МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

БЄЖИН ЄВГЕН ВОЛОДИМИРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

АЛГОРИТМ ІДЕНТИФІКАЦІЇ ТЕКСТУ ЗА РУХАМИ РУКИ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
П. К. Ніколюк, професор кафедри
інформаційних технологій,
д.ф.-м.н., професор
_____

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

Голова ЕК: _____

Вінниця - 2025

АНОТАЦІЯ

Бєжин Є.В. Розробка алгоритму розпізнавання тексту на основі рухів руки. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У кваліфікаційній роботі досліджено сучасні методи ідентифікації тексту за рухами руки з використанням технологій комп'ютерного зору та глибинного навчання. Розроблено алгоритм на базі Python, OpenCV та TensorFlow, що забезпечує розпізнавання символів у реальному часі. Система інтегрує модулі відстеження руху, попередньої обробки зображень та класифікації за допомогою згорткових нейронних мереж.

Ключові слова: розпізнавання жестів, комп'ютерний зір, нейронні мережі, OpenCV, TensorFlow.

67 ст. 27 рис., 2 табл., 5 дод., 18 джерел.

ABSTRACT

Bezhyn Y. Development of a Hand Motion-Based Text Recognition Algorithm. Specialty 122 «Computer Science», educational program «**Computer Science**». Vasyl Stus Donetsk National University, Vinnytsia, 2024.

The thesis explores modern methods of text identification through hand movements using computer vision and deep learning technologies. A Python-based algorithm utilizing OpenCV and TensorFlow is developed for real-time character recognition. The system integrates motion tracking, image preprocessing, and convolutional neural networks for classification.

Keywords: gesture recognition, computer vision, neural networks, OpenCV, TensorFlow.

ЗМІСТ

ВСТУП

У сучасному світі, де інтерактивні технології все глибше інтегруються в повсякденне життя, важливим напрямом є розробка систем, які забезпечують природну та інтуїтивно зрозумілу взаємодію між людиною та комп'ютером. Одним із перспективних напрямів є ідентифікація тексту за рухами руки, яка дозволяє користувачам вводити текст без використання клавіатури або сенсорного екрану. Ця технологія має широке застосування в інтерфейсах для людей з обмеженими можливостями, індустрії додаткової реальності (AR/VR), освітніх системах, а також у медичних і промислових середовищах, де фізичний контакт із пристроями є небажаним.

Однак існуючі методи розпізнавання жестів стикаються з кількома викликами. По-перше, точність систем залежить від змін освітлення, фону та індивідуальних особливостей почерку користувача. По-друге, традиційні алгоритми (наприклад, порогова сегментація або виявлення ознак (HOG/SIFT)) мають обмежену ефективність у динамічних умовах. У цьому контексті глибинне навчання (CNN, LSTM) пропонує більш адаптивні рішення, але вимагає значних обчислювальних ресурсів. Тому актуальним є створення гібридних моделей, які поєднують класичні методи обробки зображень із досягненнями штучного інтелекту.

Метою даної роботи є аналіз існуючих методів ідентифікації тексту за рухами руки, визначення їх переваг та недоліків, а також розробка алгоритму, який реалізує цей функціонал. Для досягнення мети вирішувалися такі задачі:

1. Дослідження сучасних підходів до розпізнавання жестів, включаючи методи глибинного навчання.
2. Аналіз ефективності алгоритмів у різних умовах (освітлення, шуми, варіативність жестів).
3. Розробка модульної архітектури програмного забезпечення на основі Python, OpenCV та MediaPipe.
4. Тестування системи на базі даних EMNIST для оцінки її точності та швидкості.

Робота структурована наступним чином: у першому розділі надано огляд методів розпізнавання жестів, у другому — описано вибір інструментів для реалізації, у третьому — деталізовано архітектуру та експерименти. Висновки містять рекомендації щодо подальшого вдосконалення системи.

Отримані результати демонструють, що гібридні підходи, які поєднують класичні алгоритми з глибинним навчанням, є перспективними для створення адаптивних систем ідентифікації тексту. Це відкриває нові можливості для розвитку безконтактних інтерфейсів у різноманітних сферах діяльності.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ МЕТОДІВ ІДЕНТИФІКАЦІЇ ТЕКСТУ ЗА РУХАМИ РУКИ

1.1 Актуальність теми, мета та задачі дослідження

У сучасному світі спостерігається стрімкий розвиток технологій розпізнавання жестів та рухів рук, що знаходять застосування в різних сферах, включаючи біометричну автентифікацію, біоцифровізацію та альтернативні методи введення даних. Особливий інтерес викликають системи, здатні ідентифікувати текст на основі рухів руки, оскільки вони відкривають нові можливості для безконтактного введення інформації та підвищення зручності користувачів.

Актуальність теми обумовлена потребою в розробці інноваційних методів введення текстової інформації, які б забезпечували природну та інтуїтивно зрозумілу взаємодію користувача з комп'ютерними системами. Традиційні методи введення, такі як клавіатури та сенсорні екрани, мають певні обмеження, особливо в умовах, коли фізичний контакт небажаний або неможливий. Технології розпізнавання рухів руки дозволяють вирішити ці проблеми, забезпечуючи безконтактне введення тексту та розширюючи можливості користувачів.

Мета дослідження полягає в аналізі існуючих методів ідентифікації тексту за рухами руки, визначенні їх переваг та недоліків, а також виявленні напрямків для подальших досліджень та вдосконалення технологій у цій галузі.

Для досягнення поставленої мети необхідно вирішити такі **задачі дослідження**:

1. Проаналізувати сучасні методи розпізнавання рухів руки та їх застосування для ідентифікації тексту.
2. Оцінити ефективність різних підходів до розпізнавання тексту на основі рухів руки, включаючи використання нейронних мереж та інших алгоритмів машинного навчання.
3. Визначити основні виклики та обмеження існуючих методів, а також можливі шляхи їх подолання.

4. Розглянути перспективи розвитку та впровадження технологій ідентифікації тексту за рухами руки в різних сферах діяльності.

1.2 Дослідження предметної області та історії розвитку технологій розпізнавання жестів

1.2.1 Дослідження предметної області розпізнавання жестів

Розпізнавання жестів є ключовим напрямом у розвитку інтерактивних систем, що забезпечують природну та інтуїтивно зрозумілу взаємодію між людиною та комп'ютером, [16] як це зображено на рис.1.1.

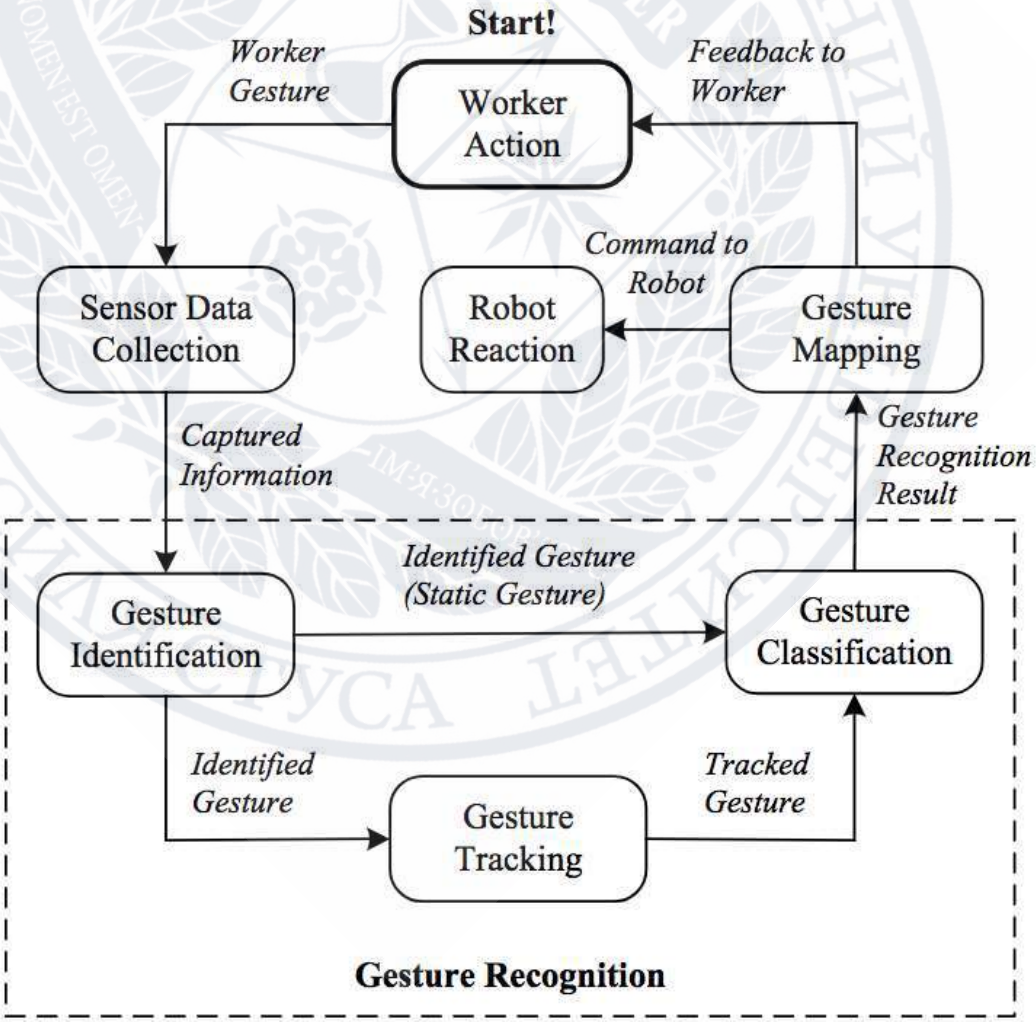


Рисунок 1.1 – Схема розпізнавання жестів

Ця технологія використовується в багатьох сферах, включаючи:

- **Інтерфейси для людей з обмеженими можливостями**: Розпізнавання жестів допомагає покращити комунікацію для осіб з вадами слуху та мовлення, забезпечуючи перетворення жестової мови на текст або аудіо [1].
- **Інтерактивні презентації та мультимедійні системи**: Використання жестів для керування презентаціями та мультимедійними матеріалами робить взаємодію більш динамічною та захоплюючою [2].
- **Робототехніка та автоматизація**: Розпізнавання жестів використовується для дистанційного керування роботами та іншими автоматизованими системами, що підвищує ефективність та безпеку операцій [3].

1.2.2 Історія розвитку технологій розпізнавання жестів

Розвиток технологій розпізнавання жестів пройшов кілька етапів:

- **Початкові дослідження (1970-1980-ті роки)**: Перші спроби розпізнавання жестів були пов'язані з використанням спеціальних датчиків та обмежених комп'ютерних можливостей. Основна увага приділялася розпізнаванню окремих жестів у контрольованих умовах.
- **Розвиток комп'ютерного зору та машинного навчання (1990-2000-ні роки)**: Завдяки прогресу в обробці зображень та розвитку алгоритмів машинного навчання, з'явилися системи, здатні розпізнавати більш складні жести в реальному часі.
- **Сучасні досягнення (2010-ті роки - сьогодення)**: Інтеграція глибокого навчання та нейронних мереж дозволила досягти високої точності та швидкості розпізнавання жестів. Сучасні системи здатні працювати в динамічних умовах з мінімальними вимогами до апаратного забезпечення [4].

1.3 Огляд сучасних алгоритмів і методів розпізнавання рухів руки

Сучасні технології розпізнавання рухів руки базуються на комплексних підходах, що інтегрують методи комп'ютерного зору, машинного та глибинного навчання. У цьому розділі розглядаються основні алгоритми та методи, які використовуються для аналізу та класифікації жестів, а також обговорюються їх переваги, недоліки та специфічні виклики.

На початкових етапах розвитку технологій розпізнавання жестів широко використовувалися методи обробки зображень, які базувалися на виділенні характерних ознак (feature extraction). До таких методів належать:

- **Histogram of Oriented Gradients (HOG)**, зображений на рис.1.2
- **Scale-Invariant Feature Transform (SIFT)**, зображений на рис.1.3
- **Speeded Up Robust Features (SURF)** зображений на рис.1.4

