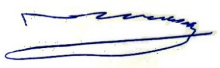
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ "КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО"

Факультет електроніки
(повна назва інституту/факультету)

Кафедра акустичних та мультимедійних електронних систем
(повна назва кафедри)

"На правах рукопису"
УДК 004.89

"До захисту допущено"
Завідувач кафедри

С.А. Найда
(ініціали, прізвище)

" 10" червня 2022 р

Магістерська дисертація

зі спеціальності (спеціалізації) 171 Електроніка
(код і назва)

на тему: "Застосування нейронних мереж в задачах розпізнавання об'єктів"

Виконав: студент II курсу, групи ДВ-01мн
(шифр групи)

Савка Максим Сергійович
(прізвище, ім'я, по батькові)

Керівник доцент, к.т.н., Філіпова Н.Ю.
(посада, вчене звання, науковий ступінь, прізвище, ініціали)

Консультант
(назва розділу) (посада, науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент доцент каф. ЕПС, к.т.н., доц. Клен К.С.
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

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

Студент
(підпис)

Київ – 2022 року

Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"

Інститут (факультет) Факультет електроніки
(повна назва)

Кафедра Акустичних та мультимедійних електронних систем
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою;

Спеціальність (освітня програма) 171 Електроніка (Електронні системи мультимедіа та засоби Інтернету речей)
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
С.А. Найда
(ініціали, прізвище)
"04" квітня 2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Савці Максиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Застосування нейронних мереж в задачах розпізнавання об'єктів

науковий керівник дисертації Філіпова Наталія Юріївна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від " 14 " квітня 2022 р. № НС-8-2022
2. Срок подання студентом роботи 01.06.2022
3. Об'єкт дослідження Веб-додаток для оцінки якості та кількості виконання фізичного навантаження на основі технологій використання штучних нейронних мереж.
4. Предмет дослідження Веб-додаток для розпізнавання положення тіла людини
5. Перелік завдань, які потрібно розробити 1. Аналіз існуючих рішень. 2. Вибір технологій для реалізації проекта. 3. Розробка плану використання технологій для реалізації проекта. 4. Створення концепту веб-застосунка. 5.Програмна реалізація проекта

6. Перелік графічного (ілюстративного) матеріалу 72 рис., 1 презентація, 13 слайдів

7. Орієнтовний перелік публікацій Савка М. С., Філіпова Н.Ю. Обумовленність вибору відеокарт замість центральних процесорів при навчанні нейронних мереж: матеріали III Всеукраїнської науково-технічної конференції «Технології кіно та аудіовізуальних систем» (9-10 грудня 2019 р). Київ, 2019 с. 23-25.

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Написання першого розділу	18.02.2022	Виконано
2	Написання другого розділу	04.03.2022	Виконано
3	Написання третього розділу	06.05.2022	Виконано
4	Оформлення пояснювальної записки	20.05.2022	Виконано
5	Підготовка доповіді і презентації	27.05.2022	Виконано

Студент _____ М.С. САВКА
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Н.Ю. ФІЛІПОВА
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

УДК 004.89

РЕФЕРАТ

Савка М.С. Застосування нейронних мереж в задачах розпізнавання об`єктів: магістерська дис.: 171 Електроніка / Савка Максим Сергійович. – Київ, 2022. – 117 с.

Ключові слова: нейронна мережа, машинне навчання, глибоке навчання, розпізнавання зображень, Python, розпізнавання об'єктів

Актуальність дослідження. Штучний інтелект – один із найголовніших напрямів розвитку сфери інформаційних технологій у сьогоденні. Штучні нейронні мережі є відгалуженням від глобального напрямку штучного інтелекту. Нейронні мережі можуть використовуватись для великого спектру задач, у тому числі для обробки зображень. Технологія обробки зображень для розпізнавання образів може забезпечити розпізнавання положення тіла людини з достатньо високою точністю, що дозволить відслідковувати її рухи та підраховувати кількість активності людини.

Метою дослідження є створення веб-застосунку з використанням нейронної мережі для визначення кількісних та якісних параметрів активності людини.

Об'єкт дослідження – веб-застосунок для оцінки якості та кількості виконання фізичного навантаження на основі технологій використання штучних нейронних мереж.

Предмет дослідження – нейронна мережа для розпізнавання положення тіла людини

Методи дослідження – розробка веб-застосунку на основі нейронної мережі та аналіз компонентів для реалізації системи.

Наукова новизна одержаних результатів: запропонована система використання технології розпізнавання об’єктів за допомогою нейронних мереж для контролю якості та кількості виконання вправ.

Практичне значення одержаних результатів: У результаті виконання дипломної роботи розроблено веб-застосунок для оцінки виконання фізичного навантаження. Проведено аналіз технологій для реалізації створеного рішення. Отримані результати можуть бути використані для реалізації проекта із самостійного контролю кількості та правильності фізичної активності людини.

Апробація результатів дисертації: Савка М. С., Філіпова Н.Ю. Обумовленність вибору відеокарт замість центральних процесорів при навчанні нейронних мереж: матеріали III Всеукраїнської науково-технічної конференції «Технології кіно та аудіовізуальних систем» (9-10 грудня 2019 р). Київ, 2019 с. 23-25.

ABSTRACT

Savka M.S. Application of neural networks in object recognition problems: master's thesis: 171 Electronics / Savka Maksym Serhiiyovich. - Kyiv, 2022. - 117 p.

Keywords: neural network, machine learning, deep learning, image recognition, Python, object recognition

This paper describes the development of a web application using a neural network to determine the quantitative and qualitative parameters of human activity. The first part is an analytical review of the prerequisites for the use of neural networks. The first section analyzes the historical preconditions for the creation of artificial intelligence systems. The principles of construction of neural network systems are considered. A common feature of many types of controlled and uncontrolled models of deep learning is that these models have many layers of latent neurons that learn in combination with the back propagation and error gradients of stochastic gradient descent. A fundamental advantage of neural networks among standard approaches to computer algorithms is the ability of the network to classify data very accurately, adjusting the strength of the connection between their neurons, ie changing the values of weights.

The second section discusses the tools for developing and training neural networks. It is advisable to use the Python programming language to develop a solution for object recognition. The TensorFlow library for Python is considered. TensorFlow provides a set of working tools for developing and learning models using Python. The expediency of using JavaScript together with Flask to build the interface of the project website is considered. The use of the BlazePose pose detection model is considered. BlazePose displays 33 key points according to the following order. The use of graphics processors for the neural network learning process is conditioned.

The third section develops a web application for estimating the quality and quantity of physical activity based on the use of artificial neural networks. The capabilities of the Flask framework were used to implement the server part of the web application. The BlazePose model was used as a model for recognizing the position of the human body. The user interface of the web application includes the number of

repetitions performed, as well as the status of the position of the hands during the exercise. Added debug mode with display of recognized parts of the human body for better visibility, as well as for the process of tracking errors in recognition.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AI — Artificial intelligence (штучний інтелект);
API — Application Programming Interface (програмний інтерфейс додатка);
AR — Augmented Reality (доповнена реальність);
BPTT — алгоритм зворотного поширення через час;
CNN — згорткова нейронна мережа;
CONV — згортковий шар;
CPU — Central Processing Unit (центральний процесор);
CSS — Cascading Style Sheets (каскадні таблиці стилів);
FC — повністю зв'язаний шар;
GPU — Graphics Processing Unit (графічний процесор);
HTML — HyperText Markup Language (мова розмітки гіпертексту);
LOSS — шар втрат;
ML — Machine Learning (машинне навчання);
MR — Mixed Reality (змішана реальність);
NaN — Not a Number (особливий стан числа);
NLP — обробка природної мови;
PCA — аналіз основних компонентів;
PDE — діференційне рівняння з частковими частинами;
POOL — об’єднання шарів;
ReLU — зрізаний лінійний вузол;
RNN — рекурентна нейронна мережа;
SDK — Software Development Kit (комплект для розробки програмного забезпечення);
SIFT — масштабно-інваріантне перетворення ознак;
SURF — прискорені надійні функції;
TF — TensorFlow;
TPU — Tensor Processing Unit (тензорний блок обробки);
VR — Virtual Reality (віртуальна реальність);

ВСТУП

Сфера штучного інтелекту все більше залучаються до нашого особистого, професійного та соціального життя. У контексті розвитку напрямку штучного інтелекту людські очікування та вимоги стають все більш конкретними, аж до бажання використовувати напрацювання штучного інтелекту у більшості сфер діяльності людини, щоб зменшити людське навантаження та помилки. У цих напрямках моделі нейронних мереж були пріоритетними для застосування по причині їх обчислювальної парадигми, яка заснована на функціонуванні та навчанні мозку. Однак дуже скоро стало очевидним, що для того, щоб комп'ютери демонстрували автономну поведінку, недостатньо використовувати парадигми навчання та функціонування людини.

Існують три важливі проблеми, які необхідно враховувати для реалізації автономної системи на основі нейронної мережі, яка виконує автоматну інтелектуальну поведінку людини. Перша пов'язана зі збором відповідної бази даних для навчання та оцінки продуктивності системи. Друга – це прийняття відповідного машинного представлення даних, що передбачає вибір відповідних характеристик даних для розглянутої проблеми. Нарешті, вибір схеми класифікації може вплинути на досягнуті результати. Все це необхідно враховувати для досягнення ефективності обчислювання та оптимальної продуктивності. Процедура, що використовується для виконання процесу навчання, називається алгоритмом навчання, функція якого полягає в тому, щоб упорядковано змінювати синаптичні ваги мережі для досягнення бажаної мети проектування.

Динамічна зміна синаптичних коефіцієнтів забезпечує традиційний метод для проектування нейронних мереж. Даний підхід є найбільш близьким до теорії лінійного адаптивного фільтра, яка вже добре сформована і успішно застосовується в багатьох різноманітних областях. Однак нейронна мережа також може змінювати свою власну топологію, що мотивується тим, що нейрони в мозку людини можуть згасати і що нові синаптичні зв'язки можуть рости.

Очевидно, що нейронна мережа отримує свою обчислювальну перевагу по причині своєї масивно паралельній розподіленій структурі, а також її здатності навчатися, тобто змінювати вагові коефіцієнти. Зміною вагових коефіцієнтів є узагальнення, яке відноситься до нейронної мережі, яка отримує на виході відповідні дані для вхідних даних, які не зустрічаються під час навчання. Ці можливості обробки інформації дозволяють нейронним мережам вирішувати складні (великі за обсягом) проблеми, які на даний момент нерозв’язні. Однак на практиці. нейронні мережі не можуть надати рішення, працюючи окремо. Навпаки, їх потрібно інтегрувати в послідовний системний інженерний підхід. Зокрема, складна проблема, що цікавить, розкладається на ряд відносно простих завдань, а нейронним мережам призначається підмножина завдань, які відповідають їхнім властивим можливостям. Важливо, однак, визнати, що нам попереду довгий шлях (якщо взагалі колись), перш ніж ми зможемо створити комп’ютерну архітектуру, яка імітує людський мозок.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Основні засади технології нейронних мереж

Алгоритми нейронних мереж для машинного навчання беруть за основу архітектуру та динамічність мереж нейронів мозку [1], [2]. Вони використовують високоідеалізовані моделі нейронів. Проте основний, або ж фундаментальний принцип залишається той самий: штучні нейронні мережі мають здатність навчатися, змінюючи зв’язки між своїми нейронами, тобто вагові коефіцієнти [3], [4] Такі мережі можуть виконувати безліч завдань з обробки інформації.

Штучні нейронні мережі поки що не наближаються до складності мозку (рис. 1.1). Однак є дві ключові схожості між біологічними та штучними нейронними мережами. По-перше, основними блоками обох мереж є прості обчислювальні пристрої (хоча штучні нейрони набагато простіші за біологічні), які дуже пов’язані між собою. По-друге, зв’язки між нейронами визначають функцію мережі [5].

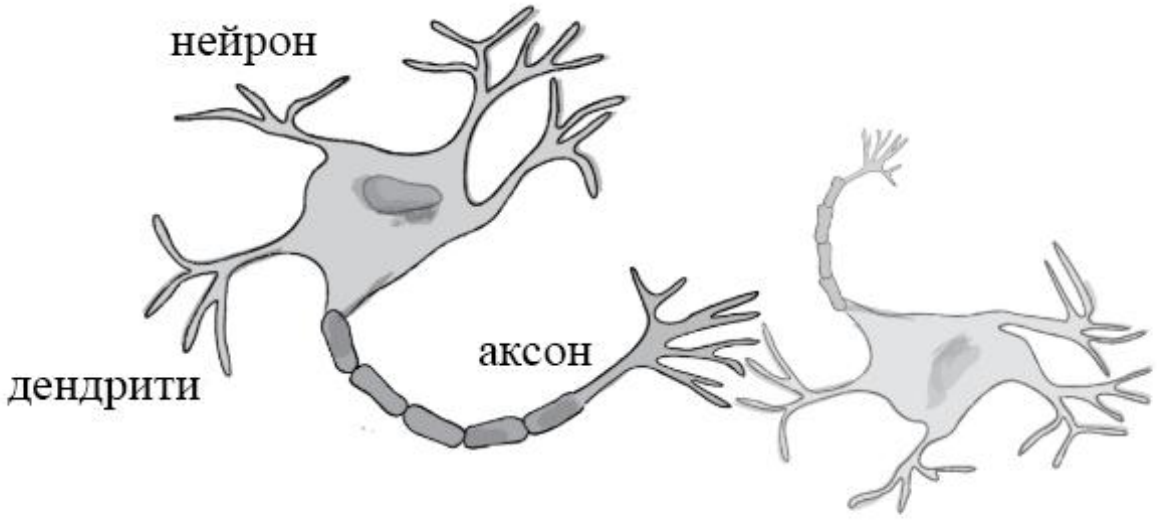


Рисунок 1.1 – Схематичне зображення будови нейронів людини

Моделі штучних нейронних мереж використовуються з середини 20-го століття [6]. Однак нинішня хвиля популярності нейронних мереж, а саме їх

підвиду – нейромереж глибокого навчання припадає на другу половину 2000-х років [7]. Загальною характеристикою багатьох видів контрольованих і неконтрольованих моделей глибокого навчання є те, що ці моделі мають багато шарів прихованих нейронів (рис. 1.2), які навчаються у поєднанні з зворотним поширенням і градієнтами помилок стохастичного градієнтного спуску [8].

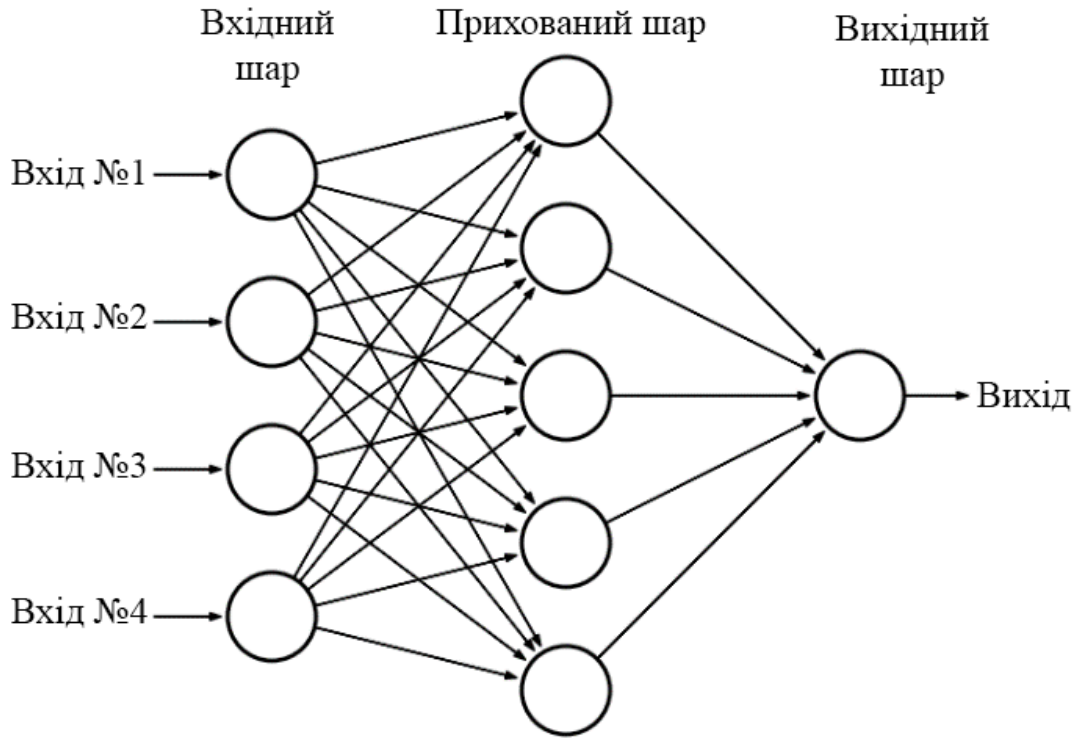


Рисунок 1.2 – Багатошаровий перцептрон

Наприклад, нейронні мережі можуть навчитися розпізнавати схожість в наборі даних і, певною мірою, узагальнювати вивчену інформацію [9]. Це фундаментальна перевага нейронних мереж серед стандартних підходів до комп’ютерних алгоритмів. Штучні нейронні мережі можна навчити дуже точно класифікувати дані, регулюючи міцність зв’язку між їхніми нейронами, тобто змінюючи значення вагових коефіцієнтів [10]. Вони також можуть узагальнювати результат на інші набори даних – за умови, що нові дані не надто відрізняються від даних навчання. Яскравим прикладом вирішення проблеми такого типу є розпізнавання об’єктів на зображеннях, наприклад у послідовності зображень камери, зроблених самокерованим автомобілем. Зростаючий останнім часом

інтерес до машинного навчання у немалій мірі зумовлений успіхом нейронних мереж саме у візуальному розпізнаванні об’єктів.

Іншою сферою, у якій нейронні мережі чудово справляються та активно використовуються, є машинний переклад [11]. Зазвичай для цього використовуються мережі рекурентної архітектури. На вхід мережі такого типу подають речення. При передачі слова за словом, мережа виводить слова в перекладеному реченні. Рекурентні мережі можна ефективно навчати на великих навчальних наборах вхідних речень та їх перекладів. Рекурентні мережі також доволі успішно використовуються для прогнозування хаотичної динаміки [12].

Навчальний набір (рис. 1.3) містить список шаблонів введення разом зі списком відповідних міток або цільових значень, які кодують властивості вхідних шаблонів, які мережа повинна засвоїти.

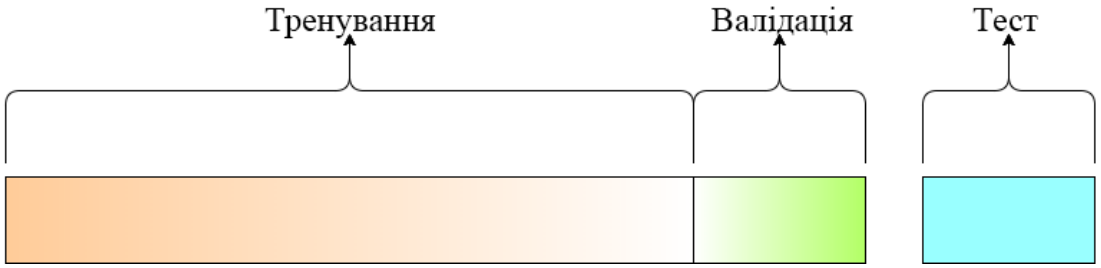


Рисунок 1.3– Схема розподілу даних навчального набору

Нейронні мережі широко використовуються з додатками для фінансових операцій, корпоративного планування, торгівлі, бізнес-аналітики та обслуговування продуктів [13]. Нейронні мережі також отримали широке поширення в бізнес-додатках, таких як рішення для прогнозування та маркетингових досліджень, виявлення шахрайства та оцінка ризиків.

1.2 Принципи розпізнавання зображень

1.2.1 Сфери використання технології розпізнавання зображень

Розпізнавання зображень – це підкатегорія комп’ютерного зору та штучного інтелекту, яка являє собою набір методів для виявлення та аналізу зображень для

автоматизації певного завдання. Це технологія, яка здатна ідентифікувати місця, людей, об’єкти та багато інших типів елементів у зображенні на основі яких робити висновки, аналізуючи їх [14]–[16] (рис. 1.4).

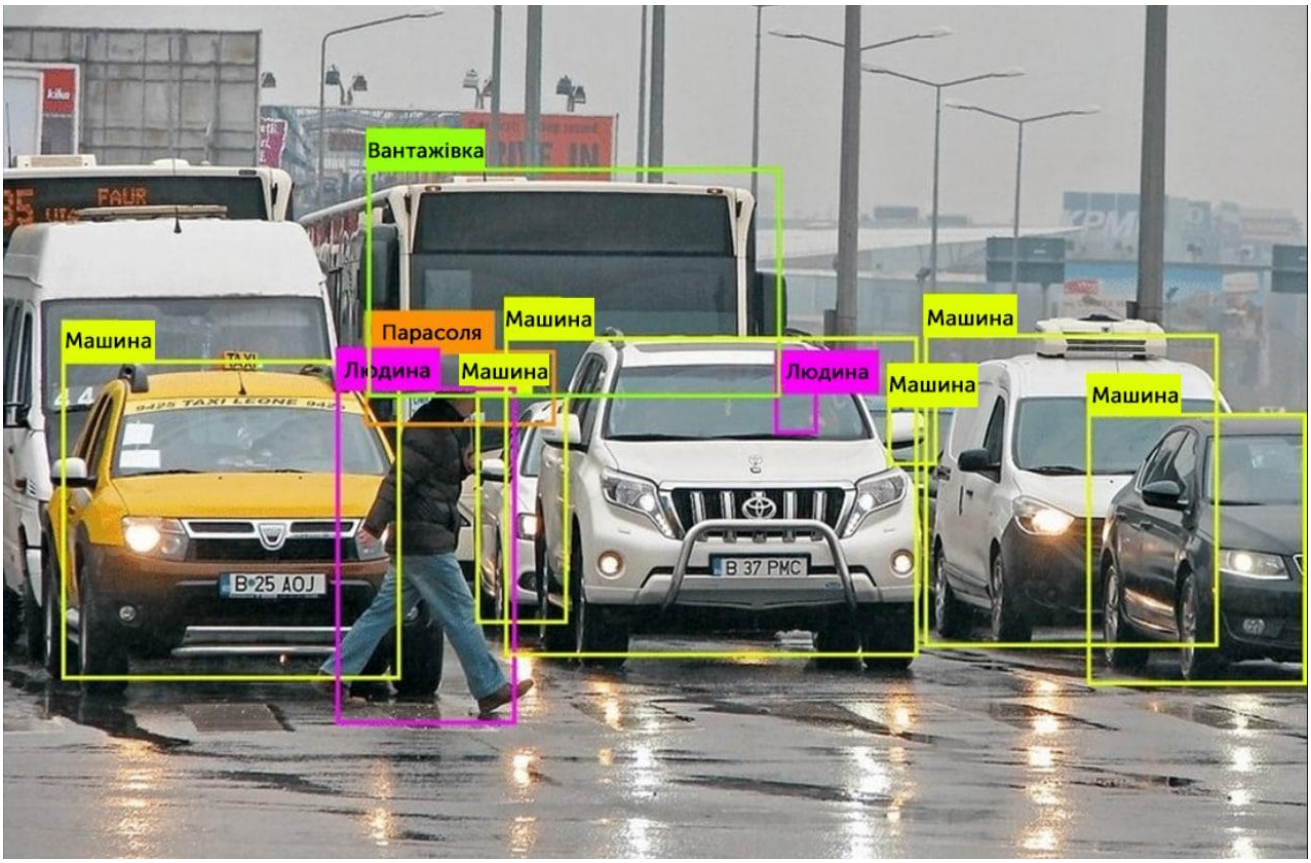


Рисунок 1.4 – Приклад розпізнавання об'єктів бібліотекою OpenCV

Розпізнавання фото- або відеоматеріалів може виконуватися з різним ступенем точності, залежно від типу необхідної інформації. Модель або алгоритм здатні виявити певний елемент, так само, як вони можуть віднести зображення до якоїсь великої категорії за відповідною ознакою.

Існують різні завдання, які може виконувати розпізнавання зображень, наприклад:

1) Класифікація. Це ідентифікація «класу», тобто категорії, до якої належить зображення. Зображення може мати тільки один клас.
2) Маркування. Це також завдання класифікації, але з вищим ступенем точності. Нейромережа може розпізнати наявність кількох понять або

об’єктів у зображенні. Тому певному зображенню можна призначити один або кілька тегів.

3) Виявлення. Це необхідно, якщо потрібно знайти об’єкт на зображенні. Зазвичай, після виявлення елемента, навколо нього розташовується обмежувальна рамка.
4) Сегментація. Це також завдання виявлення. Сегментація може визначити розташування елемента на зображенні до найближчого пікселя. Для деяких випадків необхідно бути гранично точними. Наприклад у випадках розробки автономних автомобілів.

1.2.2 Процес розпізнавання зображень

Розпізнавання зображень засноване на методах глибокого навчання (рис. 1.5). Глибоке навчання – це підкатегорія машинного навчання, що відноситься до набору методів і технологій автоматичного навчання на основі штучних нейронних мереж [17]–[20].

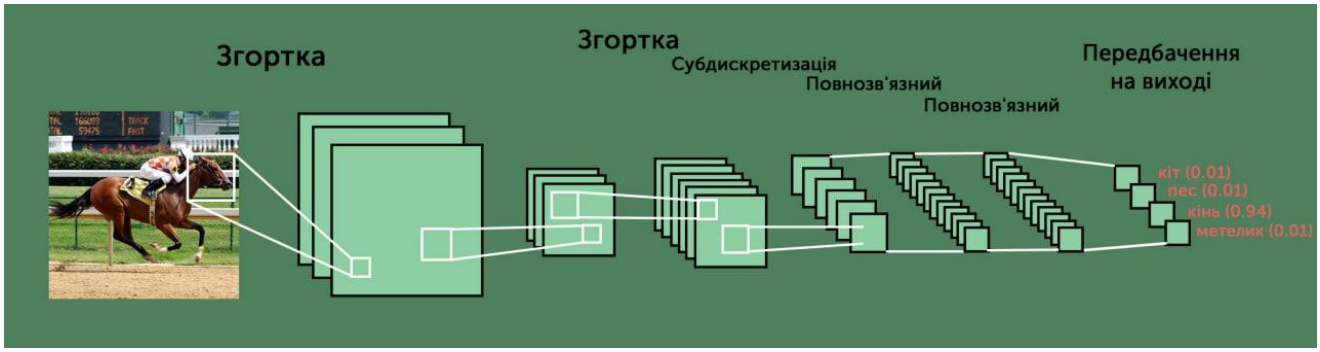


Рисунок 1.5– Схематичне зображення процесу розпізнавання методом глибокого навчання

Для того щоб нейронні мережі розпізнавали одне або кілька понять в зображенні, потрібно провести процес навчання. Для цього необхідно зібрати деякий набір візуальних даних (зазвичай у вигляді зображень) і створити набір для навчання [21].

