Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій

(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Другий (магістерський)
(рівень вищої освіти)

Розроблення програмного забезпечення для автоматизованої ідентифікації знаків дорожнього руху

Виконав:
студент 2 курсу, групи КІТПВм-21-1

Нікітін В.В
(прізвище, ініціали)

Спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)

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

Освітня програма

(повна назва освітньої програми)

Керівник доц. Максимова С.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри КІТАМ _____ Невлюдов І. Ш.
(підпис) (прізвище, ініціали)

2022р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКТ

Кафедра КІТАМ

Рівень вищої освіти другий (магістерський)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології

Тип програми Освітньо-професійна

Освітня програма

(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАМ
_____
(підпис)
«____»_____20 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Нікітіну Владиславу Віталійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення для автоматизованої ідентифікації знаків дорожнього руху

Затверджена наказом по університету від №

2. Термін подання студентом роботи до екзаменаційної комісії

3. Вихідні дані до роботи

3.2 Мови програмування JavaScript, ReactJs

4. Перелік питань, що потрібно опрацювати в роботі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій демонстраційний матеріал, представлений у форматі презентації PowerPoint (*.pptx)

6. Консультанти розділів роботи

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Отримання завдання на кваліфікаційну работу		
2	Вступ		
3	Аналіз технічного завдання		
4	Аналіз інструментів розробки та аналіз технічного завдання		
5	Проєктування системи		
6	Розробка програмного забезпечення вебдодатку		
7	Охорона праці		
8	Висновки		
9	Подання роботи на перевірку Інтернет-сервісом Unicheck		
11	Оформлення пояснювальної записки		
12	Подання роботи на рецензію		
13	Подання роботи на підпис зав. кафедри		
14	Подання атестаційної роботи в ЕК		

Дата видачі завдання _____

Студент _____ Нікітін В.В.

(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Максимова С.С.

(підпис) (посада, прізвище, ініціали

РЕФЕРАТ

Пояснювальна записка: 97 с., 10 табл., 45 рис., 1 дод., 20 джерел.

TENSERFLOW, ВЕБДОДАТОК, КОМП’ЮТЕРНИЙ ЗІР, KERAS.

Таким чином, тема атестаційної роботи, пов’язана з розробленням програмного забезпечення для автоматизованої ідентифікації знаків дорожнього руху, є актуальною.

Об’єкт дослідження – комп’ютерний зір.

Предмет дослідження – визначення об'єктів за допомогою TensorflowJS.

Мета кваліфікаційної роботи – автоматизація технологій визначення знаків дорожнього руху.

Методи дослідження – аналіз та порівняння, метод класифікації, повний факторний експеримент.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– обрати метод машинного навчання моделі;

– обрати середовище для розробки додатка;

– вивчити предметну область вже існуючих програмних рішень;

– обрати мову програмування;

– описати структуру програмного додатку;

– розробити алгоритми конвертації моделей для вебдодатка;

– розробити програмне забезпечення.

У кваліфікаційній роботі досліджено методи машинного навчання моделей для розпізнавання об’єктів у реальному часі за допомогою вебкамери комп’ютера або камери іншого пристрою який сумісний для роботи додатку.

Для цього проведено аналіз та порівняння існуючих методів машинного навчання для розпізнавання об’єктів за допомогою технологій OpenCV та похідних від цієї технологій розпізнавання об’єктів розроблених на базі OpenCV та реалізованих мовою програмування JavaScript і бібліотеки Tensorflow.

Проведено моделювання залежності зміни основних параметрів пресформи: висота неактивного шару, відстані від перегородки до камери, відстань перегородок між камерами, які впливають на конструктивно-функціонуванні параметри «пальця».

На базі проведеного дослідження розроблено додаток для створення моделей машинного навчання за допомогою бібліотеки Tensorflow. Для цього використано методи регресійного аналізу, зроблено постановку повного факторного експерименту, які найбільше впливають на функціонування та дозволяють ідентифікувати математичні моделі.

Результати кваліфікаційної роботи апробовані у 3 фахових статтях та 2 міжнародних конференціях.

ABSTRACT

Explanatory note: 97 pp., 10 tables, 45 figures, 1 appendix, 20 sources.

TENSORFLOW, WEB APP, COMPUTER VISION, KERAS.

Thus, the topic of certification work related to the development of software for automated identification of traffic signs is relevant.

The object of research is computer vision.

The subject of the study is the definition of objects using TensorflowJS.

The purpose of the qualification work is the automation of technologies for determining traffic signs.

Research methods – analysis and comparison, classification method, full factorial experiment.

To achieve the goal, the following tasks must be solved:

– choose the machine learning method of the model;

– choose an environment for application development;

– study the subject area of already existing software solutions;

– choose a programming language;

– describe the structure of the software application;

– develop model conversion algorithms for a web application;

– develop software.

In the qualification work, the methods of machine learning of models for recognizing objects in real time using a computer webcam or the camera of another device that is compatible with the operation of the application were investigated.

For this, an analysis and comparison of existing machine learning methods for object recognition using OpenCV technologies and derivatives of these object recognition technologies developed on the basis of OpenCV and

implemented in the JavaScript programming language and the Tensorflow library was carried out.

The simulation of the dependence of the change in the main parameters of the mold was carried out: the height of the inactive layer, the distance from the partition to the chamber, the distance between the partitions between the chambers, which affect the constructive and functioning parameters of the "finger".

Based on the conducted research, an application was developed for creating machine learning models using the Tensorflow library. For this, regression analysis methods were used, and a full factorial experiment was set up, which have the greatest effect on functioning and allow identifying mathematical models.

The results of the qualification work were tested in 3 professional articles and 2 international conferences.

ЗМІСТ

Перелік скорочень 10
Вступ 11
1 Аналіз технічного завдання 12
1.1 Аналіз актуальності програмного забезпечення контролю дорожніх знаків 12
1.2 Аналіз технологій розпізнавання дорожніх знаків 13
1.3 Аналіз систем аналогів 14
1.4 Обґрунтування вибору програмних засобів 15
1.5 Висновки до 1 розділу 19
2 Розробка вебдодатку для навчання моделі комп'ютерного зору 20
2.1 Створення вебдодатка за допомогою TenserFlow 20
2.2 Трансферне навчання 21
2.3 Створення структури вебдодатку 26
2.4 Висновки до 2 розділу 42
3 Розроблення програмного забезпечення розпізнавання знаків дорожнього руху 43
3.1 Основні поняття для створення React додатка 43
3.2 Створення React середовища розробки вебдодатку 56
3.3 Висновки до 3 розділу 67
4 Безпека життєдіяльності 68
4.1 Виявлення шкідливих і небезпечних факторів в приміщенні лабораторії 69
4.1.1 Несприятливі параметри мікроклімату 69
4.1.2 Відсутність або недостатня освітленість робочої зони 70
4.1.3 Підвищений рівень іонізації повітря 70
4.1.4 Підвищений рівень статичної електрики 70
4.2 Заходи щодо захисту від шкідливих і небезпечних факторів 70
4.3 Розрахунок природного освітлення 72

Висновки 75
Перелік джерел посилання 88
Додаток А Лістинг програми..... 90
Додаток Б Демонстраційний матеріал..... 98

ПЕРЕЛІК СКОРОЧЕНЬ

ВБ – веб додаток

МН – машинне навчання

НДР – науково-дослідна робота

ПЗ – програмне забезпечення

ТН – трансферне навчання

CSS – Cascading Style Sheets

IDE – Integrated Development Environment

HTML – HyperText Markup Language

ВСТУП

Сьогодні в Україні налічується приблизно 169,5 тисяч км державних доріг. Мережа основних маршрутів поширена по всій країні і з’єднує всі великі міста України, а також надає транскордонні маршрути із сусідніми країнами. Якщо брати у приклад місто Київ, то максимальна добова кількість активних автомобілів на дорогах була встановлена на січень 2022 року, вона становила 1,13 мільйона одиниць. За даними КМДА, у 2021 році порівняно з 2020 роком приріст середньодобової кількості авто становив 21 відсоток, та прогнозується ще більший приріст у майбутньому.

Через такий приріст автомобілів на дорогах України, зростає кількість правопорушень, за статистикою, кожні 15 хвилин трапляється одне правопорушення або ДТП.

Для допомоги та контролю знаків дорожнього руху найчастіше використовують навігаційні системи, такі системи використовують попереднє завантаженні мапи. Зазвичай у таких навігаційних системах для контролю пристроєм використовується сенсорний екран, якість екрана у навігаційних системах набагато гірша ніж у сучасних смартфонах, винятком можуть бути лише найсучасніші системи, але зазвичай їх вартість дуже велика. Ще одним винятком може бути несвоєчасне оновлення знаків дорожнього руху, або припинення підтримки виробником, і також застарілість карт у таких виробах.

Об’єкт дослідження – комп’ютерний зір.

Предмет дослідження – визначення об'єктів за допомогою TensorflowJS.

Методи дослідження – аналіз та порівняння, метод класифікації, повний факторний експеримент.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– обрати метод машинного навчання моделі;

– обрати середовище для розробки додатка;

– вивчити предметну область вже існуючих програмних рішень;

– обрати мову програмування;

– описати структуру програмного додатку;

– розробити алгоритми конвертації моделей для вебдодатка;

– розробити програмне забезпечення;

– оформити пояснювальну записку згідно з рекомендаціями [1], та вимогами ДСТУ 3008:2015 [2].

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз актуальності програмного забезпечення контролю дорожніх знаків

Сьогодні в Україні налічується приблизно 169,5 тисяч км державних доріг. Мережа основних маршрутів поширена по всій країні і з’єднує всі великі міста України, а також надає транскордонні маршрути із сусідніми країнами. Якщо брати у приклад місто Київ, то максимальна добова кількість активних автомобілів на дорогах була встановлена на січень 2022 року, вона становила 1,13 мільйона одиниць. За даними КМДА, у 2021 році порівняно з 2020 роком приріст середньодобової кількості авто становив 21 відсоток, та прогнозується ще більший приріст у майбутньому.

Через такий приріст автомобілів на дорогах України, зростає кількість правопорушень, за статистикою, кожні 15 хвилин трапляється одне правопорушення або ДТП.

Для допомоги та контролю знаків дорожнього руху найчастіше використовують навігаційні системи, такі системи використовують попереднє завантаженні мапи. Зазвичай у таких навігаційних системах для контролю пристроєм використовується сенсорний екран, якість екрана у навігаційних системах набагато гірша ніж у сучасних смартфонах, винятком можуть бути лише найсучасніші системи, але зазвичай їх вартість дуже велика. Ще одним винятком може бути несвоєчасне оновлення знаків дорожнього руху, або припинення підтримки виробником, і також застарілість карт у таких виробах.

Також для допомоги водіїв використовуються інтегровані в автомобіль навігаційні системи, такі системи можуть доповнюватись зовнішніми камерами, які у реальному часі аналізують ситуацію навколо автомобіля, та можуть попереджати водія при перевищенні

швидкісного режиму покладаючись на знаки дорожнього руху. Але, такі системи не мають масовий характер у автомобільному світі, та найчастіше встановлюються на автомобілі найвищого класу, яких на дорогах найменше.

1.2 Аналіз технологій розпізнавання дорожніх знаків

Для розпізнавання дорожних знаків підходять різні технології. Одна з найпопулярніші OpenCV (Open Source Computer Vision Library) – це бібліотека комп'ютерного зору та машинного навчання з відкритим кодом. OpenCV було створено, щоб забезпечити загальну інфраструктуру для програм комп'ютерного зору та прискорити використання машинного сприйняття в комерційних продуктах.

Бібліотека містить понад 2500 оптимізованих алгоритмів, що включає повний набір як класичних, так і найсучасніших алгоритмів комп'ютерного зору та машинного навчання. Ці алгоритми можна використовувати для виявлення та розпізнавання облич, ідентифікації об'єктів, класифікації дій людей у відео, відстеження рухів камери, відстеження рухомих об'єктів, вилучення 3D-моделей об'єктів, створення 3D-хмар точок із стереокамер, з'єднання зображень для отримання високої роздільної здатності. зображення цілої сцени, знаходити подібні зображення в базі даних зображень, видаляти червоні очі із зображень, зроблених за допомогою спалаху, стежити за рухами очей, розпізнавати пейзаж і встановлювати маркери для накладання на нього доповненої реальності тощо. OpenCV має понад 47 тисяч користувачів спільнота та оціночна кількість завантажень перевищує 18 мільйонів. Бібліотека широко використовується компаніями, дослідницькими групами та державними органами.

OpenCV має інтерфейси C++, Python, Java і MATLAB і підтримує Windows, Linux, Android і Mac OS, також здебільшого

орієнтується на програми бачення в реальному часі та використовує переваги інструкцій MMX і SSE, якщо вони доступні. Зараз активно розробляються повнофункціональні інтерфейси CUDA і OpenCL. Існує понад 500 алгоритмів і приблизно в 10 разів більше функцій, які створюють або підтримують ці алгоритми. OpenCV написаний на C++ і має шаблонний інтерфейс, який бездоганно працює з контейнерами STL [1].

За допомогою OpenCV можливо створювати додатки, які будуть якісно розпізнавати дорожню розмітку, та працювати на багатьох платформах. Одночасно з цим з’являються проблема у тому, що для кожного пристрою потрібно створювати свій додаток, та використовувати різні технології, через це дуже трудомісткий процес, який потребує поглибленні знання різних мов програмування. Також за допомогою OpenCV немає можливості створити додаток для пристроїв на базі IOS.

1.3 Аналіз систем аналогів

Для допомоги та контролю знаків дорожнього руху найчастіше використовують навігаційні системи, такі системи використовують попереднє завантаженні мапи. Зазвичай у таких навігаційних системах для контролю пристроєм використовується сенсорний екран, якість екрану у навігаційних системах набагато гірша ніж у сучасних смартфонах, винятком можуть бути лише найсучасніші системи, але зазвичай їх вартість дуже велика. Ще одним винятком може бути несвоєчасне оновлення знаків дорожнього руху, або припинення підтримки виробником, і також застарілість карт у таких виробах.

Також для допомоги водіїв використовуються інтегровані в автомобіль навігаційні системи, такі системи можуть доповнюватись зовнішніми камерами, які у реальному часі аналізують ситуацію

навколо автомобіля, та можуть попереджати водія при перевищенні швидкісного режиму покладаючись на знаки дорожнього руху. Але, такі системи не мають масовий характер у автомобільному світі, та найчастіше встановлюються на автомобілі найвищого класу, яких на дорогах найменше.

