Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методу і програмного засобу класифікації зображень на основі нейронних мереж»

Виконав: студент II курсу, групи 2ПІ-20м

спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Грабар С. А.
(прізвище та ініціали)

Керівник: д.т.н., професор каф. ПЗ

Ліщинська Л.Б.
(прізвище та ініціали)

«_____»_____2021 р.

Опонент: к.т.н., доцент кафедри КН

Арсенюк І. Р.
(прізвище та ініціали)

«_____»_____2021 р.

Допущено до захисту

Завідувач кафедри ПЗ

_____д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)

«_____»_____2021 р.

ВНТУ - 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 - Інформаційні технології
Спеціальність 121 - Інженерія програмного забезпечення
Освітньо-професійна програма - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____Романюк О. Н.
« 13 » вересня 2021 р.

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

Грабару Святославу Андрійовичу

1. Тема роботи - Розробка методу і програмного засобу класифікації зображень на основі нейронних мереж

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., проф. каф. ПЗ, затверджені наказом вищого навчального закладу від «24» вересня 2021 р. № 277
2. Строк подання студентом роботи __1 грудня 2021 р.
3. Вихідні дані до роботи: формат вхідних зображень - jpg, кількість шарів - 19, кількість вихідних нейронів – 1154, обсяг навчальної вибірки – від 1150, обсяг тестової вибірки – від 63, середовище програмування – об’єктно-орієнтоване.
4. Зміст розрахунково-пояснювальної записки: вступ, постановка задачі, аналіз предметної області класифікації зображень, обґрунтування вибору типу нейронної мережі для розв’язання задачі класифікації зображень, проектування основного компонента програми класифікації зображень, програмна реалізація модуля для класифікації зображень, аналіз результатів тестування програми класифікації зображень, висновки, перелік використаних джерел, додатки.
5. Перелік ілюстративного матеріалу: схема алгоритму роботи програми, структура згорткової нейронної мережі, результати роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ліщинська Л.Б., д.т.н., проф., каф. ПЗ		
5	Бурєннікова Н. В., д.е.н., проф. кафедри ЕПВМ		

7. Дата видачі завдання 14 вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів магістерського проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Аналіз предметної галузі класифікації зображень	15.09.2021 – 31.10.2021	Вик.
	Методи розв'язання задачі, проектування програмного модуля класифікації зображень	31.10.2021 – 07.11.2021	Вик.
	Програмна реалізація модуля класифікації зображень	07.11.2021 – 10.11.2021	Вик.
	Аналіз результатів тестування модуля класифікації зображень	10.11.2021 – 14.11.2021	Вик.
	Економічна частина	14.11.2021 – 30.11.2021	Вик.

Студент _____ **Грабар С. А.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Ліщинська Л. Б.**

АНОТАЦІЯ

УДК 621.374.415

Грабар С. А. Розробка методу і програмного засобу класифікації зображень на основі нейронних мереж. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2021. 113 с.

На укр. мові. Бібліогр.: 30 назв; рис.: 32; табл.: 7.

В даній кваліфікаційній роботі проводиться розробка програмного методу та програми для класифікації та розпізнавання зображень на основі побудування нейронних мереж. Базуючись на здійсненому аналізі предметної області класифікація зображень і порівняльного аналізу архітектур нейронних мереж обґрунтовано вибір архітектури згорткових мереж, як основи класифікації зображень.

Проведене тестування нейронної мережі підтвердило ефективність її використання для класифікації зображень.

Графічна частина складається з 12 плакатів з презентацією МКР.

В економічній частині визначено та представлено комерційний потенціал до розробки програми, а також витрати для того щоб виконати науково-дослідну роботу і конструкторсько-технологічну роботу, ефекти комерції причетні до реалізації усіх результатів розробки додатку, рівень важливості та ефективності отриманих інвестицій і період часу за який вони окупляться.

Ключові слова: класифікація зображень, розпізнавання зображень, згорткова нейронна мережа.

ABSTRACT

Hrabar S. A. Development of the method and software for the classification of images on the basis of neural network. Master's thesis in specialty 121 – software engineering. Vinnitsa: VNTU, 2021. 113 c.

In Ukrainian language. Bibliographer: 30 titles; fig.: 32; tabl.: 7.

In this qualification work, the development of a software method and program for the classification and recognition of images based on the construction of neural networks. Based on the analysis of the subject area image classification and comparative analysis of neural network architectures, the choice of convolutional network architecture as the basis of image classification is justified.

Tests of the neural network confirmed the effectiveness of its use for image classification.

The graphic part consists of 12 posters with a presentation of the MCR.

In the economic part, the commercial potential for program development is identified and presented, as well as the costs of research and development work, the effects of commerce are involved in the implementation of all application results, the level of importance and efficiency of investments and time period. they will pay off.

Keywords: image classification, image recognition, convolutional neural network.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. З часів коли комп’ютерний світ та мобільні технології почали набирати великих обертів у своїх функціональних можливостях та почали вирішувати одразу багато потреб людства. Серед доступних та широко використовуваних нині функцій таких як комунікації та прості обчислення тепер опинилися і складні системи, які вирішують питання та приймають рішення, подібно людині, які ми називаємо електричним розумом або штучним інтелектом.

До штучного інтелекту входить багато напрямків, серед яких є напрям, який полягає у імітування мозку людини, яке виконується серією алгоритмів, які намагаються розпізнати головні зв’язки у струкрурі сукупності даних, цей напрямок називається штучними нейронними мережами. До цього напрямку належать такі сфери застосування: прийняття інтелектуальних рішень, аналіз даних, обробка тексту та звуку, обробка текстової і звукової інформації, розпізнавання та класифікації зображень, здійснення прогнозів, оптимізація.

Машинне навчання, розпізнавання та класифікація зображень візуальні здібності комп’ютерного зору сьогодні мають всесвітньо глобальну популяризацію у світі і вони здається тільки набирають обертів. Швидкі темпи розвитку широкого аспекту технологій систем штучного інтелекту концепцій «розумного будинку» і «розумного міста», включаючи інтелект самокерованих автомобілів, оцінка глибини водяних водоймів, медичну візуалізацію, візуальний пошук розвиток інтернет-речей та інших систем ШІ означають що такі напрямкамки мають особливу роль у світі та високе місце у пріоритетах для наукового зростання.

Існує багато методів та запропоновано багато алгоритмів рішення задачі класифікації зображень, проте усі ці ідеї поступаються у точності результату, простоті та швидкодії штучним нейронним мережам. В основі сучасних глибоких нейронних мереж, як правило, лежать архітектури мереж

згорткового типу, таких як когнітрон і неокогнітрон. Їх ефективність і стрімкий розвиток обумовлено гібридним підходом до архітектурних рішень, розвитком методів навчання, додаткових методів захисту від перенавчання. Внаслідок зростаючої популярності глибоких згорткових мереж досягаються суттєві успіхи у розпізнаванні об’єктів.

Актуальність розробки обумовлена застосуванням технології класифікації зображень у пошукових системах мережі Інтернет, у системах штучного інтелекту та Data Mining, що працюють із зображеннями та їх контентом.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення точності і швидкості класифікації зображень за рахунок використання згорткової нейронної мережі.

Основними задачами дослідження є:

- проаналізувати відомі методи класифікації зображень та обрати напрямок досліджень;
- проаналізувати математичну класифікації зображень;
- розробити структуру програмного модуля;
- розробити алгоритм роботи програмного модуля класифікації зображень,
- розробити програмне забезпечення класифікації зображень за допомогою нейронної мережі;
- здійснити програмну реалізацію модуля класифікації зображень,
- провести тестування програмного модуля класифікації зображень.

Об’єкт дослідження – процес класифікації зображень.

Предмет дослідження – методи і засоби класифікації зображень на основі нейронних мереж.

Методи дослідження. В роботі використано теорію обробка зображень, теорію нейронних мереж для класифікації зображень, теорія алгоритмів і програмного забезпечення для розробки програм; комп'ютерне моделювання для перевірки отриманих результатів.

Наукова новизна отриманих результатів.

Подальшого розвитку отримав метод класифікація зображень, який відрізняється від відомих використання згорткової нейронної мережі, що дозволило підвищити точність і швидкість класифікації.

Практична цінність отриманих результатів полягає у тому, що на основі отриманих у магістерській кваліфікаційній роботі теоретичних положень запропоновано програмний засіб, який дозволяє ефективно розпізнавати об'єкти і класифікувати їх за класами.

Апробація результатів. Основні положення магістерської кваліфікаційної роботи обговорювалися на Всеукраїнській науково-практичній Інтернет-конференції "Електронні інформаційні ресурси: створення, використання, доступ" 8-9 листопада 2021 року.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У праці, що опублікована у співавторстві, автору належать: аналіз технологій та галузей застосування згорткових нейронних мереж [1].

Публікації. За результатами досліджень опубліковано тези у збірнику матеріалів конференції [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

1.1 Порівняльний аналіз відомих методів класифікації зображень

При вирішенні класичного завдання класифікації зображень заведено використовувати математичну мову, а у взаємодії з нею логічні міркування принципи математичних операцій. В протилежність відносно до математично-логічного підходу, існують ще й такі методи для класифікації зображень які працюють із використанням Deep Learning і штучних нейронних мереж, створені такою формою, що настільки добре підходять до задачі класифікації і так добре справляються з нею, що демонструють оперативність та результати класифікації на одному математично-логічним, а інколи показують себе набагато краще.

Основні методи класифікації:

• класифікація за допомогою дерев рішень;

• байєсівська (наївна) класифікація;

• класифікація методом опорних векторів;

• статистичні методи;

• класифікація за допомогою методу найближчого сусіда;

• класифікація CBR-методом (case based-reasoning);

• класифікація за допомогою нейронних мереж;

Розглянемо загальні елементи моделі класифікації.

Клас - є розширюваний програмно-коду шаблон для створення об'єктів , що забезпечує початкові значення для стану (змінних - членів) і реалізацій поведінки (функція або членів методи). Створення нового класу створює новий тип об’єкта, що дозволяє створювати нові екземпляри цього типу. Кожен екземпляр класу може мати атрибути, додані до нього для підтримки його стану. Примірники класу також можуть мати методи (визначені його класом) для зміни його стану [1].

Класифікація - процес, у якому об’єкти розуміються та об’єднуються в класи або процес присвоєння мітки класу невідомим точкам даних на основі вивчених фактів.

Класифікатор — це тип алгоритму машинного навчання, який використовується для призначення мітки класу введеним даним. Прикладом є класифікатор розпізнавання зображень для позначення зображення (наприклад, «автомобіль», «вантажівка» або «людина»). Алгоритми класифікатора навчаються за допомогою мічених даних. Класифікатор може бути безпосередньо побудований з набору прикладів шаблонів з відповідними класами, або побічно з статистичної моделі [1].

Верифікація – це процес перевірки того, що програмне забезпечення досягає своєї мети без будь-яких помилок [1]. Це процес, щоб переконатися, що розроблений продукт правильний чи ні. Він перевіряє, чи відповідає розроблений продукт вимогам, які ми висуваємо. Перевірка - це статичне тестування.

Верифікація — це процес перевірки відповідності програмного продукту значенню або, іншими словами, до продукту вимоги високого рівня. Це процес перевірки валідації продукту, тобто він перевіряє те, що ми розробляємо, є правильним продуктом. це перевірка фактичного та очікуваного продукту. Перевірка - це динамічне тестування.

У розпізнаванні образів простір ознак — це абстрактний простір, де кожен зразок шаблону представлений як точка в n-вимірному просторі . Його розмірність визначається кількістю ознак s, які використовуються для опису шаблонів. Подібні зразки групуються разом, що дозволяє використовувати оцінку щільності для пошуку закономірностей.

Враховуючи деякі дані, особливість простору - це просто набір усіх можливих значень для вибраного набору ознак із цих даних. Завжди можна представити значення ознак і, таким чином, простір ознак, використовуючи

лише числа, і далі зробити це таким чином, щоб простір ознак можна було інтерпретувати як реальний простір [2].

Іншими словами, класифікацію об’єктів можна визначити, як віднесення вихідних даних до певного класу за допомогою виділення істотних ознак або властивостей, які характеризують ці дані, із загальної маси несуттєвих деталей.

Прикладами задач класифікації класифікувати однозначно можуть бути такі: розпізнавання символів, установлення стану здоров’я або діагнозу, а також прогнозування погоди, упізнавання осіб за обличчям чи рисами, розпізнавання об’єктів у відеопотоці тощо.

Найчастіше вихідним матеріалом служить отримане із камери зображення. Якщо розглянути 2 класи об'єктів: дорослі і діти. В якості значення ознак можна вибрати зріст і вагу (рисунок 1.1).

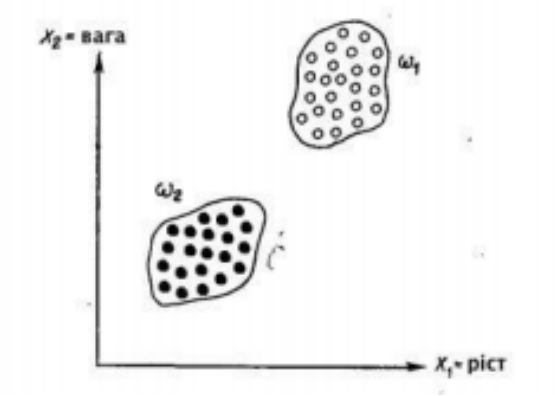


Рисунок 1.1 – Два класи, що не перетинаються

З рисунку 1.1 видно, що ці два класи утворюють дві множини, що не перетинаються, що можна пояснити обраними ознаками. Проте, не завжди є можливість вибрати правильні вимірювані параметри в якості ознак класів. Наприклад, вибрані параметри не підійдуть, щоб створити непересічні класи хокеїстів та волейболістів. Іншим завданням розпізнавання є виділення із вихідних зображень характерних ознак та/або властивостей. Це завдання можна віднести до попередньої обробки. Ознака повинна являти собою

характерну властивість конкретного класу, при цьому загальну для цього класу.

Міжкласові ознаки – це ознаки, що визначають відмінності між класами. Загальні ознаки, що властиві усім класам, не несуть корисної інформації, тому для задачі класифікації об’єктів не розглядаються як характерні. Вибрати правильні ознаки - одна із важливих задач побудови систем класифікації.

Класичні методи комп'ютерного зору класифікації зображень можна умовно розділити на три групи: методи фільтрації, методи логічного аналізу, методи навчання.

У завданнях класифікації зображень фільтрація найчастіше використовується для попередньої обробки зображення перед аналізом його внутрішніх морфологічних ознак, але зустрічаються і завдання, в яких достатнім і бажаним буде використання тільки фільтрації (наприклад, в задачах машинного зору).

Візьмемо до розгляду можливих параметрів структури та загальних елементів які можуть бути частиною моделі класифікації.

Бінаризація зображення полягає в тому, щоб встановити значення сірого для пікселів на зображенні, що дорівнює 0 або 255 (RBG зображення), що означає, що все зображення має очевидні візуальні ефекти тільки чорного та білого. Значення сірого 0: чорний; значення сірого 255: білий. Зображення включає цільовий об'єкт, фон і шум. Для безпосереднього вилучення цільового об'єкта з багатозначного цифрового зображення звичайним способом є алгоритм вибору порогу. Алгоритм полягає у встановленні порога і використання його для поділу даних зображення на дві частини. Це найбільш особливий метод вивчення перетворення на градаціях сірого [2].

Перетворення Фур'є майже не використовується при обробці зображень сам по собі окрім як буває включенням до якогось іншого методу, тому що для тощо щоб аналізувати зображення одного одновимірного

перетворення буває мало зазвичай і не вистачає, через що й виникає обов’язкова потреба у використанні набагато результативнішого перетворення у двох вимірах. Такий метод застосовується лише тільки в разі необхідності у більшому аналізі спектра, оскільки зацікавленістю у згортках є області з уже саме з готовим фільтром. Однак, одновимірне перетворення Фур'є часто застосовується для стиснення зображення.

Найпростішим із видів фільтрів низьких частот є фільтр Гаусса. При обробці 1d сигналів використовується багато типів фільтрів низьких частот. Однак фільтри Гаусса майже ніколи не використовуються. Чому вони такі популярні в програмах обробки зображень? Чи є ці фільтри результатом оптимізації будь-якого критерію чи є лише спеціальним рішенням, оскільки «пропускна здатність» зображення зазвичай не чітко визначена.

Найпростішим із видів фільтрів високих частот є фільтр Габора. Перетворення Габора є одним із багатьох так званих смугових фільтрів, що дозволяє «вирізати» перетворення Фур’є та ізолювати лише конкретну інформацію. Інша важлива інформація полягає в тому, що кожен «піксель» Фур’є є комплексним значенням (дійсною та уявною частиною). При перетворенні Фур’є зображення розглядається як утворене шляхом накладання серії синусоїдальних хвиль різної частоти, орієнтованих у всіх напрямках. Кожен «піксель» у перетворенні говорить нам про «інтенсивність» такої хвилі. Положення «пікселя» говорить нам про частоту та орієнтацію хвилі.

У фільтрах Гаусса та Габора для кожної точки на зображенні обирається вікно, в межах якого виконується перемножування вихідних фільтрованих даних того ж самого розміру (згортка). Цей підхід отримав достатньо широке поширення у застосуванні, дозволяючи знаходити на кожному зображенні важливу інформацію, відсікаючи зайву інформацію. Також, підхід став розповсюджено використовуватися як одна з сукупності реалізацій для швидкого шумозаглушення в різних сферах техніки і науки [3].

Wavelet-перетворення. Для згортування з сигналом як областю зображення застосовується така функція значень, названа wavelet або вейвлетом, що має визначення як wavelet-перетворення. Вейвлети — це короткі коливальні сигнали, які починаються з нульового значення і повертаються до нульового значення протягом кінцевої тривалості. Ця характеристика в поєднанні з «масштабуванням» і «зсувом» робить їх дуже корисними для обробки сигналів. Масштабування вейвлета – його розтягування або стиснення в часі – може дозволити йому відображати або довготривалий рух сигналу, або короткочасну флуктуацію відповідно. Зміщення вейвлетів – вирівнювання їх центрів у різних точках щодо вихідного сигналу – може дозволити їм представляти різні події в різних точках цього сигналу. Таким чином, можна апроксимувати сигнал як суму зміщених і масштабованих вейвлетів за допомогою «вейвлет-перетворення». Це схоже на інтуїцію за перетворенням Фур'є, яке представляє сигнал у вигляді суми нескінченних синусоїдних хвиль, але вейвлет-перетворення може більш ефективно працювати з сигналами, які мають різкі зміни або розриви. Вейвлет-перетворення корисні в різних програмах, включаючи обробку зображень, шумозаглушення сигналу та стиснення (наприклад, кілька) [3].

Метод обчислення кореляції, що лежить в основі вейвлет перетворення.

Метод вейвлет-когерентності для двох просторових рядів є досить корисним для аналізу локальної лінійної кореляції в заданому місці та просторовому масштабі, а також для визначення того, чи пов'язана поява частоти в послідовності X з послідовністю Y в тому ж положенні. Цей аналіз подібний до традиційного методу коефіцієнта кореляції, в той час як когерентність вейвлетів на кількох масштабах є більш надійною порівняно з одиничним значенням, отриманим за шкалою вибірки Цей метод є незамінно важливим інструментом що використовується системах комп'ютерного бачення і дуже часто використовується на практиці в своєму природному

вигляді, от як наприклад, для знаходження зрушень а також знаходження оптичних потоків що що називається кореляцією відеопотоку. На основі обчисленої корелятором різниці реалізується елементарний детектор зсуву.[3].

Розглянемо питання фільтрації зображень. Спочатку наведено загальні відомості, а потім розглянуто методи лінійної та нелінійної просторової фільтрації. Серед методів лінійної просторової фільтрації зазначено методи низько-частотної простової фільтрації, які дозволяють сгладжувати адитивний шум на зображенні, та методи виділення границь зображень об’єктів, які побудовані на основі псевдодиференціальних операторів. Серед методів нелінійної просторової фільтрації зазначено методи нелінійної усереднювальної фільтрації, методи фільтрації на основі порядкових статистик та методи медіанної та комбінованої медіанної фільтрації. [3].

Фільтрація дозволяє отримати набір даних, який може бути здатним для обробки, але галузі комп'ютерного зору більше вимагають саме аналіз структури зображення за морфологічними ознаками зображених об'єктів, для цього досліджуються і впроваджуються методи логічного аналізу зображень. Методи математичної морфології є результатом теорії та техніки аналізу й обробки геометричних структур, заснованих на теорії множин, топології та випадкових функціях [3].

Області застосування фільтрації:

- Покращення зображення, наприклад, знебарвлення, зміна розміру
- Вилучення інформації, наприклад, текстури, країв
- Виявляє закономірності, наприклад, підбір шаблону.

Контурний аналіз дозволяє обробляти зображення в прогресивному режимі. Це означає, що ми можемо відсортувати контури за якоюсь ознакою (наприклад, по площі або по градієнту кордонів, або по яскравості і т.п.). А потім обробити перший контур, і видати результат. Інші ж контури обробляти у фоновому режимі. Це означає що перший результат (а в

багатьох прикладних задачах саме він і потрібний) може бути отриманий за O(n), що є відмінною оцінкою для алгоритмів розпізнавання зображень.

Порівняно основні властивості нелінійних методів контурного аналізу за вимогами у використанні обмеженості обчислювальних ресурсів апаратних засобів. Оператор Кірша є найбільш ресурсно-витратним методом серед розглянутих, але при цьому його перевага в дуже чутливій масці, що виділяє цей метод для використання навіть за найнижчій загальній яскравості зображення. Оптимальним методом щодо обчислювальної потужності та знаходження контурів є оператор Собеля, це зумовлено використанням маски з коефіцієнтами тільки для середніх значень. Окремо можна виділити оператор Лапласа. Цей метод виконано швидше і має він меншу обчислювальну вартість. При цьому дає не набагато гірший за інші методи результат. Цей метод добре використовувати, якщо обчислювальні потужності є невисокими [4].

