Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження ефективності моделей розпізнавання товарів на інтернет ресурсах
(тема)

Виконав:
студент 2 курсу, групи СШІм-21-2
Бордюг Д.Є.
(прізвище, ініціали)

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

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Смеляков К.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

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

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бордюгу Дмитру Євгенійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження ефективності моделей розпізнавання товарів на інтернет ресурсах

затверджена наказом університету від 31 березня 20 23 р. № 306Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19 травня 20 23 р.

3. Вихідні дані до роботи Операційна система – Windows 10, частота процессору 2.5 ГГц, середовище розробки – PyCharm, мова програмування – Python, фреймворк машинного навчання – Tensorflow, бібліотеки що використовувались: keras.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка завдання
2) Дослідження методів згорткової нейронної мережі
3) Реалізація моделі згорткової нейронної мережі
4) Отримання результатів дослідження
5) Експерименти та оцінка дослідження

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

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

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

**КАЛЕНДАРНИЙ ПЛАН**

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на дипломну роботу | 05.04.2023 | Виконано |
| 2 | Аналіз предметної галузі та постановка завдання | 08.04.2023 | Виконано |
| 3 | Дослідження методів згорткової нейронної мережі | 10.04.2023 | Виконано |
| 4 | Підготовка даних | 20.04.2023 | Виконано |
| 5 | Реалізація моделі згорткової нейронної мережі | 24.04.2023 | Виконано |
| 6 | Експериментальна перевірка, обробка та аналіз | 30.04.2023 | Виконано |
| | отриманих результатів | | |
| 7 | Оформлення пояснювальної записки | 01.05.2023 | Виконано |
| 8 | Попередній захист | 10.05.2023 | Виконано |
| 9 | Захист перед ЕК | 19.05.2023 | |
| | | | |
| | | | |

Дата видачі завдання 3 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Смеляков К.С.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка: 55 с., 54 рис., 1 дод., 20 джерел.

ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, KERAS, PYTHON, TENSORFLOW.

Об’єкт дослідження – класифікація зображень.

Предмет дослідження – класифікація зображень за допомогою згорткових нейронних мереж.

Мета роботи – дослідження згорткової нейронної мережі та її розробка.

Методи дослідження – згорткові нейронні мережі, класифікація зображень.

В результаті проведених досліджень була розроблена згорткова нейронна мережа таким чином, щоб вона могла вирішувати поставлене завдання.

Отримані результати використовуються для класифікації зображень.

В якості програмної платформи використано мова програмування Python з бібліотеками машинного навчання Google Tensorflow та Keras. Пропонована розробка є корисною для класифікації зображень та має попит у сучасному часі.

## ABSTRACT

Explanatory note: 55 p., 54 fig., 1 ann., 20 sources.

CONVOLUTIONAL NEURAL NETWORKS, IMAGE CLASSIFICATION, KERAS, PYTHON, TENSORFLOW.

The object of research – image classification.

The subject of the research – image classification using convolutional neural networks.

The purpose of the work – to study the convolutional neural network and its development.

Research methods – convolutional neural networks, image classification.

As a result of the conducted research, a convolutional neural network was developed in such a way that it could solve the given task.

The obtained results are used for image classification.

The Python programming language with Google Tensorflow and Keras machine learning libraries was used as a programming platform. The proposed development is useful for image classification and is in demand in modern times.

**ЗМІСТ**

# ВСТУП

Штучний інтелект використовують комп'ютери та машини для імітації здатності людського розуму вирішувати проблеми та приймати рішення. Інакше кажучи, штучний інтелект – це метод програмування комп'ютера, робота чи інших пристроїв таким чином щоб вони думали, як високоінтелектуальна людина. Останні кілька десятиліть з'явилося багато визначень штучного інтелекту (ШІ). Цей термін часто застосовується до проектів розробки систем, наділених інтелектуальними процесами, характерними для людей [9].

На даний час штучний інтелект (ШІ) та штучні нейронні мережі (ШНМ) є з одних напрямків, які зараз активно розвиваються з кожним днем все більше і більше. Проте, мало хто насправді може розуміти, на що взагалі здатні ці технології, та що нам варто від них очікувати в нашому майбутньому. Штучний інтелект дає змогу нам допомогти вирішувати певні задачі, які раніше було неможливо вирішити, або на які потрібно було витратити багато свого часу. Він робить наше життя більш комфортним й зручним.

В сьогоденні інструменти, які використовуються в штучному інтелекті набрало широких та певних масштабів. Багато компаній та сфер почали його активно використовувати тому що вони бачать, як це в тій чи іншій діяльності спростує їм певні ресурси і зробить продукт більш вдосконаленим та сучасним.

Під час застосування штучних нейронних мереж (ШНМ) можна виділити основні області, а саме: класифікація зображень, аналіз даних, прогнозування та прийняття рішень, оптимізація.

Зараз запропоновано та існує багато методів та алгоритмів для рішення класифікації зображень, але усі вони поступаються у точності результатів та простоті, як це може зробити штучні нейронні мережі.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

## 1.1 Аналіз предметної галузі

В останній час ми недооцінюємо технології які нас оточують. Вони є по всюди, але ми можемо їх навіть не замічати тому що привикли до їх використання. Можна навести приклад того, коли ми хочемо зайти в якийсь додаток на телефоні і нам потрібно сканувати обличчя, щоб система дала доступ до використання повного функціоналу. В цьому процесі за допомогою комп’ютерного зору та машинного навчання може спростувати життя кожної людини. Цей процес складається з того, що наше зображення обличчя береться з бази даних, тому що можна використовувати багато образів в одному телефоні. Воно береться с поверхневого рівня, в той час коли проходить аналіз певних пікселів для загального зображення. Все це нам каже що саме класифікація зображень відіграє велику роль у сучасних технологіях.

На сьогоднішній день існує все більше методів та технологій, які є актуальні та важливі для вирішування класифікації зображень. Всі ці методи надбали широкого та популярного використання в різних галузях життя. Наприклад, таких як:

– медицина та екологія;

– сільське господарство;

– певні види спостереження над навколишнім світом та географічне картографування;

– наука та космічний простір.

Для класифікації зображень можуть використовувати такі методи, як штучні нейронні мережі, глибоке навчання, бустінг та баггінг методи, генетичні алгоритми. В останній час, дослідники які цим займаються, їхню увагу завойовують методи глибокого навчання. Ці методи мають велику

роль під час покращення вірогідності та класифікації.

Так як зараз все те, що було сказано попереду є популярним і користується попитом то воно має шанс на існування та подальший розвиток з кожним днем.

## 1.2 Задача класифікації зображень

В нашому розумі може скластися багато питань - які задачі використовує класифікація зображень, як виявити об’єкти на зображенні, якими способами зробити локалізацію тих самих об’єктів? Перш за все усі ці питання та терміни є компонентами комп’ютерного зору. Можна зазначити, що коли нам потрібно зробити класифікацію зображення, буде призначено мітку в той час коли локалізація головного об’єкта буде представляти певне зображення (рисунок 1.1).

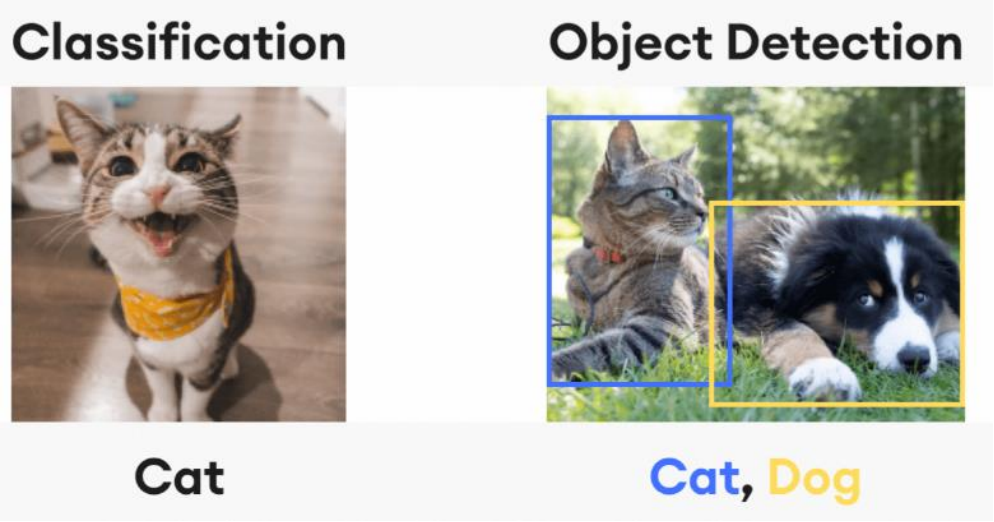



Рисунок 1.1 – Приклад локалізації об’єктів

На цьому рисунку ми можемо побачити виявлення об’єктів які взяті в обмежувальні рамки. Воно зумовлюється тим, що було призначено мітку усьому зображенню, тим самим розпізнало елементи які були у середині, та вказує на їхнє положення. Цим кроком ми можемо сказати, що класифікація

зображення певним чином пов’язана з виявленням об’єктів.

Зображення складається з пікселів. Їх може існувати дуже багато. Наприклад, як сотні так і тисячі. Щоб комп’ютерному зору визначити те чи інше зображення та зробити певний висновок, перш за все йому необхідно зробити аналіз, проаналізувати компоненти з яких складається зображення, які будуть у вигляді пікселів. Весь цей шлях досягається обробкою зображень, а саме як масивом матриць, розмір яких можливо визначити роздільною здатністю зображення. Усі пікселі які є у цифровому зображенні групуються в певні класи. Алгоритм який ми виберемо для цієї задачі, розділить зображення на атрибути, щоб гарантувати що він ніколи не залишиться на кінцевому класифікаторі. За допомогою усіх цих атрибутів, які ми будемо використовувати, вони допоможуть класифікатору зрозуміти та визначити до якого класу зображення буде відноситися та належати. Вся решта етапів які залишилися, вони будуть залежати від цього. То можна сказати що виділення ознак на зображені є найважливішим етапом у класифікації зображень (рисунок 1.2).

Рисунок 1.2 – Приклад розпізнавання пікселів

На рисунку 1.2 можна побачити роботу розпізнавання пікселів. Після

цього як пікселі будуть розпізнані, то буде певне зображення на основі виявлених пікселів (рисунок 1.3).

Рисунок 1.3 – Успішна робота розпізнавання пікселів

Задача класифікації зображень в наше сьогодення дуже може допомагати та економити наш час за допомогою правильного використання.

Успішне застосування нейронних мереж для вирішення реальних проблем комп’ютерного зору призвело до поява багатьох нових моделей згорткових нейронних мереж (ЗНМ) та їх різноманітних модифікацій використовується для виявлення та класифікації зображень в останні роки [12].

## 1.3 Постановка задачі дослідження

За допомогою аналізу предметної галузі тепер ми можемо чітко та без всяких сумнівів сформулювати постанову задачі та усі необхідні вимоги до продукту, який буде створено в цій роботі. Метою роботи є дослідження та аналіз обробки зображень за допомогою чого ми зможемо їх розпізнавати та класифікувати по певному виду, який нам потрібний.

Постанова задачі складається з наступних етапів:

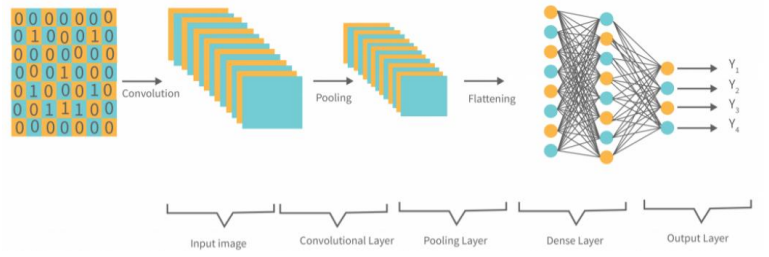
– підготовка даних для класифікації та розпізнавання;
– первинна обробка зображення;
– виявлення характеристик та пікселів зображення;
– класифікація зображення по виду діяльності.

Програмний продукт який буде створений, повинен бути зрозумілим та зручним при використанні його користувачем.

# 2 ДОСЛІДЖЕННЯ МЕТОДІВ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

## 2.1 Згорткова нейронна мережа

Згорткові нейронні мережі (CNN або ConvNet) – це тип багаторівневої нейронної мережі, яка призначена для розпізнавання візуальних шаблонів із піксельних зображень. У CNN «згортка» називається математичною функцією. Це тип лінійної операції, у якій ви можете помножити дві функції, щоб створити третю функцію, яка виражає, як форма однієї функції може бути змінена іншою. Простіше кажучи, два зображення, які представлені у вигляді двох матриць, перемножуються, щоб забезпечити вихід, який використовується для вилучення інформації із зображення. CNN схожий на інші нейронні мережі, але оскільки вони використовують послідовність згорткових шарів, вони додають рівень складності до рівняння. CNN не може функціонувати без згорткових шарів. Згорткова нейронна мережа складається з численних шарів, таких як шари згортки, шари об’єднання та повністю зв’язані шари, і використовує алгоритм зворотного поширення для автоматичного й адаптивного вивчення просторових ієрархій даних [1]. На рисунку 2.1 зображено загальне зображення архітектури згорткової нейронної мережі.




Рисунку 2.1 – Загальне зображення архітектури згорткової нейронної мережі

В згортковій нейронні мережі (CNN) можна використовувати RGB зображення, яке складається з пікселів, як матриця з трьома площинами (рисунок 2.2).