1.4 Обґрунтування вибору програмних засобів

Рішенням проблем зв’язаних з розробкою програмного забезпечення за допомогою OpenCV стає створення веб додатку. За допомогою такого підходу можливо вирішити одразу декілька проблем. Сьогодні більшість пристроїв якими людина користується кожен день, мають доступ до інтернету за допомогою браузерів. Створивши веб додаток з'являється змога користуватися одним і тим же додатком, незважаючи на те, якою операційною системою людина користується. Тобто розробник повинен володіти лише одним набором навичок, та закрити потребу для користувачів будь-яких девайсів. Інша проблема, яку вирішую створення веб додатка, це проблема пам'яті. Браузер встановлений за замовчуванням у більшості пристроїв, користувачу не потрібно завантажувати ніякі додаткові додатки, лише мати адресу до веб додатку, яка не займає додаткове місце на пристрої, та завжди працює найостанніша версія із доступних [1].

Для рішення такої задачі, можна використовувати TensorFlow.js – це бібліотека з відкритим вихідним кодом, яку можна використовувати для визначення, навчання та запуску моделей машинного навчання повністю у браузері з використанням Javascript та API-інтерфейсу шарів високого рівня.

Використання моделі TensorFlow.js експоненціально зросло за останні кілька років, і зараз багато розробників JavaScript прагнуть

використовувати існуючі найсучасніші моделі та перенавчати їх для роботи з користувацькими даними, унікальними для їх галузі. Дія взяття існуючої моделі (яку часто називають базовою моделлю) і використання її в схожій, але іншій області, відома як трансферне навчання.

Трансферне навчання має багато переваг перед початком навчання з абсолютно порожньої моделі. Ви можете повторно використовувати знання, отримані з попередньої навченої моделі, і вам знадобиться менше прикладів нового елемента, який ви хочете класифікувати. Крім того, навчання часто відбувається значно швидше через те, що потрібно перенавчати лише кілька останніх рівнів архітектури моделі замість усієї мережі. З цієї причини передавання навчання дуже добре підходить для середовища веббраузера, де ресурси можуть відрізнятися залежно від пристрою виконання, але також має прямий доступ до датчиків для легкого збору даних [1].

Враховуючи портативність JavaScript, тепер ви можете писати однією мовою та з легкістю виконувати машинне навчання на всіх наступних платформах:

Клієнтська сторона у веббраузері за допомогою vanilla JavaScript:

– серверна сторона та навіть пристрої IoT, такі як Raspberry Pi, які використовують Node.js;

– настільні програми, що використовують Electron;

– рідні мобільні програми, що використовують React Native.

TensorFlow.js також підтримує кілька серверних програм у кожному з цих середовищ (фактичних апаратних середовищ, у яких він може виконуватися, наприклад, ЦП або WebGL. «Бекенд» у цьому контексті не означає середовище на стороні сервера – серверна частина для виконання може бути на стороні клієнта, наприклад у

WebGL), щоб забезпечити сумісність, а також забезпечити швидку роботу. Наразі TensorFlow.js підтримує:

– виконання WebGL на графічній карті (GPU) пристрою – це найшвидший спосіб виконання більших моделей (розміром понад 3 МБ) із прискоренням GPU;

– виконання веб-асамблеї (WASM) на ЦП – для підвищення продуктивності ЦП на різних пристроях, включаючи, наприклад, мобільні телефони старого покоління. Це краще підходить для менших моделей (розміром менше 3 МБ), які насправді можуть працювати швидше на ЦП із WASM, ніж із WebGL, через накладні витрати на завантаження вмісту до графічного процесора;

– виконання ЦП – резервний варіант, якщо жодне з інших середовищ не буде доступним. Це найповільніший із трьох, але завжди під рукою.

Запуск TensorFlow.js у веб-браузері на клієнтській машині може призвести до кількох переваг [2].

Конфіденційність, модливо тренувати та класифікувати дані на клієнтській машині, не надсилаючи дані на веб-сервер третьої сторони. Бувають випадки, коли це може бути вимогою для дотримання місцевих законів, таких як GDPR, наприклад, або під час обробки будь-яких даних, які користувач може захотіти зберегти на своїй машині, а не надсилати третій стороні.

Швидкість, оскільки не потрібно надсилати дані на віддалений сервер, висновок (акт класифікації даних) може бути швидшим. Що ще краще, є прямий доступ до датчиків пристрою, таких як камера, мікрофон, GPS, акселерометр тощо, якщо користувач надасть доступ.

Будь-хто у світі може натиснути на посилання, відкрити веб-сторінку у своєму браузері та використати додаток. Немає потреби в складному налаштуванні Linux на стороні сервера з драйверами CUDA

та багато іншого, щоб використовувати лише систему машинного навчання.

Відсутність вартості серверів означає, що єдине, за що потрібно заплатити, – це CDN для розміщення файлів HTML, CSS, JS і моделей. Вартість CDN набагато дешевша, ніж утримання сервера (можливо з підключеною відеокартою) у цілодобовій роботі без вихідних.

Серверні функції, використовуючи реалізацію TensorFlow.js у Node.js, дають багато функцій. Повна підтримка CUDA, на стороні сервера, для прискорення відеокарти, ви повинні встановити драйвери NVIDIA CUDA, щоб увімкнути TensorFlow для роботи з відеокартою (на відміну від браузера, який використовує WebGL – інсталяція не потрібна). Однак завдяки повній підтримці CUDA ви можете повністю використовувати можливості графічної карти нижчого рівня, що призведе до швидшого навчання та часу висновку. Продуктивність на рівні з реалізацією Python TensorFlow, оскільки вони обидва використовують один і той самий сервер C++.

Для найновіших моделей із досліджень можливо працювати з дуже великими моделями, можливо, розміром у гігабайти. Ці моделі наразі не можна запускати у веб-браузері через обмеження використання пам’яті для кожної вкладки браузера. Щоб запустити ці більші моделі, треба використовувати Node.js на своєму власному сервері зі специфікаціями апаратного забезпечення, які необхідні для ефективного запуску такої моделі.

IOT, Node.js підтримується на популярних одноплатних комп’ютерах, таких як Raspberry Pi, що, у свою чергу, означає, що є можливість виконувати моделі TensorFlow.js на таких пристроях.

Node.js написаний на JavaScript, що означає, що він отримує переваги від своєчасної компіляції. Це означає, що під час використання Node.js можна спостерігати підвищення продуктивності,

оскільки він буде оптимізовано під час виконання, особливо для будь-якої попередньої обробки, яка виконується.

Запуск ML(machine learning) у браузері означає, що з точки зору користувача немає необхідності встановлювати бібліотеки або драйвери. Просто відкрийте веб-сторінку і ваша програма готова до запуску. Крім того, вона готова працювати з прискоренням GPU. TensorFlow.js автоматично підтримує WebGL і прискорить ваш код одразу, як з'явиться графічний процесор. Користувачі також можуть відкривати веб-сторінку з мобільного пристрою, і в цьому випадку ваша модель може використовувати дані датчиків, наприклад, з гіроскопа або акселерометра. Важливо ще й те, що всі дані залишаються у клієнта, що робить TensorFlow.js корисним для low-latency виводу, а також додатків, що зберігають конфіденційність [2].

Запуск TensorFlow.js у веб-браузері на клієнтській машині може призвести до кількох переваг

При роботі з TensorFlow.js можна використовувати три робочі процеси:

– можливо імпортувати існуючу, попередньо навчену модель для виведення. Якщо доступна існуюча модель TensorFlow або Keras, яка раніше навчалася в автономному режимі, можна перетворити її на формат TensorFlow.js і завантажити її у браузер для виведення;

– можливість використовувати трансферне навчання, щоб доповнити існуючу модель, навчену в автономному режимі, використовуючи невеликий обсяг даних, зібраних у браузері, з використанням методу Image Retraining. Це один із способів швидкого навчання точної моделі, використовуючи лише невеликий обсяг даних;

– створення моделі прямо у браузері за допомогою TensorFlow.js, щоб повністю визначати, навчати та запускати моделі у

браузері з використанням Javascript та API-інтерфейсу шарів високого рівня.

1.5 Висновки до 1 розділу

Виходячи з моїх досліджень у сфері веб додатків з визначення дорожніх знаків, я можу зробити висновок, що аналогів тієї системи, яку я розробляю – не існує. Перевагою такого додатка буде кросплатформність, розробляти його можна без труднощів на будь-якій платформі, чи то windows, linux чи macos.

2 РОЗРОБКА ВЕБДОДАТКУ ДЛЯ НАВЧАННЯ МОДЕЛІ КОМП'ЮТЕРНОГО ЗОРУ

2.1 Створення вебдодатку за допомогою TensorFlow

TensorFlow.js – це бібліотека машинного навчання з відкритим вихідним кодом, яка може працювати будь-де, де може працювати JavaScript. Він базується на оригінальній бібліотеці TensorFlow, написаній на Python, і має на меті відтворити цей досвід розробника та набір API для екосистеми JavaScript.

Враховуючи портативність JavaScript можна писати однією мовою та з легкістю виконувати машинне навчання на всіх наступних платформах:

– клієнтська сторона у веб-браузері за допомогою ванільного JavaScript;

– серверна сторона та навіть пристрої IoT, такі як Raspberry Pi, які використовують Node.js;

– настільні програми, що використовують Electron;

– рідні мобільні програми, що використовують React Native.

TensorFlow.js також підтримує кілька серверних програм у кожному з цих середовищ (фактичних апаратних середовищ, у яких він може виконуватися, наприклад, ЦП або WebGL. «Бекенд» у цьому контексті не означає середовище на стороні сервера – серверна частина для виконання може бути на стороні клієнта, наприклад у WebGL), щоб забезпечити сумісність, а також забезпечити швидку роботу. Наразі TensorFlow.js підтримує:

– виконання WebGL на графічній карті (GPU) пристрою – це найшвидший спосіб виконання більших моделей (розміром понад 3 МБ) із прискоренням GPU;

– виконання веб-асамблеї (WASM) на ЦП – для підвищення продуктивності ЦП на різних пристроях, включаючи, наприклад, мобільні телефони старого покоління. Це краще підходить для менших моделей (розміром менше 3 МБ), які насправді можуть працювати швидше на ЦП із WASM, ніж із WebGL, через накладні витрати на завантаження вмісту до графічного процесора;

– виконання ЦП – резервний варіант, якщо жодне з інших середовищ не буде доступним. Це найповільніший із трьох, але завжди під рукою.

2.2 Трансферне навчання

Трансферне навчання передбачає використання вже отриманих знань, щоб допомогти навчитися іншим, але схожим речам [3].

Ми, люди, робимо це постійно. У вашому мозку все життя накопичений досвід, який ви можете використовувати, щоб допомогти розпізнати нові речі, яких ви ніколи раніше не бачили. Візьмемо, наприклад, цю вербу (рис. 2.1).

Рисунок 2.1 – Перший приклад розпізнавання об’єктів

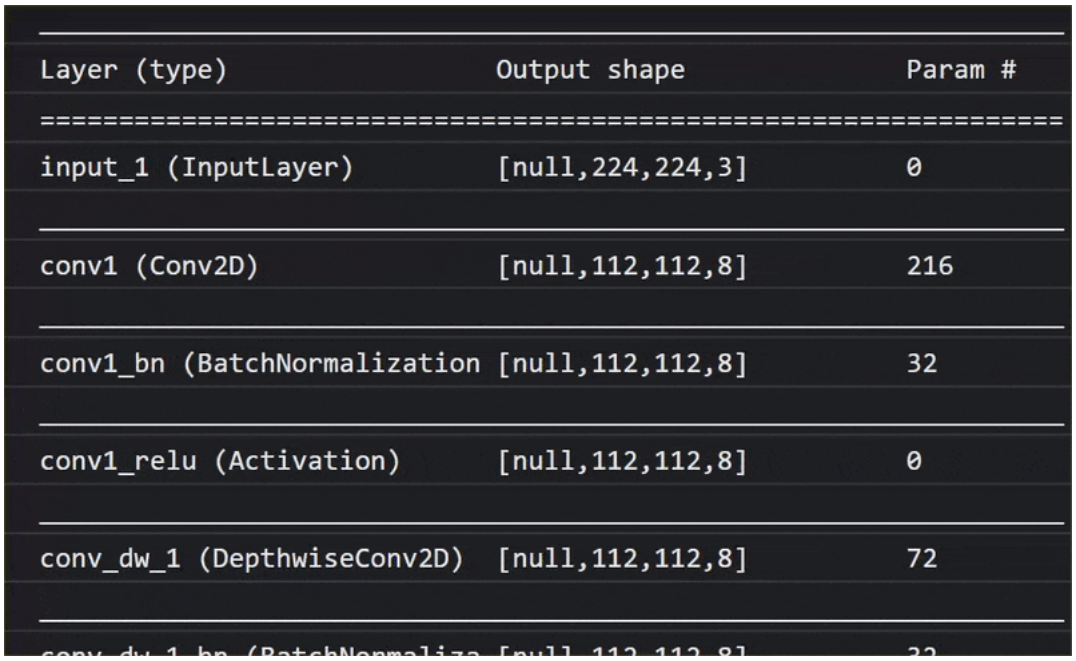
Але людина зможе досить швидко помітити різницю між цими об’єктами, навіть якщо вони під іншим кутом і трохи відрізняються від оригінального (рис. 2.2).

Рисунок 2.2 – Другий приклад розпізнавання об’єктів

Подібним чином, якщо є модель машинного навчання, яка вже навчена в домені, наприклад розпізнавання зображень, можна повторно використовувати її для виконання іншого, але пов’язаного завдання.

Можна зробити те саме за допомогою вдосконаленої моделі, як-от MobileNet, яка є дуже популярною дослідницькою моделлю, яка може виконувати розпізнавання зображень на 1000 різних типах об’єктів. Від собак до автомобілів, його навчили на величезному наборі даних, відомому як ImageNet, який містить мільйони позначених зображень [3].

На цій анімації можна побачити величезну кількість шарів у моделі MobileNet V1 (рис. 2.3).



Layer (type)	Output shape	Param #
input_1 (InputLayer)	[null,224,224,3]	0
conv1 (Conv2D)	[null,112,112,8]	216
conv1_bn (BatchNormalization	[null,112,112,8]	32
conv1_relu (Activation)	[null,112,112,8]	0
conv_dw_1 (DepthwiseConv2D)	[null,112,112,8]	72

Рисунок 2.3 – Шари у моделі MobileNet V1

Під час навчання ця модель навчилася виділяти загальні ознаки, важливі для всіх цих 1000 об’єктів, і багато функцій нижчого рівня, які вона використовує для ідентифікації таких об’єктів, можуть бути корисними для виявлення нових об’єктів, яких вона ніколи раніше не бачила. Зрештою, в кінцевому підсумку все є лише поєднанням ліній, текстур і форм.

Подивившись на традиційну архітектуру згорткової нейронної мережі (CNN) (подібну до MobileNet) можга побачити, як навчання передачі може використовувати цю навчену мережу, щоб дізнатися щось нове. На зображенні нижче показано типову архітектуру моделі CNN, яка в цьому випадку була навчена розпізнавати рукописні цифри від 0 до 9 (рис. 2.4).

Якщо відокремити попередньо навчені рівні нижнього рівня існуючої навченої моделі, як показано ліворуч, від шарів класифікації в кінці моделі, показаних праворуч (іноді їх називають головою класифікації моделі), ви можете використовувати шари нижчого рівня, щоб створити вихідні функції для будь-якого заданого зображення на основі вихідних даних, на яких воно було навчено. Ось та сама мережа з видаленою головою класифікації (рис. 2.5).

