МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Маключенко Сергій Ігорович

Програмне забезпечення системи розпізнавання зображень на основі нейронних мереж

Спеціальність: 123 «Комп'ютерна інженерія»

Освітній ступінь: бакалавр

Науковий керівник:

Мелешко Єлизавета Владиславівна _____

(підпис) (дата)

доктор технічних наук, доцент

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« 21 » грудня 2020 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Маключенку Сергію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи розпізнавання зображень на основі нейронних мереж*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 189-02 від 28.12.2020 року

2. Строк подання студентом роботи до захисту *22.05.2021 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи розпізнавання зображень на основі нейронних мереж*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « 21 » грудня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2021 р.	
3.	Розробка моделі компонента	20.03.2021 р.	
4.	Розробка структур даних	25.03.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2021 р.	
6.	Програмування алгоритмів	10.04.2021 р.	
7.	Оформлення ПЗ	17.04.2021 р.	
8.	Попередній захист роботи	22.05.2021 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Маключенко С.І. Програмне забезпечення системи розпізнавання зображень на основі нейронних мереж. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для системи розпізнавання зображень на основі нейронних мереж.

Метою роботи є створення системи розпізнавання зображень на основі нейронних мереж.

Результат роботи – програмна реалізація системи розпізнавання зображень на основі нейронних мереж.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10.

Програму розроблено на мові програмування C++.

Ключові слова: комп'ютерна інженерія, нейронні мережі, розпізнавання зображень

ABSTRACT

Makliuchenko S.I. Software of the system of image recognition based on neural networks. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2021

In this bachelor's qualification work, software that is designed for system of image recognition based on neural networks has been developed.

The purpose of the development is the software of the system of image recognition based on neural networks.

The result of the work is the software implementation of the system of image recognition based on neural networks.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

A user-friendly interface is developed. The instructions for working with the software are provided.

The program can be used on an IBM PC running Windows 10.

The software is developed in C++ programming language.

Keywords: computer engineering, neural networks, image recognition

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ…..2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи..... 5
1.2 Область застосування..... 6
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 8
2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи..... 8
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування 16
2.3 Розгорнута постановка завдання 17
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 19
3.1 Опис функціонування системи 19
3.2 Розробка структурної схеми..... 38
3.3 Розробка функціональної схеми 39
3.4 Розробка діаграми процесів 40
4 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ 42
4.1 Блок-схеми та опис алгоритмів функціонування системи..... 42
4.2 Захист розробленого програмного забезпечення..... 53
5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ 55
6 ОСНОВНІ ВИСНОВКИ 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

Back Propagation	–	нейронна мережа зворотного поширення похибки.
OLAP	–	(англ. online analytical processing) аналітична обробка у реальному часі
SURF	–	(англ. Speeded Up Robust Features) алгоритм SURF для визначення дескрипторів особливих точок на зображенні.
Гессіан	–	детермінант матриці Гессе.
НМ	–	нейронна мережа.
ОС	–	операційна система.

ВСТУП

Актуальність теми. Розпізнавання зображень – інформаційна технологія, створена для отримання, обробки, аналізу і класифікації графічних образів. Необхідність в розпізнаванні зображень виникає в самих різних областях людської діяльності – від автоматизації пошуку розважального мультимедійного контенту до розшифровки кардіограми чи розпізнавання можливих загроз безпеці на камерах стеження.

В наш час розпізнавання зображень – одна з основних і широко використовуваних задач комп'ютерного зору. Розпізнавання образів на зображеннях і одержання ознак також виступають важливою частиною інших, більш складних методів комп'ютерного зору, таких як виявлення об'єктів і сегментація зображень.

Як правило задачі розпізнавання зображень вирішуються за допомогою нейронних мереж.

Таким чином, розробка програмного забезпечення системи розпізнавання зображень на основі нейронної мережі є актуальною задачею, яка потребує вирішення у даній магістерській роботі.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи розпізнавання зображень на основі нейронних мереж.

Для досягнення мети роботи необхідно вирішити наступні задачі:

– Дослідження існуючих систем розпізнавання зображень на основі нейронних мереж.

– Проектування системи розпізнавання зображень на основі нейронних мереж.

– Програмна реалізація системи розпізнавання зображень на основі нейронних мереж.

Практичне значення одержаних результатів у роботі полягає у наступному – розроблене програмне забезпечення для розпізнавання зображень на основі нейронних мереж.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи розпізнавання зображень на основі нейронних мереж, є актуальною задачею, що потребує вирішення у цій кваліфікаційній бакалаврській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

У роботі було розроблено програмне забезпечення, що призначене для реалізації системи розпізнавання цифрових зображень на основі нейронних мереж.

Головною сферою призначення розробленої системи є класифікація зображень та пошук схожих зображень.

Класифікація зображень дозволяє віднести досліджуване зображення до певної категорії об’єктів, що може бути застосоване у наступних застосунках:

– Автоматичний контроль, наприклад, у виробничих цілях;

– Надання допомоги людям в задачах ідентифікації, наприклад, системі ідентифікації видів;

– Процеси управління, наприклад, промисловими роботами;

– Виявлення подій, наприклад, для візуального спостереження або підрахунку людей;

– Завдання взаємодії, наприклад, для пристроїв взаємодії людини з комп'ютером;

– Моделювання об'єктів або середовищ, наприклад, аналіз медичних зображень або топографічне моделювання;

– Навігація, наприклад, автономним транспортним засобом або мобільним роботом;

– Організація інформації, наприклад, для індексування баз даних зображень і послідовностей зображень.

Для пошуку зображень за зображенням-шаблоном, необхідно здійснювати розпізнавання зображень для одержання набору ключових слів, за якими потім буде здійснюватися порівняння і визначатися ступінь їх подоби – релевантності

пошуковому запиту. Пошук зображень може бути застосований, наприклад, в пошукових системах та мультимедійних базах даних.

1.2 Область застосування

Розпізнавання цифрових зображень можна використовувати у різних областях інформаційних технологій та автоматизації. Розглянемо основні з них:

Пошук зображень за шаблонами. Пошукові роботи можуть використовувати розпізнавання зображень для пошуку набору різних зображень схожих на зображення-шаблон.

Технічна діагностика. Контроль якості деталей – одна з актуальних задач автоматизації виробництва. Завдання полягає в тому, щоб виявити чи є дефект у деталі, чи немає. Також якщо деталь має дефект, часто потрібно визначити і тип цього дефекту.

Медична діагностика. Програмне забезпечення для розпізнавання зображень часто використовуються в медичній практиці. Наприклад, для діагностики захворювань на основі аналізу кардіограм, рентгенівських знімків, зображень ультразвукової діагностики тощо.

Розпізнавання тексту. Важливою задачею комп’ютерного зору є розпізнавання тексту. Системи розпізнавання тексту працюють, як правило, разом зі сканерами, що використовуються для введення в комп'ютер друкованих зображень. При введенні друкованого тексту сканер формує лише графічний файл, а щоб створити текстовий документ, з яким може працювати текстовий редактор, необхідно розпізнати на зображенні з файлу окремі букви. Аналогічним чином, розпізнавання букв є необхідним для підтримки пристроїв рукописного введення. Цими пристроями, зовні схожими на звичайну авторучку, часто комплектуються портативні комп'ютери (персональні помічники). Основна мета цих пристроїв – замінити уведення із клавіатури, що є незручним для багатьох користувачів.

Робототехніка. Застосування методів розпізнавання зображень у робототехніці є досить поширеним та необхідним, оскільки роботи повинні безпосередньо сприймати зовнішній світ та мати пристрої машинного зору.

Охоронні системи. Розпізнавання зображень в охоронних системах в першу чергу використовується для задач ідентифікації. Наприклад, потрібно ідентифікувати деяку особистість, щоб визначити, чи має вона право входити на територію об’єкту, що знаходиться під охороною.

Біометричні системи. Розпізнавання зображень також використовують біометричні системи ідентифікації особистості для отримання доступу до захищених систем та програм, які вирішують проблему ідентифікації відбитків пальців, сітківки ока тощо.

Таким чином, виходячи з вищеперерахованого, задача розробки програмного забезпечення системи розпізнавання зображень на основі нейронних мереж, є актуальною і потребує вирішення у цій кваліфікаційній бакалаврській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Нейронна мережа Google

У червні 2012 року група дослідників з Google запустила нейронну мережу на кластері 1000 комп'ютерів (16 тис. процесорних ядер; 1 млрд зв'язків між нейронами). Експеримент став одним із самих масштабних в області штучного інтелекту, причому систему створювали для рішення практичних завдань.

Самонавчальна нейронна мережа – досить універсальний інструмент, який можна використовувати на різних масивах даних. У компанії Google її застосували для поліпшення точності розпізнавання мови: «Ми одержали зменшення на 20-25% кількості помилок при розпізнаванні, – говорить Винсент Ванхоук (Vincent Vanhoucke), керівник відділу розпізнавання мови в Google. – Це значить, що багато з людей одержать безпомилковий результат». Нейронна мережа оптимізувала алгоритми для англійської мови, але Ванхоук говорить, що аналогічні поліпшення можуть бути досягнуті й для інших мов і діалектів.

Нейронна мережа використовується також у проекті Google Street View для обробки маленьких фрагментів фотографій, де потрібно визначити – є число на фрагменті номером чи вдома ні. Дивно, але в цьому завданні нейронна мережа показує кращу точність розпізнавання, ніж люди.

У майбутньому нейронна мережа буде використана в інших продуктах Google, таких як пошук зображень, окуляри Google Glass і автомобілі Google з безпілотним керуванням. Один зі співробітників проекту по розробці нейронної мережі Джефф Дин (Jeff Dean) говорить, що в автомобілі система може враховувати контекстну інформацію, у тому числі інформацію з лазерних далекомірів або, наприклад, звук мотора. Джефф Дин говорить, що потужна

нейронна мережа здатна використовувати багато контекстної інформації в процесі тренування – із цієї причини вони й вирішили створити такий великий кластер з 1000 серверів, у той час як більшість дослідників тестують нейронної мережі на одному комп'ютері.

Перші результати експерименту з нейронною мережею Google були опубліковані в червні 2012 року. Тести показали, що нейронна мережа успішно піддається самонавчанню. Після перегляду 10 мільйонів випадкових кадрів з Youtube у нейронної мережі сформувалися нейрони, що селективно реагують на присутність осіб на зображеннях. На думку вчених, нейронна мережа Google у процесі самонавчання працювала приблизно так само, як працюють нейрони в зоровій корі головного мозку ссавців. З тим застереженням, що нейронна мережа Google, незважаючи на свої масштаби, все таки набагато менше по кількості вузлів, ніж нейронна мережа зорової кори.

NeuroPro

Основними перевагами в порівнянні із сучасними індустріальними нейромережними й статистичними програмами є наступні:

– Потужні засоби аналізу й ефективної візуалізації даних, засоби керування даними й вибору аналізованих піднаборів (фрагментів) даних.

– Можливість ручного редагування значень окремих параметрів нейронної мережі, властивостей параметрів (завдання діапазону зміни значень, переваг до чутливості виходу нейронної мережі до варіювання параметра й т.д.) і інші способи включення експертних знань у нейронну мережу.

– Можливість відображення динаміки різних показників, спостережуваних у процесі навчання нейронної мережі або при зміні поколінь генетичного алгоритму оптимізації.

– Можливість вивчення різноманітних статичних властивостей побудованої нейронної мережі.

– Можливість вивчення властивостей видаваного нейронною мережею рішення для визначення адекватності або неадекватності нейронної мережі.

Помітимо, що це одне з найбільш важливих переваг, оскільки жодна зі стандартних нейромережних програм дотепер не надає навіть можливості дослідження факту закорельованості помилки прогнозу нейронної мережі з її вхідними (незалежними) змінними: значима кореляція є найбільш простою характеристикою неадекватності навченої нейронної мережі.

– Зіставлення різних моделей між собою на основі їхнього ранжирування по темі або інших властивостях.

– Підтримка ітеративного процесу над ланцюжком етапів "аналіз даних –> вибір конфігурації моделі –> адаптація моделі –> вивчення динаміки показників у ході адаптації моделі –> вивчення статичних властивостей рішення –> вивчення результатів впливу різних настроювань алгоритмів на властивості одержуваних при цьому моделей", що коли виникають на кожному етапі показники обробляються із залученням тих самих стандартних засобів статистичного аналізу й візуалізації даних. Отримані на кожному етапі характеристики й результати пояснюються виходячи з результатів попередніх етапів.

– Велике число реалізованих базових і спеціалізованих алгоритмів.

– Використання найбільш ефективних в обчислювальному плані алгоритмів і також якісне їхнє програмування дають максимальну швидкість роботи програм, що дозволяє обробляти величенні обсяги даних (десятки гігабайт) у найбільш складних сучасних завданнях на звичайному персональному комп'ютері.

– І ще безліч інших унікальних можливостей.

Neural Network Wizard

Програмна реалізація багатошарової нейронної мережі зворотного поширення (back propagation). Спосіб поширення – безкоштовно для некомерційного застосування.

Розвиток системи припинений. Закладені в ній ідеї одержали продовження в аналітичній платформі Deductor.

Можливості системи:

– Завантаження даних для навчання НМ з текстового файлу.

– Використання різних способів нормування даних НМ.

– Створення багатошарових НМ різної конфігурації та архітектури.

– Налаштування параметрів алгоритму навчання нейронної мережі.

– Можливість зберегти результати навчання НМ у файл.

– Автоматизація навчання нейронної мережі.

– Автоматичне формування навчальної й тестової вибірки.

– Відкриті сирцеві тексти програмного коду (з версії 1.7).

TensorFlow

TensorFlow –програмна бібліотека з відкритим програмним кодом для машинного навчання, що розроблена компанією Google. TensorFlow було початково розроблено відділом Google Brain для внутрішнього використання в компанії Google, поки її не було випущено під відкритою ліцензією Apache 2.0 у листопаді 2015 р.

TensorFlow може працювати на декількох центральних процесорах (на 64-розрядних Linux, macOS, Windows, Android, iOS тощо) та графічних процесорах (включно з додатковими розширеннями CUDA).

TensorFlow може застосовуватися для розробки програмного забезпечення автоматизованого опису зображень, такого як DeepDream.

Переваги TensorFlow:

– Простий вбудований високорівневий API.

– Багатофункціональність.

– Гарна підтримка під мобільні пристрої.

– Детальна документація.

Недоліки TensorFlow:

– Складне налагодження.

– Труднощі з внесенням змін (пов'язані з використанням статичних обчислювальних графів).

– Також до недоліків слід віднести відсутність підтримки заздалегідь навчених моделей.

Keras

Keras – відкрита бібліотека для створення нейронних мереж, написана на мові програмування Python. Є надбудовою над фреймворками Deeplearning4j, TensorFlow і Theano. Націлена на оперативну роботу з мережами глибинного навчання, при цьому спроектована так, щоб бути компактною, модульною та розширюватися. Вона була створена як частина дослідницьких зусиль проекту Open-ended Neuro-Electronic Intelligent Robot Operating System (ONEIROS), а її основним автором є Франсуа Шолль інженер Google.

Планувалося що Google буде підтримувати Keras в основний бібліотеці TensorFlow, однак Шолль виділив Keras в окрему надбудову, так як згідно з концепцією Keras є скоріше інтерфейсом, ніж наскрізною системою машинного навчання.

Keras робить акцент на призначеному для користувача досвіді. З її допомогою користувач може писати мінімум коду для виконання основних операцій. Бібліотека модульна і розширювана. Моделі і частини коду можна використовувати повторно і розширювати в майбутньому.

У Keras багато помічених наборів даних, які можна імпортувати і завантажити.

Переваги:

– Прототипування дійсно швидке і просте;

– Досить мало важить для побудови моделей глибокого навчання для множині прошарків;

– Має модулі, що повністю конфігуруються;

– Має простий і інтуїтивно-зрозумілий інтерфейс, відповідно, хороший для новачків;

– Має вбудовану підтримку для навчання на декількох GPU;

– Може бути налаштований в якості оцінювачів для TensorFlow і

навчений на кластерах GPU на платформі Google Cloud;

– Запускається на Spark;

– Підтримує GPU від NVIDIA, TPU від Google, GPU з Open-CL, такі як AMD.

Недоліки:

– Може виявитися занадто високорівневим і не завжди легко кастомізується;

– Обмежений бекендами Tensorflow, CNTK та Theano.

Keras не такий функціональний як TensorFlow і дає менше опцій для управління мережевим з'єднанням, що може стати серйозним обмеженням, якщо необхідно створити якусь спеціалізовану модель глибокого навчання.

MXNet

MXNet – це фреймворк для глибокого навчання створений Apache, який підтримує багато мов програмування, наприклад, Python, Julia, C++, R та JavaScript. Він застосовується в Microsoft, Intel і веб-сервісах Amazon.

Фреймворк MXNet відомий своєю високою масштабованістю, тому він використовується великими компаніями в основному для розпізнавання мови і почерку, NLP і прогнозування.

У нього є безліч переваг:

– Він досить швидкий, гнучкий і ефективний в питаннях роботи з алгоритмами глибокого навчання;

– Він забезпечує просунуту підтримку GPU;

– Він може запускатися на будь-якому пристрої;

– У нього є високопродуктивне імперативне API;

– Він забезпечує легку підтримку моделей;

– Він вкрай масштабований;

– Він забезпечує хорошу підтримку безлічі мов програмування, таких як Python, R, Scala, JavaScript і C ++ і багато інших.

Недоліки MXNet:

– Навколо нього не таке велике співтовариство, як навколо TensorFlow;

– Він не особливо популярний у науковому співтоваристві.

Виходячи з вищенаведених аргументів, MXNet – це хороший фреймворк для великих промислових проектів, але він ще досить молодий, тому можна не отримати технічну підтримку у вирішенні свого питання так швидко, як може бути необхідно.

FANN

FANN (Fast Artificial Neural Network) – безкоштовна бібліотека швидкої штучної нейронної мережі з відкритим вихідним кодом, яка реалізує багатошарові штучні нейронні мережі на мові C. Включає в себе структуру для простої обробки наборів даних для навчання.

Переваги бібліотеки для роботи з нейронними мережами FANN:

– Проста у використанні.

– Універсальна.

– Хороша документація з прикладами.

– Швидка.

– Доступні прив'язки до більш ніж 20 мов програмування.

– Для бібліотеки також доступні декілька графічних користувальницьких інтерфейсів.

До недоліків слід віднести невелику кількість різних можливих архітектур нейронних мереж, доступні тільки 1) Одношарові мережі прямого поширення; 2) Багатошарові мережі прямого поширення; 3) Мережі з радіальної базисної функцією. Інші ж архітектури недоступні.

CNTK

CNTK – зараз цей фреймворк називається Microsoft Cognitive Toolkit. Він представляє собою відкритий фреймворк для глибокого навчання, створений для роботи з дійсно великими датасетами з підтримкою Python, C ++, C # і Java.

CNTK забезпечує ефективне навчання моделей для розпізнавання голосу,

рукописного тексту і картинок, також підтримує CNN і RNN. Він використовується в Skype, Xbox і Cortana.

Переваги:

– Показує хорошу продуктивність і масштабованість;

– Надає безліч добре оптимізованих компонентів;

– Пропонує підтримку Apache Spark;

– Ефективний при використанні ресурсів;

– Має просту інтеграцію з Azure Cloud;

І в нього є один єдиний недолік:

– Обмежена підтримка спільнотою.

Chainer

Chainer – фреймворк на Python для глибокого навчання, підтримуваний фірмами Intel, IBM, NVIDIA і AWS. Він може з легкістю задіяти декілька GPU.

Він підтримує різні архітектури мереж, як наприклад CNN, fast-forward, RNN і має наступні важливі переваги:

– Набагато швидше, ніж будь-який інший фреймворк на Python;

– Гнучкий та інтуїтивно-зрозумілий;

– Існуючі мережі можуть бути модифіковані прямо під час виконання.

Недоліки:

– Дуже складний в плані дебага;

– Навколо нього відносно невелика спільнота.

PyTorch

PyTorch є наступником Python для бібліотеки Torch, написаної на Lua, і великим конкурентом TensorFlow. Він був розроблений Facebook і використовувався у Twitter, Salesforce, а також Оксфордським Університетом і багатьма іншими компаніями.

PyTorch використовується в основному, щоб навчати моделі швидко і ефективно, тому це вибір великої кількості розробників.

У нього є важливі переваги:

– Завдяки архітектурі фреймворку, процес створення моделі досить простий і прозорий;

– Режим за замовчуванням "define-by-run" – відсилання до традиційного програмування. Фреймворк підтримує популярні інструменти для дебага, такі як pdb, ipdb або дебагер PyCharm;

– Підтримує декларативний паралелізм даних;

– Має багато попередньо навчених моделей і готових модульних частин, які легко комбінувати;

– Розподілене навчання доступне з версії 0.4.

Але є і не доліки:

Недостатня підтримка моделей;

– Він ще не готовий для повноцінного виходу в продакшн, проте дорожня карта до версії 1.0 виглядає вражаюче;

– Бракує інтерфейсів для моніторингу і візуалізації, як TensorBoard - однак він має зовнішнє підключення до Tensorboard.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки системи розпізнавання образів за допомогою нейронних мереж у даній роботі було використано мову програмування C++.

C++ – об’єктно-орієнтована, структурна, статично типізована мова програмування високого рівня, що базується на мові програмування C. Мова C++ додає до C об'єктно-орієнтовані можливості. Вводить класи, які забезпечують три найважливіші властивості ООП: інкапсуляцію, успадкування і поліморфізм. Мова C++ вперше описана стандартом ISO/IEC 14882:1998, найбільш актуальним же є стандарт ISO/IEC 14882:2020.

