МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____Ігор ЖУКОВ
(підпис)

« ____ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: «Система обробки візуальної інформації для ідентифікації та аналізу образів»

_____

Виконавець:_____
(студент, група, прізвище, ім'я, по батькові)

Керівник:_____
(науковий ступінь, вчене звання , прізвище, ім'я, по батькові)

Нормоконтролер _____ _____
(підпис) (ПІБ)

Засвідчую , що у магістерській роботі немає запозичень праць інших авторів без відповідних посилань
Студент _____ _____
(підпис) (ПІБ)

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних систем та мереж

_____Ігор ЖУКОВ

« ____ » _____ 2023 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Шароватової Діани Вікторівни
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1.Тема роботи: Система обробки візуальної інформації для ідентифікації та аналізу образів
затверджена наказом ректора від « 29 » серпня 2023 року № 1521/ст.

2.Термін виконання роботи: з 02.10.2023р. по 31.12.2023р.

3.Вихідні дані до роботи: використання *Python* , *Tensorflow* , *OpenCV* , *Keras* для реалізації системи ідентифікації та аналізу образів

4.Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) Аналіз науково-інформаційних джерел стосовно теми дослідження
2) Штучні нейронні мережі, як інструмент для вирішення задачі
3) Програмне забезпечення розв'язання задачі ідентифікації образів
4) Основні етапи реалізації програмного забезпечення системи розпізнавання образів

5.Перелік обов'язкового графічного матеріалу:
1)Презентація *PowerPoint*
2)Таблиці
3)Графіки
4)Рисунки

6. Календарний план - графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1	Ознайомитися з поставленим завданням. Визначити тему та мету кваліфікаційної роботи.	02.10.23-04.10.23	
2	Провести аналіз технічної документації та спеціальної літератури. Визначити ключові аспекти області комп'ютерного зору та розпізнавання образів.	5.10.23-12.10.23	
3	Проаналізувати галузі інформаційних технологій. Визначити перспективи та тенденції розвитку. Написати перший розділ.	13.10.23-20.10.23	
4	Проаналізувати основні компоненти архітектури нейронних мереж. Обрати методи та технології реалізації практичної частини кваліфікаційної роботи. Написати другий розділ.	21.10.23-4.11.23	
5	Реалізувати практичну частину роботи, провести тестування.	5.11.23-25.11.23	
6	Написати третій та четвертий розділи. Підвести підсумки.	26.11.23-2.12.23	
7	Оформити пояснювальну записку	3.12.23-14.12.23	
8	Оформити відповідний графічний матеріал	15.12.23-21.12.23	
9	Захистити роботу	22.12.23-31.12.23	

7. Дата отримання завдання «02» жовтня 2023 р.

Керівник кваліфікаційної роботи Ігор ТЕЛЕШКО
(підпис)

Завдання прийняв до виконання Діана ШАРОВАТОВА
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Система обробки візуальної інформації для ідентифікації та аналізу образів»: 90 с., 28 рис., 18 літературних джерел.

PYTHON, TENSORFLOW, OPENCV, NEURAL NETWORKS, OBJECT DETECTION, KERAS

Об’єкт дослідження – система розпізнавання зображень**.**

Предмет дослідження – методи та підходи для ідентифікації візуальної інформації, архітектура нейронних мереж.

Мета кваліфікаційної роботи – створення системи обробки візуальної інформації для ідентифікації та аналізу образів.

Результатом виконання кваліфікаційної роботи є створена система ідентифікації образів, що має можливість розпізнавати образи із зображення, відео, в тому числі, в режимі реального часу. У роботі використано власноруч створену та навчену мережу глибинного навчання, яка спроможна ідентифікувати захисні каски на працівниках у сфері будівництва що може забезпечити контроль дотримання безпекових норм у цій галузі.

Для подальшого вдосконалення системи рекомендується розширити розмір датасету для навчання та використовувати техніки аугментації даних для поліпшення точності результатів під час тренування нейромережі. Також, можливе вдосконалення функції втрат *IoU* та оптимізація гіперпараметрів для досягнення оптимальних результатів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

AI	–	*Artificial Intelligence* (штучний інтелект)
ШІ	–	Штучний Інтелект
Хаара	–	*Haar-like Features* (функції цифрового зображення)
SIFT	–	*Scale-Invariant Feature Transform* (алгоритм комп'ютерного зору та обробки зображень)
HOG	–	*Histogram of Oriented Gradients* (гістограма орієнтованих градієнтів)
YOLO	–	*You Only Look Once* (алгоритм детекції об'єктів)
RCNN	–	*Region-based Convolutional Neural Network* (мережа зі зваженим згортковим шаром)
SSD	–	*Single Shot Detector* (одноетапний детектор)
OpenCV	–	*Open Source Computer Vision Library* (бібліотека комп'ютерного зору з відкритим вихідним кодом)
FaceNet	–	*Face Recognition System* (система розпізнавання обличчя)
ШНМ	–	Штучні нейронні мережі
CV	–	*Computer Vision* (комп'ютерне зорове сприйняття)
FNN	–	*Feedforward Neural Networks* (нейронні мережі прямого зв'язку)
RNN	–	*Recurrent Neural Networks* (рекурентні нейронні мережі)
CNN	–	*Convolutional Neural Networks* (згорткові нейронні мережі)
GAN	–	*Generative Adversarial Networks* (генеративні змагальні мережі)
LSTM	–	*Long Short-Term Memory* (мережі довготривалої короткочасної пам'яті)

ResNet	–	*Residual Networks* (залишкові мережі)
ESN	–	*Echo State Networks* (мережі Echo State)
DNN	–	*Deconvolutional Neural Network* (деконволюційна нейронна мережа)
VAE	–	*Variational Autoencoder* (варіаційний автокодер)
R-CNN	–	*Region-based Convolutional Neural Network* (регіональна згорткова нейронна мережа)
Fast R-CNN	–	*Fast Region-based Convolutional Neural Network* (швидка регіональна згорткова нейронна мережа)
Faster R-CNN	–	*Faster Region-based Convolutional Neural Network* (швидша регіональна згорткова нейронна мережа)
Mask R-CNN	–	*Mask Region-based Convolutional Neural Network* (регіональна мережа зі зваженим згортковим шаром для маскування)
FPN	–	*Feature Pyramid Network* (мережа піраміди ознак)
IoU	–	*Intersection over Union (IOU)*

ВСТУП

Технології комп'ютерного зору процвітають у сучасному техногенному світі. Перш за все, люди створюють постійно зростаючі обсяги візуальних даних. Наприклад, лише в *Google Photos* щодня завантажується 28 мільярдів зображень і відео. Величезні публічні набори даних також доступні. По-друге, обчислювальні ресурси стали дешевшими та вдосконаленими, що дозволило створювати потужні апаратні рішення для *CV*. Нарешті, спеціалісти з обробки даних постійно наполегливо працюють із системою комп'ютерного зору, яка користується великим попитом, розробляючи ефективніші алгоритми комп'ютерного зору. Прогрес у технології комп'ютерного зору змінив різні галузі.

Автомобільна сфрера. Найпомітніші переваги, які комп'ютерне бачення приносить на автомобільний ринок, походять від виявлення та класифікації об'єктів. Дані із зовнішніх камер у поєднанні з тими, які спостерігають за водієм, можуть запобігти аваріям і допомогти досягти нових стандартів безпеки на дорозі.

Tesla, один із найпередовіших виробників автомобілів сучасності, чітко продемонструвала, як передові технології можуть революціонізувати галузь. Поява безпілотних автомобілів здається неминучою. Щоб автопілот працював, транспортним засобам потрібне обладнання з підтримкою комп'ютерного зору, тоді як виробникам абсолютно необхідні ці великі набори даних для аналізу.

Сільське господарство. ООН прогнозує зростання попиту на сільськогосподарську продукцію на 70% до 2050 року. Галузь має активізувати свою технологічну гру з глобальним потеплінням, і *CV* – це вихід варіант.

Випадки використання комп'ютерного зору в сільському господарстві включають:

– контроль якості продукції;
– моніторинг тваринництва;
– краще оцінювання та сортування;
– раннє виявлення аномалій;
– прогнозування погодних умов;
– калібрування моделі культури.

Глибоке навчання може допомогти створити набагато кращі моделі для сільськогосподарського бізнесу, що забезпечить стале майбутнє розумного сільського господарства.

Банківська справа. Коли мова йде про фінансові послуги, комп'ютерний зір може багато чого запропонувати, особливо з точки зору сприяння цифровій трансформації. Банки та інші фінансові установи можуть очікувати покращень у таких сферах оцифрування документів, для швидшої обробки та вилучення даних, розпізнавання обличчя , оцінка збитків , для страхових компаній, як для автомобілів, так і для нерухомості. Технологія комп'ютерного зору може створювати багато цінних даних, які банки можуть використовувати для оптимізації своїх операцій і покращення взаємодії з клієнтами.

Охорона здоров'я. Медичні працівники часто опиняються в ситуаціях, коли кожна хвилина на рахунку, тому комп'ютерні системи, які можуть запропонувати швидкість і надійність, високо цінуються. Технологія комп'ютерного зору дає лікарям правильні інструменти для прийняття більш обґрунтованих рішень.

Наприклад, *CV* допомагає виявляти неврологічні захворювання шляхом аналізу КТ. Він також використовується для аналізу рентгенівських зображень і ультразвукового сканування. Використовуючи цю технологію, онкологи можуть з більшою точністю діагностувати рак, а також розрізняти злоякісні та доброякісні пухлини.

Спорт. Інструменти відстеження в режимі реального часу, надані *CV*, надзвичайно корисні для підрахунку результатів спортсменів. Замість того, щоб покладатися виключно на зір і пам'ять суддів, ми тепер маємо об'єктивні засоби моніторингу кожного параметра, який нам може знадобитися. Збір показників продуктивності також є неоціненним для навчання. Не можна недооцінювати досвід тренерів, але поповнення його даними з перших рук може призвести до ще кращих досягнень.

Виробництво. Завдяки ефективному моніторингу та потужній автоматизації системи комп'ютерного зору можуть оптимізувати багато аспектів виробництва.

Наприклад, рішення для контролю якості *CV* від *Quantum* допомогло клієнту скоротити виробничі втрати, досягнувши точності 99,99%. Для порівняння, попереднє рішення могло забезпечити лише 10% цієї продуктивності.

Інші способи використання *CV* у виробництві включають роботи з баченням, контроль пакування та маркування. Аналізуючи дані, зібрані під час цих процесів, виробники можуть виявити проблемні точки та прийняти рішення, які вплинуть на їхні бізнес-операції у більшому масштабі.

Логістика. У поєднанні з датчиками *IoT CV* дозволяє створювати повністю автоматизовані логістичні рішення. Ось деякі з переваг використання систем комп’ютерного зору для ланцюгів поставок: точне визначення етикеток, штрих-кодів і розмірів упаковки ; ідентифікація пошкоджених товарів; цілодобовий моніторинг, виявлення відсутніх елементів; розширене сортування; оптимізація простору в транспорті.

Аварійні служби. *CV* може допомогти екстреним службам, надаючи аналіз знімків із супутника в реальному часі. Це може бути критично важливим для планування реагування на лісові пожежі, повені, землетруси та інші стихійні лиха. Завдяки швидкому та точному аналізу ШІ керувати гуманітарними аспектами катастроф також стає легше.

Мета і завдання

Враховуючи стрімке зростання обсягів будівельних проєктів, важливо надавати пріоритет розвитку інноваційних методів для автоматизації контролю за безпекою робітників на будівництві. Наприклад, у США будівельна промисловість стикається з найвищим рівнем смертельних випадків порівняно з усіма іншими галузями промисловості.

Дотримання норм безпеки на будівельних майданчиках стає визначально важливим завданням, оскільки це напрямок, де трапляється значна частка нещасних випадків та травм. Забезпечення безпеки робітників під час будівництва не тільки рятує життя і здоров'я працівників, але також сприяє зниженню ризиків та негативного впливу на продуктивність проєкту.

Спрямованість на дослідження і розробку інноваційних методів, зокрема автоматичного контролю за дотриманням норм безпеки, відображає сучасні тенденції в галузі будівництва. Обрана тема для написання кваліфікаційної роботи про ідентифікацію касок на будівельних майданчиках з використанням відео та зображень не тільки відповідає вимогам ринку, але і вносить вагомий внесок у підвищення безпеки праці в цій галузі.

Розробка автоматизованої системи, яка може ефективно визначати присутність або відсутність касок на робітниках за допомогою відеоаналізу та обробки зображень, може значно покращити контроль за дотриманням безпекових норм. Це дозволить вчасно виявляти порушення та уникати потенційних небезпек.

Такий підхід визначається не тільки технічними можливостями, але й соціальною відповідальністю, спрямованою на збереження життя та покращення умов праці будівельних робітників.

Зосередження уваги на ідентифікації та аналізі образів відкриває нові перспективи для застосування технологій машинного навчання та комп'ютерного зору.

Мета цієї кваліфікаційної роботи полягає в розробці власної системи обробки візуальної інформації для ідентифікації та аналізу образів. В рамках даного проєкту образами, які піддаються ідентифікації та аналізу, є будівельні захисні каски.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Ознайомитися з основними теоретичними засадами стосовно теми дослідження.
2. Розглянути застосування нейромереж у завданнях розпізнавання образів.
3. Розглянути основні методи та підходи визначення образів.
4. Розглянути основні типи нейромереж.
5. Розглянути ключові поняття та види архітектур, обрати архітектуру для вирішення задачі розпізнавання.
6. Проаналізувати та порівняти різноманітні технології та інструменти для програмної реалізації системи ідентифікації образів.
7. Підготувати власний набір даних для навчання моделі.

8. Розробити та підготувати архітектуру нейронної мережі для ефективної ідентифікації образів.

9. Здійснити навчання моделі з використанням набору даних, враховуючи різні архітектурні варіанти та гіперпараметри в межах методів машинного навчання.

10. Забезпечити можливість ідентифікації образів на зображеннях , відео, провести тестування.

Об’єкт дослідження – система ідентифікації образів**.**

Предмет дослідження – методи та підходи для розпізнавання та аналізу образів засновані на використанні нейронних мереж.

Методи дослідження. Для досягнення мети використовуються методи машинного навчання та комп'ютерного зору, зокрема, використання нейронної мережі, розробленої на мові програмування *Python* з використанням бібліотек *TensorFlow* та *Keras*. Методика вимірювання ефективності системи базується на власно розробленому показнику *IoU (Intersection over Union*), який дозволяє визначити ступінь перекриття областей ідентифікованих та реальних об'єктів.

Практичне значення в результаті дослідження та розробки цієї системи досягнуті важливі результати , спрямовані на ефективну ідентифікацію захисних касок на працівниках у сфері будівництва що може забезпечити контроль дотримання безпекових норм у цій галузі. Шлях удосконалення та розширення функціональності залишається відкритим для подальших досліджень та оптимізацій.

РОЗДІЛ 1. АНАЛІЗ НАУКОВО-ІНФОРМАЦІЙНИХ ДЖЕРЕЛ СТОСОВНО ТЕМИ ДОСЛІДЖЕННЯ

1.1 Опис предметної області

Комп'ютерне бачення використовує штучний інтелект (ШІ) , щоб дозволити комп'ютерам отримувати важливі дані з візуальних вхідних даних, таких як фотографії та відео.

Дані, отримані за допомогою комп'ютерного зору, потім використовуються для виконання автоматизованих дій. Подібно до того, як ШІ дає комп'ютерам здатність «мислити» , комп'ютерний зір дозволяє їм «бачити» (рис.1.1).

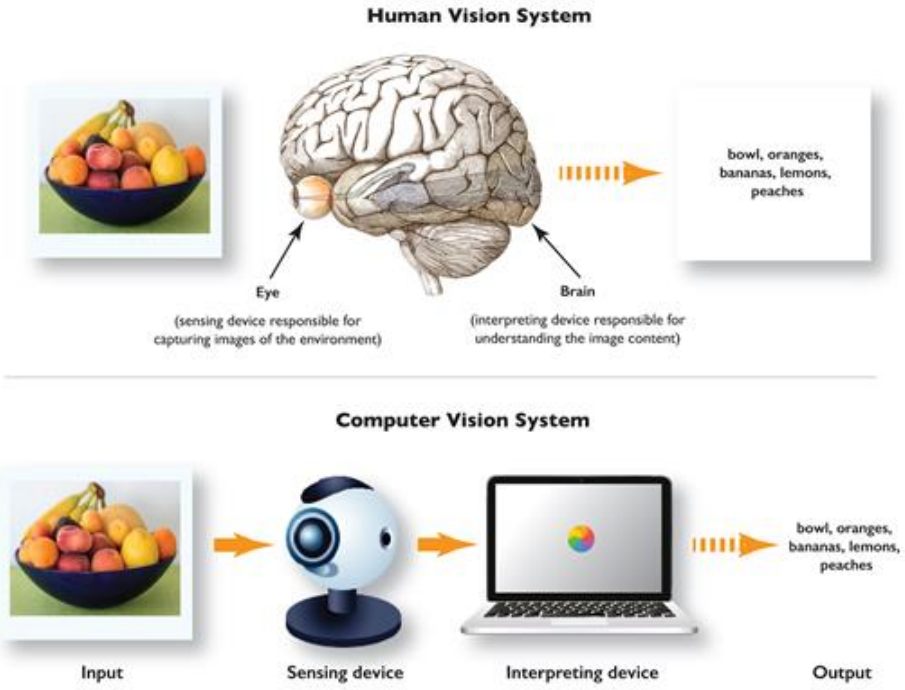


Рис.1.1 Людський зір проти комп'ютерного зору

Як люди, ми зазвичай проводимо своє життя, спостерігаючи за навколишнім середовищем за допомогою зорових нервів, сітківки та зорової кори. Ми отримуємо контекст, щоб розрізняти об'єкти, вимірювати їхню відстань від нас та інших об'єктів, розраховувати швидкість їх руху та виявляти помилки. Однак, на відміну від людини, комп'ютер не втомлюється. Ви можете навчити машини з комп'ютерним зором аналізувати тисячі виробничих активів або продуктів за лічені хвилини.

Це дозволяє виробничим підприємствам автоматизувати виявлення дефектів, непомітних для людського ока. Щоб комп’ютерний зір був справді ефективним, потрібна велика база даних. Це пояснюється тим, що ці рішення аналізують інформацію неодноразово, доки не отримають усі можливі відомості, необхідні для виконання поставленого завдання.

Наприклад, комп’ютер, навчений розпізнавати здорові посіви, мав би «бачити» тисячі візуальних еталонних даних про посіви, сільськогосподарські угіддя, тварин та інші пов’язані об’єкти. Лише тоді він зможе ефективно розпізнавати різні типи здорових культур, диференціювати їх від нездорових культур, оцінювати якість сільськогосподарських угідь, виявляти шкідників та інших тварин серед культур тощо.[5]

1.2 Найпоширеніші завдання комп’ютерного зору

Завдання в галузі комп'ютерного зору спрямовані на надання комп'ютерам здатності розуміти цифрові зображення та візуальні дані з реального світу. Ця область включає в себе вилучення, обробку та аналіз інформації з вхідних даних для прийняття рішень.

Еволюція машинного зору призвела до широкомасштабної формалізації складних проблем, перетворюючи їх на популярні та добре визначені задачі. Розподіл цих задач на групи з чіткою номенклатурою сприяв усвідомленню проблем дослідниками по всьому світу та ефективному розв'язанню цих завдань.

Серед найпопулярніших завдань комп'ютерного зору , які регулярно зустрічаються в галузі штучного інтелекту, можна визначити :

– класифікація;

– виявлення та локалізація;

– сегментація;

–розпізнавання осіб і обличчя;

– виявлення країв;

–відновлення зображення;

–відповідність ознак [6].

1.2.1 Класифікація зображень

Класифікація зображень є однією з найбільш досліджуваних тем з моменту випуску набору даних *ImageNet* у 2010 році.

