МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КУЧЕР МИКИТА ОЛЕКСАНДРОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій
д.т.н., доцент
_____ Т.В. Нескородєва
«_»_____2022 р.

РОЗРОБКА І ДОСЛІДЖЕННЯ ЗАСОБІВ АНАЛІЗУ ВІЗУАЛЬНИХ ДАНИХ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Р.М. Бабаков, доцент кафедри
інформаційних технологій,
д.т.н., доцент
_____
(підпис)

Оцінка:_____/_____/
_____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця - 2022

АНОТАЦІЯ

Кучер М.О. Розробка і дослідження засобів аналізу візуальних даних з використанням нейронних мереж. Спеціальність 122 «Комп’ютерні науки», Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній роботі досліджено методи аналізу візуальних даних різних типів зображень та розмітку знайдених об’єктів, враховуючи наявність перешкод, різних погодних явищ, різної швидкості та положень об’єктів.

Метою роботи є створення програми для знаходження кораблів на фотографіях. Таким чином можна швидко проаналізувати великий масив фотографій, в вигляді якого представлені супутникові знімки і знайти на ньому кораблі, що є корисним як з комерційної сторони, так і з військової, що у свою чергу економить витратити на аналіз цих даних вручну і покращує точність в порівнянні з звичайними алгоритмами.

У кваліфікаційній роботі розглянуто технічні особливості роботи нейронних мереж з аналізу зображень, також описані основні етапи розробки програмного продукту з прикладами коду. Були сформовані необхідні алгоритми для роботи програми.

Значну увагу було приділено роботі програми під час аналізу фотографій з поганими погодними умовами.

Основна задача даної роботи полягає в розробці нейронної мережі для програми з аналізу візуальних даних, яка повинна аналізувати надані фотографії і здійснювати розмітку кораблів на них.

В результаті був створений готовий програмний продукт для аналізу візуальних даних з використанням нейронних мереж.

Отримані після аналізу розмічені дані дозволяють аналізувати розташування, напрямок та інші параметри кораблів, це може бути використано бізнесом для аналізу завантаження транспортних артерій, знаходження проблемних місць, точного передбачення термінів поставок,

оперативного реагування на надзвичайні стани. Військовими дані можуть бути використані для розвідки та актуалізації попередніх даних з залученням меншої кількості співробітників, ніж при ручному аналізі.

Ключові слова: Python, Аналіз візуальних даних, Neural network, Deep learning, CNN model, Розмітка зображень, Ship Detection.

ANNOTATION

Kucher M.O. Development and research of visual data analysis tools using neural networks. Specialty 122 "Computer Science", Educational Program "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2022.

The qualification work examines the methods of analysis of visual data of different types of images and markings of found objects, taking into account the presence of obstacles, different weather phenomena, different speeds and positions of objects.

The aim of the work is to create a program for finding ships in photographs. In this way, you can quickly analyze a large array of photos, which shows satellite images and find ships on it, which is useful both commercially and militarily, which in turn saves time to analyze this data manually and improves accuracy. with conventional algorithms.

The qualification work considers the technical features of neural networks for image analysis, also describes the main stages of software development with examples of code. Necessary algorithms for program operation were formed.

Considerable attention was paid to the work of the program during the analysis of photos with bad weather conditions.

The main task of this work is to develop a neural network for a program for the analysis of visual data, which should analyze the provided photographs and mark the ships on them. As a result, a ready-made software product for the analysis of visual data using neural networks was created.

The data obtained after the analysis allow to analyze the location, direction and other parameters of ships, it can be used by business to analyze the load of transport arteries, finding problem areas, accurate forecasting of delivery times, rapid response to emergencies. Military data can be used to reconnoiter and update preliminary data with fewer personnel than in manual analysis.

Keywords: Python, Visual Data Analysis, Neural network, Deep learning, CNN model, Image markup, Ship Detection.

ЗМІСТ

ВСТУП **6**

РОЗДІЛ 1 ОГЛЯД ТА АНАЛІЗ ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ **8**

1.1 Типи навчання нейронних мереж **12**

1.2 Методи оцінки точності передбачень **15**

1.3 Перенавчання та недонавчання нейромереж **17**

1.4 Підготовка даних **19**

1.5 Топологія нейронних мереж **23**

1.6 Постановка основних задач дослідження **30**

Висновок до розділу 1 ..... **31**

РОЗДІЛ 2 ОГЛЯД ТЕХНІЧНОЇ БАЗИ ДЛЯ РЕАЛІЗАЦІЇ НЕЙРОННОЇ МЕРЕЖІ З АНАЛІЗУ ВІЗУАЛЬНИХ ДАНИХ **32**

2.1 Огляд мови програмування Python і специфіка її використання для нейронних мереж **32**

2.2 Огляд середовища розробки **38**

2.3 Огляд даних для навчання та бібліотек взаємодії з ними **41**

2.4 Огляд існуючих підходів до розпізнавання об'єктів **46**

Висновок до розділу 2 ..... **54**

РОЗДІЛ 3 ПРОЄКТНА ЧАСТИНА **55**

3.1 Робота з даними **55**

3.2 Розробка моделі **58**

3.3 Аналіз результатів **61**

3.4 Імплементація моделі **65**

3.5 Програмний додаток **67**

Висновок до розділу 3 ..... **69**

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ **70**

ДОДАТОК А **74**

ВСТУП

Актуальність теми.

Штучні нейронні мережі застосовуються дедалі більше в всіх сферах життя людини, проєкти на їх основі дають можливість автоматизувати чи полегшити ті задачі, які вважались недоступними для комп'ютера і були прерогативою виключною людини. З етапом виходу нейронних мереж з суто теоретичних розрахунків, до можливості їх використання в комерційних та інших цілях людство отримало інструмент, що дозволяє йому полегшити розпізнавання зображень, переклади текстів, прогнозування тих чи інших явищ та інше. Проте нейронні мережі відносно недавно стали дійсно масовими і досі є багато задач, які все ще потребують вирішення.

Мета дослідження – це підвищення ефективності застосування нейронних мереж для аналізу візуальних даних шляхом створення і застосування нової моделі нейронної мережі.

Об'єкт дослідження – це процес обробки візуальних даних з використанням нейронних мереж.

Предмет дослідження – методи, засоби та програмні модулі, що дозволяють аналізувати візуальні дані за допомогою нейронних мереж.

Методи дослідження – це створення моделі нейронної мережі, її навчання та обробка результатів навчання, розробка додатка який використовує створену модель для виконання аналізу візуальних даних.

Теоретичне і практичне значення роботи полягає в розпізнаванні кораблів в різних погодних та інших умовах. Застосовуючи модель можна аналізувати масив фотографій в вигляді якого представлені супутникові знімки, знаходячи на ньому кораблі, що є корисним як з комерційної сторони, так і з військової, що у свою чергу економить витратити на аналіз цих даних вручну і покращує точність в порівнянні з звичайними алгоритмами.

Структура роботи.

Магістерська робота складається зі вступу, трьох розділів, висновків, таблиць, рисунків, літературних джерел та додатка.

У першому розділі магістерської роботи проведено огляд та аналіз теоретичних відомостей про побудову нейромереж, актуальність та специфіку їх роботи. Описано типи навчання нейронних мереж й нюанси їх роботи, розібрані основні проблеми, підготовку даних та топології мережі.

Другий розділ дозволяє розглянути обрані технічні засоби з реалізації завдання, обрану мову її переваги, необхідні бібліотеки, середовище розробки, дані що будуть використовуватись для навчання мережі, оглянуто існуючи підходи до розпізнавання.

Третій розділ містить етапи розробки нейронної мережі, цими етапами є робота з даними, розробка основних функцій і моделі. Описано і проаналізовано результати навчання моделі, наведені приклади передбачень. Була приділена увага недолікам і шляхам їх усунення.

Дипломна робота включає в себе 74 сторінки, 57 рисунків, 1 таблицю і список літератури з 43 джерел.

РОЗДІЛ 1
ОГЛЯД ТА АНАЛІЗ ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ

До того як приступити до розгляду засобів аналізу візуальних даних необхідно описати специфіку використання нейронних мереж. Оскільки найважливішу частину програми з аналізу візуальних даних представляє собою нейронна мережа, яка має розмічати дані.

Штучні нейронні мережі – це обчислювальні системи, які були натхнені біологічною різновидністю нейронних мереж, тобто мозком людей та більшості тварин. Основна суть в наявності нейронів, тобто певних перемикачів, які поодинці є досить примітивними, але при взаємопов’язуванні в системи нейронів – можуть виконувати досить складні операції, сигнал з кожного з нейронів може передаватись на тисячі інших. Навчання відбувається через повторну активацію деяких нейронних з’єднань з корегуванням їх ваги, такий вид навчання використовує зворотній зв'язок для корегування помилки.

Ключовою в нейроні є активаційна функція, яка має характеристику тумблера з значеннями 0 і 1 (або -1 і 1) в залежності від поданого на вхід значення, це і є імітацією включення біологічної версії нейрону. Прикладом активаційної функції можна вважати сигмоїдну функцію зображену на рисунку 1.1.

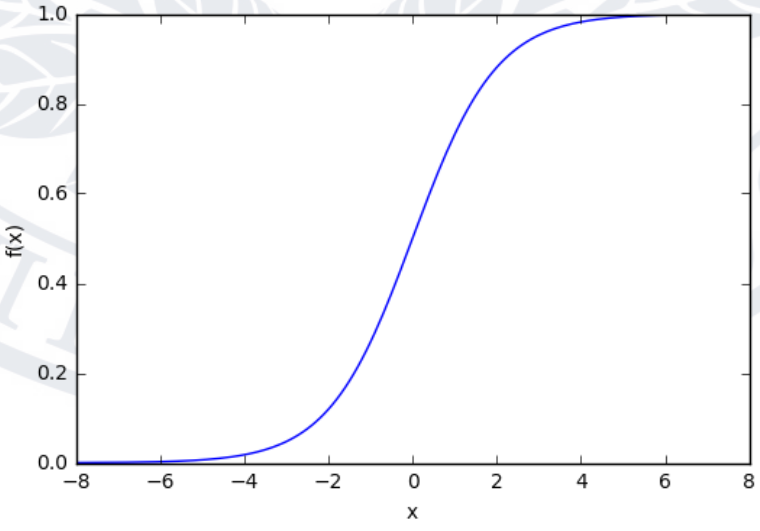


Рисунок 1.1 – Сигмоїдна функція

Нейрони (персептрони) – приймають зважені входи, сумують і

пропускають через активаційну функцію, а після цього видають певний результат. «Зваженість» входів розуміють як певний коефіцієнт на який помножаються відповідні входи, ці ж ваги корегуються під час навчання. Зважений вхід у вузол має наступний вигляд: x1w1+x2w2+x3w3+b

Де x – входи, w – їх вага, а b – зміщення.

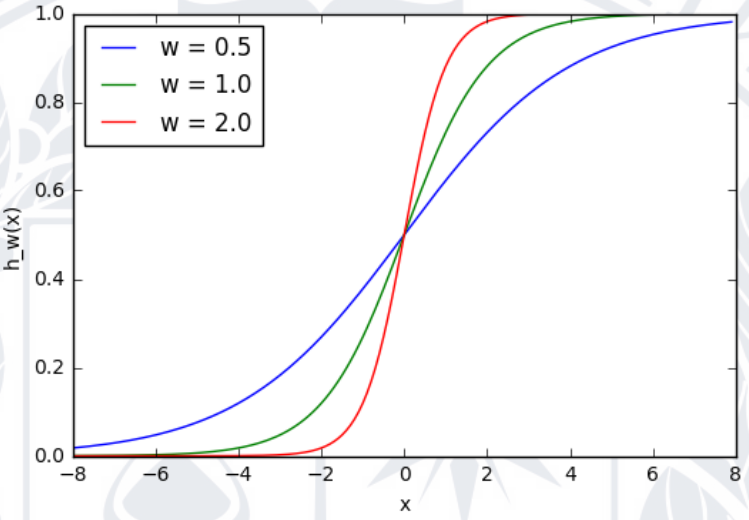


Рисунок 1.2 – Ефект від збільшення ваги

Великі нейронні мережі представляють собою безліч нейронів, що зв’язані між собою певним чином. Структура мережі і деякі її властивості випливають того, як розташовані зв’язки мін нейронами, як організовані входи і виходи, а також з кількості і величини прихованих шарів, які не є частиною ні входу, ні виходу. Приклад структури нейромережі можна побачити на рисунку 1.3

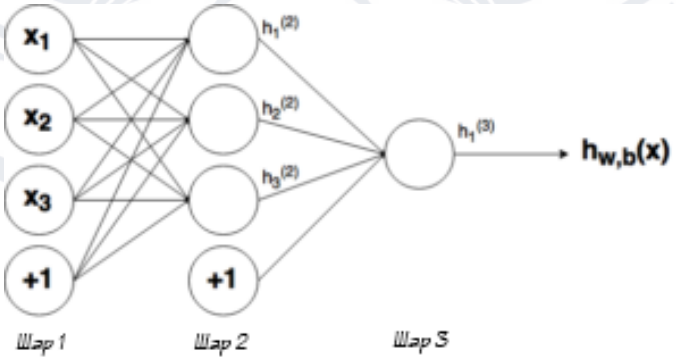


Рисунок 1.3 – Три шари нейронної мережі

Перший шар є шаром входів, в якому до мережі надходять дані зовні, другий шар – прихований, що не є частинами входу чи виходу, але безпосередньо впливає на рішення нейронної мережі, таких шарів може бути безліч, зв’язаний, в даному випадку, кожен нейрон першого шару з кожним з нейронів другого, але цей зв’язок також може бути іншим. Фінальним, вихідним шаром є третій, який вже і видає результат. Кожен зв'язок має свою вагу [1].

Розрахунок ваги з’єднань шарів в мережі є навчанням нейронної мережі. Слід розглянути градієнтний спуск в якості прикладу:

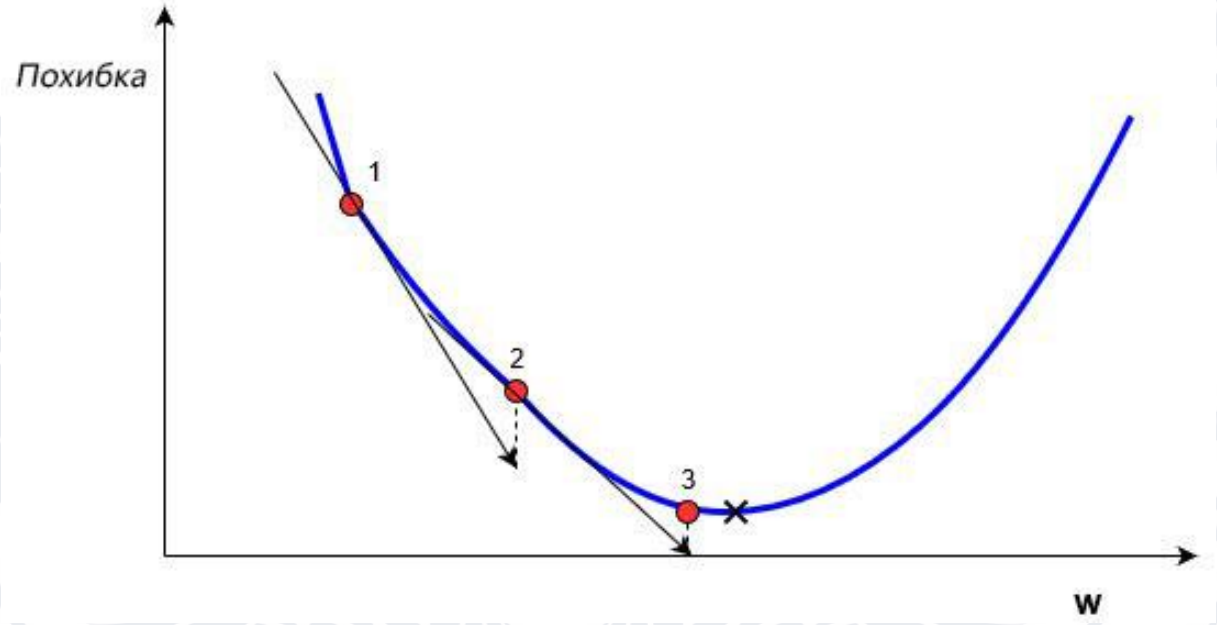


Рисунок 1.4 – Навчання нейронних мереж

На рисунку 1.4 відображено залежність помилки від значення деякої скалярної ваги *w*, мінімальне значення помилки зображене чорним хрестиком, але для дослідників заздалегідь невідоме значення ваги, яке дасть шуканий мінімум. Для знаходження беруть випадкове значення ваги, що зображене крапкою «1» на графіку і знаходять градієнт похибки відносно ваги. В даному випадку градієнтом є рівень нахилу кривої у визначеній точці, чорні стрілки на рисунку ілюструють його, крім того, це дає деяку додаткову інформацію –

тоді, коли він позитивний при збільшенні ваги, то в тому напрямку збільшується і похибка, а якщо негативний, то буде зменшуватись. Тобто з властивостей градієнта випливає, що для мінімізаціє похибки нам потрібно рухатись в сторону антиградієнта, а його величина вказує наскільки швидко функція змінюється, чим більше значення, тим швидше змінюється в залежності від ваги. Оскільки градієнтний спуск є ітеративним методом, то вага оновлюється кожен раз по формулі «$w_н=w_{ст}–α*∇error$», яка означає, що для пошуку нової ваги треба взяти стару, та відняти від неї градієнт похибки помножений на певний крок «α», який означає те, наскільки швидке буде наближення до глобального мінімума функції. Оскільки даний метод не дає ідеального наближення вихід з циклу реалізують через умову подібну до «якщо похибка менше певного числа», що, по суті, є точністю, або через певну кількість ітерацій, якщо необхідної точності не вийшло дістатись[2].

Запис надто великого «α» зазвичай приводить до того, що мінімум функції буде переступлено з неможливістю наблизитись до нього ближче, а в деяких випадках похибка буде тільки зростати з кожним кроком, маленькі ж значення кроку можуть призвести до того, що знаходження глобального мінімума буде неможливим, через зупинки в локальних мінімумах[3].

Щоб досягти прийнятного рівня точності, програми глибокого навчання вимагають доступу до величезних обсягів навчальних даних і обчислювальної потужності, жоден з яких не був легкодоступним для програмістів до ери великих даних і хмарних обчислень . Оскільки програмування глибокого навчання може створювати складні статистичні моделі безпосередньо з власного ітераційного результату, воно здатне створювати точні прогнозні моделі з великої кількості немаркованих неструктурованих даних. Це важливо, оскільки Інтернет речей продовжує набувати все більшого поширення, оскільки більшість даних, які створюють люди та машини, є неструктурованими та не позначеними.

Кількість необхідних даних також визначається типом навчання нейронної мережі.

1.1 Типи навчання нейронних мереж

Процес розрахунку ваг нейронної мережі – і є її навчанням.

Розрізняють 4 типи навчання нейромереж:

- навчання з вчителем;
- навчання без вчителя;
- навчання з частковим залученням вчителя;
- навчання з підкріпленням.

Рисунок 1.5 – Навчання нейронних мереж

Навчання з вчителем – застосовується при наявності повного набору розмічених даних для тренування моделі на всіх етапах її створення.

