НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий Фізико-технічний інститут
Кафедра математичного моделювання та аналізу даних

До захисту допущено:
Завідувач кафедри
_____ Наталія КУССУЛЬ
«___»_____2022 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Математичні методи моделювання, розпізнавання образів та безпеки даних»
спеціальності 113 «Прикладна математика»
на тему: «Побудова оптимальних моделей розпізнавання та сегментації образів на базі згорткових сіток»

Виконав:
студент IV курсу, групи ФІ-82
Кірсенко Єгор Вікторович _____
Керівник:
доцент, к.ф.-м.н. Орєхов Олександр Арсенійович _____
Рецензент:
к.т.н. Стьопочкіна Ірина Валеріївна _____

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторі без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий Фізико-технічний інститут

Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 113 «Прикладна математика»

Освітня програма «Математичні методи моделювання, розпізнавання образів та безпеки даних»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Наталія КУССУЛЬ

«__» _____ 2022 р.

ЗАВДАННЯ

на дипломну роботу студенту

Кірсенко Єгор Вікторович

1. Тема роботи «Побудова оптимальних моделей розпізнавання та сегментації образів на базі згорткових сіток», керівник роботи доцент, к.ф.-м.н. Орєхов Олександр Арсенійович, затверджені наказом по університету від «__» ____ 2022р. №_____

2. Термін подання студентом роботи _____.

3. Вхідні дані до роботи – набір зображень із опосумами та шиншилами.

4. Зміст роботи – 1. Опис предметної області; 2. Теоретичні основи роботи; 3. Реалізація поставленої задачі і аналіз результатів.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

_____

6. Консультанти розділів роботи

7. Дата видачі завдання 16.02.2022

Календарний план

№ з/п	Назва етапів виконання роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи	22.11.2021	Виконано
2.	Підготовка першого розділу	20.12.2021	Виконано
3.	Підготовка другого розділу	15.01.2022	Виконано
4.	Підготовка даних до роботи	16.02.2022	Виконано
5.	Розробка моделей нейронних мереж	3.03.2022	Виконано
6.	Підготовка третього розділу	12.04.2022	Виконано
7.	Оформлення дипломної роботи	2.05.2022	Виконано
8.	Підготовка презентації доповіді	30.05.2022	Виконано

Студент _____ Єгор КІРСЕНКО

Керівник _____ Олександр ОРЄХОВ

РЕФЕРАТ

Дипломна робота: 57 с., 31 рис.,3 табл., 3 додатки, 16 джерел.

Ключові слова: МАШИННЕ НАВЧАННЯ, ГЛИБОКЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ ТВАРИН.

Об’єкт дослідження – класифікація тварин.

Технології розпізнавання образів широко використовуються в наш час. Використання цих технологій для розпізнавання тварин може значно полегшити роботу в багатьох сферах.

Мета роботи – створити ефективний механізм класифікації схожих між собою тварин. Ключовою особливістю програми є можливість автоматичного визначення тварини по фото без втручання людей. Це можна використовувати в багатьох сферах нагляду за тваринами.

ABSTRACT

Thesis: 57 pages, 31 figures, 3 tables, 3 appendices, 16 sources.

Key words: MACHINE LEARNING, DEEP LEARNING, NEURAL NETWORKS, CONVOLVED NEURAL NETWORKS, CLASSIFICATION OF ANIMALS.

The object of research is the classification of animals.

Image recognition technologies are widely used today. The use of these technologies for animal recognition can greatly facilitate work in many areas.

The purpose of the work is to create an effective mechanism for classifying similar animals. A key feature of the program is the ability to automatically identify the animal in the photo without human intervention. It can be used in many areas of animal supervision.

Зміст

Вступ

ІНС (штучні нейронні мережі) – це математична модель функціонування традиційних для живих організмів нейронних мереж, які є мережею нервових клітин. Як і в біологічному аналогу, в штучних мережах основним елементом виступають нейрони, з'єднані між собою і утворюють шари, число яких може бути різним залежно від складності нейронної мережі та її призначення (розв'язуваних задач).

Мабуть, найпопулярнішим завданням нейронних мереж є розпізнавання візуальних образів. Сьогодні створюються мережі, у яких машини здатні успішно розпізнавати символи на папері та банківських картах, підписи на офіційних документах, детектувати об'єкти тощо. Ці функції дозволяють суттєво полегшити працю людини, а також підвищити надійність та точність різних робочих процесів за рахунок відсутності можливості припущення помилки через людський фактор.

Робота із зображеннями – важлива сфера застосування технологій Deep Learning. Глобально усі зображення з усіх камер світу становлять бібліотеку неструктурованих даних. Задіявши нейронні мережі, машинне навчання та штучний інтелект, ці дані структурують та використовують для виконання різних завдань: побутових, соціальних, професійних та державних, зокрема, забезпечення безпеки.

Розділ 1. Опис предметної області

1.1 Постановка задачі

Треба побудувати оптимальні моделі розпізнавання і сегментації образів на базі згорткових сіток і на прикладі вибраного набору даних перевірити ці моделі.

Для початку треба визначити тип задачі, яку нам необхідно вирішити. Потрібно віднести кожне з зображень до одного з двох класів: опосум або шиншила. А отже перед нами стоїть задача клисифікації.

Класифікація зображень, як випливає з назви, є проблемою введення зображення та виведення опису змісту зображення. Як вхідні дані ми маємо скінченну множину об’єктів, які розділені на класи відомим чином. Ця множина називається вибіркою. У свою чергу ця вибірка розподіляється на 2 множини: тренувальна та тестова вибірки.

Тренувальна вибірка – вибірка, завдяки якій відбувається налаштування (оптимізація) ваг і коефіцієнтів моделі.

Тестова вибірка – вибірка для оцінки якості навчання моделі. Тестова вибірка не повинна залежати від тренувальної.

Необхідно побудувати модель, яка буде здатна класифікувати довільний об’єкт із нашої вибірки. Це в свою чергу означає для кожного об’єкту указати номер або назву класу, до якого відноситься даний об’єкт.

В глибокому навчанні задачі класифікації зображень вирішується за допомогою згорткової нейронної мережі у вигляді навчання з учителем.

1.2 Опис даних

У якості даних для наших моделей ми будемо використовувати невеликий набір картинок: 28 різних картинок опосумів і шиншил, по 14 на кожен клас. Приклад картинки на якій зображено опосума представлено на рис. 1.1, шиншилу – на рис. 1.2.

Рис. 1.1 Опосум

Рис. 1.2 Шиншила

Самі зображення являють собою фото зроблені за допомогою камери. Усі зображення зберігаються у форматах .png, мають нефіксований розмір та три канали R, G, B.

1.3 Актуальність роботи

Задача є актуальною, бо моделі і методи розпізнавання образів широко використовуються у багатьох галузях. Технології розпізнавання образів використовуються в таких важливих сферах, як безпека та спостереження, сканування та створення зображень, маркетинг та реклама, доповнена реальність та пошук зображень. Ці технології здатні значно спростити життя у багатьох сферах людства.

1.4 Висновки до 1-ого розділу

У цьому розділі ми сформулювали постановку задачі, описали необхідні вхідні дані для наших моделей та розглянули актуальність даної роботи.

Розділ 2. Теоретичні основи роботи

2.1 Машинне і Глибоке навчання

Область машинного навчання виникла з питання: чи може комп'ютер вийти за межі того, «що ми й самі знаємо, як виконувати», і самостійно навчитися вирішувати певне завдання? Чи може комп'ютер без допомоги програміста, який визначає правила обробки даних, автоматично визначити ці правила, досліджуючи дані?

У машинному навчанні люди вводять дані та відповіді, що відповідають цим даним, а на виході отримують правила, що зв’язують дані і характерну їм відповідь. Ці правила можна застосувати до нових даних для отримання оригінальних відповідей.

Тобто для машинного навчання нам потрібні три складові:

- Контрольні вхідні дані – наприклад, якщо вирішується завдання класифікації зображень, такими даними можуть бути зображення.
- Приклади очікуваних результатів - у задачі класифікації зображень очікуваним результатом можуть бути теги, такі як «собака», «кішка» та ін.
- Спосіб оцінки якості роботи алгоритму - це необхідно для визначення, наскільки далеко відхиляються результати, що повертаються алгоритмом, від очікуваних. Оцінка використовується як сигнал зворотний зв'язок для коригування роботи алгоритму. Цей етап коригування ми називаємо навчанням.

Модель машинного навчання трансформує вихідні дані на значні результати, «навчаючись» на відомих прикладах вхідних даних, і результатів. Тобто головним

завданням машинного та глибокого навчання є значне перетворення даних для визначення результату.

Глибоке навчання — це особливий розділ машинного навчання: новий підхід до пошуку представлення даних, що наголошує на схемах і методах побудови послідовних шарів (чи рівнів) моделей для вирішення конкретних задач. Іншими відповідними назвами для цієї галузі машинного навчання могли б служити: багатошарове навчання та ієрархічне навчання.

У глибокому навчанні такі багатошарові структури вивчаються (майже завжди) з використанням моделей, які називаються нейронними мережами. Термін нейронна мережа запозичений з нейробіології, проте, хоча деякі основні ідеї глибокого навчання частково запозичені з науки про мозок, моделі глибокого навчання є моделями мозку.[1]

2.2 Нейронні мережі

Нейронні мережі – це один з напрямків наукових досліджень в галузі створення штучного інтелекту (ШІ), в основі якого лежить прагнення імітувати нервову систему людини. В тому числі її (нервової системи) здатність виправляти помилки і самонавчатися. Все це, хоча і дещо грубо повинно дозволити змоделювати роботу людського мозку.

Розвиток інтернету та процеси глобалізації сприяли тому, що з’явилося дуже багато інформації, опрацювати яку самотужки людина фізично не в змозі. Нейронні мережі знайшли застосування у:

- аналізі та класифікуванні даних за заданими параметрами;

- формуванні аналітичних прогнозів, керуючись вхідною інформацією;
- порівнянні та розпізнаванні ідентичних даних.

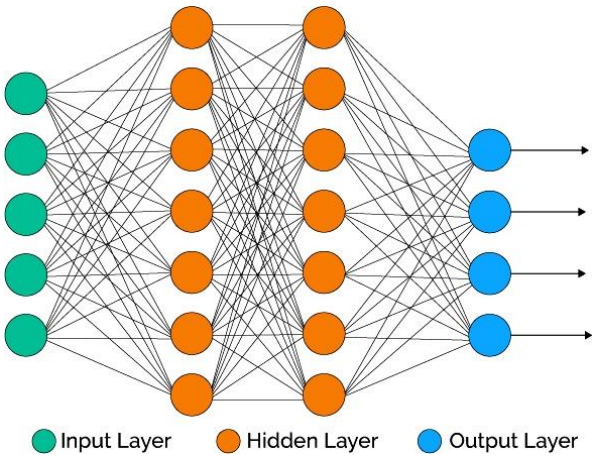

рис. 2.1 Схема нейронної мережі

Суть машинного навчання полягає у перетворенні вхідних даних (наприклад, зображень) на результат (такий, як підпис «кішка»), яке виявляється шляхом дослідження безлічі прикладів вхідних даних та результатів.

Тепер побачимо, як саме відбувається навчання нейронної мережі. Що саме шар нейронної мережі робить зі своїми вхідними даними, визначається його вагами, які фактично є набором чисел. Ваги також іноді називають параметрами шару. У цьому контексті під навчанням мається на увазі пошук набору значень ваг всіх шарів у мережі, при якому мережа правильно відображатиме зразкові вхідні дані у відповідні їм результати. Але ось у чому річ: глибока нейронна мережа може містити десятки мільйонів параметрів. Пошук правильного

значення для кожного з них може виявитися складним завданням, якщо зміна значення одного параметра впливає на поведінку всіх інших!

Щоб чимось керувати, спочатку потрібно отримати можливість спостерігати за цим. Щоб керувати результатом роботи нейронної мережі, необхідно мати можливість виміряти, наскільки цей результат далекий від очікуваного. Це завдання вирішує функція втрат мережі, яку також називають цільовою функцією. Функція втрат приймає передбачення, видане мережею, і справжнє значення (яка мережа повинна була повернути) і обчислює оцінку відстані між ними, що відображає, наскільки добре мережа впоралася з конкретним прикладом. Основна хитрість глибокого навчання полягає у використанні цієї оцінки для коригування значень вагів з метою зменшення втрат у поточному прикладі. Це коригування є завданням оптимізатора, який реалізує так званий алгоритм зворотного поширення помилки: центральний алгоритм глибокого навчання.