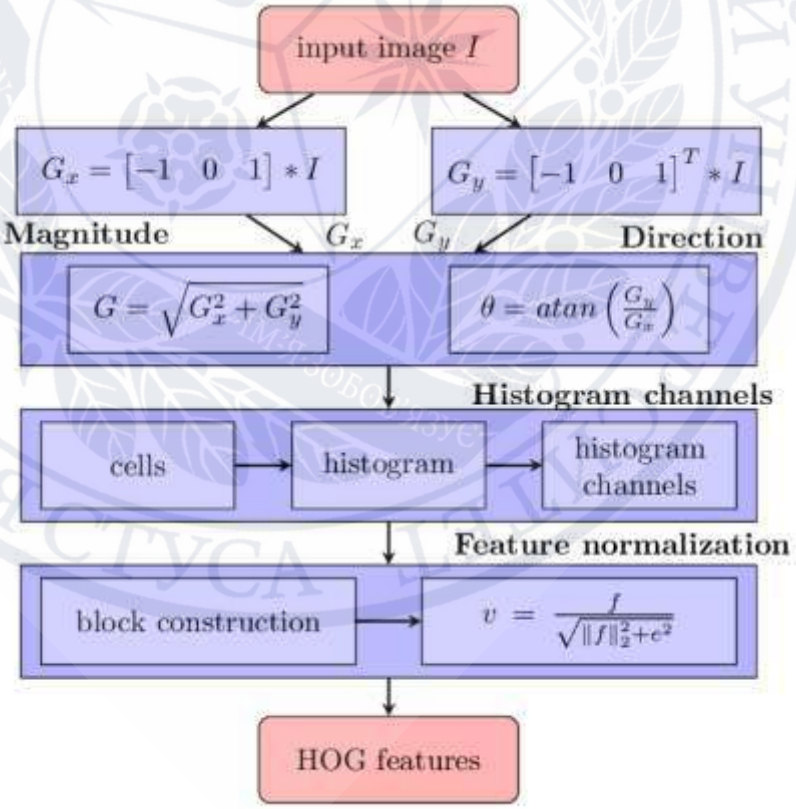


Рисунок 1.2 – Схема **HOG**

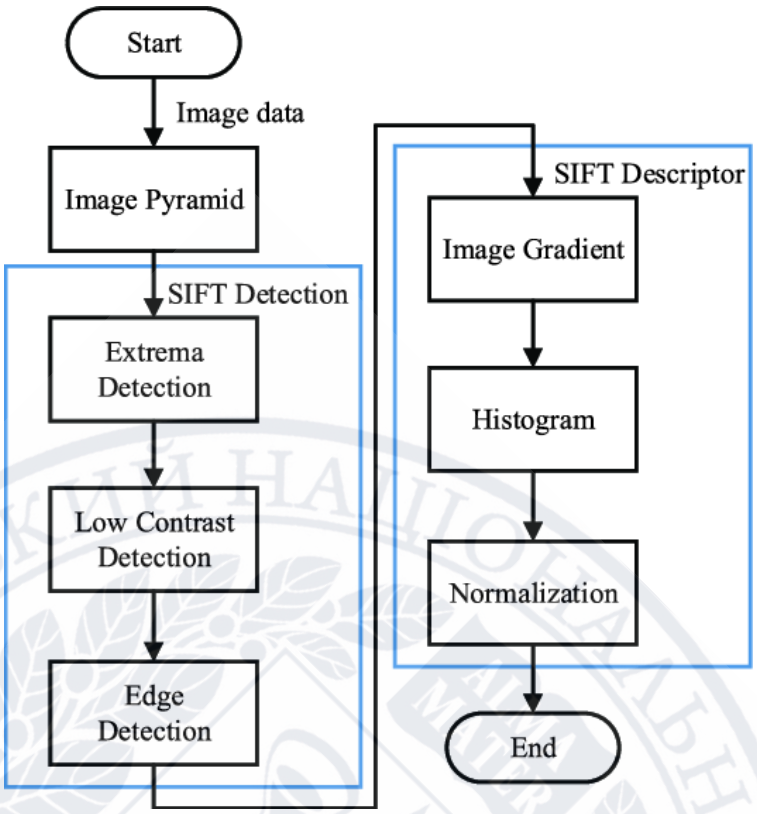


Рисунок 1.3 – Схема **SIFT**

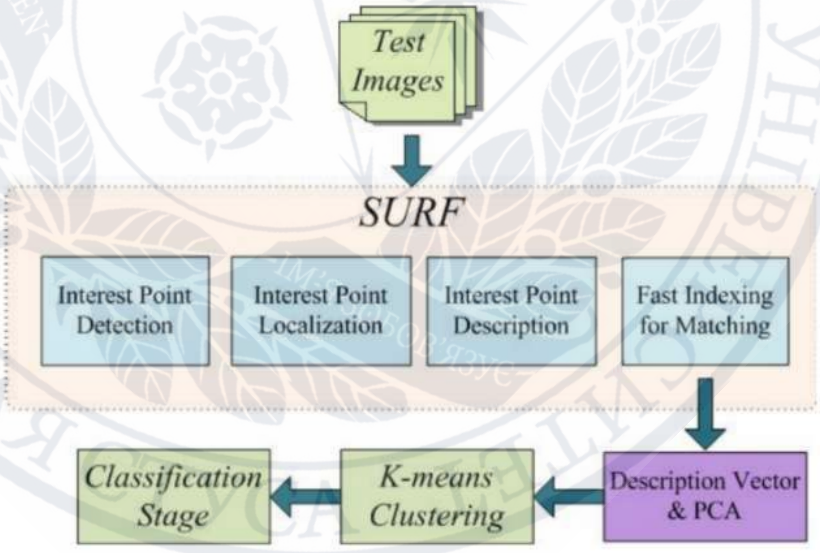


Рисунок 1.4 – Схема **SURF**

Ці алгоритми дозволяють ефективно виділяти структурні особливості зображень рук, що є важливим для подальшої класифікації жестів [5]. Проте їх застосування обмежується необхідністю попередньої підготовки даних та часто залежить від умов освітлення й фону.

Для підвищення точності розпізнавання жестів у системах почали застосовувати класичні алгоритми машинного навчання, такі як:

- **Метод опорних векторів (Support Vector Machines, SVM)**,
- **k-ближчих сусідів (k-Nearest Neighbors, k-NN)**,
- **Дерева прийняття рішень**.

Ці алгоритми часто використовуються для класифікації жестикуляційних даних, отриманих після попереднього виділення ознак за допомогою описаних вище методів. Поєднання традиційного виділення ознак із сучасними алгоритмами класифікації дозволяє досягти задовільної точності при розпізнаванні в контрольованих умовах [6].

За останнє десятиліття значний прогрес було досягнуто завдяки застосуванню глибинних нейронних мереж, зокрема:

- **Convolutional Neural Networks (CNN)**, які автоматично виділяють релевантні ознаки з зображень рук, схема якої зображена на рис.1.5
- **Recurrent Neural Networks (RNN)** та їх модифікації, зокрема **Long Short-Term Memory (LSTM)**, схема якої зображена на рис.1.6, що ефективно працюють з часовими рядами даних для аналізу послідовностей рухів.

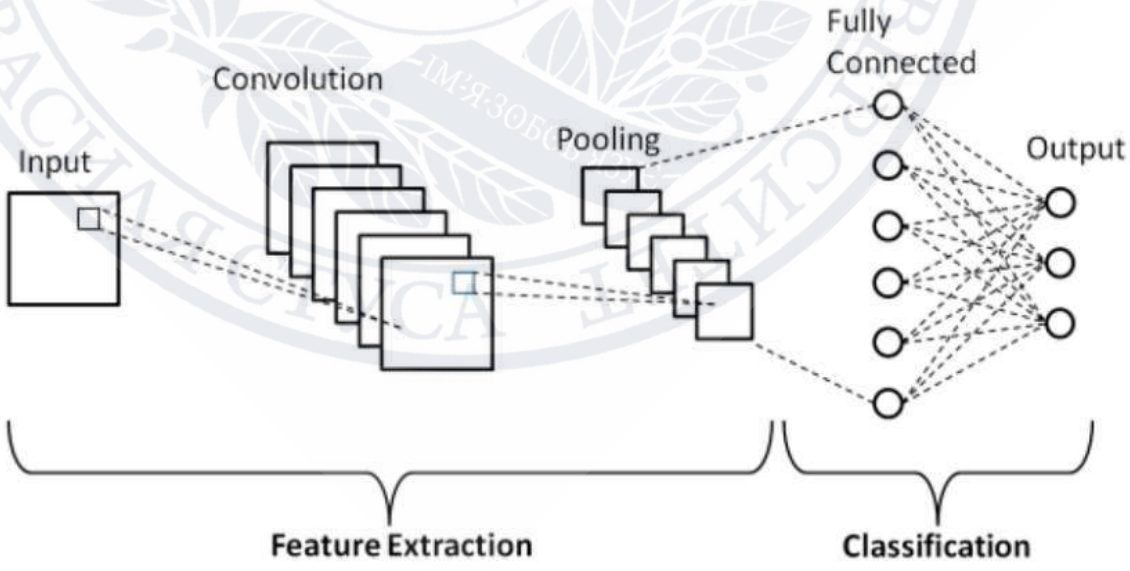


Рисунок 1.5 – Схема **CNN**

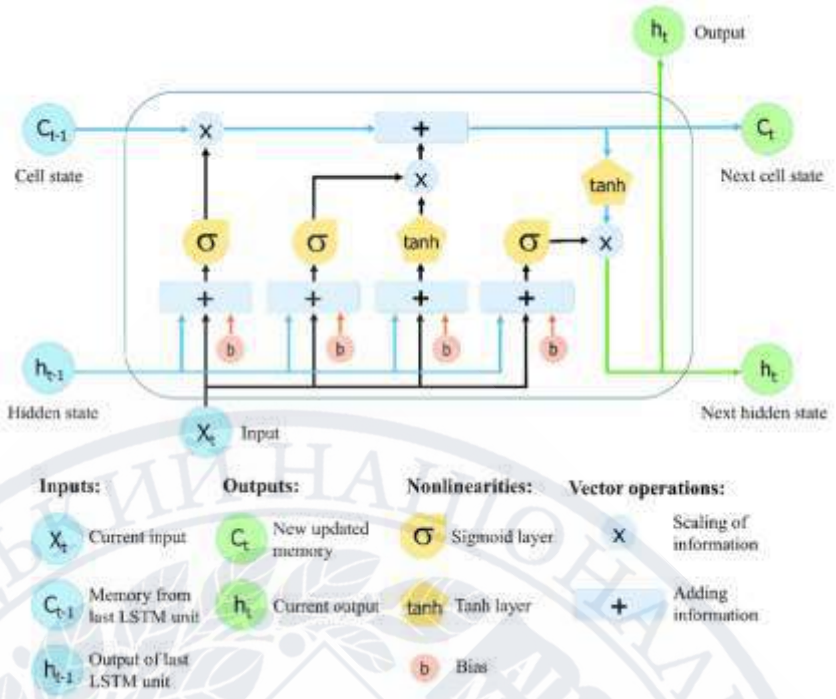


Рисунок 1.6 – Схема **LSTM**

Інтегровані архітектури, наприклад, CNN-LSTM, дозволяють не лише класифікувати окремі кадри, а й враховувати динаміку зміни положення руки у просторі, що забезпечує більш точне розпізнавання складних жестів у реальному часі [7].

З появою глибинних сенсорів (наприклад, камери Kinect) з’явилася можливість використовувати як кольорову інформацію (RGB), так і дані про глибину (Depth), що суттєво покращує якість розпізнавання. Об’єднання даних з двох джерел дозволяє системам розпізнавати жести навіть у складних умовах навколишнього середовища, забезпечуючи більшу стійкість до змін освітлення та фону [8].

Попри значні досягнення, сучасні алгоритми розпізнавання рухів руки стикаються з рядом викликів:

- **Варіативність жестів.** Різноманітність манер виконання жестів різними користувачами ускладнює створення універсальних моделей.
- **Умови середовища.** Зміни в освітленні, фонових зображеннях та часткове затемнення можуть суттєво впливати на точність розпізнавання.

- **Обмеженість навчальних даних.** Для навчання глибоких моделей необхідні великі об’єми анотованих даних, що не завжди доступні.

Порівняльний аналіз сучасних підходів свідчить про те, що гібридні моделі, які поєднують класичні алгоритми з методами глибинного навчання, мають потенціал для досягнення максимальної точності та адаптивності в реальних умовах [9].

1.4 Порівняльний аналіз існуючих підходів та технологій розпізнавання тексту

В сучасних дослідженнях, розпізнавання тексту отримало значний розвиток завдяки застосуванню як класичних методів обробки зображень, так і інноваційних алгоритмів глибинного навчання. Аналіз літератури свідчить про те, що традиційні методи, що базуються на пороговій сегментації, фільтрації та виділенні характерних ознак, забезпечують стабільні результати у контрольованих умовах, проте їх ефективність знижується при зміні умов освітлення або фонового шуму [10].

Сучасні технології розпізнавання тексту, які використовують нейронні мережі, зокрема Convolutional Neural Networks (CNN) та Recurrent Neural Networks (RNN), демонструють високу точність завдяки здатності автоматичного виділення релевантних ознак із зображень. Такий підхід дозволяє зменшити залежність від попередньої обробки даних, що є важливою перевагою в умовах динамічного середовища [11].

До того ж, існують гібридні системи, що поєднують класичні алгоритми з методами машинного та глибинного навчання, дозволяючи адаптувати алгоритми до специфічних умов експлуатації. Такі системи оптимізують процес розпізнавання тексту шляхом комбінації попередньої обробки зображень із сучасними методами аналізу даних [12].

Особливу увагу приділяють технологіям розпізнавання тексту за допомогою жестів рук. Сучасні інтерактивні системи, що інтегрують обробку відео та

аналіз динамічних послідовностей, здатні враховувати контекст жестового введення, що дозволяє підвищити зручність та інтуїтивність user interface, хоча й може дещо поступатися за точністю класичним OCR-системам [13].

Також численні дослідження, опубліковані в Інтернеті, розширюють наше розуміння даної проблематики. Сучасні статті та огляди, розміщені на таких ресурсах, як arXiv, IEEE Xplore та SpringerLink, демонструють тенденції розвитку технологій розпізнавання тексту з використанням глибинних нейронних мереж, що дозволяють досягати високої точності навіть у складних умовах експлуатації [14]. Сучасні дослідження також пропонують нові підходи до інтеграції класичних методів з інноваційними алгоритмами, що відкриває перспективи для створення більш адаптивних і ефективних систем розпізнавання тексту [15].

Порівняльний аналіз існуючих підходів свідчить про те, що вибір технології залежить від конкретних вимог до системи: в умовах стабільного середовища доцільно застосовувати традиційні методи, тоді як для динамічних та інтерактивних систем перевагу надають методам на базі глибинного навчання та інтеграції жестових технологій.

РОЗДІЛ 2. ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ АЛГОРИТМУ

2.1 Аналіз мов програмування та інструментів розробки для алгоритмічних рішень

Основними критеріями при виборі мови програмування є швидкість розробки, підтримка бібліотек для обробки зображень і відео, а також можливість інтеграції з сучасними фреймворками для машинного навчання. Найбільш вдалою в цьому контексті є:

- **Python**:
  - Широка екосистема бібліотек, таких як **OpenCV** для обробки зображень і відео, **MediaPipe** для виявлення та аналізу жестів, **TensorFlow** або **PyTorch** для реалізації нейронних мереж.
  - Простота синтаксису і велика спільнота дозволяють швидко знаходити приклади використання, рішення для налагодження та оптимізації.
  - Підтримка апаратного прискорення (наприклад, за допомогою GPU) сприяє реалізації алгоритмів реального часу.