При цьому типі навчання надається набір навчальних прикладів на вхід, які зазвичай називають навчальними наборами даних. Завдання полягає в тому, щоб перекласти відомі відповіді на новий досвід, який зазвичай представлений у вигляді тестового набору даних. Наявність повністю позначеного набору даних означає, що кожен приклад навчального набору подається разом з розв’язанням, яке потрібно отримати алгоритму. Тому

позначений набір даних, наприклад фотографій квітів, навчає нейронну мережу, яка малює ті чи інші квіти. Головний нюанс в тому, що дані, доступні для навчання, дещо схожі на дані, для яких необхідно застосувати навчену модель. В іншому випадку – узагальнення буде неможливе. Оскільки тоді, коли нейромережа отримує нову фотографію, то вона просто порівнює її з прикладом навчального набору даних і на основі цього прогнозує свою відповідь[4].

Більшою мірою навчання з вчителем застосовують тоді, коли потрібно вирішити наступні три типи задач:

- Класифікація – визначення одного з класів, при цьому кількість класів має бути кінечною, наприклад розділити фотографії різних тварин в відповідні групи: «риби», «собаки», «кішки» і т.д.;
- Регресія – напряму зв’язано з безперервними даними, прикладами може слугувати лінійна регресія, яка обчислює очікуване значення змінної *y* враховуючи конкретні значення *x*: передбачення росту людини по її вазі, передбачити ціну акцій і т.д.;
- Ранжування (learning to rank) – за наявними даними відсортувати за певною ознакою, наприклад релевантність, поданий список об’єктів.

Навчання без вчителя – відбувається з набором даних який подається без прикладеної до них бажаної відповіді, тому нейронна мережа намагається знайти кореляції в даних самостійно. Корисні ознаки модель обирає самостійно і відповідно аналізує їх, систематизація даних залежить від завдання. Найбільш поширеним випадком використання навчання без вчителя – навчання моделей кластеризації даних, тобто розбиття їх на класи, які заздалегідь невідомі для нейромережі, певною мірою так, щоб схожі між собою дані були ближче одне до одного, а точки з різних кластерів були якомога далі. Тобто алгоритм підбирає дані в кластери групуючи їх через

схожі ознаки, які нейромережа взяла як підходящі. Прикладом кластеризації можна назвати виділення сімейства генів з послідовності нуклеотидів, або кластеризація користувачів певного ресурсу чи сайту, чи медичних захворювань/карток пацієнтів[5].

Крім того, ще одним завданням вважають зниження розмірності, тобто коли вхідні дані мають більшу розмірність ніж вихідні, а задача в побудові уявлення даних меншої розмірності таким чином, щоб якомога краще представити через них дані великої розмірності. За допомогою інструментів, що використовують зменшення розмірності можна видаляти шум на фотографіях, сканах та інше.

Окремо варто зазначити, що навчання без вчителя також використовується для таких задач: оцінка щільності, виявлення аномалій.

Нюанс використання такого типу навчання полягає в складності обчислення точності алгоритму, так як якісь мітки чи вірні відповіді не існують. Але при цьому, великою перевагою є те, що нерозмічені дані знайти в рази легше, крім того, розмічені дані іноді мають помилки, а їх розмітка «вручну» іноді потребує значних фінансових витрат.

Навчання з частковим залученням представляє собою дещо середнє між навчанням без вчителя і з ним, суть його заключається в поданні тренувального датасету в якому є як розмічені дані, так і не розмічені. Зазвичай його використовують тоді, коли розмічені дані отримати складно. Наприклад для синтаксичного розбору тексту дуже легко знайти чи створити нерозмічені дані, а малювання кожного дерева – складно. Така ж ситуація з розпізнаванням мови, бо знаходження великого обсягу нерозмічених записів – легко, набагато ж складніше їх потім розмічати, адже для цього потрібно відділяти межі кожної фонеми. В таких випадках і використовують навчання з частковим залученням вчителя [6].

Навчання з підкріпленням – вид навчання, при якому нейронна мережа вчиться діяти в певному середовищі отримуючи нагороди чи штрафи за свої дії і відповідно до цього напрацьовуючи розуміння бажаної поведінки.

Цей підхід використовує довгострокову стратегію - так само як в шахах: наступний найкращий хід може не допомогти виграти в кінцевому рахунку. Тому агент намагається максимізувати сумарну нагороду.

Стратегія мережі покращується відповідно до рівнів зворотного зв'язку. Частіше всього такий підхід до навчання використовується для навчання роботів, безпілотних автомобілей, в ігровій індустрії, в рекомендаційних системах [7].

1.2 Методи оцінки точності передбачень

Помилка прогнозу – величина відхилення прогнозу від дійсного стану об'єкта. Якщо говорити про прогноз продажів, то це показник відхилення фактичних продажів від прогнозу.

Точність прогнозування є поняття прямо протилежне помилці прогнозування. Якщо помилка прогнозування велика, то точність мала і навпаки, якщо помилка прогнозування мала, то точність велика. По суті справи оцінка помилки прогнозу MAPE є зворотна величина для точності прогнозування

Обрати відповідну модель прогнозу можна за допомогою розрахунку показника точності прогнозу. Модель прогнозу, у якій показник точності прогнозу буде ближче до 100%, з більшою ймовірністю зробить більш точний прогноз.

Коефіцієнт детермінації (R2) - це частка дисперсії залежної змінної, що пояснюється розглянутою моделлю залежності, тобто пояснюючими змінними. Більш точно - це одиниця мінус частка непоясненої дисперсії (дисперсії випадкової помилки моделі, або умовної за факторами дисперсії залежної змінної) в дисперсії залежної змінної. Його розглядають як універсальну міру залежності однієї випадкової величини від безлічі інших. В окремому випадку лінійної залежності R2 є квадратом так званого множинного коефіцієнта кореляції між залежною змінною і пояснюючими змінними. Зокрема, для моделі парної лінійної регресії коефіцієнт детермінації

дорівнює квадрату звичайного коефіцієнта кореляції між y і x. Коефіцієнт детермінації R^2 — статистичний показник, що використовується в статистичних моделях як міра залежності варіації залежної змінної від варіації незалежних змінних[8].

Основна проблема застосування (вибіркового) R2 полягає в тому, що його значення збільшується (не зменшується) від додавання в модель нових змінних, навіть якщо ці змінні ніякого відношення до прогнозованої змінної не мають. Тому порівняння моделей з різною кількістю чинників за допомогою коефіцієнта детермінації - некоректно. Для цих цілей можна використовувати альтернативні показники.

Середня абсолютна помилка (mean absolute error - MAE) - це міра різниці між двома безперервними змінними. Припустимо, X і Y - змінні парних спостережень, які виражають одне і те ж явище. Приклади Y порівняно з X включають порівняння прогнозованого з спостережуваним, подальший час проти початкового часу та одну техніку вимірювання проти альтернативної методики вимірювання. Розглянемо графік розсіяння n точок, де точка i має координати (xi, yi). Середня абсолютна помилка (MAE) - середня вертикальна відстань між кожною точкою та ідентичною лінією. MAE - це також середня горизонтальна відстань між кожною точкою та ідентичною лінією[9].

Середня абсолютна процентна похибка (mean absolute percentage error - MAPE), також відома як середнє абсолютне відхилення у відсотках, є показником точності прогнозування методу прогнозування в статистиці, наприклад, в оцінці тренду, також використовується як функція втрат для проблем регресії в машинному навчанні[10].

Середньоквадратична помилка (mean square error - MSE) або середньоквадратичне відхилення (MSD) оцінювача вимірює середнє значення квадратів помилок - тобто середньоквадратична різниця між оціненими значеннями та фактичними значеннями. MSE - це функція ризику, що відповідає очікуваному значенню втрат у квадраті. Те, що MSE майже завжди є суто позитивним (а не нульовим), відбувається через випадковість або через

те, що оцінювач не враховує інформацію, яка могла б дати більш точну оцінку[11].

1.3 Перенавчання та недонавчання нейромереж

Будь-яка модель створюється для того, щоб вирішити деяку задачу. Передбачити значення, класифікувати об'єкт і т.п. Для навчання є набір даних, в якому наведено точні та розмічені дані. В результаті роботи навчання ціль отримати модель, яка буде правильно працювати на будь-яких даних, з певною точністю. Але буває, що модель прекрасно працює на навчальній вибірці, а на тестовій жахливо. Тут з'являються проблеми перенавчання або недонавчання. Недонавчання - небажане явище, яке виникає при вирішенні завдань навчання по прецедентах, коли алгоритм навчання не забезпечує досить малої величини середньої помилки на навчальній вибірці. Недонавчання виникає при використанні недостатньо складних моделей, коли навіть на навчальних даних ми не можемо досягти малої помилки моделі. Причин тому може бути багато:

- занадто рання зупинка;
- неправильно підібраний алгоритм навчання;
- неадекватна функція помилки;
- неправильно налаштовані параметри.

Перенавчання (overtraining, overfitting) - небажане явище, яке виникає при вирішенні завдань навчання по прецедентах, коли ймовірність помилки навченого алгоритму на об'єктах тестової вибірки виявляється істотно вище, ніж середня помилка на навчальній вибірці. Перенавчання виникає при використанні надмірно складних моделей[12].

На рис. 1.5 наведено приклад сигналу з шумом, який близький до лінійного, що апроксимується лінійною функцією і поліномом. Хоча поліном гарантує ідеальний збіг, лінійна апроксимація краще узагальнює закономірність і буде давати кращі передбачення.

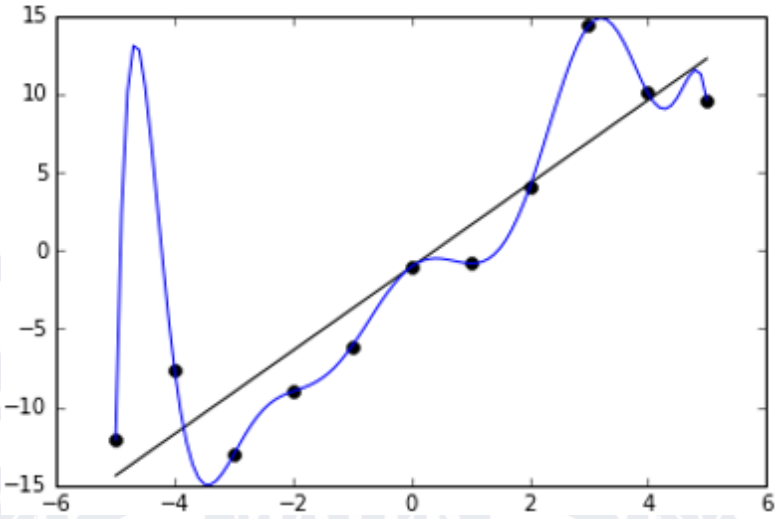


Рисунок 1.6 – Навчання нейронних мереж

Причини для перенавчання приблизно такі ж, за винятком зупинки, тут вона навпаки дуже пізня.

Потенціал перенавчання залежить не лише від кількостей параметрів та даних, але й від відповідності структури моделі формі даних, та величини похибки моделі в порівнянні з очікуваним рівнем шуму або похибки в даних.

Для того, щоб модель була ефективною, потрібно знайти той момент, коли модель вже навчена, але ще не перенавчилась (рис. 1.7). В основному використовується одночасна перевірка помилки на навчальній і тестовій вибірці, обидві вони з часом повинні зменшуватися, але якщо тестова помилка раптово починає збільшуватися, то треба звернути на це увагу і зупинити процес навчання. Тому що, в більшості випадків це означає початок перенавчання.

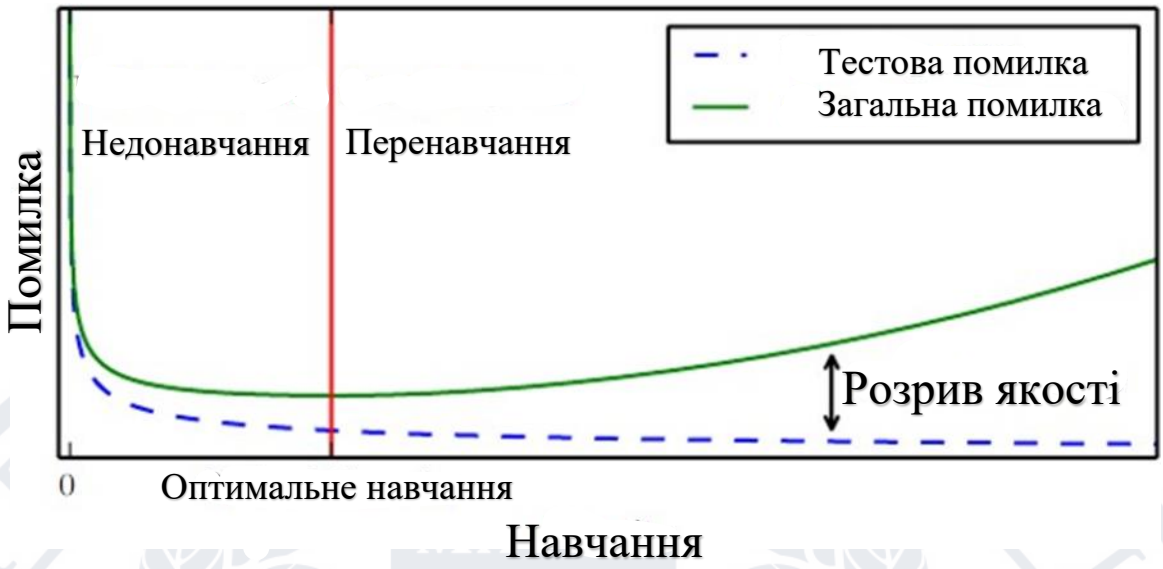


Рисунок 1.7 – Графік залежності помилки від епохи для моделей машинного навчання

Емпіричним ризиком називається середня помилка алгоритму на навчальній вибірці. Метод мінімізації емпіричного ризику (empirical risk minimization, ERM) найбільш часто застосовується для побудови алгоритмів навчання. Він полягає в тому, щоб в рамках заданої моделі вибрати алгоритм, який має мінімальне значення середньої помилки на заданій навчальній вибірці[13].

1.4 Підготовка даних

Етап підготовки даних для машинного навчання має велике значення через те, що непідготовлені дані можуть значно погіршити результати навчання через ті, чи інші проблеми необроблених даних.

Підготовка даних – це крок після збору даних у життєвому циклі машинного навчання, а також процес очищення та перетворення необроблених даних, які ви зібрали [14].

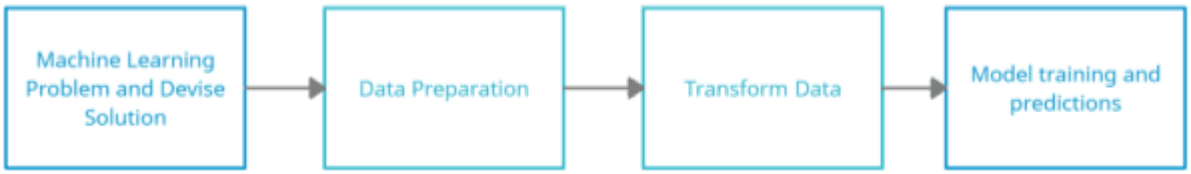


Рисунок 1.8 – Етапи створення моделі машинного навчання

Є три основні частини підготовки даних:

1. Дослідницький аналіз даних (EDA)
2. Попередня підготовка даних
3. Розбиття даних

Дослідницький аналіз даних – етап аналізу отриманих даних задля збільшення їх зрозумілості, важливість полягає в отриманні додаткової інформації про дані що може бути використана для їх зміни чи обробки, а також для розуміння специфіки навчання на них.

До цієї частини входить визначення функціональних і цільових змінних, аналіз типів даних та перевірка наявності викидів, останні можна визначити через:

- Z-показник (стандартні відхилення)
- Міжквартильний діапазон (IQR)

Попередня обробка даних – найдовший етап підготовки даних, являє собою маніпулювання даними для забезпечення можливості правильного моделювання датасету, який можна буде давати моделі. Оскільки деякі дані відсутні чи помилково вказані, існує потреба їх заміни на правдоподібні значення.

Імпутація ознаки – процес заповнення відсутніх значень:

- **Імпутація одного значення** – заміна відсутніх значень середнім, медіаною або модою стовбця.
- **Імпутація кількох значень** – функції моделювання, які мають відсутні дані, і заміна відсутніх даних тим, що знаходить ваша модель.
- **К-найближчий сусід** – заміна відсутніх даних значенням з іншої подібної вибірки.

- **Видалення рядка** – видалення рядка, достовірно замінити відсутні значення в якому неможливо.
- Інші, такі як «випадкова заміна», «рухоме вікно», тощо[15].

Функціональне кодування – процес перетворення нечислових значень ознак в числові, необхідне через те, що моделі машинного навчання вимагають, щоб всі значення були числами. Методи реалізації:

- **Кодування міток (Label Encoder) –** перетворює кожну унікальну категоріальну ознаку на відповідне число. Перевагами є простота і лаконічність, недоліком є викривлення даних внаслідок виникнення залежності між ними. Приклад вдалого використання – заміна буквених оцінок студентів на числа. Приклад невдалого використання – заміна марок автомобілів[16].
- **One-Hot Encoder –** кодування категоріальних ознак в бінарний формат. Перевагою є точність і незалежність ознак. Недоліком є збільшення числа стовбців в датасеті на n-1 де n – кількість унікальних категоріальних ознак[17].
- **Двійковий кодувальник –** частково виправляє недолік One-Hot Encoder'а можна виправити, якщо закодувати ознаки аналогічно до перетворення десяткових чисел в двійкові представляючи порядковий номер унікального значення як рядка двійкового представлення, тоді замість n-1 ми можемо отримати log(n) нових стовбців, однак в такому разі погіршується інтерпретованість ознак.
- **Контрастне кодування –** N ознак кодуються матрицею (N, N-1), де на головній діагоналі матриці і вище за неї знаходяться одиниці зі знаком мінус, а відразу під головною діагоналлю йде порядковий номер значення і нижче за нього нулі. При кодуванні Helmert Encoder іде сортування при кодуванні унікальних значень[18].

Нормалізація даних – коли числові значення знаходяться в різних шкалах, напр. зріст у сантиметрах і вага у фунтах, більшість алгоритмів машинного навчання не працюють добре. Алгоритм k-найближчих сусідів є

яскравим прикладом, коли функції з різними масштабами не працюють добре. Таким чином нормалізація або стандартизація даних може допомогти вирішити цю проблему.

- **Нормалізація функції** – перемасштабовує значення таким чином, щоб вони були в діапазоні [0,1].
- **Стандартизація ознак** – змінює масштаб даних, щоб отримати середнє значення 0 і стандартне відхилення одиниці.

Розробка функцій – це процес перетворення необроблених даних у функції, які краще представляють основну проблему, яку намагаються вирішити. Немає конкретного способу виконати цей крок, але ось які приклади можна розглянути:

- Перетворення змінної DateTime для отримання лише дня тижня, місяця року, тощо.
- Створення контейнерів або відер для змінної. (наприклад, для змінної висоти може мати 100–149 см, 150–199 см, 200–249 см тощо)
- Поєднання кількох функцій і/або значень для створення нового. Наприклад, одна з найточніших моделей для задачі титаніка створила нову змінну під назвою «Is_women_or_child», яка мала значення True, якщо особа була жінкою чи дитиною і false в іншому випадку.

Відбір функцій – визначення найбільш релевантних чи цінних функції набору даних.

- **Важливість функції** – алгоритми, такі як випадкові ліси або XGBoost, дозволяють визначити, які функції були найбільш «важливими» для прогнозування значення цільової змінної. Створивши одну з цих моделей і провівши важливість ознак, можна отримати розуміння того, які змінні корисніші за інші.
- **Зменшення розмірності** – один із найпоширеніших методів зменшення розмірності, аналіз головних компонентів (PCA), бере значну кількість функцій і використовує лінійну алгебру, щоб зменшити їх до меншої кількості[19].