Розпізнавання зображень працює шляхом аналізу кожного пікселя зображення для вилучення інформації, як це робить людське око. Тому важливо навчати мережу на якісному наборі даних.

Після створення набору даних важливо додати до нього анотації, тобто повідомити моделі, чи присутній на зображенні елемент, який нейронна мережа буде шукати, а також його розташування. Існують різні типи міток (теги, рамки або багатокутники) залежно від вибраного завдання.

Після процесу анотації всього набору даних, можна переходити до навчання. Як і у випадку з людським мозком, нейронну мережу потрібно навчити розпізнавати концепцію, показуючи їй багато різних прикладів.

Кінцева мета навчання полягає в тому, що алгоритм може робити прогнози після аналізу зображення. Іншими словами, алгоритм повинен мати можливість призначити клас зображенню або вказати, чи присутній певний елемент.

За допомогою системи або платформи розпізнавання зображень можна автоматизувати бізнес-процеси і таким чином підвищити продуктивність. Як тільки модель розпізнає елемент на зображенні, її можна запрограмувати на виконання певної дії. Варіанти використання даного типу автоматизації широко розгорнуті в різних галузях і секторах.

Наприклад, у телекомунікаційному секторі було розгорнуто рішення для автоматизації контролю якості. Фактично, польові техніки використовують систему розпізнавання зображень для контролю якості своїх установок.

Іншим прикладом є інтелектуальна система відеоспостереження, заснована на розпізнаванні зображень, яка здатна повідомляти про будь-яку незвичайну поведінку або ситуації на автостоянках.

Тому розпізнавання зображень можна застосувати як у телекомунікаціях та відеоспостереженнях (рис. 1.6), так і в будівельній та фармацевтичній промисловості.

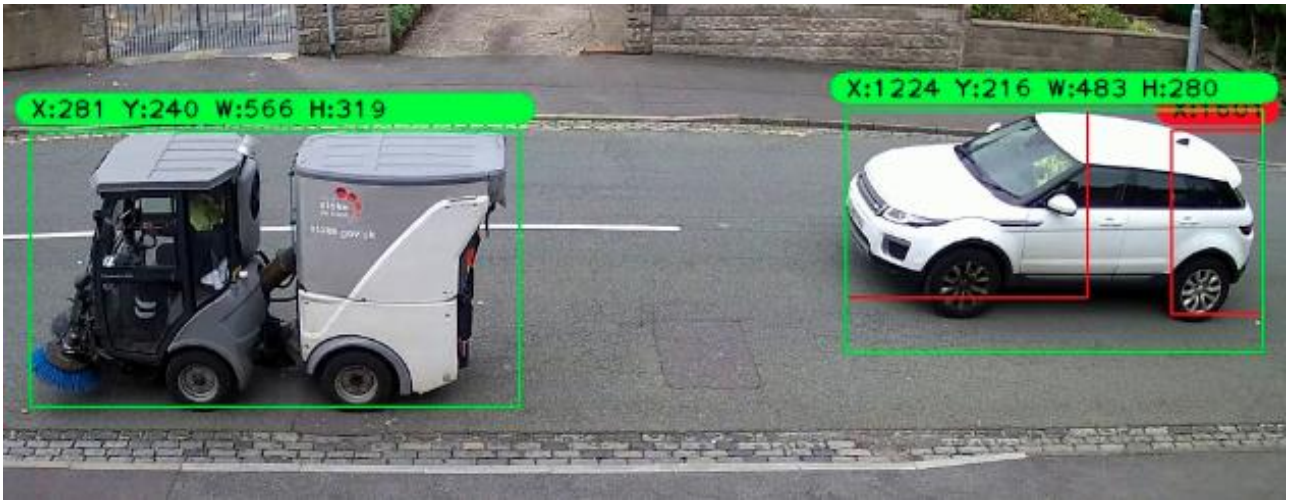


Рисунок 1.6– Приклад розпізнавання автомобілей камерою зовнішнього спостереження

Як згадувалося вище, цифрове зображення являє собою матрицю чисел (рис. 1.7). Це число представляє дані, пов’язані з пікселями зображення. Різна інтенсивність пікселів утворює середнє значення і представляє себе у форматі матриці.

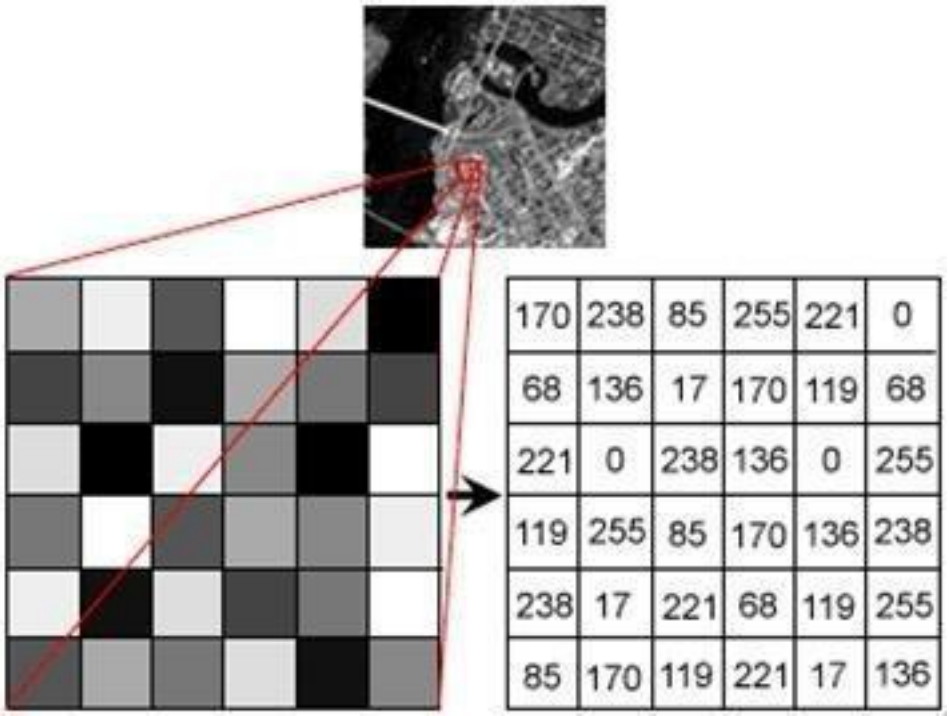


Рисунок 1.7– Матриця значення інтенсивності світла кожного окремого пікселя у чорно-білому зображенні

Дані, які надходять до системи розпізнавання, це в основному розташування та інтенсивність різних пікселів на зображенні. Відповідно можна навчити систему відображати закономірності та відносини між різними зображеннями, використовуючи цю інформацію.

Після завершення процесу навчання можна проаналізувати продуктивність системи за тестовими даними. Вагові коефіцієнти нейронної мережі оновлюються, щоб підвищити точність системи і отримати більш точні результати для розпізнавання зображення. Нейронні мережі обробляють ці числові значення за допомогою алгоритму глибокого навчання багато разів та порівнюють їх із конкретними параметрами, щоб отримати бажаний результат.

Масштабно-інваріантне перетворення ознак (SIFT), прискорені надійні функції (SURF) і PCA (аналіз основних компонентів) є одними з алгоритмів, які зазвичай використовуються в процесі розпізнавання зображень.

1.3 Види нейронних мереж

1.3.1 Нейронні мережі прямого поширення

Найпростішою формою нейронної мережі прямого поширення є одношаровий персептрон (рис. 1.8). У одношаровому персептроні серія вхідних даних після входу у шар множиться на коефіцієнти – ваги. Після цього кожне значення додається разом для отримання суми зважених вхідних значень. При сумі значень вище певного порогового значення – отримане значення дорівнює 1, а при сумі значень менше деякого встановленого порогу – вихідне значення дорівнює 0.

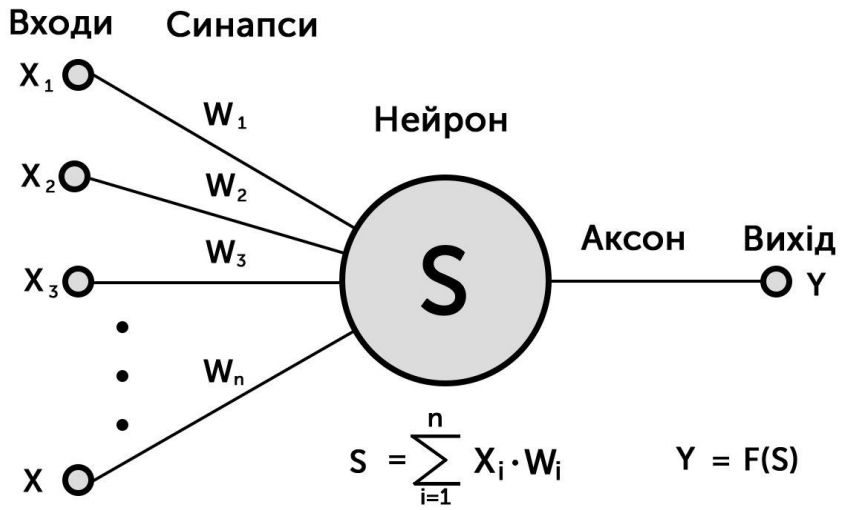


Рисунок 1.8– Одношаровий перцептрон

Одношаровий персептрон є важливою моделлю нейронних мереж із прямим поширенням і часто використовується в задачах класифікації. Крім того, одношарові персептрони навіть можуть мати аспекти машинного навчання. Використовуючи властивість, відому як правило дельта, нейронна мережа може порівнювати вихідні дані своїх вузлів із встановленними значеннями, таким чином дозволяючи мережі коригувати свої ваги за допомогою навчання, щоб отримати більш точні вихідні значення. Процес навчання даного типу створює форму градієнтного спуску. У багатошарових персептронах процес оновлення вагових коефіцієнтів майже аналогічний, однак більш коректно цей процес називається зворотним поширенням. У таких випадках кожен прихований шар у мережі коригується відповідно до вихідних значень, отриманих останнім шаром.

Не дивлячись на те, що нейронні мережі з переадресацією подачі даних є досить простими, їх спрощена архітектура може бути використана як перевага в конкретних програмах машинного навчання. Наприклад, можна створити серію нейронних мереж із прямим зв’язком з наміром запускати їх незалежно одна від одної, але з присутнім посередником для модерації. Як і людський мозок, цей процес покладається на багато окремих нейронів, щоб виконувати та обробляти більші завдання. Оскільки окремі мережі виконують свої завдання незалежно,

результати можуть бути об’єднані в кінці для отримання синтезованого та цілісного результату.

У нейронній мережі прямого поширення (рис. 1.9) інформація рухається лише в одному напрямку — від вхідного шару через приховані шари до вихідного шару [22]. Інформація переміщується прямо через мережу і ніколи не проходить через один вузел двічі.

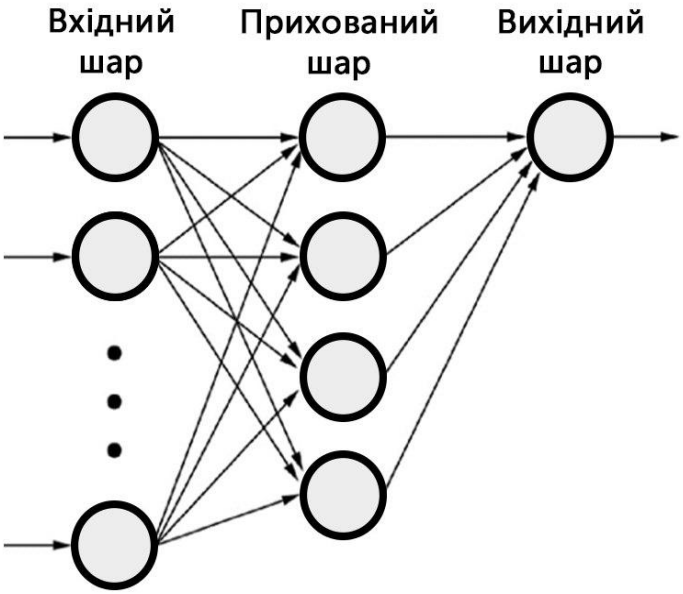


Рисунок 1.9– Схематичне зображення нейронної мережі прямого поширення

Нейронні мережі з прямим зв’язком використовуються для вивчення зв’язку між незалежними змінними, які служать вхідними для мережі, і залежними змінними, які позначаються як вихідні дані мережі. Навчання відбувається, коли набір вибірок «навчального набору», для яких відомі мітки класів, представлений мережі, а вагові коефіцієнти мережі коригуються, щоб мінімізувати відмінності між вихідними сигналами мережі та відомими правильними виходами. Після того, як вагові коефіцієнти були відкориговані за допомогою вибірок у навчальному наборі, мережу можна використовувати для прогнозування приналежності до класу невідомих вибірок.

1.3.2 Рекурентні нейронні мережі

Рекурентна нейронна мережа (RNN) — це тип штучної нейронної мережі, який використовує послідовні дані або дані часових рядів [23]. RNN і нейронні мережі прямого поширення отримали свої назви від способу передачі інформації (рис. 1.10). Як і багато інших алгоритмів глибокого навчання, рекуррентні нейронні мережі відносно старі. Вони були створені в 1980-х роках, але лише в останні роки світ побачив їх справжній потенціал. Збільшення обчислювальної потужності разом із величезними обсягами даних, з якими зараз доводиться працювати, вивели RNN на перший план.

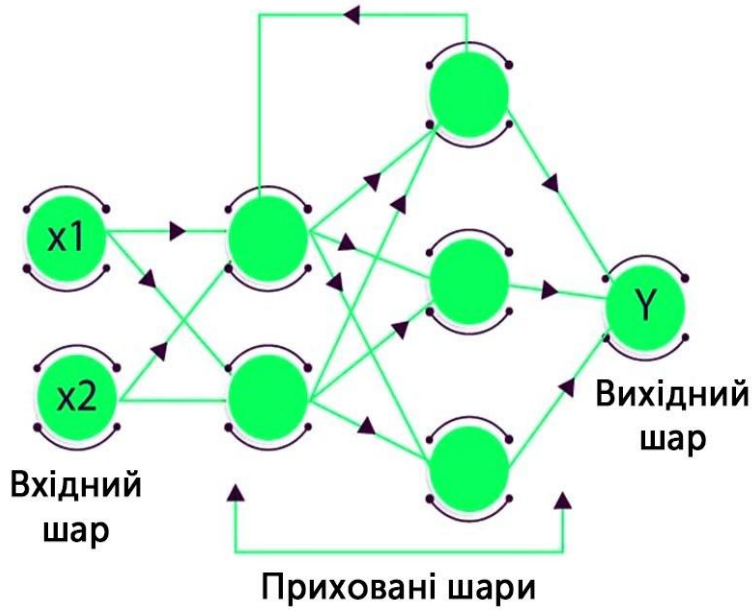


Рисунок 1.10 – Схематичне зображення архітектури рекурентної нейронної мережі

RNN архітектури глибокого навчання зазвичай використовуються для таких задач як мовний переклад, обробка природної мови (NLP) або ж розпізнавання мовлення та рукописів [24]. Вони використовуються у популярних додатках, таких як: Google Translate, голосовий пошук і Siri. Подібно до нейронних мереж прямого поширення та згорткових нейронних мереж (CNN), рекурентні нейронні

мережі використовують навчальні дані для процесу навчання. RNN є потужним і надійним типом нейронної мережі і належать до найбільш перспективних алгоритмів, які використовуються, оскільки вони єдині мають внутрішню пам'ять. Вони беруть інформацію з попередніх входів, щоб впливати на поточний вхід і вихід. Завдяки своїй внутрішній пам’яті RNN можуть запам’ятовувати важливі дані, що дозволяє їм бути дуже точним у прогнозуванні того, що буде далі. У RNN інформація проходить по циклу. Коли він приймає рішення, він враховує поточні вхідні дані, а також те, що він дізнався з вхідних даних, які він отримав раніше. Наприклад, нейронні мережі прямого поширення не пам’ятають вхідні дані, які вони отримують, і погано передбачають, що буде далі. Оскільки мережа прямого поширення враховує лише поточний вхід, вона не має поняття про порядок у часі. Ось чому RNN є кращим алгоритмом для послідовних даних, таких як часові ряди, мова, текст, фінансові дані, аудіо, відео, погода та багато іншого. Рекурентні нейронні мережі можуть сформувати набагато глибше розуміння послідовності та її контексту порівняно з іншими алгоритмами.

У той час як традиційні глибокі нейронні мережі діють за схемою незалежності входів та виходів один від одного, вихід рекуррентних нейронних мереж залежить від попередніх елементів у послідовності. Хоча майбутні події також можуть бути корисними для визначення результату послідовності, однонаправлені рекурентні нейронні мережі не можуть врахувати ці події в своїх прогнозах.

Іншою характеристикою рекурентних мереж є те, що вони мають спільні параметри на кожному рівні мережі. У той час як мережі прямого поширення мають різні ваги для кожного вузла, рекурентні нейронні мережі мають однаковий параметр вагового коефіцієнту в кожному шарі мережі. Ці вагові коефіцієнти все ще коригуються в процесі зворотного поширення та градієнтного спуску, щоб полегшити навчання з підкріпленням.

Рекурентні нейронні мережі використовують алгоритм зворотного поширення через час (BPTT) для визначення градієнтів [25]. Він відрізняється від традиційного зворотного поширення, оскільки є специфічним для даних

послідовності. Принципи BPTT не відрізняються від традиційного зворотного поширення: модель тренується, обчислюючи помилки від вихідного рівня до вхідного. Розрахунки даних помилок дозволяють належним чином налаштувати та підігнати параметри моделі. BPTT відрізняється від традиційного підходу тим, що BPTT підсумовує помилки на кожному кроці часу, тоді як мережі з прямим поширенням не повинні підсумовувати помилки, оскільки вони не поділяють параметри на кожному рівні.

Завдяки цьому процесу RNN, як правило, мають справу з двома проблемами, відомими як градієнти, що розриваються, і градієнти, що зникають. Ці проблеми визначаються розміром градієнта, який є нахилом функції втрат вздовж кривої помилки. Коли градієнт занадто малий, він продовжує зменшуватися, оновлюючи параметри ваги, доки вони не стануть незначними, тобто. 0. Коли це відбувається, алгоритм більше не навчається. Розривні градієнти виникають, коли градієнт занадто великий, створюючи нестабільну модель. У цьому випадку ваги моделі виростуть занадто великими, і в кінцевому підсумку вони будуть представлені як NaN. Одним із рішень цих проблем є зменшення кількості прихованих шарів у нейронній мережі, усуваючи частину складності моделі RNN.

1.3.3 Згорткові нейронні мережі

CNN вперше були розроблені та використані приблизно в 80-х роках 20-го століття [26]. Найбільше, на що вистачало потужності ЕОМ тих часів для роботи з CNN — це розпізнавання рукописних цифр. Здебільшого вони використовувалася в поштовому секторі для зчитування поштових індексів, пін-кодів тощо. Для навчання будь-якої моделі глибокого навчання потрібна велика кількість даних, а також багато обчислювальних ресурсів. Це було серйозним недоліком для CNN того періоду, і тому CNN обмежувалися лише поштовими секторами, їм не вдалося набути достатнього поширення.

У глибокому навчанні згорткова нейронна мережа (CNN/ConvNet) — це клас глибоких нейронних мереж, які найчастіше застосовуються для аналізу візуальних зображень [27]. Цей клас є спеціалізованим типом моделі нейронної мережі, призначеної для роботи з двовимірними масивами, тобто даними зображення, хоча їх можна використовувати з одновимірними та тривимірними даними. Коли мається на увазі нейронна мережа – зазвичай згадується про множення матриці, але це не відповідає дійності у ConvNet. Для CNN використовується спеціальна техніка під назвою згортка (рис. 1.11). Згортка у математиці — це математична операція над двома функціями, яка створює третю функцію. Вона дозволяє виражати як одна форма змінюється іншою. Центральним у згортковій нейронній мережі є згортковий шар, який і надав мережі назву. Саме цей шар виконує операцію під назвою «згортка».

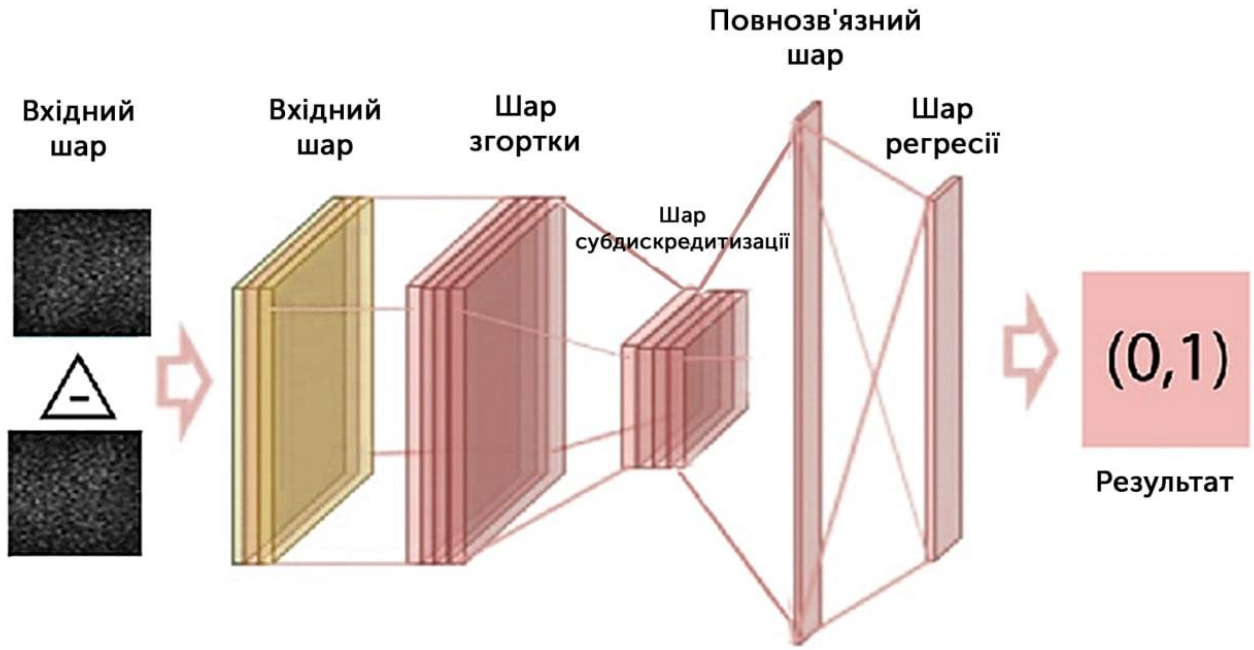


Рисунок 1.11 – Схематичне зображення архітектури згорткової нейронної мережі

Шар згортки відіграє ключову роль у CNN. Він складається з набору математичних операцій спеціалізованого типу лінійної операції. У цифрових зображеннях значення пікселів зберігаються в двовимірній (2D) сітці, тобто в масиві чисел, а невелика сітка параметрів, що називається ядром (оптимізований

екстрактор функцій) застосовується до кожної позиції зображення, що робить CNN високоефективними для обробки зображень. Оскільки один шар передає свій вихід на наступний шар, витягнуті об’єкти можуть ієрархічно та поступово ставати все більш складними.

Мережева архітектура CNN натхнена функціонуванням зорової кори тварин [28]. Аналіз поля зору виконується за допомогою набору субрегіонів, що розбивають зображення. Кожен підрегіон аналізує нейрон для попередньої обробки невеликих обсягів інформації. Це називається згортковою обробкою.

Архітектура згорткової нейронної мережі формується послідовністю будівельних блоків для виділення ознак, які розрізняють належний клас образу від інших [29]. Будівельний блок складається з однієї або кількох частин:

1) Згортковий шар (CONV), який обробляє дані рецепторного поля;
2) Корекційний шар, який часто називають «ReLU» з посиланням на функцію активації (Rectified Linear Unit);
3) Об’єднання шарів (POOL), яке стискає інформацію шляхом зменшення розміру проміжного зображення (часто шляхом підвибірки).

Будівельні блоки змінюють один одного до кінцевих шарів мережі, які виконують класифікацію зображень і обчислення похибки між прогнозом і цільовим значенням:

1) Повністю зв'язаний (FC) шар – є персептроноподібним шаром;
2) Шар втрат (LOSS).

Особливістю архітектури мережі є те, як згорткові шари, корекція та об’єднання змінюють один одного в будівельних блоках, а також спосіб, яким самі будівельні блоки змінюють один одного (рис. 1.12).

Алгоритм роботи архітектури CNN достатньо простий:

1) Розрізання на підобласті, які називаються плитками
2) Аналіз згортковим ядром

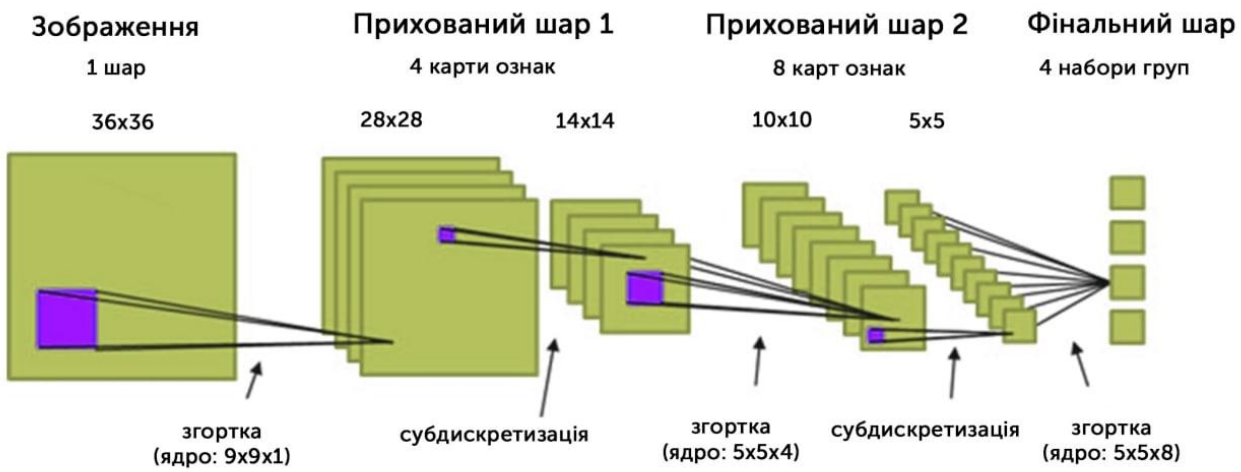


Рисунок 1.12 – Приклад роботи згорткової нейронної мережі на одному участку зображення

Згорткове ядро має розмір плитки, часто 3*3 або 5*5. Аналізована область (сприйнятливе поле) трохи більше, ніж ядро, оскільки додається крок, щоб сприйнятливі поля перекривалися. Цей прийом дозволяє краще відобразити зображення та покращити узгодженість його обробки.

Процес оптимізації таких параметрів, як ядра, називається навчанням. Воно виконується таким чином, щоб мінімізувати різницю між виходами та основними мітками за допомогою алгоритму оптимізації, який називається зворотним поширенням і градієнтним спуском.