Мова C++ є однією з найпопулярніших мов програмування, що широко використовується для розробки найрізноманітнішого програмного забезпечення.

У даній роботі її було обрано для розробки нейронної мережі розпізнавання зображень через наступні її переваги:

Швидкість роботи. Програми на C++ працюють значно швидше, ніж, наприклад, на мові Python, яку на сьогоднішній день досить часто використовують для розробки нейронних мереж через простоту коду на Python.

Масштабованість. Програмне забезпечення, розроблене на мові C++ легко підтримувати та масштабувати через велику кількість документації до цієї мови програмування та її об'єктно-орієнтованої парадигми.

Можливості для роботи на низькому рівні з пам'яттю, адресами та портами є корисними, наприклад, для використання можливостей розпаралелювання роботи нейронної мережі для її прискорення.

Можливості створення патернів програмування значно полегшує процес розробки програмного забезпечення.

Підтримка різних стилів та технологій програмування, включаючи традиційне директивне програмування, об'єктно-орієнтоване програмування, метапрограмування (шаблони, макроси), елементи функціонального програмування тощо – також полегшує процес розробки.

2.3 Розгорнута постановка завдання

Відповідно до технічного завдання у даній роботі, підлягає програмній реалізації система розпізнавання зображень на основі нейронних мереж.

Для вирішення поставленого завдання необхідно здійснити наступні кроки:

1) провести порівняльне дослідження існуючих аналогів для виявлення переваг та недоліків, врахувати результати дослідження при розробці власної системи;

2) обрати мову та середовище програмування, а також алгоритми та методи для вирішення поставленої у роботі задачі, обґрунтувати свій вибір;

3) здійснити проектування системи розпізнавання образів на основі нейронних мереж, розробити функціональну та структурну схеми системи, діаграму процесів та блок-схеми;

4) реалізувати програмне забезпечення системи розпізнавання образів на основі нейронних мереж;

5) створити інструкцію користувача для провадження системи у експлуатацію;

6) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У цій роботі для вирішення задачі розпізнавання зображень було використано багатошарову нейронну мережу (НМ) зворотного поширення похибки. Було вирішено у якості вхідного вектору використовувати не пікселі зображення, а особливі точки (дескриптори) зображення, визначені алгоритмом SURF, що дозволяє зробити систему розпізнавання зображень стійкою до змін яскравості, масштабів та поворотів об’єктів на схожих зображеннях.

Нейронна мережа зворотного поширення похибки (Back Propagation)

В наш час архітектура нейронної мережі Back Propagation найбільш популярна, ефективна та легка у навчанні для складних, багатошарових мереж. Вона використовується для вирішення різних типів задач, але одною з основних задач, що вона може вирішувати є розпізнавання зображень.

Типова НМ зворотного поширення похибки має вхідний шар (прошарок), вихідний шар та принаймні один прихований шар між ними. Теоретично обмежень відносно числа прихованих шарів не існує, їх кількість залежить від складності задачі, що треба вирішити, та кількості вхідних даних.

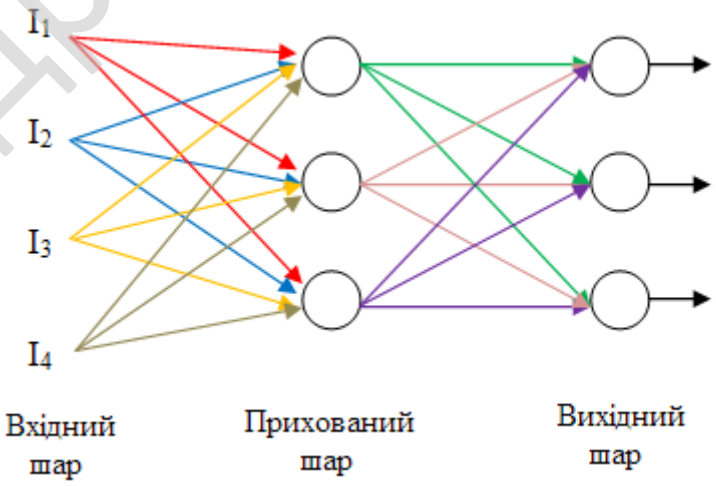

Рисунок 3.1 – Приклад нейронної мережі зворотного поширення похибки

Штучні нейрони об’єднуються у прошарки, між якими відбувається пряма передача сигналів. У кожному нейроні відбувається обчислення зваженої суми усіх сигналів на його входах. Для одержання вихідного значення зважена сума обробляється функцією активації. НМ може моделювати функцію залежності між своїми входами і виходами майже будь якої складності, причому число прошарків і нейронів у них будуть визначати складність цієї функції. Визначення числа прихованих прошарків і нейронів в кожному з них є важливою задачею при моделюванні НМ. Більшість розробників та науковців, розробляючи архітектуру НМ до вирішення конкретних задач, використовують наступні правила визначення кількості прошарків та нейронів:

1. Кількість входів та виходів НМ залежить від кількісті вхідних та вихідних параметрів досліджуваного об’єкту. Число нейронів у прихованому прошарку $n_{прих}$ обирається експериментальним шляхом. Найчастіше достатня кількість нейронів становить $n_{прих} = n_{вх} + n_{вих}$, де $n_{вх}$ – кількість нейронів у вхідному прошарку, $n_{вих}$ – кількість нейронів у вихідному прошарку.

2. Кількість нейронів у прихованому прошарку повинна збільшитись, якщо складність залежності між отриманими та бажаними даними на виході збільшується.

3. Якщо досліджуваний об’єкт має багато ознак на різних рівнях (наприклад, у зображенні можуть бути пікселі, контури, фігури, групи фігур тощо), потрібен додатковий прихований прошарок (прошарки). Якщо об’єкт не можна розділити на ознаки різних рівнів, тоді додаткові прошарки можуть призвести до перенавчання і їх краще не використовувати.

Коли визначено кількість прошарків і нейронів в кожному з них, потрібно знайти значення для ваг і порогів чутливості НМ, які зможуть мінімізувати помилку роботи мережі. Для цього використовуються алгоритми навчання НМ, які здійснюють підбір значень ваг мережі на основі опрацювання навчаючої вибірки. Помилка роботи неронної мережі визначається шляхом порівняння необхідного та одержаного результатів, необхідний результат зафіксовано у

навчаючій вибірці. Усі одержані на навчаючій вибірці помилки підсумовуються для визначення загального значення помилки за допомогою деякої функції похибки. В якості функції похибки НМ найчастіше використовується сума квадратів похибок. Вона була використана і в даній роботі.

Кожному значенню ваг і порогів чутливості НМ (N вільних параметрів мережі) відповідає один вимір у багатовимірному просторі, де вимір N+1 відповідає похибці мережі. Для різноманітних комбінацій ваг відповідну похибку НМ можна зобразити точкою в N+1-вимірному просторі, всі ці точки утворюють деяку поверхню станів мережі. Мета навчання НМ полягає у знаходженні на багатовимірній поверхні найнижчої точки за допомогою градієнтного спуску.

Поверхня станів НМ має складну будову і досить неприємні властивості, наприклад, наявність локальних мінімумів, пласких ділянок, сідлових точок і довгих вузьких ярів. Тому аналітичними засобами неможна визначити розташування глобального мінімуму на поверхні станів і навчання НМ полягає у дослідженні цієї поверхні. Відштовхуючись від початкової конфігурації ваг і порогів, тобто від випадково обраної точки на поверхні станів, алгоритм навчання з учителем поступово шукає глобальний мінімум. Обчислюється вектор градієнту поверхні похибок, який вказує напрямок найкоротшого спуску по поверхні з заданої точки. Якщо дещо просунутись по ньому, помилка зменшиться. Зрештою алгоритм зупиняється в нижній точці, що може виявитись як локальним мінімумом, так і глобальним мінімумом. Складність налаштування процесу навчання полягає у правильному виборі довжини кроків. При великій довжині кроку збіжність буде швидшою, але можна пропустити правильне рішення і піти в невірному напрямку. А при малому кроці, ймовірність виявлення правильного рішення значно зростає, але збільшується й кількість ітерацій. Найчастіше розмір кроку навчання береться пропорційним крутизні схилу з деякою константою – швидкістю навчання. Правильний вибір швидкості навчання залежить від конкретної задачі і здійснюється експериментальним шляхом. Ця константа може також залежати від часу і зменшуючись по мірі

просування по поверхні станів НМ.

Алгоритм навчання НМ діє ітеративно. Його кроки часто називають у науковій літературі епохами. На кожній епосі на вхід НМ по черзі подаються усі навчальні вхідні вектори, вихідні вектори нейронної мережі порівнюються з правильними відповідями й обчислюються похибки. Значення загальної похибки, а також градієнту поверхні станів використовують для корекції вагових коефіцієнтів мережі, після чого всі дії повторюються. Процес навчання припиняється або коли пройдена гранично допустима кількість епох, або коли похибка зменшується до прийнятного значення, або коли похибка перестає зменшуватись. Програміст сам обирає, які умови зупинки або їх комбінації використати.

Алгоритм навчання нейронної мережі злотного поширення похибки:

1. Ініціалізація нейронної мережі: вагові коефіцієнти приймають деякі випадкові значення.

2. Одержання вхідних векторів з навчаючої вибірки: входи мережі (x_1, x_2... x_N) у циклі почергово отримують різні приклади з навчальної вибірки.

3. Визначення зваженої суми входів на кожному штучному нейроні:

$$S_{i_m} = \sum_{i_{m-1}}^{N_{m-1}} w_{i_m j_{m-1}} y_{i_{m-1}} - b_{i_m}, \tag{3.1}$$

$$y_{im} = f(S_{jm}), \tag{3.2}$$

$$i_m = 1, 2, ..., N_m,\ m = 1, 2, ..., L,$$

де S – зважена сума, w – ваговий коефіцієнт, y - вихід нейрону, b – зсув нейрону, i – номер нейрону у прошарку, N – кількість нейронів у прошарку, m – номер прошарку у мережі, L – кількість прошарків, f – функція активації.

4. Налаштування вагових коефіцієнтів:

$$w_{ij}(t+1) = w_{ij}(t) + r g_j x'_i, \tag{3.3}$$

де w_{ij} – ваговий коефіцієнт зв'язку від нейрона i до нейрона j у момент часу t, x_i' - вихід нейрона i, r - швидкість навчання, g_j - значення похибки для нейрона j.

Якщо нейрон з номером *j* належить останньому прошарку, тоді:

$$g_j = y_j(1 - y_j)\cdot(d_j - y_j), \tag{3.4}$$

де d_j - бажаний вихід нейрона *j*, y_j - поточний вихід нейрона *j*.

Якщо нейрон з номером *j* належить одному з прошарків з першого по передостанній, тоді:

$$g_{j=}x'_j\left(1-x'_j\right)\sum_k g_k w_{jk}\,, \tag{3.5}$$

де *k* пробігає всі нейрони прошарку з номером на одиницю більше, ніж у того, якому належить нейрон *j*.

Зовнішні зсуви нейронів *b* налаштовуються аналогічним чином.

В НМ зворотного поширення похибки застосовуються декілька варіантів сигмоїдальних функцій активації:

Сигмоїда:

$$f(S) = \frac{1}{1+e^{-2as}}, \tag{3.6}$$

де *s* – зважена сума входів нейрону, *a* – деякий параметр.

Раціональна сигмоїда:

$$f(S) = \frac{S}{|S|+a}, \tag{3.7}$$

Гіперболічний тангенс:

$$f(S) = th\frac{S}{a} = \frac{e^{-\frac{s}{a}} - e^{-\frac{s}{a}}}{e^{\frac{s}{a}} + e^{-\frac{s}{a}}}, \tag{3.8}$$

Вищенаведені функції активації відносяться до однопараметричних, тобто значення функції залежить від аргументу й одного параметру. Також у НМ використовуються багатопараметричні функції активації, зокрема:

$$f(S) = p_1\frac{S}{|S|+p_2} + p_3\,, \tag{3.9}$$

Сигмоїдальні функції активації є монотонно зростаючими і мають відмінні від нуля похідні по всій області визначення. Ці їх властивості забезпечують можливість навчання мережі за допомогою градієнтного спуску,

оскільки даний алгоритм потребує взяття похідних.

Розглянемо принцип роботи штучного нейрону (рис. 3.2).

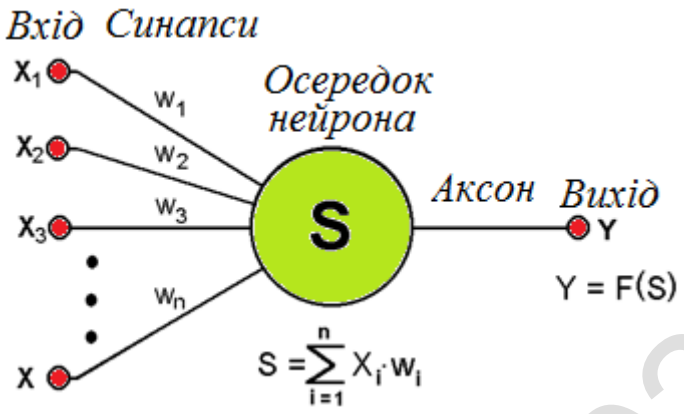

Рисунок 3.2 – Штучний нейрон

Сума вхідних сигналів x_n зважена ваговими коефіцієнтами зв'язків w_n, після проходження через функцію активації дає вихідний сигнал нейрона. У сучасних існуючих програмних реалізаціях штучні нейрони мають набагато більше можливостей, ніж простий штучний нейрон, описаний вище. На рис. 3.3 зображена детальна схема спрощеного штучного нейрону.

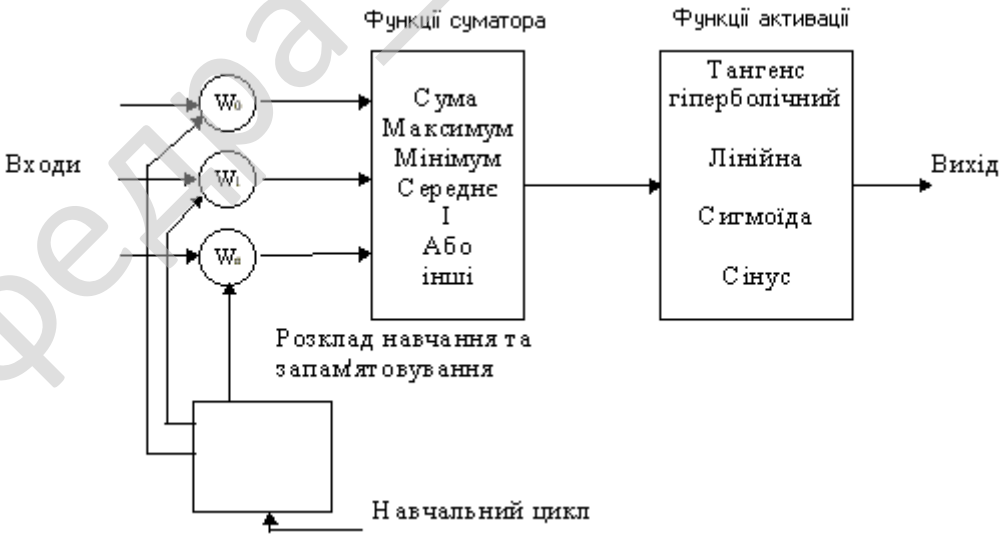

Рисунок 3.3 – Модель функціонування нейрону

Після обробки сигналу, нейрон на виході має результат функції активації, який надходить на входи інших нейронів або на вихід нейронної мережі, в залежності від того, у якому прошарку розташовано нейрон. Всі штучні нейронні мережі створюються з базового блоку – штучного нейрону.

Алгоритм SURF для визначення особливих точок на зображенні

SURF вирішує два завдання – пошук особливих точок зображення й створення їхніх дескрипторів, інваріантних до масштабу й обертання. Це значить, що опис ключової точки буде однаковий, навіть якщо зображення змінить розмір і буде повернене. І сам пошук ключових точок теж повинен мати інваріантність, щоб повернений або масштабований об'єкт зображення мав той же набір ключових точок, що й зразок.

Метод SURF дозволяє знайти особливі точки зображення за допомогою матриці Гессе. Детермінант матриці Гессе (гессіан) досягає екстремуму в точках максимальної зміни градієнта яскравості. Він добре знаходить плями, кути й краї ліній. Гессіан інваріантний щодо обертання. Але не інваріантний щодо масштабу. Тому слід використовувати різномасштабні фільтри для знаходження гессіанів.

Для кожної ключової точки вважається напрямок максимальної зміни яскравості (градієнт) і масштаб, узятий з масштабного коефіцієнта матриці Гессе.

Градієнт у точці зображення обчислюється за допомогою фільтрів Хаара.

Після знаходження ключових точок, метод SURF створює їхні дескриптори. Дескриптор є набором з 64 (або 128) чисел для кожної ключової точки, що відображають флуктуації градієнта навколо ключової точки. Так як ключова точка є максимум гессіана, то це гарантує, що в околиці точки будуть ділянки з різними градієнтами. Таким чином, забезпечується дисперсія (розходження) дескрипторів для різних ключових точок на зображенні.

Флуктуації градієнта околиць ключової точки вважаються щодо напрямку градієнта навколо точки в цілому (по всій околиці ключової точки). Таким чином, досягається інваріантність дескриптора щодо обертання. Розмір же області, на якій рахується дескриптор, визначається масштабом матриці Гессе, що

забезпечує інваріантність щодо масштабу. Флуктуації градієнта також рахують на допомогою фільтра Хаара.

Для ефективного обчислення фільтрів Гессе й Хаара – використовується інтегральне подання зображень.

Якщо коротко, то інтегральне подання є матрицею, у якої розмірність збігається з розмірністю досліджуваного зображення, а елементи розраховуються за формулою:

$$II(x,y) = \sum_{i=0,j=0}^{i\le x,j\le y} I(i,j) \quad (3.10)$$

,

де $I(i,j)$ – яскравість пікселів досліджуваного зображення.

Якщо є інтегральна матриця, можна швидко обчислити суму яскравістей пікселів довільних прямокутних областей зображення наступним чином:

$$\text{SumOfRect(ABCD)} = II(A) + II(3) - II(B) - II(D),$$

де ABCD – прямокутник, що цікавить нас.

Виявлення особливих точок в SURF засноване на обчислені детермінанта матриці Гессе (гессіана).

Матриця Гессе для двовимірної функції і її детермінант визначається в такий спосіб:

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \quad (3.11)$$

.

$$det(H) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 \quad (3.12)$$

Значення гессіана використовується для знаходження локального мінімуму або максимуму яскравості досліджуваного зображення. У цих точках значення гессіана досягає екстремуму.

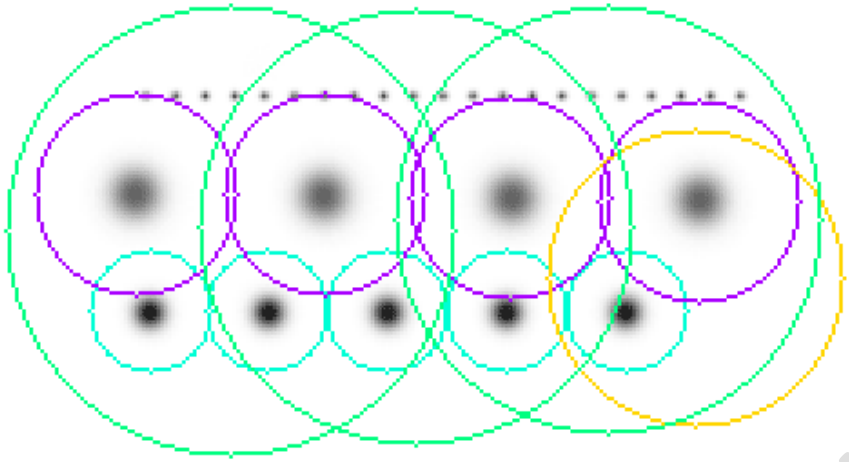

Рисунок 3.4 – Особливі точки (обкреслені різнокольоровaним колами) є локальними екстремумами яскравості досліджуваного зображення

Мілкі точки не розпізнані як особливі, через граничне відсікання по величині гессіана.

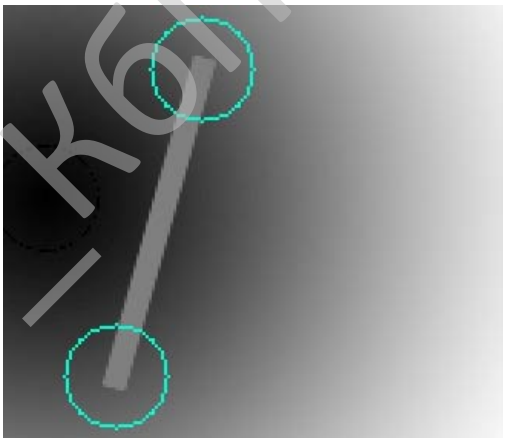

Рисунок 3.5 – Приклад особливих точок на кінці відрізку, що розпізнані як ключові точки, за допомогою матриці Гессе

Обчислення матриці Гессе зводиться до знаходження Лапласіана Гауссіан, по суті, елементи матриці Гессе визначаються як згортка (сума добутків) пікселів досліджуваного зображення на фільтри, зображені на рисунку 3.6.

