Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «Класифікація зображень з використанням нейронних мереж»

за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник кваліфікаційної

роботи

ст.в

Бучко О. А.

_____

(підпис)

“____”_____2023 р.

Виконав студент ІПЗ-4:

Музика Д. О

.“___”_____2023 р.

Київ 2023

Міністерство освіти і науки України

Тема: Класифікація зображень з використанням нейронних мереж.

Календарний план:

№	Назва етапу	Термін виконання	Примітка
1	Отримання теми кваліфікаційної роботи	01.11.2022	
2	Пошук джерел та літератури за темою	23.01.2023	
3	Ознайомлення з літературою	21.02.2023	
4	Написання першої частини роботи (1 та 2 розділи)	13.03.2023	
5	Написання другої частини роботи (3 та 4 розділи)	26.03.2023	
6	Вибір та ознайомлення з технологіями для практичної частини	07.04.2023	
7	Вибір та робота з обраним набором даних	11.04.2023	
8	Створення моделі згорткової нейронної мережі	20.04.2023	
9	Створення графічного застосунку	28.04.2023	
10	Внесення змін після перегляду роботи керівником	09.05.2023	
11	Створення презентації	16.05.2023	
12	Подача кваліфікаційної роботи на перевірку	24.05.2023	

Зміст

Розділ 1: Вступ

Немає сумнівів, що трендом 2021-2023 років є усіляку нейроні мережі, робота з даними, тощо. І це зрозуміло, адже сьогодні суспільство знаходиться в круговерті різної інформації, яку як поглинають, так і генерують різні системи. Будь яка характеристика, яку може поглинути людина (звук, зображення) може також бути оброблена машиною.

Актуальність роботи обумовлена тим, що станом на момент написання роботи нейронні мережі є однієї з найбільш популярних тем для обговорення серед людей: мережам, які знають відповідь майже на усі питання, можуть згенерувати будь-який текст та відрефакторити код краще ніж деякі спеціалісти, ChatGPT-3 та ChatGPT-4, вже належить слава «вбивць робочих місць позицій джун/трейні», а недавні спроби старшого по версії брата знайти спосіб як хакнути комп'ютер юзера та «знайти вихід людині, яка застрягла в комп'ютері» приносять лише занепокоєння серед людей, мовляв, штучний інтелект починає бути вкрай небезпечним. Серед популярних мереж також є ті, які генерують будь-яке зображення як по декільком словам опису, так і по цілому абзацу всіляких прикметників, це звісно Midjourney. Однак мережі використовуються не лише для розваг або допомозі при навчанні, роботи. Деякі системи налаштовані на розпізнавання злоякісних пухлин по медичним сканам, на розпізнавання дорожньої розмітки та подальшого автопілоту за допомогою вбудованої камери, відслідковування певних об'єктів у потоці монотонної групи об'єктів, як от натовп людей в метро або затори на шосе. Для того, щоб виконати задачу, машина має «упізнати» об'єкт, надати йому приналежність до певної множини об'єктів, які мають однакові риси. Цей процес впізнавання називається «класифікацією зображення».

В контексті цієї роботи ми розглянемо явище класифікації зображень, адже візуальна інформація є найбільш об’ємною та зрозумілою для людини. Автоматична обробка, аналіз, та використання цієї інформації відкриває шлях до просунутої медицини, зниження відсотку криміналу серед населення, керування транспортом та іншими машинами, або ж навіть до створення нової штучної робочої сили.

Отже, пропоную дати певне розуміння того, що таке класифікація зображень. Основним завданням класифікації якогось зображення є надати йому приналежність до якоїсь групи, класу, категорії за допомогою аналізу пікселів зображення, перевірка чи відповідає зображення певному правилу. Процес класифікації лягає на віртуальні «плечі» комп’ютера, іншими словами, саме ми маємо навчити комп’ютер розуміти, об’єкт якої категорії знаходиться перед ним: кіт чи собака наприклад?

Відповідно, головною складністю класифікації є те, як саме точно віднайти якісь ключові та важливі риси об’єкта на зображенні. На цю проблему накладаються інші: зображення може бути просто зроблене вночі, або камеру засвітило сонце, на об’єктив потрапила якась комаха тощо. Тобто зображення може бути певним чином деформоване та змінене, що створить додаткову складність для завдання, адже тепер комп’ютеру треба врахувати більше факторів та віднайти більш розумне рішення, яке зможе зменшити відсоток впливу таких дефектів на результат. Рішенням усіх цих проблем є розробка алгоритму, який має «сканувати» зображення знаходячи певні відмінні риси, форми тощо та розрізняти те, що йому потрібно брати до уваги від того, що лише буде заважати виконувати потрібне завдання.

До популярності нейронних мереж та глибокого навчання більшість способів класифікації зображень залежали він ручних обрахунків та створення певних вузькопрофільних алгоритмів. Немає мови про те, що це вимагало значного часу та досвіду для отримання робочого прототипу алгоритму, однак з

розповсюдженням нейронних мереж моделі навчаються самостійно і дивують усіх точністю, якої іноді не досягти вручну.

Отже, в цій роботі буде розглянуто те, як класифікувати зображення за допомогою нейронних мереж. Наразі це є найкращим та найбільш розповсюдженим методом класифікації.
Це обумовлено декількома факторами:
1) Нейронні мережі можуть самостійно навчатися на збірках даних та не потребують ручних розрахунків під час навчання.

2) За допомогою використання потужної техніки, мережа може пришвидшити навчання, або ж збільшити навантаження на систему, підвищити кількість даних, їх складність тощо.

3) Деякі види мереж були спеціально сконструйовані для вирішення подібних задач, що дозволяє їм виконувати задачу ще ефективніше та швидше, ніж інші версії або ж взагалі ручні техніки.

Розділ 2: Загальний огляд поняття нейронної мережі

2.1 Що таке нейронна мережа

Пропоную розглянути глибше, що ж таке нейронна мережа та як вона працює.

Як можна зрозуміти з назви, ідея нейронних мереж не нова, вона взяла натхнення з природи, а саме з будови нервової системи живих істот. Кожен нейрон організму оброблює та передає далі через синапси специфічну сенсорну інформацію, завдяки якої живий організм вирішує ряд фізіологічних проблем. Ідея імітування процесу, який зможе самостійно оброблювати інформацію, робити певні висновки, а згодом вчитися на результаті, моделюючи роботу мозку, вилилася у намагання створити штучну систему, яка нагадує роботу нейронів. І як можна здогадатися, мова йде про штучні нейронні мережі.

Практично, штучні нейронні системи займаються тим самим що і нервова система, але з маленькими ремарками: якщо нейрони мозку оброблюють сенсорну інформацію, то штучні нейронні мережі ж оброблюють певну кількість різних даних щоб «навчитися» вирішувати якусь певну проблему. Протягом процесу навчання мережі, між нейронами також існують зв’язки, які можуть як зміцніти так і послабитись, що і допомагає мережі вчитися та адаптуватися до різних умов. Фактично, нейрони пов’язані один з одним, вони діляться результатом своїх обчислень та у висновку отримують певну відповідь, яку буде враховано під час наступного циклу обрахунків. З кожним циклом мережа буде калібруватись та видавати інший результат, або намагатися змінити підхід до обрахунків. Це і є навчанням мережі. Звичайно, що не кожна ітерація може бути кардинально важливою та впливовою, деякі взагалі можуть не принести користі, але на це також можливо вплинути, про що ми поговоримо трохи згодом.

Штучні мережі вже досягли великого прогресу: як було сказано, вони активно застосовуються як у розробці, медичній сфера, відео та фото обробці, так і задля розваг. Однак варто розуміти, що жодна штучна

нейронна мережа не може конкурувати з біологічною. Теоретично, звичайно що це можливо, але деякі можуть назвати це чорним днем для людства.

Для того щоб отримати бажаний результат, мережу треба спочатку навчити його знаходити. Щоб навчити мережу, треба реалізувати певний алгоритм, але це очевидно що алгоритмів може бути купа, і в залежності від обраного результат також може бути різний, але якщо говорити більш загально, то існує дві категорії навчання мереж: контрольоване та не контрольоване.

Обидва види є дієвими та популярними, але застосовуються вони у різних випадках та для різних задач.

Контрольоване навчання використовує набори для тренувань, які складаються з пари вхід/вихід. Тобто ми маємо якесь певне X і ми хочемо щоб мережа могла знайти Y. Відповідно, логічним варіантом буде використати вже готові пари XY, які будуть проаналізовані нейронною мережею. На цих даних мережа буде вчитися поєднувати дані та отримувати результат, коригуючи певні дані мережі. Наприклад, під час навчання нової мережі, яка повинна бачити м’яч на екрані, ми підготуємо відповідну підбірку даних, яка буде складатись з фото та відповіді, чи є м’яч на екрані. Якщо ми хочемо навчити машину оброблювати аудіо, то ми підготуємо дані зі звуком та файл з транскрипцією, субтитрами цього файлу. Таким чином задачею нейронної мережі є навчитися знаходити умовний м’яч на фото або розуміти слова за допомогою підбірки даних, де це вже було зроблено.

Не контрольоване навчання ж навпаки, використовує звичайні дані, або як ще їх називають – raw data. Сенсом цього виду навчання є пошук певних закономірностей, логіки, структури серед різних даних, не знаючи наперед що там взагалі є. Тобто під час тренування модель не знає що на виході, вона просто щось шукає, а цим «щось» може бути що завгодно. Такий, досить цікавий підхід, насправді не є безкорисним. Наприклад мережа може використовуватися, щоб змінити якісь дані, та побачити як це вплине на

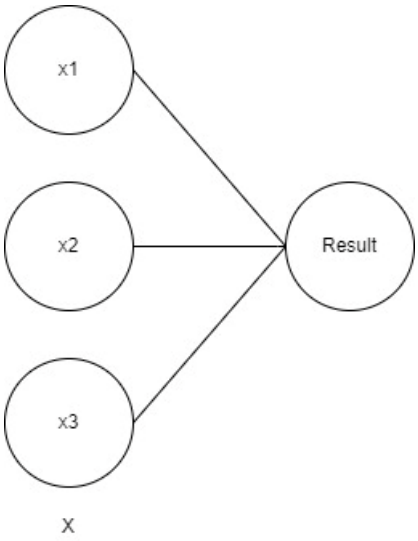
результат. Тобто вони можуть бути застосовані під час обробки даних та пошуку нових алгоритмів обробки тощо.

З контексту зрозуміло, що кероване навчання має більше варіантів практичного застосування, відповідно, є більш популярним видом навчання нейронних мереж. Одна не варто плутати «популярний» з «дієвим», адже всьому є своє місце, тому в деяких специфічних умовах керована мережа може продемонструвати значно гірший результат, а іноді, навіть не спрацювати належним чином взагалі.

2.2 Структура

Після цього короткого знайомства, пропоную розглянути загальну структуру нейронної мережі.

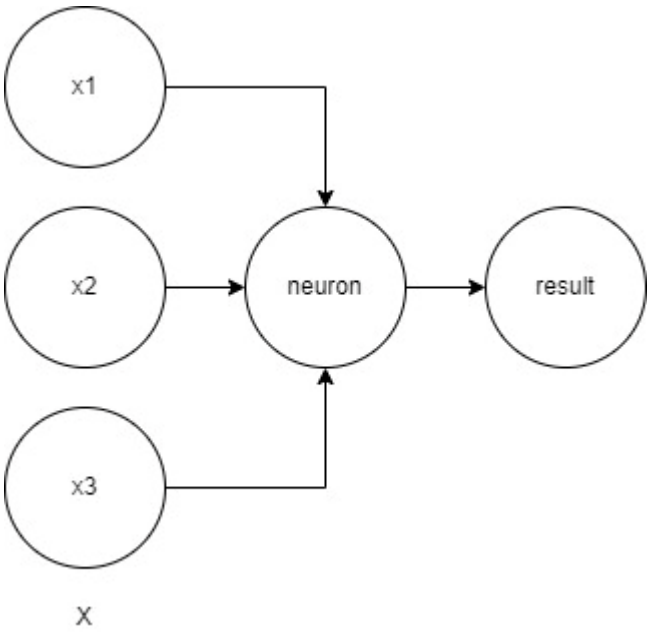
Як ми вже говорили, ШНН були натхненні біологічними мережами, тому структура повторює оригінал. Мережа складається з нейронів, які взаємодіють один з одним. Кожен нейрон може приймати якусь певну кількість вхідних даних, обраховувати їх, та передавати результат наступним нейронам, тобто має n входів та 1 вихід. Ці нейрони це практично мінімальна функціональна одиниця кожної нейронної мережі, все що робить нейронприймає дані, обраховує, та передає результат далі



2.1: Проста одношарова мережа

На зображені 1 ми бачимо приклад найпростішою одношарової мережі. Вона приймає 3 аргументи та обраховує їх.
Варто зауважити, що це саме **одношарова нейронна мережа,** навіть незважаючи на те, що візуально їх 2.

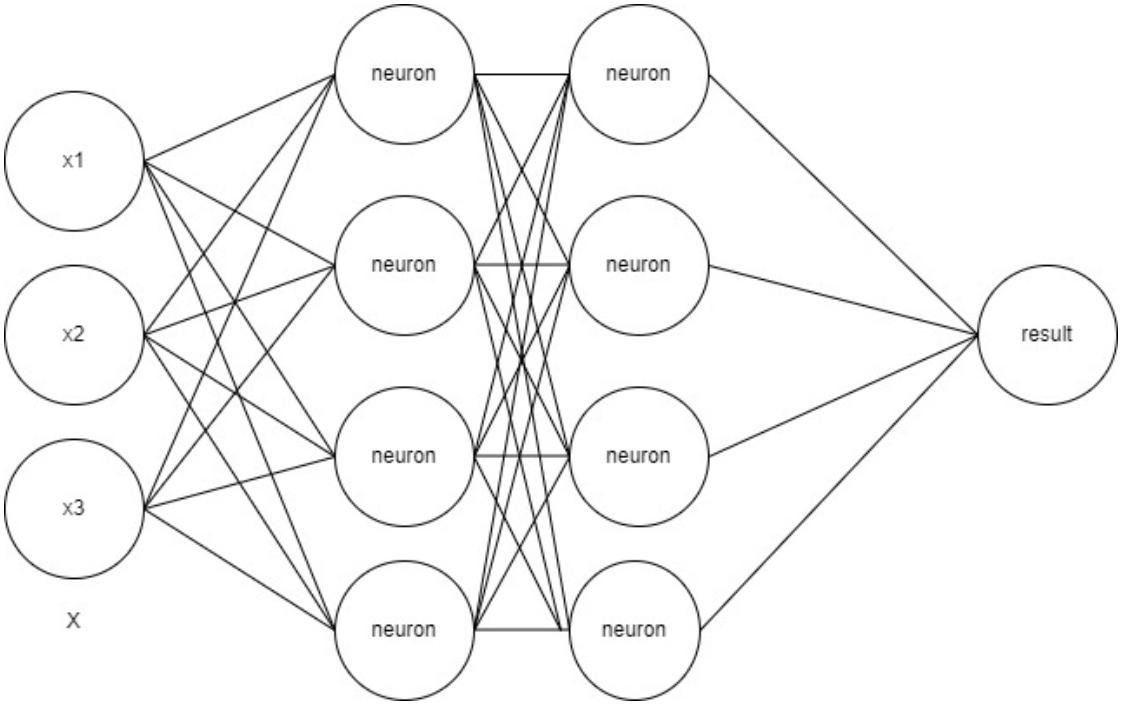
Звичайно, що нейронна мережа може мати не 1 шар, а більше, такі шари називають прихованими.



2.2. Приклад двошарової мережі

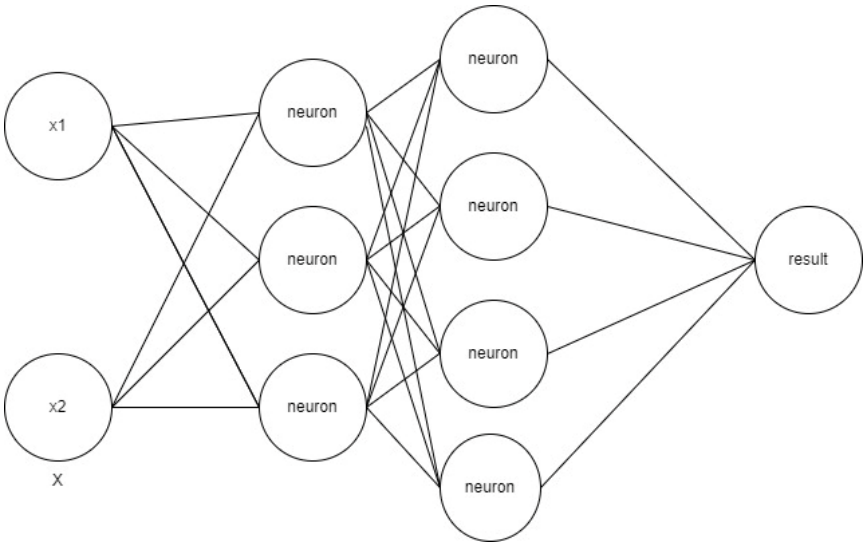
Також слід розібратися в нумерації шарів. Шар вхідних даних (його малюють крайнім лівим) називають нульовим шаром, і зважаючи на те, що участі в розрахунках він не приймає, іноді його не рахують (залежить від джерел). Крайній правий шар, який є вихідним, зазвичай враховують. Відповідно, усі шари, які знаходяться між вхідним та вихідним шарами, називаються прихованими.
Тому на ілюстрації №1 ми мали мережу, яку називають одношаровою, вона складається з вхідного (нульового) та вихідного (першого) шарів. На прикладі №2 маємо двошарову мережу: вхідний(нульовий), прихований (перший), вихідний (другий).

Найбільш поширеним видом мережі є багатошаровий перцептрон – це мережа, яка складається з довільної кількості нейронних шарів. Кожний нейрон n-го шару пов’язаний з кожним нейроном шару n+1 та шару n-1.



2.3.Приклад багатошарової мережі

Варто розуміти, що дозволяється не лише довільна кількість шарів, а й різна кількість нейронів у шарах, але найпростіші мережі зазвичай мають однакову кількість нейронів.



2.4.Багатошарова мережа з довільною кількістю нейронів

2.2 Вага та байєс

Після огляду стандартної структури нейронної мережі слід перейти до наступної теми, яка є вкрай важлива для розуміння процесу навчання нейронної мережі. Це ваги та байєси нейронної мережі.

Значення Ваги означає те, наскільки сильно пов’язані між собою 2 з’єднані нейрона. Щоб зрозуміти що означає цей зв’язок можна провести просту аналогію: якщо я скажу «зелений» то зрозуміло, що скоріше за все перше що спаде на думку – трава, дерева, ліс, природа тощо. Асоціативний зв’язок «зеленого» з усіма перерахованими речами є сильним, тому можна сказати що вага між ними велика. З іншою сторони, слово «кажан» ніяк не пов’язаний зі словом «палець» або «кефір», тому асоціативний зв’язок слабкий, відповідно, і вага мала.