Аналіз характеристик зображення згортковим ядром є операцією фільтрації з ваговими показниками, пов’язаними з кожним пікселем. Застосування фільтра до зображення називається згорткою.

Після згортки отримується карта ознак, що є абстрактним представленням зображення. Його значення залежать від параметрів застосованого ядра згортки та значень пікселів вхідного зображення.

У контексті згорткової нейронної мережі згортка — це лінійна операція, яка включає множення набору вагових коефіцієнтів на вхідні дані, так само як це відбувається і в традиційних нейронних мережах. Враховуючи, що методика була

розроблена для двовимірних вхідних даних, множення виконується між масивом вхідних даних і двовимірним масивом ваг, який називається фільтром або ядром.

Кілька згорткових ядер проходять через зображення, що призводить до кількох вихідних карт функцій. Кожне згорткове ядро має параметри, специфічні для інформації, яка шукається в зображенні (наприклад: згорткове ядро типу фільтра Собеля має параметри для пошуку контурів на зображенні).

Фільтр менший за вхідні дані, а тип множення, який застосовується між вхідним фрагментом розміром з фільтр, і фільтром є точковим добутком. Точковий добуток — це поелементне множення фрагменту розміру вхідного фільтра на фільтр, яке потім підсумовується, що призводить до єдиного значення. Оскільки це призводить до єдиного значення, операцію часто називають «скалярним добутком».

Використання фільтра, меншого за вхідний, є навмисним, оскільки дозволяє помножити один і той же фільтр (набір ваг) на вхідний масив кілька разів у різних точках входу. Зокрема, фільтр застосовується систематично до кожної частини, що перекривається, або фрагмента розміру фільтра вхідних даних, зліва направо, зверху вниз.

Вибір параметрів згорткового ядра (рис. 1.13) залежить від завдання, яке необхідно розв’язувати. При методах глибокого навчання ці параметри автоматично вивчаються алгоритмом із навчальних даних. Зокрема, завдяки техніці градієнтного зворотного поширення, яка дозволяє коригувати параметри відповідно до значення градієнта функції втрат. Функція втрат обчислює похибку між прогнозованим і цільовим значенням.

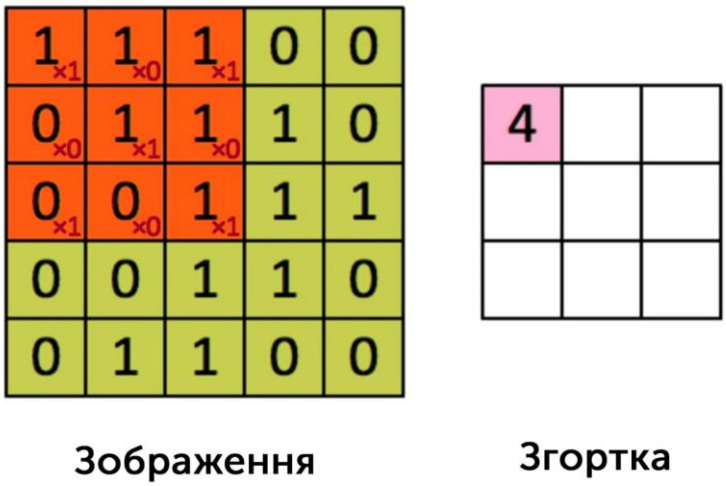


Рисунок 1.13 – Принцип роботи згорткового фільтра

Систематичне застосування одного й того самого фільтра на зображенні є фундаментальною ідеєю. Якщо фільтр призначений для виявлення певного типу об’єкта у вхідних даних, то систематичне застосування цього фільтра по всьому вхідному зображенню дає фільтру можливість виявити цю функцію в будь-якому місці зображення.

Результатом одноразового множення фільтра на вхідний масив є одне значення. Оскільки фільтр застосовується кілька разів до вхідного масиву, результатом є двовимірний масив вихідних значень, які представляють фільтрацію вхідних даних. Таким чином, двовимірний вихідний масив цієї операції називається «картою ознак».

Після створення карти об’єктів можна передати кожне значення в карті ознак через нелінійність, таку як ReLU, так само, як це робиться для вихідних даних повністю підключеного шару.

1.4 Моделі мереж для розпізнавання поз людини

1.4.1 Оцінки пози людини

Техніка комп’ютерного зору, яка визначає положення об’єкта чи людини на зображенні чи відео, називається оцінкою пози. Процес здійснюється шляхом

перегляду комбінації орієнтацій пози для даного предмета або людини. Виходячи з орієнтації точок, можна порівнювати різні моменти та пози людини чи об’єкта та робити деякі ідеї.

Оцінка пози в основному здійснюється шляхом визначення ключових точок об’єкта/люди або навіть визначення місця розташування:

Для об’єктів: ключовими точками будуть кути або краї об’єкта.

Для зображень: зображення людей, де ключовими точками можуть бути кисті рук, плечі, голова, ступні тощо.

1.4.2 Застосування оцінки пози

Наразі існує безліч рішень, які використовують технології комп’ютерного зору. Завдяки ефективній системі відстеження та вимірювання оцінки пози, інтерес до технологій оцінки пози знаходиться на достатньо високому рівні. Нижче представлені деякі застосування оцінки пози на прикладі:

1) Метавсесвіт доповненої реальності

Концепція метавсесвіту стає дедалі розповсюджуваною та активно розвивається за допомогою науково-технічного співтовариства. Вона привертає загальну увагу усіх вікових груп. Технології доповненої реальності прикріплюють тривимірні елементи до об’єкта або людини в реальному світі, для того, щоб вони виглядали реальними. Метавсесвіт створює середовище для людей, у яке можна перейти з реального світу. Тому технології оцінки пози, відстеження очей, голосу та обличчя знайшли своє застосування при розробці метавсесвіту.

2) Промисловість охорони здоров'я та фітнесу.

Швидке зростання фітнес-індустрії в часи глобальної епідемії вплинуло на багатьох споживачів, які активно приєднуються до виконання вправ задля підтримання здорового способу життя. Стрімке збільшення додатків для фітнесу, що надають ефективні діаграми моніторингу здоров’я та плани занять для покращення форми. Крім того, деякі програми забезпечують неймовірні результати у виявленні помилок при виконанні, а також зворотній зв’язок зі

споживачем. Ці програми використовують технології оцінки пози із застосуванням комп'ютерного зору, щоб звести до мінімуму можливість травми під час тренування. Також одним із корисних випадків використання оцінки пози використовується в оборонній сфері, де допомагає розрізняти ворогів та дружні війська.

3) Робототехніка

Оцінки пози інтегровані в робототехніку. Вони застосовуються при навчанні роботів, при якому вони вивчають рухи людей.

1.4.3 Причини використання оцінки пози

Виявлення людей відіграє велику роль у процесі розпізнавання. Завдяки нещодавній розробці алгоритмів машинного навчання (ML) стало достатньо легко використовувати виявлення пози та відстеження пози.

У традиційних методах, таких як виявлення об'єктів, зазвичай розпізнаий об'єкт виділяється лише обмежувальним квадратом. Завдяки вдосконаленню виявлення і відстеження пози, машини можуть легко вивчати мову тіла людини. За допомогою оцінки пози стає можливим відстежувати об'єкт на достатньо детальному рівні. Ці потужні методи відкривають широкий спектр можливостей для застосування в реальному світі.

Для відстеження рухів і активності людини оцінки поз мають кілька діапазонів застосування, наприклад, доповнену реальність, сектори охорони здоров'я та робототехніка. Наразі оцінка пози людини може використовуватися різними способами, наприклад, підтримка соціальної дистанції в чергах банку шляхом поєднання оцінки пози людини та проекції відстані. Це може допомогти людям у дотриманні правил і норм належної гігієни в банках, а також підтримувати фізичну дистанцію в переповненому людному місці.

Інший приклад доцільності та ефективності відстеження та оцінки пози, — це безпілотні автомобілі. Більшість аварій, спричинених самокерованими автомобілями, стаються коли транспортні засоби не можуть зрозуміти поведінку

пішоходів. За допомогою використання технологій оцінки пози модель може краще навчатися.

1.4.4 Методи оцінки поз кількох осіб

Для оцінки пози використовуються два поширені методи:

1) Підхід «зверху вниз»:

Спочатку алгоритм виявляє людину та створює обмежувальну рамку навколо кожної розпізнаної людини. Наступним кроком проводиться оцінка частин тіла. Після цього класифікується кожен суглоб людину.

2) Підхід «знизу вгору»:

Спочатку розпізнаються всі частини зображення, а потім зв’язуються/групуються частини, що належать окремим особам.

У більшості ситуацій підхід «зверху вниз» займає набагато більше часу, ніж підхід «знизу вгору» [30].

1.4.5 Розповсюджені моделі для оцінки пози

Завдяки швидкому прогресу розробки рішень глибокого навчання останніми роками, він випередив деякі підходи комп’ютерного зору у великій кількості завдань, таких як оцінка пози, включаючи сегментацію зображення та виявлення об’єктів.

Існує кілька моделей для оцінки пози. Вибір моделі залежить від вимоги задачі. Також присутні безліч факторів, які мають бути врахованими при виборі моделей. Вимогами можуть бути час роботи, розмір моделі та багато іншого.

Нижче перераховані найпопулярніші бібліотеки оцінки поз, що доступні для загального використання. Вони легко налаштовуються відповідно до випадку використання:

1) Deep Pose
2) PoseNet

3) Blaze pose
4) High-Resolution Net (HRNet)
5) Dense pose
6) Deep cut
7) Regional Multi-Person Pose Estimation (AlphaPose)
8) OpenPose

Найбільш розповсюдженими моделями, що використовуються є AlphaPose, HRNet, OpenPose, та Blaze pose

1) AlphaPose — це система оцінки пози людини в режимі реального часу для кількох осіб. Набір даних AplhaPose використовує доволі розповсюджений метод «зверху вниз» для оцінки пози людини. AlphaPose проявляє свої переваги у ситуації коли є неточні людські рамки, і робить це з достатньо високою точністю.

За допомогою AlphaPose можна розпізнати одну людину або групу людей за зображеннями чи відео. Для цього використовується трекер пози, відомий як PoseFlow, і він також має відкритий вихідний код.

2) HRNet відома як High-Resolution Net. Нейронна мережа HRNet використовується саме для оцінки пози людини. HRNet – це найсучасніший алгоритм в області оцінки пози людини. HRNet широко використовується для виявлення пози людини в телевізійних видах спорту.

HRNet використовує згорткові нейронні мережі (CNN). Це дозволяє масштабуватись, використовуючи великі набори даних зображень, на відміну від звичайних нейронних мереж, які працюють дуже повільно, оскільки всі приховані шари з’єднані один з одним, що призводить до повільного виконання моделі. Наприклад, при обробці зображення розміром 32323 пікселя, будуть використовуватись 3072 ваги, які є занадто великими та важкими для обчислень. Під час подачі цих даних це сповільнює нейронну мережу.

3) OpenPose, відомий як оцінка пози людини на основі відео, має відкритий вихідний код. Архітектура OpenPose популярна для передбачення оцінки пози для кількох осіб. Однією з переваг OpenPose є виявлення кількох осіб у реальному часі. Модель використовує підхід «знизу вгору» оцінки пози для кількох осіб.

OpenPose забезпечує високу точність в оцінці пози для тіла, стопи, рук і обличчя ключових точок. У загальному розпізнаються 135 ключових точок на одному зображенні. Основною перевагою OpenPose є API, який надає користувачеві контроль над вибором джерела зображення та надає кілька опцій. Є можливість вибирати зображення з полів камер, веб-камер і вбудованих систем (камер і систем відеоспостереження).

OpenPose має високу точність без шкоди для продуктивності виконання, тобто має велику швидкість при достатньо високому рівні точності (рис. 1.14).

Рисунок 1.14 – Алгоритм роботи OPenPose

4) BlazePose — це модель машинного навчання (ML), яку можна використовувати з ailia SDK. BlazePose розроблений компанією Google. Дана модель може обчислювати до 33 скелетних ключових точок, і в основному використовується у фітнес-додатках. BlazePose надає 33 ключові точки як вихідні дані моделі згідно з визначеною умовою впорядкування. Це дозволяє визначати семантику тіла лише на основі передбачення пози, що складається з моделей обличчя та рук. Модель BlazePose можна використовувати для створення додатків AI за допомогою ailia SDK, а також багатьох інших готових до використання моделей ailia.

1.4.6 Категоризація оцінки пози

Загалом оцінки пози діляться на дві групи:

1) Оцінка однієї пози: виявлення окремого об’єкта чи людини на зображенні чи відео
2) Оцінка кількох поз: виявлення кількох об’єктів або людей на зображенні чи відео

Також, існують різні види оцінки пози:

1) Оцінка пози людини
2) Оцінка жорсткої пози
3) 2D оцінка пози
4) 3D оцінка пози
5) Оцінка пози голови
6) Оцінка пози руки

Нижче детальніше розглянуто кожен з видів оцінки пози:

1) Оцінка пози людини

Оцінка ключових точок під час роботи з зображеннями або відео людини, де ключовими точками можуть бути лікті, пальці, коліна тощо.

2) Оцінка жорсткої пози

Об'єкти, які не підпадають під категорію гнучких об'єктів, є жорсткими об'єктами. Наприклад, краї будівельних блоків завжди будуть на однаковій відстані, незалежно від орієнтації цих блоків, тому прогнозування положення цих об’єктів відоме як оцінка жорсткої пози.

3) Оцінка 2D пози

Дана оцінка передбачає оцінку ключових точок зображення з точки зору рівня пікселів. У 2D-оцінці оцінюються ключові точки розташування в 2D-просторі вхідних даних. Розташування ключових точок представлено як X і Y.

4) Оцінка 3D пози

Передбачає тривимірне просторове розміщення всіх об’єктів або людей як кінцевий результат. Проводиться перетворення об’єкта в 3D-зображення з 2D-

зображення за допомогою додаткової осі z для передбачення результату (рис. 1.15).

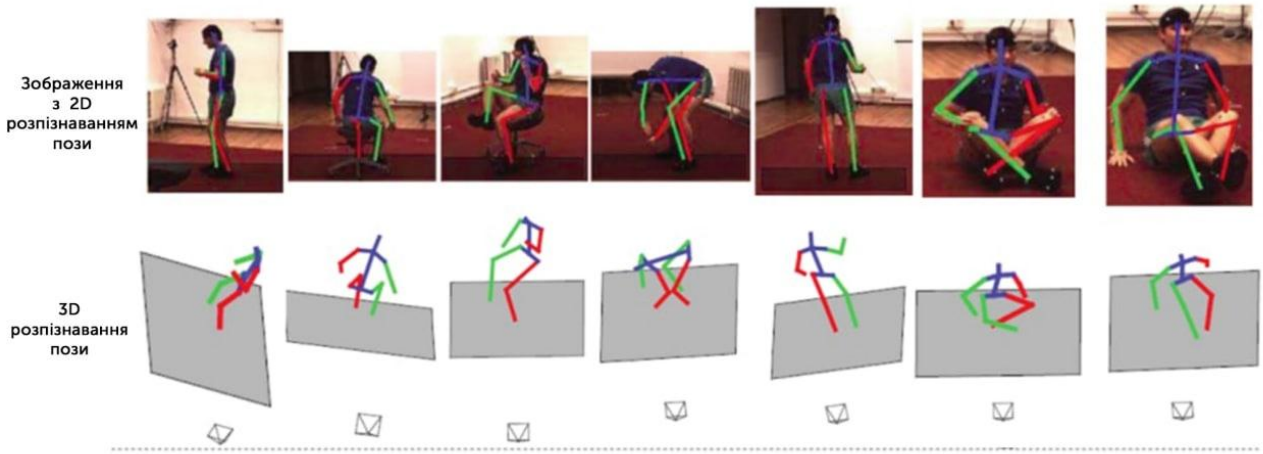


Рисунок 1.15 – Схема роботи розпізнавання пози людини у 2D та 3D

5) Оцінка пози голови

Знаходження положення голови людини набуває популярності у сфері використання комп’ютерного зору (рис. 1.16). Одним з найкращих застосувань оцінки пози голови є Snapchat, де використовуються різні види фільтрів на обличчі. Крім того, оцінка пози голови використовується у фільтрах Instagram, а також в самокерованих автомобілях, щоб відстежувати дії водіїв за кермом.

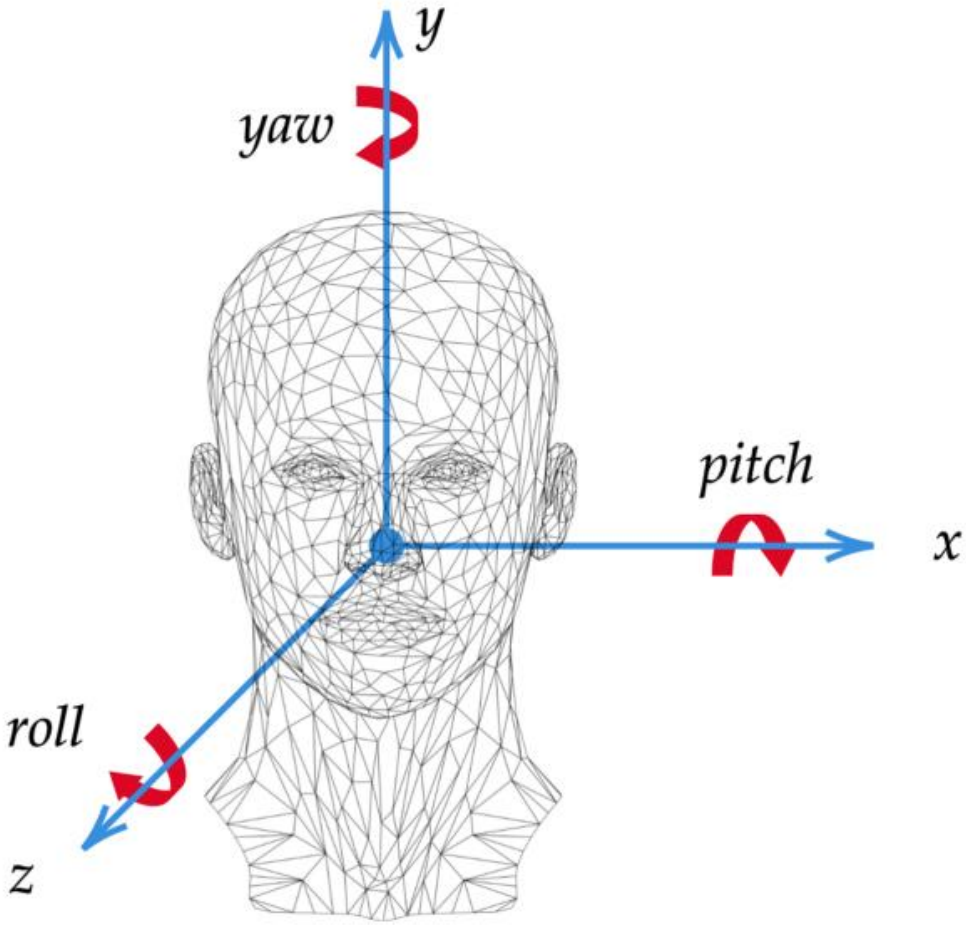


Рисунок 1.16 – Розпізнавання пози голови людини за допомогою моделі HopeNet

Оцінка пози голови має кілька застосувань, наприклад, моделювання розташування точок в 3D іграх. Щоб оцінити позу голови в 3D, потрібно знайти ключові точки обличчя з поз 2D зображення за допомогою моделей глибокого навчання.

6) Оцінка пози руки

Оцінка положення руки спрямована на прогнозування положення суглобів на руці за вхідним зображенням, і вона стала популярною через появу технологій MR/AR/VR. При оцінці пози кисті ключовими моментами є розташування

суглобів на руках. Всього рука має 21 ключову точку, яка складається з зап'ястя, і 5 пальців для розташування ключових точок (рис. 1.17).

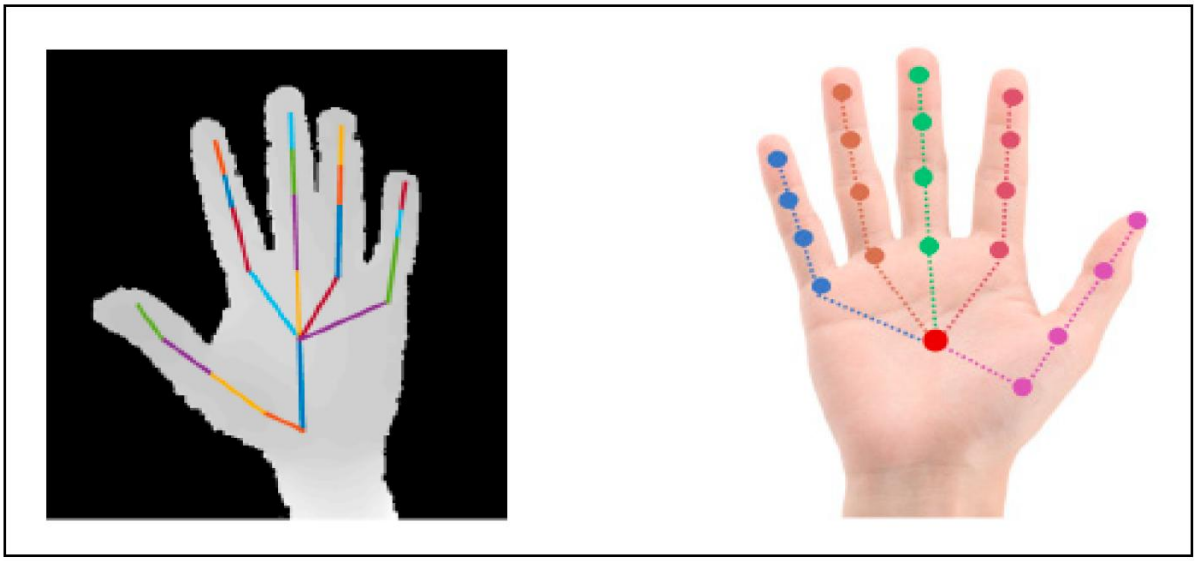


Рисунок 1.17 – Розпізнавання пози руки людини

1.4.7 Набори даних оцінки пози людини

Для оцінки пози людини за зображенням або відео доступні різні набори даних. Нижче представлені найбільш популярні з них:

1) MPII
2) COCO
3) HumanEva
4) Human3.6M
5) SURREAL – Synthetic hUmans foR REAL tasks

1) Набір даних про позу людини MPII містить дані численних поз людини, який включає близько 25 тис. зображень, що містять понад 40 тис. людей з анотованими суглобами тіла. Набір даних про пози людини MPII є найсучаснішим еталоном для оцінки артикульованої пози людини (рис. 1.18).

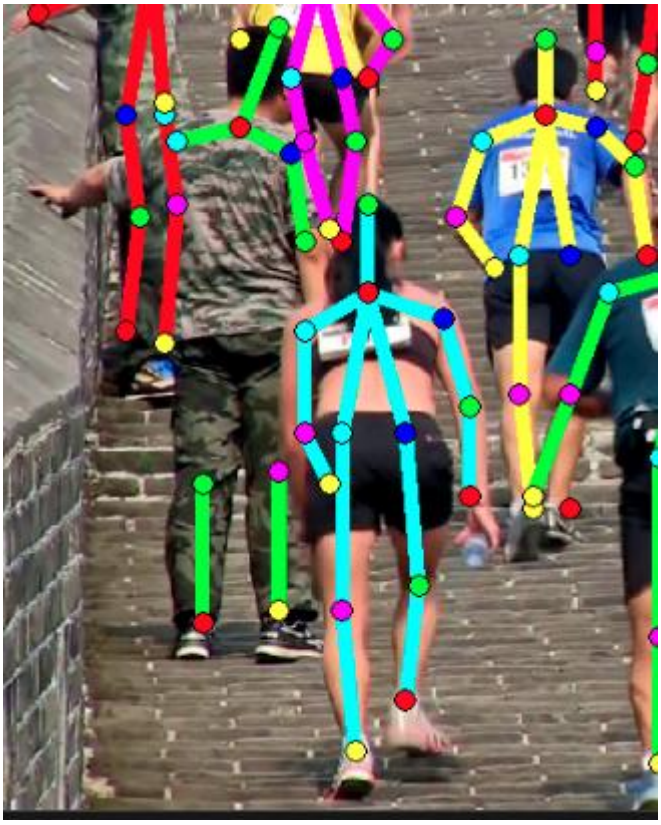


Рисунок 1.18 – Приклад розпізнавання пози людини за допомогою набору даних MPII

Першим набором даних, який був представлений у 2014 році, був MPII, і це також перший набір даних, який містить такий різноманітний діапазон поз. Дані містили зображення людської діяльності, зібрані з відео YouTube. 410 зображень діяльності людини анотовані та представлені в одній добірці. Кожне зображення було вилучено з відео YouTube і забезпечено не анотованими попередніми та наступними кадрами. Крім того, для тестового набору було представлено більш детальні анотації, включаючи оклюзії частин тіла та 3D орієнтації тулуба та голови.

2) COCO – це великомасштабний набір даних для виявлення об’єктів, сегментації та даних підписів, а також набір даних, який використовується для оцінки 2D-пози для кількох осіб із зображеннями, зібраними з Flickr. Набір даних COCO містить 80 категорій об’єктів і 91 категорію матеріалів.

Одним з найбільших наборів даних оцінки 2D пози на сьогоднішній день є COCO, і він є еталоном для тестування алгоритмів оцінки 2D пози (рис. 1.19).

Рисунок 1.19 – Приклад розпізнавання пози людини за допомогою набору даних COCO

3) Набір даних HumanEva складається з відеопослідовностей оцінки 3D-пози людини, які записуються за допомогою різних камер, таких як кілька RGB і ч/б камер. HumanEva була першим великим набором даних 3D-оцінки пози.

Набір даних HumanEva-I містить 7 відеопослідовностей: 4 у відтінках сірого та 3 кольорових. Вони синхронізовані з тривимірними позами тіла, отриманими за допомогою системи захоплення руху. База даних HumanEva містить 4 суб'єкта, які виконують 6 звичайних дій (наприклад, ходьба, біг підтюпцем, жестикуляція тощо). Також надаються показники помилок для обчислення помилки в 2D і 3D позах. Набір даних містить набори для навчання, перевірки та тестування.

4) Назва SURREAL походить від Synthetic Humans for REAL Tasks. SURREAL містить віртуальні відеоанімації для однієї особи у вигляді 2D та 3D оцінки пози. Дана модель створювалась за допомогою даних motion capture, записаних у лабораторії. Вона також є першим великомасштабним набором даних розпізнавання людини, який генерує глибину, частини тіла, оптичний потік та 2D/3D пози.

Набір даних містить 6 млн кадрів штучних людей. Зображення є фотореалістичними зображеннями людей із великими варіаціями форми, текстури, точки зору та пози. Для забезпечення реалістичності штучні тіла створені з використанням моделі тіла SMPL, параметри якої підібрані за методом MoSh на основі необроблених даних 3D-маркера MoCap.

Висновки до розділу:

У першому розділі проаналізовано історичні передумови створення систем штучного інтелекту. Розглянуто засади побудови систем нейронних мереж. Загальною характеристикою багатьох видів контрольованих і неконтрольованих моделей глибокого навчання є те, що ці моделі мають багато шарів прихованих нейронів, які навчаються у поєднанні зі зворотним поширенням і градієнтами помилок стохастичного градієнтного спуску.