Будучи найпопулярнішим завданням комп'ютерного зору, яке беруться як для початківців, так і для експертів, класифікація зображень як формулювання проблеми досить проста. Дана група зображень, завдання полягає в тому, щоб класифікувати їх у набір попередньо визначених класів, використовуючи лише набір зразків зображень, які вже були класифіковані.

На відміну від складних тем, таких як виявлення об'єктів і сегментація зображення, які повинні локалізувати (або надати позиції) ознаки, які вони виявляють, класифікація зображень має справу з обробкою всього зображення в цілому та призначенням йому певної мітки. (рис.1.2.1)

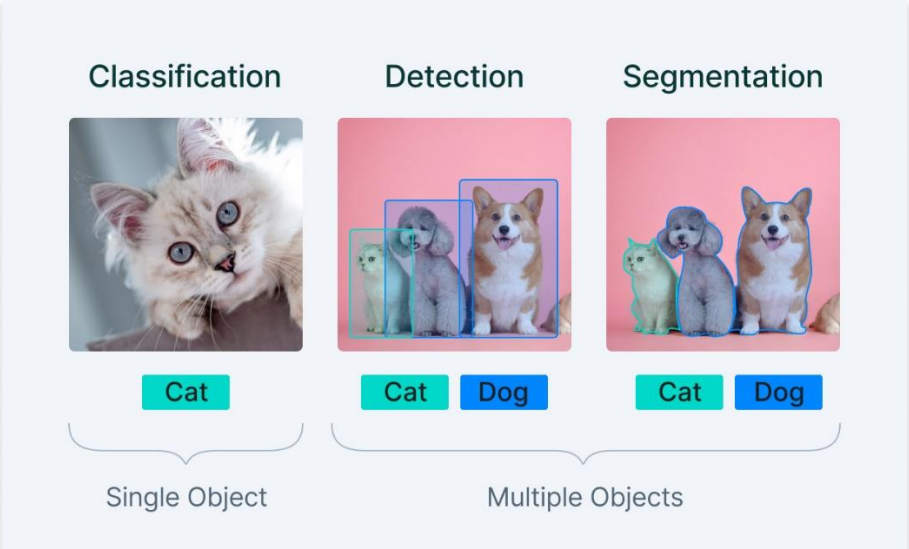


Рис.1.2.1 Класифікація , виявлення та локалізація , сегментація

Комп'ютер аналізує зображення у вигляді пікселів. Він робить це, розглядаючи зображення як масив матриць, розмір матриці яких залежить від роздільної здатності зображення. Простіше кажучи, класифікація зображень, з точки зору комп'ютера - це аналіз цих статистичних даних за допомогою алгоритмів [7]. У цифровій обробці зображень класифікація зображень здійснюється шляхом автоматичного групування пікселів у визначені категорії, так звані класи. Алгоритми поділяють зображення на низку його найвидатніших ознак, зменшуючи навантаження на кінцевий класифікатор.

Ці характеристики дають класифікатору уявлення про те, що представляє зображення та до якого класу його можна віднести. Процес виділення характеристик є найважливішим кроком у класифікації зображення, оскільки решта кроків залежить від нього.

1.2.2 Виявлення та локалізація об'єктів

Виявлення та локалізація об’єктів відбувається за допомогою обмежувальних рамок. Виявлення об’єктів шукає на зображенні чи відео деталі класу та ідентифікує їх щоразу, коли вони з’являються. Ці класи можуть бути автомобілями, тваринами, людьми або будь-чим іншим, відповідно до того набору на якому була навчена модель. (див.рис.1.2.1)

Раніше методи виявлення об’єктів використовували функції *Хаара, SIFT і HOG* для виявлення особливостей на зображенні та їх класифікації на основі класичних підходів машинного навчання.[8] Цей процес, окрім того, що він трудомісткий і значною мірою неточний, має серйозні обмеження щодо кількості об’єктів, які можна виявити.

Таким чином, моделі глибокого навчання, такі як *YOLO, RCNN, SSD*, які використовують мільйони параметрів, щоб подолати ці обмеження, широко використовуються для цього завдання. Виявлення об’єктів часто супроводжується розпізнаванням об’єктів, також відомим як класифікація об’єктів.

1.2.3 Сегментація зображення

Сегментація зображення – це поділ зображення на підчастини або підоб’єкти, щоб продемонструвати, що машина може розрізнити об’єкт на тлі та/або інший об’єкт на тому самому зображенні. «Сегмент» зображення представляє певний клас об’єктів, які нейронна мережа ідентифікувала на зображенні, представлених піксельною маскою, яку можна використовувати для його вилучення.[9]

Ця популярна область комп’ютерного бачення була широко вивчена як з використанням традиційних алгоритмів обробки зображень, таких як вододілові алгоритми, сегментація на основі кластеризації, так і з використанням популярних сучасних архітектур глибокого навчання, таких як *PSPNet, FPN, U-Net, SegNet*, тощо.

Існує кілька підходів до сегментації зображення.

Порогове значення: це простий метод, за допомогою якого пікселі класифікуються на різні сегменти на основі значень їх інтенсивності. Якщо значення інтенсивності пікселя вище певного порогу, він належить одному сегменту; інакше воно належить іншому. Це ефективно для зображень із чітко визначеною різницею інтенсивності між об'єктами та фоном.

Методи на основі країв: ці методи визначають межі між різними сегментами шляхом виявлення раптових змін інтенсивності або кольору. Такі методи, як детектор країв Canny, використовуються для ідентифікації країв, які потім можна з'єднати, щоб утворити межі об'єктів.

Методи на основі регіонів: ці методи групують пікселі в сегменти на основі критеріїв подібності, таких як колір, текстура чи інтенсивність. Для групування подібних пікселів використовуються такі методи, як кластеризація K-середніх або кластеризація середнього зсуву.

Методи на основі контурів: Ці методи зосереджені на визначенні контурів або контурів об'єктів на зображенні. Такі методи, як активні контури (змійки) або алгоритми трасування контурів, допомагають виявляти межі об'єктів.

Семантична сегментація: у цьому розширеному підході кожному пікселю присвоюється мітка класу, що вказує на конкретний об'єкт або структуру, до якої він належить. Це вимагає навчання моделей глибокого навчання, як-от згорткових нейронних мереж (*CNN)* на позначених наборах даних, щоб вивчати складні особливості різних об'єктів.

Сегментація екземплярів: Цей підхід розрізняє не лише класи об'єктів, але й різні екземпляри одного класу. Він надає унікальну мітку кожному окремому екземпляру об'єкта на зображенні.

Математичні концепції сегментації зображення

Сегментація зображення в комп'ютерному зорі передбачає поділ зображення на окремі області або сегменти для спрощення його аналізу.

Кілька математичних концепцій підтримують цей процес:

Порогове значення інтенсивності: Ця проста методика передбачає вибір порогового значення та класифікацію пікселів на основі того, чи є значення їхньої інтенсивності вищими чи нижчими за порогове значення.

Кластеризація: такі алгоритми, як K-Means або Mean-Shift, можуть групувати подібні пікселі в кластери, які відповідають різним сегментам зображення.

Теорія графів: методи на основі графів розглядають зображення як графік із пікселями як вузлами, а зв'язки між пікселями як ребрами. Розрізи графіків і мінімальні остовні дерева можна використовувати для розділення зображення на сегменти.

Зростання та розділення регіонів: ці методи починаються з початкового пікселя та розширюють або розділяють регіони на основі попередньо визначених критеріїв, таких як схожість кольорів, текстура чи інтенсивність.

Перетворення вододілу: Натхненний геологічними вододілами, цей метод розглядає зображення як топографічну карту та сегментує його на основі критеріїв затоплення.

Випадкові поля Маркова *(MRF): MRF* моделюють зв'язки між сусідніми пікселями та використовують імовірності для призначення міток пікселям, що призводить до сегментованих областей.

Мінімізація енергії на основі графіків: Цей підхід формулює сегментацію зображення як проблему мінімізації енергії. Пікселям призначаються мітки, щоб мінімізувати енергетичну функцію, яка врівноважує термін даних (схожість пікселів) і елемент гладкості (заохочення подібних міток для сусідніх пікселів).

Моделі активного контуру (змійки): ці моделі ітеративно деформують початковий контур, щоб відповідати межам об'єкта, мінімізуючи енергетичний функціонал на основі характеристик зображення та гладкості.

Методи набору рівнів: Техніки набору рівнів представляють межі об'єктів як набори нульового рівня високовимірних функцій, змінюючи набори рівнів з часом для досягнення сегментації.

1.2.4 Розпізнавання осіб та обличь

Розпізнавання обличчя – це частина виявлення об’єктів, де основним об’єктом, що виявляється, є обличчя людини. Хоча розпізнавання обличчя подібне до завдання виявлення об’єкта, де функції виявляються та локалізуються, розпізнавання обличчя виконує не лише виявлення, але й розпізнавання виявленого обличчя.(рис.1.2.2)

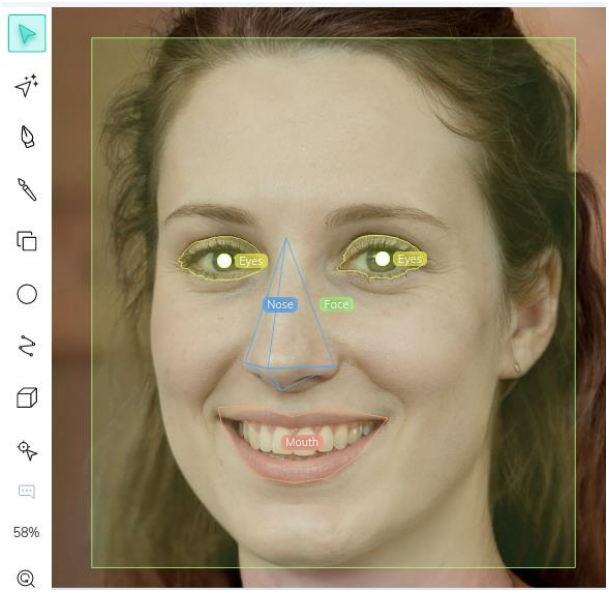


Рис.1.2.2 Розпізнавання обличчя

Системи розпізнавання обличчя шукають загальні риси та орієнтири, такі як очі, губи чи ніс, і класифікують обличчя за цими ознаками та розташуванням орієнтирів.

Традиційні методи обробки зображень для розпізнавання обличчя включають каскади Хаара, доступ до яких можна легко отримати через бібліотеку *OpenCV*. Деякі більш надійні методи, що використовують алгоритми на основі глибокого навчання, можна знайти в документах, таких як *FaceNet*.

1.2.4 Виявлення країв

Виявлення країв – це завдання виявлення меж об’єктів. Це виконується алгоритмічно за допомогою математичних методів, які допомагають виявляти різкі зміни або розриви яскравості зображення.

Виявлення країв, яке часто використовується як етап попередньої обробки даних для багатьох завдань, в основному виконується за допомогою традиційних алгоритмів обробки зображень, таких як виявлення *Canny Edge* і згорток зі спеціально розробленими фільтрами виявлення країв.

Крім того, краї на зображенні дають нам найважливішу інформацію про вміст зображення, в результаті чого всі методи глибокого навчання виконують внутрішнє виявлення країв для захоплення глобальних низькорівневих функцій за допомогою ядра, яке можна вивчати.

1.2.5 Відновлення зображення

Відновлення зображення означає реставрацію або реконструкцію вицвілих і старих друкованих копій зображень, які були зроблені та збережені неналежним чином, що призвело до втрати якості зображення. (рис.1.2.2)

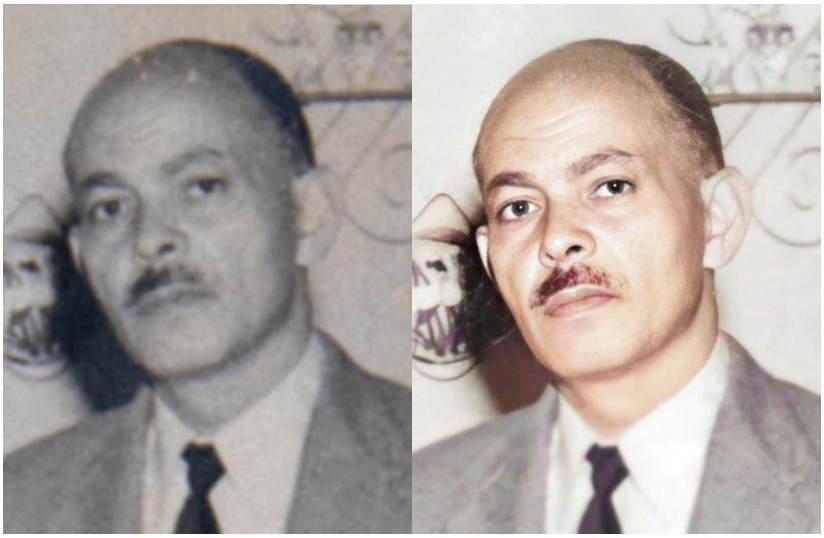


Рис.1.2.2 Відновлення зображення

Типові процеси відновлення зображень передбачають зменшення додаткового шуму за допомогою математичних інструментів, тоді як інколи реконструкція потребує серйозних змін, що веде до подальшого аналізу та використання малювання зображення. У Image inpainting пошкоджені частини зображення заповнюються за допомогою генеративних моделей, які роблять оцінку того, що зображення намагається передати. Часто процес відновлення супроводжується процесом розфарбовування, який забарвлює об’єкт зображення (якщо чорно-білий) у найбільш реалістичний спосіб.

1.2.6 Відповідність ознак

Особливості комп'ютерного зору визначаються як регіони на зображенні, які найефективніше передають інформацію про конкретний об'єкт. Краї виступають як сильні індикатори деталей об'єкта.

Таким чином, важливі особливості мають високу локалізацію та чіткість, такі як кути, і служать важливими характеристиками. Використання відповідностей між функціями дозволяє нам пов'язати особливості аналогічної області на одному зображенні з характеристиками на іншому зображенні (рис.1.2.3).

Цей процес знаходить застосування в задачах комп'ютерного зору, таких як ідентифікація об'єктів та калібрування камери.

Рис.1.2.3 Зіставлення шаблонів *SIFT, SUFT*

Завдання зіставлення ознак зазвичай вирішується в певній послідовності, де визначення відповідностей між функціями відіграє ключову роль.

1. Виявлення функцій: виявлення областей зазвичай виконується за допомогою алгоритмів обробки зображень, таких як виявлення кутів *Харріса, SIFT і SUFT.*

2. Формування локальних дескрипторів: після виявлення об’єктів фіксується область, що оточує кожну ключову точку, і отримують локальні дескриптори цих областей. Локальний дескриптор є представленням локального околиці точки, тому може бути корисним для зіставлення ознак.

3. Відповідність функцій: об’єкти та їхні локальні дескриптори зіставляються на відповідних зображеннях для завершення кроку зіставлення функцій.

1.3 Проблеми технологій комп’ютерного зору

В галузі комп'ютерного зору наукова спільнота активно досліджує декілька відкритих проблем:

– розуміння сцени;
– розуміння відео;
– самоконтрольоване навчання;
– поодиноке навчання;
– стійкість;
– бробка в режимі реального часу;
– зрозумілий штучний інтелект.

Розуміння сцени – це здатність комп’ютера розуміти контекст зображення чи відео. Це включає в себе розпізнавання об’єктів, людей і дій, а також розуміння зв’язків між ними. Розуміння відео – це здатність комп’ютера розуміти вміст відео, зокрема розпізнавати об’єкти, людей і дії, а також розуміти їхню взаємодію з часом.

Самоконтрольоване навчання – це здатність комп’ютера навчатися на основі даних без явного контролю. Це може бути навчання шляхом передбачення того, чого не вистачає на зображенні, або передбачення наступного кадру у відео.

Поодиноке навчання – це здатність комп’ютера навчатися на невеликій кількості прикладів. Це складна проблема, оскільки вона вимагає від комп’ютера узагальнення на основі прикладів, щоб розпізнавати нові об’єкти.

Стійкість. Моделі комп’ютерного зору можна легко ввести в оману невеликими змінами в зображенні, що робить їх вразливими до агресивних атак.

Обробка в режимі реального часу. Моделі комп’ютерного зору мають обробляти візуальні дані в режимі реального часу, що може бути складно через обчислювальну складність багатьох алгоритмів.

Зрозумілий штучний інтелект. моделі комп’ютерного зору стають дедалі складнішими, тому важко зрозуміти, як вони прийшли до певного рішення.

Штучний інтелект, який можна пояснити, – це область досліджень, яка спрямована на те, щоб зробити моделі більш прозорими та доступними для інтерпретації.

1.4 Методи та підходи розпізнавання образів.

Розпізнавання образів включає два основні методи класифікації.

Контрольована класифікація. У контрольованому методі розпізнавання шаблонів людина навчає комп'ютерний алгоритм розпізнавати шаблони на основі попередньо визначених мічених наборів даних. Після ідентифікації шаблону метод згодом класифікує нові дані.

Неконтрольована класифікація. При неконтрольованій класифікації модель навчається самостійно без прямого керівництва людини. Комп'ютерний алгоритм визначає кореляції між декількома елементами даних (вхідними даними) на основі їх оцінки подібності та виконує класифікацію даних.

Найпопулярніші підходи для задачі розпізнавання образів:

– Статистичне розпізнавання образів.

Цей підхід до розпізнавання шаблонів використовує історичні статистичні дані, які навчаються на шаблонах і прикладах. Метод збирає спостереження та обробляє їх для визначення моделі. Потім ця модель узагальнює зібрані спостереження та застосовує правила до нових наборів даних або прикладів.

– Синтаксичне розпізнавання образів.

Передбачає складні шаблони, які можна ідентифікувати за допомогою ієрархічного підходу. Патерни встановлюються на основі того, як примітиви (наприклад, літери в слові) взаємодіють один з одним. Прикладом цього може бути те, як примітиви збираються в слова та речення. Такі навчальні зразки дозволять виробити граматичні правила, які демонструватимуть, як читатимуть речення в майбутньому.

– Розпізнавання за допомогою нейронних мереж.

Цей метод використовує штучні нейронні мережі (ШНМ) і вчиться на основі складних і нелінійних зв'язків введення/виведення, адаптується до даних і виявляє шаблони. Найпопулярнішим і ефективним методом у нейронних мережах є метод прямого зв'язку. У цьому методі навчання відбувається шляхом надання зворотного зв'язку шаблонам введення. Це схоже на те, як люди вчаться на своєму минулому досвіді та помилках.

Модель на основі ШНМ оцінюється як найдорожчий метод розпізнавання образів порівняно з іншими методами через обчислювальні ресурси, задіяні в процесі.

– Зіставлення шаблону.

Зіставлення шаблонів є одним із найпростіших підходів до розпізнавання образів. Тут подібність між двома сутностями визначається зіставленням зразка з еталонним шаблоном. Такі методи зазвичай використовуються в цифровій обробці зображень, де невеликі ділянки зображення зіставляються зі збереженим шаблонним зображенням. Деякі з його реальних прикладів включають обробку медичних зображень, розпізнавання обличчя та навігацію роботів.

– Нечіткий підхід.

У нечіткому підході набір шаблонів розбивається на основі подібності в характеристиках шаблонів. Коли унікальні особливості шаблону правильно виявлені, дані можна легко класифікувати в цьому відомому просторі ознак. Навіть зорова система людини іноді не може розпізнати певні компоненти, незважаючи на тривале сканування об’єктів. Те саме стосується цифрового світу, де алгоритми не можуть визначити точну природу об’єкта.

Отже, нечіткий підхід має на меті класифікувати об’єкти на основі кількох подібних ознак у виявлених шаблонах.

– Гібридний підхід.

