Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
Кафедра Комп’ютерних інтелектуальних технологій та систем

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Нейромережеве розпізнавання піксельних зображень

Виконав:

Студент 2 курсу, групи КІТм-21-1

Краснов І. О.
(прізвище, ініціали)

Спеціальність 123 Комп’ютерна інженерія

Тип програми освітньо-професійна

Освітня програма Комп’ютерні інтелектуальні технології

Керівник доц., к.т.н. Борисенко В.П.
(посада, прізвище, ініціали)

Допускається до захисту

(підпис)

Зав. кафедри О. Г. Руденко
(підпис)

2022 р.

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерної інженерії та управління
Кафедра	Комп'ютерних інтелектуальних технологій та систем
Рівень вищої освіти	другий (магістерський)
Спеціальність	123 Комп'ютерна інженерія
Тип програми	освітньо-професійна
Освітня програма	Комп'ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____2022р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Краснову Іллі Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Нейромережеве розпізнавання піксельних зображень
затверджена наказом по університету від « 7 » листопада 2022р. № 1455Ст
2. Термін подання студентом роботи до екзаменаційної комісії 12 грудня 2022р.
3. Вихідні дані до роботи (проекту)
   1) Мова програмування Python
   2) Бібліотеки OpenCV, ImageNet
   3) Середовище розробки – PyCharm
   4) Документація фреймворків Microsoft CNTK та Google TensorFlow
   5) Піксельні зображення
   6) Література

4. Перелік питань, що потрібно опрацювати в роботі
   1) Аналіз предметної області
   2) Комп'ютерний зір та фреймворки глибокого навчання
   3) Структура проекту
   4) Реалізація проекту
   5) Результати та оцінка

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням кафедри)

Слайд-презентація – 16 слайдів

6. Консультанти розділів роботи (п.6 включається до завдання за наявністю консультантів згідно до наказу, зазначеному у п.1)

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з літературою	07.11.22-08.11.22	
2	Аналіз публікацій за напрямком кваліфікаційної	09.11.22-10.11.22	
	роботи		
3	Огляд існуючих рішень	11.11.22-13.11.22	
4	Дослідження технологій глибинного навчання	14.11.22-18.11.22	
	та нейронних мереж		
5	Вибір технологій розробки та інструментальних	21.11.22-23.11.22	
	засобів	24.11.22-25.11.22	
6	Розробка програми		
7	Тестування та обробка отриманих результатів	28.11.22-30.11.22	
8	Розробка інструкції користувача	01.12.22-02.12.22	
9	Оформлення матеріалів кваліфікаційної роботи	05.12.22-09.12.22	

Дата видачі завдання 7 листопада 2022р.

Студент _____
(підпис)

Керівник роботи _____ доц, к.т.н. Борисенко В.П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка магістерської кваліфікаційної роботи: 81 сторінок, 5 розділів, 1 формула, 16 рисунків, 3 додатки, 31 джерело посилання,.

Метою магістерської роботи є дослідження нейромережевого розпізнавання піксельних зображень.

В ході виконання магістерської кваліфікаційної роботи було виконано наступне:

- обґрунтовано актуальність обраної теми, оглянуто наукові публікації, які стосуються теми кваліфікаційної роботи;
- розглянуто та проаналізовано технології нейронних мереж та глибинного навчання;
- розглянуто існуючі рішення з розпізнавання зображень, бібліотеки та фреймворки, що були використані під час дослідження;
- розроблено застосунок, що дозволяє розпінавати зображення за допомогою різних фреймворків та розроблено інструкцію користувача;
- проведено тестування та проаналізовано результати використання різних технологій та їх порівняння .

НЕЙРОННА МЕРЕЖА, CNN, ТЕСТУВАННЯ ФРЕЙМВОРКІВ, IMAGENET, RESNET, GOOGLE TENSORFLOW, MICROSOFT CNTK, CNTK, API.

ABSTRACT

Master’s thesis: 81 pages, 5 chapters, 1 formula, 16 figures, 3 appendices, 31 reference sources.

The purpose of the master's work is to study the neural network recognition of pixel images.

In the course of the master's qualification work, the following was performed:

- the relevance of the chosen topic is substantiated, scientific publications related to the topic of the qualification work are reviewed;
- considered and analyzed the technologies of neural networks and deep learning;
- existing solutions for image recognition, libraries and frameworks that were used during the research were considered;
- an application was developed that allows you to expand images with the help of various frameworks and a user manual was developed;
- testing was carried out and the results of the use of various technologies and their comparison were analyzed.

NEURAL NETWORK, CNN, FRAMEWORK TESTING, IMAGENET, RESNET, GOOGLE TENSORFLOW, MICROSOFT CNTK, CNTK, API.

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
Кафедра Комп’ютерних інтелектуальних технологій та систем

АНОТАЦІЯ

КВАЛІФІКАЦІЙНОЇ РОБОТИ

Рівень вищої освіти другий (магістерський)

Нейромережеве розпізнавання піксельних зображень

Виконав:

Студент 2 курсу, групи КІТм-21-1

Краснов І. О.
(прізвище, ініціали)

Спеціальність 123 Комп’ютерна інженерія

Тип програми освітньо-професійна

Освітня програма Комп’ютерні інтелектуальні технології

Керівник доц, к.т.н. Борисенко В.П.
(посада, прізвище, ініціали)

2022 р.

АНОТАЦІЯ

Штучний інтелект і комп’ютерний зір сьогодні повсюди навколо нас: просимо Siri запланувати наступну фінансову зустріч, використовуємо розпізнавання обличчя для розблокування телефону тощо. Штучний інтелект разом із комп’ютерним зором є визначними темами індустрії технологій. AI покладається на багато різних інструментів і методів, щоб імітувати людський інтелект і відтворити його за допомогою різних алгоритмів, застосованих на різних пристроях. Комп’ютерний зір – це галузь IT, яка зосереджується на здатності машин аналізувати та розуміти зображення та відео, а також виконує завдання розпізнавання зображень у машинному навчанні.

Розпізнавання зображень – це механізм, який використовується для ідентифікації об’єкта на зображенні та класифікації його в певній категорії на основі того, як люди розпізнають об’єкти в різних наборах зображень.

Коли ми бачимо об’єкт або зображення, ми, як люди, можемо негайно і точно впізнати, що це таке. Люди класифікують усе, що бачать, за різними категоріями на основі атрибутів, які ми визначаємо в наборі об’єктів. Таким чином, навіть якщо ми точно не знаємо, що таке об’єкт, ми зазвичай можемо порівняти його з різними категоріями об’єктів, які ми вже бачили в минулому, і класифікувати його на основі його атрибутів. Візьмемо для прикладу невідому нам тварину. Навіть якщо ми не можемо чітко визначити, що це за тварина, ми все одно можемо ідентифікувати її як тварину.

Люди рідко замислюються про те, що вони спостерігають і як вони можуть ідентифікувати об'єкти, це абсолютно відбувається підсвідомо. Люди не завжди зосереджені на всьому, що їх оточує. Наш мозок навчений легко ідентифікувати об’єкти на основі нашого попереднього досвіду, тобто об’єкти, з якими ми вже стикалися в минулому. У нас є надзвичайна здатність до дедукції: коли ми бачимо щось, що нагадує об’єкт, який ми вже бачили раніше, ми можемо зробити висновок, що це належить до певної категорії предметів. Нам не обов’язково дивитися на

кожну частину зображення, щоб ідентифікувати об’єкти на ньому. Як тільки ви побачите частину предмета, який ви впізнали, ви знаєте, що це таке. Зазвичай ми використовуємо кольори та контрасти для ідентифікації предметів.

Для людей, у більшості, розпізнавання зображень працює підсвідомо. Але все набагато складніше, коли мова йде про розпізнавання зображень за допомогою машин.Машини розпізнають лише ті категорії об’єктів, які ми в них запрограмували. Вони від природи не здатні знати та ідентифікувати все, що бачать. Якщо машину запрограмовано на розпізнавання однієї категорії зображень, вона не зможе розпізнавати нічого поза програмою. Машина зможе лише вказати, чи відповідають об’єкти в наборі зображень категорії чи ні. Чи намагатиметься машина вписати об’єкт у категорію, чи ігноруватиме його зовсім.

Для машини зображення складається лише з даних, масиву значень пікселів. Кожен піксель містить інформацію про значення червоного, зеленого та синього кольорів (від 0 до 255 для кожного з них). Для чорно-білих зображень піксель матиме інформацію про значення темності та білості (від 0 до 255 для обох).

Машини не бачать всього зображення; їх цікавлять лише значення пікселів і шаблони в цих значеннях. Вони просто беруть піксельні шаблони елемента та порівнюють їх з іншими шаблонами. Якщо два шаблони досить близькі, машина зв’яже їх і розпізнає другий шаблон як щось, з чим вона вже зустрічалася в минулому. У цьому сенсі відбувається те, що машина шукатиме групи подібних значень пікселів на зображеннях і намагатиметься помістити їх у певні категорії зображень. Дуже рідко програма розпізнає зображення на 100%. Патерни пікселів дуже рідко на 100% однакові при їх порівнянні. Вирішення цих проблем і пошук покращень є роботою IT-дослідників, мета яких полягає в тому, щоб запропонувати користувачам найкращий досвід.

Метою розпізнавання зображень є ідентифікація, позначення та класифікація виявлених об’єктів за різними категоріями. Розпізнавання об’єктів або зображень – це цілий процес, який включає різні традиційні завдання комп’ютерного зору:

- класифікація зображень – маркування зображення та створення категорій;
- локалізація об’єкта – визначення розташування об’єкта на зображенні за

допомогою його обмежувальної рамки;

- виявлення об’єктів – визначення наявності об’єктів за допомогою обмежувальних рамок і класифікація в межах класу, до якого вони належать;

- сегментація об’єктів – розрізнення елементів на зображенні, без використання обмежувальних рамок, але з виділенням контуру об’єкта на зображенні;

За останні кілька років це завдання комп’ютерного зору досягло великих успіхів, головним чином завдяки програмам машинного навчання.

Таким чином, **метою** кваліфікаційної роботи є дослідження нейромережевого розпізнавання піксельних зображень.

Об’єктом дослідження є аналіз та вивчення глибинного навчання, як теорії, так і практики, через реалізацію класифікатора піксельних зображень. Оцінка повинна складатися з якомога більшої кількості відповідних факторів, з найбільшим акцентом на продуктивність фреймворків з точки зору швидкості і використання апаратних засобів.

Предметом дослідження є порівнялння фреймворків глибинного навчання, розбите на частини, де різні частини оцінюються відповідно до набору критеріїв, в поєднанні з дослідницькою частиною, що складається з вивчення глибокого навчання і нейронних мереж.

Перш за все, машина повинна точно знати, що вона має шукати. Таким чином, необхідно надати їй ті параметри, над якими потрібно працювати. Визначення розмірів обмежувальних рамок і елементів, які знаходяться всередині, має вирішальне значення. Для цього машина має бути забезпечена деякими посиланнями, якими можуть бути зображення, відео чи фотографії. Ці елементи підвищать ефективність машини під час аналізу майбутніх даних. Це створить свого роду бібліотеку даних, яка потім використовуватиметься нейронною мережею для розрізнення об’єктів. Нейронна мережа складається з кількох штучних нейронів. Ці нейрони призначені для імітації людського мозку. Нейронна мережа працює з набором різноманітних алгоритмів, також за приклвдом того, як функціонує мозок. Якщо вимагається, щоб модель розпізнавання зображень аналізувала та

класифікувала різні раси собак, моделі потрібно буде мати базу даних різних рас, щоб розпізнавати їх. По-друге, модель має пройти етап навчання. Для належної роботи набір даних потрібно ввести в програму. І ця фаза призначена лише для навчання згорткової нейронної мережі (CNN) ідентифікувати конкретні об’єкти та точно організовувати їх у відповідних класах.

Перед використанням моделі штучного інтелекту її необхідно ретельно протестувати. Для цього необхідно запропонувати зображення, які не були частиною етапу навчання. Залежно від того, чи змогла програма ідентифікувати всі елементи, а також від точності класифікації модель буде затверджено чи ні. Нижче наведено деякі з найпопулярніших розпізнавання зображень за допомогою моделей машинного навчання та принципи їх роботи.

Моделі SVM(Support Vector Machines) використовують набір методів для створення алгоритму, який визначатиме, чи відповідає зображення цільовому об’єкту чи ні. З набору даних, з яким вона була встановлена, модель SVM навчена розділяти гіперплан на кілька категорій. Під час процесу, залежно від значень пікселів, об’єкти розміщуються на гіперплані, їхня позиція передбачає категорію на основі поділу категорій, отриманого на етапі навчання.

Моделі Bag of Features враховують зображення, яке потрібно проаналізувати, і еталонний зразок фотографії. Потім алгоритм у моделі намагається зіставити піксельні шаблони зразка фотографії з деякими частинами цільового зображення для аналізу.

Алгоритм Віоли-Джонса – один із найвідоміших, який використовується для розпізнавання обличчя. Він використовувався ще до використання CNN. Він сканує обличчя людей, виділяє деякі риси з облич і класифікує їх. Він також використовує алгоритм посилення, який покликаний допомогти мати набагато точнішу класифікацію. Машинне навчання спирається на те, що дав йому людський мозок. Це в основному під наглядом людей, спочатку, коли справа доходить до доставки набору еталонних зображень, щоб навчити машину розрізняти об’єкти та перевірити метод. CNN – це конкретна архітектура моделі з методів глибокого навчання. Алгоритм CNN дозволяє машинам виявляти та класифікувати з досить вражаючою

точністю всі об’єкти, які спостерігаються на зображенні. Цей тип алгоритму працює з різними рівнями сприйняття. Часто важко інтерпретувати роль конкретного шару в остаточному прогнозі, але дослідження досягли прогресу в цьому. Ми можемо, наприклад, інтерпретувати, що один шар аналізує кольори, інший – форми, наступний – текстури об’єктів. Наприкінці процесу саме суперпозиція всіх шарів робить прогноз можливим. Глибинне навчання виявилося надзвичайно ефективним для виявлення об’єктів і їх класифікації. Існують різні підходи, і кожен має свої особливості.

Таким чином, проект став порівняльним дослідженням фреймворків, дослідженням, розбитим начастини, де різні частини оцінювалися відповідно до набору критеріїв, у поєднанні з дослідницькою частиною, що складається з вивчення глибокого навчання та нейронних мереж через впровадження класифікатора зображень.

Основна мета цього проекту полягала в оцінці двох фреймворків глибокого навчання Google TensorFlow і Microsoft CNTK, насамперед на основі їх продуктивності під час навчання нейронних мереж. У вищезазначеному аспекті CNTK показав кращі результати, ніж TensorFlow із Keras як інтерфейсом для більш детального представлення результатів порівняльного аналізу. Було вирішено використовувати сторонній API Keras замість власного API TensorFlow під час роботи з TensorFlow, оскільки власний API TensorFlow надто громіздкий для роботи. Використовуючи Keras із TensorFlow як серверною частиною, було виявилено, що процес розробки в ньому простий та інтуїтивно зрозумілий, дивіться розділ 5.2, щоб дізнатися більше про фреймворки.

На завершення, незважаючи на те, що CNTK показав кращі результати в тестах порівняльного аналізу було виявлено, що Keras із TensorFlow як серверною частиною є набагато легшим і інтуїтивно зрозумілішим у роботі. Два аспекти, які вважаються важливішими при виборі фреймворку глибокого навчання для роботи.

В додаток, той факт, що CNTK, які лежать в основі реалізації алгоритмів і функцій машинного навчання, відрізняються від літератури та інших фреймворків, робить процес розробки виснажливим, якщо ви, як і ми, щойно вивчали літературу

та новачок у машинному навчанні. Тому виходячи з щойно згаданих причин, якби довелося вибрати фреймворк для продовження роботи, краще обирати Keras із TensorFlow як бекенд, навіть незважаючи на те, що продуктивність нижча порівняно з CNTK.

ШТУЧНИЙ ІНТЕЛЕКТ, CNN, ГЛИБИННЕ НАВЧАННЯ, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, CNTK, TESNORFLOW, AI, НЕЙРОННА МЕРЕЖА, КОМП'ЮТЕРНИЙ ЗІР, АЛГОРИТМ.

Використані в роботі публікації керівника та співробітників кафедри:

1. Руденко ОГ, Бодянський ЄВ. Штучні нейронні мережі. Харків: Компанія СМІТ. 2006.