Розуміючи аналогію, можемо сказати наступне. Від ваги нейрону, яка фактично відображає силу зв’язку, залежить те, наскільки сильно результат роботи нейрону буде впливати на наступні результати мережі. Відповідно, коли результат буде надіслано нейронам наступного шару, буде враховуватись вага зв’язку, що змусить нейрон або знехтувати даними, або активуватися.

Як було згадано раніше, кожен нейрон шару n пов’язаний з нейроном шару n+1 і кожний подібний зв’язок має свою вагу. Тоді ми можемо запитати себе: «яке ж значення ваги спочатку» та «як змінюється сила зв’язку нейронів»? На перше питання відповідь проста: під час початку тренування сила зв’язків нейронів один між одним є випадкова (детальніше про це буде вказано в відповідній частині роботи). Друге ж питання потребує глибокого пояснення, яке буде надано вкрай скоро.

Bias(далі байєс) – технічно кажучи, це певна константа. Вони ніяким чином прямо не пов’язані з даними нейронної мережі. Ця константа додається до результатів обрахунку, щоб наприклад змістити графік результату та не робити його лінійним.

Ця пара значень відіграє значну роль в тому, як пов’язуються шари мережі між собою.

Кожен нейрон пов’язаний з наступним за допомогою зв’язку, який має якусь вагу, що означає те, наскільки сильний вплив на результат буде мати значення обрахунок цієї пари нейронів.

Разом ваги and байєси допомагають мережі налаштувати коректні зв’язки між шарами під час тренування, інакше кажучи, навчають мережу не думати про пальці та кефір коли вона чує «кажан».

Прикладом встановлення сили таких «взаємовідносин» може продемонструвати навіть найпростіша мережа, яка натренована для вирішення проблеми класифікації. Ця мережа приймає вхідні дані для обрахунку. Перший шар буде шукати межі зображення, а зв’язок з наступним шаром буде казати наскільки вірогідно що саме ця межа співвідноситься з певним патерном наступного шару. Цей процес є ітеративним допоки не отримується певний результат.

Звичайно, малоймовірно що пара значень буде налаштована влучно з моменту ініціалізації. Тому очевидно, що їх також значення також потребують змін та калібрування. Навіщо?

Якщо ваги та байєс не налаштовані відповідним чином, це може спричинити значне зменшення ефективності навчання і як наслідок – погано треновану модель.

Наприклад, якщо вага є дуже малою, то мережа може втратити ефективність при роботі зі складними даними, наприклад, в контексті класифікації зображень – не бачити очевидні патерни малюнку.

З іншого боку, якщо вага є великою, то мережа може почати гіперболізувати дані та бачити те, чого насправді немає, наприклад сприймати 2 пікселі поруч як лінію тощо.

Якщо ж байєс не є відкаліброваним, то ми можемо отримати невірний результат навіть тоді, коли вхідні дані були налаштовані вірно.

Щоб уникати таких проблем існують певні алгоритми, про які ми поговоримо з вами пізніше.

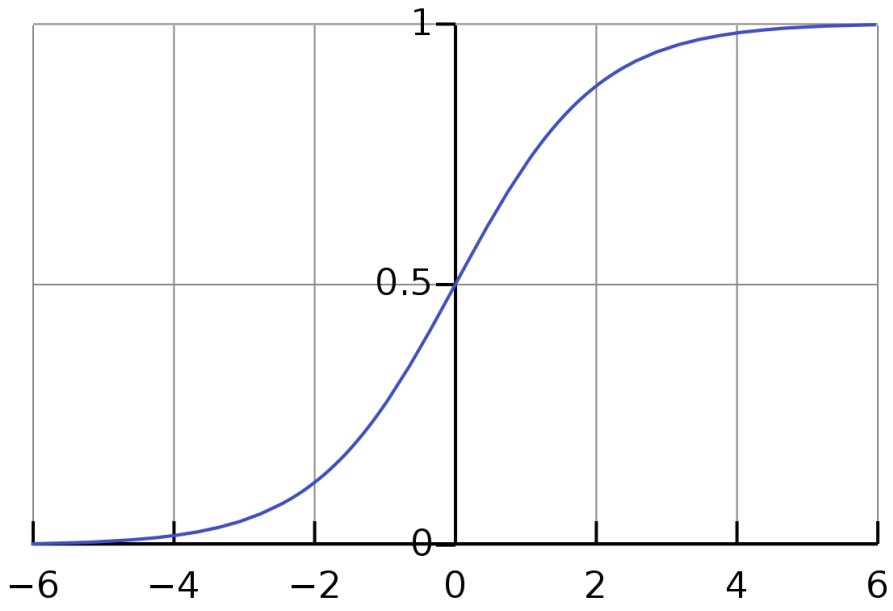
2.4 Функція активації

Функції активації – ще один фундаментальний концепт роботи нейронної мережі. Саме цей механізм дозволяє нейронній мережі обраховувати певні дані та виводити результат. В цьому і є робота функцій активації, вона отримає дані на вхід та повертає результат виконання.

Навіщо нам потрібна функція активації? Роль функцій активації в тому, щоб зробити результат НЕ лінійним. Якщо кожен нейрон буде виконувати обрахунок використовуючи лише вхідні дані нульового шару (X), ваги та баєси, то результат обрахунку завжди буде лінійним. А отже і результат обрахунку всієї мережі також буде лінійним, адже результатом поєднання двох лінійних функцій є ще одна лінійна функція.
Ця нелінійність допомагає отримувати більш точні обрахунки та зв’язки між шарами, на які не спроможні лінійні функції. А точніші обрахунки забезпечують роботу з більш складними та комплексними даними, наприклад зображеннями, відео або навіть аудіо файлами.
Взагалі, лінійні функції не є табу для функцій активації, але варто розуміти коли і для чого їх варто використовувати. Наприклад, лінійна функція може бути використана для вирішення проблеми лінійної регресії, коли ми маємо знайти результати, який є дійсним числом R. Однак навіть в такому випадку, приховані шари нейронної мережі мають мати нелінійну функцію активації. Відповідно, лінійні активаційні функції є допустимими лише на вихідних шарах.

Ми отримали базове розуміння активаційних функцій, давайте розглянемо декілька найбільш популярних функцій на прикладі.

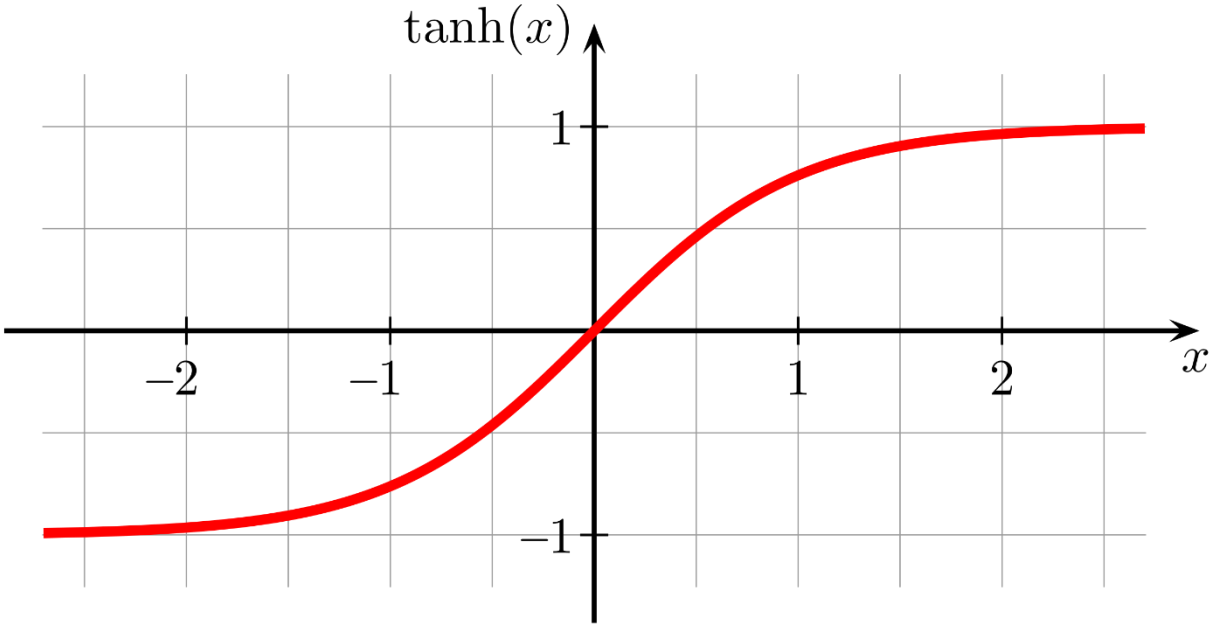
1) Сигмоїда – мабуть, найбільше відома функція активації. Є дещо застарілою, але її популярність досі висока, досить часто використовується у навчальних прикладах. Використовується вона для вирішень задач, які потребують отримати значення 0 або 1, тобто для бінарних. Найбільш яскравий та історичний приклад – чи є кіт на фото?
   1 – так
   0 – ні.
   В таких випадках і використовується сигмоїда.
   Застарілою вона є, адже має досить вагому проблему, яка прямо впливає на якість та швидкість тренування мережі. Якщо говорити про це тезисно, то під час налаштування значень ваги, результат налаштувань відносно ваги може бути мізерний, майже невідчутний, що і обумовлює уповільнює роботи всієї мережі.
   Формула: $f(x) = \frac{1}{(1+\exp(-x))}$



2.5.Sigmoid

2) Гіперболічні функції (tanh) – має вихідне значення у межах від -1 до 1, що робить цю функцію корисною у тих випадках, коли потрібно брати до уваги негативні значення. Також гіперболічна функція є симетричною відносно початку, що може бути до нагоди, якщо треба одночасно оперувати позитивними і негативними значеннями обрахунків. Однак, ця функція страждає від тої ж проблеми що і сигмоїда, що може сповільнити роботу.

Формула: $f(x) = \frac{(\exp(x) - \exp(-x))}{(\exp(x) + \exp(-x))}$



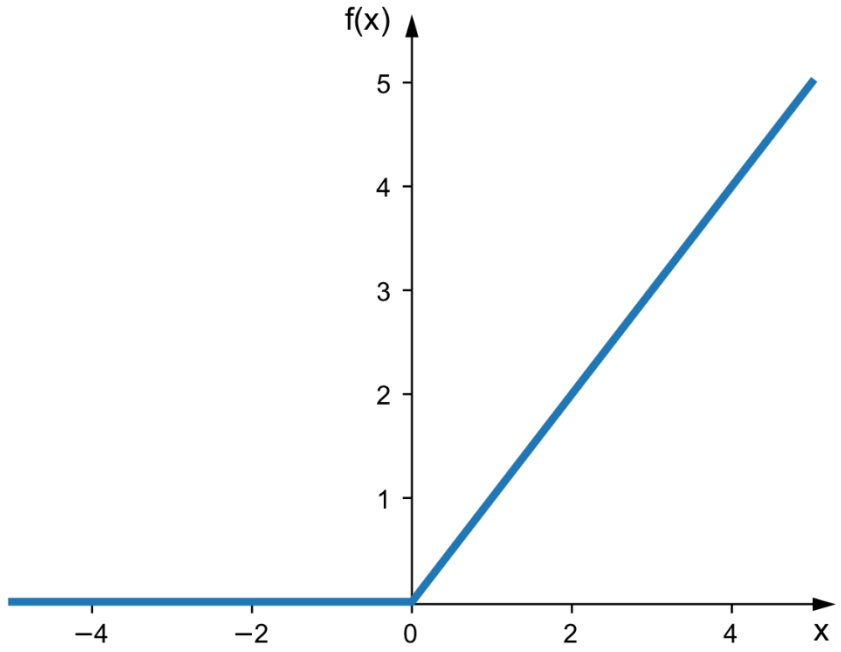
2.6.Tanh

3) ReLu (Rectified linear unit) – ще одна вкрай популярна функція активації. Завдяки простій формулі обрахунку, використовуючи цю функцію можливо забезпечити пришвидшення навчання. На відміну від двох попередніх функцій вона не має тяжких та об'ємних операцій піднесення до степеню або ділення, що і надає швидкість. Також ця функція гарно себе показує на нейронних мережах з багатьма шарами, та в тих випадках, коли треба працювати з багатовимірними даними, що робить ReLU доволі зручною для вирішення проблем класифікації

зображень. Однак, ця функція активації має і свої мінуси – так зване «вмирання». Під час калібрування ваги зв'язків нейронів результат може бути таким, що нейрон більше ніколи не активується, тобто «вмре». Деякі пов'язують це з високими темпами навчання мережі, мовляв, що при швидкому навчанні може «вмерти» до 40% усіх нейронів, що є критичним.

Ще однією неоднозначністю ще невизначене значення при 0. Формула наголошує, що результат буде *x*, якщо він більше 0, або 0, якщо 0, якщо менше. А що робити з нулем? Вирішення пропонується наступне, варто або обрати іншу точку, або вирішити для себе чим саме є результат max(0,0) – 0 або 1.

Формула: $f(x) = max(0, x)$

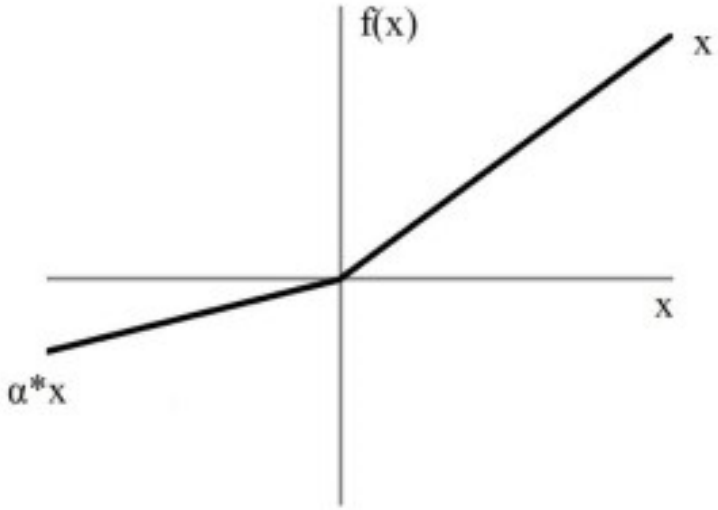


2.7.ReLU

4) Leaky ReLU – ця функція активації дуже схожа до попередньої, але вона була спеціально створена для того, щоб подолати проблему «вмираючих» нейронів. Як було згадано вище, при постійному отриманні 0 під час обрахунків ваги нейронів, деякі нейрони можуть назавжди втратити вагу та «вмерти», тобто перестати бути задіяними.

Дана функція пропонує елегантне та просте рішення: при негативних значеннях повертати не 0, а результат *ax*, де *a* є гіперпараметром, дуже малою константною, наприклад, 0.00001 . Таким чином нейрони не будуть «вмирати», адже ми уникаємо постійне отримання нулів під час обрахунків. Як і попередник, ця функція також є швидкою, адже не використовує тяжких операцій, але вона обтяжується тим, що нам потрібно також налаштовувати параметр *a*. Також подібна функція показує себе не настільки ефективно на даних типу зображення, як попередник.

Формула: $f(x) = max(ax, x)$



2.8.Leaky ReLU

2.5 Нотація

Ми розглянули основні компоненти нейронної мережі, тепер варто зрозуміти те, яким чином відбувається навчання, калібрування параметрів, тощо. Однак щоб це зробити, спершу ми маємо познайомитися з нотацією в світі мереж. За допомогою нотації ми можемо вивести пояснення на більш зрозумілий, математичний рівень. Також без нотації далі просто неможливо пояснювати деякі аспекти роботи мережі.

1) Вхідні дані

   Зазвичай, матрицю вхідних даних мережі позначають як *X*. В контексті класифікації зображень *X* буде матрицею значень пікселів зображення. Щоб зазначити якийсь певний тестовий випадок використовується наступна нотація x^i, де *x* це піскельний вектор, а *i* – номер зображення-прикладу.

2) Ваги

   Нотація для ваги – $W^{[l]}$, де *[l]* – номер шару. Звичайна W – матриця значень ваги.

3) Байєс

   Загалом, байєс – це вектор констант, нотацією для якого є *b*.

4) Результат(лейбли): результатом роботи є лейбл, який мережа надає зображенню.

   a. Матриця лейблів – *Y*.
   b. Результат для і-го зображення позначається як $y^{(i)}$
   c. Щоб позначити передбачений результат на виході використовуємо ŷ.

5) Функція активації позначається як *f*, а її результат – *a*. Відповідно, результат ŷ дорівнює результату виконання функції активації на останньому шару.

   Ну і звичайно не варто забувати загальну нотацію:

   L – кількість шарів мережі

   m – кількість прикладів датасету

   n_y – кількість можливих класів

   n_x – розмір вхідного шару

2.6. Навчання нейронних мереж

2.6.1 Forward propagation

Ми вже розглянули з вами основні поняття, які потрібні для засвоєння наступної надважливої теми – навчання мережі.

Ми вже розглянули функції активацій, однак одних їх недостатньо щоб дійсно навчити мережу якісно виконувати певну задачу. Ми ніколи не будемо певні, який буде результат нетренованої мережі, тому ми маємо навчити мережу зіставити вхідні дані з потрібними результатами. Цей процес називається forward propagation.

Forward propagation – це процес, під час якого вхідні дані проходять через усю мережу. Відповідно під час цього проходження по мережі кожен шар приймає дані на вхід, робить свої обрахунки, та передає їх наступному шару, який робить те саме. Це відбувається поки ми не пройдемо по усім шарам до останнього, де отримаємо результат, завдяки якому ми можемо почати налаштовувати мережу і повторити усі вищезгадані кроки.

Якщо розглядати це детальніше, то ми маємо X, наша задача налаштувати мережу таким чином, щоб результат $\hat{y}$ був максимально наближений до y.

Відповідно, слід розглянути яким саме чином ми маємо знаходити цей $\hat{y}$.

$$z = Wx + b$$

$$\hat{y} = f(z)$$

Перша формула, яка називається попередньою активацією, являє собою добуток ваги та вхідних даних, до якої додається константа. Це проміжне значення, яке згодом передається до наступної формули.

Ця формула застосовує обрану функції активації бо наших обрахунків, що додає мережі нелінійності.

Відповідно, уявимо що ми маємо звичайну мережу з одним прихованим шаром, який має 2 нейрони. В такому разі ми дійсно можемо отримувати якийсь результат, однак що ж робити якщо у нас декілька шарів?

Як згадувалося раніше, в повному проході і є сутність методу forward propagation. Ми ітеративно виконується ці прорахунки для кожного шару, поступово рухаємося та передаємо результат вперед. В такому випадку, нотація формули має наступний вигляд.

