується для обробки структурованих даних, таких як дерева або графи.

Згорткова нейронна мережа

- Вхідні дані
- Перший згортковий шар
- Шар пулінгу
- Додаткові згорткові та пулінгові шари
- Повністю зв’язаний шар
- Шари виводу

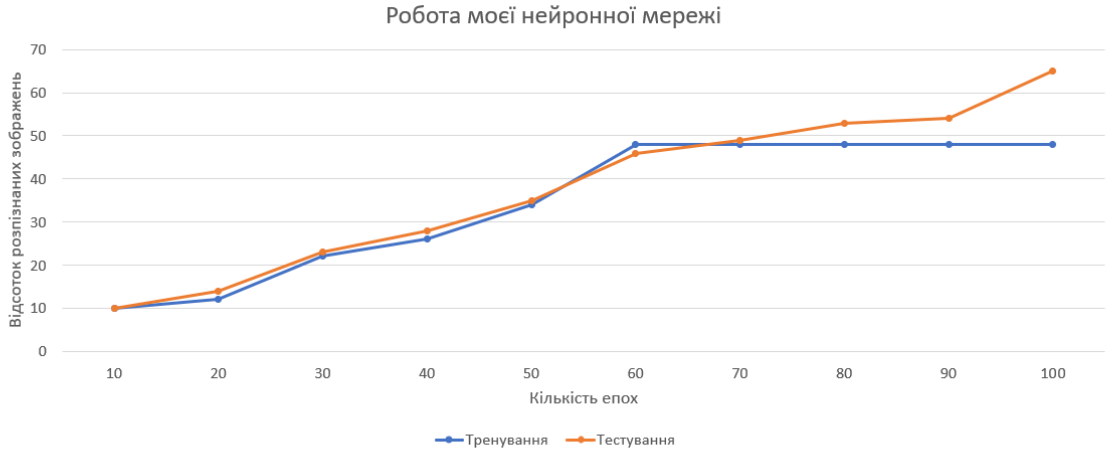


Інтерфейс



- Завантаження зображення
- Активність нейронів
- Зміна режимів
- Основна звітність мережі

Результати дослідження



Висновки

- CNN є потужним інструментом у сфері розпізнавання зображень, особливо в комп'ютерному зорі та обробці зображень.
- Використання нейронних мереж для розпізнавання зображень актуальне у зв'язку з тенденціями в області обробки зображень та широким застосуванням у сферах штучного інтелекту.
- CNN використовуються для різноманітних завдань, включаючи розпізнавання облич, класифікацію тварин та транспортних засобів у реальному часі, сегментацію зображень та інші.
- Згорткові нейронні мережі мають великий потенціал для подальшого вдосконалення та розширення застосувань у майбутньому в контексті розвитку технологій розпізнавання зображень.

Дякую що прослухали мою презентацію

ДОДАТОК Б
ЛІСТИНГ ПРОГРАМИ

Лістинг файлу uNeuronNetwork

```
unit uNeuronNetwork;

interface

uses
  Winapi.Windows,
  System.Types,
  System.SysUtils,
  System.Classes,
  System.Math,
  System.UITypes,
  VCL.Graphics,
  VCL.Dialogs,
  VCL.Forms,
  VCL.Imaging.jpeg,
  VCL.Imaging.GIFImg,
  VCL.Imaging.pngimage,
  uDLLTools;

type
  TNeuronInputs = array of PDouble;
  TNeuronWeights = array of Double;
  TActivationFunctionParams = array of Double;
  TActivationFunction = function(S: Double; Params: TActivationFunctionParams): Double;
TNeuron = class(TObject)
  private
    FTrainingSamples: TNeuronSampleList;
    FTestSamples: TNeuronSampleList;
    FColor: TColor;
    FInputs: TNeuronInputs;
    FWeights: TNeuronWeights;
```

```
  FWeight: Double; // додайте вагу для нейрона
  FActivationFunction: TActivationFunction;
  FActivationFunctionParams: TActivationFunctionParams;
  FFocused: Boolean;
  FError: Double;

  function GetInputCount: Integer;
  procedure SetInputCount(const Value: Integer);
  function GetInput(Index: Integer): PDouble;
  procedure SetInput(Index: Integer; const Value: PDouble);
  function GetWeight(Index: Integer): Double;
  procedure SetWeight(Index: Integer; const Value: Double);
  function GetActivationFunctionParams(Index: Integer): Double;
  procedure SetActivationFunctionParams(Index: Integer; const Value: Double);
  function GetActivationFunctionParamCount: Integer;
  procedure SetActivationFunctionParamCount(const Value: Integer);
  function GetWeightCount: Integer;
protected
  function GetColor: TColor;
public
  Output: Double;
  property Error: Double read FError write FError;
  property TrainingSamples: TNeuronSampleList read FTrainingSamples;
  property TestSamples: TNeuronSampleList read FTestSamples;
  property Color: TColor read FColor write FColor;
  property CurrentColor: TColor read GetColor;
  property Focused: Boolean read FFocused write FFocused;
  property InputCount: Integer read GetInputCount write SetInputCount;
  property Inputs[Index: Integer]: PDouble read GetInput write SetInput;
  property Weights[Index: Integer]: Double read GetWeight write SetWeight;
  property WeightCount: Integer read GetWeightCount;