Гібридний підхід використовує комбінацію вищевказаних методів, щоб отримати переваги всіх цих методів. Він використовує кілька класифікаторів для виявлення шаблонів, коли кожен класифікатор навчається на певному просторі ознак. Висновок робиться на основі результатів, накопичених за всіма класифікаторами.

1.5 Основні етапи процесу розпізнавання

Розпізнавання образів застосовується до даних усіх типів, включаючи зображення, відео, текст і аудіо. Оскільки модель розпізнавання образів може ідентифікувати повторювані шаблони в данних, прогнози, зроблені такими моделями, досить надійні.

Розпізнавання шаблонів включає три ключові етапи:

– аналіз вхідних даних;

– вилучення шаблонів;

– порівняння їх із збереженими даними.

Процес можна розглядати як двофазовий, переходячи від дослідження шаблонів даних до їхнього групування та приписування новим даним. Перша фаза це етап дослідження, де комп'ютерні алгоритми активно аналізують та досліджують шаблони в даних. Друга фаза, або описова, включає групування та призначення ідентифікованих шаблонів новим даним[10].

Ці етапи також можна додатково розділити на такі модулі:

– Збір даних є першим кроком розпізнавання образів.

Точність розпізнавання значною мірою залежить від якості наборів даних. Таким чином, використання наборів даних з відкритим вихідним кодом є кращим і може заощадити час замість процесів збору даних вручну. Таким чином, отримання даних з реального світу запускає процес розпізнавання.

– Попередня обробка.

Коли дані отримані , алгоритми починають етап попередньої обробки, на якому дані очищаються та виправляються від забруднення, щоб створити комплексні набори даних, які дають хороші прогнози. Попередня обробка передбачає сегментацію даних.

Наприклад, коли ви дивитеся на групову фотографію, опубліковану другом у соціальних мережах, ви розумієте, що вам знайомі деякі обличчя на фотографії, які привертають вашу увагу. Ось що означає попередня обробка.

Попередня обробка поєднується з покращенням. Це включає процес згладжування та нормалізації, який намагається виправити зображення від сильних варіацій. У результаті вхідні дані стає легко інтерпретувати для моделей.

– Виділення ознак.

Далі функції витягуються з попередньо оброблених вхідних даних. Тут вхідні дані перетворюються на вектор ознак, що представляє зменшену версію набору ознак. Цей крок вирішує проблему високої розмірності вхідного набору даних.

Це означає, що витягуються лише відповідні функції, а не використовується весь набір даних. Після виділення функцій слід вибрати функції з найвищим потенціалом для отримання точних результатів. Після відбору таких ознак вони надсилаються на подальшу класифікацію.

– Класифікація.

Витягнуті функції потім порівнюються з аналогічним шаблоном, що зберігається в базі даних. Тут навчання може відбуватися як під наглядом, так і без нагляду. Контрольований метод має попередні знання про кожну категорію шаблонів, тоді як неконтрольований метод навчання відбувається на льоту. Коли шаблони зрештою зіставляються зі збереженими даними, відбувається класифікація вхідних даних.

– Постобробка.

За класифікацією слідує етап постобробки, на якому приймаються рішення щодо найкращих способів використання результатів для ефективного керування системою. Крім того, це передбачає аналіз кожного сегмента ідентифікованих або секретних даних для отримання подальших ідей. Ці отримані відомості потім реалізуються на практиці для майбутніх завдань розпізнавання образів.

Висновки за розділом

У результаті аналізу науково-інформаційних джерел з теми дослідження в цьому розділі було визначено ключові аспекти галузі комп'ютерного зору. Зазначено, що завдання, такі як класифікація зображень, виявлення об'єктів, сегментація та розпізнавання обличь , становлять основну складову цього поля.

Виділено проблеми технологій комп'ютерного зору, що потребують уваги та вирішення для подальшого розвитку. Розглянуті різноманітні методи та підходи до розпізнавання образів, вказано на їхню різноманітність та можливості застосування в залежності від конкретних завдань.

Окреслено основні етапи процесу розпізнавання образів, підкреслено їхню важливість у структурі роботи систем комп'ютерного зору. Зроблено висновок про те, що комп'ютерний зір відіграє ключову роль у вирішенні різноманітних завдань, але його ефективність обумовлена вирішенням технічних та методологічних викликів.

Узагальнюючи, перший розділ надає повний огляд теми дослідження, визначаючи її важливість, основні завдання, проблеми та методи, що визначають контекст для подальших досліджень у галузі комп'ютерного зору.

РОЗДІЛ 2. ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, ЯК ІНСТРУМЕНТ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ

2.1. Ключові компоненти архітектури нейронної мережі.

Нейронні мережі є функціональною одиницею глибинного навчання і, як відомо, імітують поведінку людського мозку для вирішення складних проблем, керованих даними. Вхідні дані обробляються різними шарами штучних нейронів, складених разом, щоб отримати бажаний результат.[11]

Архітектура нейронної мережі(рис.2.1) стосується структури та дизайну нейронної мережі, включаючи кількість і розташування шарів, кількість нейронів у кожному шарі та зв’язки між нейронами. Архітектура суттєво впливає на здатність мережі вивчати та представляти складні шаблони в даних.

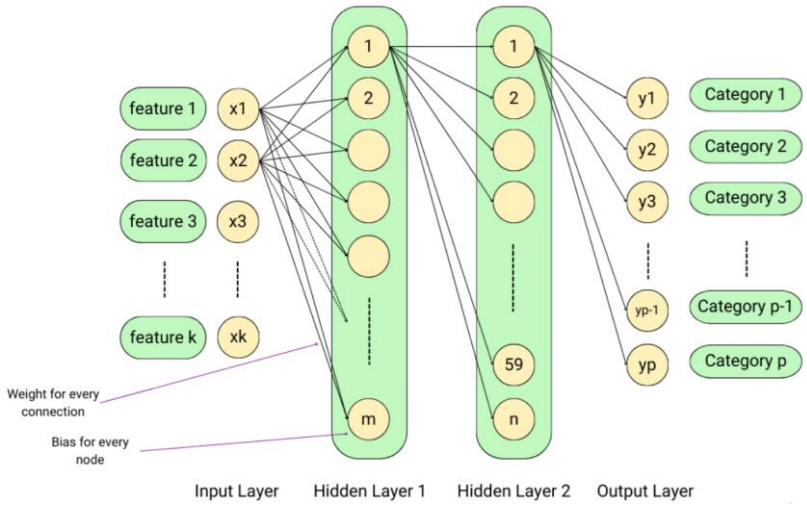


Рис.2.1 Принципова діаграма архітектури нейронної мережі.

Шари в нейронних мережах.

1. Вхідний рівень: отримує функції введення та не виконує жодних обчислень. Вхідний рівень є першим рівнем будь-якої нейронної мережі та представляє вхідні дані для мережі. Кожен нейрон, представлений у вигляді малих круглих вузлів (*x1, x2, ..., xn*) відповідає одній функції набору даних (див.рис. 2.1).

Наприклад, у моделі прогнозування ціни на житло вхідний рівень може мати нейрони для розміру будинку, відстані від залізничної станції та відстані від ринку. Розуміння вхідного рівня та його ролі в нейронній мережі має вирішальне значення для розробки та навчання ефективних моделей

2. Приховані рівні: проміжні рівні між вхідним і вихідним рівнями, де відбуваються обчислення та навчання. Іноді пари вхідних і вихідних даних можуть мати складні зв’язки, і для декодування цих зв’язків між вхідним і вихідним рівнями існують приховані шари. Приховані шари також містять нейрони, кожен нейрон з’єднується з кожним іншим нейроном у сусідніх шарах. Наприклад, нейрони прихованого шару 1 будуть підключені до кожного нейрона вхідного шару та прихованого шару 2.

3. Вихідний рівень: створює кінцевий результат мережі, часто передбачення для конкретного завдання. Кількість нейронів у цьому шарі відповідає кількості виходів, необхідних для даного входу. У задачі регресії, де очікується одне вихідне значення, у вихідному шарі буде один нейрон. Однак у завданнях класифікації, де можливі кілька класів виходу, буде кілька нейронів, по одному для кожного класу. Наприклад, у завданні на розпізнавання рукописних цифр буде 10 нейронів, які відповідають 10 можливим класам (0-9).

Нейрони (вузли).

Нейрони відіграють вирішальну роль у функціонуванні нейронної мережі, оскільки вони складають кожен рівень, включаючи вхідний, вихідний і прихований. Подібно до ядра клітин мозку, кожен нейрон, крім тих, що знаходяться на вхідному рівні, містить параметр зміщення, який нейронна мережа вивчає та коригує під час процесу навчання. Ці значення зсуву зазвичай ініціалізуються випадковими числами, і нейронна мережа точно налаштовує їх, щоб мінімізувати різницю між обчисленим і фактичним результатом.[11]

Зв'язки.

Зв’язки між нейронами в нейронній мережі мають вирішальне значення для процесу навчання. Кожен нейрон в одному шарі з’єднаний з кожним нейроном у сусідніх шарах.

Ці зв’язки представлені значенням ваги, яке визначає важливість цього зв’язку. Значення ваги — це параметри, які можна навчити, які нейронна мережа вивчає шляхом повторення набору даних навчання.

Оптимізація цих вагових значень має вирішальне значення для загальної продуктивності нейронної мережі та є ключовим аспектом процесу навчання.

Матриця ваги.

Вагова матриця, поєднання значень ваги та зсуву, є ключовим аспектом нейронних мереж. Він представляє параметри мережі, які можна дізнатися, і допомагає моделі робити прогнози. Вагова матриця використовується для відображення вхідних даних і вихідних даних у формі матриці, що полегшує її обчислення та оновлення під час процесу навчання.

Функція активації.

У нашому мозку нейрони активуються на основі сигналів, отриманих через різні органи чуття. Оскільки ці сигнали можуть бути пов’язані з різними завданнями, різні нейрони активуються та забезпечують необхідні відповіді. Подібно до цього, у нейронних мережах ми маємо функції активації для нейронів кожного рівня. Ми визначаємо функцію активації для шару, і всі нейрони в цьому шарі дотримуються тієї самої функції активації. Нейрони кожного шару отримують вхідні дані від попереднього шару, помножені на значення ваги.

Базуючись на зв’язку, який ми хочемо зберегти між зваженим входом і відповідним виходом нейрона, ми можемо розділити функції активації на два типи:

– Лінійні функції активації: в нейронній мережі функція лінійної активації використовується для прямої передачі зважених вхідних даних як вихідних даних без будь-яких додаткових перетворень. Ця функція гарантує, що зв’язок між вхідним зваженим входом і відповідним виходом цього нейрона є лінійним. Цей тип функції активації зазвичай використовується на вихідному рівні нейронної мережі.

– Нелінійні функції активації: це математичні функції, які перетворюють зважені вхідні дані, отримані нейроном, для створення нелінійного зв’язку між входом і виходом.

Ці функції допомагають нейронам виділяти складні шаблони, присутні в даних, і в основному використовуються в прихованих шарах нейронної мережі.

Функція втрати/вартості

Функції збитків і витрат є одними з найважливіших компонентів будь-якого алгоритму машинного навчання. Машини розуміють лише числа, тому нам потрібно передати наші цілі в цифрах. Якщо наші цілі у формі чисел не відображають те, чого ми хочемо, щоб наші машини навчилися, ми будемо винні в тому, що машини не навчаться.

Основні моменти під час розробки функції витрат для нейронної мережі (далі НН) :

– Функція витрат повинна представляти постановку проблеми у вигляді помилок між прогнозованим результатом НН та очікуваним результатом НН.

– У НН можуть бути тисячі параметрів. Отже, незначні зміни параметричних значень повинні відображати зміну функції витрат.

– Функція витрат має бути диференційованою. Наприклад, похибка між прогнозованим і очікуваним результатом може становити один градус. Тим не менш, ми можемо зробити помилку⁴ нашою функцією витрат, що зробить її диференційованою та зрозумілою для машин у процесі навчання.

– Функція вартості визначена лише для вихідного шару, а не для вхідного та прихованого шарів.

Параметри

Під час роботи з нейронними мережами необхідно враховувати два типи параметрів: параметри, які можна навчити, і гіперпараметри.

Параметри, які можна навчити: ключовим аспектом штучних нейронних мереж (ШНМ) є вагова матриця, яка містить параметри, які можна навчити, які можна змінювати під час процесу навчання. Ці параметри включають зміщення нейронів у всіх шарах, крім вхідного, і ваги зв’язків між нейронами.

Загальну кількість параметрів, які можна навчити, можна обчислити шляхом додавання загальної кількості нейронів у прихованому та вихідному шарах до загальної кількості з’єднань.

Гіперпараметри (параметри, які неможливо навчити) – під час створення штучної нейронної мережі (ШНМ) важливо встановити фіксовані значення, відомі як гіперпараметри, які потім точно налаштовуються за допомогою експериментування для досягнення найменшої можливої вартості.

2.2 Загальні категорії та ключові типи архітектур нейромереж.

Зі швидким розвитком глибокого навчання була створена ціла низка архітектур нейронних мереж для вирішення широкого спектру завдань і проблем. Хоча існує незліченна кількість архітектур нейронних мереж, ось одинадцять, які необхідні для розуміння будь-яким інженером глибокого навчання, розділених на чотири загальні категорії: стандартні мережі, рекурентні мережі, згорткові мережі та автокодери.

2.1.1 Стандартні мережі

1. Перцептрон.

Персептрон є найосновнішою з усіх нейронних мереж, будучи основним будівельним блоком більш складних нейронних мереж.[12] Він просто з’єднує вхідну комірку та вихідну комірку(рис.2.2). Персептрон — це алгоритм для контрольованого навчання бінарних класифікаторів. Цей алгоритм дозволяє нейронам вивчати й обробляти елементи навчального набору по одному.

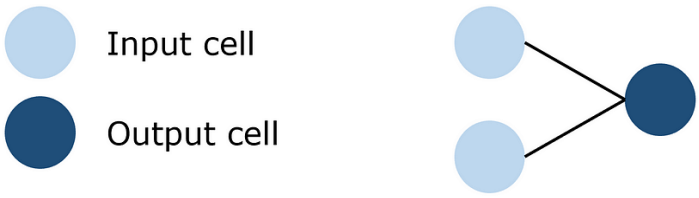


Рис.2.2 Архітектура персептрону

2. Мережа прямого зв'язку.

Мережа прямого зв’язку – це набір перцептронів, у якому є три основні типи рівнів – вхідні, приховані та вихідні. Під час кожного з’єднання сигнал із попереднього рівня множиться на вагу, додається до зміщення та пропускається через функцію активації.

Мережі прямого зв’язку використовують зворотне поширення для повторного оновлення параметрів, доки не буде досягнуто бажаної продуктивності (рис.2.3).

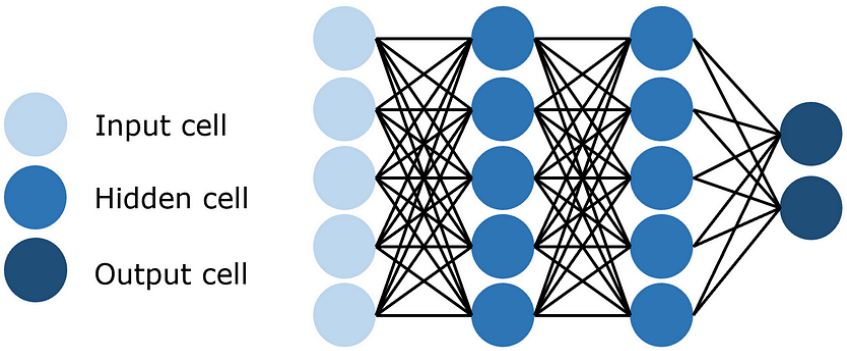


Рис.2.3. Архітектура мережі прямого зв’язку

3. Залишкові мережі (*ResNet*).

Одна з проблем нейронних мереж глибокого прямого зв’язку називається проблемою зникаючого градієнта, коли мережі занадто довгі, щоб корисна інформація могла поширюватися по всій мережі. Коли сигнал, який оновлює параметри, проходить через мережу, він поступово зменшується, доки вагові коефіцієнти на передній частині мережі не зміняться або не використовуються взагалі (рис.2.4).

Щоб вирішити цю проблему, залишкова мережа використовує пропускаючі з’єднання, які поширюють сигнали через «перескочений» рівень. Це зменшує проблему зникнення градієнта, використовуючи з’єднання, які є менш вразливими до неї. З часом мережа вчиться відновлювати пропущені шари, вивчаючи простір функцій, але ефективніша в навчанні, оскільки вона менш вразлива до зникаючих градієнтів і потребує дослідження меншої частини простору функцій.

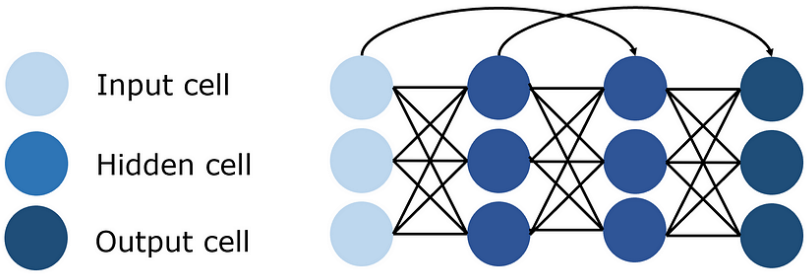


Рис.2.4. Архітектура залишкової мережі (*ResNet*)

2.1.2 Повторювані мережі

4. Повторювана нейронна мережа (*RNN*).

Рекурентна нейронна мережа — це спеціалізований тип мережі, яка містить цикли та повторюється над собою, звідси й назва «повторний». Дозволяє зберігати інформацію в мережі, *RNN* використовують міркування з попереднього навчання, щоб приймати кращі та більш обґрунтовані рішення щодо майбутніх подій. Для цього він використовує попередні передбачення як «сигнали контексту». Через свою природу *RNN* зазвичай використовуються для вирішення послідовних завдань, таких як генерування тексту буква за літерою або прогнозування даних часових рядів (наприклад, курсів акцій). Вони також можуть обробляти вхідні дані будь-якого розміру.(рис.2.5)

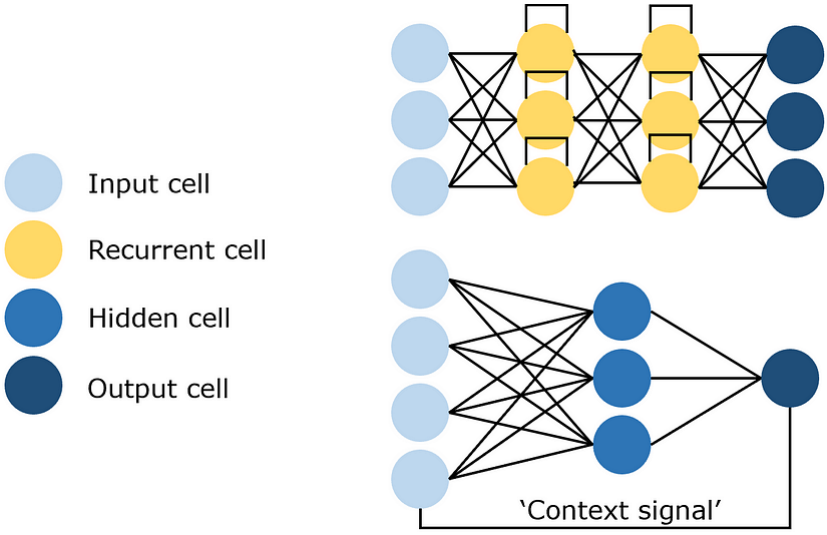


Рис.2.5 Два методи візуалізації архітектури *RNN*

5. Мережа довгострокової короткострокової пам'яті *(LSTM*).