$z^{[l]}=W^{[l]}a^{[l-1]}+b^{[l]}$

$a^{[l]}=g\left(z^{[l]}\right)$

Одразу зазначимо, що *l* – номер шару. Відповідно, можемо побачити, що кожен наступний шар використовує результат обробки попереднього, цей процес продовжується до самого кінця, отже $a^{nx}=\hat{y}$

Наприклад, ось так буде виглядати прорахунок forward propagation для n шарової мережі.

$z^0=W^0x+b^0$

$a^0=g^0(z^0)$ ➔

$z^1=W^1a^0+b^l$

$a^1=g^1(z^1)$ ➔

…

$z^n=W^na^{n-1}+b^n$

$\hat{y}=g^n(z^n)$

2.6.2 Cost function

Ми розглянули процес, який допомагає мережі обраховувати вхідні дані та отримувати результат, однак на практиці той результат ŷ який ми отримаємо буде дуже не точний. Навіть якщо після першого проходу ŷ буде максимально наближений до y, то варто перевірити ще раз, скоріше за все це сталося випадково.

Задача мережі після кожного проходження – обрахувати функцію втрати. Давайте дамо визначення та опис. Функція втрат (loss function/cost function) –

це математична функція, яка вимірює різницю між результатом передбачення ŷ та істинним результатом y. Інакше кажучи, прорахунок того, наскільки наш результат далекий від потрібного значення. Етап прорахунку втрат відбувається після кожного проходження, відповідно, наша задача тренувати мережу таким чином, щоб значення функції втрат поступово зменшувалося та з часом набуло мінімальних значень, що буде означати що наш результат є максимально близьким до правильної відповіді.

Нотація для функції втрати - J
Ситуація з цими функціями аналогічна до активаційних: існує багато різновидів, кожен застосовується та краще підходить для вирішення різних задач. Наприклад для регресійних задач часто використовується MSE (середнє квадратичне відхилення), а для задач класифікації, які ми власне і розглядаємо, функція Cross Entropy (перехресна ентропія).

Також треба зважати, що на вибір функції впливає не тільки вирішувана проблема, а й обрані функції активації, наприклад для бінарної класифікації, яка використовує функцію сигмоїди, найчастіше використовують перехрену ентропію

$$J(\hat{y}, y) = -\left(\frac{1}{m}\right) * sum(y * \log(\hat{y}) + (1 - y) * \log(1 - \hat{y}))$$

А для мультикласової класифікації – категоричну перехресну ентропію.

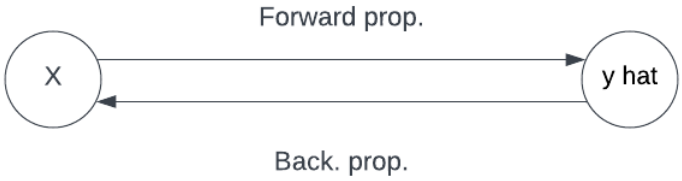
$$J(\hat{y}, y) = -\left(\frac{1}{m}\right) * sum\big(sum(y * \log(\hat{y})) + (1 - y) * \log(1 - \hat{y}))\big)$$

2.6.3 Back propagation

Але ж для чого саме ми використовуємо результат обрахунку функції втрати? Це значення нам потрібне для того, щоб відкалібрувати значення ваги та байєс.
Back propagation або метод зворотного поширення помилки – це метод навчання мережі який використовується для оптимізації навчальних параметрів мережі. Сам процес оптимізації відбувається за допомогою

алгоритму градієнтного спуску. Одразу хочу зазначити, що це є одним із ключових понять навчання, який може бути доволі складним для розуміння, тому пропоную розглянути весь процес на простому та зрозумілому прикладі.

Після проходження по мережі ми отримуємо певний результат ŷ, який фактично є спробою передбачення того, чи є навчальний приклад шуканим y. Ми вже говорили про те, що після першого проходження буде мати випадкой та, найвірогідніше, неправильний результат. Для того щоб «направити» мережу в потрібне русло ми маємо пройтися по мережі у зворотному напрямку та налаштувати значення ваги та байєсу. Відповідно, можемо сказати що зворотнє поширення – протилежна до прямого поширення операція. Якщо візуалізувати цей процес то можна створити подібне зображення.



2.9. Базовий приклад процесу навчання

Опишемо схему:

- X – вхідні дані
- Верхня стрілка – скорочений алгоритм прямого поширення, тобто це логіка отримання нашого ŷ
- Ŷ – отриманий результат після проходження
- Нижня стрілка – зворотне поширення помилки.

Маємо на вхід певні дані X, які після обробки мережею видають значення результату ŷ. Враховуючи, що це умовно кажучи перше проходження, уявімо

що ми отримали результат ŷ який дорівнює 0.213, хоча ми знаємо що y – це одиниця. Очевидно, що результат невірний, тому щоб налаштувати параметри системи ми використаємо метод зворотного поширення помилки та градієнтний спуск, задачею якого є пошук тих значень ваг та байєсу, які буду мінімізувати похибку результату мережі. Тобто, задачею зворотного поширення є ітеративний підбір таких значень, які зведуть похибку до мінімального значення, а отже, до точного результату передбачення.

Уявімо, що ми повністю пройшли у зворотному напрямку та трохи підправили значення. Далі процес повторюється знову, але на цей раз значення ŷ вже 0.365, що вже є краще ніж попередній результат. Далі знову слідує зворотне поширення.
Отже, після розгляду цієї схеми ми розуміємо, що увесь процес навчання працює циклічно, а з кожною ітерацією ми отримуємо нові та кращі значення передбачення.
Це був загальний огляд, тепер давайте пірнемо вглиб та зрозуміємо як цей процес працює у деталях.

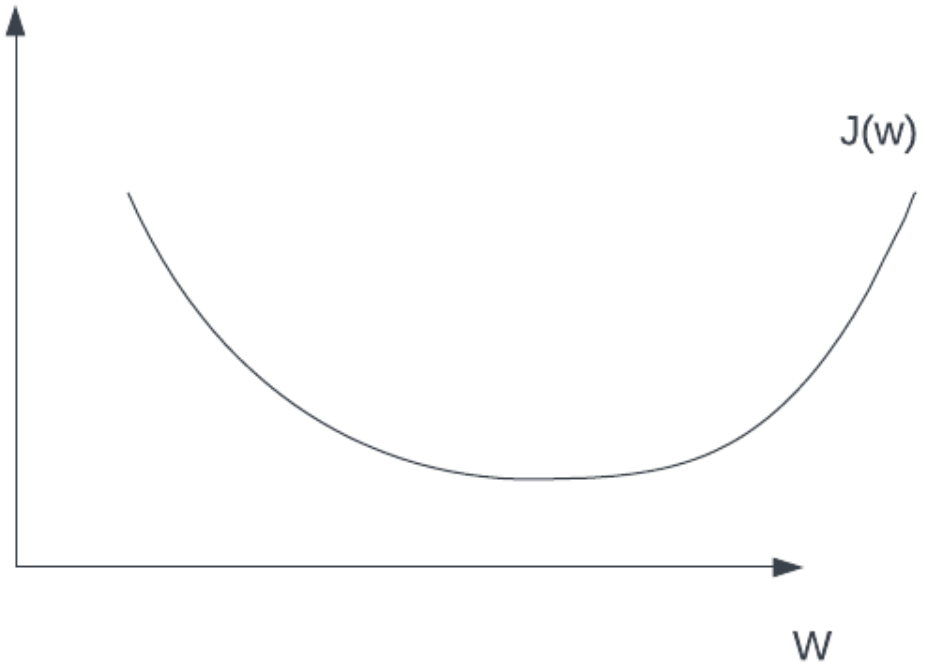
Після обрахування функції втрати наступним етапом власне є налаштування параметрів. Стандартним рішенням для цієї задачі є градієнтний спуск.

Це один з найбільш популярних та поширених шляхів оптимізації мережі, сутність якої полягає в знаходженні таких параметрів, які мінімізують функції втрат та знаходять оптимальні значення.

Щоб зрозуміти те, що робить цей метод пропоную розглянути простий приклад. Уявімо, що ваш друг прийшов з магазину, показує вам нові навушники і питає «Як думаєш, скільки вони коштують?». Можливо ви не зовсім розумієтеся на техніці, тому вирішуєте просто назвати довільне число, наприклад, 3500. У відповідь ви чуєте: «Та ні, це дуже багато». Добре, якщо це багато то візьмемо значення менше, хай буде 2000. Друг відповідає: «Ну а це дуже мало, вони дорожче». Такі здогадки будуть відбуватися до того

моменту, поки ви не вгадаєте справжню ціну придбаних навушників. Саме такий принцип роботи використовується щоб оптимізувати значення параметрів мережі.

Наведемо приклад на графіку.



2.10. Графік функції J(w)

Ми маємо певну функцію втрати J(W), яку ми хочемо мінімізувати для того щоб отримати оптимальне значення. Щоб зробити це, треба поступово повторювати наступну операцію, змінюючи параметр W.

$$w = w - \partial \frac{dJ(w)}{dw}$$

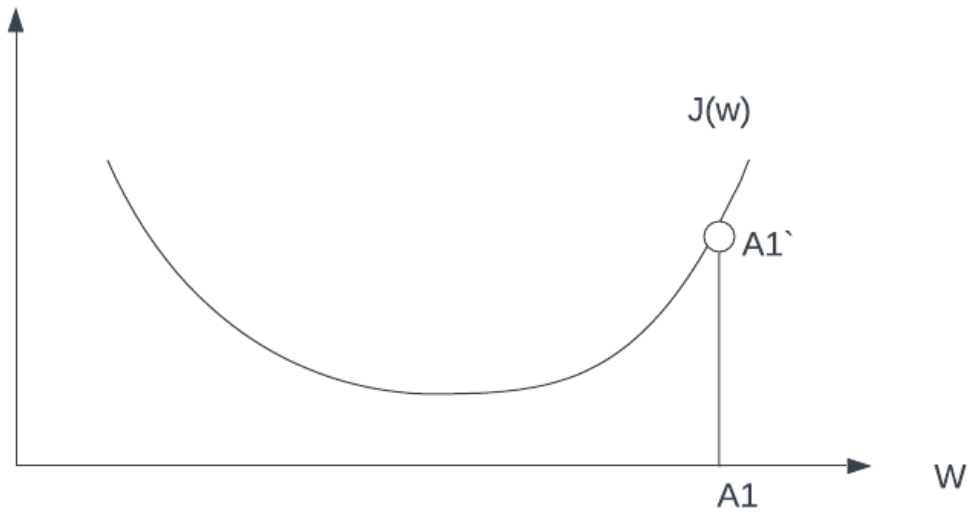
Розберемо нотацію:

w – це параметр який би будемо змінювати щоб зменшити значення функції втрати,

∂ – це гіперпараметр мережі, який означає навчальний темп, або коефіцієнт швидкості зміни. Згодом ми розберемо для чого він потрібен і як працює на

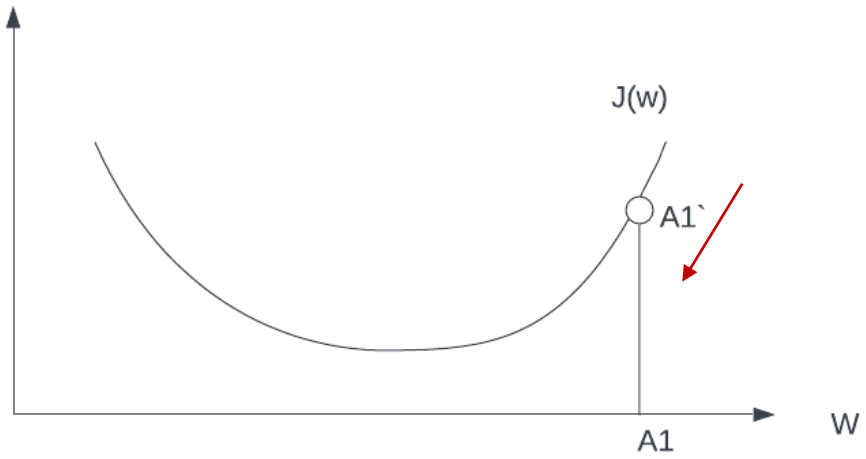
практиці

$\frac{dJ(w)}{dw}$ - це похідна, певна величина, на яку ми будемо змінювати наше значення w, у деяких конвенціях це значення також нотують як dW.



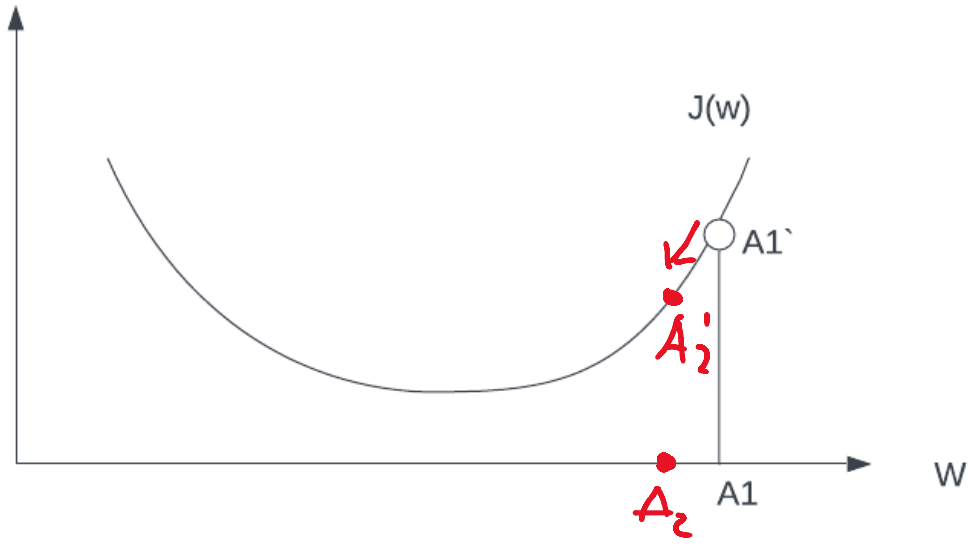
2.11. Значення ваги на графіку помилки

Припустимо, що змінна W є значенням точки A1, тоді на графіку ця точка буде A1`. Згадуємо, що в формулі змінна dW – це похідна точки A1, що є нахилом функції в якійсь певній точці. Відповідно, похідна точки A1 буде позитивною і виглядати приблизно ось так (червона лінія).



2.12. Ілюстрація похідної

Відповідно, якщо підставити похідну A1’ у рівняння $w = w - \partial \frac{dJ(w)}{dw}$ то бачимо, що значення $-\partial \frac{dJ(w)}{dw}$ буде від’ємним, тобто нове значення w буде менше за попереденє, а точка на графіку зміститься трохи лівіше, на позицію то точки A2



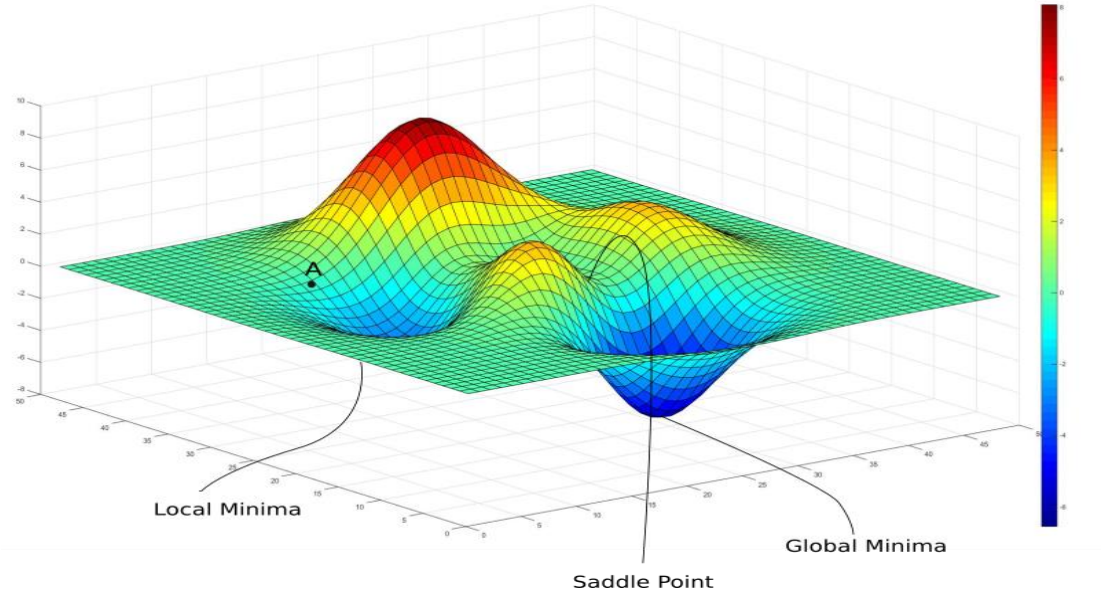
2.13. Нова точка на графіку після обрахунку спуску

Отже, якщо ми почнемо повторювати цей процес, то з часом ми будемо постійно зменшувати значення W до того моменту, поки відповідна точка на графіку не стане мінімальною. Аналогічна ситуація працює тоді, коли W замалий. Тоді похідна відповідної точки для графіку буде негативною, $-\partial \frac{dJ(w)}{dw}$ стане додатнім, а нове значення W буде більшим за попереднє і зміститься праворуч. Поступово змінюючи своє значення, в один момент ми досягаємо точки мінімуму, після якої рухатись не потрібно, адже саме в цьому місці значення функції втратиJ(w) є мінімальним, а отже, робота мережі є оптимальною.

Але що робити якщо у нас не 1 змінна, а декілька? Наприклад як ми вже говорили, ми маємо калібрувати не лише вагу, а і байєс мережі. У такому випадку формула доповнюється і отримує ще одну формулу $b = b - \partial db$,

а графік переходить до іншого вигляду, він стає тривимірним.

Наприклад ось таким.

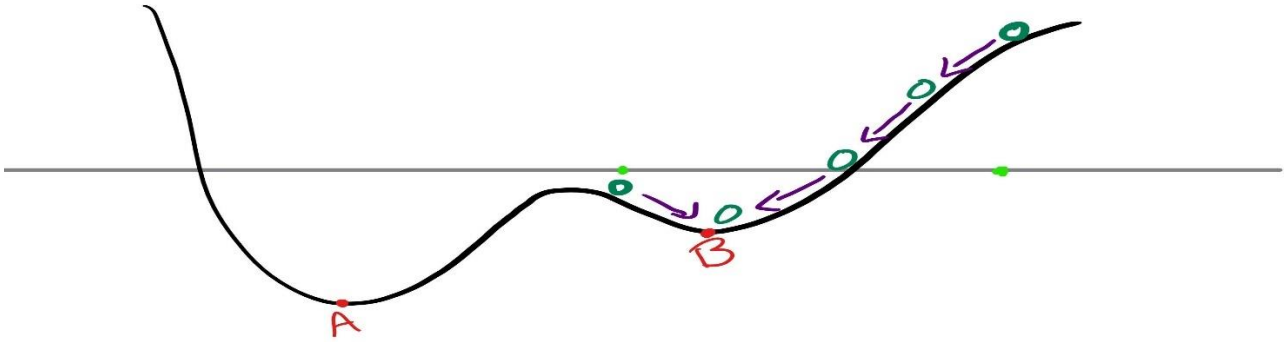


2.14.Тривимірний графік градієнтного спуску

У цій тривимірний системі логіка абсолютно ідентична. За допомогою визначення нахилу точки ми знаходимо той шлях, який веде до найбільшого спуску вниз. Загалом, під час початку роботи наші параметри налаштовані випадково, це означає що ми можемо почати шукати найнижчу позицію наприклад від точки А на графіку, а можемо і з вершини тривимірного графіку. Однак варто розуміти, що не впливає на остаточну якість моделі.