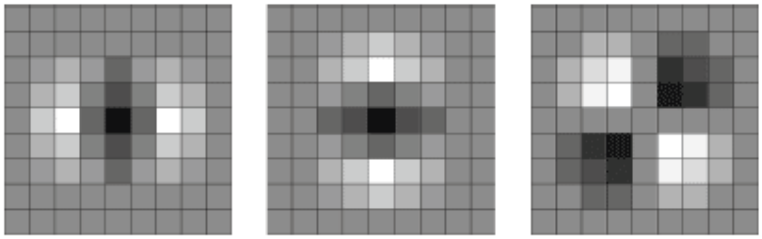

Рисунок 3.6 – Дискретизировані фільтри для знаходження чотирьох елементів матриці Гессе

На рисунку зображені дискретизировані фільтри для знаходження чотирьох елементів матриці Гессе (четвертий – збігається із третім, оскільки матриця Гессе симетрична). Фільтри мають просторовий масштаб 9x9 пікселів. Темні ділянки відповідають негативним значенням фільтра, світлі – позитивним.

Однак, Метод SURF не використовує лапласіан гауссіани в тому виді, що зображений на рисунку. По-перше, за твердженням авторів, дискретизировані лапласіан гауссіани має досить великий розкид значення детермінанта, при обертані зразка (нагадаємо, що в ідеалі гессіан повинен бути інваріантний до обертання). Особливо детермінант «просідає» у районі повороту на 45 градусів. А по-друге, і цей головне, фільтр для лапласіана гауссіани має безперервний характер. Майже всі пікселі фільтра мають різні величини яскравості. А це не дозволяє ефективно використовувати такий потужний механізм розрахунку, як інтегральну матрицю зображення.

Тому метод SURF застосовує бінарізовану апроксимацію лапласіана гауссіан (автори назвали його Fast-Hessian).

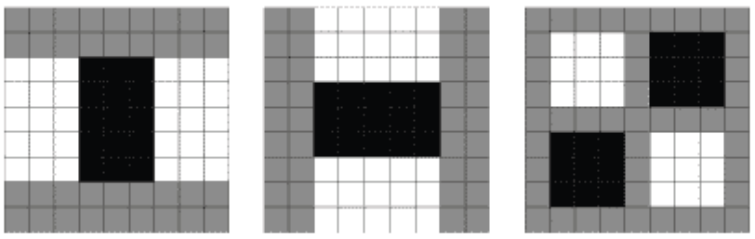

Рисунок 3.7 – Фільтри для знаходження матриці Гессе в методі SURF

Білі області відповідають значенню +1, чорні -2 (на третьому фільтрі -1), сірі – нульові. Просторовий масштаб – 9x9 пікселів.

Цей фільтр більш стійкий до обертання, і його можна ефективно обчислити за допомогою інтегральної матриці.

Таким чином, в методів SURF, гессіан обчислюється так:

$$det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \quad (3.13)$$

,

де D_{xx}, D_{yy}, D_{xy} – згортки по фільтрах, зображеним на рисунку вгорі. Коефіцієнт 0.9 має теоретичне обґрунтування, і коректує наближений характер обчислень.

Тож, для знаходження особливих точок, метод SURF пробігається по пікселях досліджуваного зображення й шукає максимум гессіана. У методі задається граничне значення гессіана. Якщо обчислене значення для пікселя вище порога – піксель розглядається як кандидат на ключову точку.

Отут ще корисно помітити наступне. Оскільки гессіан є похідній, і залежить тільки від перепаду яскравості, але не від абсолютного її рівня, то він інваріантний стосовно зрушення яскравості зображення. Таким чином, зміна рівня висвітлення зразка не впливає на виявлення особливих точок.

Крім того, властивості гессіана такі, що він досягає максимуму, як у точці білої плями на чорному тлі, так і чорної плями на білому тлі. Таким чином, метод виявляє й темні, і світлі особливості зображення.

Гессіани не інваріантні щодо масштабу. Це значить, що для того самого пікселя, гессіан може мінятися при зміні масштабу фільтра. Ця проблема має наступне вирішення – треба перебирати різні масштаби фільтрів та по черзі їх застосовувати до досліджуваного зображення.

З міркувань симетрії й дискретизації, розмір фільтра Fast-Hessian не може приймати довільні значення. Допустимі розміри цього фільтра такі: 9, 15, 21, 27 і так далі, із кроком 6. Але на практиці поступово збільшувати розмір фільтру на 6 – не вигідно, тому що для великих масштабів крок 6 виявляється занадто дрібним, а фільтри – надлишковими. Тому, й з деяких інших причин, метод SURF

розбиває всю множинуч масштабів на так звані октави. Кожна така октава покриває певний інтервал масштабів, і має свій характерний розмір фільтра.

При цьому якби на октаву доводився тільки один фільтр, це було б занадто грубим наближенням. Крім того, не можна було б знайти локальний максимум гессіана, серед різних масштабів, у різних октавах. Адже та сама точка може мати декілька локальних максимумів гессіана, у різних масштабах.

Якщо шукати максимум серед всіх гессіанів, по всіх масштабах, то було б знайдено тільки один з максимумів, у той час як їх може бути декілька. Один – в одному масштабі, іншої – в іншому.

На основі вищенаведеного, октава містить не 1 фільтр, а 4 фільтри, які добре покривають характерний масштаб октави.

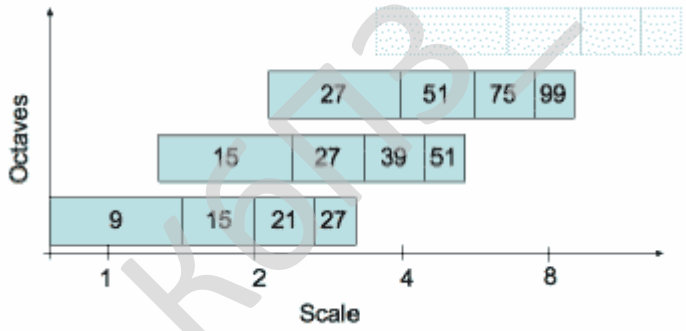

Рисунок 3.8 – Перші три октави методу SURF

Числа в прямокутниках показують розмір фільтра Fast-Hessian. Логарифмічна шкала знизу – показує масштаби, які покриваються октавами.

Крок розміру фільтра в першій октаві – становить 6, у другий – 12, у третьої – 24 і так далі.

Октави значно перекриваються одна одною, як видно з рисунку. Це збільшує надійність знаходження локальних максимумів.

Виникає питання, а скільки власне октав досить для покриття множини особливих точок різних масштабів. Теоретично, масштаби нескінчені, однак у

реальних зображеннях, вони цілком скінченні, а основна маса зосереджена в інтервалі від 1 до 10 (за даними авторів методу). Для покриття цього діапазону досить і 4 октав. Також додається одна або дві октави для покриття великих масштабів. Тобто, використовується 5-6 октав. Теоретично, цього цілком достатньо для покриття всіляких масштабів на зображені 1024x768 пікселів.

Для знаходження локального максимуму гессіана, використовується так званий метод сусідніх точок 3x3x3.

Його зміст зрозумілий з рисунка нижче.

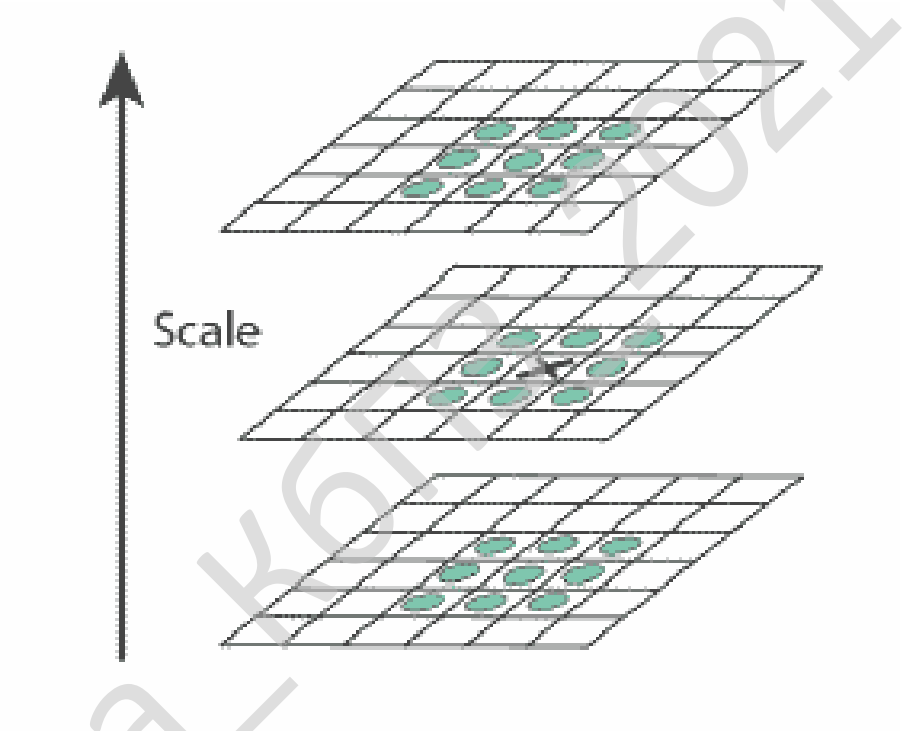

Рисунок 3.9 – Локальний максимуму гессіана

Піксель, позначений хрестиком уважається локальним максимумом, якщо його гессіан більше ніж у будь-якого його сусіда в його масштабі, а також більше кожного із сусідів масштабом менше й масштабом більше (усього 26 сусідів). Виходячи з такого визначення локального максимуму, зрозуміло, що октава повинна містити не менш трьох фільтрів, інакше ми не зможемо визначити факт знаходження локального максимуму гессіана усередині октави. Фільтри октави рахуються не для всіх пікселів підряд. Перша октава рахується для кожного другого пікселя зображення. Друга – для кожного 4-го, третя – для кожного 8-го

й так далі. Тобто, дві точки з відстанню 2 не можуть містити більше одного максимуму масштабу 2, 3 або більше високих масштабів. Тому нема сенсу перебирати всі точки зображення, для знаходження максимуму масштабу 3, наприклад.

Подвоєння кроку пікселів для октав дозволяє заощаджувати при розрахунку фільтрів. Розміри фільтрів в октавах повторюються. Так, наприклад, фільтр розміром 27 є присутнім у трьох октавах. Отож, при обчисленнях, цей фільтр буде вважатися тільки для першої октави. Друг і третя – просто використовують розрахунки першої октави. А подвоєння кроку пікселів гарантує, що точки в які потрібно вважати гессіан, уже були перелічені попередньою октавою.

Тому, незважаючи на те, що октава містить чотири фільтри, насправді кожна октава (крім першої) уважає тільки два характерних для неї розміру, два інших – завжди можна взяти з попередніх октав. Перша ж октава змушена вважати всі чотири своїх фільтри.

Отже, після знаходження максимального гессіана методом сусідніх точок 3x3x3, буде знайдено піксель, у якому цей максимум досягається. Але оскільки октава перебирає не всі точки зображення, то справжній максимум може не збігатися зі знайденим пікселем, а лежати десь поруч, у сусідніх пікселях.

Для знаходження точки справжнього максимуму, використовується інтерполяція знайдених гессіанів куба 3x3x3 квадратичною функцією. Потім обчислюється похідна методом кінцевих різниць сусідніх точок. Якщо вона близька до нуля – знайдено точку глобального максимуму. Якщо похідна велика – зрушуємося убік її зменшення, і повторюємо ітерацію, доти поки похідна не стане менше заданого порога. Якщо в процесі ітерацій відходимо від початкової точки занадто далеко, то це вважається помилковим максимумом.

Для інваріантності обчислення дескрипторів особливої точки потрібно визначити переважну орієнтацію перепадів яскравості в особливій точці. Це

поняття близьке до поняття градієнта, але метод SURF використовує небагато інший алгоритм знаходження вектора орієнтації.

Спочатку, обчислюються точкові градієнти в пікселях, сусідніх з особливою точкою. Для розгляду беруться пікселі в окружності радіуса 6*s* навколо особливої точки. Де *s* – масштаб особливої точки. Для першої октави беруться точки з околиці радіусом 12.

Для обчислення градієнта, використовується фільтр Хаара. Розмір фільтру обирається рівним 4s, де *s* – масштаб ключової точки. Вид фільтрів Хаара показаний на рис. 3.10.

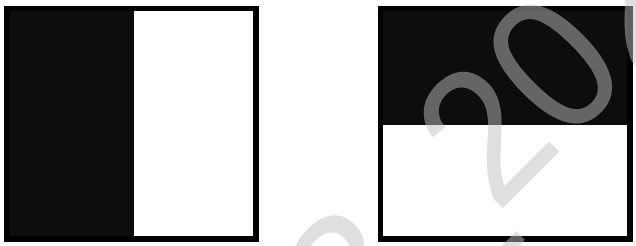

Рисунок 3.10 – Вид фільтрів Хаара

Чорні області мають значення -1, білі +1. Фільтри Хаара дають точкове значення перепаду яскравості по осі *X* і *Y* відповідно. Так як фільтри Хаара мають прямокутну форму, їхні значення легко зважають на допомогу інтегральної матриці. Для розрахунку одного фільтра довільного розміру потрібно всього 6 операцій.

Значення вейвлета Хаара d і d для кожної точки множаться на вагу й запам'ятовуються в масиві. Вага визначається як значення гауссіани із центром в особливій точці й сигмою рівної 2s. Зважування на гауссіане необхідно для відсікання випадкових перешкод на далекі від ключової точки відстанях.

Далі, всі знайдені значення *d* і *d*, умовно наносяться у виг ляді точок на площину, як показано на рисунку 3.11.

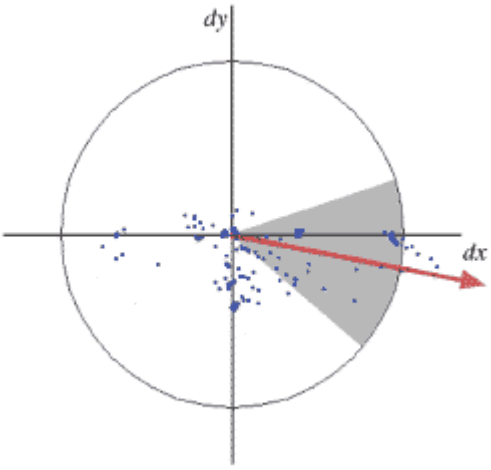

Рисунок 3.11 – Всі знайдені градієнти у вигляді точок у просторі *dXd*

Далі, береться кутове вікно (показане сірим на рисунку) розміром $\pi/3$, і обертається навколо центра координат. Вибирається таке положення вікна, при якому довжина сумарного вектора для точок, що потрапили у вікно, – максимальна. Обчислений у такий спосіб вектор нормується й приймається як пріоритетний напрямок в області особливої точки.

Маніпуляції з вікном потрібні для зменшення впливу шумових точок.

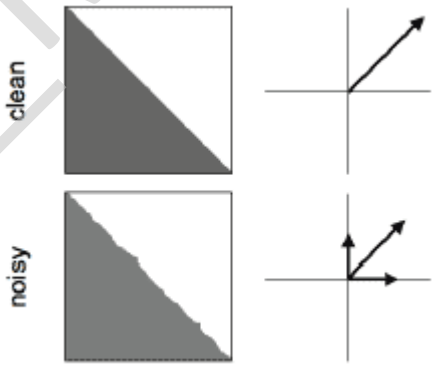

Рисунок 3.12 – Градієнт при ідеальному краї, і при краї із шумом

Як бачимо, шум дає додаткові градієнти в напрямках, що не збігаються з напрямком основного градієнта. Використання вікна дозволяє відітнути такі шумові точки, і більш точно обчислити щирий градієнт.

Не завжди потрібна інваріантність дескрипторів щодо обертання. Метод SURF має модифікацію, у якій орієнтація ключових точок не розраховується. Така модифікація дозволяє надійно ідентифікувати точки, повернені не більше ніж на ±15 градусів.

Дескриптори являють собою масив з 64 або 128 чисел, що дозволяють ідентифікувати ключову точку. Дескриптори однієї й тієї ж ключової точки на зображенні й на шаблоні повинні приблизно збігатися. Метод розрахунку дескриптора такий, що він не залежить від обертання й масштабу.

Для обчислення дескриптора, навколо ключової точки формується прямокутна область, що має розмір 20*s*, де *s* – масштаб, у якому була знайдена ключова точка. Для першої октави, область має розмір 40x40 пікселів. Квадрат орієнтується уздовж пріоритетного напрямку, обчисленого для особливої точки.

Дескриптор визначається як опис градієнта для 16 квадрантів навколо особою точки.

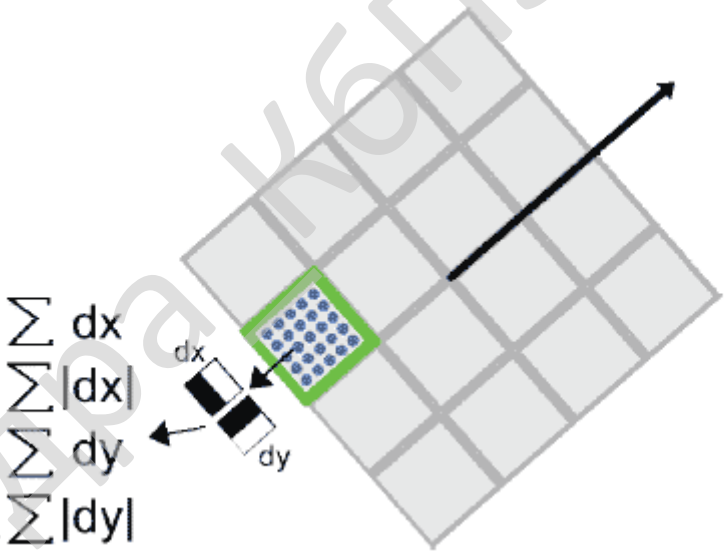

Рисунок 3.13 – Обчислення дескриптора особливої точки

Далі, квадрат розбивається на 16 більше дрібних квадрантів, як показано на рисунку. У кожному квадранті береться регулярна сітка 5x5 і для точки сітки

шукається градієнт, за допомогою фільтра Хаара. Розмір фільтра Хаара береться рівним 2s, і для першої октави становить 4x4.

Слід зазначити, що при розрахунку фільтра Хаара, зображення не повертається, фільтр уважається у звичайних координатах зображення. А от отримані координати градієнта (*d*, *d*) повертаються на кут, що відповідає орієнтації квадрата.

Разом, для обчислення дескриптора особливої точки, потрібно обчислити 25 фільтрів Хаара, у кожному з 16 квадрантів. Разом, 400 фільтрів Хаара. З огляду на, що на фільтр потрібно 6 операцій, виходить, що дескриптор обійдеться мінімум в 2400 операцій.

Після знаходження 25 точок градієнта квадранта, обчислюються 4 величини, які власне і є компонентами дескриптора:

$$\sum dX, \sum |dX|, \sum dY, \sum |dY|.$$

Дві перші з них є сумарним градієнтом по квадранту, а дві інші – це сума модулів точкових градієнтів.

Рисунок 3.14 – Поводження дескриптора для різних ділянок зображення

Рисунок показує поводження дескриптора для різних зображень. Для рівномірних областей – всі значення близькі до нуля. Для повторюваних

вертикальних ліній – всі величини, крім 2-гої близькі до нуля. При збільшені яскравості в напрямку осі *X*, дві перші компоненти мають більші значення.

Чотири компоненти на кожний квадрант, і 16 квадрантів, дають 64 компонента дескриптора для всієї області особливої точки. При занесені в масив, значення дескрипторів зважуються на гауссіану, із центром в особливій точці й із сигмою 3.3*s*. Це потрібно для більшої стійкості дескриптора до шумів у вилучені від особливої точки областях.

Плюс до дескриптора, для опису ключової точки використовується знак сліду матриці Гессе, тобто величина sign(Dxx+Dyy). Для світлих точок на темному тлі, слід негативний, для темних точок на світлому тлі – позитивний. Таким чином, метод SURF розрізняє світлі й темні плями на зображенні.

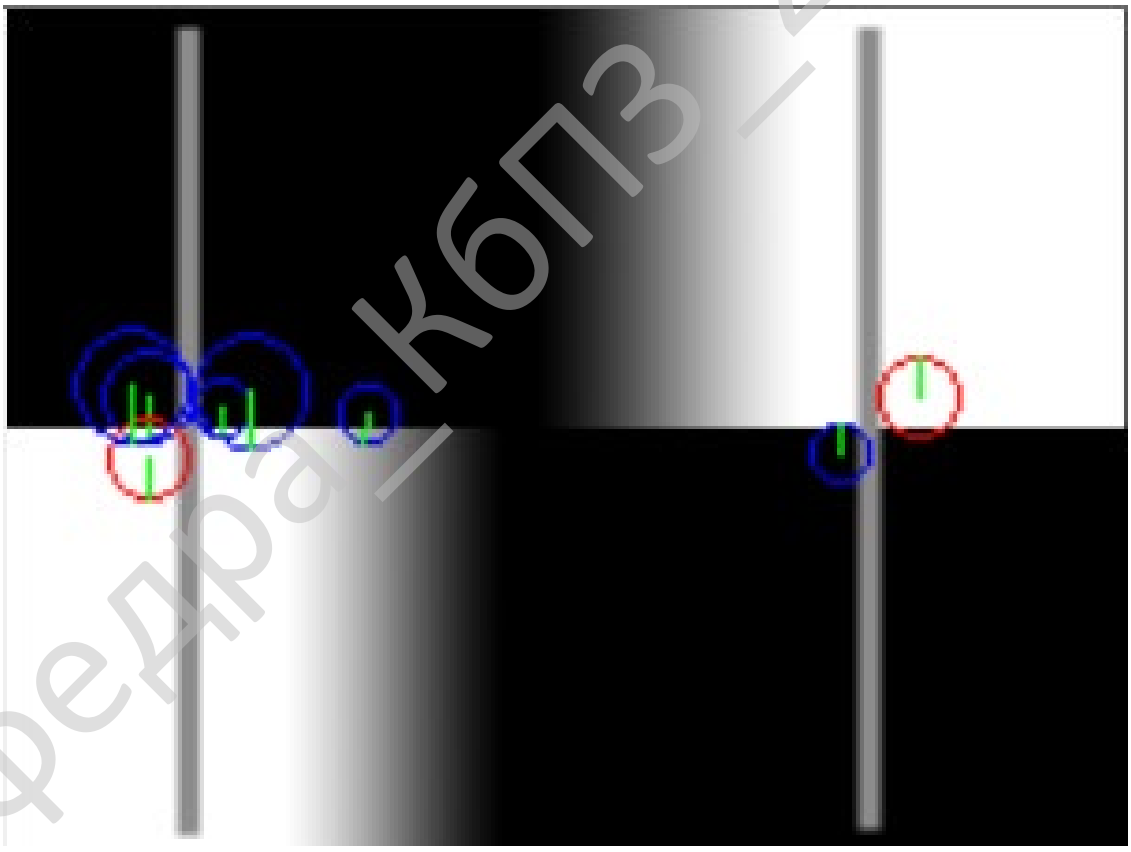

Рисунок 3.15 – Особливі точки зображення

Зелена лінія показує характерний напрямок для особливої точки. Синій колір окружності показує позитивний слід матриці Гессе, червоний – негативний слід.