2. Bodyanskiy YV, Rudenko OG. Artificial neural networks: architectures, learning, applications. TELETEX, Kharkov. 2004.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

- PC – personal computer
- ОІ – обчислювалбна система
- ШІ – штучний інтелект
- НМ – нейронна мережа
- CI – continuous Integration
- КІ – комп'ютера інженерія
- AI – artificial intelligence
- API – application programming interface
- ОС – операційна система
- CNN – convolutional neural network
- DNN – deep neural network
- ML – machine learning
- CV – computer vision
- ПЗ – програмне забезпечення

ВСТУП

Використання машинного навчання і, зокрема, нейронних мереж є зростаючою тенденцією в розробці програмного забезпечення, яка значно зросла за останні кілька років у світлі зростаючої потреби в обробці великих даних і великих інформаційних потоків. Машинне навчання має широку сферу застосування, таку як взаємодія людини і комп'ютера, прогнозування цін на акції, переклад в режимі реального часу і самокеровані транспортні засоби. Великі компанії, такі як Microsoft і Google, вже впровадили машинне навчання в деяких своїх комерційних продуктах, таких як пошукові системи та інтелектуальні персональні помічники Cortana і Google Assistant.

Машинне навчання – це величезна область, і для того, щоб це дослідження було завершено в заданий проміжок часу, сфера дослідження повинна бути обмежена деякою підмножиною машинного навчання. Тому теоретична сфера дослідження обмежується нейронними мережами, з акцентом на згорткові нейронні мережі. Практичний обсяг буде обмежений створенням та навчанням трьох еквівалентних нейронних мереж у TensorFlow та CNTK для трьох вже підготовлених та добре відомих наборів даних з метою оцінки продуктивності фреймворків на одній і тій самій нейронній мережі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Штучний інтелект (ШІ) – це широкий спектр програмної інженерії, що ототожнюється зі створенням машин, здатних виконувати завдання, які вимагають широкого людського розуміння [1]. ШІ ототожнюється з різними частинами науки з багатьма методологіями [2-5], однак сприяння ШІ та глибоке навчання змінюють модель практично в кожній сфері інноваційного бізнесу.

Симуляція інтелекту розширює ідею про те, що людський мозок може бути охарактеризований так, що машина, безсумнівно, може імітувати його та виконувати дії, від найпростіших до значно складніших [6].

Розпізнавання зображень, яке є одним із тих видів діяльності, в яких домінують глибокі нейронні мережі (DNN). Нейронні організації – це фігурні структури, призначені для сприйняття реляційних даних у зображеннях дизайнів. Основною технікою, що використовується для класифікації зображень, є згорточні нейронні мережі (CNN).

CNN складаються з кількох шарів із невеликими наборами нейронів, кожен з яких бачить невеликі фрагменти зображення. Результати кожного асортименту в шарі частково охоплюють підхід до повного аналізу зображення. Потім шар під ним повторює цей цикл на новому зображенні, дозволяючи структурі дізнатися про синтез зображення.

У магістерькій роботі представлено детальне обговорення застосування згорткових нейронних мереж на основі глибокого навчання в області класифікації зображень, а також описані різні програми класифікації зображень і сучасні CNN, які використовуються в цих програмах.

Також представлено детальне дослідження вступних термінів компонентів штучного інтелекту, тобто машинного та глибокого навчання разом із комп’ютерним зором, щоб зрозуміти основу CNN у класифікації зображень.

1.1 Комп’ютерний зір

Міждисциплінарна наукова сфера, яка визначає, яким чином комп’ютери можуть посилити розуміння високого рівня з цифрових зображень або стрічок, – це комп’ютерний зір. Він розуміє та автоматизує завдання, які може виконувати народжена зорова система. Призначення комп’ютерного зору містять методи обробки, аналізу та завершення цифрових зображень, а також накопичення первинних вимірних даних із неушкодженої земної кулі в лізі для формування числових або репрезентативних даних для зразка в розташуванні вердиктів.

Осягнення цього терену означає перетворення графічних картинок на описи світу, які привносять здоровий глузд у вивчення процесій і можуть викликати відповідні амбіції. Комп’ютерний зір дозволяє нам поступово перевіряти, чи кожна частина знаходиться на своєму місці, або, наприкінці взаємодії, чи остання зібрана правильно. Ця програма є цінною для об’єднання апаратури, апаратного забезпечення, електронних таблиць або попередніх конгрегацій із масою складності. Існує багато застосувань комп’ютерного зору, деякі з основних застосувань, які розглядаються, такі:

- виявлення об’єктів;
- розпізнавання жестів;
- класифікація зображень;

Розпізнавання предметів – це обчислювальна процедура машинного зору, яка дозволяє нам ідентифікувати та виявляти об’єкти на зображенні чи відео. За допомогою такого роду розпізнаваних доказів і обмежень розпізнавання об’єктів можна використовувати для включення об’єктів у сцену та визначення та відстеження їхніх точних зон, при цьому точно їх позначаючи.

Розпізнавання жестів є різновидом користувальницького інтерфейсу перцепційної обробки, який дозволяє комп’ютерним машинам знаходити та розшифровувати рухи людини як накази. Загальне значення підтвердження

руху – це здатність обчислювальної машини отримувати рухи та виконувати накази, залежні від цих рухів.

Класифікація зображень є одним із тих програм комп’ютерного зору, які мають дуже велике промислове значення. Класифікація зображень є основним простором, де глибокі нейронні організації беруть на себе основну роль у дослідженні зображення. Розташування зображень підтверджує подані інформаційні зображення та надає характеристику врожайності для визначення наявності захворювання. Цикл визначення характеристик планує класифікувати кожен піксель у вдосконаленому зображенні в один із кількох класів земельного покриву або «тем». Ці впорядковані дані можна застосувати для створення ефективних помічників земного покриву на зображенні. Послідовні мультиспектральні дані використовуються для відтворення запиту, і, без сумніву, модель видимості, присутня всередині даних для кожного пікселя, використовується як числове обґрунтування плану.

Явне використання штучного інтелекту включає звичайну мовну обробку, основні фреймворки, підтвердження дискурсу та комп’ютерне бачення. ML є однією з найпомітніших підмножин ШІ. ML натякає на структуру AI, яка може самонавчатися залежно від розрахунку. Явне використання штучного інтелекту включає звичайну мовну обробку, основні фреймворки, підтвердження дискурсу та комп’ютерне бачення. ML є однією з найпомітніших підмножин ШІ. ML натякає на структуру AI, яка може самонавчатися залежно від розрахунку. DL — це підмножина ML, яка зазвичай застосовується до масивних інформаційних колекцій.

1.2 Машинне навчання

Машинне навчання як сферу досліджень дещо важко описати стисло через широту цієї сфери, кількість поточних досліджень, міждисциплінарні аспекти цієї сфери та безліч інших факторів. З математичної точки зору

проблему можна сформулювати як застосування алгоритму машинного навчання для знаходження невідомої математичної функції з відомою областю значень на вході та відомою областю значень на виході. У більш неформальних термінах програма повинна знайти математичний зв'язок між даними та цільовим входом. Зв'язок залежить від поставленої задачі [1].

Відповідним прикладом вищезгаданих класів завдань є пошук зв'язку між векторизованим представленням входу та виходом, що складається з набору дискретних категорій або розподілу ймовірностей. Конкретними прикладами такої задачі може бути розробка програми для класифікації зображень та виявлення об'єктів, наприклад, розпізнавання облич на фотографіях та знаходження для них обмежувальних рамок. Іншими прикладами типів задач є виявлення аномалій, наприклад, виявлення кредитного шахрайства, та машинний переклад, наприклад, програмне забезпечення для перекладу [1].

Ефективність реалізації алгоритму машинного навчання зазвичай виражається або в точності реалізації, наприклад, який відсоток прикладів був класифікований правильно, або в коефіцієнті помилок. Складність тут полягає не стільки у виборі міри, скільки в тому, який аспект (аспекти) результату слід вимірювати в першу чергу. Чи слід, наприклад, вибирати відсоток правильно транскрибованих цілих послідовностей як єдину міру, чи враховувати правильно транскрибовані частини послідовностей при вимірюванні продуктивності [1].

Іншим ускладнюючим фактором є те, що може бути важко виміряти обрану кількість через практичні труднощі, і в таких випадках необхідно вибрати іншу міру. Існує в основному два види навчання, або досвіду, який алгоритм машинного навчання може отримати під час процесу навчання або тренування; навчання без нагляду та навчання під наглядом [1]. Процес навчання базується на наборі даних, що складається з певної кількості прикладів або точок даних.

Неконтрольоване навчання базується на наборах даних, що містять

ознаки, де метою є вивчення корисних властивостей структури набору даних або, точніше, знаходження основного розподілу ймовірностей набору даних. Навчання під контролем, навпаки, базується на наборах даних, що складаються з багатьох ознак, але з мітками, прикріпленими до кожного прикладу [1]. Метою керованого навчання є навчитися передбачати мітку, пов'язане з нею значення, на основі довільного прикладу в наборі даних після завершення тренувального сеансу.

Терміни «контрольоване» і «неконтрольоване» походять від того, що алгоритм навчання має вчителя, який показує алгоритму, що робити в першому випадку, але передбачається, що він навчається на основі даних без керівництва в другому випадку. Найскладнішою частиною машинного навчання, незалежно від алгоритму (алгоритмів), що застосовуються в процесі навчання, є створення програм, які добре узагальнюють; тобто програм, які працюють так само добре або майже так само добре на неспостережуваних вхідних даних, як і на тих, що спостерігаються під час навчання.

Головне, що відрізняє машинне навчання від оптимізації, полягає в тому, що мета полягає в мінімізації як помилки навчання, так і помилки тестування [1]. Вищезгадані набори даних розбиваються на два заздалегідь визначені набори, які будуть використовуватися в процесі навчання, один для навчання і один для тестування. Мета машинного навчання тепер може бути визначена більш точно: мінімізувати помилку навчання та мінімізувати розрив між помилкою навчання та помилкою тестування. При спробі досягти вищезгаданої мети з'являються ще дві проблеми: недостатня пристосованість та надмірна пристосованість на навчальній вибірці.

Недостатня пристосованість виникає тоді, коли модель не може досягти достатньо низької помилки навчання, а надмірна пристосованість виникає тоді, коли розрив між помилкою навчання та помилкою тесту є занадто великим (рисунок. 1.1).

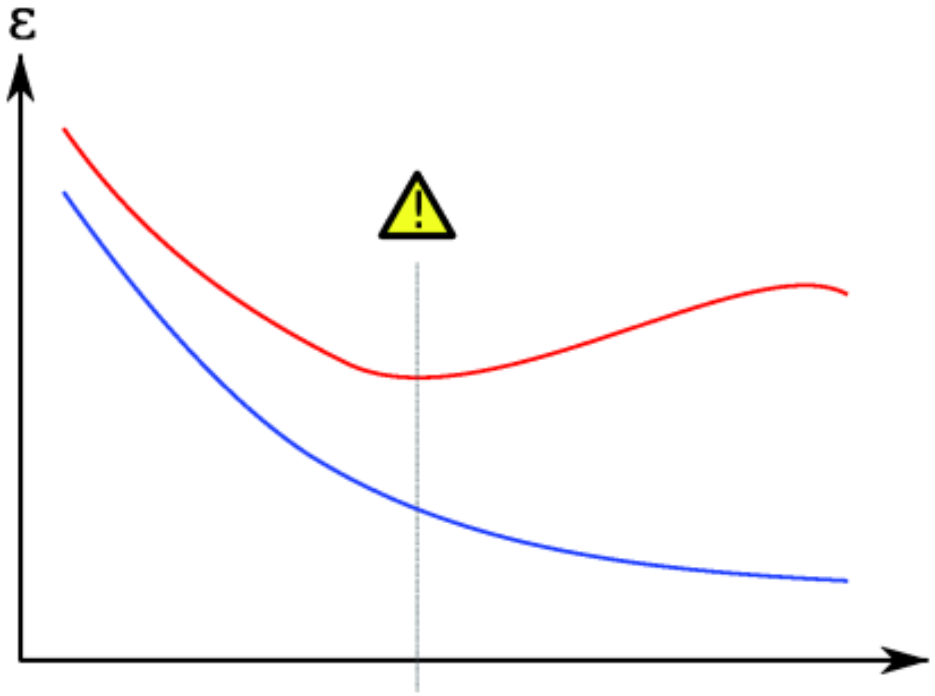

Рисунок 1.1 – Перенавчання під час навчання.

Перша проблема викликана моделлю з низькою репрезентативністю, яка не може належним чином відповідати навчальній множині, друга може бути викликана моделлю з високою репрезентативністю, яка запам'ятовує властивості з навчальної множини занадто добре, щоб працювати на тестовій множині в достатній мірі.

Надмірна пристосованість також може бути спричинена навчальною вибіркою, яка є занадто малою для належного узагальнення. Недостатню пристосованість можна досить легко виправити за допомогою моделі з достатньою пропускною здатністю, але надмірну пристосованість може бути важче виправити через те, що отримання більших обсягів даних не завжди є можливим. Яскравим прикладом техніки, яка може бути введена під час навчання, є регуляризація, яка використовується для обмеження простору потенційних функцій (оскільки мета полягає в тому, щоб знайти найкращу).

Параметри алгоритму машинного навчання, які не адаптуються і не змінюються самим алгоритмом під час навчання, називаються

гіперпараметрами, параметрами, які можуть бути налаштовані для зміни поведінки алгоритму навчання.

Тут доречно ввести ще один поділ набору даних, а саме: вихідні навчальні дані на навчальну та перевірочну множину. Навчальний набір використовується виключно для налаштування внутрішніх параметрів, наприклад, ваг та зміщень, під час навчання, а валідаційний набір використовується для вимірювання поточної помилки узагальнення та відповідного налаштування гіперпараметрів. Валідаційний набір не використовується для налаштування внутрішніх параметрів [1].

Важливою відмінністю між валідаційним набором і тестовим набором є те, що останній взагалі не використовується в процесі навчання і не дає ніяких даних для процесу навчання - він просто використовується для вимірювання продуктивності. Як останнє зауваження в цій главі: існує багато алгоритмів машинного навчання на вибір, занадто багато, щоб детально описати їх тут або навіть зробити побіжний огляд. Оскільки фреймворки в дослідженні є реалізаціями компонентів, функціональності та алгоритмів, необхідних для навчання нейронних мереж та реалізації глибокого навчання, немає необхідності.

Як видно на рисунку 1.1, втрати при навчанні (синій колір) постійно зменшуються, тоді як втрати при тестуванні (червоний колір) через деякий час починають збільшуватися.

1.2.1 Глибоке навчання та нейронні мережі

Хоча найперший крок до нейронних мереж був зроблений у 1943 році в роботі Уоррена Маккалоха та Уолтера Піттса, перше практичне застосування з використанням штучних нейронів з'явилося з винаходом Френка Розенблата, персептрона.

Перцептрон є найпростішою можливою версією штучного нейрона (рисунок. 1.2) і має такі основні суттєві атрибути [5]:

- один або більше числових входів з відповідними вагами, позитивними або негативними, для кожного входу;

- зсув, який може бути як позитивним, так і негативним. Може бути неформально описано як стійкість нейронів до «вистрілювання»;

- функція активації (у випадку персептрона - функція одиничного кроку);

- єдине вихідне значення, функція активації, застосована до суми зважених входів і зсуву.

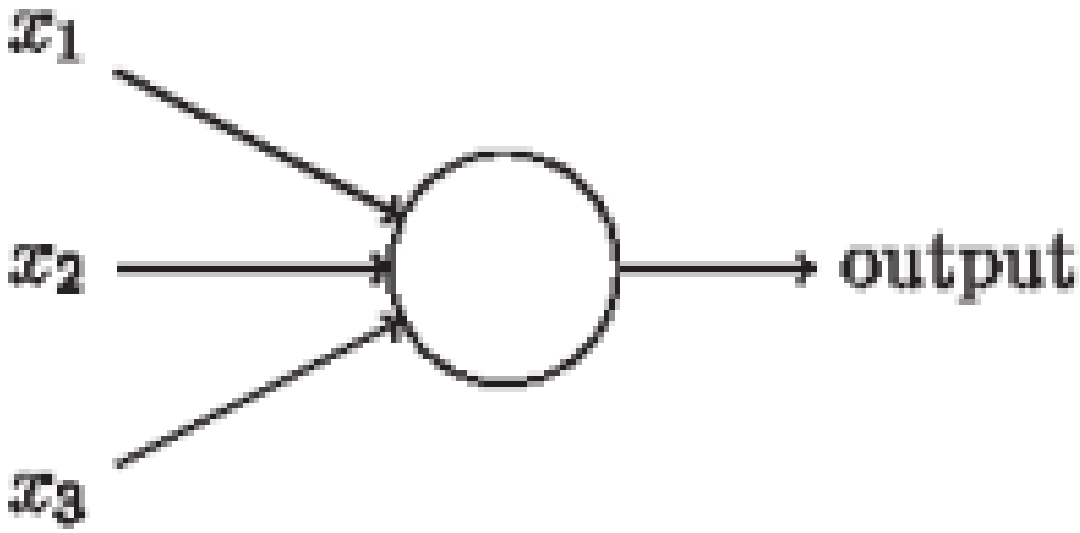

Рисунок 1.2: Штучний нейрон

Більш неформально персептрон видає 1, якщо сума зважених входів і зсуву більша за 0, і 0, якщо ні. Незважаючи на те, що персептрони не використовуються на практиці, вони призвели до наступного логічного кроку – багатошарового персептрону (MLP) або нейронної мережі прямого поширення (рисунок. 1.3).