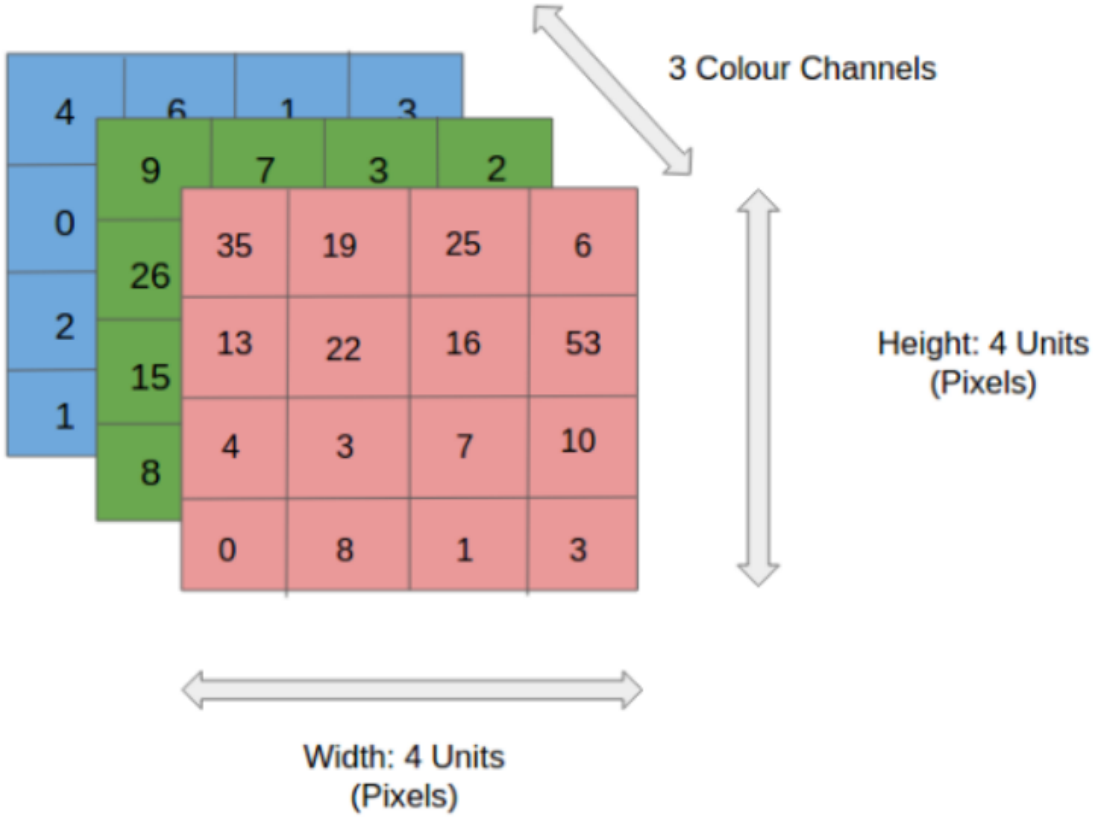



Рисунок 2.2 – Структура RGB зображення

В згорткових нейронних мережах також існують переваги, недоліки:

– знаходить та виявляє деталі закономірностей в зображеннях та їх особливості;

– наскрізне навчання, тобто немає необхідності в ручному обробленні або видаленні функцій;

– заслуговує довіру користувачів високою точністю та зумовлює обробляти великі обсяги даних;

– потребує багато пам’яті при навчанні обчислення;

– коли йде обробка з мітками потребує великих обсягів даних.

Згорткова нейронна мережа має деяку специфіку – вона полягає у

перестановці згорткових шарів та шарів субдескритизації (пулінгу) [16].

## 2.2 Згорткові шари

Згортковий шар – складається з набору згорткових фільтрів (так званих ядер). Вхідне зображення, виражене як N-вимірна метрика, згортається за допомогою цих фільтрів для створення вихідної карти ознак [2].

Ядро можна визначити за допомогою дискретних чисел, які є у сітці або з певних значень, що описує це ж саме ядро. Кожне з цих значень має назву як вага ядра.

Це робить структуру ієрархічною, оскільки наступні шари можуть візуалізувати пікселі в межах рецептивних полів попередніх шарів. Потім згорткові шари перетворюють задане зображення в числові значення та дозволяють мережі зрозуміти та витягти цінні шаблони [18].

Процес згортання складається з того, що ядро може зміщуватись вздовж матриці і якщо у нас є вектори, то ми можемо отримати скалярний добуток (рисунок 2.3).

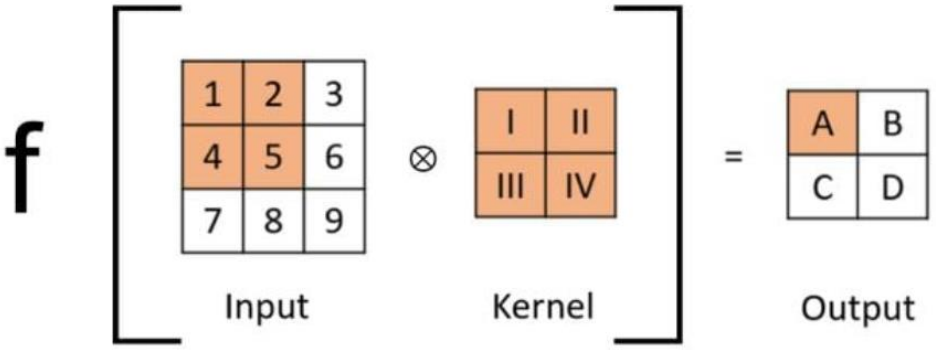



Рисунок 2.3 – Зображення як функціює згортковий шар

На рисунку 2.4 який буде зображено нижче, можна буде побачити чому скалярний добуток, який утворюється між полями, він виділений помаранчевим кольором, що далі буде виводити скаляр, який складається

з 1x4 x 4x1 буде дорівнювати 1x1.

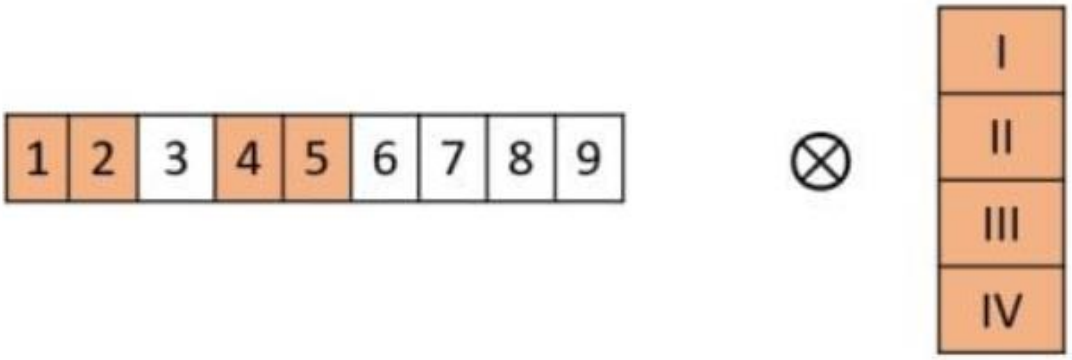



Рисунок 2.4 – Векторна форма для згортки

В нейронній мережі, де використовується згортковий шар (Convolution Layer) не всі вузли можуть бути підключені до тих самих вихідних вузлів. За допомогою цього в нас згорткові шари становляться гнучкі в навчанні. В шарі є вагові коефіцієнти  та їх кількість менша. Цим самим зображення з великими даними становиться обробляти набагато легше.

Якщо ми будемо виділяти при процесі згортки грані або контури зображення, то це нам позволить при використанні зображень з високою роздільною здатністю, зменшити кількість інформації, яка зберігається при класифікації.

На рисунку 2.5 буде зображено, як вузли будуть виглядати коли вони з’єднані. Ми можемо побачити разом з вами, що у нас є цифри від 1 до 9, які з’єднані між собою. Вони уявляють собою, як вони можуть з’єднуватись в згортковому шарі. За допомогою цієї ілюстрації ми все маємо поняття та уявлення, як вони можуть функціонувати.

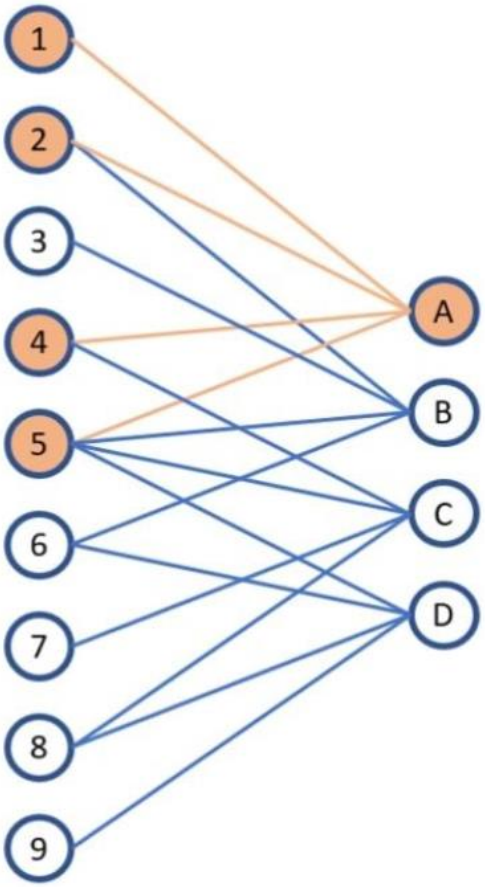



Рисунок 2.5 – Ілюстрація, як виглядають з’єднані вузли в зготковому шарі

Коли вузли нещільно з’єднані між собою, то це надає певної гнучкості в процесі навчанні згорткового шару, а кількість коефіцієнтів яка є в шарі – вона менша і тим самим допомагає даним, тобто зображенню. Всі ці переваги надають згорковим нейронним мережам певних особливостей у вивчені даних, наприклад, текстури та форми. Згортковий шар має безліч параметрів, які в подальшому можна буде змінювати для того щоб зробити адаптацію вихідного розміру.

Існує формула 1.1 для обчислення вихідного розміру згорткового шару:

$$n_{out} = \frac{n_{in}+2p-k}{s} + 1\ . \qquad (1.1)$$

Вихідний розмір дорівнює вхідному розміру плюс подвійний розмір заповнення, мінус розмір ядра над кроком плюс один. Більшу частину часу ми маємо справу з квадратними матрицями, тому це число буде однаковим для рядків і стовпців. Якщо в результаті дробу не виходить ціле число, округляємо в більшу сторону. Важливо зрозуміти рівняння. Ділення на крок має сенс з тієї причини, що коли ми пропускаємо операції, ми ділимо вихідний розмір на це число. Подвійне заповнення походить від того, що заповнення додається з обох сторін матриці, а отже, додається двічі [3].

Для того щоб знайти транспонований згортковий розмір існує формула 1.2:

$$n_{out} = (n_{in} - 1)s - 2p + (k - 1) + 1 \, . \qquad (1.2)$$

Коли потрібно зробити розширення та збільшення розмірності, то цей крок може привести до труднощів, які будуть у процесі навчанні тому що дані, будуть використовуватись для розрідження. Щоб уникнути цього потрібно використовувати транспоновані згортки.

Також ще існує формула 1.3 для того, щоб знайти згортковий шар у дискримінаторі. Вона має такий вигляд:

$$n_{out} = \frac{n_{in}+2p-k}{s} + 1 = \frac{64+2-4}{2} + 1 = 32 \, . \qquad (1.3)$$

Перша згортка, яка буде використовуватись, має розмір в 4 ядра, 2 кроки та 1 відступ. Тим самим, на виході у нас у нас буде зображення з розміром 32x32, як це написано в коді програми. Наступні 3 кроки будуть зумовлені тим, що розмір кожного шару буде становитися все менше та менше, наприклад, спочатку буде розмір 16x16, наступний 8x8, потім 4x4. Отже, при використанні цієї формули ми зможемо отримати вихідний розмір 1x1.

Формула 1.4, це є рівнянням для транспонованої згортки, тобто якщо

у нас було 4x4 стане потім 8x8, після цього 16x16, 32x32 і фінальний результат 64x64.

$$n_{out} = (1-1) * 1 - 2 * 0 + (4-1) + 1 = 4\,. \qquad (1.4)$$

Цим самим, цими усіма формулами ми можемо розуміти, як обчислюються згорткові шари та як обчислити вихідні розміри, без цього розуміння, буде тяжко та неможливо розробити свою згорткову нейронну мережу.

На рисунку 2.6 ми можемо побачити нейрони згорткового шару.

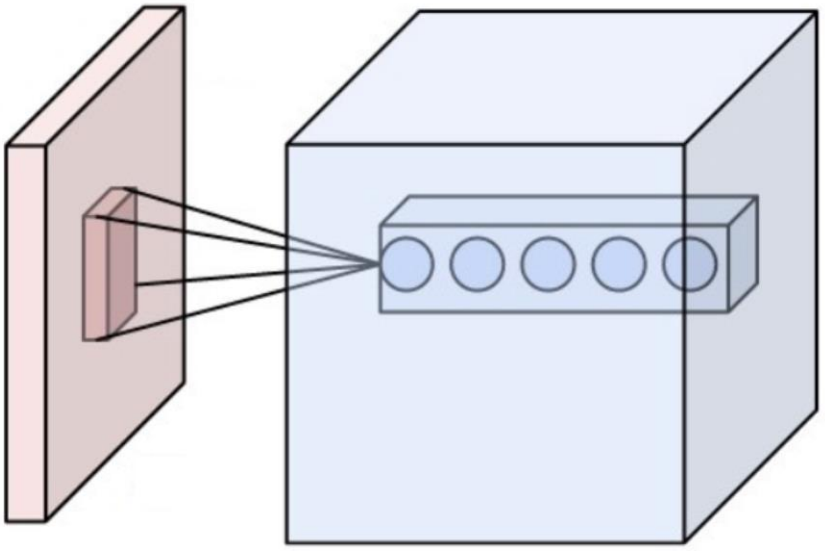


Рисунок 2.6 – Нейрони згорткового шару

На рисунку 2.6 ми бачимо що синім кольором це нейрони згорткового шару, а червоним кольором, це називається тот процес на якому буде зображено з’єднання нейронів за допомогою рецептивного поля.

