Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ МЕТОДІВ ВИДІЛЕННЯ ОБ'ЄКТІВ
НА ЗОБРАЖЕННЯХ ПРИ РОЗВ'ЯЗАННІ ЗАДАЧІ КЛАСИФІКАЦІЇ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-19-2
Сергєєв Я.С.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір С.В
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
(код і повна назва)
Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Сергєєву Ярославу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів виділення об'єктів на зображеннях при розв'язанні задачі класифікації

затверджена наказом по університету від «23» жовтня 2020 року № 1428Ст.
2. Термін подання студентом роботи до екзаменаційної комісії 25 листопада 2020 р.
3. Вихідні дані до роботи: Методи виділення об'єктів на зображенні, методі інтерпретації рішень моделей, бібліотека PyTorch, мова програмування Python, згорткові нейронні мережі,оператор Собеля, оператор Прюітта, оператор Лапласа, оператор Шарра, міри якості нейронних мереж
4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд класичних методів виділення об'єктів на зображенні
2. Вимоги методів машинного навчання
3. Вирішення задачі класифікації за допомогою нейронних мереж
4. Огляд операторів виділення границь
5. Аналіз структури згорткових нейронних мереж
6. Розробка архітектури згорткової нейронної мережі
7. Розробка моделей нейронної мережі із запропонованими модифікаціями
8. Програмна реалізація згорткової нейронної мережі з використанням фреймворку PyTorc та мови програмування Python
9. Порівняння точночті розроблених моделей

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Приклади задач комп'ютерного зору, модель згорткової нейронної мережі, архітектура запропонованої моделі, детектори границь, ілюстрація роботи програми, порівняльний аналіз побудованих моделей

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на до атестаційної роботи	23.10.2020	
2	Аналіз завдання, підбір літератури	24.10.20-25.10.20	
3	Аналіз літератури з досліджуваної проблеми	26.10.20-27.10.20	
4	Огляд існуючих алгоритмів виділення границь	28.10.20-29.10.20	
5	Огляд існуючих методів вирішення задачі класифікації	30.10.20-01.11.20	
6	Програмна реалізація	02.11.20-14.11.20	
7	Оформлення пояснювальної записки	15.11.20-19.11.20	
8	Перевірка на плагіат	20.11.20	
9	Рецензування	22.11.20	
10	Підготовка презентації та доповіді	25.11.20	
11	Занесення роботи в електронний архів	30.11.20	
12	Попередній захист атестаційної роботи	02.12.20	

Дата видачі завдання 23 10 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Машталір С.В
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка атестаційної роботи: 72 с., 1 табл., 43 рис., 1 додаток, 29 джерел.

КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, СЕГМЕНТАЦІЯ, МАШИННЕ НАВЧАННЯ, НАВЧАННЯ З УЧИТЕЛЕМ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, DATASET CIFAR-10, ФРЕЙМВОРК PYTORCH.

Дана робота присвячена вирішенню проблеми класифікації об’єктів на зображенні за допомогою згорткових нейронних мереж. Проаналізовані існуючі підходи щодо виділення об’єктів та границь на зображеннях. Проаналізовано різні архітектури згорткових нейронних мереж та розроблена власна архітектура для застосування модифікацій на її основі. Об’єктом дослідження є застосування детекторів границь замість першого згорткового шару нейронної мережі та порівняння точності побудованих моделей. Метою даної роботи є покращення точності розпізнавання об’єктів з використанням запропонованих модифікацій. Виконано порівняння точності моделей та обрано найбільш точну модель.

IMAGE CLASSIFICATION, SEGMENTATION, MACHINE LEARNING, TEACHER LEARNING, CONVOLUTION NEURAL NETWORK, DATASET CIFAR-10, PYTORCH FRAMEWORK.

This paper proposes a solution of the problem of image classification using convolution neural networks. Different objects and edges detection methods were investigated. Different architectures were analysed and custom was built for futher modifications. The object of research is to use edge detectors in the first convolutional layer of the neural network and to compare the accuracy of the constructed models. The purpose of this work is to improve the accuracy of object recognition using the proposed modifications. The accuracy of the models is compared and the most accurate model is chosen.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN – Convolutional Neural Network

ReLU – Rectified Linear Unit

GPU – Graphics Processing Unit

CPU – Central Processing Unit

ВСТУП

В даний час людство намагається автоматизувати все більше завдань і це посприяло бурхливому розвитку і повсякденного впровадження систем комп'ютерного зору. Розпізнавання образів і виділення об'єктів є основою для систем управління та обробки інформації, автоматизованих систем та систем прийняття рішень.

Ця задача вирішується у найрізноманітніших сферах діяльності людини: у медичній сфері, при знаходженні різного роду пухлин; у системах безпеки; під час стеження за об'єктами; у відновленні зображень, у системах автопілота та в багатьох інших. Задача розпізнавання складається з двох підзадач: виділення об'єкта та віднесення його до конкретного класу.

Основним ключовим моментом у розпізнаванні об'єктів є набір різних ознак (категорій) об'єктів та їх опис. Для цього потрібно набір надійних та повторюваних параметрів (дескрипторів), які отримуються з тестових зображень або з моделі об'єкта.

Під «розпізнаванням» зазвичай мається на увазі той факт, що досліджуваний об'єкт, представлений у вигляді сукупності спостережень, слід відносити до того чи іншого взаємовиключного класу [1]. Таким чином, у даному контексті, розпізнавання образів та об'єктів являє собою одну з різновидів класифікації [2], і в тих випадках, коли клас містить тільки один об'єкт, класифікація є ідентифікацією, тобто присвоєнням даного об'єкту однозначної назви. Іншими словами, отримання характерних точок на зображенні дозволить в подальшому класифікувати даний об'єкт.

Незважаючи на те, що всі предмети і ситуації унікальні, між деякими з них завжди можна знайти подібності за тією чи іншою ознакою. Отже, класифікацією є розбиття всієї множини об'єктів на підмножини – класи, елементи яких мають деякі схожі властивості, що відрізняють їх від елементів інших класів. Таким чином, завданням розпізнавання є віднесення розглянутих об'єктів або явищ за їх описом до потрібних класів.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ ВИДІЛЕННЯ ОБ'ЄКТІВ

1.1 Актуальність проблеми

Розпізнавання і виділення об'єктів на зображеннях є одними з найбільш важливих завдань комп'ютерного зору та обробки зображень. Вони знаходять своє застосування у різних сферах людської діяльності.

Одним із засобів виділення об'єктів і меж (ліній, кривих, тощо) на зображеннях є сегментація. На цьому етапі відбувається групування розрізнених ділянок або фрагментів зображення в область, що належить одному об'єкту, або поділ будь-якої ділянки зображення на області, що належать різним об'єктам. Слід зазначити, що при цьому групування здійснюється за різними ознаками, такими як яскравість, колір, текстура, рух в одному напрямку та інші подібні ознаки. Далі будуть розглянуті деякі з них [3].

1.2 Огляд підходів щодо виділення об'єктів

У якості критерію оцінки методів з точки зору їх застосування у системах комп'ютерного зору можна вважати придушення фону і виділення об'єктів у вигляді зв'язних областей. Оскільки поняття «об'єкт» у загальному випадку формалізовано не повністю і завжди апріорна інформація мінімальна, то не можна вимагати точного виділення об'єкта, що складається з декількох частин різної яскравості як однієї зв'язної області. Друга вимога полягає у тому, що повинні бути виділені принаймні ключові частини об'єкта, необхідні для його подальшого розпізнавання.

Порогові методи застосовуються за умови існування стабільних відмінностей яскравості окремих областей. Методи нарощування областей ефективні за наявністю стійкої зв'язності всередині окремих сегментів. Метод

виділення границь добре застосовувати, якщо границі досить чіткі та стабільні. Для опису та сегментації властивостей зображень (однорідності, шорсткості, регулярності) застосовують текстурні методи, які умовно поділяються на дві категорії: статистичні та структурні. Прикладом статистичного підходу є використання матриць збігів, що формуються шляхом вихідних зображень; структурного – мозаїка Вороного. Також існують комбіновані методи, засновані на аналізі кольору.

1.2.1 Метод вододілів

Основна ідея цього методу полягає у тому, щоб після побудови поля контрастності зображення, побудувати вододіли (області високої контрастності, що відповідають границям зображення), що відокремлюють один від одного басейни (області низької контрастності, що відповідають нутрощам зображень) [4]. Після виділення значущих вододілів слід послідовно провести злиття сусідніх незначущих басейнів локального характеру, які мають однорідні нутрощі сегмента об’єкта, або фону. Позитивною рисою методу вододілів є те, що розподіл контрастності для нутрощів об’єктів і фону після контрастування, як правило, носять характер рівномірного закону розподілу і, в цьому відношенні, з ними досить легко працювати. Однак при застосуванні методу вододілів виникає велика проблема з вибором порога для визначення значущості вододілів. На практиці це виражається у тому, що при заниженому значенні порога отримуємо велику кількість незначущих контурів, а при підвищеному – злиття областей різних об’єктів. Для вирішення цих проблем пропонується використовувати маркери об’єктів, що, однак, призводить до необхідності інтерактивної сегментації зображень).

1.2.2 Методи кластерного аналізу

Для сегментації областей можуть бути використані методи кластерного аналізу. У роздільній кластеризації весь набір даних вважається кластером, який рекурсивно розщеплюється на менші кластери. У агломеративної кластеризації кластером вважається кожен елемент даних, а для отримання кращого уявлення кластери рекурсивно зливаються. Основою для злиття (розщеплення) служить аналіз відстані між кластерами. Найпростіші алгоритми кластеризації засновані на ітераційному (рекурсивному) злитті (розщепленні) пар кластерів на основі критерію мінімальної (максимальної) відстані між кластерами [5].

Методи кластерного аналізу можуть застосовуватися у просторовій, і в частотній областях – для гістограми зображення. Однак, слід зауважити, що у такій загальній постановці методи кластеризації вирішують не завдання сегментації, а завдання структуризації зображення і, при цьому, характеризуються великою трудомісткістю у застосуванні. Разом із тим користь від використання методів кластеризації, що застосовуються на початковій фазі аналізу зображення (до сегментації) полягає у тому, що з введенням певних обмежень можна ефективно провести: структуризацію однорідних областей на зображенні незалежно від варіацій геометричних і фотометричних параметрів і, потім, на цій основі, локалізацію зображень об’єктів для сегментації. При цьому можуть накладатися обмеження на область застосування, на зв’язність кластерів, а також на кількість розглянутих рівнів ієрархії. З метою практичного застосування у даний час поширеними є методи кластеризації, засновані на представленні зображення у вигляді графа, і працюють за принципом ітераційного вирощування головних областей із другорядних за заданим критерієм близькості характеристик другорядних областей. Багатьма фахівцями вважається, що найкращі результати сегментації отримуються під час поєднання критеріїв кластерного і дискримінаційного аналізу, наприклад критерію мінімізації

середньоквадратичної похибки за умовою, що задано центри кластерів для зображень розглянутих класів об’єктів.

1.2.3 Методи сегментації текстури

Сегментація текстури полягає у виявленні на зображенні ділянок з постійною текстурою. Для сегментації (класифікації) текстури у даний час використовуються три основні підходи: структурний, статистичний і спектральний; в рамках цих підходів розроблено велику кількість методів, орієнтованих на виявлення і композицію фрагментів текстури [6]. Найбільшу проблему при цьому становить сегментація дрібнодисперсної текстури, оскільки неможливо виділити структурні елементи текстури і обробити їх окремо, у той час як завдання сегментації дрібнодисперсної текстури є одним з найбільш актуальних у багатьох додатках, наприклад, у задачах дистанційного зондування земної поверхні, що включають землекористування, прогнозування врожаю, моніторингу лісових масивів і інші додатки. Така ситуація складається тому, що при сегментації текстури використовуються лише числові характеристики текстури, тому ситуація помилкової класифікації є типовою для подібних областей різних класів об’єктів, а також для граничних фрагментів сусідніх областей. Така класифікація рідко дає прийнятні результати для напівтонових зображень, а також для кольорових зображень в умовах значних варіацій фотометричних параметрів зображень.

1.3 Огляд підходів щодо виділення границь

Виділення контурів використовується у якості допоміжного інструмента. Завдання полягає у побудові зображення меж об’єктів і контурів

однорідних областей. Контуром зображення вважається сукупність пікселів, в околиці яких спостерігається стрибкоподібна зміна функції яскравості. Оскільки у цифровій обробці зображення представлено як функція цілочисельних аргументів, то контури представляються лініями з шириною як мінімум в один піксель. Якщо вихідне зображення крім областей постійної яскравості містить ділянки з мінливою яскравістю, то введене визначення контуру залишається справедливим, однак у цьому разі не гарантується безперервність контурних ліній: розриви контурів будуть спостерігатися у тих місцях, де зміна функції яскравості не є достатньо різкою. З іншого боку, якщо на «кусково-постійному» зображенні присутній шум, то, він може бути зафіксований як контур. Під час розробки алгоритмів виділення контурів потрібно враховувати зазначені особливості поведінки контурних ліній.

Спеціальна додаткова обробка виділених контурів дозволяє усувати розриви і пригнічувати неправдиві контурні лінії. Однією з основних характеристик зображення є колір. За допомогою збору статистичної інформації про точки можна накопичити достатньо даних для подальшого аналізу. Шляхом побудови і порівняння колірних гістограм проводиться дослідження графічних даних.

1.3.1 Граничні детектори

Основою для побудови контурних методів сегментації вже довгий час залишаються граничні детектори, метою яких є виявлення граничних пікселів зображень за рівнем контрастності на основі використання масок. Такий підхід набув широкого поширення в силу своєї простоти і низькою трудомісткості реалізації.

Класичний граничний детектор являє собою пороговий критерій, який служить для виявлення граничного пікселя зображення за рівнем контрастності, який оцінюється на основі використання згортки маскою,

зазвичай, з центром, поєднаним з даним пікселем зображення. Для побудови граничних детекторів в даний час розроблено велику кількість масок різного типу сегментації [7]. Найпростішими масками є інваріантні повороту симетричні маски, що дозволяють оцінити контрастність пікселя щодо пікселів його околиці, яка визначається положенням маски в середньому. Такі маски використовуються для виявлення пікселів і фрагментів границь зображення.

Використання симетричних масок характеризується найнижчою трудомісткістю серед масок інших типів, але, у той самий час, і найнижчою ефективністю, оскільки поріг рівня контрастності фіксований, значення контрастності не нормовано за значенням відстані відповідно до класу функції яскравості і ступеня її спотворення. При цьому згортка з використанням маски характеризується високим розкидом допустимих відхилень, що різко зменшує ймовірність відділення пікселів зображення від пікселів фону за значенням контрастності і ускладнює вибір значення порога. Тому використання таких масок за умов низької контрастності зображень призводить до нестійкої сегментації.

Для сегментації границь зображень широке застосування в даний час знаходять спрямовані маски, наприклад, маски Прюітта, або Собеля. За схожим принципом з метою сегментації ліній широке застосування набули спрямовані лінійні маски. Загальна ідея застосування спрямованих масок полягає в тому, щоб під час обертання (тобто при використанні масок сімейства) оцінювати контрастність і потім вибрати положення маски з найбільшим рівнем контрастності, адаптуючись, таким чином, до положення границі або лінії. При цьому трудомісткість використання спрямованих масок 3×3 приблизно у п’ять разів вище трудомісткості використання симетричних масок через необхідність адаптації положення маски.