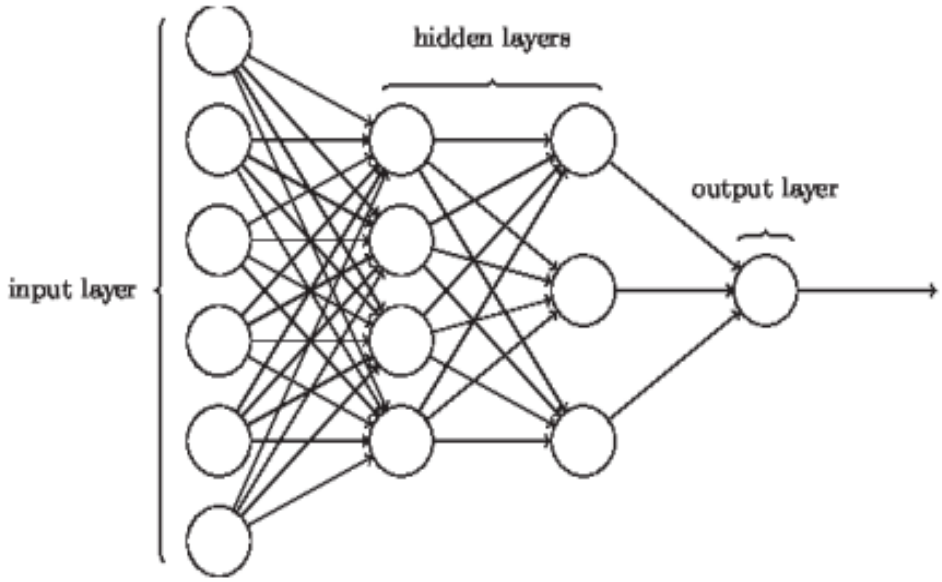

Рисунок 1.3 – Штучна (проста) нейронна мережа.

Нейронна мережа прямого поширення – це просто штучні нейрони, розташовані шарами, причому всі виходи з кожного нейрона попереднього шару подаються вперед, а не назад, на кожен нейрон наступного шару, за винятком вхідного шару (що складається з пасивних нейронів, які не перетворюють вхідні дані) і вихідного шару.

Шари між першим і останнім називаються прихованими шарами, що надає мережі глибину і, отже, призводить до першої частини назви цього розділу – глибоке навчання, що також є загальною назвою для використання глибоких нейронних мереж в цілому і пов'язаних з ними методів.

Оскільки кожен прихований шар і вихідний шар складаються з нейронів, які окремо пов'язані з виходом кожного нейрона попереднього шару, то ці шари в мережі прямого поширення називаються повністю пов'язаними шарами [2]. Всі нейрони в мережі мають унікальний набір ваг, а функції активації є нелінійними функціями.

Перш ніж розглядати більш технічні деталі прямих нейронних мереж, необхідно спочатку представити більш фундаментальну властивість прямих нейронних мереж, яка полягає в тому, що прямі нейронні мережі працюють як універсальні апроксиматори функцій [1] [2].

Окремий персептрон або будь-який інший штучний нейрон не приносить великої користі, але мережа з принаймні одним прихованим шаром може апроксимувати будь-яку неперервну функцію, що на практиці означає, що будь-яка розривна функція також може бути апроксимована [2].

Концептуально штучний нейрон можна порівняти з логічним вентилем NAND або NOR в тому сенсі, що він працює як універсальний будівельний блок. Основна відмінність полягає в тому, що штучний нейрон має параметри, які можуть бути налаштовані і, отже, можуть бути навчені. Говорячи про параметри, доречно ввести функції активації, які використовуються на практиці. Функція одиничного кроку, згадана вище, не використовується на практиці через те, що невелика зміна входу може призвести до великої зміни виходу (від 0 до 1), що є небажаною властивістю, оскільки неперервна зміна виходу є кращою [2].

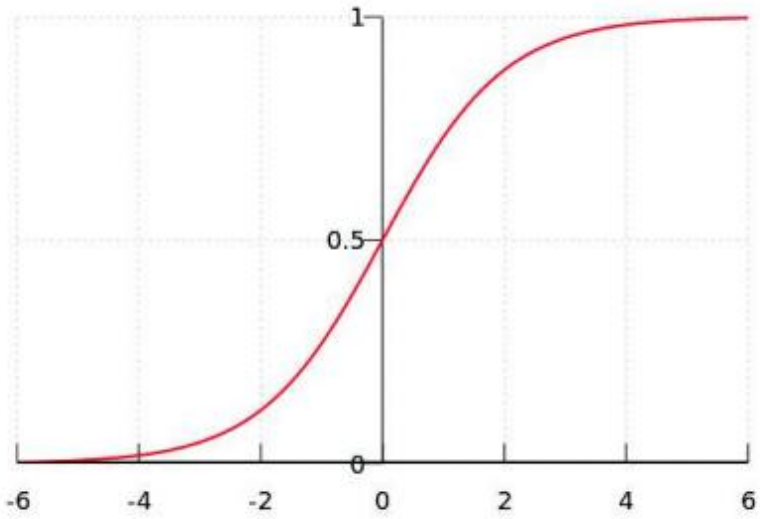

Рисунок 1.4 – Графік сигмоїдної функції.

Сигмоїдна функція (рисунок. 1.4) та гіперболічна тангенціальна функція, більш гладкі версії одиничної ступінчастої функції, використовувалися на практиці, але функцією активації вибору сьогодні є випрямлена лінійна одинична функція (ReLU), яка визначається поверненням

тільки позитивних значень, та варіанти цієї функції (рисунок 1.5) [1]. У вихідному шарі, або шарі класифікації, використовується функція softmax 10.

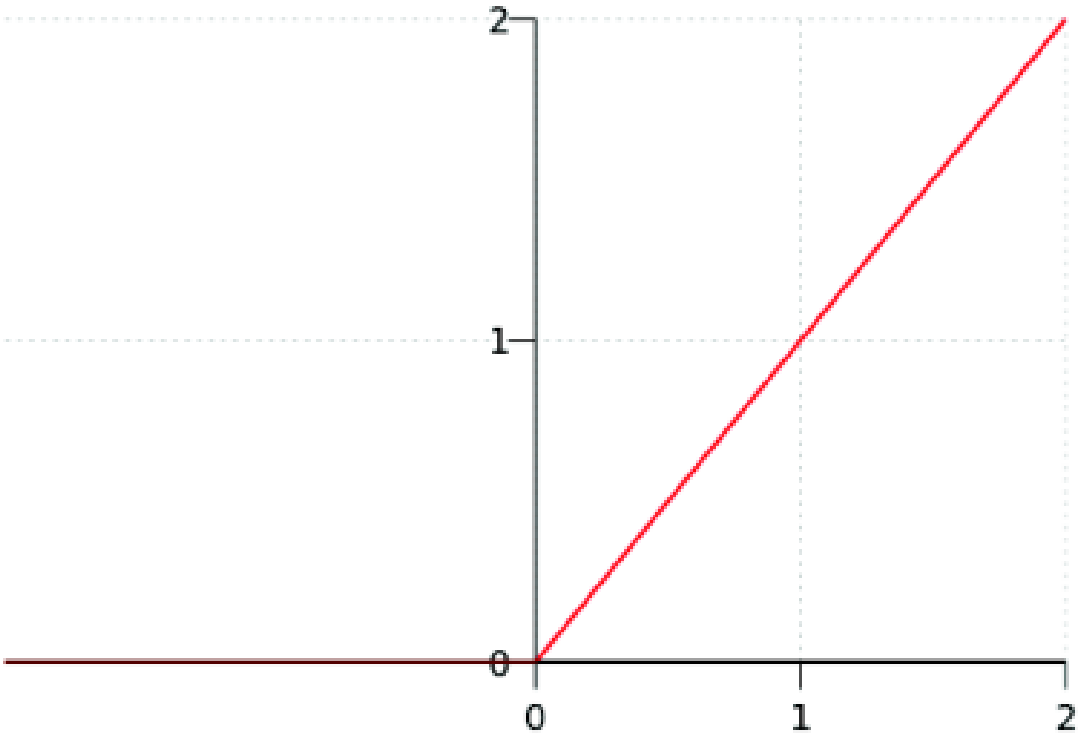

Рисунок 1.5 – Графік функції випрямленої лінійної одиниці (ReLU).

Більш неформально функція softmax виводить найбільш ймовірну категорію, класифікацію, яку мережа визнала найбільш вірогідною. Для навчання нейронної мережі потрібна якась інша міра того, наскільки великою є поточна помилка за межами помилки навчання, якась міра того, наскільки добре підібрані ваги та упередження в мережі в цілому. Для вирішення цього завдання вводиться функція вартості або цільова функція, яка вимірює загальну поточну помилку.

Дві важливі властивості, якими повинна володіти така цільова функція, полягають у тому, що вона повинна бути невід'ємною для всіх вхідних даних, і що вона дорівнює нулю або близька до нуля, якщо помилка невелика [2]. Таким чином, прямою метою навчання є мінімізація цільової функції. Простим підходом тут був би вибір середньоквадратичної помилки як

вартості, яку потрібно мінімізувати, але на практиці замість неї використовується функція перехресної ентропії, оскільки вона за своєю суттю має кращі показники [2].

Нейронні мережі можуть містити мільйони, десятки мільйонів і навіть мільярди параметрів, і не існує реального способу знайти мінімум за допомогою методів звичайного обчислення.

Для мінімізації цільової функції використовується алгоритм, який називається градієнтним спуском, точніше стохастичним градієнтним спуском [2]. Метод використовує градієнт або похідну у випадковій початковій точці для знаходження поточного «нахилу» цільової функції, і зменшує значення цільової функції з фіксованою кратністю абсолютного значення градієнта, що в більш неформальних термінах означає, що алгоритм перемістився вниз по «нахилу» на фіксовану величину в напрямку мінімуму.

Застосування градієнтного спуску до нейронної мережі означає, що кожна вага і кожне зміщення в мережі коригується з фіксованим кратним значенням часткової похідної цільової функції по відношенню до цієї конкретної ваги або зміщення під час кожного проходу алгоритму. Фіксовані множники, згадані вище, множаться на швидкість навчання, гіперпараметр мережі, який може бути налаштований для підвищення продуктивності.

Обчислення градієнта для всієї навчальної вибірки зайняло б занадто багато часу і на практиці не використовується. Замість цього градієнт обчислюється для випадково обраної підмножини навчальної вибірки, так званої міні-партії, і використовується для оновлення мережі. Цей процес повторюється для кожної міні-партії в навчальній вибірці, поки не будуть оброблені всі міні-партії. Час, необхідний для обробки всієї навчальної вибірки, називається епохою, а кількість тренувань зазвичай визначається кількістю епох. Той факт, що навчальна множина перемішується, також пояснює повну назву алгоритму – стохастичний градієнтний спуск.

Незважаючи на те, що стохастичний градієнтний спуск, як описано вище, є алгоритмічним рішенням математично нерозв'язної проблеми, він все

ще потребує подальшої роботи, щоб бути корисним на практиці. Знову ж таки, нейронні мережі мають величезну кількість параметрів і обчислення кожної часткової похідної цільової функції, як у наївній реалізації вище, є чисельно нездійсненним. Рішенням проблеми, яке робить стохастичний градієнтний спуск корисним на практиці, є зворотне поширення – алгоритм, який, простіше кажучи, обчислює похибку одного шару на основі похибки попереднього шару і відповідно оновлює свої параметри [2].

Назва алгоритму походить від тієї властивості, що помилка, і виправлення помилки, поширюється в зворотному напрямку через всю мережу від вихідного шару і назад. Елегантність алгоритму полягає в тому, що він відображає шлях, який пройшли активації в мережі, і має приблизно однакові обчислювальні витрати. Як останнє зауваження щодо градієнтного спуску, сьогодні існують більш просунуті варіанти, які базуються на стандартній версії зі зворотним розповсюдженням, варіанти, які додають додаткові елементи, такі як динамічні швидкості навчання та інші налаштування [1].

Існують вбудовані проблеми, пов'язані з процесом машинного навчання, і нейронні мережі не є винятком. Дві проблеми, які будуть розглянуті тут, – це перенавчання у випадку нейронних мереж та проблема нестабільного градієнта, проблема, характерна для алгоритму стохастичного градієнтного спуску, що застосовується для навчання нейронних мереж [2].

Надмірне пристосування в цьому контексті виправляється такими методами, як збір більшої кількості та кращої якості даних та регуляризація. Регуляризацію можна описати як обмеження кількості функцій, які може генерувати алгоритм машинного навчання. У випадку з нейронними мережами існує багато видів регуляризації, але тут ми обмежимося наступними методами: L1 регуляризація, L2 регуляризація, відсів та доповнення даних [2]. L1 та L2 регуляризація є двома варіантами однієї і тієї ж теми і побудовані на додаванні додаткового члена до функції вартості, члена, що складається із середньозваженого значення суми всіх ваг в мережі.

Різниця між ними полягає в тому, що в першому випадку сума складається з абсолютних значень ваг, а в другому – з квадратів ваг. Причиною додавання цього терміну є покарання мереж з надто великими вагами, покарання, ефект якого, за неофіційними даними, полягає в тому, що мережа краще узагальнює, змушуючи її вибирати менш складні функції, що пов'язують вхід з виходом.

Відсів – це метод, який базується на відсіві випадкової кількості нейронів у прихованих шарах під час кожного раунду міні-партій з остаточним коригуванням ваг в кінці навчального циклу. Очікуваний ефект, як і у випадку з L1 і L2, полягає в тому, щоб забезпечити краще узагальнення. Останній метод, доповнення даних, побудований на штучному розширенні наявних даних шляхом внесення випадкових змін у приклади, наприклад, перевертання зображення по горизонталі, невеликого зсуву в напрямку або невеликого обертання.

Оскільки найкращі ліки від надмірної підгонки – це більше даних, ця техніка працює в цьому напрямку, щоб зробити мережу краще узагальнюючою. Проблема нестабільного градієнта виникає через те, що градієнт, або швидкість зміни параметрів, в шарі є добутком швидкості зміни всіх шарів перед ним. Це може призвести до того, що швидкість зміни повністю зникне (проблема зникаючого градієнта), або різко зросте (проблема вибухового градієнта). Небажані ефекти накопичуються тим швидше, чим раніше в мережі знаходиться шар, про який йде мова. Ця проблема була, принаймні історично, основною перешкодою для навчання мереж понад певну глибину. На щастя, проблема, здається, частково вирішена, частково завдяки тому, що вищезгадана випрямлена лінійна функція та її різновиди стали стандартними функціями активації. Основною причиною проблеми було використання насичуючих функцій активації, таких як сигмоїдні та гіперболічні тангенціальні функції, насичення в цьому контексті означає, що швидкість зміни або похідна функції прямує до нуля, коли вхідний сигнал стає занадто великим додатним або від'ємним числом.

На відміну від цього, функція ReLU має похідну, яка дорівнює або 0, або 1, що означає, що вона не насичується в позитивній області і завжди поширює градієнт у зворотному напрямку. Терміни «нейронні мережі» і «нейронні мережі прямого поширення» використовувалися як синоніми. Однак ці терміни не є повністю синонімічними, оскільки існують нейронні мережі з петлями та зворотними зв'язками. Цей підклас нейронних мереж називається рекурентними нейронними мережами і використовує більш складні шари, ніж ті, що були згадані вище, такі як довгі одиниці короткочасної пам'яті (LSTM) [1].

1.2.2 Згорткові нейронні мережі

Одним із недоліків звичайної нейронної мережі прямого поширення з повністю з'єднаними шарами є те, що вона не має попереднього вбудованого припущення про дані, на яких вона повинна навчатися. Вона є агностичною щодо структури даних і трактує всі дані однаково [2].

Це може легко стати проблемою, частково через надлишкові параметри, частково через розмір надлишкових параметрів, оскільки нейронна мережа може, як вже неодноразово згадувалося раніше, дуже сильно розростатися. Також контрінтуїтивно використовувати цей підхід у випадку, наприклад, даних, що складаються з зображень, оскільки це означало б, що просторова структура зображення буде ігноруватися і навчання буде починатися з припущення, що всі пікселі однаково пов'язані між собою. Очевидно, що потрібен інший підхід, тому в цьому підрозділі розглядається особливий вид нейронних мереж прямого поширення – згорткові нейронні мережі.

Згорткові нейронні мережі розробляються з певними припущеннями про дані, припущеннями, які підходять не тільки для даних зображень, але й для інших даних з подібною внутрішньою структурою. Найбільш фундаментальною операцією згорткових нейронних мереж є, як це, мабуть,

не дивно, згортки.