## 2.3 Агрегувальні шари

Агрегувальні шари замінюють вихідні дані мережі в певних місцях шляхом отримання сумарної статистики найближчих виходів. Це допомагає зменшити просторовий розмір представлення, що зменшує необхідну кількість обчислень і ваги. Операція об’єднання обробляється для кожного фрагмента представлення окремо. Існує кілька функцій агрегувальних шарів, наприклад середнє значення прямокутної околиці, норма прямокутної околиці та зважене середнє на основі відстані від центрального пікселя (рисунок 2.7). Однак найпопулярнішим процесом є максимальне об’єднання, яке повідомляє про максимальний результат із сусідства [4].

Об’єднання шарів – це техніка зменшення розмірності, яка зменшує кількість вхідних параметрів. Процес об’єднання фільтрує вхідні дані так само, як і згортковий рівень. Однак цей фільтр не містить вагових коефіцієнтів, на відміну від шару згортки [19].

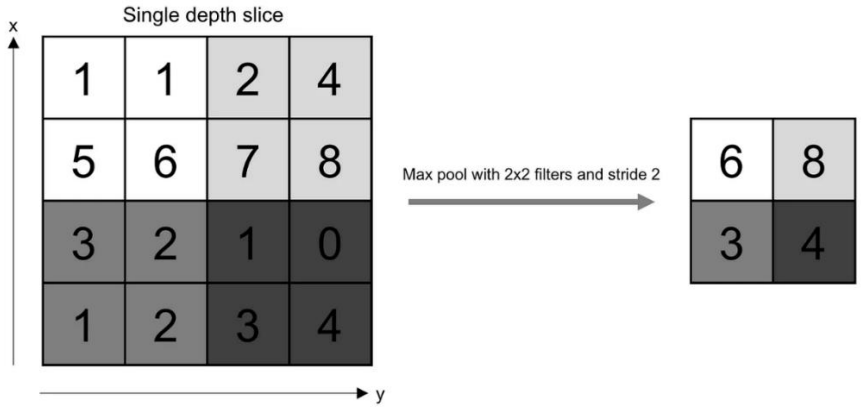



Рисунок 2.7 – Агрегувальні процеси

Щоб визначити розмір вихідного обсягу у нас є така формула 1.5:

$$w_{out} = \frac{w-f}{s}, \tag{1.5}$$

де $f$ – об’єднуюче ядро просторового розміру;

$s$ – крок.

Агрегувальний шар виконує функцію про зменшення шумів в зображенні, але також має властивість про втрачу деяких параметрів.

При згортанні карт функцій завжди зберігаються найбільш домінуючі функції (або інформація) на кожному етапі пулу. Операція об’єднання виконується шляхом вказівки об’єднаної області розмір і крок операції, подібної до операції згортки [10].

На рисунку 2.8 який зображений нижче ми можем побачити усі складаючі Pooling Layer.

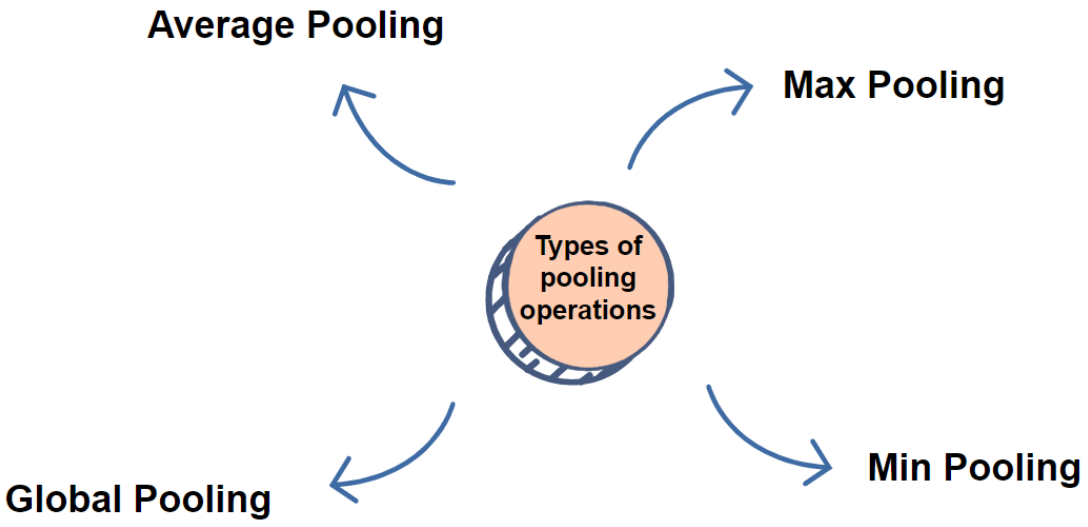



Рисунок 2.8 – Загальний вид Pooling Layer

Об’єднання взагалі існує для того, щоб підсумувати функції, а потім розмір цих функцій буде визначати фільтр об’єднання.

## 2.4 Повноз'єднані шари

Повноз'єднані шари часто зрівнюють з лінійними шарами, тому що кожний нейрон такий як вхідний, з'єднується і використовується в

нейронних мережах (рисунок 2.9). Повноз'єднані шари з'єднують кожен нейрон одного шару з кожним нейроном наступного шару [17].

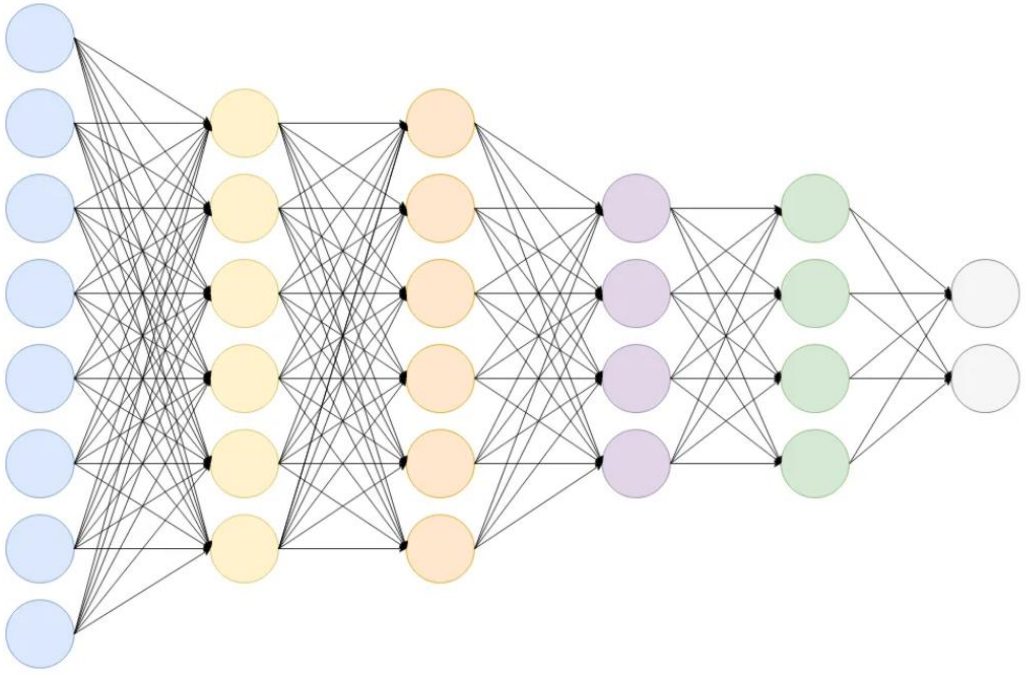


Рисунок 2.9 – Загальний вигляд повноз’єднаних шарів

Повноз'єднані шари визначать три параметри:

– розмір пакету(batch size);

– слідкує за кількістю вихідних розмірів та вхідних.

Щоб обчислити вхідний та вихідній розмір існує формула 1.6:

$$w_2 = \frac{w_1 - F + 2P}{S} + 1\ , \qquad (1.6)$$

де, $w_1$ – вхідна висота, ширина;

$F$ – висота та ширина ядра;

$P$ – відступ;

$S$ – крок;

$w_2$– вихідна висота та ширина.

Вхідними даними мається на увазі, що це вихідні дані останнього

об’єднання, які входять в повноз'єднані шари, цим самим мають властивість вирівнюватись (flattened). Приклад вирівнювання ми можем побачити на рисунку 2.10 який буде зображено нижче.

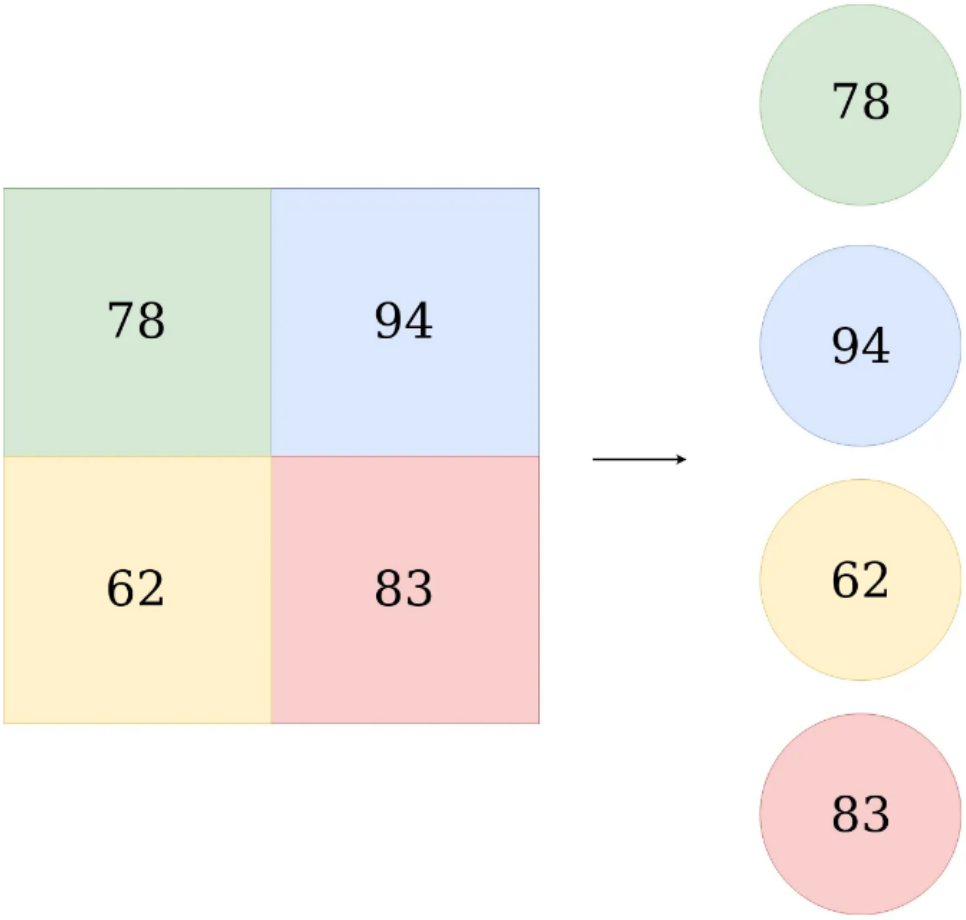



Рисунок 2.10 – Зображено процес вирівнювання

Вирівнювання (flattened) – тривимірна матриця для шару об’єднання, після чого проводиться її обрахунок для розгортання значень вектору. Зазвичай вхідні матриці доповнюються нулями, щоб ядро могло рівномірно рухатися по матриці, а результуюча матриця мала потрібний розмір. Таким чином, P=1 означає, що всі сторони вхідної матриці доповнено 1 шаром нулів, як на (рисунок 2.10), із пунктирними додатковими елементами навколо вхідної (синьої) матриці. Ядро рухається по матриці на 1 піксель за раз (рисунок 2.9). Отже, кажуть, що крок дорівнює 1. Ми можемо збільшити

крок до 2, щоб ядро рухалося по матриці на 2 пікселі за раз. Це, у свою чергу, вплине на розміри вихідного тензора та допоможе зменшити переобладнання [5].

## 2.5 Функції активації

Функції активації визначає, чи має бути активований нейрон чи ні. Це означає, що він використовуватиме деякі прості математичні операції, щоб визначити, чи є вхід нейрона в мережу релевантним чи ні в процесі прогнозування [6].

Існують такі функції активації:

– лінійна функція активації (Linear Activation Function);
– двійкова крокова функція (Binary Step Function);
– нелінійні функції активації (Non–linear Activation Functions).

Лінійна функція активації часто може ще називатися функцією активації тотожності, яка буде пропорційна входу. Лінійна функція має свій діапазон який має значення нескінченості. Суть цієї функції що вона буде додавати суму даних, на вході коли ми отримаємо і поверне результат, який ми отримаємо. Ця функція має такий вигляд, який буде зображено на рисунку 2.11.

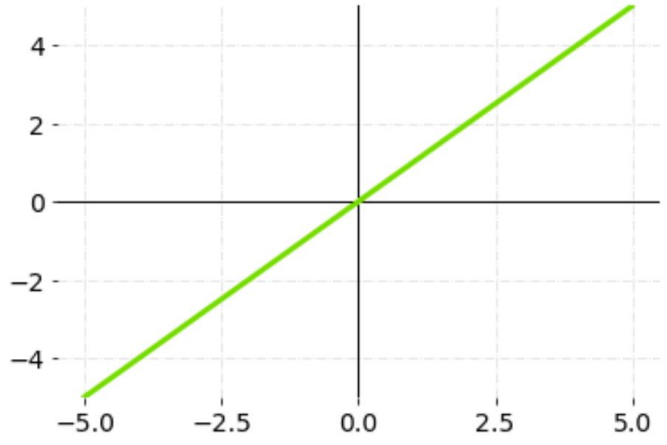



Рисунок 2.11 – Загальний вигляд лінійної функції

Якщо брати математичне зображення, то це можна представити за допомогою формули 1.7:

$$f(x) = x\,. \tag{1.7}$$

Діапазон активацій, це те що робить лінійна функція. Ми маємо можливість з’єднати певну кількість нейронів і коли є кілька активацій, то тоді можна розрахувати максимум на цій роботі що була пророблена. Використання функцій активації в першу чергу має велике значення на всю роботу мережі, тобто якщо ми би не застосовували її, то ми не могли би розпізнавати нелінійні закономірності. Коли ми зрозуміли як працює лінійна функція, то перейдемо до двійкової крокової функції.