  property ActivationFunctionParamCount: Integer
    read GetActivationFunctionParamCount write SetActivationFunctionParamCount;
  property ActivationFunction: TActivationFunction read FActivationFunction
    write FActivationFunction;
```

```
    property ActivationFunctionParams[Index: Integer]: Double
      read GetActivationFunctionParams write SetActivationFunctionParams;

    constructor Create;
    destructor Destroy; override;
    procedure Calculate(aToLog: Boolean);

    procedure SaveToStream(F: TFileStream);
    function LoadFromStream(F: TFileStream): Boolean;

TNeuronNetwork = class(TObject)
  private
    FLayers: TNeuronLayerList;
    FScreen: TBitmap;
    FScreenImageFileName: string;
    FFileName: String;
    function GetOutputLayer: TNeuronLayer;
    function GetInputLayer: TNeuronLayer;

  public
    // выходы екрана
    ScreenOutputs: array of Double;
    BiasNeuroun: Double;
    property Layers: TNeuronLayerList read FLayers;
    property OutputLayer: TNeuronLayer read GetOutputLayer;
    property InputLayer:  TNeuronLayer read GetInputLayer;
    property FileName: String read FFileName write FFileName;
    property Screen: TBitmap read FScreen;
    constructor Create;
    destructor Destroy; override;
    procedure ClearScreen;
    procedure CopyToScreen(aSource: TBitmap);

    procedure SetScreenDimentions(aWidth, aHeight: Integer);
    procedure LoadScreenFromFile;
```

```
  procedure SetScreenOutputs;
  procedure Calculate;
  procedure SaveToStream(F: TFileStream);
  function LoadFromStream(F: TFileStream): Boolean;
  procedure SaveToFile(ShowDialog: Boolean);
  function LoadFromFile(ShowDialog: Boolean): Boolean;
end;
// ###########################################################################

TConvolutionLayer = class
private
FScreen: TBitmap;

public
  procedure ApplyConvolution(FScreen: TBitmap);
  procedure ApplyPooling;
end;
// ###########################################################################
// Клас для представлення повнозв'язаного шару нейронні мережі
TFullyConnectedLayer = class
private
  FNeurons: TNeuronList; // Список нейронів у шарі
public
  constructor Create;
 destructor Destroy; override;
  // Метод для збереження шару в потік
  procedure SaveToStream(F: TFileStream);
  // Метод для завантаження шару з потоку
  function LoadFromStream(F: TFileStream): Boolean;
  // Метод для виконання операцій повнозв'язаного шару, отримуючи результат пулінгу
  procedure ApplyFullyConnectedLayer(const PooledBitmap: TBitmap);
end;
// ###########################################################################

TOutputLayer = class(TFullyConnectedLayer)
```

```
  private
   FNeuronCount: Integer;
   FNeurons: TNeuronList;

  public
    constructor Create(AInputSize, ANeuronCount: Integer);
    destructor Destroy; override;
    property NeuronCount: Integer read FNeuronCount;
    property Neurons: TNeuronList read FNeurons;
  end;
  // ###########################################################################
  // функции активации нейронов
  function SigmaActivationFunction(X: Double; Params: TActivationFunctionParams): Double;
  function LinearActivationFunction(S: Double; Params: TActivationFunctionParams): Double;
  function StepperActivationFunction(S: Double; Params: TActivationFunctionParams):