Концепція згорток в контексті нейронних мереж починається з ідеї шарів, що складаються з нейронів з локальним рецептивним полем, тобто нейронів, які пов'язані з обмеженою областю вхідних даних, а не з усією. У випадку з даними зображення це означає, що кожен нейрон пов'язаний лише з обмеженою областю пікселів, наприклад, з квадратом 3 x 3 пікселі у верхньому лівому куті зображення (рисунок 1.6).

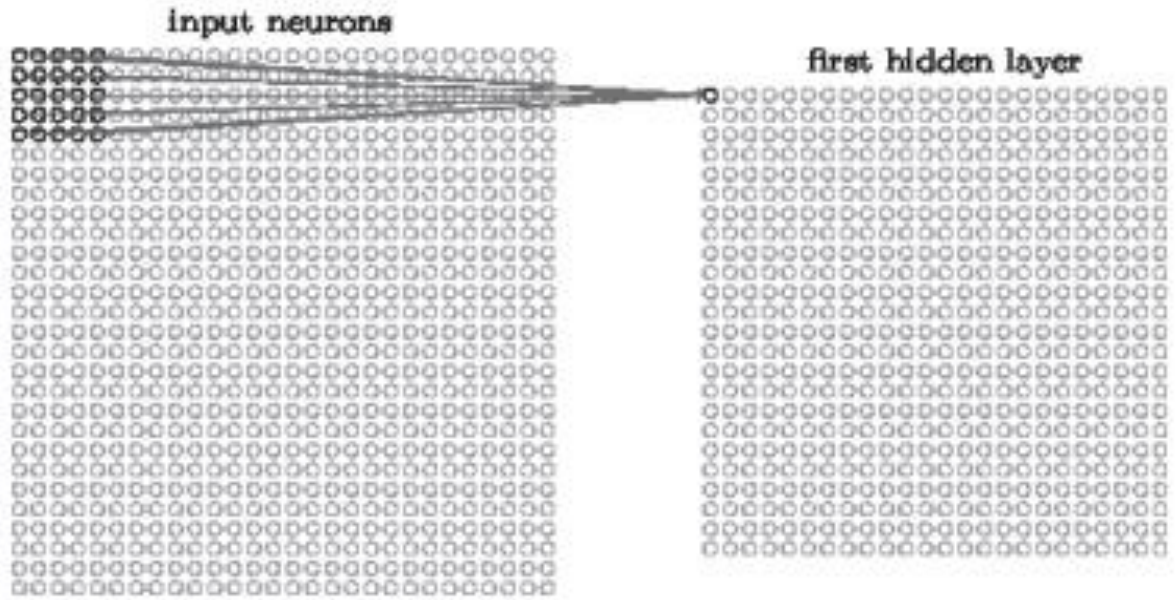

Рисунок 1.6 – Прихований шар нейронів з локальними рецептивними полями.

Рецептивні поля нейронів також певною мірою перекриваються, в тому сенсі, що (продовжуючи приклад з попереднього речення) два сусідні нейрони мають рецептивні поля, які зсунуті на один або кілька пікселів по горизонталі або вертикалі відносно один одного

Можна уявити собі розсувне вікно заданого розміру, що ковзає зліва направо і зверху вниз (відтепер можна вважати, що дані мають двовимірну структуру), з'єднуючи вихід вікна в кожному проході з нейроном. В результаті вийде власна двовимірна структура, карта або зображення активацій від кожного нейрона. «Ковзаюче вікно» в даному контексті

правильніше називати ядром, набором ваг і зсувом. Це означає, що кожен нейрон в такому шарі або карті має однакові параметри і, таким чином, можна сказати, вивчає одну і ту ж ознаку, що обумовлює правильну назву для таких структур – карти ознак.

Описана вище операція застосування ядра до вхідної карти та генерування карти ознак є згорткою. Інтуїтивно зрозуміло, що використання згорток дуже добре підходить для просторово корельованих даних, оскільки можна стверджувати, що, наприклад, два пікселі в очах і навколо очей на зображенні обличчя мають більш значущий взаємозв'язок, ніж два 16 пікселів, вибраних випадковим чином з зображення.

Згорткові шари в згорткових нейронних мережах складаються з декількох, складених карт ознак з пов'язаними нейронами. Ось чому ядра, пов'язані з картами ознак, також називаються фільтрами або каналами [3]. Згорткові шари можуть також накладатися на інші згорткові шари, що в більш неформальних термінах можна описати як побудову карт ознак на картах ознак або вивчення ознак більш високого рівня.

Ще одним важливим аспектом згорткових нейронних мереж є те, що спільне використання параметрів зменшує абсолютну кількість необхідних параметрів, на додаток до значного відносного зменшення параметрів у порівнянні з мережею, яка використовує тільки повністю з'єднані шари для виконання тієї ж задачі. Загалом, формула для кількості параметрів у згортковому шарі, що виконує двовимірну згортку, має наступний вигляд:

$$(M * X * Y + 1) * N, \tag{1.1}$$

де M – кількість карт ознак на вході (рахуючи червоно-зелено-сині канали на кольорових зображеннях), X та Y – висота та ширина згортки (3x3, 5x5 і т.д.), а N – кількість карт ознак на виході, яка додається для врахування зсуву [4].

Два останні аспекти згорток, які будуть обговорюватися в цьому розділі, – це крок і прокладка. Розмір кроку під час згортки дорівнює

одиниці, але можна використовувати і більші кроки. Прокладка входу використовується для збереження просторової роздільної здатності під час згортки, оскільки без неї він стискається на постійний коефіцієнт в обох вимірах в залежності від розміру згортки [2]. Іншим видом важливих шарів у згорткових нейронних мережах є об'єднуючі шари, в яких застосовується операція пулінгу (pooling). Об'єднання в загальному випадку полягає в перетворенні карт ознак в менші, більш агреговані карти ознак [2].

Найбільш поширеним видом об'єднання є максимальне об'єднання, яке бере області заданого розміру, що не перекриваються, і виводить найбільшу активацію в цій області, наприклад, розбиття вхідної карти на ділянки розміром 2 x 2, вибір максимального значення і створення вихідної карти, що складається з цих значень, але тільки на четверту частину від початкового розміру. Таким чином, шар об'єднання об'єднує вхідні дані з попереднього шару згортки, зберігаючи кількість карт об'єктів. Обґрунтуванням об'єднання є зменшення просторової розмірності в мережі при збереженні просторової інформації, використовуючи сильну кореляцію між точками даних, які знаходяться близько одна до одної [1].

Як і у випадку згорток, шари об'єднання можуть використовувати крок для досягнення ще більшої агрегації. З введенням шарів об'єднання структура згорткової нейронної мережі може бути більш зрозумілою. Основною метою згорткової нейронної мережі є перетворення просторового представлення вхідних даних у представлення, багате на ознаки, великий простір ознак, з якого можна класифікувати.

Таким чином, основна структура згорткової нейронної мережі – це згорткові шари, що чергуються з об'єднуючими шарами, в поєднанні з використанням методів і, як правило, пара повністю з'єднаних шарів з класифікатором в кінці. Ця структура існує вже багато років, принаймні, з часу створення архітектури Yann LeCuns LeNet5 в 1998 році [5], але ті ж самі ідеї все ще корисні і сьогодні. Варто зазначити, що базовий підхід все ще працює, тим більше, що дослідження навколо згорткових нейронних мереж є

інтенсивними, і що було зроблено і робиться багато серйозних проривів, особливо в області класифікації зображень і виявлення об'єктів.

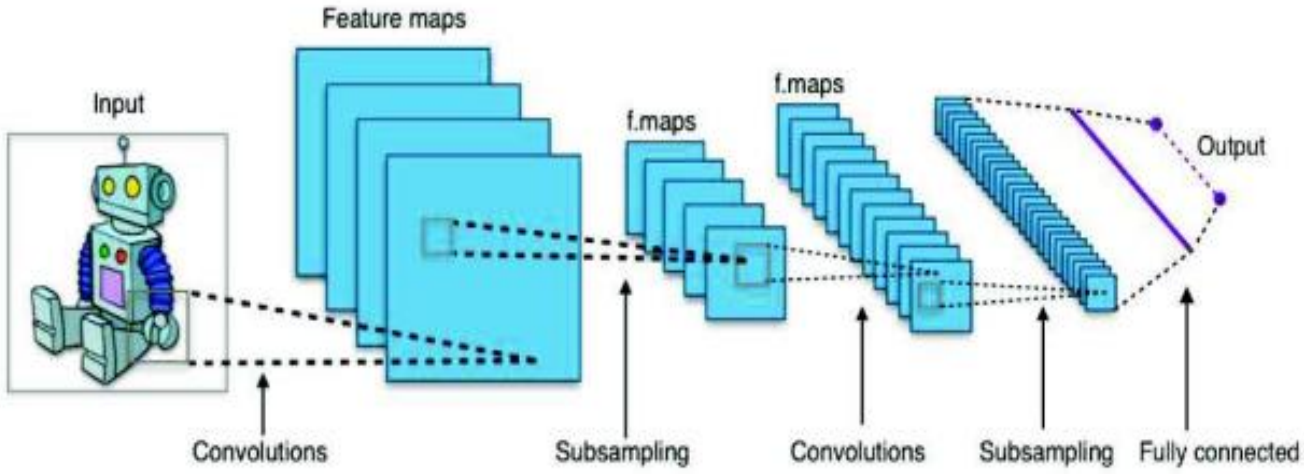

Рисунок 1.7 – Проста згорткова нейронна мережа.

1.2.3 Конструкції та архітектури нейронних мереж

У цьому розділі деякі з найбільш важливих конструкцій та архітектур нейронних мереж, особливо згорткових нейронних мереж, будуть представлені у відповідних деталях.

Загальним завданням, на яке спрямовані всі наступні архітектури, є класифікація зображень, щоб мати можливість класифікувати зображення якомога правильніше в ряд заздалегідь визначених категорій. Першим великим проривом після моделі LeNet5, згаданої в попередньому розділі, стала розробка AlexNet Алекса Крижевського в 2012 році [6], яка стала внеском у конкурс ImageNet.

ImageNet – це база даних, що складається з мільйона зображень, відсортованих за тисячею категорій, які використовуються для досліджень у галузі розпізнавання зображень та виявлення об'єктів [7]. AlexNet побудована на підході LeNet5, використовує випрямлені лінійні одиниці та працює на двох графічних процесорах [6]. Цей підхід був вдосконалений за допомогою мереж VGG (Visual Geometry Group), мереж з набагато більшою глибиною і

точністю [8].

Основним нововведенням було використання менших згорток розміром 3x3 і 5x5 замість набагато більших згорток, що використовувалися в AlexNet, і накладання шарів з використанням цих менших згорток один на одного [3]. Мережі виросли досить великими, однак, і навчання довелося проводити по частинах через величезну складність [3]. Продуктивність роботи мереж VGG під час виведення або класифікації після навчання також була досить дорогою [9].

Згаданий вище підхід має, як зазначалося, деякі очевидні недоліки, особливо щодо обчислювального навантаження при навчанні та обслуговуванні моделі. Дослідницька група в Google застосувала інший підхід з урахуванням цих міркувань, який призвів до того, що в 2014 році переможцем конкурсу ImageNet стала мережа GoogleNet – проект мережі, заснований на модулях Inception, і перша мережа, що використовує архітектуру Inception [10].

Основна ідея модуля Inception полягає в тому, щоб шари працювали над вхідними даними паралельно, як згорткові шари, так і об'єднуючі шари, шари з різними розмірами ядер, а потім об'єднували вихідні дані в один шар. GoogleNet почалася зі звичайного стека шарів, як у вищезгаданих моделях AlexNet і VGG, великої середньої частини модулів Inception, накладених один на одного, і, нарешті, глобального усереднюючого шару з класифікатором softmax, остання частина була натхненна статтею Network In Network [11].

Іншим важливим аспектом архітектури Inception є використання згорток 1x1. Згортки 1x1 служать двом корисним цілям в згорткових нейронних мережах; вони можуть бути використані для дешевого, з точки зору обчислень, додавання нелінійності за допомогою додаткових активацій і для дешевого зменшення кількості ознак у вихідному шарі [12].

Ці згортки використовуються для побудови так званих вузьких шарів,

конструкцій, що складаються з шару, який використовує згортки 1x1 для зменшення вибірки вихідних ознак, як правило, в чотири рази, другого шару з використанням більшого ядра на меншій кількості ознак і останнього шару з 1x1 згорток для збільшення вибірки ознак знову. Ці вузькі місця можуть зменшити кількість необхідних обчислень майже в десять разів [12]. Поєднання вузьких місць та модулів Inception призвело до успіху архітектури та наступних проектів мереж Inception V2 [13] та V3 [14].

Мережі Inception, в порівнянні з мережами VGG та підходом, що лежить в їх основі, підвищили точність в ImageNet Challenge, при цьому значно зменшивши кількість параметрів та кількість операцій [12].

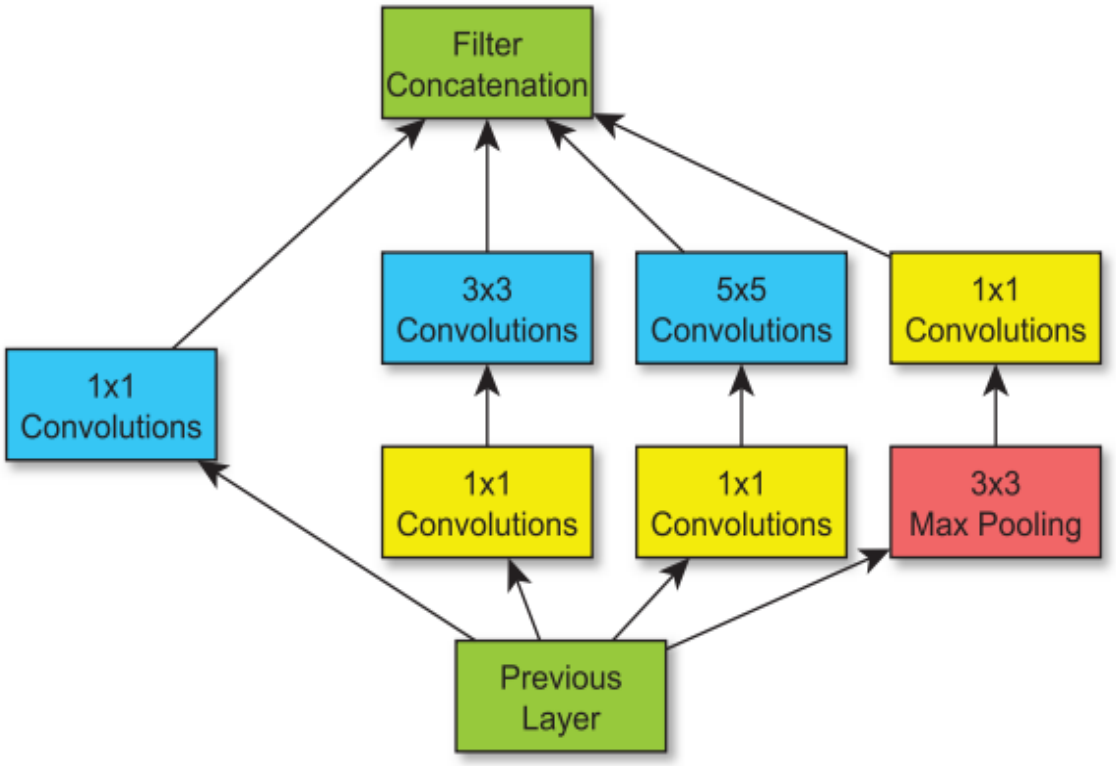

Рисунок 1.8 – Модуль Inception V1

Останній підхід, який буде представлений тут, – це підхід команди Microsoft, що стоїть за ResNet, переможцем конкурсу ImageNet 2015 року та

архітектурним підходом, який дозволив досягти відтворення зображення з безпрецедентної глибини мережі за рахунок використання нової техніки: залишкових блоків, що складаються з декількох шарів, і мереж, що складаються з цих блоків, залишкових нейронних мереж [15].

Основна ідея залишкових блоків полягає в тому, щоб пропустити вхід через два або більше шарів, а потім, що дуже важливо, додати початковий вхід до виходу другого шару, нейрон за нейроном, і застосувати функцію активації, щоб отримати вихід блоку. Можна сказати, що блок складається із залишкового зв'язку, шарів, через які проходить вхідний сигнал, та пропускного зв'язку, який обходить початковий вхідний сигнал [16].