Двійкова крокова функція (Binary Step Function) – визначає, коли і як повинен бути активований нейрон у двійковій кроковій функції. Її загальна робота визначається в тому, що вона порівнює вхідне значення з пороговим. Нейрон може активуватися в тому разі, коли вхідне значення яке у нас є, перевищує порогове значення. Ця функція зображена на рисунку 2.12.

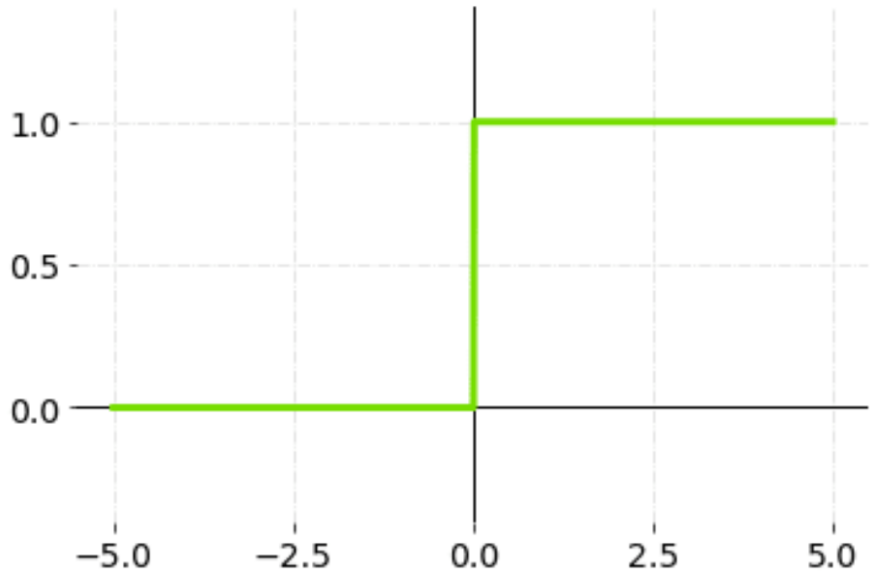



Рисунок 2.12 – Двійкова крокова функція

Плюси та мінуси цієї функції полягають в тому, що її неможливо використовувати для задач класифікації одразу для декількох класів, та може ускладнювати процедуру зворотного поширення тим, що градієнт функції має значення нуля. Це зменшить невпевненість дослідників при виборі моделі машинного навчання [14].

Нелінійні функції активації (Non-linear Activation Functions) – в останній час вони є найбільш використовуваними функціями активації. Вони можуть і спрощують адаптацію моделі нейронної мережі до великої кількості даних та також може розрізняти результати. За допомогою цієї функції вона має можливість складати декілька шарів нейронів, які проходять через безліч рівнів. Основною та більш популярною нелінійною функцію активації вважають ReLU. Її властивість складається з того, що вона оброблює дані та може більш швидше навчати мережу за усі інші функції активації котрі існують і також використовуючи нелінійність, при цьому використовувати ті самі дані, складні моделі. Як виглядає ця функція можна побачити на рисунку 2.13, який зображено нижче.

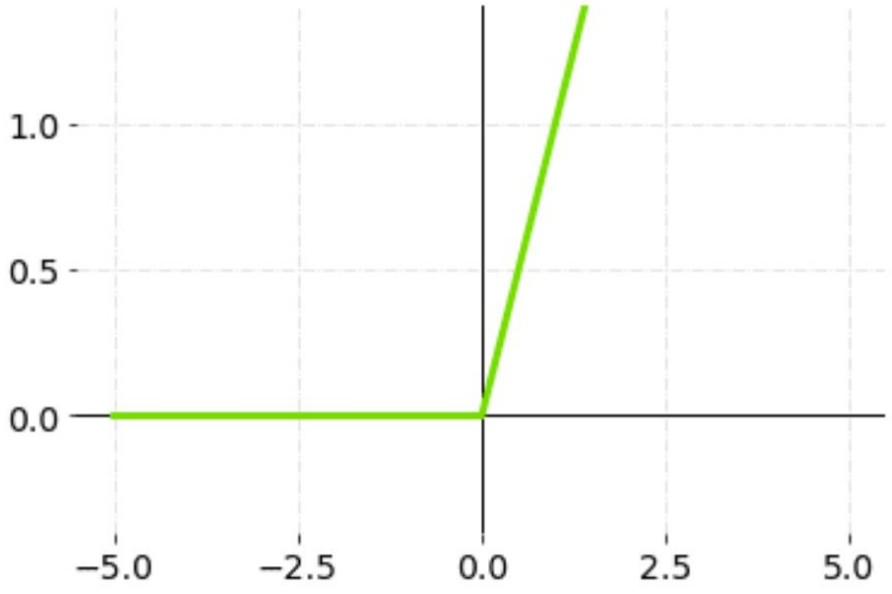



Рисунок 2.13 – Функція активації ReLU

Глибокі згорткові нейронні мережі з ReLU навчаються в кілька разів швидше, ніж їх еквіваленти з одиницями [11].

Карти згортки проходять через нелінійний рівень активації, такий як Rectified Linear Unit (ReLu), який замінює від'ємні числа відфільтрованих зображень нулями [20].

Зараз ми розглянемо формулу 1.8 для знаходження ReLU, яка має такий вигляд:

$$y = f(x) = \max(0, x)\,, \tag{1.8}$$

де $y$ – початковий рівень;

$x$ – елемент вхідних даних.

Ця формула показує нам, що вона може усувати негативні значення, та придавати їм значення, яке нульове, при цьому всьому реалізує перетворення для усіх інших значень.

# 3 РЕАЛІЗАЦІЯ МОДЕЛІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

## 3.1 Підготовка даних

В цій роботі ми з вами будемо використовувати TensorFlow. TensorFlow – це відкрита для користування бібліотека машинного навчання. Ця бібліотека може тренувати та строїти нейронні мережі. Логотип TensorFlow виглядає на рисунку 3.1.


Рисунок 3.1 – Логотип TensorFlow

TensorFlow проста у використанні. В інтернет ресурсах існує багато підручників та джерел, крім офіційної документації. Її можна зрозуміти, якщо немає певних навичок з цієї сфери навчанні. Якщо виникає питання і треба знайти відповідь, то можна легко погуглити і знайти її. Це дуже популярна в наш час бібліотека. Наприклад в ній розглядається певний метод і відразу можливо побачити, як цей метод реалізувати за допомогою коду.

Процес навчання в TensorFlow полегшило те, що він використовує TensorBoard. TensorBoard можна використовувати для перевірки навчання мережі та структури. Це уявляє собою набір інструментів, та щоб більш розуміти, як він виглядає, то подивимось на рисунку 3.2.

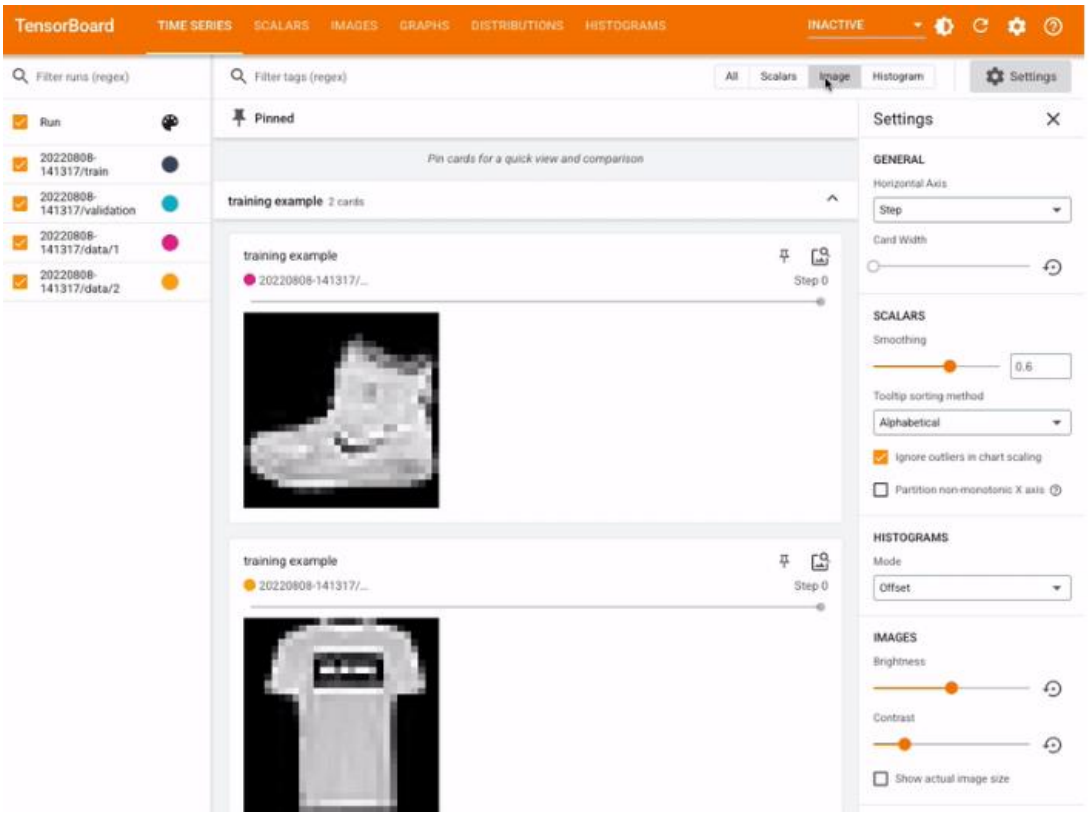



Рисунок 3.2 – Загальне уявлення TensorBoard

Ми використовуємо мову програмування Python та ще одну бібліотеку як Keras. Вона також взаємодіє з нейронними мережами і гарно працює в парі з TensorFlow.

Для початку, перед тим як ми з вами почнемо розробляти згорткову нейронну мережу нам потрібно підготовити дані. Для цього потрібно залучити та зробити імпорт даних Fashion MNIST. Цей набір даних зберігає в собі 70 000 тисяч зображень і ці зображення мають вигляд відтінку сірого. Подивимося нижче на рисунку 3.3.

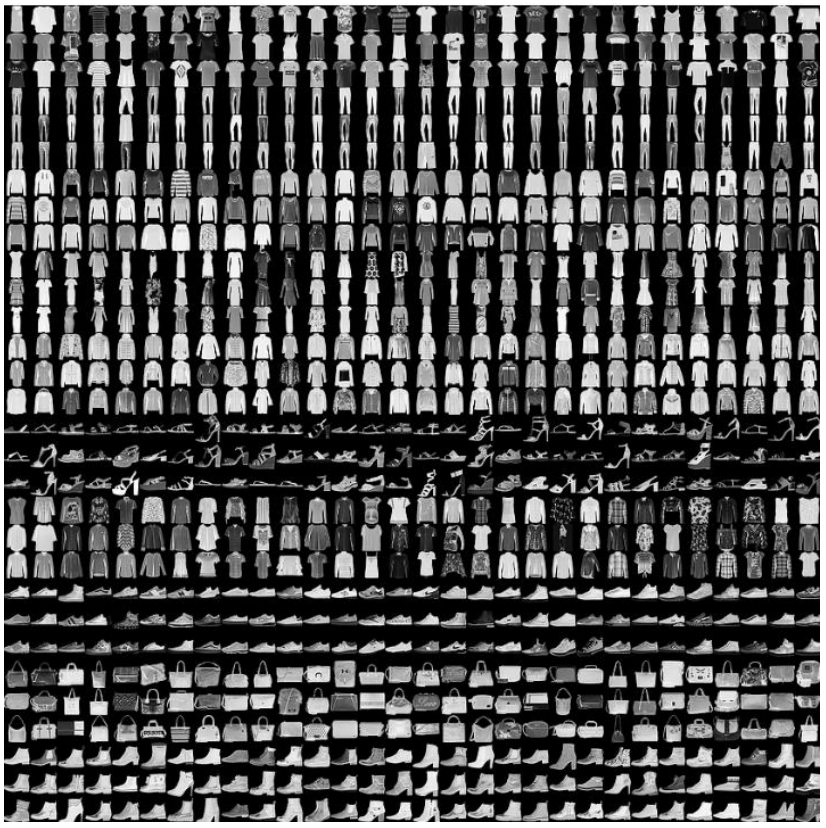


Рисунок 3.3 – Приклад набору даних Fashion MNIST

Як вже було сказано вище, що ми будемо використовувати 70 000 тисяч зображень, але треба уточнити, що 60 000 тисяч ми будемо використовувати з вами для навчання мережі, та 10 000 тисяч для її оцінки та тестування, щоб зробити висновок того, як гарно чи погано ми змогли зробити згорткову мережу для класифікації зображень.

Щоб завантажити ці дані, які ми будем використовувати, потрібно написати код, який зображено на рисунку 3.4. Після того як цей код запрацює ми побачимо наступний крок, який зобразимо на рисунку 3.5.