Інструменти розробки та середовища

Окрім вибору мови, важливими є засоби для розробки та тестування:

- **OpenCV**: Забезпечує високоефективні методи обробки зображень, фільтрації, трансформації кадрів і реалізації алгоритмів виявлення об'єктів. Приклад роботи зображено на рис.2.1

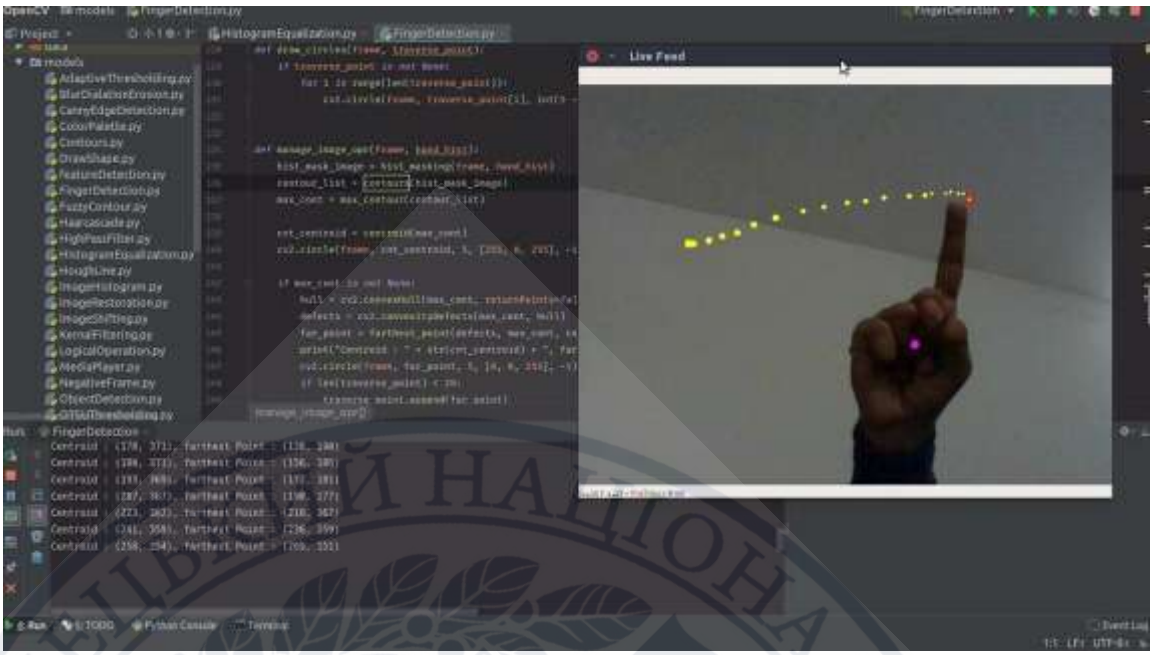


Рисунок 2.1 – Приклад роботи **OpenCV**

- **MediaPipe**: Пропонує готові рішення для трекінгу рук, що дозволяють отримати координати ключових точок руки з високою точністю. Приклад роботи зображено на рис.2.2

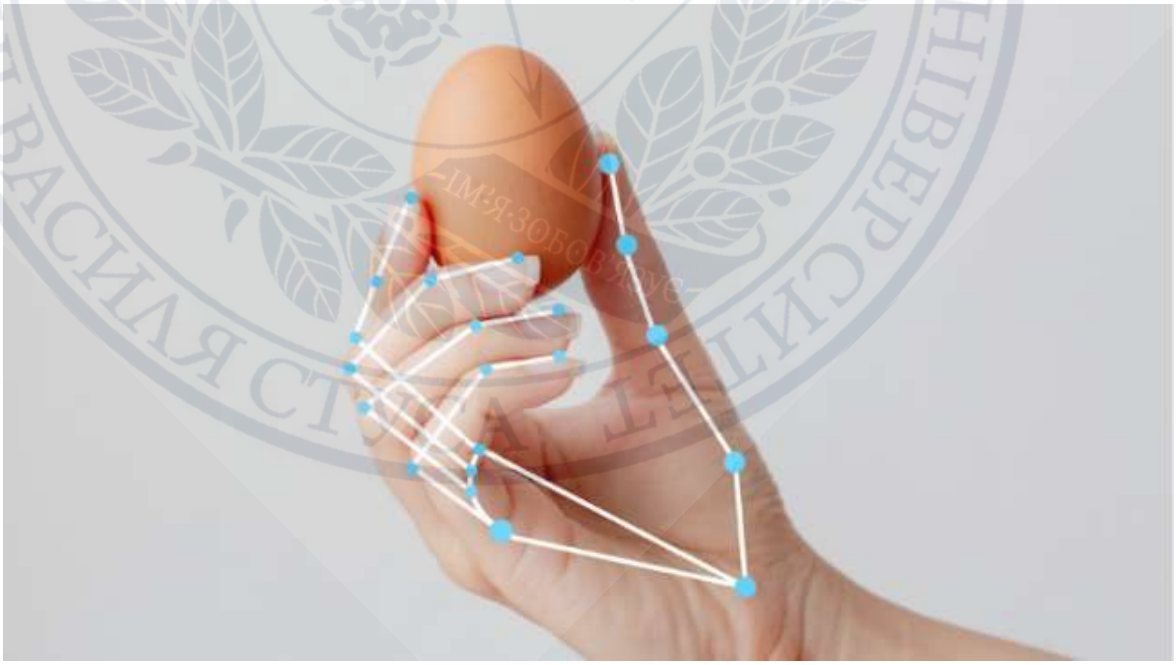


Рисунок 2.2 – Приклад роботи **MediaPipe**

- **TensorFlow/PyTorch**: Забезпечують інструменти для побудови, навчання та впровадження нейронних мереж, що класифікують послідовності рухів у конкретні символи чи слова. Приклад моделі нейромережі написаній на **TensorFlow** зображено на рис.2.3

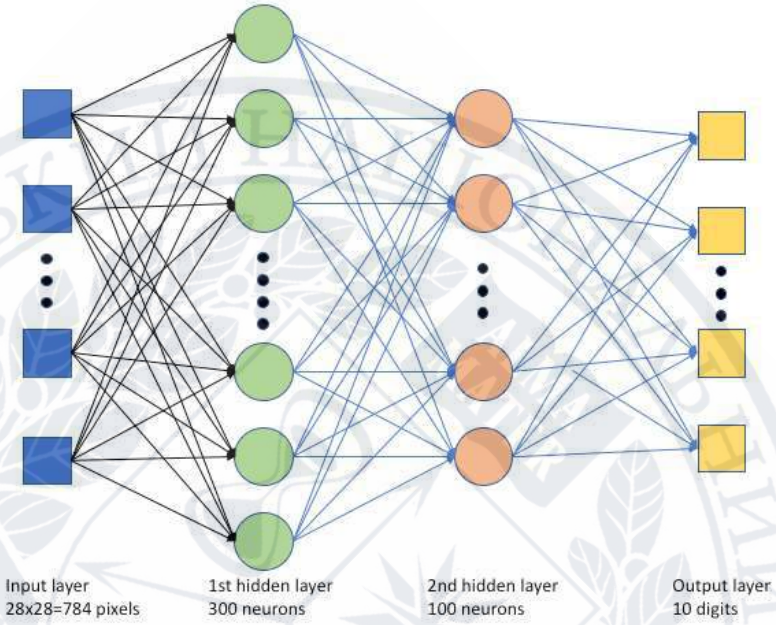


Рисунок 2.3 – Приклад роботи **TensorFlow**

- **Flask/Django** (за необхідності): Створює веб-інтерфейс або API, через який можна інтегрувати систему розпізнавання в більші проєкти. Приклад інтерфейсу, який написано на **Flask**, зображено на рис.2.4

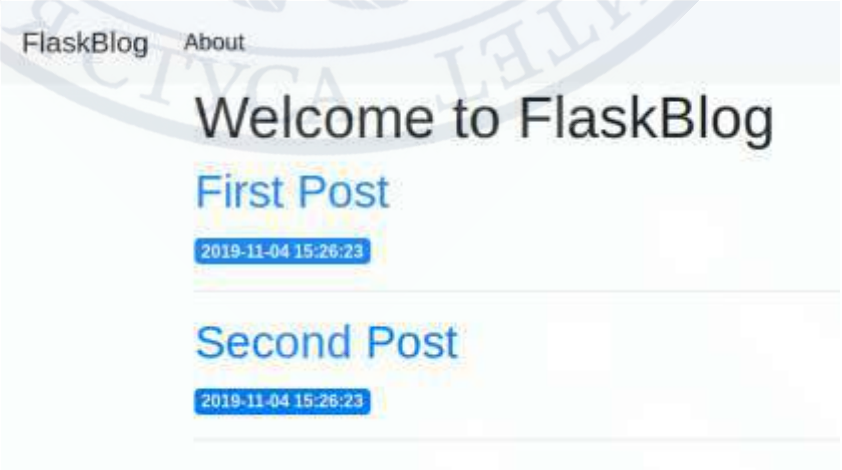


Рисунок 2.4 – Приклад роботи **Flask**

Порівняльний аналіз

Розглядаючи альтернативи, можна зазначити, що:

- **C++** може бути використано для підвищення продуктивності на етапі інференсу (продакшену), проте розробка в Python значно спрощує інтеграцію бібліотек і алгоритмів.
- **JavaScript** (через TensorFlow.js) може використовуватися для веб-орієнтованих застосунків, але обмеження браузера можуть бути проблематичними для задач обробки відео у реальному часі.

Таким чином, Python обрано як основну мову завдяки її зручності, широкій підтримці інструментів і можливостям для швидкого прототипування.

2.2 Огляд методів отримання відеоданих для аналізу рухів руки

Основні джерела відеоданих

Для отримання якісних відеоданих, необхідних для аналізу жестів і рухів руки, можуть застосовуватися різні методи:

1. **Веб-камера**:
    - Найпопулярніший і доступний метод, що дозволяє отримувати потік відео в реальному часі.
    - Сучасні веб-камери забезпечують достатню роздільну здатність і кадрову частоту, необхідні для точного розпізнавання швидкоплинних рухів.
    - Недоліком може бути вплив змін освітлення та фону, тому необхідні методи попередньої обробки зображень.

Приклад веб камер зображено на рис.2.5

Рисунок 2.5 – **Веб камера**

2. **Попередньо записані відео**:
   - Використовуються для навчання та тестування алгоритмів у контрольованих умовах.
   - Дає змогу проводити повторну валідацію моделі на однакових даних, що сприяє оптимізації алгоритмів.
3. **Глибинні камери (Intel RealSense, Kinect)**:
   - Забезпечують не лише двовимірне зображення, але й дані глибини, що дозволяють точніше визначати положення та орієнтацію руки.
   - Підвищують точність сегментації руки на фоні та дозволяють враховувати 3D-аспекти рухів, що важливо для складних жестів.

Приклад глибинної камери зображено на рис.2.6

Рисунок 2.6 – **Intel RealSense**

Аспекти обробки відеоданих

При роботі з відеоданими слід враховувати такі моменти:

- **Кадрова частота та затримка**: Важливо забезпечити обробку відео з мінімальною затримкою для реального часу, що критично для інтерактивних застосунків.
- **Якість зображення**: Впливає на точність виявлення ключових точок. Необхідно проводити попередню обробку для зменшення шумів та компенсації змін освітлення.
- **Роздільна здатність**: Баланс між деталізацією зображення та обчислювальними витратами, що визначає продуктивність системи.

У даному проєкті для забезпечення оптимальної роботи використовується веб-камера з високою кадровою частотою, що дозволяє забезпечити стабільну роботу алгоритму в режимі реального часу.

2.3 Проєктування архітектури програмного забезпечення та алгоритму розпізнавання

Основні компоненти системи

Архітектура програмного забезпечення побудована за принципом модульності, що дозволяє окремо розробляти, тестувати та вдосконалювати кожен компонент. Основні модулі включають:

1. **Модуль збору відеоданих (Data Acquisition Module)**:
    - Відповідає за захоплення відео з веб-камери або інших джерел.
    - Забезпечує буферизацію кадрів, синхронізацію та передачу відеопотоку до наступних модулів.
    - Реалізований з використанням OpenCV, що дозволяє зчитувати кадри з високою частотою.
2. **Модуль попередньої обробки (Preprocessing Module)**:
    - Виконує попередню обробку зображень: нормалізацію, зміну розміру, фільтрацію шумів, корекцію кольору.

- Застосовує методи сегментації для виділення області руки, наприклад, використання алгоритмів на базі кольорової сегментації або фонового віднімання.
- Підготовка даних до аналізу дозволяє підвищити точність подальших етапів розпізнавання.

3. **Модуль детекції та відстеження руки (Hand Detection & Tracking Module)**:
   - За допомогою MediaPipe або спеціалізованих алгоритмів на базі OpenCV здійснює визначення ключових точок руки (суглоби, пальці).
   - Забезпечує стабільне відстеження руки навіть при швидких рухах та зміні позиції.
   - Вихідними даними є координати ключових точок, що становлять скелет руки.
4. **Модуль екстракції ознак (Feature Extraction Module)**:
   - На основі координат ключових точок формується вектор ознак, що описує положення та динаміку рухів.
   - Може включати аналіз часових рядів, нормалізацію даних, застосування методів зменшення розмірності (наприклад, PCA) для спрощення моделі.
5. **Модуль класифікації рухів (Classification Module)**:
   - Використовує методи машинного навчання для перетворення послідовності ознак у конкретні символи або слова.
   - Застосовуються нейронні мережі (наприклад, згорткові мережі для просторового аналізу або рекурентні нейронні мережі, такі як LSTM, для врахування часових залежностей).
   - Модель може бути навченою на спеціально зібраному наборі даних, що містить приклади рухів різних осіб у різних умовах.
6. **Модуль інтеграції результатів та виводу (Output & Integration Module)**:

- Отримує результати класифікації, проводить постобробку (наприклад, згладжування для усунення шумів) та відображає розпізнаний текст.
- Може забезпечувати інтеграцію з іншими системами, наприклад, з мобільним застосунком або веб-інтерфейсом через API.

Взаємодія модулів та обробка в реальному часі

- **Синхронізація потоків даних**: Забезпечення безперебійної передачі даних між модулями, що є особливо важливим для роботи алгоритму в реальному часі.
- **Апаратна оптимізація**: Використання можливостей GPU для паралельної обробки кадрів та виконання обчислювально важких завдань, таких як інференс нейронних мереж.
- **Модульне тестування та налагодження**: Кожен компонент окремо тестується, що дозволяє виявити та усунути помилки на ранніх етапах розробки.

Резюме архітектурного рішення

Запропонована архітектура (на рис.2.7) дозволяє забезпечити гнучкість системи, її мас-штабованість та можливість подальшого вдосконалення. Модульна структура сприяє адаптації під різні умови експлуатації та дає можливість інтегрувати новітні алгоритми для підвищення точності розпізнавання.