Симетричні і спрямовані маски не враховують наявність тіні і гілок границі, а в якості оцінки контрастності використовується згортка з маскою, що різко зменшує здатність відділення зображення від фону за значенням

контрастності та ускладнює вибір порога. Для обліку цих недоліків у роботі було запропоновано побудовані на основі кола маски з диференціальним (спрямованим) правилом оцінювання контрастності, призначені для сегментації границь зображень і ліній за значенням контрастності. В умовах високої контрастності для зниження трудомісткості сегментації також можуть застосовуватися одномірні маски [8]. Однак зі зменшенням загального рівня контрастності зображень необхідно будувати і застосовувати сімейства одновимірних масок. Аналіз перепадів яскравості за напрямками для масок є аналогом використання системи одновимірних масок і був детально розглянутий у [9].

Найбільш досконалі граничні детектори після застосування порогового критерію можуть використовувати додаткові критерії для відсіву помилкових контрастних пікселів. Основними для цієї мети є критерії аналізу знака та напрямку зміни яскравості, а також типу спектру контрастності пікселя. Крім аналізу контрастності на основі використання масок для цілей побудови граничного детектора широкого застосування набули гістограми, а також частотні перетворення Фур'є, або вейвлети.

Використання гістограм для сегментації границь зображень розглядається нижче. Що стосується частотних детекторів границь, заснованих на використанні перетворення Фур'є, або вейвлетів, – ці детектори характеризуються надзвичайно високою трудомісткістю і не адаптуються до просторового положення границі. Крім того, контрастність уздовж границі зображення змінюється і, отже, різним фрагментам границі можуть відповідати різні коефіцієнти перетворення в частотній області. Тому для прийнятної сегментації границь потрібна складна процедура відбору коефіцієнтів перетворення. Після сегментації пікселів границі наступним завданням є зв'язування границь і побудова меж зображень. Для цих цілей розроблено ряд локальних і глобальних детекторів і методів.

1.3.2 Локальне зв’язування контурів

В умовах відсутності розривів границі основним для зв’язування є хвильовий метод [10]. Однак, граничні пікселі зображень і фону знаходяться в безпосередній близькості один від одного, і застосування хвильового методу для зв’язування суміжних пікселів призведе до об’єднання пікселів паралельних ліній границь об’єкта та фону. В таких умовах для правильного об’єднання сегментованих пікселів у компоненту границі для деякої невеликої околиці застосовуються спеціальні детектори локального аналізу подібності характеристик пікселів, що об’єднуються. Ці детектори спрямовані на перевірку близькості значень яскравості і контрастності пікселів, на встановлення фактів відсутності взаємної контрастності і збігу напрямку проходження границі. З використанням таких детекторів, зокрема, можна домогтись подолання локальних розривів, а саме: об’єднати незв’язні фрагменти границі зображення [8].

Дві ключові проблеми сегментації границь – поява розривів границі (через часткову сегментацію) і потовщення границі (через помилкову сегментацію пікселів поблизу границі). Для усунення локальних розривів границь, існують методи, засновані на використанні лінійних масок з пропусками – для зв’язування фрагментів границі, що лежать на одній прямій (кривій), а також методи гістерезису, засновані на використанні двох порогів для сегментації слабо виявленої границі в області розриву. Однак застосування таких методів нерідко призводить до сегментації великого числа пікселів поблизу справжньої границі і утворення різного роду гілок і петель, обробляти які в загальному випадку досить складно. Для усунення глобальних розривів (і усунення надмірної товщини) границь, зазвичай, застосовуються методи апроксимації контурів. Однак застосування методів апроксимації може бути обґрунтовано лише для об’єктів регулярного виду [9]. Методи, що роблять границі зображення тоншими, найчастіше орієнтовані на втілення ідеї немаксимального придушення, суть якої полягає у тому, щоб розглядати

перпендикулярні до границі відрізки і, для кожного з них, з множини граничних пікселів, що лежать на відрізку, залишати піксель з максимальним значенням контрастності. Для ефективної реалізації немаксимального придушення розроблені спеціальні маски [11]. Крім того, для побудови границі зображення використовується пошук на графі, вершинами якого є сегментовані пікселі границі, а зв'язки встановлюються між 4 і 8- зв'язковими пікселями границі.

Критерій оптимальності – мінімальна довжина границі. Однак, такий метод (з точки зору обліку форми зображення) є евристичними і, крім цього, надзвичайно трудомісткий, через необхідність перегляду величезної кількості варіантів положення границі [11].

1.4 Нейронні мережі у розпізнаванні образів

Незважаючи на те, що існуючі традиційні методи розпізнавання образів широко застосовувалися на практиці, у процесі застосування виникають деякі проблеми, такі як: незадовільні ефекти, низький рівень точності розпізнавання і слабка адаптивна здатність.

Модель глибокого навчання має потужну навчальну здатність, яка об'єднує процес вилучення ознак і класифікації в єдине ціле, що може дати великий приріст точності результатів розпізнавання. Однак цей метод має наступні проблеми в процесі застосування: по-перше, неможливо ефективно апроксимувати складні функції в моделі глибокого навчання, по-друге, модель глибокого навчання має низький класифікатор з низькою точністю.

Важливо також відзначити, що традиційні алгоритми запрограмовані явно, тобто створені на основі детального списку правил виду «якщо ..., то ...». Інженер-проектувальник ретельно продумав усі можливі комбінації і створив систему, яка приймає рішення, як результат проходу ланцюжком правил. У той час нейронні мережі мають можливість навчатися різними сценаріями без

явного програмування, що є найбільшою перевагою даного методу. Існує багато методів вирішення завдання класифікації, зокрема найпопулярніші з них були розглянуті у [12].

Велика кількість завдань, що вирішуються нейронними мережами, пояснюється в основному тим, що можливість навчання дозволяє зробити функціонування системи на порядок ефективнішим.

1.4.1 Історія розвитку нейронних мереж

Варто відзначити, що поняття «нейронна мережа» бере свій початок у роботі американських математиків, нейролінгвістів і нейропсихологів У. Маккалок і У. Питтса (1943 р.), де автори вперше згадують про неї, дають визначення і роблять першу спробу побудови моделі нейронної мережі [13]. У 1949 р. Д. Хебб пропонує перший алгоритм навчання. Далі був ряд досліджень в області нейронного навчання, і перші робочі прототипи з'явилися приблизно у 1990-1991 рр. Проте обчислювальних потужностей обладнання того часу не вистачало для досить швидкої роботи нейронних мереж, і вони були забуті майже на 20 років. До 2010 року, доки потужності GPU відеокарт сильно збільшилися і їх почали застосовувати для обчислень, що істотно (у 3-4 рази) збільшило продуктивність комп'ютерів. У 2012 р. нейронна мережа AlexNet вперше перемогла на змаганні ImageNet, що й ознаменувало їх подальший бурхливий розвиток і появу терміна Deep Learning.

У сучасному світі нейронні мережі охоплюють дуже багато галузей діяльності, вчені вважають вкрай перспективними дослідження, що проводяться в області вивчення поведінкових особливостей і станів нейронних мереж. Перелік галузей, в яких застосовують нейронні мережі, величезний: розпізнавання образів, класифікація, прогнозування, рішення апроксимаційних задач, аналіз даних – ось далеко не повний список завдань, у яких активно використовують нейронні мережі. Дослідження нейронних

мереж сьогодні активно відбувається у наукових спільнотах різних країн. У задачі розпізнавання в якості об’єкта може бути людське обличчя, рукописна цифра, а також безліч інших об’єктів, які характеризуються рядом унікальних ознак, що суттєво ускладнює процес ідентифікації. За останні кілька років глибоке навчання повністю домінувало у галузі комп’ютерного зору, досягаючи кращих результатів у багатьох завданнях і пов’язаних з ними змаганнями. Найбільш популярним і відомим з цих змагань комп’ютерного зору є ImageNet.

Змагання ImageNet ставить перед дослідниками завдання створити модель, яка найбільш точно класифікує зображення у наборі даних. Дані для нього були зібрані з Flickr та інших систем, вручну розмічені людьми, при цьому кожне зображення належить одному з 1000 класів об’єктів. За останні кілька років методи глибокого навчання дозволили швидко просунутися у цьому змаганні, і навіть перевершили людські можливості.

1.4.2 Основні принципи глибоких нейронних мереж

Спрощена версія глибокої нейронної мережі може бути представлена як ієрархічна (шарувата) структура з нейронів (подібно нейронам у мозку), пов’язаних з іншими нейронами. На основі вхідних даних одні нейрони передають команду (сигнал) іншим і, таким чином, формують складну мережу, яка навчається за допомогою певного механізму зворотного зв’язку. Схема глибокої нейронної мережі з кількістю шарів N зображена на рисунку 1.1 Вхідні дані передаються нейронам на перший (не прихований) шар, вони в свою чергу передають вихідні дані нейронам на наступному шарі і так далі до фінального виходу. Вихід може являти собою прогноз («Так»/«Ні»), представлений як ймовірність. На кожному шарі може бути один або безліч нейронів, кожен з яких обчислює невелику функцію, що зветься функцією активації. Ця функція імітує передачу сигналу наступним,

пов’язаних з попередніми, нейронам. Якщо результат вхідних нейронів перевищує поріг, вихідне значення просто ігнорується і передається далі.

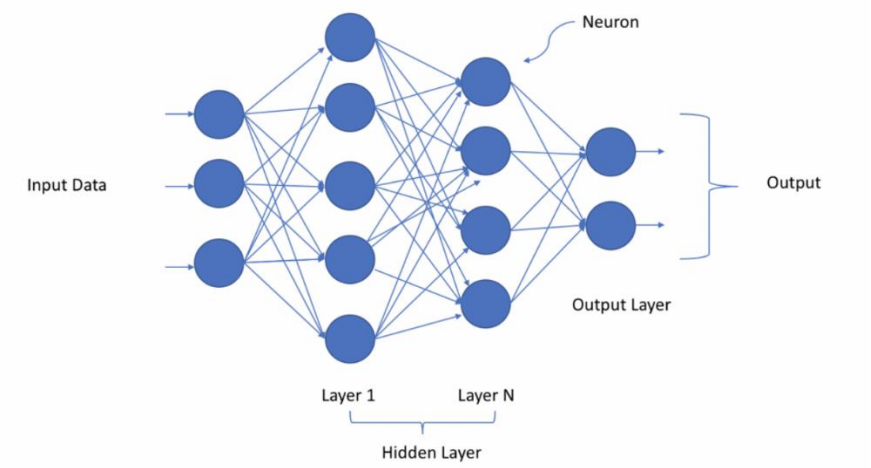

Рисунок 1.1 – Схема глибокої нейронної мережі з кількістю шарів N

Зв’язок між двома нейронами сусідніх шарів має вагу. Вага визначає вплив вхідних даних на вихід для наступного нейрона і наступний фінальний вихід. Початкові ваги нейронної мережі випадкові, проте у процесі навчання моделі вони постійно оновлюються і навчаються передбачати вірне вихідне значення. У процесі аналізу нейронної мережі можна виявити кілька логічних структурних елементів (нейрон, шар, вага, вхід, вихід, функція активації і нарешті механізм навчання, або оптимізатор), які допомагають їй поступово замінювати ваги (спочатку з випадковими значеннями) на більш підходящі для точного прогнозу виходу.

1.5 Постановка задачі дослідження

Хоча зараз існує велика кількість різних методів сегментації, проте, проведений вище огляд та аналіз ефективності їх застосування показує, що ці алгоритми мають обмеження на застосування, а результат залежить від багатьох факторів. Виділення об’єкта є частиною завдання розпізнавання

образів для їх подальшої класифікації. Раніше було розглянуто найбільш точний на сьогоднішній день метод вирішення цього завдання – нейронні мережі.

Об’єктом дослідження є застосування детекторів границь замість першого згорткового шару та порівняння точності побудованих моделей. Метою даної роботи є покращення точності розпізнавання об’єктів з використанням запропонованих модифікацій.

Задача полягає у тому, щоб побудувати згорткову нейронну мережу для розпізнавання зображень, використовуючи різні маски та оператори виділення границь у першому шарі. Для цього необхідно вирішити такі завдання:

– провести аналіз операторів виділення границь;

– розробити модель згорткової нейронної мережі;

– програмно реалізувати модель з використанням запропонованих модифікацій.

2 РОЗРОБКА МОДЕЛЕЙ ДЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ ГЛИБОКИХ НЕЙРОННИХ МЕРЕЖ

Традиційні нейронні мережі використовують повністю зв’язану архітектуру, де кожен нейрон в одному шарі з’єднується з усіма нейронами в наступному шарі. Однак, така архітектура не є ефективною для вирішення задач, пов’язаних з обробкою зображень через наступні причини:

– для середнього зображення із сотнями пікселів і трьома каналами традиційна нейронна мережа генерує мільйони параметрів, що може призвести до перенавчання;

– обчислення будуть займати досить тривалий час, що є наслідком попереднього пункту;

– важко інтерпретувати результати, налагоджувати та налаштовувати модель для покращення її продуктивності.

Згорткова нейронна мережа, CNN – основний інструмент для класифікації та розпізнавання об’єктів, обличь на фотографіях, мови, рукописного введення, візуальних об’єктів, символів, тощо. На самому базовому рівні вона є простою багаторівневою ієрархічною нейронною мережею, але на відміну від повнозв’язної нейронної мережі, у CNN нейрони в одному шарі не з’єднуються з усіма нейронами у наступному шарі.

Зазвичай, CNN використовує тривимірну структуру, де кожен набір нейронів аналізує певну область або «ознаки» зображення. CNN фільтрує з’єднання за близькістю (пікселі порівнюються лише з пікселів поблизу), що робить навчальний процес обчислювально досяжним.

Згорткова нейронна мережа за рахунок застосування спеціальної операції, власне згортки, дозволяє водночас зменшити кількість інформації, що зберігається у пам’яті, за рахунок чого точніше працює з зображеннями більш високої роздільної здатності, і виділяє такі опорні ознаки зображення як ребра, контури або грані. На наступному рівні обробки з цих ребер і граней

можна розпізнати повторювані фрагменти текстур, які далі можуть скластися у фрагменти зображення.

Кожен шар нейронної мережі використовує власне перетворення. Якщо у першому шарі мережа оперує такими поняттями як «ребра», «грані» та ін., то далі використовуються такі поняття як «текстура» та «частини об’єктів». Кожна група нейронів фокусується на одній частині зображення. Наприклад, на зображенні кота одна група нейронів може ідентифікувати голову, друга – тіло, третя – хвіст тощо. Може існувати кілька етапів сегментації, в яких алгоритм розпізнавання зображень нейронної мережі аналізує менші частини зображень, наприклад, на голові, ніс, вуса, вуха тощо. Фінальний результат – вектор імовірностей, який передбачає для кожної ознаки на зображенні ймовірність належності до певного класу. В результаті такої обробки можна правильно класифікувати зображення або виділити на останньому етапі потрібний об’єкт.