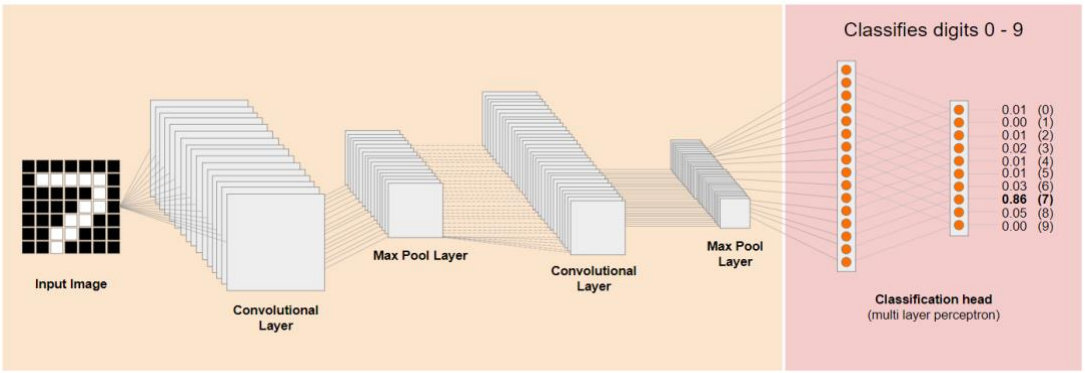


Рисунок 2.4 – Архітектуру моделі CNN

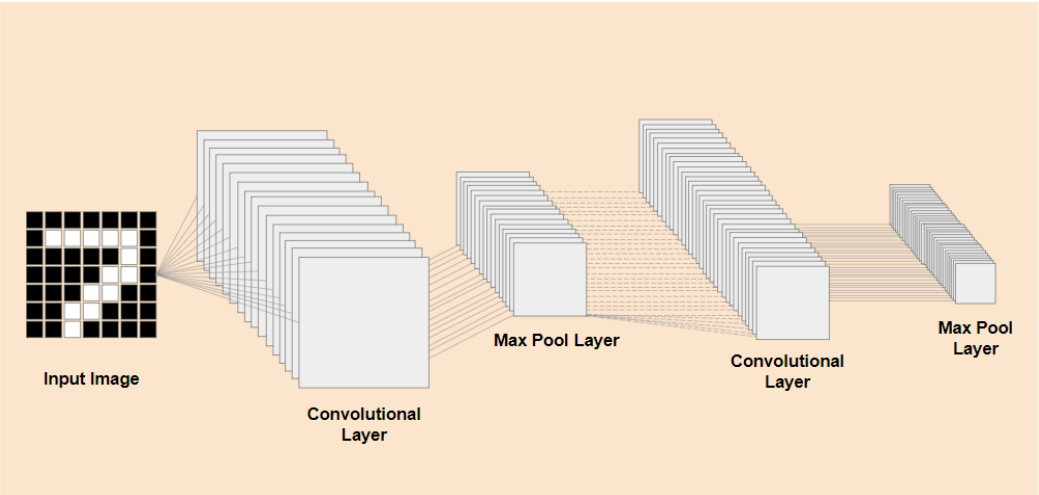


Рисунок 2.5 – Мережа з видаленою головою класифікації

Якщо припустити, що новий предмет, який ви намагаєтесь розпізнати, також може використовувати такі вихідні функції, які навчила попередня модель, тоді є хороші шанси, що їх можна повторно використати для нової мети.

На діаграмі вище цю гіпотетичну модель було навчено на цифрах, тож, можливо, те, що ми вивчили про цифри, також можна застосувати до таких літер, як a, b і c.

Отже, тепер можна додати нову головку класифікації, яка замість цього намагатиметься передбачити a, b або c, як показано (рис. 2.6).

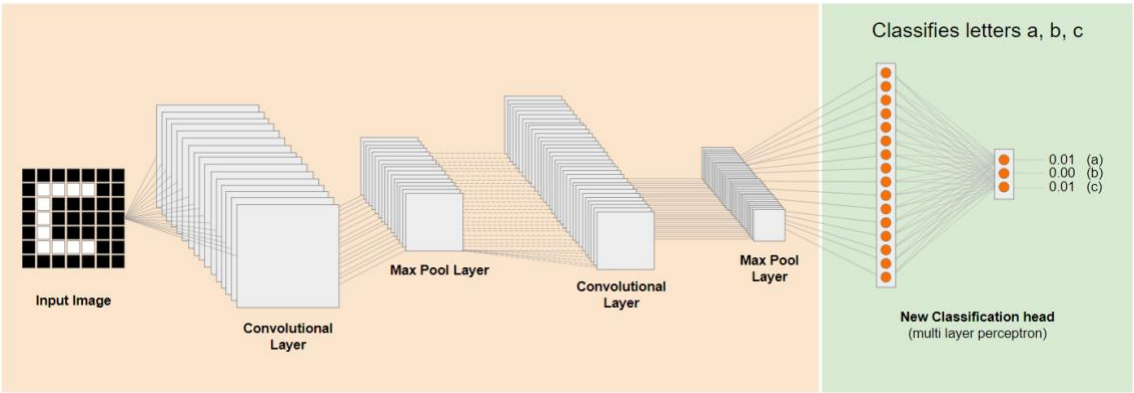


Рисунок 2.6 – Гіпотетична модель

Тут шари нижчого рівня заморожені та не навчаються, лише нова голова класифікації оновлюватиметься, щоб вивчати функції, надані з попередньо навченої нарізаної моделі ліворуч.

Акт виконання цього відомий як трансферне навчання, і це те, що Teachable Machine робить за лаштунками [4].

Можна бачити, що багатошаровий персептрон на самому кінці мережі тренується набагато швидше, ніж якби вам довелося навчати всю мережу з нуля.

2.2 Створення структури вебдодатку

Для початку потрібно створити базовий скелет вебдобатку, для цього потрібні такі файли:

– сторінка HTML (index.html);
– таблиця стилів (style.css);
– файл для написання нашого коду JavaScript (script.js).

Для зручності є імпорт у файл HTML для бібліотеки TensorFlow.js. Це виглядає так (рис. 2.7).

Усі прототипи вимагають базових платформ HTML, на яких можна відобразити свої висновки. Налаштуйте це зараз. Ви збираєтеся додати:

– заголовок для сторінки;
– трохи описового тексту;
– абзац статусу;
– відео для зберігання трансляції веб-камери після готовності;
– кілька кнопок для запуску камери, збору даних або скидання досвіду;
– імпорт файлів TensorFlow.js і JS, які ви будете кодувати пізніше.

У index.html потрібно вставити поверх наявного коду наступне, щоб налаштувати зазначені вище функції (рис. 2.8).

```
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs/dist/tf.min.js" type="text/
javascript">

</script>
```

Рисунок 2.7 – Імпорт бібліотеки TensorFlow.js.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Transfer Learning - TensorFlow.js</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Import the webpage's stylesheet -->
    <link rel="stylesheet" href="/style.css">
  </head>
  <body>
    <h1>Make your own "Teachable Machine" using Transfer Learning with MobileNet v3 in
    TensorFlow.js using saved graph model from TFHub.</h1>

    <p id="status">Awaiting TF.js load</p>

    <video id="webcam" autoplay muted></video>

    <button id="enableCam">Enable Webcam</button>
    <button class="dataCollector" data-1hot="0" data-name="Class 1">Gather Class 1
    Data</button>
    <button class="dataCollector" data-1hot="1" data-name="Class 2">Gather Class 2
    Data</button>
    <button id="train">Train &amp;amp; Predict!</button>
    <button id="reset">Reset</button>

    <!-- Import TensorFlow.js library -->
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@3.11.0/dist/tf.min.js"
    type="text/javascript"></script>

    <!-- Import the page's JavaScript to do some stuff -->
    <script type="module" src="/script.js"></script>
  </body>
</html>
```

Рисунок 2.8 – Налаштування проекту

Розібравши наведений вище HTML-код, можна виділити деякі ключові речі, які ви додали.

Додали тег <h1> для заголовка сторінки разом із тегом <p> з ідентифікатором «status», куди ви друкуватимете інформацію, оскільки ви використовуєте різні частини системи для перегляду результатів.

Додали елемент <video> з ідентифікатором «веб-камера», у який пізніше відобразите потік веб-камери.

Додали 5 елементів <button>. Перший з ідентифікатором «enableCam» вмикає камеру. Наступні дві кнопки мають клас «dataCollector», який дозволяє збирати приклади зображень для об’єктів, які ви хочете розпізнати. Код, який ви напишете пізніше, буде розроблено таким чином, що ви зможете додати будь-яку кількість цих кнопок, і вони працюватимуть за призначенням автоматично [5].

Зауваження, що ці кнопки також мають спеціальний визначений користувачем атрибут під назвою data-1hot з цілим значенням, починаючи з 0 для першого класу. Це числовий індекс, який ви використовуватимете для представлення даних певного класу. Індекс буде використовуватися для правильного кодування класів виводу за допомогою числового представлення замість рядка, оскільки моделі ML можуть працювати лише з числами.

Існує також атрибут data-name, який містить зрозумілу людині назву, яку ви хочете використовувати для цього класу, що дозволяє надати користувачеві більш значущу назву замість числового значення індексу з гарячого кодування 1.

Додали 2 імпорти <script>. Один для TensorFlow.js, а інший для script.js.

Додавши стилі для елементів HTML, щоб переконатися, що вони відображаються правильно. Ось кілька стилів, які додаються для правильного розташування та розміру елементів (рис. 2.9).

```css
body {
  font-family: helvetica, arial, sans-serif;
  margin: 2em;
}

h1 {
  font-style: italic;
  color: #FF6F00;
}


video {
  clear: both;
  display: block;
  margin: 10px;
  background: #000000;
  width: 640px;
  height: 480px;
}

button {
  padding: 10px;
  float: left;
  margin: 5px 3px 5px 10px;
}

.removed {
  display: none;
}

#status {
  font-size:150%;
}
```

Рисунок 2.9 – Стилі для розташування елементів

У створенному script.js файлі треба по-перше, додати кілька ключових констант, які ви використовуватимете в програмі. Спочатку треба замінити вміст script.js такими константами (рис. 2.10).

```
const STATUS = document.getElementById('status');
const VIDEO = document.getElementById('webcam');
const EN■ABLE_CAM_BUTTON = document.getElementById('enableCam');
const RESET_BUTTON = documen□t.getElementById('reset');
const TRAIN_BUTTON = document.getElementById('train');
const MOBILE_NET_INPUT_WIDTH = 224;
const MOBILE_NET_INPUT_HEIGHT = 224;
const STOP_DATA_GATHER = -1;
const CLASS_NAMES = [];
```

Рисунок 2.10 – Ключові константи

Ці константи потрібні для:

– STATUS просто містить посилання на тег абзацу, до якого ви будете записувати оновлення статусу;

– VIDEO містить посилання на відеоелемент HTML, який відображатиме канал веб-камери;

– ENABLE_CAM_BUTTON, RESET_BUTTON і TRAIN_BUTTON захоплюють посилання DOM на всі ключові кнопки зі сторінки HTML;

– MOBILE_NET_INPUT_WIDTH і MOBILE_NET_INPUT_HEIGHT визначають очікувану ширину введення та висоту моделі MobileNet відповідно. Зберігаючи це в константі у верхній частині файлу, як це, якщо ви вирішите використовувати іншу версію пізніше, це полегшить оновлення значень один раз замість того, щоб замінювати їх у багатьох різних місцях;

– для STOP_DATA_GATHER встановлено значення – 1. Це зберігає значення стану, щоб ви знали, коли користувач припинив натискати кнопку для збору даних із каналу веб-камери. Якщо дати цьому номеру більш значущу назву, код стане легшим для читання пізніше;

– CLASS_NAMES діє як пошук і містить зрозумілі людині імена для можливих передбачень класу. Цей масив буде заповнено пізніше.

Наступним етапом потрібно пов’язати з ними деяких слухачів подій (рис. 2.11).

```
ENABLE_CAM_BUTTON.addEventListener('click', enableCam);
TRAIN_BUTTON.addEventListener('click', trainAndPred□ict);
RESET_BUTTON.addEventListener('click', reset);
function enableCam() {
  // TOD□O: Fill this out later in the codelab!
}
function trainAndPredict() {
  // TODO: Fill this out later in the codelab!
}
function reset() {
  // TODO: Fill this out later in the codelab!
}
```

Рисунок 2.11 – Обов'язкові слухачі подій

ENABLE_CAM_BUTTON – викликає функцію enableCam при натисканні.

TRAIN_BUTTON – викликає trainAndPredict при натисканні.

RESET_BUTTON – виклики скидаються при натисканні.

За допомогою наступної функції можна знайти всі кнопки, які мають клас «dataCollector» за допомогою document.querySelectorAll(). Це повертає масив елементів, знайдених у документі, які відповідають [6] (рис. 2.12).

Потім потрібно перебрати знайдені кнопки та пов’язати 2 прослуховувачі подій з кожною. Один для «mousedown» і один для «mouseup». Це дає змогу продовжувати записувати зразки, поки натиснуто кнопку, що корисно для збору даних.

Обидві події викликають функцію gatherDataForClass, яку треба визначите пізніше.

```
let dataCollectorButtons = document.querySelectorAll('button.dataCollector');
for (let i = 0; i < dataCollectorButtons.length; i++) {
  dataCollectorButtons[i].addEventListener('mousedown', gatherDataForClass);
  dataCollectorButtons[i].addEventListener('mouseup', gatherDataForClass);

  CLASS_NAMES.push(dataCollectorButtons[i].getAttribute('data-name'));
}
function gatherDataForClass() {

}
```

Рисунок 2.12 – Функція для повернення масиву елементів

Наступним кроком потрібно додати кілька змінних для зберігання ключових речей, які використовуватимуться пізніше (рис. 2.13).