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Рисунок 3.4 – Код для імпортації даних

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 0us/step
40960/29515 [========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [================================================================================================]
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

Рисунок 3.5 – Процес загрузки даних

Далі коли ми з вами побачили, як виглядають наші з вами дані та зрозуміли, як їх почати використовувати, то можемо перейти далі, а саме до того, що всі дані які ми маємо, діляться на 10 категорій. Як це виглядає, зображено на рисунку 3.6.

| Label | Class |
|---|---|
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

Рисунок 3.6 – Зображено 10 категорій на які діляться дані та зображення

## 3.2 Перенавчання та недонавчання

Недовчання в нейронній мережі може відбуватись, коли існують можливості при яких відбувається покращення даних. Наприклад, якщо в

нас мережа недостатньо потужна, або навчалась недостатньо часу для її досконалої реалізації. Або якщо тренування мережі проходить достатньо забагато, то може відбуватися факт перенавчання. Це може відбуватися, коли тренування проходять занадто довго, то є вірогідність, що мережа може почати перенавчати наші дані. В цих ситуація, нам потрібно прагнути дотримання балансу. Ці всі моменти щоб уникнути цього ми розглянемо нижче.

Щоб уникнути перенавчанню, або можна сказати іншими словами переоснащенню, нам потрібно дані, які ми будемо використовувати, вони мають охоплювати весь діапазон наших з вами даних, котрі в подальшому буде обробляти наша за вами мережа. Є факт того, що в нас можуть буди додаткові дані, але їх використання може бути в тому випадку, коли вони всі охоплюють нові випадки.

Ми можемо використовувати з вами термін – регуляризація. Що це значить? Цей термін може і накладає обмеження на інформацію, тобто на її кількість та тип, яку зберігає наша мережа. Під час роботи з мережею, може виникнути така дія, що регуляризація дозволить запам’ятати невелику кількість даних. З цього можна зробити висновок, що процес оптимізації даних змусить спочатку зосередитись на даних, котрі можуть бути узагальненими. Далі ми з вами розглянемо роботу методів регуляції та будем використовувати їх для покращення класифікації зображень.

Ми використовуємо з вами TensorFlow, тому що він вважається більш ефективним в тому разі, коли треба працювати з великою кількістю даних.

Один з способів, щоб запобігти перенавчавчання і він являється самим простим, це почати працювати з невеликою мережею. Для початку цього буде достатньо, щоб зрозуміти як це працює. Тобто, якщо казати більш простіше, ми беремо мережу в якій є невелика кількість даних та параметрів, які навчаються. Але, якщо ми з вами не знаємо розмір та масштаб мережі, то краще буде в цій ситуації почати з використання невеликої кількості шарів та параметрів, які нам потрібно навчити. Потім за часом, ми авжеж

можемо з вами додавати дані та розширювати нашу мережу та додавати більше шарів.

Для початку, візьмемо код, де покажемо процес зниження швидкості навчання. Це в нас буде рисунок 3.7.

```
lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
  0.001,
  decay_steps=STEPS_PER_EPOCH*1000,
  decay_rate=1,
  staircase=False)

def get_optimizer():
  return tf.keras.optimizers.Adam(lr_schedule)
```

Рисунок 3.7 – Код, де зображено зниження швидкості навчання

Після цього коду зробимо графік та відобразимо, щоб можна було це побачити, як це працює на рисунках 3.8 та 3.9. Давайте подивимось на нього і почимо, що наші криві лінії, які зображені на них, вони відображені вниз, тобто с більшого значення ми бачимо, що отримуємо меньше.

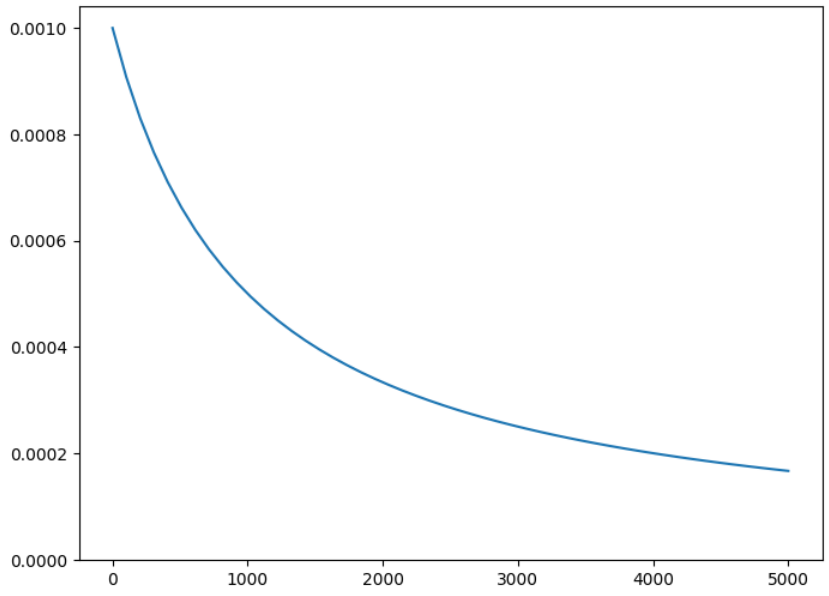



Рисунок 3.8 – Графік відображення зниження швидкості навчання

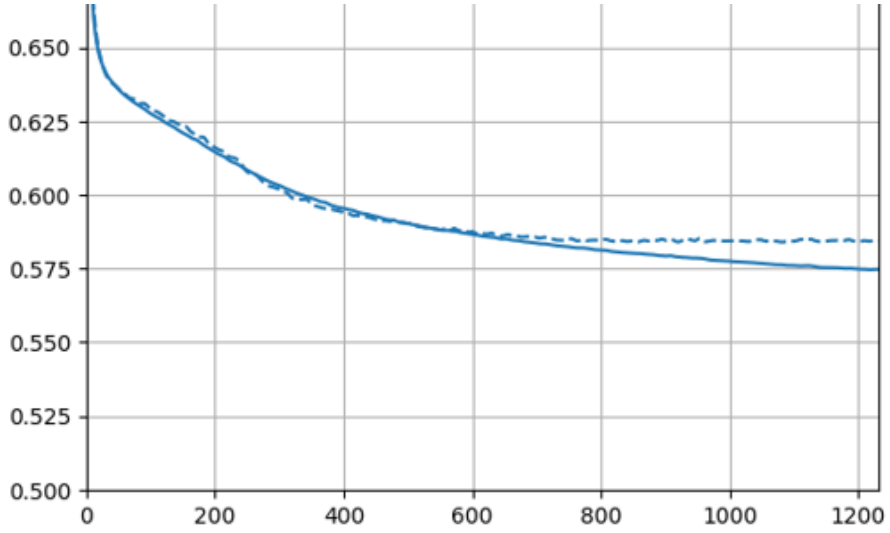



Рисунок 3.9 – Графік відображення зниження швидкості навчання

На рисунку 3.9 ми з вами можемо побачити, що в нас існують 2 лінії, це зумовлюється тим, що пунктирної лінією показує те, що існують втрати даних при перевірці. Застосування таких алгоритмів може значно покращити точність подальшого застосування згорткових нейронних мереж [13].

Помилка навчання є функцією різниці між прогнозованим результатом і фактичним результатом для кожної точки даних навчання. Помилка перевірки – це функція різниці між прогнозованим результатом і фактичним результатом для окремого набору даних перевірки з використанням мережі, що навчається [8].

На рисунку 3.10 та рисунку 3.11 ми можемо побачити, як виглядає помилка на графіку.

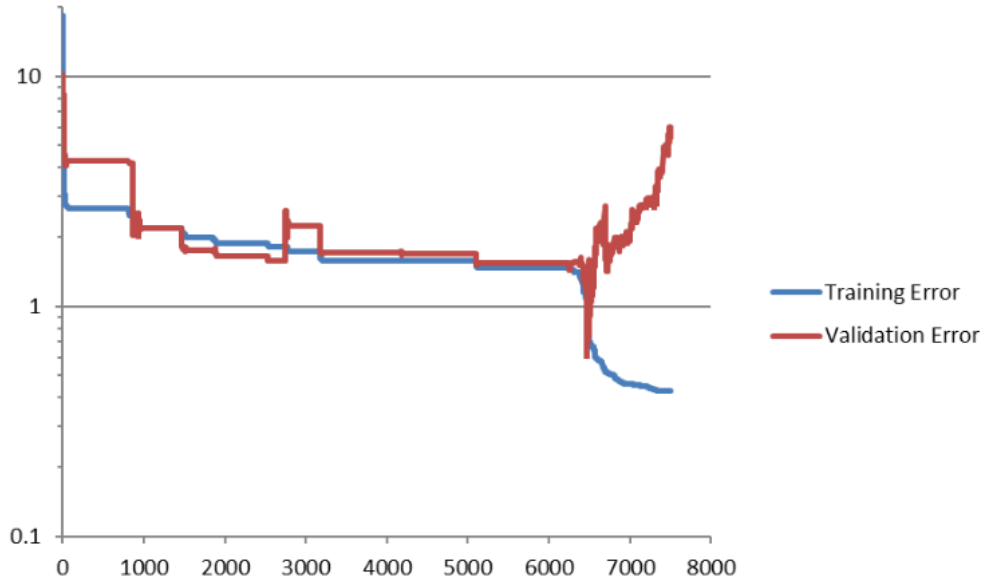



Рисунок 3.10 – Як виглядає помилка на графіку

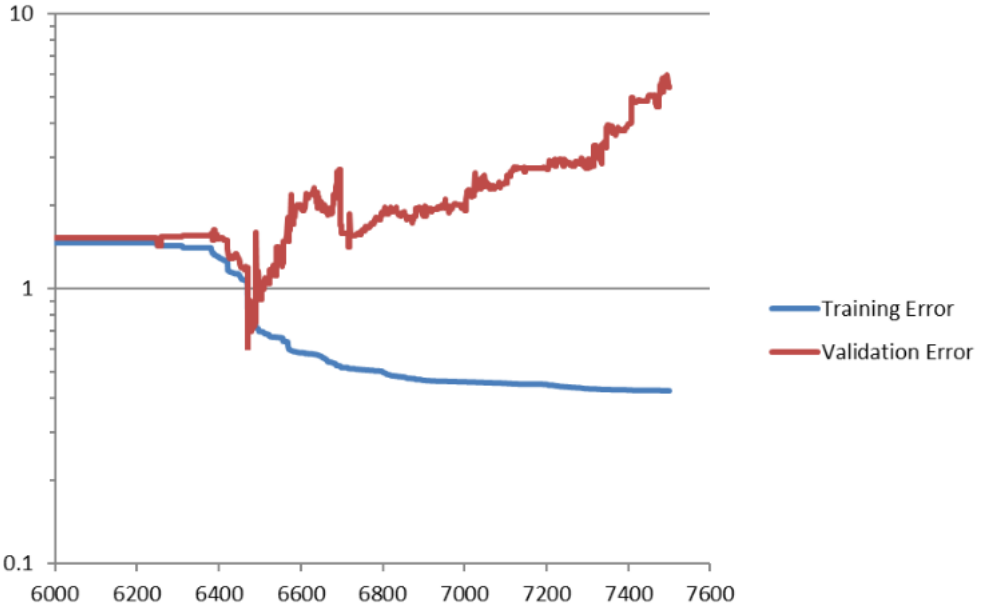



Рисунок 3.11 – Збільшення вигляду помилки на графіку

На рисунку 3.11 ми можемо з вами побачити, що розбіжність між кривими лініями становиться більше, це зумовлюється тим, що ми перетренували мережу. Тобто помилка навчання в нас стає меншою, а зростає помилка навчання. Це все може означати, що мережа гарно та досконало вивчає нові дані, але не узагальнює їх.

## 3.3 Реалізація згорткової нейронної мережі

Як вже було сказано раніше, ми будемо використовувати базу даних Fashion MNIST, яка містить в собі 60 тисяч зображень, кожне зображення буде у вигляді 28x28 пікселів. Всі ці зображення будуть ділитися на 9 категорій як це зображено на рисунку 3.12.

Для початку, щоб навчити нашу мережу, нам потрібно обробити наші зображення.

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

Рисунок 3.12 – Код для підготовки зображення

Після роботи коду, який зображено на рисунку 3.12 ми зможемо отримати результат, який буде зображено на рисунку 3.13 нижче.

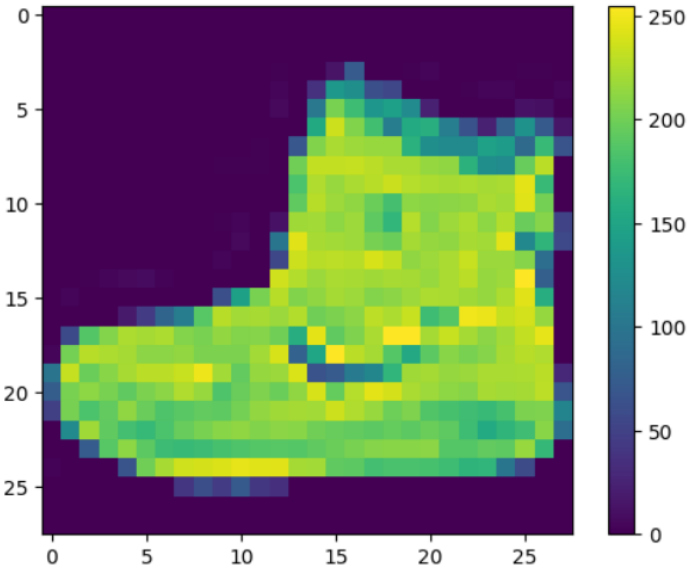



Рисунок 3.13 – Результат роботи коду

Як можно побачити на рисунку 3.13, наші значення ми масштабуємо з поділом від 0 до 1 та до 255, це робиться для того, щоб в подальшому передати ці значення в модель нейронної мережі. Наступним кроком нам потрібно переконатися що всі зображення мають правильний формат для того щоб збудувати та навчити нашу мережу. Для цього нам допоможе написаний код, який буде зображено на рисунку 3.14.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Рисунок 3.14 – Виводимо перші 25 зображень

Після цього коду, коли він запрацював ми маємо результат котрий буде зображено на рисунку 3.15 нижче.