Роботизовані системи зору можуть розрізняти частини за зразком, незалежно від орієнтації та розташування деталей. Деякі виробники пропонують тривимірні системи наведення, що використовують надійні системи зору та лазерні системи, щоб тривимірна запрограмована точка могла повторюватися, навіть якщо деталь переміщується, змінюючи її розташування, обертання та орієнтацію в робочому просторі. Незважаючи на цей розвиток подій, нинішні промислові роботи все ще не можуть надійно розпізнавати об'єкти; тобто виділити об'єкт серед об'єктів однакової форми, враховуючи не лише контур об'єкта, а й інформацію про його форму та глибину. У подібних випадках методи контурного аналізу вважаються неперевершеними у своїх сферах лідерами по швидкодії, вони вражають своїми можливостями, володіють зрозумілою і простою логікою, що обумовлює зручність їх використання в спеціалізованих областях.

Зіставлення зображень на основі ознак — це метод вирішення проблеми відповідності між двома або більше зображеннями. Сполучені точки є вимогою для оцінки параметрів орієнтації зображення, що є

передумовою для всіх геометричних застосувань у фотограмметрії, робототехніці та комп’ютерному бачення, що включає тривиміри [4]. 3D-реконструкція з кількох зображень, одночасна локалізація та відображення (SLAM), структура з руху (SfM) і створення мозаїки зображень – усі вони покладаються на координати зображення узгоджених сполучених об’єктів. Тому якість алгоритмів узгодження є життєво важливою для стабільності та якості вирішення цих проблем.

Для математичного опису та його формування використовуються дескриптори. Завдання дескрипторів є одночасний пошук feature points так званих особливих точок. Вилучення ознак включає обчислення дескриптора, яке зазвичай виконується в областях, зосереджених навколо виявлених об’єктів. Дескриптори покладаються на обробку зображень для перетворення локального піксельного околиці в компактне векторне представлення. Це нове представлення дозволяє порівнювати околиці незалежно від змін масштабу чи орієнтації. Дескриптори, такі як SIFT або SURF, покладаються на обчислення локального градієнта. Бінарні дескриптори, такі як BRISK, ORB або FREAK, спираються на пари локальних відмінностей інтенсивності, які потім кодуються у двійковий вектор.

Особливі точки це унікальні характеристики об'єкта, які дозволяють зіставляти об'єкт сам з собою або зі схожими класами об'єктів. Існує кілька десятків способів дозволяють виділити такі точки. Деякі способи виділяють особливі точки в сусідніх кадрах, деякі через великий проміжок часу і при зміні освітлення, деякі дозволяють знайти особливі точки, які залишаються такими навіть при поворотах об'єкта.

Особливі точки також можна розглядати у голоморфних функцій, визначених на рімановіх поверхнях. Зокрема, якщо змінна z пробігає сферу Рімана, то особливість на нескінченності функції {\displaystyle \ f(x)}OBJ визначається за степенем «особливості» точки 0 для функції

Проблематичність послідовності пошуку роботи росте зі стабілізацією шуканих точок

Підсумком роботи таких алгоритмів є велика кількість особливих точок, особисто кутів, для яких слід збудувати математичний опис.

Штучні нейронні мережі зазвичай складаються з взаємопов'язаних одиниць, які служать моделлю нейронів. Синапс моделюється за допомогою змінюваної ваги, пов'язаної з кожним конкретним з'єднанням. Більшість штучних мереж не відображають детальну геометрію дендритів і аксонів, і вони виражають електричний вихід нейрона як єдине число, яке представляє швидкість спрацьовування. Кожен блок перетворює шаблон вхідних дій, які він отримує, в одну вихідну дію, яку він надсилає іншим підрозділам. Це перетворення виконується в два кроки:

По-перше, він множить кожну вхідну дію на вагу з'єднання та додає всі ці зважені вхідні дані, щоб отримати загальний вхід. Це зображено на рисунку 1.2.

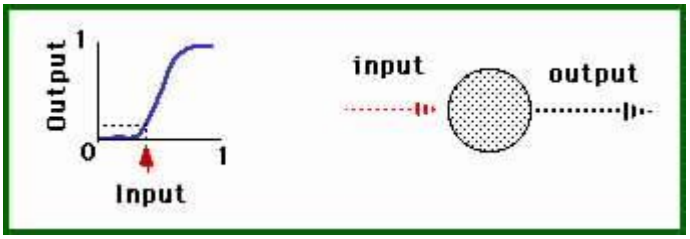


Рисунок 1.2 - Дія на вагу для отримання загального входу

По-друге, пристрій використовує функцію введення вихідних даних, яка перетворює загальний вхід у вихідну діяльність. Це зображено на рисунку 1.3.

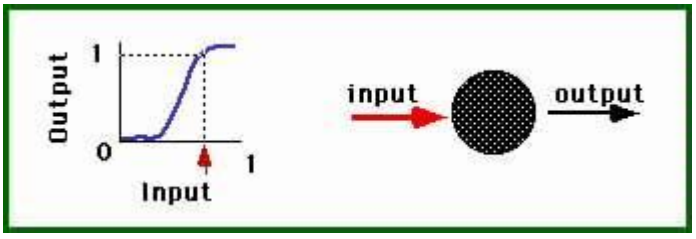


Рисунок 1.3 - функція введення вхідних даних

Глобальна поведінка штучної нейронної мережі залежить як від ваги, так і від функції введення-виводу, яка вказана для одиниці. Ця функція зазвичай відноситься до однієї з трьох категорій: лінійна, порогова або

сигмовидна. Для лінійних одиниць вихідна активність пропорційна загальному зваженому входу. Для порогових одиниць вихід встановлюється на одному з двох рівнів, залежно від того, чи є загальний вхід більшим або меншим за деяке порогове значення. У сигмовидних одиницях вихід змінюється безперервно, але не лінійно, коли змінюється вхідний сигнал. Сигмовидні одиниці мають більшу схожість з реальними нейронами, ніж лінійні або порогові одиниці.

Щоб нейронна мережа виконувала конкретне завдання, необхідно вказати зв’язок між блоками та правильно встановити ваги зв’язків. Зв'язки визначають, чи може один блок впливати на інший. Ваги визначають силу впливу.

Один поширений тип штучної нейронної мережі складається з трьох груп або шарів одиниць. Вхідний шар з'єднаний з шаром проміжних одиниць (званих прихованими одиницями), який, у свою чергу, з'єднаний з шаром вихідних одиниць. Діяльність вхідних блоків представляє собою вхідну зовнішню інформацію, яка подається в мережу. Діяльність кожного прихованого блоку визначається діяльністю вхідних блоків і ваговими показниками зв’язків між вхідними та прихованими блоками. Поведінка вихідних одиниць, у свою чергу, залежить від активності прихованих одиниць і ваг між прихованими та вихідними одиницями.

Варто звернути увагу на те, що приховані одиниці можуть вільно створювати власне представлення вхідних даних. Вагові коефіцієнти між вхідними та прихованими одиницями визначають, коли кожна прихована одиниця активна. Таким чином, змінюючи ці ваги, прихована одиниця може вибрати, що вона представляє.

Тришарову мережу можна навчити наступним чином: спочатку мережа представляється навчальним прикладом, що складається з шаблону дій для вхідних блоків разом із шаблоном, який представляє потрібний вихід. Потім визначається, наскільки фактичний вихід відповідає бажаному.

Далі змінюється вага кожного з’єднання, щоб отримати краще наближення до бажаного результату.

Варто зауважити те, що в цій процедурі експериментатор повинен заздалегідь знати бажаний вихід, а потім повинен змусити мережу вести себе відповідним чином. Тому потрібне правило навчання. Правило навчання керує способом зміни кожного зв’язку (ваги), щоб ефективно досягати мети (вихідного шаблону).

1.2 Обгрунтування вибору аналогу до програмного модуля класифікації зображень

Існує багато реалізацій класифікації зображень і дуже важливо знати те, якими вони бувають. Усі вони поділені між собою характеристиками. Перше це те що різняться між собою вони через мову програмування якою вони реалізовані, бо звідси й алгоритмічні структури самих усіх ідей та методів реалізації. Отже, за спостереженнями дослідження виявилося що всього як правило реалізація розпізнавання зображення здійснюється однією з двох мов програмування Python або .Net.

Сьогодні можна запропонувати до уваги декілька відомих програмних реалізацій, які мають здатність здійснювати розпізнавання та класифікацію зображень.

Серед таких програм можна представити BytePace, який є програмним додатком написаним на мові Python. На початку алгоритм було розроблено для того, щоб виявляти постатей на картинках, але його можливість до навчання дозволяє натренувати його на виявлення також й інших об'єктів. В основі його роботи лежить розбиття (splitting) зображення на області, оцінку яскравості в цих областях і відсікання областей, де класифікований об'єкт однозначно не знаходиться. З цього випливає головний недолік системи – її низька точність. Це може призводити до помилок в роботі. [10].

Вікно програми BytePace подане на рисунку 1.4.

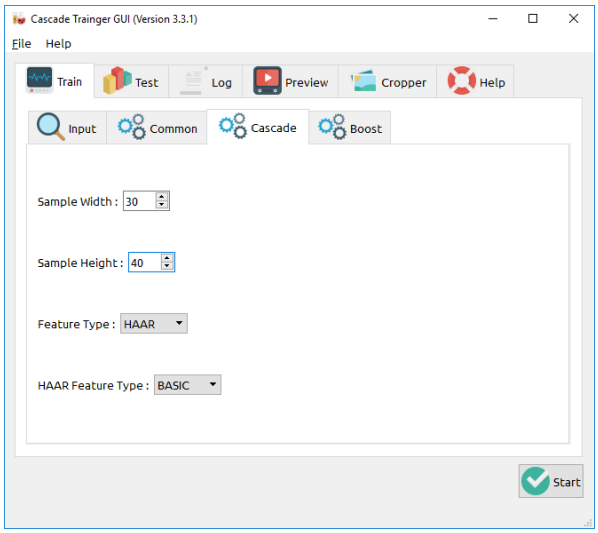


Рисунок 1.4 – Вікно програми BytePace

Іншою реалізацією для класифікації зображень є програма ResNeXt-101. Принцип роботи програми полягає в відкиданні зайвих векторів. Зайві – це ті, які не ввійшли у взаємно знайдений кластер векторів. Наприклад, на зразку може бути тінь, яка розпізнається як межа, а на наступному зразку її може не бути. До недоліків можна віднести складність роботи з програмою та низьку швидкість її роботи [11].

Вікно програми ResNeXt подане на рисунку 1.5. У таблиці 1.1 наведено порівняння програм для класифікації зображень.

Рисунок 1.5 – Вікно програми ResNeXt

Таблиця 1.1 – Порівняння програм для класифікації зображень

Можливість	BytePace	ResNeXt
Виведення пояснень	-	+
Легкість в освоєнні	-	+/-
Швидкість роботи	+	-
Точність	+/-	+

Отже, розглянуті програмні реалізації мають ряд недоліків таких як складність в освоєнні, низька швидкість та точність. З огляду на це, постає питання в розробці нового програмного засобу для класифікації зображень, який був би вільний від вказаних недоліків.

1.3 Постановка задачі

До входу пропонуються дані, які є зображеннями, формат яких jpg. Задача полягає у тому, що потрібно упізнати що зображено на кожній з поданих картинок, кожна з яких повинна мати своє ім’я за змістом як конкретний об’єкт. Такими об’єктами поданими на картинках можуть бути такі як “тигр”, “дельфін”, “водоспад”, “велосипедист” та величезна різноманітність інших. Таку задача тому і називають класифікацією через це, бо вона відповідає визначенню класифікації. Тому що відбувається розподілення екземплярів за їхніми назвами.

Мета роботи полягає у тому щоб оглянути існуючі підходи та алгоритми по вирішенню завдань розпізнавання та класифікації за допомогою штучних нейронних мереж. А також в підвищенні точності та швидкодії класифікації зображень за рахунок використання згорткової нейронної мережі.

У цій роботі планувалося вирішити такі завдання:

- Підготувати набори даних для завдання класифікації;
- Встановити залежність ефективності навчання нейронної мережі що використовує різні зображення;
- Провести порівняння результатів навчання на різних наборах даних;

Постановку завдання можна сформулювати, як одержання вихідних векторів, що залежать від ознак кожного класу на зображенні. Процес можна розглядати як процес кодування, що полягає в присвоєнні значення кожної ознаки із простору ознак для кожного класу.

Класифікація вирішує таку задачу. Визначення кінцевої множини класів і є множиною об'єктів, для кінцевої підмножини яких відомо до якого класу вони відносяться. Ця підмножина називається навчальною вибіркою. Класова приналежність інших об'єктів невідома, потрібно побудувати алгоритм, здатний класифікувати довільний об'єкт з початкової множини [1].

Класифікувати об'єкт – означає, вказати номер (або найменування класу), до якого відноситься даний об'єкт.

Вхідними даними до задачі класифікації є набір зображень, що налічує близько 665000 екземплярів. Екземпляром є кольорове зображення, що містить мітку одного з 1154 класів.

Вихідними даними задачі є клас, зображення з якого було подано на вхід нейронній мережі, і яке вона до того не бачила. Таким чином, буде побудовано програмний модуль, який здатний класифікувати зображення на 1154 класів, що були представлені в навчальній вибірці.

1.4 Висновки

У розділі було розглянуто задачу класифікації зображень за їх змістом. В ході аналізу предметної області розглянуто основні методи класифікації зображень та як найбільш перспективний, було обрано нейромережевий метод. Також було здійснено аналіз відомих програмних засобів класифікації зображень та як аналог до розроблюваного модуля було обрано програму RexNeXt.

2 МЕТОДИ РОЗВ'ЯЗАННЯ ЗАДАЧІ, ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

2.1 Обґрунтування вибору згорткової нейронної мережі для розв'язання задачі класифікації зображень

Штучний нейрон — це математична функція, заснована на моделі біологічних нейронів, де кожен нейрон приймає вхідні дані, зважує їх окремо, підсумовує і передає цю суму через нелінійну функцію для отримання результату.

Перцепетрон - це один із найпростіших типів штучних нейронних мереж прямого поширення

У будь-який момент часу кожен перцептрон має певний рівень активності, який визначається рівнями активності інших перцептронів, що передають йому сигнали зваженими зусиллями відповідної взаємодії перцептронів.

Основна його задача, яка йому вдається, - це лінійне розділення будь-яких нелінійних множин, тобто забезпечення лінійної сепарабельності.

Типова будова для багатошарового перцептрону зображена як схема на рисунку 2.1.

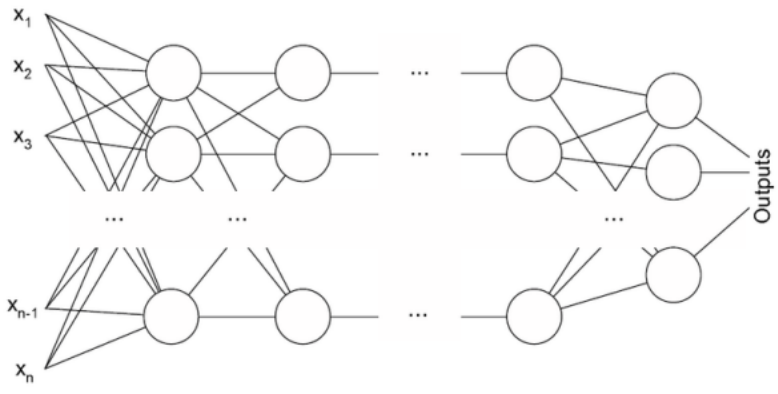


Рисунок 2.1 – Будова багатошарового перцептрону

Завдання які вирішує нейрон:

- Отримує інформацію від інших нейронів у вигляді електричних імпульсів різної сили.

- Нейрон інтегрує всі імпульси, які він отримує від інших нейронів.

- Якщо отримане підсумовування перевищує певне порогове значення, нейрон «спрацьовує», запускаючи потенціал дії, який передається іншим підключеним нейронам.

Що ж щодо перцептрона, який є штучним аналогом нейрона, він складається з трьох частин:

- Вузли введення або вхідний рівень: вхідний рівень приймає вихідні дані в систему для подальшої обробки. Кожен вхідний вузол пов'язаний з числовим значенням. Він може мати будь-яку реальну цінність.

- Вага та упередження: параметри ваги представляють силу зв'язку між одиницями. Чим вища вага, тим сильніше вплив пов’язаного вхідного нейрона, щоб визначити вихід. Зміщення виконується так само, як і перехоплення в лінійному рівнянні.

- Функція активації: функція активації визначає, спрацює нейрон чи ні. У найпростішому випадку функція активації є кроковою функцією, але на основі сценарію можна використовувати різні функції активації.

Згорткові мережі можуть часто включати шари глобального або локального які об'єднують виходи кластерів нейронів одного шару до одного нейрону наступного шару[6]. Наприклад, максимізаційне агрегування (max pooling) використовує максимальне значення з кожного з кластерів нейронів попереднього шару. Вони складаються із різних комбінацій, які включають згорткові та повноз'єднані шари, які проявляють свою активність в кінці кожного нелінійністю поточково. Для їх зниження і покращення узагальнення вводиться операція згортки на малих областях входу. Головна перевага є в тому, що є спільна вага, що означає для кожного пікселя шару, де використовується один той самий фільтр: банк ваги, через який зменшується розмір потрібної пам'яті, і покращує оперативність.

Згортка в основному використовується для вилучення ознак. У традиційній обробці зображень ми вказуємо конкретні ядра згортки для вилучення функцій зображення. Наприклад, оператор Sobe використовується для виділення країв зображення. Але в нейронній мережі ми вказуємо лише розмір ядра згортки та конкретне ядрозгортки (вміст ядра згортки), також можна сказати, що вага ядра згортки отримується шляхом навчання. Наприклад, якщо ви вкажете розмір ядра згортки як [формула](товщина x ширина x висота), ми отримаємо 27 параметрів після навчання.

З усіх архітектур було обрано архітектуру згорткової мережі, отже ж вона має можливість використи спільні ваги на шарах згорток, що впливає позитивно на оперативність та якість.

2.2 Вибір та аналіз моделі класифікації зображень на основі згорткових нейронних мереж

Ми почнемо з LeNet, який був розроблений у 1998 році, він досить добре працював у проблемах розпізнавання рукописних символів. Архітектура LeNet-5 проста, складається з 5 вагових шарів, звідси й назва LeNet-5: 3 згорткові шари + 2 повністю з’єднаних шари.

Новим проривом у розвитку моделей класифікації зображень стала нейронна мережа відома як AlexNet, розроблена у 2012 році. Ця мережа стала першою моделлю конволюційної нейронної мережі, яка досягла коефіцієнта помилок 15,4% на конкурсі ILSVRC 2012 [11]. Розроблена вона була у випалку 2012 року дослідниками Алексо Крижевським та Іллею Суцкевером, які розробили глибоку згорткову нейронну мережу AlexNet в лабораторії Джеффа Хінтона в Університеті Торонто. Вони виграли чемпіонат ImageNet LSVRC 2012 року, а показник точності значно перевищив друге місце (показник помилок топ-5). 15,3%, а другий – 26,2%), що викликало великий фурор. Можна сказати, що AlexNet є історично важливою мережевою структурою. До цього глибоке навчання тривалий час

було мовчазним. З моменту народження AlexNet у 2012 році всі наступні чемпіони ImageNet використовували згорткові нейронні мережі (CNN), щоб зробити так , і рівень стає все глибшим і глибшим, що робить CNN основною моделлю алгоритму в розпізнаванні та класифікації зображень, що принесло великий вибух глибокого навчання.

Модель Alexnet складається з 5 згорткових шарів і 3 шарів об’єднання, включаючи 3 повністю пов’язані шари. AlexNet подібний до LeNet за структурою, але використовує більше згорткових шарів і більший простір параметрів для розміщення великомасштабного набору даних ImageNet. Це лінія розмежування між поверхневою нейронною мережею і глибокою нейронною мережею.

Особливості AlexNet:

1. На задню частину кожного намотувального апарата додано функцію активації Relu, яка вирішує проблему зникнення градієнта сигмоїда і робить зближення швидшим.

2. Використано технологію випадання, щоб вибірково ігнорувати окремі нейрони під час навчання, щоб уникнути перепідгонки моделі (також використовуйте покращення даних, щоб запобігти перепідгонці)

3. Додано нормалізований шар LRN (Local Response Normalization), щоб підвищити точність.

4. Максимальний пул, який перекривається, тобто існує зв'язок між діапазоном об'єднання z і розміром кроку s z>s, щоб уникнути середнього ефекту від середнього об'єднання.

Шари CNN можуть бути чотирьох основних типів: шар згортки, шар ReLu, шар об’єднання та шар повністю підключеного.

Рівень згортки: згортка — це просте застосування фільтра до входу, що призводить до активації. Рівень згортки має набір фільтрів, які можна навчати, які мають невеликий діапазон сприйняття, але можуть бути використані для повного об’єму наданих даних. Шари згортки є основними

будівельними блоками, які використовуються в згорткових нейронних мережах.

- Рівень ReLu: шари ReLu, також відомі як випрямлені лінійні одиничні шари, є функціями активації, що застосовуються для зниження переобладнання та підвищення точності та ефективності CNN. Моделі, які мають ці шари, легше тренувати і дають точніші результати.

- Рівень об’єднання: цей шар збирає результати всіх нейронів у шарі, що передує йому, і обробляє ці дані. Основне завдання рівня об’єднання полягає в тому, щоб зменшити кількість факторів, які враховуються, і забезпечити оптимізований результат.

- Повністю підключений шар: цей шар є кінцевим вихідним шаром для моделей CNN, який вирівнює вхідні дані, отримані від шарів перед ним, і дає результат.

AlexNet модель зображена на рис 2.2 являє собою 8-шарову мережу з яких від початку перші п’ять з яких згорткові шари а інші три - повнозв’язні, та кожен із восьми із них з вагами. Останній восьмий шар містить в собі 1000 виходів, і саме цей шар визначає мітку класу. В кожному згортковому і повнозв'язному шарі застосовується функція активації *ReLU* [12]. Для звернених другого, четвертого і п'ятого шарів фільтри об’єднані з картками ознак тільки первинного шару. Забезпечено з’єднання фільтра третього шару згортки з кожною з утворених карт властивостей другого шару. Усі нейрони повністю звернених шарів поєднані з нейронами кожного попереднього шару. Поєднуючись між собою першим і другий шар згортки є передніми перед шарами нормалізації, а шари нормалізації є передніми для шарів субдискретизації а також для п'ятого шару згортки.