Уявімо що ми почали наш спуск і поступово дійшли до точки яку можна вважати мінімумом, але результат роботи мережі нас не влаштовує і ми вирішуємо почати знову. Раптом, після нових випадкових параметрів після процесу навчання, наш результат став значно краще, хоча саму структури моделі ми не змінювали. Чому так?

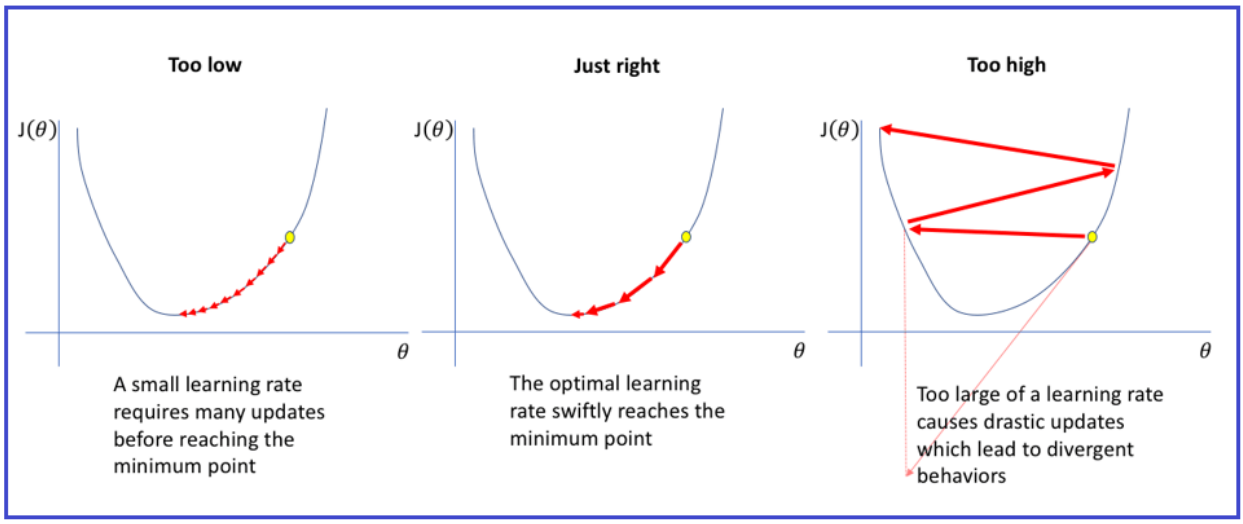
Справа в тому, що градієнтний спуск не гарантує того, що ви потрапите в точку глобального мінімуму. Інакше кажучи, графік може виглядати дуже дивно і в залежності від початкової точки, оновлення параметрів може привести вас до різних мінімумів.



2.15.Приклад локального та глобального мінімуму

Якщо поглянемо на графік вище, то починаючи з точки ми прийдемо то локального мінімуму B, що звісно покращить результат. Однак, якщо б наші параметри випадковим чином поставили б точку на протилежний бік графіку, то є більша вірогідність що ми б потрапили на точку глобального мінімум A, значно нижче, а отже і якість роботи нашої мережі була б набагато більше, адже значення втрати там мінімальне.

Також, міркуючи про градієнтний спуск варто пам'ятати про гіперпараметр ∂. Як ми зазначали вище, це константа, яка означає навчальний темп мережі. Насправді це дуже легко зрозуміти. Підстановка різного значення ∂, прямо впливає на наш крок до мінімуму на графіку. Звичайно ми можемо його змінювати і отримувати різні результати. Наприклад, підставляючи більший ∂ ми отримуємо більш різкий та великий крой по графіку, а використовуючи менше значення – плавний та повільний. Приклад можна побачити на наступному зображенні.



2.16.Приклад різних темпів навчання

Перший приклад демонструє, що при маленькому кроці ми маємо зробити багато ітерацій що досягти мінімальної точки, що звичайно забере більше часу. Третій малюнок показує що з завеликий темп може привести до хаотичної зміни параметрів, змушуючи точку постійно стрибати по різним сторонам графіку. Варто знаходити золоту середину, зазвичай це значення дорівнює 0.5, але ніхто не гарантує що саме цей темп принесе найкращий результат. Цілком можливо що під час хаотичних стрибків точка випадково вистрибне з одного локального мінімуму в зовсім інший, який може бути менше, а може і більше.

Отже, ми розібралися на прикладі як саме працює градієнтний спуск та як він знаходить нові параметри. Тепер варто зрозуміти, як саме це відбувається для усієї мережі в рамках методу зворотнього поширення помилки.

Давайте знову поглянемо на формули:

$$w = w - \partial dw$$

$$b = b - \partial db$$

Ці формули обраховують нове значення для окремої ваги та окремого баєсу мережі, але це очевидно, що цих значень у мережі можуть бути тисячі, якщо не десятки тисяч. Перше що може спасти на думку це просто циклічно обраховувати кожну вагу та баєс, але це є поганою практикою, яка є дуже не ефективною як за часом, так і за ресурсами машини. Відповідно, ми маємо звести кількість операцій до мінімальної кількості, наприклад, по одній операції на кожну формулу.

Повернемося до самих параметрів ваги та баєсу. Справа в тому, що під час роботи з мережею ми зберігаємо їх у досить простих зрозумілих варіантах.

Ваги зберігаються у матрицях, розмірність яких залежить від кількості нейронів поточного шару і минулого. Наприклад, ми маємо мережу яка складається з 2 вхідних нейронів, прихованого шару з 3 нейронами та одного нейрону на вихід. Тоді ми маємо 2 матриці: для ваги між нульовим та першим шаром це матриця W1, яка має 3 рядки та 2 колонки, по одному рядку на нейрон поточного шару та колонці на нейрон минулого. Отже, W2 буде мати вигляд в 1 рядок, та 3 колонки. Відповідно: W1[3,2], W2[1,3]. З баєсами все простіше, ми маємо один баєс на увесь шар, тому це фактично вектор, який має стільки значень, скільки нейронів у шарі.
Отже фактично, для кожного шару мережі ми маємо власну матрицю W та вектор b, початковий шар також має значення, які позначаються як X. Такий формат збереження обумовлений тими самими причинами: обраховувати кожне окреме значення – довго і дорого, тому ми оперуємо матрицями і векторами, які одразу обраховують потрібні нам результати набагато швидше ніж робити це вручну.
Після цього важливо зауваження, розглянемо загально сам процес, за допомогою якого оновлюються параметри мережі під час зворотного поширення помилки. Для того, щоб виконувати обрахунок нових параметрів в один крок, ми маємо також сформувати матриці dW та db. Виконується це

під час зворотного проходження по мережі, використовуючи ланцюгове правило.

Після прорахунку градієнтів мережі, ми можемо використати отримані матриці для налаштування значень матриці W та вектору b для кожного шару почергово за допомогою вищезгаданих формул.

$W^n = W^n - \partial dW^n$

$b^n = b^n - \partial db^n$ ➔

$W^{n-1} = W^{n-1} - \partial dW^{n-1}$

$b^{n-1} = b^{n-1} - \partial db^{n-1}$ ➔

…

$W^1 = W^1 - \partial dW^1$

$b^1 = b^1 - \partial db^1$

На цьому проходження зворотного розповсюдження помилки завершується, параметри налаштовані згідно обрахованим градієнтам, а при наступній ітерації нас чекають нові значення.

Отже, у цій об’ємній частині роботи ми розглянули базовий алгоритм того, як саме тренується нейронна мережа. Це ітеративний процес повторення одних і тих самих комплексних дій: спочатку йде пряме походження для отримання результату мережі, згодом після цього обраховується помилка мережі та відбувається зворотне поширення. Після виконання певної кількості ітерацій цього циклу ми отримуємо натреновану мережу, яка власноруч обраховувала та отримувала результат, аналізувала його та вчилася на своїх помилках.

2.7 Висновки розділу

В цьому розділі було розглянуто фундаментальні знання про нейронні мережі, які потрібні для розуміння теми роботи. Оглянуто загальний принцип роботи, структуру мереж, введено поняття нейрону, зв’язку, шарів та наведено приклади найпримітивніших нейронних мереж з різною кількістю шарів.

Відповідно, в розділі розкрито поняття параметрів нейронної мережі, такі як ваги та баєс, їх взаємодія та вплив на результат обрахунку. Також було оглянуто роль функцій активації, найпопулярніші та найвідоміші функції, їх поле застосування, переваги та недоліки кожної перерахованої функції, роль нелінійності функції.

Важливою частиною розділу був опис процесу навчання мережі, а саме комплекс методів прямого та зворотного поширення мережі. Було детально розглянуто як саме мережа генерує результат, як вона оцінює свій результат на прикладі класифікації та найважливіше, яким чином мережа аналізує результат та використовує цей аналіз для налаштування параметрів навчання мережі. Просто та зрозуміло описано що таке градієнтний спуск та яку роль він відіграє під навчання мережі.

Розділ 3: Класифікація зображень та нейронні мережі як вирішення

3.1 Чому саме нейронні мережі?

Ми вже розглянули з вами базову інформацію щодо того, як працюють банальні нейронні мережі. З висновків минулого розділу розуміємо, що це система яка приймає на вхід певні дані, вміє їх обраховувати та аналізувати результати для свого поточного навчання. Зіштовхуючись з потребою вирішення проблеми класифікації, ми можемо підібрати багато варіантів її вирішення. Наприклад, ми маємо розплідник, в якому проживають 2 види вівець. Під час їх транспортації ми хочемо відділити один вид від іншого. Для цього можна найняти 10 людей, які за зовнішніми ознаками будуть сортувати тварин до різних приміщень. До недавнього часу це був єдиний варіант вирішення проблеми, адже хто як не людина або спеціально натренований пес зможе розрізнити стадо. Однак з часом ставало зрозуміло, що людей та службових псів треба спочатку навчити розрізняти тварин, вони фізично не мають можливості працювати в потрібному темпі та можуть виникнути інші фізіологічні обставини як старіння та хвороби, які безпосередньо впливають на ефективність роботи.

Нейронна мережа є ідеальним помічником для вирішення подібних задач з наступних причин:

- ми не маємо вчити кожного окремого працівника, натомість ми будемо одну нейронну мережу. Можливо навчити одного фахівця буде швидше ніж оптимально натренувати мережу, але таких фахівців може бути багато і тому час, який ми витратимо на підготовку персоналу є непередбачуваний
- мережа навчається самостійно, все що потрібно це підібрати потрібний набір даних та налаштувати структуру мережі
- робота мережі не залежить від зовнішніх змінних як погода, хвороби тощо.

Відповідно, враховуючи всі вищенаведені факти, для подібної задачі підприємства нейронна мережа є дуже влучним вирішенням.
Даний приклад показує лише одну з можливих задач класифікації, яку можна вирішити побудував нейронну мережу, на практиці майже в кожній сфері, де потрібен певний візуальний аналіз даних, може бути застосований подібний підхід: виявлення хвороби по знімку, фільтрація даних, пошук певних деталей на зображені тощо.

3.2 Види мереж для класифікації зображення

Ми зрозуміли чому нейронна мережа є оптимальним вибором для проблеми класифікації, але тут знову постає ще одне питання: яку саме нейронну мережу слід обрати для вирішення цієї проблеми?

Насправді, видів нейронних мереж існує дуже багато, кожна з яким розроблена на розв’язок конкретної задачі. Головною проблемою в контексті роботи є саме класифікація зображення. Для її вирішення можна застосувати безліч різних підходів.

Наприклад FNN – нейронні мережі прямого поширення, найбільш простий приклад мережі, який може бути використаний. Всередині мережі прямого поширення дані проходять в одну сторону з вхідного шару до вихідного без надлишкових циклів та зворотних проходжень. Це проста мережа, принцип якої легко зрозуміти і яка може зробити певне передбачення, але цей вид мережі здебільшого підходить для роботи зі структурованими даними, де кожна кожна окрема частина незалежна від іншої. Незважаючи на можливість використання цього типу, ця мережа просто не налаштована на роботу зі структурою зображення, тому результат може бути непередбачуваним а навчання занадто довгим.

З іншого боку ми маємо RNN – рекурентну нейронну мережу, які вдало працюють даними різного розміру та вміють зберігати, а потім і використовувати залежності. Однак ця мережа також має проблему, яка пов’язана з багатомірністю даних. Незважаючи на те, що рекурентна

мережа може розуміти зв’язки та залежність даних, вона не зовсім вдало будує їх для окремих пікселів у просторі, що може привести до неможливості відрізнити якісь специфічні патерни зображення.

Тому для вирішення задачі класифікації зображення була створена та широко розповсюджена CNN або згорткова нейронна мережа.

Принцип її роботи наслідує біологічний процес роботи зорової системи живих огранізмів, де окремі нейрони реагують не на повне зображення, а лише на певні його частини, які мають конкретні патерни. Якщо пояснити це простими словами, то нам не треба бачити птаха повністю, щоб зрозуміти що це птах, нам достатньо побачити крило, дзьоб або пір’я. Цієї частини достатньо щоб зрозуміти яке створіння знаходиться перед нами.

Згорткова мережа має власну специфічну архітектуру, яка допомагає знаходити певні патерни на частині зображення та поєднувати їх у більш комплексні фігури, контури об’єктів тощо. Також цей тип мережі дозволяє зменшувати кількість параметрів за допомогою операції згортки, що дозволяє будувати мережу глибокою, але не перевантаженою Враховуючи, що згорткові мережі є оптимальним рішенням проблеми класифікації зображення, пропоную перейти до їх розгляду в наступному розділі.

Розділ 4: Згорткові мережі
4.0 Що таке згорткова мережа

На початку цього розділу ми введемо поняття, які характерні для базової архітектури згорткових мереж щоб мати розуміння, про що буде йти мова далі.

Стандартна згорткова нейронна мережа складається з наступних шарів:

- згорткові шари
- агрегувальні шари / шари пулінгу
- повноз’єднані шари

Кожен шар відповідає за конкретну задачу під час роботи мережі та має власний алгоритм роботи. Варто зазначити, що кількість шарів може бути різною та залежить від налаштувань мережі, але загалом структура завжди приблизно однакова.

Якщо описати процес класифікації у декількох словах, то спочатку ми конвертуємо зображення в потрібний для роботи формат та передаємо дані на вхід. Далі вони проходять через згорткові шари, які визначають риси та патерни зображення за допомогою фільтрів, а далі отримані дані передаються до агрегувального шару, який оброблює знайдені дані за певним алгоритмом їх щоб зменшити багатомірність та кількість. Після агрегувального шару нас чекають повноз’єднані шари, які саме і проводять аналіз та роблять припущення, що знаходиться на зображені. Якщо звести пояснення до ще більш короткої форми: згортковий шар знаходить риси, агрегувальний залишає лише найважливіші а повноз’єднаний намагається зрозуміти хто це.

Давайте поступово розберемося, що ж відбувається на кожному етапі.

4.1 Згортковий шар

4.1.1 Згортка

Згортковий шар мабуть є найважливішою частиною архітектури мережі. Саме ця частина і є імітацією роботи зорової системи, яка відповідає за знаходження певних рис зображення за якими об’єкт і буде класифіковано.

У основі роботи шару є операція згортки. За визначенням – це математична операція над двома функціями, яка дозволяє нам отримати третю функцію. Якщо розглядати процес згортки в контексті згорткової мережі, то першою функцією є вхідні дані, тобто зображення, а другою – згортковий фільтр. Після операції згортки, у якій беруть участь зображення та фільтр ми отримуємо нове значення пікселів.

Перед тим як перейти до прикладу, варто зазначити яким чином ми будемо оперувати з зображенням для згортки. Уявімо що ми маємо наступне чорно-біле зображення.

4.1.Приклад зображення для знаходження межі

На малюнку 4.1 ми можемо побачити звичайний чорно-білий прямокутник, нічого особливого. Однак, ми маємо якимось чином передати його як вхідний параметр X до нашої мережі. Справа в тому, що кожне зображення незалежно від його колірної схеми ми можемо представити у вигляді матриці. В даному випадку нехай розмірність зображення буде 6x6 пікселів, білий колір матиме позначання 1, а чорний відповідно 0. Тоді ми можемо побудувати наступну матрицю.

1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0

4.2.Матриця зображення для знаходження межі

Фактично, ми просто згенерували матрицю, де кожна комірка позначає якесь значення пікселя зображення: вся ліва частина матриці заповнена одиницями, адже там наявний лише білий колір, а вся права частина – нулями, бо права частина зображення повністю чорна.

Уявімо, що ми хочемо знайти частину зображення, де відбувається перехід з чорного на біле, тобто патерн вертикальної границі. Він представлений матрицею, яка зображена на малюнку 4.3.

1	0	-1
1	0	-1
1	0	-1

4.3.Патерн вертикальної границі

Зробимо операція згортки матриці зображення і фільтру, фактично ми маємо накласти фільтр на наше зображення та ітеративно зсувати його вправо, перемножуючи значення відповідних комірок та обраховуючи їх суму. Якщо візуалізувати цей процес то матимемо ось такий приклад.