Double;
  function PseudoRectifieActivationFunction(S: Double; Params: TActivationFunctionParams):
Double;
implementation
const
  MAX_BYTE: Byte = 255;
  SAMPLE_SET_FILE_EXTENTION = '.smpl';
  NETWORK_SETTING_FILE_EXTENTION = '.ann';
// ###########################################################################

procedure AddToLog(aFileName, S: String);
var
  F: TextFile;
begin
//  if Round(gCreateLog^) <> 1 then
//    Exit;
  if S = '' then
    Exit;
  AssignFile(F, aFileName);
  try
```

```
    try
      if FileExists(aFileName) then Append(F)
        else Rewrite(F);
      Writeln(F, S);
    except
    end;
  finally
    CloseFile(F);
  end;
end;

function RandomColor: TColor;
var
  R, G, B: Byte;
begin
  R := RandomRange(0, MAX_BYTE);
  G := RandomRange(0, MAX_BYTE);
  B := RandomRange(0, MAX_BYTE);

  Result := RGB(R, G, B);
end;
    property Weight: Double read FWeight write FWeight; // властивість для доступу до ваги
  end;
// ###########################################################################
          Деякі функції активації
// ###########################################################################
// Сігмоїда
function SigmaActivationFunction(X: Double;
  Params: TActivationFunctionParams): Double;
begin
  Result := 1 / (1 + Exp(-X));
end;
// ###########################################################################
// лінійна
function LinearActivationFunction(S: Double;
```

```
  Params: TActivationFunctionParams): Double;
begin
  Result := S;
end;
// ############################################################################
// порогова
function StepperActivationFunction(S: Double;
  Params: TActivationFunctionParams): Double;
begin
  Result := 0;
  if Length(Params) > 0 then
    if S > Params[0]  then
      Result := 1;
end;
// ############################################################################
// псевдовиправлення
function PseudoRectifieActivationFunction(S: Double;
  Params: TActivationFunctionParams): Double;
begin
  if S >= 0 then
    Result := S
  else
    Result := 0;
end;
// ############################################################################

{ TNeuornNetwork }
constructor TNeuronNetwork.Create;
begin
  BiasNeuroun := 1;
  ScreenOutputs := nil;
  FLayers := TNeuronLayerList.Create;
  FScreen := TBitmap.Create;
  FScreen.PixelFormat := pf24bit;
  FFileName := 'Default' + NETWORK_SETTING_FILE_EXTENTION;
```

```
end;
destructor TNeuronNetwork.Destroy;
begin
  FScreen.Free;
  FLayers.Count := 0;
  FLayers.Free;
  ScreenOutputs := nil;
  inherited;
end;
function TNeuronNetwork.GetInputLayer: TNeuronLayer;
begin
  if FLayers.Count > 0 then
    Result := FLayers.First
  else
    Result := nil;
end;
function TNeuronNetwork.GetOutputLayer: TNeuronLayer;
begin
  if FLayers.Count > 0 then
    Result := FLayers.Last
  else
    Result := nil;
end;

function TNeuronNetwork.LoadFromFile(ShowDialog: Boolean): Boolean;
var
  S: String;
  D: TOpenDialog;
  F: TFileStream;
  CanShowMessage: Boolean;
begin
  Result := True;
  CanShowMessage := ShowDialog;
  if ShowDialog then
  begin
```

```
  D := TOpenDialog.Create(nil);
  try
    if FFileName <> '' then
      D.FileName := FFileName;
    case gLanguage of
      lgEng: S := 'Neural network settings';
      lgRus: S := 'Настройки нейросети';