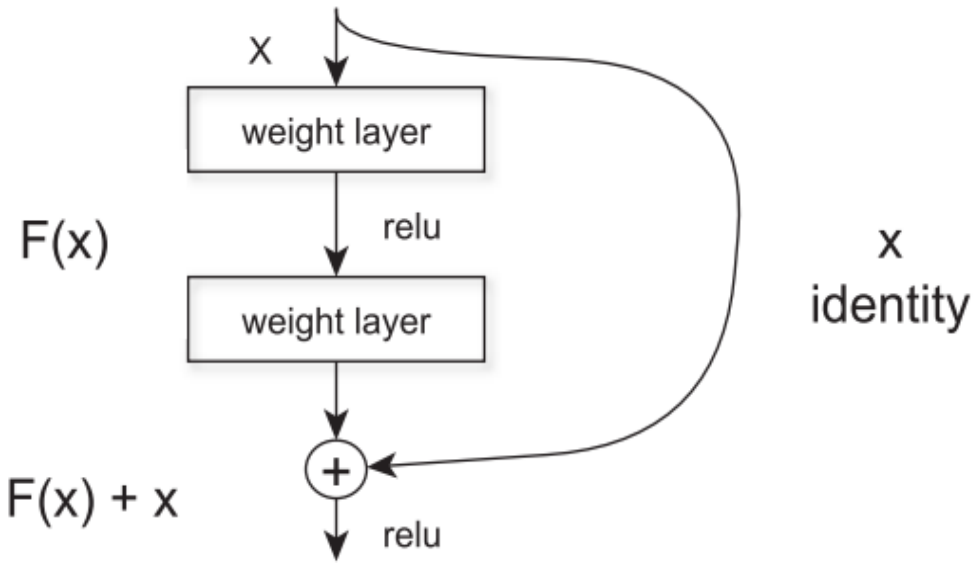

Рисунок 1.9 – Залишковий блок зі звичайним пропускним зв'язком.

Залишкове з'єднання, як правило, є вузьким шаром, а пропускне з'єднання, як правило, є тільки входом, але існують і інші варіанти, в першу чергу для скорочення просторової розмірності. Команді, що стоїть за ResNet, вдалося використати залишкові блоки для розробки та навчання згорткової

нейронної мережі з більш ніж тисячею шарів, що є рекордом у глибокому навчанні [3].

2 КОМП’ЮТЕРНИЙ ЗІР ТА ФРЕЙМВОРКИ ГЛИБОКОГО НАВЧАННЯ

Комп'ютерний зір займається автоматичним виділенням, аналізом та розумінням корисної інформації з одного зображення або послідовності зображень. Були використані згорткові нейронні мережі (CNN) в системах автоматичної класифікації зображень. У більшості випадків для класифікації використовуються ознаки з верхнього шару ШНМ, однак ці ознаки можуть не містити достатньо корисної інформації для правильного прогнозування зображення. У деяких випадках ознаки з нижнього шару мають більшу дискримінаційну здатність, ніж ознаки з верхнього. Тому застосування ознак тільки з певного шару для класифікації здається процесом, який не використовує потенційну дискримінантну силу навченого CNN в повній мірі. Через цю властивість було б доцільно об'єднати ознаки з декількох шарів.

2.1 Класифікація зображень

Класифікація зображень – це завдання, за допомогою якого обчислювальна машина може перевірити зображення та виділити «клас», до якого відноситься зображення. Групування зображень – це взаємодія обчислювальної машини, яка досліджує зображення та показує вам, що на ньому міститься. Рання характеристика зображення залежала від грубої піксельної інформації. Це означало, що обчислювальні машини розділятимуть зображення на окремі пікселі. Проблема в тому, що дві фотографії однієї речі можуть виглядати зовсім по-різному. Вони можуть мати різні основи, точки. Це стало надзвичайним випробуванням для обчислювальних машин щодо ефективного «аналізу» та впорядкування зображень.

Класифікація зображень має декілька застосувань і величезний потенціал, оскільки забезпечує незмінну якість. Безпілотні транспортні

засоби використовують характеристики зображення, щоб розпізнавати те, що навколо них, наприклад, дерева, людей, світлофори тощо. Класифікація зображень також може допомогти в медичних послугах.

Мета класифікації зображень полягає в тому, щоб розпізнати та відобразити, як надзвичайно тьмяний рівень (або затінення), світлі моменти, що відбуваються на зображенні, що стосується статті, на якій ці світлі предмети розглядаються на землі. Класифікація зображень є одним із основних компонентів комп'ютеризованого дослідження зображення [18]. Deep Learning разом із пристроями IoT зараз є найбільш використовуваною практичною реалізацією моделей класифікації [19].

Класифікація зображення з локалізацією – розміщення зображення в заданому класі та малювання рамки навколо об'єкта, щоб показати, де він розташований на зображенні (рисунок 2.1).

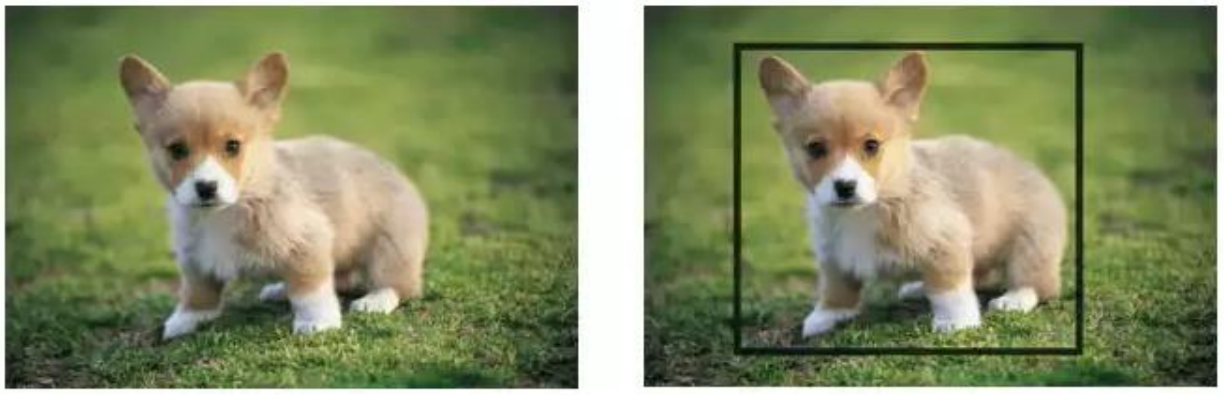

Рисунок 2.1 – Класифікація зображень проти класифікації зображень із локалізацією

Виявлення об'єктів – класифікація кількох різних об'єктів на зображенні та відображення розташування кожного з них за допомогою обмежувальних рамок. Отже, це різновид класифікації зображень із

завданнями локалізації для багатьох об'єктів.

Об’єктна (семантична) сегментація – ідентифікація конкретних пікселів, що належать кожному об’єкту на зображенні, замість малювання обмежувальних рамок навколо кожного об’єкта, як при виявленні об’єктів (рисунок 2.2).

Сегментація екземплярів – диференціація кількох об’єктів (екземплярів), що належать до одного класу (кожна особа в групі).

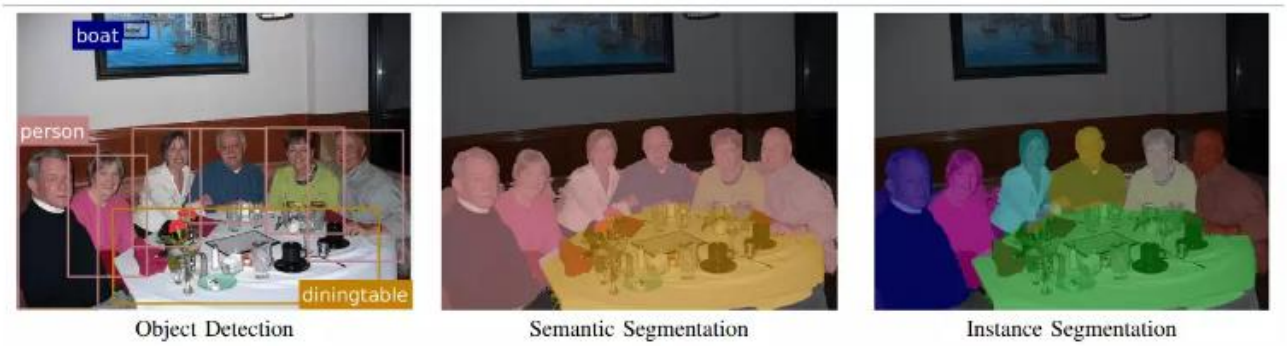

Рисунок 2.2 – Різниця між виявленням об’єктів, семантичною сегментацією та сегментацією екземплярів

Традиційні методи комп'ютерного зору і машинного навчання не можуть зрівнятися з людськими можливостями у виконанні таких завдань, як розпізнавання рукописних цифр або дорожніх знаків.

Наші біологічно правдоподібні, широкі і глибокі архітектури роботи штучних нейронних мереж можуть. Невеликі (часто мінімальні) рецептивні поля згорткових нейронів "переможець отримує все" дають велику глибину мережі, в результаті чого між сітківкою і зоровою корою утворюється приблизно стільки ж рідко з'єднаних нейронних шарів, скільки є у ссавців між сітківкою і зоровою корою.

Тренуються лише нейрони-переможці. Кілька глибоких нейронних колонок стають експертами на вхідних даних, попередньо оброблених

різними способами; їх прогнози усереднюються. Графічні карти дозволяють швидко навчатися. У дуже конкурентному тесті MNIST на розпізнавання почерку наш метод першим досягнув майже людської продуктивності. У тесті на розпізнавання дорожніх знаків він випереджає людину вдвічі. Ми також вдосконалюємо сучасні методи класифікації зображень у багатьох поширених тестах.

2.1 Фреймворки глибокого навчання

Алгоритми і функції, що використовуються в машинному навчанні, і особливо в глибокому навчанні, включають багато математики; тому реалізація нейронної мережі з нуля може бути складною і трудомісткою. Фреймворки глибокого навчання надають високорівневий API для того, щоб зробити реалізацію нейронних мереж простішою та ефективнішою. Фреймворки роблять реалізацію простішою та ефективнішою, абстрагуючись від основної математики та надаючи готові модулі та код.

Абстрагуючись від математичної реалізації, фреймворки знімають вимогу до програміста мати широку математичну підготовку, тим самим роблячи глибоке навчання більш простим у роботі і більш доступним.

2.2 Використані фреймворки

У цій магістерській роботі були використані наступні фреймворки: Google TensorFlow [17] та Microsoft Cognitive Toolkit (CNTK) [18]. Для TensorFlow використовувався сторонній API під назвою Keras. Keras надає більш високий рівень і більш зручний API, і здатний працювати як з TensorFlow, так і з іншим фреймворком глибокого навчання під назвою Theano [19], в якості бекенду.

2.2.1 Фреймворк глибокого навчання Google TensorFlow

Google TensorFlow – це фреймворк з відкритим вихідним кодом для глибокого навчання та машинного навчання, розроблений компанією Google і спочатку випущений 9 листопада 2015 року; актуальна версія була випущена 15 лютого 2017 року. TensorFlow написаний на C++ та Python і надає інтерфейси для Python, C++, Java, Haskell та Go [20]. TensorFlow спочатку розроблявся з метою проведення досліджень машинного навчання та глибоких нейронних мереж. TensorFlow підтримує обчислення на декількох GPU або CPU, як локально, так і розподілено [21].

2.2.2 Фреймворк глибокого навчання Microsoft CNTK

Microsoft CNTK – це фреймворк глибокого навчання з відкритим вихідним кодом, розроблений компанією Microsoft і спочатку випущений 25 січня 2016 року; і на момент написання цієї магістерської роботи не має стабільного випуску. CNTK написана на C++ [22], і надає інтерфейси для Python, C++, C# [23] та власної мови сценаріїв Microsoft для глибокого навчання: BrainScript. [24] CNTK спочатку був розроблений для самої Microsoft для швидкого навчання на великих наборах даних [25]. Багато власних продуктів Microsoft використовують CNTK, наприклад, Cortana, Bing та Skype. [18] CNTK підтримує обчислення на CPU [26] та декількох GPU, як локально, так і розподілено [27], а також моделі навчання та хостингу на Azure [28].

Існує багато інших фреймворків на вибір, включаючи: Theano [19], Torch [29], Caffe [30] та Deeplearning4j [31].

3 СТРУКТУРА ПРОЕКТУ

Метою цієї магістерської роботи є оцінка двох фреймворків для машинного навчання: Microsoft CNTK та Google TensorFlow. У цьому розділі буде представлено огляд різних частин оцінки та метод оцінки, що використовується для різних частин. Розділ 3.1 охоплює встановлення та системні вимоги двох фреймворків та їх залежності; а також підтримку мов програмування фреймворками. Розділ 3.2 охоплює особливості, функціональні можливості та документацію фреймворків, а також їх підтримку сторонніх API.

У підрозділі 3.3 розглянуто роботу фреймворків на двох широко використовуваних наборах даних для бенчмаркінгу моделей машинного навчання: змішаному наборі даних рукописних чисел Національного інституту стандартів і технологій (MNIST) та наборі даних мініатюрних кольорових зображень Канадського інституту перспективних досліджень (CIFAR-10).

У підрозділі 3.4 представлено розробку власного класифікатора зображень, процес розробки та ефективність якого буде використано при оцінці продуктивності та зручності використання фреймворків.

3.1 Інсталяція та системні вимоги

У цій частині оцінки проведено порівняння системних вимог Microsoft CNTK та Google TensorFlow, включаючи їх залежності.

При оцінці враховується наступне: простота та швидкість встановлення, системні вимоги, підтримка програмного та апаратного забезпечення, а також підтримка мов програмування.

Додатково будуть оцінюватися системні вимоги для можливості виконання необхідних розрахунків на графічному процесорі замість

центрального процесора, а також простота його налаштування. Також буде створено середовище розробки з необхідними інструментами.

3.2 Можливості, функціональні можливості та документація

У цій частині роботи оцінені та порівняні між собою можливості, функціональні можливості та документація CNTK та TensorFlow. Оцінка розпочинається з вивчення документації фреймворків для з'ясування наступного:

- які алгоритми машинного навчання забезпечують фреймворки, наскільки інтуїтивно зрозумілими є API фреймворків;
- наскільки добре задокументовані фреймворки та інші можливості та функції, які вони надають;
- а також їхня підтримка API сторонніх розробників.

Результати оцінки використані для порівняння фреймворків.

3.3 Бенчмаркінгові тести

У порівняльному дослідженні програмних фреймворків, таких як TensorFlow та CNTK, можна розрізняти більш м'які критерії, наприклад, особистий досвід та сприйнята простота використання, від більш жорстких критеріїв, наприклад, чисельні дані бенчмаркінгу та об'єктивна неспроможність програми, розробленої за допомогою цих фреймворків, виконати завдання за певний проміжок часу. На практиці не завжди легко провести таке розмежування, і воно не завжди є доречним для поставленого завдання.

Дані повинні бути інтерпретовані і поміщені в належний контекст, а інші суб'єктивні фактори майже завжди вступають у гру, щоб затуманити картину. Тим не менш, правильним підходом є визначення якомога більшої кількості об'єктивних, або «жорстких», аспектів дослідження, з результатами,

які здебільшого говорять самі за себе. Частиною магістерської роботи, яка повинна бути розроблена тут, є лабораторне середовище, де, наскільки це можливо, усувається або перетворюються на константи якомога більше змінних і параметрів у всіх аспектах процесу, від програмного забезпечення до апаратного забезпечення, щоб надійно порівняти продуктивність з точки зору витраченого часу, використання GPU / CPU, використання пам'яті та інших відповідних факторів.

Частково ця структура складається з розробки однакової точної моделі, аж до кожного окремого шару і кожного параметра, в API обох фреймворків. Інша частина полягає в тому, щоб переконатися, що відповідні набори даних обробляються однаково в парах моделей, які навчаються на них, і що відповідні набори даних добре підібрані з точки зору якості та доступності.

Набори даних, які будуть використовуватися для порівняльного аналізу фреймворків, – це база даних MNIST, що містить 70000 зображень, і база даних CIFAR-10, що містить 60000 зображень.

База даних MNIST складається з 70000 чорно-білих зображень рукописних цифр від 0 до 9 розміром 28x28 пікселів, і завданням, яке повинна вирішити відповідна модель, є класифікація зображень на десять класів, по одному для кожної цифри.

База даних CIFAR-10 складається з 60000 кольорових зображень розміром 32x32 пікселя, а завдання, яке повинна вирішити відповідна модель, полягає в тому, щоб класифікувати зображення за десятьма класами: літак, автомобіль, птах, кішка, олень, собака, жаба, кінь, корабель і вантажівка.

Значна частина роботи в цьому розділі проекту полягатиме в тому, щоб навчитися розбирати та переробляти існуючі моделі. Оскільки набори даних, згадані вище, MNIST і CIFAR-10, є загальними академічними еталонами продуктивності, вже є приклади, що постачаються разом з фреймворками, приклади, які готові до запуску за допомогою однієї команди.