```
let mobilenet = undefined;
let gatherDataState = STOP_DATA_GATHER;
let videoPlaying = false;
let trainingDataInputs = [];
let trainingDataOutputs = [];
let examplesCount = [];
let predict = false;
```

Рисунок 2.13 – Збереження ключових змінних

По-перше, тепер є змінна mobilenet для зберігання завантаженої моделі mobilenet. Спочатку встановіть значення undefined.

Далі є змінна під назвою gatherDataState. Якщо натиснути кнопку «dataCollector», замість цього буде 1 гарячий ідентифікатор цієї кнопки, як визначено в HTML, щоб ви знали, який клас даних ви збираєте в цей момент. Спочатку це встановлено на STOP_DATA_GATHER, щоб цикл

збору даних, який ви записуєте пізніше, не збирав жодних даних, якщо не натиснуто жодної кнопки.

videoPlaying відстежує, чи потік веб-камери успішно завантажено та відтворено та чи доступний для використання. Спочатку встановлено значення false, оскільки веб-камера не вмикається, доки ви не натиснете кнопку ENABLE_CAM_BUTTON.

Далі треба визначити 2 масиви, trainingDataInputs і trainingDataOutputs. Вони зберігають зібрані значення навчальних даних, коли ви натискаєте кнопки «dataCollector» для вхідних функцій, згенерованих базовою моделлю MobileNet, і вихідного класу, що вибирається відповідно.

Потім визначається один останній масив examplesCount, щоб відстежувати, скільки прикладів міститься для кожного класу, коли ви починаєте їх додавати.

Нарешті, у додатку є змінна під назвою predict, яка керує циклом передбачення. Спочатку встановлено значення false. Жодні передбачення не можуть мати місце, доки пізніше не буде встановлено значення true.

Тепер, коли всі ключові змінні визначено, треба завантажити попередньо нарізану базову модель MobileNet v3, яка надає вектори ознак зображення замість класифікацій.

Потрібно визначити нову функцію під назвою loadMobileNetFeatureModel. Це має бути асинхронна функція, оскільки акт завантаження моделі є асинхронним (рис. 2.14).

```
/**
 * Loads the MobileNet model and warms it up so ready for use.
 **/
async function loadMobileNetFeatureModel() {
  const URL =
    'https://tfhub.dev/google/tfjs-model/imagenet/mobilenet_v3_small_100_224/
    feature_vector/5/default/1';

  mobilenet = await tf.loadGraphModel(URL, {fromTFHub: true});
  STATUS.innerText = 'MobileNet v3 loaded successfully!';

  // Warm up the model by passing zeros through it once.
  tf.tidy(function () {
    let answer = mobilenet.predict(tf.zeros([1, MOBILE_NET_INPUT_HEIGHT,
    MOBILE_NET_INPUT_WIDTH, 3]));
    console.log(answer.shape);
  });
}

// Call the function immediately to start loading.
loadMobileNetFeatureModel();
```

Рисунок 2.14 – Завантаження моделі

У цьому коді визначається URL-адресу, де знаходиться модель для завантаження з документації TFHub.

Потім можна завантажити модель за допомогою await tf.loadGraphModel(), не забувши встановити для спеціальної властивості fromTFHub значення true під час завантаження моделі з цього веб-сайту Google. Це особливий випадок лише для використання моделей, розміщених на TF Hub, де потрібно встановити цю додаткову властивість.

Після завершення завантаження ви можете встановити внутрішній текст елемента STATUS із повідомленням, щоб ви могли візуально побачити, що він завантажився правильно, і ви готові розпочати збір даних [6].

Тепер залишилося лише розігріти модель. З такими великими моделями, як ця, під час першого використання моделі може знадобитися деякий час, щоб усе налаштувати. Тому це допомагає передати нулі через

модель, щоб уникнути очікування в майбутньому, коли час може бути більш критичним.

Можна використовувати tf.zeros(), загорнений у tf.tidy(), щоб забезпечити правильне розміщення тензорів із розміром пакета 1 і правильною висотою та шириною, які ви визначили у своїх константах на початку. Нарешті, ви також вказуєте колірні канали, які в даному випадку дорівнюють 3, оскільки модель очікує зображення RGB.

Далі треба зареєструвати отриману форму тензора, отриману за допомогою answer.shape(), щоб допомогти вам зрозуміти розмір характеристик зображення, створених цією моделлю.

Після визначення цієї функції можна негайно викликати її, щоб ініціювати завантаження моделі під час завантаження сторінки.

Якщо зараз переглянути попередній перегляд у реальному часі, через кілька секунд можна побачити, що текст статусу зміниться з «Очікується завантаження TF.js» на «MobileNet v3 завантажено успішно!» як показано нижче (рис. 2.15).

Рисунок 2.15 – Статус про успішну роботу

Тепер настав час визначити модель голови, яка, по суті, є дуже мінімальним багатошаровим перцептроном (рис. 2.16).

```
let model = tf.sequential();
model.add(tf.layers.dense({inputShape: [1024], units: 128, activation: 'relu'}));
model.add(tf.layers.dense({units: CLASS_NAMES.length, activation: 'softmax'}));

model.summary();

// Compile the model with the defined optimizer and specify a loss function to use.
model.compile({

  optimizer: 'adam',
  // Use the correct loss function. If 2 classes of data, must use binaryCrossentropy.
  // Else categoricalCrossentropy is used if more than 2 classes.
  loss: (CLASS_NAMES.length === 2) ? 'binaryCrossentropy': 'categoricalCrossentropy',
  // As this is a classification problem you can record accuracy in the logs too!
  metrics: ['accuracy']
});
```

Рисунок 2.16 – Визначення мінімального багатошарового перцептрона

Починається з визначення моделі tf.sequential, до якої можна буде додавати шари моделі.

Далі додавши щільний шар як вхідний шар до цієї моделі. Це має вхідну форму 1024, оскільки виходи з функцій MobileNet v3 мають такий розмір. Виявивши це на попередньому кроці після проходження одиниць через модель. Цей рівень містить 128 нейронів, які використовують функцію активації ReLU.

Наступний шар, який потрібно додати, це вихідний шар. Кількість нейронів має дорівнювати кількості класів, які ви намагаєтесь передбачити. Для цього можна використовувати CLASS_NAMES.length, щоб знайти, скільки класів ви плануєте класифікувати, що дорівнює кількості кнопок збору даних, знайдених в інтерфейсі користувача. Оскільки це проблема класифікації, ви використовуєте активацію softmax на цьому вихідному

рівні, яку потрібно використовувати під час спроби створити модель для вирішення проблем класифікації замість регресії.

Тепер потрібно надрукувати model.summary(), щоб вивести на консоль огляд щойно визначеної моделі.

Нарешті, треба скомпілювати модель, щоб вона була готова до навчання. Тут оптимізатор налаштований на adam, і втрата буде або binaryCrossentropy, якщо CLASS_NAMES.length дорівнює 2, або він використовуватиме categoricalCrossentropy, якщо є 3 або більше класів для класифікації. Також запитуються показники точності, щоб пізніше їх можна було відслідковувати в журналах для налагодження.

У консолі повинно бути щось на зразок цього (рис. 2.17).

```
_________________________________________________________________
Layer (type)                 Output shape              Param #
=================================================================
dense_Dense1 (Dense)         [null,128]                131200
_________________________________________________________________
dense_Dense2 (Dense)         [null,2]                  258
=================================================================
Total params: 131458
Trainable params: 131458
Non-trainable params: 0
```

Рисунок 2.17 –Вивід в консоль

Настав час конкретизувати функцію enableCam(), яка були визначена раніше. Додавши нову функцію під назвою hasGetUserMedia(), як показано нижче, а потім треба замінити вміст попередньо визначеної функції enableCam() на відповідний код нижче (рис. 2.18).

```
function hasGetUserMedia() {
  return !!(navigator.mediaDevices && navigator.mediaDevices.getUserMedia);
}

function enableCam() {
  if (hasGetUserMedia()) {
    // getUsermedia parameters.
    const constraints = {
      video: true,
      width: 640,
      height: 480
    };

    // Activate the webcam stream.
    navigator.mediaDevices.getUserMedia(constraints).then(function(stream) {
      VIDEO.srcObject = stream;
      VIDEO.addEventListener('loadeddata', function() {
        videoPlaying = true;
        ENABLE_CAM_BUTTON.classList.add('removed');
      });
    });
  } else {
    console.warn('getUserMedia() is not supported by your browser');
  }
}
```

Рисунок 2.18 – Конкретизуваня функції enableCam()

Тепер запустивши код, та натиснувши кнопку увімкнути камеру та дозволити доступ до веб-камери.

Тепер настав час заповнити наразі порожню функцію під назвою gatherDataForClass() (рис. 2.19).

```
/**
 * Handle Data Gather for button mouseup/mousedown.
 **/
function gatherDataForClass() {
  let classNumber = parseInt(this.getAttribute('data-1hot'));
  gatherDataState = (gatherDataState === STOP_DATA_GATHER) ? classNumber :
  STOP_DATA_GATHER;
  dataGatherLoop();
}
```

Рисунок 2.19 – Функція gatherDataForClass()

Тепер треба визначити функцію dataGatherLoop(). Ця функція відповідає за вибірку зображень із відео з веб-камери, пропускання їх через модель MobileNet і захоплення вихідних даних цієї моделі (1024 вектора ознак).

Потім він зберігає їх разом із ідентифікатором gatherDataState кнопки, яка зараз натиснута, щоб ви знали, який клас представляють ці дані (рис. 2.20).

```
function dataGatherLoop() {
  if (videoPlaying && gatherDataState !== STOP_DATA_GATHER) {
    let imageFeatures = tf.tidy(function() {
      let videoFrameAsTensor = tf.browser.fromPixels(VIDEO);
      let resizedTensorFrame = tf.image.resizeBilinear(videoFrameAsTensor,
      [MOBILE_NET_INPUT_HEIGHT,
          MOBILE_NET_INPUT_WIDTH], true);
      let normalizedTensorFrame = resizedTensorFrame.div(255);
      return mobilenet.predict(normalizedTensorFrame.expandDims()).squeeze();
    });

    trainingDataInputs.push(imageFeatures);
    trainingDataOutputs.push(gatherDataState);

    // Intialize array index element if currently undefined.
    if (examplesCount[gatherDataState] === undefined) {
      examplesCount[gatherDataState] = 0;
    }
    examplesCount[gatherDataState]++;

    STATUS.innerText = '';
    for (let n = 0; n < CLASS_NAMES.length; n++) {
      STATUS.innerText += CLASS_NAMES[n] + ' data count: ' + examplesCount[n] + '. ';
    }
    window.requestAnimationFrame(dataGatherLoop);
  }
}
```

Рисунок 2.20 – Функція dataGatherLoop()

Якщо запустити код зараз, можна натиснути кнопку «Увімкнути камеру», дочекатися завантаження веб-камери, а потім натиснути й

утримувати кожну кнопку збору даних, щоб зібрати приклади для кожного класу даних [2].

Тепер визначивши функцію dataGatherLoop(). Ця функція відповідає за вибірку зображень із відео з веб-камери, пропускання їх через модель MobileNet і захоплення вихідних даних цієї моделі (1024 вектора ознак).

Потім він зберігає їх разом із ідентифікатором gatherDataState кнопки, яка зараз натиснута, щоб ви знали, який клас представляють ці дані (рис. 2.21).

```
function dataGatherLoop() {
  if (videoPlaying && gatherDataState !== STOP_DATA_GATHER) {
    let imageFeatures = tf.tidy(function() {
      let videoFrameAsTensor = tf.browser.fromPixels(VIDEO);
      let resizedTensorFrame = tf.image.resizeBilinear(videoFrameAsTensor,
      [MOBILE_NET_INPUT_HEIGHT,
          MOBILE_NET_INPUT_WIDTH], true);
      let normalizedTensorFrame = resizedTensorFrame.div(255);
      return mobilenet.predict(normalizedTensorFrame.expandDims()).squeeze();
    });

    trainingDataInputs.push(imageFeatures);
    trainingDataOutputs.push(gatherDataState);

    // Intialize array index element if currently undefined.
    if (examplesCount[gatherDataState] === undefined) {
      examplesCount[gatherDataState] = 0;
    }
    examplesCount[gatherDataState]++;

    STATUS.innerText = '';
    for (let n = 0; n < CLASS_NAMES.length; n++) {
      STATUS.innerText += CLASS_NAMES[n] + ' data count: ' + examplesCount[n] + '. ';
    }
    window.requestAnimationFrame(dataGatherLoop);
  }
}
```

Рисунок 2.21 – Функція dataGatherLoop()

Наступним кроком є впровадження коду для наразі порожньої функції trainAndPredict(), де відбувається навчання передачі (рис. 2.22).

```
async function trainAndPredict() {
  predict = false;
  tf.util.shuffleCombo(trainingDataInputs, trainingDataOutputs);
  let outputsAsTensor = tf.tensor1d(trainingDataOutputs, 'int32');
  let oneHotOutputs = tf.oneHot(outputsAsTensor, CLASS_NAMES.length);
  let inputsAsTensor = tf.stack(trainingDataInputs);

  let results = await model.fit(inputsAsTensor, oneHotOutputs, {shuffle: true,
  batchSize: 5, epochs: 10,
      callbacks: {onEpochEnd: logProgress} });

  outputsAsTensor.dispose();
  oneHotOutputs.dispose();
  inputsAsTensor.dispose();
  predict = true;
  predictLoop();
}

function logProgress(epoch, logs) {
  console.log('Data for epoch ' + epoch, logs);
}
```

Рисунок 2.22 – Функція навчання передачі

Тут реалізується основний цикл прогнозування, який відбирає кадри з веб-камери та постійно передбачає, що міститься в кожному кадрі, з результатами в реальному часі в браузері [3] (рис. 2.23).

Останнім кроком є впровадження кнопки скидання, щоб почати все спочатку. Нижче наведено код наразі порожньої функції reset(). Його треба оновити наступним чином (рис. 2.24).

Таким чином, аналіз методів інструментів тестування дає смогу сформувати необхідні елементи вебдодатку та створити його для подальшого використання в якості тестового стенду при вивченні можливостей того чи іншого інструменту автоматизованого тестування.