      else S := 'Налаштування нейронні мережі';
    end;
    D.Filter := S + ' (*' + NETWORK_SETTING_FILE_EXTENTION + ') |*'
      + NETWORK_SETTING_FILE_EXTENTION;
    if D.Execute(Application.Handle) then
    begin
      FFileName := D.FileName;
      if ExtractFileExt(FFileName) = '' then
        FFileName := FFileName + NETWORK_SETTING_FILE_EXTENTION;
    end else
    begin
      Result := False;
      CanShowMessage := False;
    end;
  finally
    D.Free;
  end;
end;
if Result then
  Result := FileExists(FFileName);
if Result then
begin
  F := TFileStream.Create(FFileName, fmOpenRead);
  try
    Result := LoadFromStream(F);
  finally
    F.Free;
  end;
```

```
  end;

  if CanShowMessage and (not Result) then
  begin
    case gLanguage of
      lgEng: S := 'The file is missing or damaged';
      lgRus: S := 'Файл отсуствует или поврежден';
      else S := 'Файл вітсутній або пошкоджений';
    end;
    ShowMessage(S);
  end;
end;

function TNeuronNetwork.LoadFromStream(F: TFileStream): Boolean;
begin
  Result := Assigned(F);
  if Result then
    Result := FLayers.LoadFromStream(F);
end;
procedure TNeuronNetwork.LoadScreenFromFile;
var
  D: TOpenDialog;
  S: String;
  aGraphic: TGraphic;
  aRect: TRect;
begin
  aGraphic := nil;

  D := TOpenDialog.Create(nil);
  try
    if FileExists(FScreenImageFileName) then
      D.FileName := FScreenImageFileName;

    case gLanguage of
      lgEng: S := 'Image files';
```

```
  lgRus: S := 'Файлы изображений';
  else S := 'Файли зображень';
end;
D.Filter := S + ' (*.bmp; *.jpg; *.jpeg; *.png) |*.bmp; *.jpg; *.jpeg; *.png';

if D.Execute(Application.Handle) then
begin
  FScreenImageFileName := D.FileName;
  if FileExists(FScreenImageFileName) then
  begin
    try
      S := AnsiUpperCase(ExtractFileExt(FScreenImageFileName));

      if S = '.BMP' then
        aGraphic := TBitmap.Create;
      if S = '.JPG' then
        aGraphic := TJPEGImage.Create;
      if S = '.JPEG' then
        aGraphic := TJPEGImage.Create;
      if S = '.PNG' then
        aGraphic := TPngImage.Create;

      if Assigned(aGraphic) then
      begin
        aGraphic.LoadFromFile(FScreenImageFileName);

        aRect.Left := 0;
        aRect.Top := 0;
        aRect.Right := FScreen.Width;
        aRect.Bottom := FScreen.Height;

        FScreen.Canvas.StretchDraw(aRect, aGraphic);
      end;
    except
      on E: Exception do
```

```
          begin
            case gLanguage of
              lgEng: S := 'Error opening file:' + #13 + E.Message;
              lgRus: S := 'Ошибка при открытии файла:' + #13 + E.Message;
              else S := 'Помилка під час відкриття файлу:' + #13 + E.Message;
            end;
           ShowMessage(S);
          end;
        end;
      end;
    end;
  finally
    D.Free;

    if Assigned(aGraphic) then
      aGraphic.Free;
  end;
end;

procedure TNeuronNetwork.Calculate;
var
  i, j: Integer;
begin
  SetScreenOutputs;
  for i := 0 to FLayers.Count - 1 do
  begin
    for j := 0 to FLayers[i].Neurons.Count - 1 do
    begin
      FLayers[i].Neurons[j].Calculate(i > 0);
     end;
  end;
end;

procedure TNeuronNetwork.ClearScreen;
var
```

```
  X, Y: Integer;
begin
  for Y := 0 to FScreen.Height - 1 do
    for X := 0 to FScreen.Width - 1 do
      FScreen.Canvas.Pixels[X, Y] := RandomColor;
