МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра програмування
(повна назва кафедри)

Магістерська робота

Створення застосунку для розпізнавання рукописного тексту та зображень

Виконав: студент групи ПМіМ-21с
спеціальності
122 Комп'ютерні науки
(шифр і назва спеціальності)

Самардак Н.Б.
(підпис) (прізвище та ініціали)

Керівник Пасічник Т.В.
(підпис) (прізвище та ініціали)

Рецензент Мельничук Н.В.
(підпис) (прізвище та ініціали)

ДЕКАН

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
Кафедра програмування
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри Яро Ярошко С.А.
"13" 09 2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Самардак Назар Богданович
(прізвище, ім'я, по батькові)

1 .**Тема роботи** Створення застосунку для розпізнавання рукописного тексту та зображень

керівник роботи Пасічник Тимофій Васильович, кандидат фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від "13" вересня 2022 року №15

2. Строк подання студентом роботи 12 грудня 2022 р.

3. Вихідні дані до роботи

4.Зміст магістерської роботи (перелік питань, які потрібно розробити)
1. Ознайомитись та проаналізувати функціональність та інтерфейси наявних систем і сервісів для розпізнавання тексту та зображень.
2. Розглянути методи здійснення розпізнавання тексту та зображень.
3. Розробити серверну частину програмного забезпечення та описати її функціональність.
4. Розробити прикладний програмний інтерфейс (API) та графічний інтерфейс з використанням одного з фреймворків.
5. **Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)**

6. **Консультанти розділів роботи**

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

7. **Дата видачі завдання 2 вересня 2022 року**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	Вивчення літературних джерел з предмету досліджень	05.09.2022	
2.	Складання розгорнутого плану магістерської роботи	12.09.2022	
3.	Підготовка вступу до роботи	19.09.2022	
4.	Підготовка розділу 1	26.09.2022	
5.	Підготовка розділу 2	05.10.2022	
6.	Програмна реалізація серверної частини	21.10.2022	
7.	Програмна реалізація клієнтської частини	28.10.2022	
8.	Підготовка розділу 3	18.11.2022	
9.	Оформлення роботи	08.12.2022	

Студент _____ **Самардак Н.Б.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Пасічник Т.В.**
(підпис) (прізвище та ініціали)

ЗМІСТ

Вступ 6

1. Аналіз теми 8

1.1. Аналіз актуальності нейронних мереж 8

1.2. Аналіз актуальності використання комп’ютерного бачення 8

1.3. Аналіз архітектури 9

1.4. Аналіз способів та методів розпізнавання тексту 10

1.6. Аналіз даних для навчання 12

1.6.1. Аналіз даних для навчання розпізнавання тексту 12

1.6.2. Аналіз даних для навчання розпізнавання зображень 13

1.7. Аналіз UI фреймворків для реалізації поставленої задачі 13

1.7.1. NET MAUI 13

2. Нейронні мережі 16

2.1. Згорткова нейронна мережа 16

2.2. Навчання нейронних мереж 16

3. Програмна реалізація 18

3.1. Розпізнавання образів 18

3.2. Розпізнавання зображень у файлі 21

3.3. Опис алгоритму навчання розпізнавання зображень 21

3.4. Алгоритм навчання розпізнавання тексту 25

3.5. Результати навчання моделі нейронної мережі 25

3.6. Опис архітектури програми 25

3.7. Опис інтерфейсу програми 28

3.8. Тестування застосунку 29

3.8.1. Тестування на ОС Windows 29

3.9. Результати тестування 35

Висновки 36

Список використаних джерел 37

Додаток А. Клас ImageService 39

Додаток Б. Методи розпізнавання 39

Додаток В. MAUI Program 40

Додаток Г. Модель нейронної мережі для військових та цивільних транспортних засобів 40

Додаток Д. Тренування нейронної мережі для розпізнавання військових та цивільних об'єктів 41

Вступ

Сучасне життя неможливе без програмного забезпечення, доступу до інформації в електронному вигляді, електронних комунікацій тощо. Задля пошуку необхідної інформації для власних потреб люди зазвичай здійснюють пошук у всесвітній мережі інтернет та дізнаються інформацію звідти.

Існують також випадки коли не має необхідних даних в електронних джерелах. Зазначимо, що велика кількість людей досі використовує звичайні рукописні або ж друковані документи, тексти, книги тощо. Інколи робітникам потрібно вручну переписувати об’ємні тексти через відсутність або неможливість отримати електронні джерела. Цей процес займає переважно велику кількість часу та є вірогідність в отриманні незадовільного результату. Загалом тексти та документи можуть містити: рукописні символи, друковані, графіки, картинки, формули, які зрозумілим чином можуть ускладнювати даний процес. Для автоматизації даного процесу існують багато додатків та сервісів, які використовуються на різних платформах та пристроях, що лиш підтверджує актуальність даної проблеми.

Метою даної роботи є розробка застосунку для розпізнавання тексту та зображень. Для цього були використані напрацювання з попередньої курсової роботи, а саме: нейронні мережі та технології комп’ютерного бачення.

Для досягнення мети визначимо такі етапи розроблення, що зображено на рис. 1.