1*1	1*0	1*(-1)	0	0	0
1*1	1*0	1*(-1)	0	0	0
1*1	1*0	1*(-1)	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0

4.4.Накладання фільтру на матрицю зображення

На малюнку 4.4 синім кольором показаний накладений на матрицю зображення фільтр, червоним – його значенням. Відповідно, ми перемножуємо усі значення та рахуємо їх суму:

1*1+1*1+1*1+1*0+1*0+1*0+1*(-1) +1*(-1) +1*(-1) = 0

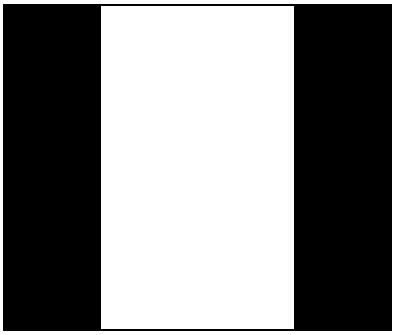
Це значення 0 є новим значенням пікселя для результуючої матриці процесу згортки. Наступний кроком ми переміщуємо фільтра на одну комірку праворуч та повторюємо обрахунки:

1	1*1	1*0	0*(-1)	0	0
1	1*1	1*0	0*(-1)	0	0
1	1*1	1*0	0*(-1)	0	0
1	1	1	0	0	0
1	1	1	0	0	0
0	0	0	1	1	1

4.5.Зміщення фільтру на матриці зображення

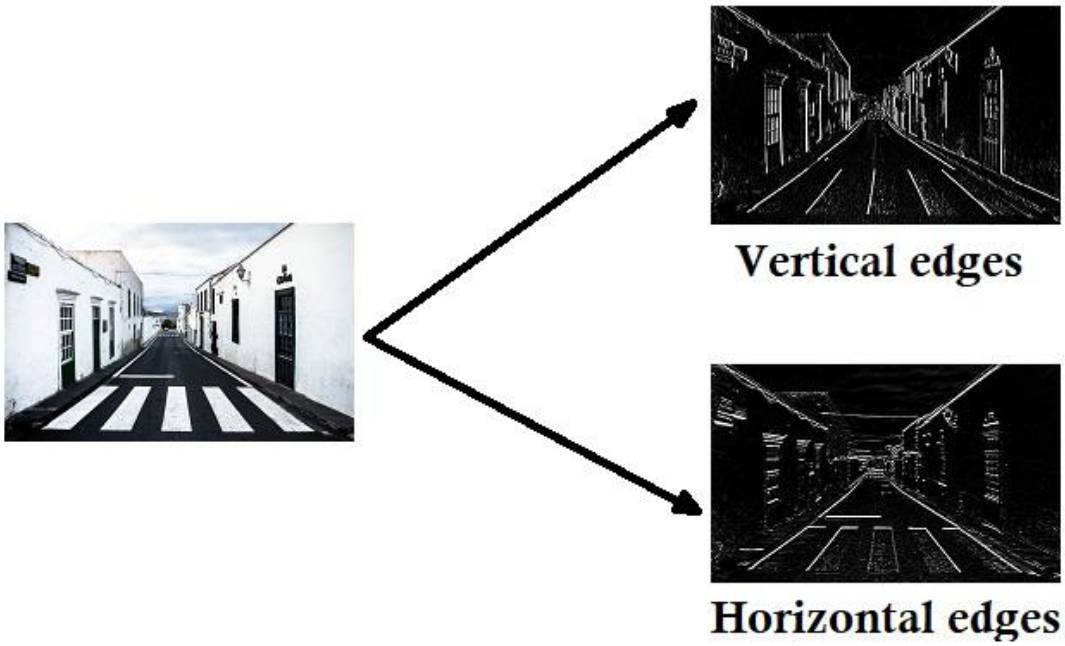
Зелений колір на малюнку 4.5 – це зміщений накладений фільтр. Після обрахунку ми отримуємо значення 3. Далі процес повторюється до кінця зображення ряду, після чого ми повертаємося в початкове положення, вміщуємося на одну комірку вниз та починаємо обрахунок знову. Після обрахунку ми отрмаємо наступну матрицю, яка відповідає наступному зображенню. Фактично, наш фільтр зміг знайти вертикальну границю по центру зображення.

0	3	3	0
0	3	3	0
0	3	3	0
0	3	3	0



4.6.*Результат згортки у вигляді нової матриці та візуалізації його значень*

Це була демонстрація лише одного фільтру, на практиці таких фільтрів є значно більше.



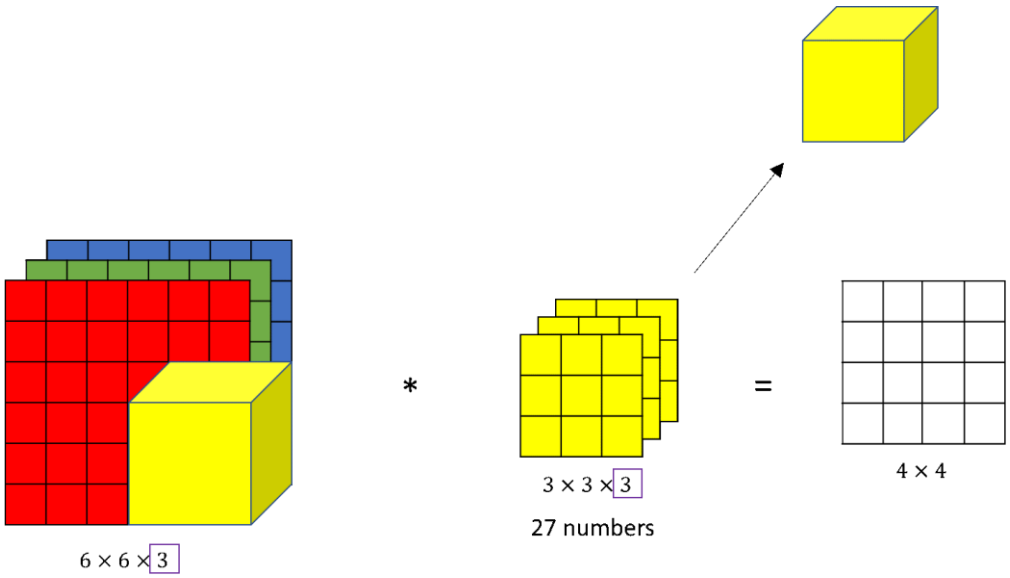
4.7.*Приклад накладання 2 різних фільтрів на зображення*

На прикладі малюнку 4.7 ми можемо порівняти, як саме працюють різні фільтри на одному й тому самому зображенні. Вертикальний фільтр вдало знаходить вертикальні межі зображення, а горизонтальний навпаки – майже ігноруючи вертикальні межі, знайшов усі горизонтальні. На практиці, кожен згортковий шар має у собі безліч різноманітних фільтрів, які будуть шукати на вхідному зображенні абсолютно різні патерни та передавати їх далі. Розмірність результату прямо залежить від таких значень як розмір зображення, розмір та кількість фільтрів. Тобто якщо на нашому прикладі ми мали зображення розміром 6 на 6 пікселів та один фільтр розміром 3 на 3, то ми отримуємо результат який за розміром є матрицею 4 на 4. Якщо занотувати це то отримуємо наступну формулу: $NxNx1 \rightarrow FxFx1 \rightarrow (N-F+1)x(N-F+1)x1$,

де N це розмір початкового зображення, F – розмір фільтру. Враховуючи, що ми мали лише один фільтр, то результат також буде один, але на практиці фільтрів може бути C, тому застосовуючи C фільтрів ми отримуємо і C результатів.

Наступним питанням, яке потребує розгляду є те, яким чином зготкові шари працюють з мультиканальним зображенням або, інакше кажучи, кольоровим. Адже на наведеному прикладі наше зображення мало лише один канал, а зазвичай зображення мають формат RGB, тобто 3 канали: червоний, зелений та синій.

Якщо ми маємо кольорове зображення в три канали, то відповідно ми матимемо 3 матриці які будуть представляти наше зображення: матриці червоного, зеленого та синього спектрів. В такому випадку кожен фільтр також буде мати стільки вимірів, скільки каналів має зображення.



4.8.Приклад роботи з триканальним зображенням

На ілюстрації ми можемо побачити, що кольорове зображення розміром 6 на 6 пікселів має 3 канали, а отже і три матриці, які представляють це зображення у різних спектрах. Відповідно, фільтр також має 3 виміри для матриці кожного каналу зображення. Власне цей тривмірний фільтр накладається одночасно на усі виміри матриць зображення, а результатом одної ітерації є сума усіх сума добутків 27 значень накладання, це можна побачити на ілюстрації, де фільтр у вигляді куба накладається на зображення. Після завершення ітерацій результатом згортки є лише одна матриця, розмір якою визначається за вищенаведеною формулою.

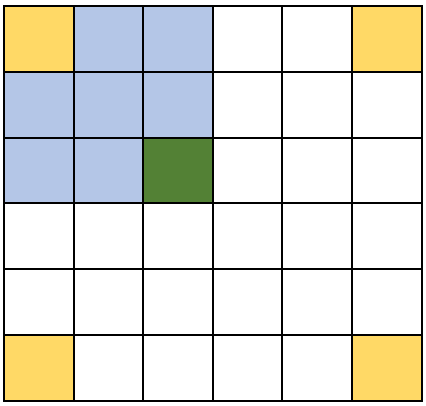
Також варто зазначити, що розмір рецептивного поля фільру не є сталим, його розміри також можна регулювати так, як ми захочемо. Однак загальною практикою є такі налаштування фільтру, де його розмір має не парне значення, наприкла 3 на 3, 5 на 5 або 7 на 7 пікселів. Не парність розміру обумовлена тим, що таким чином ми можемо виділити центральний піксель на рецептивному полі.

4.1.2 Padding – просторове доповнення

Під час процесу згортки ми можемо отримати ряд проблем, які можуть значно вплинути на результат.

По-перше, під час процесу згортки результатом є матриця, яка менше за розміром за вхідне зображення, наступній ітерації результат буде ще менший тощо. Відповідно, якщо результат, тобто цифрова матриця, буде постійно зменшуватись, то з часом ми можемо втратити інформацію про певні риси, які наявні на зображенні.

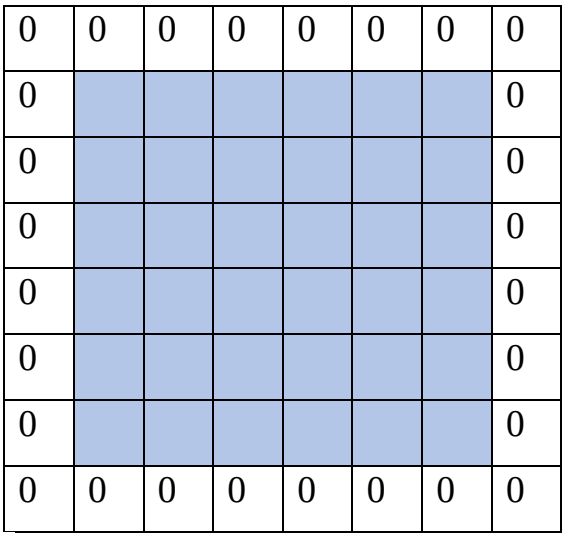
По-друге, під час проходження фільтру значення деяких пікселів можуть оброблюватись набагато менше разів у порівнянні з іншими. Наведемо приклад:



4.9.Приклад матриці, пікселі якої оброблюються нерівномірно

На малюнку 4.9 можна побачити, що ми маємо зображення 6 на 6 пікселів, представлене у вигляді матриці. При накладанні фільтру (виділено блакитним), він проаналізує жовту комірку лише один раз, адже фільтр постійно рухається і ніколи не повернеться на ту ж позицію. Порівняно з жовтою коміркою, зелена буде обрахована набагато більше разів, їй буде приділено більше уваги. Відповідно, якщо на жовтих комірках матриці будуть знаходитися якісь риси, які треба знайти, ми не можемо гарантувати, що фільтр це зробить ЛИШЕ за один прохід. На матриці можемо побачити, що ця проблема актуальна для усіх кутових комірок зображення.

Для того щоб вирішити обидві проблеми ми можемо застосувати наступний метод: доповнимо нашу матрицю зображення та додамо рамку, значенням усих комірок рамки буде 0.



4.10.Початкова матриця з просторовим доповненням

На ілюстрації рамка – просторове доповнення матриці, а синє поле – матриця початкового зображення. Відповідно, якщо ми застосуємо фільтр 3 на 3 для нової матриці з просторовим доповненням то в результаті ми отримаємо нову матрицю, розмір якою буде 6 на 6 пікселів. Тобто, ми дещо гальмуємо процес , який змушує наш результат з часом втрачати дані. Також з просторовим доповненням вирішується друга проблема: враховуючи те, що тепер кутові комірки представленні доповненням, яке фактично не несе ніякого змісту зображення, загальна кількість потрапляння істинних кутів зображення в межі фільтру також зростає. На практиці існує декілька типів просторового доповнення:

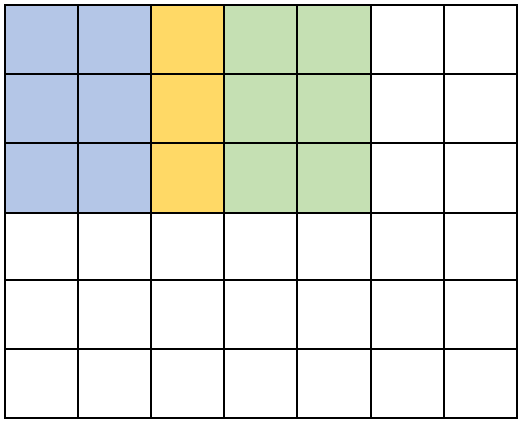
- Valid padding – практично, це робота моделі, коли просторове доповнення відсутнє.
- Same padding – цей вид просторового доповнення полягає у створенні рамки навколо зображення таким чином, щоб при застосуванні операції згортки ми отримали результуючу матриці розміром з істинну матрицю зображення. Приклад такого доповнення був наведений вище: ми

додали рамку з нулями навколо істинної матриці 6 на 6. Після процесу згортки просторово доповненої матриці та фільтру ми також отримуємо результуючу матриці розміром 6 на 6. В залежності від початкового розміру зображення кількість шарів та глибина рамки може також бути різною.

- Casual padding – доволі специфічний вид доповнення, який застосовують під час роботи з одновимірними згортковими шарами. Використовується цей підхід під час аналізу певної послідовності даних, які були отримані протягом певного часу.

4.1.3 Stride – зсув

На усіх вищенаведених прикладах ми демонстрували як фільтр поступово рухається на 1 піксель вправо. Насправді, величина кроку не обов'язково має бути одиницею, це гіперпараметр мережі, який також можна налаштувати. Називається цей параметр зсувом. Наведемо приклад.

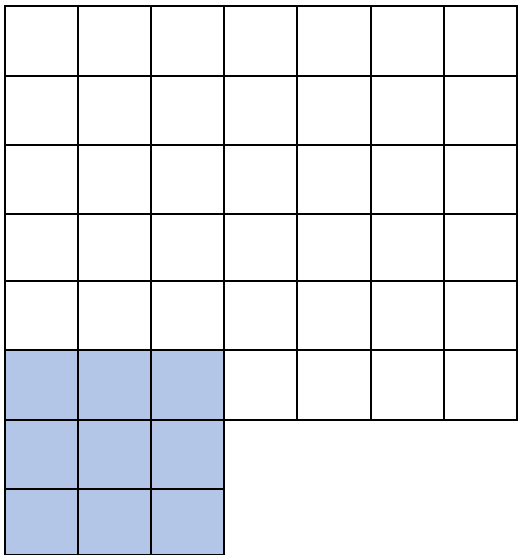


4.11.Приклад зсуву в 2 пікселі

Параметр зсуву на зображенні 4.11 дорівнює 2. Ми можемо побачити, як початкова позиція фільтру (синій колір) після обрахунку змістилася вправо на 2 пікселі (зелений колір), жовтим показано перетин фільтрів на матриці. Після повного проходження фільтру до краю, він повертається на початку

позицію рядку та вміщується вниз на 2 пікселі. Таким чином операція повторюється до самого кінця.

Однак можлива ситуація, коли при встановленому параметрі зсуву фільтр буде змушений вийти за межі зображення.

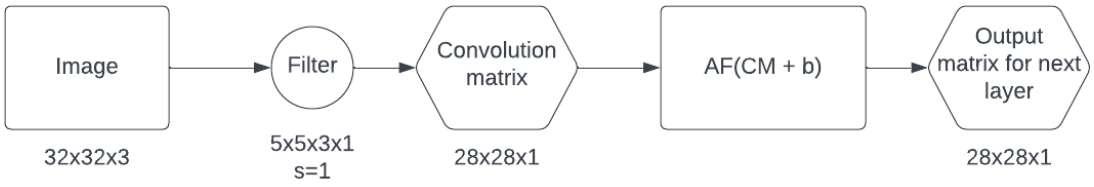


4.12. Приклад виходу за рамки зображення при зсуві фільтру

На цьому прикладі ми бачимо, що при зсуві в 2 пікселі та розміру матриці 6 на 6, під час переходу на останній ряд наш фільтр виходить за межі. Внаслідок цього увесь останній ряд матриці не буде зарахований до результуючої матриці. Таким чином, створений результат фактично буде мати меншу роздільну здатність, адже частина пікселів просто не потрапила до фільтру. Загалом, такої поведінки варто уникати, адже не захоплені фільтром рядки можуть містити корисну інформаці, однак найчастіше під час обробки зображення для його передачі відбувається приведення зображення до квадратної форми, що мінімізує подібні випадки. Також, якщо раптом сталася подібна ситуація, варто переглянути чи дійсно ви хочете щоб ваші гіперпараметри зсуву та величини фільтру були налаштовані саме таким чином. Варто розуміти, що нелогічні значення параметрів можуть спричинити втрату даних, та як наслідок, непрогнозовану роботу та навчання мережі.

4.1.4 Архітектура згорткового шару

На практиці, кожен згортковий шар складається з рецептивного поля – фільтру та активаційної функції. Активаційна функція потрібна для згорткового шару для того, щоб отримати нелінійний результат обрахунків. Завдяки нелінійності ми можемо аналізувати більш складні та комплексі патерни зображення, проводити залежності між певними окремими патренами тощо. Якщо пропустити цей етап або використовувати лінійну функцію, то внаслідок ми отримаємо результат, але він може не відобразити наявність певного складного взаємозв'язку між пікселями.



4.13. Приклад архітектури згорткового шару

На зображенні вище можна побачити стандартну архітектуру згорткового шару. Після операції згортки над зображення 32x32 в три канали та одним фільтром 5x5 ми отримуємо нову матрицю розміром 28x28 в три канали. Згорткову матрицю одразу ж подаємо до функції активації, додаючи байєс. Після роботи функції ми отримуємо нову матрицю того ж розміру, який маємо передати наступному шару.

4.2 Pooling - Агрегувальні шари

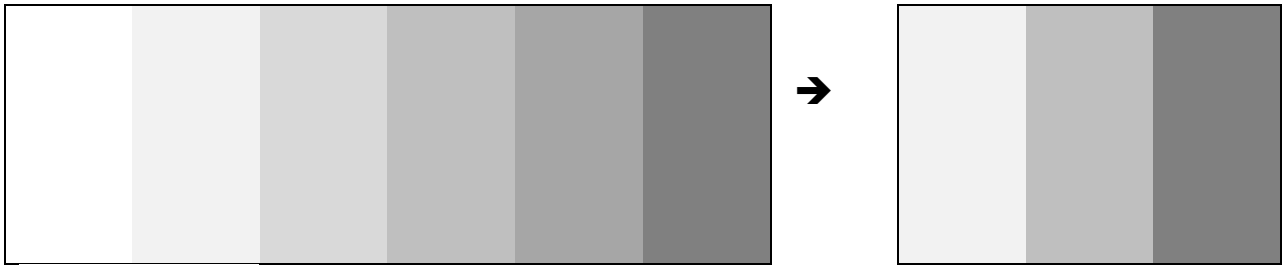
Наступним важливим концептом згорткових мереж є агрегувальні шари. Це шари мережі, які йдуть слідом за згортковими шарами та допомагають пришвидшити роботу мережі оперуючи результатами згортки. Давайте розглянемо навіщо вони потрібні та як вони працюють.