Істотна робота, яка виконана – це рефакторинг прикладів, визначення того, що робить кожна частина коду, і переписування кожної моделі до тих

пір, поки архітектура, параметри і всі інші фактори проектування не стануть однаковими.

Заключна частина – власне бенчмаркінг та збір даних вимірювань – виконана за допомогою комбінації інструментів системного моніторингу для відстеження використання графічного/процесорного процесора та використання пам'яті.

Найбільш важливими для проведення бенчмаркінгу є наступні дані:

- час, необхідний для завершення повного навчального прогону з заданою моделлю на наборі даних;

- використання графічного процесора і, в особливих випадках, використання центрального процесора в операціях в секунду і у відсотках від загальної потужності з плином часу;

- використання пам'яті, знову ж таки, вимірюється як в гігабайтах, так і у відсотках від загального обсягу з плином часу.

Наведені дані повинні бути усереднені за кількома тренувальними прогонами; наприклад, п'ять тренувальних прогонів на сеанс бенчмаркінгу та модель. Візуалізація даних у вигляді діаграм буде необхідна в належний час, але тип діаграм і програмне забезпечення, яке буде використовуватися для їх побудови, виходить за рамки розробки магістерської роботи.

3.4 Впровадження класифікатора зображень

Доцільно впровадити певний класифікатор зображень; але яких саме зображень, і наскільки специфічна класифікація зображень; наприклад, класифікація різних об'єктів або, більш конкретно, класифікація різних типів одного і того ж об'єкту, залишається частиною реалізації.

Достатня кількість даних достатньої якості також необхідна для того, щоб мати можливість навчати і тестувати класифікатор зображень; що означає «достатня» в числовому вираженні і на практиці, також може бути віднесено до розділу «Впровадження». Тому перше завдання в цій частині

магістерської роботи полягає в тому, щоб знайти і розглянути різні джерела наборів даних; визначитися з проблемою розумної складності, де можна отримати достатню кількість даних для навчання і тестування класифікатора образів. Після цього дані, звичайно, потрібно завантажити. Друге завдання – приступити до реалізації класифікатора образів. Класифікатор образів буде реалізований як в Microsoft CNTK, так і в Google TensorFlow, використовуючи TensorFlow як back end, а Keras, сторонній API для глибокого навчання, як front end. Keras можна використовувати як зовнішній інтерфейс для TensorFlow.

Всі різні моделі класифікатора зображень в різних фреймворках реалізовані і розроблені на одній мові програмування і в одному середовищі розробки, щоб зробити моделі більш порівнянними. Мова програмування, яка використовується для реалізації – Python 3 , а середовище розробки – Microsoft Visual Studio 2015 Enterprise з встановленим плагіном Python Tools for Visual Studio.

Окрім використання однієї мови програмування та IDE 29 (інтегрованого середовища розробки). Третє і останнє завдання – навчання та тестування різних моделей, розроблених у двох фреймворках, Microsoft CNTK та Google TensorFlow з Keras в якості фронт-енду.

Зрештою, процес розробки моделей, їх продуктивність та точність тестування використані як частина оцінки та порівняння продуктивності, та зручності використання фреймворків.

У цій частині магістерської роботи більш м'які аспекти фреймворків представляють більший інтерес, оскільки початкова продуктивність фреймворків перевірена в ході бенчмаркінгових тестів.

До більш м'яких аспектів, які розглядаються, відносяться: наскільки інтуїтивно зрозумілими є API фреймворків, простота розробки та швидкість розробки.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1 Інсталяція та системні вимоги

Для того, щоб зібрати необхідну інформацію для оцінки системних вимог, програмної та апаратної підтримки, а також підтримки мови програмування, було вивчено документацію TensorFlow, Keras та CNTK.

З метою оцінки легкості та швидкості інсталяції фреймворки були завантажені та встановлені. Це більш суб'єктивна частина цієї частини оцінки. Аспекти, на яких ґрунтуються висновки: кількість кроків, необхідних для того, щоб мати можливість використовувати фреймворк, та сприйнята легкість виконання вищезгаданих кроків.

Нижче описані кроки, які використовувалися для встановлення кожної відповідної системи.

По-перше, необхідно було налаштувати середовище розробки. Оскільки розробка ведеться на мові Python, з домашньої сторінки Python «https:// www.python.org/» було завантажено та інстальовано Python 3. В якості IDE було використано Microsoft Visual Studio 2015 Enterprise. Для використання Python у Visual Studio необхідно встановити розширення Python Tools for Visual Studio (PTVS). Для встановлення PTVS інсталяція Visual Studio була модифікована через панель керування Windows з додаванням розширення PTVS. Google TensorFlow був завантажений та встановлений через Visual Studio, з PTVS, за допомогою вбудованої панелі інструментів. Для можливості використання графічного процесора було встановлено версію GPU TensorFlow 0.12.1. Для використання GPU-версії TensorFlow потрібно додатково завантажити два пакети: NVIDIA CUDA Toolkit 8.0 та бібліотеку NVIDIA cuDNN v5.1 (CUDA Deep Neural Network), які завантажені з сайтів «https://developer.nvidia.com/cuda-downloads» та «https://developer.nvidia.com/cudnn» відповідно. Dll-файл cuDNN розміщено в

CUDA-теку, створену після інсталяції CUDA Toolkit. Keras завантажено та встановлено за допомогою Visual Studio, з PTVS, за допомогою вбудованого tool pip. Встановлено версію 1.2.2. Pip обробляє та встановлює всі залежності, пов'язані з python, проте версії scipy та numpy, встановлені через pip, були неправильними, і їх потрібно було завантажити та встановити вручну. Правильні версії scipy та numpy, необхідні для Keras, завантажені з сайту: «www.lfd.uci.edu/~gohlke/pythonlibs/». Завантажені whl-файли потрібних версій scipy та numpy були встановлені за допомогою pip через командний рядок Windows. Microsoft CNTK завантажений з сайту «github.com/Microsoft/CNTK/wiki/Setup-Windows-Binaryand» та встановлений вручну. Встановлена версія Beta GPU версії 10 з використанням бінарної установки за допомогою скриптів. Бінарна інсталяція CNTK зі скриптами включає всі залежності, включаючи NVIDIA CUDA Toolkit 8.0 та NVIDIA cuDNN v5.1, які були встановлені разом із самою CNTK; інсталяція також включала середовище Anaconda, в якому повинна працювати CNTK.

4.2 Особливості, функціональні можливості та документація

Робота, яка була виконана для порівняння двох фреймворків, CNTK та TensorFlow, згідно з критеріями, наведеними у відповідному підрозділі плану проекту, складалася з поглибленого читання документації (у випадку TensorFlow, в першу чергу, документації компанії Keras) для розуміння принципів роботи фреймворків, нотаток, зроблених під час процесу розробки та тестування, та остаточної оцінки зібраної інформації на основі уточненого набору критеріїв.

Було обрано наступні чотири критерії:

- особливості та функціональні можливості;
- кількість та якість документації;
- розширюваність та інтероперабельність;
- а також когнітивне навантаження.

Оскільки документація є необхідним інструментом у процесі розробки, фреймворки оцінювалися за кількістю та якістю наявної документації. Кроки, необхідні для пошуку потрібної інформації в документації, кількість доступної інформації, легкість доступу до навчальних посібників, частота оновлень документації та легкість навігації по вихідному коду – все це аспекти документації, за якими оцінювалися фреймворки в процесі роботи.

Кількість відсутньої та неповної інформації в документації фреймворків також розглядалася в контексті цього критерію. Розширюваність та інтероперабельність систем не була критерієм, який безпосередньо перевірявся ні в процесі порівняльного аналізу, ні в процесі розробки та впровадження класифікатора зображень. Аспекти, які оцінювалися тут, полягали в тому, як може працювати процес розгортання і як фреймворки інтегруються з іншими мовами, бібліотеками і фреймворками.

Інформація, отримана тут, була взята з документації фреймворків. Когнітивне навантаження – це поняття, що виникло в когнітивній психології, що відноситься до загальної кількості розумових зусиль, які використовуються в робочій пам'яті. У контексті цього звіту цей термін означає кількість розумових зусиль, необхідних для вивчення, використання та розвитку в рамках системи. Мотивація цього критерію полягає в тому, що загалом, за інших рівних умов, краще мати просту у використанні систему, ніж систему, яка вимагає багато зусиль від розробника.

Інші аспекти, згруповані в рамках цього критерію, який використовувався в процесі оцінювання, – це те, наскільки система відповідає принципу найменшого здивування, наскільки необхідний шаблон і наскільки добре реалізація в системі відповідає літературним джерелам.

4.3 Бенчмаркінгові тести

Система, що використовується в цій частині магістерської роботи:

- операційна система: 64-розрядна «Windows 8.1»;

- процесор «Intel Core i7-4800MQ» 2.80GHz;
- оперативна пам'ять 32GB GPU: «NVIDIA Quadro K2100M» 2GB.

Інше: «TensorFlow-gpu 1.0.1», «Keras 2.0.2», «CNTK 2.0.beta15.0», «Numpy 1.12.1» + «mkl», «Scipy 0.19.0», «PyYaml 3.12», графічний драйвер «NVIDIA» версії 376.51, «NVIDIA CUDA Toolkit 8.0», «NVIDIA cuDNN v5.1». Ми оновили TensorFlow, Keras та CNTK з моменту їх першого встановлення перед початком бенчмаркінгових тестів. Версії фреймворків під час цієї частини проекту були такими: «TensorFlow GPU 1.0.1», «Keras 2.0.2», «CNTK GPU beta 15». Як Keras, так і CNTK надали готові приклади коду для навчання нейронних мереж на наборах даних MNIST та CIFAR-10.

Для початку було використано наведені приклади коду для кожного відповідного набору даних і намагалися зробити їх максимально схожими в обох фреймворках, враховуючи знання про базову реалізацію фреймворків та теоретичну базу. Під час реалізації нейронних мереж як для MNIST, так і для CIFAR-10 дотримувалася однакова робоча процедура.

Робоча процедура була наступною: вивчення коду та документації для розуміння кожного кроку в коді, визначення того, де приклади коду відрізняються, та вивчення документації для пошуку заміни рішення, яке можна зробити максимально схожим в обох середовищах.

Основним фактором у цій частині проекту є не точність тестування моделей, а оцінка продуктивності фреймворків, особливо часу навчання. Моніторинг системи в процесі навчання здійснювався за допомогою «ASUS GPU Tweak». Час навчання для кожної епохи виводився на стандартний вивід за допомогою вбудованих у фреймворки функцій. Для кожної моделі в кожному фреймворку було проведено по п'ять окремих тренувальних та тестових прогонів, всього по чотири моделі та двадцять прогонів. Моделі, навчені на MNIST, пройшли через двадцять навчальних епох у кожному прогоні. Моделі, навчені на CIFAR-10, пройшли через сорок навчальних епох у кожному прогоні.

4.4 Реалізація класифікатора зображень

Система, що використовується в цій частині магістерської роботи:

- операційна система: 64-розрядна «Windows 8.1»;
- процесор «Intel Core i7-4800MQ» 2.80GHz;
- оперативна пам'ять 32GB GPU: «NVIDIA Quadro K2100M» 2GB.

Інше: «TensorFlow-gpu 1.0.1», «Keras 2.0.2», «CNTK 2.0.beta15.0», «Numpy 1.12.1» + «mkl», «Scipy 0.19.0», «PyYaml 3.12», графічний драйвер «NVIDIA» версії 376.51, «NVIDIA CUDA Toolkit 8.0», «NVIDIA cuDNN v5.1». Основна відмінність між підходом, викладеним у відповідному розділі технічного завдання, та фактично виконаною роботою полягає в тому, що результуючий класифікатор зображень був написаний на мові Keras, з використанням TensorFlow в якості бекенду, а не на CNTK.

Проблеми, що виникли під час роботи з CNTK, у поєднанні з часовими обмеженнями унеможливили реалізацію класифікатора в обох фреймворках. Це означає, що порівняльний аспект у цій частині магістерської роботи довелося виключити, залишивши лише дослідницький аспект. Решта процесу відбувалася відповідно до плану магістерської роботи і складалася з двох частин: пошук і вибір набору даних, а потім реалізація класифікатора зображень шляхом кодування і навчання отриманої мережі.

Набір даних, який ми вирішили використовувати, був варіантом набору даних CIFAR, тим самим набором даних, який ми використовували в процесі порівняльного аналізу, як описано в попередньому розділі, але з іншим маркуванням. Набір даних CIFAR-100 має 100 класів, класів, які згруповані у 20 суперкласів. Кожен приклад у наборі даних позначений як класом, до якого він належить, так і суперкласом, до якого він належить. Мотивація вибору використання CIFAR-100 в якості набору даних для навчання класифікатора зображень може бути розділена на декілька причин. Першою і головною причиною є те, що цей набір даних широко використовується в наукових дослідженнях і розробках, а це означає, що якість даних не була

невідомою і на неї можна було розраховувати. Інша причина полягає в тому, що використання існуючого, легкодоступного набору даних дозволило заощадити значний час і зусилля, усунувши роботу, необхідну для збору, підготовки та попередньої обробки даних, необхідних для навчання класифікатора зображень. Третя і остання причина полягає в тому, що при розробці і впровадженні мережі можна було використати результати роботи, виконаної в рамках порівняльного аналізу, оскільки відповідні проблеми були опрацьовані і протестовані в рамках цієї магістерської роботи.

Мережа була спроектована та реалізована з використанням цілого ряду різних методів, а саме: експоненціальних лінійних блоків, залишкових блоків, глобального середнього об'єднання, відсіву та доповнення даних. Вибір функції активації, експоненціальної лінійної одиниці (ELU), є варіантом функції ReLU, яка дозволяє протікати невеликий градієнт, якщо вхідний сигнал є негативним, властивість функції, яка, як передбачається, має нормалізуючий вплив на вихід, роблячи активації ближчими до середнього значення.

Було використано два варіанти, перший з яких складався з незміненого пропускного з'єднання і звичайного шару вузького місця, а другий використовував згортки 1x1 з кроком 2 для зменшення просторової розмірності. Причина, по якій були обрані залишкові блоки, частково пов'язана з вражаючими результатами, досягнутими при використанні мереж з використанням залишкових зв'язків, частково з відносною простотою розуміння і реалізації їх в класифікаторі.

Замість використання повністю з'єднаних рівнів перед рівнем класифікатора в кінці мережі був використаний рівень глобального усередненого об'єднання. Операція глобального середнього об'єднання – це варіант об'єднання, який усереднює всі карти ознак і виводить одновимірний тензор, розмір якого дорівнює кількості ознак на вході. Вихідні дані можуть бути подані безпосередньо до шару класифікації. Причиною використання цього методу є те, що кількість параметрів у повністю пов'язаних шарах має

тенденцію до швидкого зростання, а також те, що операція об'єднання використовує просторову кореляцію в даних.

Регуляризація у вигляді відсіювання використовувалася один раз для кожного залишкового блоку і в початкових згорнутих шарах. Доповнення даних у вигляді випадкових вертикальних та горизонтальних зсувів, а також горизонтальних перевертань також використовувалось при навчанні мережі. Причиною використання доповнення даних була регуляризація мережі та штучне роздування кількості навчальних вибірок. Замість звичайного стохастичного градієнтного спуску була випробувана і використана нова версія алгоритму. Він був доповнений планувальником швидкості навчання, який зменшував швидкість навчання поетапно під час навчальної сесії, причому вся навчальна сесія складала двісті епох.

Дизайн доопрацьованої мережі або класифікатора зображень, як показано в додатку В, складався з невеликої кількості звичайних, послідовних згорткових шарів на початку і більшої частини мережі, що складалася із залишкових блоків зі зростаючою кількістю ознак у міру поглиблення мережі, і, нарешті, глобального середнього шару об'єднання, за яким слідував шар softmax для класифікації.

Важливим аспектом проектування було зробити його якомога меншим з точки зору кількості параметрів та операцій. Оскільки комп'ютери, що були в розпорядженні проекту, мали порівняно слабкі графічні карти порівняно з апаратним забезпеченням, що використовується у відповідних дослідженнях і розробках, отримана мережа повинна була бути здатною до навчання принаймні на одному з наявних комп'ютерів. Таким чином, завдання полягало в тому, щоб зробити якомога кращий дизайн при обмежених ресурсах. Мережа була реалізована за допомогою бібліотек, перелічених вище. Точність, виміряна у відсотках, і втрати як на навчальному, так і на тестовому наборах були візуалізовані з використанням Matplotlib, бібліотеки з відкритим вихідним кодом, написаної на мові Python.