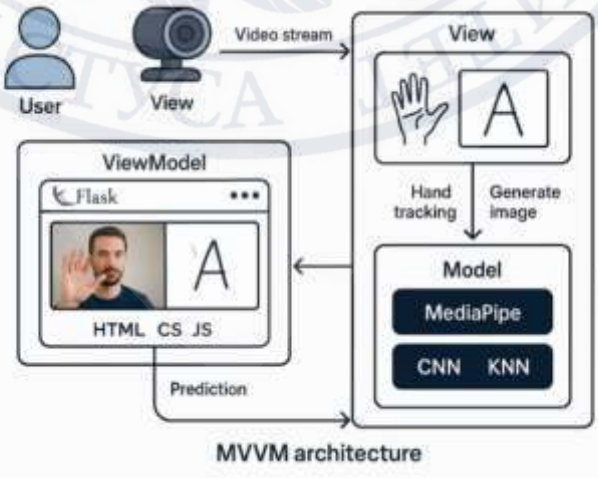


Рисунок 2.7 – **Архітектура**

РОЗДІЛ 3. РЕАЛІЗАЦІЯ АЛГОРИТМУ ІДЕНТИФІКАЦІЇ ТЕКСТУ ЗА РУХАМИ РУКИ

3.1 Розробка блок-схеми алгоритму

На першому етапі було створено детальну блок-схему, яка ілюструє послідовність дій системи від захоплення відеокадру до відображення розпізнаного тексту, зображена на рис. 3.1.

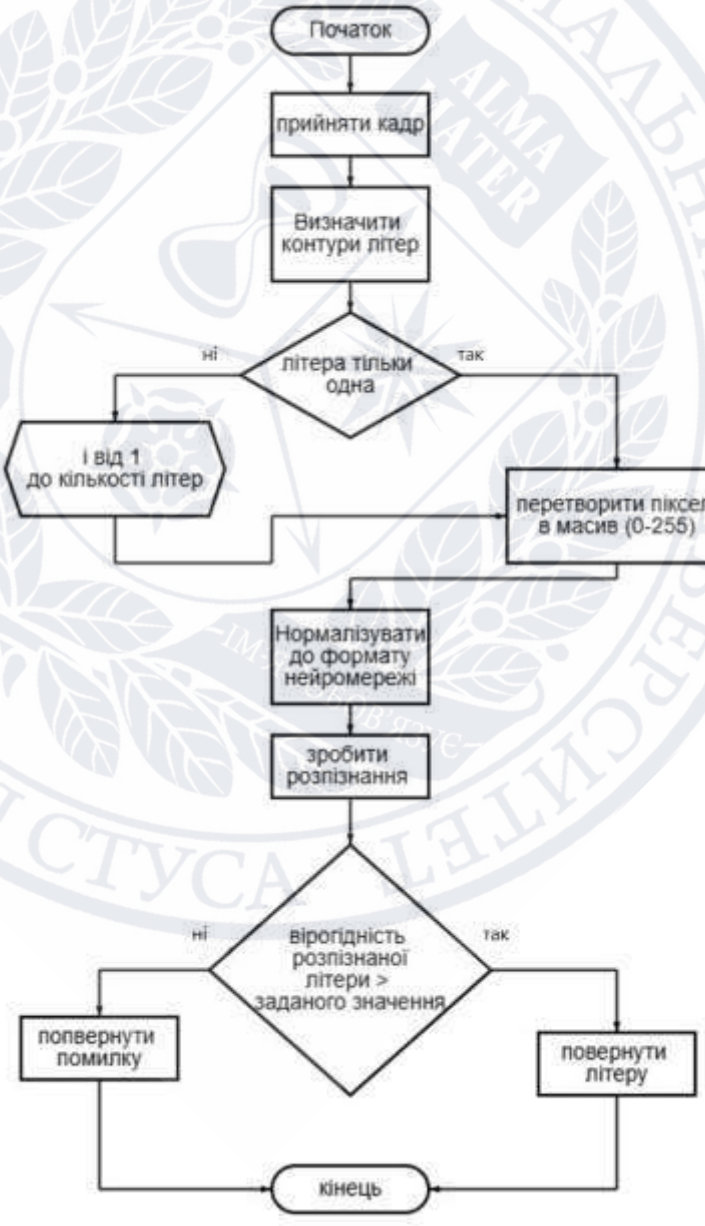


Рисунок 3.1 – Блок-схема алгоритму ідентифікації тексту за рухами руки.

На блок-схемі зображено наступне:

1. Ініціалізація системи:
    - Запуск веб-камери, калібрування області малювання.
    - Створення об’єктів класів AirWritingApp (основний контролер) та DrawingCanvas (віртуальне полотно).
2. Цикл обробки кадрів:
    - Захоплення кадру: використання бібліотеки OpenCV для читання відео потоку.
    - Попередня обробка:
        - Дзеркальне відображення (cv2.flip) для імітації дзеркала.
        - Перетворення у градації сірого (cv2.cvtColor) та бінаризація (cv2.threshold).
        - Видалення шумів за допомогою морфологічної операції ерозії (cv2.erode).
3. Виділення траєкторії руху:
    - Фільтрація контурів за розміром та ієрархією (cv2.findContours).
    - Запис траєкторії на віртуальне полотно.
4. Сегментація символів:
    - Розділення суцільного руху на окремі символи за допомогою аналізу пауз у русі.
    - Обрізка кожного символу з подальшим масштабуванням до 28x28 пікселів.
5. Підготовка даних для нейромережі:
    - Нормалізація пікселів у діапазон [0, 1].
    - Перетворення зображення у 1D масив (784 елементи).
6. Класифікація:
    - Використання попередньо навченої згорткові нейромережі (CNN) для передбачення символу.
    - Відображення результату через інтерфейс користувача.
7. Оновлення інтерфейсу:

- JavaScript-код для періодичного запиту результатів (fetch кожні 5 секунд).
- Відображення проміжних результатів у реальному часі.

Обґрунтування циклічності: система працює в режимі реального часу завдяки асинхронній обробці кадрів та періодичним запитам до сервера. Це дозволяє користувачеві бачити результати без затримок, навіть під час безперервного малювання.

3.2 Реалізація алгоритму обробки сигналів та розпізнавання рухів

3.2.1 Архітектура системи

У цьому проєкті для організації взаємодії між інтерфейсом користувача (View), логікою представлення даних і сервісами розпізнавання жестів застосовано архітектурний паттерн MVVM (Model–View–ViewModel).

Переваги застосування MVVM у даному випадку:

1. **Чітке розділення обов'язків**: UI-код не містить жодної логіки обробки зображень чи розпізнавання — він лише відображає те, що отримав від ViewModel.
2. **Полегшене тестування**: усі ключові сценарії (розпізнавання жестів, нормалізація даних, оновлення тексту) можна протестувати на рівні ViewModel без використання браузера чи камери.
3. **Масштабованість**: при необхідності можна легко замінити частину UI (змінити веб-сторінку на desktop client) або реалізувати нові режими вводу, не змінюючи Model.
4. **Підтримка асинхронності**: MVVM дозволяє красиво вбудувати async-методи обробки кадрів та запитів до нейромережі, не блокуючи відображення та забезпечуючи плавний UX.

Таким чином, впровадження MVVM зробило архітектуру програми стійкішою, зрозумілішою для подальшого розвитку і полегшило інтеграцію різних технологічних шарів (OpenCV → GestureWriter → AirWritingApp → MainPage).

Нижче на рис.3.2 наведено UML-діаграму основних компонентів програми, що ілюструє взаємодію модулів для захоплення, обробки відео потоку та розпізнавання рукописних символів у повітрі:

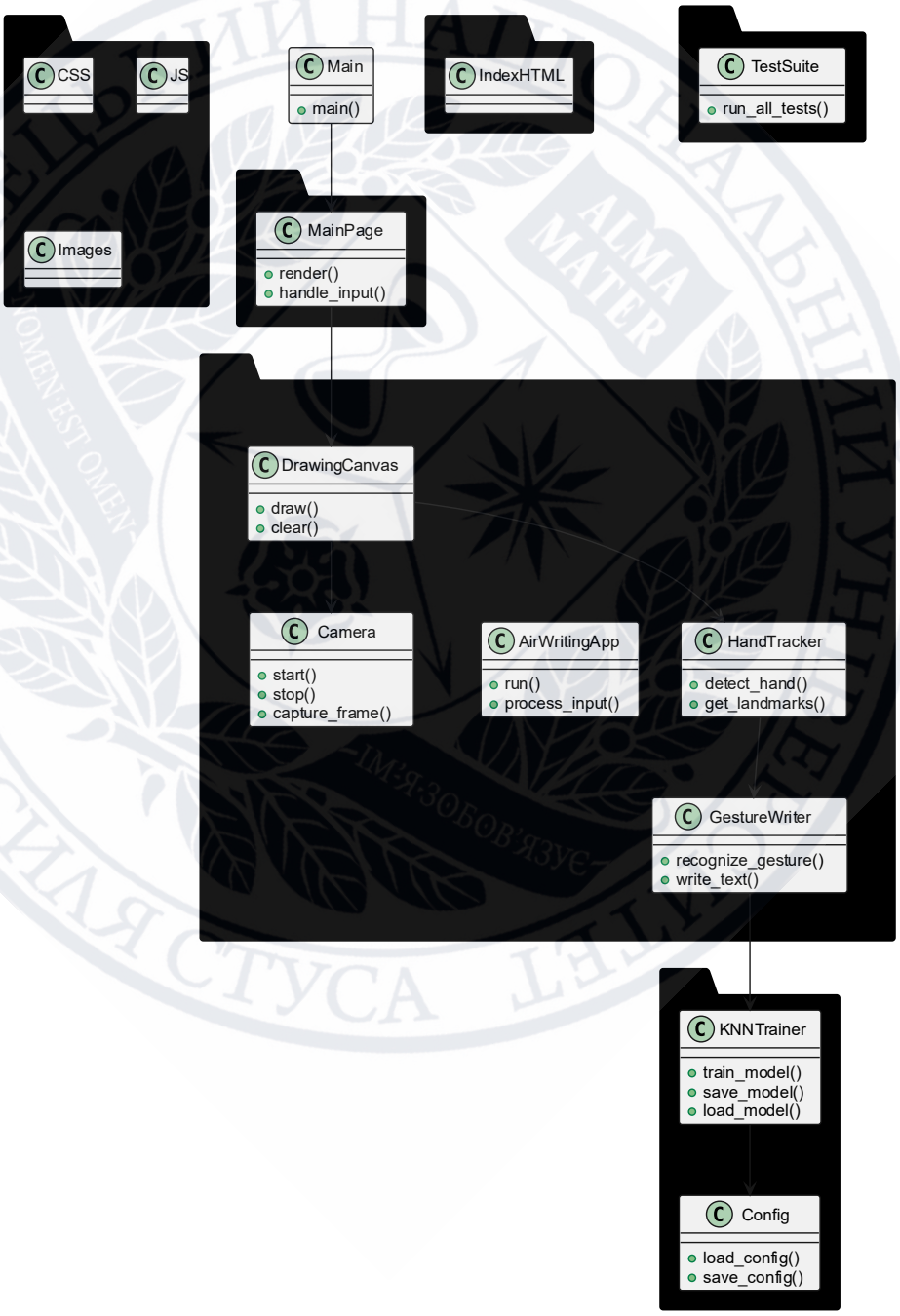


Рисунок 3.2– Архітектура компонентів системи розпізнавання рухів руки (UML-діаграма).

Компоненти діаграми:

- **Camera** — керує підключенням до камери та отриманням відео потоку. Підключення відбувається у функції find_cameras, що наведена нижче.

```
# Функція не використовується - залишена для можливого розширення
функціоналу програми
def find_cameras(max_cameras: int = 2) -> List[Dict[str, str]]:
    cameras = []
    for i in range(max_cameras):
        cap = cv2.VideoCapture(i)
        if cap.isOpened():  камеру
    cameras.append({'id': i, 'name': cap.getBackendName()})
    cap.release()
        else:
    print(f"Не вдалося відкрити камеру з індексом {i}.")  # Виво-
димо повідомлення, якщо камеру не вдалося відкрити
    return cameras  # Повертаємо список знайдених камер
```

- **DrawingCanvas** — реалізує відрисовку руху руки на віртуальному полотні за допомогою малювання крапок та ліній. Функція малювання draw_line наведено нижче:

```
def draw_line(self, x: int, y: int, color: Tuple[int, int, int] =
(255, 255, 255), size: int = 20):
    # Якщо попередні координати існують
    if self.prev_x is not None and self.prev_y is not None:
    # Малюємо лінію
    cv2.line(self.canvas, (self.prev_x, self.prev_y), (x, y),
color, size)
    self.prev_x, self.prev_y = x, y # Оновлюємо попередні коорди-
нати
```

- **HandTracker** — відслідковує положення руки та витягує координати ключових точок (код наведено нижче):

```
def get_finger_position(self, frame: np.ndarray) ->
Optional[Tuple[int, int]]:
    h, w, _ = frame.shape
    # Отримуємо розміри зображення
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    # Перетворюємо зображення у формат RGB
    results = self.hands.process(frame_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
```

```
    index_finger_tip =
hand_landmarks.landmark[self.mp_hands.HandLandmark.INDEX_FINGER_TI
P]
    x, y = int(index_finger_tip.x * w), int(index_finger_tip.y *
h)
    return x, y
    return None
```

- **GestureWriter** — розпізнає жести та перетворює їх у символи або текст за допомогою нейромережі. В цьому класі реалізована логіка, яка приймає дані у вигляді масивів, нормалізує їх – тобто приводить у робочій стан для нейромережі. Далі відбувається обробка наступним чином:

```
def recognize_letter(self, data: List[int]) -> Tuple[str, float]:
    # Прогнозуємо мітку для зображення
    prediction = self.model.predict(self.normalize_image(data))
    predicted_class = np.argmax(prediction)
    # Знаходимо індекс найбільш ймовірного класу
    confidence = np.max(prediction)
    # Впевненість у прогнозі
    return emnist_labels[predicted_class], confidence
    # Повертаємо літеру та впевненість
```

- **AirWritingApp** — основний клас додатку, координує роботу камери, трекінгу та візуалізації. В цьому класі у головній функції run ініціалізовано нескінченний цикл, який щосекундно приймає кадр із камери користувача та передає цей кадр до вищеперерахованих класів, які в свою чергу виконують свою роботу по відображенню руху пальця та повертають результат для подальшого передбачення.
- **MainPage** — відповідає за відображення та логіку головної веб-сторінки таким чином:

```
@app.route('/')
def index() -> str:
    return render_template('index.html')
    # Відображення головної сторінки за допомогою шаблону
```