1. Пошук даних для тренування нейронних мереж.
2. Тренування моделей нейронних мереж.
3. Реалізація модулів для виявлення зображень символів.
4. Створення веб-АПІ.

5. Налагодження зв’язку між веб-АПІ та python програмами.
6. Створення користувацького інтерфейсу за допомогою MAUI.
7. Створення зв’язку між веб-API та користувацьким інтерфейсом.
8. Тестування застосунку.

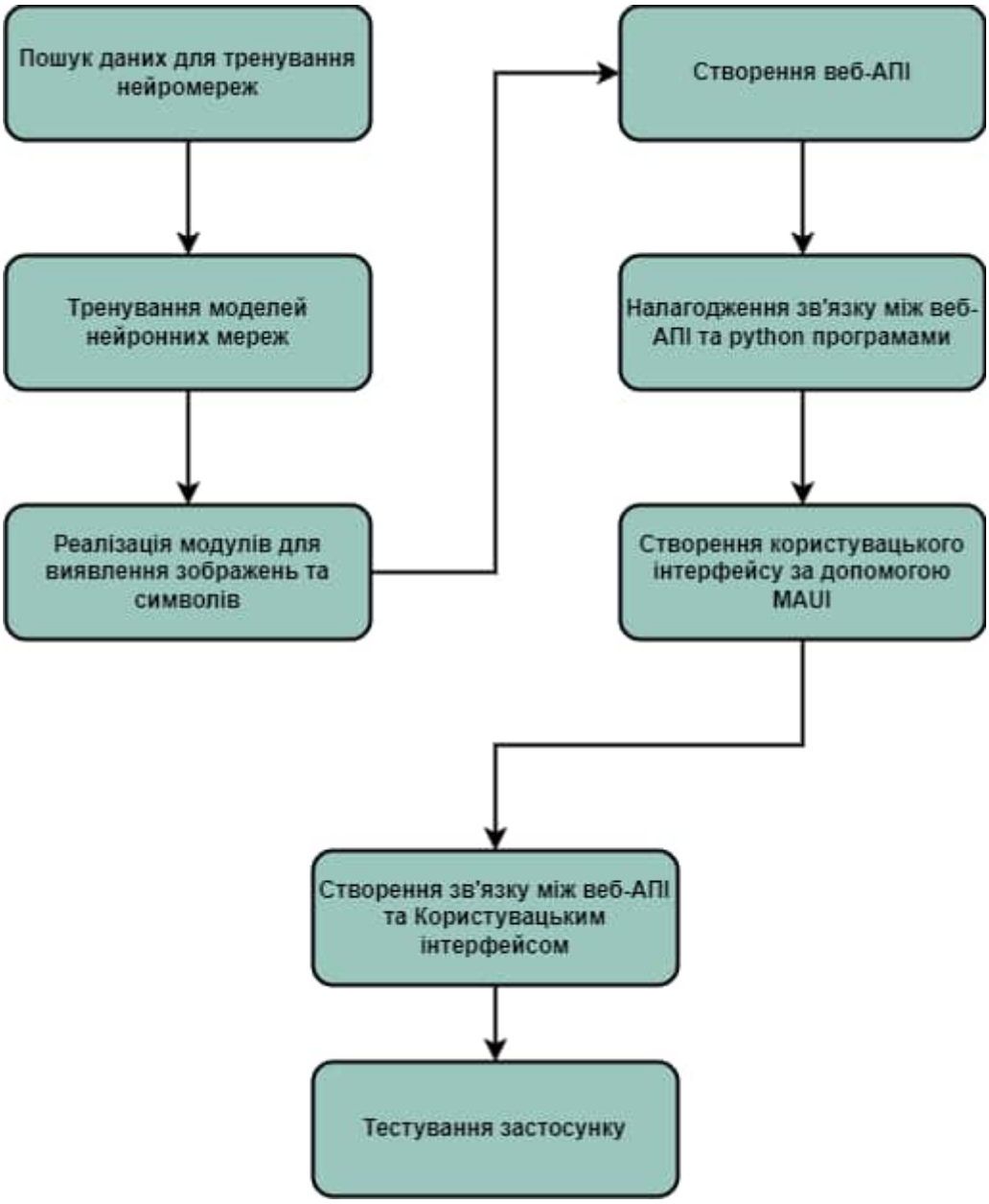

Рис.1. Загальна структурна схема розробки системи.

1. Аналіз теми

1.1. Аналіз актуальності нейронних мереж

У сучасному світі велика кількість завдань може бути розв’язними використовуючи програмні застосунки. Наприклад, ми можемо розв’язати рівняння використовуючи інструмент Wolfram Alpha, побудувати графік функції за допомогою бібліотеки matplotlib або ж дізнатися прогноз погоди використовуючи мобільний застосунок “погода”.

На даний момент, задачі, які колись вважалися нерозв’язними або ж дуже складними для людини, тепер розв’язує без проблем комп’ютер. Однією з таких задач є розпізнавання рукописних текстів, перетворення їх у друковані та класифікація зображень. Для вирішення цих та схожих задач були винайдені штучні нейронні мережі.