5 РЕЗУЛЬТАТИ ТА ОЦІНКА

5.1 Встановлення та системні вимоги

Системні вимоги «Google TensorFlow». Операційна система: 64-розрядна «Windows». «Ubuntu 14.04+». «Mac OS X». Використання графічного процесора: «CUDA Toolkit 8.0. cuDNN v5.1». Графічний процесор з «CUDA Compute Capability 3.0+». Додатково щодо Ubuntu: бібліотека «libcupti-dev». Підтримувані мови: Повністю підтримуються: «Python». Частково підтримуються: «C++», «Java», «Go». Підтримувані спільнотою: «C#», «Haskell», «Julia», «Ruby», «Rust». Системні вимоги «Microsoft CNTK:s». Операційна система: «Windows 64-bit 8.1», «64-bit 10». «Windows Server 2012 R2+». «Ubuntu 64-bit 14.04+». З використанням графічного процесора: «CUDA Toolkit 8.0. cuDNN v5.1» «NVIDIA CUB v. 1.4.1» GPU з «CUDA Compute Capability 3.0+». Інше: «Windows: Visual Studio 2015 Update 3+», «Microsoft MPI (Message Passing Interface)» версія 7.0. «Linux: GNU C++ 4.8.4+», «Open MPI v. 1.10.3+». Підтримувані мови: Повністю підтримуються: «Python», «C++», «BrainScript». Тільки для оцінки моделі: «C#» + інші мови «.NET».

Як видно з наведених вище системних вимог фреймворків, фреймворки мають схожі вимоги щодо використання графічного процесора, також вони мають схожу підтримку операційних систем, за винятком того, що TensorFlow має підтримку «Mac OS X», що є сильним плюсом для TensorFlow.

Що стосується підтримки мов програмування, обидва фреймворки мають спільну підтримку «Python» та «C++», в той час як TensorFlow, здається, має більшу підтримку спільноти, і тому підтримує більше мов в цілому, однак, CNTK має більше мов, які він повністю підтримує. Нарешті, CNTK має більше інших системних вимог. Простота встановлення

Фреймворки сильно змінилися з того часу, як їх вперше встановили, і їх встановлення, особливо CNTK, стало набагато простішим; тому початкове порівняння встановлення більше не є дійсним. Також варто зазначити: було встановлено та використано фреймворки лише у Windows.

Спочатку інсталяція CNTK:s вимагала набагато більше кроків, і кроки також були складнішими для виконання, через необхідність вводити багато команд у командному рядку; крім того, необхідне завантаження було великим, приблизно 2 ГБ, єдиним полегшуючим фактором було те, що програмні залежності CNTK:s йшли в тому ж пакеті і були встановлені одночасно з CNTK.

Інсталяція TensorFlow залишилася практично такою ж, і вимагає лише простої команди pip, а залежності TensorFlow (CUDA Toolkit 8.0 і cuDNN v5.1), однак, потрібно встановлювати вручну. Однак тепер установка CNTK набагато простіша і швидша, і може бути встановлена за допомогою pip, також необхідний обсяг завантаження набагато менший, приблизно 250 МБ.

Налаштування для виконання обчислень на GPU є простим в обох фреймворках, потрібно лише встановити GPU-версію кожного фреймворку, а також виконати перераховані вище системні вимоги для використання GPU. GPU-версія CNTK:s працює без ручного встановлення «NVIDIA CUB (CUDA Unbound)», тому вона або не потрібна, або входить до складу «NVIDIA Toolkit».

Підсумовуючи, можна сказати, що на момент написання цієї магістерської роботи, фреймворки є рівними з точки зору простоти інсталяції. Що стосується налаштування обчислень на GPU, то можна зробити висновок, що фреймворки є рівними і в цьому плані, оскільки вони вимагають однакової кількості кроків та залежностей для встановлення.

Що стосується системних вимог та підтримки, то рішення про те, який фреймворк є кращим у цьому відношенні, в основному зводиться до того, яку операційну систему та мову програмування використовувати; оскільки CNTK та TensorFlow підтримують мови, які не підтримує інший, а також те, що

TensorFlow підтримує «Mac OS X», а CNTK не підтримує.

5.2 Особливості, функціональні можливості та документація

Було виявлено, що з точки зору інструментів, можливостей та загальної функціональності, фреймворки глибокого навчання, оцінені в дослідженні, по суті, еквівалентні за здатністю реалізовувати нейронні мережі та глибоке навчання.

Як CNTK, так і TensorFlow/Keras мають весь необхідний функціонал для використання, модифікації та реалізації будівельних блоків нейронних мереж, таких як функції активації, шари нейронів, функції вартості, стохастичний градієнтний спуск та його різновиди, регуляризація в різних її формах та інші. Простіше кажучи, всі методи та концепції, можуть бути реалізовані за допомогою або CNTK, або TensorFlow, або Keras з використанням TensorFlow в якості рушія.

Далі було виявлено, що API як CNTK, так і Keras є переважно об'єктно-орієнтованими, з основною відмінністю в тому, що Keras інкапсулює мережу в об'єкт Model. Модельний об'єкт інстанціюється на мережевій конфігурації, конфігурації, яка може бути виконана послідовним або функціональним способом.

Процес навчання конфігурується і запускається через виклики методів до інкапсульованого об'єкта мережі, а також є виклики методів для інших функціональних можливостей. CNTK, навпаки, було виявлено, що реалізація процесу навчання розбита на значно більшу кількість класів, ніж у Keras, однак конфігурація мережі все ще може бути виконана послідовним або функціональним способом, як і в Keras. [30]

Однією з помітних відмінностей, яку було виявлено, це те, що такі набори даних, як «MNIST» і «CIFAR-10/100», включені в структуру Keras як модулі, але не в CNTK. Обробка даних щодо вищезгаданих наборів даних виявилася громіздкою і залежною від спеціально розроблених скриптів у випадку CNTK, до такої міри, що це негативно вплинуло на наступні частини

магістерської роботи. Загальна якість та кількість документації в обох системах була визнана недостатньою в декількох аспектах, але загальна оцінка полягала в тому, що документація TensorFlow та Keras була кращою, ніж документація CNTK на момент написання цього звіту.

Було виявлено, що в документації обох систем не вистачає інформації про те, як інформація підтримується в актуальному стані порівняно з оновленнями програмного забезпечення, а також наскільки ретельною та повною є інформація в кожному розділі документації. Однією з переваг Keras та TensorFlow над CNTK було те, що документація обох систем була зібрана в одному місці – на відповідних веб-сайтах кожної системи.

Документація CNTK, навпаки, була розпорошена між веб-сайтом для API Python та репозиторієм Github, що одночасно збільшувало кількість кроків, необхідних для пошуку необхідної інформації в документації, та ускладнювало її пошук.

Було виявлено, що навчальні посібники, надані в документації обох фреймворків, не містять стільки відповідної інформації, скільки прикладів, наданих у репозиторіях вихідного коду. Щодо вихідного коду, то було виявлено, що Keras має більш читабельний та доступний вихідний код з двох фреймворків, які безпосередньо використовувалися авторами.

В процесі бенчмаркінгу час від часу виникала потреба зазирнути в реалізацію фреймворків, а в репозиторії CNTK знайти код, що цікавив, було значно складніше. В якості останнього спостереження автори також відзначили, що, здається, було докладено більше зусиль, щоб зробити документацію Keras і TensorFlow більш презентабельною і зручною для навігації, на відміну від CNTK.

5.3 Розширюваність та інтероперабельність

Розширюваність фреймворків виявилася досить широкою, головним чином завдяки тому, що основні API фреймворків написані на мові Python і

тому можуть бути розширені за допомогою обраних пакетів та бібліотек з усієї екосистеми Python. Кількість інших мов, що підтримуються в обох фреймворках, як детально пояснено в попередньому розділі, також було визнано покращенням у цьому відношенні. Що стосується розгортання, було виявлено, що моделі, розроблені як в Keras, так і в TensorFlow, можуть використовувати TensorFlow Serving, обслуговуючу систему для машинного навчання, як на Windows, так і на Linux. Моделі, розроблені за допомогою CNTK, можуть бути розгорнуті за допомогою CNTK NuGet Package для Visual Studio та API, який він надає в Windows.

Azure, служба хмарних обчислень Microsoft, також може бути використана для розгортання моделей, підготовлених за допомогою CNTK, а двійкові файли можуть бути використані для розгортання моделей в Linux.

5.4 Когнітивне навантаження

З точки зору простоти вивчення, використання та розвитку, було виявлено, що Keras значно зручніший для користувача, ніж CNTK. Основними факторами, які, свідчать на користь Keras, є гнучкість у виборі між конвенцією та конфігурацією, кількість коду, який був ефективно та належним чином інкапсульований, та зрозумілі назви функцій, класів, методів та модулів. Було виявлено те, що як конвенція, у вигляді рядків для неявної інстанції поведінки за замовчуванням, так і конфігурація, у вигляді явно створених об'єктів, можуть використовуватися і взаємозамінюватися в міру необхідності при розробці в Keras.

Також було виявлено, що модулі та функції нижчого рівня з TensorFlow можуть використовуватися та взаємодіяти з кодом, написаним на Keras, на додаток до його використання в якості обчислювального бекенду, хоча у магістерській роботі не використовували ці можливості на практиці в проекті.

Як згадувалося раніше в цьому розділі, Keras інкапсулює конфігурацію

нейронної мережі в клас Model, клас, з яким було легко працювати завдяки зрозумілим назвам методів і ключових слів-параметрів. Оскільки багато функціональних можливостей було реалізовано в класі Model, кількість необхідних шаблонів також було зведено до мінімуму.

Як згадувалося раніше, функціональність в CNTK була розпорошена по декількох класах, багато з яких, не відповідали принципу найменшої несподіванки щодо імен, а іноді і поведінки. Було також виявлено, що численні класи, згадані вище, в поєднанні з іменами, які не завжди були зрозумілими, іноді ускладнювали розуміння того, що насправді відбувається в коді. Постійна потреба у явному конфігуруванні та інстанціюванні численних об'єктів, які у багатьох випадках одразу ж використовувалися для інстанціювання інших об'єктів, робила реалізацію частини навчального процесу.

Можна зауважити, що ні явна конфігурація, ні функціональність, розподілена в декількох класах, не обов'язково є поганими, але реалізація у випадку з CNTK була визнана недостатньо хорошою. Щодо того, наскільки добре реалізації відповідають літературним джерелам, було виявилено, що Keras виграв і в цьому відношенні, через те, що CNTK має явно іншу реалізацію, яка вимагає перерахунку певних параметрів.

Підсумовуючи, було виявилено, що фреймворки надають еквівалентний набір можливостей та функціоналу, і фреймворки більш ніж здатні будувати нейронні мережі.

Було виявлено, що документація обох фреймворків є недостатньою як за якістю, так і за кількістю, однак Keras має невелику перевагу в тому, що його документація зібрана в одному місці, тоді як документація CNTK розподілена по різних сайтах. Було виявлено, що Keras є більш дружнім до новачків і з ним легше працювати, порівняно з CNTK.

Було виявлено, що CNTK не відповідає літературі, маючи натомість власну реалізацію, яка, в свою чергу, вимагає перенавчання, а також вимагає перерахунку параметрів для того, щоб функціонувати відповідно до

літератури; Keras, з іншого боку, не вимагає такого переналагодження, що є великим плюсом на наш погляд.

5.5 Порівняльні тести

Нижче в таблицях 5.1 – 5.4 наведено результати часу навчання, по одному для кожного набору даних та фреймворку.

Таблиця 5.1 – Результати CNTK MHICT

Тест (№)	Загальний час (с)	Середній час (с)	Максимальний час (с)
1	328	16.4	18.1
2	324	16.2	18.1
3	324	16.2	18.7
4	324	16.2	18.1
5	322	16.1	18.1
1-5	1622	16.2	18.7

Таблиця 5.2 – Результати Keras/TensorFlow MNIST

Тест (№)	Загальний час (с)	Середній час епохи(с)	Максимальний час епохи(с)
1	444	22.2	26
2	451	22.6	26
3	447	22.4	26
4	446	22.3	26
5	444	22.2	26
1-5	2232	22.3	26

Таблиця 5.3 – Результати CNTK CIFAR-10

Тест (№)	Загальний час (с)	Середній час епохи(с)	Максимальний час епохи(с)
1	2776	69.4	74.7
2	2766	69.2	74.2
3	2756	68.9	73.4
4	2756	68.9	73.1
5	2760	69.0	74.6
1-5	13814	69.1	74.7

Таблиця 5.4 – Результати Keras/TensorFlow CIFAR-10

Тест (№)	Загальний час (с)	Середній час епохи(с)	Максимальний час епохи(с)
1	3731	93.3	98
2	3745	93.6	99
3	3759	94.0	98
4	3758	94.0	101
5	3701	92.5	97
1-5	18694	93.5	101

5.4 Впровадження класифікатора зображень

Вихідний код, використаний для кожного набору даних та фреймворку був описаний у додатку А та Б. Вихідний код здебільшого збігається з прикладами коду для наборів даних, наведеними у фреймворках. До коду прикладів були внесені зміни, щоб зробити нейронні мережі в прикладах настільки схожими, наскільки це можливо зробити. Знайти необхідну

інформацію в документації фреймворків не вдалося, щоб з'ясувати деякі аспекти функціонування та реалізації прикладу коду, тому залишаються деякі аспекти коду, щодо яких важко бути впевненим, ці аспекти перераховані нижче для прозорості.

«Рандомізувати» в CNTK; яка еквівалентна функція в Keras? Чи виконується вона за замовчуванням у Keras, чи не виконується взагалі? Перемішування виконується за замовчуванням в Keras, як це робиться в CNTK? Параметри до оптимізаторів, наприклад, епсилон (ϱ) в Adagrad, де він задається у CNTK? Чи задається взагалі? Як він називається в CNTK, і яке його значення за замовчуванням? У CNTK можна задати як розмір навчальної міні-партії, так і розмір тестової міні-партії, чи можливо це в Keras, і як, і де це робиться? У CNTK розмір мінібатча динамічно змінюється під час навчання, де це робиться в коді, чи це робиться автоматично? Які значення за замовчуванням? Чи можна це зробити в Keras?

Враховуючи перелічені вище невизначеності та експериментальну установку, представлену в розділі 4.3, час навчання з використанням CNTK є стабільно швидшим, ніж з використанням Keras з TensorFlow як бекендом, як можна побачити в таблицях 5.1 – 5.4.

Варіація часу навчання є відносно низькою в обох фреймворках, хоча варіація є дещо вищою при використанні Keras з TensorFlow в якості бекенда, особливо виділяється останній прогін на CIFAR-10 з використанням Keras з TensorFlow в якості бекенда, з різницею в 30 секунд до найближчого сусіда, див. таблицю 5.4. Цікаво, що перша епоха постійно була епохою, яка займала найбільше часу для завершення, див. колонку "Максимальний час епохи" в таблицях 5.1 – 5.4. Провівши деяке тестування після того, як були зібрані результати, представлені в таблицях 5.1 – 5.4, можна прийти до висновку, що перші епохи займали більше часу.

При запуску скриптів без налагодження перші епохи займали приблизно такий же час, як і решта епох. Щодо порівняння CPU та GPU, то було виявлено, що GPU настільки переважає за швидкістю навчання, що

подальші дослідження в цьому питанні були визнані зайвими.

Цікавим аспектом щодо використання пам'яті є те, що Keras за замовчуванням використовує всю доступну VRAM, в той час як CNTK - ні, проте за допомогою декількох рядків коду можна налаштувати Keras так, щоб він використовував лише необхідну йому пам'ять.

Підсумовуючи, було виявлено, що CNTK дає коротший час навчання мереж порівняно з Keras з TensorFlow як бекендом, див. таблиці 5.1 – 5.4, враховуючи перелічені невизначеності та експериментальну установку, представлену в розділі 4.3.

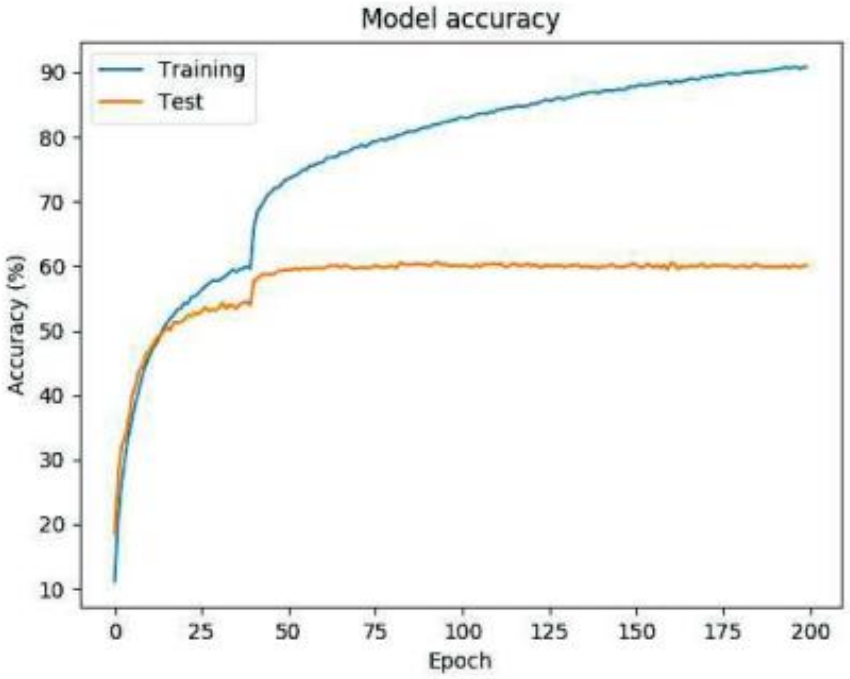

Рисунок 5.1 – Графік точності для першого прогону

Було виявлено, що графічний процесор настільки перевершує центральний процесор за швидкістю навчання, що якщо хтось має графічний процесор, він повинен використовувати його замість центрального процесора. Використання GPU та VRAM було однаковим в обох фреймворках.