RNN є проблемою, оскільки діапазон контекстної інформації на практиці дуже обмежений. Вплив (помилка зворотного поширення) даного вхідного сигналу на вхід на прихованому рівні (і, отже, на вихід мережі) або зростає експоненціально, або зникає нанівець, оскільки циклічно обертається навколо з'єднань мережі. Рішенням цієї проблеми зникнення градієнта є мережа довготривалої короткочасної пам'яті, або *LSTM* (рис.2.6).

Ця архітектура *RNN* спеціально розроблена для вирішення проблеми зникнення градієнта, додаючи структуру блокам пам’яті. Ці блоки можна розглядати як мікросхеми пам’яті в комп’ютері — кожна з них містить кілька періодично з’єднаних комірок пам’яті та три вентилі (вхід, вихід і забуття, еквіваленти запису, читання та скидання).

Мережа може взаємодіяти зі стільниками лише через кожен шлюз, і, отже, шлюзи вчаться розумно відкриватися та закриватися, щоб запобігти вибуху або зникненню градієнтів, а також поширювати корисну інформацію через «каруселі постійних помилок», а також відкидати нерелевантний вміст пам’яті.

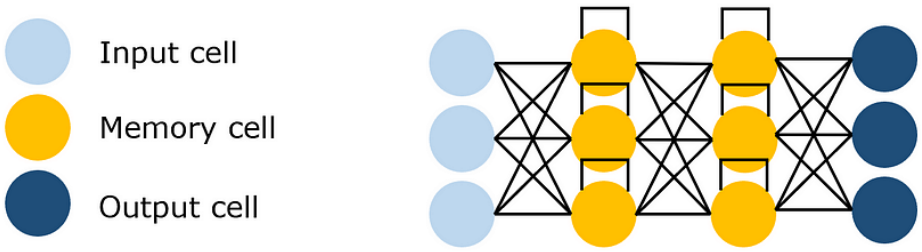


Рис.2.6. Архітектура довгострокової короткострокової пам'яті *(LSTM)*

Там, де стандартні *RNN* не можуть визначити наявність часових затримок, що перевищують п’ять-десять часових кроків між вхідними подіями та цільовими сигналами, це не впливає на *LSTM* і може навчитися з’єднувати часові затримки навіть на 1000 часових кроків, забезпечуючи корисний постійний потік помилок.

6. Мережі *Echo State (ESN).*

Мережа ехо-стану є варіантом рекурентної нейронної мережі з дуже рідко зв’язаним прихованим шаром (зазвичай одновідсоткова зв’язність). Зв’язок і ваги нейронів призначаються випадковим чином і ігнорують розбіжності шарів і нейронів (пропуск з’єднань). Вага вихідних нейронів вивчається таким чином, що мережа може виробляти та відтворювати певні часові шаблони. Обґрунтування цієї мережі випливає з того факту, що, незважаючи на те, що вона є нелінійною, єдині вагові коефіцієнти, які змінюються під час навчання, це з’єднання синапсів, і, отже, функцію помилок можна диференціювати на лінійну систему (рис.2.7).

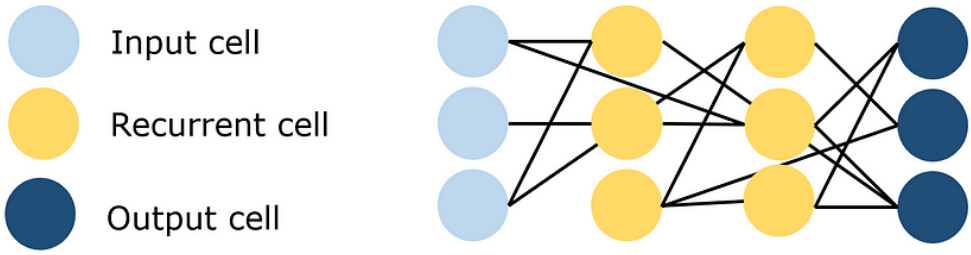


Рис.2.7. Архітектура Мережі *Echo State (ESN)*

2.1.3 Згорткові мережі

7. Згорткова нейронна мережа (*CNN*).

Зображення мають дуже високу розмірність, і, отже, навчання стандартної прямої мережі розпізнаванню зображень потребуватиме сотень тисяч вхідних нейронів, що, окрім явно високих витрат на обчислення, може спричинити багато проблем, пов’язаних із прокляттям розмірності в нейронних мережах. . Конволюційна нейронна мережа (*CNN*) забезпечує вирішення цієї проблеми шляхом використання згорткових і об’єднаних шарів, щоб зменшити розмірність зображення.

Оскільки згорткові шари можна навчати, але вони мають значно менше параметрів, ніж стандартний прихований шар, він здатний виділяти важливі частини зображення та передавати кожну з них вперед. Традиційно в *CNN* кілька останніх шарів є прихованими, які обробляють «згорнуту інформацію про зображення» (рис.2.8).

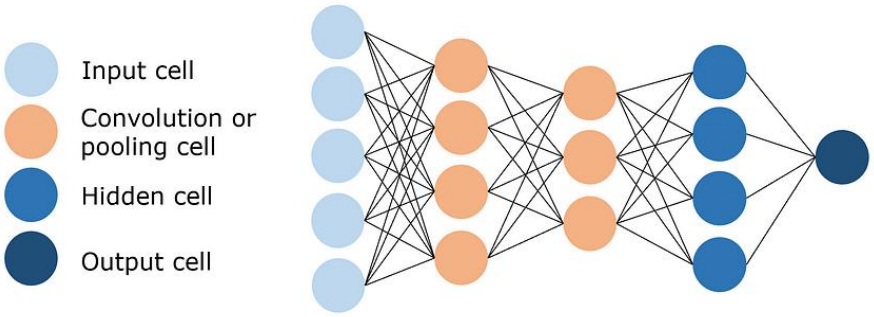


Рис.2.8. Архітектура згорткової нейронної мережі (*CNN*)

Згорткові нейронні мережі добре виконують завдання на основі зображень, наприклад, класифікують зображення як собаку чи кота.

Цей тип алгоритму глибокого навчання, який в основному використовується для завдань класифікації зображень і розпізнавання об’єктів.

8. Деконволюційна нейронна мережа (*DNN*)

Деконволюційні нейронні мережі, як випливає з назви, працюють протилежно згортковим нейронним мережам. Замість виконання згорток для зменшення розмірності зображення, *DNN* використовує деконволюції для створення зображення, зазвичай із шуму (рис.2.9).

Це за своєю суттю складне завдання; розглянемо завдання *CNN* написати короткий виклад із трьох речень повної книги Орвелла *1984* , тоді як завдання *DNN* полягає в написанні повної книги з трьох структура речення.

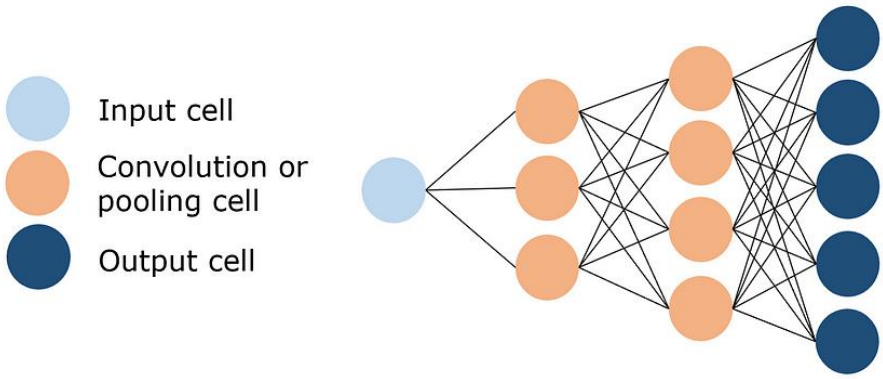


Рис.2.9 Архітектура деконволюційної нейронної мережі *(DNN)*

9. *Generative Adversarial Network (GAN).*

Генеративна змагальна мережа — це спеціалізований тип мережі, створений спеціально для створення зображень і складається з двох мереж — дискримінатора та генератора. Завдання дискримінатора полягає в *розрізненні* між тим, чи отримано зображення з набору даних, чи воно було згенеровано генератором, і завданням генератора є генерувати зображення, достатньо переконливі, щоб дискримінатор не міг розрізнити, справжні вони чи ні (рис.2.10).

З часом, завдяки ретельному регулюванню, ці два противники конкурують між собою, кожен прагне досягти успіху, покращуючи іншого. Кінцевим результатом є добре навчений генератор, який може створити реалістичне зображення.

Дискримінатор — це згорточна нейронна мережа, метою якої є максимізація точності ідентифікації справжніх/фальшивих зображень, тоді як генератор — це згорточна нейронна мережа, метою якої є мінімізація продуктивності дискримінатора.

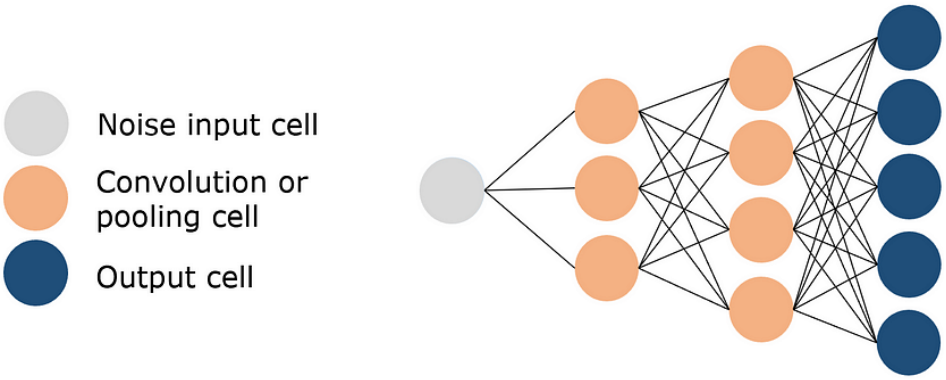


Рис.2.10 Архітектура генеративної змагальної мережі *(GAN)*

2.1.4 Автоматичні кодери

10. Автоматичний кодер (АК).

Фундаментальна ідея автокодувальника полягає в тому, щоб отримати оригінальні високорозмірні дані, «стиснути» їх у високоінформативні та низькорозмірні дані, а потім спроектувати стиснуту форму в новий простір.

Існує багато застосувань автокодерів, включаючи зменшення розмірності, стиснення зображень, усунення шумів у даних, виділення ознак, генерацію зображень і системи рекомендацій. Він може використовуватися як неконтрольований або контрольований метод, може бути дуже проникливим щодо природи даних (рис.2.11).

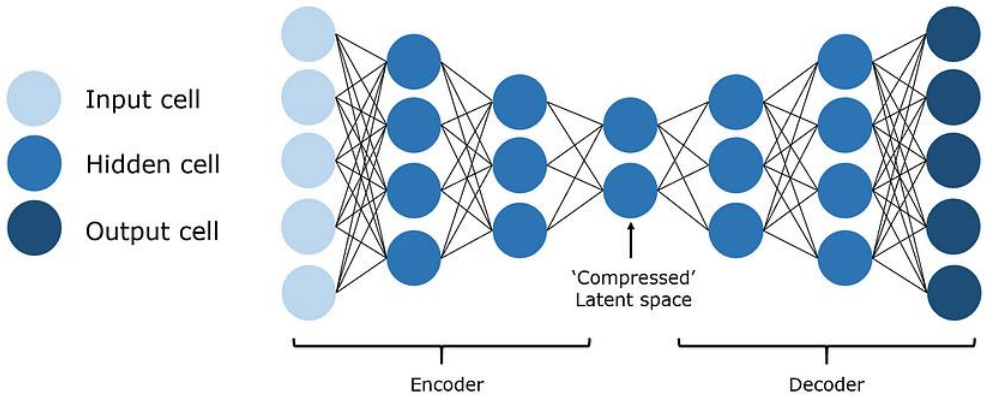


Рис.2.11 Архітектура автоматичного кодера

Приховані комірки можна замінити згортковими шарами для обробки зображень.

11. Варіаційний автоматичний кодер (*VAE*).

У той час як автокодер вивчає стисне представлення вхідних даних, якими можуть бути зображення або текстові послідовності, наприклад, стискаючи вхідні дані, а потім розпаковуючи їх назад, щоб відповідати вихідним вхідним даних, варіаційний автокодер (*VAE*) вивчає параметри розподілу ймовірностей представлення даних.

Замість того, щоб просто вивчати функцію, що представляє дані, він отримує більш детальне та нюансоване уявлення про дані, вибірку з розподілу та генерацію нових вибірок вхідних даних. У цьому сенсі це більше суто «генеративна» модель, як *GAN* (рис.2.12).

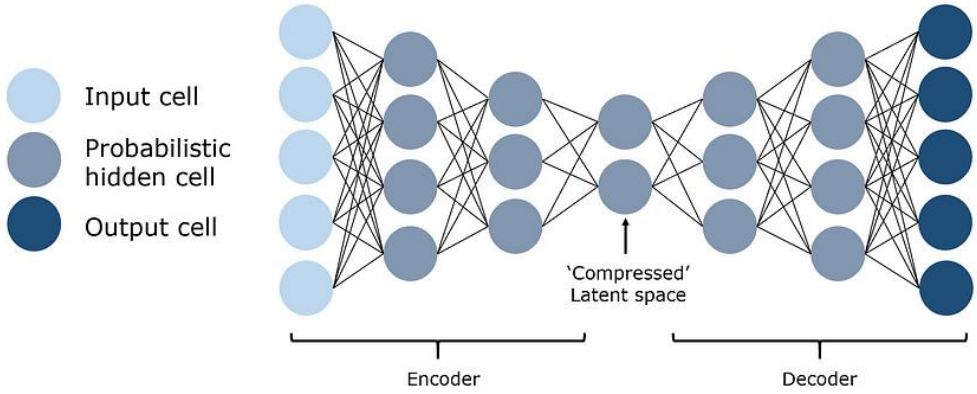


Рис.2.12 Архітектура варіаційного автоматичного кодера (*VAE*)

VAE використовує імовірнісну приховану клітинку, яка застосовує радіальну базисну функцію до різниці між тестовим випадком і середнім значенням клітинки.

Висновки за розділом

У цьому розділі було проведено докладний аналіз внутрішньої структури штучних нейронних мереж (ШНМ) як ключового елементу машинного навчання. Увага була акцентована на визначенні та розгляді всіх суттєвих компонентів цієї техніки, включаючи вхідний рівень, прихований рівень, вихідний рівень, вагові матриці, функцію витрат, параметри та гіперпараметри.

В ході аналізу кожного компонента, надано детальне розгортання вагових матриць, визначено значущість функцій витрат та особливості налаштування параметрів та гіперпараметрів для досягнення оптимальної ефективності ШНМ. Основна мета полягає в наданні повного технічного висвітлення внутрішньої механіки мережі.

В процесі аналізу я визначила ключовий напрямок подальших досліджень та розробок у галузі систем ідентифікації, вибравши згорткові нейронні мережі (*CNN*) для подальшої реалізації. Обрана архітектура *CNN* вважається оптимальною для досягнення високої точності та ефективності в розпізнаванні образів, що є ключовим завданням у вибраній області дослідження.

РОЗДІЛ 3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЗАДАЧІ ІДЕНТИФІКАЦІЇ ОБРАЗІВ

3.1 *Python* як основна мова програмування в галузі машинного навчання.

У динамічному світі штучного інтелекту та рішень, керованих даними, машинне навчання (МН) є революційною технологією, яка формує майбутнє багатьох галузей. В основі цієї трансформації лежить *Python*, мова програмування, яка зміцнила свою позицію як кращий вибір для дослідників, спеціалістів із обробки даних і розробників у сфері машинного навчання. *Python* - інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом.[13]

Python заслужив своє місце як одна з найпопулярніших мов програмування серед професіоналів МН завдяки легкому для читання синтаксису, великим бібліотекам і крос-платформній сумісності. Як мова програмування високого рівня з відкритим вихідним кодом, *Python* став найкращим вибором для широкого спектру завдань машинного навчання, від аналізу даних до глибокого навчання. Зростаюча популярність *Python* у проектах зі штучного інтелекту (ШІ) і МН не випадкова, оскільки вона забезпечує чудове середовище для розробників, щоб вирішувати навіть найскладніші завдання машинного навчання.

Розгалужена бібліотечна екосистема *Python*, надійні можливості візуалізації, низький бар’єр входу, сильна підтримка спільноти, гнучкість, читабельність і незалежність від платформи роблять його ідеальним вибором для цілей машинного навчання. Як наслідок, *Python* спостерігав сплеск використання в додатках ШІ і МН, включаючи розпізнавання зображень і мови, прогнозну аналітику та автономні транспортні засоби.

Синтаксис *Python* інтуїтивно зрозумілий і простий, що робить його популярною мовою програмування, яку легко читати. Об’єктно-орієнтоване програмування надає розробникам логічний метод організації, обробки та планування коду відповідно. Це полегшує розробку чистого та лаконічного коду для проектів будь-якої складності.

У результаті *Python* став популярною початковою мовою для розробників-початківців і вибором для досвідчених програмістів. Легкий для читання синтаксис Python не тільки робить його доступним для початківців, але також дозволяє швидше розробляти та налагоджувати. З *Python* код стає більш розбірливим і легшим для налагодження, що полегшує виявлення та виправлення помилок і оперативну розробку нових функцій. Цей зручний характер *Python* значно сприяв його широкому впровадженню в спільноті машинного навчання.

Великі бібліотеки та фреймворки.

Одним із ключових факторів, які відрізняють *Python* від інших мов програмування, є його комплексна бібліотечна екосистема. *Python* пропонує широкий спектр бібліотек і фреймворків, спеціально розроблених для машинного навчання, що полегшує розробникам впровадження алгоритмів МН. Деякі популярні бібліотеки *Python* для машинного навчання представлено нижче.

NumPy – це фундаментальна бібліотека *Python* для ефективних числових обчислень і операцій із масивами.

Scikit-learn – це комплексна бібліотека машинного навчання, яка пропонує широкий спектр інструментів для різних завдань, зокрема класифікації, регресії, кластеризації тощо.

Pandas – це потужна бібліотека для аналізу та обробки даних, що забезпечує інтуїтивно зрозумілі структури даних, такі як *DataFrames і Series*.

TensorFlow – це передова бібліотека глибокого навчання, відома своїми можливостями розподіленого обчислення та надійною екосистемою.

Theano – це бібліотека *Python*, призначена для швидких чисельних обчислень, особливо корисна для навчання моделей глибокого навчання.

Keras –це простий у використанні *API* глибокого навчання, який діє як інтерфейс для *TensorFlow, Theano* або *Microsoft Cognitive Toolkit (CNTK),* спрощуючи створення та навчання нейронних мереж.

PyTorch –це динамічна бібліотека глибокого навчання з гнучким графіком обчислень, що робить її ідеальною для розробки та навчання складних нейронних мереж.

Ці бібліотеки та фреймворки *Python* надають потужні можливості для аналізу даних, машинного та глибокого навчання, дозволяючи розробникам зосередитися на вирішенні складних завдань без необхідності винаходити колесо.

Кросплатформна сумісність *Python* дозволяє розробникам створювати код, який можна використовувати на різних платформах, таких як *Windows, Mac і Linux.* Ця гнучкість полегшує розробку програм, які можна використовувати в різних операційних системах без необхідності переписувати вихідний код. Таким чином, це дозволяє розробникам використовувати той самий код для різних платформ, заощаджуючи час і зусилля.

Однак кросплатформна сумісність пов'язана з певними труднощами. На різних платформах можуть бути встановлені різні версії *Python*, що може призвести до проблем із сумісністю під час виконання коду на різних платформах. Щоб подолати ці проблеми, важливо переконатися, що код написаний у спосіб, сумісний з усіма підтримуваними версіями, і що він протестований на всіх платформах, щоб гарантувати, що він функціонує належним чином.