end;

procedure TNeuronNetwork.CopyToScreen(aSource: TBitmap);
var
  aRect: TRect;
begin
  if not Assigned(aSource) then
    Exit;

  aRect.Left := 0;
  aRect.Top := 0;
  aRect.Right := FScreen.Width;
  aRect.Bottom := FScreen.Height;

  FScreen.Canvas.StretchDraw(aRect, aSource);
end;

procedure TNeuronNetwork.SaveToFile(ShowDialog: Boolean);
var
  S: String;
  D: TSaveDialog;
  F: TFileStream;
  CanSaveNetwork: Boolean;
begin
  CanSaveNetwork := True;

  if ShowDialog then
  begin
    D := TSaveDialog.Create(nil);
    try
```

```
    D.Options := D.Options + [ofOverwritePrompt];

    if FileExists(FFileName) then
      D.FileName := FFileName;

    case gLanguage of
      lgEng: S := 'Neural network settings';
      lgRus: S := 'Настройки нейросети';
      else S := 'Налаштування нейронні мережі';
    end;

    D.Filter := S + ' (*' + NETWORK_SETTING_FILE_EXTENTION + ') |*'
      + NETWORK_SETTING_FILE_EXTENTION;

    if D.Execute(Application.Handle) then
    begin
      FFileName := D.FileName;

      if ExtractFileExt(FFileName) = '' then
        FFileName := FFileName + NETWORK_SETTING_FILE_EXTENTION;
    end else
      CanSaveNetwork := False;
  finally
    D.Free;
  end;
end;

if CanSaveNetwork then
begin
  F := TFileStream.Create(FFileName, fmCreate);
  try
    SaveToStream(F);
  finally
    F.Free;
  end;
```

```
  end;
end;

procedure TNeuronNetwork.SaveToStream(F: TFileStream);
begin
  if not Assigned(F) then
    Exit;
  FLayers.SaveToStream(F);
end;

procedure TNeuronNetwork.SetScreenDimentions(aWidth, aHeight: Integer);
begin
  if (aWidth = FScreen.Width) and (aHeight = FScreen.Height) then
    Exit;

  if (aWidth < 0) or (aHeight < 0) then
    Exit;

  FScreen.Width := aWidth;
  FScreen.Height := aHeight;
  ClearScreen;

  SetLength(ScreenOutputs, aWidth * aHeight);
end;

procedure TNeuronNetwork.SetScreenOutputs;
var
  X, Y, Index: Integer;
  R, G, B: Byte;
begin

  Index := 0;

  for Y := 0 to FScreen.Height - 1 do
    for X := 0 to FScreen.Width - 1 do
```

```
    begin
      R := GetRValue(FScreen.Canvas.Pixels[X, Y]);
      G := GetGValue(FScreen.Canvas.Pixels[X, Y]);
      B := GetBValue(FScreen.Canvas.Pixels[X, Y]);

      if Index <= High(ScreenOutputs) then
        ScreenOutputs[Index] := (R + G + B) / (3 * MAX_BYTE);

      Inc(Index);
    end;
end;

{ TNeuron }

procedure TNeuron.Calculate (aToLog: Boolean);
var
  i: Integer;
  S: Double;
begin
  Output := 0;

  S := 0;
  for i := 0 to High(FInputs) do
    if Assigned(FInputs[i]) then
      S := S + FInputs[i]^ * FWeights[i];

  if Assigned(FActivationFunction) then
    Output := FActivationFunction(S, FActivationFunctionParams);


//   if aToLog then
//     AddToLog('1.txt', 'W = ' + FloatToStr());

end;
```

```
constructor TNeuron.Create;
begin
  FTrainingSamples := TNeuronSampleList.Create;
  FTestSamples := TNeuronSampleList.Create;

  FColor := TColors.Red;
  Output := 0;
  FFocused := False;

  FInputs := nil;
  FWeights := nil;
  FActivationFunctionParams := nil;
  FActivationFunction := @LinearActivationFunction;
end;

destructor TNeuron.Destroy;
begin
  FTrainingSamples.DeleteAll;
  FTrainingSamples.Free;

  FTestSamples.DeleteAll;
  FTestSamples.Free;

  FInputs := nil;
  FWeights := nil;
  FActivationFunctionParams := nil;