```
function predictLoop() {
  if (predict) {
    tf.tidy(function() {
      let videoFrameAsTenso r = tf.browser.fromPixels(VIDEO).div(255);
      let resizedTensorFrame =  tf.image.resizeBilinear(videoFrameAsTensor,
      [MOBILE_NET_INPUT_HEIGHT,
          MOBILE_NET_INPUT_WIDTH], true);

      let imageFeatures = mobilenet.predict(resizedTensorFrame.expandDims());
      let prediction = model.predict(imageFeatures).squeeze();
      let highestIndex = prediction.argMax().arraySync();
      let predictionArray = prediction.arraySync();

      STATUS.innerText = 'Prediction: ' + CLASS_NAMES[highestIndex] + ' with ' + Math.
      floor(predictionArray[highestIndex] * 100) + '% confidence';
    });

    window.requestAnimationFrame(predictLoop);
  }
}
```

Рисунок 2.23 – Основний цикл прогнозування

```
/**
 * Purge data and start over. Note this does not dispose of the loaded
 * MobileNet model and MLP head tensors as you will need to reuse
 * them to train a new m odel.
 **/
function reset() {
  predict = false;
  examplesCount.length = 0;
  for (let i = 0; i < trainingDataInputs.length; i++) {
    trainingDataInputs[i].dispose();
  }
  trainingDataInputs.length = 0;
  trainingDataOutputs.length = 0;
  STATUS.innerText = 'No data collected';

  console.log('Tensors in memory: ' + tf.memory().numTensors);
}
```

Рисунок 2.24 – Впровадження кнопки скидання

Таким чином тепер за допомогою TensorFlow.js є змога додавати власні моделі машинного навчання та потім використовувати їх у додатку для розпізнавання знаків дорожнього руху.

2.3 Висновки до 2 розділу

Розробивши додаток для навчання tensorflow моделі, тепер маємо змогу зробити свою унікальну навчену модель, яку можливо використовувати у JavaScript середовищі. Сама бібліотека tensorflow пропонує деякі готові моделі, але вони вміють відрізнити різні об’єкти, такі як людей, велосипеди, машини, різні меблі. Також існує модель яка розпізнає емоції. Тобто для виконання поставленої мети обов’язковим етапом було створення такого програмного забезпечення.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РОЗПІЗНАВАННЯ ЗНАКІВ ДОРОЖНЬОГО РУХУ

3.1 Основні поняття для створення React додатка

Цей синтаксис тегів не є ні рядком, ні HTML (рис. 3.1). Він має назву JSX, і це розширення синтаксису для JavaScript. Його потрібно використовувати в React, щоб описати, як повинен виглядати інтерфейс користувача. JSX може нагадувати мову шаблонів, але з усіма перевагами JavaScript. JSX створює “React-елементи”.

Рисунок 3.1 – Приклад синтаксису JSX

React використовує той факт, що логіка виводу пов’язана з іншою логікою інтерфейсу користувача: як обробляються події, як змінюється стан з часом і як дані готуються для рендерингу.

Замість того, щоб штучно відокремити технології, розмістивши розмітку і логіку в окремих файлах, React розділяє відповідальність між вільно зв’язаними одиницями, що містять обидві технології і називаються “компонентами”.

React не вимагає використання JSX, але більшість людей цінують його за візуальну допомогу при роботі з інтерфейсом користувача в коді JavaScript.

У прикладі нижче оголошується змінна name, а потім використовуємо її в JSX, обертаючи її у фігурні дужки (рис. 3.2).

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Рисунок 3.2 – Приклад створення змінної в React

React дозволяє помістити будь-який валідний JavaScript вираз всередину фігурних дужок у JSX. Наприклад, 2 + 2, user.firstName, або formatName(user) є валідними виразами JavaScript.

У наведеному нижче прикладі вставляється результат виклику функції JavaScript, formatName(user) в елемент <h1> (рис. 3.3).

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Рисунок 3.3 – Приклад виклику фунцкії в React

Після компіляції вирази JSX стають звичайними викликами функцій JavaScript, що повертають об’єкти JavaScript.

Це означає, що можна використовувати JSX всередині if виразів і циклів for, присвоювати його змінним, приймати його як аргументи і повертати його з функцій.

Можна використовувати лапки для задання рядкових літералів як атрибутів або використовувати фігурні дужки, щоб вставити вираз JavaScript у атрибут.

Якщо тег порожній, його можна одразу закрити за допомогою />, як XML (рис. 3.4).

```jsx
const element = <img src={user.avatarUrl} />;
```

Рисунок 3.4 – JSX змінні

Теги JSX можуть мати дочірні елементи (рис. 3.5).

```jsx
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

Рисунок 3.4 – JSX дочірні елементи

За замовчуванням, React DOM екранує будь-які значення, що включені в JSX, перед їх рендерингом. Таким чином, це гарантує, що користувач ніколи не включить в код те, що явно не написано у додатку. Перед виводом все перетворюється на рядок. Це допомагає запобігти атакам XSS (cross-site-scripting).

JSX – це об’єкти по суті. Babel компілює JSX до викликів React.createElement(). Ці два приклади ідентичні між собою (рис. 3.5).

```jsx
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);

const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Рисунок 3.5 – Приклад JSX елементів

Ці об’єкти називаються “React-елементами”. можна вважати їх описами того, що хочете бачити на екрані. React читає ці об’єкти і використовує їх для побудови DOM і підтримки його в актуальному стані.

Для початку потрібно створити JSX елемент з айді – id="root". Цей елемент називається “кореневим” DOM вузлом, тому що все всередині нього буде керуватись за допомогою React DOM.

Додатки, створені за допомогою самого лише React, зазвичай мають лише один кореневий вузол DOM. Якщо інтегрується React в існуючий додаток – можна мати будь-яку кількість ізольованих кореневих DOM вузлів.

Для рендерингу React-елементу в кореневому DOM вузлі, потрібно викликати функцію ReactDOM.render() з React-елементом та кореневим DOM вузлом у якості аргументів (рис. 3.6).

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Рисунок 3.6 – Рендеринг React-елементу

React-елементи є незмінними. Як тільки елемент створений, не можна змінювати його дочірні елементи чи атрибути. Елемент схожий на кадр із фільму: він відображає інтерфейс користувача в певний момент часу.

Через це, єдиний спосіб оновити інтерфейс користувача – створити новий елемент і передати його в ReactDOM.render().

React оновлює лише те, що необхідно, він порівнює елемент і його дочірні елементи з попередніми та вносить в DOM тільки необхідні зміни для приведення DOM у бажаний стан.

Компоненти невід’ємні елементи будь-якого React додатку, вони дозволяють розділити інтерфейс користувача на незалежні частини, придатні до повторного використання, і сприймати їх як такі, що функціонують окремо один від одного.

Концептуально компоненти є подібними до функцій JavaScript. Вони приймають довільні вхідні дані (так звані “пропси”) і повертають React-елементи, що описують те, що повинно з’явитися на екрані.

Існують два види, функціональні та класові компоненти. Найпростішим способом визначення компонента є написання функції JavaScript (рис. 3.7).

Рисунок 3.7 – Функціональний компонент

Ця функція є валідним React-компонентом, оскільки вона приймає єдиний аргумент “пропс” (скорочено від *properties* – властивості), який є об’єктом з даними і повертає React-елемент. Такі компоненти ми називаємо “функціональними компонентами”, оскільки вони буквально є JavaScript функціями.

Компоненти можуть посилатися на інші компоненти під час виведення. Це дозволяє використовувати одну і ту ж абстракцію компонентів для будь-якого рівня деталізації. Кнопка, форма, діалогове вікно, екран: у React-додатках всі вони зазвичай виражаються як компоненти.

Як правило, нові React-додатки мають єдиний компонент App, що знаходиться зверху дерева ієрархій елементів. Однак, якщо потрібно інтегрувати React у існуючий додаток, можна почати знизу вгору з невеликим компонентом, наприклад Button, і поступово працювати у верхній частині ієрархії перегляду.

Пропси можна тільки читати. Незалежно від того як оголошен компонент як функція чи клас, він ніколи не повинен змінювати свої власні пропси. Наприклад (рис. 3.8).

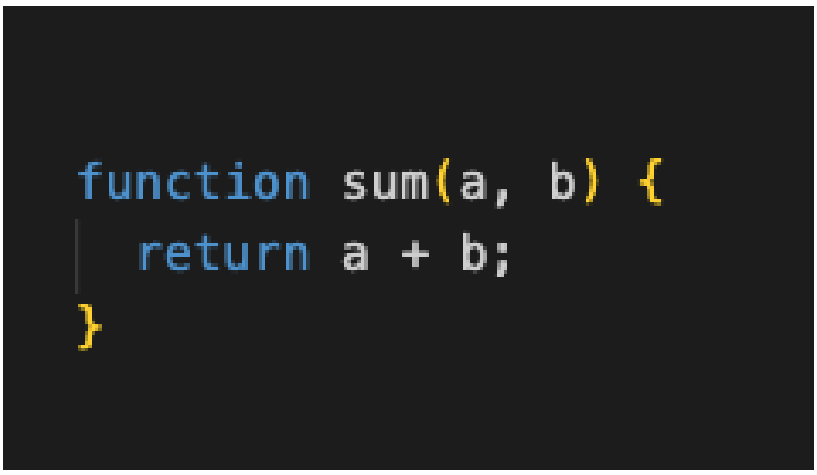


Рисунок 3.8 – Приклад пропсів

Такі функції називаються “чистими”, оскільки вони не намагаються змінити свої аргументи і завжди повертають один і той же результат для тих же аргументів.

Кожен компонент має декілька “методів життєвого циклу”, які ви можете перевизначати, щоб запускати код в певний момент часу. Ці методи викликаються в наступному порядку, коли екземпляр компонента створюється і вставляється в DOM:

– constructor();
– static getDerivedStateFromProps();
– render();
– componentDidMount().

Оновлення може бути спричиненим зміною пропсів чи стану. Ці методи викликаються в наступному порядку, коли компонент повторно рендериться:

– static getDerivedStateFromProps();
– shouldComponentUpdate();
– render();
– getSnapshotBeforeUpdate();
– componentDidUpdate().

Метод render() – єдиний необхідний метод в класових компонентах. Під час виклику він перевіряє this.props та this.state і повертає один з наступних типів:

– react-елементи. Зазвичай створені за допомогою JSX. Наприклад, і <MyComponent /> є React-елементами, які інструктують React відрендерити вузол DOM або інший компонент визначений користувачем, відповідно;

– масиви та фрагменти. Дозволяють повернути декілька елементів під час рендерингу. Перегляньте документацію для фрагментів, щоб дізнатися більше;

– портали. Дозволють рендерити дочірні елементи в іншому піддереві DOM. Перегляньте документацію для порталів, щоб дізнатися більше;

– рядки і числа. Будуть відрендерені як текстові вузли в DOM;

– логічні значення чи null. Не рендерять нічого. (Існують, здебільшого, для підтримки шаблону return test && <Child />, де test – логічне значення).

Функція render() має бути чистою, а це означає, що вона не змінює стан компонента, повертає однаковий результат при кожному виклику і не взаємодіє з браузером напряму.

Якщо потрібно взаємодіяти з браузером, потрібно виконувати необхідні дії в componentDidMount() чи інших методах життєвого циклу. Збереження render() чистим, робить компонент легшим для розуміння.

Конструктор для React-компонента викликається до того, як він буде примонтований. При реалізації конструктора для підкласу React.Component, ви маєте викликати super(props) перед будь-яким іншим виразом. У іншому випадку, this.props буде невизначеним в конструкторі, що може призвести до помилок.

Як правило, у React конструктори використовуються лише для двох цілей:

– ініціалізація локального стану шляхом присвоєння об’єкта this.state;

– прив’язка методів обробника подій до екземпляру.

Не варто викликати setState() у constructor(). Натомість, якщо компонент потребує використання локального стану, присвоюйте початкове значення this.state безпосередньо в конструкторі (рис. 3.9).

Рисунок 3.9 – setState()

Конструктор – це єдине місце, де потрібно присвоювати this.state напряму. У всіх інших методах для цього слід використовувати this.setState().

Потрібно уникати додавання будь-яких побічних ефектів чи підписок у конструкторі. Для таких випадків використовується componentDidMount().

componentDidMount() викликається відразу після монтування компонента (вставки в DOM-дерево). Ініціалізація, яка потребує DOM-вузли, має знаходитись тут. Якщо потрібно завантажити дані з віддаленого сервера, це гарне місце для створення мережевого запиту.

Також цей метод є вдалим місцем для налаштування підписок. Якщо це зробити, то потрібно не забути відписатись в componentWillUnmount().

Мжона негайно викликати setState() в componentDidMount(). Це запустить додатковий рендер, але це станеться до того, як браузер оновить екран. Це гарантує те, що навіть якщо render() в цьому випадку буде

викликаний двічі, користувач не побачить проміжного стану. Потрібно обережно використовувати цей підхід, тому що він часто приводить до проблем з продуктивністю. У більшості випадків, замість цього у вас має бути можливість присвоїти початковий стан у constructor(). Однак, це може бути необхідно для таких випадків як модальні вікна і спливаючі підказки, коли вам потрібно відрендерити щось, що залежить від розмірів та позиції вузла DOM.

componentDidUpdate() викликається відразу після оновлення. Цей метод не викликається під час першого рендеру.

Використовувати це як можливість працювати з DOM при оновленні компонента. Також це хороше місце для мережевих запитів, якщо ви порівнюєте поточні пропси з попередніми (наприклад, мережевий запит може бути не потрібним, якщо проп не змінився).

Також можна відразу викликати setState() у componentDidUpdate(), але потрібно звернути увагу, що цей виклик має бути обгорнутий в умову, інакше можна спричинити безкінечний цикл. Крім того, це спричинить додатковий повторний рендер який, хоч і не буде видимий користувачу, може вплинути на продуктивність компонента. Якщо потрібно “дзеркально відобразити” певний стан в пропі, що приходять зверху, розгляньте безпосереднє використання пропу.

Якщо компонент реалізує метод життєвого циклу getSnapshotBeforeUpdate() (що трапляється доволі рідко), значення, яке він повертає, буде передане третім “snapshot” параметром в componentDidUpdate(). В іншому випадку цей параметр буде невизначеним.

componentWillUnmount() викликається безпосередньо перед тим як компонент буде демонтовано і знищено. Виконувати будь-яку необхідну очистку в цьому методі, таку як скасування таймерів, мережевих запитів чи підписок створених у componentDidMount().

Не потрібно викликати setState() у componentWillUnmount(), тому що компонент не буде повторно рендеритись. Як тільки екземпляр компонента буде демонтований, він ніколи не буде примонтованим знову.

Використовувати shouldComponentUpdate(), щоб дати знати React, чи поточна зміна стану і пропсів не впливає на виведення компонента. Поведінка за замовчуванням полягає в повторному рендері при кожній зміні стану і в переважній більшості випадків ви маєте покладатись на поведінку за замовчуванням.

shouldComponentUpdate() викликається перед рендерингом при отриманні нових пропсів і стану. За замовчуванням має значення true. Цей метод не викликається при першому рендері чи коли використовується forceUpdate().

Цей метод існує лише в якості оптимізації продуктивності. Не треба покладатись на нього, щоб “запобігти” рендерингу, оскільки це може привести до помилок. Можна розглянути можливість використання вбудованого PureComponent замість написання власного shouldComponentUpdate(). PureComponent виконує поверхове порівняння пропсів та стану і зменшує шанс того, що буде пропущено необхідне оновлення.

Якщо немає впевненості, що потрібно реалізувати його власноруч, можна порівняти this.props із nextProps та this.state із nextState, і повернути false, щоб сказати React, що це оновлення можна пропустити. Але повернення false не запобігає повторному рендерингу дочірніх компонентів, коли їх стан змінюється.

React не рекомендує робити глибокі порівняння або використовувати JSON.stringify() у shouldComponentUpdate(). Це надзвичайно неефективно і негативно вплине на продуктивність.

Наразі, якщо shouldComponentUpdate() повертає false, тоді UNSAFE_componentWillUpdate(), render(), і componentDidUpdate() не будуть викликані. У майбутньому React може розглядати shouldComponentUpdate()

як пораду, а не строгу вимогу, і повернення false може спричинити повторний рендеринг компоненту, як зазвичай.

getDerivedStateFromProps викликається безпосередньо перед викликом методу render, як при першому рендерингу, так і при всіх наступних оновленнях. Він має повернути об’єкт для оновлення стану або null, щоб не оновлювати нічого.

Цей метод існує для малопоширених випадків коли стан залежить від змін в пропсах з часом. Наприклад, він може бути корисним для реалізації компоненту <Transition> котрий порівнює свої попередні і наступні дочірні елементи, щоб вирішити котрі з них потрібно анімувати для появи і зникнення.

getSnapshotBeforeUpdate() викликається безпосередньо перед тим, як останній відрендерений вивід буде зафіксовано, наприклад в DOM. Він дозволяє вашому компоненту захопити деяку інформацію з DOM (наприклад, позицію прокрутки) перед її можливою зміною. Будь-яке значення повернуте цим методом життєвого циклу, буде передане як параметр в componentDidUpdate().

Цей випадок не поширений, але він може бути в інтерфейсах користувача, таких як ланцюжок повідомлень в чаті, який має оброблювати позицію прокрутки особливим чином.

Запобіжники – це React-компоненти, котрі перехоплюють помилки JavaScript будь-де в їхньому дереві дочірніх компонентів, логують їх і відображають резервний інтерфейс користувача, замість невалідного дерева компонентів. Запобіжники перехоплюють помилки протягом рендерингу, в методах життєвого циклу і конструкторах всього дерева під ними.

Класовий компонент стає запобіжником, якщо він визначає один (або обидва) з методів життєвого циклу – static getDerivedStateFromError() чи componentDidCatch(). Оновлення стану з цих методів дозволить вам перехопити необроблену помилку JavaScript у дереві нижче і відобразити резервний інтерфейс користувача.

Використовувати запобіжники тільки для відновлення від несподіваних виключних ситуацій; не потрібно використовувати їх для управління потоком.

setState() ставить в чергу оновлення стану компонента і повідомляє React, що цей компонент і його нащадки мають бути повторно відрендерені з оновленим станом. Це основний метод, який використовується для оновлення інтерфейсу користувача у відповідь на обробники подій і відповіді сервера.

setState() це як запит, а не як команду, що має бути негайно виконана для оновлення компонента. Для кращої наочної продуктивності, React може відкласти виклик і тоді оновити кілька компонентів за один прохід. React не гарантує негайного застосування змін стану.

setState() не завжди відразу оновлює компонент. Цей метод може групувати чи відкладати оновлення на потім. Це робить зчитування this.state відразу після виклику setState() потенційною пасткою. Натомість, використовуеться componentDidUpdate чи функцію зворотнього виклику setState (setState(updater, callback)), обидва підходи гарантовано запустяться після застосування оновлення. Якщо потрібно оновити стан на основі поперенього стану.

setState() завжди призводить до повторного рендерингу за умови, що shouldComponentUpdate() не повертає false. Якщо використовуються змінні об’єкти і логіка умовного рендерингу не може бути реалізована у shouldComponentUpdate(), виклик setState() тільки тоді, коли новий стан відрізняється від попереднього, може запобігти непотрібних повторних рендерингів.

state – це посилання на стан компонента в момент часу, коли зміна застосовується. Воно не має змінюватись напряму. Замість цього, зміни повинні бути представлені шляхом створення нового об’єкту на основі вхідних даних із state та props.

Як state, так і props, отримані функцією оновлення, гарантовано будуть в актуальному стані. Результат функції оновлення буде поверхово об’єднаний із state.

Другий параметр setState() – це необов’язкова функція зворотнього виклику, яка буде виконана після того, як setState завершив роботу і компонент повторно відрендерився. Зазвичай ми радимо використовувати componentDidUpdate() для подібної логіки.

Стан містить дані, конкретні для цього компонента і які можуть змінюватися з часом. Стан визначається користувачем і має бути простим JavaScript-об’єктом.

Якщо певне значення не використовується для рендерингу чи потоку даних (наприклад, ідентифікатор таймера), вам не потрібно вставляти його в стан. Такі значення можуть бути визначені як поля екземпляру компонента.

Обробка подій для React-елементів дуже схожа до обробки подій для DOM-елементів. Але є деякі синтаксичні відмінності. Події React іменуються в camelCase, а не в нижньому регістрі. З JSX передається функція як обробник події замість рядка.

Інша відмінність полягає в тому, що не можна повернути false для того, щоб запобігти поведінці за замовчуванням у React. Потрібно явно викликати preventDefault. Наприклад, для звичайного HTML, щоб запобігти поведінці посилання за замовчуванням (відкриття нової сторінки).

Зазвичай, коли використовується React, не потрібно викликати addEventListener, щоб додати обробник до DOM-елементу після його створення. Натомість, просто треба вказати обробник, коли елемент вперше відрендерився.

Коли визначається компонент як клас ES6, поширеною практикою є оголошення обробника подій як методу цього класу.

Потрібно бути обережним із значенням this у JSX функціях зворотнього виклику. У JavaScript класові методи за замовчуванням не

зв’язані. Якщо забути зв’язати this.handleClick і передати її до onClick, this буде undefined під час виклику функції.

Така поведінка не є особливою лише для React – вона є частиною того, як функції працюють в JavaScript. В загальному випадку, якщо посилатися на метод без () після нього, як наприклад onClick={this.handleClick}, потрібно зв’язати цей метод.

Якщо виклики bind немає змоги використовувати – є два шляхи обійти їх. Якщо використовується експериментальний синтаксис відкритих полей класу, то можно використовувати поля класу, щоб правильно прив’язувати функції зворотнього виклику.

Всередині циклу доволі часто потрібно передати додатковий параметр до обробника події. Наприклад, якщо id це ID рядка, можна застосувати один з нижченаведених прикладів (рис. 3.10).