Фундаментальною перевагою нейронних мереж серед стандартних підходів до комп'ютерних алгоритмів є здатність мережі дуже точно класифікувати дані, регулюючи міцність зв'язку між їхніми нейронами, тобто змінюючи значення вагових коефіцієнтів.

Розглянуто принципи технології розпізнавання зображень, що являє собою набір методів для виявлення та аналізу зображень для автоматизації певного завдання. Розпізнавання фото- або відеоматеріалві може виконуватися з різним ступенем точності. Модель здатна виявити певний елемент або віднести зображення до якоїсь великої категорії за відповідною ознакою.

Для задач з розпізнавання зображень використовують методи глибокого навчання, що є підкатегорією машинного навчання, яка відноситься до набору методів і технологій автоматичного навчання на основі штучних нейронних мереж.

Проведено огляд основних архітектур нейронних мереж. Серед них архітектури прямого поширення, рекурентні та згорткові. Найпростішою формою нейронної мережі прямого поширення є одношаровий персептрон. У нейронній мережі прямого поширення інформація рухається лише в одному напрямку — від вхідного шару через приховані шари до вихідного шару. Інформація переміщується прямо через мережу і ніколи не проходить через один вузел двічі.

Рекурентна нейронна мережа — це тип штучної нейронної мережі, який використовує послідовні дані або дані часових рядів. У RNN інформація

проходить по циклу. Коли система приймає рішення, вона враховує поточні вхідні дані, а також те, що вона дізналась з вхідних даних, які вона отримала раніше.

У глибокому навчанні згорткова нейронна мережа являє собою клас глибоких нейронних мереж, які найчастіше застосовуються для аналізу візуальних зображень. Архітектура згорткової нейронної мережі формується послідовністю будівельних блоків для виділення ознак, які розрізняють належний клас образу від інших. Особливістю CNN є те, що вхідні дані CNN не є абсолютними числовими значеннями пікселів зображення. Замість цього повне зображення поділено на невеликі набори, де кожен набір діє як нове зображення. Розглянуто набір мереж для розпізнавання поз людини. Серед них найбільш використовуваними є HRNet, OpenPose та BlazePose.

2 ТЕОРЕТИЧНІ ЗАСАДИ

2.1 Python

Для розробки рішення з розпізнавання об'єктів доцільно використовувати мову програмування Python. Python – це мова комп’ютерного програмування, яка використовується для створення програмного забезпечення і веб-сайтів, автоматизації і аналізу даних. Python можна використовувати для створення різноманітних програм, тобто вона є мовою загального призначення. Python не спеціалізується на будь-яких конкретних задачах. Ця універсальність разом зі зручністю для людей, що тільки починають або зовсім не мали справу з програмуванням зробили її однією з найбільш використовуваних мов програмування сьогодні. Python успішно використовується багатьма працівниками фінансової сфери та вченими для виконання різноманітних повсякденних завдань [31]–[33].

Деякі зі сфер використання Python:

1) Розробка веб-додатків та веб-сайтів
2) Тестування програмного забезпечення
3) Аналіз даних
4) Автоматизація або написання сценаріїв
5) Машинне навчання
6) Повсякденні завдання

Python став основною мовою у сфері науки о даних (data science), дозволяючи аналітикам даних та іншим фахівцям проводити складні статистичні обчислення, створювати візуалізації даних, створювати алгоритми машинного навчання, аналізувати дані, а також виконувати інші завдання, пов’язані з даними.

Бібліотеки Python дозволяють створювати широкий спектр різноманітних візуалізацій даних, наприклад: гістограми, кругові діаграми, та тривимірні графіки. Також Python дозволяє програмістам писати програми для аналізу даних

і машинного навчання швидше та ефективніше, наприклад за допомогою TensorFlow і Keras.

Python часто використовується для розробки внутрішнього функціоналу веб-сайту або програми. Python у веб-розробці може виконувати сервісні функції, такі як: надсилання даних на сервери та відправлення даних з серверу у бік користувача, обробку даних і зв’язок з базами даних, маршрутизацію URL-адрес і забезпечення безпеки. Python пропонує кілька фреймворків для веб-розробки, таких як: Django і Flask.

Сфери веб-розробки, які використовують Python, включають інженерів серверної частини, інженерів повного стека, розробників Python, інженерів програмного забезпечення та інженерів DevOps.

2.2 TensorFlow

TensorFlow надає набір робочих інструментів для розробки та навчання моделей з використанням мови Python або JavaScript. Також дозволяє доволі швидко проводити розгортання в хмарному середовищі, локально, в браузері або на пристрої, незалежно від того, яку мова використовується [34].

TensorFlow дозволяє навчати та ініціалізувати глибокі нейронні мережі для рукописної класифікації, розпізнавання зображень, моделей від послідовності до послідовності для машинного перекладу, обробки природньої мови та моделювання на основі PDE (діференційне рівняння з частковими частинами) [35].

TensorFlow пропонує кілька рівнів абстракції, для більш чіткого вибору за потребами. TensorFlow дозволяє навчати моделі за допомогою високорівневого API Keras, що полегшує роботу з машинним навчанням.

TensorFlow надає гнучкість і контроль за допомогою таких функцій, як Keras Functional API і Model Subclass API для створення складних топологій. Для легкого створення прототипів і швидкого налагодження можна використовувати eager execution [36].

TensorFlow також підтримує екосистему потужних бібліотек надбудов і моделей для експериментів, включаючи Ragged Tensors, TensorFlow Probability, Tensor2Tensor і BERT.

Програми TensorFlow можна запускати майже на будь-якому пристрої: локальній машині, кластері в хмарі, пристроях iOS і Android, ЦП або графічних процесорах. Якщо використовувати власну хмару Google, можна запустити TensorFlow на спеціальному блоці обробки TensorFlow (TPU) від Google для подальшого прискорення. Отримані моделі, що створені за допомогою TensorFlow, можна розгорнути на будь-якому пристрої, де вони будуть використовуватися для прогнозування [37].

Найбільшою перевагою, яку TensorFlow надає для розвитку машинного навчання, є абстракція. Замість того, щоб мати справу з дрібними деталями реалізації алгоритмів або з'ясовувати правильні способи прив'язати вихід однієї функції до входу іншої, розробник може зосередитися на загальній логіці програми.

TensorFlow пропонує додаткові функції для розробників. Набір візуалізації TensorBoard дає змогу перевіряти та профілювати роботу TensorFlow за допомогою графіків на інтерактивній веб-інформаційній панелі.

TensorFlow також має багато переваг завдяки підтримці Google. Google не лише інвестує у розвиток проекту, але й створює багато важливих сервісів навколо TensorFlow, які полегшують розгортання та використання. Наприклад віртуальний графічний процесор TPU для прискореної роботи в хмарі Google, онлайн-хаб для обміну моделями, створеними за допомогою фреймворка, фреймворк для браузера та мобільних пристроїв; і багато іншого [38].

Деталі реалізації TensorFlow ускладнюють отримання повністю детермінованих результатів навчання моделі. Іноді модель, яка була навчена на одній системі, буде дещо відрізнятися від моделі, навченої на іншій, навіть якщо вони отримують однакові вхідні дані. Причини цього є доволі динамічними, наприклад, як і куди заповнюються випадкові числа, чи певна недетермінована поведінка під час використання графічних процесорів). Тим не менш, у більшості

випадків ці проблеми можна обійти, і команда TensorFlow розглядає додаткові засоби контролю, щоб впливати на детермінізм у робочому процесі.

2.3 Mediapipe

Широкий спектр програм машинного навчання сьогодні покладається на кілька основних базових завдань машинного навчання. Наприклад, і навігація жестами, і розпізнавання мови жестів покладаються на здатність програми ідентифікувати та відстежувати руки людини. З огляду на те, що створення моделі відстеження рук займає багато часу та ресурсів, існують проблеми розвитку у створенні всіх програм, які покладаються на відстеження частин тіла людини. Щоб вирішити цю проблему, Google винайшов MediaPipe. MediaPipe — це проект Google, який пропонує «кросплатформні рішення з відкритим кодом, які можна налаштувати для роботи в релаьному часі та у форматі потокового медіа. MediaPipe надає доступ до широкого спектру потужних моделей машинного навчання, створених з урахуванням апаратних обмежень мобільних пристроїв [39].

MediaPipe надає моделі машинного навчання для доволі простих завдань, таких як відстеження окремих частин тіла, таким чином вирішуючи проблеми розвитку, які існують для багатьох програм машинного навчання. Ці моделі, а також їхні надзвичайно прості у використанні API, у свою чергу, спрощують процес розробки та скорочують тривалість розробки проекту для багатьох програм, які покладаються на комп’ютерний зір [40]–[42]. Ці моделі включають:

1) Анатомічні моделі
2) Відстеження сітки обличчя
3) Сегментація волосся
4) Сегментація зображення з передньої камери
5) Відстеження пози
6) Моделі сегментації
7) Відстеження рук

8) Holstic Tracking

9) Об'єктні моделі

10) Виявлення/відстеження 2D об'єктів

11) Виявлення 3D об’єктів та оцінка пози

Google MediaPipe з відкритим кодом було вперше представлено в червні 2019 року. Метою проекта є спрощення розробки проектів з використанням комп’ютерного зору способом надання деяких інтегрованих функцій комп’ютерного зору та машинного навчання. MediaPipe — це фреймворк для створення мультимодальних (наприклад, відео, аудіо), кросплатформних (наприклад, Android, iOS, веб) застосованих конвеєрів ML. Mediapipe також полегшує розгортання технології машинного навчання в демонстраціях і додатках на різноманітних апаратних платформах. Тобто MediaPipe є кросплатформенною платформою, що працює на настільних комп’ютерах/серверах, Android, iOS та вбудованих пристроях, таких як Raspberry Pi та Jetson Nano.

MediaPipe дозволяє ефективно керувати використовуваними ресурсами (ЦП і графічним процесором), щоб досягти продуктивності з низькою затримкою відповіді та обробляти синхронізацію даних, таких як аудіо- та відеокадри. MediaPipe абстрагує кожну модель сприйняття в модуль і з'єднує їх. MediaPipe також підтримує механізми роботи з TensorFlow і TF Lite. У MediaPipe можна використовувати будь-яку модель TensorFlow і TF Lite. Важливою перевагою MediaPipe є підтримка прискорення GPU самого пристрою мобільних платформ.

2.4 Flask

Flask — це легкий інтерфейс веб-серверного шлюзу, WSGI-фреймворк веб-додатків, який був створений, щоб полегшити роботу з веб-розробкою. Фреймворк — це сховище коду, яке має допомогти розробникам досягти необхідного результату, спрощуючи роботу, масштабованість, ефективність і обслуговування веб-додатків, надаючи багаторазовий код або розширення для звичайних операцій [43]. Flask має підтримку вбудованого масштабування та

збільшення складності проектів. Наразі, Flask став одним із найпопулярніших фреймворків веб-додатків Python [44].

Flask надає розробникам різні варіанти вибору під час розробки веб-додатків. Він надає інструменти, бібліотеки та механіку, які дозволяють створювати веб-додаток, але без встановлювання жодних залежностей та жорстких обмежень до вигляду проекта.

Веб-додаток, що розроблений за допомогою Flask може мати будь-яке практичне направлення, наприклад: бути блогом, комерційним веб-сайтом або веб-сторінками [45]. Фреймворк має підтримку використання розробниками деяких розширень, надані спільнотою, що дозволяє додати більше функціональних можливостей до веб-додатку.

Як було зазначено раніше, Flask класифікується як мікро-фреймворк. Зазвичай мікро-фреймворк — це фреймворк з відсутністю залежностей від зовнішніх бібліотек. У будь-якому випадку, оскільки Flask – це фреймворк для розробки веб-додатка, він має свої переваги і недоліки.

Перевагою використання Flask як платформи для розробки веб-програми, — це майже відсутність залежності від оновлення та виявлення помилок безпеки.

Недоліком використання Flask є те, що при додаванні розширень, збільшується список залежностей.

Flask заснований на Werkzeug, бібліотеці утиліти WSGI, і Jinja2, який є його механізмом шаблонів. Можна використовувати цю структуру веб-додатків для компіляції модулів і бібліотек, які також допоможуть писати веб-додатки без написання низькорівневого коду, такого як керування потоками та протоколами.

Для використання Flask потрібно створити веб-програму та підключити її до HTML. Щоразу, коли будь-який користувач надсилає інформацію в мережі або переходить до рядка пошуку, HTML виступає посередником для використання Flask. Фреймворк Flask шукає файли HTML (шаблони) у папці під назвою Templates. Перед відправкою шаблону виконується код Python, який вводить змінні та код.

Його також можна використовувати для створення соціальних мереж, платформ для ведення блогів або ж сайтів звичайного контенту. Однак, якщо задачею є створення об’ємного веб-додатку, є сенс вибрати інший веб-фреймворк, оскільки Flask є мікрофреймворком.

2.5 Web

JavaScript — це мова сценаріїв, яка дозволяє реалізувати складні функції на веб-сторінках [46]. Код виконується щоразу при завантаженні або роботі веб-сторінки, відображаючи своєчасні оновлення вмісту, інтерактивні карти, анімовані 2D/ 3D-графіки, прокрутки. JavaScript є третім шаром використання стандартних веб-технологій, наряду з HTML та CSS.

JavaScript робить веб-сторінки динамічними. До представлення JavaScript веб-сторінки створювали лише за допомогою HTML і CSS. HTML і CSS здатні створювати лише статичні сторінки, які можна стилізувати, але не інтерактивні. Велика кількість найпопулярніших веб-сайтів створені за допомогою JavaScript, наприклад: Google, YouTube та Facebook.

JavaScript дозволяє розробникам впроваджувати такі функції, як:

1) Відображення та приховування меню або інформації
2) Додавання анімації
3) Відтворення аудіо чи відео на веб-сторінці
4) Створення галерей зображень
5) Створення спадного меню та стилізованих динамічних меню
6) Додавання ефектів наведення
7) Збільшення або зменшення масштабу зображення

Завдяки великій колекції фреймворків JavaScript розробники можуть ефективно створювати додатки для мобільних пристроїв та Інтернету [47]. Фреймворки JavaScript — це бібліотеки попередньо написаного коду JavaScript, які розробники використовують для стандартних функцій.

Нижче перераховані ключові переваги JavaScript для використання при побудові веб-сайтів або веб-сторінок та приклади їх застосування:

1) Швидкість: швидше виконувати код у веб-браузері на локальному пристрої, ніж виконувати код на сервері та передавати оброблену інформацію на пристрій.
2) Незалежне перезавантаження певних частин сторінки: прикладом застосування є пропозиції продовження запросу при введенні текста в рядку пошуку пошукової системи.
3) Автозаповнення: JavaScript допомагає заповнювати адресу електронної пошти або особисті дані кожного разу при заповнені онлайн-форми.
4) Динамічні сторінки: сторінки, з якими користувачі можуть взаємодіяти.
5) Перевірка форми: обов’язкові для заповнення поля перевіряються JavaScript для того, щоб користувач не пропустив жодної інформації, наприклад при замовленні чогось на веб-сайті.
6) Відтворення аудіо та відео
7) Адаптивний вміст: JavaScript дозволяє змінювати розмір вмісту на сторінці при зміні розміру вікна браузера
8) Зменшення використання пам’яті: виконання коду в браузері допомагає звільнити місце на серверах, що допомагає скоротити витрати на забезпечення потужних серверів.
9) Створення адаптивних інтерфейсів користувача: майже всі інтерфейси користувачів соціальних мереж покладаються на JavaScript.

2.6 BlazePose

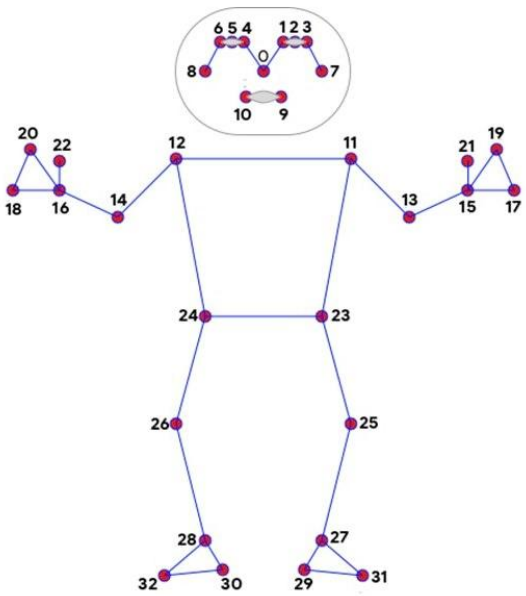
BlazePose (Full Body) — це модель виявлення пози, що була розроблена Google . Вона може обчислювати координати 33 ключових точок скелета у 3D-просторі. Цю модель можна використовувати у фітнес-додатках, або у будь-якому додатку, який потребує розпізнавання пози людини [48], [49].

BlazePose складається з двох моделей машинного навчання: детектора та оцінювача. Детектор вирізає область людини з вхідного зображення, тоді як оцінювач приймає зображення виявленої людини з роздільною здатністю 256x256 як вхід і виводить розпізнані ключові точки.

BlazePose виводить 33 ключові точки відповідно до визначеного впорядкування. Кількість точок BlazePose у два рази більша ніж 17 ключових точок набору даних COCO.

BlazePose на виході надає координати 33-ох ключових точок тіла людини, що описують приблизне розташування тіла (рис. 2.1):

1) Ніс
2) Праве око (3 ключові точки): внутрішній, центральний, зовнішній
3) Ліве око (3 ключові точки): внутрішній, центральний, зовнішній
4) Вуха (2 ключові точки): праве, ліве
5) Рот (2 ключові точки): правий кут, лівий кут
6) Плече (2 ключові точки): праве, ліве
7) Лікоть (2 ключові точки): правий, лівий
8) Зап'ястя (2 ключові точки): праве, ліве
9) Кістка пальця (2 ключові точки): права, ліва
10) Вказівний суглоб (2 ключові точки): праве, ліве
11) Кістки великого пальця (2 ключові точки): праве, ліве
12) Стегна (2 ключові точки): праве, ліве
13) Коліно (2 ключові точки): праве, ліве
14) Гомілковостопний суглоб (2 ключові точки): правий, лівий
15) Каблуки (2 ключові точки): праве, ліве
16) Стопи (2 ключові точки): права, ліва



Координати точок тіла

0) Ніс
1) Внутрішня ліва частина ока
2) Ліве око
3) Зовнішня частина лівого ока
4) Внутрішня права частина ока
5) Праве око
6) Зовнішня частина правого ока
7) Ліве вухо
8) Праве вухо
9) Ліва сторона рота
10) Права сторона рота
11) Ліве плече
12) Праве плече
13) Лівий лікоть
14) Правий лікоть
15) Ліве зап'ястя
16) Праве зап'ястя
17) Лівий мізинець
18) Правий мізинець
19) Вказівний палець лівої руки
20) Вказівний палець правої руки
21) Великий палець лівої руки
22) Великий палець правої руки
23) Ліве стегно
24) Праве стегно
25) Ліве коліно
26) Праве коліно
27) Ліва щиколотка
28) Права щиколотка
29) Ліва п'ята
30) Права п'ята
31) Індекс лівої стопи
32) Індекс правої стопи

Рисунок 2.1 – Список координат пози людини моделі BlazePose

Детектор — це архітектура на основі Single-Shot Detector (SSD). Враховуючи вхідне зображення розміром масива (1,224,224,3), детектор виводить обмежувальну рамку (1,2254,12) і оцінку точності (1,2254,1). 12 елементів обмежувальної рамки мають вигляд (x,y,w,h,kp1x,kp1y,…,kp4x,kp4y), де kp1x до kp4y є додатковими ключовими точками. Кожен із 2254 елементів має свою власну назву, масштаб прив’язки та зміщення, які потрібно застосувати.

Існує два способи використання детектора. У прямокутному режимі обмежувальна рамка визначається з її положення (x,y) і розміру (w,h). У режимі вирівнювання масштаб і кут визначаються з (kp1x,kp1y) і (kp2x,kp2y), що дозволяє передбачити обмежувальну рамку, включаючи нахил об’єкта.

Оцінювач використовує теплову карту для навчання, але обчислює ключові точки не використовуючи її для більш оптимізованого, а відповідно швидкого передбачення. Архітектура мережі відстеження показана на рис. 2.2.

Рисунок 2.2 – Архітектура мережі відстеження BlazePose

Перший вихідний результат оцінювача – це 1195 орієнтирів. Орієнтири складаються з 165 елементів для параметрів x, y, z, значення видимості та присутності для кожної з 33 ключових точок.

Значення z засновані на координатах стегна людини, причому ключові точки знаходяться між стегнами і камерою, коли значення z негативне, і позаду стегон, коли значення z є позитивним.

Видимість і присутність зберігаються в діапазоні min_float та max_float і перетворюються на значення ймовірності шляхом застосування сигмоїдної функції. Значення видимості повертає ймовірність розпізнавання ключових точок, які існують у кадрі і не перекриті іншими об’єктами. Значення наявності повертає ймовірність ключових точок, які існують у кадрі.

Модель BlazePose подібна до MobileNetV2 з налаштованими блоками для продуктивності в реальному часі.

На виході нейронної мережі отримуємо:

1) Тензор 33x5

2) Тензор 33x3
3) Скаляр, що вказує на ймовірність присутності особи на переданому зображенні.

У BlazePose використовується двоетапний конвеєр детектор-трекер ML, який довів свою ефективність у рішеннях MediaPipe Hands і MediaPipe Face Mesh. Використовуючи детектор, конвеєр спочатку визначає особу/позу, що представляє інтерес (ROI) у кадрі. Наступним кроком трекер прогнозує орієнтири пози і маску сегментації в межах ROI. Для випадків використання відео детектор викликається лише за потреби, тобто для самого першого кадру і коли трекер більше не може ідентифікувати присутність пози тіла в попередньому кадрі.

BlazePose має такі параметри:

1) STATIC_IMAGE_MODE – Якщо встановлено значення false, рішення розглядає вхідні зображення як відеопотік;
2) MODEL_COMPLEXITY – Складність моделі орієнтира пози: 0, 1 або 2. Точність орієнтиру, а також затримка висновку зазвичай зростають зі складністю моделі;
3) SMOOTH_LANDMARKS – Якщо встановлено значення true, фільтри рішення створюють орієнтири на різних вхідних зображеннях, щоб зменшити тремтіння, але ігноруються, якщо для static_image_mode також встановлено значення true. За замовчуванням значення true.;
4) ENABLE_SEGMENTATION – Якщо встановлено значення true, окрім орієнтирів пози, рішення також генерує маску сегментації. За замовчуванням false;
5) SMOOTH_SEGMENTATION – Якщо встановлено значення true, рішення фільтрує маски сегментації між різними вхідними зображеннями, щоб зменшити тремтіння. Ігнорується, якщо enable_segmentation – false або static_image_mode – true. За замовчуванням значення true;

6) MIN_DETECTION_CONFIDENCE – Мінімальне значення довіри ([0,0, 1,0]) з моделі виявлення людини, щоб виявлення вважалося успішним. За замовчуванням 0,5;
7) MIN_TRACKING_CONFIDENCE – Мінімальне значення довіри ([0,0, 1,0]) з моделі відстеження орієнтирів для того, щоб орієнтири пози вважалися відстеженими успішно, або в іншому випадку виявлення людини буде автоматично викликано на наступному вхідному зображенні. Встановлення більшого значення може підвищити надійність рішення за рахунок більшої затримки. Ігнорується, якщо static_image_mode має значення true, де виявлення людей просто виконується на кожному зображенні. За замовчуванням 0,5;

У свою чергу параметр «MODEL_COMPLEXITY» впливає на використовувану для розпізнавання модель. Значення «0» відповідає моделі BlazePose GHUM Lite, значення «1» – BlazePose GHUM Full, а значення «2» – відповідно використанню моделі BlazePose GHUM Heavy. Найдетальнішою та найточнішою моделлю є BlazePose GHUM Heavy, проте вона потребує достатньо великої потужності для своєї роботи, тому найчастіше використовують модель BlazePose GHUM Full, яка за своїми потребами до потужності та точністю розпізнавання є середньою між BlazePose GHUM Lite та BlazePose GHUM Heavy.

2.7Дані для тренування

Навчання нейронної мережі являє собою знаходження найкращого значення вагового коефіцієнта нейронних з’єднань завдяки циклу зворотного зв’язку, який також називається градієнтним зворотним поширенням (рис. 2.3).

Основною метою поділу даних на набори є запобігання надмірному тренуванню моделі. При надмірному тренуванні модель добре класифікує вибірки у навчальному наборі, але не може узагальнювати та робити точні класифікації даних, які вона раніше не бачила.

Рисунок 2.3 – Процес навчання нейронної мережі за допомогою навчального набору

Для навчання та тестування будь-якої моделі нейронної мережі дані повинні бути поділені на три окремих набори: тренувальний, навчальний та валідаційний.

Тренувальний набір – це набір даних, який використовується для того щоб навчити модель знаходити приховані шаблони в даних.

З кожною епохою навчання одні й ті самі дані за навчального набору циклічно надходять в архітектуру нейронної мережі. Таким чином модель вивчає особливості представлених даних.

Для того, щоб модель навчалася в більшій кількості можливих сценаріїв і могла передбачити будь-яку нову вибірку даних, яка з’явиться в майбутньому, навчальний набір повинен мати достатньо різноманітний набір вхідних даних.

Набір перевірки — це виокремлений від навчального набору набір даних. Його задача – це перевірка ефективності моделі під час навчання. Процес

перевірки надає дані, які допомагають краще налаштувати коефіцієнти та конфігурації моделі.

Алгоритм навчання достатньо примітивний – модель навчається на навчальному наборі та одночасно з цим оцінка моделі виконується на наборі перевірки після кожної епохи навчання. Три варіанти пропорцій поділу навчальних даних у навчальному наборі показані на рис. 2.4.