Спочатку вагам мережі надаються випадкові значення, тобто фактично мережа реалізує послідовність випадкових перетворень. Звичайно, отриманий нею результат далекий від ідеалу, і оцінка втрат, відповідно, дуже висока. Але з кожним прикладом, що обробляється мережею, ваги коригуються у потрібному напрямку, і оцінка втрат зменшується. Це цикл навчання, який повторюється достатньо разів і породжує вагові значення, що мінімізують функцію втрат.

Мережа з мінімальними втратами, що повертає результати, близькі до справжніх, називається навченою мережею. Процес навчання нейронної мережі наведено на рис. 2.2.

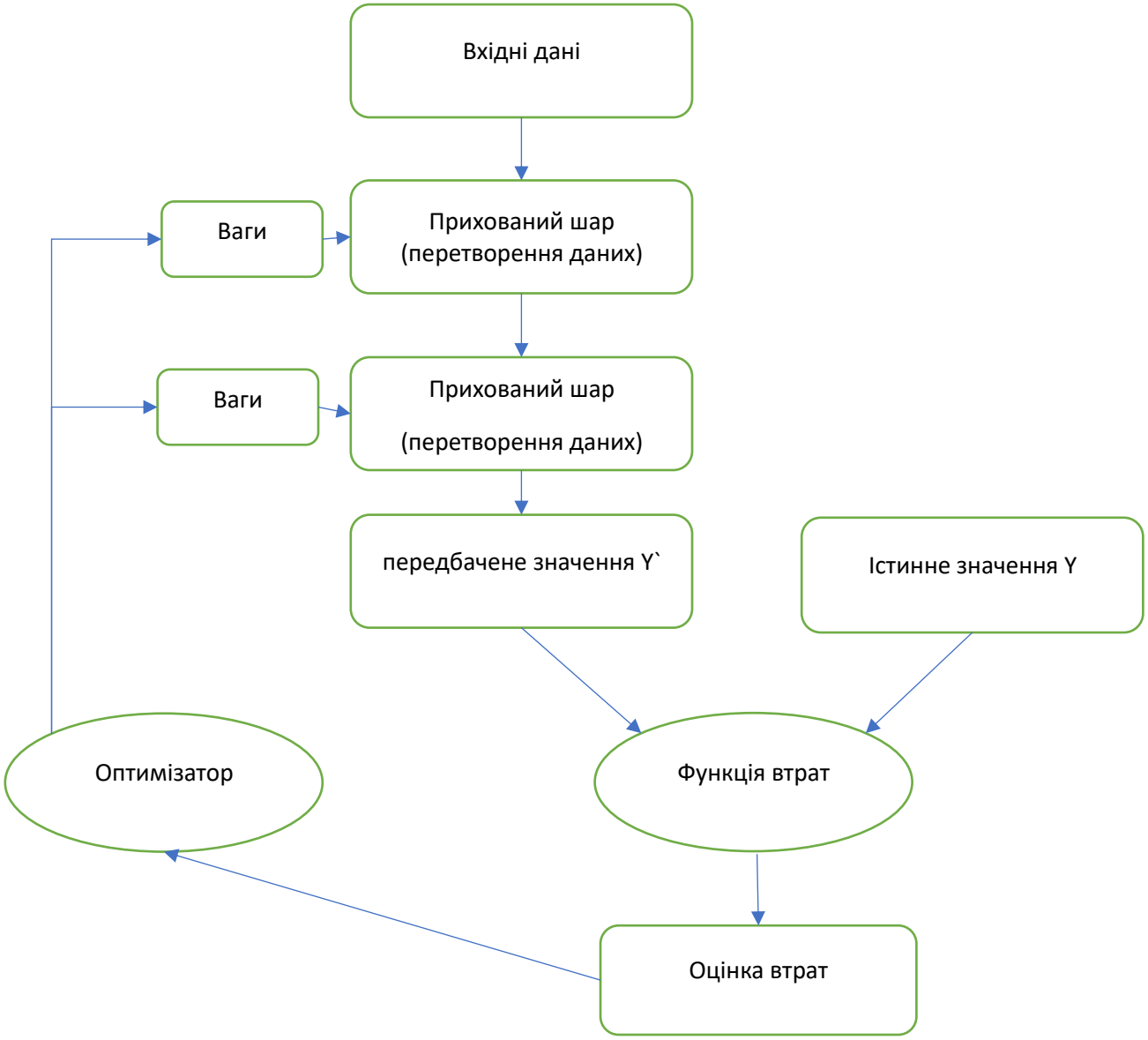

Рис. 2.2 Процес навчання нейронної мережі

2.3 Згорткові нейронні мережі (Convolutional Neural Network, CNN)

Згорткова нейронна мережа (англ. convolutional neural network, CNN) - спеціальна архітектура штучних нейронних мереж, запропонована Яном Лекуном в 1988 році і націлена на ефективне розпізнавання образів, входить до складу

технологій глибокого навчання (англ. deep learning). Використовує деякі особливості зорової кори, у якій були відкриті прості клітини, реагують на прямі лінії під різними кутами, і складні клітини, реакція яких пов'язані з активацією певного набору простих клітин. Таким чином, ідея згорткових нейронних мереж полягає в чергуванні згорткових шарів (англ. convolution layers) та субдискретизуючих шарів (англ. pooling layers). Структура мережі – односпрямована (без зворотних зв'язків).

Назву архітектура мережі отримала через наявність операції згортки, суть якої в тому, що кожен фрагмент зображення множиться на матрицю (ядро) згортки поелементно, а результат сумується і записується в аналогічну позицію вихідного зображення.

Робота згорткової нейронної мережі зазвичай інтерпретується як перехід від конкретних особливостей зображення до більш абстрактних деталей, і далі до ще більш абстрактних деталей аж до виділення понять високого рівня. При цьому мережа самоналаштовується і виробляє найнеобхіднішу ієрархію абстрактних ознак, фільтруючи маловажливі деталі та виділяючи суттєві.[2]

2.4 Архітектура згорткової нейронної мережі

У звичайному перцептроні, який є повнозв'язною нейронною мережею, кожен нейрон пов'язаний з усіма нейронами попереднього шару, причому кожен зв'язок має свій персональний ваговий коефіцієнт. У згортковій нейронній мережі в операції згортки використовується лише обмежена матриця ваг невеликого розміру, яку «рухають» по всьому шару (на самому початку - безпосередньо по вхідному зображенню), формуючи після кожного зсуву сигнал активації для нейрона наступного шару з аналогічною позицією. Тобто для різних нейронів

вихідного шару використовуються та сама матриця ваг, яку також називають ядром згортки. Її інтерпретують як графічне кодування будь-якої ознаки, наприклад, наявність похилої лінії під певним кутом. Тоді наступний шар, що вийшов у результаті операції згортки такою матрицею ваг, показує наявність даної ознаки в шарі, що обробляється, і її координати, формуючи так звану карту ознак (англ. feature map). Природно, у згортковій нейронній мережі наборів ваг набагато більше одного. У цьому такі ядра згортки не закладаються дослідником заздалегідь, а формуються самостійно шляхом навчання мережі класичним шляхом зворотного поширення помилки. Прохід кожним набором вагів формує свій власний екземпляр карти ознак, роблячи нейронну мережу багатоканальної (багато незалежних карт ознак на одному шарі). Також слід зазначити, що при переборі шару матрицею вагів її пересувають зазвичай не на повний крок (розмір цієї матриці), а на невелику відстань. Так, наприклад, при розмірності матриці ваги 5×5 її зсувають на один або два нейрони (пікселі) замість п'яти, щоб не «переступити» шуканий ознака.

Операція субдискретизації (англ. subsampling, англ. pooling, також перекладається як «операція підвиборки» або операція об'єднання), виконує зменшення розмірності сформованих карт ознак. У даній архітектурі мережі вважається, що інформація про факт наявності шуканої ознаки важливіша за точне знання його координат, тому з кількох сусідніх нейронів карти ознак вибирається максимальна і приймається за один нейрон ущільненої карти ознак меншої розмірності. За рахунок цієї операції, крім прискорення подальших обчислень, мережа стає інваріантнішою до масштабу вхідного зображення.[3]

Розглянемо типову структуру згорткової нейронної мережі докладніше. Мережа складається з великої кількості шарів. Після початкового шару (вхідного зображення) сигнал проходить серію згорткових шарів, у яких чергується власне згортка та субдискретизація (пулінг). На кожному наступному шарі картка

зменшується у розмірі, але збільшується кількість каналів. Зазвичай після проходження кількох шарів карта ознак вироджується у вектор або навіть скаляр, але таких карт ознак стають сотні. На виході згорткових шарів мережі додатково встановлюють кілька шарів пов'язкової нейронної мережі (перцептрон), на вхід якому подаються кінцеві карти ознак.[4]

Типова архітектура згорткової нейронної мережі зображена на рис. 2.

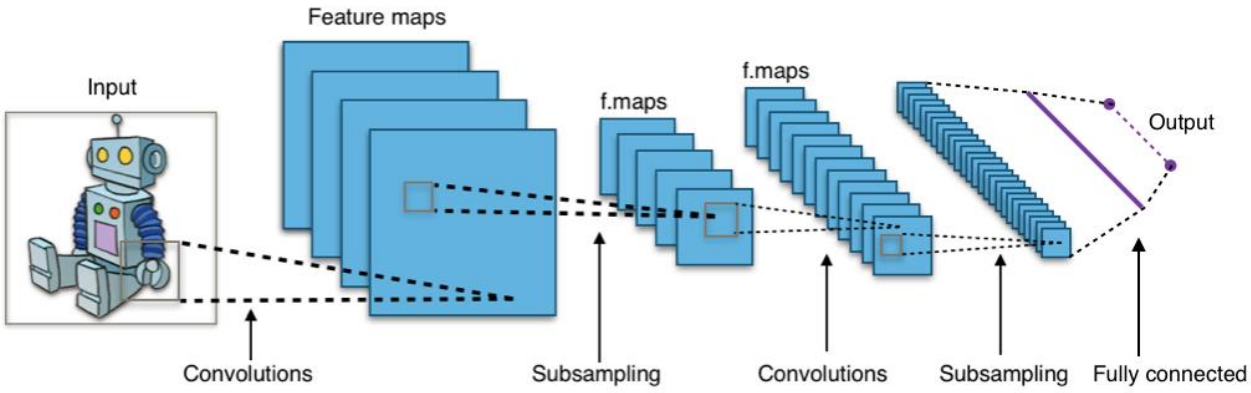

Рис. 2.3 Архітектура згорткової нейронної мережі

Тепер розглянемо деякі переваги і недоліки CNN:

Переваги:

1. Один з кращих алгоритмів розпізнавання та класифікації зображень.
2. У порівнянні з повнозв'язною нейронною мережею (типу перцептрону) - набагато менша кількість ваг, що настроюються, так як одне ядро ваг використовується цілком для всього зображення, замість того, щоб

робити для кожного пікселя вхідного зображення свої персональні вагові коефіцієнти.

3. Зручне розпаралелювання обчислень, а отже, можливість реалізації алгоритмів роботи та навчання мережі на графічних процесорах.

4. Навчання за допомогою класичного методу зворотного розповсюдження помилки.

Недоліки:

1. Занадто багато параметрів мережі, що варіюються: функції активації, кількість шарів, оптимізатор, функція втрат та ін.

2. Занадто багато потрібно часу для навчання моделі.

3. Необхідність у великій кількості прикладів для навчання.

2.5 Види шарів

Розглянемо деякі види шарів згорткових нейронних мереж, їх принципи роботи і основні властивості.

2.5.1 Convolution

Шар згортки (англ. convolutional layer) – це основний блок згорткової нейронної мережі. Шар згортки включає в себе для кожного каналу свій фільтр, ядро згортки якого обробляє попередній шар фрагментами (підсумовуючи результати поелементного твору для кожного фрагмента). Вагові коефіцієнти ядра згортки (невеликої матриці) невідомі та встановлюються у процесі навчання.

Особливістю згорткового шару є порівняно невелика кількість параметрів, що встановлюється під час навчання. Так наприклад, якщо вихідне зображення має розмірність 100×100 пікселів по трьох каналах (це означає 30 000 вхідних нейронів), а згортковий шар використовує фільтри з ядром 3×3 пікселя з виходом на 6 каналів, тоді в процесі навчання визначається лише 9 ваг ядра , проте по всіх поєднаннях каналів, тобто 9×3×6=162, у разі даний шар вимагає знаходження лише 162 параметрів, що значно менше кількості шуканих параметрів пов'язкової нейронної мережі. На рис. 2.4 зображено приклад роботи шару згортки.