Рисунок 3.15 – Виведено 25 зображень з розпізнаванням класу

Наступним кроком для розробки нашої згорткової нейронної мережі нам потрібно построїти та налаштувати модель даних, а саме її шари. Взагалі самим головним вважаються шари в нейронній мережі (рисунок 3.16).

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

Рисунок 3.16 – Параметри слоїв

Тут ми використовуємо Flatten для того, щоб перетворити формат зображення, а саме у нас є двомірний масив, який складається з 28x28 пікселів і цим самим ми отримуємо одновимірний масив, де наші пікселі будуть множитись та в результаті буде отримано 784 пікселів. Цей шар нам потрібний тільки щоб переформувати наші усі дані.

Після вирівнювання пікселів мережа складається з послідовності двох шарів. Це тісно пов'язані або повністю пов'язані нейронні шари. Перший шар Dense має 128 вузлів (або нейронів). Другий (і останній) шар повертає масив довжиною 10. Кожен вузол містить оцінку, що вказує, що поточне зображення належить до одного із 10 класів [7].

Щоб наша модель була готова до використання її ще потрібно скомпілювати. Компіляція моделі має такі функції:

– функція втрат – це нам потрібно щоб дізнатися наскільки досконало та точна модель під час навчання;

– оптимізатор – модель бачить дані та на основі цих даних вона оновлюється;

– метрики – їхня функціональність залежить від того, що вони використовуються, коли нам потрібно моніторити етапи навчання а потім

проводити тестування;

На рисунку 3.17 ми можемо побачити процес компіляції моделі.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Рисунок 3.17 – Компіляція моделі

Щоб навчити нашу модель нам потрібно зробити наступні кроки, а саме:

– вивести зображення, тобто подати дані до моделі;

– навчити модель пов'язувати зображення що ми отримали та відповідні мітки до них;

– модель робить свій прогноз щодо тестового набору, треба переконатися щоб мітки відповідали до масивів.

Для початку навчання моделі нам потрібно визвати метод який буде зображено на рисунку 3.18.

```
model.fit(train_images, train_labels, epochs=10)
```

Рисунок 3.18 – Метод для навчання моделі

Далі буде зображено рисунок 3.19 для відображення результатів навчання.

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4986 - accuracy: 0.8253
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3751 - accuracy: 0.8651
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3364 - accuracy: 0.8769
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3124 - accuracy: 0.8858
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2949 - accuracy: 0.8913
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2776 - accuracy: 0.8977
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2669 - accuracy: 0.9022
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2552 - accuracy: 0.9046
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2463 - accuracy: 0.9089
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2376 - accuracy: 0.9117
<keras.callbacks.History at 0x7f5f2c785110>
```

Рисунок 3.19 – Результат після навчання

На рисунку 3.19 ми можемо побачити, що наша модель успішно пройшла навчання та загальний результат точності досягає 0,91 або 91%. За допомогою графіку відобразимо цей результат, що зображено на рисунку 3.20.

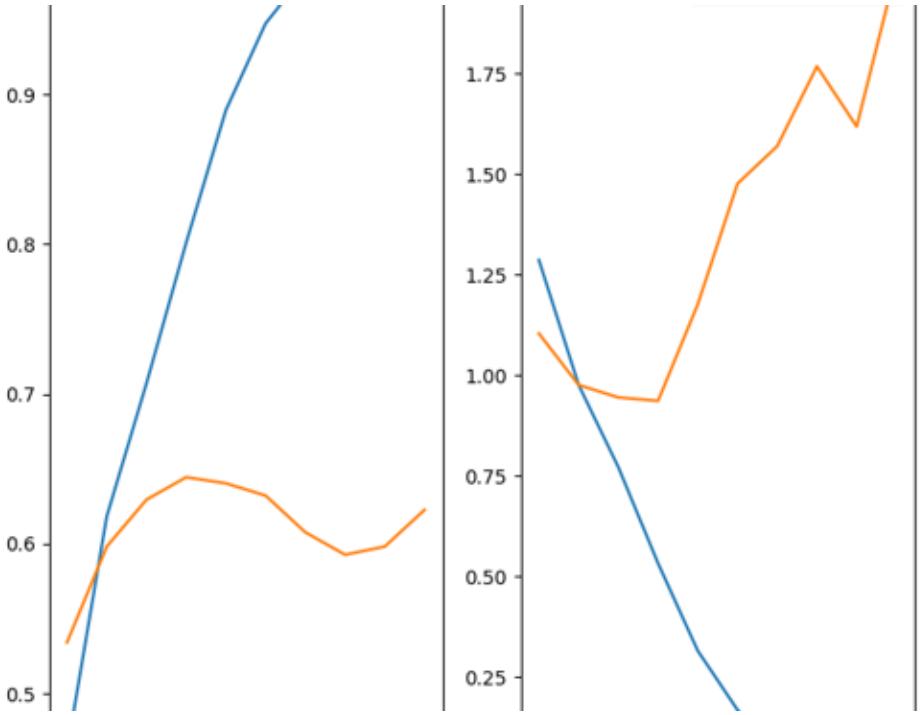



Рисунок 3.20 – Відображення графіку

## 3.4 Отримання результатів дослідження та розробки

Ми змогли успішно розробити класифікацію зображень за допомогою згорткових нейронних мереж та навчити її. Тепер давайте виведемо результати котрі ми змогли отримати під час цих процесів. Для цього нам потрібно такий код, який зображено на рисунках 3.21 та 3.22.

```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Рисунок 3.21 – Код для отримання результатів

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

Рисунок 3.22 – Код для отримання результатів

Після роботи цього коду можна побачити на рисунку 3.23 результати дослідження.

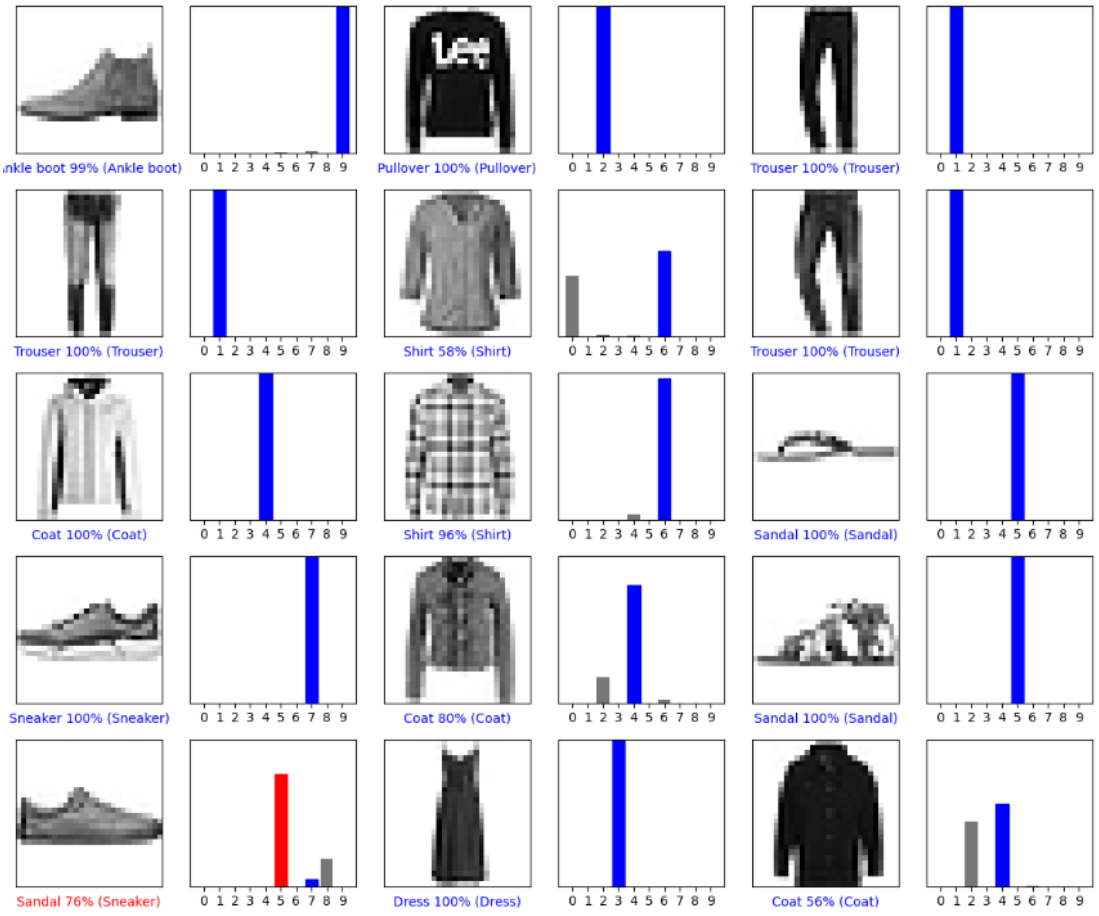



Рисунок 3.23 – Результати дослідження

На рисунку 3.23 можна побачити результати, котрі ми отримали. Після цього зробимо прогнози роботи. Пропоную взяти декілька зображень. Наприклад те зображення що буде на рисунку 3.25, але для початку нам потрібно написати код який можна побачити на рисунку 3.24.

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

Рисунок 3.24 – Код для прогнозу

Після цього отримуємо рисунок 3.25.

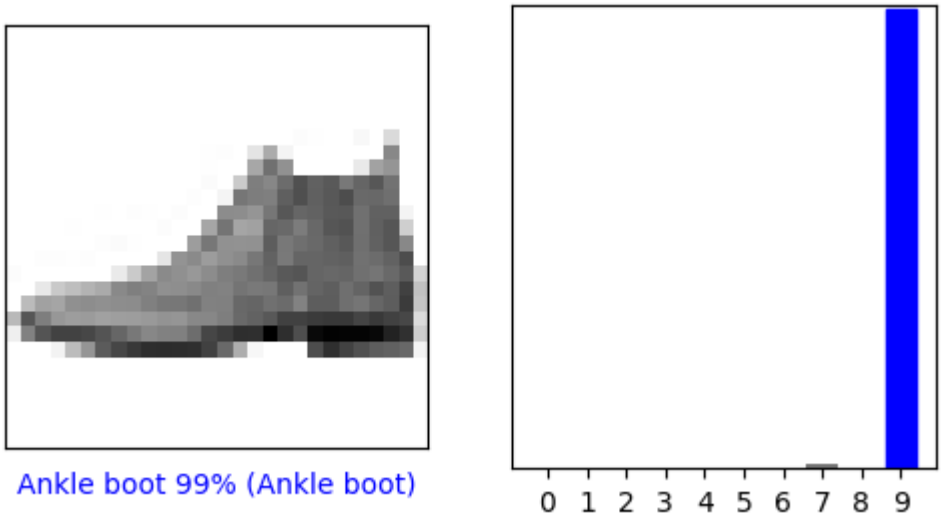



Рисунок 3.25 – Результат роботи коду

На рисунку 3.25 ми можем побачити синій стовбець. Він показує мітки правильних прогнозів. Цей прогноз нам показав 99% результат розпізнавання. Давайте повернемося до рисунку 3.23 та бачим що там є і