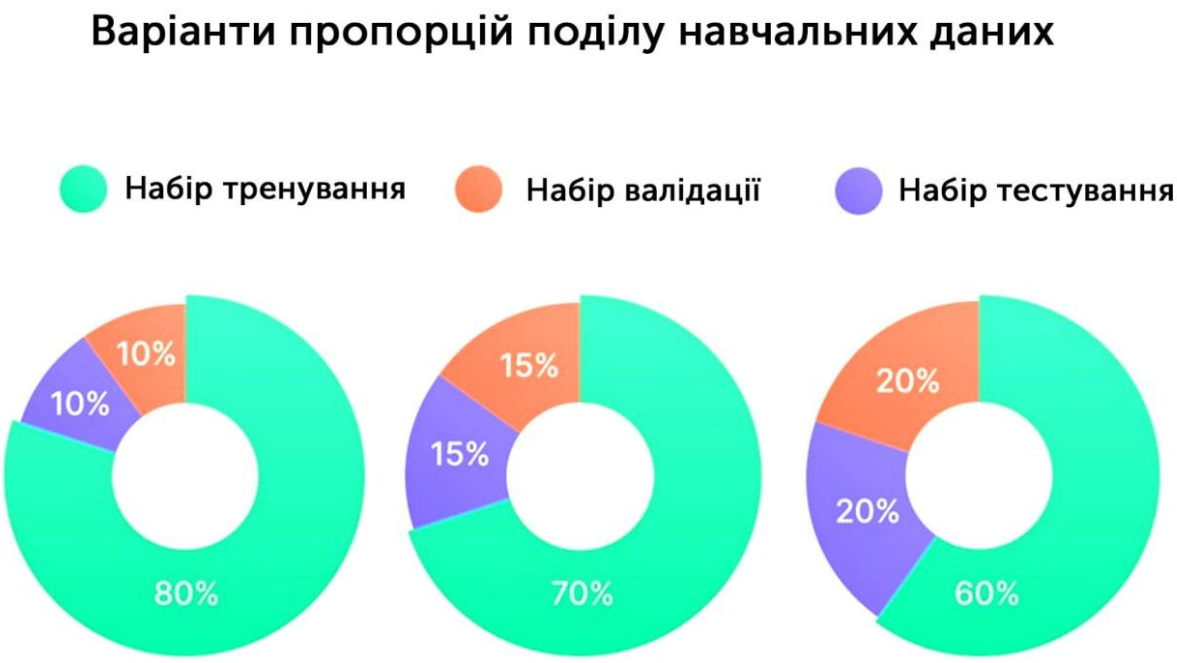


Рисунок 2.4 – Варіанту пропорцій поділу навчальних даних у навчальному наборі

2.7 Процесори для навчання нейронних мереж

Для завдань навчання нейронних мереж оптимальніше використовувати саме графічні, а не центральні процесори. Обумовлене це правило відмінністю у архітектурі роботи вищеназваних (рис. 2.5)

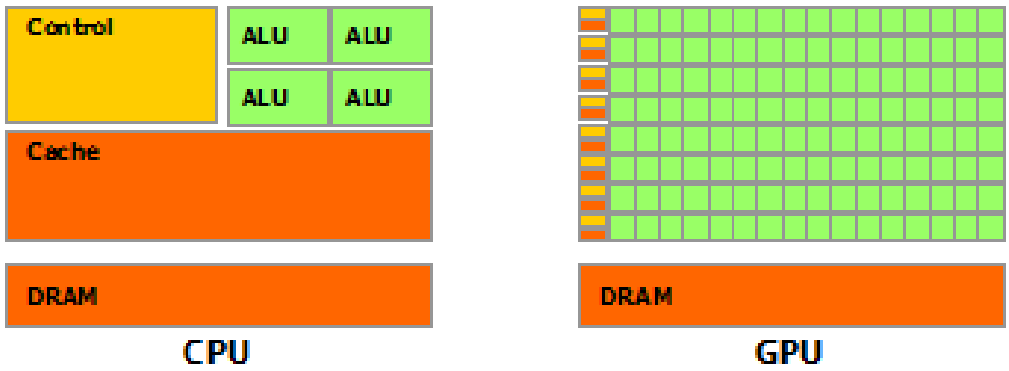


Рисунок 2.5 – Архітектура CPU та GPU

Графічні процесори значно відрізняються від звичайних процесорів. GPU (Graphics Processing Unit, графічний процесор) має більшу кількість арифметико-логічних пристроїв для обробки даних, у порівнянні з CPU (Central Processing Unit, центральний процесор) [50].

Якщо для процесора звичайна кількість ядер станом на 2021 рік – 8-16 ядер, то для графічних процесорів цей показник починається від 512 і вище.

Також присутні відмінності у принципах роботи: процесор є послідовним пристроєм, який чергу задач виконує одну за одною, у той час як графічний процесор є мультипоточним, тобто принципом роботи є паралельна обробка даних великою кількістю не дуже потужних ядер.

Нейронні мережі – це паралельні алгоритми, тому саме GPU використовують для машинного навчання.

Ще однією перевагою графічних процесорів є оптимізованість для матричних операцій – це основний тип операцій, який необхідний нейронним мережам для отримання результату. Для оцінки ефективності GPU у порівнянні з CPU доцільно використовувати параметр кількості операцій з плаваючою комою в секунду (рис. 2.6)

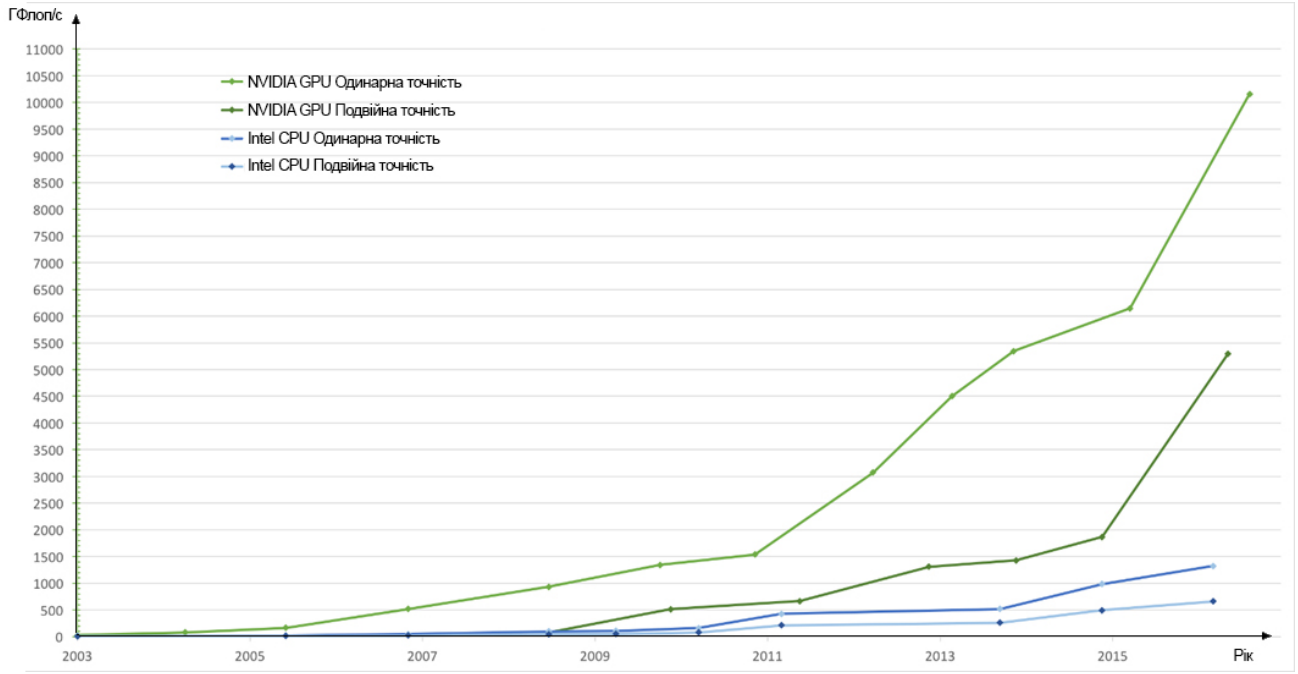


Рисунок 2.6 – Швидкість операцій з плаваючою комою в секунду для CPU та GPU

Навчання багатошарових нейронних мереж на CPU може займати тижні, у той час як графічні процесори з тими ж операціями можуть впоратись за один день. При цьому, у перерахунку графічні процесори споживають менше енергії, що говорить про більший ККД обчислень.

Висновки до розділу:

У другому розділі розглянуто інструменти розробки та навчання нейронних мереж. Для розробки рішення з розпізнавання об'єктів доцільно використовувати мову програмування Python. Python – це мова комп’ютерного програмування, яка використовується для створення програмного забезпечення, зокрема, нейронних мереж.

Розглянуто бібліотеку TensorFlow для Python. TensorFlow надає набір робочих інструментів для розробки та навчання моделей з використанням мови Python. TensorFlow дозволяє навчати моделі за допомогою високорівневого API Keras, що полегшує роботу з машинним навчанням. Однією з основних переваг TensorFlow є можливість запускати програми, створені за допомогою цієї бібліотеки майже на будь-якому пристрої: локальній машині, кластері в хмарі, пристроях iOS і Android, ЦП або графічних процесорах. Однак особливістю TensorFlow є деталі реалізації TensorFlow, які ускладнюють отримання повністю детермінованих результатів навчання моделі. Іноді модель, яка була навчена на одній системі, буде дещо відрізнятися від моделі, навченої на іншій, навіть якщо вони отримують однакові вхідні дані.

Створення моделі відстеження рук займає багато часу та ресурсів, існують проблеми розвитку у створенні всіх програм, які покладаються на відстеження частин тіла людини, тому для побудови нейронних мереж доцільно використовувати MediaPipe. MediaPipe надає моделі машинного навчання для доволі простих завдань, таких як відстеження окремих частин тіла, таким чином оптимізуючи програми машинного навчання. MediaPipe можна використовувати з будь-якою моделлю TensorFlow і TF Lite. Важливою перевагою MediaPipe є підтримка прискорення GPU самого пристрою мобільних платформ.

Для побудови веб-частини проекту доцільно використовувати фреймворк Flask. Flask надає інструменти, бібліотеки та механіку, які дозволяють створювати веб-додаток, але без встановлювання жодних залежностей та жорстких обмежень до вигляду проекта.

Обумовлено доцільність використання JavaScript разом з Flask для побудови інтерфейсу веб-сайту проекта. Важливою перевагою JavaScript є можливість робити веб-сторінки динамічними.

Розглянуто використання моделі виявлення пози BlazePose. BlazePose виводить 33 ключові точки відповідно до наведеного нижче порядку впорядкування.

Проведено огляд процесу створення даних для тренування нейронної мережі. Для навчання та тестування будь-якої моделі нейронної мережі дані повинні бути поділені на три окремих набори: тренувальний, навчальний та валідаційний.

Обумовлено використання графічних процесорів для процесу навчання нейронної мережі. Процесор є послідовним пристроєм, який чергу задач виконує одну за одною, у той час як графічний процесор є мультипоточним, тобто принципом роботи є паралельна обробка даних великою кількістю не дуже потужних ядер. Нейронні мережі – це паралельні алгоритми, тому саме GPU використовують для машинного навчання.

3 РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Розробка веб-застосунку

Розробляти проект будемо у редакторі вихідного коду Visual Code. Для написання коду програми використаємо мову програмування Python, функціонал якої розширено за допомогою бібліотеки TensorFlow. Python надає можливість написання ефективного та оптимізованого коду, тому для задач розробки проекту було використано саме Python. TensorFlow у свою чергу надає набір робочих інструментів для розробки, навчання та використання моделей з використанням мови Python.

Для побудови веб-сайту, а саме його розмітки використаємо гіпертекстову мову розмітки HTML. Також, для деяких елементів інтерфейсу застосуємо можливості мови програмування JavaScript з доданими стилями каскадної таблиці стилів CSS.

Для того, щоб об'єднати код, написаний мовою програмування Python та веб-частину застосуємо фреймворк Flask, що являє собою легкий інтерфейс веб-серверного шлюзу. Фреймворк надає інструменти, бібліотеки та механіку, які дозволяють створювати веб-додаток, але без встановлювання жодних залежностей та жорстких обмежень до вигляду проекта, тому це значно спросить розробку даного проекта.

Для задачі з розпізнавання положення тіла людини у реальному часі використаємо модель BlazePose, яка може обчислювати координати 33 ключових точок скелета у 3D-просторі.

За попереднім тестуванням було виявлено, що при значеннях параметра «min_detection_confidence» та «min_tracking_confidence» рівному «0,55» модель достатньо точно та оптимізовано розпізнає присутність та позу тіла людини при роботі із зображенням з веб-камери. Тому у якості параметрів моделі BlazePose використаємо «min_detection_confidence=0.55», «min_tracking_confidence=0.55».

Для успішного розпізнавання пози людини потрібно правильно підібрати модель для використання у веб-застосунку. У випадку розробки веб-застосунку є

необхідність мати можливість запускати додаток на великій кількості пристроїв. Але у той же час модель має бути достатньо точною для правильного розпізнавання пози людини у кожен момент часу. Тому за заздалегідь проведеними тестами було виявлено, що найоптимальнішим варіантом для використання у розроблюваному веб-застосунку є модель BlazePose GHUM Full. Вона є достатньо легкою для безпроблемного запуску на більшості сучасних пристроїв та у той же час надає високий рівень точності, що задовольняє потреби для реалізації даного веб-додатку.

Для розробки та тестування веб-застосунку використано ноутбук Xiaomi Air 13,3 з характеристиками, наведеними у табл.3.1.

Таблиця 3.1 – Технічні характеристики Xiaomi Air 13,3

Центральний процесор	Intel Core i5-8250U 1,6Ghz-3,4Ghz
ОЗП	8Gb DDR4 2400Mhz
Графічний процесор	nVidia MX150 2Gb
ОС	Windows 10 Enterprise

3.2 Налаштування Flask

3.2.1 Інсталювання фреймворку

Для роботи з фреймворком Flask потрібно провести його інсталяцію. Для цього скористаємось вбудованим у Python менеджером пакетів pip. Запускаємо команду «pip install flask», результатом виконання буде встановлений фреймворк Flask (рис. 3.1).

```
(testenv) C:\Users\fdl\Desktop\Test>pip install flask
Collecting flask
  Using cached Flask-2.1.2-py3-none-any.whl (95 kB)
Collecting Werkzeug>=2.0
  Using cached Werkzeug-2.1.2-py3-none-any.whl (224 kB)
Collecting Jinja2>=3.0
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting click>=8.0
  Using cached click-8.1.3-py3-none-any.whl (96 kB)
Collecting itsdangerous>=2.0
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting colorama
  Using cached colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.1.1-cp310-cp310-win_amd64.whl (17 kB)
Installing collected packages: Werkzeug, MarkupSafe, itsdangerous, colorama, Jinja2, click, flask
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.1.2 click-8.1.3 colorama-0.4.4 flask-2.1.2 itsdangerous-
2.1.2
WARNING: You are using pip version 22.0.4; however, version 22.1 is available.
You should consider upgrading via the 'C:\Users\fdl\Desktop\Test\testenv\Scripts\python.exe -m pip install --upgrade pip
' command.

(testenv) C:\Users\fdl\Desktop\Test>
```

Рисунок 3.1 – Результат виконання команди «pip install flask»

3.2.2 Ініціалізація Flask

При використанні Flask ми будемо використовувати шаблон для роботи зі стандартизованою мовою розмітки HTML. Для ініціалізації роботи Flask створюємо файл, у якому буде викликатись Flask – «app.py».

У файлі імпортуємо усі потрібні для роботи бібліотеки (рис. 3.2).

```
1  import cv2
2  import mediapipe as mp
3  import numpy as np
4
5  import counter
6  import stage
```

Рисунок 3.2 – Імпортування необхідних для роботи бібліотек

Також імпортуємо необхідні для роботи Flask функції: Flask, render_template, Response. Функція Video з файла camera.py необхідна для постійного оновлення зображення з веб-камери, адже воно передається покадрово (рис. 3.3).

```
7
8   from flask import Flask, render_template, Response
9   from camera import Video
```

Рисунок 3.3 – Імпортування функцій Flask

Основа для використання Flask зображена на (рис. 3.4).

```
1   from flask import Flask, render_template
2
3   app = Flask(__name__)
4
5   @app.route('/')
6   def index():
7       return render_template('index.html')
8
```

Рисунок 3.4 – Шаблон для запуску веб-сервера Flask

В нашому випадку починаємо з опису «@app.route('/')», це корінь сайту, тобто основна та перша сторінка браузера, яка відкривається після старту сервера Flask. У ній вказуємо, що при переході на основну сторінку сайта серверу потрібно віддавати файл «index.html» (рис. 3.5).

```
12  app = Flask(__name__)
13
14
15  @app.route('/')
16  def index():
17      return render_template('index.html')
18
```

Рисунок 3.5 – Налаштування корінної адреси веб-сайта

Далі у «app.route» використовуємо конструкцію «while True» для постійного виклику функції get.frame з файлу camera.py для оновлення зображення з веб-камери. Оголошуємо змінну frame якій присвоїмо результат виконання функції get.frame. Після цього використовуємо метод Flask для роботи із зображеннями, у якій вказано використання змінної frame (рис. 3.6).

```
20  def gen(camera):
21      while True:
22          frame = camera.get_frame()
23          yield(b'--frame\r\n'
24                b'Content-Type:  image/jpeg\r\n\r\n' + frame +
25                b'\r\n\r\n')
26
```

Рисунок 3.6 – Використанння конструкції while True

Наступним кроком вказуємо шлях «/video» у якому буде лише оброблене зображення з веб-камери. Цей шлях потрібен для процесу відладки можливих проблем при розробці (рис. 3.7).

```
28  @app.route('/video')
29  def video():
30      return Response(gen(Video()),
31                      mimetype='multipart/x-mixed-replace; boundary=frame')
32
```

Рисунок 3.7 – Додавання шляху /video

Також додаємо рядок, який вмикає режим «debug» (рис. 3.8).

```
33
34      app.run(debug=True)
35
```

Рисунок 3.8 – Вмикання режиму debug

3.3 Налаштування web-інтерфейсу

3.3.1 Основна сторінка

Для коректної роботи програми потрібно налаштувати веб-інтерфейс та створити необхідні елементи для відображення даних. Створюємо папку templates у якій розміщуємо файл index.html, на який посилається фреймворк Flask при запуску веб-сервера. У документі використовуємо стандартну конструкцію html та додаємо параметри з мета-даними, такі як: мова за замовчуванням англійська, кодування UTF-8, та параметр viewport – видиму користувачу область веб-сторінки, яку він може побачити, не застосовуючи прокрутку. Також вказуємо параметр «title», який відповідає за відображення назву веб-сторінки у браузері (рис. 3.9).

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Необхідні meta теги -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x" crossorigin="anonymous">

    <title>CountFit</title>
  </head>
```

Рисунок 3.9 – Мета-дані файлу index.html

Також зверху сторінки додаємо навігаційний бар, який залишається зверху сторінки навіть при прокручуванні сторінки. Для цього оголошуємо відповідний клас та вказуємо необхідні параметри, такі як style="color: aliceblue;" та align="left". Тобто було обрано білий колір навігаційної панелі та вирівнювання тексту на панелі по лівій стороні (рис. 3.10). Наступним кроком додаємо елемент <img> з посиланням на емблему програми для відображення її на навігаційній панелі з параметрами ширини та висоти 40 пікселів. Також вписуємо назву «NeuraTrain», це буде назвою програми, яка буде відображатись на навігаційній панелі.

```
<nav class="navbar navbar-light bg-dark">
    <div class="container">
      <a class="navbar-brand" href="#" style="color: aliceblue;" align="left">
        <img src="https://icon-library.com/images/calories-icon/calories-icon-23.jpg" alt="" width="40"
        height="40">
        NeuraTrain
    </a>
    </div>
  </nav>
```

Рисунок 3.10 – Створення класу для елемента navbar

Створюємо header-елемент html-сторінки типу h1, у якому вказуємо назву проекта «Віртуальний тренер» та вирівнюємо текст по лівій стороні.

За допомогою класу div додаємо елемент img зі змінною «url_for('video')» , яка автоматично буде розпізнана web-сервером Flask як посилання на оброблене зображення з веб-камери. Додаємо опції «justify-content: left; display: flex; margin-top: 30px; margin-left: 30px» для вирівнювання елементу по лівій стороні та відступам від інших елементів зверху та зліва по 30 пікселів (рис. 3.11).

```
<h1 style="text-align: left;">Віртуальний тренер</h1>
<div style="justify-content: left; display: flex; margin-top: 30px; margin-left: 30px;"><img src="{{ url_for
('video') }}"></div>
```

Рисунок 3.11 – Створення header-елементу

3.4 Написання Python скрипту

3.4.1 Імпортування бібліотек

Для роботи програми потрібно імпортувати необхідні для роботи бібліотеки. Бібліотека cv2 (OpenCV) використовується для доступу до веб-камери пристрою, а також для розпізнавання вхідного зображення. Бібліотека mediapipe потрібна для оптимізації використання ресурсів пристрою для розпізнавання образів. NumPy потрібен для швидкого обчислювання формул прорахунку кута положення частин тіла. Також імпортуємо потрібні функції з інших файлів програми (calcAngle, counter, stage). Імпортуємо потрібні для роботи програми бібліотеки (рис. 3.12):

```
1   import cv2
2   import mediapipe as mp
3   import numpy as np
4   from calcAngle import calculate_angle
5   import counter
6   import stage
```

Рисунок 3.12 – Імпортування необхідних бібліотек у файл camera.py

Проводимо ініціацію імпортованих функцій з файлів counter та stage (рис. 3.13):

```
7   counter.init()
8   stage.init()
```

Рисунок 3.13 – Ініціалізація функцій counter та stage

Для відображення на екрані положеня частин тіла скористаємось функціями drawing_utils та pose з бібліотеки mediapipe (рис. 3.14):

```
10  mediapipe_draw = mp.solutions.drawing_utils
11  mediapipe_pose = mp.solutions.pose
```

Рисунок 3.14 – Додавання відображення на екрані положення частин тіла

3.4.2 Оголошення основного класу

Оголошуємо клас Video, це основний каркас, який буде викликатись фреймворком Flask для відображення потокового відео з веб-камери. Також оголошуємо функції init та del, які відповідають за початок та кінець обробки відео відповідно. Функція get_frame викликається фреймворком Flask та використовується для постійного оновлення зображення з веб-камери (рис. 3.15).

```
14  class Video(object):
15      def __init__(self):
16          self.video = cv2.VideoCapture(0)
17      def __del__(self):
18          self.video.release()
19      def get_frame(self):
```

Рисунок 3.15 – Оголошення класу Video

Використовуємо mediapipe з налаштуваннями min_detection_confidence=0.55 та min_tracking_confidence=0.55. Ці параметри відповідають за коефіцієнти трекінгу положення тіла, тому беремо близькі до середніх значення задля того, щоб отримати баланс між правильним розпізнаванням та відсутністю помилок. Змінна frame потрібна для збереження щосекундних змін значень масиву розпізнаних точок тіла (рис. 3.16).

```
27          # Налаштування mediapipe
28          with mediapipe_pose.Pose(min_detection_confidence=0.55, min_tracking_confidence=0.55) as pose:
29              ret, frame = self.video.read()
30
```

Рисунок 3.16 – Налаштування коефіцієнтів трекінгу положення тіла

Змінюємо кольорову схему вхідного зображення на RGB задля відпрацювання функції process. Використовуємо функцію process для процесу детекції зображення та конвертуємо кольорову схему назад до BGR (рис. 3.17).

```
31              # Зміна кольорово схеми на RGB
32              image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
33              image.flags.writeable = False
34
35              # Процес розпізнавання
36              results = pose.process(image)
37
38              # Зміна кольорової схеми назад до BGR
39              image.flags.writeable = True
40              image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

Рисунок 3.17 – Зміна кольорової схеми вхідного зображення

3.4.3 Алгоритм розпізнавання

Використовуємо метод try для запобігання перебоїв у роботі алгоритму, адже розпізнавання може відбуватися не кожен кадр та записуємо масив положень розпізнаних точок тіла у змінну landmarks (рис. 3.18)

```
42          # Отримання точок положення частин тіла
43          try:
44              landmarks = results.pose_landmarks.landmark
45
```

Рисунок 3.18 – Використання методу try для виклику функції обробки вхідного зображення

Для розпізнавання положення потрібних контрольних точок тіла потрібно виокремити необхідні координати відповідних частин тіла за схемою із загального масиву розпізнаних точок (рис. 3.19).

```
46          # Отримання координатів для правої половини тіла
47
48          hip_right = [landmarks[mediapipe_pose.PoseLandmark.RIGHT_HIP.value].x,
49                       landmarks[mediapipe_pose.PoseLandmark.RIGHT_HIP.value].y]
50          shoulder_right = [landmarks[mediapipe_pose.PoseLandmark.RIGHT_SHOULDER.value].x,
51                            landmarks[mediapipe_pose.PoseLandmark.RIGHT_SHOULDER.value].y]
52          elbow_right = [landmarks[mediapipe_pose.PoseLandmark.RIGHT_ELBOW.value].x,
53                         landmarks[mediapipe_pose.PoseLandmark.RIGHT_ELBOW.value].y]
54
55          # Отримання координатів для лівої половини тіла
56
57          hip_left = [landmarks[mediapipe_pose.PoseLandmark.LEFT_HIP.value].x,
58                      landmarks[mediapipe_pose.PoseLandmark.LEFT_HIP.value].y]
59          shoulder_left = [landmarks[mediapipe_pose.PoseLandmark.LEFT_SHOULDER.value].x,
60                           landmarks[mediapipe_pose.PoseLandmark.LEFT_SHOULDER.value].y]
61          elbow_left = [landmarks[mediapipe_pose.PoseLandmark.LEFT_ELBOW.value].x,
62                        landmarks[mediapipe_pose.PoseLandmark.LEFT_ELBOW.value].y]
63
```

Рисунок 3.19 – Розпізнавання положення контрольних точок тіла

Запускаємо веб-сервер Flask та у вікні термінала отримуємо посилання на сторінку нашого сайту (рис. 3.20).

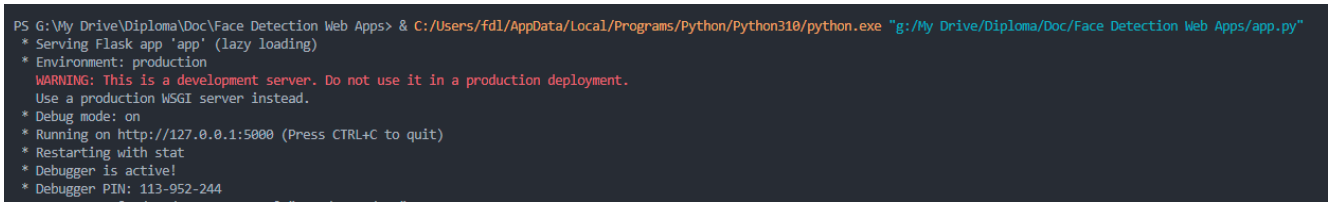


Рисунок 3.20 – Запуск веб-сервера Flask у вікні термінала

Перевіряємо коректність відображення відеосигналу. Як видно з рис. 3.21 , функція відображення потокового сигналу з веб-камери працює коректно.