У прикладі поданих даних представлено зображення розміром 256x256, яке радномно розрізають на 224 x 224 та горизонтально провертають, тому тут дані збільшені, а структура згину тут вводиться для 3 x 224 x 224, у той час як під час розрахунку необхідно включати відступ.

Деякі скрипти на сайтах мережі Інтернет розглядають саме таку структуру як 3 x 227 x 227, тому не виникає необхідності в заповненні.

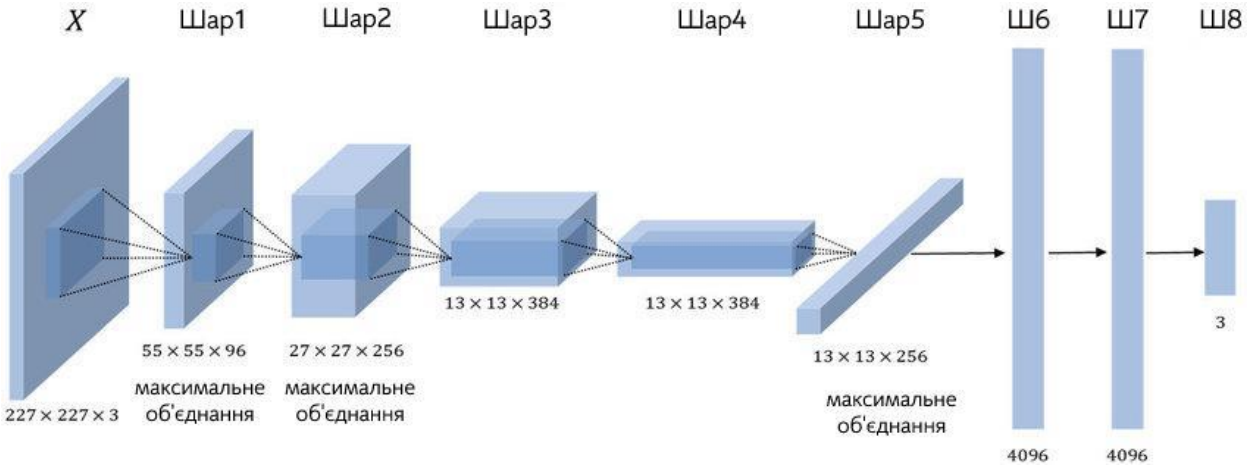


Рисунок 2.2 - Архітектура мережі AlexNet

У мережу AlexNet включено 60 млн різних параметрів, у той час як вартість розрахунку тоді була досить високою. Через те що ємність відеокарти обмежена, дані параметри для навчання не виходить вмістити на одній відеокарті, отож щоб навчити мережу використано два GPU.

Збільшення даних часто використовується як метод часто з метою зменшити перенавчання. Такий метод полягає в повторювальному перетворенні навчальних зображень для того щоб збільшити кількість вхідних зображень. У парі з цим застосовується дзеркальне відображення самих зображень по горизонталі і варіювання частоти кольору пікселів.

Встановлення нульовоно значення кожного із нейронів у схованому шарі з імовірністю в 0,5. При застосуванні дропаута нейрони, до яких застосовано дропаут, не приймають участь у поширенні помилки з кінця.

Якщо говорити про AlexNet трансферне навчання, то можна виділити те, що зазвичай таке навчання використовується програмами глибокого навчання. На практиці ви можете взяти заздалегідь підготовлену мережу і використовувати її як відправну точку для вивчення нового завдання. Тонка настройка мережі за допомогою навчання передачі зазвичай набагато швидша і легша, ніж навчання мережі з випадково ініціалізованими

ваговими показниками з нуля. Ви можете швидко перенести вивчені функції до нового завдання, використовуючи меншу кількість навчальних зображень. Для прикладу можна використовувати набір даних зображення, який містить 4 мітки: коло, прямокутник, трикутник і зірка, або інші цього типу, в залежності від ідеї [9].

Після успішного прориву AlexNet в 2012 році, коли проводили ILSVRC 2012 Workshop різносторонньо та різнобічно велося обговорення про те як можливо узагальненити та правильніше структуризувати увесь обсяг тих отриманих результатів згорткових нейронних мереж коли постало завдання розпізнавання зображувальних об'єктів у змаганнях PASCAL VOC. Тоді було розглянуто та вирішено що в PASCAL VOC значно краще та якісніше може виявляти об’єкти інша запропонована структура - R-CNN, що стало означати що методи, побудовані на гістограмах орієнтованих градієнтів мають великі недоліки в якості [12].

R-CNN, яка вирішила проблему — це така стуктура яка заснована на алгоритмі для виявлення об’єктів, яка використовує згорткову нейронну мережу (CNN) для класифікації областей зображення в зображенні [1]. Замість того, щоб класифікувати кожну область за допомогою ковзного вікна, детектор R-CNN обробляє лише ті області, які, ймовірно, містять об’єкт. Це значно зменшує обчислювальні витрати під час роботи CNN.

Під час трансферного навчання мережа, навчена на великій колекції зображень, наприклад ImageNet [2], використовується як відправна точка для вирішення нової задачі класифікації або виявлення. Перевага використання цього підходу полягає в тому, що попередньо навчена мережа вже засвоїла багатий набір функцій зображення, які можна застосувати до широкого кола зображень. Це навчання можна перенести на нове завдання шляхом точного налаштування мережі. Мережа точно налаштована шляхом внесення невеликих коригувань до ваг, таким чином, щоб представлення функцій, засвоєні для вихідного завдання, трохи коригувалися для підтримки нового

завдання. Перевага трансферного навчання полягає в тому, що зменшується кількість зображень, необхідних для навчання, і час навчання.

У моделі R-CNN показано, що саме ЗНМ здатні значно якісніше виявляти об'єкти в PASCAL VOC якщо їх порівняннювати з тими методами, які побудовані на підґрунті гістограм які працюють за допомогою орієнтованих градієнтів [9]. Згорткові нейронні мережі працюють за допомогою згорткових шарів які відділяються картами і з’єднуються між собою в одну структуру [11].

Ранні моделі згорткових нейронних мереж працювали за допомогою методу ковзного вікна. Ці мережі мали, зазвичай, тільки 2 згорткових шари і шар субдискретизації. Метод R-CNN - архітектура яка запропонувала використовувати замість методу ковзного вікна метод "розпізнавання всередині області" вибірковий пошук для вилучення лише 2000 регіонів із зображення, і воно називається пропозиціями регіонів. Тому тепер замість того, щоб намагатися класифікувати величезну кількість регіонів, можна просто працювати з 2000 регіонами. Ці 2000 пропозицій регіонів генеруються за допомогою алгоритму вибіркового пошуку.

Вибірковий пошук включає в собі:

1. Створення початкової субсегментації, генеруємо багато регіонів-кандидатів.

2. Користування жадібним алгоритмом для рекурсивного об’єднання подібних регіонів у більші.

3. Використовує згенерованих регіонів, щоб створити остаточні пропозиції регіонів-кандидатів.

Ці 2000 пропозицій регіонів-кандидатів деформуються в квадрат і подаються в згорткову нейронну мережу, яка створює 4096-вимірний вектор ознак як вихідний результат. CNN діє як екстрактор ознак, а щільний вихідний шар складається з об’єктів, витягнутих із зображення, а вилучені об’єкти надходять у машині опорних векторів (МОВ) відому як SVM для класифікації присутності об’єкта в цій пропозиції регіону-кандидата. На

додаток до передбачення присутності об’єкта в пропозиціях регіону, алгоритм також передбачає чотири значення, які є значеннями зміщення, щоб збільшити точність рамки. Наприклад, з огляду на пропозицію регіону, алгоритм передбачав би присутність людини, але обличчя цієї особи в пропозиції регіону можна було б скоротити навпіл. Таким чином, значення зміщення допомагають налаштувати обмежувальну рамку так званий Bbox пропозиції регіону. Схема роботи взаємодій для R-CNN зображена на рисунку 2.3.

Схема взаємодій між вхідним зображенням, зрортковими нейронними мережами, SVM та Bbox регіонами моделі R-CNN зображена на рисунку 2.3.

Проблеми з R-CNN та недоліки архітектури

- Навчання мережі все ще займає величезну кількість часу, оскільки на практиці доводиться класифікувати 2000 пропозицій регіонів на одне зображення.

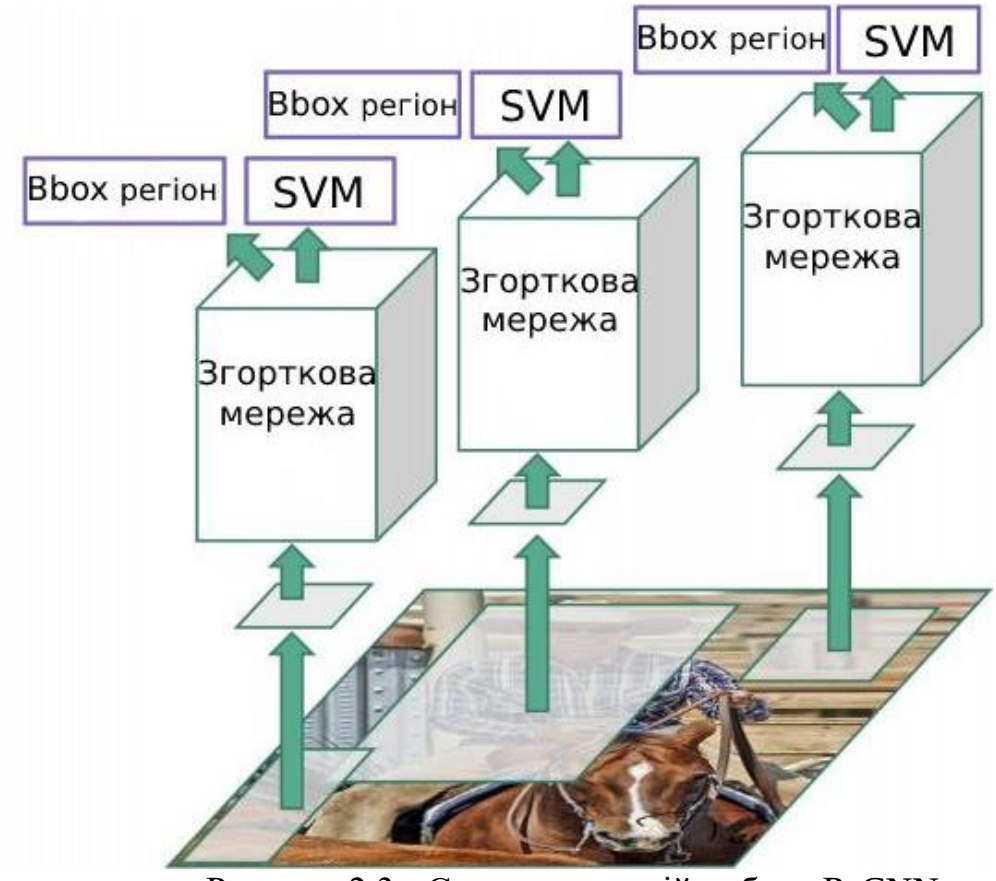


Рисунок 2.3 - Схема взаємодій роботи R-CNN

- Його неможливо реалізувати в реальному часі, оскільки для кожного тестового зображення потрібно близько 47 секунд.

- Алгоритм селективного пошуку є фіксованим алгоритмом. Тому на цьому етапі навчання не відбувається. Це може призвести до створення поганих пропозицій щодо регіонів-кандидатів.

У моделі Fast R-CNN запропоновані методи поліпшення мережі R-CNN [10]. У Fast R-CNN запропоновано новий алгоритм навчання, який виправляє недоліки R-CNN, підвищуючи його швидкість і точність. Алгоритм Fast R-CNN зображений на рис. 2.4. Вхідне зображення і область множинного інтересу (ROI) надходять на вхід згорткової мережі. Область множинного інтересу - це шар, що виконує субдискретизацію вхідного зображення і створює карту ознак фіксованого розміру 7x7. Основна мета цього шару - прискорення часу навчання і тестування.

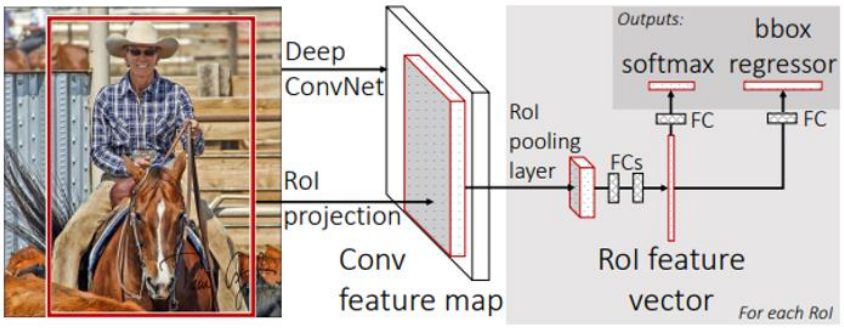


Рисунок 2.4 Алгоритм Fast R-CNN.

Fast R-CNN має такі переваги:

- Якість виявлення вище, ніж R-CNN.
- Навчання проходить в один етап.
- При навчанні оновлюються ваги всіх шарів мережі.
- Не потрібно простір жорсткого диска для зберігання ознак.

У попередніх моделях (R-CNN, Fast R-CNN) використовувалися методи пропозиції областей для первинного створення обмежувального прямокутника для об'єктів, потім ці області класифікувалися.

YOLO - це передова система виявлення об'єктів в реальному часі. Завдяки широкому діапазону доступних варіантів можна вибрати версію, що

найбільше підходить для ваших потреб. Наприклад, YOLO - це "компактний" варіант, який може працювати швидко навіть на смартфонах або Raspberry Pi [11].

Одна згорткова мережа водночас передбачає велетенську сукупність обмежувальних прямокутників і ймовірності класів для них.

Архітектура ЗНМ моделі YOLO зображена на рисунку 2.5.

Переваги YOLO:

- YOLO більш ніж у двічі збільшив швидкість Faster R-CNN, але MAP трохи знизився.;

- Приймаюче поле, здатне сприймати контекстну інформацію, зазвичай являє собою повне зображення, тому його здатність до узагальнення буде кращою. В той же час YOLO може «бачити» загальну інформацію всього зображення під час навчання та міркувань. Що знижує ймовірність помилкового виявлення.

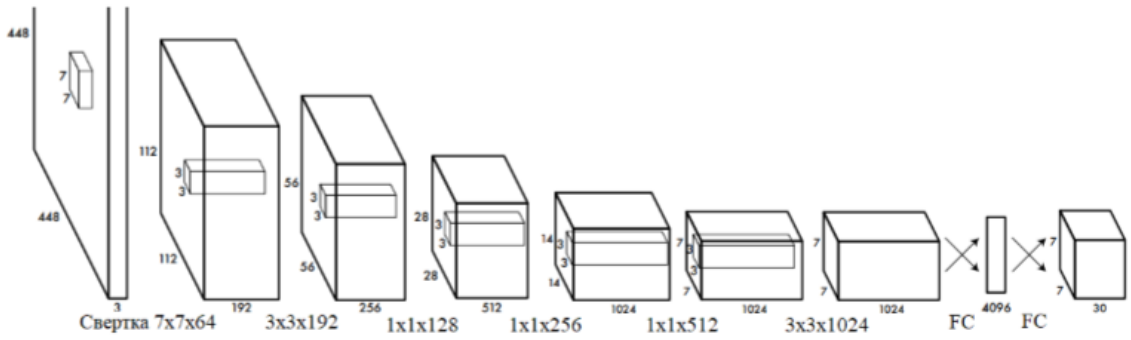


Рисунок 2.5 - Архітектура згорткової мережі в YOLO

- YOLO робить великий прогноз по всьому відтворенню, різних методів ковзного вікна, та технік заснованих на всіх областях пропозиції.

Продовженням розвитку ідеї YOLO стали успіхи досягнень у моделі SSD, яка використовує такий самий алгоритм, який полягає у виявленні потрібних об'єктів одночасно для всього зображення, як і відображено в такій її назві SSD, - Single Shot Detector [12]. CNN складається з серії шарів,

де кожен рівень визначає конкретне обчислення. Deep Learning Toolbox забезпечує функціональність для легкого пошарового проектування CNN. Модель Одноразовий детектор (SSD) класифікація зображень і виявлення об'єктів як класифікація зображень у комп’ютерному зорі бере зображення та прогнозує об’єкт на зображенні, тоді як виявлення об’єктів не тільки передбачає об’єкт, але й знаходить його розташування в термінах обмежувальних рамок. Для ілюстрації, якщо припустити, що в зображенні є щонайбільше один клас і один об’єкт, вихідні дані моделі виявлення об’єктів мають містити:

Ймовірно, що є предмет, висота обмежувальної рамки, ширина обмежувальної рамки, горизонтальна координата центральної точки обмежувальної рамки, вертикальна координата центральної точки обмежувальної рамки.

На рис. 2.6 зображено порівняння архітектур YOLO і SSD. У моделі SSD додаються додаткові шари, що дозволяють виявляти об'єкти різного розміру.

При порівнянні моделей можна отримати однозначної відповіді, яка модель є кращою. Для застосування в реальному житті, необхідний баланс між точністю та швидкістю. Крім того, необхідно врахувати й інші фактори, які вносять вклад в якість моделі:

• Модель вилучення ознак (VGG, ResNeXt, Inception, MobileNet).

• Дозвіл вхідного зображення.

• Як пророкування виключається з розрахунку функції втрат.

• Число передбачень в моделі.

• Кодування обмежувальних прямокутників.

• Розширення (аугментація) наборів даних.

• Набір даних, що використовується для навчання.

• Використання багатомасштабних зображень в навчанні або тестуванні.

• Локалізація функції втрат.

• Робота з програмним забезпеченням для глибокого навчання.

• Параметри навчання, такі як розмір батча (кількість даних для навчання на одному кроці), крок навчання, падіння швидкості навчання.

Також на адекватність порівняння впливає те, що технології розвиваються швидко, і будь-яке порівняння швидко застаріває. Результати PASCAL VOC 2012 наведені в таблиці 2.1 [13]. Використовується середня точність по всіх класах (*mAP*) як критерій якості, найкращі результати у моделей SSD512 (*mAP* = 74,9) і SSD300 (*mAP* = 72,4).

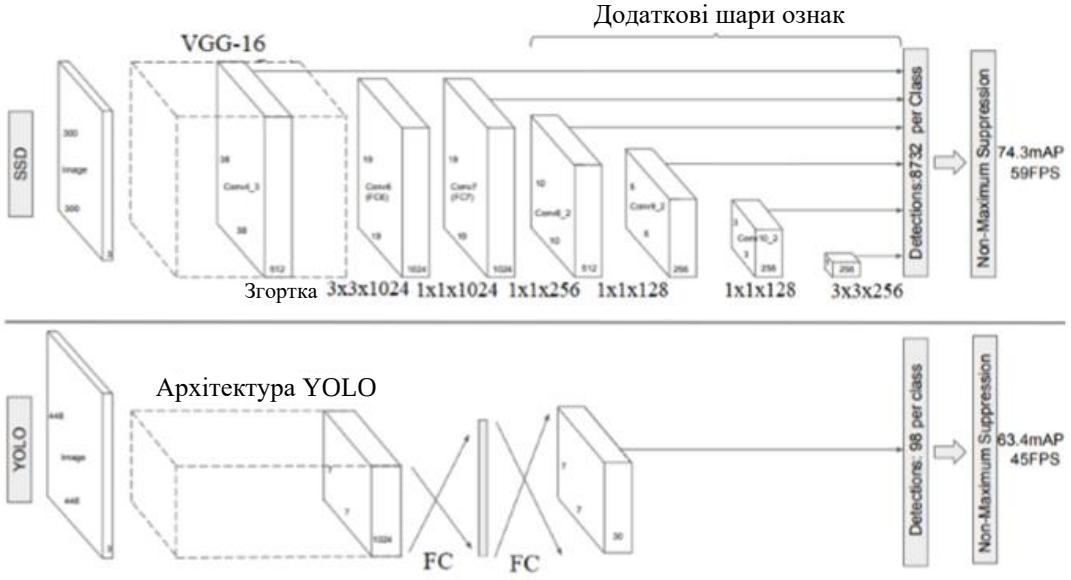


Рисунок 2.6 - Порівняння архітектури SSD (зверху) і YOLO (знизу)

На рис. 2.7 показаний порівняльний результат точності моделей, які навчалися на наборі PASCAL VOC 2007 і PASCAL VOC 2012 [13]. Значення вимірювалися на перевірочному наборі даних PASCAL VOC 2012. Для SSD, показані результати для вхідного зображення 300×300 і 512×512. Для YOLO показані результати для вхідного зображення 288×288, 416×416 і 514×514. Більш високий дозвіл вхідного зображення для моделі забезпечує більш високе значення *mAP*, але меншу продуктивність.

Таблиця 2.1 - Результати порівняння моделей по *mAP*

Модель	*mAP*
Fast R-CNN	68.4
YOLO	67.9
SSD-300	72.4
SSD-512	74.9

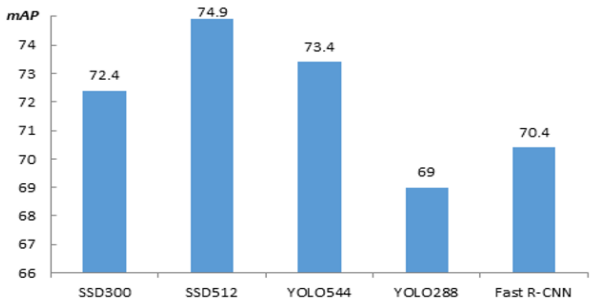


Рисунок 2.7 – Результати точності в PASCAL VOC 2007 і PASCAL VOC 2012

На рисунку 2.8 показаний порівняльний результат продуктивності, що вимірюється в FPS (кадрів в секунду) [13]. Навчальні та перевірочні набори даних аналогічні попереднім.

.