3.2 Розробка структурної схеми

На рисунку 3.16 зображена структурна схема розроблюваної системи.

Як видно з рисунку, у системі є база даних зображень, на основі яких здійснюється навчання, тестування та використання нейронної мережі. Вхідні вектори для нейронних мереж формуються за допомогою методу SURF, що обчислює дескриптори особливих точок на зображенні. Архітектура нейронної мережі, що розробляється – багатошарова нейронна мережа зворотного поширення.

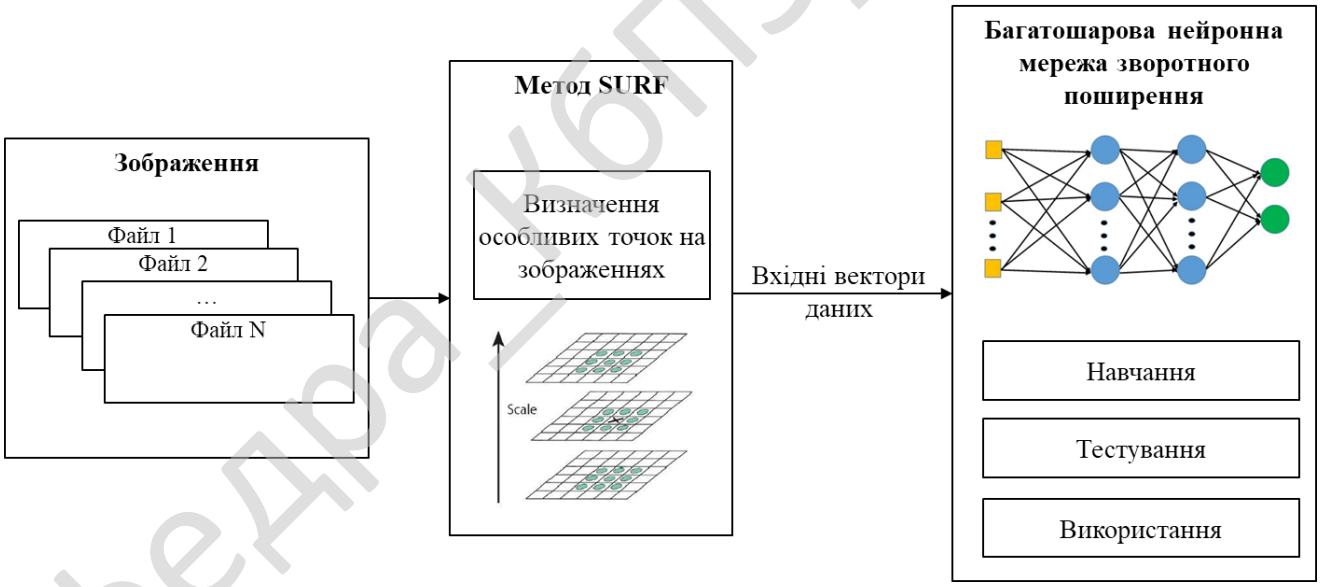

Рисунок 3.16 – Структурна схема системи

Оскільки метод SURF може знайти будь-яку кількість особливих точок, то було вирішено на входи нейронної мережі одержувати максимум N дескрипторів особливих точок (наприклад, $N = 128$, виходячи з розмірів зображень). Таким

чином у вхідному шарі буде N нейронів. Якщо дескрипторів буде менше, то «зайві» нейрони одержить нульові значення на входах.

3.3 Розробка функціональної схеми

На рисунку 3.17 зображена функціональна схема розроблюваної системи. Як видно з рисунку розроблювана програма складається з наступних елементів:

1. Блок навчання нейронної мережі.
2. Блок тестування нейронної мережі.
3. Блок використання нейронної мережі.
4. База даних з наборами зображень.

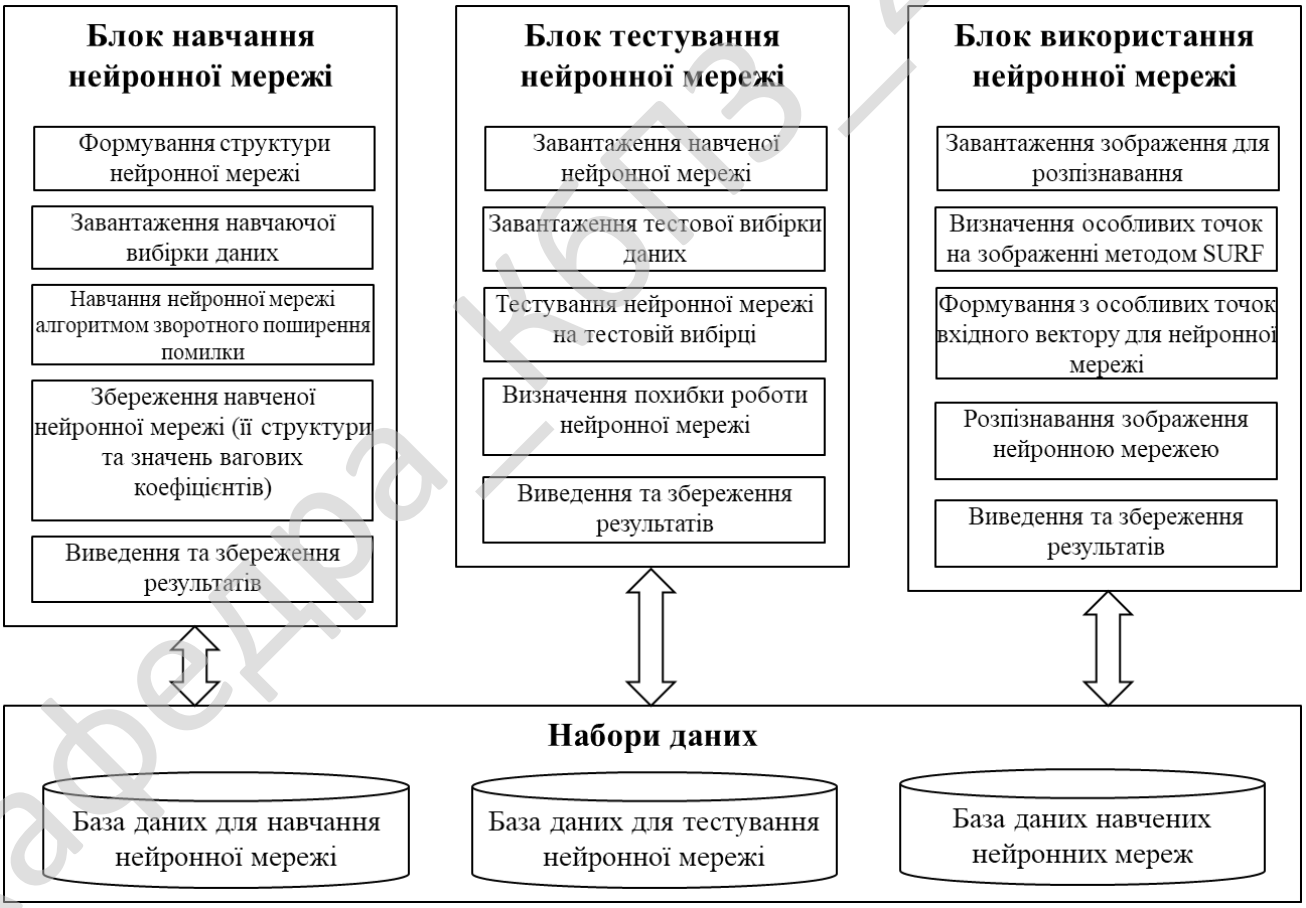

Рисунок 3.17 – Функціональна схема системи

3.4 Розробка діаграми процесів

На рисунку 3.18 зображена діаграма взаємодії процесів.

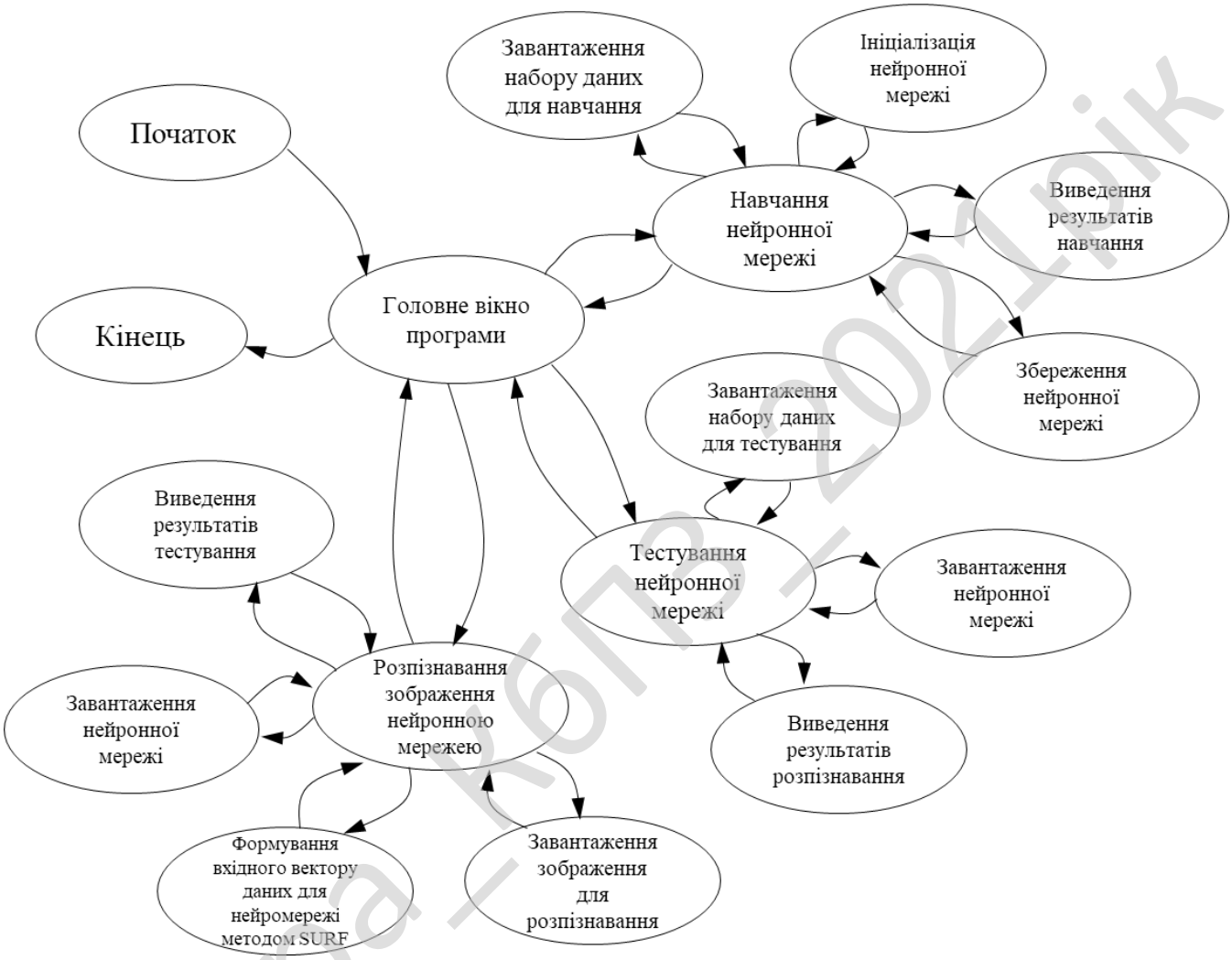

Рисунок 3.18 – Діаграма взаємодії процесів

Як видно з рисунку розроблювана система складається з наступних процесів:

1. Головне вікно програми.
2. Навчання нейронної мережі.
3. Завантаження набору даних для навчання.
4. Ініціалізація нейронної мережі.

5. Виведення результатів навчання.
6. Збереження нейронної мережі.
7. Тестування нейронної мережі.
8. Завантаження набору даних для тестування.
9. Завантаження нейронної мережі.
10. Виведення результатів розпізнавання.
11. Розпізнавання зображення нейронною мережею.
12. Завантаження зображення для розпізнавання.
13. Формування вхідного вектору даних для нейромережі методом SURF.
14. Завантаження нейронної мережі.
15. Виведення результатів тестування.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1.

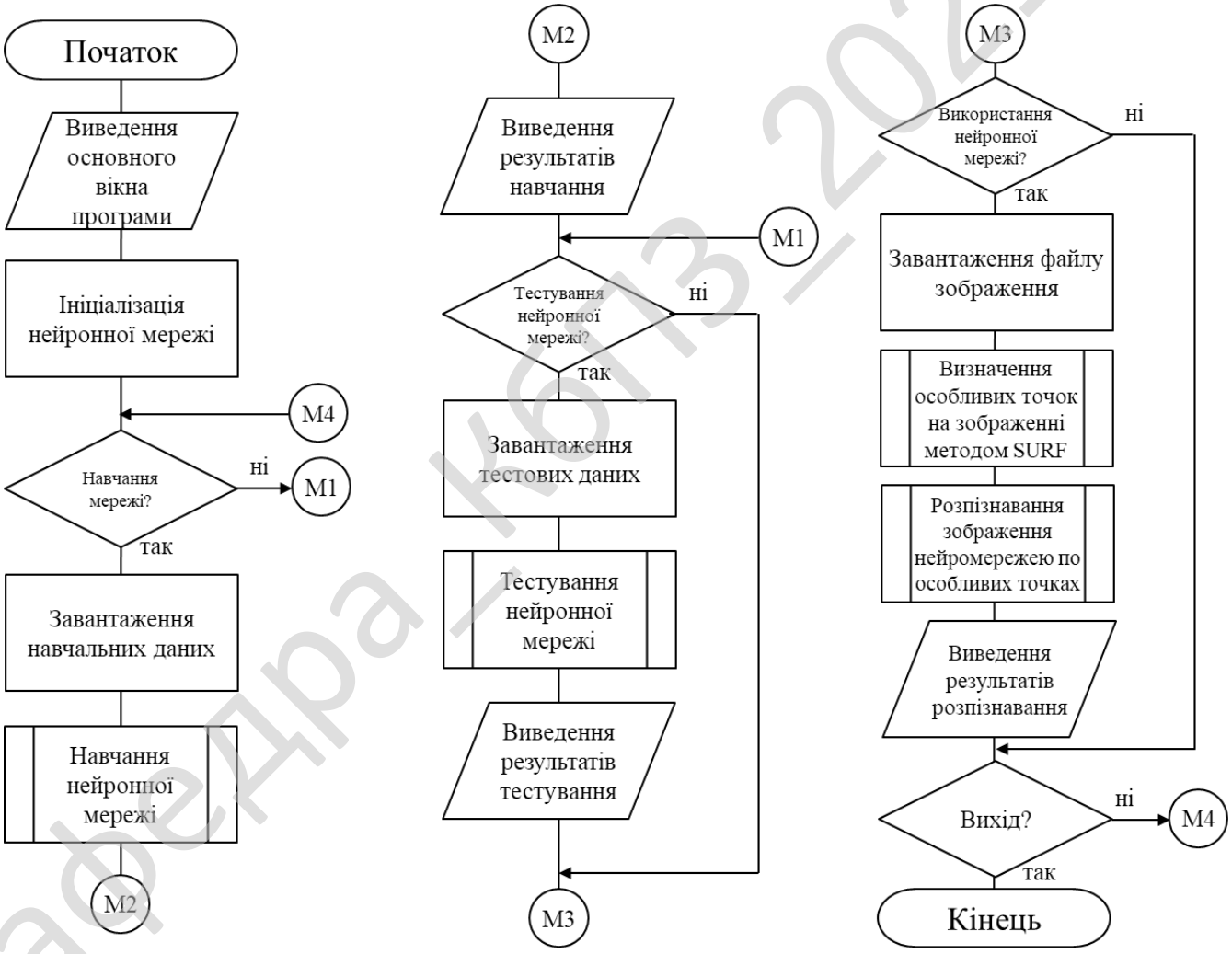

Рисунок 4.1 – Блок-схема основної програми

Як видно з рисунку, користувач програми може обрати наступні дії:

– Начання нейронної мережі.

– Тестування нейронної мережі.

– Використання нейронної мережі.

Основна програма використовує підпрограми навчання нейронної мережі, тестування нейронної мережі, визначення особливих точок на зображенні методом SURF та розпізнавання зображення нейромережею по особливих точках.

На рисунку 4.2 наведена підпрограма навчання нейронної мережі.

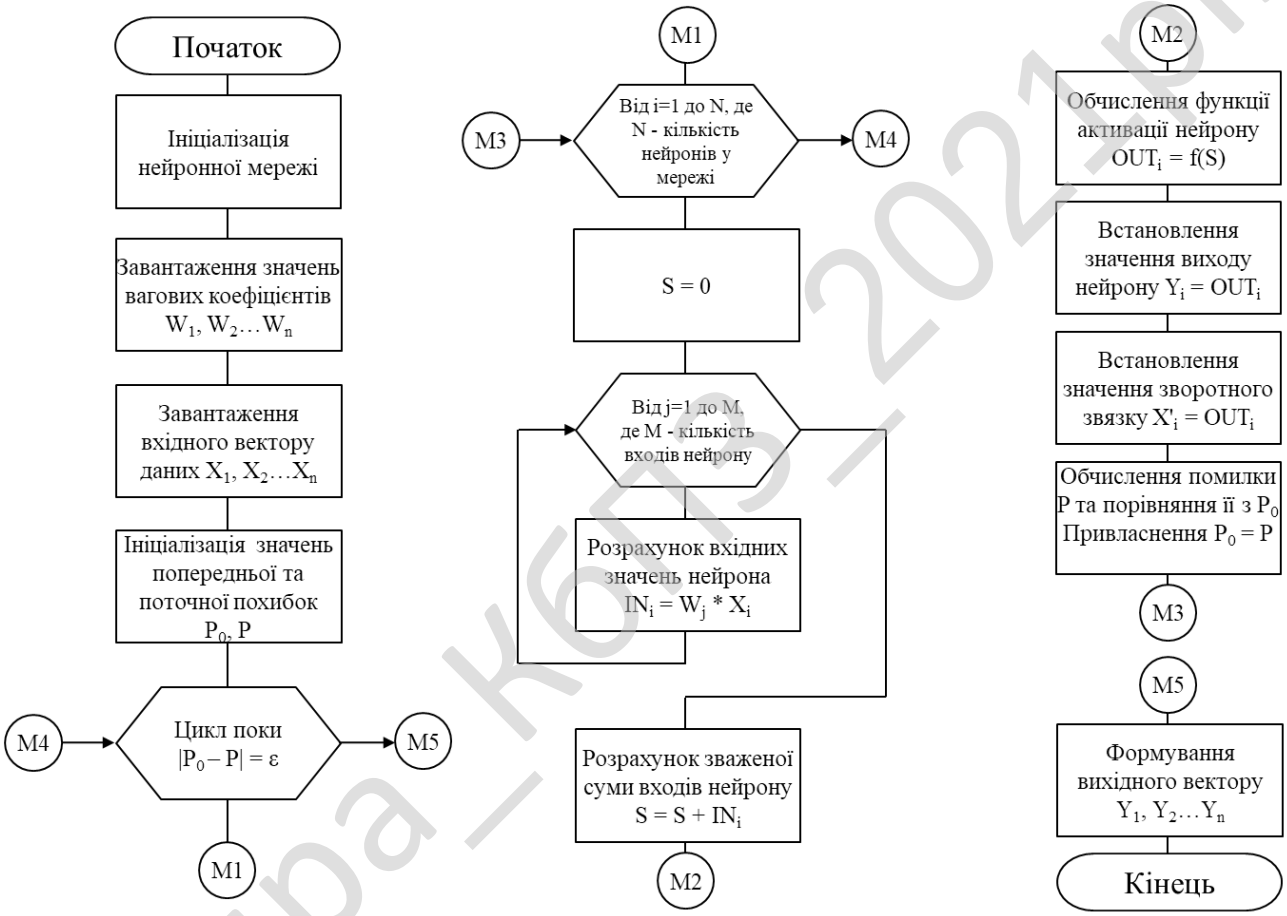

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми навчання нейронної мережі

Навчання нейронної мережі полягає у підлаштуванні вагових коефіцієнтів синапсів таким чином, щоб мінімізувати помилку, що виникає при розпізнаванні зображення. Якщо помилку не вдається знизити до необхідного значення за встановлену максимальну кількість ітерацій, то приймається рішення про зміну

архітектури нейронної мережі. Архітектура нейронної мережі встановлюється вручну, та в даному випадку під нею розуміється кількість прихованих прошарків мережі та кількість нейронів у них.