Вони знаходять застосування не лише у задачах класифікації та розпізнавання образів, а й у наступних сферах: системи асоціативної пам‘яті, компресія даних, оптимізаційні задачі, теорія керування, розробка нейрокомп‘ютерів, наближення функцій з високою точністю, екстраполяція, прогнозування тощо.

Дані факти наводять на те, що нейронні мережі є актуальними та широко використовуваними у різних сферах.

1.2. Аналіз актуальності використання комп’ютерного бачення

Галузь комп'ютерного зору може бути охарактеризована як молода та різноманітна, не зважаючи на те, що існують ранні роботи, можна визначити, що лише з кінця 1970-х почалось інтенсивне вивчення цієї проблеми. У більшості практичних застосувань комп'ютерного зору, комп'ютери попередньо запрограмовані для вирішення окремих завдань, але методи, що базуються на знаннях, стають все більше узагальненими.

Однією з найбільш важливих сфер, де застосовується комп’ютерне бачення є обробка зображень в медицині. Ця область характеризується отриманням інформації з відеоданих для визначення медичного діагнозу пацієнту. В більшості випадків, відеодані отримують за допомогою мікроскопії, рентгенографії, ангіографії, ультразвукових досліджень та томографії. Прикладом інформації, яка може бути отримана з таких відеоданих є виявлення пухлин, атеросклерозу чи інших злоякісних змін. Також прикладом може слугувати вимірювання розмірів органів, кровообігу тощо. Дана прикладна галузь також сприяє медичним дослідженням, наданням нової інформації, наприклад, про будову мозку чи якості медичного лікування.

Крім медичної сфери комп’ютерний зір використовують у таких галузях як: промисловість, військова галузь, автономні транспортні застосування та в бібліотечній справі для збереження стародруків тощо. Таким чином можна дійти до висновку що використання даної технології є актуальним.

1.3. Аналіз архітектури

Розглянемо визначення API, API (Application Programming Interfaces, APIs) – це готові конструкції мов програмування, які дозволяють розробнику створювати складнішу функціональність з меншими зусиллями. В випадку з розробкою мобільних додатків в ролі API може виступати бібліотека для роботи з “розумним будинком” – вся логіка реалізована в бібліотеці і ви лише звертаєтеся до API в своєму коді. [7]

У випадку веб-додатків, API може повертати дані в форматі відмінному від HTML, завдяки чому ним зручно користуватися при написанні власних програмних застосунків. Частіше всього API повертають дані в одному з двох форматів: XML або JSON. Останній вважають більш лаконічним і простішим в читанні, аніж XML. Крім цього сервіси які надають доступ до даних в XML-

форматі з часом відмовляються від нього. Тому актуальнішим буде використання саме JSON-формату.

Сучасні технології створення веб-застосунків і веб-сервісів надають можливість створювати Веб-API.

1.4. Аналіз способів та методів розпізнавання тексту

Розпізнавання друкованих та рукописних символів різних зображень забезпечує вирішення низки наукових і прикладних задач при ідентифікації об'єктів різної природи. Сучасні методи розпізнавання символів використовуються для вирішення як типових задач, наприклад розпізнавання тексту, так і спеціалізованих завдань, орієнтованих на розпізнавання символьної інформації, нанесеної на поверхню різних об'єктів. В даний час існує досить велика кількість програм, призначених для розпізнавання тексту (наприклад, FineReader, Readiris, ScanSoft OmniPaeg, CuneiForm та ін.). Окрім цього набувають популярності вбудовані програми у операційну систему для розпізнавання тексту, як от наприклад в операційній системі IOS. Проаналізуємо кілька найбільш відомих та розповсюджених алгоритмів розпізнавання тексту.

1.5. Аналіз розпізнавання військових транспортних засобів

Окрім розпізнавання символів чи тексту існують також розпізнавання класів об’єктів. Сучасність вказує на те що необхідним є розпізнавання військових об’єктів, транспортних засобів і т.д. В даний час у відкритому доступі не має офіційних застосунків що використовуються саме для даного типу розпізнавання. Тож за необхідне поставимо задачу розпізнавання військових транспортних засобів.

1. Шаблонні методи.

Першим етапом роботи шаблонного методу є перетворення сканованих зображень в растрове, так зване “поточкове”. Далі проводиться його порівняння з усіма шаблонами, наявними в базі системи та обирають шаблон, який відповідний до даного зображення або ж символу. Найбільш доречним шаблоном вважається той, у якого буде найменша кількість точок, відмінних від вхідного зображення. Шаблон для кожного класу зазвичай отримують шляхом вибору найбільш відповідного зображення символів навчальної вибірки. Слід зазначити, що дані методи визначають, як ті, які мають досить високу точність розпізнавання дефектних символів (склеєних або розірваних).

2. Структурні або топологічні методи

Даний тип розпізнавання тексту зберігає інформацію не про поточне написання символу, а про його топологію. Іншими словами стандарт містить інформацію про відносне розташування окремих компонентів. У процесі роботи алгоритму, який побудований на структурному методі, символ, який необхідно розпізнати проходить етап зменшення (“скелетизації”).

3. Ознакові методи.

Ознакові методи будуються на тому, що картинці ставиться у відповідність вектор ознак. Розпізнавання полягає у порівнянні вектору ознак зображення з набором прикладів векторів тієї ж розмірності. До переваг даного методу відносять те, що він простий у реалізації, у нього добра узагальнююча спроможність, висока швидкість розпізнавання та є стійким до зміни форм символів.