2.1 Історія змін у архітектурі згорткових нейронних мереж

Архітектура нейронної мережі AlexNet [14] стала революційною та почала нову епоху розвитку машинного навчання. У ній було втілено багато новаторських ідей, які стали стандартами та викостивуються і досі як базові принципи для побудови глибокий нейронних мереж, а саме:

– вперше використали в якості функції активації ReLU, також встановили, що нелінійні функції активації працюють краще і дозволяють скоротити час навчання у порівнянні з функцією tanh. Нелінійність ReLU тепер має тенденцію бути функцією активації за замовчуванням для глибоких нейронних мереж;

– запропонували методи збільшення даних (data augmentation), які складалися з перекладів зображень, горизонтальних відображень і віднімання

середнього. Ці методи сьогодні також дуже широко використовуються для вирішення багатьох завдань комп'ютерного зору;

– використання шарів виключення для вирішення проблеми перенавчання;

– запропонували чередування шарів згортки з пулінгом та використання повноз'єднаних шарів наприкінці, що все ще є основою багатьох глибоких мереж сьогодні.

Наступні інновації принесла архітектура VGGNet, зокрема:

– використання лише фільтрів розміром 3×3 замість 11×11, що використовувалися у AlextNet. Вони показали, що дві послідовні згортки 3×3 мають еквівалентне сприйнятливе поле або «поле зору», як єдину згортку 5×5; аналогічно, три послідовні згортки 3×3 еквівалентні одній 7×7. Перевага полягає в тому, що імітується більший фільтр, зберігаючи переваги менших розмірів фільтра [15]. Перша перевага менших фільтрів – зменшення кількості параметрів. Друга – можливість використання функції ReLU між кожною згорткою, що робить систеу біль нелінійною, а функцію прийняття рішень більш дискримінаційною;

– використання пулінгу, а саме максимального пулінгу (Max Pooling) для зменшення просторової інформації та зберігання найхарактерніших ознак, що сприяє більш точній класифікації;

– представлено новий вид збільшення даних – «тремтіння масштабу» (scale jittering). Він включає запуск моделі з кількома зміненими у масштабі версіями тестового зображення з подальшим усередненням результуючого класифікатора.

Наступним кроком у розвитку нейронних мереж стали моделі GoogleNet Inception, вони привнесли такі зміни:

– використання згорток 1×1 перед кожними 3×3 та 5×5, початковим модулем зменшує кількість карт функцій, що проходять кожним шаром, зменшуючи таким чином обчислення та споживання пам'яті;

– початковий модуль має параллельне обчислення згорток 1×1, 3×3 та 5×5. Ідея полягала в тому, щоб дозволити мережі вирішувати під час навчання, яку інформацію слід вивчати та використовувати. Це також дозволяє виконувати багатомасштабну обробку: модель може відновлювати як локальні ознаки за допомогою менших згорток, так і високі абстрактні ознаки за допомогою великих згорток;

– представлено ідею про те, що шари CNN не завжди потрібно складати послідовно. Автори показали, що збільшення ширини, а не лише глибини мережі, призводить до кращої продуктивності [16].

Архітектура ResNet [17] була першою, яка змогла досягти точності людського рівня в змаганні ImageNet, їх основні висновки наступні:

– представлено ідею про те, що шари CNN не завжди потрібно складати послідовно. Автори показали, що збільшення ширини, а не лише глибини мережі, призводить до кращої продуктивності;

– показано, що наївне укладання шарів, що робить мережу дуже глибокою може погіршити ситуацію.

2.2 Архітектура згорткової нейронної мережі

У звичайному перцептроні, який представляє собою повнозв’язну нейронну мережу, кожен нейрон пов’язаний з усіма нейронами попереднього шару, причому кожен зв’язок має свій персональний ваговий коефіцієнт. У згортковій нейронній мережі операція згортки використовується лише обмежену матрицю ваг невеликого розміру, яка ковзає вздовж усього шару (на самому початку – безпосередньо вхідним зображенням), формуючи після кожного зсуву сигнал активації для нейрона наступного шару з аналогічною позицією. Тобто для різних нейронів вихідного шару використовуються загальні ваги – матриця, яку також називають набором ваг або ядром згортки. Вона побудована таким чином, що графічно кодує якусь одну ознаку,

наприклад, наявність похилої лінії під певним кутом. Тоді, наступний шар, що вийшов в результаті операції згортки з такою матрицею ваг, показує наявність даної похилої лінії в оброблюваному шарі і її координати, формуючи так звану карту ознак. Певна річ, що у згортковій нейронній мережі набір ваг не один, а ціла гама, що кодує всілякі лінії і дуги під різними кутами.

При цьому такі ядра згортки не закладаються дослідником заздалегідь, а формуються самостійно шляхом навчання мережі класичним методом зворотного поширення помилки. Прохід кожним набором ваг формує свій власний примірник карти ознак, роблячи нейронну мережу багатовимірною (багато незалежних карт ознак в одному шарі). Також слід зазначити, що під час перебору шару матрицею ваг її пересувають зазвичай не на повний крок (розмір цієї матриці), а на невелику відстань. Так, наприклад, при розмірності матриці ваг 5×5 її зрушують на один або два пікселя замість п’яти, щоб не пропустити шукану ознаку. Загальна схема представлена на рисунку 2.1.

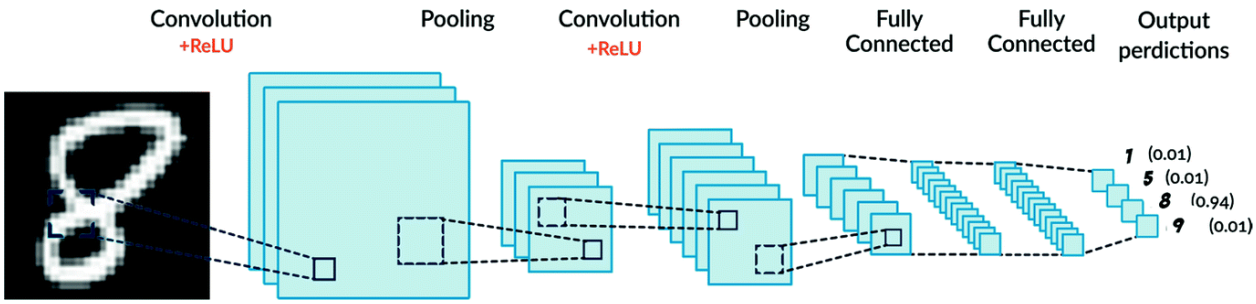

Рисунок 2.1 – Загальна схема CNN

Операція пулінгу (англ. Subsampling, англ. Pooling, також перекладається як «операція підвибірки» або операція об’єднання), виконує зменшення розмірності сформованих карт ознак. В данному типі мережі факт наявності шуканої ознаки важливіший за точне знання її координат, тому з декількох сусідніх нейронів карти ознак вибирається максимальний і приймається за один нейрон карти ознак зменшеної розмірності.

Таким чином, повторюючи один за одним кілька шарів згортки і пулінгу, будується згорткова нейронна мережа. Чергування шарів дозволяє складати

карти ознак з карт ознак, що на практиці означає здатність розпізнавання складних ієрархій ознак. Зазвичай після проходження декількох шарів карта ознак вироджується у вектор або навіть скаляр, але такі карти ознак налічують сотні.

На виході мережі часто додатково встановлюють декілька повнозв'язних шарів (fully connected layer), на вхід яких подаються кінцеві карти ознак. Якщо на першому шарі ядро згортки проходить тільки одним вихідним зображенням, то на внутрішніх шарах одне і те ж ядро проходить паралельно всіма картами ознак цього шару, а результат згортки підсумовується, формуючи (після проходження функції активації) одну карту ознак наступного шару, що відповідає цьому ядру згортки.

Функція активації визначає вихідне значення нейрона в залежності від результатів зваженої суми входів і порогового значення. Існує багато функцій активації, зокрема ReLU, tanh, linear, softmax, sigmoid та різні їх варіації. Найбільш поширеною практикою є використання функції активації ReLU між згортковими шарами.

2.2.1 Згортковий шар

Центральним для згорткової нейронної мережі є згортковий шар, завдяки якому мережа дістала свою назву. Цей шар виконує операцію, яка називається згорткою. У контексті згорткової нейронної мережі згортка – це лінійна операція, яка передбачає множення набору ваг із вхідним потоком даних, подібно до традиційної нейронної мережі. Основна мета полягає у зменшенні зображень у форму, яку легше обробити, не втрачаючи ознак, які є критично важливими для отримання задовільненого прогнозу. Це важливо, коли ми хочемо розробити архітектуру, яка не тільки добре володіє функціями навчання, але й може масштабуватися до масивних наборів даних.

Враховуючи, що методика була розроблена для двовимірних вхідних даних, множення виконується між вхідним потоком даних та матрицею ваг, яка зветься фільтром або ядром. Зазвичай ядро згортки є квадратною матрицею $n \times n$, де n – непарне.

Зазвичай вхідний потік для згорткового шару складається з d каналів, наприклад, red / green / blue для вхідного шару (рис. 2.2), і в цьому випадку ядра теж розширюють таким чином, щоб вони також складалися з d каналів.

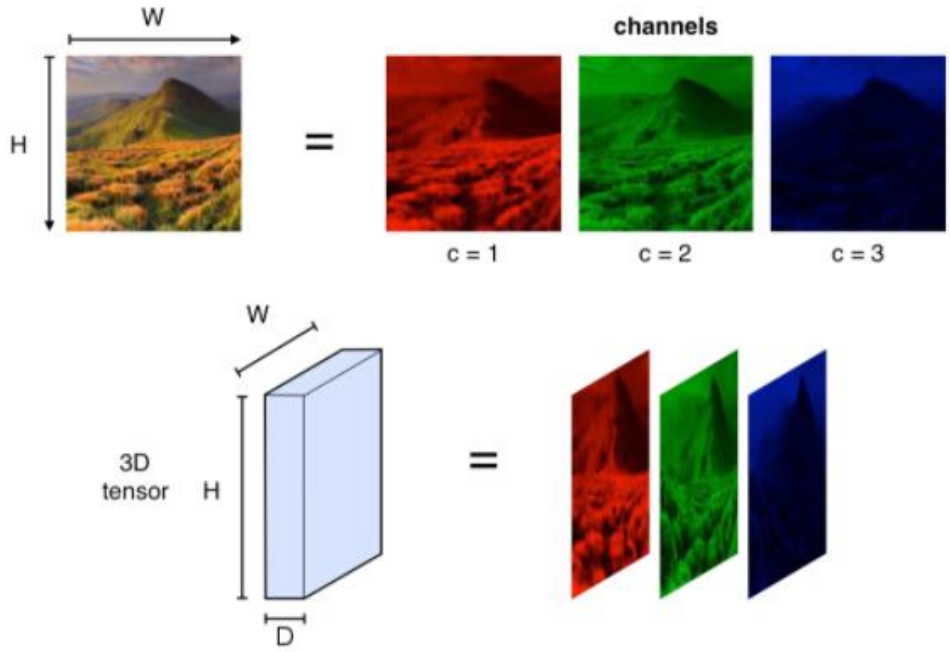

Рисунок 2.2 – RGB зображення у першому шарі CNN

Розмір фільтра K менший за вхідні дані I , він накладається на вхідні дані як маска – у результаті маємо фрагмент такого ж розміру. Далі фрагмент та фільтр помножуються і в результаті отримаємо скалярний добуток. він отримується шляхом множення відповідних елементів двох матриць, потім результат підсумовується і, таким чином результатом завжди є єдине значення.

Використання фільтра, меншого за вхідні дані, є навмисним, оскільки дозволяє множити той самий фільтр (набір вагових коефіцієнтів) на вхідний масив кілька разів у різних точках входу. Зокрема, фільтр застосовується

систематично до кожної частини, що перекривається, або ділянки вхідних даних розміром із фільтр, зліва направо, зверху вниз. Якщо застосувати згортку до кожного пікселя зображення, то в результаті вийде певний ефект, що залежить від обраного ядра згортки. Приклад застосування операції згортки до вхідних даних показано на рисунку 2.3.

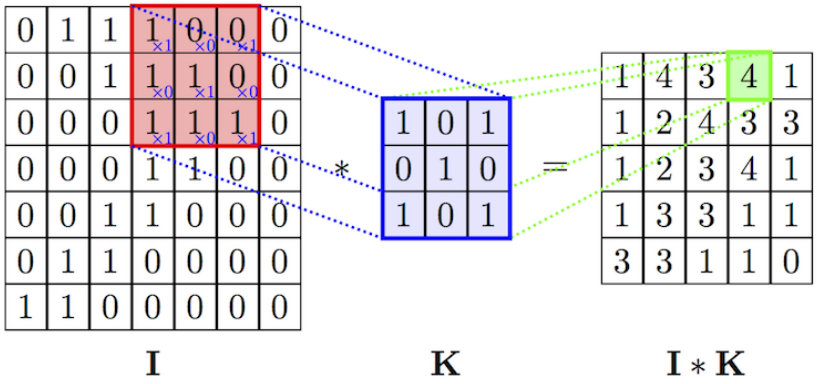

Рисунок 2.3 – Операція згортки

Результатом множення фільтра з вхідним масивом кожен раз є одне значення. Оскільки фільтр застосовується кілька разів до вхідного масиву, результатом є двовимірний масив вихідних значень $I \times K$, що представляють фільтрування вхідних даних. Таким чином, двовимірний вихідний масив цієї операції називається «картою ознак» (feature map), приклад таких карт наведено на рисунку 2.4.

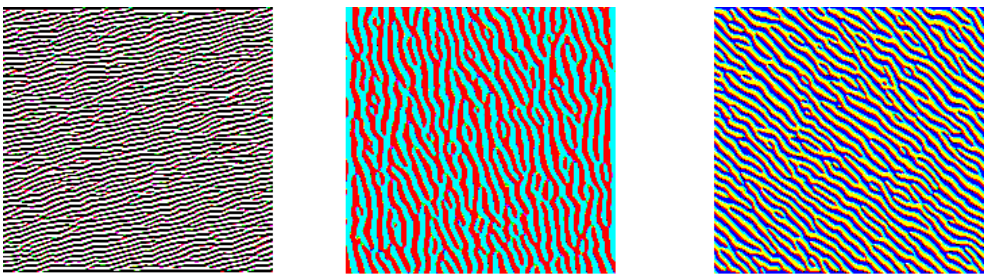

Рисунок 2.4 – Приклади карт ознак як результат роботи першого слою згортки

Систематичне застосування одного і того ж фільтра на зображенні є дуже потужним механізмом. Якщо фільтр призначений для виявлення певного

типу ознак у вхідних даних, то застосування цього фільтра систематично усім вхідним зображенням дає можливість фільтру виявити цю функцію в будь-якому місці зображення. Цю можливість зазвичай називають інваріантністю перекладу, наприклад загальний інтерес до того, чи присутня ознака, а не де вона присутня. Під час операції згортки часто застосовують дві техніки, так звані Padding та Striding, від вибору техніки буде залежати розмір вихідних даних [18].

Padding. Як видно на рисунку 2.5, під час ковзання, краї обрізаються, перетворюючи матрицю ознак розміром 5×5 у матрицю 3×3. Крайні пікселі ніколи не опиняються у центрі ядра, тому що тоді ядру нема над чим буде ковзати за краєм. Це не задовільний варіант, коли потрібно, щоб розмір на виході дорівнював вхідному. Padding додає до країв підроблені (fake) пікселі (зазвичай нульового значення, внаслідок цього до них застосовується термін «ульовий додаток» – «zero padding»). Таким чином, ядро під час ковзання дозволяє непідробним пікселям знаходитися у своєму центрі, а потім поширюється на підроблені пікселі за межами краю, створюючи вихідну матрицю того ж розміру, що й на вході.