  inherited;
end;

procedure TNeuron.SetWeight(Index: Integer; const Value: Double);
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    FWeights[Index] := Value;
end;
```

```
function TNeuron.GetActivationFunctionParamCount: Integer;
begin
  Result := Length(FActivationFunctionParams);
end;

function TNeuron.GetActivationFunctionParams(Index: Integer): Double;
begin
  if (Index >= 0) and (Index < Length(FActivationFunctionParams)) then
    Result := FActivationFunctionParams[Index]
  else
    Result := 0;
end;

function TNeuron.GetColor: TColor;
const
  MIN_VALUE = 0;
  MAX_VALUE = 1;
var
  R, G, B: Byte;
begin
  R := Round(GetRValue(FColor) * Output / (MAX_VALUE - MIN_VALUE));
  G := Round(GetGValue(FColor) * Output / (MAX_VALUE - MIN_VALUE));
  B := Round(GetBValue(FColor) * Output / (MAX_VALUE - MIN_VALUE));

  Result := RGB(R, G, B);
end;

function TNeuron.GetInput(Index: Integer): PDouble;
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    Result := FInputs[Index]
  else
    Result := nil;
end;
```

```
function TNeuron.GetInputCount: Integer;
begin
  Result := Length(FInputs);
end;

function TNeuron.GetWeight(Index: Integer): Double;
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    Result := FWeights[Index]
  else
    Result := 0;
end;

function TNeuron.GetWeightCount: Integer;
begin
  Result := Length((FWeights));
end;

function TNeuron.LoadFromStream(F: TFileStream): Boolean;
var
  i, aCount, aReadByteCount: Integer;
begin
  Result := Assigned(F);

  // цвет
  if Result then
  begin
    aReadByteCount := F.Read(FColor, SizeOf(FColor));
    Result := aReadByteCount = SizeOf(FColor);
  end;

  // количество весовых коэффициентов
  if Result then
  begin
```

```
    aReadByteCount := F.Read(aCount, SizeOf(aCount));
    Result := aReadByteCount = SizeOf(aCount);
  end;

  if Result then
    Result := aCount = Length(FWeights);

  // весовые коеффициенты
  if Result then
    for i := 0 to aCount - 1 do
    begin
      aReadByteCount := F.Read(FWeights[i], SizeOf(FWeights[i]));
      Result := aReadByteCount = SizeOf(FWeights[i]);

      if not Result then
        Break;
    end;

  // учебная выборка
  if Result then
    Result := FTrainingSamples.LoadFromStream(F);

  // тестовая выборка
  if Result then
    Result := FTestSamples.LoadFromStream(F);
end;

procedure TNeuron.SaveToStream(F: TFileStream);
var
  i, aCount: Integer;
begin
  if not Assigned(F) then
    Exit;

  // Цвет
```

```
  F.Write(FColor, SizeOf(FColor));

  // кількість вагових коефіцієнтів
  aCount := Length(FWeights);
  F.Write(aCount, SizeOf(aCount));

  // впшові коеффцієнти
  for i := 0 to aCount - 1 do
    F.Write(FWeights[i], SizeOf(FWeights[i]));

  // учебная выборка
  FTrainingSamples.SaveToStream(F);

  // тестовая выборка
  FTestSamples.SaveToStream(F);
end;

procedure TNeuron.SetActivationFunctionParamCount(const Value: Integer);
var
  i: Integer;
begin
  if Value = Length(FActivationFunctionParams) then
    Exit;

  if Value >= 0 then
  begin
    SetLength(FActivationFunctionParams, Value);

    for i := 0 to High(FActivationFunctionParams) do
      FActivationFunctionParams[i] := 0;
  end;
end;

procedure TNeuron.SetActivationFunctionParams(Index: Integer;
  const Value: Double);
```

```
begin