Розглянемо реалізацію нейронної мережі зворотного поширення похибки у розроблюваному програмному забезпеченні:

```
//ініціалізація
CBackProp::CBackProp(int nl,int *sz,double b,double a):speed(b),alpha(a)
{

   //   множина шарів та їх розміри
   numl_layer=nl;
   layer_size=new int[numl_layer];

   for(int i=0;i<numl_layer;i++){
        layer_size[i]=sz[i];
   }

   //   виділити пам'ять для виведення кожного нейрона
   out = new double*[numl_layer];

   for( i=0;i<numl_layer;i++){
        out[i]=new double[layer_size[i]];
   }

   //   виділити пам'ять для дельти
   dlt = new double*[numl_layer];

   for(i=1;i<numl_layer;i++){
        dlt[i]=new double[layer_size[i]];
   }

   //   виділити пам'ять для ваг
   weight = new double**[numl_layer];

   for(i=1;i<numl_layer;i++){
        weight[i]=new double*[layer_size[i]];
   }
   for(i=1;i<numl_layer;i++){
        for(int j=0;j<layer_size[i];j++){
             weight[i][j]=new double[layer_size[i-1]+1];
        }
   }

   //  allocate memory for previous weights
   prevDwt = new double**[numl_layer];

   for(i=1;i<numl_layer;i++){
        prevDwt[i]=new double*[layer_size[i]];

   }
   for(i=1;i<numl_layer;i++){
        for(int j=0;j<layer_size[i];j++){
             prevDwt[i][j]=new double[layer_size[i-1]+1];
        }
   }

   //   проініціалізувати ваги випадковими значеннями
```

```
   srand((unsigned)(time(NULL)));
   for(i=1;i<numl_layer;i++)
        for(int j=0;j<layer_size[i];j++)
             for(int k=0;k<layer_size[i-1]+1;k++)
                  weight[i][j][k]=(double)(rand())/(RAND_MAX/2) - 1;
   for(i=1;i<numl_layer;i++)
        for(int j=0;j<layer_size[i];j++)
             for(int k=0;k<layer_size[i-1]+1;k++)
                  prevDwt[i][j][k]=(double)0.0;

}

CBackProp::~CBackProp()
{
   //   видалити out
   for(int i=0;i<numl_layer;i++)
        delete[] out[i];
   delete[] out;

   //   видалити dlt
   for(i=1;i<numl_layer;i++)
        delete[] dlt[i];
   delete[] dlt;

   //   видалити weight
   for(i=1;i<numl_layer;i++)
        for(int j=0;j<layer_size[i];j++)
             delete[] weight[i][j];
   for(i=1;i<numl_layer;i++)
        delete[] weight[i];
   delete[] weight;

   //   видалити prevDwt
   for(i=1;i<numl_layer;i++)
        for(int j=0;j<layer_size[i];j++)
             delete[] prevDwt[i][j];
   for(i=1;i<numl_layer;i++)
        delete[] prevDwt[i];
   delete[] prevDwt;

   //   видалити інфорацію про прошарки
   delete[] layer_size;
}

// сигмовидна функція активації
double CBackProp::sigmoid(double in)
{
        return (double)(1/(1+exp(-in)));
}

// середньоквадратична помилка
double CBackProp::mse(double *tgt) const
{
   double mse=0;
   for(int i=0;i<layer_size[numl_layer-1];i++){
        mse+=(tgt[i]-out[numl_layer-1][i])*(tgt[i]-out[numl_layer-1][i]);
   }
   return mse/2;
}


// повертає i-ий результат мережі
double CBackProp::Out(int i) const
```

```
{
   return out[numl_layer-1][i];
}

// подати один набір вхідних даних
void CBackProp::ffwd(double *in)
{
   double sum;

   //   призначити вхідний вектор вхідному шару
   for(int i=0;i<layer_size[0];i++)
        out[0][i]=in[i];  // output_from_neuron(i,j) Jth neuron in Ith Layer

// присвоюємо вихідне (активаційне) значення
// до кожного нейрону застосовуємо функцію активації
   for(i=1;i<numl_layer;i++){                  // Для кожного шару
        for(int j=0;j<layer_size[i];j++){     // Для кожного нейрона в
поточному шарі

             sum=0.0;
             for(int k=0;k<layer_size[i-1];k++){       // Для одержання
даних з кожного нейрона попереднього шару
                  sum+= out[i-1][k]*weight[i][j][k];    // Обчислення
зваженої суми
             }
             sum+=weight[i][j][layer_size[i-1]];
             out[i][j]=sigmoid(sum);                   // Функція активації
        }
   }
}


// повернення значень помилок
void CBackProp::bpgt(double *in,double *tgt)
{
   double sum;

   //   оновити вихідні значення для кожного нейрона
   ffwd(in);

   //   знайти дельту для вихідного шару
   for(int i=0;i<layer_size[numl_layer-1];i++){
        dlt[numl_layer-1][i]=out[numl_layer-1][i]*
        (1-out[numl_layer-1][i])*(tgt[i]-out[numl_layer-1][i]);
   }

   //   знайти дельту для прихованих шарів
   for(i=numl_layer-2;i>0;i--){
        for(int j=0;j<layer_size[i];j++){
             sum=0.0;
             for(int k=0;k<layer_size[i+1];k++){
                  sum+=dlt[i+1][k]*weight[i+1][k][j];
             }
             dlt[i][j]=out[i][j]*(1-out[i][j])*sum;
        }
   }

   //   застосувати імпульс (нічого не робить, якщо alpha=0)
   for(i=1;i<numl_layer;i++){
        for(int j=0;j<layer_size[i];j++){
             for(int k=0;k<layer_size[i-1];k++){
                  weight[i][j][k]+=alpha*prevDwt[i][j][k];
             }
```

```
            weight[i][j][layer_size[i-
1]]+=alpha*prevDwt[i][j][layer_size[i-1]];
        }
   }

   //   відрегулюйте вагу, використовуючи крутий спуск
   for(i=1;i<numl_layer;i++){
        for(int j=0;j<layer_size[i];j++){
             for(int k=0;k<layer_size[i-1];k++){
                  prevDwt[i][j][k]=speed*dlt[i][j]*out[i-1][k];
                  weight[i][j][k]+=prevDwt[i][j][k];
             }
             prevDwt[i][j][layer_size[i-1]]=speed*dlt[i][j];
             weight[i][j][layer_size[i-1]]+=prevDwt[i][j][layer_size[i-1]];
        }
   }
}
```

Навчання мережі реалізовано наступним чином:

```
// ---навачальні дані---

//Створюємо файловий потік і пов'язуємо його з файлом
   ifstream in("train.txt");

   //Відкриття файлу з навчальними даними

   if (in.is_open())
   {

        int count = 0;// кількість чисел у файлі
        int temp;

        //підрахунок кількості чисел у файлі
        while (!in.eof())
        {
             in >> temp;
             count++;
        }

        //Підрахунок кількості чисел у рядку

        //переведемо каретку в потоці в початок файлу
        in.seekg(0, ios::beg);
        in.clear();

        //Число пробілів в першому рядку спочатку дорівнює 0
        int count_space = 0;
        char symbol;
        while (!in.eof())
        {
             //посимвольно читаємо дані
             in.get(symbol);
             if (symbol == ' ') count_space++;//Якщо це пробіл, то число
пробілів збільшуємо
             if (symbol == '\n') break;//Якщо кінець рядка - виходимо з
циклу
        }
        //cout << count_space << endl;

        //Переходимо в потоці в початок файлу
```

```
        in.seekg(0, ios::beg);
        in.clear();

        //Читаємо з файлу матрицю

        int n = count / (count_space + 1);//число рядків
        int m = count_space + 1;//число стовпців на одиницю більше числа
пробілів
        double **data;
        data = new double*[n];
        for (int i = 0; i<n; i++) data[i] = new double[m];

        //Вважаємо матрицю з файлу
        for (int i = 0; i < n; i++)
             for (int j = 0; j < m; j++)
                  in >> data[i][j];

        in.close();//закриваємо файл
   }
   else
   {
        //Якщо відкриття файлу пройшло не успішно
        cout << "Файл не знайдено";
   }

   system("pause");
   return 0;
}
// визначення мережі з 5 шарами, що мають 128,200,200,50 та 10 нейронів
// відповідно, перший шар вхідний, він просто одержувач вхідних параметрів
// і повинен мати такий самий розмір, як кількість вхідних параметрів

   int numl_layerayers = 5, lSz[5] = {128,200,200,50,10};


   // speed - швидкість навчання
   // alpha - імпульс
   // thresh – поріг чутливості нейрону (цільове значення помилки mse,
навчання нейромережі припиняється після його досягнення)

   double speed = 0.3, alpha = 0.1, Thresh =  0.00001;

   // максимум ітерацій під час тренування
   long num_iter = 2000000;


   // Створення мережі
   CBackProp *bp = new CBackProp(numl_layerayers, lSz, speed, alpha);

   cout<< endl <<  " Зараз нейронна мережа навчається...." << endl;
   for (long i=0; i<num_iter ; i++)
   {

        bp->bpgt(data[i%8], &data[i%8][m]);

        if( bp->mse(&data[i%8][m]) < Thresh) {
             cout << endl << " Мережа навчена. Порогове значення, досягнуте
за " << i << " ітерацій." << endl;
             cout << "MSE:  " << bp->mse(&data[i%8][m])
                  <<  endl <<  endl;
             break;
        }
        if ( i%(num_iter/10) == 0 )
```

```
            cout<<  endl <<  "MSE:  " << bp->mse(&data[i%8][m])
                 << "... Training..." << endl;
   }

   if ( i == num_iter )
        cout << endl << i << ітерацій завершено..."
        << "MSE: " << bp->mse(&data[(i-1)%8][m]) << endl;

   cout<< "Навчання завершилося. Тепер нейронну мережу можна використовувати
....." << endl << endl;
   for ( i = 0 ; i < 8 ; i++ )
   {
        bp->ffwd(testData[m2]);
        cout << testData[i][0]<< "  " << testData[i][1]<< "  "  <<
testData[i][2]<< "  " << bp->Out(0) << endl;
   }
```

Реалізація алгоритму SURF для пошуку дескрипторів особливих точок зображення з метою формування вхідного вектору нейронної мережі:

```
// SURF константи
const float pi = 3.14159f;

const double gauss25 [7][7] = {

0.02350693969273,0.01849121369071,0.01239503121241,0.00708015417522,0.00344
628101733,0.00142945847484,0.00050524879060,

...

0.00131955648461,0.00103799989504,0.00069579213743,0.00039744277546,0.00019
345616757,0.00008024231247,0.00002836202103
};

const double gauss33 [11][11] = {

0.014614763,0.013958917,0.012162744,0.00966788,0.00701053,0.004637568,0.002
798657,0.001540738,0.000773799,0.000354525,0.000148179,

...

0.000148179,0.000141529,0.000123318,9.80224E-05,7.10796E-05,4.70202E-
05,2.83755E-05,1.56215E-05,7.84553E-06,3.59452E-06,1.50238E-06
};


// Конструктор
Surf::Surf(IplImage *img, IpVec &ipts)
: ipts(ipts)
{
  this->img = img;
}

// Опишемо усі особливості у векторі, що поставляється
void Surf::getDescriptors(bool upright)
{

// Перевіряємо Ipoints на опис
```

```
  if (!ipts.size()) return;

  // Беремо розмір вектору для фіксованих меж циклу
  int ipts_size = (int)ipts.size();

  if (upright)
  {
    // U-SURF цикл тільки отримує дескриптори
    for (int i = 0; i < ipts_size; ++i)
    {
      // Встановлюємо Ipoint на опис
      index = i;

      // Витягуємо  вертикальні (тобто інваріант не обертання) дескриптори
      getDescriptor(true);
    }
  }
  else
  {
    // Головний SURF-64 цикл визначення орієнтації та отримання
дескрипторів
    for (int i = 0; i < ipts_size; ++i)
    {
      // Встановлюємо Ipoint на опис
      index = i;

      // Призначаємо орієнтацію і витягуємо дескриптори інваріанту
обертання
      getOrientation();
      getDescriptor(false);
    }
  }
}

// Призначаємо поставляння Ipoint на орієнтацію
void Surf::getOrientation()
{
  Ipoint *ipt = &ipts[index];
  float gauss = 0.f, scale = ipt->scale;
  const int s = fRound(scale), r = fRound(ipt->y), c = fRound(ipt->x);
  std::vector<float> resX(109), resY(109), Ang(109);
  const int id[] = {6,5,4,3,2,1,0,1,2,3,4,5,6};

  int idx = 0;
  // розраховуємо відповідні для точок Хаара в межах радіусу 6*масштаб
  for(int i = -6; i <= 6; ++i)
  {
    for(int j = -6; j <= 6; ++j)
    {
      if(i*i + j*j < 36)
      {
        gauss = static_cast<float>(gauss25[id[i+6]][id[j+6]]);
        resX[idx] = gauss * haarX(r+j*s, c+i*s, 4*s);
        resY[idx] = gauss * haarY(r+j*s, c+i*s, 4*s);
        Ang[idx] = getAngle(resX[idx], resY[idx]);
        ++idx;
      }
    }
  }

  // розраховуємо основний напрямок
  float sumX=0.f, sumY=0.f;
  float max=0.f, orientation = 0.f;
```

```
  float ang1=0.f, ang2=0.f;

  // цикл слайдів pi/3 вікно біля точки, яка може бути
  for(ang1 = 0; ang1 < 2*pi;  ang1+=0.15f) {
    ang2 = ( ang1+pi/3.0f > 2*pi ? ang1-5.0f*pi/3.0f : ang1+pi/3.0f);
    sumX = sumY = 0.f;
    for(unsigned int k = 0; k < Ang.size(); ++k)
    {
      // беремо angle з x-axis для точки прикладу
      const float & ang = Ang[k];

      // визначаємо чи є точка в межах вікна
      if (ang1 < ang2 && ang1 < ang && ang < ang2)
      {
        sumX+=resX[k];
        sumY+=resY[k];
      }
      else if (ang2 < ang1 &&
        ((ang > 0 && ang < ang2) || (ang > ang1 && ang < 2*pi) ))
      {
        sumX+=resX[k];
        sumY+=resY[k];
      }
    }

    // якщо вектор робив від цього вікна довше, ніж усі попередні вектори
потім це формує новий домінуючий напрям
    if (sumX*sumX + sumY*sumY > max)
    {
      // запам'ятовуємо найбільшу орієнтацію
      max = sumX*sumX + sumY*sumY;
      orientation = getAngle(sumX, sumY);
    }
  }

  // призначаємо орієнтацію домінуючого відповідному вектору
  ipt->orientation = orientation;
}

// Беремо модифікований дескріптор.

void Surf::getDescriptor(bool bUpright)
{
  int y, x, sample_x, sample_y, count=0;
  int i = 0, ix = 0, j = 0, jx = 0, xs = 0, ys = 0;
  float scale, *desc, dx, dy, mdx, mdy, co, si;
  float gauss_s1 = 0.f, gauss_s2 = 0.f;
  float rx = 0.f, ry = 0.f, rrx = 0.f, rry = 0.f, len = 0.f;
  float cx = -0.5f, cy = 0.f; //Підобласть зосереджується для 4x4 блока
гауса

  Ipoint *ipt = &ipts[index];
  scale = ipt->scale;
  x = fRound(ipt->x);
  y = fRound(ipt->y);
  desc = ipt->descriptor;

  if (bUpright)
  {
    co = 1;
    si = 0;
  }
  else
```

```
{
  co = cos(ipt->orientation);
  si = sin(ipt->orientation);
}

i = -8;

//Розраховуємо дескриптор для цієї особливої точки
while(i < 12)
{
  j = -8;
  i = i-4;

  cx += 1.f;
  cy = -0.5f;

  while(j < 12)
  {
    dx=dy=mdx=mdy=0.f;
    cy += 1.f;

    j = j - 4;

    ix = i + 5;
    jx = j + 5;

    xs = fRound(x + ( -jx*scale*si + ix*scale*co));
    ys = fRound(y + ( jx*scale*co + ix*scale*si));
    for (int k = i; k < i + 9; ++k)
    {
      for (int l = j; l < j + 9; ++l)
      {
        //Беремо координати визначеної точки та повертаємо їх
        sample_x = fRound(x + (-l*scale*si + k*scale*co));
        sample_y = fRound(y + ( l*scale*co + k*scale*si));

        //Беремо the gaussian weighted x and y responses
        gauss_s1 = gaussian(xs-sample_x,ys-sample_y,2.5f*scale);
        rx = haarX(sample_y, sample_x, 2*fRound(scale));
        ry = haarY(sample_y, sample_x, 2*fRound(scale));

        //Беремо блок Гусса x та y відповідно на вісь обертання
        rrx = gauss_s1*(-rx*si + ry*co);
        rry = gauss_s1*(rx*co + ry*si);
        dx += rrx;
        dy += rry;
        mdx += fabs(rrx);
        mdy += fabs(rry);
      }
    }

    //Додаємо значення до дескриптора вектора
    gauss_s2 = gaussian(cx-2.0f,cy-2.0f,1.5f);
    desc[count++] = dx*gauss_s2;
    desc[count++] = dy*gauss_s2;
    desc[count++] = mdx*gauss_s2;
    desc[count++] = mdy*gauss_s2;
    len += (dx*dx + dy*dy + mdx*mdx + mdy*mdy) * gauss_s2*gauss_s2;
    j += 9;
  }
  i += 9;
}
```

```
  //конвертуємо до Unit Vector
  len = sqrt(len);
  for(int i = 0; i < 64; ++i)
    desc[i] /= len;
}

//-------------------------------------------------------

// Розраховуємо значення в  2d гаусціані в x,y
inline float Surf::gaussian(int x, int y, float sig)
{
  return (1.0f/(2.0f*pi*sig*sig)) * exp( -(x*x+y*y)/(2.0f*sig*sig));
}

//-------------------------------------------------------

// Розраховуємо значення в  2d гаусціані в x,y
inline float Surf::gaussian(float x, float y, float sig)
{
  return 1.0f/(2.0f*pi*sig*sig) * exp( -(x*x+y*y)/(2.0f*sig*sig));
}

//-------------------------------------------------------

// Розраховуємо вейвлет Хаара в  x напрямку
inline float Surf::haarX(int row, int column, int s)
{
  return BoxIntegral(img, row-s/2, column, s, s/2)
    -1 * BoxIntegral(img, row-s/2, column-s/2, s, s/2);
}

//-------------------------------------------------------

// Розраховуємо вейвлет Хаара в  y напрямку
inline float Surf::haarY(int row, int column, int s)
{
  return BoxIntegral(img, row, column-s/2, s/2, s)
    -1 * BoxIntegral(img, row-s/2, column-s/2, s/2, s);
}

//-------------------------------------------------------

// Беремо вугол з +ve x-axis для вектора  (X Y)
float Surf::getAngle(float X, float Y)
{
  if(X > 0 && Y >= 0)
    return atan(Y/X);

  if(X < 0 && Y >= 0)
    return pi - atan(-Y/X);

  if(X < 0 && Y < 0)
    return pi + atan(Y/X);

  if(X > 0 && Y < 0)
    return 2*pi - atan(-Y/X);

  return 0;
}
```

4.2 Захист розробленого програмного забезпечення

Захист програмного забезпечення – це комплекс заходів, спрямованих на убезпечення його від несанкціонованого копіювання, використання, поширення, модифікації, вивчення і відтворення аналогів.

Реалізується захист програмного забезпечення як система заходів, спрямованих на протидію його нелегальному використанню, що можуть застосовувати організаційні, юридичні, програмні та програмно-апаратні засоби.

У даній роботі пропонується захищати розроблене програмне забезпечення від несанкціонованого використання шляхом *обфускації коду.*

Обфускація коду – заплутування коду для унеможливлення його несанкціонованого аналізу, модифікації, відтворення та подальшого використання.

Бувають різні технології обфускації програмного забезпечення:

– *На рівні машинного коду*. Обфускація застосовується в таких частинах програми, як перевірка реєстраційного коду, тобто некритичних до швидкості, але критичних до безпеки. Найпростіший метод заплутати машинний код – вставити в нього недіючі конструкції.

– *На рівні вихідних текстів*. Оригінальний текст програми на скриптових мовах. Менш читаним його можна зробити шляхом форматування і заміни імен.

– *На рівні проміжного коду*. Мови програмування для платформи .NET, а також NetP і Java, компілюють вихідний код в проміжний (байт-код), що містить достатньо інформації для відновлення вихідного коду. Тому для перерахованих мов використовується обфускація проміжного коду.

Для захисту розробленого програмного коду пропонується використовувати обфускацію на рівні вихідних текстів.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Перш ніж почати використовувати розроблене програмне забезпечення, потрібно створити навчаючий та тестовий набори даних зображень.

На рис. 5.1 зображене головне вікно програми.