Робота з дисбалансом даних – це невідповідність кількості прикладів одного класу до інших, можна усунути, наприклад так:

- **Збір додаткових даних** — це завжди працює накраще, але часто не має можливості зібрати більше даних.
- **За допомогою пакета scikit-learn-contrib Python** можна збільшувати або зменшувати вибірку даних[20].

Поділ даних – це етап розділення всіх даних на три набори:

- Навчальний набір: дані, на яких вчиться модель
- Набір перевірки: дані, на яких налаштовуються гіперпараметри моделі
- Тестовий набір: дані, на яких оцінюється кінцева якість моделі.

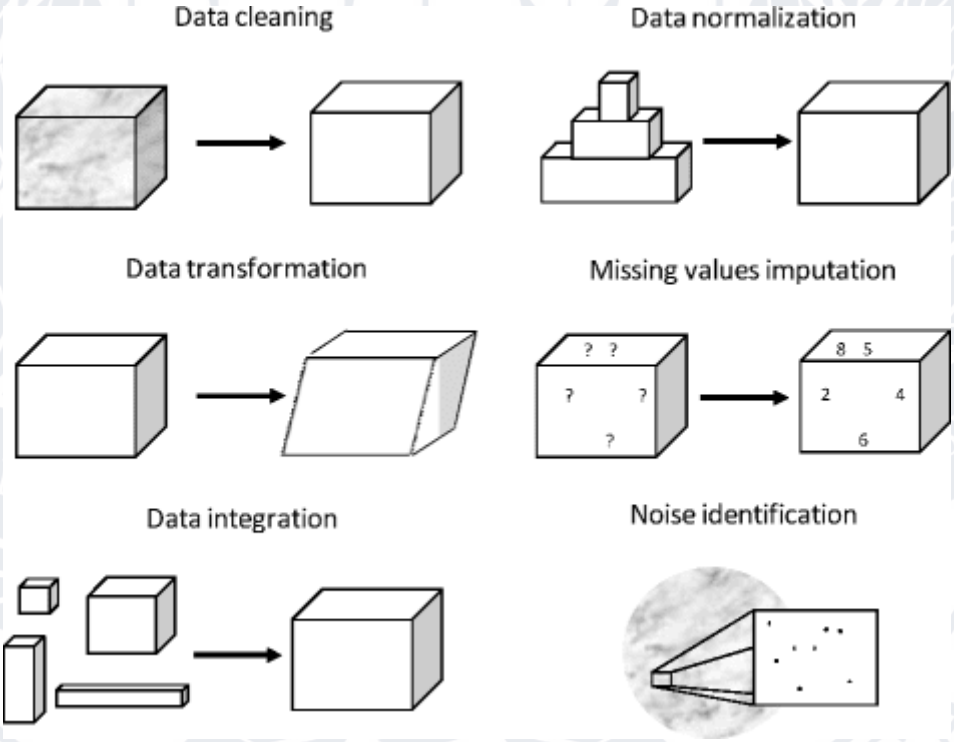


Рисунок 1.9 – Ілюстрування деяких процесів підготовки даних

1.5 Топологія нейронних мереж

Топологія характеризує організацію вузлів нейромереж. Основними типами є повнозв’язні мережі, багатошарові та нейромережі з локальними зв’язками. Розглянемо деякі можливі топології нейромереж:

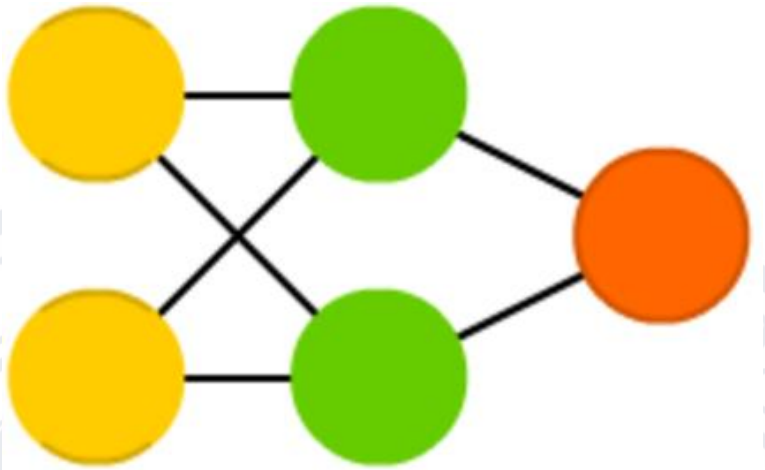


Рисунок 1.10 – Мережа прямого зв’язку

Нейронна мережа прямого зв’язку (FF) бере свій початок із 50-х років, дотримується таких правил:

- всі вузли повністю з'єднані
- активація проходить від вхідного рівня до вихідного, без зворотних циклів
- існує один шар між входом і виходом (прихований шар)

В більшості випадків цей тип мереж навчається методом зворотного поширення. Її підвидом можна вважати RBF, яка використовує радіальну базисну функцію, як функцію активації замість логістичної функції, різниця в тому, що логістична функція відображає деяке довільне значення в діапазоні від 0 до 1 відповідаючи на питання «так чи ні», що добре для систем класифікації та прийняття рішень, але погано для безперервних значень. Радіальні базисні функції відповідають на питання «наскільки далеко це від цілі», що чудово підходить для апроксимації функцій і керування автомобілем, наприклад[21].

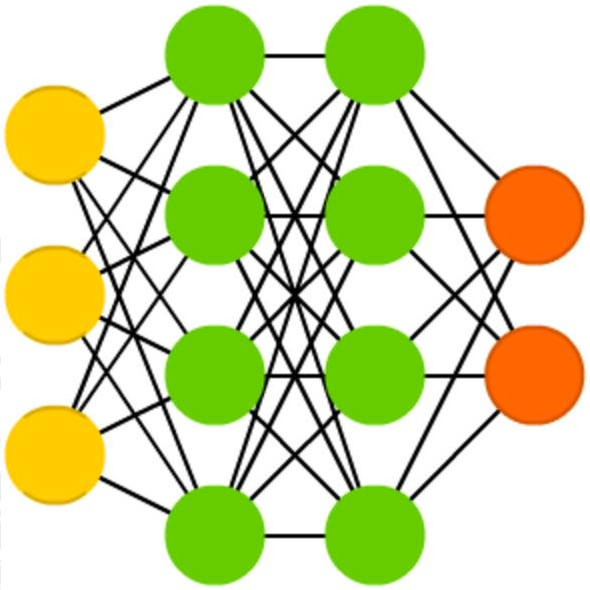


Рисунок 1.11 – Нейронна мережа DFF

Нейронна мережа DFF бере свій початок із 90-х років, є FF мережею з більш ніж одним прихованим шаром, довгий час їх навчання було майже неможливим через надто стрімке зростання часу навчання від кожного додаткового шару і ближче до 00-х було розроблено підходи до навчання, які дозволили більш ефективно тренувати цей тип мереж, що вкупі з більш потужною апаратною частиною переросло в бум нейронних мереж. Оскільки DFF є, по суті, варіацією FF мережі – вона виконує ті самі цілі, однак робить це з набагато кращими результатами.

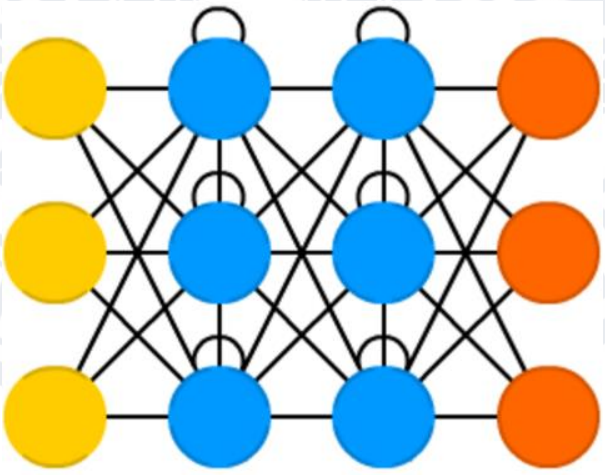


Рисунок 1.12 – Рекурентна нейронна мережа

Рекурентна нейронна мережа (RNN), має деяке нововведення для прихованих вузлів відносно FNN, рекурентність, що реалізована наступним чином: кожна прихована комірка отримує власний вихід з фіксованою затримкою (одна, або кілька ітерацій). Існує багато варіацій реалізації і використання цього виходу, такі як: передача стану вхідним вузлам, змінні затримки, тощо. Основна ж перевага відносно FNN в тому, що RNN значно краще себе показує в випадках, коли важливий контекст, тобто коли рішення з минулих ітерацій або зразків можна з користю використати для поточних. Приклад використання таких мереж – тексти, адже в них слова несуть додаткову інформацію виходячи з попередніх слів та речень.

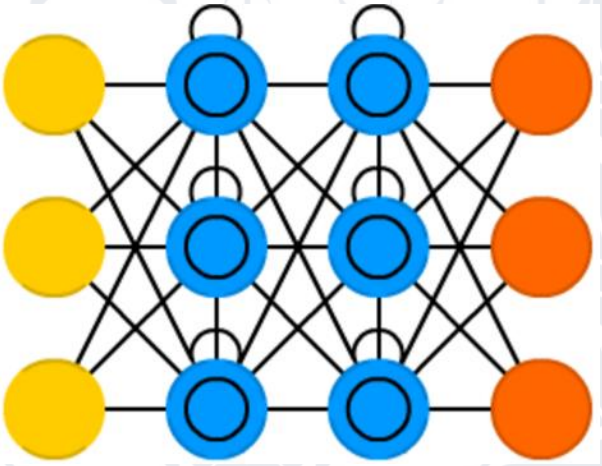


Рисунок 1.13 – Мережа LSTM

Нейронна мережа з комірками довгої короткочасної пам’яті (LSTM) має спеціальну комірку, яка може обробляти дані, що мають затримки. Таким чином можуть запам’ятовувати не просто деякий контекст попередніх обробок чи циклів, як в випадку з RNN, а цілі попередні відеокадри, наприклад. Однак мережі LSTM також часто використовуються для розпізнавання письма та мови. Сама ж комірка пам’яті має так звані «ворота», які повторюються і контролюють як інформація буде запам’ятовуватись чи забуватись. Структуру вузла видно на рисунку 1.14

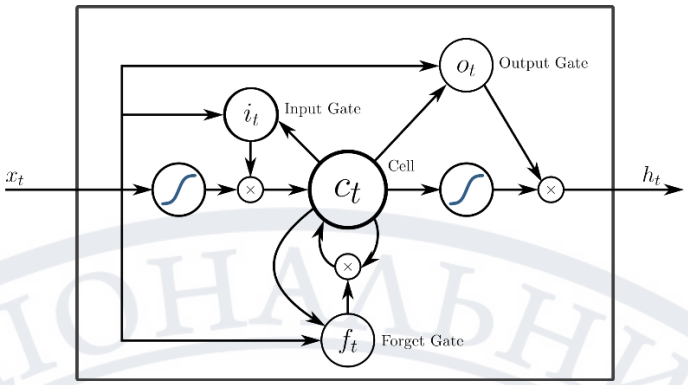


Рисунок 1.14 – Структура вузла мережі LSTM

Де x – ворота, що мають власні ваги, а іноді й функції активації, кожен раз вони вирішують чи передавати дані вперед, стирати пам'ять, тощо. Вхідні ворота/вентиль визначає скільки інформації з останнього зразка буде зберігатись в пам’яті, вихідні ворота регулюють обсяг даних, що передаються на наступний рівень, а ворота забування контролюють швидкість руйнування збереженої пам’яті.

Також існує мережа GRU, що є дуже схожою на LSTM, але всі ворота об’єднані в шлюз оновлення, а шлюз скидання тісно пов’язаний з входом, найчастіше використовуються в музиці та синтезі мови, менш вимогливі до ресурсів, але майже однакові по точності в порівнянні з LSTM.

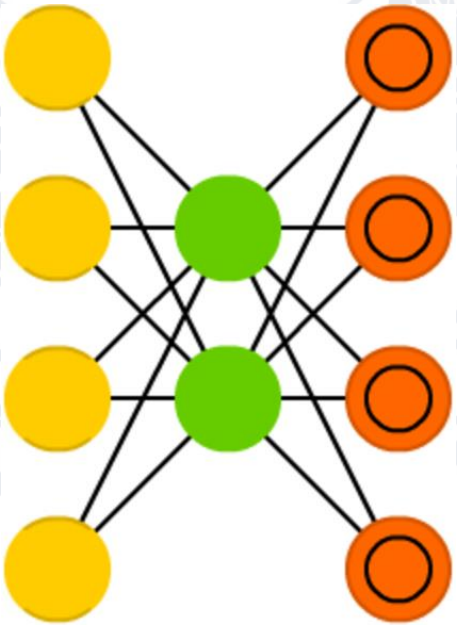


Рисунок 1.15 – Мережа AE

Автокодувальники AE використовуються для класифікації, кластеризації та стиснення ознак. На відміну від FF мереж, в яких відбувається навчання під наглядом AE можна тренувати і без нього. Їх головна особливість структури – кількість прихований комірок менша за кількість вхідних комірок, що змушує нейромережу узагальнювати вхідні дані і шукати певні шаблони.

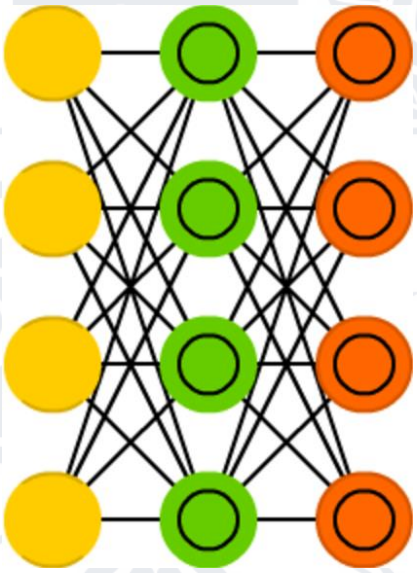


Рисунок 1.16 – Мережа VAE

Відмінність варіаційного автокодера (VAE) в тому, що він фокусується на знаходженні сили зв’язку між подіями/даними, а не узагальнює їх.

Ще одною AE-подібною мережею є DAE мережа, що призначена для вирішення проблеми адаптування до вхідних даних. В вхідний шар домішується трохи шуму – змінюються дані випадковим бітом чи перемикають випадковий біт, через це мережі необхідно реконструювати вихідні дані з трохи зашумлених вхідних, що призводить до необхідності відбору мережею більш загальних рис.

SAE мережа теж є AE-подібною, однак має протилежну ідею і, як наслідок, має кількість прихованих клітин більшу, ніж кількість шарів введення і виведення, застосовується для виявлення прихованих шаблонів групування даних.

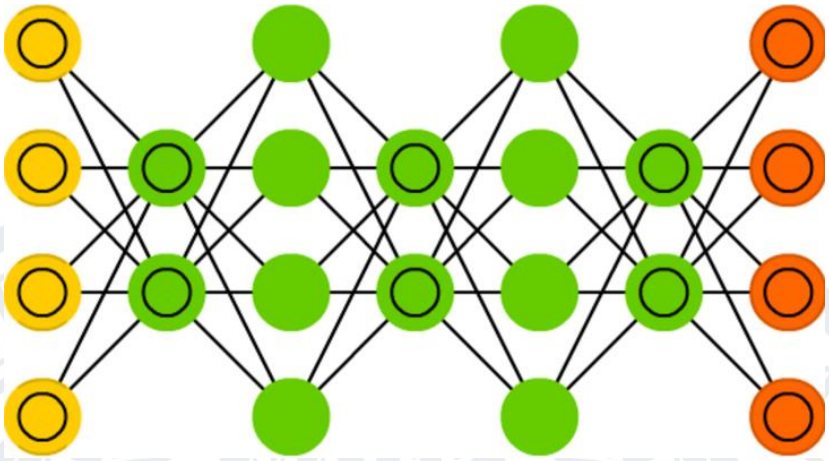


Рисунок 1.17 – Мережа DBN

DBN мережа є набором машин Больцмана, які оточені VAE, ці мережі можна з'єднувати разом для навчання однієї мережі іншою за вже вивченим шаблоном. Машина Болцмана, в свою чергу, самі по собі не вважаються повноцінними нейромережами внаслідок принципів її роботи, але дозволяють зменшувати шум, відновлювати вхідні дані та виявляти і вилучати ознаки. Це перша топологія мережі, яка була успішно створена за допомогою підходу імітованого відпалу (Simulated annealing). Багатокомпонентні машини Больцмана можуть створити мережу глибоких переконань що вже є нейромережою, в загальному розумінні.

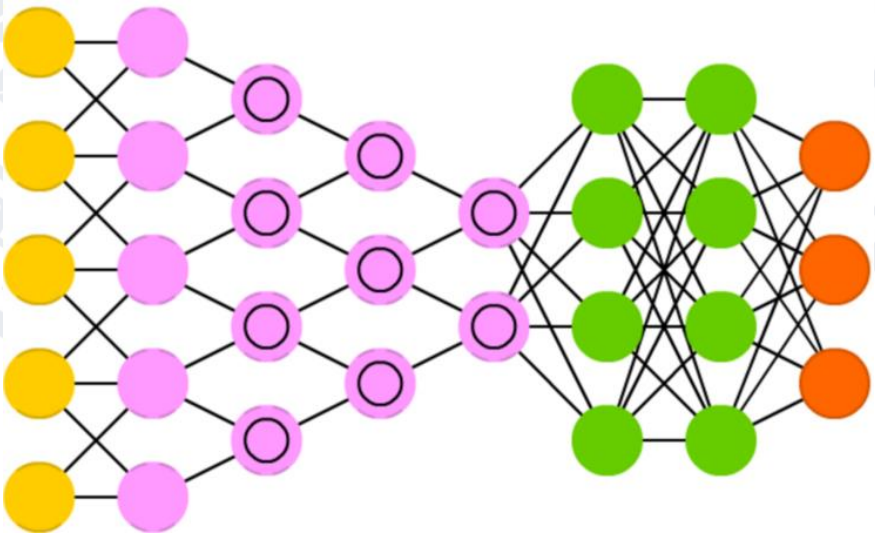


Рисунок 1.18 – Мережа DCN

Мережа DCN складається з комірок згортки і ядер. Комірки згортки обробляються дані, котрі надходять до них і спрощують/узагальнюють, ці дані потім передаються на обробку в ядра, які дають відповідь. Зазвичай використовуються для обробки зображень і працюють з невеликою підмножиною пікселів рухаючи вікно введення по зображенню піксель за пікселем. Часто DFF приєднується до кінцевого згорткового рівня для подальшої обробки даних.

Схожим чином виглядає і DN модель, що виглядає як перевернута DCN, вона по малому входу намагається дати великій вихід, тобто по назві відтворити малюнок, наприклад.

DCIGN є поєднанням DCN і DN, хоча являє собою аналог автокодера, ця мережа використовується для обробки фото і може обробляти навіть ті, які з якими раніше мережа не зустрічалась. Завдяки виділенню ознак і рівням абстракції ця мережа може видаляти певні об’єкти з зображення, перефарбовувати їх, змінювати стилі та інше.

1.6 Постановка основних задач дослідження

Виходячи з теоретичної частини наведеної в першому розділі отримана задача дослідження, яке досягнуто в наступних розділах. Проаналізувавши аналоги та технічну базу розробити програмне забезпечення, яке буде розпізнавати кораблі на супутникових чи інших знімках використовуючи нейронні мережі що навчаються з вчителем, додати можливість доступу до інструментів аналізу з мережі інтернет. Проаналізувати вплив структури нейронних мереж та якість підготовки даних на результат розпізнавання і його точність.