Діаграми, представлені тут, є результатами трьох тренувальних сесій

або прогонів у вигляді двох діаграм для кожного прогону, одна діаграма відображає точність тренування і тесту, а інша – втрати тренування і тесту.

Під точністю в цьому контексті мається на увазі відношення кількості правильно класифікованих прикладів до кількості прикладів у наборі в цілому, а під втратою – розмір поточної помилки або розмір цільової функції. Мета тут, як і у всіх видах машинного навчання, полягає в мінімізації втрат і максимізації точності як на навчальній, так і на тестовій множині.

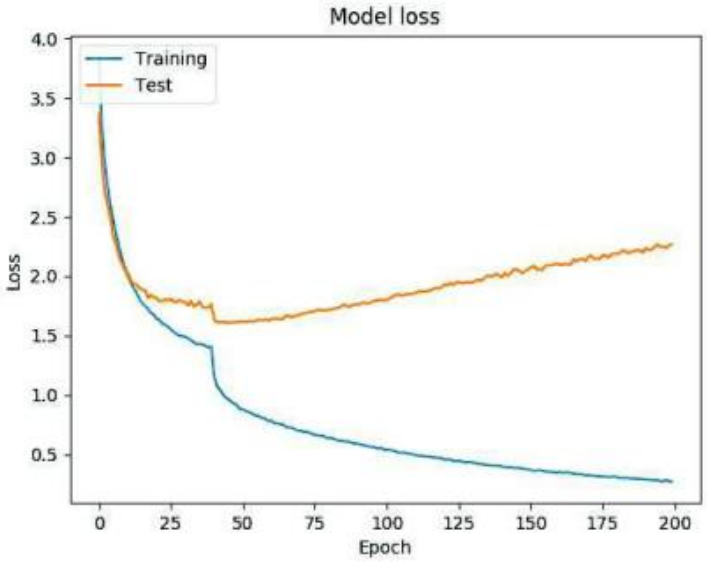

Рисунок 5.2 – Графік втрат для першого прогону

Процес навчання, як видно з діаграм, призвів до кінцевої точності близько 60 відсотків на тестовому наборі та близько 90 відсотків на навчальному наборі. Втрати при навчанні плавно зменшувалися, а точність навчання плавно зростала в усіх трьох прогонах, з тимчасовим збільшенням збіжності за обома показниками близько 40-го періоду після уповільнення, що почалося приблизно в 12-13-му періодах. Точність на тестовому наборі, навпаки, не збільшилася в жодному з трьох прогонів після епохи 50, епохи, після якої тестова втрата почала неухильно зростати.

Було також більш вираженим у випадку як точності тесту, так і втрат. Описана вище поведінка є класичним прикладом перенавчання моделі на

навчальній вибірці, коли вона втрачає здатність узагальнювати дані, яких не бачила під час навчання.

Особливо показовими в цьому відношенні є графіки втрат при навчанні та тестуванні, оскільки криві чітко розходяться після епохи 50 у всіх трьох прогонах. Зменшення швидкості навчання після епохи 40 помітно покращило характеристики, з точки зору втрат і точності, на тренувальному наборі і тимчасово покращило характеристики на тестовому наборі, що свідчить про ефективність графіка швидкості навчання.

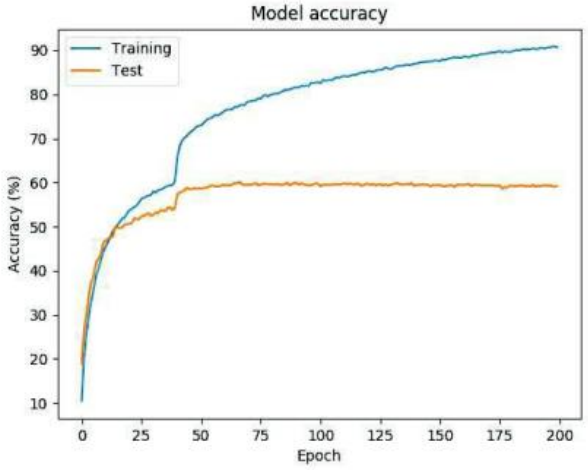

Рисунок 5.3 – Графік зміни точності для другого прогону.

Обґрунтуванням використання такого графіка є те, що він покращує збіжність втрат до мінімуму, і це твердження було підтверджено в даному дослідженні. Небажаним ефектом графіка швидкості навчання могло бути збільшення перенавчання, оскільки розбіжність результатів на тренувальному тесті з набором тестів прискорилася після зменшення швидкості навчання. Останнє спостереження, яке можна було б тут зробити, полягає в тому, що всі три тренувальні сесії були проведені для всіх 200 епох, кількість епох, яка, як видається, була надлишковою або навіть шкідливою для продуктивності моделі з огляду на спостережуване перенастроювання.

Результати з точки зору точності на тестовому наборі досить далекі від результатів найкращих претендентів (сучасний рівень – точність 75,72%)

У цій частині магістерської роботи не ставилося завдання отримати найкращу модель з точки зору точності, метою було зробити якомога кращу модель при заданих обмеженнях. Якщо брати до уваги реалізацію самого дизайну та кодування, що лежить в його основі, то було знайдено кілька способів його покращення, імпровізацій, які за наявності достатнього часу та кращого обладнання можна було б легко реалізувати та протестувати.

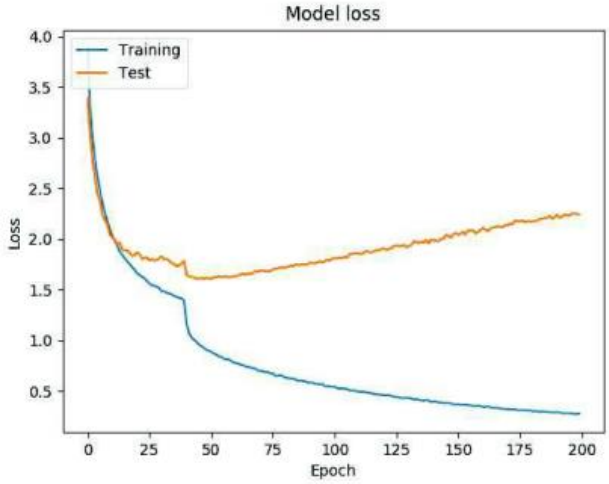

Рисунок 5.4 – Графік втрат для другого прогону.

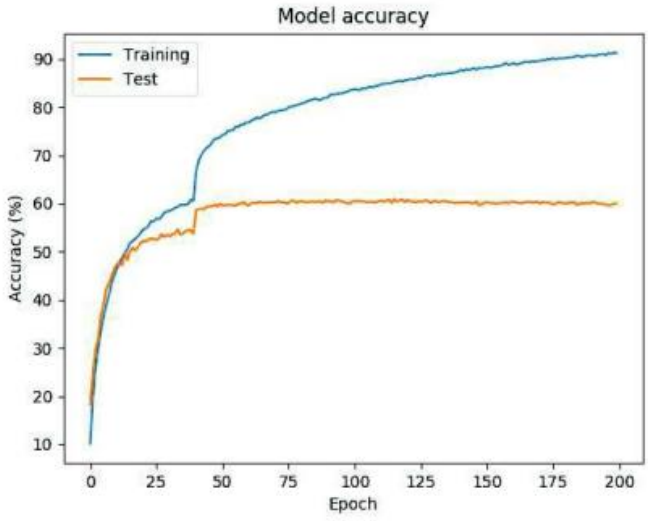

Рисунок 5.5 – Графік точності для третього прогону.

Одним з прикладів недоліків, які можна було б виправити, є те, що

через часові та апаратні обмеження просторова роздільна здатність карт об'єктів на більш пізніх етапах створення мережі могла бути занадто малою.

Іншим покращенням могло б бути використання ранньої зупинки, методу, який зупиняє навчання після певної кількості епох без покращення певної метрики, наприклад, втрат при тестуванні або точності. Інші вдосконалення, які можна було б зробити, включають більш досконалий планувальник швидкості навчання та більш жорстку регуляризацію, як у формі існуючих методів, так і інших. На завершення цього аналізу можна зробити короткий підсумок: було виявлено, що модель досить швидко перенавчається на навчальній вибірці, графік швидкості навчання виявився корисним, а кількість епох, схоже, негативно вплинула на продуктивність моделі.

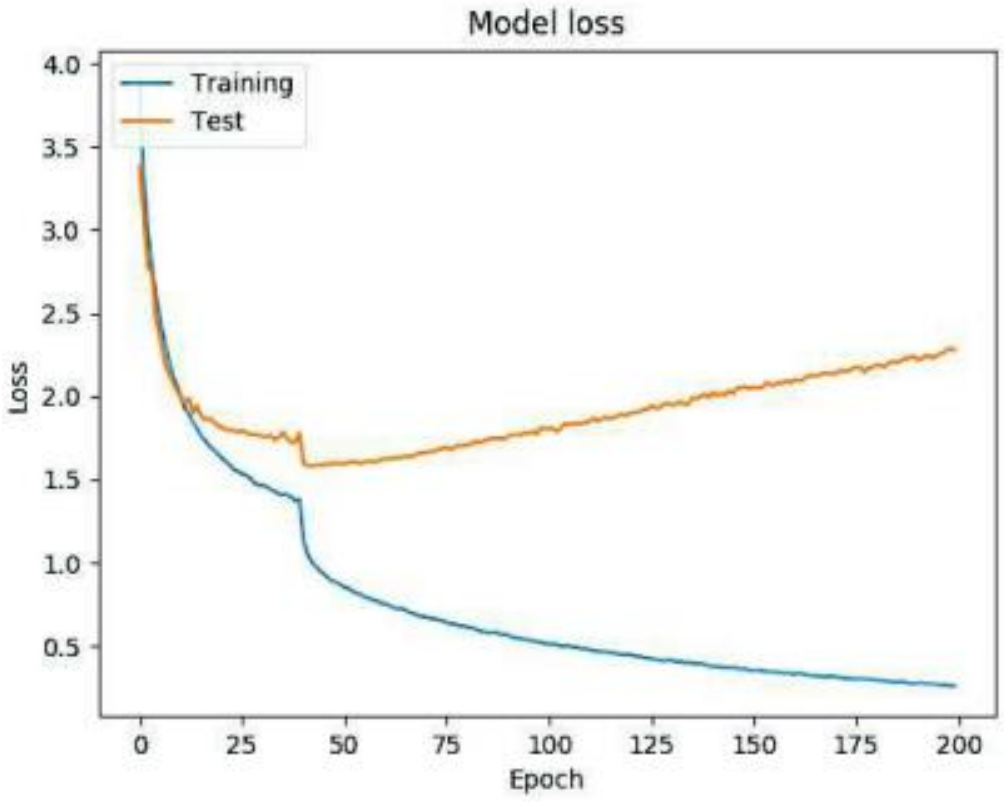

Рисунок 5.6: Графік втрат для третього прогону.

Було виявлено та обговорено ряд можливих покращень: більша

регуляризація, графік швидкості навчання з більшою кількістю кроків, рання зупинка та вища просторова роздільна здатність у пізніх частинах мережі.

ВИСНОВКИ

На момент написання цієї магістерської роботи фреймворки є рівноцінними з точки зору простоти інсталяції.

Що стосується налаштування обчислень на GPU (Graphics Processing Unit), то можна вважати, що фреймворки є рівними і в цьому відношенні, оскільки вони вимагають однакової кількості кроків та залежностей для встановлення.

Що стосується системних вимог та підтримки, то рішення про те, який фреймворк є кращим у цьому відношенні, значною мірою зводиться до того, яку операційну систему та мову програмування використовуватиметься; враховуючи, що CNTK та TensorFlow підтримують мови, які не підтримує інший, а також те, що TensorFlow підтримує «Mac OS X», а CNTK – ні.

Було виявлено, що фреймворки надають еквівалентний набір можливостей та функцій, і вони більш ніж здатні будувати нейронні мережі. Документація обох фреймворків була визнана недостатньою як за якістю, так і за кількістю, однак Keras має невелику перевагу в тому, що його документація зібрана в одному місці, в той час як CNTK має документацію, розподілену по різних сайтах. Було виявлено, що Keras є більш дружнім до новачків і з ним легше працювати, порівняно з CNTK.

Було виявлено, що CNTK не відповідає літературі, маючи натомість власну реалізацію, яка в свою чергу вимагає перенавчання, якщо хтось вивчав літературу, а також вимагає перерахунку параметрів для того, щоб функціонувати відповідно до літератури; Keras, з іншого боку, не вимагає такого переналагодження, що є великим плюсом, на наш погляд.

Було виявлено, що CNTK дає менший час навчання мереж порівняно з Keras з TensorFlow як бекендом, враховуючи невизначеності, та

експериментальну установку. Було виявлено, що графічний процесор настільки перевершує центральний процесор (CPU) за швидкістю навчання, що якщо у когось є графічний процесор, то слід використовувати його замість центрального процесора.

Використання GPU та VRAM (відеопам'яті) було однаковим як в CNTK, так і в Keras з TensorFlow в якості бекенду. Щодо реалізації класифікатора зображень, було виявлено, що модель досить швидко перенавчається на навчальній вибірці, графік швидкості навчання виявився корисним, а кількість епох, схоже, негативно вплинула на продуктивність моделі. Було виявлено та обговорено ряд можливих покращень: більша регуляризація, графік швидкості навчання з більшою кількістю кроків, рання зупинка та вища просторова роздільна здатність в останніх частинах 2 мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

2. Michael A Nielsen. Neural networks and deep learning. 2017. http:// neuralnetworksanddeeplearning.com/.

3. Руденко ОГ, Бодянський ЄВ. Штучні нейронні мережі. Харків: Компанія СМІТ. 2006.

4. Bodyanskiy YV, Rudenko OG. Artificial neural networks: architectures, learning, applications. TELETEX, Kharkov. 2004.

5. Gradient based learning applied to document recognition / Y. LeCun, L. Bottou, Y. Bengio, Y. Haffner. // Proceedings of the IEEE. – 1998. – №86. – С. 2278 – 2324.

6. Krizhevsky A. Imagenet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. E. Hinton. // Communications of the ACM. – 2017. – №60. – С. 84–90.

7. ImageNet. Imagenet. http://image-net.org/, 2017.

8. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

9. Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678, 2016.

10. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

11. Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. arXiv preprint arXiv:1312.4400, 2013.

12. Tom M Mitchell et al. Machine learning. wcb, 1997.

13. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

14. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2818–2826, 2016.

15. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

16. Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In Advances in Neural Information Processing Systems, pages 550–558, 2016.

17. Google. Tensorflow homepage. https://www.tensorflow.org/, 2017.

18. Microsoft. CNTK homepage. https://www.microsoft.com/en-us/research/product/ cognitive-toolkit/, 2017.

19. Theano. Theano documentation. http://deeplearning.net/software/theano/index. html, 31 March, 2017.

20. Google. Distributed tensorflow. https://www.tensorflow.org/deploy/distributed, 8 March, 2017.

21. Microsoft. CNTK CPU-version installation. https://github.com/Microsoft/CNTK/ wiki/Setup-CNTK-on-your-machine, 11 April, 2017.

22. Microsoft. Distributed CNTK. https://github.com/Microsoft/CNTK/wiki/ Multiple-GPUs-and-machines, April, 2017.

23. Microsoft. Azure CNTK. https://github.com/Microsoft/CNTK/wiki/CNTK-on-Azure, March, 2017.

24. Microsoft. Azure. https://azure.microsoft.com/en-us/, 2017.

25. Torch. Torch homepage. http://torch.ch/, n.d.

26. Evan Shelhamer Yangqing Jia. Caffe homepage. http://caffe.berkeleyvision.org/, n.d.

27. Skymind. Deeplearning4j homepage. https://deeplearning4j.org/, 2017.

28. François Chollet. Twitter - the author of keras confirms keras' integration into tensor- flow. https://twitter.com/fchollet/status/820746845068505088, January 15, 2017.

29. Python. Python. https://www.python.org/, 2017.

30. Microsoft. Visual studio. https://www.visualstudio.com/downloads/, 2017.

31. Microsoft. Python tools for visual studio. https://www.visualstudio.com/vs/python/, 2017.