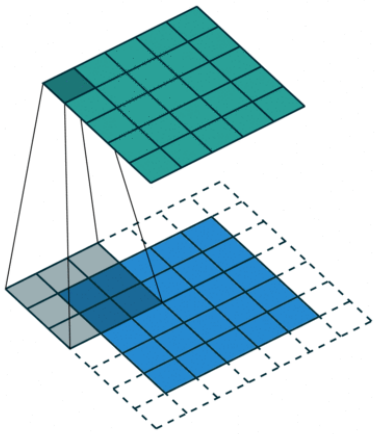

Рисунок 2.5 – Приклад техніки Padding

Striding. Застосовується у випадках коли потрібно отримати вихідні дані меншого розміру, ніж вхідні. У згорткових нейронних мережах, розмір

просторових розмірів зменшується при збільшенні кількості каналів. Цього можна досягти декількома способами, зокрема, використовувати шари субдискретизіції (pooling layer), наприклад, приймати середнє / максимальне значення кожної гілки розміром 2×2, для зменшення всіх просторових розмірів у два рази. Альтернативний спосіб – використовувати stride (крок).

Ідея stride полягає у тому, щоб пропустити деякі області, якими ковзає ядро. Крок 1 означає, що зсув складає один піксель, крок 2 – що зсув складає два пікселя, пропускаючи всі інші прольоти в процесі і зменшуючи їх кількість приблизно в 2 рази, крок 3 означає пропуск трьох пікселів, скорочуючи кількість у 3 рази тощо. Приклад stride з кроком 2 наведено на рисунку 2.6.

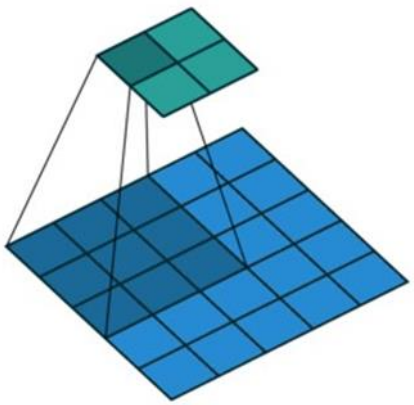

Рисунок 2.6 – Stride з кроком 2

Варто також відзначити, що незважаючи на те, що згортковий шар скорочує кількість параметрів у порівнянні з повноз’єднаним шаром, він використовує більше гіперпараметрів – параметрів, які обирають до початку навчання. Зокрема, вибираються такі гіперпараметри:

– глибина (depth) – кількість ядер і коефіцієнтів зміщення, котрі будуть задіяні в одному шарі;

– висота (height) і ширина (width) кожного ядра;

– крок (stride);

– відступ (padding).

2.2.2 Шар пулінгу

Подібно до згорткового шару, шар об’єднання (субдискретизації) відповідає за зменшення просторового розміру ознаки, над якою застосовується згортка. Це має зменшити обчислювальну потужність, необхідну для обробки даних, завдяки зменшенню розмірності. Крім того, це корисно для вилучення домінантних ознак, які є інваріантними щодо обертання та позиції, таким чином підтримуючи ефективний процес навчання моделі.

Існує два типи об’єднання: максимальне об’єднання (Max Pooling) та середнє об’єднання (Average Pooling) [19]. Перше повертає максимальне значення частини зображення, охопленої ядром, тоді як друге повертає середнє значення всіх значень із частини зображення, охопленої ядром.

Max Pooling також виконує функцію придушення шуму. Він взагалі відкидає активізації шумів, а також зменшує рівень шуму разом із зменшенням розмірності. З іншого боку, Average Pooling придушує шум просто за допомогою зменшення розмірності. Отже, можна сказати, що Max Pooling працює набагато краще, ніж Average Pooling. Обидва типи наведено на рисунку 2.7.

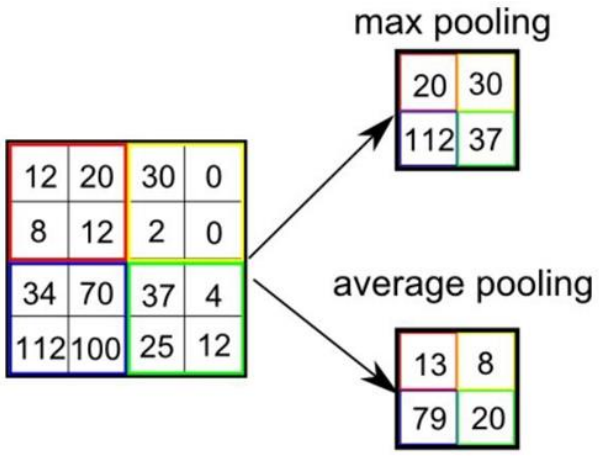

Рисунок 2.7 – Max Pooling та Average Pooling

Згортковий шар та шар об’єднання разом утворюють *i*-й шар згорткової нейронної мережі, один такий прохід впливає на зображення наступним чином: він скорочує довжину і ширину певного каналу, але збільшує його значення (глибину). Залежно від складності зображень, кількість таких шарів може бути збільшена для подальшого захоплення деталей низького рівня, але ціною більшої обчислювальної потужності. Вищевказаний процес дозволяє моделі виділяти ознаки. Для подальшого шару потрібно зробити перетворення двовимірних матриць в одномірні вектори. Даний вектор формується з матриці шляхом запису її елементів через підрядник в один рядок, як показано на рисунку 2.8. Результат цього перетворення подається на вхід наступному шару.

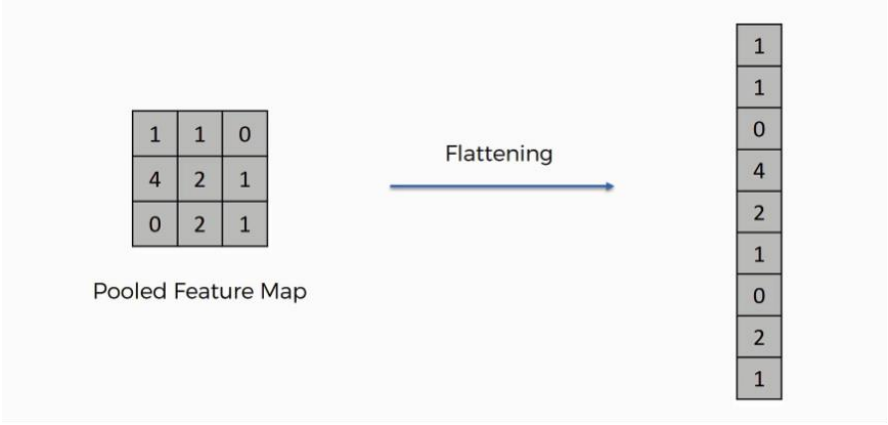

Рисунок 2.8 – Формування вхідних даних для повноз’єднаного шару

2.2.3 Повноз’єднаний шар

Додавання повноз’єднаного шару – це, зазвичай, простий спосіб вивчення нелінійних комбінацій особливостей високого рівня, представлених результатами згорткового шару. Він навчається використовуючи нелінійну функцію в цьому просторі. Результатом попереднього етапу є одномірний вектор, який подається в нейронну мережу прямого поширення зв’язку, застосовуючи алгоритм зворотного поширення помилки на кожній ітерації

навчання. Протягом серії епох модель здатна розрізнити домінуючі та певні низько рівневі ознаки у зображеннях та класифікувати їх, використовуючи функцію Softmax для классифікації. Приклад шару наведено на рисунку 2.9.

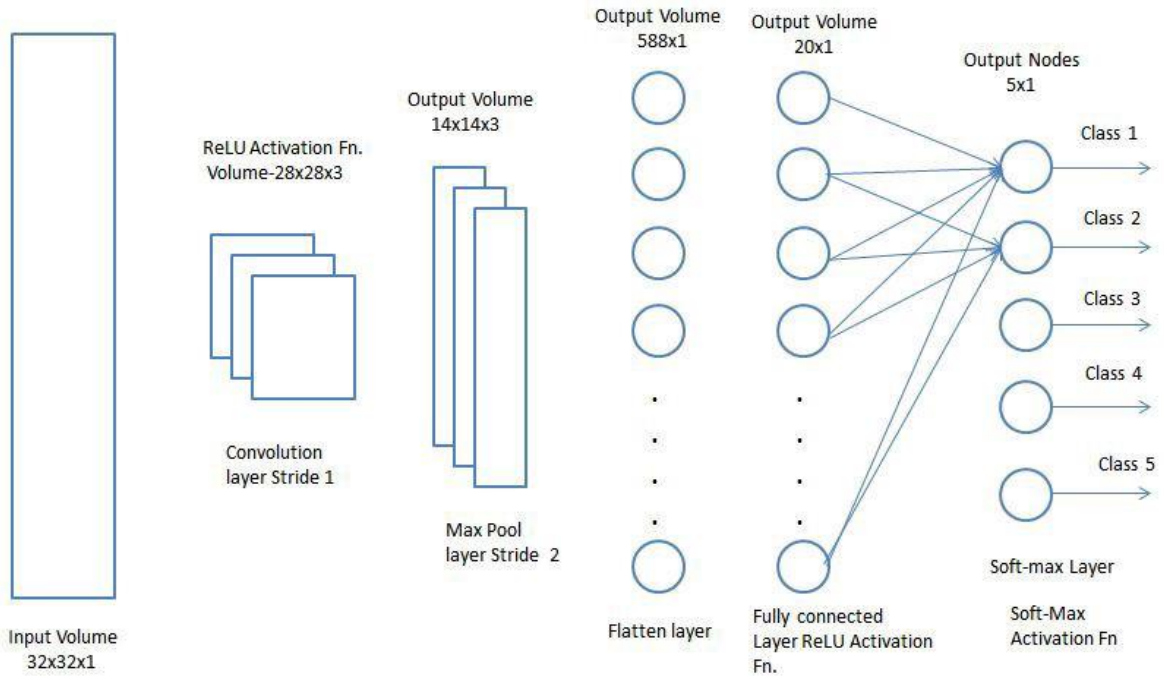

Рисунок 2.9 – Повноз’єднаний шар

2.3 Навчання нейронної мережі

Головні проблеми, які виникають в процесі навчанні моделі – це проблеми недостатнього навчання(underfitting) та перенавчання (overfitting) моделі. У першому випадоку модель має низький рівень точності на навчальному наборі, тобто недостатньо добре описує навчальний набір. Причиною цієї проблеми можуть бути різні, серед них неправильно побудована архітектура моделі, або ж через недостатню кількість даних для навчання – модель не змогла «вивчити» закономірності та характерні ознаки.

Перенавчання моделі – це явище, коли з моделі на тестовому наборі істотно вище помилки на навчальному наборі. У цьому випадку мережа не вчиться відокремлювати ознаки та будувати шаблони, а просто запамятовує правильні відповіді, іншими словами модель добре розрізняє вже знайомі дані,

але на нових точність істотно падає [20]. На рисунку 2.10 наведено приклад перенавчання моделі: видно, що значення помилки з часом починає збільшуватися на тестовому наборі даних, у той час як продовжує знижуватися на тренувальному.

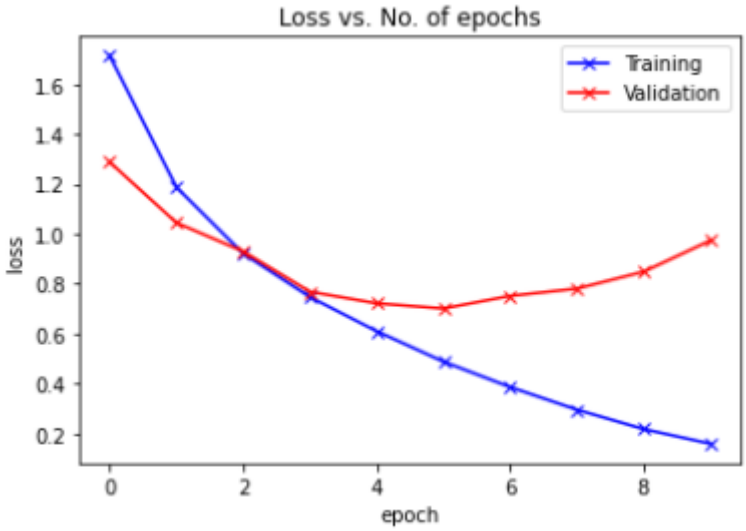

Рисунок 2.10 – Приклад перенавчання

Розповсюджені наступні способи боротьби із перенавчанням:

– збільшення даних(data augmentation);
– використання техніки виключення(dropout);
– додавання даних.

2.3.1 Збільшення даних

Вперше цей підхід також було запроваджено у нейронній мережі AlexNet: вони використовували методи збільшення даних, таких як складалися з поворотів зображень, горизонтальних відбиттів та середнього віднімання [14]. Ці методи дуже широко використовуються сьогодні для багатьох завдань комп’ютерного зору. Приклади цієї техніки наведено на рисунку 2.11.

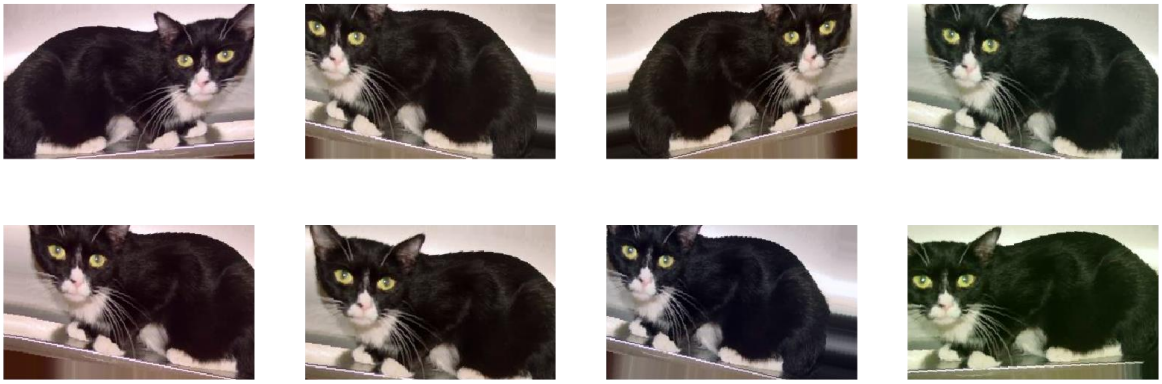

Рисунок 2.11 – Приклад збільшення даних

Збільшення даних включає такі прийоми, як випадкове обертання зображення, відображення, додавання кольорового фільтра, обрізання тощо. Збільшення даних застосовується лише на навчальному наборі, але не на тестовому. Основною метою є створення нових даних на основі існуючих. Таким чином, тренувальний набір стає більшим та різноманітнішим, в наслідок цього мережа здатна вивчити більше ознак, що позитивно впливає на точність. Наприклад, використання повороту зображення дозволяє у майбутньому розпізнати кота, зображеного не тільки у профіль, а й напівбоком.

У той час існують перетворення, яких слід уникати, а саме наближення, бо можна наблизити так сильно, що риси кота вже не буде видно і модель не стане кращою від тренувань на цих зображеннях.

2.3.2 Шар виключення як засіб боротьби із перенавчанням

Виключення(dropout) – це метод регуляризації, який випадково «виключає» або «деактивує» кілька нейронів у нейронній мережі, щоб уникнути проблеми перенавчання.