- **KNNTrainer** — проводить навчання моделі KNN, збереження та завантаження ваг, код створення моделі наведено нижче:

```
# Створюємо модель
    self.model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)),
        # Перший згортковий шар
        MaxPooling2D((2, 2)),
        # Шар підвибірки
        Conv2D(64, (3, 3), activation='relu'),
        # Другий згортковий шар
        MaxPooling2D((2, 2)),
        # Шар підвибірки
        Flatten(),
        # Перетворюємо дані в одномірний вектор
        Dense(128, activation='relu'),
        # Повнозв'язний шар
        Dropout(0.5),
        # Шар регуляризації (відключення половини нейронів)
        Dense(num_classes, activation='softmax')
        # Вихідний шар з кількістю нейронів = кількості класів
    ])
    # Компіляція моделі
    self.model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    print("Починаємо навчання...")
    # Навчаємо модель
    self.model.fit(X_train, y_train_categorical, epochs=10,
batch_size=128, validation_data=(X_test, y_test_categorical))
    # Зберігаємо модель
    self.model.save("emnist_cnn_model.h5")
    print(f"Модель збережено у файл {model_file}")
```

- **Config** — завантажує та зберігає налаштування програми, код наведено нижче:

```
# Імпортуємо модуль os для роботи з файловою системою
import os
# Імпорт dotenv для завантаження змінних оточення
# from dotenv import load_dotenv
# Завантажуємо змінні оточення з .env-файлу
# load_dotenv()

# Отримуємо абсолютний шлях до папки, де знаходиться цей скрипт
parent_dir = os.path.dirname(os.path.abspath(__file__))
# Приклад формування шляху до файлу
resultFile = os.path.join(parent_dir, '', "result.png")
# Формуємо шлях до папки knn для зберігання навчальних даних та
моделі
train_data_folder = os.path.join(parent_dir, '', "knn")
```

3.2.2 Захоплення відео потоку та трекінг траєкторії руки

Першим кроком є безперервне отримання кадрів з камери. У головному контролері програми встановлюється зв’язок із пристроєм захоплення (веб-камера або зовнішня камера), зчитаними ним в режимі реального часу та відображенними кадрами. Відповідний код наведено нижче:

```
ret, frame = self.cap.read()  # Читання кадру з відеопотоку
if not ret:  # Якщо кадр не отримано, виходимо
    break
frame = cv2.flip(frame, 1)  # Дзеркальне відображення зображення
по горизонталі
h, w, _ = frame.shape  # Отримуємо розміри кадру
if self.canvas is None:  # Якщо полотно ще не створено, створюємо
його
    self.canvas = DrawingCanvas(400, 400)
```

Для зручності відображення та більш інтуїтивного керування користувач бачить власні руки дзеркально, тому кожен кадр відразу відображається по горизонталі.

Далі виконується виявлення положення руки на зображенні — у простішій реалізації це може бути відстеження кольору рукавички чи ключових точок (наприклад, кінчиків пальців). В даному ж випадку, як і було зазначено раніше, була використана бібліотека mediapipe, яка має можливість розпізнавати руку, або інші частини тіла людини та фіксувати ключові точки як це зображено на рис.3.3.

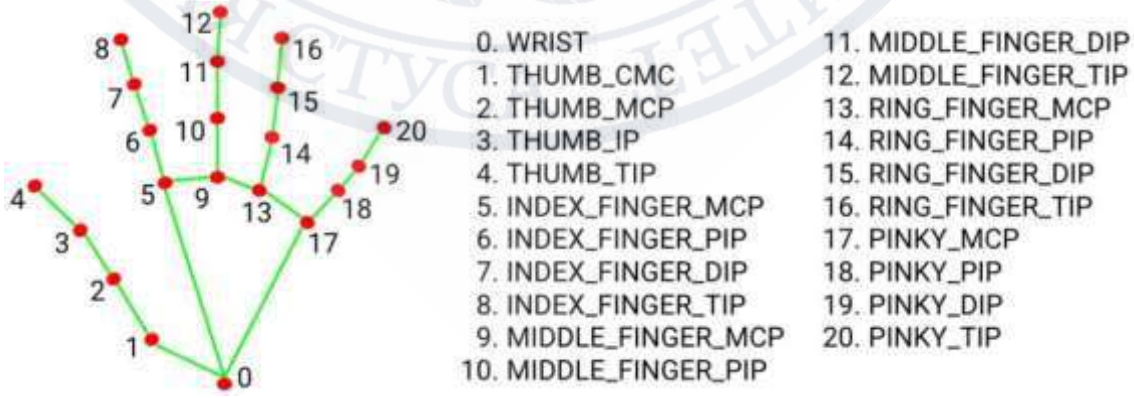


Рисунок 3.3– Ключові точки на лівій руці людини

Пройшовши верифікацію на наявність руки, система намагається розпізнати, чи намагається рука продемонструвати який-небудь жест, наприклад великий палець піднятий догори, що за логікою програми означає *збереження намальованої літери/літер*. Нижче приводиться код для розпізнавання великого пальця піднятого догори

```
def is_thumb_up(self, frame: np.ndarray) -> bool:
    # Обробляємо зображення для пошуку рук
    results = self.hands.process(cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB))
    if results.multi_hand_landmarks:
        # Якщо знайдено руки
        for hand_landmarks in results.multi_hand_landmarks:
self.mp_hands.HAND_CONNECTIONS)
            landmarks = hand_landmarks.landmark
            # Отримуємо координати всіх точок на руці
            thumb_tip =
landmarks[self.mp_hands.HandLandmark.THUMB_TIP]
            thumb_cmc =
landmarks[self.mp_hands.HandLandmark.THUMB_CMC]
            index_tip =
landmarks[self.mp_hands.HandLandmark.INDEX_FINGER_TIP]
            index_mcp =
landmarks[self.mp_hands.HandLandmark.INDEX_FINGER_MCP]
            middle_tip =
landmarks[self.mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
            middle_mcp =
landmarks[self.mp_hands.HandLandmark.MIDDLE_FINGER_MCP]
            ring_tip =
landmarks[self.mp_hands.HandLandmark.RING_FINGER_TIP]
            ring_mcp =
landmarks[self.mp_hands.HandLandmark.RING_FINGER_MCP]
            pinky_tip =
landmarks[self.mp_hands.HandLandmark.PINKY_TIP]
            pinky_mcp =
landmarks[self.mp_hands.HandLandmark.PINKY_MCP]
            # Перевіряємо, чи великий палець вище його основи
            thumb_up = thumb_tip.y < thumb_cmc.y
            # Перевіряємо, чи інші пальці зігнуті
            fingers_folded = (
                index_tip.y > index_mcp.y
                and middle_tip.y > middle_mcp.y
                and ring_tip.y > ring_mcp.y
                and pinky_tip.y > pinky_mcp.y
            )
            if thumb_up and fingers_folded:
                return True
        return False
    return False
```

Якщо ж жодного жесту не розпізнано, це означає, що вмикається режим малювання, наведений нижче:

```
if (not self.tracker.fist_detect(frame)) and self.is_write:  #
Якщо кулак не виявлений і малювання увімкнене
    finger_pos = self.tracker.get_finger_position(frame)  # Отри-
муємо координати вказівного пальця
    if finger_pos:
        # Якщо координати пальця знайдені
        x, y = finger_pos
        # Витягуємо координати
        self.canvas.draw_line(x, y)
        # Малюємо лінію на полотні по координатах
else:
    self.canvas.clear_prev()
    # Якщо кулак виявлений, очищуємо попередні координати
canvas_img = self.canvas.get_canvas()
# Отримуємо зображення полотна
```

Система фіксує координати руки на робочому полотні (DrawingCanvas), перетворюючи їх у відповідні піксельні значення. Таким чином, рух пальця сприймається як *ледве помітний пензель*, що малює в повітрі.

3.2.3 Попередня обробка зображення та зменшення шуму

Оскільки потік відео може містити значні перешкоди — змінне освітлення, тремтіння камери, фоновий рух — перед сегментацією символів необхідно провести низку операцій фільтрації. По-перше, зображення переводять у відтінки сірого, щоб позбавитися від колірної інформації, неважливої для виявлення контуру. Нижче наведено код, що реалізує даний функціонал.

```
def get_single_letter(self)  -> np.ndarray:
    gray_canvas = cv2.cvtColor(self.canvas, cv2.COLOR_BGR2GRAY)  #
Перетворюємо у градації сірого
    resized_canvas = cv2.resize(gray_canvas, (28, 28),
interpolation=cv2.INTER_NEAREST)
    # Змінюємо розмір на 28x28
    flattened_array = resized_canvas.flatten()
# Перетворюємо у 1D масив
    return flattened_array
```

Далі застосовується порогова бінаризація, яка чітко відділяє *малюнок* (темні штрихи) від фону — достатньо буде чітко підібрати значення порогу, щоб помах руки *відображався* насичено.

Щоб видалити дрібні точки шуму, які з’являються внаслідок невеликих коливань камери чи артефактів обчислень, застосуємо морфологічні операції: ерозію для усунення зайвих частинок та, за необхідності, дилатацію для відновлення товщини ліній. В результаті полотно набуває вигляду суцільної чорно-білої маски, де чітко виокремлені області майбутніх символів.

3.2.4 Сегментація контурів та виділення символів

Після фільтрації алгоритм виконує пошук контурів. Для цього використовується метод, який перебирає усі знайдені контури та відкидає вкладені (ті, що є прогалинами всередині інших). Лише зовнішні контури вважаються потенційними літерами.

Кожен зовнішній контур обмежується прямокутником, після чого зображення символу *вирізається* з основного полотна. Оскільки траєкторії букв можуть бути нерівномірними (деякі літери *високі*, інші — *широкі*), для збереження пропорцій обрізані області відцентровують у квадраті, що заповнений білим фоном, та масштабують до стандартного розміру 28×28 пікселів. Це гарантує, що модель отримає однаковий розмір вхідного зображення, незалежно від початкової форми символу. Реалізація функціоналу наведена нижче:

```
def get_several_letters(self) -> List[np.ndarray]:
    gray_canvas = cv2.cvtColor(self.canvas, cv2.COLOR_BGR2GRAY)
    # Перетворюємо у градації сірого
    ret, thresh = cv2.threshold(gray_canvas, 0, 255,
cv2.THRESH_BINARY)
    # Бінаризація
    img_erode = cv2.erode(thresh, np.ones((3, 3), np.uint8),
iterations=1)
    # Видаляємо шуми
    # Отримуємо контури
    contours, hierarchy = cv2.findContours(img_erode,
cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    letters = []
```

```
    for idx, contour in enumerate(contours):
        (x, y, w, h) = cv2.boundingRect(contour)
        # Визначаємо координати та розміри букви
        if hierarchy[0][idx][3] == -1:
            # Перевіряємо, чи це зовнішній контур
            letter_crop = gray_canvas[y:y + h, x:x + w]
            # Вирізаємо букву
            size_max = max(w, h)
            # Визначаємо максимальну сторону квадрата
            # Створюємо квадратне зображення букви
            letter_square = 255 * np.ones(shape=[size_max,
size_max], dtype=np.uint8)

            if w > h: # Якщо буква ширша, ніж вища
                y_pos = size_max // 2 - h // 2
                letter_square[y_pos:y_pos + h, 0:w] = letter_crop
            elif w < h: # Якщо буква вища, ніж ширша
                x_pos = size_max // 2 - w // 2
                letter_square[0:h, x_pos:x_pos + w] = letter_crop
            else: # Якщо буква квадратна
                letter_square = letter_crop
            # Додаємо букву у список (x-координата + саме зобра-
ження)
            letters.append((x, cv2.resize(letter_square, (28, 28),
interpolation=cv2.INTER_AREA)))
    # Сортуємо букви за їх позицією на полотні
    letters.sort(key=lambda x: x[0])
    letters_array = [letter[1] for letter in letters]
    # Масив зображень букв (28x28)
    return letters_array
```

3.2.5 Формування вхідних векторів для моделі

Після сегментації кожне квадратне зображення символу перетворюється на одномірний масив пікселів (flatten). Піксельні значення нормалізуються до інтервалу [0, 1], щоб уникнути впливу абсолютних інтенсивностей на передбачення моделі. Додатково до цього додається вимір *каналу* — у готовому векторі формується *частка* для параметрів розпізнавання, але основний фокус припадає на піксельні інтенсивності. Побачити цей масив можна за допомогою службових функцій, наприклад print_data_array.

```
    s = ','.join(map(str, flattened_array))
    # Готовий масив для подачі у нейромережу
    print(s)
```

3.2.6 Компонент розпізнавання нейромережею

Після перетворення у вектор (або у масив векторів залежно від кількості контурів літер, що були розпізнані) передається у клас для роботи з нейромережею, де сам вектор нормалізується до формату необхідного для нейромереж. Цей код наведено таким чином:

```
def normalize_image(self, data: List[int]) -> np.ndarray:
    image_data = np.array(data)
    # Перетворюємо дані в numpy масив
    image_data = image_data.reshape(28, 28)
    # Перетворюємо в розмір 28x28
    image_data = image_data.astype("float32") / 255.0
    # Нормалізуємо зображення
    image_data = np.expand_dims(image_data, axis=-1)
    # Додаємо розмірність для каналу
    image_data = np.expand_dims(image_data, axis=0)
    # Додаємо розмірність для партії
    return image_data.T
    # Повертаємо транспоноване зображення
```

Нормалізований вектор подається на вхід згорткової нейронної мережі (CNN), побудованої на основі двох згорткових блоків. Архітектура мережі представлена на рис.3.4.

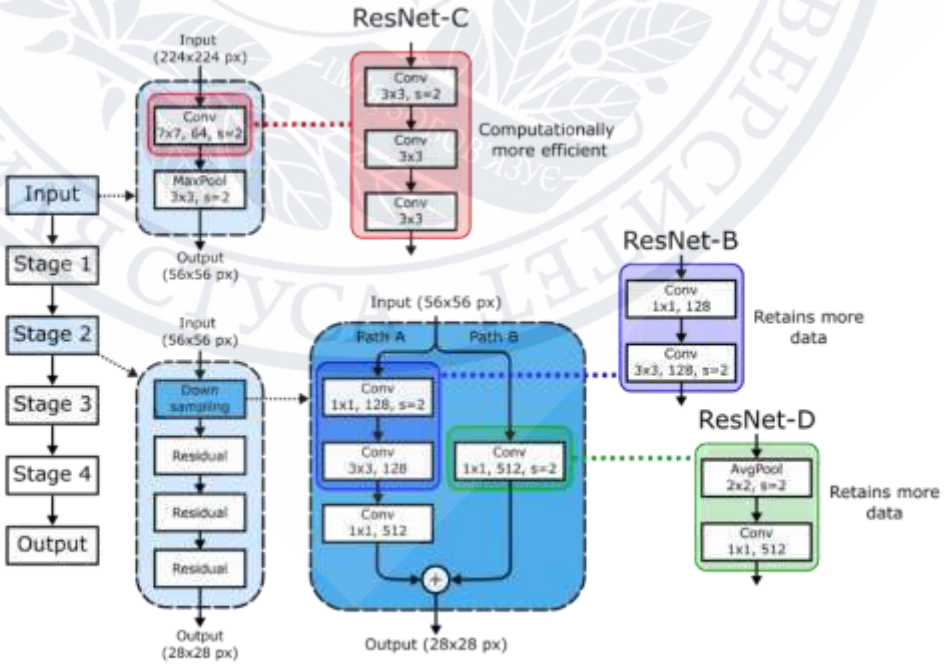


Рисунок 3.4– Архітектура згорткової нейронної мережі

1. Вхідний шар: 28×28×1 (відтінки сірого).
2. Перший згортковий блок:
    - Conv2D (32 фільтри 3×3), активація ReLU;
    - MaxPooling2D (2×2).
3. Другий згортковий блок:
    - Conv2D (64 фільтри 3×3), активація ReLU;
    - MaxPooling2D (2×2).
4. Flatten: перетворення карти ознак у вектор довжиною 3136 елементів.
5. Dense: 128 нейронів, активація ReLU.
6. Dropout: ймовірність 0,5.
7. Вихідний шар: Dense з K нейронами (кількість класів), активація softmax.