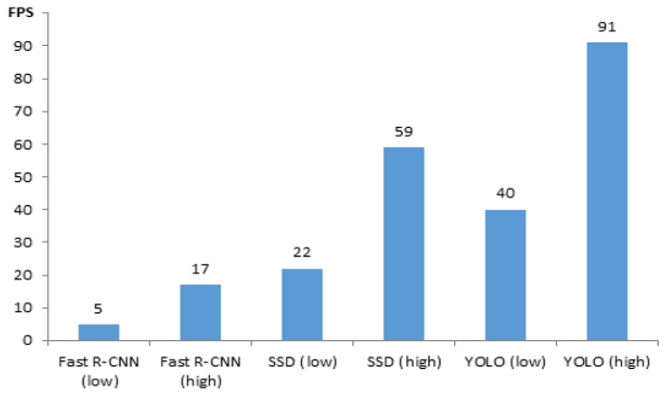


Рисунок 2.8 - Порівняння швидкодії моделей

За наведеними вище результатами видно, що найбільшу продуктивність забезпечує модель YOLO, але вона має точність нижче в порівнянні з іншими моделями. Високу точність має модель R-CNN і її модифікації, але вона має дуже низьку продуктивність. Модель SSD має

оптимальне співвідношення точності і продуктивності, тому ця модель взята за основу для розробки моделі класифікації зображень.

Для того щоб розробити власну модель на основі згорткових нейронних мереж, необхідно розглянути також принцип навчання згорткових нейронних мереж.

2.3 Навчання згорткових нейронних мереж

Згорткові нейронні мережі (CNN) — це особливий тип нейронної мережі. Вони дуже добре працюють при застосуванні до наборів даних зображень.

Процес навчання згорткових нейронних мереж починається з ініціалізації CNN випадковими вагами. Це в основному означає, що мережа повністю здогадується. Після того як він вона свій прогноз, вона перевіряє, наскільки неправильно вона використовує функцію втрати, а потім оновить свої ваги, щоб наступного разу зробити кращий прогноз.

Згортковий шар створює купу карт об’єктів. Для CNN, який використовується для опису різних зображень, наприклад тварин або облич. Ознаками, які шукає перший згортковий шар, можуть бути різні краї об’єктів. Це як скласти список різних країв на малюнку. Потім цей список передається до іншого згорткового шару, який виконує подібну дію, за винятком того, що він шукає більші форми на зображенні. Це може бути нога тварини або око на обличчі. Зрештою, функції поглинаються повністю пов’язаним шаром, який класифікує зображення [12].

Об’єднані шари також використовуються зі згортковими шарами. Для цього потрібна ще одна лупа, але вона не шукає особливостей. Натомість, щоб зменшити складність зображення, потрібно найбільше значення в регіоні. Об’єднання шарів показано на рисунку 2.9.

Це корисно, оскільки більшість зображень великі. Вони мають масу пікселів, що ускладнює роботу з ними для процесора. Об’єднання дозволяє

зменшити розмір зображення, зберігаючи при цьому більшість важливої нформації. Об’єднання також використовується для запобігання переобладнанню, коли модель стає занадто вдалою в ідентифікації даних, на яких ми її навчаємо, і погано працює для інших прикладів, які ми їй надаємо.

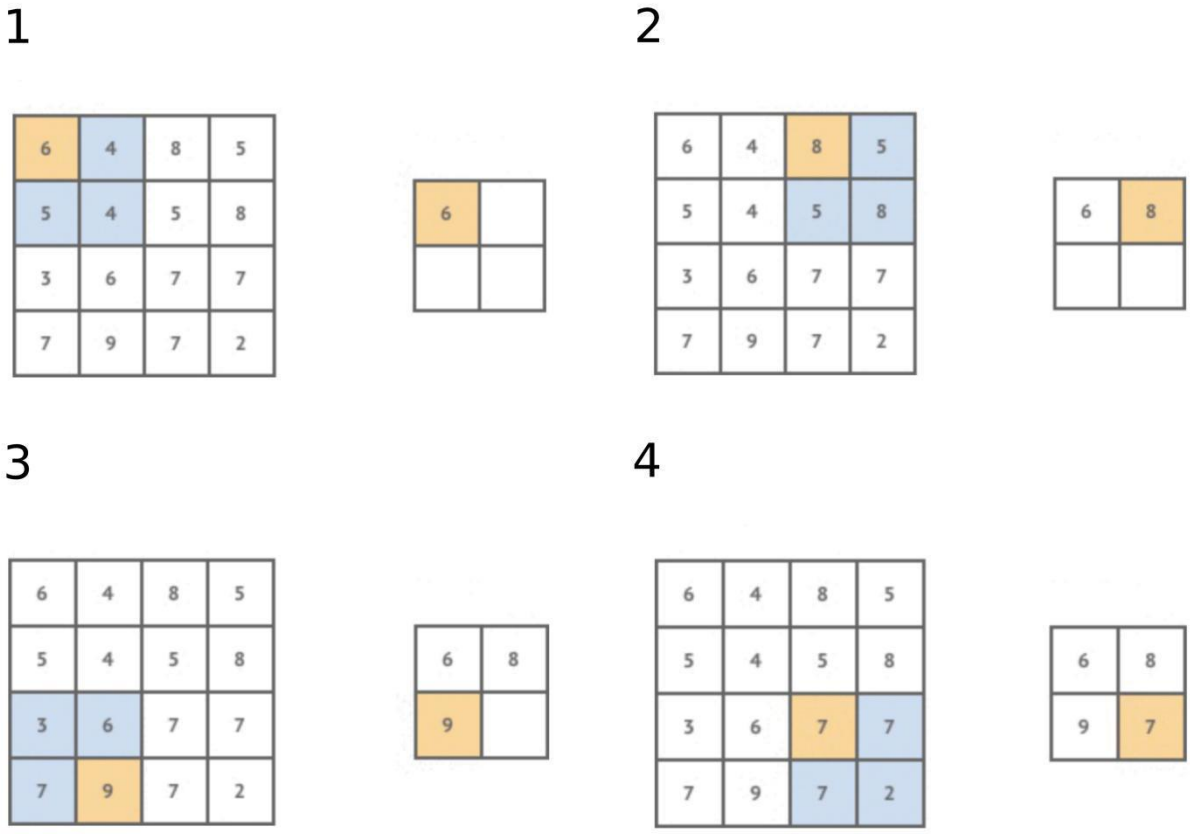


Рисунок 2.9 - Об’єднання шарів згортки

Мета навчання полягає в тому, щоб налаштувати всі ваги та зміщення в мережі, коли була зроблена помилка класифікації, щоб помилка на виході була зведена до мінімуму. Це робиться за допомогою градієнтного спуску для ваг і ухилів

$$w_{ij}(\ell) = w_{ij}(\ell) - \alpha \frac{\partial E}{\partial w_{ij}(\ell)} \tag{2.1}$$

та

$$b_i(\ell) = b_i(\ell) - \alpha \frac{\partial E}{\partial b_i(\ell)}, \tag{2.2}$$

де α – скалярний приріст корекції, який називається константою швидкості навчання. На жаль, зміна вихідної помилки щодо змін ваг і зміщень у прихованих шарах невідома. Зворотне поширення є схемою, яка:

1) поширює помилку у вихідних даних, яка відома, назад через усі приховані шари мережі

2) використовує зворотну помилку для того щоб виразити обидві частини у формулах (2.1) і (2.2).

З точки зору функції активації, вихідної помилки та поточних значень ваг і зміщень, усі вони є відомими величинами на кожному рівні мережі під час навчання.[16].

Глибока повнозв'язна згорткова мережа (FCN) складається з шарів штучних нейронів, у яких вихід кожного нейрона в шарі пов’язаний з входом кожного нейрона наступного шару, звідси термін повний зв’язок. Вхідний шар формується з компонентів вектора шаблону X1,x2,...,xn, а кількість нейронів у вихідному шарі дорівнює кількості класів шаблону в даній програмі. Вхідний і вихідний шари видимі, тому що ми можемо спостерігати значення їх вихідних даних.

Усі інші шари нейронної мережі є прихованими шарами. Зауважте, що CNN пов’язані не повністю в тому сенсі, що кожен елемент карти в одному шарі не пов’язаний з кожним елементом карт наступного шару.

Головним задумом навчання мережі CNN/FCN є визначення ваги та зміщення об’ємів згортки в першому, а також ваг і зміщень нейронів в останньому, які вирішують дану проблему. Ці параметри оцінюються за допомогою зворотного поширення, методології ітеративного коригування коефіцієнтів на основі значень помилки, що спостерігається на вихідних нейронах FCN.

Для задачі навчання нейронних мереж як першочерговим завданням виступає мінімізування заданої функції помилки [14]. Зазвичай, при цьому потрібно оптимізувати апостеріорну імовірність.

$$p(\theta|D) = p(\theta)\prod_{d\in D} p(d|\theta),$$

де функція правдоподібності $p(\theta)$ - помилка на тренувальній вибірці, функція апріорного розподілення $p(d \mid \theta)$ — функція регуляризації.

Оптимізація - це таке завдання що полягає у пошуку аргументів за заданою функцією, де для функції застосовується мінімізація. Локально у нейронних мережах є багато екстремумів функції помилки. Для пошуку оптимального екстремуму, застосовують такий метод оптимізації як градієнтний спуск, він є евристичним.

Після представлення поверхневого ділення функції помилки, наступним кроком оптимізації постає завдання обчислення градієнта. Функцію, яка визначає поверхню, позначаємо через $E(\boldsymbol{\theta})=E(\theta_1, \theta_2,\ldots, \theta_n)$,

де $(\theta_1, \theta_2,\ldots, \theta_n)$ – є параметрами самої функції, а ∇E ϵ вектором похідних функції індивідуально для кожної компоненти та виступає як градієнт.

$$\nabla_\theta E = \begin{pmatrix} \frac{\partial E}{\partial \theta_1} \\ \vdots \\ \frac{\partial E}{\partial \theta_n} \end{pmatrix}.$$

Градієнтний спуск є найпростішим, але найбільш використовуваним алгоритмом оптимізації. Він активно використовується в алгоритмах лінійної регресії та класифікації. Зворотне поширення в нейронних мережах також використовує алгоритм градієнтного спуску.

Градієнтний спуск — це алгоритм оптимізації першого порядку, який залежить від похідної першого порядку функції втрат. Він обчислює, у який бік слід змінити ваги, щоб функція досягла мінімуму. За допомогою зворотного поширення втрати переносяться з одного шару на інший, а параметри моделі, також відомі як ваги, змінюються залежно від втрат, щоб можна було мінімізувати втрати.

алгоритм: $\theta = \theta - \alpha \cdot \nabla J(\theta)$

Переваги :

- Легке обчислення.
- Легко реалізувати.
- Легко зрозуміти.

Недоліки :

- Може потрапити в пастку на локальних мінімумах.
- Вага змінюється після розрахунку градієнта для всього набору даних.

Отже, якщо набір даних занадто великий, то для досягнення мінімуму можуть знадобитися роки.

Для обчислення градієнта для всього набору даних потрібна велика пам’ять.

У простому градієнтного спуску швидкість навчання вказується вручну, і це може стати причиною таких проблем:

- Якщо крок є маленьким то навчання відбуватиметься занадто довго і ймовірність потрапити в в таку ситуація як невдалий локальний функціональний мінімум зростає (див. Рисунок 2.9(а)),
- Якщо крок завеликий, то є ризик пропуску необхідного локального функціонального мінімуму (див. рисунок 2.9(б)).

Для вирішення таких проблем використовується саме стохастичний градієнтний спуск, а не стандартний. Він полягає у підрахунку помилки і перезавантаження ваг відбувається не після проходження усієї навчальної множини, а вже після проходження кожного з прикладів:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \nabla E(f(\mathbf{x}_t, \boldsymbol{\theta}_{t-1}), y_t)$$

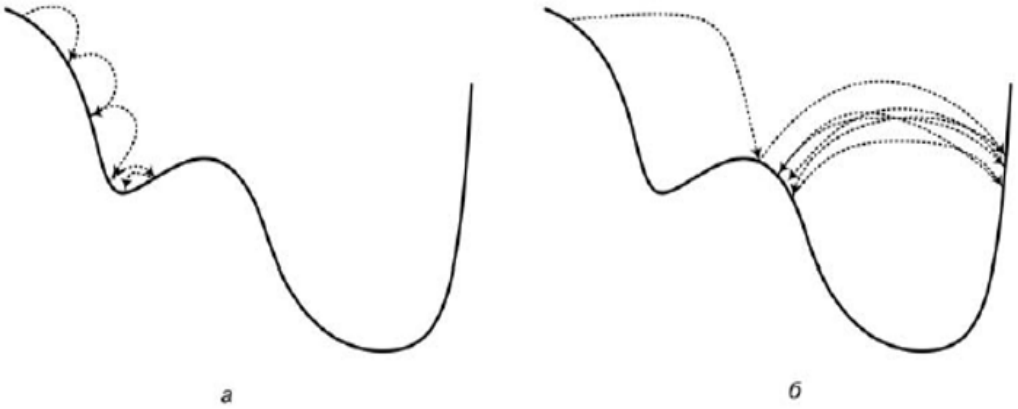


Рисунок 2.9 - Проблеми у швидкості градієнтного спуску:
а) — занадто маленькі кроки,
б) — занадто великі кроки.

Найголовніша перевага стохастичного градієнтного спуску - дуже висока швидкість обрахунку помилки, тому що ваги оновлюються після кожного кроку, і це є причиною збільшення швидкості навчання. Також перевагою є те, що даний спуск вимагає менше пам’яті, оскільки не потрібно зберігати значення функцій втрат також входить до переваг.

Найкращим серед усіх варіантів алгоритмів градієнтного спуску є міні-пакетний градієнтний спуск. Це покращення як SGD, так і стандартного градієнтного спуску. Він оновлює параметри моделі після кожної партії. Таким чином, набір даних ділиться на різні пакети, і після кожного пакету параметри оновлюються.

$\theta=\theta-\alpha\cdot\nabla J(\theta; B(i))$, де $\{B(i)\}$ — пакети навчальних прикладів .

Переваги міні-пакетного градієнтного спуску в тому, що спуск часто оновлює параметри моделі, має меншу дисперсію і для його роботи потрібен середній обсяг пам’яті.

Усі типи градієнтного спуску мають деякі проблеми:

- Вибір оптимального значення швидкості навчання. Якщо швидкість навчання занадто мала, то градієнтний спуск може зайняти багато років для зближення.

- Мати постійну швидкість навчання для всіх параметрів. Можуть бути деякі параметри, які ми не хочемо змінювати з тією ж швидкістю.

- Може потрапити в пастку локальних мінімумів.

2.4 Розробка алгоритму формування архітектури ЗНМ

При проектуванні однозначно потрібно дотримуватись усіх рекомендацій які вимагає архітектура побудови згорткових нейронних мереж коли питання постає у виборі параметрів щоб забезпечити та гарантувати повну вправність, ефективність та точність щодо одержуваних результаті під час роботи з об’єктами у мережах нейронів. Подані рекомендації складалися на підґрунті аналізу найефективніших архітектур, які взагалі існують у ЗНМ [5].

Рекомендації про побудову архітектури вхідного шару згорткової мережі:

· Усі вхідні двомірні зображення повинні бути квадратними.

· Кожне зображення що подається на вході має ділитися на 2 потрібну велику кількість разів аж поки результат поділу досягне однозначних чисел пікселів зображення.

Рекомендації щодо побудови архітектури структурних елементів шару згортки:

· Щодо використання фільтрів потрібно використовувати малий розмір у порядку таких розмірностей: 3 х 3 або 5 х 5, але якщо на вході подано зображення більше ніж середніх, великих зображень то для першого згорткового шару параметри розмірів можуть бути змінені до порядку цифр 7 х 7.

· Для згорткових шарів фільтри слід використовувати у кількості залежній від розміру вхідного мережевого шару і чим він більший, використовувати більше фільтрів і якщо шари згортки є глибокі, значить і кількість фільтрів має бути більшою, а якщо вхідний шар мережі менший, то і фільтрів має бути менше.

Рекомендації до архітектури шару субдискретизації:

- Переважно в якості опції шару субдискретизації використовувати вибір максимального значення, замість розрахунку середнього значення.
- Переважно використовувати розміри вікна шару субдискретизації 2 x 2, тому що великі розміри вікна надають значний вплив на руйнування ознак, виділених зготковими шарами мережі.

Рекомендації до архітектури згорткової нейронної мережі в цілому:

- Процес формування архітектури мережі повинен враховувати глобальні параметри архітектури, наведені на рисунку 2.11.

На рисунку 2.11 значення змінних n, m, k і d – додатні цілі числа, причому n [1; 3], m [0; 1], k [0; 2].

- Бажано, щоб мережа містила якомога більше шарів (Більше значення d), тобто була якомога глибше.
- Чим більше і складніше зображення, тим більше значення n.
- Бажано використовувати якомога менше значення k.

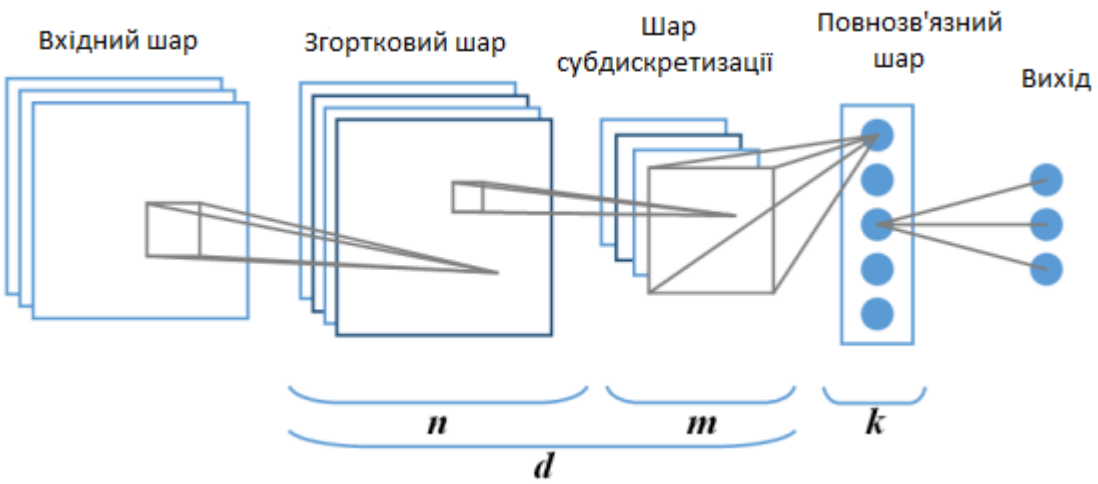


Рисунок 2.11 – Шаблон архітектури згорткової мережі

Складені рекомендації є основною для розробки алгоритму формування найбільш ефективної архітектури згорткової нейронної мережі.

Об’єднання, або підвибірка, мотивовано дослідженнями, які припускають, що мозок ссавців виконує аналогічні операції під час візуального пізнання. Об’єднана карта – це просто карта об’єктів з нижчою роздільною здатністю. Типовим методом об’єднання є заміна значень кожного околиці розміром, скажімо, 2 x 2, на картах об’єктів на середнє значення в околиці. Використання зони сусідства розміром 2 x 2 призводить до об’єднаних карт розміром наполовину в кожному просторовому вимірі розміру карт об’єктів. Таким чином, наслідком об’єднання є значне скорочення даних, що сприяє прискоренню обробки. Однак основним недоліком є те, що розмір карти також значно зменшується щоразу, коли виконується об’єднання. Навіть з околицями розміром 2 × 2 скорочення вдвічі в кожному просторовому вимірі швидко стає проблемою, коли кількість шарів велика по відношенню до розміру вхідних зображень. Це одна з причин, чому об’єднання в пул використовується лише спорадично у великих системах CNN. Як і у випадку з функціями активації, тип використовуваного об'єднання також відіграє роль у визначенні архітектури CNN. На додаток до усереднення по сусідству, на практиці

використовуються два додаткові методи об’єднання: максимальне об’єднання, яке замінює значення в околиці максимальним значенням його елементів, і об’єднання 12, при якому об’єднане значення в околиці є квадратним коренем з сума їх значень у квадраті. Було продемонстровано, що максимальне об’єднання є особливо ефективним при класифікації великої бази даних зображень, а також має додаткову перевагу — простота та швидкість. Як зазначалося раніше, коли об’єднання використовується в шарі, кожна об’єднана карта створюється лише з однієї карти об’єктів, тому кількість об’єктів та об’єднаних карт однакова.

Основна архітектура кожного етапу CNN визначається шляхом вказівки кількості карт ознак і того, чи використовується об’єднання на цьому етапі. Також вказуються розміри ядра та пулу, а також крок згортки, що визначається як кількість приростів зміщення ядра між операціями згортки і. Наприклад, крок у два означає, що згортка виконується в кожному іншому просторовому місці введення.

Процес будування архітектури ЗНМ для розподілення об’єктів вхідного зображення по групах полягає у «згортуванні» мережевого вхідного шару до угрупувань з найбільш малими розмірами. Розміри таких угрупувань для прикладу можуть бути такими як, 2x2xD і lxlxD, звідки D – це глибина шару. Продовженням процесу будування ЗНМ є перетворення до С сигналам які належать до одного виміру і які є сигналами ймовірності приналежності еталону(зразка), що підійшов на вхід мережі прямо до одного з С класів.

Пропонований підхід представляє собою алгоритм, який пояснює та дає інструкцію для вибору параметрів у функціях та методах архітектури ЗНМ залежно від головних заданих характеристик даних що на вході. Це відноситься для кожного з етапів в якому формуються шари мережі у кожний у свою чергу у послідовному порядку, що схиляється на інструкцію для побудови архітектури згорткових нейронних мереж. В основі алгоритму передбачено функціональне введення основних значень імен та величин у

параметрах мережі і рандомне введення деяких з них. що означає, що результатами алгоритму внаслідок його виконання для різних вхідних даних може бути набір архітектур ЗНМ[5].

В описі алгоритму прийняті наступні позначення:

– N – розмір квадратного попереднього шару мережі (для першого згорткового шару є розміром вхідного шару).
– D – глибина вхідного шару мережі (кількість колірних каналів зображення)
– C – кількість класів, що належність до яких визначається класифікатором.
– P – кількість рядків і стовпців, що містять нулі, що додаються до меж шару, що передує згортковому шару (параметр архітектури згорткового шару).
– S – зміщення між фільтрами при формуванні сигналів нейронів згорткового шару/зсув між вікнами при формуванні сигналів шару субдискретизації.
– F – розмір квадратних фільтрів згорткового шару, причому F може набувати значень 3, 5 або 7.
– Filters – глибина згорткового шару (кількість фільтрів).
– Filtersp – глибина попереднього згорткового шару (глобальний параметр циклу формування шарів мережі).
– AF_c – функція активації нейронів згорткового шару.
– U – розмір квадратного вікна для шару субдискретизації.
– Subf – тип функції шару субдискретизації (max – максимум, або avg– розрахунок середнього).
– K – кількість нейронів в повнозв'язному шарі.
– AFf_c – функція активації повнозв'язного шару.