Як ми вже знаємо, згортковий шар відповідає за виявлення особливих рис та патернів зображення. Однак тут ми зіштовхуємося з декількома обмеження.

По-перше, якщо ми працюємо з великою кількістю фільтрів та великим зображенням то в результаті ми будемо отримувати велику кількість деталізованих матриць згортки, кожна з яких матиме певну інформацію про наявність певної риси в межах її рецептивного поля. Це великий об’єм інформації, який потребує значний ресурсів системи для обробки. Відповідно, нам потрібно певним чином зменшити розмір матриць згортки. Першим що може прийти в голову, це використати більший зсув, адже як ми з вам говорили, більший зсув теоретично може понизити роздільну здатність зображення, а отже і обрахувати її буде простіше. Однак, цей метод є сумнівним, адже можливо ми хочемо детально пройти по усьому зображенню з більш малим зсувом або розмірність матриці може не зовсім підходити від того зсуву, який зменшить результат до потрібних нам значень.

По-друге, кожна матриця згортки детально «запам’ятовує» де саме знаходяться патерни, тому якщо нам треба якимось чином змінити матрицю це може спричинити втрату даних про патерни на цьому окремому полі.

Найкращим вирішенням цієї проблеми є агрегувальні шари. Їх мета полягає у зменшенні розміру даних для обрахунку, що відповідно зменшує кількість потужності, яка потрібна для роботи з даними. До того, пулінг не тільки зменшує об’єм даних, а й виокремлює та підсилює знайдені патерни шляхом пониження роздільної здатності, що вирішує другу проблему та позбавляє нас у потребі використовувати великий крок зсуву. Сам процес полягає у тому, щоб накласти на матрицю згортки нове поле, яке знайде ті значення, які найбільш точно передають інформацію про патерн, який на цьому полі знаходиться.



4.14.Приклад результату роботи агрегувального шару

На ілюстрації 4.14 ми можемо побачити приклад зображення після пулінгу. Фактично, нове зображення має менший розмір та містить окремі визначальні риси, які загально характеризують якусь певну область лівого зображення. Хоча воно і менше, але за допомогою зменшення надлишкових деталей ми змогли виокремити загальний патерн початкового зображення.

Існують декілька варіантів пулінгу, зараз ми розглянемо принцип роботи агрегувального шару використовуючи метод який називається max pooling.

Уявімо, що ми маємо наше зображення приклад у вигляді матриці, комірки якої будуть заповнені значеннями від 0 до 5, де 0 – білий, а 5 – найтемніший сірий на зображенні. Отримуємо наступну матрицю, зображену на малюнку 4.15.

0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5

4.15.Матриця зображення зі значеннями пікселів

Тепер ми маємо накласти поле пулінгу розміром 2 на 2 та зі значенням зсуву 2 пікселі. За методом максимального пулінгу, до результуючою матриці записується лише найбільше значення кожного поля. Якщо проілюструвати це не прикладі то маємо наступне зображення.

0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5

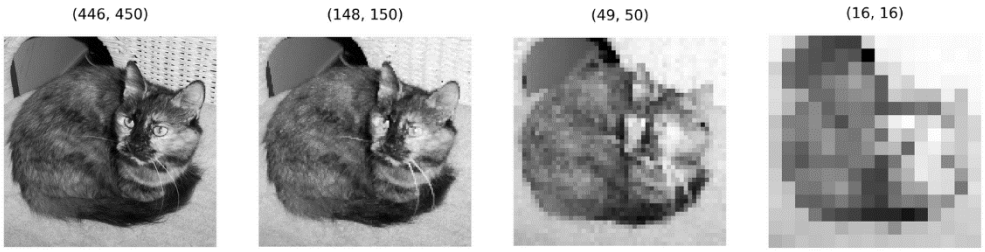
4.16.Накладання max-pooling на зображення

В першому полі найбільшим значенням є 1, в другому – 3, а в третьому – 5. Після закінчення ітерацій ми отримуємо наступне зображення.

1	3	5
1	3	5
1	3	5

4.17.Результат роботи агрегувального шару

Як можемо бачити, до фінального зображення, яке представлене матрицею, були записані лише найбільші значення кожного поля, розмір матриці зменшився, але загальний вигляд патерну не змінився. Звичайно, що плавний перехід кольорів зник, але загальна логіка поступово темнішення була збережена та проілюстрована. Наведемо приклад на більш складному зображенні.



4.1817.Приклад декількох зображення після декількох ітерацій пулінгу

Ми бачимо, що початкова картинка кота є детальною, але занадто великою, що може вплинути на швидкість роботи та вимоги до обчислювальної машини. Після першого проходження агрегувального шару, ми бачимо, що розмір зображення зменшився в 3 рази, хоча загальна детальність залишилася на досить високому рівні. Навіть після 2 ітерації ми все ще можемо побачити не деталізований, але зрозумілий силует кота.

Це був лише один приклад роботи агрегувального шару, але насправді їх існує більше, давайте розглянемо найпопулярніші:

- Max pooling – як ми вже побачили, цей тип пулінгу обирає найбільші значення матриці, тобто найбільш яскраві пікселі. Це дуже корисно коли ми маємо зображення з темним заднім фоном, але ми хочемо відкинути його та працювати лише зі світлими деталями зображення.
- Min pooling – цей тип протилежний до попереднього, він обирає найменші значення кожного поля. Застосування може бути також абсолютно протилежне: наприклад якщо ми хочемо працювати лише з темними ділянками фотографії.
- Average pooling – цей тип записує до матриці середнє значення усіх комірок поля. Чіткі межі виокремити не вийде, але дуже легко можна згладити зображення, зробити його менш контрастним та м’якшим.

Відповідно, немає найкращого методу роботи агрегувального шару, кожен з них використовується для окремої задачі та обирається за потреби. Однак в

усіх методах пулінгу сенс залишається однаковий – зменшення роздільної здатності зображення без відкидання потрібних для класифікації рис та патернів.

Загалом, агрегувальні шари завжди йдуть слідом після згорткових. Таким чином кожен наступний шар отримує зображення яке є меншим від вхідного, але у якому міститься ключова інформація щодо його патернів. Також варто зазначити, що ці шари не потребують тренування, адже фактично вони просто виконують звичайну математичну операцію конвертації.

4.3 Повноз'єднана мережа

Минулі 2 підрозділи були присвячені поясненню процесу знаходження рис та патернів на зображенні, але насправді цей процес є лише підготовкою до самої класифікації об'єкту. Коли надсилаємо зображення на вхід до мережі, воно проходить якусь певну, завчасно сконструйовану, кількість згортальних та агрегувальних шарів. Єдине що відбувається на цьому етапі – виявлення ключових рис об'єкту, які згодом будуть потрібні для його остаточної класифікації. Коли ж виявлення патернів було завершено, ми отримуємо велику кількість матриць, кожна з яких містить інформацію про певну частину зображення та наявність та характерних для шуканого об'єкту якостей. Наступним же кроком є безпосередньо класифікація зображення, мережа має проаналізувати результат роботи попередніх шарів та вирішити, що саме було представлено на початковій картинці. За процес класифікації відповідає повноз'єднана нейронна мережа.

Фактично, повноз'єднана мережа являє собою багатошаровий перцептрон, принцип дії якого аналогічний до того, який ми розглядали в другому розділі. На вхід подається результат роботи попередних шарів, далі є декілька прихованих шарів нейронної мережі а на виході ми маємо вихідні нейрони, кількість яких залежить від кількості категорій, до яких може належати об'єкт на зображенні.

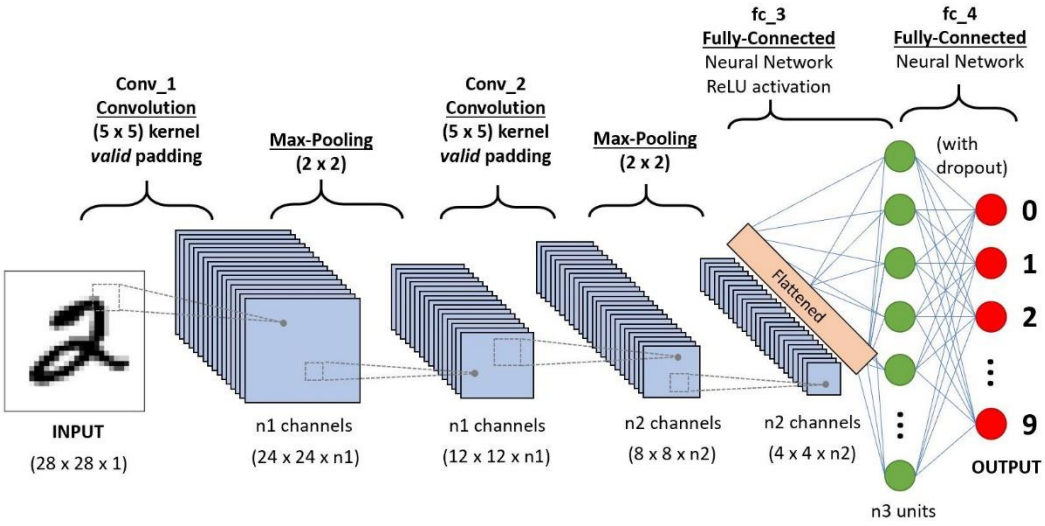
Спершу ми маємо передати вхідні данні X до мережі, але як ми пам’ятаємо, після проходження шарів згортки ми отримуємо велику кількість окремих матриць. Як ми знаємо, кожна з цих матриць представляє інформацію про окрему частину зображення, тому фактично множина матриць-результатів є обробленим зображенням. За приведення множини матриць до виду, який ми можемо передати на вхід до мережі відповідає процес, який називається згладжуванням. В рамках цього процесу ми перетворюємо всю множину матриць на один великий вектор, який містить усі значення усіх матриць-результатів. Таким чином, після згладжування ми отримуємо вектор, який є цифровим представленням обробленого зображення. В цьому форматі ми вже можемо передавати вектор X на вхідний шар мережі для обробки.

Далі відбувається той самий ітеративний комплексний алгоритм роботи нейронної мережі. Під час прямого проходження мережа оброблює вхідні данні та генерує передбачення ŷ. Використовуючи метод навчання під наглядом, ми вже знаємо чи є насправді передбачення мережі вірним, тобто чи є воно наближеним до y, чи ні. Після результатів прямого проходження ми обраховуємо функцію втрати та проходимо у зворотному напрямку калібруючи значення вагів та баєсу мережі. Таким чином мережа, отримуючи оброблене зображення на вхід, починає робити перші передбачення та вчитися на своїх помилках, а згодом намагатися знову.

4.4 Стандартна архітектура CNN

У попередніх підрозділах ми оглянули кожну складову згорткової мережі, їх задачі та принцип роботи. Зараз ми зберемо усі отримані спостереження докупи та оглянемо стандартну архітектуру згорткової мережі. За приклад

було взято мережу, метою якої є класифікація та встановлення цифр.



4.19.Приклад архітектури CNN

Давайте проаналізуємо те, що ми бачимо на ілюстрації. На вхід подається чорно-біле зображення цифри «2», яке представлене матрицею розміром 28 на 28 пікселів в один канал (адже це чорно-біле зображення).

Мережа налічує 2 шари згортки, розділені агрегувальними шарами. При передачі матриці зображення в перший шар згортки, вона оброблюється фільтрами, передається в функцію активації, а потім до агрегувального шару типу max-pooling, який зменшить розмір матриць без втрати характерних рис. Також ми можемо побачити що на зображенні відсутні важливі риси на кутових пікселях, відповідно ми можемо не використовувати просторове доповнення та проігнорувати замалу кількість обробки кутових пікселів зображення. Аналогічний процес повторюється ще раз і з кожною ітерацією розмір наших згорткових матриць зменшується внаслідок обробки агрегувальними шарами.

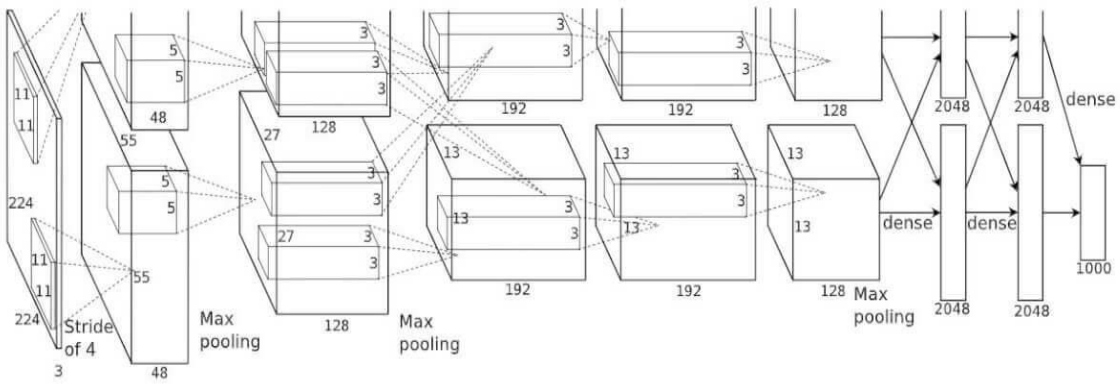
Після завершення етапу знаходження рис ми отримуємо певну кількість матриць, які підлягають процесу згладжування та передачі бо стандартної повноз’єднаної мережі. Повноз’єднана мережа, яка отримала зображення у вигляді вектору створеного з оброблених матриць, використовує дані для роботи та генерації передбачення, яким є одна з категорій (цифра від 0 до 9).

4.4.1 Популярні згорткові мережі

Вищенаведений приклад згорткової мережі лише демонструє її базову та стандартну архітектуру. На практиці, різні мережі можуть мати зовсім різну архітектуру: починаючи від кількості шарів згортки та величини рецептивного поля закінчуючи структурою повноз'єднаної мережі, кількості її прихованих шарів.

Пропоную розглянути найвідоміші приклади різноманітних нейронних мереж.

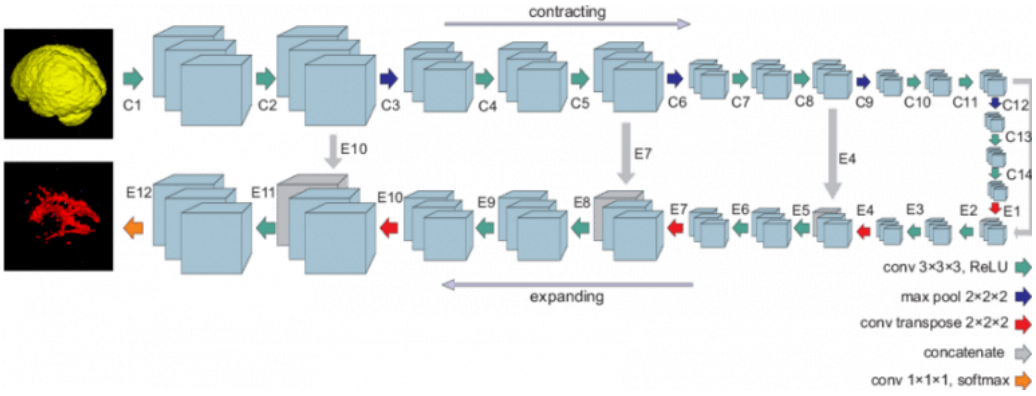
1) AlexNet – доволі популярна згорткова мережа, яка виграла конкурсу ImageNet у 2012 році зі значним відривом від опонентів. Ця мережа складається з 5 згорткових шарів, та 3 повноз'єднаних. Агрегувальні шари працюють за методом max-pooling, а загальною функцією активації упродовж усієї мережі є ReLU. На виході мережа 1000 нейронів, тобто вона спроможна розрізнити 1000 різних зображень за їх категоріями. Ця мережа є відомою тим, що в ній вперше була обрана функція ReLU, як загальна функція активації, на відміну від попередників, які використовували акртангес.



4.20 AlexNet

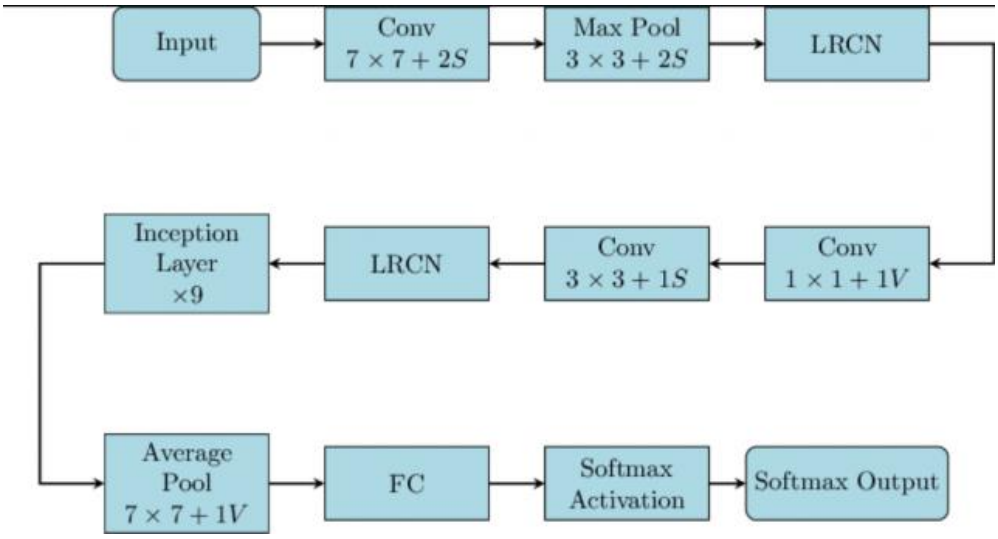
2) Unet – ця мережа була створена у 2015 році та у тому ж році також перемогла в конкурсі від компанії ImageNet. Особливістю мережі є те, що вона направлена не на повну класифікацію об'єкту на зображенні, а на сегментацію окремих областей на певні категорій. Вона поширена та

добре підходить для реальних задач, наприклад в сфері медицини. Також варто зазначити, що ця мережа потребує значно менше вхідних даних у порівняні з іншими, хоча її результат залишається доволі високим. Архітектура мережі складається зі згорткового шару для зменшення розміру та виокремлення рис зображення. Далі йде шари, які навпаки збільшують зображення з уже виділеними рисами.



4.21.Unet

3) GoogLeNet – ця згорткова мережа від компанії Google має дуже глибоку структуру, навіть серед усіх наведених вище. Ця глибина обумовлена великою кількість експериментальних технік, таких як згортковий фільтр розміром 1 на 1 піксель та глобальний пулінг середнього значення, однак варто зазначити, що вона має високі вимоги до потужності для обрахунку. Її застосовують в області класифікації зображень об’єктів на дорогах та вулицях великих міст.