Висновок до розділу 1

У даному розділі були розглянуті теоретичні відомості про побудову нейромереж, актуальність та специфіку їх роботи. Описано типи навчання нейронних мереж й нюанси їх роботи, розібрані основні проблеми, в тому числі недонавчання та перенавчання мереж. Була приділена увага підготовці даних та топології мережі, виконана постановка основних задач дослідження.

РОЗДІЛ 2
ОГЛЯД ТЕХНІЧНОЇ БАЗИ ДЛЯ РЕАЛІЗАЦІЇ НЕЙРОННОЇ МЕРЕЖІ З АНАЛІЗУ ВІЗУАЛЬНИХ ДАНИХ

Для реалізації програмної частини необхідно було обрати мову програмування що буде гнучкою, мати необхідні бібліотеки і актуальну підтримку. Мова програмування Python відповідає необхідним умовам і має значно більшу кількість бібліотек для машинного навчання ніж конкуренти[22].

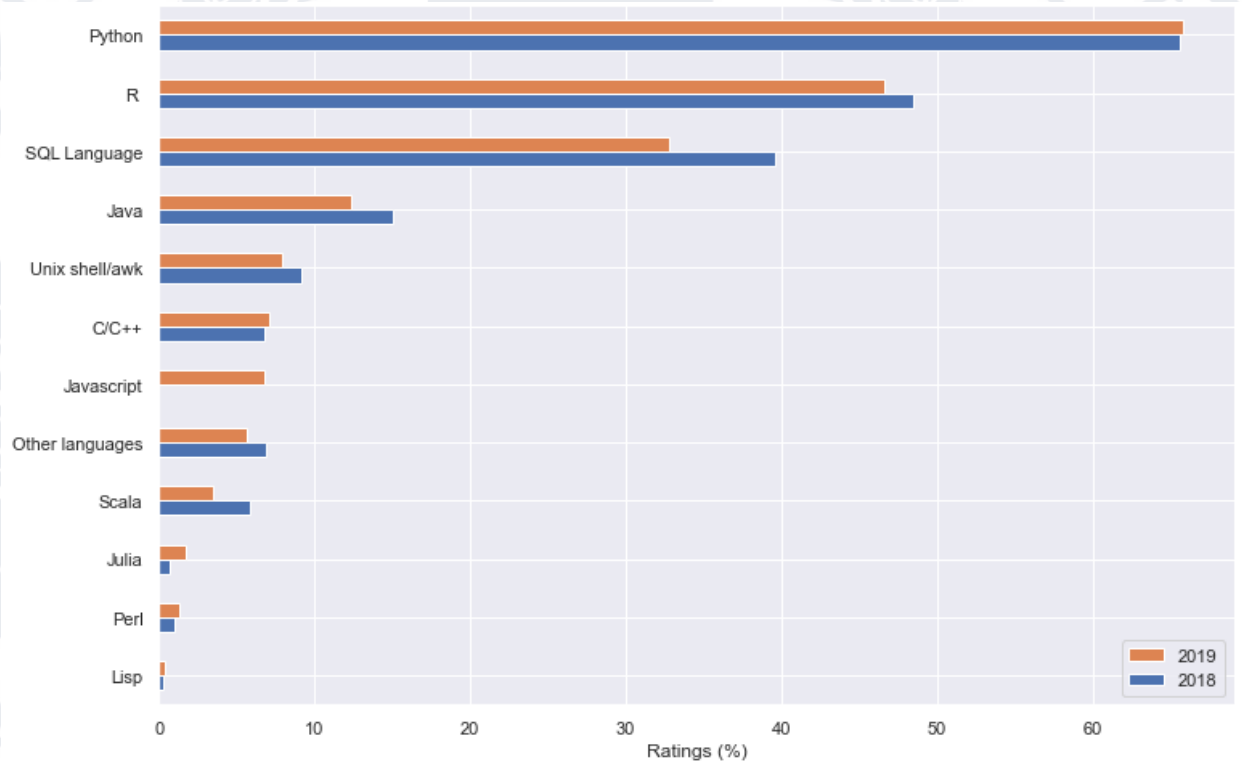


Рисунок 2.1 – Популярність мов програмування в сфері науки про дані

Крім того вибір даних для навчання нейронної мережі теж є важливим етапом. Необхідним був також і готових рішень.

2.1 Огляд мови програмування Python і специфіка її використання для нейронних мереж

Python — це інтерпретована об’єктно-орієнтована мова програмування

високого рівня з динамічною семантикою. Його високорівневі вбудовані структури даних у поєднанні з динамічною типізацією та динамічним зв'язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання як мови сценаріїв або з'єднувальної мови для з'єднання існуючих компонентів. Простий, легкий для вивчення синтаксис Python підкреслює читабельність і, отже, знижує вартість обслуговування програми. Python підтримує модулі та пакети, що заохочує модульність програми та повторне використання коду. Інтерпретатор Python і обширна стандартна бібліотека доступні у вихідному або двійковому вигляді безкоштовно для всіх основних платформ і можуть вільно поширюватися[23].

Оскільки етапу компіляції немає, цикл редагування-тестування-налагодження відбувається неймовірно швидко. Налагоджувати програми на Python легко: помилка чи неправильний вхід ніколи не спричинить помилку сегментації. Натомість, коли інтерпретатор виявляє помилку, він викликає виняток. Якщо програма не вловлює виняток, інтерпретатор друкує трасування стека. Налагоджувач рівня вихідного коду дозволяє перевіряти локальні та глобальні змінні, оцінювати довільні вирази, встановлювати контрольні точки, покроково виконувати код по рядках і так далі. Сам налагоджувач написаний на Python, що свідчить про інтроспективну силу Python. З іншого боку, часто найшвидшим способом налагодження програми є додавання кількох операторів друку до джерела.

Зазвичай очікується, що програми на Python працюватимуть повільніше, ніж програми на Java, але їх розробка потребує набагато менше часу. Програми на Python зазвичай у 3-5 разів коротші за еквівалентні програми на Java. Цю різницю можна пояснити вбудованими високорівневими типами даних Python і його динамічною типізацією. Наприклад, програміст на Python не витрачає час на оголошення типів аргументів або змінних, а потужні поліморфні типи списків і словників Python, для яких розширена синтаксична підтримка вбудована прямо в мову, знаходять застосування майже в кожній програмі на Python. Через типізацію

під час виконання, час виконання Python має бути більше, ніж Java. Наприклад, коли обчислюється вираз a+b, він повинен спочатку перевірити об’єкти a і b, щоб дізнатися їхній тип, який невідомий під час компіляції. Потім він викликає відповідну операцію додавання, яка може бути перевантаженим методом, визначеним користувачем. Java, з іншого боку, може виконувати ефективне додавання цілих чисел або з плаваючою комою, але вимагає оголошення змінних для a і b і не дозволяє перевантажувати оператор + для екземплярів визначених користувачем класів.

З цих причин Python набагато краще підходить як «склеюча» мова, тоді як Java краще характеризується як мова низькорівневої реалізації. Фактично, обидва разом утворюють чудове поєднання. Компоненти можна розробляти на Java та комбінувати для створення додатків на Python [24].

Згідно з опитувань основними бібліотеками спеціалісти з аналізу даних називають NumPy, Pandas і Matplotlib. Популярність цих та інших бібліотек продемонстрована в таблиці 1.1

NumPy	62%
Pandas	51%
Matplotlib	46%
SciPy	38%
SciKit-learn	31%
TensorFlow	25%
Keras	15%
Seaborn	15%
NLTK	13%
Gensim	4%
Theano	3%
Інші	27%

Таблиця 2.1 – Популярність бібліотек і фреймворків в сфері науки про дані

Pandas — це одна з тих бібліотек для аналізу даних, яка містить високорівневі структури даних та інструменти для простого маніпулювання даними. Забезпечення легкого, але ефективного способу аналізу даних потребує здатності індексувати, витягувати, розділяти, об'єднувати, реструктурувати та здійснювати різноманітні інші аналізи як багатовимірних, так і одновимірних даних [25].

Плюси використання Pandas

- Pandas дозволяє легко та простіше представляти дані, покращуючи аналіз і розуміння даних. Для наукових проєктів даних таке просте представлення даних допомагає отримати кращу інформацію.
- Pandas є високоефективним, оскільки дозволяє виконувати будь-яке завдання, написавши лише кілька рядків коду.
- Pandas надає користувачам широкий набір команд для швидкого аналізу даних.

NumPy — це бібліотека Python для числових і наукових обчислень. NumPy надає численні функції, які ентузіасти та програмісти Python можуть використовувати для роботи з високопродуктивними масивами та матрицями. Масиви NumPy забезпечують векторизацію математичних операцій, що дає приріст продуктивності порівняно з циклічними конструкціями Python[26].

Плюси використання NumPy

- NumPy забезпечує ефективне та масштабоване зберігання даних і краще керування даними для математичних обчислень.
- Масив Numpy містить безліч функцій, методів і змінних, які спрощують обчислення матриць.

SciPy — це набір математичних функцій і алгоритмів, побудованих на розширенні Python NumPy. SciPy надає різні високорівневі команди та класи для маніпулювання та візуалізації даних. SciPy корисний для систем обробки даних і створення прототипів. Він містить модулі для статистики, оптимізації,

інтеграції, лінійної алгебри, перетворень Фур'є, обробки сигналів і зображень, розв'язувачів ODE тощо [27].

Плюси використання SciPy

- Візуалізація та маніпулювання даними за допомогою команд і класів високого рівня.
- Для паралельного програмування існують класи, веб-процедури та процедури бази даних.

Sci-Kit Learn має контрольовані та неконтрольовані алгоритми машинного навчання для виробничих програм. Sci-Kit Learn зосереджується на якості коду, документації, простоті використання та продуктивності, оскільки ця бібліотека надає алгоритми навчання[28].

- Бібліотека навчання scikit є корисним інструментом для прогнозування поведінки клієнтів, розробки нейрообразів тощо.
- Він простий у використанні та абсолютно безкоштовний.

TensorFlow — безкоштовна наскрізна платформа з відкритим вихідним кодом для машинного навчання, яка включає широкий спектр інструментів, бібліотек і ресурсів. Команда Google Brain вперше випустила його 9 листопада 2015 року. TensorFlow спрощує розробку та навчання моделей машинного навчання за допомогою API високого рівня, таких як Keras. Він також пропонує різні рівні абстракції, що дозволяє вибрати найкращий підхід для вашої моделі. TensorFlow також дозволяє розгортати моделі машинного навчання в різних середовищах, включаючи хмару, браузер і персональний пристрій. TensorFlow надає набір робочих процесів із інтуїтивно зрозумілими API високого рівня для початківців і експертів для створення моделей машинного навчання багатьма мовами. Розробники мають можливість розгортати моделі на низці платформ, наприклад на серверах, у хмарі, на мобільних і периферійних пристроях, у браузерах і на багатьох інших платформах JavaScript. Це дозволяє розробникам набагато легше переходити

від створення моделі та навчання до розгортання. [29].

OpenCV, ліцензований BSD, є безкоштовною бібліотекою машинного навчання та комп'ютерного зору. Він пропонує спільну архітектуру для додатків комп'ютерного бачення для оптимізації впровадження комп'ютерного бачення в комерційних продуктах[30].

SQLAlchemy — це набір інструментів бази даних на Python, який допомагає ефективно отримувати доступ до сховищ даних. Він містить найбільш широко реалізовані шаблони для високопродуктивного доступу до бази даних. SQLAlchemy ORM і SQLAlchemy Core є двома основними компонентами SQLAlchemy. Охоплюючи API та характеристики баз даних Python, ядро SQLAlchemy додає рівень абстракції. Він також надає користувачам інструкції та схеми SQL. SQLAlchemy ORM є автономним об'єктно-реляційним картографом. SQLAlchemy дозволяє розробникам контролювати свої бази даних, одночасно автоматизуючи надлишкові дії[31].

BeautifulSoup — це бібліотека для збирання та аналізу даних Python , яка збирає вихідні дані HTML і XML. Це дозволяє дослідникам даних розробити веб-сканер, який сканує веб-сайти. BeautifulSoup може отримувати дані та структурувати їх у потрібному форматі. Зібрані дані HTML містять багато зашифрованих веб-даних, які користувачі не можуть інтерпретувати. Його остання версія, BS4 (BeautifulSoup 4), упорядковує перемішані веб-дані в прості для розуміння структури XML, що дозволяє аналізувати дані. BeautifulSoup автоматично визначає кодування та плавно інтерпретує HTML-документи, включно зі спеціальними символами[32].

Ggplot — це бібліотека візуалізації даних Python , заснована на реалізації ggplot2 для мови програмування R, з рейтингом 3 тисячі зірок на Github. Ggplot може створювати візуалізації даних, такі як гістограми, секторні діаграми, гістограми, діаграми розсіювання, діаграми помилок тощо за допомогою API високого рівня. Це також дозволяє об'єднувати різні компоненти або шари візуалізації даних у комбіновану візуалізацію.

2.2 Огляд середовища розробки

В якості середовища розробки програмної частини обрано середовище Jupyter. Project Jupyter — це некомерційний проєкт із відкритим вихідним кодом, який народився на основі проєкту IPython у 2014 році, оскільки він розвивався для підтримки інтерактивної науки про дані та наукових обчислень усіма мовами програмування, Jupyter завжди буде на 100% відкритим програмним забезпеченням, безкоштовним для всіх і випущеним на ліберальних умовах модифікованої ліцензії BSD [33]. Jupyter розробляється відкрито на GitHub завдяки консенсусу спільноти Jupyter. Усі онлайнові та особисті взаємодії та спілкування, безпосередньо пов'язані з проєктом, регулюються Кодексом поведінки Jupyter . Цей Кодекс поведінки встановлює очікування, які дозволять різноманітній спільноті користувачів і співавторів брати участь у проєкті з повагою та безпекою.

- Редагування коду в браузері з автоматичним підсвічуванням синтаксису, відступами та завершенням табуляції/самоаналізом.
- Можливість виконання коду з браузера з додаванням результатів обчислень до коду, який їх згенерував.
- Відображення результату обчислення з використанням мультимедійних представлень, таких як HTML, LaTeX, PNG, SVG тощо. Наприклад, показники якості публікації, відтворені бібліотекою matplotlib , можна включити вбудовано.
- Редагування форматованого тексту в браузері за допомогою мови розмітки Markdown , яка може надавати коментарі до коду, не обмежується звичайним текстом.
- Здатність легко включати математичну нотацію в комірки розцінки за допомогою LaTeX і відтворювати їх у MathJax .

Документи Jupyter блокнота містять вхідні та вихідні дані інтерактивного сеансу, а також додатковий текст, який супроводжує код, але

не призначений для виконання. Таким чином, файли блокнота можуть служити повним обчислювальним записом сеансу, чергуючи виконуваний код з пояснювальним текстом, математикою та багатими представленнями отриманих об’єктів. Ці документи є внутрішніми файлами JSON і зберігаються з .ipynb розширенням[34].

Крім того, будь-який .ipynb документ блокнота, доступний за загальнодоступною URL-адресою, можна поширити через Jupyter Notebook Viewer <nbviewer>. Ця служба завантажує документ блокнота з URL-адреси та відображає його як статичну веб-сторінку. Таким чином, результатами можна поділитися з колегою або у вигляді публічної публікації в блозі, без необхідності іншим користувачам самостійно встановлювати блокнот Jupyter.

Таким чином, звичайний робочий процес у блокноті дуже схожий на стандартний сеанс IPython, з тією різницею, що ви можете редагувати комірки на місці кілька разів, поки не отримаєте бажаних результатів, а не повторно запускати окремі сценарії за допомогою %run магічної команди[35].

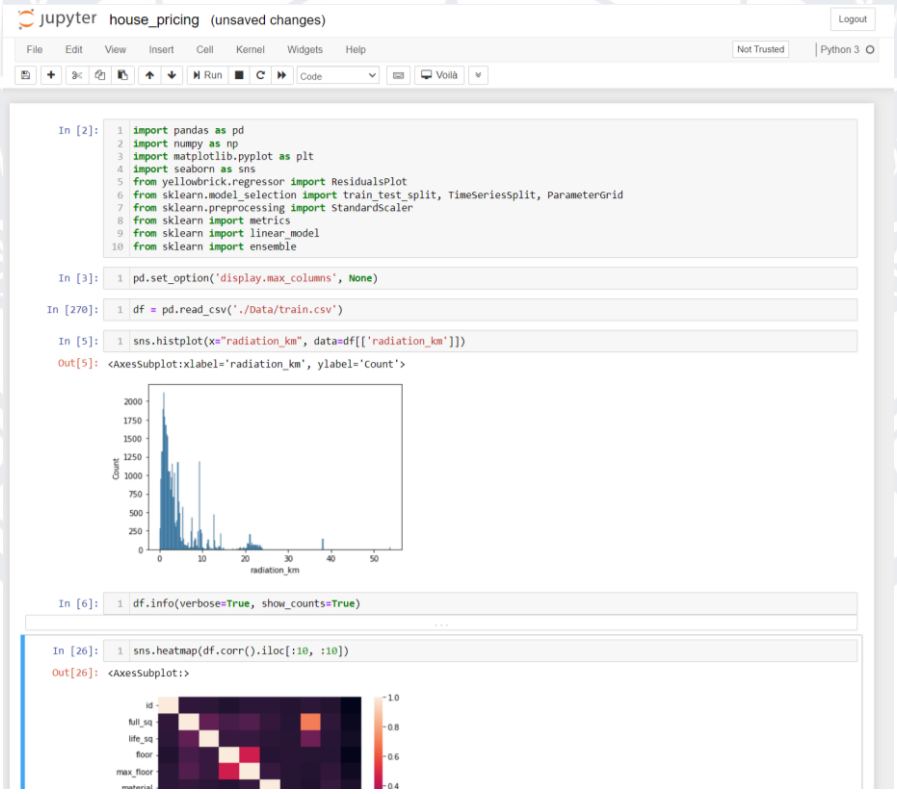


Рисунок 2.2 – Інтерфейс Jupyter ноутбука

Як правило, ви працюватимете над обчислювальною задачею по частинах, організовуючи пов’язані ідеї в клітинки та рухаючись вперед, коли попередні частини працюватимуть правильно. Це набагато зручніше для інтерактивного дослідження, ніж розбивати обчислення на сценарії, які повинні виконуватися разом, як це було необхідно раніше, особливо якщо частина їх виконується довго.

JupyterLab — це користувальницький інтерфейс наступного покоління для Project Jupyter, який пропонує всі знайомі будівельні блоки класичного Jupyter Notebook, а саме блокнот, термінал, текстовий редактор, файловий браузер, розширені виходи у гнучкому користувацькому інтерфейсі.

JupyterLab можна розширити за допомогою пакетів npm, які використовують публічні API. Попередньо зібрані розширення можна поширювати через PyPI, conda та інші менеджери пакетів. Вихідні розширення можна встановити безпосередньо з npm, але для цього потрібен додатковий етап збірки.