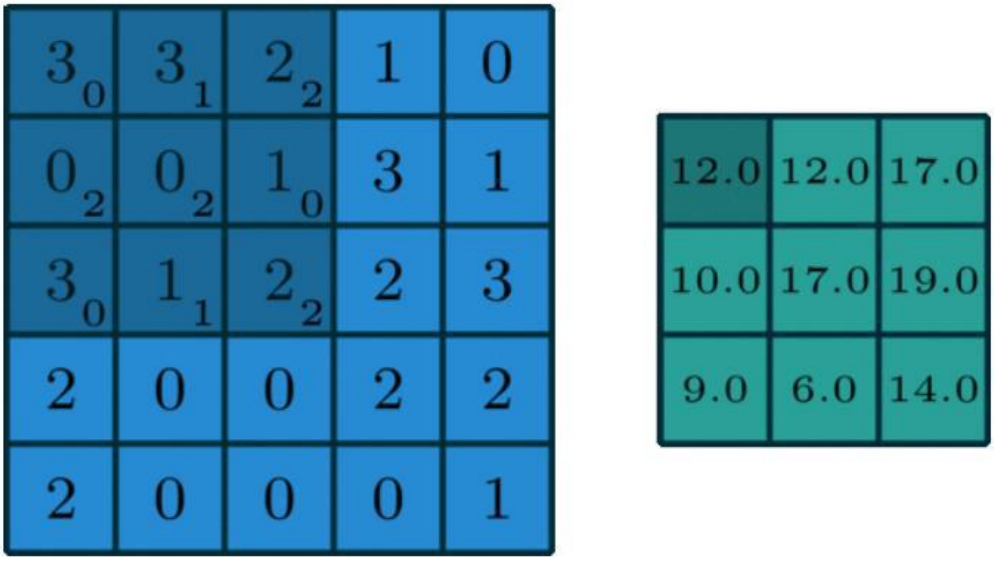

Рис. 2.4 Шар згортки

2.5.2 Шар активації

Скалярний результат кожної згортки потрапляє на функцію активації, яка є якоюсь нелінійною функцією. Шар активації зазвичай поєднують із шаром згортки (вважають, що функція активації вбудована у шар згортки). Функція нелінійності може бути будь-яким на вибір дослідника, традиційно для цього

використовували функції типу гіперболічного тангенсу(рис. 2.5а) або сигмоїди(рис. 2.5б). Однак у 2000-х роках було запропоновано та досліджено нову функцію активації — ReLU (скорочення від англ. rectified linear unit), яка дозволила суттєво прискорити процес навчання та одночасно спростити обчислення (за рахунок простоти самої функції). Тобто, по суті, це операція відсікання негативної частини скалярної величини. Станом на 2017 рік ця функція та її модифікації (Noisy ReLU, Leaky ReLU та інші) є функціями активації, що найчастіше використовуються, у глибоких нейромережах, зокрема, у згорткових. Ця функція активації зображена на рис. 2.6.

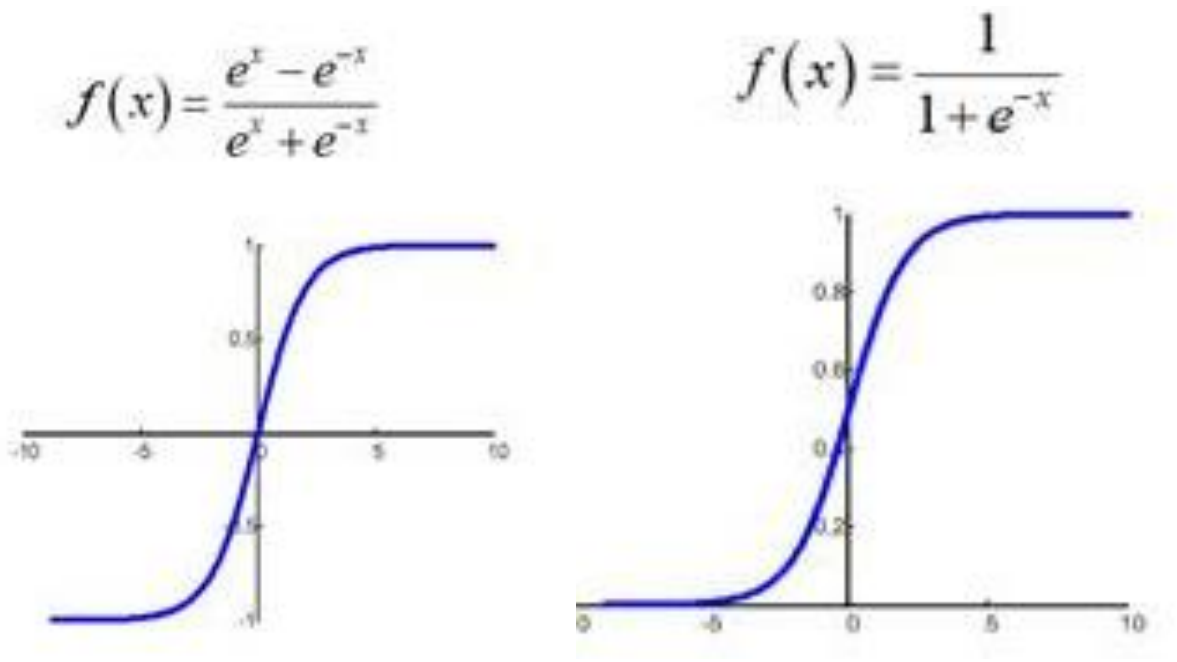

Рис. 2.5: а - функції типу гіперболічного тангенсу, б - функції типу сигмоїди

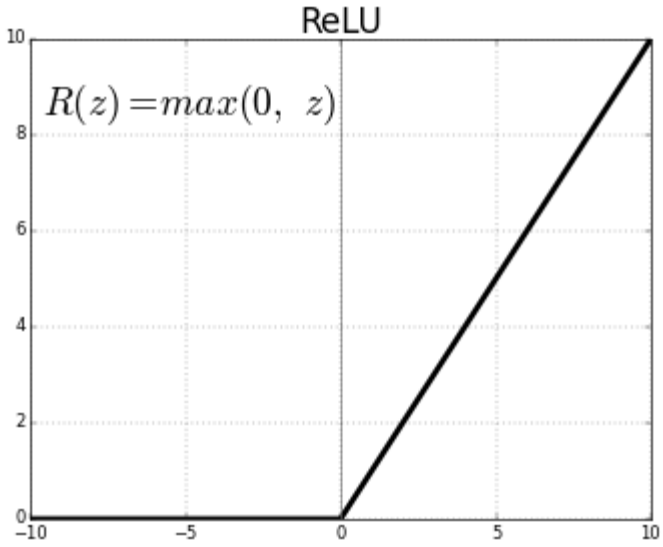

Рис. 2.6 Функція активації ReLU

Також слід виділити функцію активації Softmax. Вона перетворює вектор z розмірності k у вектор σ тієї ж розмірності, де кожна координата σ_i отриманого вектора представлена дійсним числом в інтервалі [0,1] та сума координат дорівнює 1.

Координати σ_i обчислюються таким чином:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_i}}$$

2.5.3 Pooling

Шар пулінгу (інакше субдискретизації) є нелінійним ущільненням карти ознак, при цьому група пікселів (зазвичай розміру 2×2) ущільнюється до одного пікселя, проходячи нелінійне перетворення. Найбільш уживана функція максимуму. Перетворення зачіпають прямокутники або квадрати, що не

перетинаються, кожен з яких утискається в один піксель, при цьому вибирається піксель, що має максимальне значення. Операція пулінгу дозволяє суттєво зменшити просторовий обсяг зображення. Пулінг інтерпретується так: якщо на попередній операції згортки вже були виявлені деякі ознаки, то для подальшої обробки настільки докладне зображення вже не потрібне, і воно ущільнюється до менш детального. До того ж, фільтрація вже непотрібних деталей допомагає не перевчитися. Шар пулінгу, як правило, вставляється після шару згортки перед шаром наступної згортки. На рис. 2.7 зображено приклад роботи шару MaxPool.

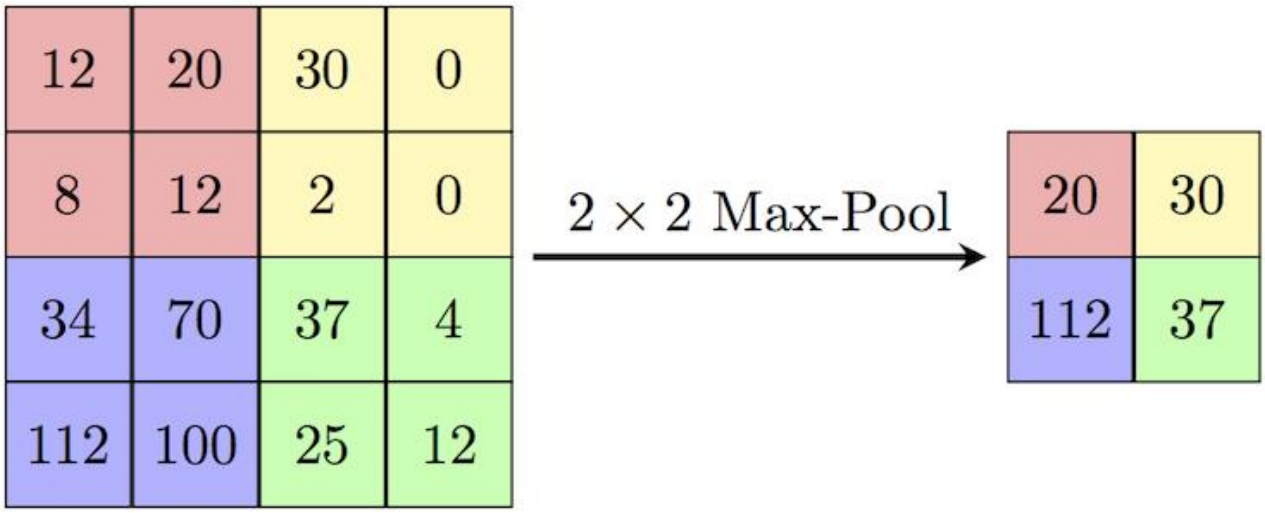

Рис. 2.7 Шар MaxPool

Крім пулінгу з функцією максимуму можна використовувати й інші функції, наприклад, середнього значення або L2-нормування. Однак практика показала переваги саме пулінгу з функцією максимуму, що включається до типових систем.

2.5.4 Fully connected neural network

Після кількох проходжень згортки зображення та ущільнення за допомогою пулінгу система перебудовується від конкретної сітки пікселів з високою

роздільною здатністю до більш абстрактних карт ознак, як правило, на кожному наступному шарі збільшується кількість каналів і зменшується розмірність зображення в кожному каналі. Зрештою, залишається великий набір каналів, що зберігають невелику кількість даних (навіть один параметр), які інтерпретуються як абстрактні поняття, виявлені з вихідного зображення.

Ці дані об'єднуються і передаються на звичайну повнозв'язну нейронну мережу (рис. 2.8), яка також може складатися з кількох шарів. При цьому повнозв'язні шари вже втрачають просторову структуру пікселів і мають порівняно невелику розмірність (стосовно кількості пікселів вихідного зображення).

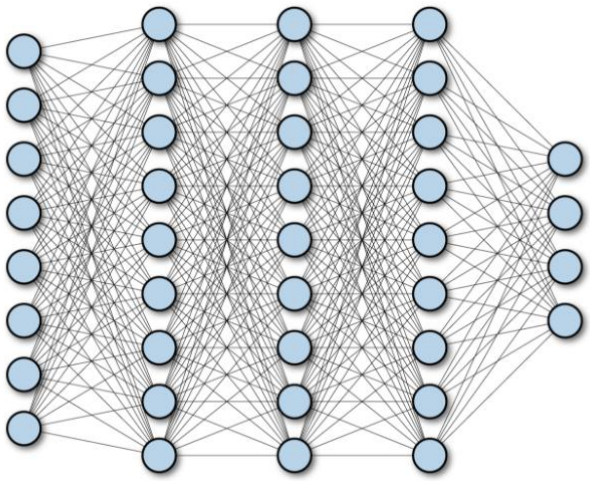

Рис. 2.8 Звичайну повнозв'язна нейронна мережа

2.5.5 Dropout layer

Дропаут (від англ. dropout) - метод регуляризації штучних нейронних мереж, призначений для зменшення перенавчання мережі.

Термін "dropout" (вибивання, викидання) характеризує виключення певного відсотка (наприклад 30%) випадкових нейронів (які знаходяться як у прихованих, так і видимих шарах) на різних ітераціях (епохах) під час навчання нейронної мережі. Це дуже ефективний спосіб усереднення моделей усередині нейронної мережі. В результаті найбільш навчені нейрони отримують у мережі більшу вагу. Такий прийом значно збільшує швидкість навчання, якість навчання на тренувальних даних, а також підвищує якість передбачень моделі на нових тестових даних. На рис. 2.9 зображено як працює цей метод.