Python широко відомий своєю масштабованістю та винятковою продуктивністю в машинному навчанні. Його універсальність, зручний характер і великі бібліотеки роблять його ідеальним вибором для масштабування операцій машинного навчання. Завдяки таким бібліотекам, як *NumPy, pandas і TensorFlow*, *Python* дає змогу виконувати складні операції з масивними наборами даних, демонструючи свою високу масштабованість. Його вміння працювати з великими даними сприяє його широкому застосуванню.

Простота й читабельність *Python* ще більше сприяють швидкому створенню прототипів, прискорюючи ітераційний процес розробки й тонкого налаштування моделей МН. Однак продуктивність *Python* створює проблеми. Як інтерпретована мова, *Python* є відносно повільнішим порівняно з такими мовами, як *C++* або *Java*. Тим не менш, такі бібліотеки, як *NumPy* і *Cython*, вирішують цю проблему, виконуючи обчислення зі швидкістю, близькою до C. Крім того, інфраструктури розподілених обчислень, такі як *Apache Spark* і *Dask*, значно покращують продуктивність *Python* у програмах МН.

У порівнянні з іншими мовами, простий синтаксис *Python* забезпечує швидку розробку та налагодження, полегшуючи розробникам створення читабельного коду та швидке виявлення та виправлення помилок.

Крім того, *Python* не залежить від платформи, тобто він може працювати в різних операційних системах, включаючи *Windows, Mac і Linux*. Ця крос-платформна сумісність дає розробникам можливість створювати програми МН, які можна використовувати на кількох платформах без необхідності переписувати вихідний код.

Переваги *Python* над іншими мовами програмування.

Простота і читабельність.

Python: синтаксис *Python* чіткий, лаконічний і легко читається. Це полегшує як початківцям, так і досвідченим розробникам писати та розуміти код, сприяючи швидшій розробці та співпраці.

R: R призначено в основному для статистичного аналізу та має крутішу криву навчання для нестатистів. Його синтаксис може бути менш інтуїтивно зрозумілим для завдань програмування загального призначення порівняно з *Python.*

Java/C++: I Java, і C++ мають складніший синтаксис і вимагають більше шаблонного коду, що робить їх менш придатними для швидкого створення прототипів і експериментів у машинному навчанні.

Велика екосистема бібліотек і фреймворків.

Python: Python має багату екосистему бібліотек машинного навчання, таких як *TensorFlow, Keras, PyTorch, scikit-learn* тощо. Ці бібліотеки забезпечують попередньо реалізовані функції та моделі, які значно прискорюють розробку.

R: R також має деякі корисні бібліотеки машинного навчання, як-от *caret і randomForest,* але екосистема *Python* є більш обширною та активно підтримується.

Java/C++: хоча *Java і C++* мають бібліотеки машинного навчання, їхні пропозиції, як правило, не такі повні, як *Python.*

Підтримка спільноти та документація.

Python: Python має величезну та активну спільноту, що означає, що для ентузіастів і практиків машинного навчання доступні численні ресурси, навчальні посібники та обговорення.

R: Спільнота *R* також активна, але вона може бути не такою великою, як *Python,* коли йдеться про машинне навчання.

Інтеграція та прототипування.

Python: простота *Python* і легкість інтеграції з іншими технологіями, такими як веб-фреймворки, інструменти аналізу даних і хмарні служби, роблять його чудовим вибором для створення прототипів і створення наскрізних програм машинного навчання.

R: R є винятковим для статистичного аналізу та візуалізації, але може бути не так плавно інтегрований у нестатистичні програми, як *Python.*

Java/C++: хоча ці мови добре підходять для великих систем, вони можуть вимагати більше зусиль у розробці для швидкого створення прототипів і експериментів.

Продуктивність і ефективність.

Python: продуктивність *Python* значно покращилася завдяки таким бібліотекам, як *NumPy* і методи компіляції *JIT*. Хоча він не такий швидкий, як *Java* або *C++,* він забезпечує прийнятний рівень продуктивності для більшості завдань машинного навчання.

R: R може працювати повільніше, ніж *Python* для певних операцій, особливо під час обробки великих наборів даних.

Java/C++: Java i C++ відомі своєю високою продуктивністю та ефективністю, особливо в обчислювально інтенсивних завданнях, але вони часто вимагають більше коду для досягнення тієї самої функціональності.

3.2 Порівняння фреймворків глибокого навчання.

Keras - це ефективний високорівневий інтерфейс програмування прикладних програм (*API)* нейронної мережі, написаний мовою *Python*.[14] Ця бібліотека нейронних мереж із відкритим вихідним кодом створена для швидкого експериментування з глибокими нейронними мережами та може працювати поверх *CNTK, TensorFlow і Theano*. *Keras* фокусується на тому, щоб бути модульним, зручним і розширюваним. Він не обробляє низькорівневі обчислення, натомість він передає їх іншій бібліотеці під назвою *Backend. Keras* було прийнято та інтегровано в *TensorFlow* у середині 2017 року. Користувачі можуть отримати доступ до нього через модуль *tf.keras*. Однак бібліотека *Keras* все ще може працювати окремо та незалежно.

PyTorch — це відносно нова структура глибокого навчання на основі *Torch.* Розроблено групою досліджень штучного інтелекту *Facebook* і відкрито на *GitHub* у 2017 році, воно використовується для програм обробки природної мови.[15] *PyTorch* має репутацію простоти, легкості використання, гнучкості, ефективного використання пам'яті та динамічних обчислювальних графіків. Він також виглядає рідним, роблячи кодування легшим і збільшуючи швидкість обробки.

TensorFlow – це наскрізна платформа глибокого навчання з відкритим вихідним кодом, розроблена *Google* і випущена в 2015 році.[16] Вона відома підтримкою документації та навчання, масштабованим виробництвом і можливостями розгортання , кілька рівнів абстракції та підтримка різних платформ, наприклад *Android*. *TensorFlow* – це символічна математична бібліотека, яка використовується для нейронних мереж і найкраще підходить для програмування потоків даних у ряді завдань. Він пропонує кілька рівнів абстракції для створення та навчання моделей. Крім того, як згадувалося раніше, *TensorFlow* прийняв *Keras*, що робить їх порівняння проблематичним. Тим не менш, користувачам *Keras* необов'язково використовувати *TensorFlow.*

3.2.1 *PyTorch* проти *TensorFlow*.

І *TensorFlow,* і *PyTorch* пропонують корисні абстракції, які полегшують розробку моделей за рахунок скорочення шаблонного коду. Вони відрізняються тим, що *PyTorch* має більш "пітонічний" підхід і є об’єктно-орієнтованим, тоді як *TensorFlow* пропонує різноманітні варіанти.

PyTorch сьогодні використовується для багатьох проектів глибокого навчання, і його популярність зростає серед дослідників ШІ, хоча з трьох основних фреймворків він є найменш популярним. Тенденції показують, що незабаром це може змінитися. Коли дослідники хочуть гнучкості, можливостей налагодження та короткої тривалості навчання, вони обирають *PyTorch*. Він працює на *Linux, macOS* і *Windows*.[17]

Завдяки добре задокументованому фреймворку та великій кількості навчених моделей і посібників *TensorFlow* є улюбленим інструментом багатьох професіоналів галузі та дослідників. *TensorFlow* пропонує кращу візуалізацію, що дозволяє розробникам краще налагоджувати та відстежувати процес навчання.

Однак *PyTorch* забезпечує лише обмежену візуалізацію. *TensorFlow* також перемагає *PyTorch* у розгортанні навчених моделей у виробництві завдяки фреймворку *TensorFlow Serving. PyTorch* не пропонує такої структури, тому розробникам потрібно використовувати *Django* або *Flask* як внутрішній сервер.

У сфері паралелізму даних *PyTorch* отримує оптимальну продуктивність, покладаючись на власну підтримку асинхронного виконання через *Python.* Однак із *TensorFlow* ви повинні вручну кодувати та оптимізувати кожну операцію, що виконується на конкретному пристрої, щоб забезпечити розподілене навчання. Таким чином, можна копіювати все з *PyTorch у TensorFlow*.

3.2.2 *PyTorch* проти *Keras*.

Обидва ці варіанти підходять, якщо ви тільки починаєте працювати з фреймворками глибокого навчання. Математикам і досвідченим дослідникам *PyTorch* більше сподобається.

Keras краще підходить для розробників, яким потрібен фреймворк *plug-and-play*, який дозволяє їм швидко створювати, навчати та оцінювати свої моделі. *Keras* також пропонує більше варіантів розгортання та спрощений експорт моделі. *PyTorch* швидший за *Keras* і має кращі можливості налагодження

Обидві платформи користуються достатнім рівнем популярності, тому пропонують багато навчальних ресурсів. *Keras* має чудовий доступ до багаторазового коду та посібників, тоді як *PyTorch* має видатну підтримку спільноти та активний розвиток. *Keras* є найкращим при роботі з невеликими наборами даних, швидким створенням прототипів і підтримкою кількох серверних систем. Це найпопулярніший фреймворк завдяки своїй порівняльній простоті. Він працює на *Linux, MacOS* і *Windows.*

3.2.3 *TensorFlow* проти *Keras.*

TensorFlow — це наскрізна платформа з відкритим вихідним кодом, бібліотека для багатьох завдань машинного навчання, а *Keras* — це бібліотека нейронних мереж високого рівня, яка працює на основі *TensorFlow*. Обидва надають *API* високого рівня, які використовуються для легкої побудови та навчання моделей, але *Keras* є більш зручним для користувача, оскільки він вбудований у *Python.*

Дослідники звертаються до *TensorFlow,* коли працюють із великими наборами даних і виявленням об’єктів, і їм потрібна відмінна функціональність і висока продуктивність. *TensorFlow* працює на Linux, *MacOS, Windows* і *Android.* Фреймворк був розроблений *Google Brain* і зараз використовується для дослідницьких і виробничих потреб Google. Порівняння *TensorFlow і Keras* — не найкращий спосіб підійти до питання, оскільки *Keras* функціонує як оболонка для фреймворку *TensorFlow*. Таким чином, можна визначити модель з інтерфейсом *Keras,* який простіший у використанні, а потім перейти до *TensorFlow,* коли вам потрібно використовувати функцію, якої немає у *Keras*, або ви шукаєте певну функціональність *TensorFlow.* Це дозволяє розмістити свій код *TensorFlow* безпосередньо в навчальному конвеєрі або моделі *Keras.*

Порівняння основних характеристик оглянутих фреймворків представлено у таблиці 3.1.

Таблиця 3.1

Порівняльна таблиця між *TensorFlow, PyTorch і Keras*

	TensorFlow	*PyTorch*	*Keras*
Рівень API	Високий і низький	Низький	Високий
Простота використання	Середня, крута крива навчання для початківців	Висока, з репутацією простоти та гнучкості.	Висока, створений для швидкого експериментування з глибокими нейронними мережами
Архітектура	Не простий у використанні	Складний, менш читабельний	Просто, лаконічно, читабельно
Рівні абстракції	Кілька рівнів абстракції для створення та навчання моделей	Кілька рівнів абстракції з динамічним обчислювальним графіком	*API* високого рівня з попередньо створеними шарами та моделями
Підтримка бекенда	*TensorFlow, Keras*	*PyTorch*	*TensorFlow, Theano, CNTK*
Набори даних	Великі набори даних, висока продуктивність	Великі набори даних, висока продуктивність	Менші набори даних
Налагодження	Важко проводити налагодження	Хороші можливості налагодження	Проста мережа, тому налагодження часто не потрібно
Наявність попередньо тренованої моделі	Так	Так	Так
Популярність	Другий за популярністю з трьох	Третій за популярністю з трьох	Найпопулярніший із трьох
Швидкість	Швидкий, високопродуктивно	Швидкий, високопродуктивно	Повільний, низька продуктивність

3.3 Бібліотека *OpenCV*.

OpenCV (Open Source Computer Vision Library) — бібліотека комп'ютерного зору та машинного навчання з відкритим кодом.[18] *OpenCV* було створено, щоб забезпечити загальну інфраструктуру для програм комп'ютерного зору та прискорити використання машинного сприйняття в комерційних продуктах. Будучи ліцензованим продуктом *Apache 2, OpenCV* дозволяє компаніям легко використовувати та змінювати код.

Бібліотека містить понад 2500 оптимізованих алгоритмів, включаючи повний набір як класичних, так і найсучасніших алгоритмів комп'ютерного бачення та машинного навчання. Ці алгоритми можна використовувати для виявлення та розпізнавання облич, ідентифікації об'єктів, класифікації дій людей у відео, відстеження рухів камери, відстеження рухомих об'єктів, вилучення *3D*-моделей об'єктів, створення *3D*-хмар точок із стереокамер, з'єднання зображень для отримання високої роздільної здатності. зображення цілої сцени, знаходити подібні зображення в базі даних зображень, видаляти червоні очі із зображень, зроблених за допомогою спалаху, стежити за рухами очей, розпізнавати пейзаж і встановлювати маркери для накладання на нього доповненої реальності тощо.

OpenCV має понад 47 тисяч користувачів спільноти, а оціночна кількість завантажень перевищує 18 мільйонів. Бібліотека широко використовується компаніями, дослідницькими групами та державними органами. Поряд із такими відомими компаніями, як *Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota*, які використовують бібліотеку, є багато стартапів, таких як *Applied Minds, VideoSurf* і *Zeitera,* які широко використовують *OpenCV.*

Застосування *OpenCV* охоплює широкий діапазон: від з'єднання зображень вуличного перегляду, виявлення вторгнень у відеоспостереження в Ізраїлі, моніторингу шахтного обладнання в Китаї, допомоги роботам у навігації та підбиранні об'єктів у *Willow Garage*, виявлення нещасних випадків утоплення в басейні в Європі, використання інтерактивного мистецтва в Іспанія та Нью-Йорк, перевірка злітно-посадкових смуг на наявність сміття в Туреччині, перевірка етикеток на продуктах на фабриках по всьому світу та швидке виявлення облич у Японії.

Застосування *OpenCV*.

Короткий огляд деяких типових завдань, які можна виконувати за допомогою *OpenCV*:

– Завантаження та відображення зображень: *OpenCV* дозволяє завантажувати зображення з файлів і відображати їх на екрані.

– Маніпулювання зображеннями: Ви можете виконувати різноманітні операції із зображеннями, як-от змінювати розмір, обрізати, повертати та гортати.

– Фільтрування та покращення зображень: *OpenCV* надає функції для застосування фільтрів до зображень, таких як розмивання, підвищення різкості та зменшення шумів. Він також підтримує вирівнювання гістограми та налаштування контрастності.

– Виявлення та розпізнавання об’єктів: *OpenCV* містить попередньо підготовлені моделі для виявлення об’єктів, розпізнавання обличчя та розпізнавання пішоходів. Ви можете використовувати ці моделі для виявлення та розпізнавання об’єктів на зображеннях або відеопотоках.

– Витяг і зіставлення функцій: *OpenCV* пропонує алгоритми вилучення функцій, такі як *Scale-Invariant Feature Transform (SIFT*) і *Speeded Up Robust Features (SURF)*. Ці функції можна використовувати для відстеження об’єктів, зшивання зображень і реєстрації зображень.

– Калібрування камери: *OpenCV* надає функції для калібрування камери, що є важливим для таких завдань, як *3D*-реконструкція та оцінка пози камери.

– Обробка відео: *OpenCV* дозволяє обробляти відео, маніпулюючи окремими кадрами, застосовуючи фільтри та виконуючи відстеження об’єктів або аналіз руху.

– Інтеграція машинного навчання: *OpenCV* легко інтегрується з такими платформами машинного навчання, як *Tensor Flow і PyTorch*, що дозволяє поєднувати завдання комп’ютерного зору з алгоритмами глибокого навчання.

Щоб використовувати *OpenCV*, вам потрібно встановити бібліотеку та налаштувати середовище розробки для вибраної мови програмування. В Інтернеті доступні численні навчальні посібники, документація та приклади коду, які допоможуть вам розпочати роботу з *OpenCV* на вашій конкретній мові програмування.

Бібліотека має інтерфейси *C++, Python, Java і MATLAB* і підтримує *Windows, Linux, Android і Mac OS*. *OpenCV* здебільшого орієнтується на програми бачення в реальному часі та використовує переваги інструкцій *MMX і SSE*, якщо вони доступні.

Переваги OpenCV:

– Відкритий вихідний код: OpenCV є відкритим вихідним кодом, тобто він є вільнодоступним і може використовуватися, змінюватися та поширюватися будь-ким.

– Широкий набір функцій: пропонує велику бібліотеку функцій комп'ютерного зору та обробки зображень, що робить його придатним для різноманітних програм.

– Кросплатформенність: OpenCV сумісний із багатьма платформами, включаючи Windows, Linux, macOS, Android та iOS.

– Підтримка спільноти: має велику й активну спільноту, яка надає підтримку, навчальні посібники та ресурси для користувачів.

– Ефективність: OpenCV написано мовою C/C++ і оптимізовано для продуктивності, що робить його ефективним для завдань у режимі реального часу та ресурсомістких завдань.

Недоліки OpenCV:

– Крива крива навчання: OpenCV може бути складним для початківців через його велику бібліотеку та різноманітні функції.

– Відсутність високорівневих абстракцій: хоча він забезпечує низькорівневий контроль, він може потребувати більше зусиль у кодуванні для певних завдань високого рівня.

– Прогалини в документації: Деякі частини OpenCV можуть мати обмежену або застарілу документацію, що може бути складним для нових користувачів.

– Змінність продуктивності: Продуктивність може відрізнятися залежно від конкретної платформи та апаратного забезпечення, що може потребувати оптимізації для різних випадків використання.

– Обмежене машинне навчання: Хоча OpenCV пропонує деякі функції машинного навчання, OpenCV не є спеціальною бібліотекою для машинного навчання та може бути не настільки повним, як спеціалізовані бібліотеки ML, як-от TensorFlow або PyTorch

3.4 *Jupyter Notebook.*

Jupyter Notebook — це веб-обчислювальна платформа, яка дозволяє розробникам кодувати, візуалізувати, ділитися та вбудовувати мультимедіа з пояснювальним текстом в один документ. Це відомий інструмент для демонстрації вашої роботи, оскільки ви можете бачити як код, так і результати в одному файлі.

Основною особливістю *Jupyter Notebook* є можливість виконувати код почергово (комірка за коміркою) та спостерігати за результатами в режимі реального часу, що робить його ідеальним для розробки, експериментів та візуалізації даних.

Jupyter Notebook дозволяє своїм користувачам завантажувати блокнот у різних форматах файлів, таких як *PDF, HTML, Python, Markdown* або *an.* файл *ipynb.*

Основні характеристики та переваги *Jupyter Notebook:*

– Інтерактивність. Надає можливість виконувати код по частинах (комірка за коміркою), перевіряти результати та виправляти помилки негайно. Це спрощує відлагодження та розробку.

– Підтримка різних мов програмування , таких як *Python, R, Julia* та інші.

– Візуалізація даних. Інтеграція з бібліотеками візуалізації даних, такими як *Matplotlib, Seaborn* та *Plotly*, дозволяє швидко створювати графіки та діаграми прямо в документі.

– Розширена підтримка тексту. Є можливість вставляти текст, форматувати його за допомогою *Markdown* або *LaTex*, що дозволяє створювати навчальні матеріали, звіти чи статті, а також розміщувати пояснення поруч із кодом.

– Легке збереження та обмін результатами. Результати виконання коду, графіки та висновки можна легко зберігати у вигляді ноутбуків, які легко обмінювати та використовувати для подальших досліджень.

Існує дві основні частини блокнота *Jupyter* : комірка *Markdown* і комірки коду.

Комірка *Markdown* використовується для пояснення роботи, надаючи зручний спосіб введення тексту та форматування його за допомогою мови розмітки. Вміння писати пояснення є важливим для науковців з області даних.

За замовчуванням всі комірки вважаються комірками коду, але їх можна легко перетворити в комірки Markdown, натиснувши клавішу *ESC* для входу в командний режим та потім клавішу *M* для перетворення. Це дозволяє легко комбінувати пояснення текстом та виконання коду в одному документі.