  if (Index >= 0) and (Index < Length(FActivationFunctionParams)) then
    FActivationFunctionParams[Index] := Value;
end;

procedure TNeuron.SetInput(Index: Integer; const Value: PDouble);
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    FInputs[Index] := Value;
end;

procedure TNeuron.SetInputCount(const Value: Integer);
var
  i: Integer;
begin
  if Value = Length(FInputs) then
    Exit;

  if Value >= 0 then
  begin
    SetLength(FInputs, Value);
    SetLength(FWeights, Value);
  end;

  for i := 0 to High(FInputs) do
  begin
    FInputs[i] := nil;
    FWeights[i] := 0;
  end;
end;
{ TConvolutionLayer }

procedure TConvolutionLayer.ApplyConvolution;
var
  Source: TBitmap;
```

```
  Filter: TBitmap;
  X, Y, I, J: Integer;
  PixelColor: TColor;
  FilterValue, ResultColor: Integer;
begin
  if not Assigned(FScreen) then
    Exit;

  Source := TBitmap.Create;
  try
    Source.Assign(FScreen); // Створюємо копію вхідного зображення

    Filter := TBitmap.Create;
    try
      // Ініціалізуємо Filter - встановлюємо його розмір і значення пікселів фільтра
      // З цієї точки ви можете встановити власні значення фільтра
      Filter.SetSize(3, 3);
      // Наприклад, встановимо значення фільтра для розмиття
      Filter.Canvas.Pixels[0, 0] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[0, 1] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[0, 2] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[1, 0] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[1, 1] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[1, 2] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[2, 0] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[2, 1] := RGB(1, 1, 1);
      Filter.Canvas.Pixels[2, 2] := RGB(1, 1, 1);

      // Застосування згорткового фільтра
      for X := 1 to Source.Width - 2 do
        for Y := 1 to Source.Height - 2 do
        begin
          // Застосування фільтра до пікселя (X, Y) на вхідному зображенні
          ResultColor := 0;
```

```
        for I := 0 to 2 do
          for J := 0 to 2 do
          begin
            PixelColor := Source.Canvas.Pixels[X - 1 + I, Y - 1 + J];
            FilterValue := Filter.Canvas.Pixels[I, J] and $FF; // отримати "чистий" колір

            // Додати ваговане значення пікселя та ваги фільтра до результату
            ResultColor := ResultColor + (GetRValue(PixelColor) * FilterValue);
          end;

        // Зберегти результат в оригінальному зображенні (FScreen)
        FScreen.Canvas.Pixels[X, Y] := RGB(ResultColor, ResultColor, ResultColor);
      end;
    finally
      Filter.Free;
    end;
  finally
    Source.Free;
  end;
end;

procedure TConvolutionLayer.ApplyPooling;
var
  X, Y, I, J: Integer;
  MaxValue: Integer;
  Source: TBitmap;
  Destination: TBitmap;
begin
  if not Assigned(FScreen) then
    Exit;

  Source := TBitmap.Create;
  try
    Source.Assign(FScreen); // Створюємо копію вхідного зображення
```

```
    Destination := TBitmap.Create;
    try
      // Розмір області пулінгу (зазвичай 2x2 або 3x3)
      for X := 0 to Source.Width div 2 - 1 do
        for Y := 0 to Source.Height div 2 - 1 do
        begin
          MaxValue := 0;

          // Знаходження максимального значення у кожній області пулінгу
          for I := 0 to 1 do
            for J := 0 to 1 do
            begin