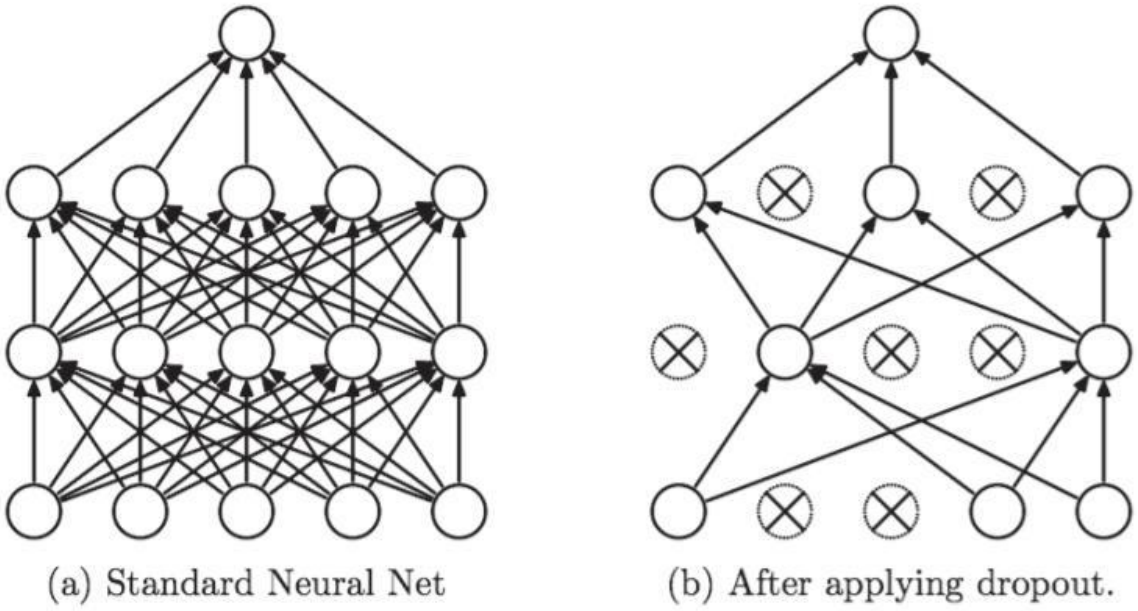

Рис. 2.9 Метод Dropout

2.6 Попередня обробка даних

Будь-яке зображення складається із пікселів. Під кожним пікселем розуміють трійку чисел (R,G,B), кожне з яких є цілим числом на інтервалі [0;255] і позначає інтенсивність червоного, зеленого або синього кольору. Отже, будь-яке

зображення можна подати у вигляді тривимірної матриці розміром (3,Height,Weight).

2.6.1 Нормалізація

Нормалізація — це процес перетворення числової ознаки в стандартний діапазон значень. Діапазон значень може бути [-1, 1] або [0, 1]. Хоча нормалізація не є обов'язковою, але це може допомогти вам двома способами, а саме:

Нормалізація даних збільшить швидкість навчання. Це збільшить швидкість як під час навчання моделі, так і під час тестування даних.

Це дозволить уникнути переповнення чисел. Насправді це означає, що нормалізація гарантує, що наші вхідні дані будуть у відносно невеликому діапазоні. Це дозволить уникнути проблем, оскільки комп'ютери зазвичай мають проблеми з дуже малими чи дуже великими числами.[17]

Одним із найпоширеніших методів нормалізації є maxmin-нормалізація. Метод полягає в тому, що значення інтенсивності кожного кольору для кожного пікселю нормується на діапазон [0,1] шляхом використання формули:

$$x_{new} = \frac{x - 0}{255 - 0} = \frac{x}{255}$$

2.6.2 Аугментація

При глибокому навчанні іноді можна зіткнутися із ситуацією, коли набір даних має обмежений розмір. Але щоб отримати кращі результати узагальнення моделі, необхідно мати більше даних, у тому числі різні їх варіації. Тобто

необхідно збільшити розмір вихідного набору штучно, і це можна зробити за допомогою аугментації даних.

Аугментація даних (англ. data augmentation) - це методика створення додаткових даних із наявних даних.[16]

Найчастіше проблема обмеженого набору даних виникає при вирішенні завдань, пов'язаних з обробкою зображень. Наступні способи аугментації зображень є найпопулярнішими:

- Відображення по вертикалі або горизонталі (англ. flipping);
- Повертання зображення на певний кут (англ. rotation);
- Створення відступу (англ. Padding);
- Вирізання частини зображення (англ. cropping);
- Додавання шуму (англ. adding noise);
- Маніпуляції із кольором (англ. color jittering).

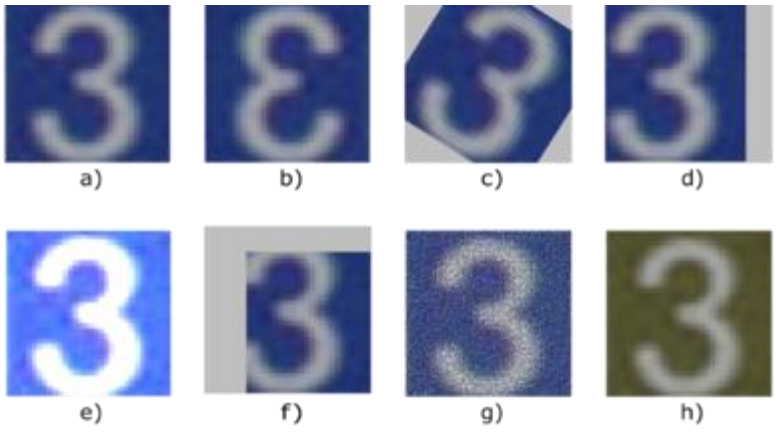

Рис. 2.10: a) Вихідне зображення. b) Відображення по горизонталі. c) Повертання зображення на певний кут. d) Створення відступу. e) Збільшення

яскравості та контрастності. f) Вирізання частини зображення. g) Додавання шуму. h) Зміна каналів RGB.

Також можна використовувати різні комбінації, наприклад, вирізати частину зображення, повернути його і змінити колір фону.

2.7 Відомі архітектури CNN

2.7.1 LeNet

LeNet - це структура згорткової нейронної мережі, запропонована Янном ЛеКуном у 1998 році. Загалом, LeNet відноситься до lenet-5 і являє собою просту згорткову нейронну мережу. Її архітектура наведена на рис. 2.11.[12][13]

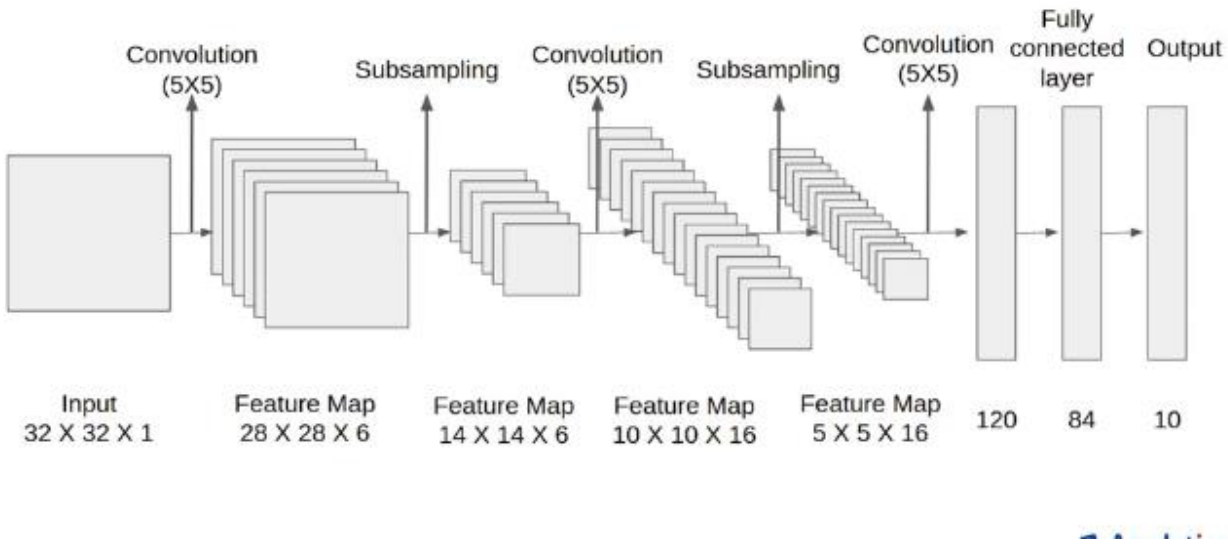

Рис. 2.11 Архітектура Lenet

Основні характеристики LeNet:

- Кожен згортковий шар включає три частини: згортку, об'єднання та нелінійні функції активації
- Рівень об'єднання підвиборів середнього значення
- tanh функція активації
- Використання MLP як останній класифікатор
- Порівняно невелика кількість параметрів мережі

2.7.2 AlexNet

AlexNet — згорткова нейронна мережа, яка вплинула на розвиток машинного навчання, особливо — на алгоритми комп'ютерного зору. Мережа з великим відривом виграла конкурс із розпізнавання зображень ImageNet LSVRC-2012 у 2012 році (з кількістю помилок 15,3% проти 26,2% у другого місця).

Архітектура AlexNet схожа із створеною Yann LeCum мережею LeNet. Однак у AlexNet більше фільтрів на шарі та вкладених згорткових шарів. Мережа включає згортки, максимальне об'єднання, дропаут, аугментацію даних, функції активацій ReLU та стохастичний градієнтний спуск (рис**.** 2.12).[6][7]

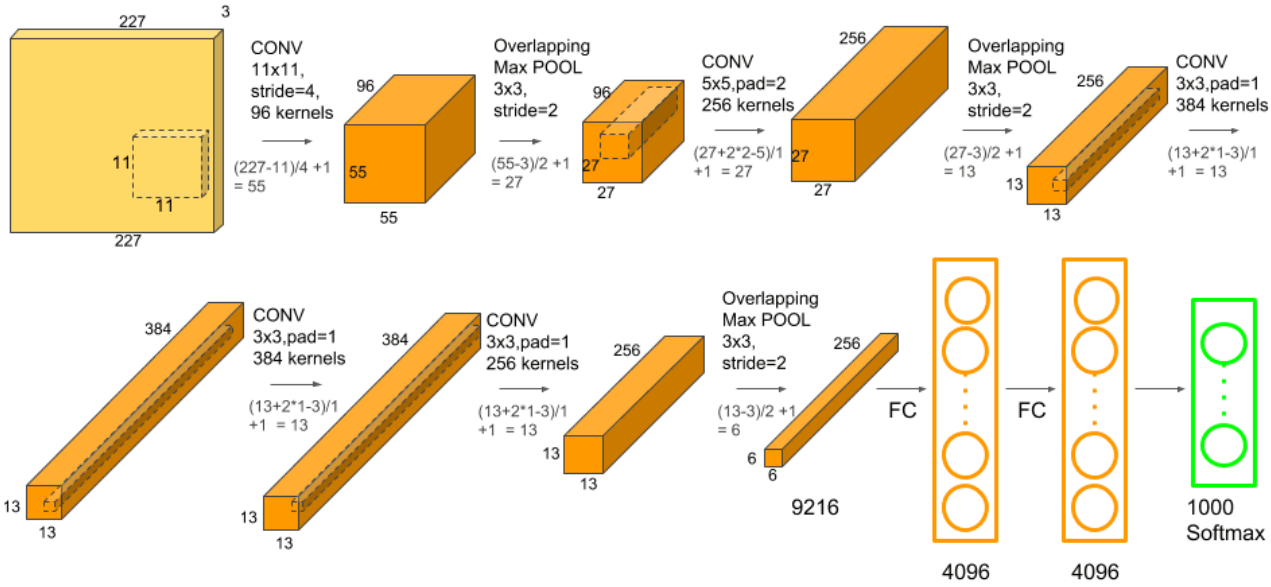

Рис. 2.12 Архітектура Alexnet

Основні характеристики AlexNet:

- Як функція активації використовується Relu замість арктангенсу для додавання моделі нелінійності.
- Використання дропауту вирішує проблему перенавчання.
- Мережа містить 62,3 мільйона параметрів.

2.7.3 VGG

VGG16 – модель згорткової нейронної мережі, запропонована K. Simonyan та A. Zisserman з Оксфордського університету у статті “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Модель досягає точності 92.7% - топ-5, при тестуванні на ImageNet у задачі розпізнавання об'єктів на зображенні.

Цей датасет складається з більш ніж 14 мільйонів зображень, що належать до 1000 класів.