Основна ідея полягає у наступному: кожному нейрону повноз’єднаного шару(крім тих, що належать до вихідного) надається ймовірність p тимчасово на випадковості ігноруватися в розрахунках [21]. Гіперпараметр p називається

коефіцієнтом відсіву, і дуже часто значенням за замовчуванням є 0,5. Потім на кожній ітерації випадковим чином відбираються нейрони, які будуть відкинуті відповідно до призначеної ймовірності. Як результат, кожного разу ми працюємо з меншою нейронною мережею. На рисунку 2.12 продемонстровано приклад нейронної мережі, яка зазнала відсіву.

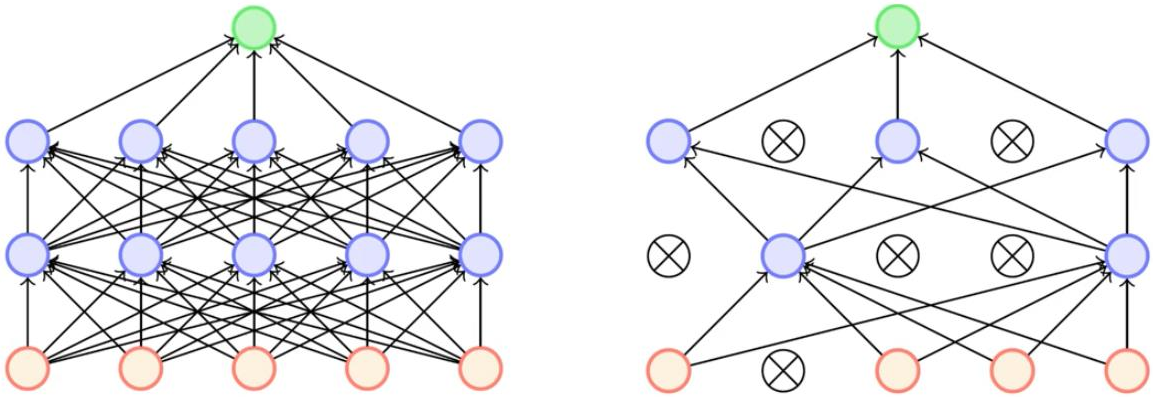

Рисунок 2.12 – Використання dropout у повноз’єднаних шарах

Давайте розглянемо цю проблему з точки зору окремого нейрона. Оскільки в кожній ітерації будь-яке введене значення може бути випадковим чином усунено, нейрон намагатиметься збалансувати ризик, а не надавати перевагу жодній з характеристик. В результаті значення у ваговій матриці стають більш рівномірно розподіленими. Таким чином, модель уникає ситуації, коли рішення, яке вона пропонує, більше не має сенсу, оскільки інформація з неактивних функцій більше не надається.

Також, попри корисність цього підходу, він може стати причиною проблеми недостатнього навчання моделі через наступну причину. Виключення відбувається під час тренувального процесу, щоб уникнути переобладнання, але не відбувається під час прогнозування на тестовому наборі. Якщо це так, потрібно видалити dropout і додавати dropout поступово до отримання прийнятної точності.

2.3.3 Метод навчання Adam

Вибір алгоритму оптимізації для моделі може означати різницю між хорошими результатами в хвилинах, годинах і навіть днях. Класичним вибором в якості методу навчання є алгоритм зворотного поширення помилки.

Алгоритм зворотного поширення помилки є одним із методів навчання багатошарових нейронних мереж прямого поширення. Навчання за допомогою цього алгоритма передбачає два проходи всіма шарами мережі: прямого і зворотного. Під час прямого проходу вхідний вектор подається на вхідний шар нейронної мережі, після чого поширюється по мережі від шару до шару. В результаті генерується набір вихідних сигналів, який і є фактичною реакцією мережі на даний вхідний образ. Під час прямого проходу всі синаптичні ваги мережі фіксовані. Під час зворотного проходу всі синаптичні ваги налаштовуються відповідно до правила корекції помилок, а саме: фактичний вихід мережі віднімається від бажаного, в результаті чого формується сигнал помилки.

Цей сигнал згодом поширюється мережею у напрямку, протилежному напрямку синаптичних зв’язків, саме тому метод має назву – алгоритм зворотного поширення помилки. Синаптичні ваги налаштовуються з метою максимального наближення вихідного сигналу мережі до бажаного.

Алгоритм оптимізації Адам(Adam) – це розширення методу стохастичного градієнтного спуску, яке нещодавно стало широко застосовуватися для програм глибокого навчання в комп’ютерному зорі та обробці природних мов [22]. Автори описують його як поєднання переваг двох інших розширень стохастичного градієнтного спуску. Зокрема:

– адаптивний алгоритм градієнта (AdaGrad), який підтримує швидкість навчання з параметром, що покращує ефективність роботи під час виникнення проблем із рідкими градієнтами (наприклад, проблеми з природною мовою та комп’ютерним зором);

– алгоритм середньоквадратичного коріння (RMSProp), який також підтримує коефіцієнти навчання з параметрами, які адаптуються використовуючи середнє значення останніх величин градієнтів для ваги (наприклад, наскільки швидко вона змінюється). Це означає, що алгоритм стікий до нестаціонарних проблемам, наприклад, шумів.

Адам реалізує переваги як AdaGrad, так і RMSProp. Замість того, щоб адаптувати швидкість навчання параметрів на основі середнього першого моменту (середнього значення), як у RMSProp, Адам також використовує середнє значення моментів другого порядку градієнтів (нецентрова дисперсія). Зокрема, алгоритм обчислює експоненціальну ковзке середнє для градієнта та квадратичний градієнт, а параметри *beta1* та *beta2* контролюють швидкість спаду цих ковзких середніх. Початкові значення ковзких середніх та значень *beta1* та *beta2*, близькі до одиниць (рекомендовані), призводять до зміщення оцінок моменту до нуля. Цей зсув долається спочатку обчисленням упереджених оцінок, а потім обчисленням, скоригованим із зміщенням. На рисунку 2.13 наведено порівняння Adam та інших алгоритмів навчання у багатошаровому перцептроні.

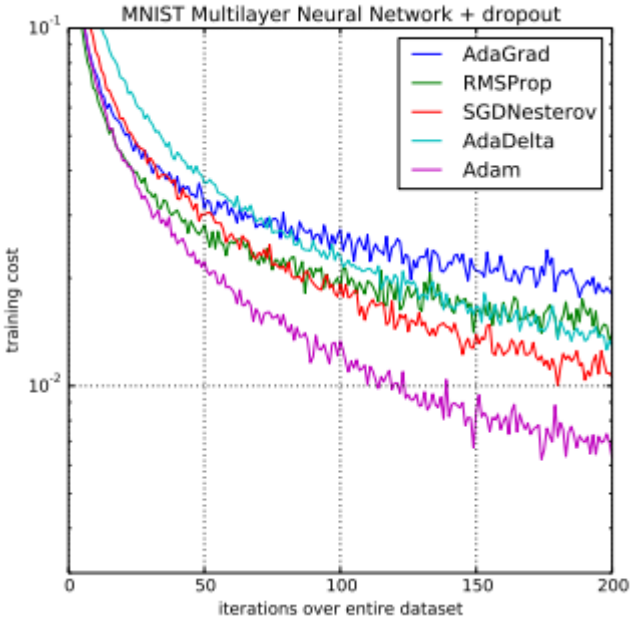

Рисунок 2.13 – Порівняння Adam з іншими алгоритмами навчання у багатошаровому перцептроні

У оригінальній роботі Адам був застосований до алгоритму логістичної регресії на наборах даних для розпізнавання цифр MNIST та аналізу настроїв IMDB багатошарового перцептрона на наборі даних MNIST та згорткових нейронних мереж на наборі даних для розпізнавання зображень CIFAR-10. Вони дійшли висновку, що використовуючи великі моделі та масиви даних, Адам ефективніше вирішує практичні проблеми глибокого навчання.

2.4 Оператори виділення границь

В якості виділення границь біло вирішено використовувати матричні фільтри, тому що вони дієві, досить швидкі та можуть бути використані у згортковому слої в якості фільтра.

Методи, що використовують диференційні оператори, ґрунтуються на одному з базових властивостей сигналу яскравості – розривності. Найбільш загальним способом пошуку розривів є обробка зображення за допомогою ковзної маски, яку називають також фільтром або ядром та представляє собою деяку квадратну матрицю, що відповідає зазначеній групі пікселів оригінального зображення. При виявленні перепадів яскравості використовуються дискретні аналоги похідних першого і другого порядку. Перша похідна одновимірної функції *f(x)* визначається як різниця значень сусідніх елементів:

$$\frac{\delta f}{\delta x} = f(x+1) - f(x). \qquad (2.1)$$

Тут використано запис у вигляді часткової похідної для того, щоб зберегти ті ж позначення у разі двох змінних *f(x,y)*, де доведеться мати справу з частковими похідними по двом просторовим осям. Друга похідна визначається як різниця сусідніх значень першої похідної:

$$\frac{\delta^2 f}{\delta x} = f(x+1) - f(x-1) - 2f(x). \quad (2.2)$$

Обчислення першої похідної цифрового зображення засноване на різних дискретних наближеннях двовимірного градієнта. За визначенням, градієнт зображення *f (x, y)* в точці *(x, y)* – це вектор [11]:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{bmatrix}. \quad (2.3)$$

Як відомо, напрямок вектора градієнта збігається з напрямком максимальної швидкості зміни функції *f* у точці *(x, y)* [11]. Важливу роль при виявленні контурів відіграє модуль цього вектора, який позначається ∇f і дорівнює

$$\nabla f = |\nabla f| = \sqrt{G_x^2 + G_y^2}\ . \quad (2.4)$$

Ця величина дорівнює значенню максимальної швидкості зміни функції *f* у точці *(x, y)*, причому максимум досягається в напрямку вектора ∇f. Величину ∇f також часто називають градієнтом. Напрямок вектора градієнта також є важливою характеристикою. Позначимо α як кут між напрямком вектора ∇f в точці *(x,y)* і віссю *x* [11]. Його можна обчислити за формулою:

$$\alpha(x, y) = arctg\left(\frac{G_x}{G_y}\right). \quad (2.5)$$

Звідси легко знайти напрям контуру в точці *(x,y)*, який перпендикулярний до напрямку вектора градієнта у цій точці. Обчислити

градієнт зображення можна, обчисливши величини часткових похідних *δf/δx* та *δf/δy* для кожної точки [23]. Далі будуть розглянуті найбільш часто використовувані оператори, а саме Робертса, Прюітта. Собеля. Усі перелічені далі оператори працюють з півтоновим зображенням.

Оператор Робертса. Оператор виділення границь Робертса був введений Лоуренсом Робертсом в 1964 році. Він виконує прості та швидкі обчислення двовимірного просторового виміру на зображенні. Цей метод підкреслює області високої просторової частоти, які часто відповідають межам. Значення пікселів вихідного зображення в кожній точці передбачає якусь величину просторового градієнта вхідного зображення (значення яскравості) у цій же точці в околиці деякої області зображення. Уявімо область зображення 3×3 у вигляді $\begin{pmatrix} z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 \\ z_7 & z_8 & z_9 \end{pmatrix}$, у такому разі для знаходження перших часткових похідних у точці z_5 застосуємо перехресний градієнтний оператор Робертса [24]:

$$G_x = (z_9 - z_5). \tag{2.6}$$

$$G_y = (z_8 - z_6). \tag{2.7}$$

Похідні також можна реалізувати шляхом обробки всього зображення за допомогою оператора, описуваного масками виду:

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.8}$$

$$G_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{2.9}$$

Проте, реалізація масок розмірами 2×2 не дуже зручна, тому що вони не мають чітко вираженого центрального елемента, що значною мірою впливає на результат виконання фільтрації, але при цьому має досить високу швидкість обробки зображення.

Оператор Прюітта. Цей оператор також оперує областю зображення 3×3, тільки використання такої маски задається іншими виразами:

$$G_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3). \tag{2.10}$$

$$G_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7). \tag{2.11}$$

У цих формулах різниця між сумами верхнього та нижнього рядків околиці 3×3 є наближеним значенням похідної по осі *x*, а різниця між сумами першого та останнього стовпця цієї околиці – похідною по осі *y*. Для реалізації цих формул використовується оператор, що використовує маски:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix}. \tag{2.12}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}. \tag{2.13}$$

Варто відзначити, що недоліком оператора Прюітта є його чутливість до шуму на зображенні.

Оператор Собеля. Межі відзначаються там, де градієнт набуває максимального значення. Вони можуть мати різне спрямування, тому алгоритм Канні використовує чотири фільтри для виявлення горизонтальних, вертикальних і діагональних ребер розмитого зображення. Оператор Собеля

використовує дві маски розмірності 3×3. Основою є згортка зображення невеликими цілочисельними фільтрами у вертикальному і горизонтальному напрямках, тому його відносно легко обчислювати. Ядра фільтрів мають наступний вигляд:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{2.14}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{2.15}$$

Величина градієнта і його напрямок визначаються за формулами (2.4) та (2.5) відповідно. Кут напрямку вектора градієнта округляється і може приймати такі значення: 0, 45, 90, 135. Наприклад, якщо кут лежить між 1 до 20, то він відноситься до значення 0, а якщо він більше 20, то до значення 45.

Оператор Шарра. Цей оператор є модифікацією оператора Собеля, що мала знизити негативні ефекти попередника кластерами [25]. В отриманих матрицях згортки ваги центральних пікселів перевершують ваги крайніх пікселів в 3,3 рази. Ядра фільтрів виглядають наступним чином:

$$G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}. \tag{2.16}$$

$$G_x = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}. \tag{2.17}$$

Оператор Лапласа. Оператор є найпростішим ізотропним оператором, тобто інваріантним до повороту, в тому сенсі, що поворот зображення і подальше застосування фільтра дає той самий результат, що і початкове застосування фільтра із подальшим поворотом [26]. Оператор заснований на обчисленні похідних та у двомірному просторі визначається наступним чином:

$$\nabla^2 f = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2}. \tag{2.18}$$

Для розрахування лапласиану використовуються часні похідні другого порядку:

$$\begin{aligned} \frac{\delta^2 f}{\delta x^2} &\approx ((f(x+1,y) - f(x,y)) - (f(x,y) - f(x-1,y))) = \\ &= f(x+1,y) - 2f(x,y) + f(x-1,y), \end{aligned} \tag{2.19}$$

$$\begin{aligned} \frac{\delta^2 f}{\delta y^2} &\approx ((f(x,y+1) - f(x,y)) - (f(x,y) - f(x,y-1))) = \\ &= f(x,y+1) - 2f(x,y) + f(x,y-1). \end{aligned} \tag{2.20}$$

Сума яких визначається наступним чином:

$$\begin{aligned} \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2} &= f(x+1,y) - f(x-1,y) - \\ -4f(x,y) &+ f(x,y+1) + f(x,y-1). \end{aligned} \tag{2.21}$$

Рівняння 2.21 може бути реалізовано за допомогою маски, представленої формулою 2.22, яка дає ізотропний результат для поворотів на кути, кратні 90°.

$$\nabla^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \qquad (2.22)$$

Діагональні напрямки можуть бути включені в формулу дискретного Лапласіана шляхом додаванням ще двох членів – по одному для кожного з діагональних напрямків попередника. Кожен з них має вигляд, як у рівнянні 2.19 або 2.20 відповідно, але вказуються координати точок, що розташовані по діагоналях. Оскільки кожне додавання діагонального елемента включає член, $-2f(x,y)$, то сумарний віднімаємий з суми член складе $-8f(x,y)$. Маска, яка відповідає новому визначенню, визначається формулою 2.23. Такий фільтр є ізотропноим для поворотів на кути, кратні 45°.

$$\nabla^2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \qquad (2.23)$$

Оскільки оператор Лапласа по суті є другою похідною, його застосування підкреслює розриви рівнів яскравості на зображенні і пригнічує області зі слабкими змінами яскравості. Це призводить до отримання зображення, що містить сіруваті лінії на місці контурів і інших розривівна темному тлі. Але тло можна «відновити», зберігши при цьому ефект підвищення різкості, що досягається лапласіаном.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ДОСЛІДЖЕНЬ МОДЕЛЕЙ НЕЙРОННОЇ МЕРЕЖІ

3.1 Середовище розробки

Як було зазначено у попередньому розділі, один із найважливіших факторів для розробки нейронної мережі є час навчання. Використання CPU задля навчання мережі є неефективним, тож потрібно, щоб машина, на якій буде проводитися навчання, мала потужний графічний процессор, на який можна б було перенести трудомісткі обчислення. На щастя, компанія Google розробила безкоштовне хмарне середовище розробки, яке має назву Colaboratory.