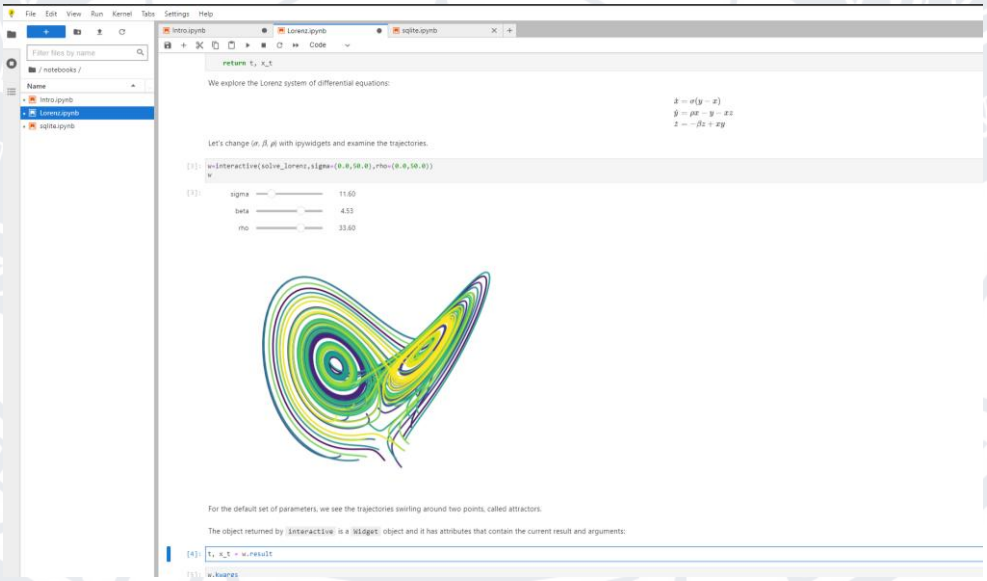


Рисунок 2.3 – Інтерфейс Jupyter lab

Крім того, JupyterLab дозволяє використовувати текстові редактори, термінали, засоби перегляду файлів даних та інші компоненти, що настроюються поруч із блокнотами в розбитій на вкладки робочій області.

- Використовується drag-and-drop для того, щоб можна було переупорядкувати комірки та скопіювати їх між блокнотами.
- Блоки коду виконуються інтерактивно прямо з текстових файлів (.py, .R, .md, .tex тощо).
- Консоль коду можна зв'язати з ядром блокнота, щоб вивчати код в інтерактивному режимі, не захаращуючи блокнот тимчасовими правками.
- Можливість редагування популярних форматів файлів із попереднім переглядом у реальному часі, такі як Markdown, JSON, CSV, Vega, VegaLite та інші.

JupyterLab побудований на системі розширень, які дозволяють налаштовувати та покращувати середовище під себе. Фактично, вся вбудована функціональність самого JupyterLab, а саме блокноти, термінали, браузер файлів, система меню і інше, забезпечується набором основних розширень.

Оскільки значна частина навчання нейронної мережі потребує частих виправлень, підбору параметрів та інших схожих речей – розробка програмної частини саме в цьому середовищі є обґрунтованою.

2.3 Огляд даних для навчання та бібліотек взаємодії з ними

В якості навчальних даних був обраний датасет Airbus Ship Detection з сайту для змагань kaggle.com, на це було декілька причин:

1) Обраний датасет містить вже підготовлені дані, тому не потрібно витрачати час на очистку і можна зосередитись на меті роботи.

2) Датасет налічує близько 193 тисяч тренувальних зображень, що є достатнім для навчання нейронної мережі

3) Надані зображення мають різні погодні умови, наближення, якість та інші відмінності, що дозволяє навчити мережу в більшості можливих ситуацій

4) Платформа дає можливість порівняти результат розпізнавання з

іншими учасникам, хоча багато з фіналістів користуються ансамблями нейромереж чи вдаються до хитрощів, але можна отримати загальне розуміння про те, наскільки створена нейронна мережа точно справляється з своєю задачею

Kaggle — це платформа онлайн-спільноти для науковців із обробки даних та ентузіастів машинного навчання. Kaggle дозволяє користувачам співпрацювати з іншими користувачами, знаходити та публікувати набори даних, використовувати блокноти з інтегрованим графічним процесором і конкурувати з іншими дослідниками даних для вирішення завдань науки про дані. Мета цієї онлайн-платформи (заснованої в 2010 році Ентоні Голдблумом і Джеремі Говардом і придбаної Google у 2017 році) полягає в тому, щоб допомогти професіоналам і учням досягти своїх цілей у своїй подорожі наукою про дані за допомогою потужних інструментів і ресурсів, які вона надає. Станом на сьогодні (2021) на Kaggle зареєстровано понад 8 мільйонів користувачів[36].

На наступному рисунку зображене одне з фото обраного датасету, що дозволяє зрозуміти, що окрім водних пейзажів серед зображень трапляються і обширні ділянки сущі, що є корисним, адже при навчанні нейронної мережі в різних умовах отримана програма буде менше помилятись, коли подібні умови буде бачити на реальних даних.

Рисунок 2.4 – Приклад фото з навчальної вибірки

Окрім наведених вище даних були також використані відібрані вручну малі картинки кораблів, а саме 4 тисячі картинок, з яких 1 тисячу можна ідентифікувати як «корабель», а 3 тисячі – елементи ландшафту, шматки кораблів, шматки інфраструктури та інше.

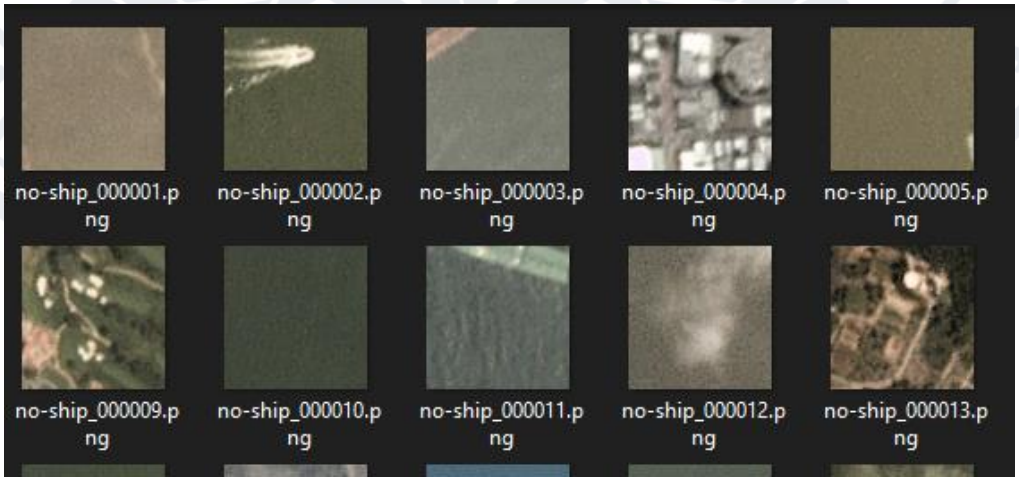


Рисунок 2.5 – Приклад даних з навчальної вибірки «не кораблів»

Різноманіття прикладів важливе для якісного навчання нейронної мережі, однак оскільки є деякий дисбаланс кількості прикладів з кораблями і без то в подальшому необхідно буде працювати з цими даними додатково.

Рисунок 2.6 – Приклад даних з навчальної вибірки кораблів

Всі зображення мають роздільну здатність 80x80 пікселів, формат файлу PNG, розмір цієї частини вибірки – близько 50 мегабайт.

Далі розглянуто основну вибірку, а саме кількість кораблів на кожному фото датасеті.

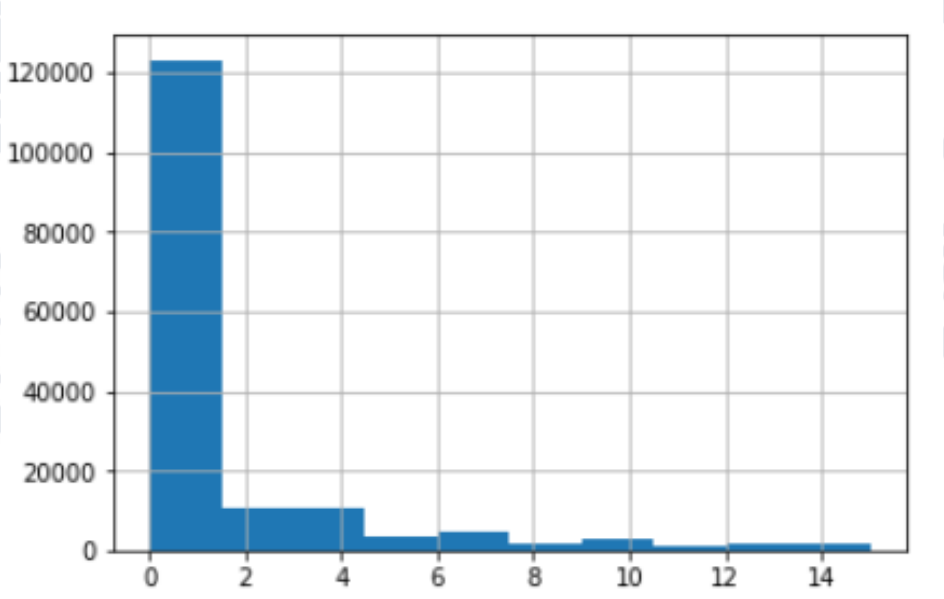


Рисунок 2.7 – Кількість кораблів на фото

Також бачимо що є дисбаланс який треба буде усувати, адже більшість фото мають або 0 кораблів, або 1. Подібне розподілення буде проблемою, адже нейромережа буде мало стикатись з тими випадками, коли кораблів на фото багато.

Можливе вирішення цієї ситуації – поєднання багатьох фото де є один корабель в одне велике задля створення штучних фото з більшою кількістю кораблів.

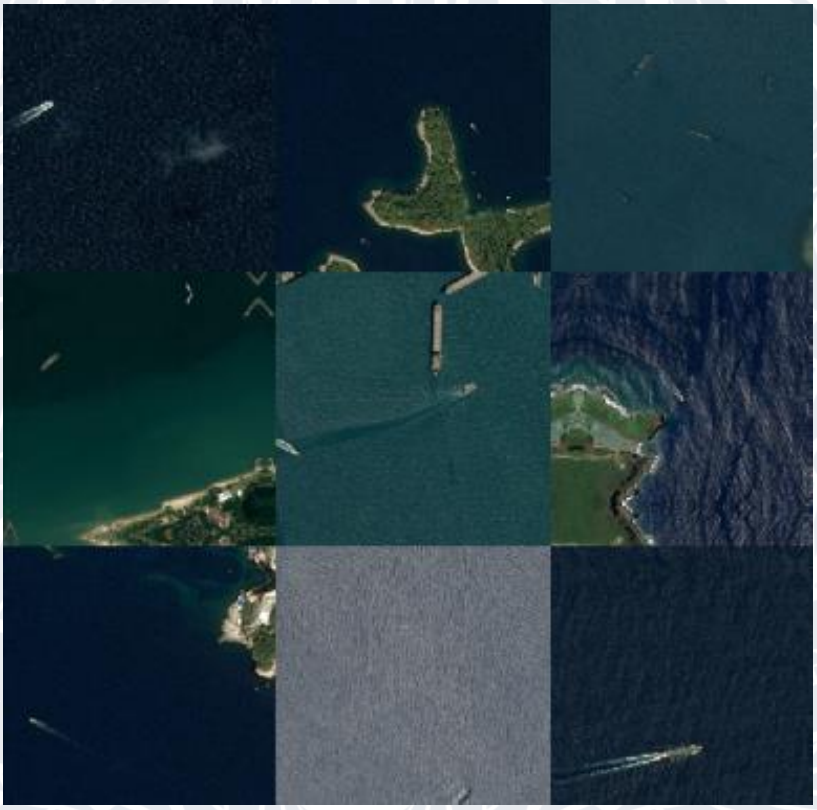


Рисунок 2.8 – Приклад поєднання зображень

Зміна ваги класу – теж може бути вирішенням проблеми, для розпізнавання кораблів важливіше розпізнати всі що є на фото, навіть якщо мережа при цьому позначить кораблями деякі об’єкти які не є ними, ніж упустити деякі з кораблів, але при цьому не позначити хибні об’єкти, адже в першому випадку людина зможе швидко відсіяти непотрібне і залишити для подальшої роботи тільки кораблі.

2.4 Огляд існуючих підходів до розпізнавання об’єктів

Розпізнавання об’єктів — це загальний термін для опису набору пов’язаних завдань комп’ютерного зору, які передбачають ідентифікацію об’єктів на цифрових фотографіях[37].

Класифікація зображень передбачає передбачення класу одного об’єкта на зображенні. Локалізація об’єкта означає визначення розташування одного чи кількох об’єктів на зображенні та малювання прямокутника навколо їх розміру. Виявлення об’єктів поєднує ці два завдання та локалізує та класифікує один або декілька об’єктів на зображенні.

Таким чином, ми можемо розрізнити ці три завдання комп’ютерного зору:

1 Класифікація зображень : завдання присвоєння мітки або класу всьому зображенню. Зображення повинні мати лише один клас для кожного зображення. Моделі класифікації зображень приймають зображення як вхідні дані та повертають прогноз про те, до якого класу належить зображення. У класифікаторах на основі CNN початкові рівні — це згорткові шари нейронної мережі, що складаються від кількох шарів до 100 шарів (наприклад, ResNet 101), і це залежить від програми, кількості даних і доступних обчислювальних ресурсів. Кількість шарів сама по собі є великою областю дослідження. Після шарів CNN є шар об’єднання, а потім один або два повністю з’єднаних шари. Останній шар є вихідним шаром, який дає ймовірність присутності об’єкта на зображенні. Наприклад, припустимо, що алгоритм ідентифікує 100 різних об’єктів на даному зображенні. Потім останній шар дає масив довжиною 100 і значеннями в діапазоні від 0 до 1, які позначають ймовірність присутності об’єкта на зображенні[38].

Вхід : зображення з одним об’єктом, наприклад фотографія.

Вихід: мітка класу (наприклад, одне або кілька цілих чисел, які зіставляються з мітками класу).

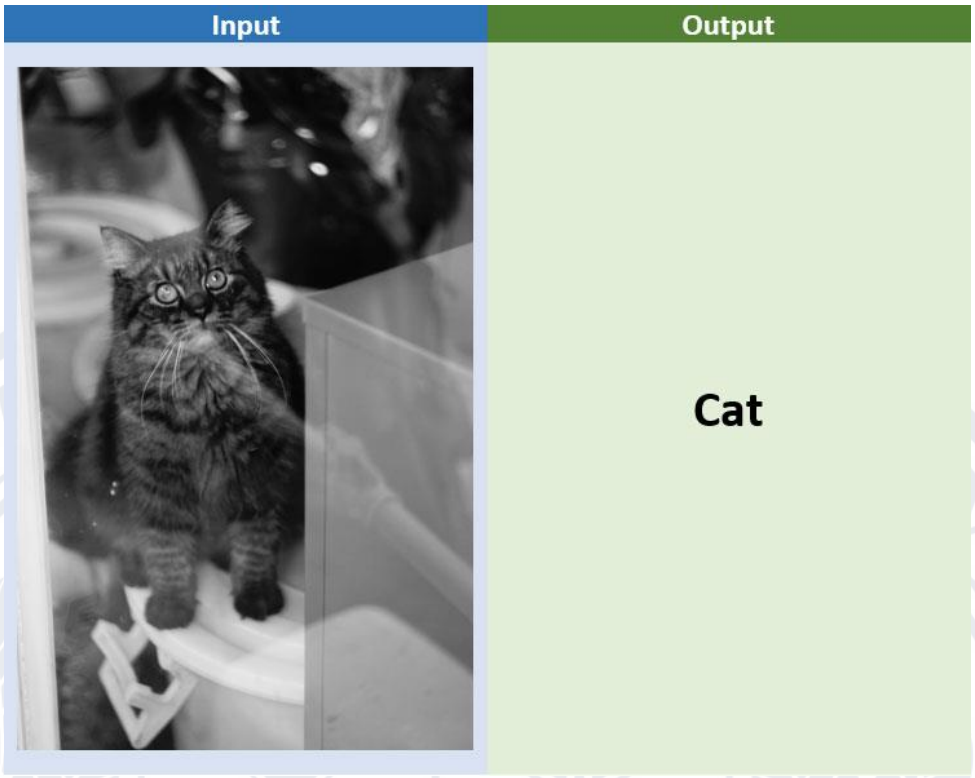


Рисунок 2.9 – Приклад класифікації зображень

2 Локалізація об’єктів : визначення наявності об’єктів на зображенні та вказування їх розташування за допомогою обмежувальної рамки. Локалізація зображення є побічним результатом звичайних алгоритмів зору CNN. Ці алгоритми передбачають класи з дискретними числами. Під час локалізації об’єкта алгоритм передбачає набір із 4 чисел, а саме координату x, координату y, висоту та ширину, щоб намалювати обмежувальну рамку навколо цільового об’єкта[39].

Вхід : зображення з одним або кількома об’єктами, наприклад фотографія.

Вихід : одна або кілька обмежувальних рамок (наприклад, визначених точкою, шириною та висотою).

Рисунок 2.10 – Приклад локалізації об’єктів

3 Виявлення об’єктів : техніка комп’ютерного зору для визначення місцезнаходження об’єктів на зображеннях або відео. Алгоритми виявлення об’єктів зазвичай використовують машинне або глибоке навчання для отримання значущих результатів. Це визначення наявності об’єктів за допомогою обмежувальної рамки та типів або класів розташованих об’єктів на зображенні. По своїй суті це комплексна задача, яка поєднує поняття локалізації та класифікації зображення. Маючи зображення, алгоритм виявлення об’єктів повертає обмежувальні рамки навколо всіх цікавих об’єктів і призначає їм клас[40].

Вхід : зображення з одним або кількома об’єктами, наприклад фотографія.

Вихід : один або кілька обмежувальних рамок (наприклад, визначених точкою, шириною та висотою) і мітка класу для кожної обмежувальної рамки.

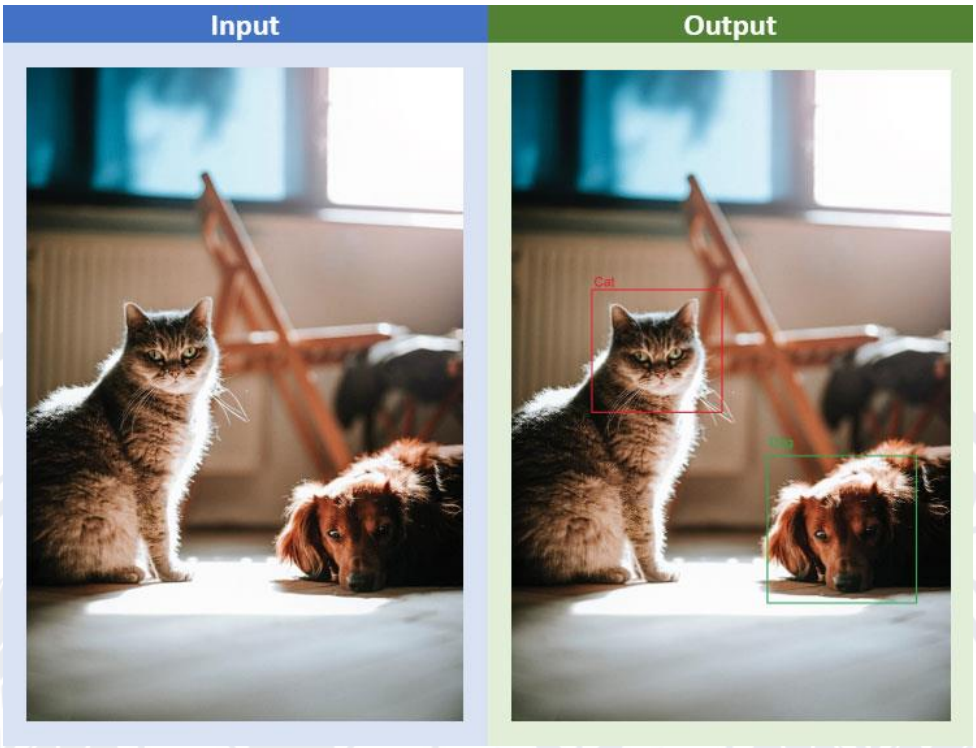


Рисунок 2.11 – Приклад виявлення об’єктів

Продуктивність моделі для класифікації зображень оцінюється за допомогою середньої помилки класифікації для прогнозованих міток класу. Ефективність моделі для локалізації одного об’єкта оцінюється за допомогою відстані між очікуваною та прогнозованою обмежуючою рамкою для очікуваного класу. У той час як продуктивність моделі для розпізнавання об’єктів оцінюється за допомогою точності та запам’ятовування по кожній із найкращих відповідних обмежувальних рамок для відомих об’єктів на зображенні.