Рівень мережі - актуальна послідовність з n згорткових шарів і m шарів субдискретизації, причому для рівня дані змінні приймають цілочисельні значення з діапазонів: n э [1; 2], m э [0; 1].

– d – глибина мережі – кількість рівнів мережі (глобальний параметр циклу формування шарів мережі).

Текстовий опис етапів алгоритму формування архітектури ЗНМ:

1. Опциональне введення основних параметрів алгоритму: F – чим більші особливості необхідно детектувати на зразках даних, тим більше рекомендується ставити значення F (по замовчуванню F не задається), параметри рівня мережі: n, m – чим складніше зображення, тим більше рекомендується ставити значення n (за замовчуванням n = 1, m = 1).

2. Формування параметрів вхідного шару мережі: введення розміру вхідного зображення, що має квадратну форму, зі значенням розміру боку, рівним N, причому N має багаторазово ділитися на 2 аж до однозначних чисел. Введення глибини вхідного шару D, зазвичай рівного кількості колірних каналів зображення. Введення значення кількості класів C.

3. Ініціалізація глобальних змінних алгоритму: d = 0, Filtersp = 0.

4. Ініціалізація змінної циклу формування згорткових шарів in = 0.

5. Якщо значення in не дорівнює n, то проводиться формування параметрів згорткового шару, інакше проводиться перехід до пункту 9.

F обирається в залежності від N за формулою (2.3).

$$F = \begin{cases} 7, if(N \geq 64) \\ 5, if(32 \leq N < 64) \\ 3, if(N < 32) \end{cases} \quad (2.3)$$

Filters вибирається залежно від значень параметрів F і d по формулі (2.4). Після вибору значення F, проводиться операція Filtersp = Filters.

$$FIlters\{ \quad \begin{matrix} 8, if(F = 7d = 0) \\ 16, if(F = 5d = 0) \\ 24, if(F = 5d = 0) \end{matrix} \quad (2.4)$$

$$FIliters_p = \| 1.25 * Filiters_p, if(d > 0)$$

Якщо in = 0, то проводиться формування першого згорткового шару, інакше перехід P і S розраховуються шляхом вирішення системи рівнянь – формула (2.5), для шарів в яких ширина і висота рівні N і розмір попереднього шару дорівнює розміру згорткового шару.

$$\begin{cases} S = (N - F + P * 2)/(N - 1)) \\ P = ((N - 1)S - N + F)/2 \end{cases} \quad (2.5)$$

Проводиться формування параметрів інших згорткових шарів: P вибирається мінімальним цілочисельним з відрізка [0; F] таким чином, щоб розміри згорткового шару були цілочисельними. S вибирається мінімальним цілочисельним з відрізка [1; F] таким чином, щоб розміри згорткового шару були цілочисельними.

6. Розрахунок N для сформованого згорткового шару за формулою (2.6), де ширина і висота зображення рівні.

$$N = (N_P - F + P * 2)/S + 1 \quad (2.6)$$

де N_p – розмір квадратного попереднього шару мережі.

7. Вибір в якості функції активації нейронів згорткового шару порогової функції, згідно з формулою (2.7).

$$AF_c = (0, x) \quad (2.7)$$

де – x – сигнал нейрона згорткового шару, отриманий в результаті фільтрації.

8. $i_n = i_n + 1$ і перехід до пункту 5.

9. Якщо $m = 1$ і $N > 1$, то відбувається формування параметрів слою субдискретизації, інакше перехід до пункту 11:

   U вибирається рівним значенню 2.

   S Вибирається рівним U.

   Subf вибирається як функція max – вибір максимального значення.

10. Розрахунок N для сформованого шару субдискретизації мережі за формулою (2.8), де ширина і висота зображення рівні.

$$N = (N_P - U)/S + 1 \tag{2.8}$$

де N_p – розмір квадратного попереднього слою мережі.

11. $d = d + 1$

12. Ухвалення рішення про формування наступного рівня мережі: якщо $N \geq 3$, то проводиться формування нового рівня мережі – перехід до пункту 4, інакше проводиться перехід до пункту 13.

13. Формування параметрів повнозв'язного шару мережі:

   K вибирається рівним C

Як функції активації нейронів повнозв'язного шару використовується сігмоїдної функції згідно з формулою (2.9).

$$AF_{fc} = \frac{1}{1+e^{-x}} \tag{2.9}$$

де x – сигнал нейрона повнозв'язну шару, отриманий в результаті зваженого підсумовування вхідних сигналів нейрона.

Розроблений алгоритм представлений на рисунку 2.12

Модифікація архітектури мережі проводиться шляхом її ускладнення і застосування послідовності правил:

1. Додавання згорткового шару до одного з рівнів мережі згідно з алгоритмом, починаючи з останнього найбільш глибокого рівня.
2. Додавання більшої кількості фільтрів згорткового шару мережі (збільшення відбувається на 25% від попереднього значення кількості фільтрів), починаючи з останнього найбільш глибокого рівня.
3. Додавання нового рівня мережі, який приєднується до останнього, найбільш глибокого рівня сформованої мережі.

Для всіх операцій навчання в даному підході була застосована технологія яка є однією з кращих за ефективністю, зручністю в налаштуваннях параметрів швидкості одна з існуючих модифікацій незамінного за якістю алгоритму реверсного поширення помилки, її назва – Adadelta. Ця технологія базується на динамічній, довільній зміні швидкості навчання мережі в рамках значень помилки кожного фільтра згорткових шарів [6].

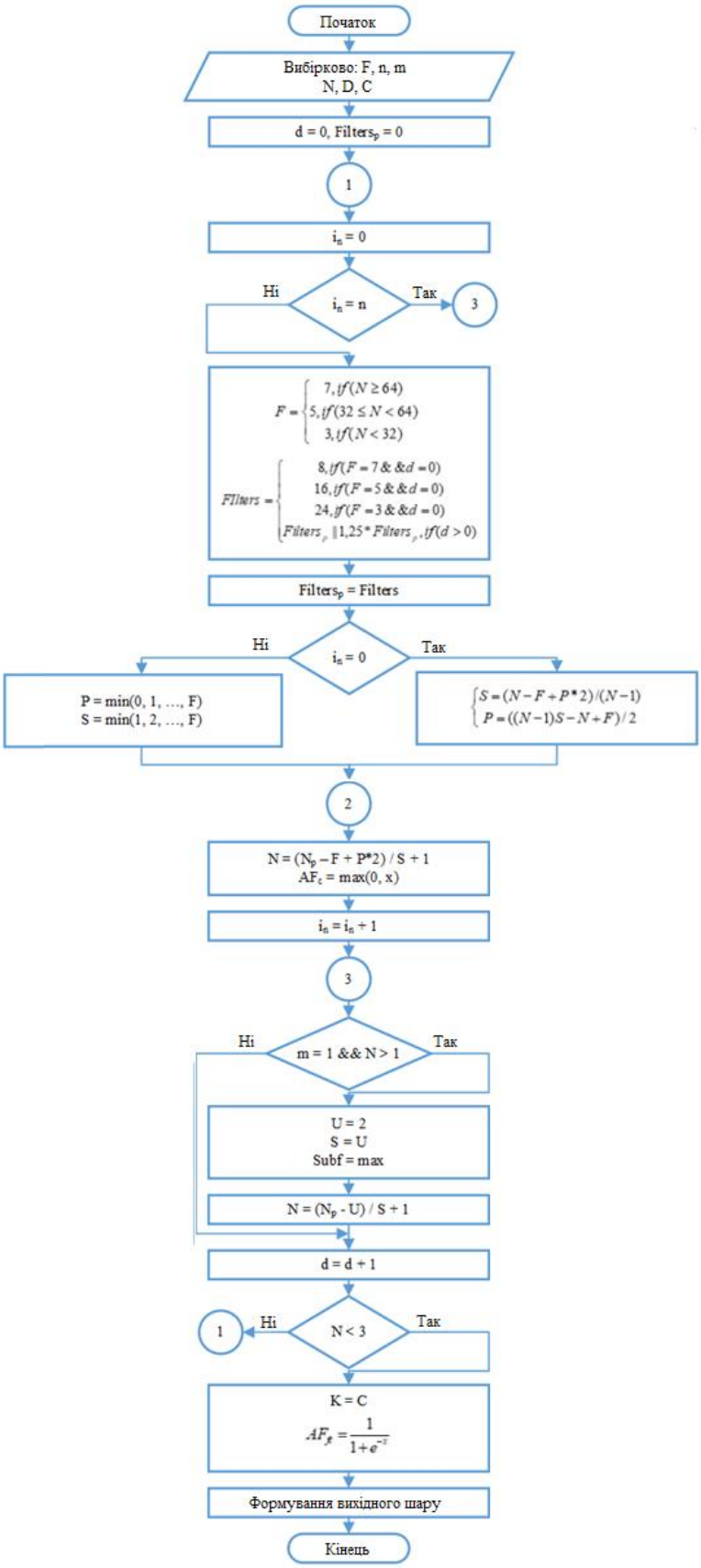


Рисунок 2.12 – Схема алгоритму формування архітектури ЗНМ

Обмеженість застосування розробленого алгоритму для класу задач класифікації зображень пояснюється основними характеристиками вхідних даних, що використовуються для підбору значень параметрів архітектури, а також особливостями використовуваних в основі алгоритму рекомендацій до побудови архітектур ЗНМ, призначених тільки для класифікації об’єктів на кольорових зображеннях.

Представлений апарат формування ефективної архітектури ЗНМ ліг в основу пропонованого в даній роботі підходу до створення класифікатора на основі згорткових нейронних мереж.

2.5 Висновки

У розділі було проаналізовано та обґрунтовано вибір згорткової нейронної мережі для класифікації зображень. Було розглянуто всі основні моделі здійснено їх порівняння за їх перевагами та недоліками. Було проаналізовано процеси навчання та оптимізації навчання нейронних мереж. Було досліджено формування архітектури ЗНМ та запропоновано підхід.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

3.1 Вибір мови програмування для реалізації класифікатора зображень

C# — сучасна об'єктно-орієнтована мова програмування, розроблена в 2000 році Андерсом Хейлсбергом у Microsoft як конкурент Java (на яку вона дуже схожа). Він був створений тому, що Sun (пізніше придбана Oracle) не хотіла, щоб Microsoft вносила зміни в Java , тому Microsoft вирішила створити власну мову. C# швидко виріс з моменту його створення, завдяки широкій підтримці з боку Microsoft, що допомогло йому отримати велику кількість прихильників; зараз це одна з найпопулярніших мов програмування у світі.

Це мова загального призначення, розроблена для розробки додатків на платформі Microsoft, і для роботи потрібна платформа .NET в Windows. C# часто розглядають як гібрид, який використовує найкраще з C і C++ для створення справді модернізованої мови. Хоча платформа .NET підтримує кілька інших мов кодування, C# швидко став однією з найпопулярніших.

C# можна використовувати для створення майже чого завгодно, але він особливо сильний у створенні настільних програм та ігор Windows. C# також можна використовувати для розробки веб-додатків і стає все більш популярним і для мобільної розробки . Міжплатформні інструменти, такі як Xamarin, дозволяють використовувати програми, написані на C#, майже на будь-якому мобільному пристрої.

Microsoft створила C# для Microsoft. Отже, немає сумніву, чому він популярний у створенні додатків Windows. Це робить процес розробки гладким, а такі функції, як збирання сміття C#, працюють чудово.

Крім того, розробники можуть розраховувати на підтримку спільноти та документацію щодо розробки додатків і програм, які є специфічними для архітектури платформи Microsoft.

C# широко використовується для створення ігор за допомогою ігрового движка Unity , який є найпопулярнішим ігровим движком сьогодні. Більше третини найкращих ігор створено за допомогою Unity, і існує приблизно 770 мільйонів активних користувачів ігор, створених за допомогою двигуна Unity. Unity також використовується для віртуальної реальності: 90% усіх ігор Samsung Gear і 53% усіх ігор Oculus Rift VR розроблено за допомогою Unity.

C# є дуже популярним інструментом для створення цих програм, тому він є чудовим вибором для будь-якого програміста, який сподівається проникнути в індустрію розробки ігор, або для тих, хто цікавиться віртуальною реальністю.

Будучи потужним, гнучким і добре підтримуваним, C# швидко став однією з найпопулярніших доступних мов програмування. Сьогодні це четверта за популярністю мова програмування, приблизно 31% усіх розробників використовують її регулярно.

C# має кілька переваг програмування. Одна з цих переваг пов’язана зі «збиранням сміття», а така потреба є дуже висока у більшості програм. У деяких програмах, таких як C++, програміст повинен постійно знати про пам'ять і виділяти її або звільняти за потреби. C# займається цією проблемою. Це допомагає оптимізувати код і удосконалювати та спрощувати обчислювальну складність та покращує швидкість роботи програми та сприяє зручності для розробки та задоволеність користувачів зроблених програм.

Ще однією перевагою є попередження компілятора. C++ дозволяє програмістам вносити будь-які зміни в систему, якщо вони сформулювали її, використовуючи правильний синтаксис програмування. Тому цілком можливо, що програміст може ненавмисно створити щось, що пошкодить

внутрішню операційну систему або частину базового програмування комп’ютера. Для порівняння, C# має певні запобіжні заходи, щоб уникнути цього.

Таблиця 4.1 – Порівняння мов програмування

Можливості	C#	C++	Java
Імперативність	+	+	+
Об’єктна-орієнтованість	+	+/-	+
Функціональність	+	+/-	+/-
Рефлексивність	+/-	+/-	+/-
Декларативність	+/-	-	-
Інтерпретованість командного рядка	+	+/-	-
Ручного керування пам'яттю	+	+	-

Для розробки програмного забезпечення було обрано C#, так як вона по більшості параметрів можливості підходить для найоптимальнішого вирішення поставлених задач.

Переваги C# при реалізації системи:

- Мова програмування C # претендує на справжню об’єктну орієнтованість.
- Принципово особливо важливою ознакою відмінності від попередників у мові C# є те що у першу чергу вона орієнтується на безпеку коду.
- C# це «рідною» мова програмування для розробки ПЗ в середовищі розробки Microsoft .NET, так як найтісніше інтегрована з ним.

3.2 Вибір середовища програмування для класифікатора зображень на основі згорткових нейронних мереж

SharpDevelop (також оформлений як #develop) — це безкоштовне інтегроване середовище розробки з відкритим кодом (IDE) для платформ .NET Framework, Mono, Gtk# і Glade#. Він підтримує розробку на мовах програмування C#, Visual Basic .NET, Boo, F#, IronPython та IronRuby .

Він був розроблений як безкоштовна і легка альтернатива Microsoft Visual Studio і містить еквівалентну функцію майже для кожної важливої функції Visual Studio Express, включаючи функції для керування проектами, редагування коду, компіляції та налагодження програм. Щоб полегшити міграцію проекту, SharpDevelop працює з проектами Visual Studio і файлами коду. Він здатний компілювати програми для .NET Framework версії 2.0, 3.0, 3.5, 4.0 і .NET Compact Framework 2.0 і 3.5.

Незважаючи на те, що SharpDevelop не настільки поширений, як лінійка продуктів Visual Studio, SharpDevelop є досить популярним і був завантажений щонайменше 8 мільйонів разів у всьому світі, і це було задокументовано в книзі Dissecting a C# Application: Inside SharpDevelop (2003), опублікованій її творцями.

SharpDevelop повністю написаний на C# і складається з близько 20 компонентів, які інтегруються для формування програми. Компонент редактора вихідного коду відомий як AvalonEdit і може використовуватися іншими програмами. На початку свого розвитку проект був розділений для розробки Mono і Gtk# на проект MonoDevelop.

Інші цікаві особливості:

- Мови програмування, які також підтримуються: C, ASP.NET, ADO.NET, XML, HTML.

-Висвітлення синтаксису для C, HTML, ASP, ASP.NET, VBScript, VB.NET і XML.

- Конвертер для C у VB.NET і навпаки

- Конструктор форм для C, Visual Basic.NET і Boo.

-І багато іншого.

SharpDevelop сумісний з Visual Studio Express і Visual Studio 2005, оскільки використовує той самий тип формату для файлів проекту та вихідного коду.

Microsoft Visual Studio — це IDE, створена Microsoft і використовується для різних типів розробки програмного забезпечення, наприклад комп'ютерних програм, веб-сайтів, веб-програм, веб-сервісів та мобільних додатків. Він містить інструменти завершення, компілятори та інші функції для полегшення процесу розробки програмного забезпечення.

Visual Studio IDE (інтегроване середовище розробки) — це програмна програма для розробників для написання та редагування свого коду. Його інтерфейс користувача використовується для розробки програмного забезпечення для редагування, налагодження та створення коду. Visual Studio включає редактор коду, який підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Інтегрований налагоджувач працює і як налагоджувач на рівні джерела, і як налагоджувач на рівні машини. Інші вбудовані інструменти включають профайлер коду, конструктор для створення додатків із графічним інтерфейсом користувача, веб-дизайнер, конструктор класів і дизайнер схем баз даних.

Visual Studio надає передові інструменти та технології для створення програм, які використовують переваги новітніх можливостей платформи, будь то Windows, Android, iOS або Linux. Visual Studio 2019 також орієнтована на попередні платформи, тому ви можете створювати нові програми або модернізувати існуючі програми, які виконуються в попередніх версіях Windows, одночасно використовуючи розширені інструменти розробки, забезпечення якості та можливості спільної роботи в Visual Studio 2019.

Переваги Visual Studio

-Для C++ це одна з найкращих IDE і підтримує багато мов

-Багато розширень

-Має безкоштовну версію

-Використовується багатьма розробниками

Найбільша відмінність між цими середовищами спостерігається при налагодженні помилкової логіки програми. Поки що Sharp Develop відсутній такий потужний вбудований налагоджувач, як Visual Studio.

Порівнюючи середовища Visual Studio, та Sharp Develop, було вирішено що Visual Studio 2019 більш комфортне сукупністю потрібних інструментів, а також воно не має свого кінця у своїй привабливості для розробників тому що воно часто оновлюється і розробка ПЗ у такому середовищі відбувається швидше та без витрачання додаткових зусиль у пошуках серед сторонніх ресурсів функцій, яких могло б не вистачати у середовищі.

3.3 Реалізація нейронної мережі в C#

Класи, які підтримують нейромережеве програмування, визначені в просторі імен Aforge.NET, яке містить різні типи, що дозволяють створювати нейромережеві програми.

AForge.NET — це бібліотека з відкритим вихідним кодом, розроблена мовою C#, яка використовується розробниками та дослідниками при вирішенні завдань, пов'язаних із комп'ютерним зором. А також у бібліотеці AForge.NET є можливості для вирішення завдань у галузі штучного інтелекту. Діапазон засобів, що застосовуються бібліотекою, досить різноманітний: обробка зображень, нейронні мережі, генетичні алгоритми, нечітка логіка, машинне навчання, робототехніка та багато іншого. [10].

Фреймворк складається з набору бібліотек і прикладів додатків, які демонструють їх можливості:

- Aforge.Imaging – бібліотека з процедурами обробки зображень і фільтрами;
- Aforge.Vision – бібліотека комп’ютерного зору;
- Aforge.Video – набір бібліотек для обробки відео;
- Aforge.Neuro – бібліотека обчислень нейронних мереж;
- Aforge.Genetic – бібліотека еволюційного програмування;
- Aforge.Fuzzy – бібліотека нечітких обчислень;
- Aforge.Robotics – бібліотека, яка забезпечує підтримку деяких комплектів робототехніки;
- Aforge.MachineLearning – бібліотека машинного навчання; і т.п.

Технологія комп'ютерного зору використовується у різних системах, пов'язаних зі зчитуванням та інтерпретацією візуальної інформації в оброблювані комп'ютером команди та дані. Для практичної реалізації подібних систем можна використовувати різні бібліотеки комп'ютерного зору. Найбільш поширеними та функціональними з них є бібліотека з відкритим вихідним кодом AForge.NET.

Бібліотека Aforge.NET являє собою набір функцій для завдання практично будь-якої складності що стосується згорткових нейронних мереж, включаючи навчання та тестування ЗНМ, імпорту, експорту у мережі. Є велика сукупність прикладів розроблених додатків, які можуть навчати і тестувати згорткові нейронні мережі що дозволяє швидко і просто реалізувати програмний задум, який являє собою програму, яка втілює в життя прототип для навчання, тестування і візуалізації результатів роботи ЗНМ. Програмна система, що представляє собою експериментальний стенд для тестування роботи згорткових нейронних мереж, що мають певні архітектурні параметри.

До цієї бібліотеки є актуальна HTML-документація, яка може допомогти розробникам-початківцям при її використанні у своїх проектах.

Однією з переваг є наявність власної спільноти, учасники якої можуть відповісти на питання, що стосуються роботи з цією бібліотекою, а також діляться своїми напрацюваннями з практичного застосування та модернізації різних алгоритмів та функцій бібліотеки.

Бібліотека AForge.NET постійно покращується та розвивається, розробляються нові методи, функції тощо, які розширюють її можливості.

Параметри навчання мережі задаються окремо в змінній trainer. У наведеній системі використовується алгоритм навчання Adadelta.