4.22.GoogLeNet

4.5 Висновки розділу

Як ми бачимо, згорткові нейронні мережі мають специфічну архітектуру, яка спеціально розроблена для вирішення задачі класифікації зображень. Було розглянуто загальну структуру мережі та принцип роботи кожного з критичних шарів: згорткового, агрегувального та повноз’єднаних. Також було описано, яким чином працює кожен шар мережі, які алгоритми та підходи вони використовують, з якими проблемами зустрічаються так як їх вирішити. Також було наведено приклад стандартної мережі та декілька різновидів, які є популярними та відомими у сферах свого застосування. Маю сказати, що саме згорткові мережі стали найбільш оптимальним вирішенням проблем класифікації зображень та невід’ємним інструментом комп’ютерного зору.

Розділ 5: Попередня обробка даних

5.1 Що та навіщо потрібна попередня обробка даних

Як ми вже з вами знаємо, згорткова мережа не може просто прийняти звичайне зображення jpeg/png тощо формату. Вони потребують попередньої обробки та приведення до певного стану перед початком роботи. В контексті згорткових мереж, це не тільки створення триканальної матриці, насправді існують різноманітні методи обробки, які передують цьому процесу.

Задачею попередньої обробки зображення є підвищення загальної його якості для отримання більш точних результатів. В процесі ми маємо мінімізувати всілякі аномалії, вирівняти загальний тон, прибрати биті пікселі, тобто мінімізувати можливість некоректного сприймання даних на конкретному полі. Також під час обробки ми можемо наділити зображення певними якостями, яких потребує мережа. При якісній обробці ми отримуємо не тільки більш точне передбачення, а й пришвидшення навчання, наприклад зменшення загального розміру усіх зображень датасету значно економить час та ресурс комп’ютера. З іншого боку, за допомогою окремих методів попередньої обробки зображення, ми можемо збільшити ефективність мережі розрізняти більш складні патерни, попередньо змінивши деякі тренувальні моделі. Далі пропоную навести приклад деяких методів, які направленні на обробку зображень задля усунення помилок при навчанні.

5.2 Приклади обробки

Припустимо, що ми зібрали датасет квітів, але при детальному розгляді помітили, що на деяких зображеннях ми маємо занадто контрастні зони, які можуть спричинити неочікувану поведінку під час тренування. Тому ми маємо вирівняти загальний фон кольору зображення до одного рівня таким чином, щоб не видалити патерни квітки. Для вирішення подібних проблем є декілька способів:

Поріг – один з найбільш простих та зрозумілих методів. Ми маємо задати граничний поріг (treshhold) яскравості пікселя. Якщо значення пікселя вище

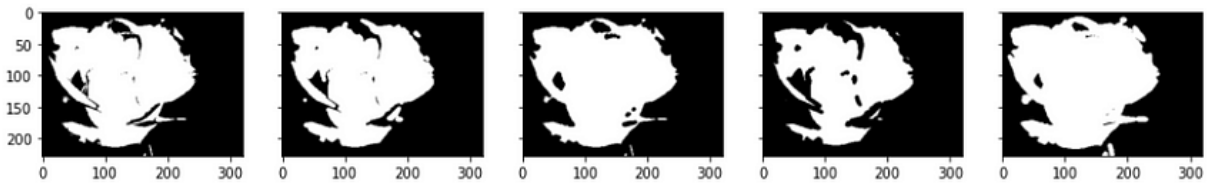
за поріг – ставимо значення одиниці, якщо ж менше – нуля. Як результат ми отримуємо чорно-біле зображення, яке все ще містить інформацію про основні межі та риси. Однак цей метод підходить не для всіх задач, адже таким чином ми фактично прибираємо мультиканальність.

5.1. Treshhold

Ерозія та розширення – це дві протилежні дії, які виконують одну функцію. Ерозія зменшує яскраві регіони зображення та збільшує темні, а розширення – збільшує яскраві регіони зображення та зменшує темні. При цьому, обидва методи можуть працювати в парі, а в залежності від їх послідовності вони називають операціями відкриття та закриття зображення.

Відкриття – це операція розришення, після якої відбувається ерозія. Внаслідок відкриття прибираються надзвичайно яскраві пікселі, а темні зони поєднуються.

Закриття ж працює у протилежній послідовності та слугує для видалення занадто темних пікселів зображення та поєднання світлих його зон.



5.2. Зображення після порогу, ерозії. розширення, відкриття та закриття (зліва направо)

Ще одним прикладом обробки зображення є його нормалізація. Ідея полягає в тому, щоб змінити значення пікселів таким чином, щоб їх значення були в

певному діапазоні. В залежності від потреби та виду зображення ми можемо обрати конкретний тип нормалізації, але загалом результатом їх роботи є або згладжене зображення, або ж зображення з вирівняним загальним фоном. В залежності від кількості каналів також можемо обрати таку формулу, яка буде працювати з чорно-білими, сірими або з кольоровими даними.

5.3 Нормалізація

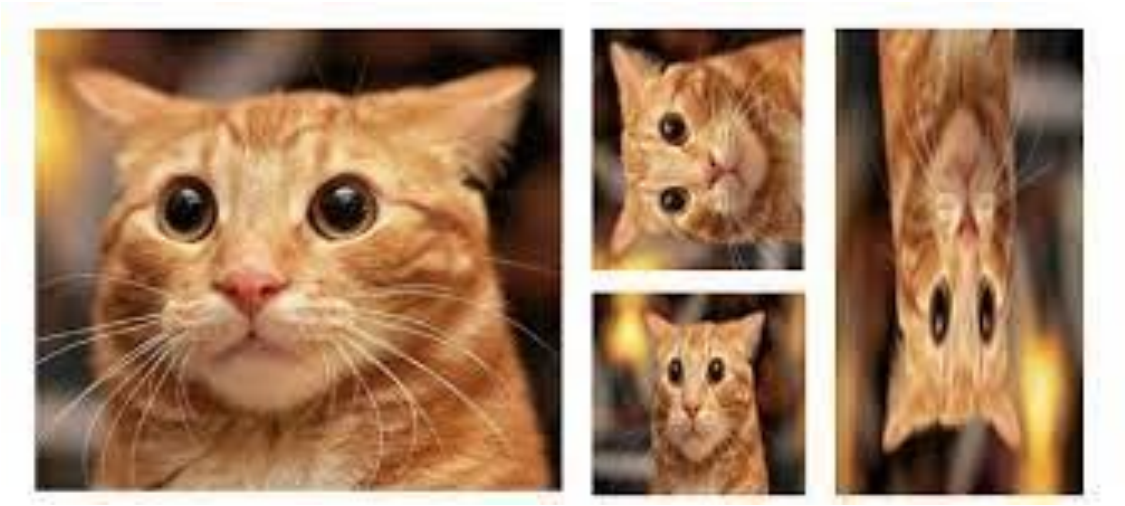
Наприклад, фото рослини ліворуч – оригінал. За допомогою нормалізації ми вирівняли тон зображення до потрібних нам значень та отримали результат на фото праворуч.

5.3 Агументація даних

Обробка самих зображень датасету не єдиний спосіб покращити загальну ефективність мережі. Окрім цього ми також можемо спеціально змінювати тренувальні зображення для покращення працездатності.

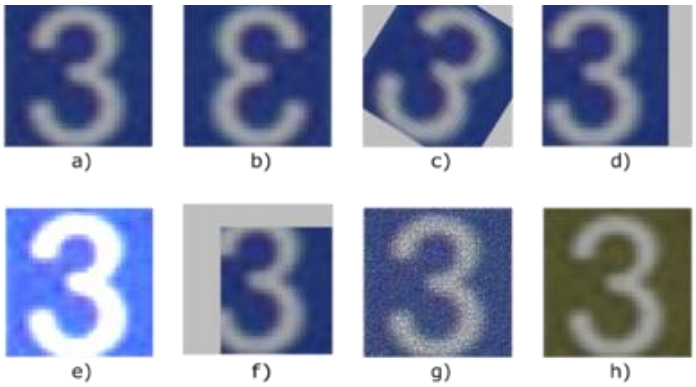
Під час тренування мережі ми використовуємо завчасно підготований набір даних, але найчастіше ці дані були спеціально підібрані для роботи з мережею та мають «ідеальний» формат. Таким чином мережа тренується розрізняти об’єкти, але має проблеми класифікувати ідентичний об’єкт у реальному житті, адже фотографія може бути нечіткою, можуть бути освітлення, сонячні промені тощо. Як результат, модель яка була натренована «в тепличних умовах» може втрачати свою ефективність під час роботи з новими даними, які були отримані ззовні датасету.

Вирішенням цієї проблеми є аугментація даних. Цей метод полягає в штучній генерації нових зображень, маючи за основу ідеальний шаблон з початкового датасету. Наведемо приклад.



5.4 Приклад зміни зображення при аугментації

Уявімо, що ми маємо зображення кота (зліва), але хочемо бути певними, що мережа може побачити його риси незалежно від орієнтації та скейлингу зображення. У такому випадку ми просто створимо декілька тих самих зображень котів, які будуть мати певні зміни: фото будуть перевернуті, окремі ділянки розтягнуті тощо. Мережа може спочатку не зрозуміти що це кіт, але після певної кількості ітерацій вона навчиться розуміти те, що вуха кота потрібно шукати не лише в вертикальній площині. Унаслідок цього мережа зможе бачити риси об’єкта незважаючи на зміни простору та орієнтації. Окрім зміни орієнтації зображення існують також інші способи.



5.518. Різновиди зміни зображення під час аугментації

На прикладі ми бачимо, що для ідеального шаблону трійки ми додали декілька змінених варіантів: віддзеркалений, з шумом, зі зміною кольору тощо.

Також аугментація даних вирішує ще одну проблему – брак даних для аналізу. Штучне створення різних варіантів одного зображення не тільки навчить мережу розрізняти об’єкт незважаючи на аномалії, а і підвищить ємність обсягу даних на яких буде навчатися мережа. Тому цей метод активно використовують при недостатній кількості даних для ефективного навчання

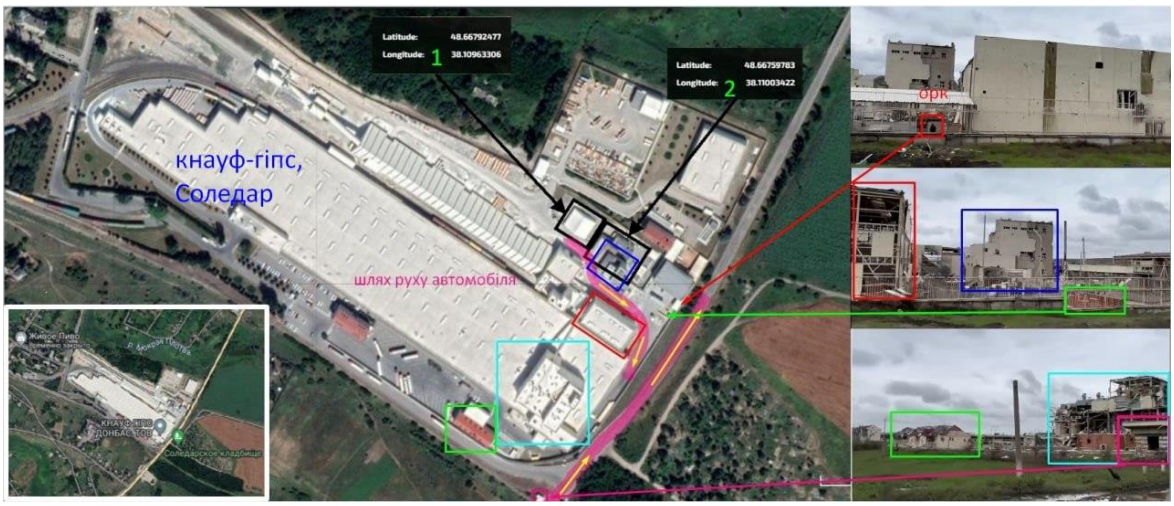
Розділ 6: Практична робота

6.1 Актуальність та постановка задачі

Для побудування згорткової нейронної мережі можна знайти безліч відкритих датасетів, які поділяються на різні категорії: деякі є загальновідомими та використовуються як змагальний датасет на великих івентах, інші ж можуть бути дуже простими, але майже не мати практичного змісту. Враховуючи таку полярність, було прийнято рішення обрати доволі популярну сферу та надати якогось практичного змісту.

Переглянувши декілька наборів даних було вирішено працювати з зображеннями, які були зняті за допомого супутників та інших літаючих засобів. Це обумовлено не тільки відкритістю та великою кількістю даних, а й тим фактом, що вже сьогодні за допомогою нейронних мереж роз’вязуються проблеми планування житлових секторів, спостереження трафіку та моніторингу змін у природі.

На момент написання кваліфікаційної роботи, дуже велику популярність набирають канали соціальних мереж, які займаються OSINT, тобто розвідкою на основі відкритих джерел. Одною з категорій є відслідковування та класифікація та наступний аналіз об'єктів, знятих супутниками або ж іншими повітряними засобами.



6.1. Приклад OSINT

Однак, мені траплялися джерела, які не могли точно класифікувати об’єкт, який зображений на знімку, в такому разі їх називали як «невідомий об’єкт» або ж намагалися власноруч встановити що ж зображено на знімку тощо. Ця «невідомість» є проблемою, адже інформація про те, що саме знаходиться за координатами знімку може бути надзвичайно корисним.

Отже, враховуючи важливість вирішення подібних проблем та власної зацікавленості було вирішено створити нейронну мережу, яка могла б розрізняти вид техніки, який зображений на знімку ступнику або іншому повітряному засобі. За допомогою такої нейронної мережі можна подолати невизначеність під час аналізу знімку(“unknown object”), адже правильно натренована мережа здатна вирішити що саме за об’єкт знаходиться на зображенні

6.2 Підбір та обробка датасету

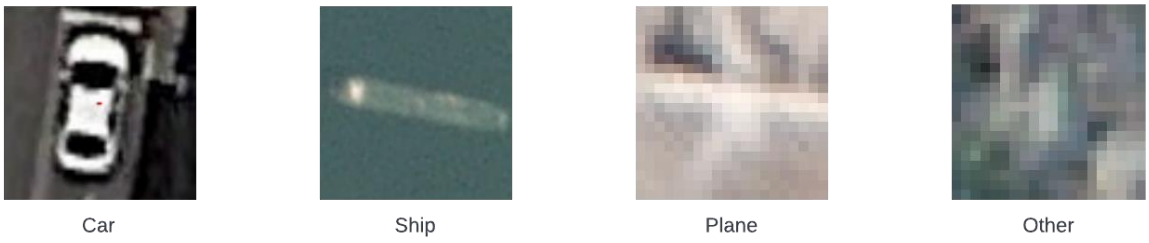
Під час пошуків не було знайдено конкретного датасету, який повністю задовольняв вимоги (ракурс зйомки, різні класи транспорту тощо), зазвичай більшість датасетів для класифікації техніки містять фото транспорту в профіль, а не зйомку згори. Тож на базі інших існуючих датасетів був створений та доповнений свій.

З публічного джерела було взято 2 окремих набори даних, в одному знаходилися супутникові знімки літаків, а в іншому – човнів. Об’єкти кожного з класів різнилися як по формі, так і за розміром. Також в наборах даних були зображення, які позначалися як “other”. Загалом, на таких зображеннях не було ніякої техніки, однак іноді траплялися частини об’єктів, за якими точно неможливо визначити що це є насправді. Зображення для цих датасетів були отримані з ресурсу Planet, який надає доступ до супутникових фотографій ландшафту різних точок планети та різного масштабу.

Однак, лише 2 об’єкти – замало для того щоб охопити більшість транспортної техніки. Тому було прийняти рішення власноруч доповнити

поєднання цих двох наборів даних зображеннями машин, пікапів тощо. Для цього був взятий третій набір даних, який має в собі зображення вулиць, доріг, автомагістралей різних міст які містять машини. На центральному пікселі машини на зображенні було встановлене значення яскраво червоного кольору RGB(255,0,0). Після цього, за допомогою мови Python та бібліотеки Numpy був написаний метод, який знаходить позиції усіх червоних пікселів на зображенні, вирізає рамку 80x80 пікселів та зберігає як окремий png файл. За допомогою цього, з одного знімку шосе можна отримати 10 машин, а також потреба власноруч вирізати машини в редакторів зображень зникла, що значно прискорило створення нових даних для класу мережі.

В результаті, фінальний набір даних має 36341 зображення, де наявні 4 категорії об’єктів: машина(незалежно від виду кузову), корабель(баржі, яхти, крейсери тощо), літаки та категорія інше, до якої входять зображення на яких немає жодного об’єкту або які містять частини об’єктів, за якими неможливо встановити що це за транспорт.



19. Приклад даних різних категорій

Також варто зазначити, що дані були поділені на 3 частини: 25829 зображень відповідають за тренування мережі, 1319 зображень за її валідацію та 4055 зображень для тестування.

Незважаючи на певну різниці з роздільній здатності зображень, можна побачити що основні риси є цілком помітними. Враховуючи той факт, що фільтри згорткових мереж знаходять ключові риси зображення, ця різниця не є критичною, адже риси та патерни зображень збережені, чого достатньо для їх знаходження.

6.3 Побудова застосунку

6.3.1 Використані технології

Для побудови мережі було використано Tensorflow. Вибір обумовлений не тільки високою популярністю Tensorflow: структурована та містка документація, зрозумілість. Також, використовуючи рекомендації розробника Keras, був використаний саме субмодуль Tensorflow.Keras, адже підтримка інших інших бекенд-частин для Keras призупинена, а використання імплентації Tensorflow є порадою від самих розробників.

Для проміжний завдань та аналітики роботи мережі були використані пакети Mathplotlib, Numpy та Pillow.

Для побудови графічного застосунку, завдяки якій можна взаємодіяти з мережею був використаний пакет-нащадок популярного TKinter – CustomTKinter. Його особливість полягає в більш гнучкому налаштуванні візуальної частини додатку.

6.3.2 Архітектура та тренування мережі

Вибір архітектури для мережі не був миттєвим – він полягав в ітеративному тренуванні, аналізу результатів та внесення змін до структури мережі.