4. Методи на основі навчання

Дані методи направлені не на формалізацію людських знань в певний набір правил, а на виявлення закономірностей і властивостей зображень символів, застосовуючи методи математичної статистики та машинного

навчання. Задача розпізнавання символу відбувається в два етапи. Перший – пошук фрагментів, які є претендентами для класифікації. Другий – розділення знайдених фрагментів на класи. Зображенню ставиться у відповідність вектор ознак, який використовується для класифікації зображень.

1.6. Аналіз даних для навчання

Для навчання нейронної мережі слід підібрати правильну вибірку зображень. В процесі аналізу того, яким чином можна згенерувати дані для тренування нейронної мережі було визначено два основних рішення: підготувати самостійно дані для тренування та розпізнавання або ж знайти готові дані та реалізувати навчання нейронної мережі використовуючи їх. Якщо вибрати перший спосіб, то це може зайняти багато часу, оскільки потрібно згенерувати кілька випадків для кожного символу: пошкоджений, повернутий та різні варіації того, яким може бути символ. Тобто даний спосіб буде дуже довгим та можливо неточним.

Процес пошуку та аналіз даних розгорнуто представлений у попередній курсовій роботі.

1.6.1. Аналіз даних для навчання розпізнавання тексту

Проаналізувавши готові рішення для вирішення даного питання було знайдено базу даних EMNIST (Extended Mixed National Institute of Standards and Technology) – це об'ємна база даних зразків рукописного написання букв англійського алфавіту та цифр. Вона є стандартом, запропонованим Національним інститутом стандартів і технологій США з метою калібрування і зіставлення методів розпізнавання зображень за допомогою машинного навчання, в першу чергу на основі штучних нейронних мереж. [9]

1.6.2. Аналіз даних для навчання розпізнавання зображень

Для розпізнавання зображень було використано набір даних “Military and Civilian Vehicles Classification”. Дана колекція зображень містить класи військових і цивільних транспортних засобів. Набір даних містить загалом 6772 зображень військових вантажівок, військових танків, військових літаків, військових вертольотів, цивільних автомобілів і цивільних літаків у різних розмірах. [2]

1.7. Аналіз UI фреймворків для реалізації поставленої задачі

На даний момент існує велика кількість фреймворків, платформ, застосунків тощо для створення застосунків. Існує великий різновид типів застосунків, головні з яких: веб, мобільні та настільні. Кожен з них має свої операційні системи(якщо говорити про веб та настільні додатки). Оскільки їх є великий різновид, поставимо задачу знаходження найбільш оптимальної платформи задля створення додатку що вирішує поставлену задачу.

Для задач розпізнавання великих текстів та зображень зручніше використовувати настільні застосунки. За даними дослідницький центру П'ю (Вашингтон, США) 73% американців користуються комп’ютерами [3,4]. Також використовуючи настільні додатки є перевага в тому що користувач може користуватися ними без інтернету.

Отже, можна дійти до висновку що використання настільних додатків актуальне.

1.7.1. NET MAUI

Необхідно знайти інструмент задля виконання поставленої задачі а саме створення настільного застосунку.

Проаналізувавши готові рішення, було знайдено “.NET Multi-platform App UI (.NET MAUI)”. Даний фреймворк дозволяє створювати власні програми за допомогою крос платформного набору інструментів інтерфейсу користувача .NET, який націлений на створення мобільних та настільних додатків на Android, iOS, macOS, Windows і Tizen за допомогою C# та XAML. [1]

Використовуючи .NET MAUI, ми можемо розробляти програми, які можуть одночасно працювати на Android, iOS, macOS та Windows з єдиної спільної бази коду. Даний інструмент є відкритим програмним забезпеченням та є еволюцією Xamarin.Forms, розширеною від мобільних до настільних додатків. Працюючи з цим фреймоворком, ми маємо можливість створювати багато платформні програми за допомогою одного проекту, де за потреби можна додати вихідний код і ресурси для певної платформи. Однією з ключових цілей .NET MAUI є можливість реалізувати якомога більшу частину логіки програми та макета інтерфейсу користувача в одній базі коду. Фреймворк об’єднує API Android, iOS, macOS і Windows в єдиний API, що дозволяє розробнику виконувати одноразовий запуск і на будь-якій операційній системі.