Метою будь-якого алгоритму машинного навчання є прогнозування максимально близьких до базових значень істини. Для цього будь-який керований алгоритм машинного навчання використовує функцію втрат, а алгоритми навчаються шляхом мінімізації втрат за допомогою оптимізації ваги або параметрів.

Оскільки локалізація об’єкта є проблемою регресії, можна використовувати будь-яку функцію втрат регресії, застосовну до N-вимірного масиву. Наприклад, втрати відстані L1, відстані L2, втрати Губера тощо[41].

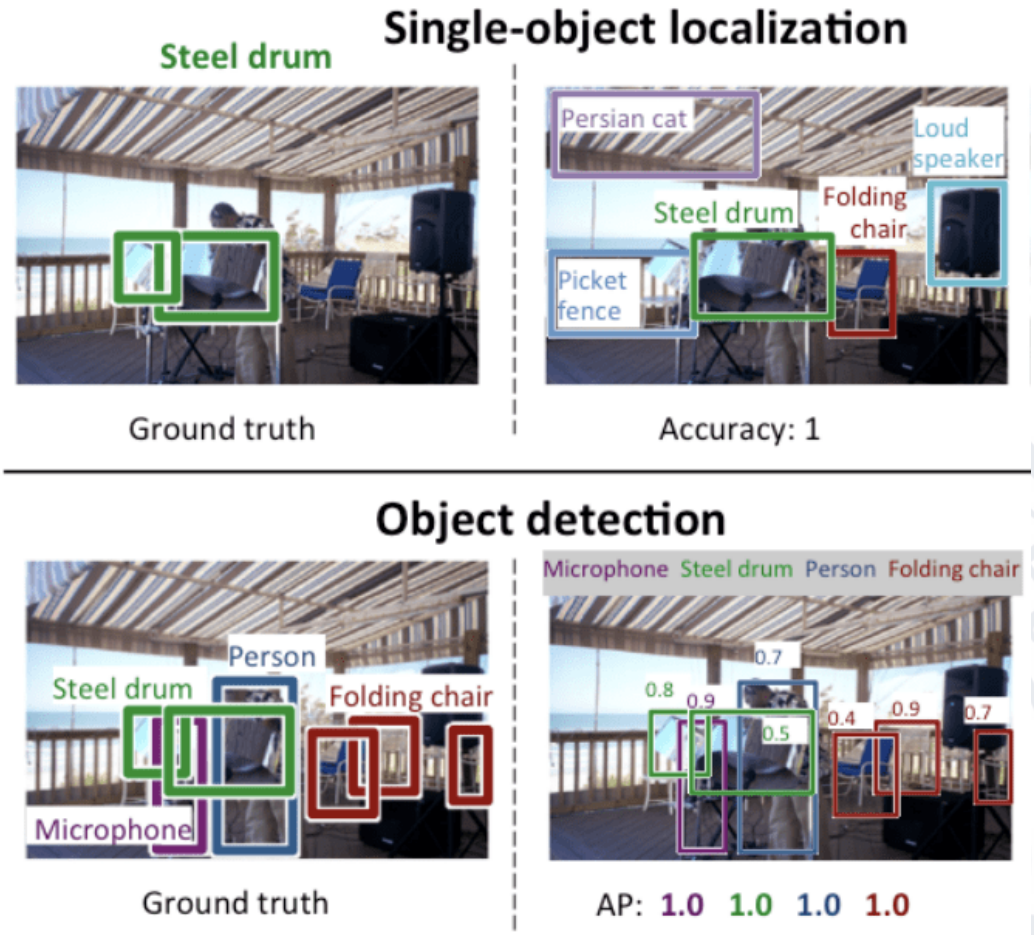


Рисунок 2.12 – Порівняння між локалізацією окремого об’єкта та виявленням об’єкта

R-CNN мережа є одним з існуючих підходів до розпізнавання, вона була першим великим і успішним застосуванням згорткових нейронних мереж до проблеми локалізації, сигментації та виявлення об’єктів[42].

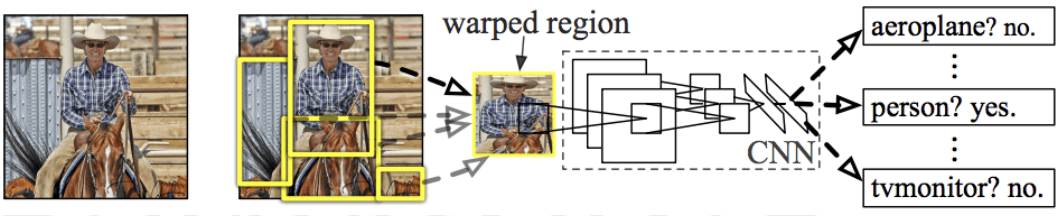


Рисунок 2.13 – Архітектура R-CNN моделі

Будову R-CNN мережі можна поділити на 3 модулі:

- Модуль 1: Регіональна пропозиція – створення та вилучення пропозиції регіонів незалежних від категорій, наприклад рамки-кандидати.

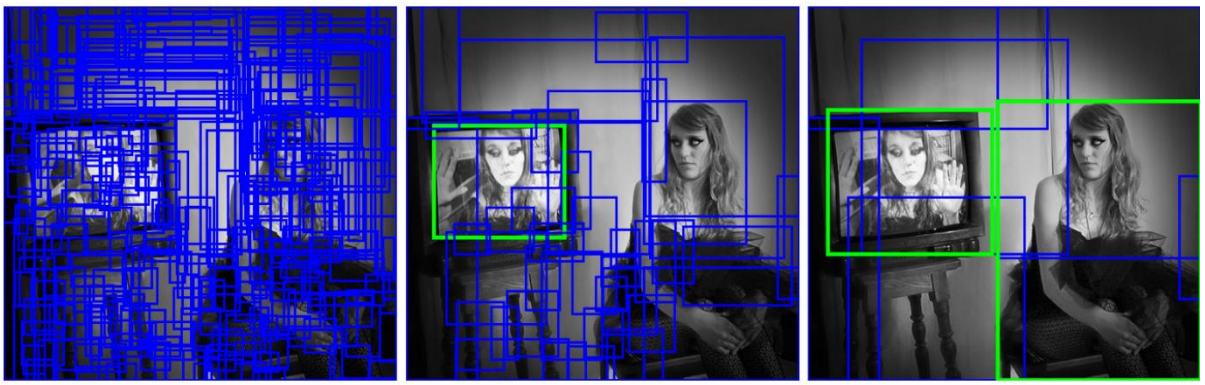


Рисунок 2.14 – Регіональна пропозиція

- Модуль 2: Екстрактор функцій – витягання ознак з кожної кандидатської області, наприклад використовуючи глибоку згортову нейронну мережу. Під час навчання CNN «вивчає» оптимальні значення для матриць фільтрів, які дозволяють йому витягувати значущі характеристики (текстури, краї, форми) із вхідної карти функцій. Зі збільшенням кількості фільтрів (глибина вихідної карти ознак), застосованих до вхідних даних, збільшується кількість функцій, які CNN може отримати. Однак компроміс полягає в тому, що фільтри складають більшість ресурсів, які витрачає CNN, тому час навчання також збільшується, коли додається більше фільтрів. Крім того, кожен фільтр, доданий до мережі, забезпечує меншу додаткову цінність, ніж попередній, тому інженери прагнуть побудувати мережі, які використовують мінімальну кількість фільтрів, необхідних для виділення функцій, необхідних для точної класифікації зображень.

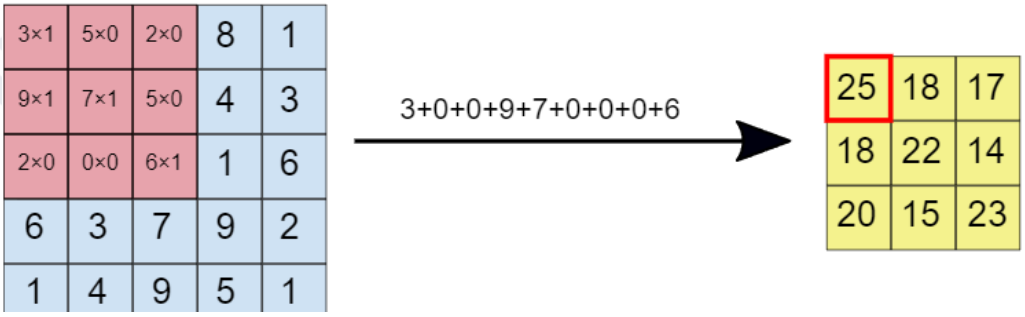


Рисунок 2.15 – Принцип згортки зображення

- Модуль 3: Класифікатор – класифікація об’єкта як один із відомих класів, наприклад за допомогою лінійної моделі класифікатора SVM.

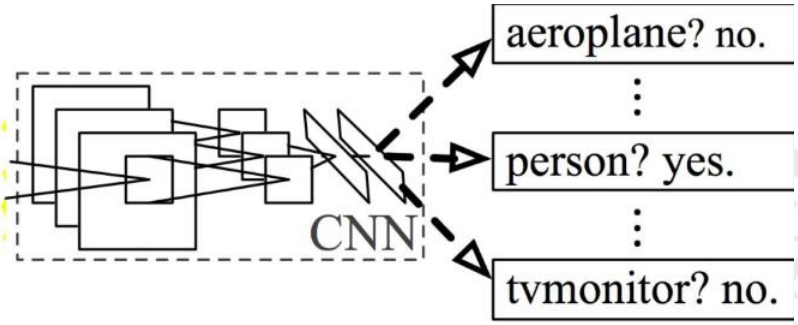


Рисунок 2.16 – Передбачення останнього модуля

Наскрізна структура моделі CNN показана на наступному рисунку.

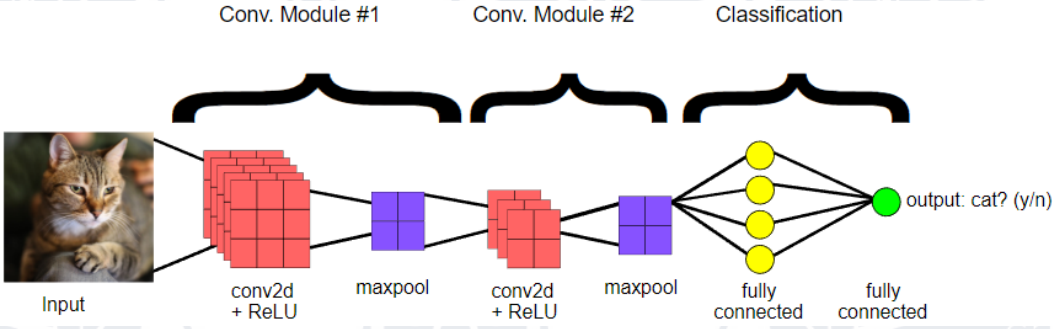


Рисунок 2.17 – Наскрізна структура CNN

До недоліків даної моделі, про які написано в її описі відносять наступне:

- Навчання є багатоетапним конвеєром. Передбачає підготовку та роботу трьох окремих моделей.
- Навчання дороге в просторі та часі. Навчання глибокої CNN на такій кількості регіональних пропозицій на одне зображення відбувається дуже повільно.
- Виявлення об'єктів відбувається повільно. Робляться прогнози за допомогою глибокої CNN на такій кількості регіональних пропозицій дуже повільно.

Іншим підходом до розпізнавання можна вважати сімейство моделей

YOLO. Модель YOLO вперше була описана Джозефом Редмоном та іншими в статті 2015 року під назвою « You Only Look Once: Unified, Real-Time Object Detection ». Зауважте, що Росс Гіршик, розробник R-CNN, також був автором і учасником цієї роботи, тоді у Facebook AI Research.

Підхід включає в себе єдину нейронну мережу, навчену наскрізно, яка приймає фотографію як вхідні дані та прогнозує обмежувальні рамки та мітки класів безпосередньо для кожної обмежувальної рамки. Техніка забезпечує меншу точність прогнозування, наприклад більше помилок локалізації, хоча працює зі швидкістю від 45 кадрів на секунду та до 155 кадрів на секунду для версії моделі, оптимізованої за швидкістю.

Зокрема це дає змогу використовувати цю модель на вуличних камерах, безпілотниках і іншим подібним чином.

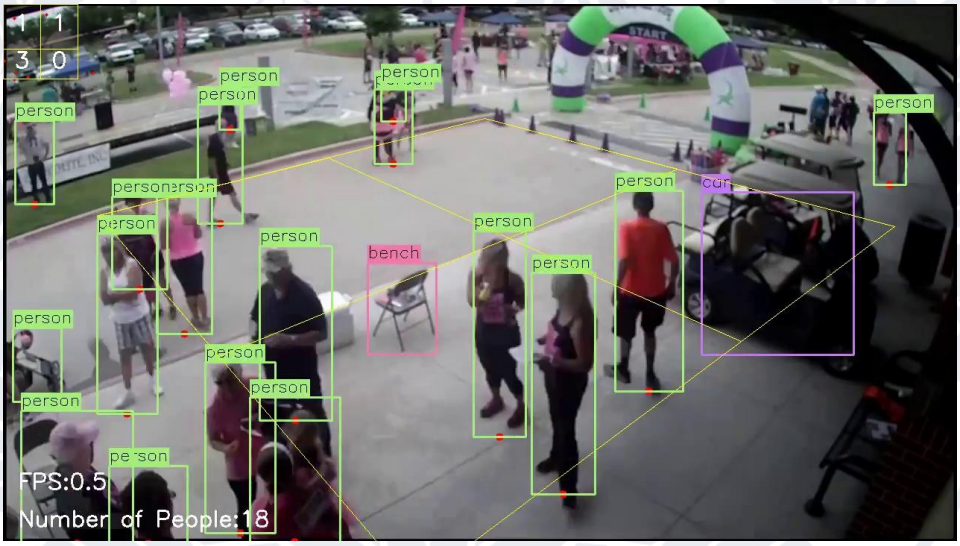


Рисунок 2.18 – робота YOLO моделі

Модель працює спочатку розбиваючи вхідне зображення на сітку клітинок, де кожна клітинка відповідає за прогнозування обмежувальної рамки, якщо центр обмежувальної рамки потрапляє в клітинку. Кожна клітинка сітки передбачає обмежувальну рамку, що включає координати x, y, ширину, висоту та достовірність. Прогноз класу також базується на кожній клітинці.

Наприклад, зображення можна розділити на сітку 7×7, і кожна клітинка

в сітці може передбачати 2 обмежувальні прямокутники, що призводить до 94 запропонованих обмежувальних рамок. Карта ймовірностей класів і обмежувальні прямокутники з достовірністю потім об’єднуються в остаточний набір обмежувальних рамок і міток класів.

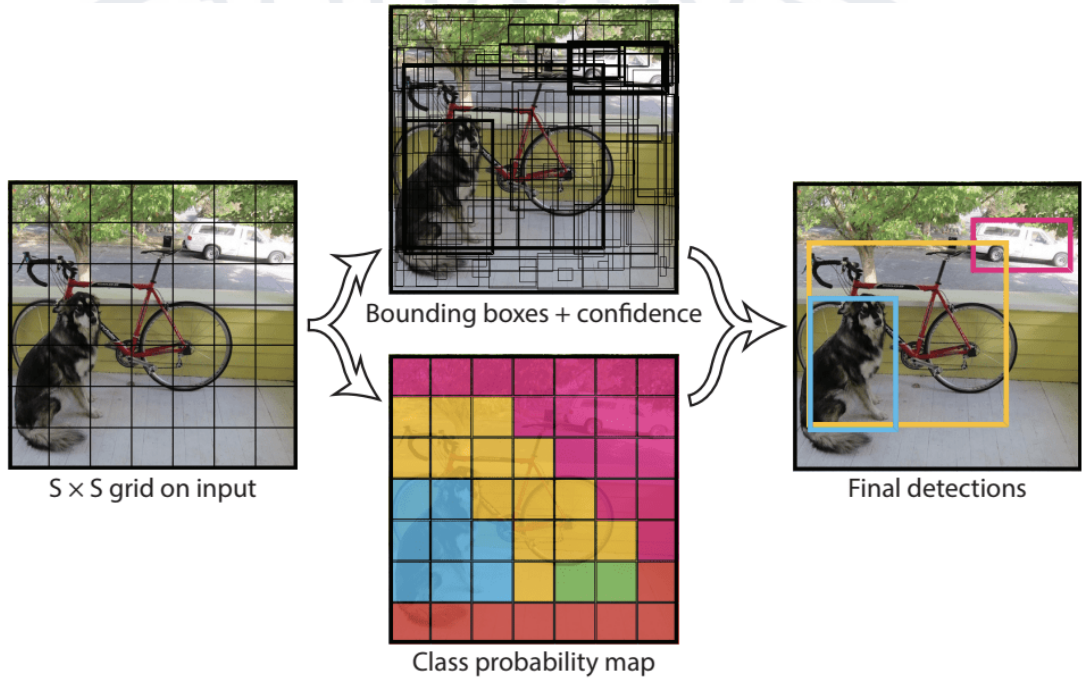


Рисунок 2.19 – Структура YOLO моделі

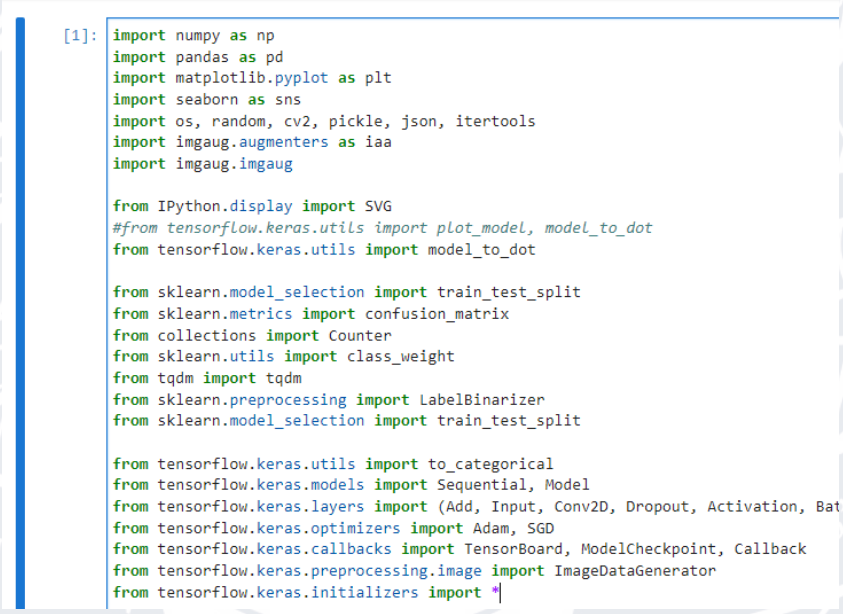
Висновок до розділу 2

У даному розділі були розглянуті обрані технічні засоби з реалізації нейронної мережі. Описано обрану мову її переваги та деякі необхідні бібліотеки. Була приділена увага середовищу розробки, та даним що будуть використовуватись для навчання, оглянуто деякі існуючи підходи до розпізнавання.