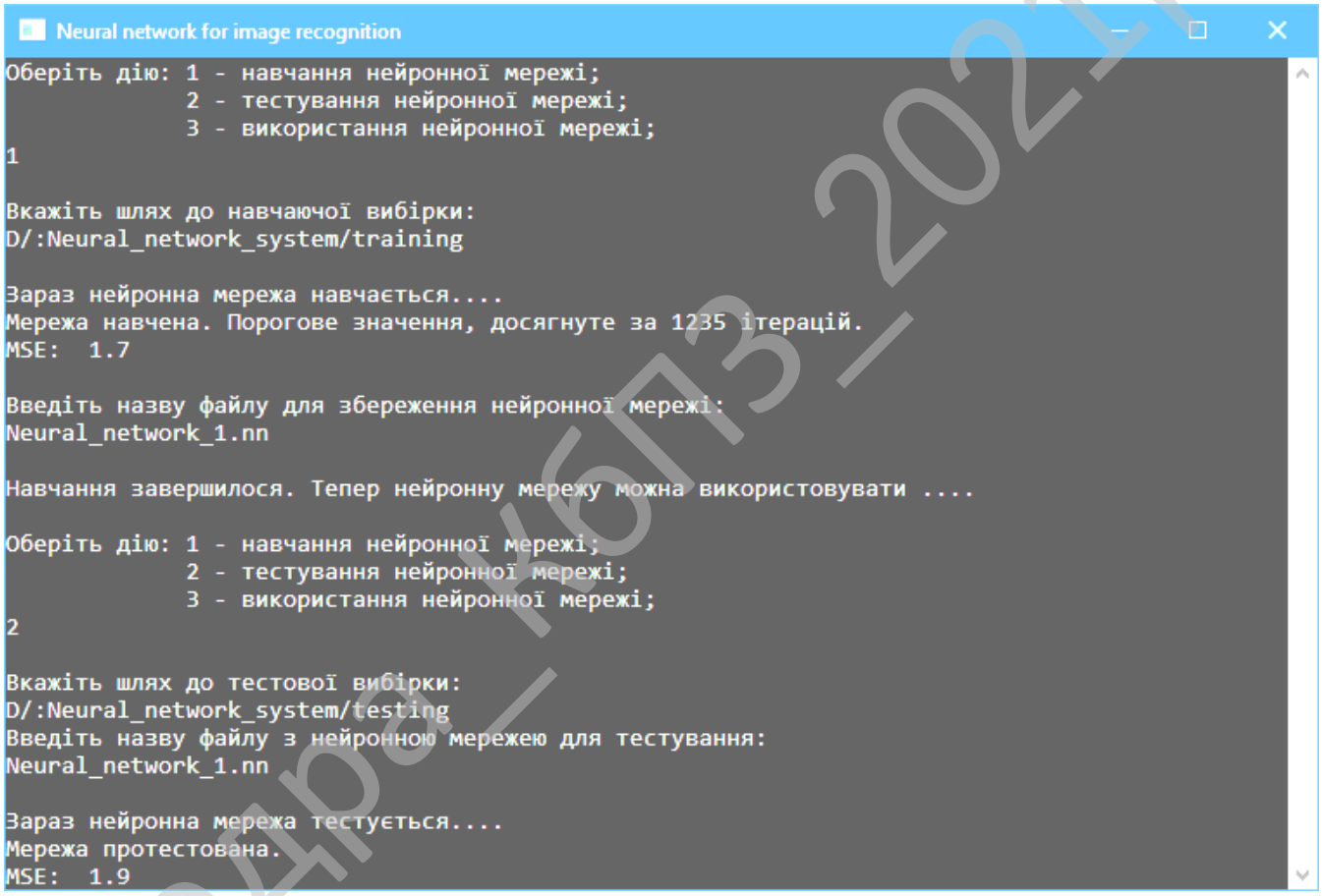

Рисунок 5.1 – Головне вікно програми

Робота з програмою складається з наступних кроків.

Крок 1. Встановити розроблене програмне забезпечення.

Крок 2. Додати набір зображень для навчання та тестування у базу даних.

Крок 3. Обрати архітектуру нейронної мережі.

Крок 4. Запустити навчання нейронної мережі на навчаючій вибірці та переглянути результати навчання. Якщо розмір похибки влаштовує можна продовжувати робити наступні кроки. Інакше треба змінити архітектуру нейронної мережі (крок 3) та повторити даний крок.

Крок 5. Запустити тестування нейронної мережі на тестовій вибірці та переглянути результати навчання. Якщо розмір похибки влаштовує можна продовжувати робити наступні кроки. Інакше треба змінити архітектуру нейронної мережі (крок 3) та повторити кроки 4 та 5.

Крок 6. Зберегти архітектуру нейронної мережі та значення її ваг для можливості подальшого повторного використання.

Крок 7. Завантажити зображення для розпізнавання, запустити процес класифікації зображення навченою нейронною мережею та переглянути результат.

6 ОСНОВНІ ВИСНОВКИ

Програмний застосунок, створений в ході виконання цієї кваліфікаційної бакалаврської роботи, призначений для реалізації системи розпізнавання зображень на основі нейронних мереж.

Для реалізації мети цієї роботи було вирішено наступні задачі:

– Проведено дослідження існуючих систем та методів розпізнавання зображень на основі нейронних мереж.

– Проведено проектування системи розпізнавання зображень на основі нейронних мереж.

– На основі отриманих результатів досліджень створена програмна реалізація системи розпізнавання зображень на основі нейронних мереж.

Розроблені під час виконання цієї роботи алгоритми дозволили реалізувати систему розпізнавання зображень на основі нейронних мереж. Було використано багатошарову нейронну мережу зворотного поширення помилки та метод SURF для визначення дескрипторів особливих точок на зображенні, що дало змогу сформувати вхідний вектор для нейронної мережі, який дозволяє мережі бути стійкою до модифікацій зображення, що розпізнається.

Програмне забезпечення системи розпізнавання зображень було реалізовано на мові програмування C++. Основною причиною вибору цієї мови програмування є її швидкодія та можливість низькорівневого програмування, а також об’єктно-орієнтована парадигма. Її використання дозволило зробити нейронну мережу швидкою у роботі.

Розроблене програмне забезпечення призначене для виконання під управлінням ОС Windows 10.

У пояснювальній записці даються необхідні рекомендації з установки та використання розробленої системи.

У даній роботі пропонується захищати розроблене програмне забезпечення від несанкціонованого використання програмно шляхом обфускації коду.

У роботі було вирішено поставлену мету – реалізовано програмне забезпечення системи розпізнавання зображень на основі нейронних мереж. Розроблений програмний застосунок може використовуватися для пошуку та класифікації зображень у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Барский А.Б. Логические нейронные сети: Учебное пособие / А.Б. Барский. - М.: Бином, 2013. - 352 с.

2. Вороновский Г. К., Махотило К. В., Петрашев С. Н., Сергеев С. А. Генетические алгоритмы, искусствение нейронные сети и проблемы виртуальной реальности. – заказное. – Х.: ОСНОВА, 1997. – С. 112.

3. Галушкин А.И. Нейронные сети: история развития теории: Учебное пособие для вузов. / А.И. Галушкин, Я.З. Цыпкин. - М.: Альянс, 2015. - 840 с.

4. Жданов А.А. Формальная модель нейронна и нейросети в методологии автономного адаптивного управления// Сб. Вопросы кибернетики. Вып. 3. М.: 1997. С. 258-274.

5. Каллан Р. Основные концепции нейроних сетей = The Essence of Neural Networks First Edition. – 1-е. – «Вильямс», 2001. – С. 288. – ISBN 5-8459-0210-X.

6. Миркес Е.М. Нейрокомпьютер. Проект стандарта. – Новосибирск: Наука, 1999. – 337 с. ISBN 5-02-031409-9.

7. Редько, В.Г. Эволюция, нейронные сети, интеллект: Модели и концепции эволюционной кибернетики / В.Г. Редько. - М.: Ленанд, 2015. - 224 с.

8. Розенблат Ф. Принципы нейродинамики. – М.: Мир, 1965.

9. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. - М.: РиС, 2013. - 384 с.

10. Уоссермен Ф. Нейрокомпьютерная техника. – М.: Мир, 1992.

11. Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика = Neural Computing. Theory and Practice. – М.: Мир, 1992. – 240 с. – ISBN 5-03-002115-9.

12. Хайкин С. Нейронные сети: полный курс = Neural Networks: A Comprehensive Foundation. – 2-е. – М.: «Вильямс», 2006. – С. 1104. – ISBN 0-13-273350-1.

13. Чечкин А.В. Математическая информатика. – М.: Наука. 1991.

14. Ширяев, В.И. Финансовые рынки: Нейронные сети, хаос и нелинейная динамика / В.И. Ширяев. - М.: КД Либроком, 2016. - 232 с.

15. Ясницкий Л.Н. Введение в искусствений интеллект. – 1-е. – Издательский центр "Академия", 2005. – С. 176. – ISBN 5-7695-1958-4.

16. Тархов Д.А. Нейросетевые модели и алгоритмы. Справочник / Д.А. Тархов. - М.: Радиотехника, 2014. - 643 с.

17. Гренандер У. Лекции по теории образов (Том 2. Анализ образов) / У. Гренандер. - М.: [не указано], 2016. - 342 с.

18. Елисеева И.И. Группировка, корреляция, распознавание образов (статистические методы классификации и измерения связей) / И.И. Елисеева, В.О. Рукавишников. - Москва: РГГУ, 2014. - 144 с.

19. Фукунага К. Введение в статистическую теорию распознавания образов / К. Фукунага. - М.: Главная редакция физико-математической литературы издательства "Наука", 2013. - 368 с.

20. Потапов А.А. Анализ изображений и распознавание образов / А. ыПотапов. - М.: LAP Lambert Academic Publishing, 2017. - 292 с.

21. Бишоп К.М. Распознавание образов и машинное обучение, том 1 / К.М. Бишоп – М.: Диалектика-Вильямс, 2020. - 480 с.

22. Нгуен Т.Т., Спицын В.Г. Алгоритмическое и программное обеспечение для распознавания формы руки в реальном времени с использованием surfcдескрипторов и нейронной сети // Известия ТПУ. 2012. №5. URL: https://cyberleninka.ru/article/n/algoritmicheskoe-i-programmnoe-obespechenie-dlya-raspoznavaniya-formy-ruki-v-realnom-vremeni-s-ispolzovaniem-surfcdeskriptorov-i

23. Джгаркава Г.М., Лавров Д.Н. Использование метода SURF для обнаружения устойчивых признаков изображения при создании сферических панорамных снимков // МСиМ. 2011. №1 (22). URL: https://cyberleninka.ru/article/n/ispolzovanie-metoda-surf-dlya-obnaruzheniya-

ustoychivyh-priznakov-izobrazheniya-pri-sozdanii-sfericheskih-panoramnyh-snimkov

24. Обнаружение устойчивых признаков изображения: метод SURF. [Электронный ресурс] URL: http://twinpeppers.blogspot.com/2012/03/surf-httphabrahabrrupost103107.html

25. Сидорина Т.С. Microsoft Visual Studio C++ и MFC / Т.С. Сидорина. - М.: "БХВ-Петербург", 2014. - 848 с.

26. Полубенцева М. С/С++. Процедурное программирование / М. Полубенцева. - М.: БХВ-Петербург, 2014. - 448 с.

27. Дьюхарст С.С. Программирование на С++. - М.: ДиаСофт, 2015. – 272 с.

28. Понамарев В. Программирование на С++/С# в Visual Studio .NET 2003 / В. Понамарев. - М.: БХВ-Петербург, 2013. - 352 с.

29. Хьюз К. Параллельное и распределенное программирование с использованием С++: моногр. / К. Хьюз , Т. Хьюз. - М.: Вильямс, 2016. - 672 с.

30. Хенкеманс Д. Программирование на С++ / Д. Хенкеманс, М. Ли. - СПб.: Символ-плюс, 2015. - 416 с.

31. Игошин В.И. Теория алгоритмов: Учебное пособие / В.И. Игошин. - М.: ИНФРА-М, 2013. - 318 с.

32. Канцедал С.А. Алгоритмизация и программирование : Учебное пособие / С.А. Канцедал. - М.: ИД Форум, НИЦ Инфра-М, 2013. - 352 с.

33. Крупский В.Н. Математическая логика и теория алгоритмов: Учебное пособие для студентов учреждений высшего проф. образования / В.Н. Крупский, В.Е. Плиско. - М.: ИЦ Академия, 2013. - 416 с.

34. Семакин И.Г. Основы алгоритмизации и программирования. Практикум: Учебное пос. для студ. учреждений сред. проф. образования / И.Г. Семакин, А.П. Шестаков. - М.: ИЦ Академия, 2013. - 144 с.

35. Судоплатов С.В. Математика: математическая логика и теория алгоритмов : учебник и практикум для среднего профессионального образования / С. В. Судоплатов Е.В. Овчинникова. — 5-е изд., стер. — Москва : Издательство

Юрайт, 2019. — 255 с.

36. Семакин И.Г. Основы алгоритмизации и программирования: Учебник для студ. учреждений сред. проф. образования / И.Г. Семакин, А.П. Шестаков. - М.: ИЦ Академия, 2013. - 304 с.

37. Трофимов В.В. Алгоритмизация и программирование : учебник для академического бакалавриата / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — Москва : Издательство Юрайт, 2019. — 137 с.

38. Черняк А.А. Методы оптимизации: теория и алгоритмы : учебное пособие для академического бакалавриата / А. А. Черняк, Ж. А. Черняк, Ю. М. Метельский, С. А. Богданович. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 357 с.

39. Пароджанов В. Учись писать, читать и понимать алгоритмы / В. Пароджанов. - М.: ДМК Пресс, 2016. - 458 с.

40. Ахо А.В. Структуры данных и алгоритмы / А .В. Ахо, Д.Э. Хопкрофт, Д.Д. Ульман. - М.: Вильямс, 2016. - 400 с.

41. Чернов А.В. Интегрированная среда для исследования "обфускации" программ. Доклад на конференции, посвящённой 90-летию со дня рождения А.А. Ляпунова. Россия, Новосибирск, 8-11 октября 2001 года. http://www.ict.nsc.ru/ws/Lyap2001/2350/

42. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. Современное состояние исследований в области обфускации программ: определения стойкости обфускации // Труды ИСП РАН. 2014. №3. URL: https://cyberleninka.ru/article/n/sovremennoe-sostoyanie-issledovaniy-v-oblasti-obfuskatsii-programm-opredeleniya-stoykosti-obfuskatsii

43. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. О стойкой обфускации компьютерных программ // Экономика. Информатика. 2009. №15 (70). URL: https://cyberleninka.ru/article/n/o-stoykoy-obfuskatsii-kompyuternyh-programm-1

44. Козачок А.В., Бочков М.В., Лай М .Т. Теоретическое обоснование

стойкости неразличимой обфускации // Вопросы кибербезопасности. 2016. №1 (14). URL: https://cyberleninka.ru/article/n/teoreticheskoe-obosnovanie-stoykosti-nerazlichimoy-obfuskatsii

45. Поздеев А.Г., Кривопалов В.Н., Ромашкин Е.В., Радченко Е.Д. Математические и программные средства обфускации программ // ПДМ. Приложение. 2009. №1. URL: https://cyberleninka.ru/article/n/matematicheskie-i-programmnye-sredstva-obfuskatsii-programm

46. Конкин Ю.В., Колесенков А.Н. Распознавание изображений на основе текстурных признаков Харалика и искусственных нейронных сетей // Известия ТулГУ. Технические науки. 2016. №2. URL: https://cyberleninka.ru/article/n/raspoznavanie-izobrazheniy-na-osnove-teksturnyh-priznakov-haralika-i-iskusstvennyh-neyronnyh-setey

47. Сагитов Р.Р., Мустафина С.А. Обучение нейронных сетей: распознавание изображений // Марчуковские научные чтения. 2020. №2020. URL: https://cyberleninka.ru/article/n/obuchenie-neyronnyh-setey-raspoznavanie-izobrazheniy

48. Шустов В.А. Алгоритмы обучения нейронных сетей распознаванию изображений по равномерному критерию // КО. 2003. №25. URL: https://cyberleninka.ru/article/n/algoritmy-obucheniya-neyronnyh-setey-raspoznavaniyu-izobrazheniy-po-ravnomernomu-kriteriyu

49. Горин В.В. Распознавание изображений на основе персептрона // Science Time. 2015. №5 (17). URL: https://cyberleninka.ru/article/n/raspoznavanie-izobrazheniy-na-osnove-perseptrona

50. Ферцев А.А. Реализация нейронной сети для распознавания изображений с помощью технологии Nvdia CUDA // Прикладная информатика. 2011. №6 (36). URL: https://cyberleninka.ru/article/n/realizatsiya-neyronnoy-seti-dlya-raspoznavaniya-izobrazheniy-s-pomoschyu-tehnologii-nvdia-cuda

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи розпізнавання зображень на основі нейронних мереж.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 189-02 від 28.12.2020 року).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи розпізнавання зображень на основі нейронних мереж.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи розпізнавання зображень на основі нейронних мереж;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мові програмування C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 63 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 9.06.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ Мелешко Є.В.

Програмне забезпечення системи розпізнавання зображень на основі нейронних мереж

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 25

Літера: РП

Кропивницький – 2021 року

MainNeuralNet.cpp – навчання та тестування нейронної мережі

```
#include "BackProp.h"
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{

// ---навачальні дані---

//Створюємо файловий потік і пов'язуємо його з файлом
      ifstream in("train.txt");

      //Відкриття файлу з навчальними даними

      if (in.is_open())
      {

            int count = 0;// кількість чисел у файлі
            int temp;

            //підрахунок кількості чисел у файлі
            while (!in.eof())
            {
                  in >> temp;
                  count++;
            }

            //Підрахунок кількості чисел у рядку

            //переведемо каретку в потоці в початок файлу
            in.seekg(0, ios::beg);
            in.clear();

            //Число пробілів в першому рядку спочатку дорівнює 0
            int count_space = 0;
            char symbol;
            while (!in.eof())
            {
                  //посимвольно читаємо дані
                  in.get(symbol);
                  if (symbol == ' ') count_space++;//Якщо це пробіл, то число
пробілів збільшуємо
                  if (symbol == '\n') break;//Якщо кінець рядка - виходимо з
циклу
            }
            //cout << count_space << endl;

            //Переходимо в потоці в початок файлу
            in.seekg(0, ios::beg);
            in.clear();

            //Читаємо з файлу матрицю

            int n = count / (count_space + 1);//число рядків
            int m = count_space + 1;//число стовпців на одиницю більше числа
пробілів
            double **data;
            data = new double*[n];
            for (int i = 0; i<n; i++) data[i] = new double[m];

            //Вважаємо матрицю з файлу
            for (int i = 0; i < n; i++)
                  for (int j = 0; j < m; j++)
                        in >> data[i][j];
```

```
			in.close();//закриваємо файл
		}
		else
		{
			//Якщо відкриття файлу пройшло не успішно
			cout << "Файл не знайдено";
		}

		system("pause");
		return 0;
}


		// ---тестові дані---

		//Створюємо файловий потік і пов'язуємо його з файлом
		ifstream in("test.txt");

		//Відкриття файлу з тестовими даними

		if (in.is_open())
		{

			int count = 0;// кількість чисел у файлі
			int temp;

			//підрахунок кількості чисел у файлі
			while (!in.eof())
			{
				in >> temp;
				count++;
			}

			//Підрахунок кількості чисел у рядку

			//переведемо каретку в потоці в початок файлу
			in.seekg(0, ios::beg);
			in.clear();

			//Число пробілів в першому рядку спочатку дорівнює 0
			int count_space = 0;
			char symbol;
			while (!in.eof())
			{
				//посимвольно читаємо дані
				in.get(symbol);
				if (symbol == ' ') count_space++;//Якщо це пробіл, то число
пробілів збільшуємо
				if (symbol == '\n') break;//Якщо кінець рядка - виходимо з
циклу
			}
			//cout << count_space << endl;

			//Переходимо в потоці в початок файлу
			in.seekg(0, ios::beg);
			in.clear();

			//Читаємо з файлу матрицю

			int n2 = count / (count_space + 1);//число рядків
			int m2 = count_space + 1;//число стовпців на одиницю більше числа
пробілів
			double **test_data;
			test_data = new double*[n2];
			for (int i = 0; i<n2; i++) test_data[i] = new double[m2];

			//Вважаємо матрицю з файлу
			for (int i = 0; i < n2; i++)
				for (int j = 0; j < m2; j++)
```

```
                        in >> test_data[i][j];

            in.close();//закриваємо файл
      }
      else
      {
            //Якщо відкриття файлу пройшло не успішно
            cout << "Файл не знайдено";
      }

      system("pause");
      return 0;
}


// визначення мережі з 5 шарами, що мають 128,200,200,50,10 нейронів відповідно,
// перший шар вхідний, тобто просто одержувач вхідних параметрів
// і повинен мати такий самий розмір, як кількість вхідних параметрів

      int numl_layerayers = 5, lSz[5] = {128,200,200,50,10};


      // speed - швидкість навчання
      // alpha - імпульс
      // thresh – поріг чутливості нейрону (цільове значення помилки mse,
навчання нейромережі припиняється після його досягнення)

      double speed = 0.3, alpha = 0.1, Thresh =  0.00001;

      // максимум ітерацій під час тренування
      long num_iter = 2000000;


      // Створення мережі
      CBackProp *bp = new CBackProp(numl_layerayers, lSz, speed, alpha);

      cout<< endl <<  " Зараз нейронна мережа навчається...." << endl;
      for (long i=0; i<num_iter ; i++)
      {

            bp->bpgt(data[i%8], &data[i%8][m]);

            if( bp->mse(&data[i%8][m]) < Thresh) {
                  cout << endl << " Мережа навчена. Порогове значення, досягнуте
за " << i << " ітерацій." << endl;
                  cout << "MSE:  " << bp->mse(&data[i%8][m])
                        <<  endl <<  endl;
                  break;
            }
            if ( i%(num_iter/10) == 0 )
                  cout<<  endl <<  "MSE:  " << bp->mse(&data[i%8][m])
                        << "... Training..." << endl;
      }

      if ( i == num_iter )
            cout << endl << i << ітерацій завершено..."
            << "MSE: " << bp->mse(&data[(i-1)%8][m]) << endl;

      cout<< "Навчання завершилося. Тепер нейронну мережу можна використовувати
...." << endl << endl;
      for ( i = 0 ; i < 8 ; i++ )
      {
            bp->ffwd(testData[m2]);
            cout << testData[i][0]<< "  " << testData[i][1]<< "  "  <<
testData[i][2]<< "  " << bp->Out(0) << endl;
      }
      system("pause");
      return 0;
}
```