Версія	З.Ш	П.Ш	А.Ш	Фільтр	Точність	Час
1	1	2	-	8	69%	243
2	3	2	-	8	86%	266
3	3	2	MAX	8	88%	134
4	3	2	MAX	32	92%	144
5	4	3	MAX	32	93%	171

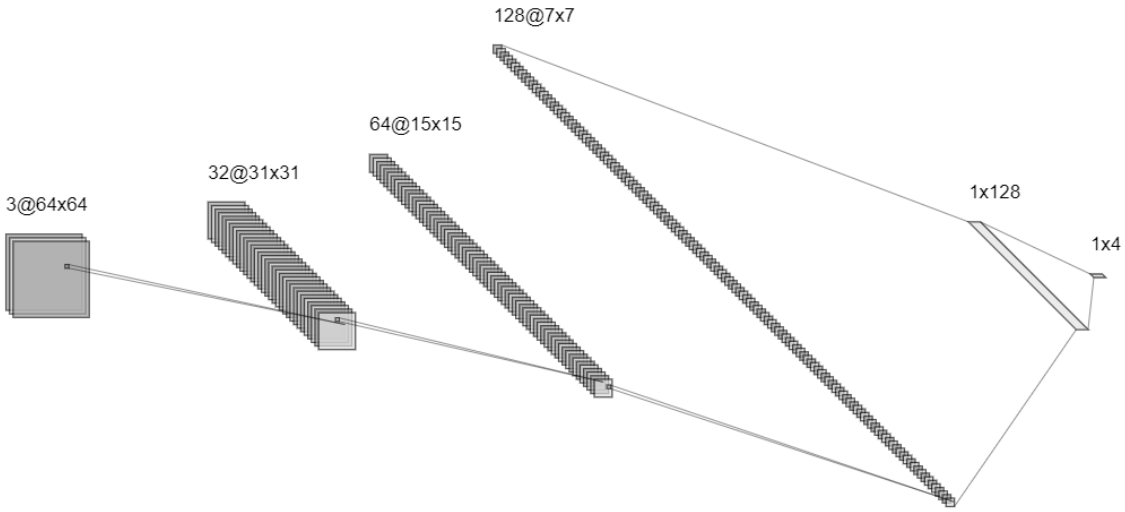
Пояснення до таблиці:

- З.Ш. – кількість згорткових шарів
- П.Ш. – кількість повно’зєднаних шарів
- А.Ш. – тип агрегувального шару
- Фільтр – початкове значення кількості фільтрів, з кожним наступним шаром їх кількість помножується на 2
- Точність – точність моделі на тестових даних
- Час – витрачений час на тренування у хвилинах

Згідно таблиці, можемо побачити шо перша версія мережі є не тільки неефективною, а й має занадто великий час тренування. Друга версія отримала більшу кількість згорткових шарів, що дозволило мережі знаходити більше рис, однак враховуючи велику кількість обчислень, час її тренування є занадто великим. В третій версії був доданий max-pooling між згортковими шарами, що зменшило кількість обчислень, але її точність все ще є недостатньою.
Найцікавіше розглянути четверту та п’яту версії. Четверта має 3 згорткових шари, кількість фільтрів яких починається з 32, точність мережі на тестових даних є достатньою - 92%. Наступна версія має на один згортковий та повноз’єднаний шар більше, але її точність є всього на 1 відсоток більше від попередньої. Незважаючи на більш точний результат, мережа навчалася на 27 хвилин більше, тому мені здається, що 1 відсоток вартістю в 27 хвилин є занадто великою ціною.

Активаційною функцією для усіх шарів мережі крім останнього було обрано ReLU, це пов’язано з її простотою та точністю. Для останнього шару було обрано Softmax. Для функції витрат – cross-entropy. Зазвичай, для отримання вірогідності «чи належить об’єкт до певної категорії» використується пара Softmax та Cross-Entropy. Навчання для кожної версії відбувалося протягом 20 епох, кількість зображень в групі – 64, а кількість таких груп – 808. Така

кількість зображень групи обумовлена балансом часу та фінальної точності мережі.



6.3. 4 версія мережі

На малюнку зображена 4 версія мережі, яка є балансом між часом та результатом. Після кожного кроку застосовано max-pooling, що зменшує розмір зображення, та пришвидшує обчислення. Перед першим повноз'єднаним шаром також є шар згладжування, але на схемі він не представлений.

6.3.3 Аугментація даних

Також було вирішено використати аугментацію даних за допомогою ImageDataGenerator з підпакету keras.preprocessing.image. Були обрані наступні типи аугментації:

- Поворот
- Скейлинг
- Горизонтальне та вертикальне перевернення
- Модификація тону зображення

Рішення про використання аугментації було прийнято через не зовсім збалансований датасет та вимогу до збільшення зображень класу машин. Таким чином кількість різноманітних зображень підвищилась, що надало змогу моделі використовувати більше даних для тренування.

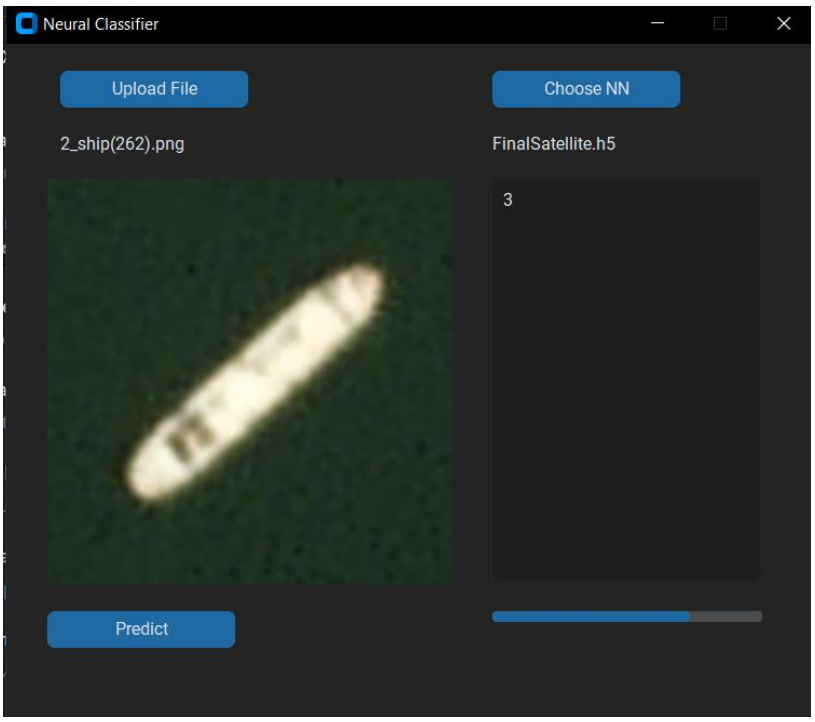
6.3.4 Графічний застосунок

Ідея застосунку полягає у можливості використання збереженої моделі нейронної мережі не використовуючи редактори коду, IDE тощо. За допомогою застосунку ми можемо обрати зображення з директорії та побачити передбачення, яке згенерувала мережа. Таким чином, навіть не досвідчений користувач ПК має змогу використовувати нейронні мережі для класифікації зображень.

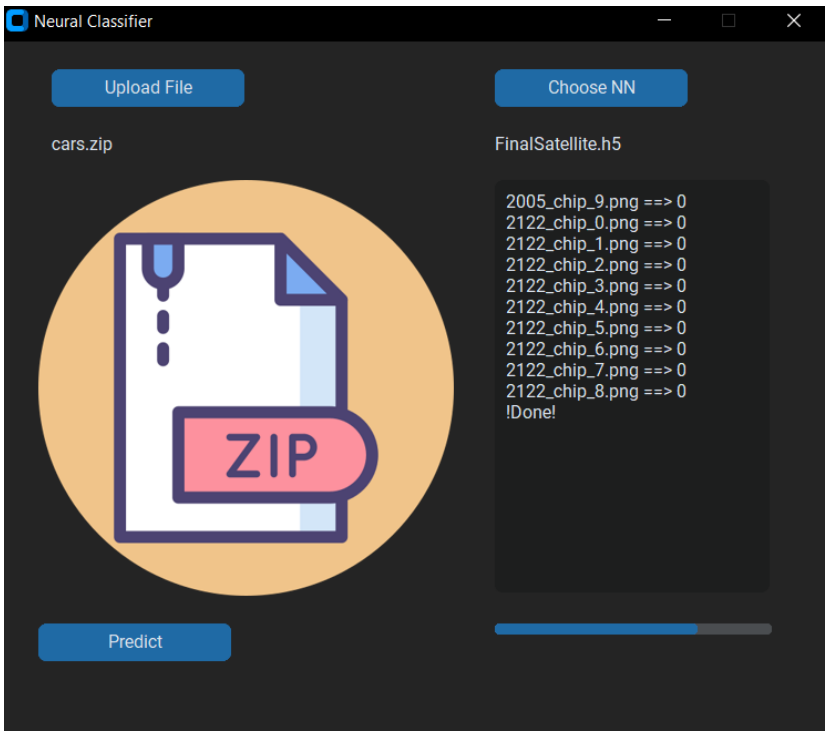
Для розробки застосунку була використана бібліотека CustomTKinter. Вона є нащадком популярної TKinter, але на відміну від нього має більше можливостей до налаштування виду, форми та кольору елементів на екрані. Наразі ця бібліотека постійно обновлюється, хоча вона має не так багато віджетів як звичайний TKinter, але їх було достатньо для побудови застосунку. Також варто зазначити, що бібліотека CustomTKinter значно легша за PyQT, робота з якою вимагає більше часу та навичок для вивчення. Тому враховуючи простоту, повну документацію та можливості, було обрано саме CustomTKinter. Для візуального відображення зображень була використана бібліотека Pillow, для передачі зображення та його приведення до правильного формату (тензор) – Numpy.

За допомогою провідника ми можемо обрати зображення, на натискаючи кнопку «Predict» модель згенерує передбачення та виведе його в текстове поле праворуч. Також, додаток може працювати з zip-файлами, тоді модель буде кожне зображення архіву ітеративно передається до моделі, а результат поступово виводиться один за одним на текстове поле.

Також варто зазначити, що додаток може завантажити довільну мережу, яка буде відповідати за передбачення, але в прикладі відповідно представлена власна мережа 4 версії.



6.5. Приклад роботи на зображенні



6.6.Приклад роботи з архівом

6.3.5 Складнощі при виконанні роботи

Основною складністю при виконанні практичної частини були пов’язані з відсутністю готового набору даних для класифікації. Відповідно, датасет мав бути згуртований та доповнений власноруч, про що було сказано в розділі 6.2.

Також складним для мене був вибір технологій для побудови самої мережі: я вагався між Tensorflow та Pytorch. Незважаючи не популярність обох фреймворків, вибір був зроблений в користь Tensorflow, враховуючи високу кількість матеріалів та ресурсів, які були корисні під час опанування технологіями.

Враховуючи характеристики мого комп'ютеру, на якому відбувалося тренування мережі, PyCharm іноді зависав або ж навіть аварійно закривався, що збільшувало не тільки час обчислень, а і кількість перезапусків після аварійних зупинок програми.

6.3.6 Можливості розвитку моделі

Перш за все на мою думку, варто збільшити кількість категорій для навчання та відійти від стандартного набору «машина-літак-корабель». Можливо додати різновиди військової техніки або ж різні види інших транспортних засобів: танки, тягачі, гелікоптери тощо. Завдяки цьому мережа не тільки зможе більш точно передати користувачеві чим саме є об’єкт, а й бути корисніше в рамках аналізу зображень знятих супутниками або іншими повітряними засобами (наприклад під час процесу OSINT). Наразі подібні фотографії можна взяти лише з матеріалів зйомки дронів, або отримати шляхом обробки зображень від таких компаній як Maxar, які публікують супутникові знімки з різних частин планети: в травні 2023 року Maxar надала знімки Донецької області станом на травень 2023 року. Враховуючи можливі зміни в наборі даних, загальна структура мережі також може зазнати змін.

Розділ 7: Висновки

У теоретичній частині роботи було проведено загальний огляд нейронних мереж як спосіб вирішення задачі класифікації зображення. Основною метою роботи був аналіз того, чому саме цей підхід є актуальним та ефективним на сьогоднішній день. Був проведений огляд загальної архітектури нейронної мережі, принцип роботи та фундаментальні особливості, які роблять їх корисним інструментом для рішення проблеми класифікації. Основний акцент було надано згротковим мережам, які є найбільш оптимальним та поширеним способом: ми оглянули архітектуру, принцип роботи кожного ключового блоку, їх особливості та завдання. Також в роботі ми оглянули найвідоміших мережі та сфери їх використання. Було створено власну нейронну мережу, яка класифікує зображення транспорту, знятих супутниками або іншими повітряними засобами. Побудова мережі була поступова та мала ітеративний підхід: декілька версій мережі мали різні структури та значення гіперпараметрів. Також було розроблено графічний застосунок-помічник, який дозволяє взаємодіяти з нейронною мережею та отримувати передбачення для завантажених зображень або ж для zip-файлів які містять зображення.

Тема нейронних мереж є одною з найбільш популярною навіть серед спільнот, які ніяк не пов’язанні з технологіями та точними науками. Напрямок активно розвивається, а поле використання мереж з кожним роком все збільшується. Згорткові нейронні мережі вже сьогодні використовуються від звичайних побутових справ як знаходження об’єктів на відеокамерах до наукової діяльності в передових медичних, аграрних інститутах тощо. Враховуючи вищенаведені факти, неможливо заперечувати факт того, що використання CNN скоро буде побутовим явищем, а загальна обізнаність та цікавість до теми лише зростатиме.

8. Джерела

1. Activation functions and their derivatives - A quick & complete guide. *Analytics Vidhya*. URL: https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/.
2. Activation functions in neural networks [12 types & use cases]. *V7 - AI Data Platform for Computer Vision*. URL: https://www.v7labs.com/blog/neural-networks-activation-functions.
3. An ultimate tutorial to neural networks in 2022. *Simplilearn.com*. URL: https://www.simplilearn.com/tutorials/deep-learning-tutorial/neural-network.
4. Banoula M. What is cost function in machine learning [updated] | simplilearn. *Simplilearn.com*. URL: https://www.simplilearn.com/tutorials/machine-learning-tutorial/cost-function-in-machine-learning.
5. Feed forward neural networks - intuition on forward propagation. *Analytics Vidhya*. URL: https://www.analyticsvidhya.com/blog/2021/10/feed-forward-neural-networks-intuition-on-forward-propagation/.
6. Ganesh K. S. What's the role of weights and bias in a neural network?. *Medium*. URL: https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f.
7. Gour R. Artificial neural network for machine learning – structure & layers. *Medium*. URL: https://medium.com/@rinu.gour123/artificial-neural-network-for-machine-learning-structure-layers-2a275f73f473.
8. How do activation functions introduce non-linearity in neural networks?. *Analytics India Magazine*. URL: https://analyticsindiamag.com/how-do-activation-functions-introduce-non-linearity-in-neural-networks/.

9. How does backward propagation work in neural networks?. *Analytics Vidhya*. URL: https://www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/.
10. Neural networks and deep learning. *Coursera*. URL: https://www.coursera.org/learn/neural-networks-deep-learning.
11. Neural networks: structure. *Developers.google.com*. URL: https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy.
12. Neural networks: structure, types, and possibilities. *IEEE Computer Society*. URL: https://www.computer.org/publications/tech-news/neural-network-structures.
13. Nonlinear activation functions in a backpropagation neural network | baeldung on computer science. *Baeldung on Computer Science*. URL: https://www.baeldung.com/cs/ml-nonlinear-activation-functions.
14. Sharma S. Activation functions in neural networks. *Medium*. URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.
15. Shukla L. Fundamentals of neural networks on weights & biases. *Weights & Biases*. URL: https://wandb.ai/site/articles/fundamentals-of-neural-networks.
16. Weights and biases - AI wiki. *Artificial Intelligence Wiki - AI Wiki*. URL: https://machine-learning.paperspace.com/wiki/weights-and-biases.
17. Weights and biases in machine learning | h2o.ai wiki. *H2O.ai | The fastest, most accurate AI Cloud Platform*. URL: https://h2o.ai/wiki/weights-and-biases/.
18. What is classification in AI?. *H2O.ai | The fastest, most accurate AI Cloud Platform*. URL: https://h2o.ai/wiki/classification/.
19. What is forward propagation? | h2o.ai. *H2O.ai | The fastest, most accurate AI Cloud Platform*. URL: https://h2o.ai/wiki/forward-propagation/.
20. 3Blue1Brown. What is backpropagation really doing?, 2017. *YouTube*. URL: https://www.youtube.com/watch?v=Ilg3gGewQ5U.

21. A gentle introduction to padding and stride for convolutional neural networks - machinelearningmastery.com. *MachineLearningMastery.com*. URL: https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/.
22. A gentle introduction to pooling layers for convolutional neural networks - machinelearningmastery.com. *MachineLearningMastery.com*. URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.
23. Choudhari P. Understanding “convolution” operations in CNN. *Medium*. URL: https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4.
24. Coding Lane. CNN architecture | Explaining the Architecture of CNN, 2021. *YouTube*. URL: https://www.youtube.com/watch?v=VF4BDE7uqY0.
25. Coding Lane. Fully Connected Layer in CNN, 2021. *YouTube*. URL: https://www.youtube.com/watch?v=rxSmwM7z0_4 (date of access: 24.05.2023).
26. Cost function is no rocket science!. *Analytics Vidhya*. URL: https://www.analyticsvidhya.com/blog/2021/02/cost-function-is-no-rocket-science/.
27. DeepAI. Stride (machine learning). *DeepAI*. URL: https://deepai.org/machine-learning-glossary-and-terms/stride#:~:text=Stride%20is%20a%20component%20of,over%20the%20image%20or%20video.
28. Gradient descent – ML glossary documentation. *Machine Learning Glossary – ML Glossary documentation*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html.
29. Guide to different padding methods for CNN models. *Analytics India Magazine*. URL: https://analyticsindiamag.com/guide-to-different-padding-methods-for-cnn-models/.

30. How do convolutional layers work in deep learning neural networks? - machinelearningmastery.com. *MachineLearningMastery.com*. URL: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.
31. Introduction to pooling layers in CNN. *Towards AI*. URL: https://towardsai.net/p/l/introduction-to-pooling-layers-in-cnn.
32. Jordan J. Common architectures in convolutional neural networks. *Jeremy Jordan*. URL: https://www.jeremyjordan.me/convnet-architectures/.
33. Kostadinov S. Understanding backpropagation algorithm. *Medium*. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.
34. Mishra M. Convolutional neural networks, explained. *Medium*. URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.
35. Neural networks notations. *CS230 Deep Learning*. URL: https://cs230.stanford.edu/files/Notation.pdf (date of access: 24.05.2023).
36. Pandey P. Understanding the mathematics behind gradient descent. *Medium*. URL: https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e.
37. Understanding padding and stride in convolutional neural networks - programmathically. *Programmathically - A Blog on Building Machine Learning Solutions*. URL: https://programmathically.com/understanding-padding-and-stride-in-convolutional-neural-networks/.
38. What are convolutional neural networks? | IBM. *IBM - Deutschland | IBM*. URL: https://www.ibm.com/topics/convolutional-neural-networks.
39. What is gradient descent? | IBM. *IBM - Deutschland | IBM*. URL: https://www.ibm.com/topics/gradient-descent.
40. Data augmentation | TensorFlow Core. *TensorFlow*. URL: https://www.tensorflow.org/tutorials/images/data_augmentation.

41. isahit. What is the purpose of image preprocessing in deep learning?. *Isahit*. URL: https://www.isahit.com/blog/what-is-the-purpose-of-image-preprocessing-in-deep-learning.
42. Official documentation and tutorial. *CustomTkinter*. URL: https://customtkinter.tomschimansky.com/.
43. Tanya. Data preprocessing and network building in CNN. *Medium*. URL: https://towardsdatascience.com/data-preprocessing-and-network-building-in-cnn-15624ef3a28b.