РОЗДІЛ 3
ПРОЄКТНА ЧАСТИНА

3.1 Робота з даними

Спочатку необхідно імпортувати всі необхідні бібліотеки, для роботи були використані, зокрема, наступні бібліотеки: numpy, pandas, matplotlib, seaborn, os, random, cv2, imgaug, sklearn та tensorflow. Остання наведена бібліотека відіграє ключову роль. На наступному малюнку представлений перший блок коду, який містить в собі імпорт.



```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import os, random, cv2, pickle, json, itertools
     import imgaug.augmenters as iaa
     import imgaug.imgaug

     from IPython.display import SVG
     #from tensorflow.keras.utils import plot_model, model_to_dot
     from tensorflow.keras.utils import model_to_dot

     from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix
     from collections import Counter
     from sklearn.utils import class_weight
     from tqdm import tqdm
     from sklearn.preprocessing import LabelBinarizer
     from sklearn.model_selection import train_test_split

     from tensorflow.keras.utils import to_categorical
     from tensorflow.keras.models import Sequential, Model
     from tensorflow.keras.layers import (Add, Input, Conv2D, Dropout, Activation, Bat
     from tensorflow.keras.optimizers import Adam, SGD
     from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint, Callback
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras.initializers import *
```

Рисунок 3.1 – Імпорт бібліотек

Крім того, через необхідність подбати про оцінку якості моделі, було створено функції show_final_history() і plot_confusion_matrix (). Перша з яких буде відображати історію навчання моделі.

```
]: #Для побудови втрат/точності наборів даних для навчання та перевірки
def show_final_history(history):
    plt.style.use("ggplot")
    fig, ax = plt.subplots(1,2,figsize=(15,5))
    ax[0].set_title('Похибка')
    ax[1].set_title('Точність')
    ax[0].plot(history.history['loss'],label='Похибка на тренувальній вибірці')
    ax[0].plot(history.history['val_loss'],label='Похибка на валідаційній вибірці')
    ax[1].plot(history.history['accuracy'],label='Точність на тренувальній вибірці')
    ax[1].plot(history.history['val_accuracy'],label='Точність на валідаційній вибірці')

    ax[0].legend(loc='upper right')
    ax[1].legend(loc='lower right')
    plt.show();
    pass
```

Рисунок 3.2 – Функція малювання історії навчання

Друга є матрицею заплутаності, вона дозволяє легко побачити, чи модель плутає два класи, тобто неправильно позначає один як інший. Як виглядає в програмному коді реалізаціє цієї функції наведено на наступному малюнку.

```
#Для побудови відсотка справжніх позитивних результатів на клас, щоб краще зрозуміти, як модель передбачила дані
def plot_confusion_matrix(cm,classes,title='Матриця заплутаності',cmap=plt.cm.Blues):
    cm = cm.astype('float')/cm.sum(axis=1)[:,np.newaxis]
    plt.figure(figsize=(10,10))
    plt.imshow(cm,interpolation='nearest',cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes,rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f'
    thresh = cm.max()/2.
    for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
        plt.text(j,i,format(cm[i,j],fmt),
                horizontalalignment="center",
                color="white" if cm[i,j] > thresh else "black")
        pass
    plt.ylabel('Правда')
    plt.xlabel('Передбачення')
    pass
```

Рисунок 3.3 – Функція малювання матриці заплутаності

Після того було перетворено всі картинки в формат матриць з якими більшості модулів набагато легше працювати. Реалізовано даний функціонал було за допомогою opencv-python пакета, завдяки маніпуляції з картинками отримали датасет з закодованими в ньому фото в трьох каналах кольору.

Для навчання моделі було відібрано 4 тисячі фотографій, близько 1000

з них містили кораблі, в той час як близько 3000 не містили кораблі, але містили зображення води, будівель і шматків кораблів, бо яких не можна впевнено розпізнати корабель.

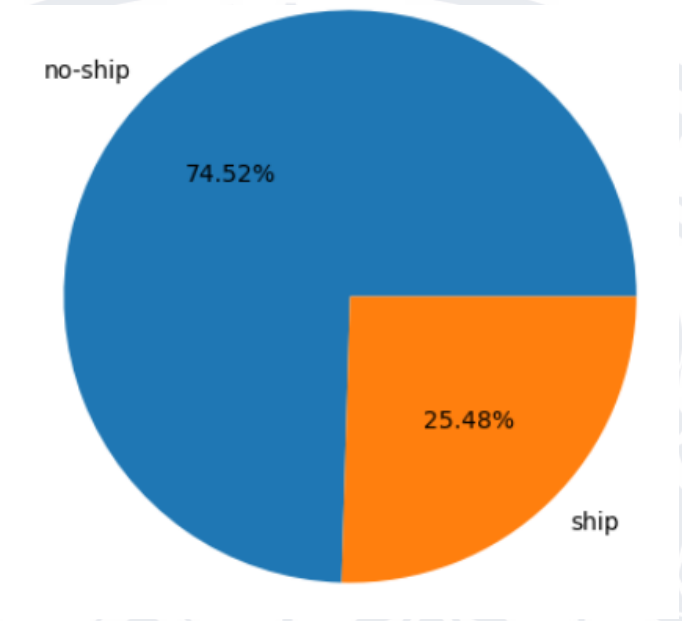


Рисунок 3.4 – Секторна діаграма дисбалансу класів

Через дисбаланс досліджуваних двох класів навчання моделі може бути викривлене, так як мережа буде частіше схилятись до передбачення в бік відсутності корабля, через те, що звичайним вгадуванням, не дивлячись на малюнок вона зможе досягти приблизно 75% точності.

Подібна поведінка є небажаною, її можна усунути через аугментацію даних, тобто через дублювання меншого з двох класів до тих пір, поки вони не вирівняються, або через введення вагових коефіцієнтів для кожного класу, таким чином штраф за невірне передбачення зросте для меншого з класів.

Був обраний метод аугментації даних, хоча і реалізовані обидва, вибір обумовлювався більшою зручністю роботи з класами.

Після обробки даних розподілення класів стало близьким до 1/1, що підходить для подальшої роботи з ними.

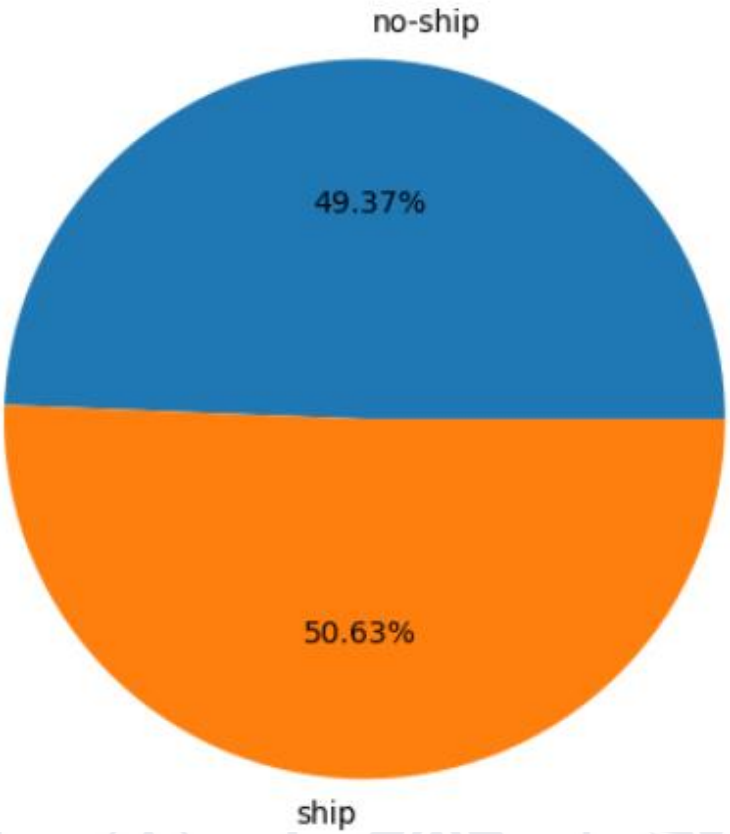


Рисунок 3.5 – Нова секторна діаграма класів

Перед тим, як далі працювати з даними їх необхідно розбити на 3 різні групи, про що було описано в першому розділі, це тренувальні дані, дані валідації та тестові дані.

Зручніше всього це робити розглянутою раніше бібліотекою sklearn, а саме її частиною sklearn.model_selection.train_test_split.

```
train_images, test_images, train_labels, test_labels= train_test_split(images, labels, test_size=0.4, random_state=42)
test_images, val_images, test_labels, val_labels= train_test_split(test_images, test_labels, test_size=0.5, random_state=42)
```

Рисунок 3.6 – Розбиття даних

В даному випадку ми відділяємо 60% даних для навчання, а решту порівну розподіляємо на валідацію і тестування.

3.2 Розробка моделі

Для розпізнавання було вирішено модифікувати R-CNN нейронну

мережу сімейство якої було розглянуто в другому розділі. В ній карта функцій створена згортковими шарами, яка передається до мережі регіональних пропозицій і обробляється повністю зв'язаними шарами, замість логістичної регресії, опорних вершин чи випадкових лісів, для того, щоб пришвидшити мережу.

```
def conv_block(X,k,filters,stage,block,s=2):

    conv_base_name = 'conv_' + str(stage)+block+'_branch'
    bn_base_name = 'bn_'+str(stage)+block+"_branch"

    F1 = filters

    X = Conv2D(filters=F1, kernel_size=(k,k), strides=(s,s),
               padding='same',name=conv_base_name+'2a')(X)
    X = BatchNormalization(name=bn_base_name+'2a')(X)
    X = Activation('relu')(X)

    return X
    pass
```

Рисунок 3.7 – Створення рівня згортки

conv_block – функція містить згортковий рівень, пакетну нормалізацію та рівні активації.

basic_model – функція створює модель за допомогою вищезгаданої функції, має максимальну кількість шарів об'єднання та вилучення.

Conv2D – двовимірний згортковий шар, кількість фільтрів визначає, що вивчає згортковий рівень. Чим більше фільтрів, тим більше отриманої інформації.

MaxPooling2D – це зменшує просторові розміри карти функцій, створеної згортковим шаром, без втрати інформації про діапазон. Це дозволяє зробити модель трохи надійнішою.

Dropout – видаляє визначений користувачем відсоток зв'язків між нейронами послідовних шарів. Його можна використовувати як у повністю згорткових, так і в повнозв'язаних шарах.

BatchNormalization – цей рівень нормалізує значення, присутні в прихованій частині нейронної мережі. Це схоже на масштабування MinMax/Standard, що застосовується в алгоритмах машинного навчання.

Padding – це доповнює карту об'єктів/вхідне зображення нулями, дозволяючи межі залишатися доступною.

```
def basic_model(input_shape,classes):

    X_input = Input(input_shape)
    X = ZeroPadding2D((5,5))(X_input)
    X = Conv2D(16,(3,3),strides=(2,2),name='conv1',padding="same")(X)
    X = BatchNormalization(name='bn_conv1')(X)

#    2
    X = conv_block(X,3,32,2,block='A',s=1)
    X = MaxPooling2D((2,2))(X)
    X = Dropout(0.25)(X)

#     3
    X = conv_block(X,5,32,3,block='A',s=2)
    X = MaxPooling2D((2,2))(X)
    X = Dropout(0.25)(X)

#     4
    X = conv_block(X,3,64,4,block='A',s=1)
    X = MaxPooling2D((2,2))(X)
    X = Dropout(0.25)(X)

#   Блок виходу
    X = Flatten()(X)
    X = Dense(64)(X)
    X = Dropout(0.5)(X)
    X = Dense(128)(X)
    X = Activation("relu")(X)
    X = Dense(classes,activation="softmax",name="fc"+str(classes))(X)

    model = Model(inputs=X_input,outputs=X,name='Feature_Extraction_and_FC')

    return model
    pass
```

Рисунок 3.8 – Створення моделі

Спершу вхідний рівень ініціалізується за допомогою вхідного шару Keras, це визначає кількість нейронів, присутніх на вхідному рівні. ZeroPadding застосовується до вхідного зображення, щоб обмежувальні елементи не були втрачені.

Потім згорткові шари, починаються з 16 фільтрів, розміром ядра (3,3) і кроками (2,2). Відступи зберігаються однаковими, тому зображення не змінюється просторово до наступного блоку, у якому відбувається MaxPooling і Dropout.

3-4 етап має подібну структуру з згортковим шаром, за яким слідують шари MaxPooling і Dropout.

Останній крок бере карту об'єктів, створену попередніми згортковими

шарами, перетворюється в один стовпець за допомогою Flatten Layer і класифікується за допомогою Dense (вихідного шару) з кількістю класів, присутніх у наборі даних і сигмою як функцією активації.

Всього фінальна модель розпізнавання має 74818 параметрів з яких усі окрім 288 є гнучкими для тренування.

Модель доповнено оптимізатором Adam з швидкістю навчання 0.001, використана бінарна кросентропія, так як в нас тільки 2 класи.

3.3 Аналіз результатів

Після тренування модель розміщується в окремому файлі «model_weight.h5».

```
Epoch 1/45
221/226 [============================>.] - ETA: 0s - loss: 0.4448 - accuracy: 0.7963
Epoch 1: val_accuracy improved from -inf to 0.78394, saving model to model_weights.h5
226/226 [==============================] - 4s 11ms/step - loss: 0.4400 - accuracy: 0.7990 - val_loss: 0.4476 - val_accuracy: 0.7839
Epoch 2/45
226/226 [==============================] - ETA: 0s - loss: 0.2540 - accuracy: 0.9041
Epoch 2: val_accuracy improved from 0.78394 to 0.92384, saving model to model_weights.h5
226/226 [==============================] - 2s 9ms/step - loss: 0.2540 - accuracy: 0.9041 - val_loss: 0.2273 - val_accuracy: 0.9238
Epoch 3/45
226/226 [==============================] - ETA: 0s - loss: 0.1972 - accuracy: 0.9258
Epoch 3: val_accuracy improved from 0.92384 to 0.94205, saving model to model_weights.h5
226/226 [==============================] - 2s 10ms/step - loss: 0.1972 - accuracy: 0.9258 - val_loss: 0.1432 - val_accuracy: 0.9421
Epoch 4/45
222/226 [============================>.] - ETA: 0s - loss: 0.1642 - accuracy: 0.9388
Epoch 4: val_accuracy improved from 0.94205 to 0.96275, saving model to model_weights.h5
226/226 [==============================] - 2s 10ms/step - loss: 0.1641 - accuracy: 0.9391 - val_loss: 0.0997 - val_accuracy: 0.9627
```

Рисунок 3.9 – Фрагмент навчання

Вище наведений фрагмент навчання моделі , як видно з історії – з кожним кроком помилка зменшується, а точність зростає, це означає що модель навчається нормально. Для більш детального розбору процесу навчання варто скористатись візуалізацією історії навчання.

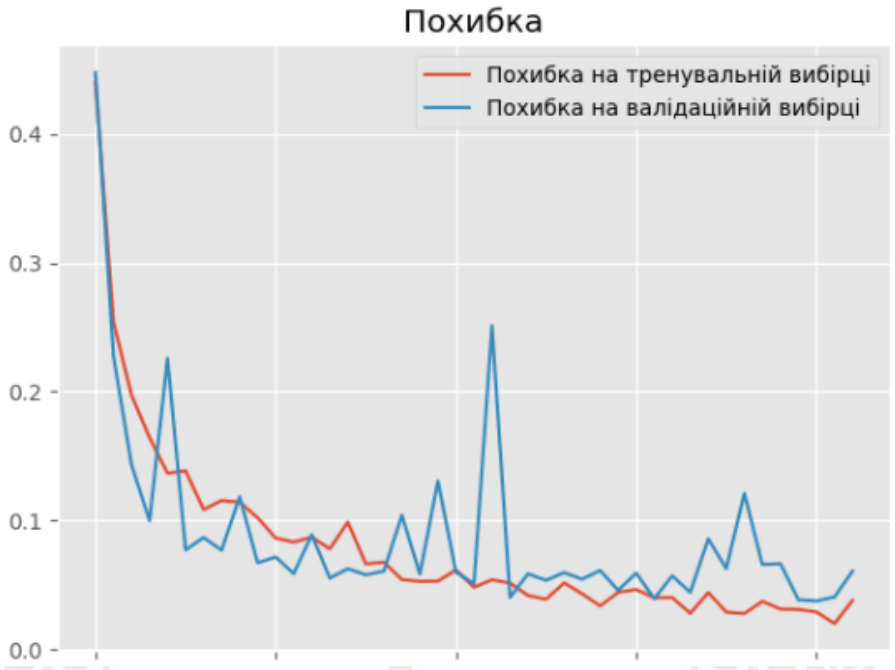


Рисунок 3.10 – Історія похибки

Крім історії похибки проаналізовано і історію точності, що я різними параметрами які не завжди мають стійку протилежність.

Рисунок 3.11 – Історія точності

З отриманих візуалізацій ми бачимо що модель добре розрізняє отримані дані в тому числі на валідаційній вибірці, що говорить про те, що ця нейромережа дійсно навчилась відрізняти кораблі, а не просто запам’ятала

дані.

Щоб дійсно впевнитись в якості передбачення поглянемо на матрицю заплутаності, за допомогою якої зможемо побачити всі картину передбачень, адже вона являє собою матрицю 2 на 2 в якій видно відношення реальності/правди, до нашого передбачення. Тобто матриця має 4 клітинки, які є комбінацією правди і вірного вгадування, правди і невірного вгадування, неправди і вірного вгадування, а також неправди в невірного вгадування[43].

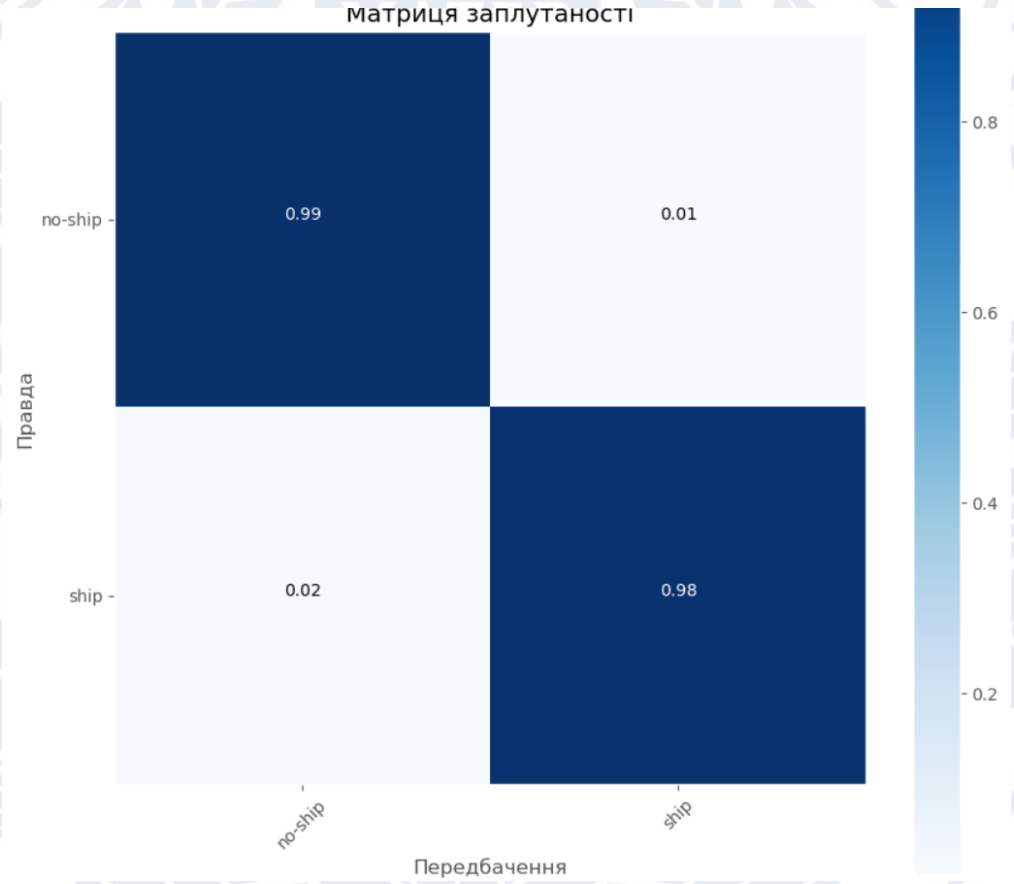


Рисунок 3.12 – Матриця заплутаності

В якості прикладу було виведено декілька зображень по яких можна зробити певні висновки про представлені дані і якість моделі, фрагменти цих зображень наведені нижче.