.NET MAUI надає єдину структуру для побудови інтерфейсу для мобільних та настільних програм. На рис. 1.6.1 показано високорівневе представлення архітектури програми .NET MAUI:

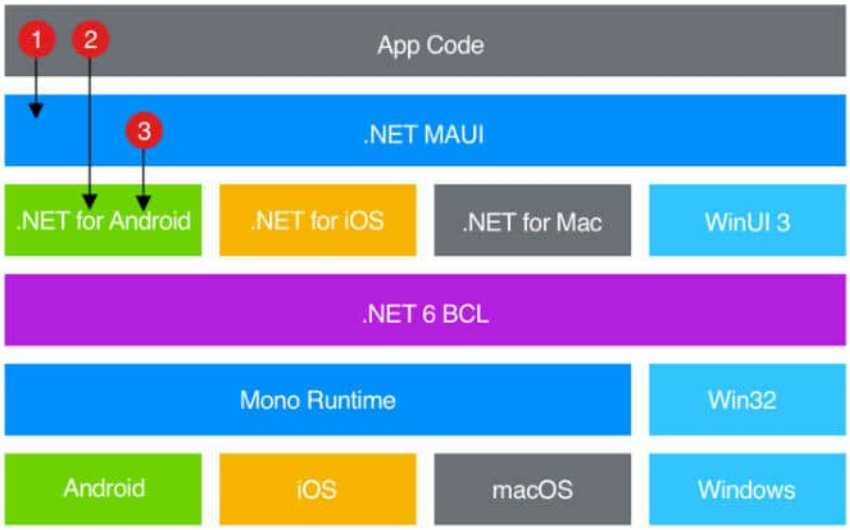

Рис. 1.6.1

Слід також зазначити, що MAUI досі перебуває в процесі розробки і доступний лише у попередньому перегляді, проте існує велика кількість документацій та прикладів як використовувати даний інструмент. Вартує вказати, що попередній перегляд доступний лише з 2022 року.

Проаналізувавши вище наведену дану інформацію та актуальність використання даного інструмента було вирішено що застосунок буде створено за допомогою .NET MAUI.

2. Нейронні мережі

Теорія нейронних мереж розпочала своє існування у 1943 році в результаті спільних спроб фізіологів і кібернетиків зрозуміти і змоделювати роботу мозку. Вони створили обчислювальну модель для нейронних мереж на основі математики та алгоритмів, названою пороговою логікою. Основною ідеєю є представлення складних біологічних процесів математичними моделями.

2.1. Згорткова нейронна мережа

У попередній курсовій роботі було обрано клас згорткових нейронних мереж. Даний клас нейронних мереж є класом глибинних штучних нейронних мереж прямого поширення. Вибір даного класу нейронних мереж буде обґрунтованим, скільки вхідним параметром у даній системі буде зображення, а даний клас працює краще в аналізуванні зображень, аніж інші.

В даному типі нейромережі використовується три види шарів: згортковий шар (convolution), шар субдискретизації (subsampling, pooling) та повнозв’язний шар (fully-connected). Послідовність цих шарів визначає якість роботи ЗНМ. Також для зменшення розмірності зображення застосовують шар пулінгу (субдискретизації). Останнім шаром іде повнозв'язний шар, що формує вихід нейромережі.[8].

Слід зазначити, що до вхідного зображення також застосовується набір фільтрів із заданим розміром вікна (ядра) після чого отримується карта ознак побудована на основі цих фільтрів.

2.2. Навчання нейронних мереж

Сама собою нейронна мережа не варта абсолютно нічого. Для того, щоб вона могла видавати якийсь результат – її потрібно навчити. Процес навчання

нейронної мережі – це процес, в якому її параметри налаштовуються за певним алгоритмом (цей алгоритм називається алгоритмом навчання) таким чином, щоб вона на виході давала правильні результати.

3. Програмна реалізація

3.1. Розпізнавання образів

Кожен день нашого життя полягає у розпізнанні різних образів. Наприклад, коли ми зупинилися, щоб подивитися на колір світлофора, сітківка ока знаходить образ світлофора та у ньому знаходить образи кожного з сигналів та визначає, той який світиться. Після процесу розпізнавання людина приймає рішення чи переходити їй дорогу чи зупинитися зважаючи на те, який колір світиться на світлофорі. Цей приклад наведено використовуючи біологічну складову, проте на даний момент існують також штучні системи розпізнавання образів. Необхідність у розпізнаванні є в різних областях починаючи від військової сфери та закінчуючи повсякденним життям.

У системі штучного розпізнавання образів виділяють такі три методи:

1. Метод перебору – даний метод полягає у порівнянні зображення об’єкта з базою даних, яка містить різні модифікації кожного з зображень, як от: різні кути нахилу, масштаби, зсуви та інші властивості, які можуть визначати відмінності між образами. Для букв потрібно перебирати шрифт, властивості шрифту та інші властивості.

2. Наступний підхід полягає у дослідженні властивостей образів (зв’язність, наявність кутів і т.д.) та знаходженні контуру об’єкта.

3. Та останній, це метод з використанням нейронних мереж. Він потребує великої кількості прикладів задачі розпізнавання з правильними відповідями або ж спеціальної структури нейронної мережі, що враховує специфіку даного завдання.

У багатьох випадках вхідне зображення може мати шуми та об’єкти, які заважають у розпізнаванні символів, тому слід перед тим, як займатися

безпосередньою задачею розпізнавання образів, зайнятися етапом обробки зображення до вигляду у якому даний процес буде швидким та не буде сприймати не потрібні об’єкти, що містяться на вхідному зображенні, як символи. Даний процес слід розбити на певні кроки для структуризації.