красний стовбець. Пропоную дізнатися що він значить та для цього напишем відповідний код котрий зображено на рисунку 3.26.

```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

Рисунок 3.26 – Результат роботи коду

Для виведення  зображення будемо дивитися на рисунок 3.27.

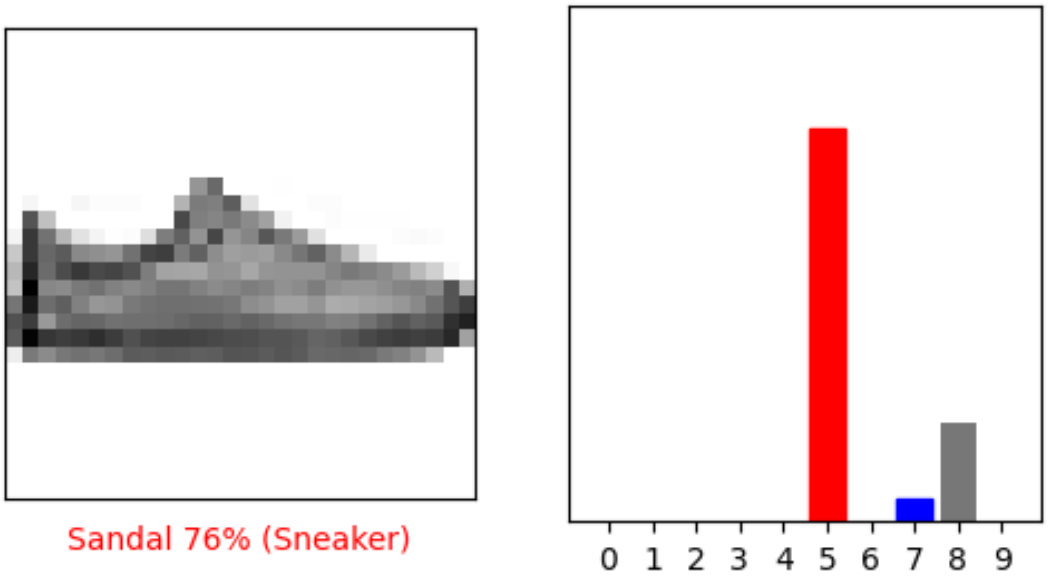



Рисунок 3.27 – Демонстрування  прогнозу

На рисунку 3.27 ми можемо побачити, що модель може робити помилки та виводити неправильні прогнози. В такому випадку бачимо, що процент розпізнавання склав 76%.

## 3.5 Експерименти та оцінка ефективності

Модель навчається на навчальних даних (без будь-якого шуму), а перевірка виконується на тестових даних (оскільки я зосереджуся лише на аналізі результатів перевірки без виконання остаточних тестів). Модель була навчена в 20 епохах і розмірі партії 64. Значення точності та втрат для навчання та перевірки представлені на графіках нижче [15] на рисунках 3.28 та 3.29.

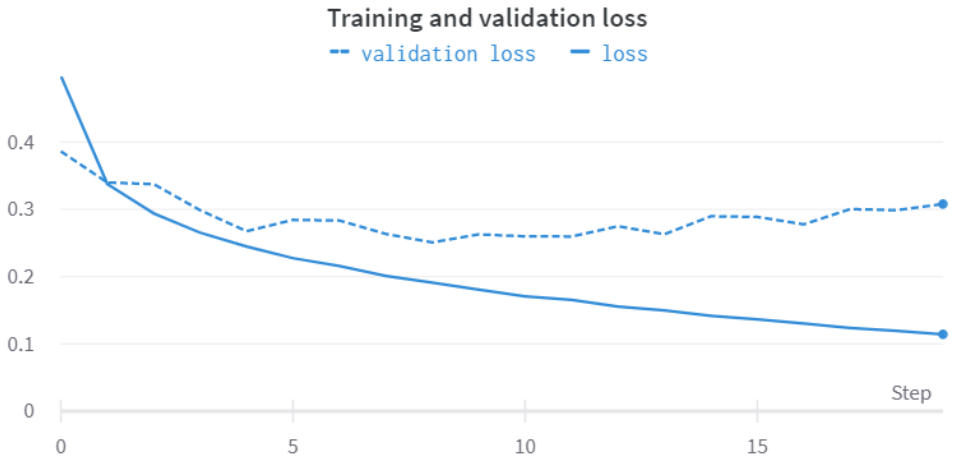



Рисунок 3.28 – Втрата навчання та перевірка

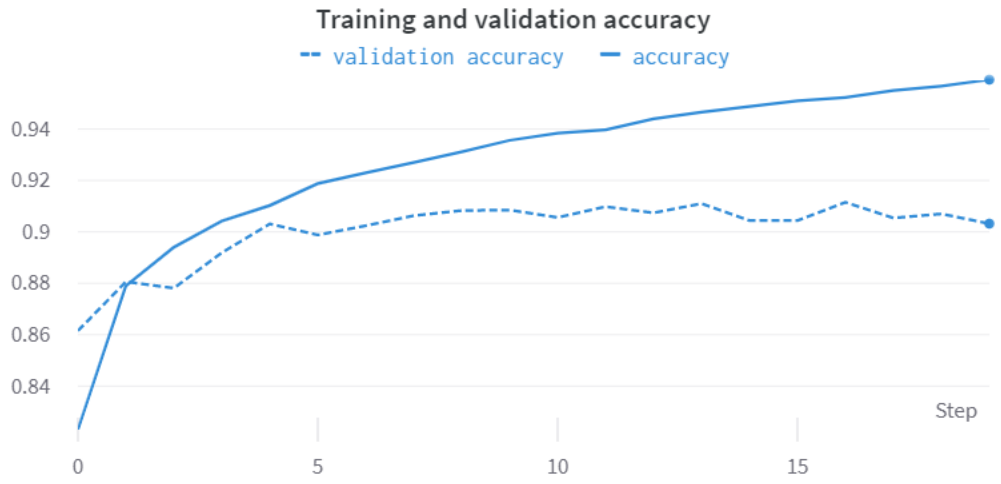



Рисунок 3.29 – Перевірка точності мережі

Судячи з наведених результатів вище, можна побачити, що мережа переповнена, та точність навчання складає 0,9592, але точність перевірки після навчання показує нам 0,9083. Функція втрат на рисунку 3.29 показує нам такий результат 0,2508, но потім вона стає збільшуватись.

Далі я пропоную взяти 10 зображень и додати до них ефект шуму, тобто зробити шум-фактор і вивести його на рисунок 3.30.

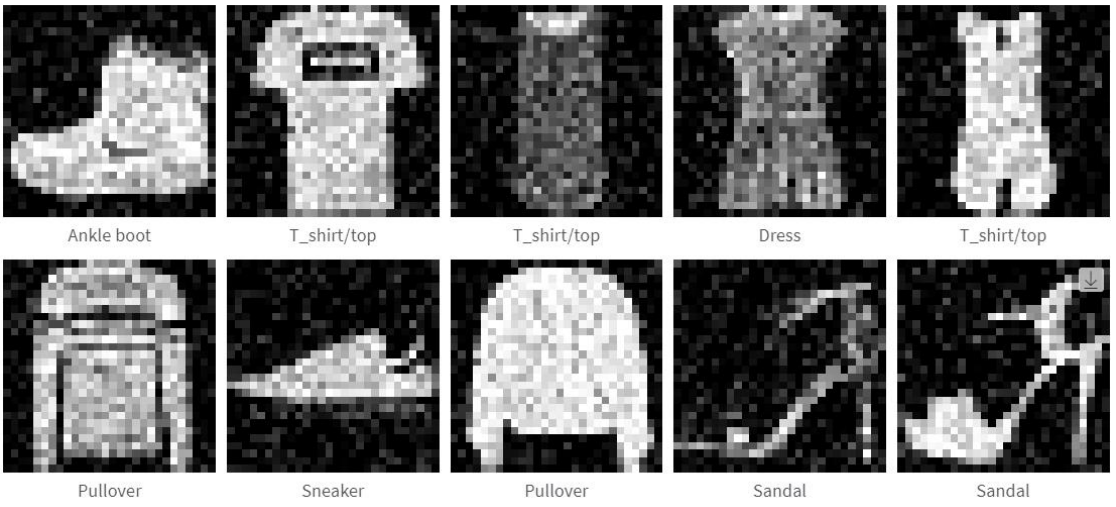



Рисунок 3.30 – Зображення з шум-фактором

Після цього, як ми вивели з вами 10 зображень з шум-фактором, пропоную відобразити це на графіках, як вони будуть функціонувати в мережі, це показано на рисунках 3.31 та 3.32.

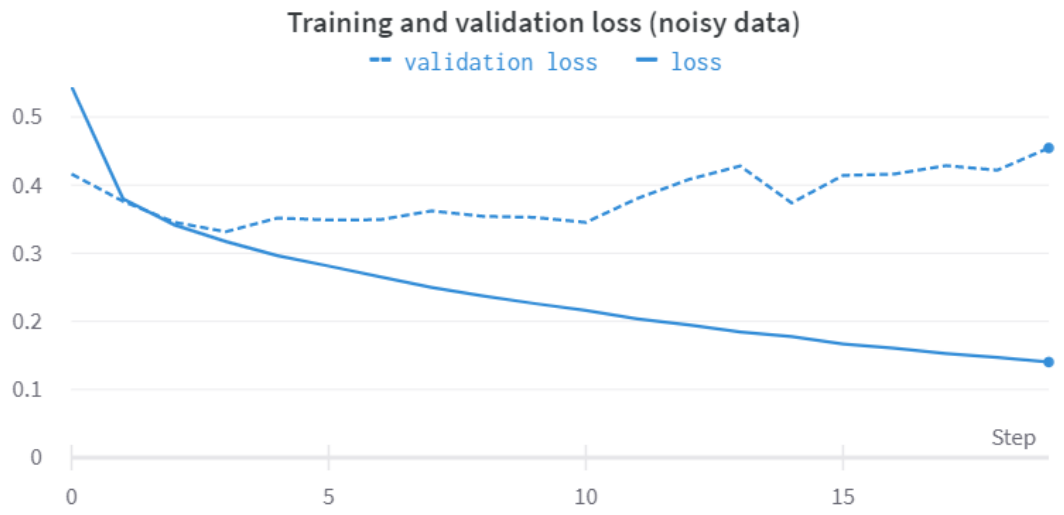



Рисунок 3.31 – Перевірка зашумлених даних

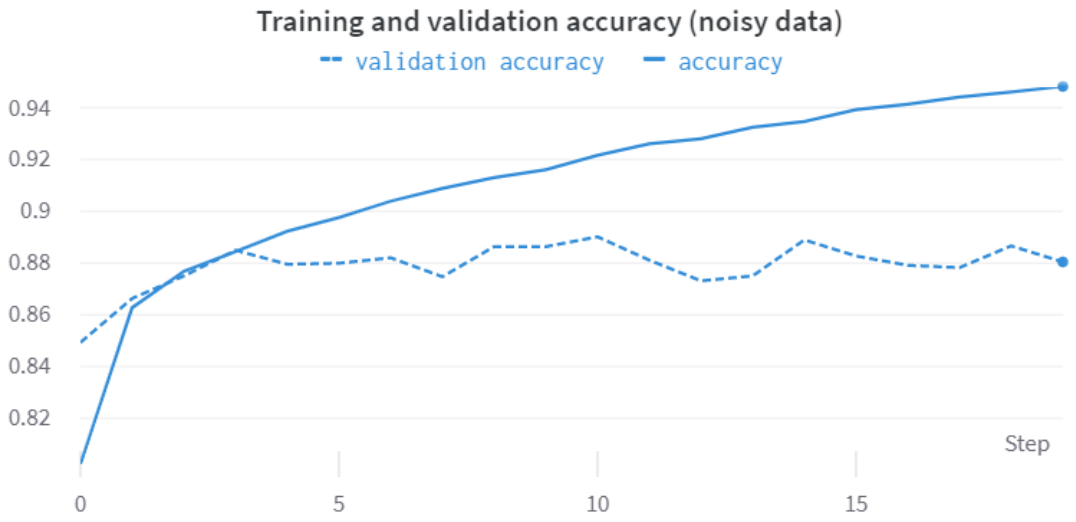



Рисунок 3.32 – Точність навчання зашумлених даних

Можна побачити, що результати котрі ми отримали, вони сталі гірше, ніж ті, шо в нас буди без шум-фактору, але це потрібно було перевірити та дослідити, як мережа поведе себе в цій ситуації.

Після цього можна зробити експеримент, як мережа буде себе поводити, коли ми будем використовувати випадаючі шари (Dropout layers). Іншими словами, коли наша мережа буде використовувати 2 шари вилучення, тобто, по одному після кожному згортковому шару. Це ми

можемо побачити на рисунках 3.33, 3.34, як це буде виглядати.

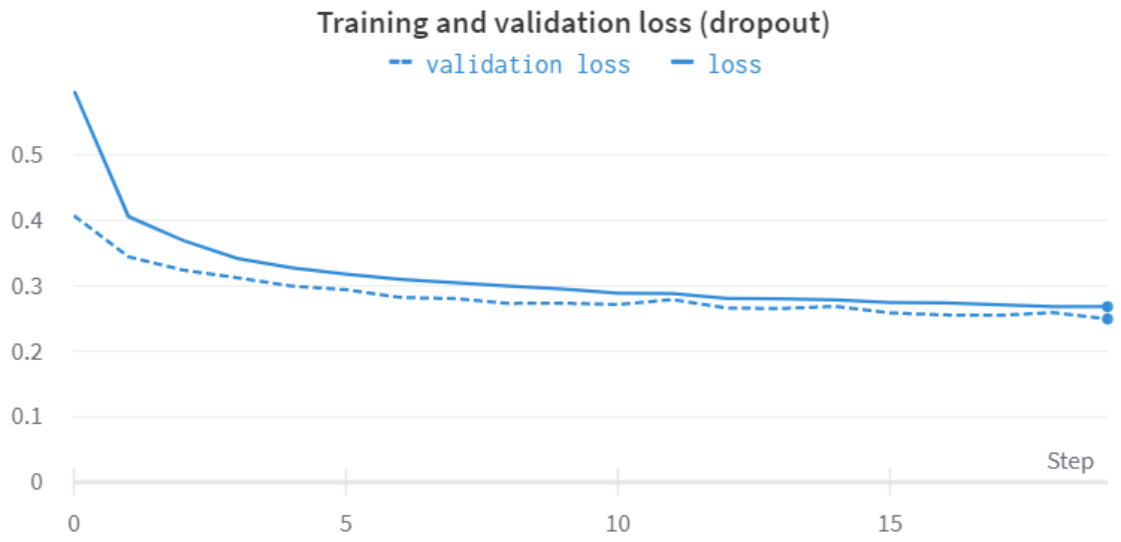



Рисунок 3.33 – Навчання та перевірка втрати за допомогою Dropout layers

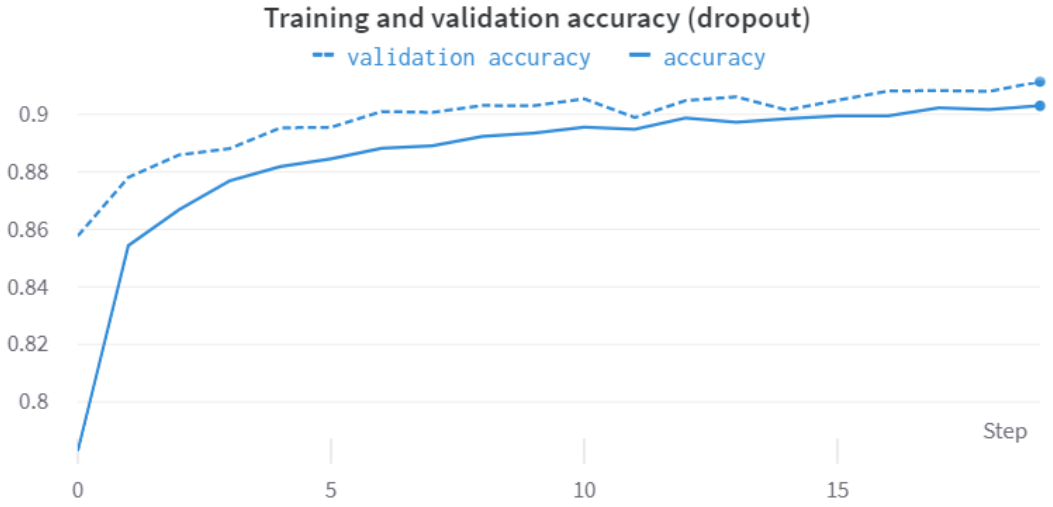



Рисунок 3.34 – Точність навчання за допомогою Dropout layers

Тим, що ми дослідили, як мережа може працювати та функціонувати за допомогою Dropout layers, нам показало, що це зменшило перенавчання мережі, та показало результат точності 0,9114, а втрати зменшились до значення 0,2494.

Тепер після цих всіх досліджень та експериментів, пропоную зробити

узагальнений експеримент, та підсумуємо всі показники, що ми отримали раніше, це ми відобразимо на рисунках 3.35, 3.36.

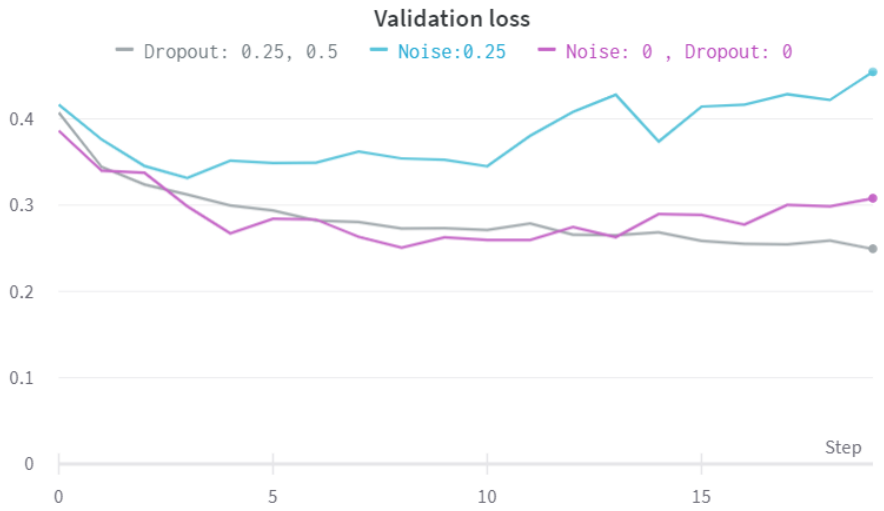



Рисунок 3.35 – Узагальнений графік втрат навчання

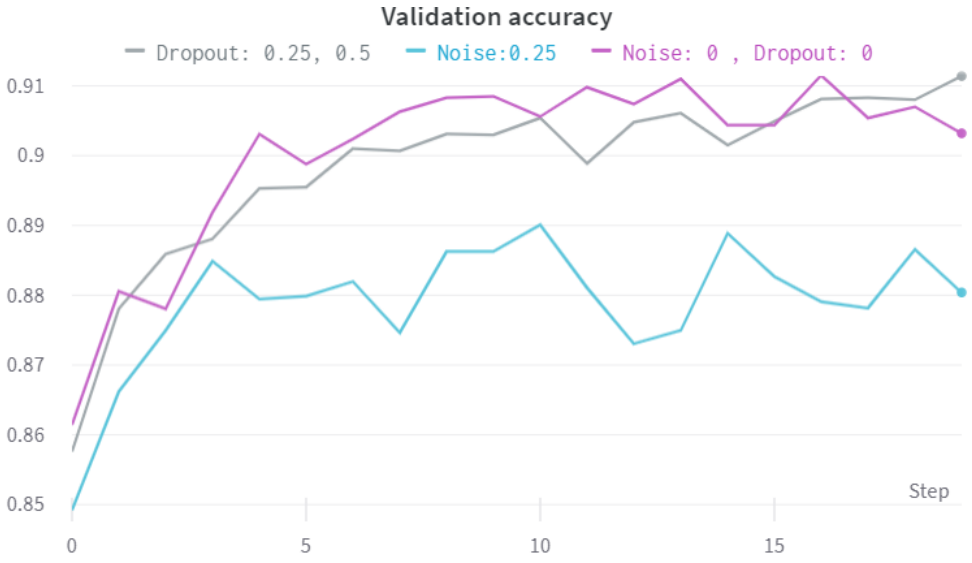



Рисунок 3.36 – Узагальнений графік точність перевірки

З цих досліджень можна зробити висновок, що ми змогли досягти найвищу точність перевірки нашої мережі, та найменшу втрату навчання мережі (рисунок 3.37).

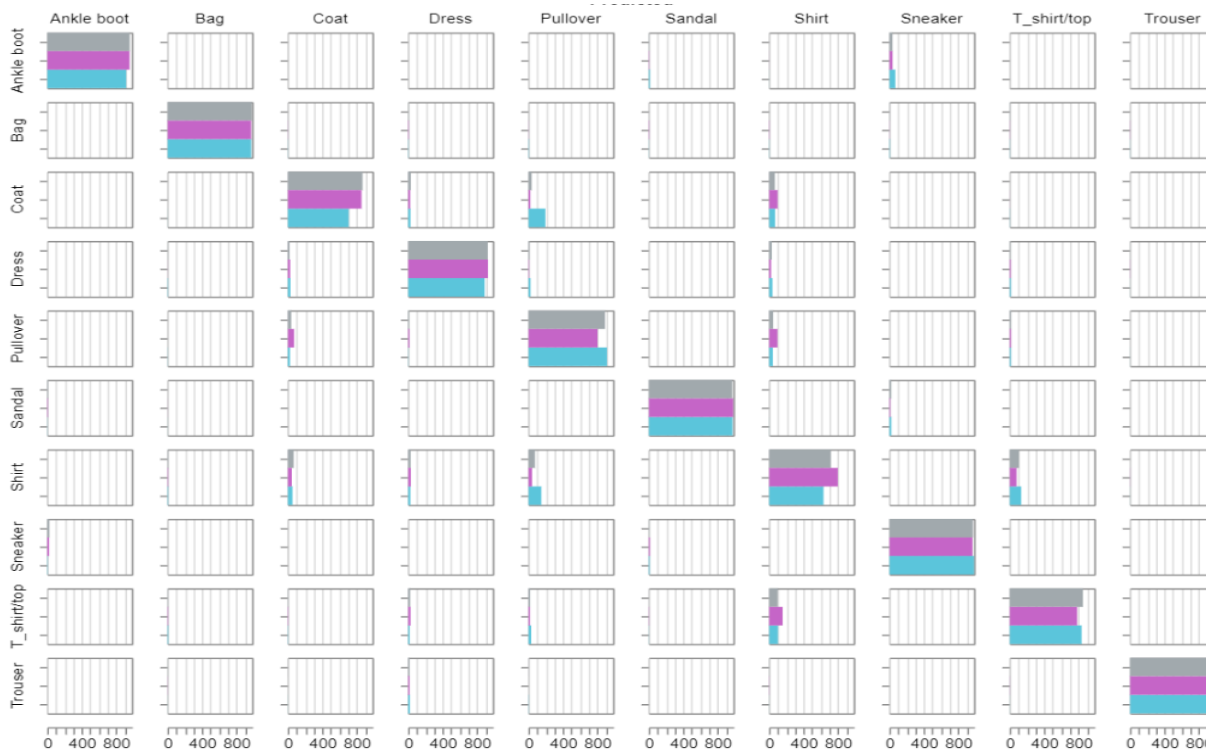



Рисунок 3.37 – Матриця класів

На рисунку 3.37 Ми можемо побачити та зробити висновок скільки елементів нашої мережі віднесло та спрогнозувало до правильно класу, а скільки елементів було віднесено до іншого.

# ВИСНОВКИ

В даній кваліфікаційній роботі було досліджено, як застосовувати і працювати з згортковими нейронними мережами. Було розглянуто більш детально та досконало, як вони построєні з середини і як вони застосовуються для класифікації зображень.

Так як на наш час класифікація зображень є популярною темою та має свій шанс на існування, то ми роздивилися та дослідили як класифікувати RGB зображення.

Було реалізації було використано бібліотеку з відкритим кодом для всіх користувачів в інтернет мережі TensorFlow та язик програмування Python.

Було також задіяно бібліотеку Keras, яка теж є у відкритому доступі. Вона гарно працює в парі з TensorFlow, як ми це і здійснили та продемонстрували в даній кваліфікаційній роботі.

За допомогою цих бібліотек розробили свою згорткову нейронну мережу та використовували набір даних Fashion MNIST. Цей набір даних складається з 70 000 тисяч зображень, а саме 60 000 – це основні, а 10 000 – для тестування та демонстрації згорткової нейронної мережі.