Рисунок 3.12 – Істино вірне передбачення

На малюнку досить добре видно корабель незважаючи на низьку роздільну здатність, ніяких проблем у моделі не виникло, передбачення вірне. Більш корисними для нас є ті випадки, де передбачення не відповідало дійсності. Наступний малюнок демонструє один з таких випадків.

Рисунок 3.13 – Помилкове передбачення

Незважаючи на те, що корабель досить сильно зливається з кольором води, але людина досить легко може його розрізнити, на відміну від нашої мережі, яка це зробити не змогла. Причиною даної поведінки може бути мала вибірка даних на яких мережа навчалась, збільшення датасету може змінити ситуацію на краще, однак подібних помилок, як видно з матриці заплутаності, в моделі досить мало, тому ними можна просто знехтувати.

3.4 Імплементація моделі

Оскільки модель розпізнавання кораблів була навчена і результат її навчання хороший, наступним кроком є застосування цієї моделі для розпізнавання зображень на великих супутникових знімках з малюванням в відповідних місцях рамки, для зручності перегляду результату людиною.

Спочатку йде процес розбиття отриманого зображення на «регіони інтересу», які потенційно можуть бути кораблями, після чого визначається за допомогою створеної і навченої моделі чи є дана область інтересу кораблем, в випадку якщо так – даний регіон позначається як корабель і створюється відповідна рамка.

```
def find_regions(image, method):

    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
    ss.setBaseImage(image)

    if method == 'fast':
        ss.switchToSelectiveSearchFast()
    else:
        ss.switchToSelectiveSearchQuality()

    rects = ss.process()
    boxes = []
    for (x,y,w,h) in rects:

        boxes.append([x,y,w,h])
        pass

    return boxes
    pass
```

Рисунок 3.14 – Блок пошуку регіонів інтересу на знімку

Для блоку пошуку регіонів інтересу було використано функціонал бібліотеки OpenCV, після знаходження регіонів інтересу які задаються в вигляді рамки з координатами її кутів, регіони аналізуються створеною раніше моделлю, якщо регіон є кораблем – рамка зберігається, якщо ні – видаляється. Оскільки на кожен об’єкт часто припадає більше однієї рамки з дещо зміненими координатами – було вирішено не обирати одну з них, а залишати всі, таким чином результат обробки зображення стає більш наглядним.

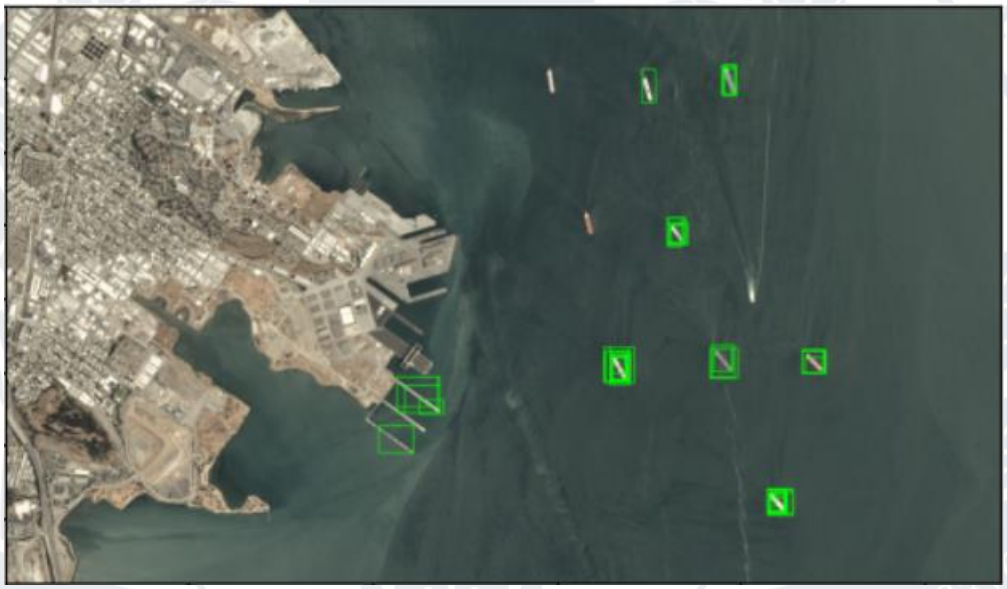


Рисунок 3.15 – Результат обробки зображення

На рисунку бачимо результат обробки зображення, бачимо що більшість кораблів успішно виділені, не виділеними залишаються 2 кораблі з 9, крім того є моторний човен, однак його модель і не мала виділяти. Одначе, окрім кораблів були виділені елементи інфраструктури порту, розглянемо частину оригіналу зображення, для кращого розуміння помилки.

Рисунок 3.16 – Помилково визначена інфраструктура

З рисунку видно, що інфраструктура досить схожа по формі на кораблі, ймовірно це і стало причиною помилкового визначення. Ймовірною причиною даної помилки і помилки з не визначення існуючих двох кораблів є малий розмір тренувального датасету, який складався з 4000 зображень з прикладами кораблів і не кораблів. Найкращим рішенням цієї проблеми буде збільшення тренувального датасету. При неможливості знаходження необхідних даних проміжним рішенням є взяття помилкових областей і донавчання моделі з врахуванням нових прикладів «не кораблів».

3.5 Програмний додаток

Було розроблено додаток з графічним інтерфейсом задля збільшення зручності роботи з нейромережею серед сторонніх дослідників, додаток має необхідний для аналізу зображень функціонал і побудований за допомогою крос-платформового інструментарія розробки Qt.

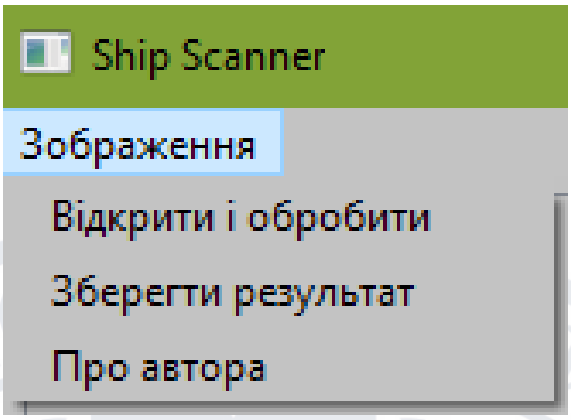


Рисунок 3.17 – Випадаюче меню

В випадаючому меню присутня сторінка інформації про автора додатку, можливість відкривати і відразу ж обробляти зображення, функціонал збереження результату.

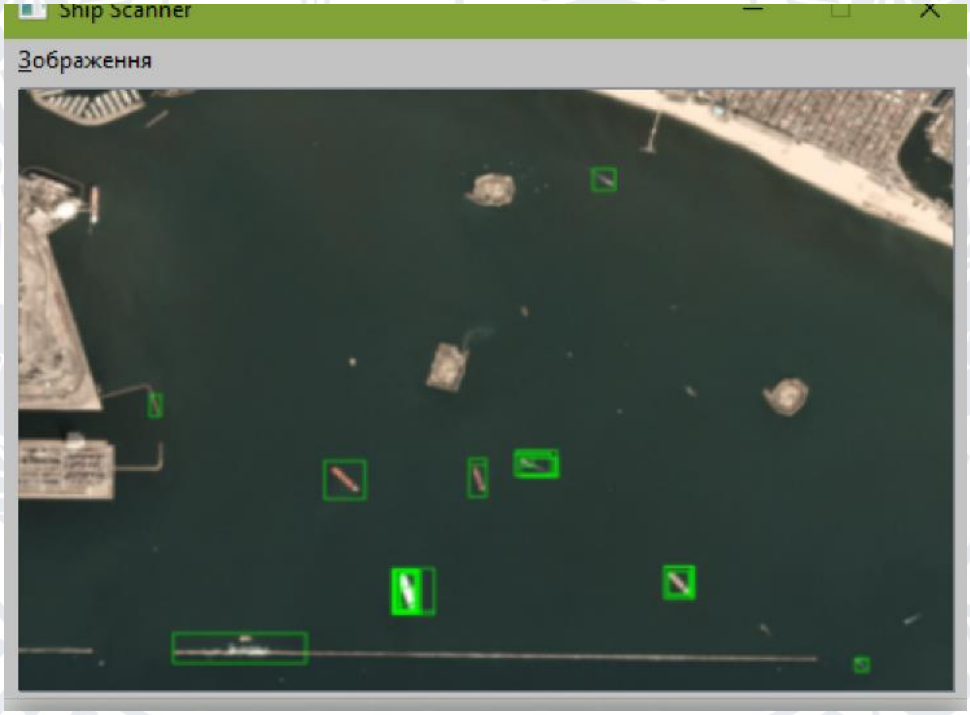


Рисунок 3.18 – Головне вікно

В головному вікні є можливість переглядати результат обробки зображення до його зберігання, зображення з’являється тільки після його відкриття за допомогою меню, до того – поле пусте.

Висновок до розділу 3

У даному розділі були розглянуті етапи розробки нейронної мережі, а саме роботу з даними, розробку основних функцій і моделі. Описано і проаналізовано результати навчання моделі, наведені приклади передбачень. Була приділена увага недолікам і шляхам їх усунення, інтерфейсу додатка.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Нейронні мережі - шлях до глибинного навчання. URL: codeguida.com/post/739
2. Градієнтний спуск. URL: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html
3. Gradient Descent in Machine Learning. URL: https://www.javatpoint.com/gradient-descent-in-machine-learning
4. Guido S. Introduction to Machine Learning with Python / S. Guido, A. Müller. Sebastopol, United States: O'Reilly Media, Inc, USA, 2016. – 392 с.
5. Навчання без вчителя. URL: https://mykhno.at.ua/news/obuchenie_bez_uchitelja/2019-01-30-9
6. Навчання з вчителем, без вчителя з підкріпленням. URL: https://neurohive.io/en/osnovy-data-science/obuchenie-s-%20uchitelem-bez-uchitelja-s-podkrepleniem/
7. Навчання з підкріпленням у машинному навчанні. URL: https://evergreens.com.ua/ua/articles/reinforcement-learning.html
8. NIST/SEMATECH e-Handbook of Statistical Methods. URL: http://www.itl.nist.gov/div898/handbook/
9. Mean Absolute Error. URL: https://www.sciencedirect.com/topics/engineering/mean-absolute-error
10. Mean Absolute Percentage Error. URL: https://docs.oracle.com/en/cloud/saas/planning-budgeting-cloud/pfusu/insights_metrics_MAPE.html
11. Mean Squared Error : Overview, Examples, Concepts and More. URL: https://www.simplilearn.com/tutorials/statistics-tutorial/mean-squared-error
12. Дослідження ефекту перенавчання нейронних мереж. URL: http://ir.kneu.edu.ua/handle/2010/20355

13. Empirical Risk Minimization with Relative Entropy Regularization: Optimality and Sensitivity Analysis. URL: https://arxiv.org/abs/2202.04385

14. Підготовка даних і необроблені дані в машинному навчанні. URL: https://www.kdnuggets.com/2022/07/data-preparation-raw-data-machine-learning.html

15. How To Prepare Your Data for Your Machine Learning Model. URL: https://towardsdatascience.com/how-to-prepare-your-data-for-your-machine-learning-model-b4c9fd4e7ea

16. ML | Label Encoding of datasets in Python. URL: https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

17. One-Hot Encoding in Scikit-Learn with OneHotEncoder. URL: https://datagy.io/sklearn-one-hot-encode/

18. Категоріальні ознаки. URL: https://habr.com/ru/post/666234/

19. Зниження розмірності. URL: https://www.wiki.uk-ua.nina.az/%D0%97%D0%BD%D0%B8%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F_%D1%80%D0%BE%D0%B7%D0%BC%D1%96%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%96.html

20. Scikit-learn-contrib, an umbrella for scikit-learn related projects. URL: https://fa.bianp.net/blog/2016/scikit-learn-contrib-an-umbrella-for-scikit-learn-related-projects/

21. The mostly complete chart of Neural Networks, explained. URL: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

22. Your First Machine Learning Project in Python Step-By-Step. URL: https://machinelearningmastery.com/machine-learning-in-python-step-by-step/

23. Python is powerful... and fast. URL: https://www.python.org/about/

24. What is Python? Executive Summary. URL: https://www.python.org/doc/essays/blurb/

25. 15 Python Libraries for Data Science and Machine Learning. URL: https://www.projectpro.io/article/top-5-libraries-for-data-science-in-python/196#mcetoc_1fu5v5t8dl

26. NumPy Introduction. URL: https://www.w3schools.com/python/numpy/numpy_intro.asp

27. Scipy 1.9.3. URL: https://pypi.org/project/scipy/

28. Scikit Learn Tutorial. URL: https://www.tutorialspoint.com/scikit_learn/index.htm

29. TensorFlow. URL: https://opensource.google/projects/tensorflow

30. Learning OpenCV. URL: https://www.oreilly.com/library/view/learning-opencv/9780596516130/

31. The Python SQL Toolkit and Object Relational Mapper. URL: https://www.sqlalchemy.org/

32. Beautiful Soup: Build a Web Scraper With Python. URL: https://realpython.com/beautiful-soup-web-scraper-python/

33. About Us. Project Jupyter's origins and governance. URL: https://jupyter.org/about

34. 7 Reasons Why You Should Use Jupyterlab for Data Science. URL: https://towardsdatascience.com/7-reasons-why-you-should-use-jupyterlab-for-data-science-7c2a3db8755a

35. Top 8 magic commands in jupyter notebook. URL: https://www.aboutdatablog.com/post/top-8-magic-commands-in-jupyter-notebook

36. WHAT IS KAGGLE? URL: https://www.datacamp.com/blog/what-is-kaggle

37. A Gentle Introduction to Object Recognition With Deep Learning. URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/

38. Image Classification. URL: https://huggingface.co/tasks/image-classification

39. Understanding Object Localization with Deep Learning. URL: https://www.einfochips.com/blog/understanding-object-localization-with-deep-learning/

40. What Is Object Detection? URL: https://www.mathworks.com/discovery/object-detection.html

41. Object Localization using PyTorch, Part 1. URL: https://blog.paperspace.com/object-localization-using-pytorch-1/

42. Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN. URL: https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html

43. Confusion Matrix. URL: https://www.sciencedirect.com/topics/engineering/confusion-matrix

ДОДАТОК А

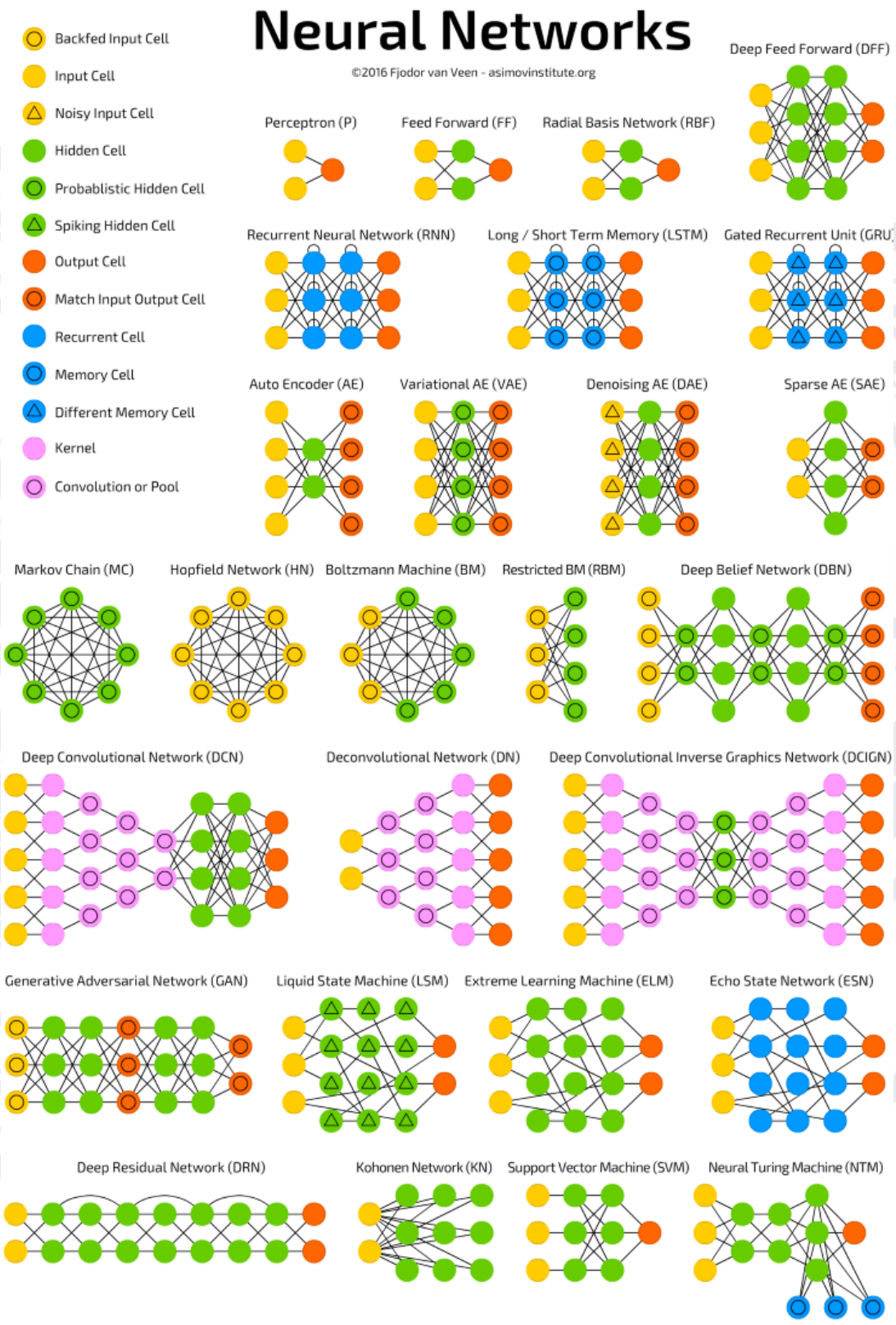


Рисунок 1 – Неповний список всіх топологій нейромереж

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Кучер Микита Олександрович
Прізвище, ім’я, по батькові

Факультет інформаційних і прикладних технологій
Факультет

122 Комп’ютерні науки
Шифр і назва спеціальності

Комп'ютерні технології обробки даних(Data Science)
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему: «Розробка і дослідження засобів аналізу візуальних даних з використанням нейронних мереж» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувались іншими особами, а також дані та інформація неотримувались у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна (магістерська) робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

дата	підпис