Активаційні функції та функції втрат

- ReLU використовується для усунення проблеми затухаючого градієнта. Вона залишає позитивні значення без змін, а всі негативні замінює на нуль. Це допомагає нейронній мережі краще та швидше навчатись.:

$$f(x) = \max(0, x) \tag{3.1}$$

де: f(x)— вихідне значення після застосування активації; x — вхідне значення нейрона.

- Softmax:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, i = 1, \dots, K \tag{3.2}$$

де: $\sigma(z)_i$ — ймовірність належності до класу i; z_i— логіт (вихід перед активацією) для класу i; K — загальна кількість класів.

- Крос-ентропія:

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{3.3}$$

де: L — значення функції втрат; K — кількість класів; y_i — справжнє значення для класу i (у one-hot представленні: 1 або 0); $\hat{y}_i$ — передбачена ймовірність належності до класу i.

Після forward pass мережа генерує вектор ймовірностей розмірністю 62 елементи — по 10 для цифр, 26 для великих та 26 для малих літер. Якщо векторів декілька, вони обробляються послідовно функцією **recognize_letter**.

```
def recognize_letter(self, data: List[int]) -> Tuple[str, float]:
    # Прогнозуємо мітку для зображення
    prediction = self.model.predict(self.normalize_image(data))
    predicted_class = np.argmax(prediction)
    # Знаходимо індекс найбільш ймовірного класу
    confidence = np.max(prediction)
    # Впевненість у прогнозі
    return emnist_labels[predicted_class], confidence
    # Повертаємо літеру та впевненість
```

З вихідного вектору вибирається індекс максимального значення — передбачена мітка. Цей індекс використовують для пошуку відповідного символу у словнику EMNIST:

```
# Словник відповідності індексів символам EMNIST
emnist_labels = {
    **{i: str(i) for i in range(10)},  # 0-9 (цифри)
    **{i + 10: chr(65 + i) for i in range(26)},  # 10-35 (A-Z)
    **{i + 36: chr(97 + i) for i in range(26)}  # 36-61 (a-z)
}
```

Відповідність числових міток та символів у наборі даних EMNIST наступна:

- мітки 0–9 → символи '0'–'9';
- мітки 10–35 → великі літери латиниці ‘A’–‘Z’;
- мітки 36–61 → малі літери латиниці ‘a’–‘z’.

Таким чином, будь-яке передбачення моделі у вигляді числа від 0 до 61 автоматично переводиться в символ, який розуміє людина.

3.2.7 Інтеграція з користувацьким інтерфейсом

Для відображення результатів у режимі реального часу клієнтська частина на JavaScript періодично надсилає запит на серверний endpoint, який повертає останню розпізнану літеру. Отримане значення записується у визначене поле у XHTML-розмітці. Приклад коду наведено нижче:

```
// Отримуємо дані з сервера з інтервалом 5 секунд (можна скоротити
або збільшити)
setInterval(() => {
    fetch('/get_letters', { method: 'POST' })
        .then(response => {
            if (!response.ok) throw new Error(`Помилка сервера:
${response.status}`);
            return response.json(); // Перетворюємо JSON
        })
        .then(data => {
            valueRez.innerText = data.letter; // Відображаємо
отриману букву
        })
        .catch(error => console.error('Помилка:', error));
}, 5000);
```

Такий підхід дозволяє оновлювати відображення без перезавантаження сторінки, забезпечуючи плавний UX. Інтервал запиту (5 с за замовчуванням) можна коригувати залежно від швидкості написання користувачем та параметрів камери.

3.3 Тестування додатку

3.3.1 Тестування інтерфейсу

Запуск програми відбувається на сервері, що має відповідний домен адресу, для тесту, програма була запущена на localhost наступним чином:

- В main було вказано порт host програми як 0.0.0.0 та порт на якому буде працювати програма, 5000, код конфігурації наведено нижче :

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
    # запуск Flask програми на localhost
```

При коректному запуску в консолі відображається інформація як на рис.3.5, де вказано інформацію про завантаження або навчання нейромережі та інформацію щодо host на якому запущена програма.

Рисунок 3.5 – Log запуску програми

- 2) Відкрити будь-який браузер (Chrome, Firefox, Opera або Microsoft Edge) за посиланням, яке вказане в рядку *Running on*, та побачити результат. Зовнішній вигляд додатку зображено на рис.3.6.

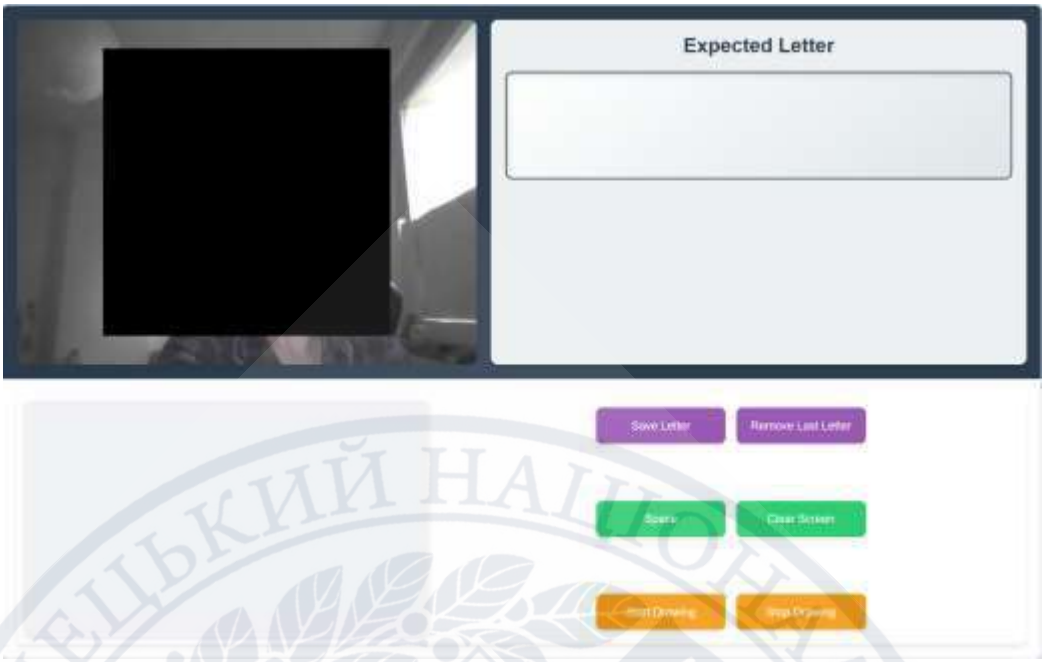


Рисунок 3.6 – Зовнішній вигляд програми

Розпізнавання цифри 4 зображено на рис.3.7.

Рисунок 3.7– Розпізнавання звичайної цифри

Розпізнавання нечітко намальованої літери А зображено на рис.3.8.

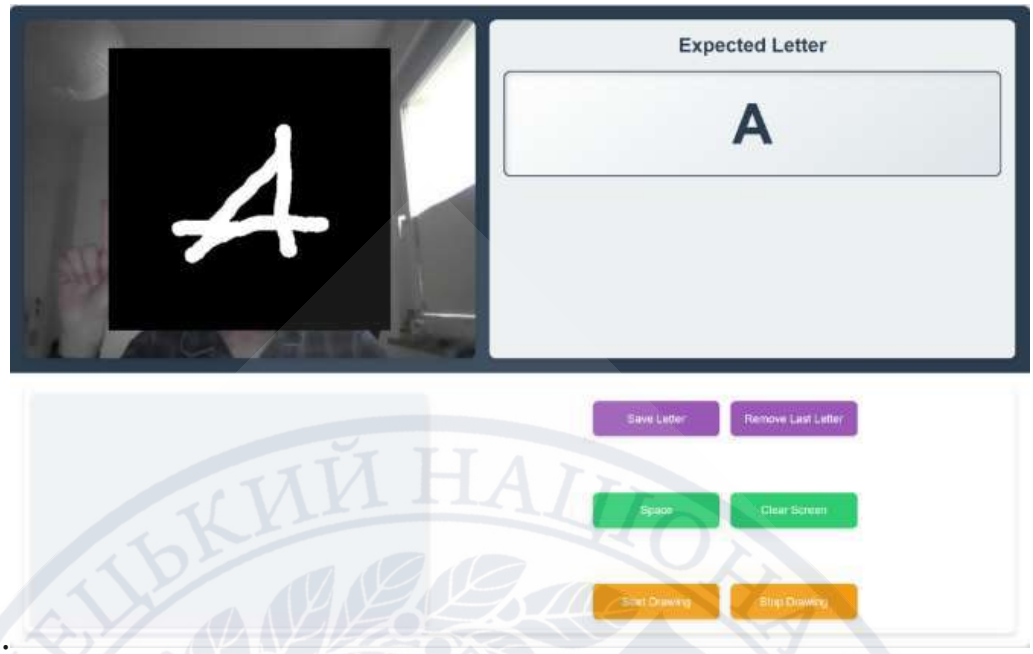


Рисунок 3.8– Розпізнавання нечітко намальованої літери

3.3.2 Тестування, валідація та оцінка ефективності алгоритму

Для оцінки ефективності алгоритму було проведено низку експериментів у контрольованих умовах. Основна мета полягала у визначенні точності розпізнавання, стабільності роботи системи та часу відгуку.

1. **Умови проведення**:
   - **Обладнання**: веб-камера з роздільною здатністю 1280×720 пікселів, персональний комп'ютер з процесором Intel Core i7-11800H та 32 ГБ ОЗП.
   - **Програмне забезпечення**: система реалізована на Python 3.9 з використанням бібліотек OpenCV 4.5.5 та TensorFlow 2.8. Веб-інтерфейс розроблено на Flask 2.0.
   - **Освітлення**: штучне освітлення без різких перепадів яскравості.
   - **Фон**: однорідний світлий фон без рухомих об'єктів.
2. **Протокол тестування**:
   - Для кожного символу (**k**, **j**, **z**, **i**, **Q**) виконано **10 повторень**.
   - Символи малювались послідовно з паузою 2 секунди між спробами.
3. **Критерії оцінки**:
   - **Успішна спроба**: символ розпізнано правильно з першого разу.

- **Невдала спроба**: система повернула неправильний символ або не вивела результат.
- **Точність**: Співвідношення вдало виконаних спроб до невдалих.

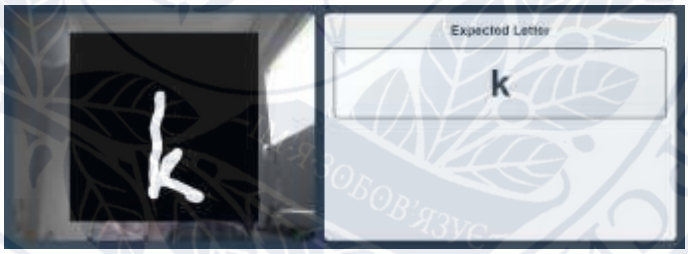
Результати експерименту

Результати тестування представлені у таблиці 3.1:

Таблиця 3.1 Результати тестування

Символ	Успішні спроби	Невдалі спроби	Точність
k	8	2	80%
j	7	3	70%
z	10	0	100%
i	6	4	60%
Q	10	0	100%

Загальна точність: **82%** (41/50). Нижче наведено зображення інтерфейсу під час розпізнавання **k**, **j**, **z**, **i**, **Q** на рис.3.9-3.14



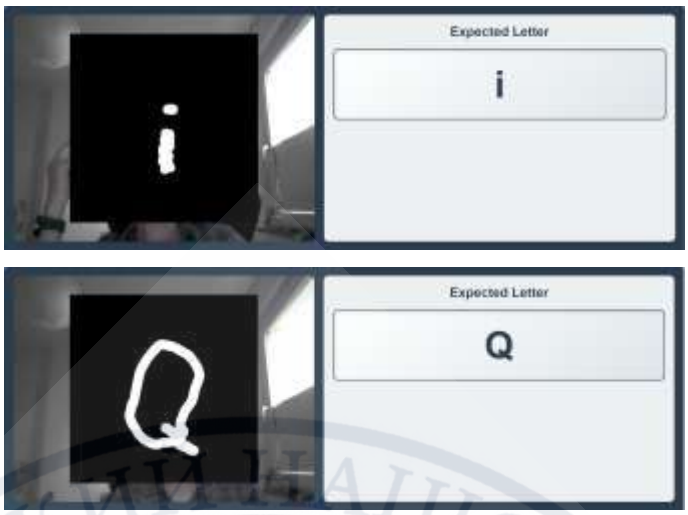


Рисунок 3.9-3.14 – Скріншоти програми, що розпізнає k, j, z, i, Q

Аналіз результатів

1. **Висока точність для символів *z* та *Q* (100%)**:
   - **z**: унікальна зигзагоподібна форма зменшує ймовірність плутанини.
   - **Q**: чіткий замкнутий контур із хвостиком легко ідентифікується.
2. **Помилки для *k* та *j***:
   - **k → r**: нижня петля *k* часто недостатньо розроблялася, що призводило до схожості з *r*.
   - **j → g**: крапка над *j* інтерпретувалась як частина іншого символу через швидкий рух руки.
3. **Низька точність для *i* (60%)**:
   - Вертикальна лінія без крапки система сприймала як *l*.
   - Проблема виникла через недостатню чутливість до дрібних деталей (наприклад, крапки).

Джерела похибок

1. **Особливості почерку**:
   - Нахил, розмір символу та швидкість малювання впливають на якість контурів.
2. **Обмеження бінаризації**:
   - Адаптивний поріг не завжди ефективний при змінній яскравості.
3. **Шум у кадрі**:

- Навіть незначні рухи фону можуть створювати помилкові контури.

У таблиці 3.2 порівнюються результати створеної системи із вже існуючими рішеннями:

Таблиця 3.2 Порівняння з **існуючими рішеннями**

Параметри	**Моя система**	**AirScript [17]**	**RetinaNet ResNet-50 [18]**
Точність	82%	85%	79%
Вимоги до ПЗ	Низькі	Середні	Високі

Переваги системи:

- Оптимізована робота з низькою роздільною здатністю камер.
- Мінімальні затримки завдяки ефективній нормалізації даних.

Пропозиції для оптимізації

1. **Вдосконалення сегментації**:
   - Впровадження алгоритму Watershed для точнішого розділення перекриваючих контурів.
2. **Адаптивна бінаризація**:
   - Використання **Otsu's method** для динамічного вибору порогу яскравості.
3. **Контекстне розпізнавання**:
   - Інтеграція **мовних моделей** для корекції помилок на основі послідовності символів.
4. **Калібрування інтерфейсу**:
   - Додавання інструментів для коригування чутливості до розміру символу та швидкості руху.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведено комплексне дослідження методів ідентифікації тексту за рухами руки, проаналізовано сучасні підходи до розпізнавання жестів та розроблено алгоритм, який реалізує цю функціональність.

Основні результати та висновки:

1. Аналіз існуючих технологій

   Виявлено, що традиційні методи (наприклад, порогова сегментація, виділення ознак HOG/SIFT) ефективні у контрольованих умовах, але мають обмеження при зміні освітлення або фону. Методи глибинного навчання (CNN, LSTM) забезпечують вищу точність завдяки автоматичному аналізу просторових і часових характеристик рухів, що робить їх більш адаптивними до реальних умов.

2. Розробка системи

   Створено модульну архітектуру програмного забезпечення, яка включає:

   - Захоплення та попередню обробку відеопотоку (OpenCV, MediaPipe для трекінгу рук);
   - Сегментацію символів та формування вхідних векторів для нейронної мережі;
   - Класифікацію рухів за допомогою CNN на основі бази даних EMNIST.

3. Експериментальні результати

   Тестування показало загальну точність 82% у розпізнаванні символів (наприклад, *z*, *Q* — 100%, *i* — 60%). Найбільші виклики виникли через шуми, особливості почерку користувача та недосконаліть бінаризації зображення.

4. Оптимізація та перспективи

   Запропоновано шляхи підвищення ефективності:

- Впровадження алгоритмів Watershed та адаптивної бінаризації Otsu’s method для покращення сегментації;
- Використання LSTM для аналізу часових залежностей рухів;
- Інтеграцію контекстного розпізнавання для зменшення кількості помилок.

5. Практична значимість

   Розроблений алгоритм може бути використаний у інтерактивних системах для людей з обмеженими можливостями, в освіті, індустрії додаткової реальності тощо. Його гнучка архітектура дозволяє адаптувати систему до різних умов експлуатації.

Отримані результати підтверджують доцільність використання гібридних підходів, що поєднують класичні методи обробки зображень із глибинним навчанням, для створення ефективних систем ідентифікації тексту за рухами руки. Подальші дослідження слід зосередити на оптимізації продуктивності та розширенні підтримки багатомовних баз даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bayegizova A., Murzabekova G., Ismailova A., Aitimova U., Mukhanova A., Beldeubayeva Z., Ainagulova A., Naizagarayeva A. "Efficiency of using algorithms and methods of artificial technologies for sign language recognition for people with disabilities" // Eastern-European Journal of Enterprise Technologies. – 2022. – Режим доступа: https://doi.org/10.15587/1729-4061.2022.262509

2. Underkoffler: управління комп’ютером за допомогою жестів // Презентация. – 2014. – Режим доступа: https://www.slideserve.com/frisco/underkoffler

3. Коваленко О. В. Дослідження методів розпізнавання жестів та формування системи управління на їх основі [Електронний ресурс] // Харківський національний університет радіоелектроніки. – 2023. – Режим доступу: https://openarchive.nure.ua/server/api/core/bitstreams/bb060daa-a00e-473f-a217-665f798b9f0d/content

4. Олексійко Ю. Р. Розробка програмної системи розпізнавання жестів для взаємодії з комп'ютерними системами [Електронний ресурс] // Тернопільський національний технічний університет імені Івана Пулюя. – 2023. – Режим доступу: https://elartu.tntu.edu.ua/bitstream/lib/44465/1/dyplom_Oleksiyko_Y_2023.pdf

5. Петров П. І. «Методи обробки зображень для розпізнавання жестів». – Київ: Наукова думка, 2019.

6. Іваненко О. С. «Глибокі нейронні мережі в розпізнаванні жестів». – Львів: Видавництво Львівського університету, 2020.

7. Ковальчук М. В. «Аналіз методів розпізнавання рухів руки з використанням CNN-LSTM». – Одеса: Технічна література, 2021.

8. Шевченко А. Л. «Використання глибоких сенсорів у системах розпізнавання жестів». – Харків: Наукова думка, 2022.

9. Сидоренко В. П. «Порівняльний аналіз алгоритмів розпізнавання рухів руки». – Дніпро: Технічна думка, 2023.

10. Петренко О. В. «Сучасні технології розпізнавання тексту». – Київ: Наукова думка, 2020.

11.Смирнов А. П. «Методи оптичного розпізнавання символів». – Харків: Видавництво Харківського університету, 2019.

12.Козак І. О. «Розпізнавання тексту за допомогою глибинного навчання». – Львів: Львівський політехнічний інститут, 2021.

13.Гончаренко М. В. «Порівняльний аналіз систем розпізнавання тексту». – Одеса: Технічна література, 2022.

14.Liu, W. et al. «Scene Text Recognition: Recent Advances and Future Trends». [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/2102.09672, 2020.

15.Балалаєва О. Ю., ЧичкарьовЄ.А., ЗінченкоО.В., Сергієнко А.В., Ковальов О.О. ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДОЛОГІЙ РОЗПІЗНАВАННЯ РУКОПИСНИХ СИМВОЛІВ З ВИКОРИСТАННЯМ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ [Електронний ресурс]. – Режим доступу: https://journals.uran.ua/vestnikpgtu_tech/article/view/299989.

16.Hongyi Liu, Lihui Wang. «Gesture recognition for human-robot collaboration: A review." ». [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0169814117300690

17.Ayushman Dash13, Amit Sahu13, Rajveer Shringi13, John Gamboa4, Muhammad Zeshan Afzal4, Muhammad Imran Malik2, Sheraz Ahmed2 and Andreas Dengel2 «AirScript - Creating Documents in Air» [Електронний ресурс]. – Режим доступу: https://ar5iv.labs.arxiv.org/html/1705.11181

18.Kapitanov Alexander, Kvanchiani Karina, Nagaev Alexander, Kraynov Roman, Makhliarchuk Andrei «HaGRID – HAnd Gesture Recognition Image Dataset" ». [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/2206.08219

ДОДАТКИ

ДОДАТОК А

Клас для роботи з полотном для малювання