              MaxValue := Max(MaxValue, GetRValue(Source.Canvas.Pixels[X * 2 + I, Y * 2 +
J]));
              // Ви можете вибрати інший канал або застосовувати пулінг до всіх каналів RGB
            end;

          // Результат зберігається у Destination.Canvas.Pixels[X, Y]
          Destination.Canvas.Pixels[X, Y] := RGB(MaxValue, MaxValue, MaxValue);
        end;

      // Копіювати результат обробки назад у вихідне зображення (FScreen)
      FScreen.Assign(Destination);
    finally
      Destination.Free;
    end;
  finally
    Source.Free;
  end;
end;



{ TFullyConnectedLayer }
```

```
constructor TFullyConnectedLayer.Create;

const NumberOfNeurons = 10; // Встановіть бажану кількість нейронів

var
  i: Integer;
  Neuron: TNeuron;
begin
  FNeurons := TNeuronList.Create;


  // Ініціалізація нейронів та їх параметрів
  for i := 0 to NumberOfNeurons - 1 do
  begin
    Neuron := TNeuron.Create;

    // Ініціалізація ваг нейронів
    // Тут ви можете встановити ваги за своїми умовами або отримувати їх з іншого джерела
    Neuron.Weight := Random; // Наприклад, ініціалізація ваги нейрона випадковим значенням

    FNeurons.Add(Neuron);
  end;
end;

destructor TFullyConnectedLayer.Destroy;
begin

  FNeurons.Free; // Звільнення списку нейронів

  inherited;
end;
```

```
procedure TFullyConnectedLayer.SaveToStream(F: TFileStream);
begin
  // Збереження нейронів шару в потік
  FNeurons.SaveToStream(F);
end;

function TFullyConnectedLayer.LoadFromStream(F: TFileStream): Boolean;
begin
  // Завантаження нейронів шару з потоку
  Result := FNeurons.LoadFromStream(F);
end;

procedure TFullyConnectedLayer.ApplyFullyConnectedLayer(const PooledBitmap: TBitmap);
var
  X, Y: Integer;
  Neuron: TNeuron;
  OutputValue: Double;
begin
  // Ітерація по нейронам шару
  for Y := 0 to PooledBitmap.Height - 1 do
    for X := 0 to PooledBitmap.Width - 1 do
    begin
      // Отримання нейрона, який відповідає певному пікселю
      Neuron := FNeurons[Y * PooledBitmap.Width + X];

      // Виконання операцій над пікселем та отримання виходного значення
      // Наприклад, можна помножити значення пікселю на вагу нейрона
      OutputValue := PooledBitmap.Canvas.Pixels[X, Y] * Neuron.Weight;

      // Далі можна використовувати OutputValue за потреби
      // Наприклад, передати його іншому шару чи вивести результат
      // ...
    end;
end;
```

```
{ TOutputLayer }

constructor TOutputLayer.Create(AInputSize, ANeuronCount: Integer);
var
  i: Integer;
  Neuron: TNeuron;
begin
  // Виклик конструктора базового класу (TFullyConnectedLayer) для налаштування входу та кількості нейронів
  inherited Create;

  // Ініціалізація параметрів нейронів виходового шару, наприклад, їх ваг
  FNeuronCount := ANeuronCount;
  FNeurons := TNeuronList.Create;

  for i := 0 to NeuronCount - 1 do
  begin
    Neuron := TNeuron.Create;
    Neuron.Weight := Random;
    FNeurons.Add(Neuron);
  end;
end;


destructor TOutputLayer.Destroy;
var
  i: Integer;
begin
  // Звільнення ресурсів, пов'язаних із кожним нейроном в вихідному шарі
  for i := 0 to NeuronCount - 1 do
    Neurons[i].Free;

  // Звільнення списку нейронів
```

```
    Neurons.Free;

    // Виклик деструктора базового класу (TFullyConnectedLayer), щоб звільнити його ресурси
    inherited;
  end;

  end.
```