```
<button onClick={(e) => this.deleteRow(id, e)}>Видалити рядок</button>
<button onClick={this.deleteRow.bind(this, id)}>Видалити рядок</button>
```

Рисунок 3.10 – Виклик onClick

Обидва приклади є еквівалентними і використовують стрілкові функції та Function.prototype.bind відповідно.

В обох випадках аргумент e, який відповідає події React, буде переданий другим аргументом після ID. Для стрілкової функції потрібно зробити передачу явною, але для bind будь-які наступні аргументи будуть передані автоматично.

Умовний рендеринг у React працює так само, як і умовні вирази працюють в JavaScript. Іноді потрібно пояснити React, як стан впливає на те, які компоненти треба сховати, а які – відрендерити, та як саме. Для цього використовуйте умовний оператор JavaScript, або вирази подібні до if.

Можно використовувати змінні для того, щоб зберігати елементи React. Це допоможе умовно рендерити лише частину компонента, в той час, як інша частина компонента залишається незмінною.

Незважаючи на те, що оголошення змінної та використання if-виразів для умовного рендерингу є гарним варіантом, але іноді хочеться коротший синтаксис. Для цього існують декілька інших способів для написання умов прямо в JSX.

Можно вставляти будь-який вираз у JSX, охопивши його фігурними дужками. Це правило поширюється і на логічний оператор && JavaScript, яким можна зручно вставити елемент в залежності від умови.

Іншим методом для умовного рендерингу елементів є використання тернарного оператора condition ? true : false.

Зрідка може виникнути потреба в тому, щоб дозволити компоненту сховати себе, навіть якщо він вже був відрендереним іншим компонентом. Для цього поверніть null замість того, що зазвичай йде на рендеринг. Повернення null із методу render ніяк не впливає на спрацювання методів життєвого циклу компонента.

3.2 Створення React середовища розробки вебдодатку

React – це JavaScript-бібліотека для створення інтерфейсів користувача. З самого початку React був спроектований так, щоб його можна було впроваджувати поступово. Тобто можно додавати так мало або так багато React-у, як потрібно.

Create React App – це найкращий шлях щоб почати будувати нові односторінкові додатки за допомогою React. Він встановлює осередок для розробки таким чином, щоб використовувати найновіші можливості JavaScript, робить розробку комфортнішою, а також оптимізує додаток для продакшну. Для цього знадобиться Node версії >= 8.10 та npm версії >= 5.6 на комп’ютері. Для створення проекту потрібно виконати (рис. 3.11).

```
npx create-react-app my-app
cd my-app
npm start
```

Рисунок 3.11 – Створення React проекту

Create React App не опрацьовує бекенд логіку чи логіку баз данних, а лише надає команди для побудови фронтенду, тому можно використовувати його з будь-яким бекендом. Під капотом він використовує Babel та webpack, але для роботи не треба нічого знати про них.

Коли додаток буде готовий для розгортання на продакшн, запустіть команду npm run build, це створить оптимізовану версію додатку у папці build.

Після створення проекту, спочатку потрібно ще створити папку "model" в корневії папці public, та додати в неї три файли вже згенерованої моделі дорожніх знаків (рис. 3.12). Ці файли будуть використовуватися для визначення знаків за допомогою вебкамери.

Для читання моделі буде використовуватися бібліотека ml5.js. Це нова бібліотека яка була створена і працює за схожим принципом як і Tensorflow, для зчитування об’єктів може використовуватися не лише процессор, а й відоекарти пристрою, що робить процесс значно швидшим.

Буде використовувати метод imageClassifier для передачі файлу моделі. Цей виклик методу повертає об’єкт класифікатора, який буде використовуватися для класифікації живих зображень через деякий час.

Також, після успішного завантаження моделі буде ініціалізуватися пристрій веб-камери, щоб була змога збирати зображення з прямої трансляції.

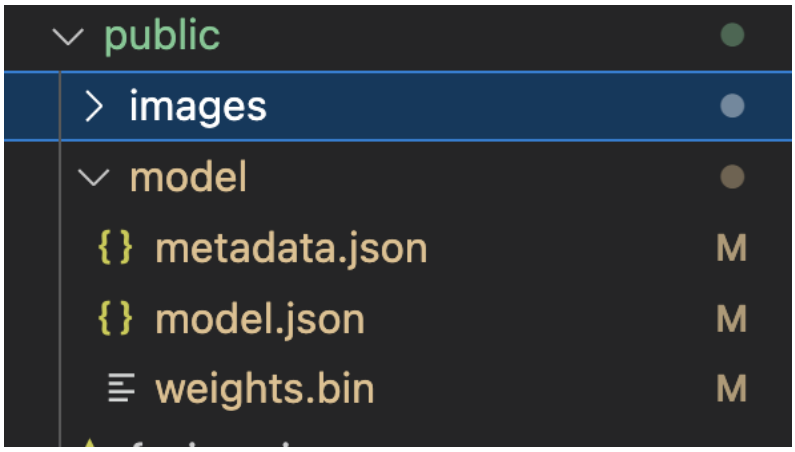


Рисунок 3.12 – Файли згенерованої моделі

Головний код проекту буде писатися в папці src у файлі App.js, також у цьому файлі буде прописата головна логіка проекту, для деяких окремих елементів потрібно зробити інші файли, але це робиться більше для зручності читання проекту, на швидкість самого додатку це ніяк не вплаває.

Спочатку у функції function App() {} треба створити основні змінні для кнопки "старт", для виводу результатів та для анімації загрузки (рис. 3.13).

```
const [start, setStart] = useState(false);
const [result, setResult] = useState([]);
const [loaded, setLoaded] = useState(false);
```

Рисунок 3.13 – Основні змінні

Наступним кроком потрібно створити новий стан useEffect(), у ньому будуть ініціалізуватися такі компоненти як ml5.imageClassifier та videoRef. Спочатку за допомогою imageClassifier зчитується раніше загружена модель, для цього прописується шлях до папки model – "./model/model.json", після визивається коллбек функція, які бере вже зчитану модель, та додає її до компоненту videoRef, приклад такого коду (рис. 3.14).

```
useEffect(() => {
 classifier = ml5.imageClassifier("./model/model.json", () => {
   navigator.mediaDevices
     .getUserMedia({ video: true, audio: false })
     .then((stream) => {
       videoRef.current.srcObject = stream;
       videoRef.current.play();
       setLoaded(true);
     });
 });
}, []);
```

Рисунок 3.14 – Ініціалізування ml5.imageClassifier

Також потрібно вивести результат цього кода до компоненту video. Сам компонент виводиться у визов функції return (). У компонент video потрібно додати чотири атрибути. Перший атрибут ref, він получає змінну videoRef в яку раніше був записаний результат роботи ml5 бібліотеки з моделью. Другий атрибут style, в ньому додається { transform: "scale(-1, 1)" } для зміни розміру вікна, в якому буде відображатися відео. За допомогою останніх двух атрибутів width="200" та height="150" визначається розмір самого вікна, приклад (рис. 3.15).

```
<video
ref={videoRef}
style={{ transform: "scale(-1, 1)" }}
width="400"
height="300"
/>
```

Рисунок 3.15 – Компонент video

Одним з важних кроків, окрім виводу результатів, це перевірка на помилки. Ці два етапи можно об’єднати в одну функції. Сама функція повинна викликатися з невеликою затримкою, для цього буде використовуватися хук useInterval(). Результати будуть записуватися в ранішу створену змінну results за допомогою setResult(results). У випадку якщо виникне помилка, вони буде виводитися у консоль за допомогою console.error(error). Приклад такого коду (рис. 3.16).

```
useInterval(() => {
 if (classifier && start) {
   classifier.classify(videoRef.current, (error, results) => {
     if (error) {
       console.error(error);
       return;
     }
     setResult(results);
     console.log(results)
   });
 }
}, 500);
```

Рисунок 3.16 – Перевірка на помилки

Масив результатів може виглядати так (рис. 3.17).

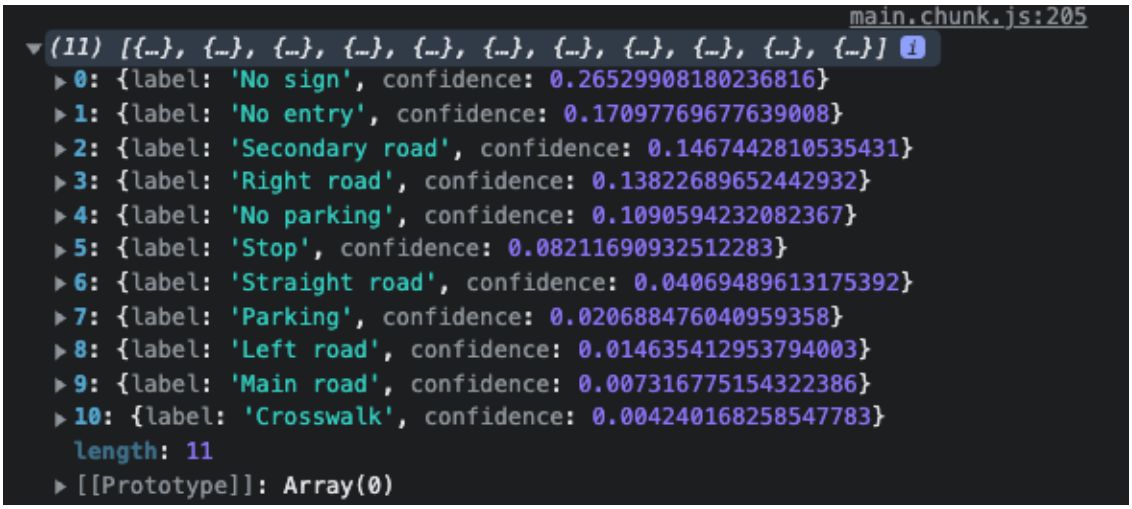


Рисунок 3.17 – Масив результатів

Наступним кроком потрібно створити компонент який буде показувати на екрані ідентифікований знак. Для цього буде використовуватися окремий файл під назвою Signs.js. Спочатку потрібно створити цей файл у корневій папці src. У цьому файлі буде

використовуватися функціональний компенент. Спочатку в цьому файлі потрібно експортувати React бібліотеку, приклад (рис. 3.18).

Рисунок 3.18 – Експортувати React бібліотеки

Після цього треба створити коллбек функцію, та передати в неї атрибут(props). Самі props потрібно буде передавати у настопному кроці. Зараз одразу потрібно експортувати цю фунцію, для того, щоб була змога використовувати цей файл будь де у цій програмі, приклад (рис. 3.19).

Рисунок 3.19 – Експортувати компоненту Signs

У колбек функції спочатку потрібно сворити змінні, у яких буде зберігатися інформація про усі знаки які моделі існують, також список відсортованими елементами, та елементом який зараз визначений, приклад (рис. 3.20).