```
# Імпортуємо OpenCV для роботи із зображеннями
import cv2
# Імпортуємо NumPy для роботи з масивами
import numpy as np
# Імпортуємо Matplotlib для візуалізації деяких зображень
from matplotlib import pyplot as plt
# Імпортуємо типи для анотації функцій
from typing import List, Dict, Tuple, Set
class DrawingCanvas:
    def __init__(self, width: int, height: int):
        # Створюємо чорне полотно
        self.canvas = np.zeros((width, height, 3), dtype=np.uint8)
        # Координати попередньої точки для малювання лінії
        self.prev_x = None
        self.prev_y = None
    def draw_line(self, x: int, y: int, color: Tuple[int, int,
int] = (255, 255, 255), size: int = 20):
        # Якщо попередні координати існують
        if self.prev_x is not None and self.prev_y is not None:
                # Малюємо лінію
                cv2.line(self.canvas, (self.prev_x, self.prev_y),
(x, y), color, size)
        self.prev_x, self.prev_y = x, y # Оновлюємо попередні ко-
ординати
    def clear(self):
        self.canvas[:] = 0
    def clear_prev(self):
        self.prev_x = None
        self.prev_y = None
    def get_canvas(self) -> np.ndarray:
        return self.canvas
    def get_single_letter(self)  -> np.ndarray:
        gray_canvas = cv2.cvtColor(self.canvas,
cv2.COLOR_BGR2GRAY)  # Перетворюємо у градації сірого
        resized_canvas = cv2.resize(gray_canvas, (28, 28),
interpolation=cv2.INTER_NEAREST)
        # Змінюємо розмір на 28x28
        flattened_array = resized_canvas.flatten()  # Перетворюємо
у 1D масив
        return flattened_array
        def print_data_array(self, flattened_array: np.ndarray):
        s = ','.join(map(str, flattened_array))
        # Готовий масив для подачі у нейромережу
        print(s)

    def get_several_letters(self) -> List[np.ndarray]:
        # Перетворюємо зображення для виділення букв
```

```
        gray_canvas = cv2.cvtColor(self.canvas,
cv2.COLOR_BGR2GRAY)
        # Перетворюємо у градації сірого
        ret, thresh = cv2.threshold(gray_canvas, 0, 255,
cv2.THRESH_BINARY)
        # Бінаризація
        img_erode = cv2.erode(thresh, np.ones((3, 3), np.uint8),
iterations=1)
        # Видаляємо шуми
        # Отримуємо контури
        contours, hierarchy = cv2.findContours(img_erode,
cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        letters = []

        for idx, contour in enumerate(contours):
            (x, y, w, h) = cv2.boundingRect(contour)
            # Визначаємо координати та розміри букви
            if hierarchy[0][idx][3] == -1:
                # Перевіряємо, чи це зовнішній контур
                letter_crop = gray_canvas[y:y + h, x:x + w]
                # Вирізаємо букву
                size_max = max(w, h)
                # Визначаємо максимальну сторону квадрата
                # Створюємо квадратне зображення букви
                letter_square = 255 * np.ones(shape=[size_max,
size_max], dtype=np.uint8)

                if w > h: # Якщо буква ширша, ніж вища
                    y_pos = size_max // 2 - h // 2
                    letter_square[y_pos:y_pos + h, 0:w] =
letter_crop
                elif w < h: # Якщо буква вища, ніж ширша
                    x_pos = size_max // 2 - w // 2
                    letter_square[0:h, x_pos:x_pos + w] =
letter_crop
                else: # Якщо буква квадратна
                    letter_square = letter_crop
                # Додаємо букву у список (x-координата + саме зо-
браження)
                letters.append((x, cv2.resize(letter_square, (28,
28), interpolation=cv2.INTER_AREA)))
        # Сортуємо букви за їх позицією на полотні
        letters.sort(key=lambda x: x[0])
        letters_array = [letter[1] for letter in letters]
        # Масив зображень букв (28x28)
        return letters_array
    def show_leters(self, letters_array: List[np.ndarray]):
        fig, axes = plt.subplots(1, len(letters_array),
figsize=(len(letters_array) * 2, 2))
        if len(letters_array) == 1:  # Якщо одна буква, перетворю-
ємо axes у список
            axes = [axes]
        # Перебираємо букви
```

```
for ax, letter_img in zip(axes, letters_array):
    # Відображаємо зображення у відтінках сірого
    ax.imshow(letter_img, cmap='gray')
    ax.axis("off") # Видаляємо осі
plt.show() # Показуємо результат
```

ДОДАТОК Б

Клас для роботи з нейромережею

```
import os  # Модуль для роботи з операційною системою (файли, шляхи тощо)
import cv2  # Бібліотека для роботи з комп'ютерним зором
import numpy as np  # Бібліотека для роботи з масивами та матрицями даних
import pandas as pd  # Бібліотека для роботи з даними у форматі таблиць
import matplotlib.pyplot as plt  # Модуль для створення графіків і візуалізації даних
from PIL import Image, ImageDraw, ImageFilter  # Бібліотека для роботи із зображеннями
# Модуль для завантаження та створення нейромережевих моделей
from tensorflow.keras.models import Sequential, load_model # type: ignore
# Шари для згорткових нейромереж
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout # type: ignore
# Функція для перетворення міток у категоріальні значення
from tensorflow.keras.utils import to_categorical # type: ignore
from data.config import train_data_folder, emnist_labels  # Шляхи до даних
# Імпортуємо типи для анотації функцій
from typing import List, Tuple
class GestureWriter:
    def __init__(self, training_data_folder: str = None):
        self.model = None # Модель нейромережі (за замовчуванням None)
        self.get_model(training_data_folder)  # Завантажуємо або навчаємо модель
        # self.test_model(training_data_folder)  # Перевірка моделі

    def get_model(self, training_data_folder: str):
        # Шлях до файлу моделі
        model_file = f"{training_data_folder}\\emnist_cnn_model.h5"
        # Перевіряємо, чи існує збережена модель
        if os.path.exists(model_file):
            print("Завантажуємо раніше навчену модель...")
            self.model = load_model(model_file)
# Завантажуємо модель
        else:
            print("Навчаємо нову модель...")
            # Шлях до файлу навчальних даних
            train_file = f"{train_data_folder}\\emnist-byclass-train.csv"
            # Шлях до файлу тестових даних
            test_file = f"{train_data_folder}
```