Середовище дозволяє отримати видалений доступ до машини з підєднаним графічним процесором NVIDIA Tesla K80, що робить машинне навчання доступним для всіх. Середовище дозволяє працювати з документами Jupiter Notebook (.ipynb). Цей формат дозволяє створювати комірки з кодом на мові програмування Python і текстові, які також можуть включати графічну інформацію. Jupiter Notebook також дозволяж встановлювати будь-які пакети та бібліотеки для Python, у тому числі популярні бібліотеки та фреймворки машинного навчання, такі як Tensorflow, Keras та PyTorch.

3.1.1 Огляд мови програмування Python

Мова програмування Python 3.7 є високорівневою мовою програмування, головна особливість якої полягає в її простоті. Вона орієнтована на підвищення продуктивності розробника і забезпечує легку читаність коду. Основні архітектурні риси включають в себе: динамічну типізацію, автоматичне керування пам’яттю, механізм обробки виключень та підтримку багатопотокових обчислень. Python є портованою мовою, це

означає, що він може виконуватися майже на всіх відомих платформах. Коли платформа застаріває, нові версії Python перестають її підтримувати (оновлення виходять раз у два роки), але на таких платформах все ще можна використовувати старі версії Python. Ще однією особливістю мови є відсутність операторних дужок, виділення блоків коду відбувається за рахунок відступів або табуляцій, що дозволяє скоротити код. Зважаючи на те, що Python є інтерпретуючою мовою – всі рядки в ньому виконуються послідовно (що повільно), але існують і компілятори, наприклад PyPy.

PyPy спочатку був інтерпретатором Python написаним на Python, що дозволяло легше розширювати функціональність, але незабаром PyPy став компілятором. Ідея полягає в тому, щоб підвищити продуктивність, тому що інтерпретація виконується по рядках і нездатна досить оптимізувати програму, тоді як все повинно робитися «на льоту». При компіляції такої проблеми не виникає, бо оптимізація відбувається саме у цей час, після чого на виході отримаємо готовий додаток, готовий до запуску і не потребуючий повторної компіляції або інтерпретації. PyPy не компілює Python, а транслює код в мову C і компілює вже його, саме це є причиною такої швидкої роботи застосувань.

3.1.2 Огляд можливостей фреймворка PyTorch

PyTorch – фреймворк машинного навчання для мови Python з відкритим вихідним кодом, створений на базі Torch командою розробників Facebook. Використовується для вирішення різних завдань машинного навчання, таких як комп’ютерний зір, обробка мови та багато інших. На відміну від інших популярних рішень, таких як Tensorflow та Keras, має наступні переваги:

– краща підтримка Python;

– більш легких процесс відлагодження;

– тісна інтеграція з такими важливими бібліотеками, як numpy, scipy, scikit-learn та ін.

На відміну від TensorFlow, PyTorch дає можливість створювати свої шари, функції активації та інші потрібні об’єкти у вигляді звичайних класів і функцій Python.

Операції над тензорами можуть виконуватися на GPU, що значно підвищує швидкість навчання мережі. PyTorch тензор – особливий тип даних, який використовується в бібліотеці для всіх операцій з даними і вагами всередині нейронної мережі. По суті, тензор є проста багатовимірною матрицею.

Для преносу обчислень на графічний процесор досить одного рядка: device = torch.device ("cuda"). Щоб в подальшому використовувати саме цей пристрій, досить вказати його як аргумент при ініціалізації тензорів наступним чином: torch.randn(10, 10, device = device).

PyTorch використовує автоматичне диференціювання для обчислень градієнтів, які використовуються в алгоритмі зворотного поширення помилки (backpropogation) – цей підхід називається автоградієнтом(autograd) [27]. Перевагою такого підходу є швидкість обчислення. Для його активації слід аргументу тензора *requires_grad* встановити значення True насутпним чином: torch.randn(2, 2, requires_grad=True). Також слід відзначити велику кількість пакетів, які постачаються разом із фреймворком, деякі з них будуть згадані нижче.

3.1.3 Огляд набору даних CIFAR-10

Набір даних (Dataset) CIFAR10 поставляється разом з пакетом torchvision, який є частиною PyTorch та має 50 000 навчальних образів і 10000 тестових зображень 10 класів, таких як літаки, автомобілі, птиці, коти, олені, собаки, жаби, коні, кораблі і вантажівки) [28]. На рисунку 3.1 наведено програмний код, за допомогою якого можна здійснити завантаження і підготовку даних CIFAR-10 для подальшої обробки за допомогою нейронних

мереж. CIFAR-10 міститься у пакеті torchvision.datasets, який є частиною PyTorch.

```
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize([0, 0, 0], [1, 1, 1])
])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0, 0, 0], [1, 1, 1])
])

dataset = CIFAR10(root='data/', download=True, transform=train_transform)
test_dataset = CIFAR10(root='data/', train=False, transform=test_transform)

batch_size = 128
train_ds = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_ds = DataLoader(val_ds, batch_size*2, num_workers=4, pin_memory=True)
```

Рисунок 3.1 – Код загрузки CIFAR-10 dataset та розбиття на тестовий та тренувальний набори даних

На етапі завантаження можна зробити модифікації даних, відповідно до параметра transform. Базовим та обвязковим перетворенням є перетворення зображення у тензор за допомогою функції ToTensor. Як було зазначено раніше, всі операції у PyTorch використовують тензори, тож без цього подальше використання фреймворку неможливе.

Інші перетворення є необовязковими та застосовуються з метою покращення точності роботи мережі. В даному випадку використано такі перетворення: RandomHorizontalFlip (горизонтальний поворот), RandomCrop(обрізає зображення у довільному місці.) та Normalize(нормалізує тензорне зображення із середнім та стандартним відхиленням для кожного каналу). Перетворення об’єднуються за допомогою функції Compose. Усі перетворення використано із пакету transforms, який також є частиною фреймворка PyTorch.

Зображення у наборі даних мають розмір 32×32. На рисунку 3.2 наведено кілька прикладів зображень для кожного класу.

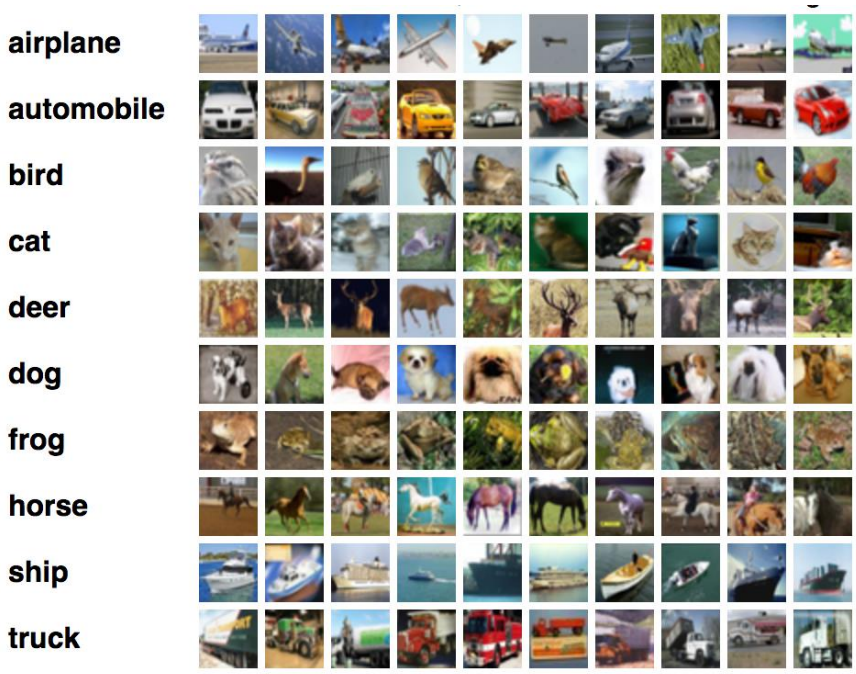

Рисунок 3.2 – Приклади зображень з набору даних CIFAR-10

Усі зображення поділяються на навчальний набір (*train_ds*), валідаційний (*val_ds*) та тестовий (*test_dataset*), розмір яких 45000, 5000 та 10000 відповідно. Навчальний набір містить 5000 зображень кожного класу, валідаційний по 500, та тестовий по 1000, відповідно. Навчальний набір використовується для тренування моделі, обчислень втрат та регулювання ваг мережі, валідаційний – для оцінки моделі з гіперпараметрами та вибору найкращої моделі під час навчання, тестовий – для порівняння різних моделей та звітування про остаточну точність.

3.2 Створення моделі

Найпростіший спосіб створити структуру нейронної мережі в PyTorch – створити власний клас від базового класу nn.Module. Клас nn.Module містить усе необхідне для конструювання типової глибокої

нейронної мережі. Крім цього, клас має зручні функції: способи переміщення змінних та операцій на GPU або навпаки на CPU, застосування рекурсивних функцій (наприклад, перебудова всіх вагових коефіцієнтів), створення оптимізованих інтерфейсів для тренувань тощо.

Як можна бачити на рисунку 3.3, модель складається з трьох великих послідовностей виду згортка – ReLU – згортка – ReLU – пулінг.

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3,stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Flatten(),
            nn.Linear(256 * 4 * 4, 1024),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(512, 10))

    def forward(self, x):
        return self.network(x)
```

Рисунок 3.3 – Код створення моделі

Згортковий шар визначається за допомогою функції nn.Conv2d, яка в якості першого аргумента приймає кількість вхідних каналів, у якості

другого – кількість вихідних каналів; аргумент *kernel_size* визначає розмір ядра згортки, що буде застосовано до вхідних даних; *padding* визначає скільки пікселів буде додано до зображення (детально був розлядений у розділі 2.2.1); *stride* визначає крок зміщення (детально був розлядений у розділі 2.2.1).

На вхід першого згорткового шару мережі подається тензор виду $[B,C,H,W]$, де B – розмір партії данних для навчання (*batch_size*), C – кількість каналів , H – висота зображення у пікселях, W – ширина зображення у пікселях. В даному випадку C дорівнює трьом, бо зображення має 3 канали (RGB), число вихідних каналів дорівнює 32, розмір ядра згортки 3×3, *padding stride* дорівнюють одиниці. Візуалізація фільтрів першого згорткового шару зображена на рисунку 3.4.

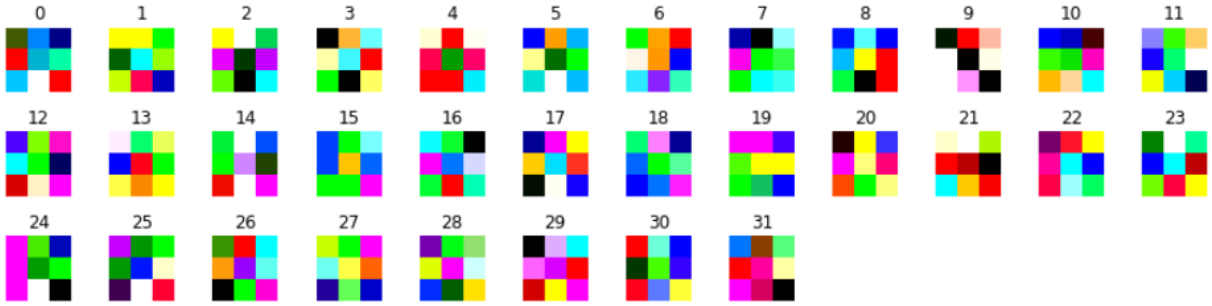

Рисунок 3.4 – Фільтри першого згорткового шару

Далі визначається функція активації за допомогою функції nn.ReLU. Останнім елементом блока є операція max pooling, яка у якості аргументів приймає розмір області об’єднання та *stride*. Область об’єднання дорівнює 2×2, отже, аргумент дорівнює 2 та *stride* дорівнює 2. У такий самий спосіб будуються наступні три блока. Четвертий блок є ланцюгом повнозв’язних шарів, що чередуються з функцією ReLU [29].

На рисунку 3.5 наведено інформацію про шари нейронної мережі, розмір вихідних даних та кількість параметрів для кожного шару. Усього мережа має понад 5,5 мільйонів параметрів, що навчаються, також видно, що 4 мільйони припадає на перший повнозв’язний шар.

Layer (type)	Output Shape	Param #	Tr. Param #
Conv2d-1	[128, 32, 32, 32]	896	896
ReLU-2	[128, 32, 32, 32]	0	0
Conv2d-3	[128, 64, 32, 32]	18,496	18,496
ReLU-4	[128, 64, 32, 32]	0	0
MaxPool2d-5	[128, 64, 16, 16]	0	0
Conv2d-6	[128, 128, 16, 16]	73,856	73,856
ReLU-7	[128, 128, 16, 16]	0	0
Conv2d-8	[128, 128, 16, 16]	147,584	147,584
ReLU-9	[128, 128, 16, 16]	0	0
MaxPool2d-10	[128, 128, 8, 8]	0	0
Conv2d-11	[128, 256, 8, 8]	295,168	295,168
ReLU-12	[128, 256, 8, 8]	0	0
Conv2d-13	[128, 256, 8, 8]	590,080	590,080
ReLU-14	[128, 256, 8, 8]	0	0
MaxPool2d-15	[128, 256, 4, 4]	0	0
Flatten-16	[128, 4096]	0	0
Linear-17	[128, 1024]	4,195,328	4,195,328
Dropout-18	[128, 1024]	0	0
ReLU-19	[128, 1024]	0	0
Linear-20	[128, 512]	524,800	524,800
Dropout-21	[128, 512]	0	0
ReLU-22	[128, 512]	0	0
Linear-23	[128, 10]	5,130	5,130

Total params: 5,851,338
Trainable params: 5,851,338
Non-trainable params: 0

Рисунок 3.5 – Опис моделі

На вхід першого шару надходить тензор 3×32×32, який у результаті роботи першого блока набуває вигляду 64×16×16, що означає 64 канали та зображення 16×16, у результаті застосування другого блока – 128×8×8, третього – 256×4×4. Далі тензор, отриманий у результаті роботи останнього згорткового шару, перетворюється на одномірний вектор за допомогою функції Flatten.

На наступному кроці створюється три повнозв’язних шари за допомогою функції nn.Linear. Перший аргумент методу – число вузлів в даному шарі, другий аргумент - число вузлів в наступному шарі. Таким чином, перший шар має розмір 256×4×4 вузла і з’єднується з другим шаром з 1024 вузлами, наступний з’єднує 1024 вузла з 512, а останній 512 з 10-тьма, де 10 – кількість класів, до яких потрібно віднести заданий об’єкт.

Після того, як мережа побудована, треба визначити як вхідні дані будуть проходити мережею під час прямого розповсюдження. Для цього існує стандартна функція forward, треба відзначити, що функція повинна мати саме таку назву для того, щоб використати базову функцію і дозволити всьому функціоналу nn.Module працювати коректно. В даному випадку, уся структура мережі міститься в одному обєкті nn.Sequential з назвою *network*, тому достатньо просто передати йому дані. Після визначення архітектури мережі необхідно її навчити.

3.3 Навчання моделі

Перш за все треба створити екземпляр моделі за допомогою інструкції model = CNN(), на рисунку 3.6 представлено код методу fit, який відповідає за навчання мережі. Метод має наступні аргументи: *epochs* – кількість епох навчання, *learning_rate* – визначає шидкість навчання, *model* – власне екземпляр класу нейронної мережі, *train_loader* – вибірка для тренування моделі, *val_loader* – вибірка для перевірки, *opt_func* – функція оптимізації. В даному випадку парметри мають наступні значення: *epochs* – 20, *learning_rate* – 0.001, *opt_func* – torch.optim.Adam.

Алгоритм навчання полягає в тому, що кожну епоху через мережу поступово проходить весь тренувальний набір *train_loader*, який розбивається на партії (batch). Для кожної епохи створюється оптимізатор Adam, який першим аргументом приймає параметри моделі, що потрібно навчити (*model.parameters*) та другим – швидкість навчання. Далі для кожної партії подається на вхід до мережі і обчислюється значення втрат за допомогою функції cross_entropy. Треба відзначити, що ця функція об’єднує функцію активації SoftMax, яка використовується для класифікації у останньому шарі, і крос-ентропійну функцію втрат в єдину функцію. Далі втрати додаються у список, який буде використаний пізніше для відстеження прогресу навчання.