```
const mostMatched = props.data[0];
const allLabels = props.data.map((elem) => elem.label);
const sortedLabels = allLabels.sort((a, b) => a.localeCompare(b));
```

Рисунок 3.20 – Змінні для моделі

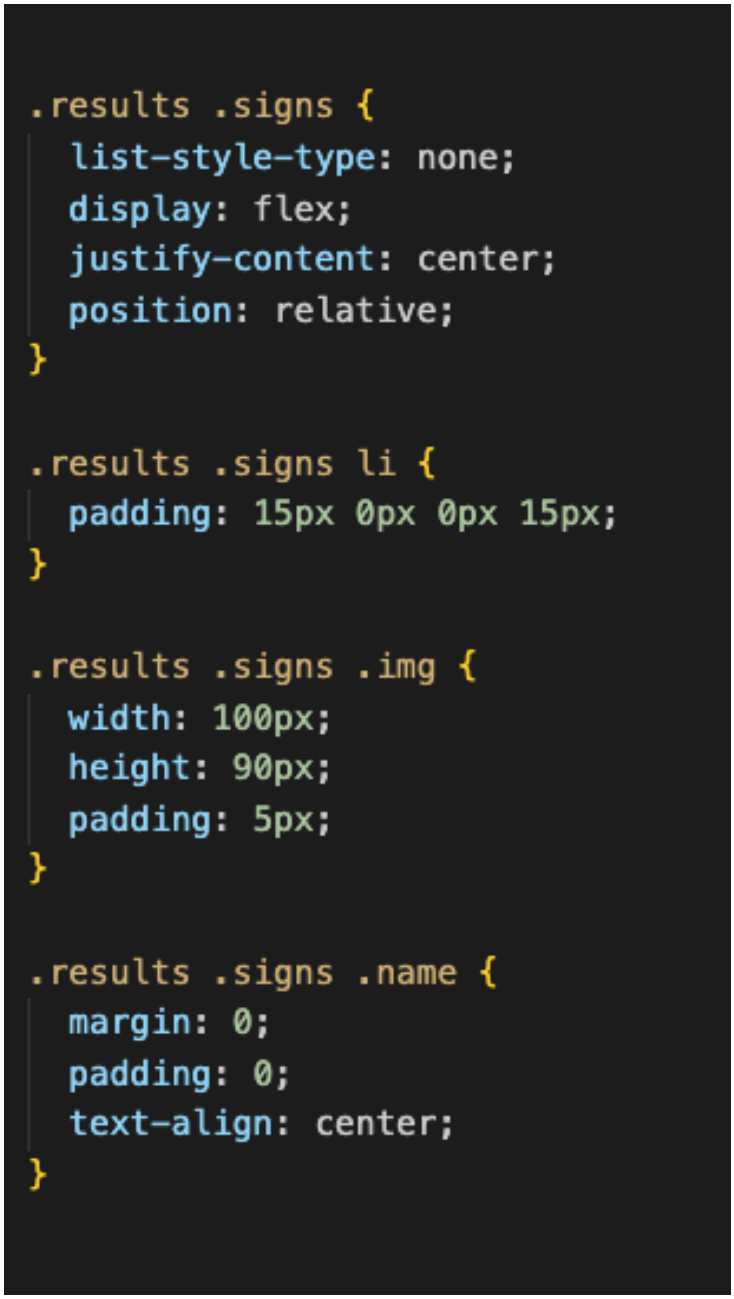
Далі потрібно створити JSX розмітку, яка буде виводити елементи на екран. Вона виглядає як список елементів, та за допомогою JSX до кожного елементу додається свій клас, якщо один з елементів списку співпадає з визначеним елементом, у ньому змінюється його клас. Також під зображенням буде виводиться його назва, приклад (рис. 3.21).

```
<ul className="signs">
        {sortedLabels.map((label) => (
          <li key={label}
            className={`item ${
              label === mostMatched.label ? "selected" : null
            }`}
            >

            <span>
              <img
                className={`img ${
                  label === mostMatched.label ? "selected" : null
                }`}
                src={
                  label === "No sign"
                    ? "./images/No sign.png"
                    : `./images/${label}.png`
                }
                alt={label}
              />
              <p className="name">{label}</p>
            </span>
          </li>
        ))}
      </ul>
```

Рисунок 3.21– JSX розмітка для виведення елементів

Щоб скривати усі елементи окрім визначених було використано CSS, приклад такої розмітки (рис. 3.22).



```css
.results .signs {
  list-style-type: none;
  display: flex;
  justify-content: center;
  position: relative;
}

.results .signs li {
  padding: 15px 0px 0px 15px;
}

.results .signs .img {
  width: 100px;
  height: 90px;
  padding: 5px;
}

.results .signs .name {
  margin: 0;
  padding: 0;
  text-align: center;
}
```

Рисунок 3.22 – Видалення елементів за допомогою CSS

Для того, щоб вивести цей компонент, потрібно використовувати JSX розмітку, та передати в атрибут data результати, які використовуються всередині самого компоненту. Також було створено перевірку на рахунок того, чи взагалі існують результати, якщо виникла помилка, та результатів

нема, то компонент не буде рендеритись зовсім, приклад такої розмітки (рис. 3.23).

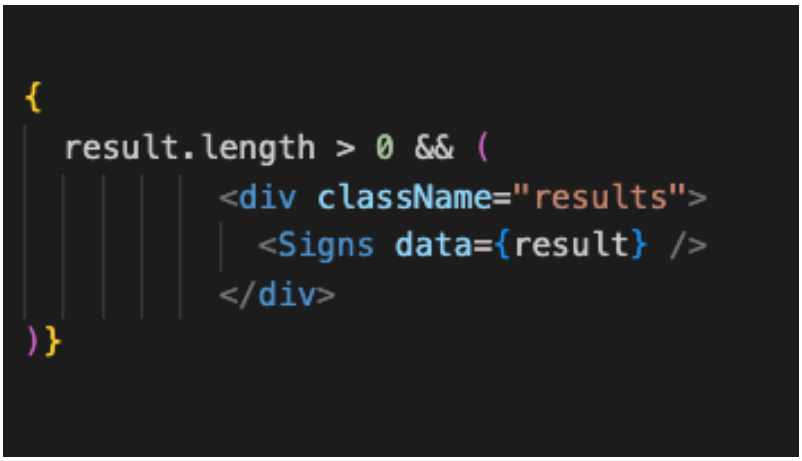


Рисунок 3.23 – Вивід результатів

Також ml5.js дозволяє визначити наскільки чітко той чи інший елемент був визначений. Результат цих розрахунків варіюється від 0 до 1. Для наглядності було створено діаграму яка постійно наглядно демонструє результат цих розрахунків.

Для створення такого функціоналу було використано Chart компонент. Цей компонент вторений в реакт, тому додатково нічого встановлювати не потрібно. Було створено окремий файл Chart.js у кореневій папці src. У цьому файлі потрібно імпортувати реакт та реакт-діаграму, щоб її відображати, приклад (рис. 3.24).

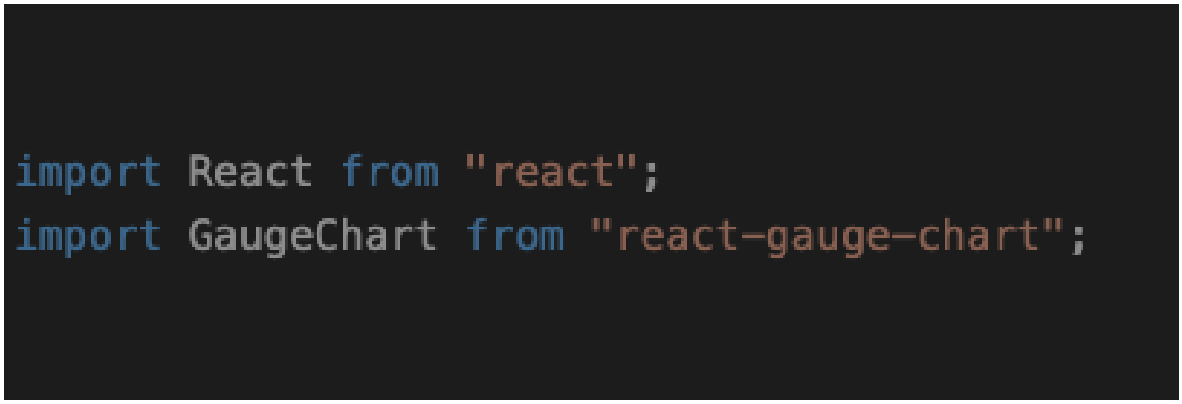


Рисунок 3.24 – Chart діаграма

Наступним кроком потрібно створити колбек функції, та передати в неї результати за допомогою props. Всередині цієї функції потрібно створити змінні, та передати назву поточного елементу, та те, наскільки він був визначений. Також треба перерахувати ці результати у проценти, приклад такого коду (рис. 3.25).

```
const data = props.data;
const label = data.label;
const confidence = parseFloat(data.confidence.toFixed(2));
console.log(label, confidence);
```

Рисунок 3.25 – Результати вичеслення Chart

Після цього потрібно створити JSX розмітку з виводом цього компонент. По-перше додавши заголовок, а потів вивід діаграми за допомогою компоненту GaugeChart. У цьому компоненті потрібно створити п’ять атрибутів. Першим атрибутом створюється власний id. Другий атрибут, це кількість ділянок діаграми nrOfLevels. Третій атрибут, це кольори якими будуть підкрашені ділянки діаграми colors. Четвертий атрибут, це ширина діаграми arcWidth. П’ятий атрибут, це тип виводу результатів percent, приклад такого коду (рис. 3.26).

```
<div>
    <h3>Classification Confidence: {label}</h3>
    <GaugeChart
      id="gauge-chart3"
      nrOfLevels={3}
      colors={["#FF5F6D", "#FFC371", "rgb(26 202 26)"]}
      arcWidth={0.3}
      percent={confidence}
    />
  </div>
```

Рисунок 3.26 – JSX компонент діаграми

Вивід компоненту з діаграмою також потрібно створити через перевірки чи існують результати, якщо результатів немає, то цей компонент не буде навіть рендериться, також у сам компонент потрібно передати результати через атрибут, приклад такого коду (рис. 3.27).

```
{
  result.length > 0 && (
      <div>
        <Chart data={result[0]} />
      </div>
    )}
```

Рисунок 3.27 – Перевірки виводу діаграми

Крім цього для загрузки самої моделі та підключення камери додатку потрібен деякий час, на різних пристроях, а також через різницю швидкості інтернет з'єднання цей може займати від декількох секунд до 10. Для того, щоб у користувача не складалося враження того, що додаток зламаний або

не працює, потрібно додати елемент загрузки сторінки. Для цього можна використовувати компонент під назвою Loader. Цей компонент потрібно додатково встановити за допомогою терміналу. Після встановлення потрібно додати експорт цього компоненту до головного js файлу App.js. Після цього за допомогою JS X потрібно додати цей компонент.

Також цей компонент підтримую різні налаштування. Для додатку було використано шість атрибутів з налаштуванням. Перший атрибут type, за допомогою цього атрибуту можна вибрати дизайн загрузки. Другий атрибут color, за допомогою його вибирається колір загрузки. Третій атрибут height, він визначає висоту елементу. Четвертий атрибут width, за аналогією з третім елементом, за його допомогою визначається ширина елементу. П’ятий атрибут visible, за допомогою цього атрибуту визначається час, коли цей елемент буде на сторінці. В атрибуті приймаються два параметри true або false. Якщо в ньому знаходиться false то цей атрибут не відображається, і навпаки, якщо у середині знаходиться true. Для того, щоб динамічно змінювати значення атрибуту, та не відображати цей елемент на сторінці, було використано useState. У функції

useEffect, яка конвертує загружену модель за допомогою ml5 після успішної конвертації потрібно додати виклик функції для зміни значення змінною loaded. Після цього у атрибут для відображення елемента загрузки потрібно помістити змінну loaded, вона динамічно буде змінюватися після успішної конвертації моделі, та скривати елемент загрузки.

Останній атрибут елемента з загрузкою style, цей атрибут був використаний для відображення елементу посередині сторінки, за допомогою CSS. Приклад коду (рис. 3.28).