Опишемо основні кроки реалізації програмного модуля:

Спочатку завантажуємо модель за допомогою конструктора. Оскільки завантаження мільйонів параметрів з файлу моделі може зайняти деякий час, ми викликаємо конструктор в окремий потік, щоб не блокувати інтерфейс користувача (рисунок 3.1):

Розроблена програма завантажує шари в пам’ять як ланцюг шарів. Ланцюг – це пов’язаний список, перший і останній вузли якого – Input та Ouput. Щоб класифікувати зображення, воно встановлюється як вхідне, і шари перетворюються на виклик функції feedNext () для подачі наступного шару на кожному кроці. Коли дані надходять до шару Ouput, вони знаходяться у формі вектора ймовірності. Виклик getDecision () сортує ймовірності від найвищої до найнижчої, і тоді ми можемо розглядати кожну ймовірність як відповідність. Важливо зробити ці виклики всередині потоку знову, щоб не блокувати інтерфейс користувача. Крім того, оскільки потік не може змінювати елементи інтерфейсу, потоки повинні викликати коди, що змінюють елементи інтерфейсу (додавання рядків до lv_KeywordList, оновлення ddLabel.Text), і викликати потік GUI (рисунок 3.3).

```
Thread t = new Thread(() =>
{
    try
    {
        cnn = new CNN(«imagenet-matconvnet-vgg-f.cenin»);
        ddLabel.Invoke((MethodInvoker)delegate ()
        {
            cbClasses.Items.AddRange(cnn.outputLayer.classes);
            dropToStart();
        });
    }
    catch (Exception exp)
    {
        ddLabel.Invoke((MethodInvoker)delegate ()
        {
            ddLabel.Text = «Missing model file!»;
            if (MessageBox.Show(this, «Couldn’t find model file.
                Do you want to be redirected to download page?», «Missing Model File»,
                MessageBoxButtons.YesNo,MessageBoxIcon.Error) == DialogResult.Yes)
                Process.Start(«http://huseyinatasoy.com/y.php?bid=71»);
        });
    }
});
t.Start();

```

Рисунок 3.1 - Викликання інтерфейсу користувача

Також нам потрібна структура для збереження результатів (рисунок 3.2):

```

private struct Match
{
    public int ImageIndex { set; get; }
    public string Keywords { set; get; }
    public float Probability { set; get; }
    public string ImageName { set; get; }
    public Match(int imageIndex, string keywords, float probability, string imageName)
    {
        ImageIndex = imageIndex;
        Keywords = keywords;
        Probability = probability;
        ImageName = imageName;
    }
}
```

Рисунок 3.2 - Структура для збереження результатів

```
Thread t = new Thread(() =>
{
    int imCount = imageFullPaths.Length;
    for (int j = 0; j < imCount; j++)
    {
        Bitmap b = (Bitmap)Image.FromFile(imageFullPaths[j]);
        ddLabel.Invoke((Action<int,int>)delegate (int y, int n)
        {
            ddLabel.Text = «Processing [« + (y + 1) + «/» + n + «]…\n\n» +
                           getImageName(imageFullPaths[y]);
        }, j, imCount);
        Application.DoEvents();
        cnn.inputLayer.setInput(b, Input.ResizingMethod.ZeroPad);
        b.Dispose();
        Layer currentLayer = cnn.inputLayer;
        while (currentLayer.nextLayer != null)
        {
            currentLayer.feedNext();
            currentLayer = currentLayer.nextLayer;
        }
        Output outputLayer = (Output)currentLayer;
        outputLayer.getDecision();
        lv_KeywordList.Invoke((MethodInvoker)delegate ()
        {
            int k = 0;
            while (outputLayer.probabilities[k] > 0.05)
            {
                Match m = new Match(
                    j,
                    outputLayer.sortedClasses[k],
                    (float)Math.Round(outputLayer.probabilities[k], 3),
                    getImageName(imageFullPaths[j])
                );
                matches.Add(m);
                k++;
            }
        });
    }
```

Рисунок 3.3 - сортування ймовірності, виклик GUI

Тепер усі зображення позначені ключовими словами, які фактично є описом класу моделі, яку ми використовуємо. Нарешті, ми повторюємо Match, щоб знайти кожну відповідність, що містить ключове слово, написане користувачем (рисунок 3.3):

```
    lv_KeywordList.Invoke((MethodInvoker)delegate ()
    {
        groupBox2.Enabled = true;
        btnFilter.PerformClick();
        int k;
        for (k = 0; k < lv_KeywordList.Columns.Count - 1; k++)
            if(k!=1)
              lv_KeywordList.Columns[k].Width = -2;
        lv_KeywordList.Columns[k].Width = -1;
        dropToStart();
    });
});
t.Start();
float probThresh = (float)numericUpDown1.Value;
string str = cbClasses.Text.ToLower();
lv_KeywordList.Items.Clear();
pictureBox1.Image = null;
List<int> imagesToShow = new List<int>();
int j = 0;
bool stringFilter = (str != «»);
for (int i = 0; i < matches.Count; i++)
{
    bool cond = (matches[i].Probability >= probThresh);
    if (stringFilter)
        cond = cond && matches[i].Keywords.Contains(str);
    if (cond)
    {
        addMatchToList(j, matches[i]);
        int ind = matches[i].ImageIndex;
        if (!imagesToShow.Contains(ind))
            imagesToShow.Add(ind);
        j++;
    }
}
if (lv_KeywordList.Items.Count > 0)
    lv_KeywordList.Items[0].Selected = true;
```

Рисунок 3.4 - Знаходження відповідності за ключевим словом

3.4 Розробка інтерфейсу класифікатора зображень

При розробці інтерфейсу було розроблено клас MainForm і створено інтерфейс Fount. Також під час розробки інтерфейсу програмного модуля було використано стандартну бібліотеку MetroFramework.Forms.

Клас MainForm – це клас що відповідає за інтерфейс інтелектуального модуля та основних елементів функціонування модуля, а конкретно за ініціалізацію додатку та функціонування основних елементів інтерфейсу модуля класифікації зображень.

При розробці модуля було використано Windows Forms.

Windows Forms дозволяє розробляти інтелектуальні клієнти. Інтелектуальний клієнт – це програма з повнофункціональним графічним інтерфейсом, просте в розгортанні і оновленні, здатне працювати при наявності або відсутності підключення до Інтернету і використовує більш безпечний доступ до ресурсів на локальному комп'ютері в порівнянні з традиційними додатками Windows [9].

Windows Forms – це технологія інтелектуальних клієнтів для .NET Framework. Вона являє собою набір керованих бібліотек, що спрощують виконання стандартних завдань, таких як читання з файлової системи і запис в неї. За допомогою середовища розробки типу Visual Studio, було створено програму з використанням Windows Forms, які відображають інформацію, запитують введення інформації від користувача У програмі, форма – це видима поверхня, на якій виводиться інформація для користувача.

Елемент управління – це окремий елемент призначений для користувача інтерфейсу, призначений для відображення та введення даних.

При виконанні користувачем якої-небудь дії з формою або одним з її елементів управління створюється подія. Програма реагує на ці події за допомогою коду і обробляє події при їх виникненні [9].

Windows Forms включає широкий набір елементів управління, які можна додавати на форми: текстові поля, кнопки, списки, що розкриваються, перемикачі та навіть веб-сторінки [11].

Завдяки підтримці перетягування конструктора Windows Forms в Visual Studio можна легко створювати додаток Windows Forms для аналізу фондового ринку акцій. Досить виділити елемент керування курсором і помістити його в потрібне місце на формі. Для подолання труднощів,

пов’язаних з вирівнюванням елементів управління, конструктор надає такі елементи управління, як лінії сітки і лінії прив’язки. І при використанні Visual Studio, і при компіляції з командного рядка ви можете використовувати елементи керування FlowLayoutPanel, TableLayoutPanel і SplitContainer для створення складних макетів форм за менший час [11].

Оброблювач подій – це процедура в коді, що визначає, які дії повинні виконуватися при виникненні тих чи інших подій, наприклад, якщо користувач натискає кнопку то повідомлення надходить в чергу. При породженні події запускається обробник або кілька обробників. Події можуть призначатися відразу декільком обробникам, а методи, які керують конкретними подіями, можна змінювати динамічно.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМИ КЛАСИФІКАТОРА ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

4.1 Тестування аналіз та порівняння результатів розробленої програми з результатами аналогу

Основною задачею даного тестування є перевірка того, чи підвищилась точність класифікації зображень. Для тестування виконаємо класифікацію семи зображень для розробленої програми (рисунок 4.1).

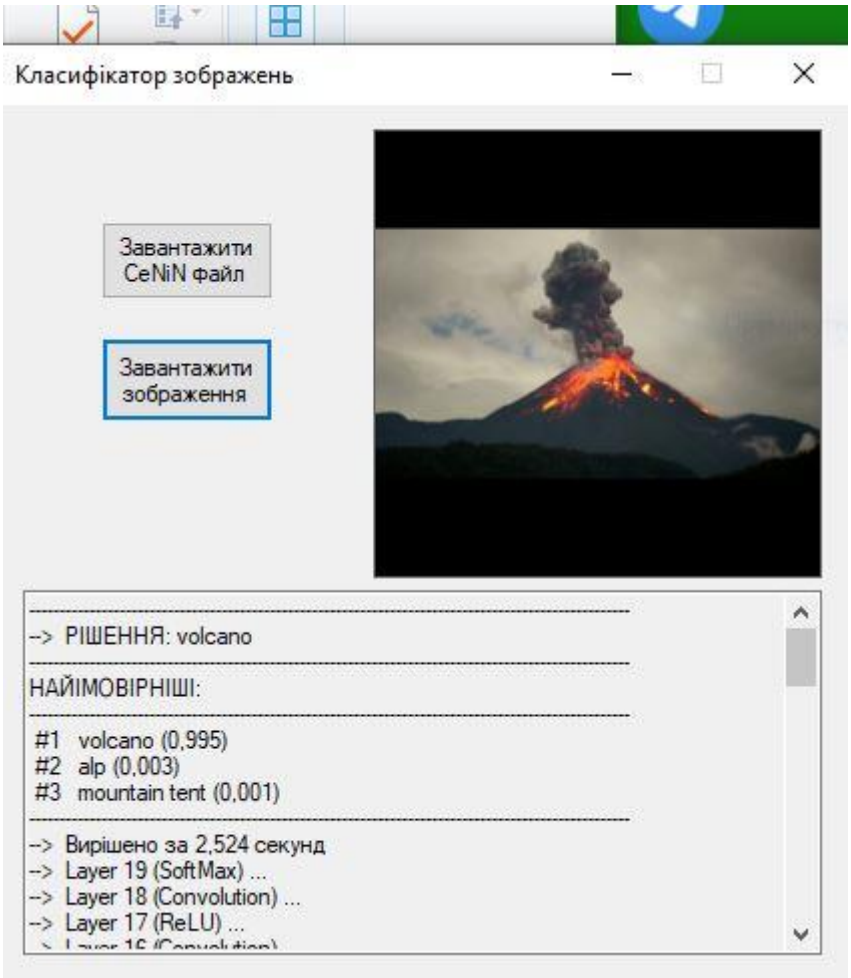


Рисунок 4.1– Результат роботи класифікатора зображень

Для того щоб виміряти час використаємо стандартний клас Stopwatch, тому що він дозволяє більш точно виміряти час роботи модуля.

Код, що використовує клас Stopwatch для виміру часу виконання модуля має вигляд:

```
var sw = Stopwatch.StartNew();
SomeOperation();
sw.Stop();
```

Виходячи із результатів роботи програми, бачимо що точність дорівнює 0.806, а швидкодія (час розпізнавання) становить 1,374 секунд. Ці показники є вищими ніж в аналогічних програмних модулях, що свідчить про підвищення точності та швидкодії класифікації. Для доведення цього було проведено подібні експерименти по розпізнаванню 7 різних зображень розробленим модулем та програмою-аналогом ResNeXt. Результати експериментів занесено до таблиці 4.1.

Таблиця 4.1 – Порівняння результатів класифікації зображень розробленим модулем та програмою-аналогом

	Програма ResNeXt	Розроблений програмний засіб	Номер зображення
Точність класифікації (ймовірність належності вхідного зображення визначеному класу)	0.588	0.668	1
	0.812	0.943	2
	0.227	0.233	3
	0.975	0.988	4
	0.936	0.989	5
	0.785	0.989	6
	0.817	0.912	7
Середня точність класифікації	0,73	0,80	

Швидкодія	1,651 с.	1,437 с.	1
	1,723 с.	1,457 с.	2
	1,689 с.	1,391 с.	3
	1,675 с.	1,439 с.	4
	1,586 с.	1,429 с.	5
	1,579 с.	1,411 с.	6
	1,678 с.	1,433 с.	7
Середня швидкодія	1,615 с.	1,432 с.	

Із табл. 4.1 видно, що розроблена програма має середню точність класифікації зображень 0,80 (тобто 80%), а аналогічна програма - 0,73 (73%), тобто розроблена програма має на 7% вищу середню точність класифікації. Також із табл. 4.1 видно, що розроблена програма має середню швидкодію 1,432 с., а аналогічна програма - 1,615 с., тобто має на 13% (((1,615-1,432)/1,432)*100%=13%) вищу середню швидкодію. Тобто мета роботи досягнута – точність та швидкодія класифікації зображень підвищена у розробленому програмному засобі.

Тестування розробленої програми для класифікації зображень показало її надійну роботу. Програма повністю відповідає технічному завданню.

4.2 Висновки

Тестування розробленої програми для класифікації зображень показало її надійну роботу. Програма повністю відповідає технічному завданню.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Для проведення технологічного аудиту з метою оцінювання комерційного потенціалу розробки було залучено 3-х незалежних експертів. Такими експертами будуть Войтко В. В. (Експерт 1), Бабюк Н. П. (Експерт 2) та Майданюк В. П. (Експерт 3). Оцінка комерційного потенціалу розробки проводиться за 12-ма критеріями за 5-ти бальною шкалою [35]. Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Бали, виставлені		
	експертом 1	експертом 2	експертом 3
1	4	4	4
2	3	3	3
3	3	4	3
4	4	3	3
5	3	3	4
6	4	4	3
7	3	4	3
8	4	4	4
9	4	4	4
10	4	4	4
11	4	4	4
12	4	4	4
Сума балів	$СБ_1$=4	$СБ_2$=4	$СБ_3$=4

	4	5	3
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{СБ_1 + СБ_2 + СБ_3}{3} = 44$		

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу. Цей рівень досягається завдяки зростанню продуктивності, ефективності нової науково-технічної розробки.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи

Для розробки нового програмного продукту необхідні певні витрати.

Основна заробітна плата розробників визначається за формулою (5.1)

$$З_о = \frac{M}{T_Р} \cdot t, \qquad (5.1)$$

де M – місячний посадовий оклад конкретного розробника, $T_Р$ – кількість робочих днів у місяці (вважаємо, що $T_Р = 21$ день), t – кількість днів роботи. Дана формула використовується для знаходження об’єму витрат на оплату праці, результати обчислень наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Працівник	Оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на оплату праці, грн.
Науковий керівник	12 000	571,43	10	5714,29
Інженер-програміст	000	285,71	45	12857,14

Всього	18571,43

Додаткові витрати на оплату праці становлять 10% від основних витрат на оплату праці, тобто

$$З_{дод} = 0{,}1 \cdot 18571{,}43 = 1857{,}14 \text{ (грн.)}.$$

Єдиний соціальний внесок розраховується як 22% від суми основної та додаткової заробітної плати. Сума нарахувань становить

$$Н_{зп} = 0{,}22 \cdot (18571{,}43 + 1857{,}14) = 4494{,}29 \text{ (грн.)}.$$

Витрати на комплектуючі вироби $К_в$, які використовують при дослідженні нового технічного рішення, розраховуються, згідно з їхньою номенклатурою, за формулою

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j,$$

де $Н_j$ – кількість комплектуючих j-го виду, шт.; $Ц_j$ – покупна ціна комплектуючих j -го виду, грн; $К_j$ – коефіцієнт транспортних витрат, (встановимо однаковим для всіх комплектуючих та рівним 1,3). Отримані розрахунки зведено до таблиці 5.3.

Таблиця 5.3 – Витрати на комплектуючі

Найменування	Кількість, шт.	Ціна за штуку, грн.	Сума, грн.
Флешка	1	100	100

Пачка паперу	2	125	250
Ручка	1	25	25
Всього з урахуванням транспортних витрат			487,50

До статті «Амортизація обладнання» відносять амортизаційні відрахування по кожному виду обладнання для проведення науково-дослідної роботи, за його наявності в дослідній організації або на підприємстві.

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12},$$

де $Ц_б$ – балансова вартість обладнання, яке використовувалось для проведення досліджень, грн.; $t_{вик}$ – термін використання обладнання під час досліджень, місяців; $T_в$ – строк корисного використання обладнання, років.

Дослідження проводились на переносному персональному комп'ютері, вихідні дані для якого $Ц_б = 12000$ грн., $T_в = 2$ роки [35], $t_{вик} = 3$ місяці, тому амортизаційні відрахування становлять

$$A_{обл} = \frac{12000}{2} \cdot \frac{3}{12} = 1560{,}50 \text{ (грн.)}.$$

Витрати на силову електроенергію $B_е$ розрахуємо за формулою:

$$B_e = W_y \cdot t \cdot Ц_e \cdot K_{вп},$$

де W_y – встановлена потужність обладнання на певному етапі розробки, кВт; t – тривалість роботи обладнання на етапі дослідження, год; $Ц_e$ – вартість 1 кВт-години електроенергії, грн.; $K_{вп}$ – коефіцієнт, що враховує використання потужності, $K_{вп} < 1$. Вихідні дані: $W_y = 0{,}065$ кВт; $t = 720$ год; $Ц_e = 1{,}68$ грн./кВт; $K_{вп} = 0{,}8 < 1$. Витрати на силову електроенергію становлять

$$B_e = 0{,}065 \cdot 720 \cdot 1{,}68 \cdot 0{,}8 = 62{,}90 \text{ (грн.)}.$$

Інші витрати $I_в$ – витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Вони розраховуються як 50% від суми основної заробітної плати розробників і становлять

$$I_в = 0.5 \cdot 18571{,}43 = 9285{,}72 \text{ (грн.)}.$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат і становлять

$$B_{заг} = 18571{,}43 + 1857{,}14 + 4494{,}29 + 487{,}50 + 1560{,}50 + 62{,}90 + 9285{,}72 = 36319{,}48$$

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta},$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науководослідної роботи. На стадії впровадження $\eta = 0{,}9$. Загальні витрати становитимуть

$$ЗВ = \frac{36319{,}48}{0{,}9} = 40354{,}98 \text{ (грн.)}.$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Можливе збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П_i = (\Delta П_я \cdot N + П_я \cdot \Delta N)_i,$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження на підприємстві результатів науково-технічної розробки в аналізованому році; N – основний кількісний показник, який визначає обсяг діяльності підприємства у році до впровадження результатів нової науково-технічної розробки; $П_я$ – основний якісний показник, який визначає результати діяльності підприємства у кожному із років після впровадження науково-технічної розробки; ΔN – зміна основного кількісного показника діяльності підприємства в результаті впровадження науково-технічної розробки в аналізованому році [35].

У результаті впровадження результатів наукової розробки витрати на виготовлення програмного продукту зменшаться на 50 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 50 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 500 користувачів, протягом другого року – на 350 користувачів, протягом третього року – на 200 користувачів. Реалізація програмного продукту до впровадження результатів наукової розробки складала 3000 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 600 грн. Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Протягом перших трьох років:

$$\Delta П_1 = 50 \cdot 3000 + (600 + 50) \cdot 500 = 475000,$$
$$\Delta П_2 = 50 \cdot 3000 + (600 + 50) \cdot (500 + 350) = 702500,$$
$$\Delta П_3 = 50 \cdot 3000 + (600 + 50) \cdot (500 + 350 + 200) = 832500.$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їхньої окупності

Розрахуємо приведену вартість збільшення всіх чистих прибутків $ПП$, що їх може отримати розробник (замовник) від можливого впровадження науково-технічної розробки на власному підприємстві:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t},$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн; T – період часу, протягом якого очікується отримання позитивних

результатів від впровадження науково-технічної розробки, роки; τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні (встановимо $\tau = 0{,}05$); t – період часу (в роках) від моменту початку впровадження науковотехнічної розробки до моменту отримання підприємством збільшеної величини чистого прибутку в аналізованому році.

$$ПП = \frac{40354{,}98}{(1+0{,}05)^0} + \frac{475000}{(1+0{,}05)^1} + \frac{702500}{(1+0{,}05)^2} + \frac{832500}{(1+0{,}05)^3} = 1849068{,}94 \text{ (грн.)}.$$

Рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, як на рисунку 5.1.

Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Далі розраховують величину початкових інвестицій PV, які розробник (замовник) має вкласти для здійснення науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{розр} \cdot ЗВ,$$

де $k_{розр}$ – коефіцієнт, що враховує витрати розробника (замовника) на впровадження науково-технічної розробки (візьмемо $k_{розр} = 2$); $ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 \cdot 40354,98 = 80709,96 \text{ (грн.)}.$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для розробника (замовника) від можливого впровадження науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV,$$

де $ПП$ – приведена вартість збільшення всіх чистих прибутків від можливого впровадження науково-технічної розробки, грн.; PV – теперішня вартість початкових інвестицій, грн.

$$E_{абс} = 1849068,94 - 80709,96 = 1768358,98 \text{ (грн.)}.$$

Величина $E_{абс}$ має велике додатне значення. Це може свідчити про потенційну доцільність у впровадженні цієї науковотехнічної розробки. Але для остаточного прийняття рішення про впровадження цього недостатньо.

Для остаточного прийняття рішення в такому випадку необхідно розрахувати внутрішню економічну дохідність $E_в$ Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені розробником

(замовником) у впровадження науково-технічної розробки, розраховується за формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1,$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн; PV – теперішня вартість початкових інвестицій, грн; $T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_в = \sqrt[3]{1 + \frac{1768358,98}{80709,96}} - 1 = 1,84 \text{ або } 180\%.$$

Далі визначають бар'єрну ставку дисконтування $\tau_{мін}$. Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках (в 2021 році в Україні $d = 0,9$); f – показник, що характеризує ризикованість вкладення інвестицій (використаємо $f = 0,05$).

$$\tau_{мін} = 0,9 + 0,05 = 0,95 \text{ або } 95\%.$$

Оскільки величина $E_в > \tau_{мін}$, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки.

Далі розраховуємо період окупності інвестицій $T_{ок}$:

$$T_{ок} = \frac{1}{E_в},$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{1{,}84} = 0{,}54 \text{ роки}.$$

Оскільки $T_{ок}$ < 3-х років, то це свідчить про економічну ефективність впровадження науково-технічної розробки її розробником (замовником).

5.5 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки

за темою «Методи розпізнавання та класифікації зображень» становить 44 бали, і це свідчить про комерційну важливість проведення таких досліджень, тому що рівень комерційного потенціалу розробки високий. Також термін окупності становить 0,67 р., що менше 3-х років, що

свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-

дослідної роботи за темою «Методи та засоби розподілення даних між хмарними сховищами».