Провели ряд експериментів та зробили оцінку ефективності згорткової нейронної мережі, яку ми змогли отримати після вдалого навчання та тестування на її ефективність.

Експерименти нам показали, що ми змогли справитися з нашою метою роботи та достигли успіхів в класифікації зображень.

Результат, який ми змогли отримати, показав нам, що наші зображення, які ми загрузили за допомогою Fashion MNIST складає 0,91 або 91%.

Після такого результату, можна вважати що класифікація зображень є дуже цікавою сферою для пізнавання та опанування навичок з згорткової нейронної мережі.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. CNN Architecture – Detailed Explanation. URL: https://www.interviewbit.com/blog/cnn-architecture/ (date of access: 10.04.2023).

2. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8 (date of access: 10.04.2023).

3. Fully Connected Layer vs. Convolutional Layer: Explained. URL: https://builtin.com/machine-learning/fully-connected-layer (date of access: 13.04.2023).

4. Convolutional Neural Network Architecture. URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20CNN%20typically%20has%20three,and%20a%20fully%20connected%20layer (date of access: 15.04.2023).

5. Convolutional Neural Network. An Introduction to Convolutional Neural Networks. URL: https://medium.com/towards-data-science/convolutional-neural-network-17fb77e76c05 (date of access: 15.04.2023).

6. How Activation Functions Work in Deep Learning. URL: https://www.kdnuggets.com/2022/06/activation-functions-work-deep-learning.html (date of access: 16.04.2023).

7. Базова классифікація: Класифікувати зображення одягу. URL: https://www.tensorflow.org/tutorials/keras/classification?hl=ru (date of access: 18.04.2023).

8. Machine Learning Overtraining. URL: https://vortarus.com/machine-learning-overtraining/ (date of access: 18.04.2023).

9. Штучний інтелект: сьогодення та майбутнє. URL: https://ula.lantec.ua/statti/shtuchnij-intelekt-sogodennya-ta-majbutne (date of

access: 20.04.2023).

10. Ghosh, Anirudha & Sufian, A. & Sultana, Farhana & Chakrabarti, Amlan & De, Debashis. (2020). Fundamental Concepts of Convolutional Neural Network. 10.1007/978-3-030-32644-9_36.

11. Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.

12. Machine Learning Models Efficiency Analysis for Image Classification Problem Smelyakov, K., Honchar, Y., Bohomolov, O., Chupryna, A. CEUR Workshop Proceedingsthis link is disabled, 2022, 3171, pp. 942–959.

13. Lung X-Ray Images Preprocessing Algorithms for COVID-19 Diagnosing Intelligent Systems. Smelyakov, K., Chupryna, A., Bohomolov, O., Vakulik, E. CEUR Workshop Proceedingsthis link is disabled, 2022, 3171, pp. 1233–1250.

14. Identification of Modern Facial Emotion Recognition Models. Smelyakov, K., Bohomolov, O., Kizitskyi, M., Chupryna, A. CEUR Workshop Proceedingsthis link is disabled, 2022, 3171, pp. 1267–1281.

15. Експерименти та оцінка ефективності. URL: https://wandb.ai/katarzyna-zbroinska/fashion-mnist-kreas/reports/Fashion-MNIST-clothing-classification--VmlldzozMTQ3NDc2 (date of access: 01.05.2023).

16. Тимошин, Юрій Афанасійович, та Сергій Петрович Орленко. «Алгоритм розпізнавання обличчя людей на базі згорткової нейронної мережі». Адаптивні системи автоматичного управління 1, № 32 (21 вересня 2018): 166–73.

17. Повноз'єднані шари. URL: https://www.wiki-data.uk-ua.nina.az/Агрегування_(нейронні_мережі).html#Згорткові_шари (date of access: 04.05.2023).

18. Шари в CNN. URL: https://techukraine.net/згорткові-нейронні-мережі-cnn-вступ/ (05.05.2023).

19. Convolutional Neural Network Layers. URL: https://intellipaat.com/blog/tutorial/artificial-intelligence-tutorial/convolution-neural-network/?US#no3 (date of access: 05.05.2023).

20. ReLU Activation Layer. URL: https://www.run.ai/guides/deep-learning-for-computer-vision/deep-convolutional-neural-networks#ReLU-Activation-Layer (date of access: 07.05.2023)

# ДОДАТОК А

Відомість кваліфікаційної роботи

| Позначення | Найменування | Дод. відомості |
|---|---|---|
| | Текстові документи | |
| **1.** | Пояснювальна записка | 55 с. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | Інші документи | |
| | | |
| **2.** | Презентаційні матеріали | __ плакатів |
| | | |
| | | |
| | | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| . | . | Прізвище та ініціали. | Підп. | Дата | |
| Розробив | | Бордюг Д.Є. | | | Дослідження ефективності моделей розпізнавання товарів на інтернет ресурсах |
| Перевірив | | Смеляков К.С. | | | |
| Н.контр. | | Малєєва І.А. | | | |
| Затв. | | Філатов В.О. | | | |

| Шифр групи | Код напр./спец. |
|---|---|
| СШІм-21-2 | 122 |
| ХНУРЕ кафедра ШІ | |