BackProp.cpp – реалізація нейронної мережі зворотнього поширення

```
#include "backprop.h"
#include <time.h>
#include <stdlib.h>

//ініціалізація
CBackProp::CBackProp(int nl,int *sz,double b,double a):speed(b),alpha(a)
{

	//	множина шарів та їх розміри
	numl_layer=nl;
	layer_size=new int[numl_layer];

	for(int i=0;i<numl_layer;i++){
		layer_size[i]=sz[i];
	}

	//	виділити пам'ять для виведення кожного нейрона
	out = new double*[numl_layer];

	for( i=0;i<numl_layer;i++){
		out[i]=new double[layer_size[i]];
	}

	//	виділити пам'ять для дельти
	dlt = new double*[numl_layer];

	for(i=1;i<numl_layer;i++){
		dlt[i]=new double[layer_size[i]];
	}

	//	виділити пам'ять для ваг
	weight = new double**[numl_layer];

	for(i=1;i<numl_layer;i++){
		weight[i]=new double*[layer_size[i]];
	}
	for(i=1;i<numl_layer;i++){
		for(int j=0;j<layer_size[i];j++){
			weight[i][j]=new double[layer_size[i-1]+1];
		}
	}

	//	allocate memory for previous weights
	prevDwt = new double**[numl_layer];

	for(i=1;i<numl_layer;i++){
		prevDwt[i]=new double*[layer_size[i]];

	}
	for(i=1;i<numl_layer;i++){
		for(int j=0;j<layer_size[i];j++){
			prevDwt[i][j]=new double[layer_size[i-1]+1];
		}
	}

	//	проініціалізувати ваги випадковими значеннями
	srand((unsigned)(time(NULL)));
	for(i=1;i<numl_layer;i++)
		for(int j=0;j<layer_size[i];j++)
			for(int k=0;k<layer_size[i-1]+1;k++)
				weight[i][j][k]=(double)(rand())/(RAND_MAX/2) - 1;
	for(i=1;i<numl_layer;i++)
		for(int j=0;j<layer_size[i];j++)
			for(int k=0;k<layer_size[i-1]+1;k++)
				prevDwt[i][j][k]=(double)0.0;

}
```

```
CBackProp::~CBackProp()
{
	//	видалити out
	for(int i=0;i<numl_layer;i++)
		delete[] out[i];
	delete[] out;

	//	видалити dlt
	for(i=1;i<numl_layer;i++)
		delete[] dlt[i];
	delete[] dlt;

	//	видалити weight
	for(i=1;i<numl_layer;i++)
		for(int j=0;j<layer_size[i];j++)
			delete[] weight[i][j];
	for(i=1;i<numl_layer;i++)
		delete[] weight[i];
	delete[] weight;

	//	видалити prevDwt
	for(i=1;i<numl_layer;i++)
		for(int j=0;j<layer_size[i];j++)
			delete[] prevDwt[i][j];
	for(i=1;i<numl_layer;i++)
		delete[] prevDwt[i];
	delete[] prevDwt;

	//	видалити інфорацію про прошарки
	delete[] layer_size;
}

//	сигмовидна функція активації
double CBackProp::sigmoid(double in)
{
		return (double)(1/(1+exp(-in)));
}

//	середньоквадратична помилка
double CBackProp::mse(double *tgt) const
{
	double mse=0;
	for(int i=0;i<layer_size[numl_layer-1];i++){
		mse+=(tgt[i]-out[numl_layer-1][i])*(tgt[i]-out[numl_layer-1][i]);
	}
	return mse/2;
}


//	повертає i-ий результат мережі
double CBackProp::Out(int i) const
{
	return out[numl_layer-1][i];
}

// подати один набір вхідних даних
void CBackProp::ffwd(double *in)
{
	double sum;

	//	призначити вхідний вектор вхідному шару
	for(int i=0;i<layer_size[0];i++)
		out[0][i]=in[i];  // output_from_neuron(i,j) Jth neuron in Ith Layer

// присвоюємо вихідне (активаційне) значення
// до кожного нейрону застосовуємо функцію активації
	for(i=1;i<numl_layer;i++){                    // Для кожного шару
```

```
            for(int j=0;j<layer_size[i];j++){        // Для кожного нейрона в
поточному шарі

                  sum=0.0;
                  for(int k=0;k<layer_size[i-1];k++){      // Для одержання
даних з кожного нейрона попереднього шару
                        sum+= out[i-1][k]*weight[i][j][k]; // Обчислення
зваженої суми
                  }
                  sum+=weight[i][j][layer_size[i-1]];
                  out[i][j]=sigmoid(sum);                  // Функція активації
            }
      }
}


// повернення значень помилок
void CBackProp::bpgt(double *in,double *tgt)
{
      double sum;

      //    оновити вихідні значення для кожного нейрона
      ffwd(in);

      //    знайти дельту для вихідного шару
      for(int i=0;i<layer_size[numl_layer-1];i++){
            dlt[numl_layer-1][i]=out[numl_layer-1][i]*
            (1-out[numl_layer-1][i])*(tgt[i]-out[numl_layer-1][i]);
      }

      //    знайти дельту для прихованих шарів
      for(i=numl_layer-2;i>0;i--){
            for(int j=0;j<layer_size[i];j++){
                  sum=0.0;
                  for(int k=0;k<layer_size[i+1];k++){
                        sum+=dlt[i+1][k]*weight[i+1][k][j];
                  }
                  dlt[i][j]=out[i][j]*(1-out[i][j])*sum;
            }
      }

      //    застосувати імпульс (нічого не робить, якщо alpha=0)
      for(i=1;i<numl_layer;i++){
            for(int j=0;j<layer_size[i];j++){
                  for(int k=0;k<layer_size[i-1];k++){
                        weight[i][j][k]+=alpha*prevDwt[i][j][k];
                  }
                  weight[i][j][layer_size[i-
1]]+=alpha*prevDwt[i][j][layer_size[i-1]];
            }
      }

      //    відрегулюйте вагу, використовуючи крутий спуск
      for(i=1;i<numl_layer;i++){
            for(int j=0;j<layer_size[i];j++){
                  for(int k=0;k<layer_size[i-1];k++){
                        prevDwt[i][j][k]=speed*dlt[i][j]*out[i-1][k];
                        weight[i][j][k]+=prevDwt[i][j][k];
                  }
                  prevDwt[i][j][layer_size[i-1]]=speed*dlt[i][j];
                  weight[i][j][layer_size[i-1]]+=prevDwt[i][j][layer_size[i-1]];
            }
      }
}
```

BackProp.h – файл заголовків

```
// файл заголовків для реалізації багатошарової нейронної мережі з використанням
алгоритму навчання зворотного поширення помилки

#ifndef backprop_h
#define backprop_h
#include<assert.h>
#include<iostream.h>
#include<stdio.h>
#include<math.h>


class CBackProp{

//    вихід кожного нейрону
      double **out;

//    значення дельти-помилки для кожного нейрона
      double **dlt;

//    вектор ваг для кожного нейрона
      double ***weight;

//    кількість шарів у нейронній мережі
//    including input layer
      int numl_layer;

//    вектор кількості елементів у кожному шарі нейронної мережі
      int *layer_size;

//    швидкість навчання
      double speed;

//    параметр імпульсу
      double alpha;

// збереження значень для зміни ваг
// в попередню епоху
      double ***prevDwt;

//    функція стиснення
      double sigmoid(double in);

public:

      ~CBackProp();

//    ініціалізує та виділяє пам'ять
      CBackProp(int nl,int *sz,double b,double a);

//    Помилка для одного набору вхідних даних
      void bpgt(double *in,double *tgt);

//    активація для одного набору входів
      void ffwd(double *in);

//    повертає середньоквадратичну помилку мережі mse
      double mse(double *tgt) const;

//    повертає i-й результат мережі
      double Out(int i) const;
};

#endif
```

mainSURF.cpp – створення вхідного вектору для нейронної мережі за допомогою алгоритму визначення особливих точок SURF

```
#include "surflib.h"
#include "kmeans.h"
#include <ctime>
#include <iostream>
#include <fstream>


int mainImage(void)
{
  // Оголошення Ipoints та інших змінних
  IpVec ipts;
  IplImage *img=cvLoadImage("imgs/sf.jpg");

  // Визначення та описання потрібних ключових точок у зображенні
  clock_t start = clock();
  surfDetDes(img, ipts, false, 5, 4, 2, 0.0004f);
  clock_t end = clock();

  std::cout<< "Програма визначення ключових точок у зображенні методом SURF
знайшла: " << ipts.size() << " особливі точки" << std::endl;
  std::cout<< "Програма визначення ключових точок у зображенні методом SURF
виконується: " << float(end - start) / CLOCKS_PER_SEC  << " секунд" <<
std::endl;

  // Відображення знайдених особливих точок
  drawIpoints(img, ipts);

  std::ofstream out;          // поток для запису у файл
  out.open("D:\\train.txt" , std::ios::app); // відкриваємо файл для дозапису
  if (out.is_open())
  {
     out << ipts << std::endl;
   }

   out.close ();

  // Виведення результату на екран
  showImage(img);

  return 0;
}

//-------------------------------------------------------

int mainVideo(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("No Capture");

  // Ініціалізація пристрою відеозапису
  //cv::VideoWriter vw("c:\\out.avi",
CV_FOURCC('D','I','V','X'),10,cvSize(320,240),1);
  //vw << img;

  // Створюємо вікно
  cvNamedWindow("Програма визначення ключових точок у зображенні методом SURF",
CV_WINDOW_AUTOSIZE );

  // Оголошення Ipoints та інших змінних
  IpVec ipts;
  IplImage *img=NULL;

  // Прокручуємо головну картинку
  while( 1 )
```

```
  {
    // Вирізаємо фрейм з джерела картинки
    img = cvQueryFrame(capture);

    // Отримуємо точки для методу SURF
    surfDetDes(img, ipts, false, 4, 4, 2, 0.004f);

    // Відображення знайдених особливих точок
    drawIpoints(img, ipts);

    std::ofstream out;          // поток для запису у файл
    out.open("D:\\train.txt" , std::ios::app); // відкриваємо файл для дозапису
    if (out.is_open())
    {
        out << ipts << std::endl;
     }

   out.close ();

    // Виводимо фігуру FPS
    drawFPS(img);

    // Виведення результату на екран
    cvShowImage("Програма визначення ключових точок у зображенні методом SURF",
img);

    // Якщо нажата клавіша ESC перериваємо прокручування
    if( (cvWaitKey(10) & 255) == 27 ) break;
  }

  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма визначення ключових точок у зображенні методом
SURF" );
  return 0;
}


//-------------------------------------------------------


int mainMatch(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("No Capture");

  // Оголошення Ipoints та інших змінних
  IpPairVec matches;
  IpVec ipts, ref_ipts;

  // Описуємо пошук необхідних  точок на  відеокадрі
  // Це описано у рядку IplImage *img = cvLoadImage("imgs/object.jpg");
  // де object.jpg потрібний нам кадр з відео
  IplImage *img = cvLoadImage("imgs/object.jpg");
  if (img == NULL) error("Потрібно завантажити довідкове зображення для того,
щоб управляти відповідністю процедури");
  CvPoint src_corners[4] = {{0,0}, {img->width,0}, {img->width, img->height},
{0, img->height}};
  CvPoint dst_corners[4];

  // Витягуємо довідковий об’єкт Ipoints
  surfDetDes(img, ref_ipts, false, 3, 4, 3, 0.004f);
  drawIpoints(img, ref_ipts);
  showImage(img);

  std::ofstream out;          // поток для запису у файл
  out.open("D:\\train.txt" , std::ios::app); // відкриваємо файл для дозапису
  if (out.is_open())
```

```
  {
     out << ref_ipts << std::endl;
   }

   out.close ();

  // Створюємо вікно
  cvNamedWindow("Програма визначення ключових точок у зображенні методом SURF",
CV_WINDOW_AUTOSIZE );

  // Прокручуємо головну картинку
  while( true )
  {
    // Вирізаємо фрейм з джерела картинки
    img = cvQueryFrame(capture);

    // Визначаємо та описуємо особливі точки у фреймі
    surfDetDes(img, ipts, false, 3, 4, 3, 0.004f);

    // Рисуємо відповідний вектор
    getMatches(ipts,ref_ipts,matches);

    // Цей виклик знаходить, де об'єктні кути мають бути у фреймі
    if (translateCorners(matches, src_corners, dst_corners))
    {
      // Рисуємо фігуру вокруг об'єкту
      for(int i = 0; i < 4; i++ )
      {
        CvPoint r1 = dst_corners[i%4];
        CvPoint r2 = dst_corners[(i+1)%4];
        cvLine( img, cvPoint(r1.x, r1.y),
          cvPoint(r2.x, r2.y), cvScalar(255,255,255), 3 );
      }

      for (unsigned int i = 0; i < matches.size(); ++i)
        drawIpoint(img, matches[i].first);
    }

    // Виводимо фігуру FPS
    drawFPS(img);

    // Виведення результату на екран
    cvShowImage("Програма визначення ключових точок у зображенні методом SURF",
img);

    // Якщо нажата клавіша ESC перериваємо прокручування
    if( (cvWaitKey(10) & 255) == 27 ) break;
  }

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма визначення ключових точок у зображенні методом
SURF" );
  return 0;
}


//---------------------------------------------------------


int mainMotionPoints(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("No Capture");

  // Створюємо вікно
```

```
  cvNamedWindow("Програма визначення ключових точок у зображенні методом SURF",
CV_WINDOW_AUTOSIZE );

  // Оголошення Ipoints та інших змінних
  IpVec ipts, old_ipts, motion;
  IpPairVec matches;
  IplImage *img;

  // Прокручуємо головну картинку
  while( 1 )
  {
    // Вирізаємо фрейм з джерела картинки
    img = cvQueryFrame(capture);

    // Визначення та описання потрібних ключових точок у зображенні
    old_ipts = ipts;
    surfDetDes(img, ipts, true, 3, 4, 2, 0.0004f);

    // Рисуємо відповідний вектор
    getMatches(ipts,old_ipts,matches);
    for (unsigned int i = 0; i < matches.size(); ++i)
    {
      const float & dx = matches[i].first.dx;
      const float & dy = matches[i].first.dy;
      float speed = sqrt(dx*dx+dy*dy);
      if (speed > 5 && speed < 30)
        drawIpoint(img, matches[i].first, 3);
    }

    // Виведення результату на екран
    cvShowImage("Програма визначення ключових точок у зображенні методом SURF",
img);

    // Якщо нажата клавіша ESC перериваємо прокручування
    if( (cvWaitKey(10) & 255) == 27 ) break;
  }

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма визначення ключових точок у зображенні методом
SURF" );
  return 0;
}


//---------------------------------------------------

int mainStaticMatch()
{
  IplImage *img1, *img2;
  img1 = cvLoadImage("imgs/img1.jpg");
  img2 = cvLoadImage("imgs/img2.jpg");

  IpVec ipts1, ipts2;
  surfDetDes(img1,ipts1,false,4,4,2,0.0001f);
  surfDetDes(img2,ipts2,false,4,4,2,0.0001f);

  IpPairVec matches;
  getMatches(ipts1,ipts2,matches);

  for (unsigned int i = 0; i < matches.size(); ++i)
  {
    drawPoint(img1,matches[i].first);
    drawPoint(img2,matches[i].second);

    const int & w = img1->width;

cvLine(img1,cvPoint(matches[i].first.x,matches[i].first.y),cvPoint(matches[i].se
cond.x+w,matches[i].second.y), cvScalar(255,255,255),1);
```

```
    cvLine(img2,cvPoint(matches[i].first.x-
w,matches[i].first.y),cvPoint(matches[i].second.x,matches[i].second.y),
cvScalar(255,255,255),1);
  }

  std::cout<< "Відповідає: " << matches.size();

  cvNamedWindow("1", CV_WINDOW_AUTOSIZE );
  cvNamedWindow("2", CV_WINDOW_AUTOSIZE );
  cvShowImage("1", img1);
  cvShowImage("2",img2);
  cvWaitKey(0);

  return 0;
}

//--------------------------------------------------

int mainKmeans(void)
{
  IplImage *img = cvLoadImage("imgs/img1.jpg");
  IpVec ipts;
  Kmeans km;

  // Беремо Ipoints
  surfDetDes(img,ipts,true,3,4,2,0.0006f);

  for (int repeat = 0; repeat < 10; ++repeat)
  {

    IplImage *img = cvLoadImage("imgs/img1.jpg");
    km.Run(&ipts, 5, true);
    drawPoints(img, km.clusters);

    for (unsigned int i = 0; i < ipts.size(); ++i)
    {
      cvLine(img, cvPoint(ipts[i].x,ipts[i].y),
cvPoint(km.clusters[ipts[i].clusterIndex].x
,km.clusters[ipts[i].clusterIndex].y),cvScalar(255,255,255));
    }

    showImage(img);
  }

  return 0;
}

//--------------------------------------------------

int main(void)
{

mainImage();

}
```