Маючи оброблене підготовлене зображення, можна починати сам процес розпізнавання образів. Спершу слід визначити контури кожного з символів у тексті. Контури в реалізації бібліотеки OpenCV можна пояснити кривою, яка з'єднує всі безперервні точки (уздовж країв), що мають однаковий колір або інтенсивність. Також вони є корисним інструмент для аналізу фігур, виявлення та розпізнавання об'єктів. Пошук контурів схожий на шукання білого об’єкта на чорному тлі. Саме для цього попередньо вхідне зображення слід було перевести у бінарний формат, де шукані об’єкти повинні бути білим, а фон – чорним.

Для знаходження контурів скористаємося функцією *cv.findContours*, яка отримує зображення і повертає список всіх контурі. Кожен з окремих є масивом з (x, y) – координат граничних точок об’єкта. При виклику даної функції нам слід задати параметри, які будуть використовуватися при розпізнаванні образів – це режими групування та упаковки зображення. У програмі задамо, що групування буде відбуватися лише по зовнішніх контурах зображення, для цього скористався полем *cv2.RETR_EXTERNAL.* Для упакування використаємо *cv2.CHAIN_APPROX_SIMPLE* – дана властивість дає нам можливість отримати чотири точки, навколо кожного з символів, це нам в майбутньому буде корисним, оскільки саме їх ми будемо використовувати для виділення кожного з символів на зображенні окремо. Також знаходження саме чотирьох точок дає шанс на оптимізацію пам’яті, оскільки інші властивості, які використовуються для упакування зображення можуть повертати всі точки, тобто це може бути дуже велика кількість точок.

Для того щоб обвести контури окремими квадратами скористаємося методом *cv2.boundingRect*, яка приймає контур і на виході повертає точку центру даного контуру та ширину з висотою[10]. Отримавши дані значення пройшлися по всіх знайдених контурах і для них використали функцію. Після цього використали чотири знайдених значення для побудови прямокутника навколо кожного з символів. Для цього скористалися методом *cv2.rectangle*. Отримавши побудовані прямокутники навколо кожного з символів збережемо їх у зображенні. Результат роботи програми можна побачити на рис. 3.1.2.а.

Рис. 3.1.2.а. – розпізнавання символів.

Отже, ми отримали зображення на якому кожен з символів обведено контуром, тепер спробуємо отримати кожен з них та зберегти окремо один від одного.

Для виконання вище наведеного процесу для кожного з символів отриманих за допомогою знаходження контурів переведу кожен з цих символів у квадрат розміру 28*28 оскільки в процесі пошуку даних для навчання нейронної мережі було вирішено використовувати базу даних EMNIST, а зображення у ній містяться саме у такому розширенні. Для цього спершу знаходимо те, яка сторона символу(ширина або висота) є більшою і відповідно до цього будемо робити рішення чи потрібно дане зображення робити ширшим або ж навпаки довшим. Даний процес будемо робити за допомогою так званого кадрування.
Перед самим кадрування створимо масив який буде містити лише білі

пікселі, тобто заповнимо його значеннями *rgb(255,255,255)*. Дане “пусте” зображення використовуватиметься для зберігання контурів кожного з символів окремо.

Кадрування буде проводитися по знайдених контурах, оскільки нам не потрібно навантажувати систему або ж займатися обчисленнями у використовуючи білий фон (у нашому випадку). Пройшовшись ітерацією по кожному з контурів і провівши процес кадрування збережемо кожен з символів окремо. Даний процес можна побачити на зображенні рис. 3.1.2.б.

Рис. 3.1.2.б. – окремі символи.

У подальшому дана програма буде використовуватися для того щоб отримати масив символів з зображення і використовуючи нейронну мережу отримувати з кожного зображення об’єкта текстовий вміст.

3.2. Розпізнавання зображень у файлі

Перед тим як займатися безпосереднім розпізнаванням окремих властивостей тексту та зображень було створено модулі для обробки вхідних файлів. Дані процеси були реалізовані за допомогою бібліотеки OpenCV, що використовується для алгоритмів комп’ютерного зору та алгоритмів обробки зображень. В результаті було отримано кожне з вкладених зображень окремо.

3.3. Опис алгоритму навчання розпізнавання зображень

Для розпізнавання зображень було обрано модель згорткової нейронної мережі (CNN). Для програмної реалізації використаємо зокрема такі бібліотеки як TensorFlow та Keras.

Tensorflow – відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди. [5]

Keras – відкрита нейромережна бібліотека. Вона здатна працювати поверх TensorFlow.[6]

Також, опишемо процес створення моделі, яка буде використуватися при навчанні нейронних мереж для розпізнавання зображень.

Архітектура передбачає укладання згорткових шарів (convolution) з невеликими фільтрами 3×3 з подальшим максимальним шаром субдискретизації (pooling). Разом ці шари утворюють блок, і ці блоки можна повторювати, де кількість фільтрів у кожному блоці збільшується із збільшенням глибини мережі, наприклад 32, 64, 128, 256 для перших чотирьох блоків моделі. Заповнення використовується на згорткових шарах, щоб забезпечити, щоб висота і ширина вихідних карт відповідала вхідним.