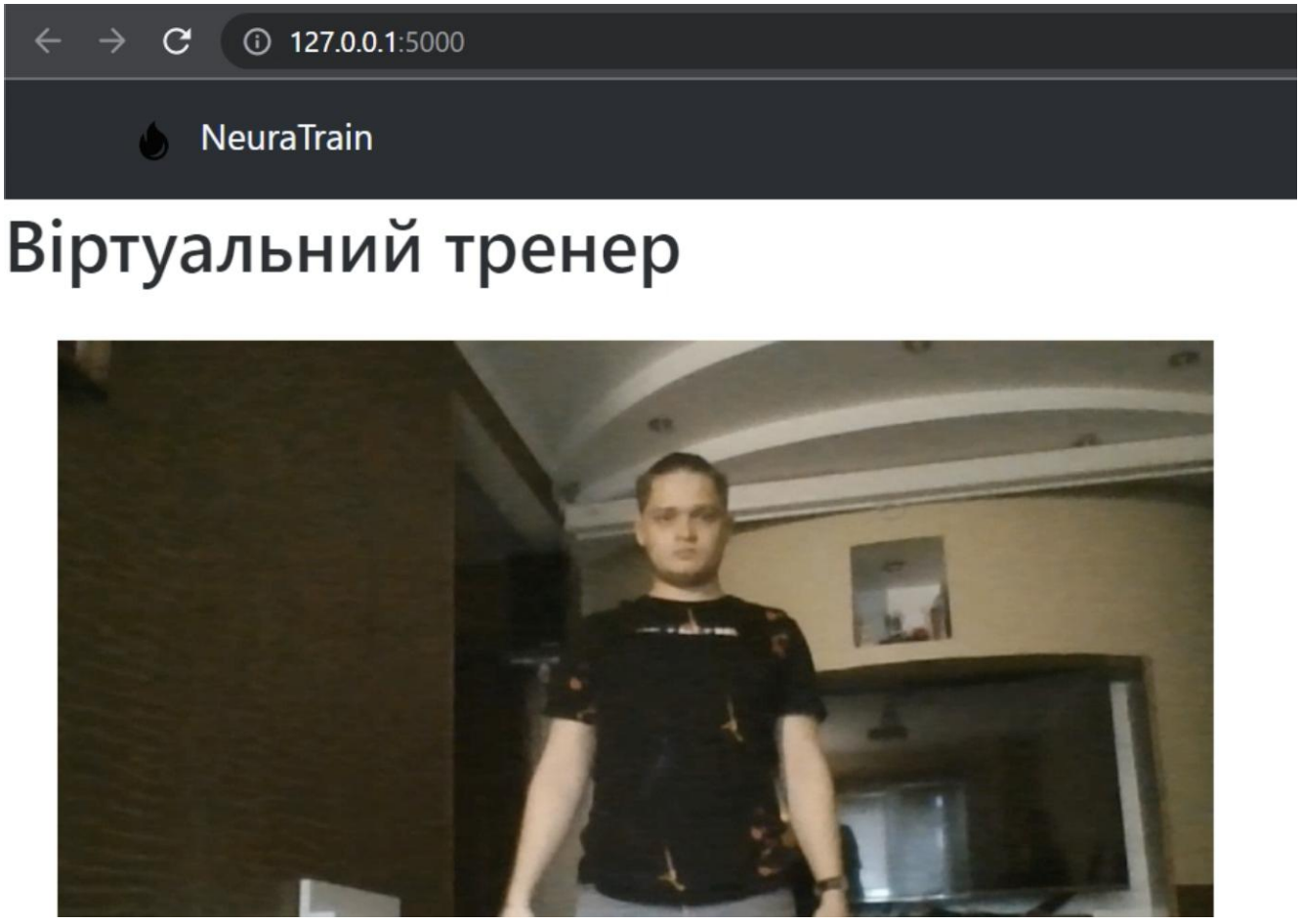


Рисунок 3.21 – Запуск веб-сервера Flas

Для реалізації системи підрахунку виконаних разів та контрою правильності виконання вправи скористаємось формулою знаходження кута між трьома точками: np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1], a[0]-b[0]). Для цього створюємо окремий файл під назвою calcAngle.py. У ньому імпортуємо бібліотеку для роботи з математичними формулами NumPy (рис. 3.22):

```
1    import numpy as np
2
```

Рисунок 3.22 – Імпортування бібліотеки numpy

Далі оголошуємо функцію calculate_angle для підрахунку кута між трьома точками (рис. 3.23).

```
3
4    def calculate_angle(first, mid, end):
```

Рисунок 3.23 – Оголошення функції calculate_angle

Оголошуємо змінні first, mid та end, у які будуть записуватись масиви з координатами точок відповідних частин тіла (рис. 3.24)

```
5        first = np.array(first)  # Перша точка
6        mid = np.array(mid)  # Друга точка
7        end = np.array(end)  # Третя точка
```

Рисунок 3.24 – Оголошення змінних first, mid та end

Змінна radians, використовуючи функцію np.arctan виконує обчислення кута у радіанах (рис. 3.25)

```
8
9        radians = np.arctan2(end[1]-mid[1], end[0]-mid[0]) - np.arctan2(first[1]-mid[1], first[0]-mid[0])
```

Рисунок 3.25 – Обчислення кута у радіанах

Змінна angle конвертує значення у радіанах в звичне значення у градусах (рис. 3.26)

```
10       angle = np.abs(radians*180.0/np.pi)
11
```

Рисунок 3.26 – Конвертування значення у радіанах у значення в градусах

Також потрібно додати умову для позбавлення від від'ємних значень координат (рис. 3.27)

```
12      if angle > 180.0:
13          angle = 360-angle
14
```

Рисунок 3.27 – Умова позбавлення від від'ємних значень кута

Для використання функції calculate_angle у початку основного файлу camera.py було додану рядок з імпортом цієї функції (рис. 3.28)

```
4   from calcAngle import calculate_angle
```

Рисунок 3.28 – Імпортування функції calculate_angle з файлу calcAngle

Після отримання масивів даних по положенню потрібних точок тіла скористаємось функцією calculateAngle для підрахунку значень кутів у кожний момент часу коли викликається клас Video для лівої та правої руки (рис. 3.29).

```
64
65              # Прорахунок кутів
66              angle_left = calculate_angle(hip_left, shoulder_left, elbow_left)
67              angle_right = calculate_angle(hip_right, shoulder_right, elbow_right)
68
```

Рисунок 3.29 – Підрахунок кутів у кожен момент часу

Для відладки використаємо функцію cv2.putText. У аргументи функції передається змінна image з зображенням отриманим з веб-камери, а також значення кута, знайденого за допомогою функції calculate_angle, переведеного у формат string. Задаємо відображення кутів для обох рук (рис. 3.30).

```
69                  # Візуалізація значення кута
70                  cv2.putText(image, str(angle_left),
71                              tuple(np.multiply(
72                              shoulder_left, [640, 480]).astype(int)),
73                              cv2.FONT_HERSHEY_SIMPLEX, 0.5, (
74                              255, 255, 255), 2, cv2.LINE_AA
75                              )
76                  cv2.putText(image, str(angle_right),
77                              tuple(np.multiply(
78                              shoulder_right, [640, 480]).astype(int)),
79                              cv2.FONT_HERSHEY_SIMPLEX, 0.5, (
80                              255, 255, 255), 2, cv2.LINE_AA
81                              )
```

Рисунок 3.30 – Візуалізація значення кута

Перевіряємо коректність відображення кутів між трьома заданими точками (рис. 3.31).

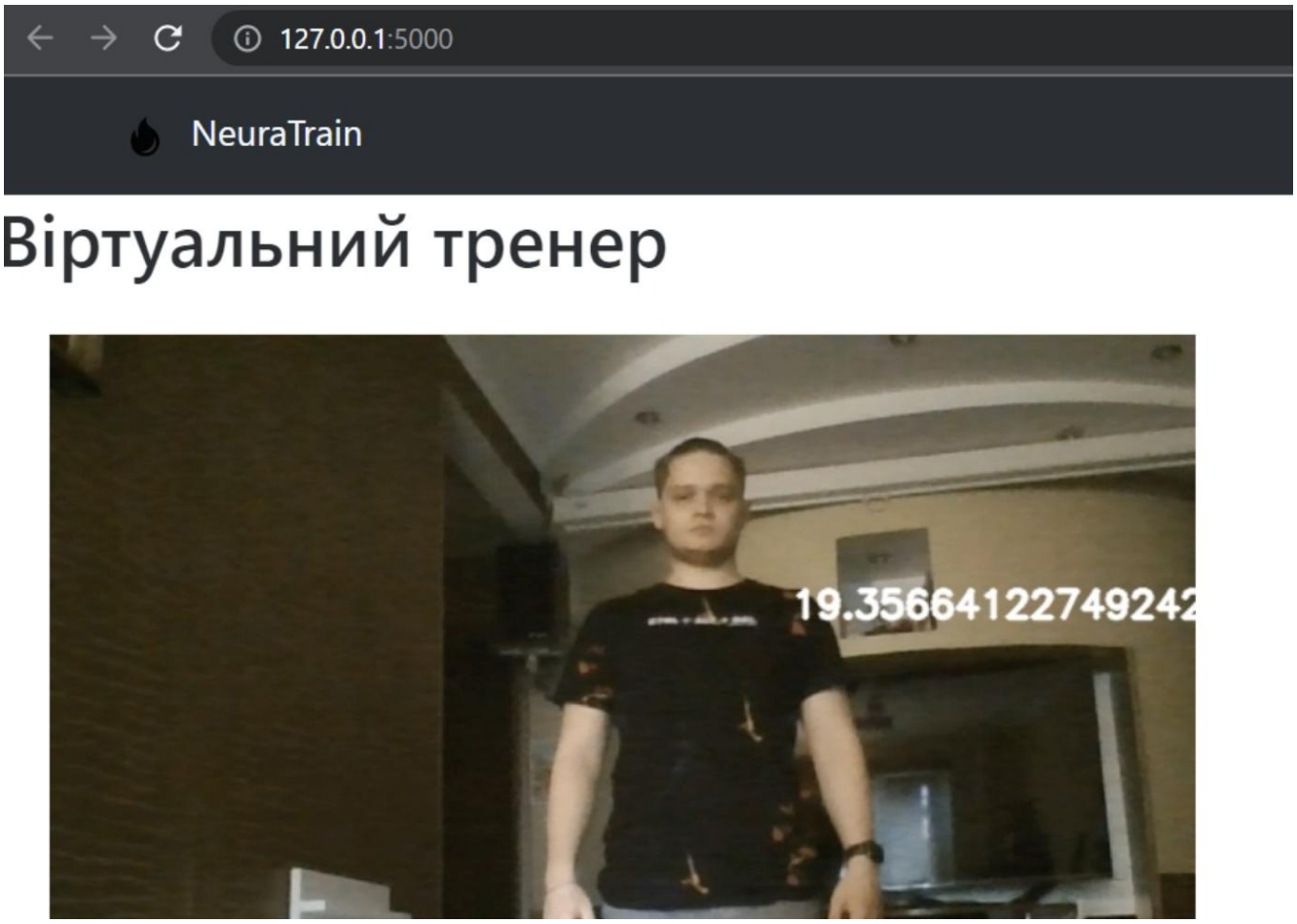


Рисунок 3.31 – Перевірка відображення значення кута

Прописуємо логіку підрахунку кількості виконаних разів. Для тесту було обрано махи обох рук тіла у різні боки (можлива варіація з гантелями). Алгоритмом за замовчуванням вважається, що руки опущені донизу, тому кут між руками та тулубом менше 30-ти градусів. Якщо умова виконана, то змінній stage у файлі stage.py задається значення “DOWN”. Якщо обидві руки підняті догори (тобто обидві руки фіксовані у верхньому положенні і кут між руками та тулубом більше за 90 градусів), то алгоритм зараховує це як одне повторення вправи. У такому випадку змінній stage у файлі stage.py надається значення «UP», а у змінній counter у файлі counter.py з початковим значенням 0 значення збільшується на 1. Оновлене значення виводиться на екран (рис. 3.32).

```
83              # Логіка підрахунку виконаних разів
84              if angle_left < 30 and angle_right < 30:
85                  stage.stage = "DOWN"
86              if angle_left > 90 and angle_right > 90 and stage.stage == 'DOWN':
87                  stage.stage = "UP"
88                  counter.counter += 1
89                  print(counter.counter)
```

Рисунок 3.32 – Алгоритм підрахунку виконаних разів

У випадку, якщо метод try не зміг провести підрахунок кута та змінити значення змінних підрахунку по причині відсутності даних – алгоритм пропускає цей кадр (рис. 3.33)

```
91              except:
92                  pass
```

Рисунок 3.33 – Конструкція except

3.4.4 Відображення статистики

Для індикації разів виконання вправи та нинішнього положення рук скористаємось вбудованими у бібліотеку OpenCV функціями rectangle та putText.

У функцію rectangle передаємо змінну image, у якій зберігається зображення з веб-камери, а також вказуємо початкову та кінцеву точку прямокутника, що

буде зображений поверх зображення з веб камери. Так як роздільна здатність вікна 640*480 пікселів, то вказуємо початкові координати (0,0), а кінцевими вказуємо (640,95). Колір вказуємо білий, у кольоровій схемі RGB це значення пікселів (255,255,255). Також вказуємо значення -1 для того, щоб прямокутник був залитий цим кольором (рис. 3.34).

```
94          # Відображнення виконаних разів
95          # Налаштування поля відображення
96          cv2.rectangle(image, (0, 0), (640, 95), (255, 255, 255), -1)
```

Рисунок 3.34 – Налаштування поля відображення результатів

Перезапускаємо сервер Flask та перевіряємо правильність заданих координат функції rectangle (рис. 3.35)

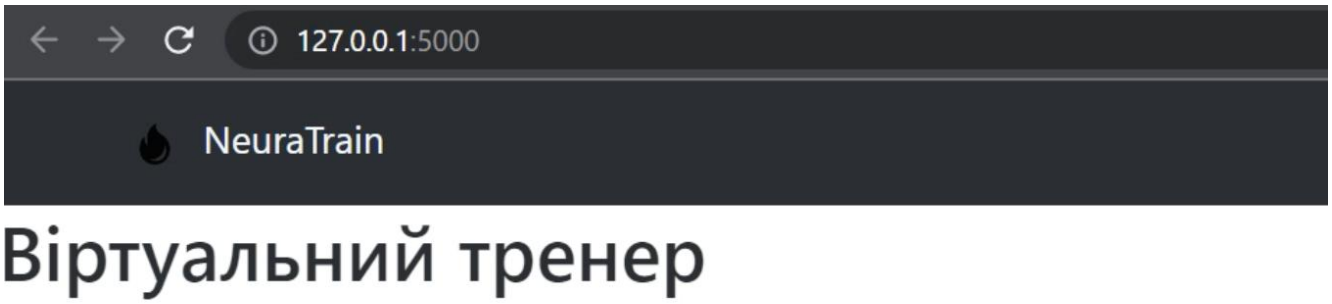


Рисунок 3.35 – Перевірка наявності поля відображення результатів

Для відображення кількості виконаних разів скористаємось функцією putText. У функцію передається змінна image, результатом є накладена поверх зображення графіка з потрібними значеннями.

Викликаючи функцію аргументами image, «'РАЗ(ІВ)', (25, 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA» задаємо параметри відображення. Тобто відобразиться поле з текстом «РАЗ(ІВ) з початковими координатами (25,15) пікселів зі шрифтом Hershey Simplex шириною 0.5 пікселя чорного кольору, непрозорий (рис. 3.36)

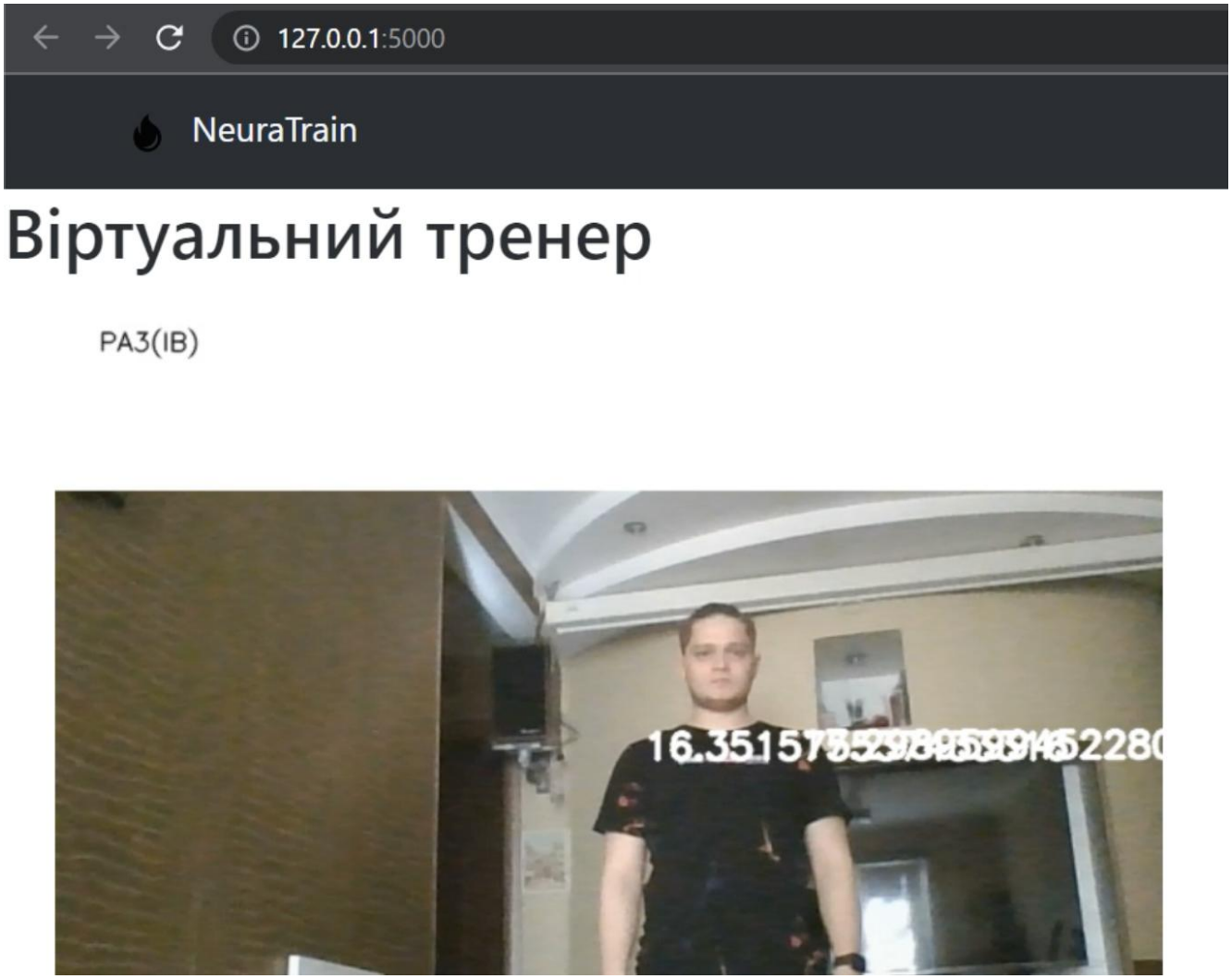


Рисунок 3.36 – Відображення підпису кількості виконаних разів

При повторному виклику функції відображаємо строкове значення змінної counter у полі з початковими координатами (25,15) пікселів зі шрифтом Hershey Simplex шириною 0.5 пікселя чорного кольору, непрозорий (рис. 3.37)

```
100             # Відображення кількості виконаних разів
101             cv2.putText(image, 'PA3(IB)', (25, 15),
102                         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
103             cv2.putText(image, str(counter.counter),
104                         (25, 75),
105                         cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 0), 2, cv2.LINE_AA)
```

Рисунок 3.37 – Додавання строкове значення змінної counter

Перевіряємо коректність відображення поля підрахунку кількості разів та впевнюємось у правильній роботі (рис. 3.38)

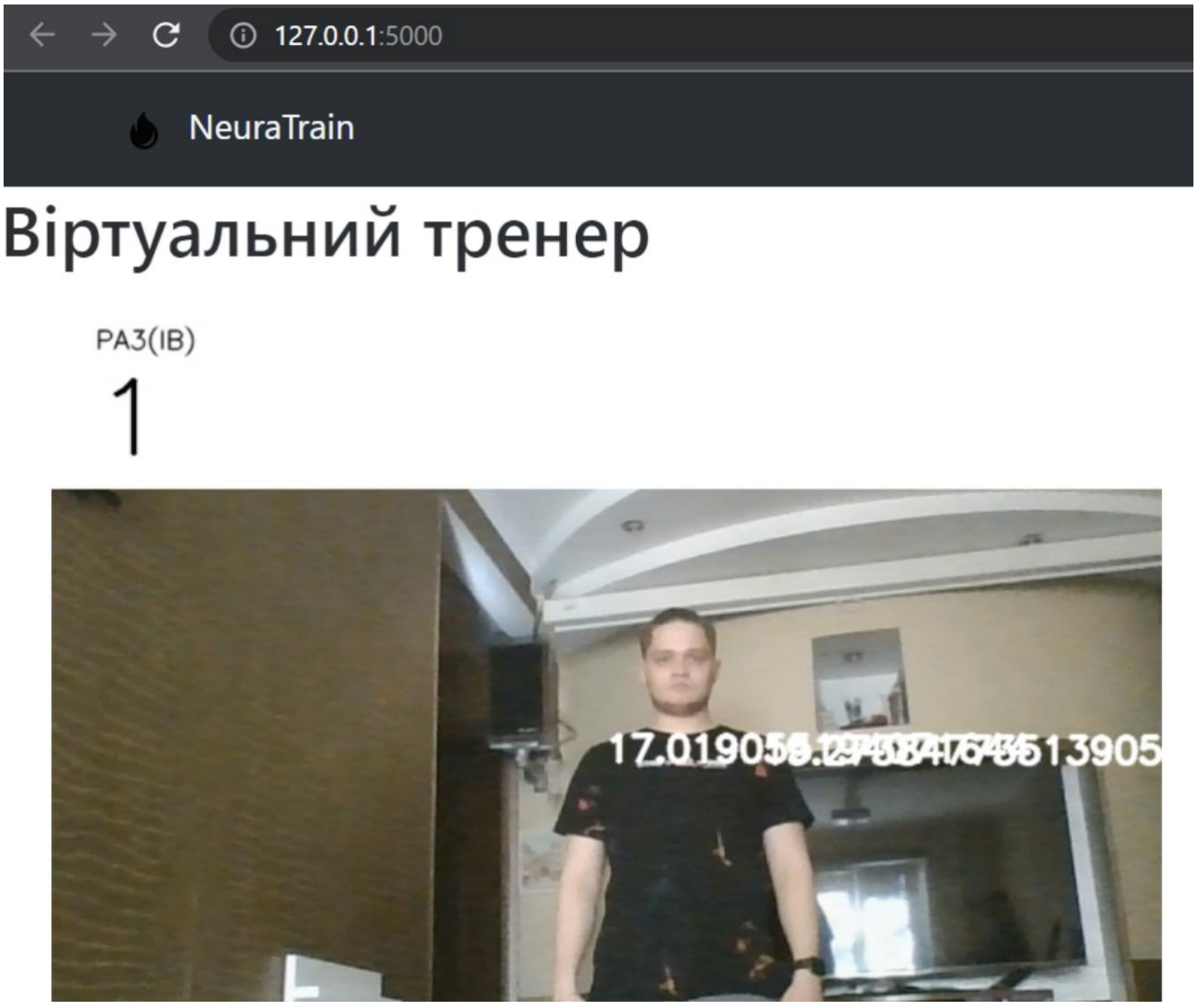


Рисунок 3.38 – Перевірка відображення підрахунку кількості виконаних разів

Далі потрібно додати поля для відображення статусу положення рук. Для цього аналогічно до попередніх дій скористаємось функцією putText.

Задаємо параметри для рядку «СТАН»: початкові координати (515,15) зі шрифтом Hershey Simplex шириною 0.5 пікселя чорного кольору, непрозорий рис. 3.39).