```
def fit(epochs, learning_rate, model, train_loader, val_loader, opt_func):
    history = []
    optimizer = opt_func(model.parameters(), lr=learning_rate)
    for epoch in range(epochs):
        model.train()
        train_losses = []
        for batch in train_loader:
            images, labels = batch
            out = model(images)
            loss = F.cross_entropy(out, labels)
            train_losses.append(loss)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        model.eval()
        outputs = []
        for batch in val_loader
            images, labels = batch
            out = model(images)
            loss = F.cross_entropy(out, labels)
            acc = accuracy(out, labels)
            outputs.append({'val_loss': loss.detach(), 'val_acc': acc})
        result = model.validation_epoch_end(outputs)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

Рисунок 3.6 – Код методу навчання мережі

На наступному етапі виконується зворотнє поширення помилки і оптимізований крок тренування. Градієнти ініціалізуються нулями, використовуючи функцію оптимізатора zero_grad, потім викликається функція backward для змінної *loss* для виконання зворотного поширення. Завершуючим етапом навчання є виклик метода optimizer.step для виконання кроку навчання оптимізатора Adam [22]. Наостанок кожної епохи навчання відбувається оцінка точності моделі за допомогою методу evaluate і зберігаємо дані для їх подальшої візуалізації.

Під час навчання та подальшій перевірці на валідаційному наборі даних, мережі вдалося досягти 82% точності, графік втрат під час тренування представлено на рисунку 3.7.

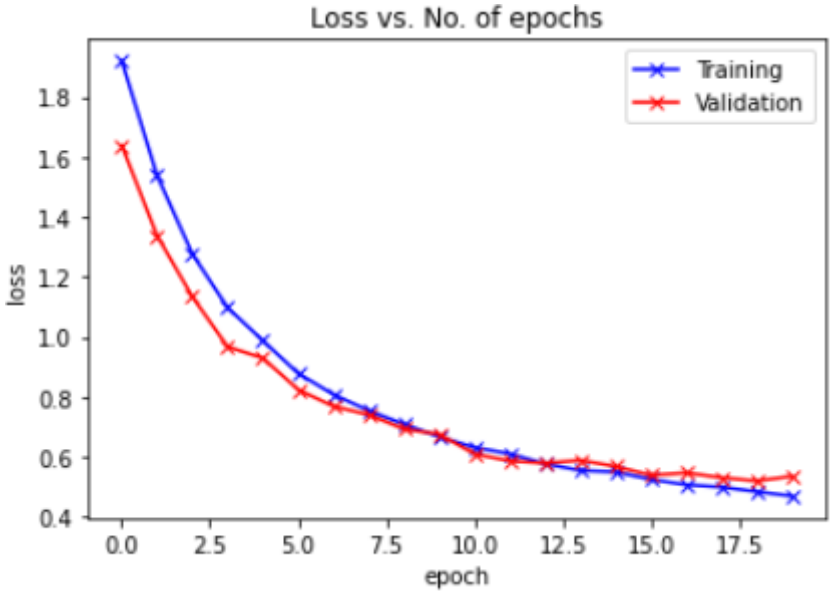

Рисунок 3.7 – Графік втрат на тренувальній та тестовій вибірках

Як видно, втрати на валідаційному наборі трохи вищі за тестові, що свідчить про перенавчання, але незначне. Оцінка моделі на тестовому наборі показала, що модель має 82% точності. Таким чином, можна зробити висновок, що модель має здатність до узагальнення та навчання відбулось вдало.

3.4 Використання алгоритмів виділення границь у нейронній мережі

В якості основи взято структуру моделі, яка була представлена у розділі 3.2, проте буть проводитися модифікації до першого згорткового шару, а саме: до згорткового шару, який має 32 канали на вході. Будуть додані додаткові згорткові шари, один з яких буде реалізацією оператора виділення границі.

На рисунку 3.8 наведено код реалізації операторів на прикладі оператора Собеля. За вийнятком різної кількості фільтрів та їхніх ваг, реалізація не буде відрізнятися від фільтра до фільтра.

```
grayscale = nn.Conv2d(3, 1, kernel_size=1, stride=1, padding=0)
grayscale.weight.data.fill_(1.0 / 3.0)
grayscale.bias.data.zero_()

sobel_filter = nn.Conv2d(1, 2, kernel_size=3, stride=1, padding=1)
sobel_filter.weight.data[0,0].copy_(
    torch.FloatTensor([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
)
sobel_filter.weight.data[1,0].copy_(
    torch.FloatTensor([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
)
sobel_filter.bias.data.zero_()

self.sobel = nn.Sequential(grayscale,
                           sobel_filter,
                           nn.ReLU(),
                           nn.Conv2d(2, 32, kernel_size=3, stride=1, padding=1),
                           nn.ReLU())
```

Рисунок 3.8 – Код реалізації операторів на прикладі оператора Собеля

Блок складається з трьох згорткових шарів, що чередуються із функціями активації ReLU. Перед тим як застосовувати оператори виділення кордонів, зображення потрібно перевести із кольорового у напівтонове. За це відповідає перший згортковий шар *grayscale*. Далі відбувається ініціалізація ваг початковими значеннями таким чином, щоб кожен Тобто перший шар має на вході три канали, а на виході – один.

Наступним згортковим шаром є, власне, реалізація оператора виділення границь, в даному випадку він має назву *sobel_filter*. Ваги шару ініціалізубться відповідно до ваг матриць фільтрів. При ініціалізації ваг згорткового шару, необхідно градієнти шару заповнити нулями задля їх коректного обчислення в подальшому при виклику методу *backward* на етапі навчання, якщо цього не зробити – градієнти будуть накопичуватися і матимуть некоректні значення. Шар приймає один і повертає N фільтрів, де N – кількість фільтрів конкретного оператора. Останній згортковий шар було введено задля переходу між N фільтами шару виділенння границь та наступного, який приймає на вході 32 фільтри. Для візуальної оцінки роботи операторів було вибрано випадкове зображення з набору даних CIFAR-10, яке зображено на рисунку 3.9

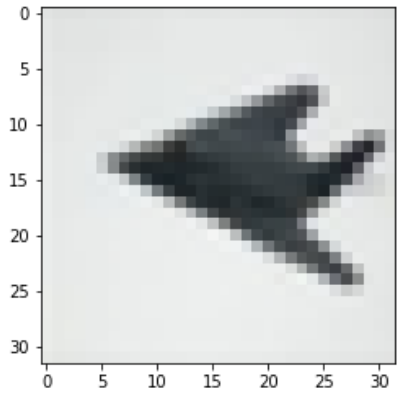

Рисунок 3.9 – Зображення для візуалізації роботи операторів виділення кордонів

Кожен оператор буде застосовано до зображення 3.9, та виконано порівняння результатів роботи оператора до та після навчання.

3.4.1 Використання оператору Собеля

На рисунку 3.10 показано графічне представлення фільтрів оператора Собеля.

Рисунок 3.10 – Візуалізація фільтрів оператора Собеля

Як видно на рисунку 3.11, до навчання оператор зумів виділити основні зовнішні границі літака, але бачимо, що також з’явилося багато ложних границь: виділенна частина неба та область посередині корпусу літака, де крила кріпляться до ф’юзеляжу – це ледь можливо розрізнити через невеликий перепад рівня яскравості між ф’юзеляжем та крилами. Також слід відзначити, що хвостова частина літака та кінцівки крил злилися в одну область. Неможливо розпізнати літак на рисунку.

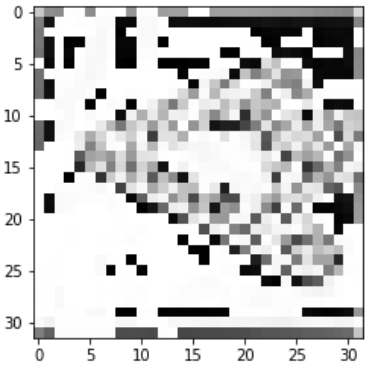

Рисунок 3.11 – Результат роботи оператора Собеля до навчання

Результат роботи оператора наведено на рисунку 3.12, як бачимо, пропали шуми на тлі та досить чітко розрізняються границі літака, зокрема крила та хвіст, хоча ліва кінцівка крила й розмита надто сильно. Пляма по середені все ще присутня, але не виділяється як окрема границя. В даний момент можна зробити висновок, що це літак.

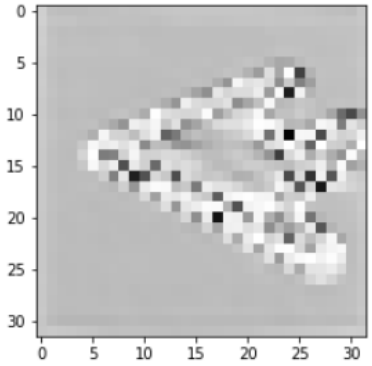

Рисунок 3.12 – Результат роботи оператора Собеля після навчання

Під час навчання ваги фільтрів зкорегувалися наступним чином (рис. 3.13).

```
tensor([
        [[[ 0.9553,  0.0420, -0.8239],
          [ 2.0624,  0.0032, -2.0887],
          [ 0.9344,  0.0304, -0.9168]]],


        [[[ 0.9618,  2.1256,  0.9436],
          [ 0.0454,  0.0670,  0.0459],
          [-0.8128, -1.9917, -0.8480]]]])
```

Рисунок 3.13 – Скореговані значення ваг після навчання

На рисунку 3.14 наведено графік втрат під час навчання. В результаті навчання точність моделі склала 80%, що на 2% нижче за точність базової моделі.

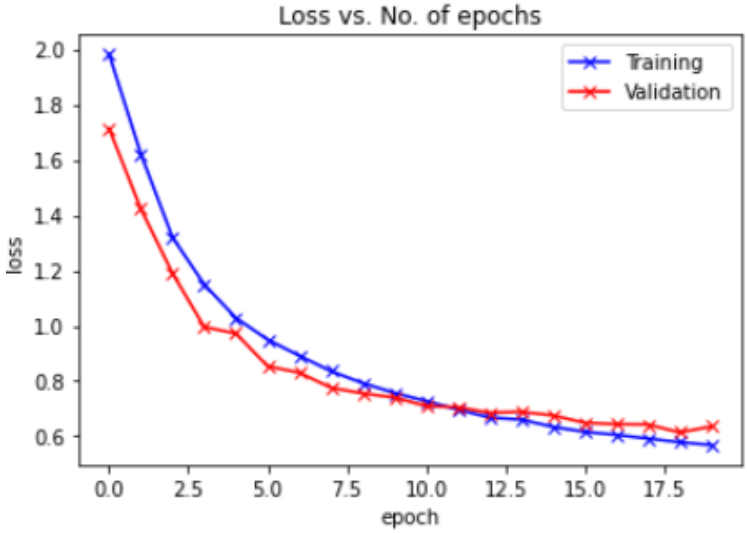

Рисунок 3.14 – Графік втрат на тренувальному та валідаційному наборах

3.4.2 Використання оператору Прюітта

На рисунку 3.15 показано графічне представлення фільтрів оператора Прюітта.

Рисунок 3.15 – Візуалізація фільтрів оператора Прюітта

Отримані результати до рис. 3.16 та після (рис. 3.17 навчання майже не відрізняються від попереднього оператора, через те, що єдиною відмінністю є різна вага центральних пікселів, але варто відзначити, що оператор Прюітта трохи чіткіше виділив зовнішній контур. Також на рисунку 3.18 наведено значення ваг фільтрів після тренування. На рисунку 3.19 наведено графік втрат під час навчання.

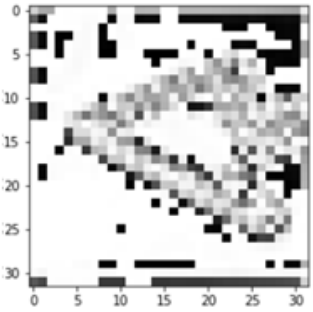

Рисунок 3.16 – Результат роботи оператора Прюітта до навчання

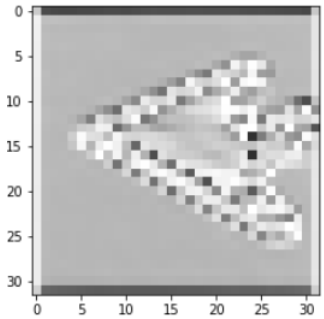

Рисунок 3.17 – Результат роботи оператора Прюітта після навчання

```
tensor([
    [[[ 0.9066,  0.0209, -0.8973],
    [ 1.0943,  0.0258, -1.1312],
    [ 0.9519,  0.0116, -0.9560]]],

    [[[ 1.0062,  1.1358,  0.9230],
      [ 0.0262,  0.0317,  0.0587],
      [-0.8498, -1.1012, -0.8584]]]])
```

Рисунок 3.18 – Скореговані значення ваг після навчання

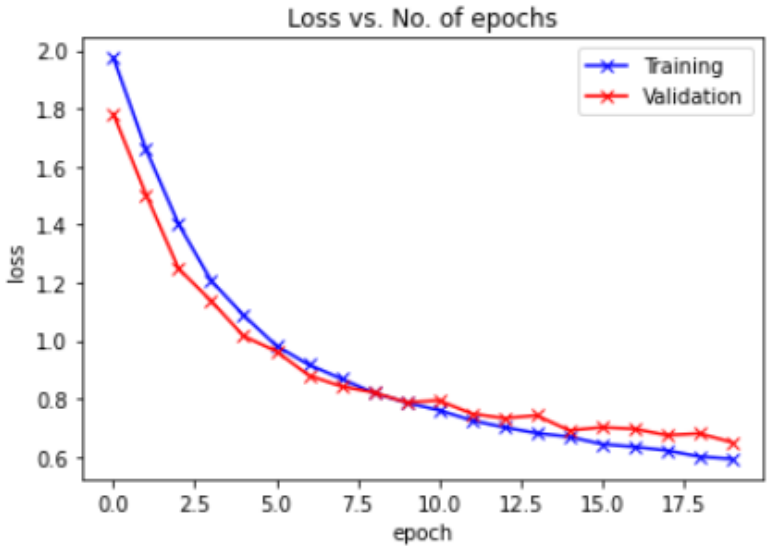

Рисунок 3.19 – Графік втрат на тренувальному та валідаційному наборах

Точність моделі складає 80%, що майже не відрізняється від результату робти оператора Собеля.