surf.cpp – реалізація алгоритму SURF

```
// Програма визначення ключових точок у зображенні методом SURF

#include "utils.h"

#include "surf.h"

// SURF константи
const float pi = 3.14159f;

const double gauss25 [7][7] = {

0.02350693969273,0.01849121369071,0.01239503121241,0.00708015417522,0.0034462810
1733,0.00142945847484,0.00050524879060,

0.02169964028389,0.01706954162243,0.01144205592615,0.00653580605408,0.0031813183
4134,0.00131955648461,0.00046640341759,

0.01706954162243,0.01342737701584,0.00900063997939,0.00514124713667,0.0025025136
4222,0.00103799989504,0.00036688592278,

0.01144205592615,0.00900063997939,0.00603330940534,0.00344628101733,0.0016774850
5986,0.00069579213743,0.00024593098864,

0.00653580605408,0.00514124713667,0.00344628101733,0.00196854695367,0.0009581946
7066,0.00039744277546,0.00014047800980,

0.00318131834134,0.00250251364222,0.00167748505986,0.00095819467066,0.0004664034
1759,0.00019345616757,0.00006837798818,

0.00131955648461,0.00103799989504,0.00069579213743,0.00039744277546,0.0001934561
6757,0.00008024231247,0.00002836202103
};

const double gauss33 [11][11] = {

0.014614763,0.013958917,0.012162744,0.00966788,0.00701053,0.004637568,0.00279865
7,0.001540738,0.000773799,0.000354525,0.000148179,

0.013958917,0.013332502,0.011616933,0.009234028,0.006695928,0.004429455,0.002673
066,0.001471597,0.000739074,0.000338616,0.000141529,

0.012162744,0.011616933,0.010122116,0.008045833,0.005834325,0.003859491,0.002329
107,0.001282238,0.000643973,0.000295044,0.000123318,

0.00966788,0.009234028,0.008045833,0.006395444,0.004637568,0.003067819,0.0018513
53,0.001019221,0.000511879,0.000234524,9.80224E-05,

0.00701053,0.006695928,0.005834325,0.004637568,0.003362869,0.002224587,0.0013424
83,0.000739074,0.000371182,0.000170062,7.10796E-05,

0.004637568,0.004429455,0.003859491,0.003067819,0.002224587,0.001471597,0.000888
072,0.000488908,0.000245542,0.000112498,4.70202E-05,

0.002798657,0.002673066,0.002329107,0.001851353,0.001342483,0.000888072,0.000535
929,0.000295044,0.000148179,6.78899E-05,2.83755E-05,

0.001540738,0.001471597,0.001282238,0.001019221,0.000739074,0.000488908,0.000295
044,0.00016243,8.15765E-05,3.73753E-05,1.56215E-05,

0.000773799,0.000739074,0.000643973,0.000511879,0.000371182,0.000245542,0.000148
179,8.15765E-05,4.09698E-05,1.87708E-05,7.84553E-06,

0.000354525,0.000338616,0.000295044,0.000234524,0.000170062,0.000112498,6.78899E
-05,3.73753E-05,1.87708E-05,8.60008E-06,3.59452E-06,
```

```
  0.000148179,0.000141529,0.000123318,9.80224E-05,7.10796E-05,4.70202E-
05,2.83755E-05,1.56215E-05,7.84553E-06,3.59452E-06,1.50238E-06
};

//-------------------------------------------------------

//-------------------------------------------------------

// Конструктор
Surf::Surf(IplImage *img, IpVec &ipts)
: ipts(ipts)
{
  this->img = img;
}

//-------------------------------------------------------

// Опишемо усі особливості у векторі, що поставляється
void Surf::getDescriptors(bool upright)
{
  // Перевіряємо Ipoints на опис
  if (!ipts.size()) return;

  // Беремо розмір вектору для фіксованих меж циклу
  int ipts_size = (int)ipts.size();

  if (upright)
  {
    // U-SURF цикл тільки отримує дескриптори
    for (int i = 0; i < ipts_size; ++i)
    {
      // Встановлюємо Ipoint на опис
      index = i;

      // Витягуємо  вертикальні (тобто інваріант не обертання) дескриптори
      getDescriptor(true);
    }
  }
  else
  {
    // Головний SURF-64 цикл визначення орієнтації та отримання дескрипторів
    for (int i = 0; i < ipts_size; ++i)
    {
      // Встановлюємо Ipoint на опис
      index = i;

      // Призначаємо орієнтацію і витягуємо дескриптори інваріанту обертання
      getOrientation();
      getDescriptor(false);
    }
  }
}

//-------------------------------------------------------

// Призначаємо поставляння Ipoint на орієнтацію
void Surf::getOrientation()
{
  Ipoint *ipt = &ipts[index];
  float gauss = 0.f, scale = ipt->scale;
  const int s = fRound(scale), r = fRound(ipt->y), c = fRound(ipt->x);
  std::vector<float> resX(109), resY(109), Ang(109);
  const int id[] = {6,5,4,3,2,1,0,1,2,3,4,5,6};

  int idx = 0;
  // розраховуємо відповідні для точок Хаара в межах радіусу 6*масштаб
  for(int i = -6; i <= 6; ++i)
  {
    for(int j = -6; j <= 6; ++j)
```

```
    {
      if(i*i + j*j < 36)
      {
        gauss = static_cast<float>(gauss25[id[i+6]][id[j+6]]);
        resX[idx] = gauss * haarX(r+j*s, c+i*s, 4*s);
        resY[idx] = gauss * haarY(r+j*s, c+i*s, 4*s);
        Ang[idx] = getAngle(resX[idx], resY[idx]);
        ++idx;
      }
    }
  }

  // розраховуємо основний напрямок
  float sumX=0.f, sumY=0.f;
  float max=0.f, orientation = 0.f;
  float ang1=0.f, ang2=0.f;

  // цикл слайдів pi/3 вікно біля точки, яка може бути
  for(ang1 = 0; ang1 < 2*pi;  ang1+=0.15f) {
    ang2 = ( ang1+pi/3.0f > 2*pi ? ang1-5.0f*pi/3.0f : ang1+pi/3.0f);
    sumX = sumY = 0.f;
    for(unsigned int k = 0; k < Ang.size(); ++k)
    {
      // беремо angle з x-axis для точки прикладу
      const float & ang = Ang[k];

      // визначаємо чи є точка в межах вікна
      if (ang1 < ang2 && ang1 < ang && ang < ang2)
      {
        sumX+=resX[k];
        sumY+=resY[k];
      }
      else if (ang2 < ang1 &&
        ((ang > 0 && ang < ang2) || (ang > ang1 && ang < 2*pi) ))
      {
        sumX+=resX[k];
        sumY+=resY[k];
      }
    }

    // якщо вектор робив від цього вікна довше, ніж усі попередні вектори потім
це формує новий домінуючий напрям
    if (sumX*sumX + sumY*sumY > max)
    {
      // запам'ятовуємо найбільшу орієнтацію
      max = sumX*sumX + sumY*sumY;
      orientation = getAngle(sumX, sumY);
    }
  }

  // призначаємо орієнтацію домінуючого відповідному вектору
  ipt->orientation = orientation;
}

//-------------------------------------------------------

// Беремо модифікований дескріптор.

void Surf::getDescriptor(bool bUpright)
{
  int y, x, sample_x, sample_y, count=0;
  int i = 0, ix = 0, j = 0, jx = 0, xs = 0, ys = 0;
  float scale, *desc, dx, dy, mdx, mdy, co, si;
  float gauss_s1 = 0.f, gauss_s2 = 0.f;
  float rx = 0.f, ry = 0.f, rrx = 0.f, rry = 0.f, len = 0.f;
  float cx = -0.5f, cy = 0.f; //Підобласть зосереджується для 4x4 блока гауса

  Ipoint *ipt = &ipts[index];
  scale = ipt->scale;
```

```
x = fRound(ipt->x);
y = fRound(ipt->y);
desc = ipt->descriptor;

if (bUpright)
{
  co = 1;
  si = 0;
}
else
{
  co = cos(ipt->orientation);
  si = sin(ipt->orientation);
}

i = -8;

//Розраховуємо дескриптор для цієї особливої точки
while(i < 12)
{
  j = -8;
  i = i-4;

  cx += 1.f;
  cy = -0.5f;

  while(j < 12)
  {
    dx=dy=mdx=mdy=0.f;
    cy += 1.f;

    j = j - 4;

    ix = i + 5;
    jx = j + 5;

    xs = fRound(x + ( -jx*scale*si + ix*scale*co));
    ys = fRound(y + ( jx*scale*co + ix*scale*si));

    for (int k = i; k < i + 9; ++k)
    {
      for (int l = j; l < j + 9; ++l)
      {
        //Беремо координати визначеної точки та повертаємо їх
        sample_x = fRound(x + (-l*scale*si + k*scale*co));
        sample_y = fRound(y + ( l*scale*co + k*scale*si));

        //Беремо the gaussian weighted x and y responses
        gauss_s1 = gaussian(xs-sample_x,ys-sample_y,2.5f*scale);
        rx = haarX(sample_y, sample_x, 2*fRound(scale));
        ry = haarY(sample_y, sample_x, 2*fRound(scale));

        //Беремо блок Гусса x та y відповідно на вісь обертання
        rrx = gauss_s1*(-rx*si + ry*co);
        rry = gauss_s1*(rx*co + ry*si);
        dx += rrx;
        dy += rry;
        mdx += fabs(rrx);
        mdy += fabs(rry);

      }
    }

    //Додаємо значення до дескриптора вектора
    gauss_s2 = gaussian(cx-2.0f,cy-2.0f,1.5f);

    desc[count++] = dx*gauss_s2;
    desc[count++] = dy*gauss_s2;
    desc[count++] = mdx*gauss_s2;
```

```
      desc[count++] = mdy*gauss_s2;

      len += (dx*dx + dy*dy + mdx*mdx + mdy*mdy) * gauss_s2*gauss_s2;

      j += 9;
    }
    i += 9;
  }

  //конвертуємо до Unit Vector
  len = sqrt(len);
  for(int i = 0; i < 64; ++i)
    desc[i] /= len;

}

//-------------------------------------------------------

// Розраховуємо значення в  2d гауссіані в x,y
inline float Surf::gaussian(int x, int y, float sig)
{
  return (1.0f/(2.0f*pi*sig*sig)) * exp( -(x*x+y*y)/(2.0f*sig*sig));
}

//-------------------------------------------------------

// Розраховуємо значення в  2d гауссіані в x,y
inline float Surf::gaussian(float x, float y, float sig)
{
  return 1.0f/(2.0f*pi*sig*sig) * exp( -(x*x+y*y)/(2.0f*sig*sig));
}

//-------------------------------------------------------

// Розраховуємо вейвлет Хаара в  x напрямку
inline float Surf::haarX(int row, int column, int s)
{
  return BoxIntegral(img, row-s/2, column, s, s/2)
    -1 * BoxIntegral(img, row-s/2, column-s/2, s, s/2);
}

//-------------------------------------------------------

// Розраховуємо вейвлет Хаара в  y напрямку
inline float Surf::haarY(int row, int column, int s)
{
  return BoxIntegral(img, row, column-s/2, s/2, s)
    -1 * BoxIntegral(img, row-s/2, column-s/2, s/2, s);
}

//-------------------------------------------------------

// Беремо вугол з +ve x-axis для вектора  (X Y)
float Surf::getAngle(float X, float Y)
{
  if(X > 0 && Y >= 0)
    return atan(Y/X);

  if(X < 0 && Y >= 0)
    return pi - atan(-Y/X);

  if(X < 0 && Y < 0)
    return pi + atan(Y/X);

  if(X > 0 && Y < 0)
    return 2*pi - atan(-Y/X);

  return 0;
}
```

surf.h – файл заголовків

```
// Програма визначення ключових точок у зображенні методом SURF

#ifndef SURF_H
#define SURF_H

#include <cv.h>
#include "ipoint.h"
#include "integral.h"

#include <vector>

class Surf {

public:

// Стандартний констуктор(img є цілочислене зображення)
Surf(IplImage *img, std::vector<Ipoint> &ipts);

// Опишемо усі особливості у векторі, що поставляється
void getDescriptors(bool bUpright = false);

private:

//---------------- Private Functions -----------------//

// Призначаємо поточнеIpoint на орієнтацію
void getOrientation();

// Беремо дескриптор.
void getDescriptor(bool bUpright = false);

// Розраховуємо значення в  2d гаусciані в x,y
inline float gaussian(int x, int y, float sig);
inline float gaussian(float x, float y, float sig);

// Розраховуємо вейвлет Хаара в  x and y directions
inline float haarX(int row, int column, int size);
inline float haarY(int row, int column, int size);

// Беремо the angle з +ve x-axis of the vector given by [X Y]
float getAngle(float X, float Y);


//---------------- Private Variables -----------------//

// Цілочисельне зображення  де Ipoints визначений
IplImage *img;

// Ipoints вектор
IpVec &ipts;

// Індексуємо поточнеIpoint в вектор
int index;
};

 #endif
```

utils.cpp – візуалізація особливих точок на зображенні, знайдених методом SURF для нейронної мережі

```
#include <highgui.h>
#include <iostream>
#include <fstream>
#include <time.h>
#include "utils.h"
using namespace std;

//-------------------------------------------------------

static const int NCOLOURS = 8;
static const CvScalar COLOURS [] = {cvScalar(255,0,0), cvScalar(0,255,0),
cvScalar(0,0,255), cvScalar(255,255,0),
cvScalar(0,255,255), cvScalar(255,0,255),
cvScalar(255,255,255), cvScalar(0,0,0)};

//-------------------------------------------------------

// Виводимо повідомлення про помилку, та завершуємо програму
void error(const char *msg)
{
cout << "\nError: " << msg;
getchar();
exit(0);
}

//-------------------------------------------------------

// Показуємо зображення й чекаємо натискання клавіши
void showImage(const IplImage *img)
{
cvNamedWindow("Surf", CV_WINDOW_AUTOSIZE);
cvShowImage("Surf", img);
cvWaitKey(0);
}

//-------------------------------------------------------

// Показуємо зображення у головному вікні й чекаємо натискання клавіши
void showImage(char *title,const IplImage *img)
{
cvNamedWindow(title, CV_WINDOW_AUTOSIZE);
cvShowImage(title, img);
cvWaitKey(0);
}

//-------------------------------------------------------

// Конвертуємо зображення по одному каналу32F
IplImage *getGray(const IplImage *img)
{
// Перевіряємо, ми поставляли ненульовий img покажчик
if (!img) error("Не в змозі створити зображення у градаціях сірого кольору.
Немає зображення, що поставляється");

IplImage* gray8, * gray32;

gray32 = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );

if( img->nChannels == 1 )
gray8 = (IplImage *) cvClone( img );
else {
gray8 = cvCreateImage( cvGetSize(img), IPL_DEPTH_8U, 1 );
cvCvtColor( img, gray8, CV_BGR2GRAY );
}
```

```
cvConvertScale( gray8, gray32, 1.0 / 255.0, 0 );

cvReleaseImage( &gray8 );
return gray32;
}

//-------------------------------------------------------

// Виводимо всі Ipoints у забезпеченому векторі
void drawIpoints(IplImage *img, vector<Ipoint> &ipts, int tailSize)
{
Ipoint *ipt;
float s, o;
int r1, c1, r2, c2, lap;

for(unsigned int i = 0; i < ipts.size(); i++)
{
ipt = &ipts.at(i);
s = (2.5f * ipt->scale);
o = ipt->orientation;
lap = ipt->laplacian;
r1 = fRound(ipt->y);
c1 = fRound(ipt->x);
c2 = fRound(s * cos(o)) + c1;
r2 = fRound(s * sin(o)) + r1;

if (o) // Зелена лінія вказує орієнтацію
cvLine(img, cvPoint(c1, r1), cvPoint(c2, r2), cvScalar(0, 255, 0));
else // Зелена точка, якщо, користуючись вертикальною версією
cvCircle(img, cvPoint(c1,r1), 1, cvScalar(0, 255, 0),-1);

if (lap == 1)
{ // Блакитні круги вказують темні краплі на світлих фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(255, 0, 0),1);
}
else if (lap == 0)
{ // Червоні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(0, 0, 255),1);
}
else if (lap == 9)
{ // Червоні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(0, 255, 0),1);
}

// Виводимо рух з  ipoint dx та dy
if (tailSize)
{
cvLine(img, cvPoint(c1,r1),
cvPoint(int(c1+ipt->dx*tailSize), int(r1+ipt->dy*tailSize)),
cvScalar(255,255,255), 1);
}
}
}

//-------------------------------------------------------

// Виводимо єдину особливість на зображенні
void drawIpoint(IplImage *img, Ipoint &ipt, int tailSize)
{
float s, o;
int r1, c1, r2, c2, lap;

s = (2.5f * ipt.scale);
o = ipt.orientation;
lap = ipt.laplacian;
r1 = fRound(ipt.y);
c1 = fRound(ipt.x);

// Зелена лінія вказує орієнтацію
```

```
if (o) // Зелена лінія вказує орієнтацію
{
c2 = fRound(s * cos(o)) + c1;
r2 = fRound(s * sin(o)) + r1;
cvLine(img, cvPoint(c1, r1), cvPoint(c2, r2), cvScalar(0, 255, 0));
}
else // Зелена точка, якщо, користуючись вертикальною версією
cvCircle(img, cvPoint(c1,r1), 1, cvScalar(0, 255, 0),-1);

if (lap >= 0)
{ // Блакитні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(255, 0, 0),1);
}
else
{ // Червоні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(0, 0, 255),1);
}

// Виводимо рух з  ipoint dx and dy
if (tailSize)
{
cvLine(img, cvPoint(c1,r1),
cvPoint(int(c1+ipt.dx*tailSize), int(r1+ipt.dy*tailSize)),
cvScalar(255,255,255), 1);
}
}

//-------------------------------------------------------

// Виводимо єдину особливість на зображенні
void drawPoint(IplImage *img, Ipoint &ipt)
{
float s, o;
int r1, c1;

s = 3;
o = ipt.orientation;
r1 = fRound(ipt.y);
c1 = fRound(ipt.x);

cvCircle(img, cvPoint(c1,r1), fRound(s), COLOURS[ipt.clusterIndex%NCOLOURS], -
1);
cvCircle(img, cvPoint(c1,r1), fRound(s+1),
COLOURS[(ipt.clusterIndex+1)%NCOLOURS], 2);

}

//-------------------------------------------------------

// Виводимо єдину особливість на зображенні
void drawPoints(IplImage *img, vector<Ipoint> &ipts)
{
float s, o;
int r1, c1;

for(unsigned int i = 0; i < ipts.size(); i++)
{
s = 3;
o = ipts[i].orientation;
r1 = fRound(ipts[i].y);
c1 = fRound(ipts[i].x);

cvCircle(img, cvPoint(c1,r1), fRound(s), COLOURS[ipts[i].clusterIndex%NCOLOURS],
-1);
cvCircle(img, cvPoint(c1,r1), fRound(s+1),
COLOURS[(ipts[i].clusterIndex+1)%NCOLOURS], 2);
}
}
```

```
//-------------------------------------------------------

// Виводимо дескрипторні вікна навкруги Ipoints у забезпеченому векторі
void drawWindows(IplImage *img, vector<Ipoint> &ipts)
{
Ipoint *ipt;
float s, o, cd, sd;
int x, y;
CvPoint2D32f src[4];

for(unsigned int i = 0; i < ipts.size(); i++)
{
ipt = &ipts.at(i);
s = (10 * ipt->scale);
o = ipt->orientation;
y = fRound(ipt->y);
x = fRound(ipt->x);
cd = cos(o);
sd = sin(o);

src[0].x=sd*s+cd*s+x; src[0].y=-cd*s+sd*s+y;
src[1].x=sd*s+cd*-s+x; src[1].y=-cd*s+sd*-s+y;
src[2].x=sd*-s+cd*-s+x; src[2].y=-cd*-s+sd*-s+y;
src[3].x=sd*-s+cd*s+x; src[3].y=-cd*-s+sd*s+y;

if (o) // Виводимо лінію орієнтації
cvLine(img, cvPoint(x, y),
cvPoint(fRound(s*cd + x), fRound(s*sd + y)), cvScalar(0, 255, 0),1);
else // Зелена точка, якщо, користуючись вертикальною версією
cvCircle(img, cvPoint(x,y), 1, cvScalar(0, 255, 0),-1);

// Виводимо квадрат навколо точки
cvLine(img, cvPoint(fRound(src[0].x), fRound(src[0].y)),
cvPoint(fRound(src[1].x), fRound(src[1].y)), cvScalar(255, 0, 0),2);
cvLine(img, cvPoint(fRound(src[1].x), fRound(src[1].y)),
cvPoint(fRound(src[2].x), fRound(src[2].y)), cvScalar(255, 0, 0),2);
cvLine(img, cvPoint(fRound(src[2].x), fRound(src[2].y)),
cvPoint(fRound(src[3].x), fRound(src[3].y)), cvScalar(255, 0, 0),2);
cvLine(img, cvPoint(fRound(src[3].x), fRound(src[3].y)),
cvPoint(fRound(src[0].x), fRound(src[0].y)), cvScalar(255, 0, 0),2);
}
}

//-------------------------------------------------------

// Виводимо фігуру FPS у зображенні (вимагає щонайменше 2 виклики)
void drawFPS(IplImage *img)
{
static int counter = 0;
static clock_t t;
static float fps;
char fps_text[20];
CvFont font;
cvInitFont(&font,CV_FONT_HERSHEY_SIMPLEX|CV_FONT_ITALIC, 1.0,1.0,0,2);

// додаємо fps зображення (кожні 10 фреймів)
if (counter > 10)
{
fps = (10.0f/(clock()-t) * CLOCKS_PER_SEC);
t=clock();
counter = 0;
}

// Інкрементуємо лічильник
++counter;

// Беремо зображення з рядка
sprintf(fps_text,"FPS: %.2f",fps);
```

```
// Виводимо рядок на зображенні
cvPutText (img,fps_text,cvPoint(10,25), &font, cvScalar(255,255,0));
}

//--------------------------------------------------

// Зберігаємо SURF зображення до файлу
void saveSurf(char *filename, vector<Ipoint> &ipts)
{
ofstream outfile(filename);

// виводимо довжину дескриптора
outfile << "64\n";
outfile << ipts.size() << "\n";

// створюємо лінію виведення: координати x y
for(unsigned int i=0; i < ipts.size(); i++)
{
outfile << ipts.at(i).scale << " ";
outfile << ipts.at(i).x << " ";
outfile << ipts.at(i).y << " ";
outfile << ipts.at(i).orientation << " ";
outfile << ipts.at(i).laplacian << " ";
outfile << ipts.at(i).scale << " ";
for(int j=0; j<64; j++)
outfile << ipts.at(i).descriptor[j] << " ";

outfile << "\n";
}

outfile.close();
}

//--------------------------------------------------

// Завантажуємо SURF зображення з файлу
void loadSurf(char *filename, vector<Ipoint> &ipts)
{
int descriptorLength, count;
ifstream infile(filename);

// очищуємо ipts перший вектор
ipts.clear();

// читаємо дескриптор довжини/числа ipoints
infile >> descriptorLength;
infile >> count;

// для кожної ipoint
for (int i = 0; i < count; i++)
{
Ipoint ipt;

// читаємо значення
infile >> ipt.scale;
infile >> ipt.x;
infile >> ipt.y;
infile >> ipt.orientation;
infile >> ipt.laplacian;
infile >> ipt.scale;

// читаємо дескриптор компонент
for (int j = 0; j < 64; j++)
infile >> ipt.descriptor[j];

ipts.push_back(ipt);

}
}
```

utils.h – файл заголовків

```
#ifndef UTILS_H
#define UTILS_H
#include <cv.h>
#include "ipoint.h"
#include <vector>


// Виводимо повідомлення про помилку, та завершуємо програму
void error(const char *msg);

// Показуємо зображення й чекаємо натискання клавіши
void showImage(const IplImage *img);

// Показуємо зображення у головному вікні й чекаємо натискання клавіши
void showImage(char *title,const IplImage *img);

// Конвертуємо зображення по одному каналу32F
IplImage* getGray(const IplImage *img);

// Виводимо єдину особливість на зображенні
void drawIpoint(IplImage *img, Ipoint &ipt, int tailSize = 0);

// Виводимо всі Ipoints у забезпеченому векторі
void drawIpoints(IplImage *img, std::vector<Ipoint> &ipts, int tailSize = 0);

// Виводимо дескрипторні вікна навкругиIpoints у забезпеченому векторі
void drawWindows(IplImage *img, std::vector<Ipoint> &ipts);

// Виводимо фігуру FPS on the image (вимагає щонайменше 2 викликів)
void drawFPS(IplImage *img);

// Виводимо точку в позиції на зображенні
void drawPoint(IplImage *img, Ipoint &ipt);

// Виводимо точку для всіх зображень
void drawPoints(IplImage *img, std::vector<Ipoint> &ipts);

// Зберігаємо SURF зображення до файлу
void saveSurf(char *filename, std::vector<Ipoint> &ipts);

// Завантажуємо SURF зображення з файлу
void loadSurf(char *filename, std::vector<Ipoint> &ipts);

// Округляємо дробове число до найближчого цілого числа
inline int fRound(float flt)
{
return (int) floor(flt+0.5f);
}

#endif
```