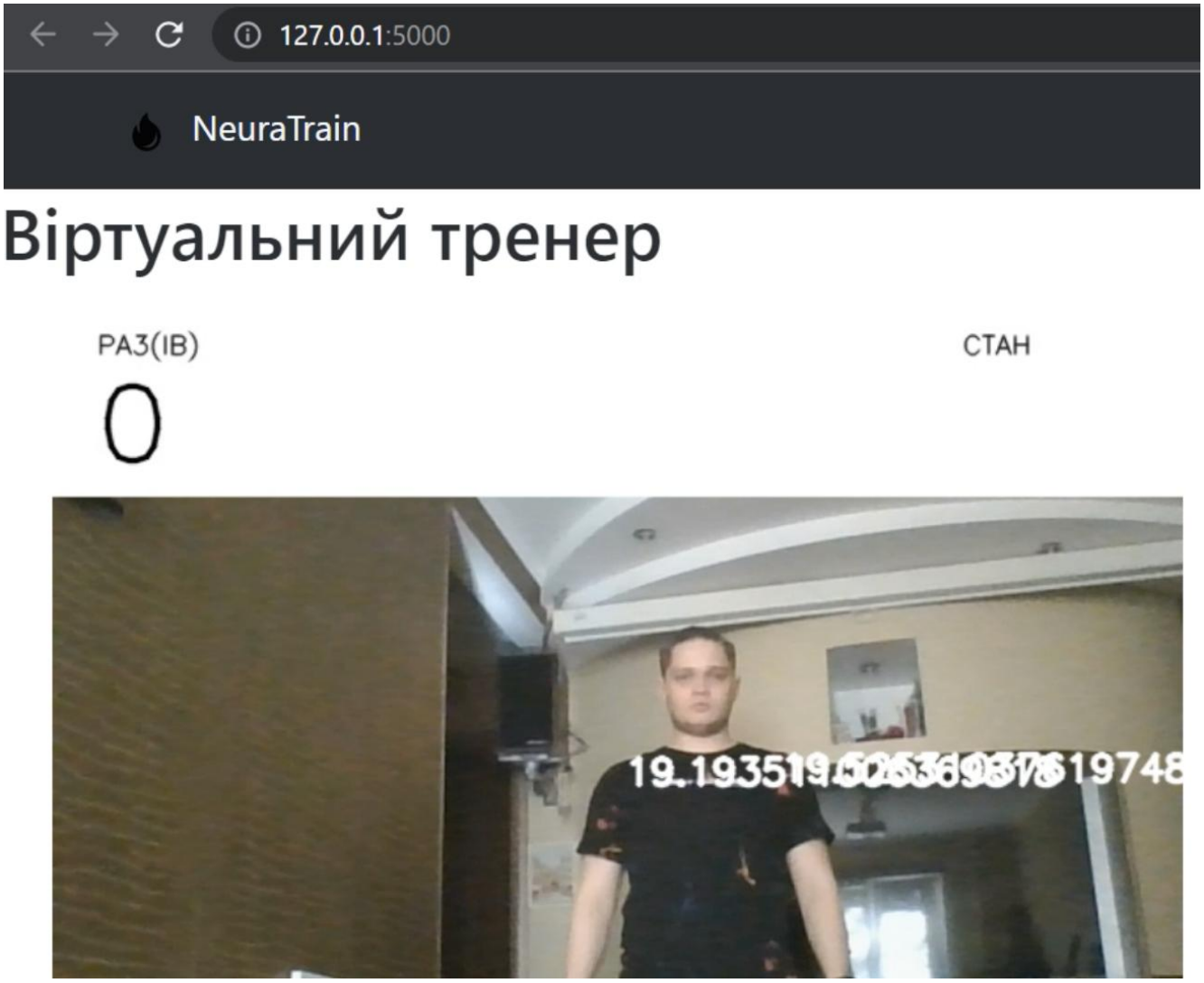


Рисунок 3.39 – Додавання підпису до стану положення рук

Також задаємо параметри для відображення змінної stage: початкові координати (450,75) зі шрифтом Hershey Simplex шириною 0.5 пікселя чорного кольору, непрозорий (рис. 3.40).

```
107             # Відображення статусу положення рук
108             cv2.putText(image, 'СТАН', (515, 15),
109                         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
110             cv2.putText(image, stage.stage,
111                         (450, 75),
112                         cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 0), 2, cv2.LINE_AA)
```

Рисунок 3.40 – Задавання параметрів відображення змінної stage

Перезапускаємо сервер Flask та перевіряємо коректність відображення стану положення рук (рис. 3.41)

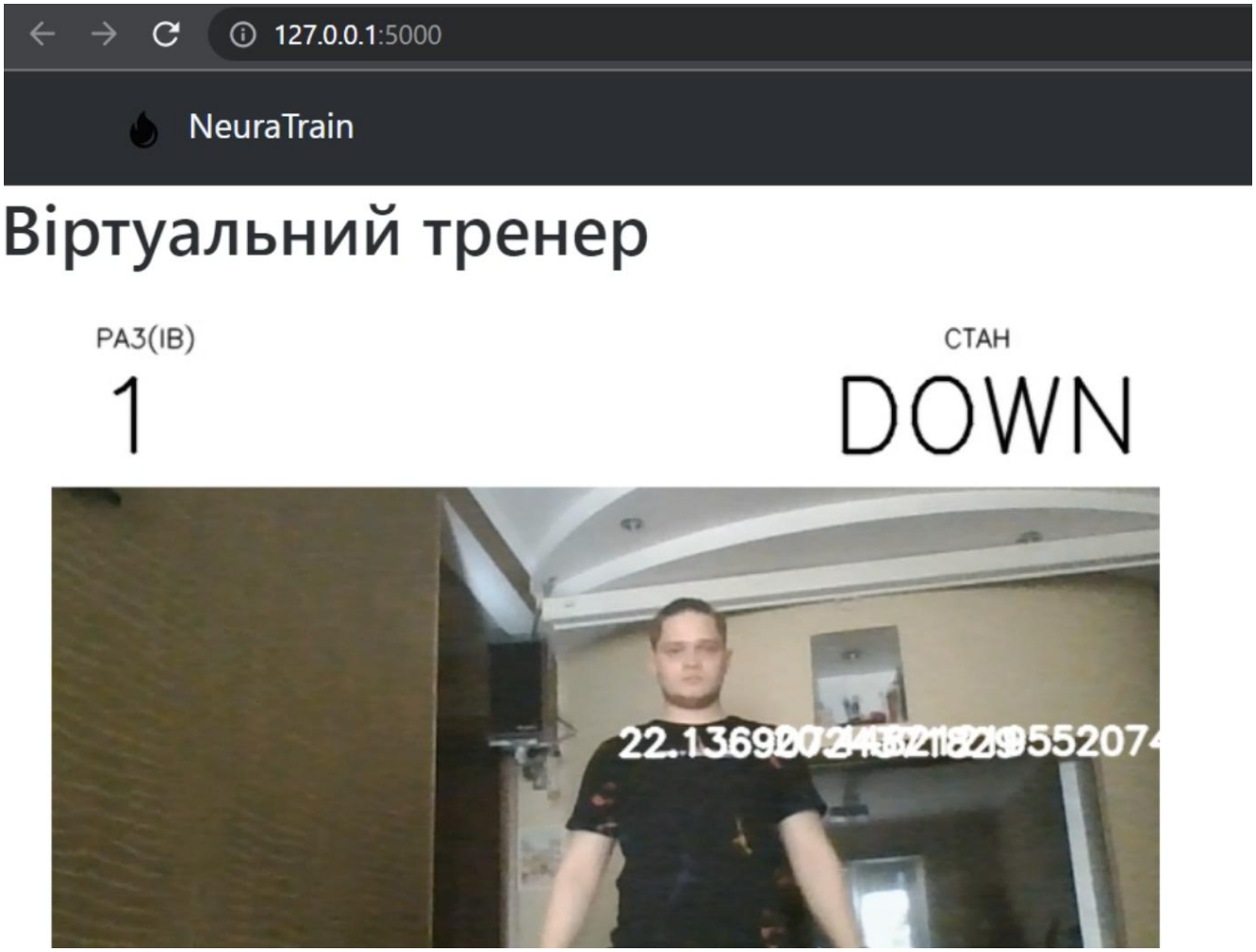


Рисунок 3.41 – Перевірка відображення стану положення рукі

3.4.5 Відображення графіки

Додаємо відображення розпізнаних точок тіла за допомогою функції draw_landmarks у mideapipe. У функцію передаємо змінну image та додаємо

параметри: колір за схемою RGB зі значенням (245, 117, 66) з товщиною 3 (рис. 3.42).

```
114             # Відображення розпізнаних точок тіла
115             mediapipe_draw.draw_landmarks(image, results.pose_landmarks, mediapipe_pose.POSE_CONNECTIONS,
116                                           mediapipe_draw.DrawingSpec(
117                                               color=(245, 117, 66), thickness=3, circle_radius=3),
118                                           mediapipe_draw.DrawingSpec(
119                                               color=(245, 66, 230), thickness=3, circle_radius=3)
120                                           )
```

Рисунок 3.42 – Додавання відображення розпізнаного скелету людини

Перевіряємо відображення розпізнаного скелету людини та як бачимо з рис. 3.43, програма вірно розпізнає положення тіла людини. Дана функція дозволяє проводити процес відладки для процесу виправлення помилок розпізнавання.

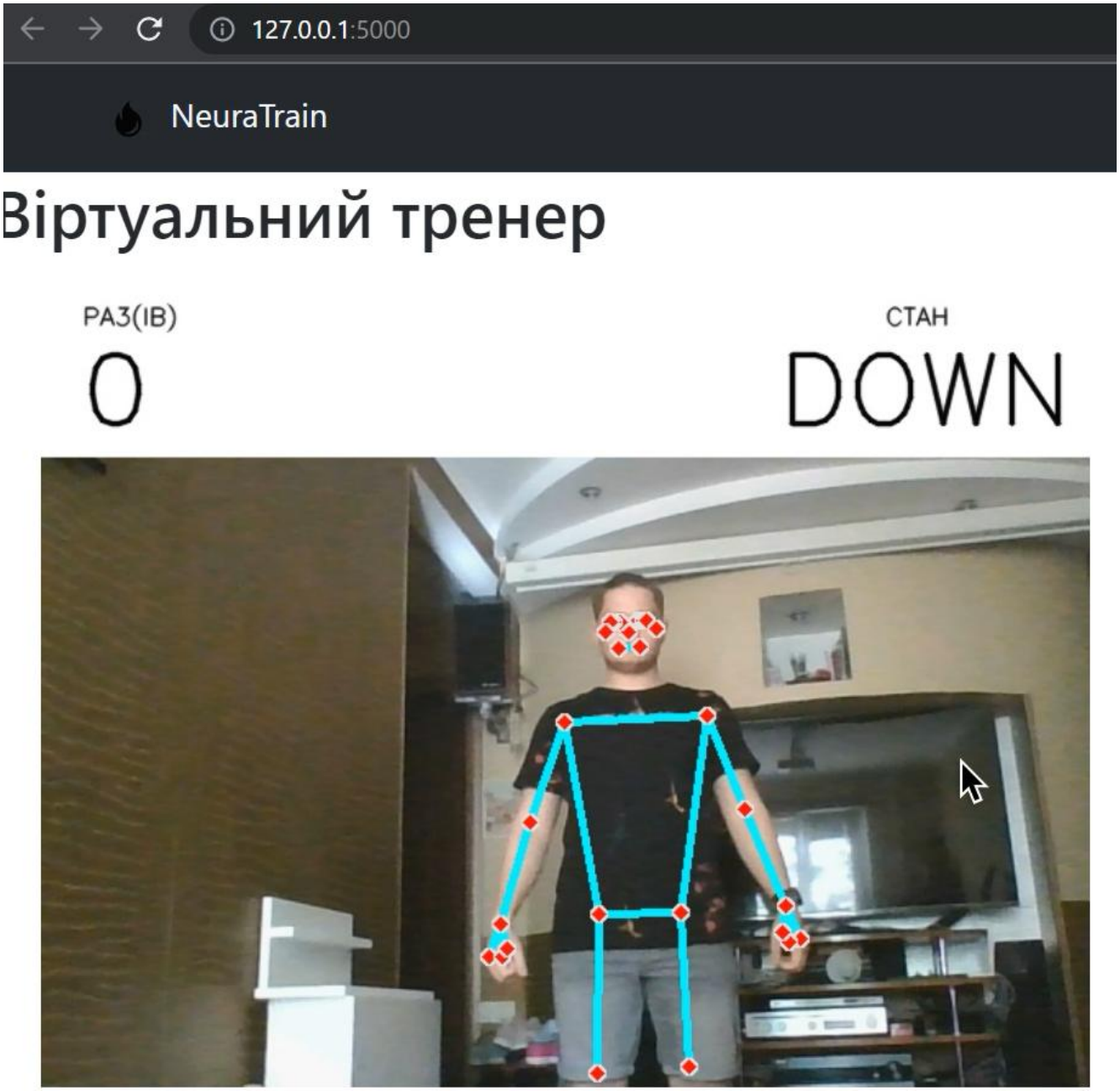


Рисунок 3.43 – Відображення розпізнаного положення тіла

Далі конвертуємо зображення з веб-камери у формат jpg за допомогою функції imencode та записуємо у змінну jpg. Клас Video повинен повертати зображення у форматі jpg, закодоване у побітовому вигляді (рис. 3.43).

```
126             ret, jpg = cv2.imencode('.jpg', image)
127             return jpg.tobytes()
```

Рисунок 3.44 – Процес конвертації зображення у формат jpg, закодований у побітовому вигляді

3.5 Огляд результату роботи

Результатом проектування є веб-сторінка з додатком, який дозволяє підрахувати кількість та правильність виконання одного виду фізичного навантаження, а саме розведення рук у сторони з, або без додаткової ваги.

Як видно з рис. 3.44, реалізовано функціонал підрахунку кількості виконаних разів, а також статус положення рук у даний час.

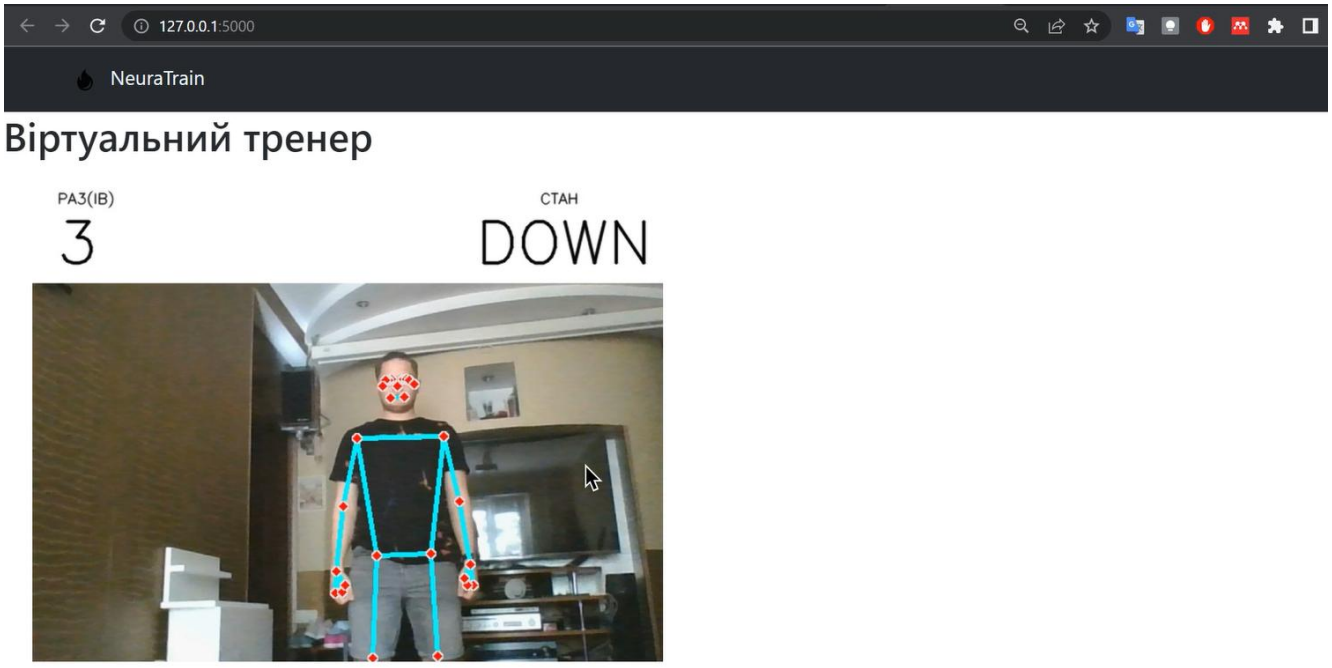


Рисунок 3.45 – Скріншот веб-сторінки проекту у веб-браузері

Підрахунок виконаних разів відбувається за таким алгоритмом: за замовчуваням при запуску програми значення кількості виконаних разів дорівнює 0, а стан статусу положення має значення «DOWN». Кожен кадр, отриманий з веб-камери проходить через процес розпізнавання зображення за допомогою моделі BlazePose з використанням MediaPipe. Кількість кадрів у секунду (кількість оновлень зображення) залежить від потужності пристроя, на якому запущений веб-сайт. Після оновлення інформації отримуються моментальні значення кута між руками та тулубом. Частини тіла для яких проводиться підрахунок кута залежать від обраної вправи, у даному випадку вправою є

розведення рук у сторони. Ці моментальні значення кута підраховуються для правої та лівої сторони тіла. Якщо значення кута менше 30-ти градусів – статус положення рук при виконанні вправи має значення «DOWN».

Правильним виконанням вправи є розведення та підняття обох рук вище рівня плечей, тому алгоритм програми зараховує виконання одного повторення тільки у випадку, коли кут між тулубом та руками більше 90 градусів (з урахуванням похибки розпізнавання), а також коли ця умова вірня для обох рук одночасно. При піднятті тільки однієї руки алгоритм програми не дозволить змінити статус стану положення у «UP» та зарахувати одне повторення. Проводимо перевірку твердження. Піднімаємо одну руку та спостерігаємо за статусом інтерфейсу (рис. 3.45).

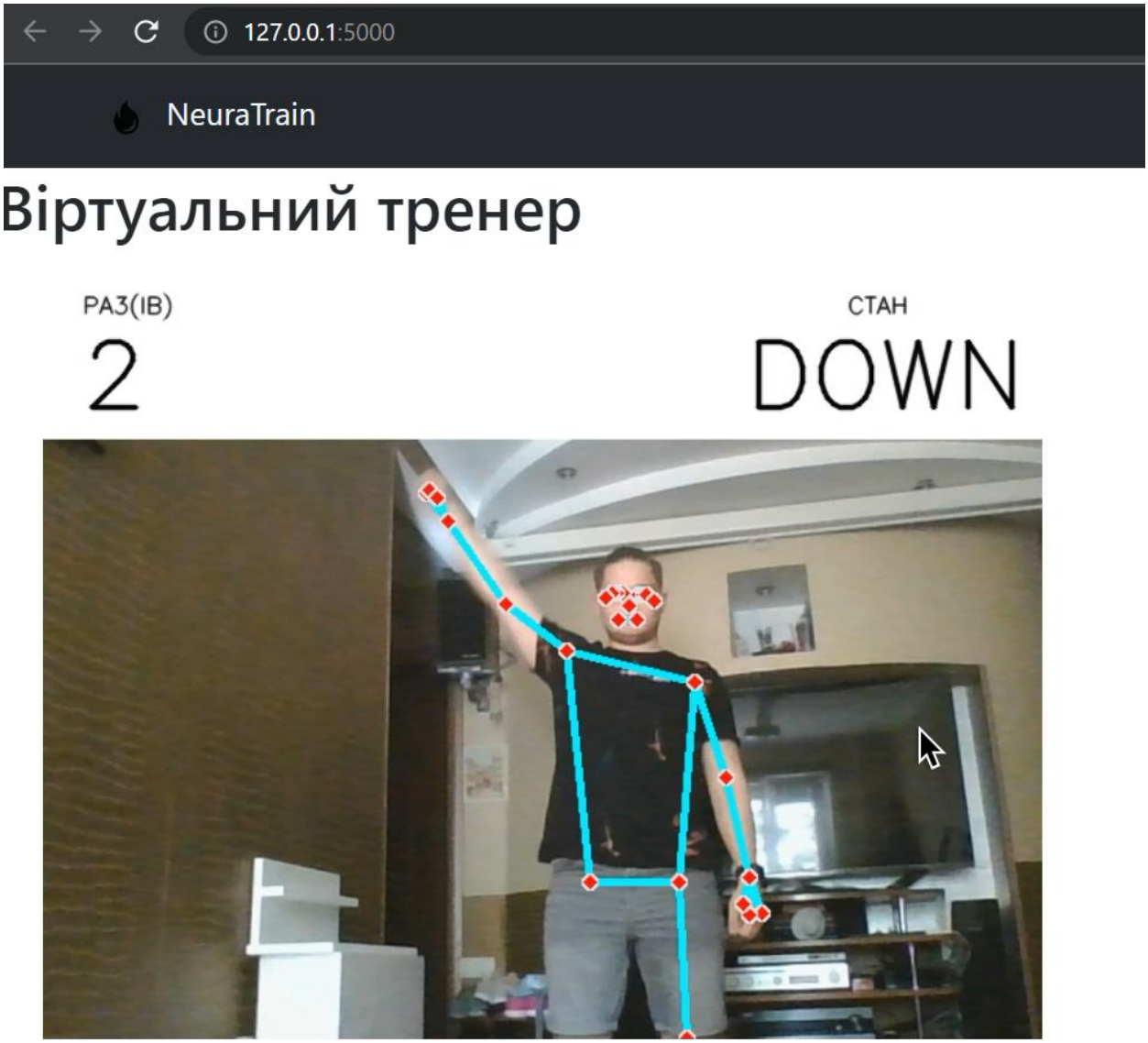


Рисунок 3.46 – Підняття однієї руки

Як бачимо, програма не зарахувала повторення, а також не змінила стан положення на значення «UP». Тепер спробуємо підняти обидві руки як того потребує техніка виконання. Як видно з рис. 3.46, програма зарахувала виконання вправи, адже були задовільнені обидві умови виконання, тобто присутня перевірка правильності виконання вправи.

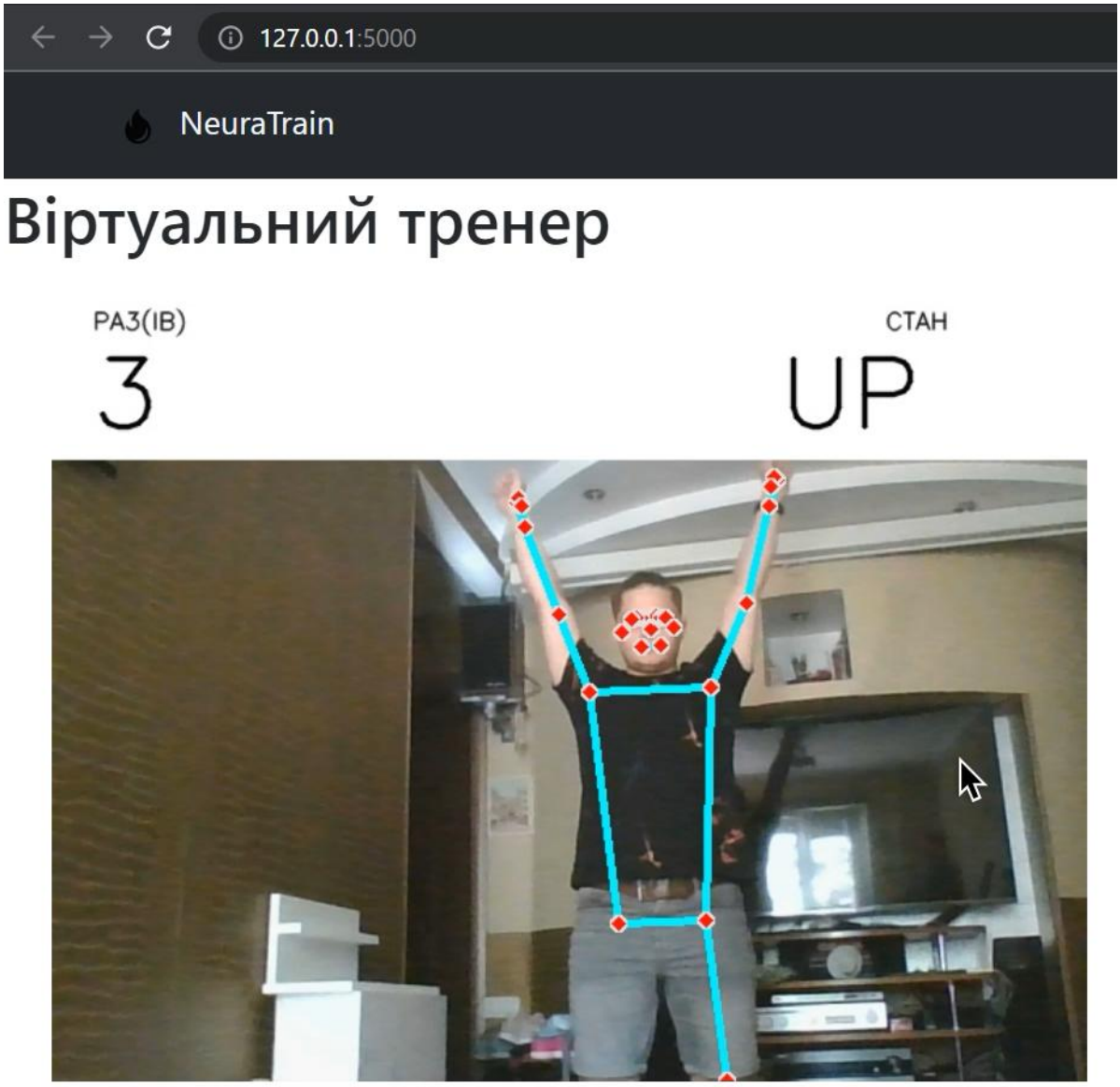


Рисунок 3.47 – Підняття обох рук

Висновки до розділу:

У третьому розділі розроблено веб-застосунок для оцінки якості та кількості виконання фізичного навантаження на основі технологій використання штучних нейронних мереж.

Код програми розроблено у редакторі вихідного коду Visual Code. Для розробки застосунку було використано мову програмування Python, а також бібліотеку TensorFlow. Бібліотека TensorFlow у великий мірі розширила функціонал мови програмування Python в задачах застосування моделей згорткових нейронних мереж та прискорила розробку та реалізацію проекта.

Для реалізації серверної частини веб-застосунка було використано можливості фреймворка Flask. Flask є інтерфейсом веб-серверного шлюзу, тобто фреймворком веб-додатків, який був створений, щоб полегшити роботу з веб-розробкою.

Інтерфейс користувача веб-застосунка реалізовано функціоналом мови сценаріїву JavaScript із застосуванням HTML та CSS технологій. Для використання Flask було створено веб-програму та підключено її до HTML. Для реалізації не знадобилось застосовувати додаткових фреймворків для мови програмування JavaScript.

У якості моделі розпізнавання положення тіла людини було використано модель BlazePose. BlazePose являє собою модель виявлення пози, що була розроблена Google . Дана модель дозволила обчислити координати 33 ключових точок скелета у 3D-просторі.

Бібліотека TensorFlow дозволяє використовувати потужність вбудованого у пристрій GPU для обчислень, необхідних для детекції пози людини. У випадку використання GPU параметр кількості кадрів на секунду значно зростає та дозволяє без видимих затримок відображати відео з вбудованої камери.

За попереднім тестуванням було виявлено, що при значеннях параметра «min_detection_confidence» та «min_tracking_confidence» рівному «0,55» модель достатньо точно та оптимізовано розпізнає присутність та позу тіла людини при

роботі із зображенням з веб-камери. Тому у якості параметрів моделі BlazePose використано «min_detection_confidence=0.55», «min_tracking_confidence=0.55».

У випадку розробки веб-застосунку є необхідність мати можливість запускати додаток на великій кількості пристроїв, тому за заздалегідь проведеними тестами було виявлено, що найоптимальнішим варіантом для використання у розроблюваному веб-застосунку є модель BlazePose GHUM Full. Вона є достатньо легкою для безпроблемного запуску на більшості сучасних пристроїв та у той же час надає високий рівень точності, що задовольняє потреби для реалізації даного веб-додатку.

Інтерфейс користувача веб-застосунку має в собі відображення кількості виконаних повторень, а також статусу положення рук при виконанні вправи. Додано режим відладки з відображенням розпізнаних частин тіла людини для кращої наочності, а також для процесу відстеження помилок при розпізнаванні.

ВИСНОВКИ

У роботі проведений аналіз історичних засад розробки систем нейронних мереж та їх підвиду – нейронних мереж із розпізнавання об’єктів. Нинішня хвиля популярності нейронних мереж, а саме їх підвиду – нейромереж глибокого навчання припадає на другу половину 2000-х років.

Досліджено засади побудови систем нейронних мереж. Алгоритми нейронних мереж для машинного навчання беруть за основу архітектуру та динамічність мереж нейронів мозку. Штучні нейронні мережі мають здатність навчатися, змінюючи зв’язки між своїми нейронами, тобто вагові коефіцієнти. Такі мережі можуть виконувати безліч завдань з обробки інформації.

Розглянуто принципи розпізнавання зображень. Розпізнавання зображень засноване на методах глибокого навчання Розпізнавання фото- або відеоматеріалів може виконуватися з різним ступенем точності, залежно від типу необхідної інформації. Модель або алгоритм здатні виявити певний елемент, так само, як вони можуть віднести зображення до якоїсь великої категорії за відповідною ознакою.

Розглянуто процес розпізнавання зображень. Розпізнавання зображень засноване на методах глибокого навчання. Розпізнавання зображень працює шляхом аналізу кожного пікселя зображення для вилучення інформації, тому важливо навчати мережу на якісному наборі даних. Кінцева мета навчання полягає в тому, що алгоритм може робити прогнози після аналізу зображення.

Проаналізовано існуючі найпоширеніші архітектури нейронних мереж. Вони включають в себе нейронні мережі прямого поширення, рекурентні нейронні мережі та згорткові нейронні мережі. Найпростішою формою нейронної мережі прямого поширення є одношаровий персептрон. У нейронній мережі прямого поширення інформація рухається лише в одному напрямку — від вхідного шару через приховані шари до вихідного шару. Інформація переміщується прямо через мережу і ніколи не проходить через один вузел двічі.

Рекурентна нейронна мережа використовує послідовні дані або дані часових рядів. У той час як традиційні глибокі нейронні мережі діють за схемою незалежності входів та виходів один від одного, вихід рекурентних нейронних мереж залежить від попередніх елементів у послідовності

Згорткова нейронна мережа являється класом глибоких нейронних мереж, які найчастіше застосовуються для аналізу візуальних зображень. Цей клас є спеціалізованим типом моделі нейронної мережі, призначеної для роботи з двовимірними масивами, тобто даними зображення. Шар згортки відіграє ключову роль у CNN. Він складається з набору математичних операцій спеціалізованого типу лінійної операції.