VGG16 – одна з найзнаменитіших моделей, відправлених на змагання ILSVRC-2014. Вона є покращеною версією AlexNet, в якій замінені великі фільтри (розміру 11 і 5 в першому і другому шарі, відповідно) на кілька фільтрів розміру 3x3, які йдуть один за одним. Архітектура даної мережі наведена на рис. 2.13.[10][11]

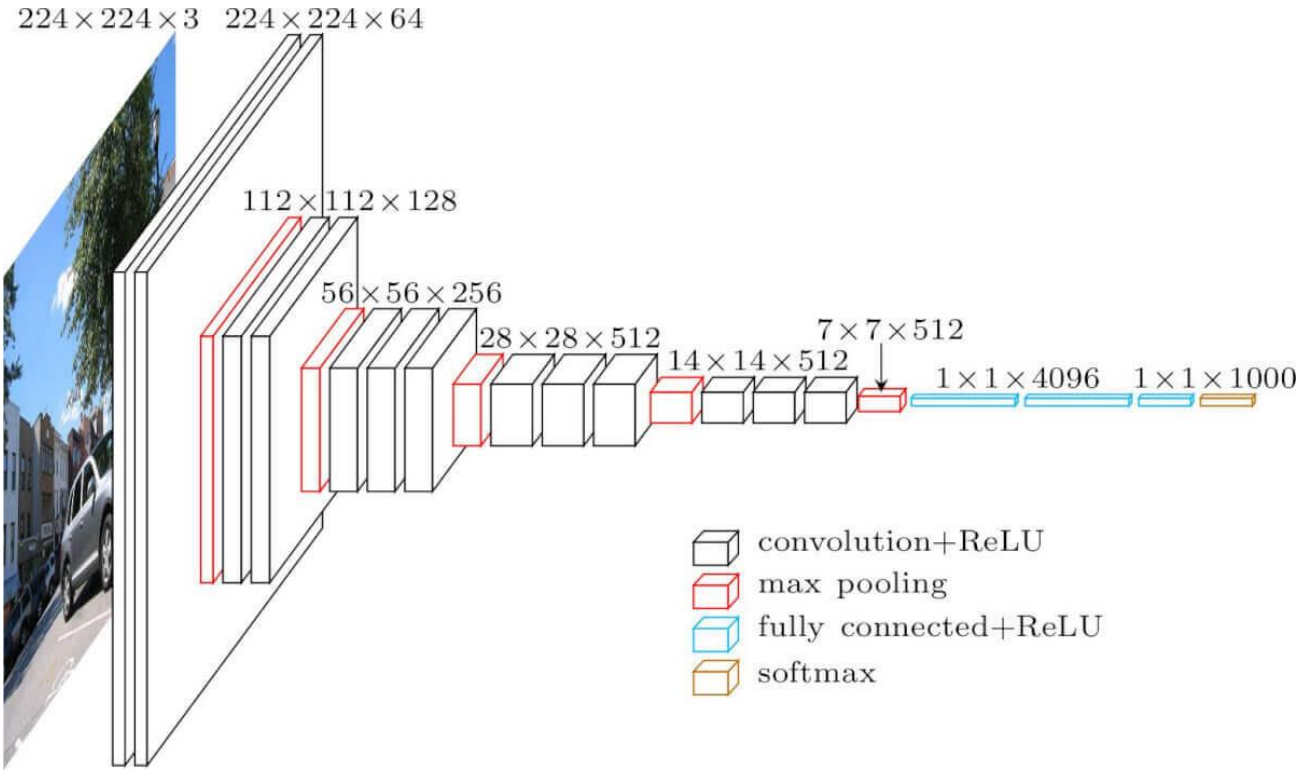

Рис. 2.13 Архітектура VGG

Основні характеристики VGG:

- Використання ядер згортки невеликого розміру (3x3, на відміну великих ядер розміру 7x7 чи 11x11).
- Дуже повільна швидкість навчання.
- Мережа містить 138 мільйона параметрів.

2.7.4 MobileNet

MobileNet є ефективною архітектурою згорткової нейронної мережі, яка зменшує кількість використовуваної пам'яті для обчислень, зберігаючи при цьому високу точність передбачень. Саме тому MobileNet є ідеальним варіантом для використання на мобільних пристроях з обмеженою кількістю пам'яті та обчислювальних ресурсів. Архітектура даної мережі наведена на рис. 2.14.[14]

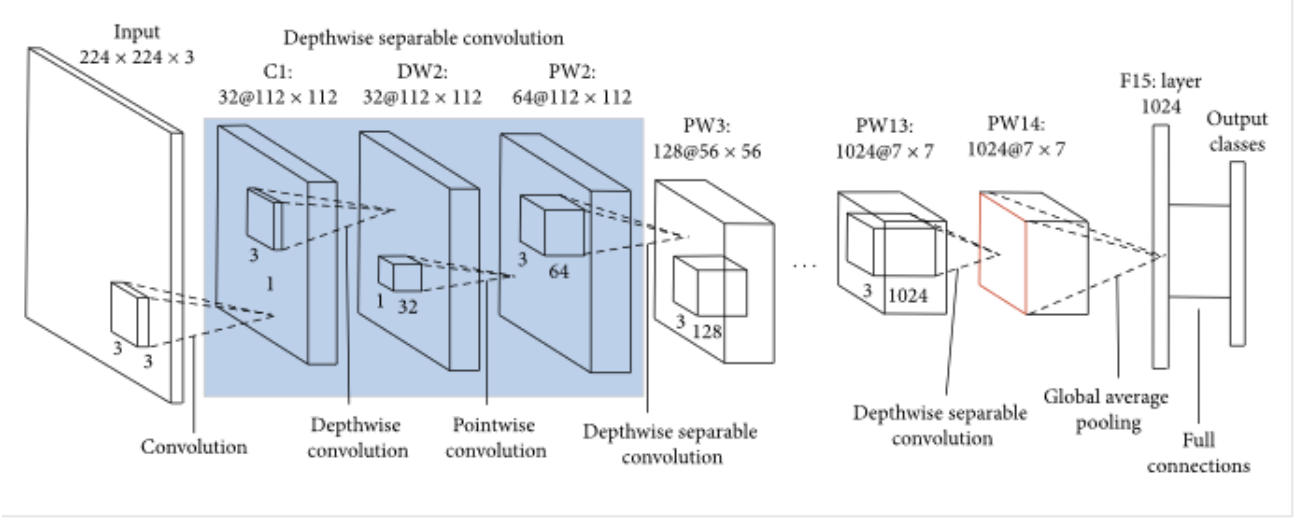

Рис. 2.14 Архітектура Mobilenet

Основні характеристики MobileNet:

- відсутність max pooling-шарів. Замість них зниження просторової розмірності використовується згортка з параметром stride, рівним 2.
- Мережа містить 3,5 мільйона параметрів.
- Використання Global average pooling (рис. 2.16).

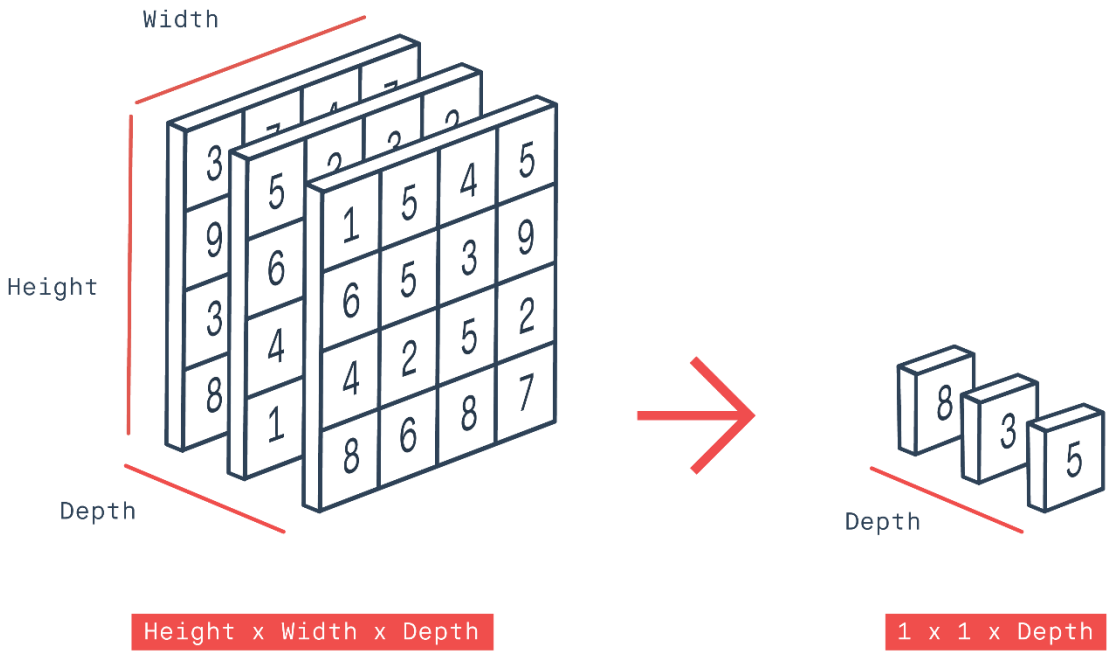

Рис. 2.15 Шар Global average pooling

2.8 Висновки до 2-ого розділу

У цьому розділі ми розглянули необхідні теоретичні основи обраної теми, а саме машинне і глибоке навчання, архітектура і принципи роботи згорткових нейронних мереж. Саме таке теоретичне підґрунтя дозволить нам вирішити поставлену задачу.

Розділ 3. Реалізація поставленої задачі і аналіз результатів

3.1 Опис інструментів для реалізації поставленої задачі

TensorFlow – відкрита програмна бібліотека для машинного навчання, розроблена компанією Google для вирішення завдань побудови та тренування нейронної мережі з метою автоматичного знаходження та класифікації образів, досягаючи якості людського сприйняття.

Keras - відкрита бібліотека, написана мовою Python і забезпечує взаємодію зі штучними нейронними мережами. Вона є надбудовою над фреймворком TensorFlow. Націлена на оперативну роботу з мережами глибинного навчання, при цьому спроектована так, щоб бути компактною, модульною та розширюваною. Її основним автором та підтримуючим є Франсуа Шолле (фр. François Chollet), інженер Google.

Планувалося, що Google підтримуватиме Keras в основній бібліотеці TensorFlow, проте Шолле виділив Keras в окрему надбудову, оскільки згідно з концепцією Keras є скоріше інтерфейсом, ніж наскрізною системою машинного навчання. Keras надає високорівневий, більш інтуїтивний набір абстракцій, який робить простим формування нейронних мереж, незалежно від бібліотеки наукових обчислень, що використовується як обчислювальний бекенд.

Ця бібліотека містить численні реалізації широко застосовуваних блоків нейронних мереж, таких як шари, цільові та передавальні функції, оптимізатори, та безліч інструментів для спрощення роботи із зображеннями та текстом.

3.2 Обгрунтування вибраних моделей для реалізації поставленої задачі

Для розв’язання поставленої задачі класифікації були вибрані представники сімейства моделей Lenet, Alexnet та MobileNet. Ці архітектури не тільки мають різні схеми побудови, а також принципово різну кількість параметрів, що в деякому сенсі говорить про структуру і швидкість навчання моделей.

Також, оскільки оптимізатор і функція втрат не входить до складу архітектур моделей CNN, ми проведемо порівняльний аналіз результатів цих моделей для різних пар оптимізатора і функції втрат. Для цієї роботи були обрані такі популярні оптимізатори, як Adam, SGD та RMSprop, а багатьох функцій втрат були обрані BinaryCrossentropy та SparseCategoricalCrossentropy.

3.3 Попередня обробка даних

Структура каталогу з даними представлена на рис. 3.1.