Комірка коду дозволяє створювати, оновлювати, читати, редагувати та видаляти код із повним підсвічуванням синтаксису та доповненням табуляції.

Стандартним ядром *Jupyter Notebook є Ipython*, який виконує код *Python*. Коли комірка коду виконує код у ній, потік переходить до ядра (*IPython* за замовчуванням), яке виконує код. Щоб виконати комірку коду, ви можете використовувати комбінацію клавіш клавіатури *SHIFT+ENTER*. Щоб установити *Jupyter Notebook* у середовищі python, просто запустіть команду *pip install jupyter*, а щоб відкрити веб-клієнт-сервер, просто введіть *jupyter notebook*. Інтерфейс платформи зображено на рис.3.1

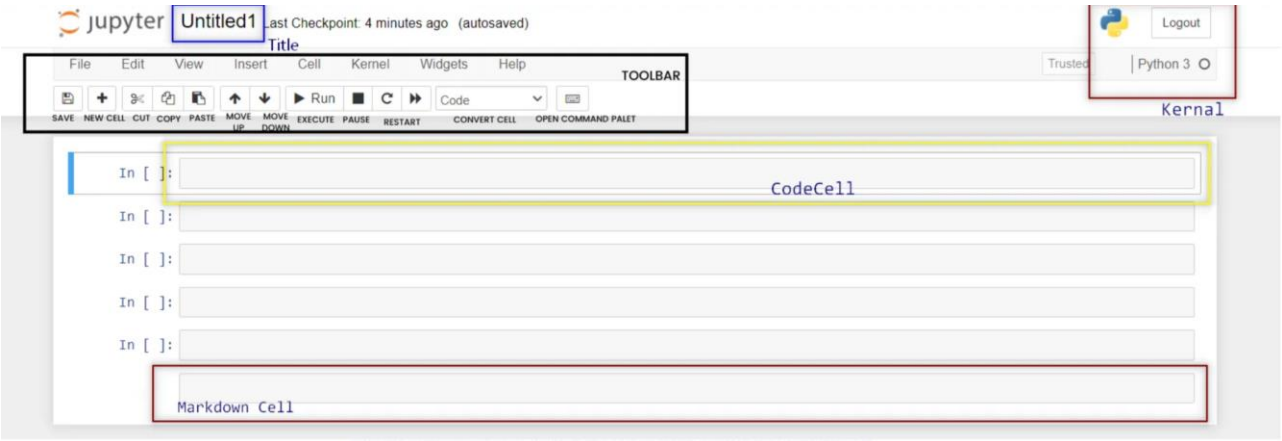


Рис.3.1 Інтерфейс платформи *Jupyter notebook*

Синій прямокутник на зображенні вище показує назву, яку можна перейменувати, клацнувши її. Жовтий прямокутник показує комірку з кодом, яку можна змінити на рамку темно-бордового кольору, схожу на комірку розцінки, за допомогою команди або за допомогою розкривного меню.

У верхньому правому куті ви можете підтвердити ядро. Під заголовком розташована панель інструментів, за допомогою якої можна виконувати кілька завдань, як-от збереження файлу, створення нової комірки, перетворення комірки коду на розмітку тощо.

Висновки за розділом

У розділі було ретельно проаналізовано та порівняно різноманітні технології та інструменти для програмної реалізації системи ідентифікації образів. Огляд почався з вибору мови програмування, і в результаті ретельного розгляду факторів, таких як універсальність, зручність та багатофункціональність, було вирішено використовувати *Python* як основну мову для реалізації проєкту.

Далі було проведено детальний аналіз трьох ключових фреймворків для машинного навчання. Виокремлено переваги кожного, а особливо гнучкість та висока продуктивність *TensorFlow*, а також простота та швидкість прототипування *Keras*. Після цього прийнято рішення використовувати обидва фреймворки для досягнення більшої ефективності в розробці моделей глибокого навчання.

Важливим аспектом стало внесення *OpenCV* до стеку технологій для обробки та аналізу зображень. Це рішення допоможе розширити можливості проєкту в галузі комп'ютерного зору та опрацювання візуальних даних.

Завершальним пунктом вибору технологій став вибір *Jupyter Notebook* для проведення експериментів та візуалізації результатів. Це забезпечить зручний та інтерактивний підхід до роботи з кодом та допоможе ефективно аналізувати та представляти дані.

РОЗДІЛ 4. ОСНОВНІ ЕТАПИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ОБРАЗІВ

4.1 Підготовка навчального набору даних.

Першим етапом у реалізації системи розпізнавання образів є створення навчального набору даних. Для цього використовувалися зображення зібрані самостійно, понад 100 фото для більш точного результату.

Зображення були анотовані, тобто позначені мітками, що відповідають класам об'єктів для подальшого навчання моделі.

Анотація даних – це процес додавання метаданих до ваших даних, наприклад зображень, аудіо чи тексту, щоб полегшити розуміння алгоритмами машинного навчання. Зазвичай це робиться шляхом вручну позначення даних описовими тегами, анотаціями або обмежувальними рамками. Анотація даних є важливою для керованих проектів машинного навчання. Без точного та послідовного маркування моделі не можуть навчатися та робити прогнози. Це може бути нудним і трудомістким завданням, але, на щастя, є такі інструменти, як *LabelImg*, які допоможуть.

LabelImg — це безкоштовний графічний інструмент анотації зображень із відкритим вихідним кодом, який дає змогу коментувати рамки обмеження об’єктів на зображеннях (рис.4.1). *LabelImg* підтримує різні формати анотацій, включаючи *PascalVOC i YOLO*, і може експортувати анотації у форматах *XML, CSV i TXT*.

Підтримка цих форматів, які зазвичай використовуються в конвеєрах виявлення об’єктів, робить його корисним інструментом для анотування даних для виявлення об’єктів.

LabelImg написаний на Python і використовує Qt для свого графічного інтерфейсу, що робить його чудовим вибором для систем на базі Linux, які багато програмного забезпечення для анотацій не підтримують. Крім того, для Windows LabelImg надає окрему програму, яка не потребує інсталяції та має розмір трохи більше 13 МБ.

LabelImg надає гарячі клавіші для швидкої навігації та анотації кількох зображень.

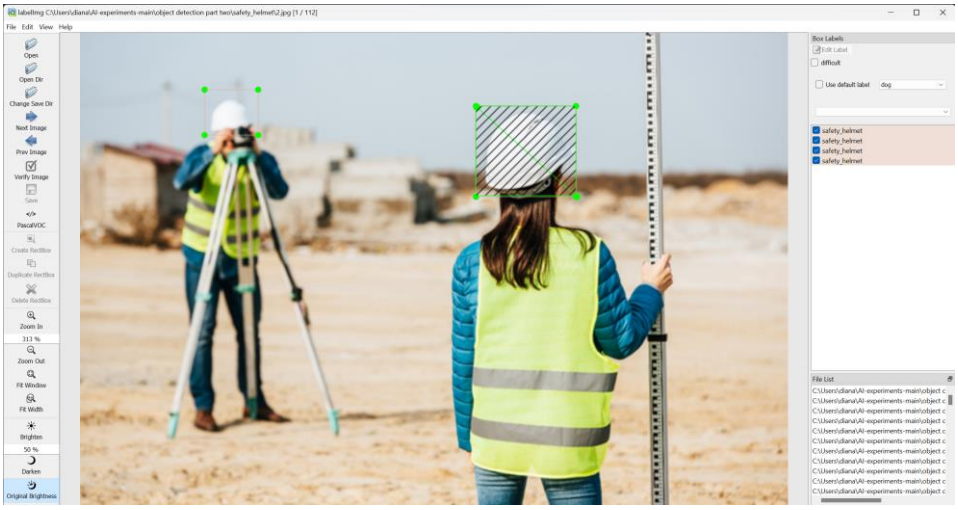


Рис.4.1 Інтерфейс *Labelimg*

Кожен об’єкт був самостійно анотований (створення обмежувальної рамки) в кожному окремому кадрі. В результаті отримали файли у форматі *XML* , що представляють собою анотацію (мітку) для зображення та містять інформацію про об'єкти, які були марковані на цьому зображенні.(див.рис.4.2)

Файл Редагувати Переглянути

```
<annotation>
        <folder>safety_helmet</folder>
        <filename>113.jpg</filename>
        <path>C:\Users\diana\AI-experiments-main\object detection part two\safety_helmet\113.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>608</width>
                <height>405</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>safety_helmet</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>198</xmin>
                        <ymin>20</ymin>
                        <xmax>334</xmax>
                        <ymax>113</ymax>
                </bndbox>
```

Рис.4.2 Вміст файл *XML* з мітками

Це стандартний формат анотацій для об'єктів на зображеннях, і часто використовується для навчання моделей машинного навчання, зокрема для областей об'єктів *(object detection).*

4.2 Створення файлу *TFRecord*

Формат *TFRecord* використовується в бібліотеці *TensorFlow* для ефективного зберігання та обробки великої кількості даних при навчанні мережі глибокого навчання. *TFRecord* - це формат для зберігання послідовностей даних у бінарному форматі. Використання *TFRecord* може мати кілька переваг.

1. Швидший час читання даних: Формат *TFRecord* дозволяє ефективно зчитувати дані партіями *(batch)* і уникати витрат часу на розпакування та конвертацію даних з текстових форматів під час навчання.

2. Менше місця для зберігання: Дані зберігаються у бінарному форматі, що може призвести до меншого обсягу даних на диску порівняно з текстовими форматами.

3. Легше змішування різних типів даних: В одному файлі *TFRecord* можна зберігати дані різних типів, таких як зображення, текст чи числа, і легко обробляти їх під час навчання.

4. Інтеграція з *TensorFlow: TFRecord* добре інтегрується з функціями *TensorFlow*, що робить його зручним для використання в тензорних потоках та інших компонентах бібліотеки.

При навчанні мережі важливо ефективно управляти великими обсягами даних, і *TFRecord* надає зручний та ефективний механізм для цього.

Лістинг коду:

```
import tensorflow as tf
import xml.etree.ElementTree as ET
from os import listdir
from os.path import isfile, join
# Задаємо цільове значення кількості об'єктів на зображенні
goal = 10
# Шлях до папки з XML файлами та зображеннями
fn = "C:\\Users\\diana\\AI-experiments-main\\object detection part two\\safety_helmet\\"
```

```
# Створюємо список усіх XML файлів у папці
p = [fn + '/' + f for f in listdir(fn) if isfile(join(fn, f)) and f[-1] == 'l']
print(p[:5])
# Функція для завантаження та обробки зображення
def load_img(img):
    img = tf.io.read_file(img)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.cast(img, tf.float32) / 256
    img = tf.image.resize(img, (128, 128))
    return img
# Створюємо запис у TFRecord файлі
writer = tf.io.TFRecordWriter('bounding_box_dataset.tfrecord')
# Проходимо по кожному XML файлу
for xml in p:
    tree = ET.parse(xml)  # Парсимо XML файл
    root = tree.getroot()  # Отримуємо кореневий елемент
        # Визначаємо ширину та висоту зображення
    w = int(root[4][0].text)
    h = int(root[4][1].text)
      # Створюємо список позицій об'єктів, які будуть включені у вибірку
    positions = []
    p = 0
    while len(positions) < goal:
      positions.append(p)
      p += 1
      if p == num_objects:
        p = 0
       # Збираємо координати об'єктів
    cords = []
    for num in positions:
```

```
            object_cords = [
                int(root[num + 6][4][0].text) / w * 2 - 1,  # x_min
                int(root[num + 6][4][1].text) / h * 2 - 1,  # y_min
                int(root[num + 6][4][2].text) / w * 2 - 1,  # x_max
                int(root[num + 6][4][3].text) / h * 2 - 1   # y_max
            ]
            cords.append(object_cords)
        img = load_img(root[2].text)
        # Перетворюємо дані в байтовий формат
        serialized_img = tf.io.serialize_tensor(img).numpy()
        serialized_cords = tf.io.serialize_tensor(cords).numpy()
        # Створюємо об'єкт Example та записуємо його у TFRecord файл
        example = tf.train.Example(features=tf.train.Features(feature={
            'img': tf.train.Feature(bytes_list=tf.train.BytesList(value=[serialized_img])),
            'cords':
tf.train.Feature(bytes_list=tf.train.BytesList(value=[serialized_cords]))
        }))
        writer.write(example.SerializeToString())
    writer.close()
```

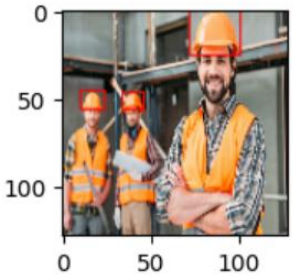
Для перевірки правильності запису файлу tfrecord та підготовки даних до навчання виконуємо наступний код.

Лістинг коду:

```
# прочитаємо запис
dataset = tf.data.TFRecordDataset('bounding_box_dataset.tfrecord')
# функція для розбору запису
def parse_record(record):
    # необхідно описати екземпляр, що надходить
    # імена елементів, як при записі
```

```
        feature_description = {
            'img': tf.io.FixedLenFeature([], tf.string),
            'cords': tf.io.FixedLenFeature([], tf.string)
        }
        parsed_record = tf.io.parse_single_example(record, feature_description)
        img = tf.io.parse_tensor(parsed_record['img'], out_type=tf.float32)
        cords = tf.io.parse_tensor(parsed_record['cords'], out_type=tf.float32)
        return img, cords
    # пройдемося по записам і розпакуємо їх
    dataset = dataset.map(parse_record)
    # ще раз перевіримо
    for i, c in dataset.take(1):
        plt.figure(figsize=(10, 6))
        ax = plt.subplot(3, 1, 1)
        i = i.numpy()
        c = c.numpy()
        c = (c + 1) / 2 * 128  # знову з -1...1 до 0...64
        c = c.astype(np.int16)  # для opencv
        for bb in c:
            i = cv2.rectangle(i, (bb[0], bb[1]), (bb[2], bb[3]), (1, 0, 0), 1)
        plt.imshow(i)
        plt.show()
        print(c)
    # оптимізуємо швидкість доступу до даних та попереднє завантаження
    dataset =
dataset.cache().prefetch(buffer_size=tf.data.AUTOTUNE).batch(32).shuffle(40)
```

Результат компіляції зображено на рисунку 4.3.




```
[[ 72   0 101  25]
 [ 34  46  46  56]
 [ 10  45  24  56]
 [ 72   0 101  25]
 [ 34  46  46  56]
 [ 10  45  24  56]
 [ 72   0 101  25]
 [ 34  46  46  56]
 [ 10  45  24  56]
 [ 72   0 101  25]]
```

Рис.4.3 Результат компіляції

Код успішно обробляє *TFRecord*-датасет, витягує зображення та координати *bounding boxes*, візуалізує їх та готує для тренування, використовуючи *TensorFlow* та бібліотеку *OpenCV*.

4.3 Написання власної нейронної мережі

Основні компоненти та їх визначення.

– Input((128, 128, 3)): Визначає вхідні дані розміром 128x128 пікселів та 3 канали *(RGB).*

– *Conv2D(...):* Конволюційні шари, які використовуються для взяття фільтрів та здійснення згортки зображення.

– *Flatten():* Перетворення фінального блоку конволюційних шарів у вектор.

– *Dropout(0.2):* Dropout шар для регуляризації та запобігання перенавчанню.

– *Dense(...):* Повнозв'язні шари, які здійснюють зв'язки між входами та виходами нейромережі.

– *Model(inputs, outputs):* Створення моделі, яка використовує визначені вхідні та вихідні дані.

Лістинг коду:

```
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten, Input, Conv2D, Conv2DTranspose, Concatenate, LeakyReLU, Dropout
    # Визначаємо вхідні дані розміром (128, 128, 3)
inputs = Input((128, 128, 3))
    # Створюємо блок конволюційних шарів
x = Conv2D(32, 3, activation='relu', padding='same')(inputs)
x = Conv2D(64, 3, activation='relu', padding='same', strides=2)(x)
x = Conv2D(64, 3, activation='relu', padding='same')(x)
x = Conv2D(64, 3, activation='relu', padding='same', strides=2)(x)
x = Conv2D(128, 3, activation='relu', padding='same')(x)
x = Conv2D(128, 3, activation='relu', padding='same', strides=2)(x)
x = Conv2D(128, 3, activation='relu', padding='same')(x)
x = Conv2D(256, 3, activation='relu', padding='same', strides=2)(x)
x = Conv2D(256, 3, activation='relu', padding='same')(x)
    # Перетворюємо фінальний блок у вектор за допомогою Flatten
x = Flatten()(x)
    # Додаємо Dropout для регуляризації
x = Dropout(0.2)(x)
    # Додавання повнозв'язного шару з активацією ReLU
x = Dense(256, activation='relu')(x)
    # Фінальний повнозв'язний шар без активації для виведення регресії (bounding box coordinates)
x = Dense(30)(x)  # 3 координати * 10 об'єктів = 30 виходів
    # Визначаємо вихід моделі
outputs = x
    # Створюємо модель, використовуючи визначені вхідні та вихідні дані
boxregressor = keras.Model(inputs, outputs)
```

Ця нейромережа призначена для регресії координат *bounding box* для 10 об'єктів на зображенні розміром 128x128 пікселів.

Регресія координат у контексті задачі об'єктного виявлення означає навчання моделі прогнозувати координати межі області об'єкта (*bounding box*) на зображенні. Тобто, замість того, щоб класифікувати області як конкретні класи об'єктів, модель спробує прогнозувати координати прямокутника (або іншої форми), який найкраще описує положення об'єкта на зображенні.

У багатьох випадках координати області об'єкта представлені чотирма значеннями: (*x_min, y_min, x_max, y_max).*

Де *(x_min, y_min)* - координати верхнього лівого кута прямокутника, а *(x_max, y_max)* - координати його нижнього правого кута.

Під час навчання модель отримує на вході зображення та виводить прогнозовані координати *bounding box* для об'єкта на цьому зображенні.

Після тренування, коли модель вивчила залежності між пікселями зображення та положенням об'єкта, вона може використовуватися для автоматичного виявлення та обведення об'єктів на нових, раніше невиданих зображеннях.