ВИСНОВКИ

У роботі було розглянуто задачу класифікації зображень за їх змістом. В ході аналізу предметної області розглянуто основні методи класифікації зображень та як найбільш перспективний, було обрано нейромережевий метод. Також було здійснено аналіз відомих програмних засобів класифікації зображень та як аналог до розроблюваного модуля було обрано програму ResNeXt-101. У другому розділі було обґрунтовано доцільність використання для даної задачі згорткової нейронної мережі. Було проаналізовано різні архітектури згорткових нейронних мереж та обгрунтовано вибір архітектури SSD. Проаналізовано процес навчання згорткових нейронних мереж та розроблено алгоритм формування архітектури згорткової нейронної мережі. Розроблено архітектуру обраного типу мережі, яка має 19 шарів та тисячу вихідних нейронів. Обґрунтовано вибір об’єктно-орієнтованої мови програмування високого рівня C# та середовища програмування MS Visual Studio. В результаті було спроектовано власну програму класифікації зображень з використанням бібліотеки Aforge.NET. Аналізи результатів тестування розробленої програми показали, що вона має середню точність класифікації зображень на 7% вищу за аналог. Також розроблена програма має середню швидкодію на 13% вищу за аналог. Тобто мета роботи досягнута – точність та швидкодія класифікації зображень підвищена у розробленому програмному засобі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Грабар С. А., Ліщинська Л. Б. Обґрунтування доцільності використання штучного інтелекту під час розпізнавання об’єктів у медицині на основі згорткових нейронних мереж. Електронні інформаційні ресурси: створення, використання, доступ: Збірник матеріалів Всеукраїнської науково-практичної Інтернет конференції (Суми/Вінниця, 9-10 листопада 2021). Суми/Вінниця: НІКО/ВНТУ, 2021. С.62-64

2. Колесницький О. К. Методичні вказівки до виконання курсової роботи з дисципліни «Нейромережеві методи обчислювального інтелекту» для студентів магістерської підготовки спеціальності 122 – „Комп’ютерні науки” для усіх форм навчання / О. К. Колесницький. – Вінниця: ВНТУ, 2018. – 38 с.

3. Месюра В.І. Основи проектування систем штучного інтелекту. Навчальний посібник / В.І. Месюра, Л.М. Ваховська. – В.:ВДТУ, 2000. – 96 с

4. Згорткова нейронна мережа [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Згорткова_нейронна_мережа

5. Глибокі нейронні мережі для вирішення завдань розпізнавання і класифікації зображення [Електронний ресурс]. URL: http://itcm.comp-sc.if.ua/2017/Sineglazov.pdf

6. Для чого потрібна згорткова нейронна мережа [Електронний ресурс]. URL: https://evergreens.com.ua/ua/articles/cnn.html

7. Модель обробки потокових даних для розпізнавання окремих зображень [Електронний ресурс]. URL: http://repository. kpi.kharkov.ua/handle/KhPI-Press/40715

8. Застосування згорткових нейронних мереж [Електроннийресурс]. URL: http://st.nmetau.edu.ua/journals/108/24_a_ua.167-171.pdf

9. Ріхтер Джефрі CLR via C #. Програмування на платформі Microsoft .NET Framework 4.5 мовою C #. — М.: Українська редакція, 2017. — 896 с.

10. Microsoft Visual Studio [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio

11. Visual Studio [Електронний ресурс]. URL: https://msdn. microsoft.com/uk-ua

12. Побудова Windows Forms. [Електронний ресурс]. URL: https://metanit.com/sharp/forms.php

13. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Computer_vision.

14. Benenson R. Ten Years of Pedestrian Detection, What Have We Learned? [Електронний ресурс]. URL: https: //rodrigob.github.io/documents/2014_eccvw_ten_years_of_pedestrian_detection_ with_supplementary_material.pdf.

15. Maenpaa T. The Local Binary Pattern Approach to Texture Analysis — Extensions and Applications. Oulu University Press, 2003.

16. Viola P. Rapid Object Detection using a Boosted Cascade of Simple Features [Електронний ресурс]. URL: https: //www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf.

17. Dalal N. Histograms of Oriented Gradients for Human Detection [Електронний ресурс]. URL: https: //lear.inrialpes.fr/pubs/2005/DT05/hog_cvpr2005.pdf.

18. Review: RetinaNet — Focal Loss (Object Detection) [Електронний ресурс]. URL: https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4.

19. Carandini M. Area V1 [Електронний ресурс]. URL: http://www:scholarpedia:org/article/Area_V.

20. Krizhevsky A. ImageNet Classification with Deep Convolutional Neural Networks [Електронний ресурс]. URL:https://papers.nips.cc/paper/4824imagenet-classification-with-deep-convolutional-neural-networks.pdf.

21. Ren S. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [Електронний ресурс]. URL: https://arxiv.org/pdf/1506.01497.

22. Girshick R. Fast R-CNN [Електронний ресурс]. URL:https://www.cvfoundation.org/openaccess/content_iccv_2015/papers/Girshick_Fast_RCNN_ICCV_2015_paper.pdf.

23. Redmon J. You Only Look Once: Unified, Real-Time Object Detection [Електронний ресурс]. URL:https://pjreddie.com/media/files/papers/yolo.pdf.

24. Liu W. SSD: Single Shot MultiBox Detector [Електронний ресурс]. URL:https://www.cs.unc.edu/~wliu/papers/ssd.pdf.

25. [Електронний ресурс]. URL: http: //host.robots.ox.ac.uk/pascal/VOC/voc2012/results/index.html.

26. Николенко С. Глубокое обучение. СПб.: Питер, 2018. 480 с.: ил. (Серия «Библиотека программиста»).

27. Гудфеллоу Я. Глубокое обучение / пер. с анг. А. А. Слинкина. 2-е изд., испр. М.: ДМК Пресс, 2018. 652 с.: цв. ил.

28. Dollar P. Pedestrian Detection: An Evaluation of the State of the Art [Електронний ресурс]. URL : http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/files/ PAMI12pedestrians.pdf.

29. Yosinski J. How transferable are features in deep neural networks? [Електронний ресурс]. URL: https://arxiv.org/pdf/1411.1792.pdf.

30. Python [Електронний ресурс]. URL:https://www.python.org/

31. OpenCV [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/OpenCV.

32. [Електронний ресурс]. URL:https://keras.io/ (дата обращения: 02.03.2019).

33. [Электронный ресурс]. URL:https://www.tensorflow.org/ (дата обращения: 30.05.2019).

Додаток А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Завідувач кафедри пз

_____проф., д.т.н. Романюк О.Н.

«____»_____2021_р.

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

«РОЗРОБКА МЕТОДУ І ПРОГРАМНОГО ЗАСОБУ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ»

Керівник роботи:

_____проф., д.т.н. Л. Б. Ліщинська

«____» _____2021р.

Виконав:

_____студент гр. 2ПІ-20М С. А. Грабар

«____» _____2021р.

Вінниця ВНТУ 2021

1. Найменування і область використання

Робота має назву «Програмний модуль класифікації зображень на основі згорткової нейронної мережі». Програма, що розробляється відноситься до класу програм, які забезпечують розв'язання задачі розпізнавання образів. Розроблену програму можна застосувати у пошукових системах Інтернет, у системах штучного інтелекту та Data mining, що працюють з зображеннями та їх контентом.

2. Підстави для розробки

Підставою для розробки даної магістерської роботи є наказ ВНТУ №___
від "___" _____ 20__р. та рішення засідання кафедри комп'ютерних наук (протокол №__ від "__"_____ 20__ року).

3. Мета та призначення розробки

Магістерська робота є навчальною і виконується для отримання практичних навичок в створенні нейрокомп'ютерних систем штучного інтелекту.

Метою розробки є спеціалізована програма, яка буде класифікувати зображення за їх контентом на основі нейронних мереж.

4 Джерела розробки

Серед джерел, які були використані в ході розробки магістерської роботи є Державний стандарт України (ДСТУ) 2481-94 «Системи оброблення інформації. Інтелектуальні інформаційні технології. Терміни та визначення», ДСТУ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і

правила оформлення», ДСТУ 2941-64 «Системи обробки інформації. Розроблення систем. Терміни та визначення», а також список використаних джерел.

5 Технічні вимоги

5.1 Вимоги до функціональних характеристик:

- вид вхідного сигналу – файл зображення,
- вид вихідного сигналу – номер та назва класу, до якого відноситься вхідне зображення,
- кількість представників вибірки для навчання – 1154,
- кількість представників вибірки для тесту – 103,
- наочне відображення результатів роботи.

5.2 Вимоги до апаратного забезпечення:

Технічні вимоги які є обов’язковими для функціонування програми:

- IBM-сумісний комп’ютер;
- операційна система Microsoft Windows XX;
- оперативна пам’ять об’ємом не менше 512 Мб;
- жорсткий диск об'ємом більше 20 Гб;
- монітор з розширенням не менше 1024*768;

5.3 Вимоги до програмного забезпечення

Для роботи програми необхідна операційна система Windows або сумісна з нею. Обов’язкова наявність у операційній системі платформи .NetFramework 2.0. Для доповнення, або зміни певної частини програми необхідне середовище програмування Microsoft Visual Studio 2015, OpenCV.

5.4 Вимоги до уніфікації та стандартизації

Робота повинна відповідати ДСТУ, які представлені вище у пункті «Джерела розробки» та вимогам до магістерського проектування.

5.5 Вимоги до безпеки

В ході виконання, дослідження та розробки програмного комплексу повинні виконуватись безпечні для життя та діяльності людини заходи при роботі з персональними ЕОМ.

5.6 Умови експлуатації

Умови експлуатації повинні відповідати умовам організації роботи людини в офісних приміщеннях.

6 Стадії і етапи розробки

1. Ідентифікація проблеми.
2. Розробка структури системи.
3. Розробка алгоритму роботи програми.
4. Програмна реалізація модуля для класифікації зображень на основі згорткової нейронної мережі.
5. Тестування програмних засобів.

7 Порядок прийому та контролю

1. Рубіжний контроль.
2. Попередній захист.
3. Захист на ДЕК.

Додаток Б. Протокол перевірки навчальної (кваліфікаційної) роботи

Додаток В. Лістинг програмного додатку

```
using System;
using System.Windows.Forms;
using CeNiN;
using System.Drawing;


namespace CeNiN_CSharp_Example
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        DateTime dateTime;

        CNN cnn;

        private void button1_Click(object sender, EventArgs e)
        {
            OpenFileDialog opf = new OpenFileDialog();
            opf.Filter = "CeNiN файл|*.cenin";
            if (opf.ShowDialog() != DialogResult.OK) return;

            textBox1.Clear();
            prependLine("Парсинг CeNiN файла...");
            Application.DoEvents();
            tic();
            cnn = new CNN(opf.FileName);
            prependLine("Нейронная сеть успешно загружена за " + toc() + "
секунд.");
            prependLine(cnn.layerCount + "+2 слоя, "
                    + cnn.totalWeightCount + " ваги і"
                    + cnn.totalBiasCount + " базиси завантажувались "
                    + toc() + " секунд.");

            button2.Enabled = true;
        }

        private void button2_Click(object sender, EventArgs e)
        {
```

```
                OpenFileDialog opf = new OpenFileDialog();
                opf.Filter = "Файл зображення|*.bmp;*.jpeg;*.jpg;*.png";
                if (opf.ShowDialog() != DialogResult.OK) return;
                Bitmap b = new Bitmap(opf.FileName);
                cnn.inputLayer.setInput(b, Input.ResizingMethod.ZeroPad);
                pictureBox1.Image = (Image)cnn.inputLayer.ResizedInputBmp.Clone();
                tic();
                Layer currentLayer = cnn.inputLayer;
                int i = 0;
                while (currentLayer.nextLayer != null)
                {
                    if (i == 0)
                        prependLine("Завантаження даних...");
                    else
                        prependLine("Layer " + i + " (" + currentLayer.type + ") ...");

                    Application.DoEvents();

                    currentLayer.feedNext();
                    currentLayer = currentLayer.nextLayer;
                    i += 1;
                }

                Output outputLayer = (Output)currentLayer;
                prependLine("Вирішено за " + toc().ToString() + " секунд");

                string decision = outputLayer.getDecision();
                string hLine = new string('-', 100);
                prependLine(hLine, "");
                for (i = 2; i >= 0; i--)
                    prependLine(" #" + (i + 1) + "   " + outputLayer.sortedClasses[i] +
" (" + Math.Round(outputLayer.probabilities[i], 3) + ")", "");
                prependLine(hLine, "");
                prependLine("НАЙІМОВІРНІШІ: ", "");
                prependLine(hLine, "");
                prependLine("РІШЕННЯ: " + decision);
                prependLine(hLine, "");
            }

            private DateTime tic()
            {
                dateTime = DateTime.Now;
                return dateTime;
```

```
        }
        private double toc()
        {
            return Math.Round((DateTime.Now - dateTime).TotalSeconds, 3);
        }

        private void prependLine(string text, string prefix = "-->  ")
        {
            textBox1.Text = prefix + text + "\r\n" + textBox1.Text;
        }
    }
}

namespace NeuralNetworkNET.cpuCNN
{
    /// <inheritdoc cref="CpuDnn"/>
    public static partial class CpuDnn
    {
        /// <summary>
        /// Executes the forward pass on a max pooling layer with a 2x2 window and a stride of 2
        /// </summary>
        /// <param name="x">The input <see cref="Tensor"/> to pool</param>
        /// <param name="xInfo">The info on the input <see cref="Tensor"/></param>
        /// <param name="y">The resulting pooled <see cref="Tensor"/></param>
        /// <exception cref="ArgumentException">The size of one of the input <see cref="Tensor"/> instances isn't valid</exception>
        public static unsafe void PoolingForward(in Tensor x, in TensorInfo xInfo, in Tensor y)
        {
            int h = x.Entities, w = x.Length;
            if (h < 1 || w < 1) throw new ArgumentException("The input tensor
```

```
isn't valid");
                int
                    depth               =
xInfo.Channels,
                    imgSize = w % depth ==
0   ?   w   /   depth   :   throw   new
ArgumentException("Invalid depth parameter
for the input tensor", nameof(x)),
                    imgAxis             =
imgSize.IntegerSquare();  // Size of an edge
of one of the inner images per sample
                if (imgAxis * imgAxis !=
imgSize) throw new ArgumentException("The
size of the input tensor isn't valid",
nameof(x));
                int
                    poolAxis = imgAxis / 2
+ (imgAxis % 2 == 0 ? 0 : 1),
                    poolSize = poolAxis *
poolAxis,
                    poolFinalWidth = depth
* poolSize,
                    edge = imgAxis - 1;
                if       (!y.MatchShape(h,
poolFinalWidth))          throw          new
ArgumentException("The output tensor shape
isn't valid", nameof(y));


                // Pooling kernel
                float* px = x, py = y;
                void Kernel(int sample)
                {
                    int
                        sourceBaseOffset =
sample * w,
                        resultBaseOffset =
sample * poolFinalWidth;
                    for (int z = 0; z <
depth; z++)
                    {
                        int
                            sourceZOffset
= sourceBaseOffset + z * imgSize,
```

```
                                resultZOffset
= resultBaseOffset + z * poolSize,
                                c = 0;
                            for (int i = 0; i
< imgAxis; i += 2)
                            {
                                int

sourceIOffset = sourceZOffset + i * imgAxis,

resultXOffset   =   resultZOffset   +   c   *
poolAxis,
                                    r = 0;
                                if (i == edge)
                                {
                                    //    Last
row
                                    for (int j
= 0; j < imgAxis; j += 2)
                                    {
                                        float
max;
                                        if  (j
== w - 1) max = px[sourceIOffset + j]; //
Last column
                                        else
                                        {

float

left = px[sourceIOffset + j],

right = px[sourceIOffset + j + 1];

max = left > right ? left : right;
                                        }

py[resultXOffset + r++] = max;
                                    }
                                }
                                else
                                {
                                    int
```

```
sourceI_1Offset = sourceZOffset + (i + 1) *
imgAxis;
                                    for (int j
= 0; j < imgAxis; j += 2)
                                    {
                                        float
max;
                                        if  (j
== edge)
                                        {
                                            //
Last column

float

up = px[sourceIOffset + j],

down = px[sourceI_1Offset + j];

max = up > down ? up : down;
                                        }
                                        else
                                        {


float

upLeft = px[sourceIOffset + j],

upRight = px[sourceIOffset + j + 1],

downLeft = px[sourceI_1Offset + j],

downRight = px[sourceI_1Offset + j + 1],

maxUp = upLeft > upRight ? upLeft : upRight,

maxDown = downLeft > downRight ? downLeft :
downRight;

max = maxUp > maxDown ? maxUp : maxDown;
                                        }


py[resultXOffset + r++] = max;
                                    }
                                }
```

```
                        c++;
                    }
                }
            }
            Parallel.For(0, h, Ker).AssertCompleted();
        }

        /// <summary>
        /// Executes the backward pass on a max pooling layer with a 2x2 window and a stride of 2
        /// </summary>
        /// <param name="x">The original input <see cref="Tensor"/> used during the forward pass</param>
        /// <param name="xInfo">The info on the input <see cref="Tensor"/></param>
        /// <param name="dy">The output error for the current layer</param>
        /// <param name="dx">The resulting backpropagated error</param>
        /// <exception cref="ArgumentException">The size of one of the input <see cref="Tensor"/> instances isn't valid</exception>
        public static unsafe void PoolingBackward(in Tensor x, in TensorInfo xInfo, in Tensor dy, in Tensor dx)
        {
            // Prepare the resul
            if (!dx.MatchShape(x)) throw new ArgumentException("The result tensor must have the same shape as the input", nameof(dx));
            int n = x.Entities, l = x.Length;
            if (n < 1 || l < 1) throw new ArgumentException("The input tensor isn't valid");
            int
                depth =
```

```
xInfo.Channels,
                    imgSize = l % depth ==
0   ?   l   /   depth   :   throw   new
ArgumentException("Invalid depth parameter
for the input tensor", nameof(x)),
                    imgAxis             =
imgSize.IntegerSquare();  // Size of an edge
of one of the inner images per sample
                if (imgAxis * imgAxis !=
imgSize) throw new ArgumentException("The
size of the input tensor isn't valid",
nameof(x));
                int
                    poolAxis = imgAxis / 2
+ (imgAxis % 2 == 0 ? 0 : 1),
                    poolSize = poolAxis *
poolAxis,
                    poolFinalWidth = depth
* poolSize,
                    edge = imgAxis - 1;
                int
                    pn = dy.Entities,
                    pl = dy.Length;
                if (pn != n || pl !=
poolFinalWidth)          throw          new
ArgumentException("Invalid pooled tensor",
nameof(dy));


                // Pooling
                float* px = x, pdy = dy,
pdx = dx;
                void Ker(int sample)
                {
                    int
                        sourceBaseOffset =
sample * l,
                        resultBaseOffset =
sample * poolFinalWidth;
                    for (int z = 0; z <
depth; z++)
                    {
                        int
                            sourceZOffset
```

```
= sourceBaseOffset + z * imgSize,
                                resultZOffset
= resultBaseOffset + z * poolSize,
                                c = 0;
                            for (int i = 0; i
< imgAxis; i += 2)
                            {
                                int

sourceIOffset = sourceZOffset + i * imgAxis,

resultXOffset   =   resultZOffset   +   c   *
poolAxis,
                                    r = 0;
                                if (i == edge)
                                {
                                    //    Last
row
                                    for (int j
= 0; j < imgAxis; j += 2)
                                    {
                                        if  (j
== l - 1)
                                        {

pdx[sourceIOffset + j] = pdy[resultXOffset +
r++];
                                        }
                                        else
                                        {

float

left = px[sourceIOffset + j],

right = px[sourceIOffset + j + 1];
                                            if
(left > right)
                                            {

pdx[sourceIOffset + j] = pdy[resultXOffset +
r++];

pdx[sourceIOffset + j + 1] = 0;
```

```
                                        }

else
                                        {

pdx[sourceIOffset    +    j    +    1]    =
pdy[resultXOffset + r++];

pdx[sourceIOffset + j] = 0;
                                        }
                                    }
                                }
                            }
                            else
                            {
                                int
sourceI_1Offset = sourceZOffset + (i + 1) *
imgAxis;
                                for (int j
= 0; j < imgAxis; j += 2)
                                {
                                    if  (j
== edge)
                                    {
                                        //
Last column

float

up = px[sourceIOffset + j],

down = px[sourceI_1Offset + j];
                                        if
(up > down)
                                        {

pdx[sourceIOffset + j] = pdy[resultXOffset +
r++];

pdx[sourceI_1Offset + j] = 0;
                                        }


else
```

```
                                        {
pdx[sourceI_1Offset + j] = pdy[resultXOffset
+ r++];
pdx[sourceIOffset + j] = 0;
                                        }
                                    }
                                    else
                                    {
int offset = sourceIOffset + j;
float
max = px[offset],
next = px[sourceIOffset + j + 1];
                                        if
(next > max)
                                        {
max = next;
pdx[offset] = 0;
offset = sourceIOffset + j + 1;
                                        }
else pdx[sourceIOffset + j + 1] = 0;
next = px[sourceI_1Offset + j];
                                        if
(next > max)
                                        {
max = next;
pdx[offset] = 0;
offset = sourceI_1Offset + j;
                                        }
else pdx[sourceI_1Offset + j] = 0;
```

```
next = px[sourceI_1Offset + j + 1];
                                        if
(next > max)
                                        {

pdx[offset] = 0;

offset = sourceI_1Offset + j + 1;
                                        }

else pdx[sourceI_1Offset + j + 1] = 0;

pdx[offset] = pdy[resultXOffset + r++];
                                            }
                                    }
                                }
                                c++;
                            }
                        }
                    }
                    Parallel.For(0,         n,
Kernel).AssertCompleted();
                }
            }
        }
```

Додаток Г. Ілюстративні матеріали

Ілюстративний матеріал до захисту магістерської кваліфікаційної роботи

РОЗРОБКА МЕТОДУ І ПРОГРАМНОГО ЗАСОБУ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

РОЗРОБКА МЕТОДУ І ПРОГРАМНОГО ЗАСОБУ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Магістерська кваліфікацій робота
спеціальність 121 «Інженерія програмного забезпечення»

Виконав студент гр. 2ПІ-20М Грабар С.А.
Керівник: д.т.н., проф. Ліщинська Л.Б.