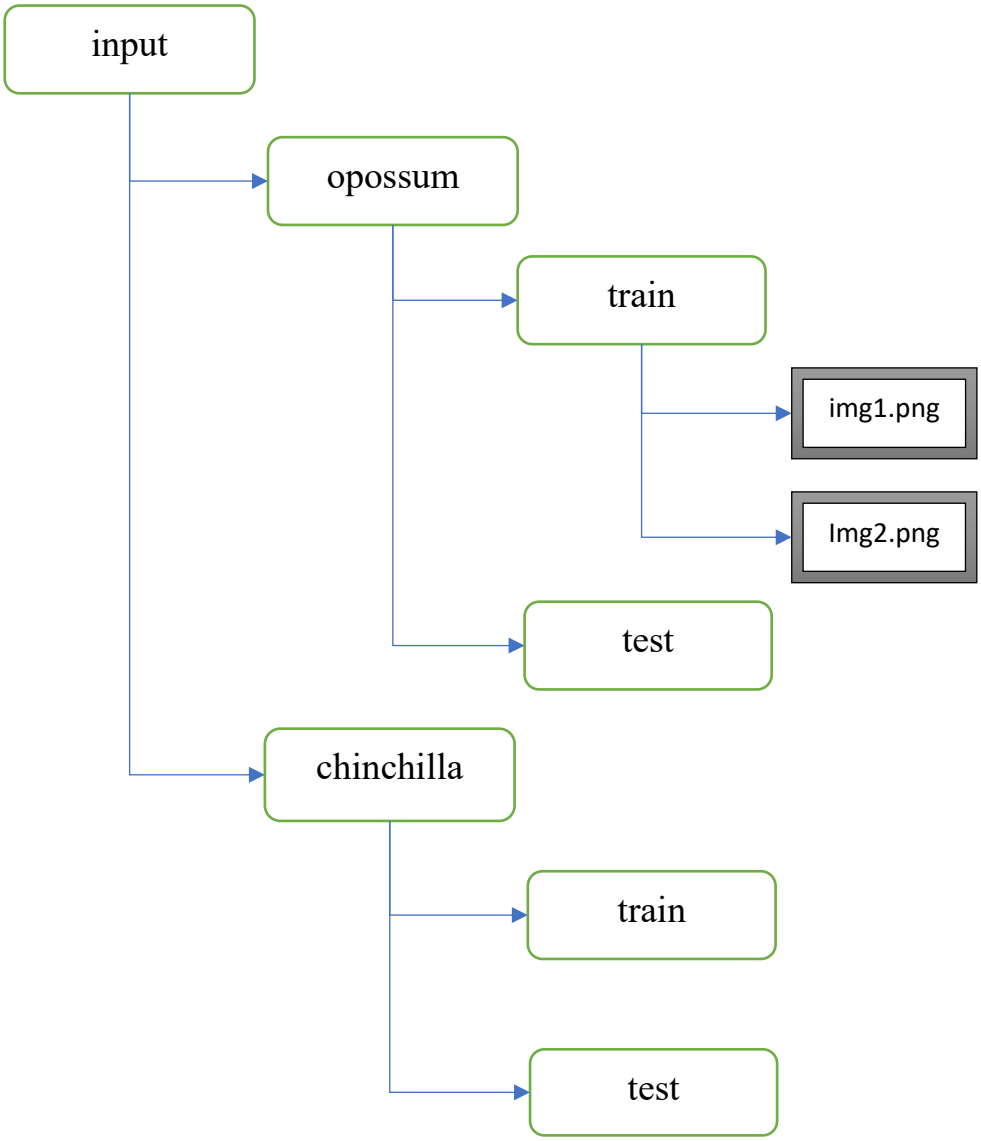

рис. 3.1 Структура каталогу з даними

Після зчитування даних необхідно привести наші картинки до конкретного розміру в залежності від обраної моделі:

- Базова модель – (150, 150);
- AlexNet – (227, 227);
- MobileNet – (224, 224).

Для покращення наших моделей ми будемо використовувати механізм генерування нових картинок за допомогою певних маніпуляцій зі старими. У python цей механізм реалізовано за допомогою об’єкта ImageDataGenerator(). Приклад роботи цього механізму представлено на рис. 3.2.

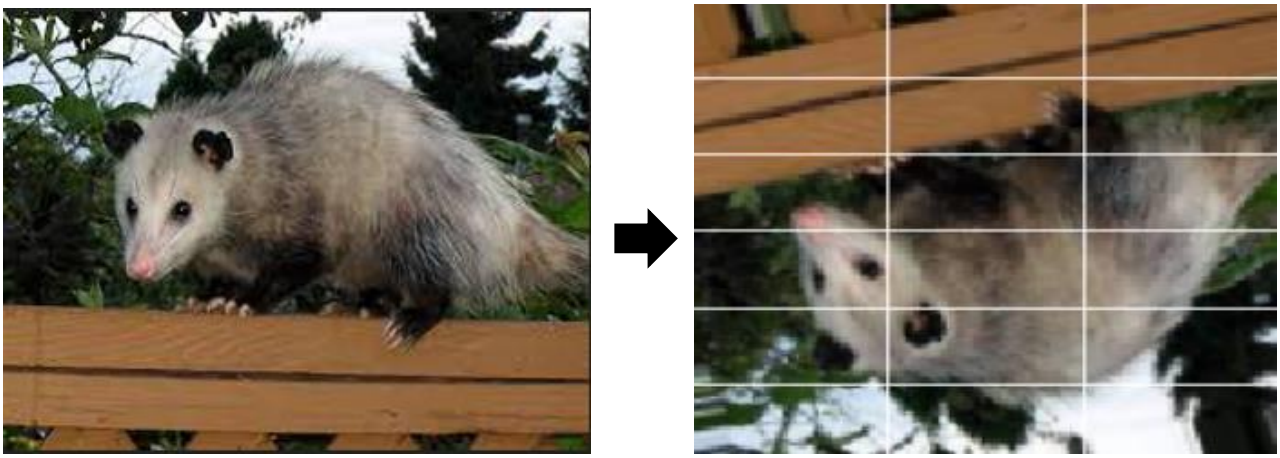

Рис. 3.2: a – базове зображення, b – нове згенероване зображення

3.4 Реалізація моделей і аналіз результатів

3.4.1 AlexNet: Порівняльний аналіз результатів в залежності від оптимізатора і функції втрат

Loss Optimizator	Adam	SGD	RMSprop
BinaryCrossentropy	Train: acc = 0.85 loss =14.7915 Test: acc = 1 loss = 3.5e-23	Train: acc = 0.95 loss = 0.1124 Test: acc = 1 loss = 0.1247	Train: acc = 0.8 loss = 16.7269 Test: acc = 1 loss = 6.1e-22

SparseCategorical Crossentropy	Train: acc = 0.8 loss = 33.7457 Test: acc = 0.75 loss = 19.3108	Train: acc = 0.95 loss = 0.3229 Test: acc = 1 loss = 0.0012	Train: acc = 0.9 loss = 5.9453 Test: acc = 0.8750 loss = 0.1969

Табл. 3.1

Кількість параметрів моделі становить 41,513,730. Програмний код наведено в додатку А.

Для функція втрат BinaryCrossentropy найкраще себе показав оптимізатор SGD. Графік точності моделі і значення функції втрат на тренувальній вибірці зображено на рис. 3.3.

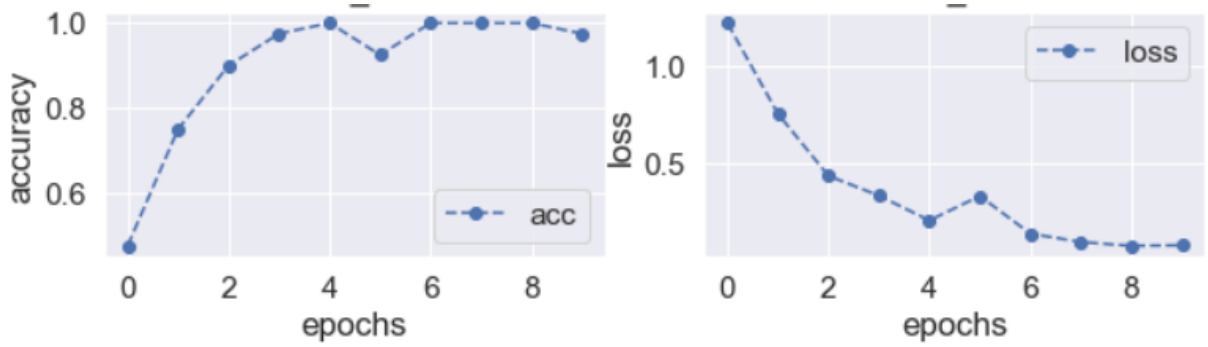

Рис. 3.3 Графік точності моделі і значення функції втрат на тренувальній вибірці

Значення точності моделі можна також перевірити проаналізувавши матрицю невідповідностей на рис. 3.4.

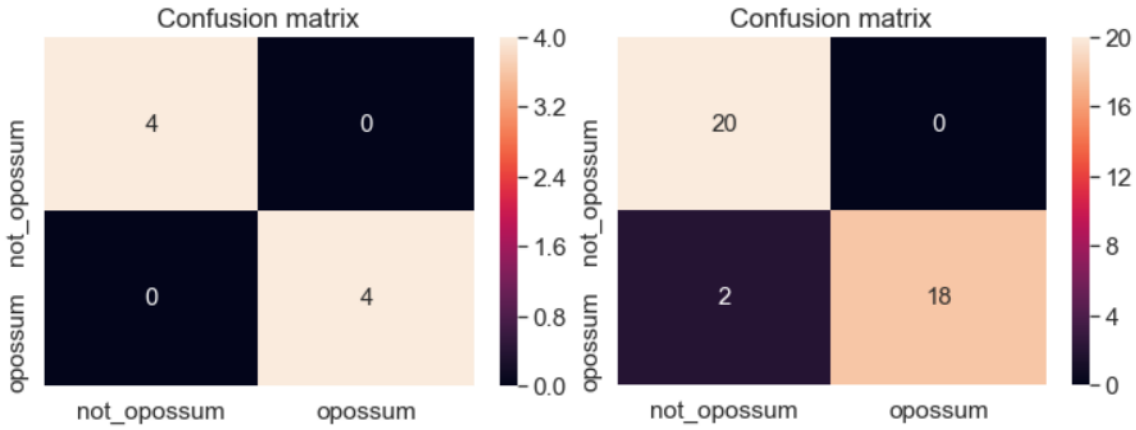

Рис. 3.4 Матриця невідповідностей

Для функція втрат SparseCategoricalCrossentropy найкраще себе показав оптимізатор SGD. Графік точності моделі і значення функції втрат на тренувальній вибірці зображено на рис. 3.5.

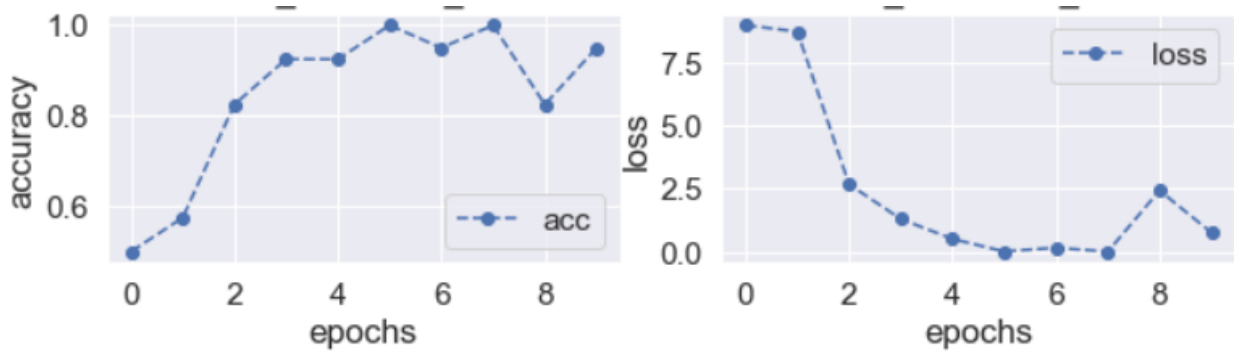

Рис. 3.5 Графік точності моделі і значення функції втрат на тренувальній вибірці

Значення точності моделі можна також перевірити проаналізувавши матрицю невідповідностей на рис. 3.6.

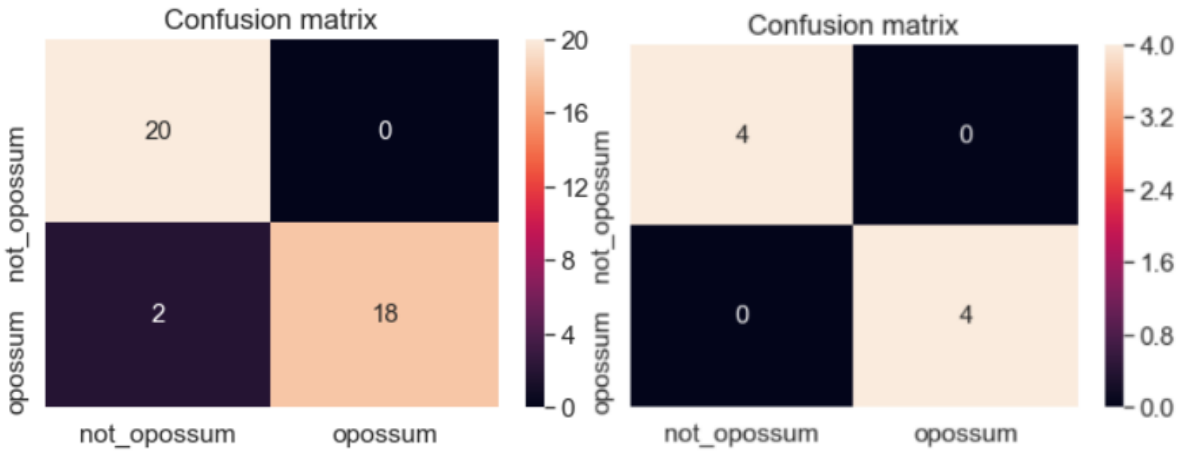

рис. 3.6 Матриця невідповідностей

3.4.2 Базова модель: Порівняльний аналіз результатів в залежності від оптимізатора і функції втрат

Loss Optimizator	Adam	SGD	RMSprop
BinaryCrossentropy	Train: acc = 1 loss = 0.0004 Test: acc = 1 loss = 0.0496	Train: acc = 0.95 loss = 0.3578 Test: acc = 0.875 loss = 0.4288	Train: acc = 1 loss = 0.0045 Test: acc = 1 loss = 0.1292
SparseCategorical Crossentropy	Train: acc = 1 loss = 5e-5 Test: acc = 0.875 loss = 0.1423	Train: acc = 1 loss = 0.1548 Test: acc = 1 loss = 0.3636	Train: acc = 1 loss = 5e-6 Test: acc = 1 loss = 0.0956

Табл. 3.2

Кількість параметрів моделі становить 2,664,546. Програмний код наведено в додатку Б.