Виведемо граф написаної мережі, код для виведення зображено на рисунку 4.4.

```
import os

# Вказати шлях до виконуваного файлу dot.exe в Graphviz
graphviz_path = 'C:/Downloads Apps/Graphviz/bin/'
os.environ["PATH"] += os.pathsep + graphviz_path

# Намалювати граф моделі та зберегти його у файл
plot_model(boxregressor, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

Рис.4.4 Код для виведення архітектури у вигляді графу.

Архітектура нейронної мережі зображена на рисунку 4.5.

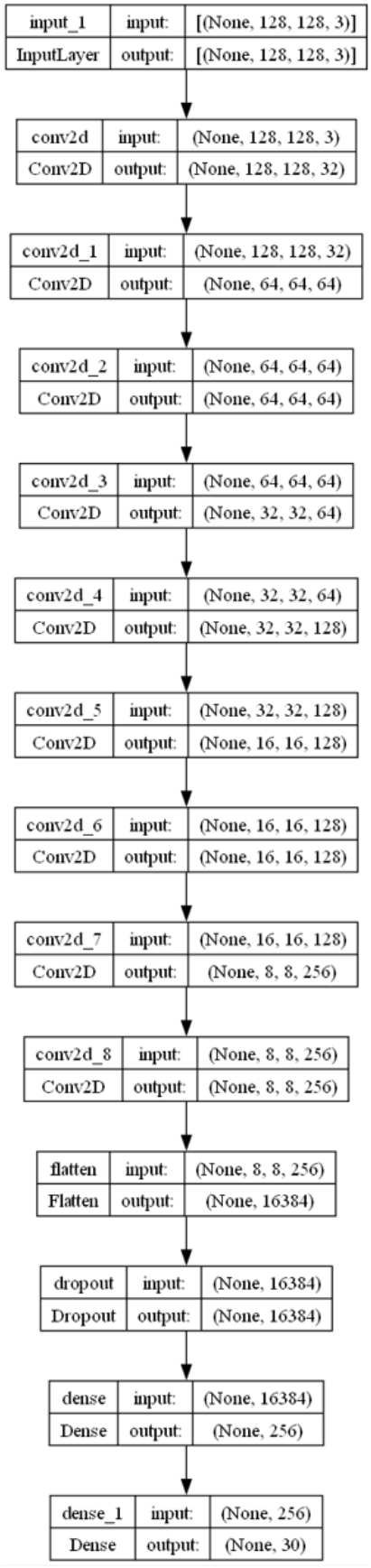


Рис.4.5 Архітектура створеної нейромережі

4.4 Розрахунок функції втрат *IoU Loss*

Intersection over Union (IOU) — це показник продуктивності, який використовується для оцінки точності анотацій, сегментації та алгоритмів виявлення об’єктів. Він кількісно визначає перекриття між прогнозованою обмежувальною рамкою або сегментованою областю та обмежувальною рамкою істини землі або анотованою областю з набору даних.

IoU визначає, наскільки добре прогнозований об’єкт узгоджується з анотацією фактичного об’єкта, що дозволяє оцінити точність моделі та точно налаштувати алгоритми для покращення результатів

Loss IoU працює лише тоді, коли передбачені обмежувальні прямокутники накладаються на базову рамку правди. *Loss IoU* не забезпечить жодного рухливого градієнта для випадків, що не перекриваються. *IOU* обчислюється шляхом ділення площі перетину між прогнозованою та наземною областями істинності на площу їх об’єднання. Формула *IOU* може бути виражена таким чином: *IOU* = Площа перетину / Площа об'єднання (рис.4.6).

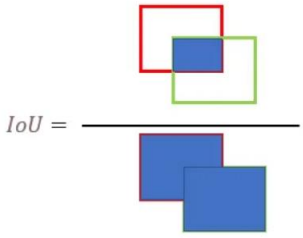


Рис.4.6 Функція втрат *Iou Loss*

Червоний — прогнозована обмежувальна рамка, а зелений — обмежувальна рамка правдивості. Більше значення *IOU* вказує на краще узгодження між прогнозованими та фактичними регіонами, що відображає точнішу модель.

Лістинг коду:

```
def IoU_Loss(true, pred):
    # Розбиваємо тензори на компоненти для координат bounding box
  minx1, miny1, maxx1, maxy1 = tf.split(true, 4, axis=2)
  fminx, miny2, fmaxx = tf.split(pred, 3, axis=2)
```

```
        # Обчислюємо min та max для предсказаних bounding box
    minx2 = tf.minimum(fminx, fmaxx)
    maxx2 = tf.maximum(fminx, fmaxx)
        # Обчислюємо delta (відстань між min та max)
    delta = maxx2 - minx2
    maxy2 = miny2 + delta
    intersection = 0.0
        # Обчислюємо перетин для кожної пари bounding box
    for i1 in range(10):
       for i2 in range(10):
           x_overlap = tf.maximum(0.0, tf.minimum(maxx1[:, i1], maxx2[:, i2]) -
tf.maximum(minx1[:, i1], minx2[:, i2]))
           y_overlap = tf.maximum(0.0, tf.minimum(maxy1[:, i1], maxy2[:, i2]) -
tf.maximum(miny1[:, i1], miny2[:, i2]))
           intersection += x_overlap * y_overlap

        # Обчислюємо "хитрості" для об'єднання та площі bounding box
    beta1 = 0.0
    for i1 in range(10):
       for i2 in range(10):
           x_overlap = tf.maximum(0.0, tf.minimum(maxx1[:, i1], maxx1[:, i2]) -
tf.maximum(minx1[:, i1], minx1[:, i2]))
           y_overlap = tf.maximum(0.0, tf.minimum(maxy1[:, i1], maxy1[:, i2]) -
tf.maximum(miny1[:, i1], miny1[:, i2]))
           if i1 == i2:
              beta1 += (x_overlap * y_overlap) ** 2
           else:
              beta1 += x_overlap * y_overlap
    beta2 = 0.0
    for i1 in range(10):
```

```
        for i2 in range(10):
            x_overlap = tf.maximum(0.0, tf.minimum(maxx2[:, i1], maxx2[:, i2]) - tf.maximum(minx2[:, i1], minx2[:, i2]))
            y_overlap = tf.maximum(0.0, tf.minimum(maxy2[:, i1], maxy2[:, i2]) - tf.maximum(miny2[:, i1], miny2[:, i2]))
            if i1 == i2:
                beta2 += (x_overlap * y_overlap) ** 2
            else:
                beta2 += x_overlap * y_overlap
        # Обчислюємо втрату за допомогою квадратичної різниці площ та перетину
    loss = (beta1 - beta2) ** 2 - intersection
    return loss
```

4.5 Реалізація моделі для завдання об’єктного виявлення.

Створення класу *Model* є ключовим компонентом для тренування моделі регресії *bounding box* на зображеннях у завданні об'єктного виявлення. Його основна мета - керувати процесом навчання, оновлюючи параметри моделі з кожним кроком. Використовуючи зворотне поширення та оптимізатор Adam, клас визначає метод training_step, в якому викликається модель для отримання прогнозів та обчислення втрат.

Особливість класу в тому, що він використовує експоненційний розпад швидкості навчання, що дозволяє динамічно адаптувати швидкість навчання протягом процесу тренування. Це може сприяти більш ефективному навчанню моделі, особливо коли виникає необхідність у більш чутливому контролі за швидкістю збіжності.

Створення екземпляру класу *Model* із підготованою моделлю для регресії *bounding box (nn_box)* дозволяє легко управляти та конфігурувати процес тренування моделі, використовуючи інтерфейс цього керівного класу.

Лістинг коду:

```
class Model(tf.keras.Model):
    def __init__(self, nn_box):
        super(Model, self).__init__()
        self.nn_box = nn_box
          # Змінення швидкості навчання
        initial_learning_rate = 3e-4
        lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate, decay_steps=10000, decay_rate=0.9, staircase=True
        )
        self.box_optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
    @tf.function
    def training_step(self, x, true_boxes):
        with tf.GradientTape() as tape_box:
            # Викликаємо модель для отримання прогнозів
            pred = self.nn_box(x, training=True)
            pred = tf.reshape(pred, [-1, 10, 3])  # Перетворення форми відповіді моделі
            # Обчислюємо втрату за допомогою функції втрати IoU_Loss
            loss = IoU_Loss(true_boxes, pred)
        # Застосовуємо зворотнє поширення градієнтів та оптимізуємо ваги моделі
        grads = tape_box.gradient(loss, self.nn_box.trainable_variables)
        self.box_optimizer.apply_gradients(zip(grads, self.nn_box.trainable_variables))
        return loss
# Створюємо екземпляр класу Model, передаючи йому підготовану модель
boxregressor
model = Model(boxregressor)
```

Для демонстрації роботи моделі на одному пакеті даних і виведення середньої втрати для цього пакету використаємо наступну команду зображену на рисунку 4.7.

```
[81]: for i, c in dataset.take(1):
          print(tf.reduce_mean(model.training_step(i, c)))

tf.Tensor(5.3548627, shape=(), dtype=float32)
```

Рис.4.7. Код для демонстрації роботи моделі.

dataset.take(1): Бере один пакет даних (i - зображення, c - координати *bounding box) з дамасету dataset. take(1)* вказує, що береться тільки один елемент з датасету.

model.training_step(i, c): Викликає метод *training_step* класу *Model* для одного кроку навчання з вхідними даними i (зображення) та з (координати *bounding box*).

tf.reduce_mean(...): Обчислює середнє значення втрат для отриманого пакету даних. Зазвичай втрати обчислюються на рівні окремих зразків, і *reduce_mean* використовується для обчислення середньої втрати на всьому пакеті.

print(...): Виводить обчислене середнє значення втрати.

Цей фрагмент може використовуватися для моніторингу якості моделі на одному пакеті даних під час тренування. Середнє значення втрати є показником того, наскільки добре модель вирішує завдання для даного пакету.

4.6 Тренування нейронної мережі.

Основні компоненти що використовуються для тренування нейромережі:

Цикл навчання *«for epoch in range(1, epochs + 1)»* запускає цикл для кожної епохи тренування. Цикл пакетів даних *«for step, (i, c) in enumerate(dataset)»:* внутрішній цикл ітерується по пакетах даних у датасеті. *«i»* представляє зображення, а c - координати *bounding box* для кожного зображення у пакеті.

Виклик методу тренування: *«loss += tf.reduce_mean(model.training_step(i, c))»* викликає метод *training_step* класу *Model* для обчислення втрат і додає їх до загальних втрат для поточної епохи. *«lc += 1»* збільшує лічильник кількості кроків для обчислення середніх втрат.

Оновлення виведення: *«clear_output(wait=True)»* видаляє попередні виведення з екрану, щоб можна було відобразити нову інформацію. *«print(epoch)»* виводить номер поточної епохи.

Моніторинг швидкості навчання: *«current_learning_rate = model.box_optimizer.lr.numpy()»* отримує поточне значення швидкості навчання. *« hist = np.append(hist, current_learning_rate)»* додає це значення до історії швидкостей навчання.

Лістинг коду :

```
from IPython.display import clear_output
import matplotlib.pyplot as plt
    # Ініціалізація пустого масиву для збереження значень швидкості навчання
hist = np.array(np.empty([0]))
epochs = 1000  # Кількість епох
for epoch in range(1, epochs + 1):
  loss = 0
  lc = 0
      # Цикл по пакетам даних у датасеті
  for step, (i, c) in enumerate(dataset):
      # Обчислення втрат та їх додавання до загальних втрат для поточної епохи
    loss += tf.reduce_mean(model.training_step(i, c))
    lc += 1
      # Виведення номеру поточної епохи
  clear_output(wait=True)
  print(epoch)
      # Збереження значень швидкості навчання
  current_learning_rate = model.box_optimizer.lr.numpy()
  hist = np.append(hist, current_learning_rate
      # Візуалізація графіку швидкості навчання
  plt.plot(np.arange(0, len(hist)), hist)
  plt.show()
testing(
```

Результат тренування нейронної мережі зображено на рисунку 4.8.

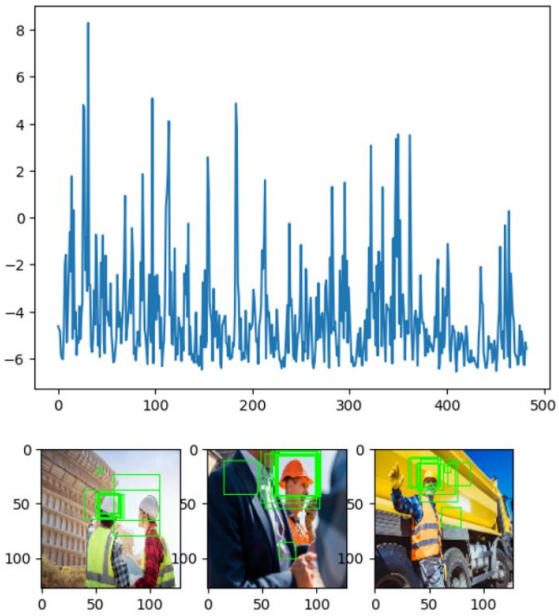


Рис.4.8.Результат тренування нейронної мережі.

Зменшення значень функції втрат на графіку помилок свідчить про те, що модель здатна зменшувати відхилення між прогнозами і справжніми значеннями координат рамок. Модель навчається і показує покращення у виявленні та локалізації об'єктів на зображеннях. Для більш точних результатів варто збільшити обсяг тренувального набору даних шляхом додавання нових прикладів чи розширення існуючих зображень для покращення роботи моделі на різних умовах зйомки.

Також шляхом збільшення епох можна досягнути покращення результатів, проте процес тренування буде значно довше. Порівняємо результати тестування до тренування нейромережі до та після 1000 епох. (рис.4.9)

Рис.4.9 Порівняння результатів до та після тренування.

Після тренування нейромережі для завдання виявлення та локалізації об'єктів, результати демонструють успішність у прогнозуванні координат рамок на зображеннях. Візуалізація п'яти прикладів з тестового набору даних показує, що модель ефективно локалізує об'єкти, відображаючи прямокутники, що охоплюють ці об'єкти.

Кожен прогноз відображається на оригінальному зображенні, можна зробити висновки, що модель навчилася визначати положення об'єктів з досить гарною точністю.

4.7 Реалізація моделі класифікатора.

Для поліпшення системи ідентифікації об'єктів та зменшення видимості зайвих рамок можна розглянути використання моделі класифікації, яка визначає клас об'єкту в кожній рамці. Після цього можна об'єднати цю модель класифікації з моделлю для регресії координат, яка визначає розташування рамок.

Для вирішення цієї задачі як і в попередній моделі реалізуємо код, для створення файлу *tfrecord* для класифікатора.

Лістинг коду:

```
import tensorflow as tf
import xml.etree.ElementTree as ET
from os import listdir
from os.path import isfile, join
import random
# Створення простору імен для міток класів
namespace = {'NOTHING': 0, 'safety_helmet': 1 }
# Шлях до папки з XML-файлами та зображеннями
fn = "C:\\Users\\diana\\AI-experiments-main\\object detection part two\\safety_helmet\\"
# Формування списку усіх XML-файлів у папці
p = [fn + '/' + f for f in listdir(fn) if isfile(join(fn, f)) and f[-1] == 'l']
# Функція для завантаження та обробки зображень
def load_img(img):
```

```
    img = tf.io.read_file(img)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.cast(img, tf.float32)/256
    img = tf.image.resize(img, (1024, 1024))
    return img
# Створення TFRecordWriter для збереження даних
writer = tf.io.TFRecordWriter('classifier_dataset.tfrecord')
# Функція для збереження зображень та міток у форматі TFRecord
def saveinrecord(img, name):
    # Підготовка даних та запис у файл TFRecord
    serialized_img = tf.io.serialize_tensor(img).numpy()
    serialized_name = tf.io.serialize_tensor(name).numpy()
    # Створення екземпляру tf.train.Example
    example = tf.train.Example(features=tf.train.Features(feature={
        'img': tf.train.Feature(bytes_list=tf.train.BytesList(value=[serialized_img])),
        'name': tf.train.Feature(bytes_list=tf.train.BytesList(value=[serialized_name]))
    }))
    # Запис у файл TFRecord
    writer.write(example.SerializeToString()
# Цикл для обробки кожного XML-файлу
for xml in p:
    # Обробка кожного XML-файлу
    tree = ET.parse(xml)  # Адреса файлу
    root = tree.getroot()  # Парсинг XML
    # Кількість об'єктів (з областями інтересу)
    num_objects = len(root) - 6
    # Розміри зображення
    w = int(root[4][0].text)
    h = int(root[4][1].text)
    # Завантаження та обробка зображення
```

```
    img = load_img(root[2].text)

    # Цикл для обробки кожного об'єкта
    for num in range(num_objects):
        # Отримання координат області інтересу та обробка зображення
        xmin = tf.clip_by_value(int(int(root[num + 6][4][0].text)/w*1024), 0, 1024)
        ymin = tf.clip_by_value(int(int(root[num + 6][4][1].text)/h*1024), 0, 1024)
        xmax = tf.clip_by_value(int(int(root[num + 6][4][2].text)/w*1024), 0, 1024)
        ymax = tf.clip_by_value(int(int(root[num + 6][4][3].text)/h*1024), 0, 1024)
        # Обрізка та зміна розмірів області інтересу
        offset_height = ymin
        offset_width = xmin
        target_height = ymax - ymin
        target_width = xmax - xmin
        cropped = tf.image.crop_to_bounding_box(img, offset_height, offset_width,
target_height, target_width)
        cropped = tf.image.resize(cropped, (128, 128))
        # Мітка класу
        name = namespace[root[num + 6][0].text]
        # Збереження в TFRecord
        saveinrecord(cropped, name)
    # Створення та збереження фонових областей
    counter = 0
    goal = 5

    while counter < goal:
        # Генерація випадкових координат та розмірів фонової області
        gxmin = random.randint(1, 900)
        gymin = random.randint(1, 900)
        gxsize = random.randint(10, 100)
```

```
        gysize = random.randint(10, 100)
        gxmax = gxmin + gxsize
        gymax = gymin + gysize
        # Перевірка перетину з реальною областю інтересу
        notintersect = True
        for num in range(num_objects):
            xmin = tf.clip_by_value(int(int(root[num + 6][4][0].text)/w*1024), 0, 1024)
            ymin = tf.clip_by_value(int(int(root[num + 6][4][1].text)/h*1024), 0, 1024)
            xmax = tf.clip_by_value(int(int(root[num + 6][4][2].text)/w*1024), 0, 1024)
            ymax = tf.clip_by_value(int(int(root[num + 6][4][3].text)/h*1024), 0, 1024)
            # Перевірка перетину координат
            x_overlap = tf.maximum(0, tf.minimum(gxmax, xmax) - tf.maximum(gxmin, xmin))
            y_overlap = tf.maximum(0, tf.minimum(gymax, ymax) - tf.maximum(gymin, ymin))
            if x_overlap > 0 and y_overlap > 0:
                notintersect = False
                break
        # Збереження фонової області, якщо немає перетину
        if notintersect:
            cropped = tf.image.crop_to_bounding_box(img, gymin, gxmin, gysize, gxsize)
            cropped = tf.image.resize(cropped, (128, 128))
            name = 0
            saveinrecord(cropped, name)
            counter += 1
# Закриття TFRecordWriter
writer.close()
```

Для цієї задачі було використано попередньо навчену нейромережу як базову модель (відому як base_model), яка призначена для взяття призначених для завдань зображень та вилучення важливих функцій з них. Зазвичай ці базові моделі попередньо навчаються на великих наборах даних, таких як *ImageNet*, і мають здатність добре узагальнювати функції.

Лістинг коду:

```
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten, Input, Conv2D,
GlobalAveragePooling2D, Dropout
inputs = Input((128,128,3))
x = base_model(inputs)
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation = 'relu')(x)
x = Dropout(0.2)(x)
x = Dense(3, activation = 'softmax')(x)
outputs = x
classifier = keras.Model(inputs, outputs)
```

Тестування моделі класифікатора зображено на рис. 4.10

Рис.4.10 Тестування класифікатора.

Архітектура моделі класифікатора зображена на рисунку 4.11.

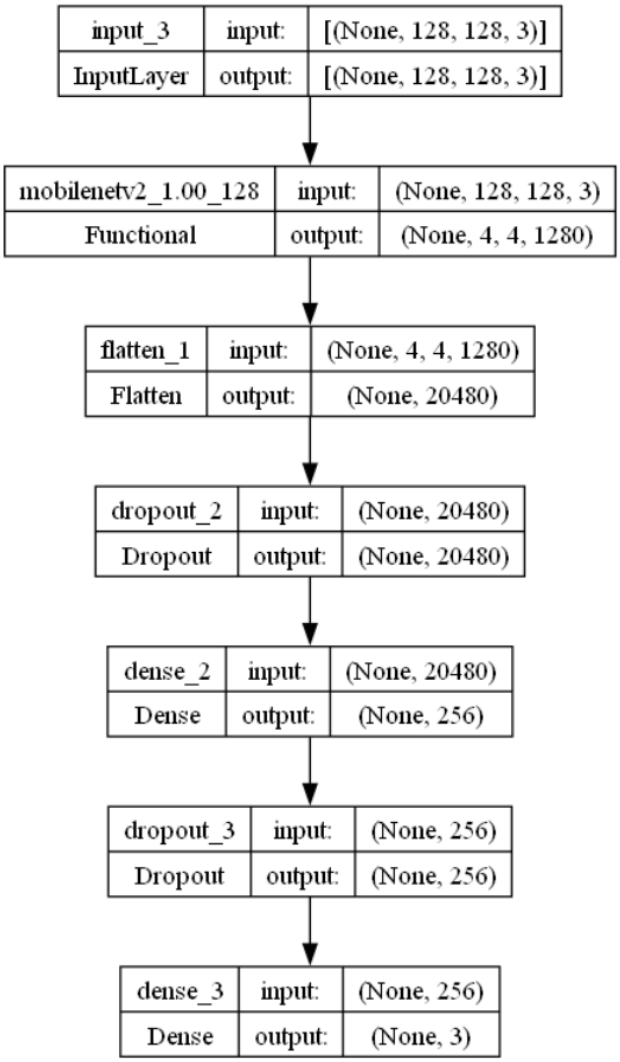


Рис.4.11 Архітектура моделі класифікатора

Класифікатор, навчений для розпізнавання класів на зображеннях, демонструє добрі результати. Під час тестування зображень з датасету він визначає класи з високою ймовірністю, що свідчить про його ефективність.

Модель правильно класифікує зображення. Додатково, виведення зображень разом із передбаченнями та ймовірностями допомагає візуалізувати результати роботи класифікатора, дозволяючи легко спостерігати за точністю його передбачень.

4.8 Тестування системи ідентифікації образів.

Тестування системи ідентифікації образів виконується шляхом об'єднання результатів двох моделей. Спочатку використовується перша модель для локалізації об'єктів на зображенні, а отримані координати обробляються для визначення регіонів інтересу.

Далі ці регіони проходять через другу модель — класифікатор, який визначає класи об'єктів та повертає ймовірності. Подальше об'єднання результатів двох моделей надає системі повну інформацію про локалізацію та класифікацію об'єктів на зображенні, що робить систему більш ефективною в завданні ідентифікації образів.

Тестування системи було проведено у двох режимах: перш за все, на статичних зображеннях, щоб оцінити здатність системи ідентифікації до коректного визначення об'єктів на фотографіях , а також на відео, щоб визначити її ефективність у реальному часі та здатність працювати з послідовними зображеннями.(див.рис.4.11-4.14)

Рис.4.11. Тестування системи (зображення) .

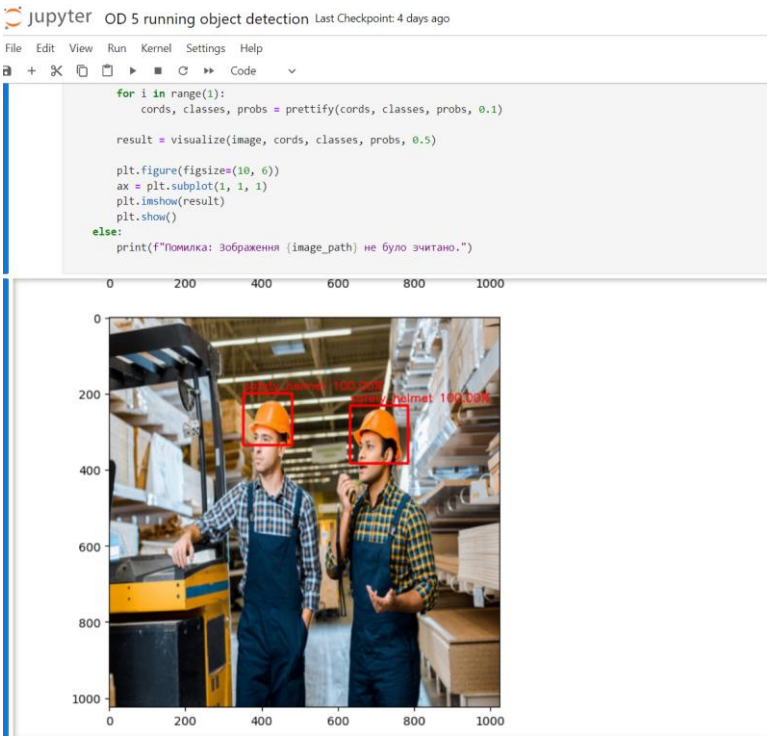


Рис.4.12. Тестування системи (зображення) .

Рис.4.13. Тестування системи(зображення) .

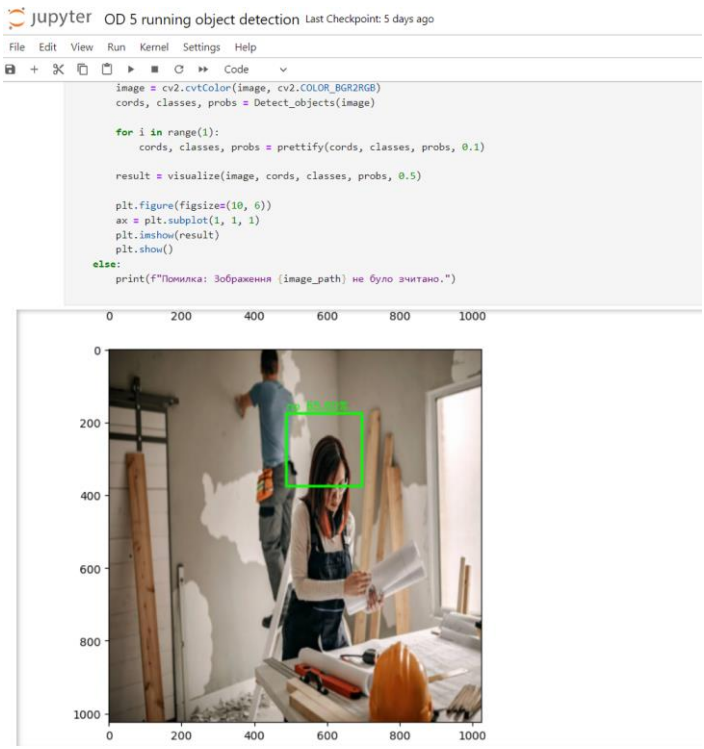


Рис.4.14. Тестування системи (зображення).

В результаті тестування системи ідентифікації зображень було виявлено високий рівень ефективності, навіть в умовах великої кількості шуму на зображеннях. Система успішно розпізнала всі об'єкти, незважаючи на їхню віддаленість та низьку якість наявних зображень. Отже, можна сказати що нейромережа тепер знає дн

Важливо відзначити, що навіть при наявності різної кількості об'єктів на кожному фото, система здатна точно визначати їхні характеристики та локалізувати на зображенні. Це свідчить про високу стійкість та надійність системи в умовах складних сценаріїв, що включають в себе різноманітні ускладнення, такі як шуми та обмежені умови зйомки.

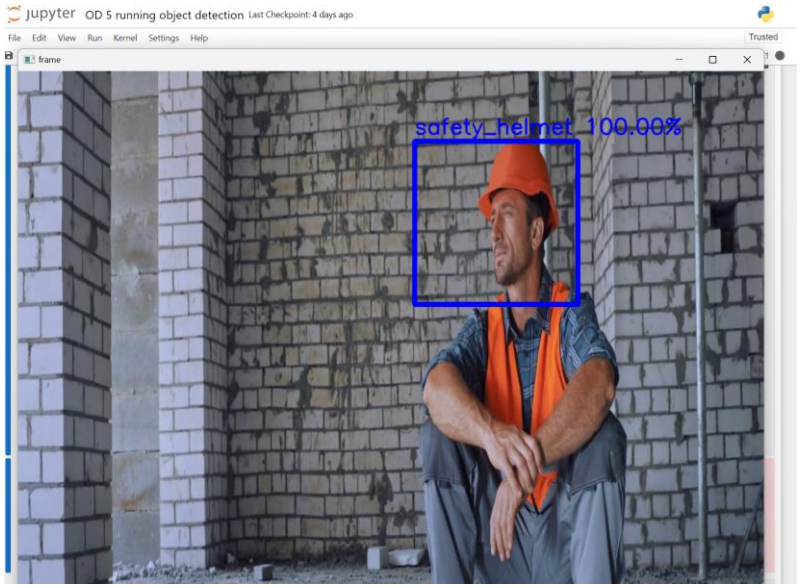


Рис.4.12. Тестування системи (відео).

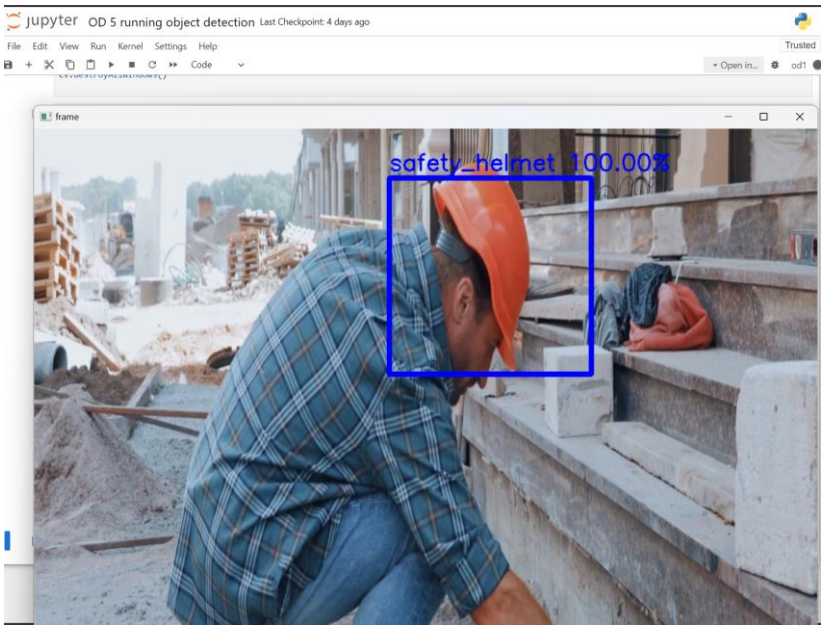


Рис.4.13. Тестування системи (відео)

Висновки за розділом

Цей розділ присвячений практичному аспекту дослідження, я детально описала етапи розробки програмного забезпечення для системи розпізнавання образів. Почавши з підготовки навчального набору даних, я виконала збір та обробку вхідних даних, ключових для ефективного навчання нейронної мережі.

Далі був розглянутий процес створення файлу TFRecord як оптимізованого формату для швидкого та ефективного читання даних під час тренування моделі.

Важливим етапом було написання власної нейронної мережі, спеціально адаптованої для розпізнавання будівельних касок на зображеннях.

Зокрема, був розглянутий розрахунок функції втрат IoU Loss, визначаючої точність розташування об'єктів. Далі подано реалізацію моделі для завдання об’єктного виявлення, що визначає положення та форму будівельних касок.

До складу роботи також увійшло тренування розробленої нейронної мережі на підготовленому навчальному наборі даних, а також реалізація моделі класифікатора для ідентифікації різних класів об’єктів.

В результаті вдалої реалізації була створена система, спрямована на розпізнавання будівельних касок на зображеннях у сфері будівництва, що сприяє забезпеченню безпекових норм та стандартів. Проведені тести підтвердили точність та функціональність розробленої системи.

ВИСНОВКИ

Мета роботи полягала у створенні власної системи обробки візуальної інформації для ідентифікації та аналізу образів. В рамках даного проєкту образами, які піддаються ідентифікації та аналізу, є будівельні захисні каски.

Для досягнення цієї мети було виконано наступні задачі:

1. Проведено огляд основних теоретичних засад стосовно теми дослідження.
2. Розглянуто застосування нейромереж у завданнях розпізнавання образів.
3. Розглянуто основні методи та підходи визначення образів.
4. Розглянуто основні типи нейромереж.
5. Розглянуто ключові поняття та види архітектур, обрано CNN архітектуру для вирішення задачі розпізнавання.
6. Проаналізовано різноманітні технології та інструменти
для програмної реалізації системи ідентифікації образів, .
7. Підготовано власний набір даних для навчання моделі.
8. Розроблено архітектуру нейронної мережі для ефективної ідентифікації образів.
9. Здійснено навчання моделі з використанням набору даних, враховуючи різні архітектурні варіанти та гіперпараметри в межах методів машинного навчання.
10. Забезпечено можливість ідентифікації образів на зображеннях , відео, проведено тестування.

У першому розділі роботи проведено детальний аналіз основних аспектів галузі комп'ютерного зору. З'ясовано, що завдання, такі як класифікація зображень, виявлення об'єктів, сегментація та розпізнавання обличь, є важливою частиною цього напрямку. Висвітлені проблеми, які вимагають уваги та вирішення для подальшого розвитку технологій комп'ютерного зору.

У розділі розглянуті різноманітні методи та підходи до розпізнавання образів, звертаючи увагу на їхню різноманітність та можливості застосування в залежності від конкретних завдань. Особливий акцент зроблено на важливості основних етапів процесу розпізнавання образів у структурі роботи систем комп'ютерного зору.

У висновку даного розділу підкреслено, що комп'ютерний зір відіграє ключову роль у вирішенні різноманітних завдань, однак його ефективність залежить від успішного вирішення технічних та методологічних викликів. Загальна мета розділу полягає в наданні повного огляду теми дослідження, визначенні важливих аспектів, ідентифікації проблем та огляду різноманітних методів, що створює контекст для подальших досліджень у галузі комп'ютерного зору.

У другому розділі здійснено детальний аналіз внутрішньої структури штучних нейронних мереж (ШНМ), яка є ключовим елементом у галузі машинного навчання. Особлива увага була приділена визначенню та розгляду всіх значущих компонентів цієї техніки, таких як вхідний рівень, прихований рівень, вихідний рівень, вагові матриці, функція витрат, параметри та гіперпараметри.

Під час аналізу кожного з компонентів здійснено детальний розгортання вагових матриць, визначено важливість функцій витрат та особливості налаштування параметрів та гіперпараметрів для досягнення оптимальної ефективності ШНМ.

Основна мета полягала у наданні повного технічного огляду внутрішньої механіки мережі. У ході аналізу було визначено ключовий напрямок подальших досліджень та розробок у галузі систем ідентифікації, і вибір пав на згорткові нейронні мережі (CNN) для подальшої реалізації.

Обрана архітектура CNN вважається оптимальною для досягнення високої точності та ефективності у розпізнаванні образів, що є ключовим завданням у визначеній області дослідження.

У третьому розділі було докладно проаналізовано та порівняно різноманітні технології та інструменти для розробки програмної системи ідентифікації образів. Процес огляду розпочався з вибору мови програмування, і після ретельного розгляду таких факторів, як універсальність, зручність та багатофункціональність, було визначено, що Python стане основною мовою для реалізації проєкту.

Далі був проведений докладний аналіз трьох ключових фреймворків для машинного навчання. Виділено переваги кожного, зокрема гнучкість та висока продуктивність TensorFlow, а також простота та швидкість прототипування Keras.

В результаті було прийнято рішення використовувати обидва фреймворки для досягнення більшої ефективності в розробці моделей глибокого навчання.

Важливим кроком стало включення OpenCV до стеку технологій для обробки та аналізу зображень. Це рішення розширить можливості проєкту у сфері комп'ютерного зору та обробки візуальних даних.

Завершальним етапом вибору технологій було обрано використання Jupyter Notebook для проведення експериментів та візуалізації результатів. Це забезпечить зручний та інтерактивний підхід до роботи з кодом та допоможе ефективно аналізувати та візуалізувати дані.

Четвертий розділ присвячений практичному аспекту дослідження, в якому я докладно описала етапи розробки програмного забезпечення для системи розпізнавання образів.

Починаючи з підготовки навчального набору даних, я виконала збір та обробку вхідних даних, які є ключовими для ефективного тренування нейронної мережі.

Далі був розглянутий процес створення файлу TFRecord як оптимізованого формату для швидкого та ефективного читання даних під час тренування моделі. Наступним важливим етапом було написання власної нейронної мережі, яка була спеціально адаптована для розпізнавання будівельних касок на зображеннях.

Зокрема, був розглянутий розрахунок функції втрат IoU Loss, яка визначає точність розташування об'єктів. Подано реалізацію моделі для завдання об’єктного виявлення, яка визначає положення та форму будівельних касок.

До складу роботи також увійшло тренування розробленої нейронної мережі на підготовленому навчальному наборі даних, а також реалізація моделі класифікатора для ідентифікації різних класів об’єктів.

У висновках дослідження можна зазначити, що розроблена система розпізнавання образів на основі згорткових нейронних мереж (CNN) виявилася ефективною в розв'язанні завдань ідентифікації об'єктів.

Отримані результати свідчать про значний внесок у сферу комп'ютерного зору та створюють підґрунтя для подальших досліджень.

Обмеженням роботи виявилося обмежене кількість доступних даних для навчання, і в цьому контексті можливі перспективи розширення дослідження включають в себе збільшення обсягу даних та вдосконалення архітектур нейронних мереж.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Комп’ютерна інженерія: методичні рекомендації до виконання дипломних проєктів для студентів освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 “Комп’ютерна інженерія” / Уклад.: І.А. Жуков, М.М. Проценко – К.: НАУ, 2015. - 36 с.

2. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи випускників Національного авіаційного університету. – К.: НАУ, 2017. 63 с.

3. Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» від 05.07.1994 № 80/94-ВР (Редакція від 19.04.2014). Архів оригіналу за 22 червня 2017. Процитовано 8 вересня 2017.

4. ДСТУ 2392-94 Інформація та документація. Базові поняття.

5. *What Is Computer Vision? Meaning, Examples, and Applications in 2022* [Електронний ресурс] – Режим доступу до ресурсу: *https://kinit.sk/multimodal-processing-can-artificial-intelligence-learn-the-meaning-and-relationship-between-several-different-modalitie*

6. Комп'ютерний зір [Електронний ресурс] – Режим доступу до ресурсу: *https://uk.wikipedia.org/wiki/*

7. Як працює класифікація зображень? [Електронний ресурс] – Режим доступу до ресурсу: *https://www.unite.ai/uk/how-does-image-classification-work/*

8. Класифікація зображень глибокого навчання за допомогою *Tensorflow* і *Keras* в *Python* [Електронний ресурс] – Режим доступу до ресурсу: *https://hashdork.com/uk/deep-learning-image-classification-with-tensorflow-keras-in-python/*

9. Сегментація зображення [Електронний ресурс] – Режим доступу до ресурсу: *https://www.wikidata.uk-ua.nina.az*

10. Основи теорії розпізнавання образів : навч. посіб. : у 2 ч. / А. С. Довбиш, І. В. Шелехов. – Суми : Сумський державний університет, 2015. – Ч. 1. – 109 с.

11. Нейронні мережі - шлях до глибинного навчання [Електронний ресурс] – Режим доступу до ресурсу: *https://devzone.org.ua/post/neironni-merezi-sliax-do-glibinnogo-navcannia*

12. Терейковський, І. А. Штучні нейронні мережі: базові положення [Електронний ресурс] : навчальний посібник для здобувачів ступеня бакалавра за освітньою програмою «Системне програмування та спеціалізовані комп'ютерні системи» спеціальності 123 Комп'ютерна інженерія / І. А. Терейковський, Д. А. Бушуєв, Л. О. Терейковська ; КПІ ім. Ігоря Сікорського, 2022. – 123 с. – Назва з екрана.

13. *Python* [Електронний ресурс] – Режим доступу до ресурсу: *https://uk.wikipedia.org/wiki/Python*

14. *About Keras 3* [Електронний ресурс] – Режим доступу до ресурсу: *https://keras.io/about/*

15. *Pytorch documentation* [Електронний ресурс] – Режим доступу до ресурсу: *https://pytorch.org/docs/stable/index.html*

16. Що таке *Tensorflow?* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.oksim.ua/2023/08/07/shho-take-tensorflow/*

17. Порівняння найкращих інструментів машинного навчання Електронний ресурс] – Режим доступу до ресурсу: *https://www.zfort.com.ua/blog/porivnyannya-naikrashikh-instrumentiv-mashinnogo-navchannya-tensorflow-keras*

18. Вивчення основ *OpenCV* і розпізнавання обличчя Електронний ресурс] – Режим доступу до ресурсу: *https://ts2.space/uk/opencv*