- Об’єкт - процес класифікації зображень.
- Предмет - програмні засоби класифікації зображень на основі нейронних мереж та точність їх роботи.
- Мета роботи – підвищення точності та швидкодії класифікації зображень за рахунок використання згорткової нейронної мережі.

АКТУАЛЬНІСТЬ

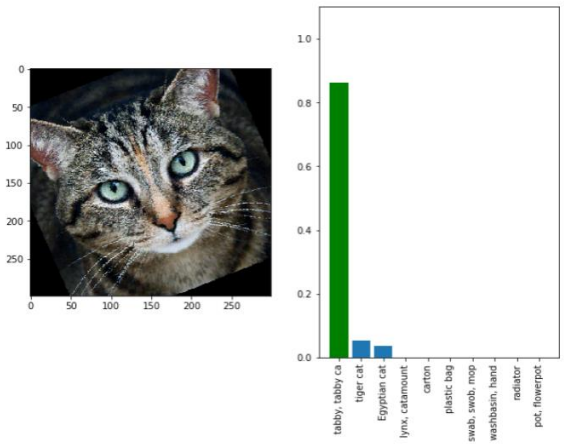
Актуальність теми магістерської роботи обумовлена застосуванням технології класифікації зображень у пошукових системах мережі Інтернет, у системах штучного інтелекту та Data Mining, що працюють із зображеннями та їх контентом.

Аналіз предметної області класифікації зображень

Основні методи класифікації:

- класифікація за допомогою дерев рішень;
- байєсівська (наївна) класифікація;
- класифікація методом опорних векторів;
- статистичні методи;
- класифікація за допомогою методу найближчого сусіда;
- класифікація CBR-методом (case based-reasoning);
- класифікація на основі нейронних мереж;

Було обрано метод на основі нейронних мереж



Вибір і обґрунтування аналогу

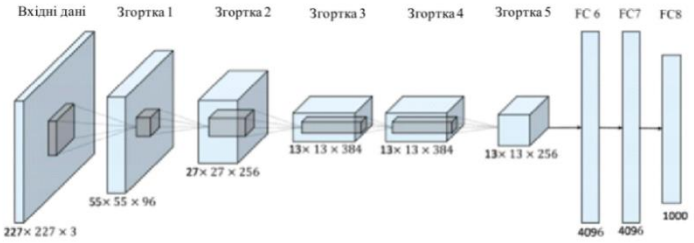
Програма ResNeXt. Принцип її роботи полягає у відкиданні зайвих векторів. Зайві – це ті, які не увійшли у взаємно знайдений кластер векторів.

Обґрунтування вибору згорткової нейронної мережі для розв'язання задачі класифікації зображень

Ідея даних моделей мотивована дослідженнями про зорову кору головного мозку.

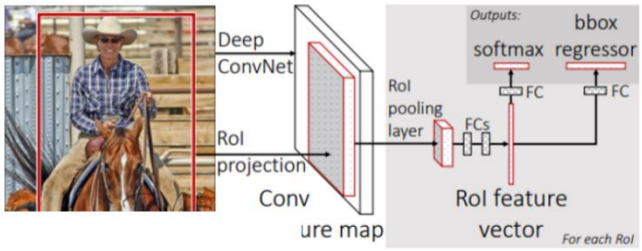
Обробка зображень в мозку влаштована як глибока нейронна мережа. Це наближення згорткових нейронних мереж до роботи зорової кори головного мозку дозволяє їм розпізнавати об'єкти на більш високому рівні абстракції з високою точністю

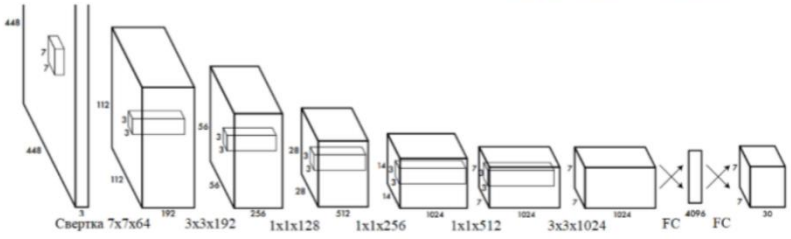
Обґрунтування вибору моделі згорткової нейронної мережі



AlexNet

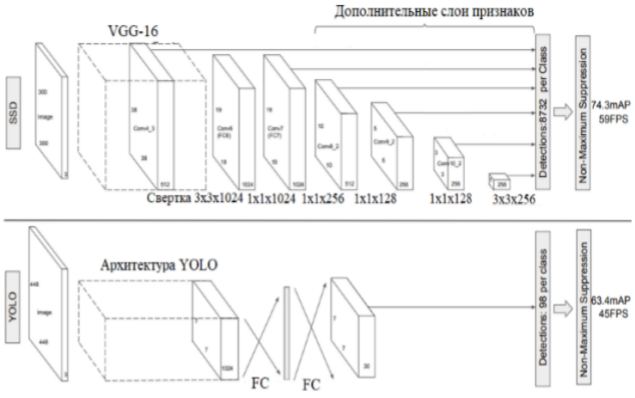
Fast R-CNN

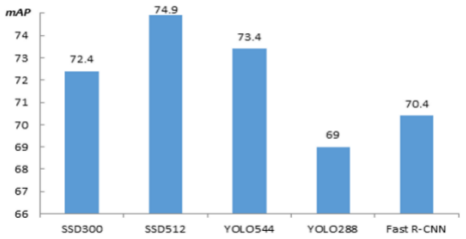




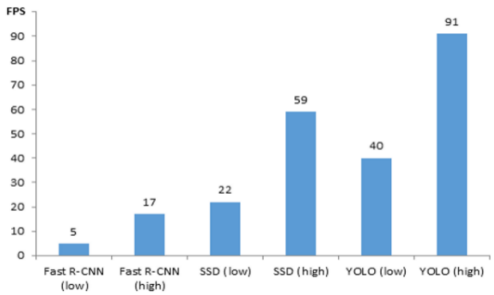
YOLO

Обґрунтування вибору моделі згорткової нейронної мережі





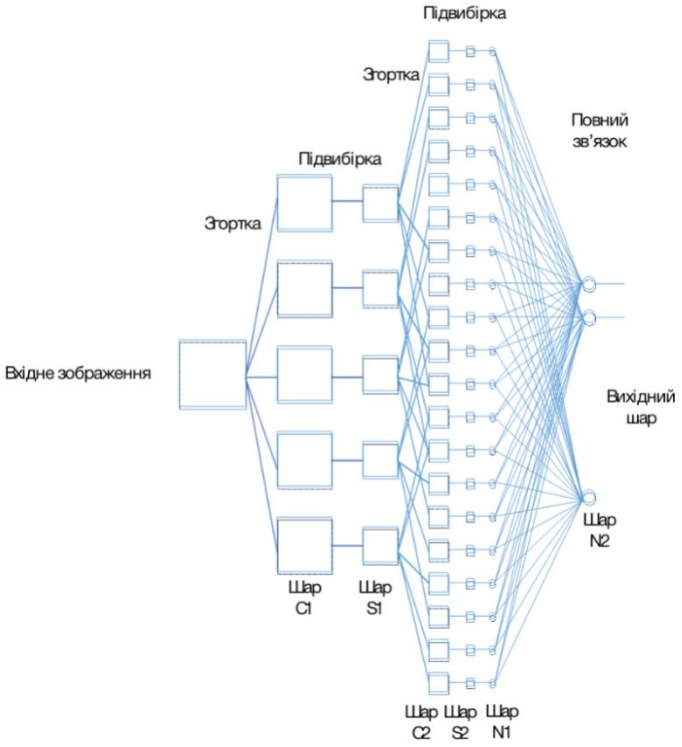
Порівняння точності різних моделей



Порівняння швидкодії різних моделей

Найбільшу продуктивність забезпечує модель YOLO, але вона має точність нижче в порівнянні з іншими моделями. Високу точність має модель R-CNN і її модифікації, але вона має дуже низьку продуктивність. Модель SSD має оптимальне співвідношення точності і продуктивності, тому ця модель взята за основу для розробки програми класифікації зображень

МАТЕМАТИЧНА МОДЕЛЬ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ



Функціонування нейрона згорткової площині описується виразом [10]:

$$y_k^{(i,j)} = f\left(\sum_{s=1}^{K}\sum_{t=1}^{K}\left(w_{k,s,t}x^{((i-1)*s,(j-1)*t)}\right) + b_k\right),$$

де $y_k^{(i,j)}$ - вихід нейрона з координатами (i, j) k-ої площині згорткового шару, b_k - нейрони зміщення k-ої площини, K - розмір рецептивного поля нейрона, $w_{k,s,t}$ - синаптична вага k-ої площини, для зв'язку з s-им, t- им значенням рецептивного поля, x - виходи нейронів попереднього шару.

Скільки карт містить згортковий шар C1, стільки буде здійснено згорток вхідного зображення. Розмір згорткової площині визначається наступними виразами:

$$w_c = w_{in} - K + 1,$$
$$h_c = h_{in} - K + 1,$$

де w_c і h_c - відповідно ширина і висота згорткової площини, w_{in} і h_{in} - відповідно ширина і висота площини, що сканується, K - розмір рецептивного поля.

Згортка можлива не тільки з одиничним кроком, при цьому:

$$w_c = (w_{in} - K)/step + 1,$$
$$h_c = (h_{in} - K)/step + 1,$$

де step - крок згортки.

Згортка з кроком, відмінним від одиничного, можлива тільки при певних співвідношеннях параметрів w_{in}, h_{in}, K і step. Крок не повинен перевищувати розмір рецептивного поля K і w_c і h_c повинні бути цілими числами. Така згортка може зменшити загальну кількість обчислень, зберігши при цьому прийнятний рівень сканування зображення.

Функціонування нейрона площини підвибірки описується виразом [15]:

$$y_k^{(i,j)} = f\left(\frac{1}{K^2}w_k\sum_{s=1}^{K}\sum_{t=1}^{K}\left(x^{((i-1)*K+s,(j-1)*K+t)}\right) + b_k\right),$$

Навчання згорткових нейронних мереж

Основне завдання в навчанні нейронних мереж - мінімізація заданої функції помилки [14]. Зазвичай, оптимізується апостеріорна імовірність.

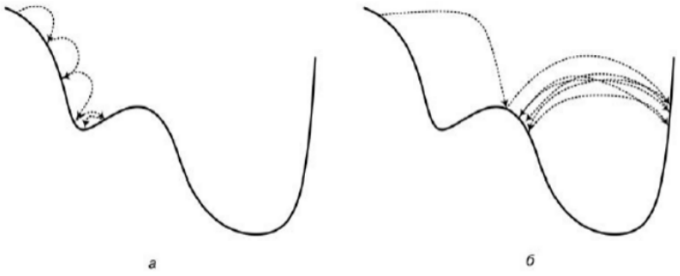
$$p(\theta|D) = p(\theta)\prod_{d\in D} p(d|\theta),$$

де функція правдоподібності $p(\theta)$ - помилка на навчальній вибірці, апріорне розподіл $p(d \mid \theta)$ — регуляризація.

Завдання оптимізації - по заданій функції знайти аргументи, в яких ця функція мінімізується. Функція помилки в нейронних мережах має багато локальних екстремумів. Для того, щоб знайти оптимальний екстремум, використовується евристичний метод оптимізації - градієнтний спуск. Представивши поверхню функції помилки, завдання оптимізації зводитися до обчислення градієнта. Якщо функцію, що визначає поверхню, позначити через

$E(\boldsymbol{\theta})=E(\theta_1, \theta_2,\ldots, \theta_n)$,

де $(\theta_1, \theta_2,\ldots, \theta_n)$ – параметри функції , то її градієнт ∇E – це вектор похідних функції декількох змінних по кожній з компонент

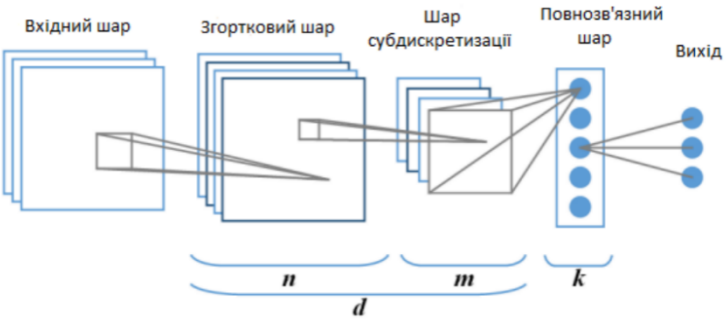


Проблеми зі швидкістю градієнтного спуску:
а) — занадто маленькі кроки,
б) — занадто великі кроки.

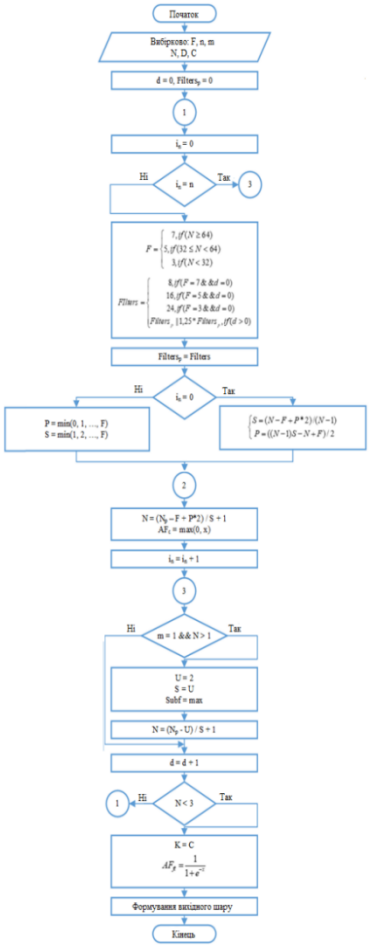
Для вирішення цих проблем використовується стохастичний градієнтний спуск - підрахунок помилки і оновлення ваг відбувається не після проходу всієї тренувальної множини, а після кожного прикладу:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta\nabla E(f(\mathbf{x}_t, \boldsymbol{\theta}_{t-1}), y_t)$$

СХЕМА АЛГОРИТМУ ФОРМУВАННЯ АРХІТЕКТУРИ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ



Шаблон архітектури згорткової мережі



Обгрунтування вибору мови та середовища програмування

Можливість	C#	C++	Java
Імперативна	+	+	+
Об'єктно- орієнтована	+	+/-	+
Функціональна	+	+/-	+/-
Рефлексивна	+/-	+/-	+/-
Декларативна	+/-	-	-
Інтерпретатор командного рядка	+	+/-	-
Ручне керування пам'ятю	+	+	-

Було обрано C#, так як вона найбільш відповідає поставленій задачі.

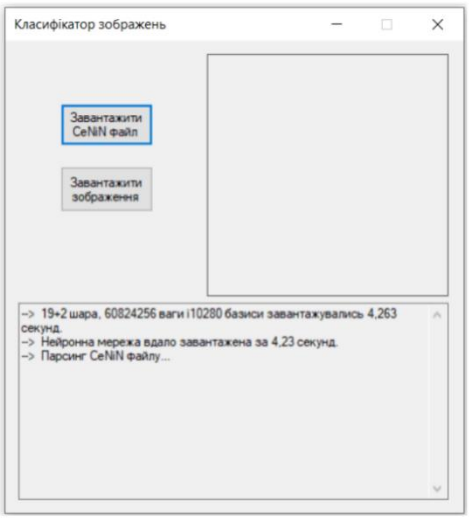
Також була використана бібліотека Aforge.NET,

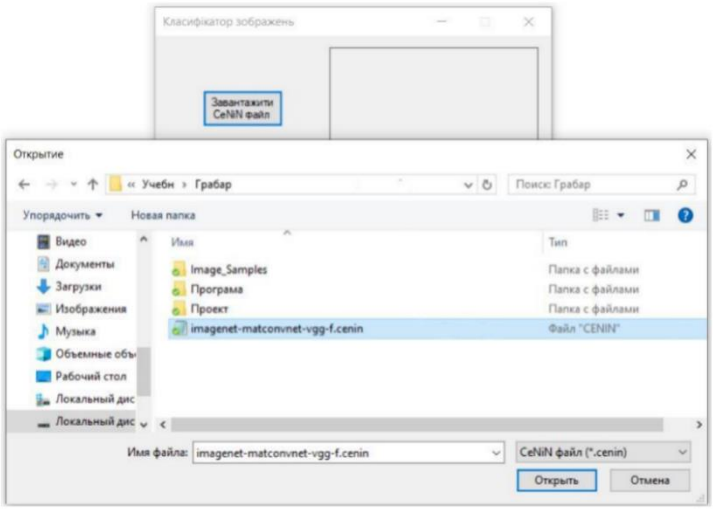
яка являє собою набір функцій для задання практично будь-яких конфігурацій згорткових нейронних мереж, навчання та тестування даних мереж, а також експорту та імпорту будь-яких конфігурацій мережі

Особливість	Sharp Develop	MS Visual Studio
Автозаповнення коду	+	+
Підсвітка синтаксису коду	+	+
Конструктор Windows Forms	+	+
Дизайнер веб-форм	-	+
Покриття коду	+	-
Модульне тестування	+	-
Довідкова інформація	-	+
Оглядач рішень	+	+
Браузер об'єктів	+	+
Перегляд джерел даних	+/-	+
Підтримка плагінів	+	+/-
Майстер додавання джерела даних	-	+

Було обрано середовище програмування Microsoft Visual Studio, так як воно регулярно оновлюється, підтримується розробниками та має зручний інтерфейс

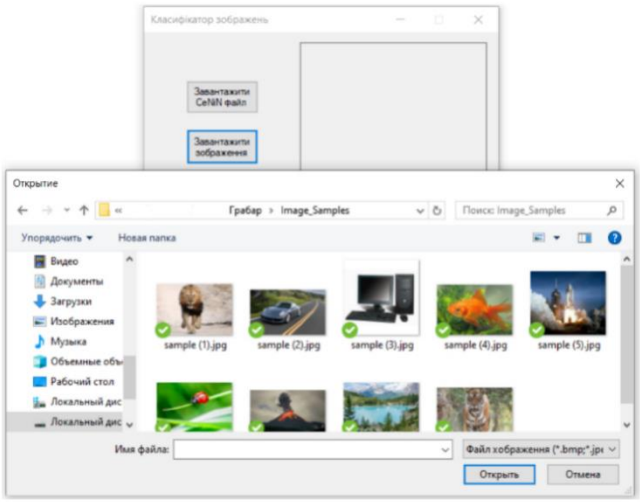
СТАРТОВІ ВІКНА ПРОГРАМИ

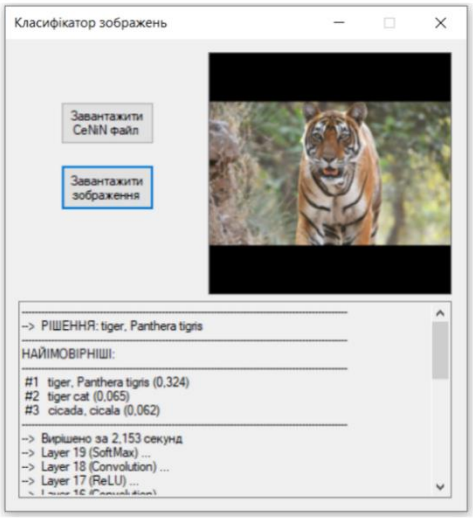




Для вимірювання часу виконання програми використано стандартний клас Stopwatch

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ





У вікні легко побачити структуру шарів згорткової нейронної мережі та використовувані функції активації:

--> Layer 19 (SoftMax) ...
--> Layer 18 (Convolution) ...
--> Layer 17 (ReLU) ...
--> Layer 16 (Convolution) ...
--> Layer 15 (ReLU) ...
--> Layer 14 (Convolution) ...
--> Layer 13 (Pool) ...
--> Layer 12 (ReLU) ...
--> Layer 11 (Convolution) ...
--> Layer 10 (ReLU) ...
--> Layer 9 (Convolution) ...
--> Layer 8 (ReLU) ...
--> Layer 7 (Convolution) ...
--> Layer 6 (Pool) ...
--> Layer 5 (ReLU) ...
--> Layer 4 (Convolution) ...
--> Layer 3 (Pool) ...
--> Layer 2 (ReLU) ...
--> Layer 1 (Convolution) ...

Тестування та аналіз результатів роботи програмного модуля

Порівняння результатів роботи розробленого модуля та програми-аналога

	Програма RexNeXt	Розроблений програмний засіб	Номер зображення
Точність класифікації (імовірність належності вхідного зображення визначеному класу)	0.762	0.806	1
	0.875	0.938	2
	0.230	0.231	3
	0.925	0.996	4
	0.956	0.999	5
	0.855	0.999	6
	0.817	0.927	7
Середня точність класифікації	0,73	0,80	
Швидкодія	1,611 с.	1,437 с.	1
	1,678 с.	1,457 с.	2
	1,659 с.	1,391 с.	3
	1,645 с.	1,439 с.	4
	1,586 с.	1,429 с.	5
	1,579 с.	1,411 с.	6
	1,571 с.	1,427 с.	7
Середня швидкодія	1,615 с.	1,432 с.	

Було проведено експерименти по класифікації 7 різних зображень розробленим модулем та програмою-аналогом ResNeXt

Із табл. видно, що розроблена програма має середню точність класифікації зображень 0,80 (тобто 80%), а аналогічна програма - 0,73 (73%), тобто розроблена програма має на 7% вищу середню точність класифікації. Також із табл. видно, що розроблена програма має середню швидкодію 1,432 с., а аналогічна програма - 1,615 с., тобто має на 13% (((1,615-1,432)/1,432)*100%=13%) вищу середню швидкодію. Тобто мета роботи досягнута – точність та швидкодія класифікації зображень підвищена у розробленому програмному засобі.

ВИСНОВОК

В результаті виконання магістерської роботи розроблено програмний модуль класифікації зображень за контентом на основі згорткової нейронної мережі. Модуль створено мовою програмування C# у середовищі Microsoft Visual Studio з використанням бібліотеки Aforge.NET. Модуль має середню точність класифікації зображень на 7% вищу за аналог, а також середню швидкодію на 13% вищу за аналог. Тобто мета роботи досягнута – точність та швидкодія класифікації зображень підвищена.