Для функція втрат BinaryCrossentropy найкраще себе показав оптимізатор Adam. Графік точності моделі і значення функції втрат на тренувальній вибірці зображено на рис. 3.7.

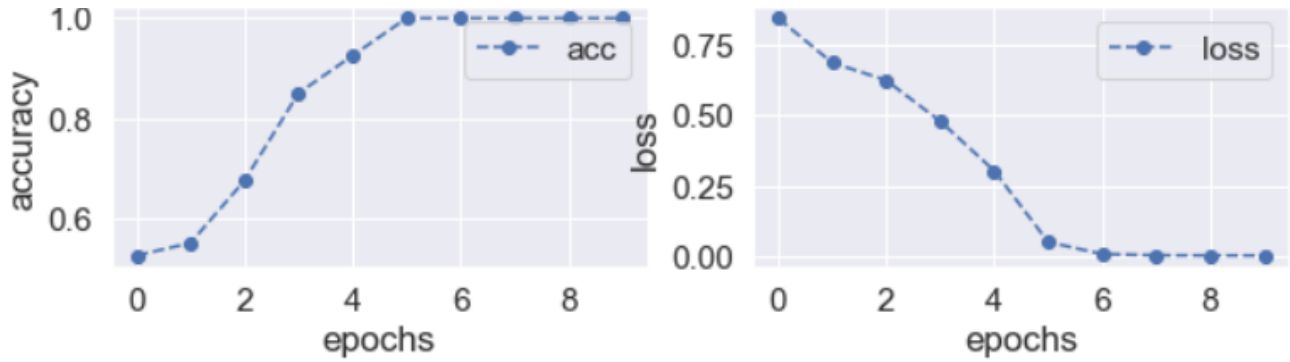

Рис. 3.7 Графік точності моделі і значення функції втрат на тренувальній вибірці

Значення точності моделі можна також перевірити проаналізувавши матрицю невідповідностей на рис. 3.8.

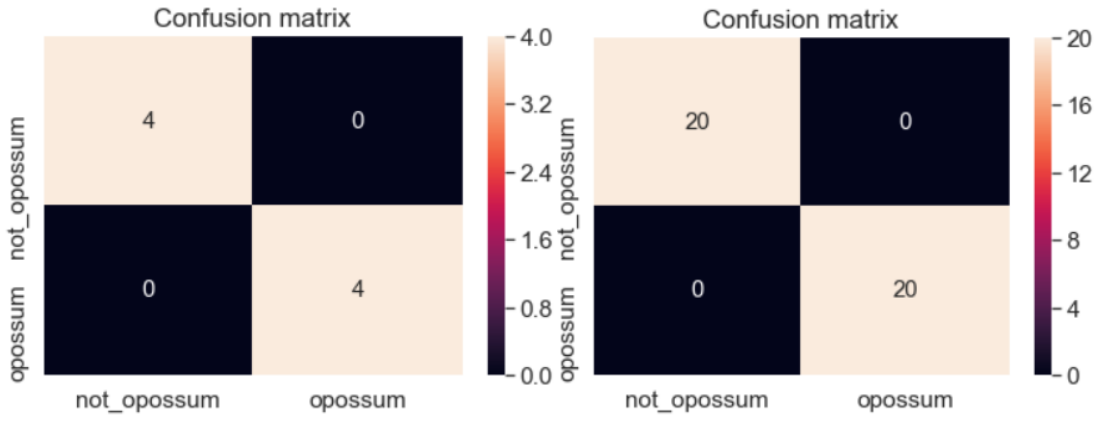

Рис. 3.8 Матриця невідповідностей

Для функція втрат SparseCategoricalCrossentropy найкраще себе показав оптимізатор RMSprop. Графік точності моделі і значення функції втрат на тренувальній вибірці зображено на рис. 3.9.

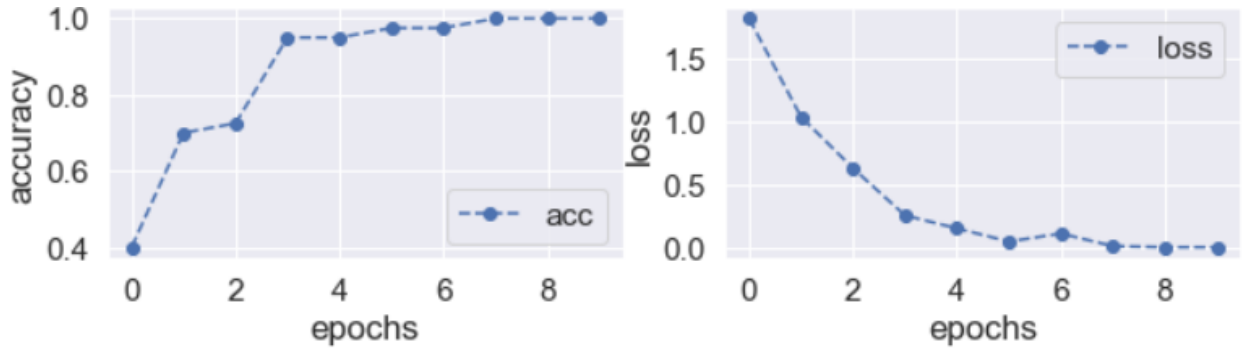

Рис. 3.9 Графік точності моделі і значення функції втрат на тренувальній вибірці

Значення точності моделі можна також перевірити проаналізувавши матрицю невідповідностей на рис. 3.10.

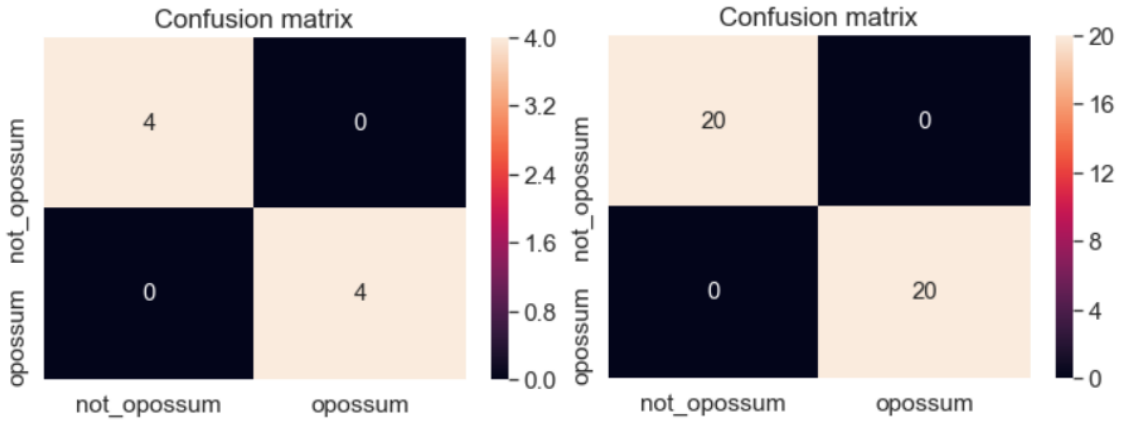

Рис. 3.10 Матриця невідповідностей

3.4.3 MobileNet: Порівняльний аналіз результатів в залежності від оптимізатора і функції втрат

Loss Optimizator	Adam	SGD	RMSprop
BinaryCrossentropy	Train: acc = 1 loss = 0.5379 Test: acc = 1 loss = 0.5374	Train: acc = 0.525 loss = 0.6926 Test: acc = 0.875 loss =0.6868	Train: acc = 1 loss = 0.5823 Test: acc = 1 loss = 0.5812
SparseCategorical Crossentropy	Train: acc = 1 loss = 0.4037 Test: acc = 1 loss = 0.4030	Train: acc = 0.675 loss = 0.6892 Test: acc = 0.75 loss = 0.6838	Train: acc = 1 loss = 0.5025 Test: acc = 1 loss = 0.5026

Табл. 3.3

Кількість параметрів моделі становить 3,540,986. Програмний код наведено в додатку В.

Для функція втрат BinaryCrossentropy найкраще себе показав оптимізатор Adam. Графік точності моделі і значення функції втрат на тренувальній вибірці зображено на рис. 3.11.

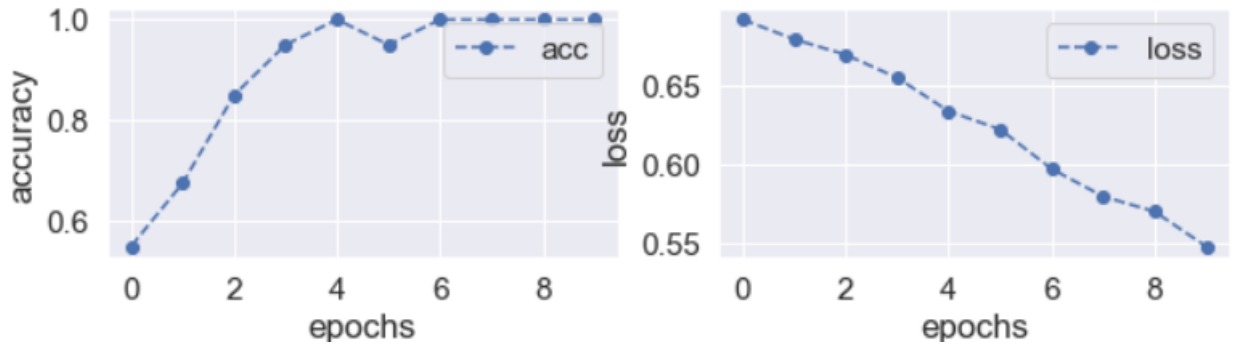

Рис. 3.11 Графік точності моделі і значення функції втрат на тренувальній вибірці

Значення точності моделі можна також перевірити проаналізувавши матрицю невідповідностей на рис. 3.12.

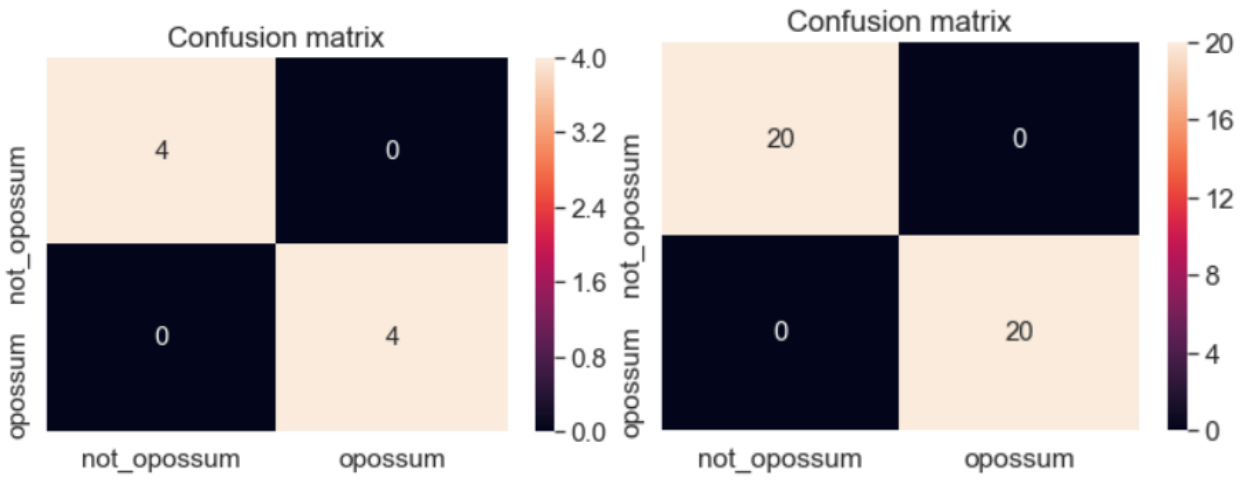

Рис. 3.12 Матриця невідповідностей

Для функція втрат SparseCategoricalCrossentropy найкраще себе показав оптимізатор Adam. Графік точності моделі і значення функції втрат на тренувальній вибірці зображено на рис. 3.13.