3.4.3 Використання оператору Лапласа

На рисунку 3.20 показано графічне представлення фільтрів оператора Лапласа.

Рисунок 3.20 – Візуалізація фільтрів оператора Лапласа

Як видно на рисунку 3.21, оператор Лапласа, на відміну від операторів Собеля та Прюітта виділив ще більше ложних границі за межами об’єкта, злив у єдину область кінцівки крил та хвістову частину та щось намагався та намагався щось виділити у області з низькою контрастностю посередині.

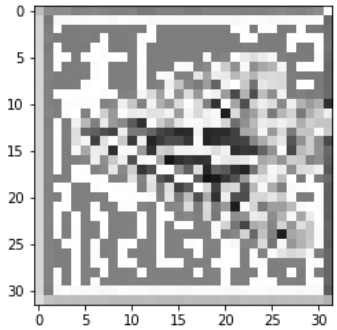

Рисунок 3.21 – Результат роботи оператора Лапласа до навчання

Після навчання (рис. 3.22) бачимо, що нема ложних виділень на тлі, зовнішні границі стали чіткими, також оператор виділив опорні точки, на відміну від Собеля та Прюітта, не виділяв внутрішні кордони як сильні.

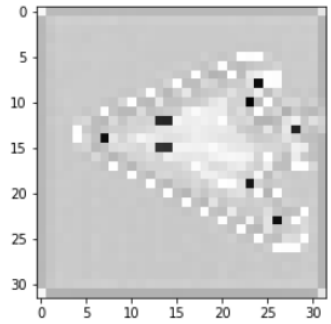

Рисунок 3.22 – Результат роботи оператора Лапласа після навчання

Скореговані під час навчання значення ваг фільтрів наведені на рисунку 3.23.

```
tensor([
    [[[ 0.8965,  0.9461,  0.8831],
    [ 0.9900, -8.0601,  1.0019],
    [ 0.8912,  0.8875,  0.8924]]],

    [[[ 1.0631,  1.1063,  1.0502],
    [ 1.1176, -7.9473,  1.0740],
    [ 0.9954,  0.9929,  0.9461]]]])
```

Рисунок 3.23 – Скореговані значення ваг після навчання

В результаті тренування модель досягла точності 81%, що лише на 1% меньше за базову модель. Графік втрат під час тренування наведено на рисунку 3.24.

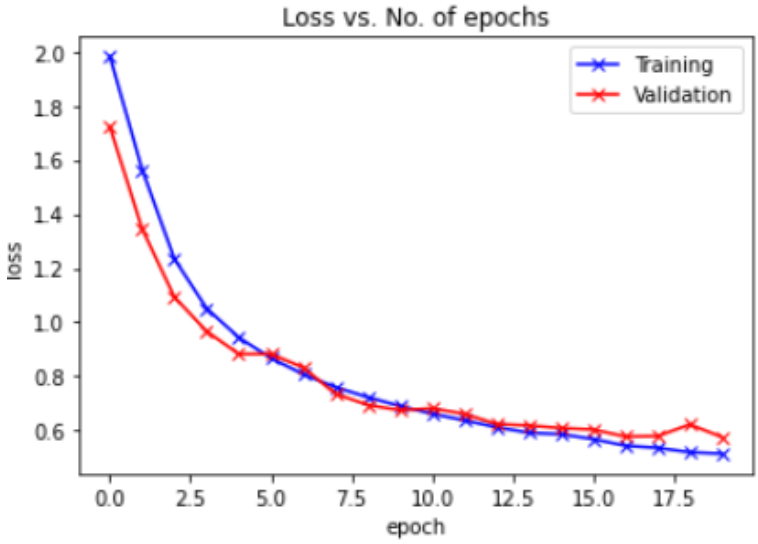

Рисунок 3.24 – Графік втрат на тренувальному та валідаційному наборах

3.4.4 Використання оператору Шарра

На рисунку 3.25 показано графічне представлення фільтрів оператора Шарра.

Рисунок 3.25 – Візуалізація фільтрів оператора Шарра

Як бачимо на рисунку 3.26, до навчання з результату роботи оператора в загалі неможливо зрбити хоч якісь висновки про об’єкт, багато ложних границь у небі та хаотично розкидані точки по всьому об’єкту. Після навчання ж (рис. 3.27) бачимо, що, як і у попередніх випадках, на тлі нема виділених ложних границь, контури об’єкта досить чіткі, проте оператор погано впорався з областіми з низьким рівнем контрастності.

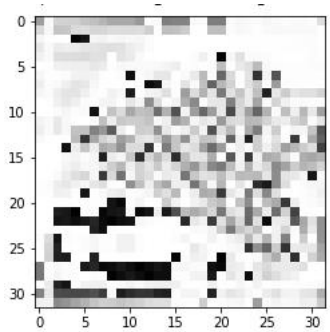

Рисунок 3.26 – Результат роботи оператора Шарра до навчання

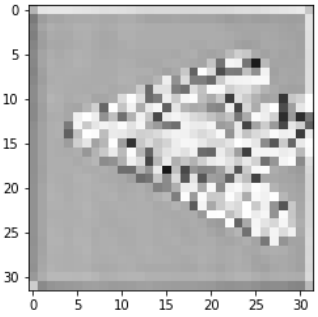

Рисунок 3.27 – Результат роботи оператора Шарра після навчання

Скореговані під час навчання значення ваг фільтрів наведені на рисунку 3.28.

```
tensor([
        [[[-2.8260e+00,  6.6881e-02,  2.9620e+00],
        [-9.9578e+00,  1.2402e-01,  1.0113e+01],
        [-2.9140e+00, -9.4309e-03,  2.8934e+00]]],


        [[[-2.5599e+00, -9.7488e+00, -2.6053e+00],
        [ 2.5659e-01,  2.7103e-01,  1.4994e-01],
        [ 3.0597e+00,  1.0357e+01,  3.0698e+00]]]])
```

Рисунок 3.28 – Скореговані значення ваг після навчання

У результаті навчання точність моделі склала трохи меньше за 81%. Графік втрат під час тренування наведено на рисунку 3.29.

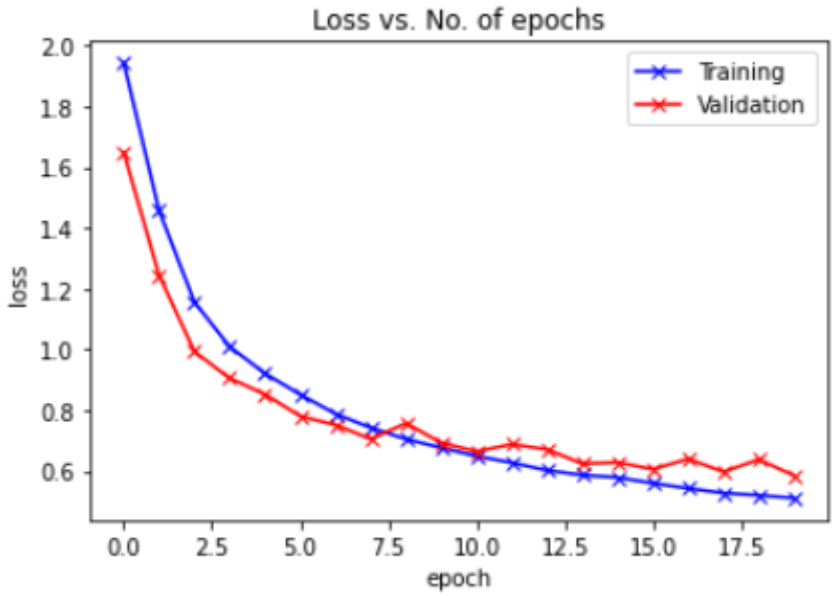

Рисунок 3.29 – Графік втрат на тренувальному та валідаційному наборах

У таблиці 3.1 наведено точність кожної кожної модифікації. Результати експериментів показали, що жодна мобифікація не показала кращої ефективності за базову модель, але при цьому різниця у точності була невелика. Серед модифікованих моделей модель із використанням оператора Лапласа показала найвищу точність. Це можна пояснити інваріантністю Лапласиану до повороту, на відміну від усіх інших розглянутих операторів, що

дозволяє виділяті границі незалежно від розташування у просторі. Також слід зазначити, що після навчання скореговані значення фільтрів не стали однаковими, лише трохи змінивши значення від початкових.

Таблиця 3.1 – Загальна середня оцінка роботи моделей

Назва моделі (модифікації)	Точність, %
Базова	0,824
Прюітт	0,795
Собель	0,803
Лаплас	0,812
Шарр	0,806

ВИСНОВКИ

У рамках атестаційної роботи було проведено дослідження впливу детекторів границь на точність розпізнавання об’єктів під час розв’язання задачі классифікації, а також реалізовано моделі з використанням запропонованих модифікацій. Були виконані наступні:

– проаналізовані різні архітектури згоркових нейронних мереж;

– проаналізовано різні оператори виділення границь;

– побудована згорткова нейронна мережа за допомогою фреймворка PyTorch та використанням мови програмування Python;

– побудовані модифіковані моделі з використанням операторів виділення границь;

– досліджено вплив операторів виділення границь на точність розпізнавання об’єктів.

В ході експериментів було встановлено, що точність розпізнавання знижується при використанні детекторів границь, у порівнянні з базовою моделлю. Було наведено результати роботи кожного оператора на випадково вибраному зображенні з набору даних CIFAR-10 до та після навчання модифікованої моделі. Було встановлено, що найкращій результат серед запропонованих модифікацій показав оператор Лапласа, лише на 1% поступившись точності базової моделі.

Дослідження показали, що модифікації поступаються базовій моделі на 1-2%, але варто зауважити, що усі моделі навчалися однакову кількість єпох, та мали однакову архітектуру – це було зроблено, щоб вони залишались порівнюваними. Однією з причин могла стати низька розподільна здатність зображень в цілому, тож у подальших достідженнях можна спробувати використати інший набір даних, але це у свою чергу потребує перебудови архітектури усієї мережі. Результати були апробовані на XXIV-ому Міжнародному молодіжному форумі [12].

Таким чином, ця ідея є перспективною для подальших досліджень і може бути втілена на іншому наборі даних, або ж на CIFAR-10, але зробивши корегування в архітектурі мережі та процесі навчання для кожного окремого випадку для максимізації точності. Ще один варінт для подальших досліджень у цьому напрямку – використання інших ізотропних фільтрів. Іншим варіантом є створення нейронної мережі, мета якої є тільки виділення границь, і використати її у якості першого шару.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сергеев, В. В. (1998). Применение методологии распознавания образов в задачах цифровой обработки изображений. *Автометрия*, (2), 63-76.

2. Верхаген, К., Дейн, Р., Грун, Ф., Йостен, Й., & Вербек, П. (1985). Распознавание образов: состояние и перспективы. *М.: Радио и связь*, 104.

3. Романов, С. А., Лепешкин, О. М., & Стоянов, Ю. П. (2010). Анализ методов сегментации изображений. *Молодой ученый*, (6), 26-28. [Електронний ресурс] – Режим доступу: https://moluch.ru/archive/17/1534/, (дата звернення: 28.10.2020), вільний. – Заголов.з екрану.

4. Gonzalez 3rd, R. (2008). Digital Image Processing 3rd edition/R. Gonzalez, R. Woods.

5. Форсайт, Д., & Понс, Ж. (2004). Компьютерное зрение. Современный подход. *М.: Вильямс*, *928*, 22.

6. Путятин, Е. П. (1990). Обработка изображений в робототехнике. Москва.

7. Семенов, С. И. (2002). Теория неадаптивных масок для обработки изображений. *Биомедицинские технологии и радиоэлектроника*, (12), 33-40.

8. Смеляков, К. С. (2005). Модели и методы сегментации границ изображений нерегулярного вида на основе адаптивных масок. *Харьков, 2005.–162 с*.

9. Sonka, M., Hlavac, V., & Boyle, R. (2014). Image processing, analysis, and machine vision. Cengage Learning.

10. Смеляков, К. С., Романенко, И. А., Рубан, И. В., Кириллова, Н. И., & Шитова, О. В. (2010). Методы сегментации изображений объектов нерегулярного вида, особенности их применения и перспективы развития.

11. Кудрявцев, Л. Д. (2010). Краткий курс математического анализа. Том 2. Дифференциальное и интегральное исчисление функций многих переменных. Гармонический анализ. Учебное пособие. Физматлит.

12. Сергеев Я.С., Машталір С.В. Методы классификации изображений. *Радіоелектроніка та молодь у XXI столітті*: матеріали 24 молодіжного міжнар. форуму (Харків, 7–9 квітня 2020 р.). Харків: ХНУРЕ, 2020. С. 50–51.

13. Хайкин, С. (2008). *Нейронные сети: полный курс, 2-е издание*. Издательский дом Вильямс.

14. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.

15. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

16. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

17. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

18. Как работает сверточная нейронная сеть: архитектура, примеры, особенности. URL: https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/, (дата звернення: 07.11.2020).

19. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, (дата звернення: 08.11.2020).

20. Deep Learning #3: More on CNNs & Handling Overfitting. URL: https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d, (дата звернення: 13.11.2020).

21. Preventing Deep Neural Network from Overfitting. URL: https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a, (дата звернення: 14.11.2020).

22. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, (дата звернення: 06.11.2020).

23. Алгоритмы выделения контуров изображений. URL: https://habr.com/ru/post/114452/, (дата звернення: 27.10.2020).

24. Гонсалес, Р., & Вудс, Р. (2012). Цифровая обработка изображений.

25. Хрящев, Д. А. (2010). Об одном методе выделения контуров на цифровых изображениях. Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика, (2), 181-187.

26. Линейные методы выделения границ областей. Лапласиан. URL: https://studfile.net/preview/1874349/page:5/, (дата звернення: 30.10.2020).

27. Краткий обзор PyTorch. URL: https://medium.com/@bigdataschool/краткий-обзор-pytorch-97400bc58ed8, (дата звернення: 13.11.2020).

28. The CIFAR-10 dataset. URL: https://www.cs.toronto.edu/~kriz/cifar.html, (дата звернення: 04.11.2020).

29. Batch Normalization and Dropout in Neural Networks with Pytorch. URL: https://towardsdatascience.com/batch-normalization-and-dropout-in-neural-networks-explained-with-pytorch-47d7a8459bcd, (дата звернення: 10.11.2020).