В результаті наша модель повинна виглядати наступним чином:

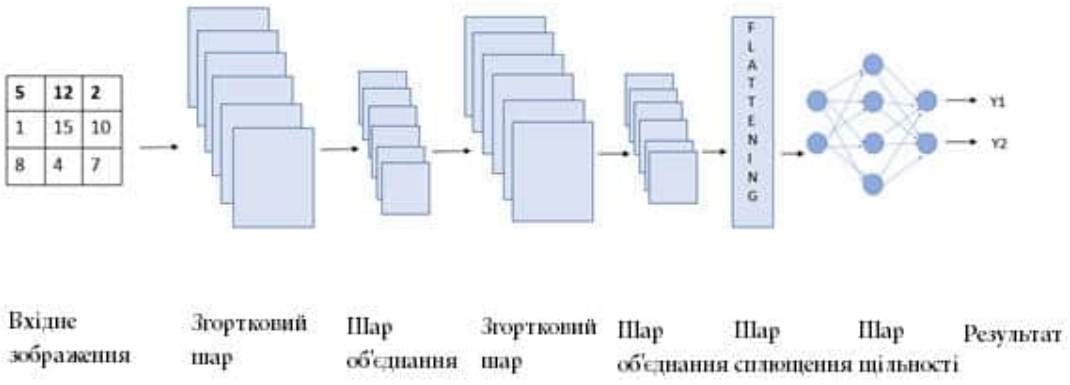

Рис. 3.3.1.

Модель CNN працює в три етапи. На першому етапі згортковий шар (convolution) витягує особливості зображення / даних. На другому етапі шар об'єднання (pooling) зменшує розмірність зображення, тому невеликі зміни

не створюють великих змін на моделі. Простіше кажучи, це запобігає надмірній обробці. На третьому етапі шар сплющення (flattening) перетворює нашу модель в одновимірну і подає її до повністю зв’язаного щільного шару. Потім цей щільний шар (dense) виконує прогнозування зображення. Хороша модель має кілька згорткових шарів та шарів об'єднання.

Ми використовуємо згорткову нейронну мережу (CNN), тому будемо використовувати згортковий шар (convolution). Найпоширенішим є Conv2D. Conv2D означає, що згортка відбувається по 2 осі. Він поширює згортку на трьох шарах – червоний, зелений та синій. Conv2D зазвичай використовується для "зображень".

Розглянемо детальніше параметри які передаються в функцію Conv2D. Перший параметр - «фільтри». Це значення вказує кількість фільтрів, з яких навчатимуться модель CNN та згортковий шар. З кожного такого фільтра згортковий шар дізнається щось про зображення, наприклад, відтінок, межу, форму або ж певну особливість. Значення параметрів повинно мати силу 2.

Другий параметр - “розмір ядра”. Ядро означає фільтр, який рухатиметься по зображенню та витягуватиме деталі. Розмір ядра означає розмірність (висота x ширина) цього фільтра. Тут ми використовували розмір ядра 3, що означає розмір фільтра 3 x 3.

Наступний параметр - «заповнення». Тут ми використаємо значення за замовчуванням “valid”. Він вказує що вхідними даними будуть лише валідні дані.

В процесі аналізу тих функцій, які можна використовувати як активаційні було знайдено функцію *relu* (Rectified Linear Unit).

Окрім цього наша модель містить об’єднуючий шар (pooling). Він використовується для зменшення розміру зображення, а також для збереження важливих параметрів. Таким чином це допомагає зменшити обчислення в моделі. Виконуючи згортку, згортковий шар зберігає

інформацію про точне положення об'єкта. Завдяки Max Pooling ми звужуємо область застосування та всі функції, де враховуються лише найважливіші особливості. Об’єднання здійснюється двома способами: Середнє об’єднання або Максимальне об’єднання. Ми будемо використовувати Max Pooling.[12]

Останнім шаром є шар вирівнювання (flattening). Він додається після набору згорткових шарів та шарів об'єднання. Шар вирівнювання або згладжування перетворює вектор 3d-зображення в 1d. Після шару вирівнювання є шар щільності.

Щільний шар – це повністю зв’язаний шар, який подає всі вихідні результати попереднього функціонування до всіх нейронів. Щільний шар має вагу W, ухил B і активацію, яка передається кожному елементу. Цей шар використовує всі функції, вилучені раніше та виконує роботу з навчання моделі.

Тепер, щоб запобігти надмірній обробці, додається шар виключення (dropout). Під час навчання даних деякі нейрони відключаються випадковим чином. Значення, яке передається нейронам, означає, яку частку нейрона необхідно видалити під час ітерації. Таким чином, після тренування нейрони не впливають значимо на ваги інших нейронів. В результаті чого модель може бути краще узагальненою. [11]

Нарешті ми дійшли до результуючого шару. На виході шар використовує кількість одиниць відповідно до кількості класів у наборі даних. Тут ми використовуємо 6, оскільки є 6 класів. Крім цього ми використовуємо активацію SOFTMAX, оскільки вона дає ймовірності для кожного класу.

Після тренування нейронної мережі було отримано точність 76% що є досить хорошим результатом. Також натреновану модель було збережено

для її подальшого використання програмою. Код моделі знаходиться у додатку Г.

3.4. Алгоритм навчання розпізнавання тексту

У попередній курсовій роботі було розглянуто детально алгоритм навчання розпізнавання тексту. Для програмної реалізації також було використано такі бібліотеки як TensorFlow та Keras.