```
\\emnist-byclass-test.csv"
            model_file = "emnist_cnn_model.h5"
 # Шлях для збереження моделі
            # Завантажуємо дані для навчання
            X_train, y_train = self.load_emnist_data(train_file)
            # Завантажуємо дані для тестування
            X_test, y_test = self.load_emnist_data(test_file)
            num_classes = len(set(y_train)) # Кількість класів
(унікальних міток)
            # Перетворюємо мітки в категоріальні
            y_train_categorical = to_categorical(y_train,
num_classes=num_classes)
            y_test_categorical = to_categorical(y_test,
num_classes=num_classes)
            # Створюємо модель
            self.model = Sequential([
                Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)),
                # Перший згортковий шар
                MaxPooling2D((2, 2)),
                # Шар підвибірки
                Conv2D(64, (3, 3), activation='relu'),
                # Другий згортковий шар
                MaxPooling2D((2, 2)),
                # Шар підвибірки
                Flatten(),
                # Перетворюємо дані в одномірний вектор
                Dense(128, activation='relu'),
                # Повнозв'язний шар
                Dropout(0.5),
                # Шар регуляризації (відключення половини нейро-
нів)
                Dense(num_classes, activation='softmax')
                # Вихідний шар з кількістю нейронів = кількості
класів
            ])
            # Компіляція моделі
            self.model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
            print("Починаємо навчання...")
            # Навчаємо модель
            self.model.fit(X_train, y_train_categorical,
epochs=10, batch_size=128, validation_data=(X_test,
y_test_categorical))
            # Зберігаємо модель
            self.model.save("emnist_cnn_model.h5")
            print(f"Модель збережено у файл {model_file}")

    def test_model(self, training_data_folder: str):
        # Шлях до файлу тестових даних
        test_file = f"{training_data_folder}\\emnist-byclass-
test.csv"
        X_test, y_test = self.load_emnist_data(test_file)
```

```
        # Оцінка моделі на тестових даних
        loss, accuracy = self.model.evaluate(X_test, y_test)
        print(f"Тестовий loss: {loss}, Точність: {accuracy}")

    def load_emnist_data(self, data_path: str) ->
Tuple[np.ndarray, np.ndarray]:
        print(f'Завантаження даних із {data_path}...')
        df = pd.read_csv(data_path, header=None)  # Читання даних
із CSV-файлу
        y = df.iloc[:, 0].values  # Мітки класів (перший стовпець)
        X = df.iloc[:, 1:].values  # Решта стовпців — пікселі
        X = X.reshape(-1, 28, 28, 1).astype("float32") / 255.0  #
Нормалізація
        return X, y

    def show_sample_images(self, X: np.ndarray, y: np.ndarray,
number: int, num_samples: int = 5):
        t = 0  # Лічильник виведених зображень
        for i in range(len(X)):
            if y[i] == number:  # Якщо мітка зображення збігається
із заданою
                image = X[i].reshape(28, 28).T  # Перетворюємо в
28x28 (EMNIST використовує цей розмір)
                plt.imshow(image, cmap='gray')  # Відображаємо зо-
браження
                # Заголовок із міткою
                plt.title(f"Тестове зображення, мітка: {y[i]} -
{emnist_labels[y[i]]}")
                plt.show()  # Показуємо зображення
                t += 1
            if t == num_samples:
                # Якщо виведено потрібну кількість зображень
                break

    def recognize_letter(self, data: List[int]) -> Tuple[str,
float]:
        # Прогнозуємо мітку для зображення
        prediction =
self.model.predict(self.normalize_image(data))
        predicted_class = np.argmax(prediction)
        # Знаходимо індекс найбільш ймовірного класу
        confidence = np.max(prediction)
        # Впевненість у прогнозі
        return emnist_labels[predicted_class], confidence
        # Повертаємо літеру та впевненість

    def normalize_image(self, data: List[int]) -> np.ndarray:
        image_data = np.array(data)
        # Перетворюємо дані в numpy масив
        image_data = image_data.reshape(28, 28)
        # Перетворюємо в розмір 28x28
        image_data = image_data.astype("float32") / 255.0
        # Нормалізуємо зображення
```

```
        image_data = np.expand_dims(image_data, axis=-1)
        # Додаємо розмірність для каналу
        image_data = np.expand_dims(image_data, axis=0)
        # Додаємо розмірність для партії
        return image_data.T
        # Повертаємо транспоноване зображення

    def recognize_letters(self, data: List[List[int]]) ->
Tuple[str, int]:
        s = ''
        for e in data:
            s += self.recognize_letter(e)[0]  # Розпізнаємо кожну
букву та додаємо в рядок
        return s, 0  # Повертаємо рядок і 0, щоб відповідь була
схожа на recognize_letter
```

ДОДАТОК В

Клас для розпізнавання жестів

```
# Бібліотека для роботи із зображеннями та відео
import cv2
# Для роботи з масивами
import numpy as np
# Бібліотека для комп'ютерного зору, включаючи розпізнавання рук
import mediapipe as mp
# Імпортуємо типи для анотації функцій
from typing import List, Dict, Tuple, Set, Optional


class HandTracker:
    def __init__(self, min_detection_confidence: float = 0.8,
min_tracking_confidence: float = 0.8):
        # Використовуємо рішення для розпізнавання рук у MediaPipe
        self.mp_hands = mp.solutions.hands
        self.hands = self.mp_hands.Hands(
            min_detection_confidence=min_detection_confidence,
            min_tracking_confidence=min_tracking_confidence
        )
        self.mp_draw = mp.solutions.drawing_utils # Використовуємо
утиліту для малювання на зображеннях
    def get_finger_position(self, frame: np.ndarray) ->
Optional[Tuple[int, int]]:
        h, w, _ = frame.shape
        # Отримуємо розміри зображення
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        # Перетворюємо зображення у формат RGB
        results = self.hands.process(frame_rgb)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                index_finger_tip =
hand_landmarks.landmark[self.mp_hands.HandLandmark.INDEX_FINGER_TI
P]
                x, y = int(index_finger_tip.x * w),
int(index_finger_tip.y * h)
                return x, y
        return None

    def fist_detect(self, frame: np.ndarray) -> bool:
        # Обробляємо зображення для знаходження рук
        results = self.hands.process(cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB))
        if results.multi_hand_landmarks:  # Якщо знайдено руки
            for hand_landmarks in results.multi_hand_landmarks:
                # self.mp_draw.draw_landmarks(frame,
hand_landmarks, self.mp_hands.HAND_CONNECTIONS)
                tips = [
                    self.mp_hands.HandLandmark.INDEX_FINGER_TIP,
```

```
                    self.mp_hands.HandLandmark.MIDDLE_FINGER_TIP,
                    self.mp_hands.HandLandmark.RING_FINGER_TIP,
                    self.mp_hands.HandLandmark.PINKY_TIP
                ]
                mcps = [
                    self.mp_hands.HandLandmark.INDEX_FINGER_MCP,
                    self.mp_hands.HandLandmark.MIDDLE_FINGER_MCP,
                    self.mp_hands.HandLandmark.RING_FINGER_MCP,
                    self.mp_hands.HandLandmark.PINKY_MCP
                ]

                is_fist = True  # Припускаємо, що це кулак
                for tip, mcp in zip(tips, mcps):  # Для кожного
пальця перевіряємо, чи його кінчик нижче основи
                    if hand_landmarks.landmark[tip].y <
hand_landmarks.landmark[mcp].y:
                        is_fist = False  # Якщо хоча б одна умова
не виконується, це не кулак
                        break

                return is_fist if True else False
        return False

    def is_thumb_up(self, frame: np.ndarray) -> bool:
        # Обробляємо зображення для знаходження рук
        results = self.hands.process(cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB))
        if results.multi_hand_landmarks:
            # Якщо знайдено руки
            for hand_landmarks in results.multi_hand_landmarks:
                # self.mp_draw.draw_landmarks(frame,
hand_landmarks, self.mp_hands.HAND_CONNECTIONS)
                landmarks = hand_landmarks.landmark
                # Отримуємо координати всіх точок на руці
                thumb_tip =
landmarks[self.mp_hands.HandLandmark.THUMB_TIP]
                thumb_cmc =
landmarks[self.mp_hands.HandLandmark.THUMB_CMC]
                index_tip =
landmarks[self.mp_hands.HandLandmark.INDEX_FINGER_TIP]
                index_mcp =
landmarks[self.mp_hands.HandLandmark.INDEX_FINGER_MCP]
                middle_tip =
landmarks[self.mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
                middle_mcp =
landmarks[self.mp_hands.HandLandmark.MIDDLE_FINGER_MCP]
                ring_tip =
landmarks[self.mp_hands.HandLandmark.RING_FINGER_TIP]
                ring_mcp =
landmarks[self.mp_hands.HandLandmark.RING_FINGER_MCP]
                pinky_tip =
landmarks[self.mp_hands.HandLandmark.PINKY_TIP]
                pinky_mcp =
```

```
landmarks[self.mp_hands.HandLandmark.PINKY_MCP]
                # Перевіряємо, чи великий палець вище його основи
                thumb_up = thumb_tip.y < thumb_cmc.y
                # Перевіряємо, чи інші пальці зігнуті
                fingers_folded = (
                    index_tip.y > index_mcp.y
                    and middle_tip.y > middle_mcp.y
                    and ring_tip.y > ring_mcp.y
                    and pinky_tip.y > pinky_mcp.y
                )
                if thumb_up and fingers_folded:
                    return True
            return False
        return False
    def write_text_on_canvas(self, frame: np.ndarray, text: str,
x: int, y: int):
        cv2.putText(frame, text, (x, y), cv2.FONT_HERSHEY_SIMPLEX,
                    2, (0, 255, 0), 2, cv2.LINE_AA)
```

ДОДАТОК Г

Головний клас логіки програми

```
import cv2  # Імпорт бібліотеки OpenCV для обробки зображень та відео
from logic.workWithHand.GestureWriter import GestureWriter  # Імпорт класу для розпізнавання тексту нейромережею
from logic.workWithHand.HandTracker import HandTracker  # Імпорт класу для відстеження положення рук
from logic.canva.DrawingCanvas import DrawingCanvas  # Імпорт класу для малювання на полотні
from data.config import train_data_folder  # Імпорт шляхів до файлів конфігурації

class AirWritingApp:
    def __init__(self):
        self.cap = cv2.VideoCapture(0)  # Відкриття відеопотоку з камери за замовчуванням
        self.tracker = HandTracker()  # Ініціалізація трекера рук
        self.canvas = None  # Змінна для полотна, на якому буде малюватися
        self.writer = GestureWriter(train_data_folder)  # Ініціалізація розпізнавача жестів
        self.is_write = True  # Прапор для включення/виключення режиму малювання

    def set_is_write(self, value: bool):
        self.is_write = value
    def generate_frames(self):
        while True:
            ret, frame = self.cap.read()  # Читання кадру з відеопотоку
            if not ret:  # Якщо кадр не отримано, виходимо
                break
            frame = cv2.flip(frame, 1)  # Дзеркальне відображення зображення по горизонталі
            h, w, _ = frame.shape  # Отримуємо розміри кадру
            if self.canvas is None:  # Якщо полотно ще не створено, створюємо його
                self.canvas = DrawingCanvas(400, 400)

            if (not self.tracker.fist_detect(frame)) and self.is_write:  # Якщо кулак не виявлений і малювання увімкнене
                finger_pos = self.tracker.get_finger_position(frame)  # Отримуємо координати вказівного пальця
                if finger_pos:
                    # Якщо координати пальця знайдені
                    x, y = finger_pos
                    # Витягуємо координати
                    self.canvas.draw_line(x, y)
```

```
                    # Малюємо лінію на полотні по координатах
            else:
                self.canvas.clear_prev()
# Якщо кулак виявлений, очищуємо попередні координати
            canvas_img = self.canvas.get_canvas()
            # Отримуємо зображення полотна
            # Перевіряємо кількість каналів (уникаємо помилки
OpenCV)
            if len(canvas_img.shape) == 2:  # Якщо зображення по-
лотна в градаціях сірого (1 канал)
                canvas_bgr = cv2.cvtColor(canvas_img,
cv2.COLOR_GRAY2BGR)  # Перетворюємо в 3 канали (кольорове зобра-
ження)
            else:
                canvas_bgr = canvas_img  # Якщо зображення вже ко-
льорове, просто використовуємо його

            # Розміри полотна
            ch, cw, _ = canvas_bgr.shape
            x_offset = (w - cw) // 2
            y_offset = (h - ch) // 2
            # Вставляємо полотно по центру
            frame[y_offset:y_offset + ch, x_offset:x_offset + cw]
= canvas_bgr
            # Код для відправки зображення у вигляді потоку
            _, buffer = cv2.imencode('.jpg', frame)  # Кодуємо
кадр у формат JPEG
            frame_bytes = buffer.tobytes()  # Перетворюємо в байти
для відправки по мережі
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' +
frame_bytes + b'\r\n')  # Відправка кадру як частини HTTP-
відповіді
```

ДОДАТОК Ґ

Допоміжні функції

```
from logic.AirWritingApp import AirWritingApp  # Імпорт основного
класу програми для малювання в повітрі
# Імпорт бібліотек Flask для створення веб-додатку
from flask import Flask, Response, jsonify, render_template,
request
# Ініціалізація Flask додатку
app = Flask(__name__, template_folder="../templates",
static_folder="../static")
# Створюємо екземпляр додатку для малювання в повітрі
app_instance = AirWritingApp()

@app.route('/')
def index() -> str:
    return render_template('index.html')
    # Відображення головної сторінки за допомогою шаблону
@app.route('/video_feed')
def video_feed() -> Response:
    # Генерація кадрів і передача їх як відеопотоку
    return Response(app_instance.generate_frames(),
mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/clear_canvas', methods=['POST'])
def clear_canvas() -> str:
    app_instance.canvas.clear()  # Очищаємо полотно
    return '', 204  # Повертаємо порожню відповідь
@app.route('/get_letter', methods=['POST'])
def get_letter() -> str:
    # Виводимо розпізнану букву в консоль
print(app_instance.writer.recognize_letter(app_instance.canvas.get
_single_letter()))
    return '', 204  # Повертаємо порожню відповідь
@app.route('/set_is_write', methods=['POST'])
def set_is_write() -> str:
    is_write = request.get_json().get('is_write', False)  # Отри-
муємо значення прапора з JSON запиту
    app_instance.set_is_write(is_write)  # Встановлюємо прапор ма-
лювання
    return '', 204
@app.route('/get_letters', methods=['POST'])
def get_letters() -> Response:
    if app_instance.canvas is None:
        return jsonify({'error': 'Canvas is not initialized'}),
500  # Повертаємо помилку, якщо полотно не ініціалізовано
    # Отримуємо результат розпізнавання однієї букви
    result, confidence =
app_instance.writer.recognize_letter(app_instance.canvas.get_singl
e_letter())
    # Отримуємо результат розпізнавання кількох букв
```

```
result, confidence =
app_instance.writer.recognize_letters(app_instance.canvas.get_seve
ral_letters())
    return jsonify({'letter': str(result), 'confidence':
float(confidence)})  # Повертаємо JSON з результатом
```

Декларація щодо унікальності текстів роботи
та невикористання матеріалів інших авторів без посилань

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, Бєлкін Євген Володимирович, за спеціальністю 122 „Комп'ютерні науки"

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

02 червня 2025 (дата)

(підпис)