З’ясовано, що для оцінки пози використовуються два поширені методи: підхід «зверху вниз» та підхід «знизу вгору». У більшості ситуацій підхід «зверху вниз» займає набагато більше часу, ніж підхід «знизу вгору», тому доцільно використовувати методику підходу «знизу вгору»

Розглянуто набір мереж для розпізнавання поз людини. Серед них найбільш використовуваними є HRNet, OpenPose та BlazePose.

Проведено огляд інструментів розробки та навчання нейронних мереж. Для розробки рішення з розпізнавання об'єктів доцільно використовувати мову програмування Python. Розглянуто бібліотеку TensorFlow для Python. TensorFlow надає набір робочих інструментів для розробки та навчання моделей.

Зроблено висновок, що створення моделі відстеження рук займає багато часу та ресурсів, тому для побудови нейронних мереж доцільно використовувати MediaPipe

Для побудови веб-частини проекту доцільно використовувати фреймворк Flask. Flask надає інструменти, бібліотеки та механіку, які дозволяють створювати веб-додаток

Розглянуто використання моделі виявлення пози BlazePose. BlazePose виводить 33 ключові точки відповідно до наведеного нижче порядку впорядкування. За заздалегідь проведеними тестами було виявлено, що найоптимальнішим варіантом для використання у розроблюваному веб-

застосунку є модель BlazePose GHUM Full. Вона є достатньо легкою для безпроблемного запуску на більшості сучасних пристроїв та у той же час надає високий рівень точності, що задовольняє потреби для реалізації даного веб-додатку.

Проведено огляд процесу створення даних для тренування нейронної мережі та обумовлено використання графічних процесорів для процесу навчання нейронної мережі

Розроблено веб-застосунок для оцінки якості та кількості виконання фізичного навантаження на основі технологій використання штучних нейронних мереж. Для розробки застосунку використано мову програмування Python, а також бібліотеку TensorFlow. Для реалізації серверної частини веб-застосунка використано можливості фреймворка Flask.

У якості моделі розпізнавання положення тіла людини використано модель BlazePose Дана модель дозволила обчислити координати 33 ключових точок скелета у 3D-просторі.

Інтерфейс користувача веб-застосунку має в собі відображення кількості виконаних повторень, а також статусу положення рук при виконанні вправи. Додано режим відладки з відображенням розпізнаних частин тіла людини для кращої наочності, а також для процесу відстеження помилок при розпізнаванні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Haykin S. S. Neural networks : a comprehensive foundation. Prentice Hall, 1999. 842 с. ISBN 0139083855.
2. Neural Networks and Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: http://neuralnetworksanddeeplearning.com (дата звернення: 30.03.2022)
3. Afaq S., Rao S. Significance Of Epochs On Training A Neural Network. *International Journal of Scientific and Technology Research*. 2020. Вип. 19, № 6.
4. Günther F., Fritsch S. Neuralnet: Training of neural networks. *R Journal*. 2010. Вип. 2, № 1.
5. Martin T., Howard B., Mark H. Neural network design. *Boston, Massachusetts: PWS*. 1996. С. 4–15.
6. Ramgulam A., Ertekin T., Flemings P. B. An Artificial Neural Network Utility for the Optimization of History Matching Process. 2007.
7. Rawat W., Wang Z. Deep convolutional neural networks for image classification: A comprehensive review. 2017.
8. Cheridito P., Jentzen A., Rossmannek F. Non-convergence of stochastic gradient descent in the training of deep neural networks. *Journal of Complexity*. 2021. Вип. 64.
9. Hupkes D., Dankers V., Mul M., Bruni E. Compositionality Decomposed: How do Neural Networks Generalise? *Journal of Artificial Intelligence Research*. 2020. Вип. 67.
10. Soares F. M., Souza A. M. F. Neural network programming with Java : unleash the power of neural networks by implementing professional Java code. 2016. ISBN 9781785880902.
11. Dabre R., Chu C., Kunchukuttan A. A Survey of Multilingual Neural Machine Translation. *ACM Computing Surveys*. 2020. Вип. 53, № 5.
12. Vlachas P. R., Pathak J., Hunt B. R., Sapsis T. P., Girvan M., Ott E., Koumoutsakos P. Forecasting of spatio-temporal chaotic dynamics with recurrent

neural networks: A comparative study of reservoir computing and backpropagation algorithms. 2019.

13. Timmerman A. Neural networks in finance and investing. Using artificial intelligence to improve realworld performance. *International Journal of Forecasting*. 1997. Вип. 13, № 1.
14. Szeliski R. Computer vision : algorithms and applications. 2011. C. 812.
15. Prince S. J. D. (Simon J. D. Computer vision : models, learning, and inference. 2012. C. 580.
16. Forsyth D., Ponce J. Computer vision : a modern approach. 2012. C. 761.
17. Atienza R. Advanced Deep Learning with Keras : Applying GANs and other new deep reinforcement learning, policy gradients, and more. 2018.
18. Burns S., Amazon Kindle Publishing. Python deep learning : develop your first neural network in Python using TensorFlow, Keras, and PyTorch : step-by-step tutorial for beginners. C. 170.
19. Sewak M., Karim R., Pujari P. Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models using Python. 2018. C. 200.
20. Michelucci U. Applied deep learning: A case-based approach to understanding deep neural networks. Apress Media LLC, 2018. 1–410 c. ISBN 9781484237908.
21. Foxwell H. J. Creating Good Data. *Creating Good Data*. 2020.
22. Lavine B. K., Blank T. R. Feed-Forward Neural Networks. *Comprehensive Chemometrics*. 2009.
23. Kanagachidambaresan G. R., Ruwali A., Banerjee D., Prakash K. B. Recurrent Neural Network. *EAI/Springer Innovations in Communication and Computing*. 2021.
24. Suleiman D., Etaiwi W., Awajan A. Recurrent Neural Network Techniques: Emphasis on Use in Neural Machine Translation. *Informatica (Slovenia)*. 2021. Вип. 45, № 7.
25. Guo J. BackPropagation Through Time. *Manuscript*. 2013. № 1.
26. Dai D. An Introduction of CNN: Models and Training on Neural Network Models. 2021.

27. Hossain M. A., Alam Sajib M. S. Classification of Image using Convolutional Neural Network (CNN). *Global Journal of Computer Science and Technology*. 2019.
28. LISA Lab. Convolutional Neural Networks (LeNet) — DeepLearning 0.1 documentation. *LISA lab*. 2018.
29. Shrestha A., Mahmood A. Review of deep learning algorithms and architectures. 2019.
30. Javadiha M., Andujar C., Lacasa E., Ric A., Susin A. Estimating player positions from padel high-angle videos: Accuracy comparison of recent computer vision methods. *Sensors*. 2021. Вип. 21, № 10.
31. Choi B. Introduction to Python Network Automation. *Introduction to Python Network Automation*. 2021.
32. Mózo B. S. Python Automation Cookbook. 2017.
33. Neethidevan V., Chandrasekaran G. Web automation using selenium web driver python. *International Journal of Recent Technology and Engineering*. 2019. Вип. 7, № 6.
34. Singh P., Manure A. Learn TensorFlow 2.0. 2020.
35. Pang B., Nijkamp E., Wu Y. N. Deep Learning With TensorFlow: A Review. 2020.
36. Géron A. Book Review: Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow, 2nd edition. *O'Reilly Media, Inc.* 2019. С. 838.
37. Tutorials Point. TensorFlow Tutorial. *Tutorials Point (I) Pvt. Ltd.* 2019.
38. Kanter D. Google TPU boosts machine learning. *Microprocessor Report*. 2017. № May.
39. Bazarevsky V., Zhang F. On-Device, Real-Time Hand Tracking with MediaPipe. 2019.
40. Ivan Grishchenko, Bazarevsky V. MediaPipe Holistic — Simultaneous Face, Hand and Pose Prediction, on Device. 2021.
41. Singh A. K., Kumbhare V. A., Arthi K. Real-Time Human Pose Detection and Recognition Using MediaPipe. 2022.

42. Anilkumar A., K.T. A., Sajan S., K.A. S. Pose Estimated Yoga Monitoring System. *SSRN Electronic Journal*. 2021.
43. Taneja S., Gupta P. R. Python as a Tool for Web Server Application Development. 2014.
44. Aslam F. A., Mohammed H. N., Lokhande P. S. Efficient Way Of Web Development Using Python And Flask. *International Journal of Advanced Research in Computer Science*. 2015. Вип. 6, № 2.
45. Trianti C. A., Kristianto B., Hendry Integration of Flask and Python on the Face Recognition Based Attendance System. 2021.
46. MDN Web Docs What is JavaScript? - Learn web development | MDN. 2020.
47. Mariano C. L. Benchmarking JavaScript Frameworks. *Dissertations*. 2017.
48. Pauzi A. S. B., Mohd Nazri F. Bin, Sani S., Bataineh A. M., Hisyam M. N., Jaafar M. H., Ab Wahab M. N., Mohamed A. S. A. Movement Estimation Using Mediapipe BlazePose. 2021.
49. Cabo E. E., Salsalida J. T., Alar H. S. Utilizing Mediapipe Blazepose for a Real-Time Pose Classification of Basic Arnis Striking and Blocking Techniques. *SSRN Electronic Journal*. 2021.
50. Савка М.С. Обумовленність вибору відеокарт замість центральних процесорів при навчанні нейронних мереж. 2019.

Додаток А

SUMMARY

SUMMARY

Algorithms of neural networks for machine learning are based on the architecture and dynamics of neural networks of the brain. They use highly idealized models of neurons. However, the basic or fundamental principle remains the same: artificial neural networks have the ability to learn by changing the connections between their neurons, ie weights. Models of artificial neural networks have been used since the middle of the 20th century. However, the current wave of popularity of neural networks, namely their subspecies - deep learning neural networks dates back to the second half of the 2000s. A common feature of many types of controlled and uncontrolled models of deep learning is that these models have many layers of latent neurons that learn in combination with the back propagation and error gradients of stochastic gradient descent.

Artificial neural networks can be taught to classify data very accurately by adjusting the strength of the connection between their neurons, ie by changing the values of the weights. They can also generalize the result to other data sets, provided that the new data are not too different from the training data. A good example of this type of problem is recognizing objects in images, such as a sequence of camera images taken by a self-driving car. The growing interest in machine learning is largely due to the success of neural networks in the visual recognition of objects.

Image recognition is a subcategory of computer vision and artificial intelligence, which is a set of methods for detecting and analyzing images to automate a task. Recognition of photos or videos can be performed with varying degrees of accuracy, depending on the type of information required

There are various tasks that image recognition can perform, such as:

1) Classification. This is the identification of the "class", ie the category to which the image belongs. An image can only have one class.

2) Marking. This is also a classification task, but with a higher degree of accuracy. The neural network can recognize the presence of several concepts or objects in an image. Therefore, you can assign one or more tags to a specific image.

3) Detection. This is necessary if you want to find an object in the image. Usually, after detecting an item, a bounding box is placed around it.

4) Segmentation. This is also a task of detection. Segmentation can determine the location of an element in an image to the nearest pixel. In some cases it is necessary to be extremely accurate. For example, in the case of autonomous vehicles.

Image recognition is based on deep learning methods. In order for neural networks to recognize one or more concepts in an image, a learning process is required. Image recognition works by analyzing each pixel of an image to extract information, as the human eye does. Therefore, it is important to train the network on a quality data set.

The ultimate goal of learning is that the algorithm can make predictions after image analysis. In other words, the algorithm must be able to assign a class to the image or indicate whether a particular element is present.

With an image recognition system or platform, you can automate business processes and thus increase productivity. Once the model recognizes the element in the image, it can be programmed to perform a specific action.

The simplest form of a neural network of direct propagation is a single-layer perceptron

In a neural network of direct propagation, information moves in only one direction - from the input layer through the hidden layers to the output layer. Information moves directly through the network and never passes through one node twice.

Direct-link neural networks are used to study the relationship between independent variables that serve as input to the network and dependent variables that are referred to as network output. Learning occurs when a set of "training set" samples for which class labels are known is presented to the network and the network weights are adjusted to minimize differences between network outputs and known correct outputs.

A recurrent neural network (RNN) is a type of artificial neural network that uses sequential data or time series data. RNN deep learning architectures are commonly used for tasks such as language translation, natural language processing (NLP), or speech and manuscript recognition.

While traditional deep neural networks operate on a scheme of independence of inputs and outputs from each other, the output of recurrent neural networks depends on the previous elements in the sequence. Although future events may also be useful in determining the outcome of the sequence, unidirectional recurrent neural networks cannot take these events into account in their predictions.

Another characteristic of recurrent networks is that they have common parameters at each network level. While direct propagation networks have different weights for each node, recurrent neural networks have the same weighting parameter in each network layer. These weights are still adjusted in the process of back propagation and gradient descent to facilitate reinforcement training.

Recurrent neural networks use the time backpropagation algorithm (BPTT) to determine gradients. It differs from traditional backpropagation in that it is specific to sequence data. The principles of BPTT do not differ from the traditional backpropagation: the model is trained by calculating errors from source to input.

CNN was first developed and used around the 1980s

In deep learning, convolutional neural network (CNN / ConvNet) is a class of deep neural networks that are most commonly used to analyze visual images. This class is a specialized type of neural network model designed to work with two-dimensional arrays, ie image data, although they can be used with one-dimensional and three-dimensional data.

The convolution layer plays a key role in CNN. It consists of a set of mathematical operations of a specialized type of linear operation

CNN's network architecture is inspired by the functioning of the visual cortex of animals. The field of view analysis is performed using a set of subregions that divide the image. Each subregion analyzes a neuron to pre-process small amounts of information.

Posture assessment is mainly done by identifying key points of the object / people or even determining the location:

1) For objects: The key points will be the corners or edges of the object.

2) For images: images of people, where the key points may be the hands, shoulders, head, feet, etc.

There are many solutions today that use computer vision technology. Thanks to an effective system of tracking and measuring posture assessment, interest in posture assessment technologies is quite high. The following are some applications of posture assessment by example:

1) Metaverse of augmented reality

The concept of the metauniverse is becoming more widespread and is actively developing with the help of the scientific and technological community. It attracts the general attention of all age groups. Augmented reality technologies attach three-dimensional elements to an object or person in the real world to make them look real. The metauniverse creates an environment for people to move into from the real world. Therefore, the technologies of posture assessment, eye, voice and face tracking have found their application in the development of the metaverse.

2) Health and fitness industry.

The rapid growth of the fitness industry during the global epidemic has affected many consumers who are actively joining the exercise to maintain a healthy lifestyle. Rapid increase in fitness supplements that provide effective health monitoring charts and fitness plans. In addition, some programs provide incredible results in the detection of execution errors, as well as customer feedback. These programs use computer-assisted posture assessment technologies to minimize the risk of injury during exercise. Also, one of the useful uses of posture assessment is used in the defense field, where it helps to distinguish between enemies and friendly troops.

3) Robotics

Pose scores are integrated into robotics. They are used in the training of robots, in which they study the movements of people.

Identifying people plays a big role in the recognition process. Due to the recent development of machine learning algorithms (ML), it has become fairly easy to use posture detection and posture tracking.

In traditional methods, such as object detection, a recognizable object is usually highlighted only by a bounding box. Thanks to improved posture detection and tracking, machines can easily learn human body language. Pose evaluation makes it possible to track an object at a sufficiently detailed level. These powerful techniques open up a wide range of applications for real-world applications.

To track human movement and activity, posture assessments have several ranges of application, such as augmented reality, healthcare, and robotics. Currently, the assessment of human posture can be used in different ways, for example, maintaining social distance in the queues of the bank by combining the assessment of human posture and distance projection. This can help people follow the rules and regulations of good hygiene in banks, as well as maintain physical distance in a crowded place.

Another example of the feasibility and effectiveness of tracking and assessing posture is unmanned vehicles. Most accidents caused by self-driving cars occur when vehicles cannot understand the behavior of pedestrians. With the use of posture assessment technologies, the model can learn better.

To track human movement and activity, posture assessments have several ranges of application, such as augmented reality, healthcare, and robotics. Currently, the assessment of human posture can be used in different ways, for example, maintaining social distance in the queues of the bank by combining the assessment of human posture and distance projection. This can help people follow the rules and regulations of good hygiene in banks, as well as maintain physical distance in a crowded place.

Two common methods are used to assess posture:

1) Top-down approach:

2) Bottom-up approach:

There are several models for assessing posture. The choice of model depends on the requirements of the problem. There are also many factors to consider when choosing models. Requirements can be operating time, model size and more.

The following are the most popular pose evaluation libraries that are available for general use. They are easily customizable according to the case of use:

1) Deep Pose

2) PoseNet

3) Blaze pose

4) High-Resolution Net (HRNet)

5) Dense pose

6) Deep cut

7) Regional Multi-Person Pose Estimation (AlphaPose)

8) OpenPose

The most common models used are AlphaPose, HRNet, OpenPose, and BlazePose.

Додаток Б

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську дисертацію

"Застосування нейронних мереж в задачах розпізнавання об`єктів"

1 Назва роботи

Веб-застосунок з розпізнавання та контролю виконання фізичних вправ людиною

2 Підстави для виконання

Робота проводиться на підставі завдання на магістерську дисертацію відповідно до наказу КПІ ім. Ігоря Сікорського від " 14 " квітня 2022 р. № НС-8-2022

3 Мета та актуальність роботи

Метою роботи є створення веб-застосунку з використанням нейронної мережі для визначення кількісних та якісних параметрів активності людини.

Актуальність роботи: штучний інтелект – один із найголовніших напрямів розвитку сфери інформаційних технологій у сьогоденні. Штучні нейронні мережі є відгалуженням від глобального напрямку штучного інтелекту. Нейронні мережі можуть використовуватись для великого спектру задач, у тому числі для обробки зображень. Технологія обробки зображень для розпізнавання образів може забезпечити розпізнавання положення тіла людини з достатньо високою точністю, що дозволить відслідковувати її рухи та підраховувати кількість активності людини.

4 Основні технічні вимоги до виконання роботи

Особливості використання технології розпізнавання об’єктів за допомогою нейронних мереж зумовлюють такі фактори: необхідність правильного підбору моделі розпізнавання задля коректного розпізнавання об’єктів, використання моделі не має навантажувати систему, на якій вона запущена задля покриття більшої кількості підтримуваних для запуску пристроїв.

При розробки веб-застосунку для розпізнавання об'єктів для реалізації у вигляді веб-сайта потрібно враховувати такі обмежуючі фактори:

- підтримка запуску на максимальній кількості сучасних пристроїв;
- навантаження при використанні моделі має бути оптимізованим відповідно до обчислювальних можливостей пристроїв, з якими вона буде використовуватись;
- точність розпізнавання об'єктів;
- відкрита для всіх ліцензія на використання.

5 Вимоги до технологічності

Веб-застосунок має бути достатньо оптимізованим, щоб забезпечити безперебійну роботу на великій кількості пристроїв. Використовувані для реалізації технології мають бути сучасними задля можливості додавати новий функціонал, а також задля підтримки на більшій кількості сучасних пристроїв. З цієї причини є потреба у проведенні аналізу існуючих рішень для реалізації задачі. Модель нейронної мережі та використовувані технології не мають потребувати забагато ресурсів для використання.

6 Вимоги до рівня уніфікації

При розробці проекта необхідно намагатись максимально використовувати найпоширеніші засоби реалізації без використання технологій із закритим вихідним кодом та закритими ліцензіями.

7 Стадії та етапи розробки

1. Аналіз існуючих рішень
2. Вибір технологій для реалізації проекта

3. Розробка плану використання технологій для реалізації проекта
4. Створення концепту веб-застосунка
5. Програмна реалізація проекта

8 Порядок приймання роботи

Робота приймається Державною екзаменаційною комісією.

Додаток В

Програмний код веб-застосунку з розпізнавання та контролю виконання фізичних вправ людиною

general.html

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css"
rel="stylesheet"                                                    integrity="sha384-
+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x"
crossorigin="anonymous">

    <title>CountFit</title>
  </head>
  <body>

    <nav class="navbar navbar-light bg-dark">
        <div class="container">
          <a class="navbar-brand" href="#" style="color: aliceblue;" align="left">
            <img  src="https://icon-library.com/images/calories-icon/calories-icon-
23.jpg" alt="" width="40" height="40">
            NeuraTrain
        </a>
        </div>
      </nav>

{% block content %}

{% endblock %}

  </body>
</html>
```

index.html

```
<!doctype html>
<html lang="en">
```

```
  <head>
    <!-- Необхідні meta теги -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css"
rel="stylesheet"                                                integrity="sha384-
+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x"
crossorigin="anonymous">

    <title>CountFit</title>
  </head>
  <body>

    <nav class="navbar navbar-light bg-dark">
        <div class="container">
          <a class="navbar-brand" href="#" style="color: aliceblue;" align="left">
            <img  src="https://icon-library.com/images/calories-icon/calories-icon-
23.jpg" alt="" width="40" height="40">
            NeuraTrain
        </a>
        </div>
      </nav>


    <h1 style="text-align: left;">Віртуальний тренер</h1>
    <div  style="justify-content:  left;  display:  flex;  margin-top:  30px;  margin-
left: 30px;"><img src="{{ url_for('video') }}"></div>

  </body>
</html>
```

app.py

```
import cv2
import mediapipe as mp
import numpy as np

import counter
```

```
import stage

from flask import Flask, render_template, Response
from camera import Video


app = Flask(__name__)


@app.route('/')
def index():
    return render_template('index.html')


def gen(camera):
    while True:
        frame = camera.get_frame()
        yield(b'--frame\r\n'
              b'Content-Type:  image/jpeg\r\n\r\n' + frame +
              b'\r\n\r\n')


@app.route('/video')
def video():
    return Response(gen(Video()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')


app.run(debug=True)
```

calcAngle.py

```
import numpy as np


def calculate_angle(first, mid, end):
    first = np.array(first)  # Перша точка
    mid = np.array(mid)  # Друга точка
    end = np.array(end)  # Третя точка
```

```
    radians = np.arctan2(end[1]-mid[1], end[0]-mid[0]) - np.arctan2(first[1]-
mid[1], first[0]-mid[0])
    angle = np.abs(radians*180.0/np.pi)

    if angle > 180.0:
        angle = 360-angle

    return angle
```

counter.py

```
def init():
    global counter
    counter = 0
```

stage.py

```
def init():
    global stage
    stage = None
```

camera.py

```
import cv2
import mediapipe as mp
import numpy as np
from calcAngle import calculate_angle
import counter
import stage
counter.init()
stage.init()

mediapipe_draw = mp.solutions.drawing_utils
mediapipe_pose = mp.solutions.pose


class Video(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)
```

```
    def __del__(self):
        self.video.release()
    def get_frame(self):


        # Налаштування mediapipe
        with                    mediapipe_pose.Pose(min_detection_confidence=0.55,
min_tracking_confidence=0.55 , model_complexity=2) as pose:
            ret, frame = self.video.read()

            # Зміна кольорово схеми на RGB
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image.flags.writeable = False

            # Процес розпізнавання
            results = pose.process(image)

            # Зміна кольорової схеми назад до BGR
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

            # Отримання точок положення частин тіла
            try:
                landmarks = results.pose_landmarks.landmark

                # Отримання координатів для правої половини тіла

                hip_right                                                    =
[landmarks[mediapipe_pose.PoseLandmark.RIGHT_HIP.value].x,
                        landmarks[mediapipe_pose.PoseLandmark.RIGHT_HIP.value].y]
                shoulder_right                                               =
[landmarks[mediapipe_pose.PoseLandmark.RIGHT_SHOULDER.value].x,

landmarks[mediapipe_pose.PoseLandmark.RIGHT_SHOULDER.value].y]
                elbow_right                                                  =
[landmarks[mediapipe_pose.PoseLandmark.RIGHT_ELBOW.value].x,

landmarks[mediapipe_pose.PoseLandmark.RIGHT_ELBOW.value].y]

                # Отримання координатів для лівої половини тіла
```

```
                hip_left                                                    =
[landmarks[mediapipe_pose.PoseLandmark.LEFT_HIP.value].x,
                        landmarks[mediapipe_pose.PoseLandmark.LEFT_HIP.value].y]
                shoulder_left                                               =
[landmarks[mediapipe_pose.PoseLandmark.LEFT_SHOULDER.value].x,

landmarks[mediapipe_pose.PoseLandmark.LEFT_SHOULDER.value].y]
                elbow_left                                                  =
[landmarks[mediapipe_pose.PoseLandmark.LEFT_ELBOW.value].x,

landmarks[mediapipe_pose.PoseLandmark.LEFT_ELBOW.value].y]


                # Прорахунок кутів
                angle_left = calculate_angle(hip_left, shoulder_left, elbow_left)
                angle_right    =    calculate_angle(hip_right,    shoulder_right,
elbow_right)

                # Візуалізація значення кута
                cv2.putText(image, str(angle_left),
                            tuple(np.multiply(
                            shoulder_left, [640, 480]).astype(int)),
                            cv2.FONT_HERSHEY_SCRIPT_SIMPLEX, 0.7, (
                            255, 255, 255), 2, cv2.LINE_AA
                            )
                cv2.putText(image, str(angle_right),
                            tuple(np.multiply(
                            shoulder_right, [640, 480]).astype(int)),
                            cv2.FONT_HERSHEY_SCRIPT_SIMPLEX, 0.7, (
                            255, 255, 255), 2, cv2.LINE_AA
                            )

                # Логіка підрахунку виконаних разів
                if angle_left < 30 and angle_right < 30:
                    stage.stage = "DOWN"
                if angle_left > 90 and angle_right > 90 and stage.stage == 'DOWN':
                    stage.stage = "UP"
                    counter.counter += 1
                    print(counter.counter)

            except:
                pass
```

```
            # Відображнення виконаних разів
            # Налаштування поля відображення
            cv2.rectangle(image, (0, 0), (640, 95), (255, 255, 255), -1)
            #cv2.rectangle(image, (0, 0), (125, 95), (255, 255, 255), -1)
            #cv2.rectangle(image, (425, 0), (640, 95), (255, 255, 255), -1)

            # Відображення кількості виконаних разів
            cv2.putText(image, 'РАЗ(ІВ)', (25, 15),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
            cv2.putText(image, str(counter.counter),
                        (25, 75),
                        cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 0), 2, cv2.LINE_AA)

            # Відображення статусу положення рук
            cv2.putText(image, 'СТАН', (515, 15),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
            cv2.putText(image, stage.stage,
                        (450, 75),
                        cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 0), 2, cv2.LINE_AA)

            # Відображення розпізнаних точок тіла
            mediapipe_draw.draw_landmarks(image,           results.pose_landmarks,
mediapipe_pose.POSE_CONNECTIONS,
                                      mediapipe_draw.DrawingSpec(
                                          color=(0,    0,    255),    thickness=3,
circle_radius=2),
                                      mediapipe_draw.DrawingSpec(
                                          color=(255,    255,    0),    thickness=3,
circle_radius=3)
                                      )


            ret, jpg = cv2.imencode('.jpg', image)
            return jpg.tobytes()
```