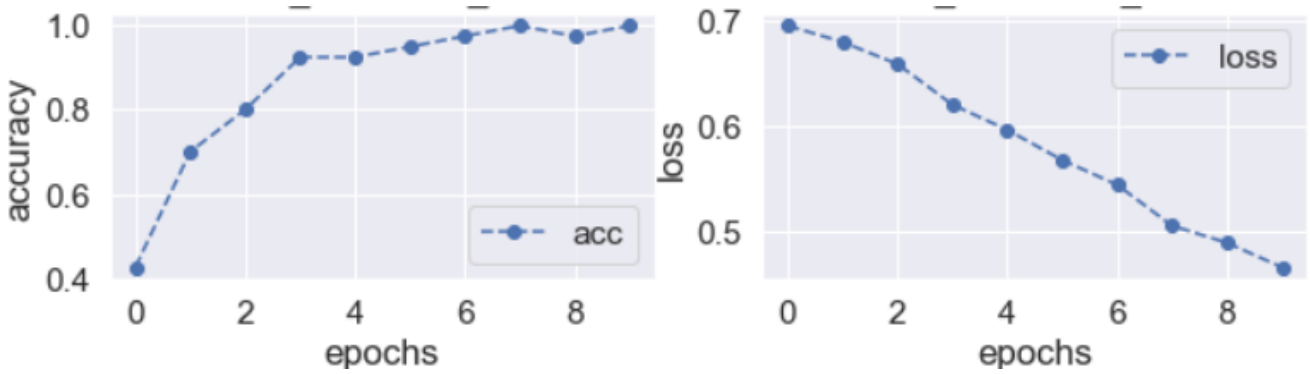

Рис. 3.13 Графік точності моделі і значення функції втрат на тренувальній вибірці

Значення точності моделі можна також перевірити проаналізувавши матрицю невідповідностей на рис. 3.14.

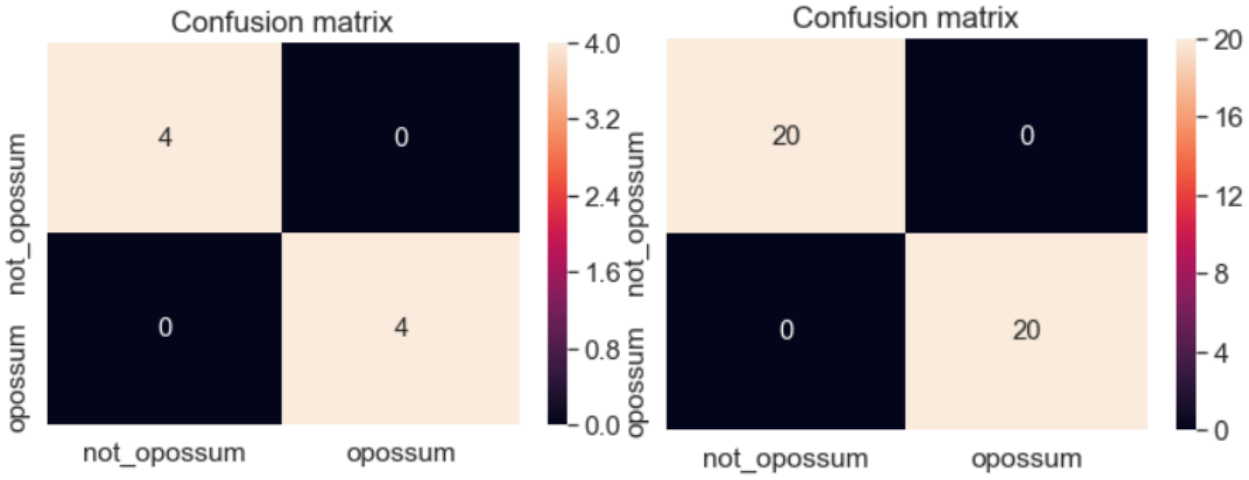

Рис. 3.14 Матриця невідповідностей

3.5 Висновки до 3-ого розділу

У цьому розділі ми на практиці за допомогою обраного набору даних розглянули деякі відомі архітектури CNN. Як бачимо, найкраще себе показали моделі із відносно невеликою кількістю параметрів і глибиною мережі, а саме моделі Lenet та Mobilenet. Тому можна сказати, в силу нашого невеликого набору даних для вирішення нашої задачі класифікації нам достатньо більш простих архітектур CNN.

Висновки

В даній дипломній роботі було розглянуто моделі класифікації за допомогою згорткових нейронних мереж. В першому розділі було розглянуто обраний датасет для наших моделй та наведено приклади зображень. Також було описано постановку задачі класифікації зображень та основні принципи. Було розглянуто актуальність поставленої задачі.

У другому розділі більше уваги приділялося математичним основам роботи, що формулюють фундамент для практичної частини. Було розглянуто основні принципи машинного та глибокого навчання, нейронних і згорткових мереж. Також основну увагу приділялося на принципи роботи і побудову згорткових нейронних мереж: види шарів, функцій активацій, оптимізаторів, функцій втрат, основні архітектури. Для більш глибокого опису було наведено графічну ілюстрацію оглянутих явищ.

Третій розділ було присвячено практичній частині роботи. Перш за все було обрано основні програмні продукти для вирішення поставленої задачі, зокрема це мова програмування, середовище розробки та основні бібліотеки. Далі було описано які саме архітектури нейронних мереж ми використовуємо, принципи їх будови та роботи. На кінець, було представлено результати моделей на прикладі нашого набору даних із зазначенням необхідних графіків. Також було зроблено висновок про найкращу модель з боку вибраного оптимізатора та функції втрат.

Літературні джерела

1. Шолле Франсуа, Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Библиотека программиста»).
2. Convolution neurol network wiki: https://en.wikipedia.org/wiki/Convolutional_neural_network
3. Convolution neurol network:

   https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
4. Сверточные нейронные сети: https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%91%D1%80%D1%82%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C
5. Сверточные нейронные сети:

   https://habr.com/ru/post/348000/
6. Alexnet:

   https://en.wikipedia.org/wiki/AlexNet
7. Alexnet architecture: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/
8. Alexnet:

   https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/

9. Alexnet:

   https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/

10. VGG architecture:

    https://russianblogs.com/article/3773549397/

11. VGG python:

    https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951

12. Lenet:

    https://en.wikipedia.org/wiki/LeNet

13. Lenet:

    https://uk.wikipedia.org/wiki/LeNet

14. Mobilenet:

    https://habr.com/ru/post/352804/

15. Mobilenet python:

    https://russianblogs.com/article/60511631815/

16. Data augmentation:

    https://neptune.ai/blog/data-augmentation-in-python

17. Data normalization:

    https://en.wikipedia.org/wiki/Database_normalization

Додатки

Додаток А

```
import numpy as np
import os
from sklearn.metrics import confusion_matrix
import seaborn as sn; sn.set(font_scale=1.4)
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tqdm import tqdm
class_names = ['not_opossum', 'opossum']
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}

nb_classes = len(class_names)

IMAGE_SIZE = (227, 227)

def load_data():
    """
        Load the data:
            - 14,034 images to train the network.
            - 3,000 images to evaluate how accurately the network learned to
classify images.
    """

    datasets = ['input/train', 'input/test']
    output = []

    # Iterate through training and test sets
    for dataset in datasets:

        images = []
        labels = []

        print("Loading {}".format(dataset))

        # Iterate through each folder corresponding to a category
        for folder in os.listdir(dataset):
            label = class_names_label[folder]

            # Iterate through each image in our folder
            for file in tqdm(os.listdir(os.path.join(dataset, folder))):
```

```
                # Get the path name of the image
                img_path = os.path.join(os.path.join(dataset, folder), file)

                # Open and resize the img
                image = cv2.imread(img_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, IMAGE_SIZE)

                # Append the image and its corresponding label to the output
                images.append(image)
                labels.append(label)

        images = np.array(images, dtype = 'float32')
        labels = np.array(labels, dtype = 'int32')

        output.append((images, labels))

    return output

(train_images, train_labels), (test_images, test_labels) = load_data()

n_train = train_labels.shape[0]
n_test = test_labels.shape[0]

print ("Number of training examples: {}".format(n_train))
print ("Number of testing examples: {}".format(n_test))
print ("Each image is of size: {}".format(IMAGE_SIZE))

import pandas as pd

_, train_counts = np.unique(train_labels, return_counts=True)
_, test_counts = np.unique(test_labels, return_counts=True)
pd.DataFrame({'train': train_counts, 'test': test_counts},
index=class_names).plot.bar()
plt.show()

train_images = train_images / 255.0
test_images = test_images / 255.0

from keras.preprocessing.image import ImageDataGenerator
shift_fraction=0.005
batch_size = 1
num_classes = 2
epochs = 10
datagen = ImageDataGenerator(
        rotation_range=45,
        horizontal_flip = True,
```

```
        vertical_flip = True
        )

a = datagen.flow(train_images,train_labels, batch_size=1)
train_images_full = train_images.copy()
train_labels_full = train_labels.copy()
for _ in range(20):
    img, label = a.next()
    train_images_full = np.concatenate((train_images_full, img), axis=0)
    train_labels_full = np.concatenate((train_labels_full, label), axis=0)

train_labels_full_new = tf.keras.utils.to_categorical(train_labels_full,
num_classes=2)
test_labels_new = tf.keras.utils.to_categorical(test_labels, num_classes=2)

import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.advanced_activations import LeakyReLU
import tensorflow as tf
from tensorflow import keras
import keras.layers as layers
model = keras.Sequential()
model.add(layers.Conv2D(filters=96, kernel_size=(11, 11),
                        strides=(4, 4), activation="relu",
                        input_shape=(227, 227, 3)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool2D(pool_size=(3, 3), strides= (2, 2)))
model.add(layers.Conv2D(filters=256, kernel_size=(5, 5),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)))
model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=256, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)))
```

```
model.add(layers.Flatten())
model.add(layers.Dense(4096, activation="relu"))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation="softmax"))
model.compile(loss='BinaryCrossentropy',
              optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.0001),
              metrics=['accuracy'])

model.summary()

history = model.fit(train_images_full, train_labels_full_new, batch_size=1,
validation_split = 0, epochs=10)

train_loss = model.evaluate(train_images_full, train_labels_full_new)
test_loss = model.evaluate(test_images, test_labels_new)

def plot_accuracy_loss(history):
    """
        Plot the accuracy and the loss during the training of the nn.
    """
    fig = plt.figure(figsize=(10,5))

    # Plot accuracy
    plt.subplot(221)
    plt.plot(history.history['accuracy'],'bo--', label = "acc")
    #plt.plot(history.history['val_accuracy'], 'ro--', label = "val_acc")
    plt.title("train_acc")
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    plt.legend()

    # Plot loss function
    plt.subplot(222)
    plt.plot(history.history['loss'],'bo--', label = "loss")
    #plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
    plt.title("train_loss")
    plt.ylabel("loss")
    plt.xlabel("epochs")

    plt.legend()
    plt.show()

plot_accuracy_loss(history)

predictions = model.predict(test_images)
pred_labels = np.argmax(predictions, axis = 1)
```

```
print(pred_labels)

CM = confusion_matrix(test_labels, pred_labels)
ax = plt.axes()
sn.heatmap(CM, annot=True,
           annot_kws={"size": 15},
           xticklabels=class_names,
           yticklabels=class_names, ax = ax)
ax.set_title('Confusion matrix')
i, k = ax.get_ylim()
ax.set_ylim(i+0.5, k-0.5)
plt.show()

predictions = model.predict(train_images_full)
pred_labels = np.argmax(predictions, axis = 1)
print(pred_labels)

CM = confusion_matrix(train_labels_full, pred_labels)
ax = plt.axes()
sn.heatmap(CM, annot=True,
           annot_kws={"size": 15},
           xticklabels=class_names,
           yticklabels=class_names, ax = ax)
ax.set_title('Confusion matrix')
i, k = ax.get_ylim()
ax.set_ylim(i+0.5, k-0.5)
plt.show()
```

Додаток Б

```
model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150,
150, 3)),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation=tf.nn.relu),
        tf.keras.layers.Dense(2, activation=('softmax'))
        ])
```

Додаток В

```
from tensorflow.keras.models import Model
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense


model = MobileNetV2(input_shape=(224, 224,3))
model.layers.pop()
for layer in model.layers[:-4]:
    layer.trainable = False
output = Dense(2, activation="softmax")
output = output(model.layers[-1].output)
model = Model(inputs=model.inputs, outputs=output)
```