```
<Loader
        type="Watch"
        color="#00BFFF"
        height={200}
        width={200}
        visible={!loaded}
        style={{display:'flex', justifyContent:'center', marginTop:'30px' }}
      />
```

Рисунок 3.28 – Компонент загрузки

3.3 Висновки до 3 розділу

За допомогою такого додатку можливо розробляти веб додатки, у яких будуть працювати будь-які навчені моделі TensorFlow. Такий код дозволяє визначати різноманітні об’єкти, при цьому вносити зміни у код майже не потрібно. Навіть якщо модель для використання не оптимізована для ml5 бібліотеки, змінивши лише метод конвертації моделі, все одно такий додаток буде працювати.

Отже можливо зробити висновок, що такий досить оптимізований для різних середовищ. Ще одним великим плюсом використання React, є його мультиплатформенність. Такий код можливо скомпілювати для використання не тільки у браузері, а навіть на платформі Android або iOS, що полегшує роботу розробника.

Для запуску такого додатку на локальній машині процесс налаштування середовища є досить простим. Для запуску коду можливо використовувати майже будь-який редактор коду, навіть IDE не є обов’язковою вимогою. Для запуску достатньо завантажити на комп’ютер пакети npm, та встановити бібліотеку node. Після цього у розробника є змогу запустити локально додаток, також зміни коду будуть одразу відображатися у браузері навіть без перезавантаження сторінки.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

4.1 Виявлення шкідливих і небезпечних факторів в приміщенні лабораторії

Небезпечні і шкідливі виробничі фактори за природою виникнення поділяються на такі групи:

– фізичні;

– хімічні;

– психофізіологічні;

– біологічні.

Робота, яка безпосередньо пов'язана з роботою з комп'ютерами, пов'язана зі шкідливими впливами цілої специфічної групи чинників. Серед усіх факторів виділяють фізичні та психофізичні.

Фізичні фактори, до яких відносяться:

– підвищений рівень електромагнітного випромінювання;

– недостатня освітленість робочого місця;

– високий рівень статичної електрики;

– надмірна запиленість і загазованість повітря;

– небезпека ураження електричним струмом;

– блякла екрана дисплея;

– підвищена або знижена вологість повітря;

– підвищена або знижена температура повітря;

– підвищена або знижена рухливість повітря.

Психофізичні чинники, до яких відносяться:

– фізіологічно недостатня рухова активність (гіподинамія);

– монотонність праці;

– перевантаження окремих систем і органів;

– перевантаження аналізаторів;

– нервово-психічні перенапруження (емоційне, розумовий).

При роботі з ПЕОМ користувач піддається впливу наступних шкідливих фізичних факторів. Підвищений рівень шуму.

Шум погіршує умови праці, роблячи шкідливий вплив на організм людини. Працівник в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, зниження апетиту, біль у вухах. Такі порушення в роботі ряду органів і систем організму людини можуть викликати негативні зміни в емоційному стані людини аж до стресових. Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється втома у зв'язку з підвищеними енергетичними витратами і нервово-психічним напруженням, погіршується мовна комутація. Все це знижує працездатність людини і його продуктивність, якість і безпеку праці. Рівень шуму на робочих місцях не повинен перевищувати 50 дБ.

У ПЕОМ джерелами шуму служать охолоджувальні вентилятори та накопичувачі даних з рухомими частинами.

4.1.1 Несприятливі параметри мікроклімату

Параметри мікроклімату можуть змінюватися в широких межах, в той час, як необхідною умовою життєдіяльності людини є підтримка постійності температури тіла завдяки терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище. Принцип нормування мікроклімату – створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості в приміщенні. У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату

4.1.2 Відсутність або недостатня освітленість робочої зони

Недостатність освітлення призводить до напруги зору, послаблює увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає осліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працівника. Всі ці причини можуть призвести до нещасного випадку або профзахворювань, тому настільки важливий правильний розрахунок освітленості.

4.1.3 Підвищений рівень іонізації повітря

У приміщеннях необхідно контролювати рівень аероіонізації, тому що напруга на аноді монітора викликає іонізацію повітря зі створенням позитивних іонів, які вважаються шкідливими для людини. Позитивні іони, проникаючи в організм, пригнічують обмінні процеси.

4.1.4 Підвищений рівень статичної електрики

При роботі з ВДТ розрядні струми статичної електрики частіше за все виникає при дотику до поверхні екрана або корпусу системного блоку. Такі розряди небезпеки для людини не представляють, але крім неприємних відчуттів вони можуть привести виходу з ладу комп'ютера.

4.2 Заходи щодо захисту від шкідливих і небезпечних факторів

Нормалізацію повітряного середовища виробничих приміщень здійснюють наступними способами:

– підтриманням нормованої величини барометричного тиску;

– природною і механічною вентиляцією;

– кондиціонуванням повітря;

– локалізацією шкідливих чинників;

– опаленням;

– автоматичним контролем і сигналізацією;

– дезодорацією повітря (усуненням неприємних запахів);

– нормування мікроклімату і чистоти повітряного середовища.

Відповідне освітлення забезпечується шляхом раціонального комбінування і застосування природного і штучного освітлення, правильного розміщення монітора на робочому місці щодо віконних прорізів.

Штучне освітлення за своїм устроєм буває двох систем: загальне і комбіноване. При виборі системи освітлення враховують психологічні, фізіологічні, економічні та конструктивні чинники. Так як в приміщенні виконуються роботи високої точності, то доцільніше використовувати систему загального освітлення. У неї включаються стельові і підвісні люмінесцентні світильники загальної освітленості 400 лк. Світильники розподіляються рівномірно рядами і паралельно джерел прямого світла, так щоб екран монітора знаходився в зоні захисного кута світильника, і його проекція не доводилося на екран монітора. Причому, для таких світильників рекомендується використовувати люмінесцентні лампи потужністю по 40 Вт серій ЛП013, ЛП031, ЛПОЗЗ.

Для поліпшення освітленості важливо правильно підібрати колірну обробку інтер'єру і обладнання. Зазвичай стелю і стіни вище панелей від 1,5 м до 1,7 м, якщо вони не облицьовані звукопоглинальним матеріалом, фарбуються водоемульсійною фарбою світлих, холодних тонів.

З огляду на дедалі зростаючу важливість комп'ютера як невід'ємної частини сучасного життя, не можна забувати про небезпеку, яку несе в собі будь-яке електроустаткування. Чим частіше ви маєте справу з ПК та іншою комп'ютерною технікою, тим більше уваги слід приділяти власній безпеці при подібного виду роботі.

Не слід перевантажувати одну розетку, включаючи туди одночасно все обладнання. Краще рівномірно розподіліть навантаження на кілька розеток. Не зайвим буде заземлення ПК. Під час експлуатації уважно стежте за цілісністю ізоляції силового проводу: чи не переламувати їх і не піддавайте додаткового навантаження.

Працюючи за комп'ютером стежте, щоб ваші руки були сухі, не допускайте попадання будь-якої рідини на корпус та допоміжні пристрої ПК. Не вмикайте і не вимикайте техніку занадто часто і без потреби. Чистити від пилу і витирати включений комп'ютер або екран категорично заборонено. Не можна класти на комп'ютерну техніку сторонні предмети і закривати вентиляційні отвори. При найменшій підозрі несправності будь-якого компонента ПК слід припинити роботу з ним і відключити обладнання від мережі.

Зменшення впливу психофізіологічних навантажень на організм людини досягається шляхом правильного оформлення робочого місця [12], раціонального розподілу робочого часу (через кожні 2 години проведені за ПЕОМ необхідно забезпечувати від 10 хв до 15 хв відпочинку), правильним колірним оформленням (коефіцієнти відображення повинні бути: від 60 % до 70 % для стелі, від 40 % до 50 % для стін, 30 % для підлоги, від 30 % до 40 % для інших поверхонь, що відбивають), забезпеченням відповідної настройки параметрів термінального обладнання (контрастність зображення знака не менше 0,8; яскравість освітлення екрана не менше 10 кд/м^2; роздільна здатність екрана 640 × 480 і більше; частота регенерації зображення щонайменше 72 МГц).

4.3 Розрахунок природного освітлення

Природне освітлення – це освітлення, що створюється спрямованим або розсіяним сонячним світлом або світлом неба, що потрапляє через світлові прорізи приміщення. Згідно з санітарними нормами всі

приміщення, в яких постійно знаходяться люди, повинні мати природне освітлення.

Природне освітлення поділяється на такі види:

– верхнє природне освітлення;

– бокове природне освітлення;

– комбіноване природне освітлення.

Розрахунок природного освітлення повинен забезпечити оптимальне виконання вимог, що пред'являються для природного освітлення конкретних приміщень.

Величина природного освітлення змінюється в залежності від широти місцевості, пори року і дня, стан погоди. Тому природне освітлення не можна кількісно ставити величиною освітленості. Природне освітлення в приміщенні визначається коефіцієнтом природного освітлення (КПО), тобто відношення природної освітленості всередині будівлі $E_в$ до одночасно виміряної зовнішньої освітленості горизонтальної поверхні ($E_н$). КПО позначається через "е":

$$e = \frac{E_в}{E_н} \cdot 100\ \%. \qquad (4.1)$$

Зниження розрахункового коефіцієнта природного освітлення стосовно щодо нормованого допускається тільки на 10 %.

Розраховувати норму природного освітлення потрібно виключаючи перешкоди, створювані меблями і деревами.

Верхнє і комбіноване природне освітлення мають ту перевагу, що забезпечують більш рівномірне освітлення приміщення. Бічне природне освітлення створює значну нерівномірність у висвітленні ділянок, розташованих поблизу вікон або далеко від них.

Для встановлення необхідного нормативного значення КПО, тобто ен необхідно врахувати розмір об'єкта розрізнення, тобто розряд зорової

роботи, контраст об'єкта розрізнення і фону, а також характеристику фону. Крім цього, враховується географічна широта місця розташування будівлі і орієнтування приміщення по сторонах горизонту. При природному освітленні нормується його нерівномірність, тобто відношення максимальної до мінімальної освітленості e_{max} / e_{min}.

Що розряд зорової роботи, тим менше допускається нерівномірність освітленості.

Для визначення необхідних площ світлових прорізів бічного освітлення використовується залежність:

$$S_O = \frac{S_п \cdot e_н \cdot h_o \cdot K_{зд} \cdot K_з}{\tau_o \cdot r_i \cdot 100}, \qquad (4.2)$$

де $S_п$ площа підлоги, м^2:

$e_н$ – нормоване значення КПО;

h_o – світлова характеристика відповідно вікон і ліхтарів;

$K_{зд}$ – коефіцієнт обліку затінення вікон протилежними будівлями;

$K_з$ – коефіцієнт запасу;

r_i – коефіцієнт, що враховує підвищення КПО при бічному освітленні завдяки світлу, відбитому від поверхонь приміщення;

τ_o – загальний коефіцієнт світлопропускання светопроемов;

$S_п = 4 \cdot 6 = 24$ м^2.

$$e_н = e_н^{III} \cdot c \cdot m, \qquad (4.3)$$

де c – коефіцієнт сонячності клімату;

m – коефіцієнт світлового клімату.

$e_н = 3 \cdot 0{,}6 \cdot 0{,}9 = 1{,}62$;

$h_o = 15$;

L / B = 3/6 = 0,5;

B / h = 6 / 1,62 = 3,7;

r_i = 1,2;

$К_з$ – для офісного приміщення при бічному освітленні = 1,2;

$К_{зд}$ – при відсутності протилежних будинків = 1.

$$\tau_o = \tau_1 \cdot \tau_2 \cdot \tau_3 \cdot \tau_4, \quad (4.4)$$

де τ_1 – вид світлопропускаючого матеріалу, для віконного листового одинарного скла $\tau_1 = 0,9$;

τ_2 – вид палітурки, для дерев'яних одинарних переплутав $\tau_2 = 0,75$;

τ_3 – несучі конструкції покриттів (при бічному освітленні $\tau_3 = 1$);

τ_4 – сонцезахисні пристрої, вироби і матеріали (з причини відсутності сонцезахисних пристроїв приймаємо $\tau_4 = 1$).

$$\tau_o = 0,9 \cdot 0,75 \cdot 1 \cdot 1 = 0,675.$$

Підставивши всі необхідні дані можна розрахувати необхідну площу світлових прорізів:

$$S_O = \frac{24 \cdot 1,62 \cdot 15 \cdot 1 \cdot 1,2}{0,675 \cdot 1,2 \cdot 100} = 8,64 \text{ м}^2. \quad (4.5)$$

Можна зробити висновок, що площа світлових прорізів офісного приміщення, яка становить 2 · (2 · 2,6) = 10,4 м^2 в повній мірі відповідає необхідним параметрам.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи був виконаний аналіз сучасних видів машинного навчання для визначення об’єктів за допомогою камери пристрою, проаналізовані методи конвертації моделей для використання у вебдодатках.

В ході виконання роботи була вивчена предметна область машинного навчання. Було проведено аналіз сучасних існуючих бібліотек для інтеграції моделей машинного навчання у вебдодатки. Були розроблені алгоритми роботи програм. Були виконані наступні завдання:

− обрано метод машинного навчання моделі;

− обрані середовища для розробки;

− вивчена предметна область вже існуючих програмних рішень;

− обрано мову програмування;

− описана структура програмного додатку;

− розроблено алгоритми конвертації моделей для вебдодатків;

− розроблено програмне забезпечення.

У результаті розроблено програмне забезпечення для машинного навчання моделей для розпізнавання об’єктів за допомогою бібліотек розроблених мовою програмування JavaScript.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Документація. Звіти у сфері науки та техніки. структура та правила оформлення [Текст]. – Введ. 2015-06-22. – К.: Держстандарт України, 2017. – 29 с.

2. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології» [Текст]: довід. / І.Ш. Невлюдов, А.О. Андрусевич, О.В. Токарєва, Г.В. Пономарьова. – К.: Київ-56, пр. Космонавта Комарова, 1, 2016. – 320 с.

3. Методичні вказівки до підготовки атестаційної роботи бакалавра для студентів усіх форм навчання спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» освітньої програми "Системна інженерія" [Текст] / упоряд.: І.Ш. Невлюдов, О.В. Токарєва, Г.В. Пономарьова. – Харків: ХНУРЕ, 2019. – 36 с.

4. Положення про протидію академічному плагіату в ХНУРЕ [Електронний ресурс]. – Режим доступу: www/ URL: https//nure.ua/wp-content/uploads/Main_Docs_NURE/polozhennnjapro-akademichnu-dobrochesnist. pdf – 05.06.2021 р. – Загол. з екрану.

5. Моделі з Layers API [Электронный ресурс] / Lits. – Режим доступа: www/ URL: https://www.tensorflow.org/js/guide/models_and_layers – 08.03.2022 р. – Загол. з екрана.

6. TensorFlow.js layers API for Keras users [Электронный ресурс] / TensorFlow. – Режим доступа: www/ URL: https://www.testing-whiz.com/ – 21.03.2021 р. – Загол. з екрана.

7. TensorFlow is an end-to-end open source platform for machine learning [Электронный ресурс]] / Selenium. – Режим доступа: www/ URL https://opensource.google/projects/tensorflow – 21.03.2022 р. – Загол. з екрана.

8. Framework for machine learning [Електронний ресурс] / Testing. – Режим доступу: www/ URL: https://www.databricks.com/glossary/tensorflow-

guide – 10.05.2022 р. – Загол. з екрана.

9. Tensors are the core datastructure [Електронний ресурс] / Software. – Режим доступу: www/ URL: https://js.tensorflow.org/api/latest/ – 23.05.2022 р. – Загол. з екрана.

10. Keras: the Python deep learning API [Електронний ресурс] / Selenium. – Режим доступу: www/ URL: https://keras.io/api/ – 11.04.2022 р. – Загол. з екрана.

11. Нікітін В. В. Аналіз елементів навчального вебдодатка для автоматизації тестування. – Автоматизація та приладобудування. («Automation and Development of Electronic Devices» ADED-2021) [Текст] : збірник студентських наукових статей / Харківський національний університет радіоелектроніки; [редкол.: І.Ш. Невлюдов та ін.]. – Харків : ХНУРЕ, 2021. – Вип. 1. – С. 88-92.

12. Комплекс навчально-методичного забезпечення навчальної дисципліни «Основи охорони праці та цивільний захист підготовки освітнього рівня бакалавр усіх спеціальностей та усіх напрямів університету напрямів університету» [Текст] / ХНУРЕ; розроб.: Т.Є. Стиценко, В.А. Айвазов, О. В. Мамонтов, Н.М. Сердюк. – Харків, 2017. – 517 с.