3.5. Результати навчання моделі нейронної мережі

Після тренування нейронної мережі для розпізнавання символів було отримано точність 87% що є дуже хорошим результатом, а для розпізнавання зображень відповідно 76%. Результати тренування наведені у додатку Д.

3.6. Опис архітектури програми

В процесі аналізу архітектури та визначення етапів розроблення застосунку було обрано необхідність створення веб-API та застосунку який буде використовувати дане веб-API.

Моделі розпізнавання символів та зображень були написані на мові python. Для зв’язку між веб-API було створено клас *ScriptRunner*, який має метод *RunFromCmd*, що приймає в якості параметрів шлях до python програми та шлях до файлу.

Наступним кроком було додано проміжний модуль для приймання аргументів з веб-API і передавання їх в методи що написані на мові python. Для цього з кожного метода були написані класи що приймають аргументи з консолі та повертають результат, який в подальшому отримується C# класом та виводиться в результаті у веб-API.

На стороні веб-API було створено контролер *RecognitionController,* що містить такі методи: Get(), PredictText, PredictImage, GetImages, GetImageClasses, GetImageClassesSymbols. PredictText, PredictImage, GetImages, GetImageClasses, GetImageClassesSymbols – кожен з даних методів приймає в якості параметра *IFormCollection,* який являє собою значення проаналізованої форми, надісланої за допомогою HttpRequest.

- Get() – повертає кількість файлів які були створені в процесі розпізнавання одним з методів.
- PredictText – даний метод розпізнає текст та у результаті повертає текст що міститься у зображенні.
- PredictImage – даний метод розпізнає зображення та у результаті повертає клас вхідного зображення.
- GetImages – даний метод у результаті повертає список з вкладених зображень та зберігає кожен з них окремо у файловій системі.
- GetImageClasses – даний метод розпізнає зображення та у результаті повертає список з вкладених зображень з розпізнаними класами. Також він зберігає кожен з розпізнаних вкладених зображень окремо у файловій системі.
- GetImageClassesSymbols – даний метод по суті є узагальненням всієї програми. Він розпізнає зображення та символи у вхідному файлі. У результаті повертає список з вкладених зображень з розпізнаними класами та розпізнаними символами. Також він зберігає кожен з розпізнаних вкладених зображень окремо у файловій системі.

Кожен з даних методів в результаті повертає користувачу текст з інформацією про розпізнавання.

На MAUI стороні було додано сервіс, який за допомогою *HTTPClient* надсилає запити у попередньо створену веб-API. Для цього у класі *ImageService* було створено метод *sendRequest,* який приймає у якості аргументів метод який ми намагаємося викликати з веб-API, шлях до файлу, що був вибраний користувачем та назву файлу. Далі відбувається надсилання запиту у веб-API з вмістом зображення та отримання результату. Реалізацію додано у додаток А.

Окрім цього було створено декілька сторінок для відображення необхідних даних та можливості роботи з ними кінцевим користувачем, а саме: “Головна сторінка”, “Розпізнавання тексту”, “Розпізнавання зображень”, “Класи та символи”, “Список Зображень”. Вікно меню зображено на рис. 3.4.1.

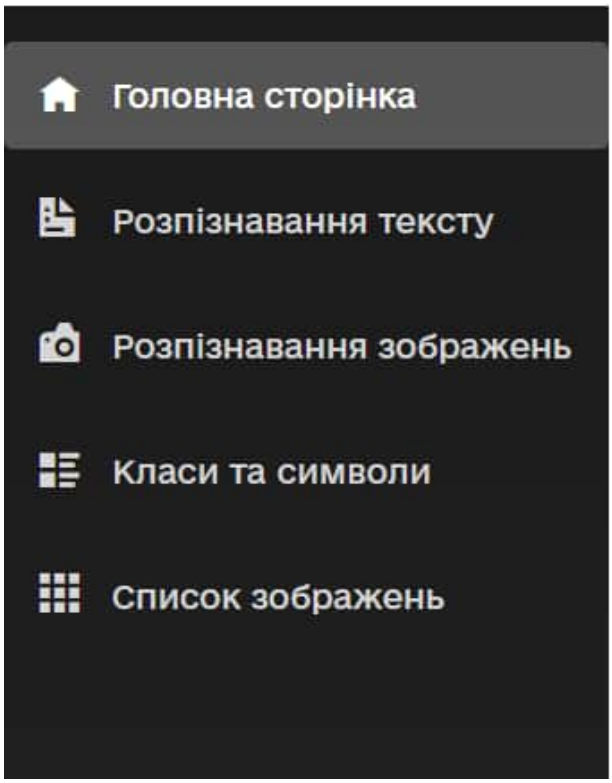

Рис. 3.6.1.

При переході на сторінку “Розпізнавання тексту” та обирання зображення відбувається запит до методу веб-API під назвою *PredictText*; на сторінку “Розпізнавання зображень” та обирання зображення відбувається запит до методу веб-API під назвою *PredictImage*; на сторінку “Класи та символи” та обирання зображення відбувається запит до методу веб-API під назвою *GetImageClassesSymbols*.
Методи що надсилають запити з MAUI застосунку наведені у додатку Б. Загалом архітектура проєкту виглядає так як зображено на рис.3.4.2.

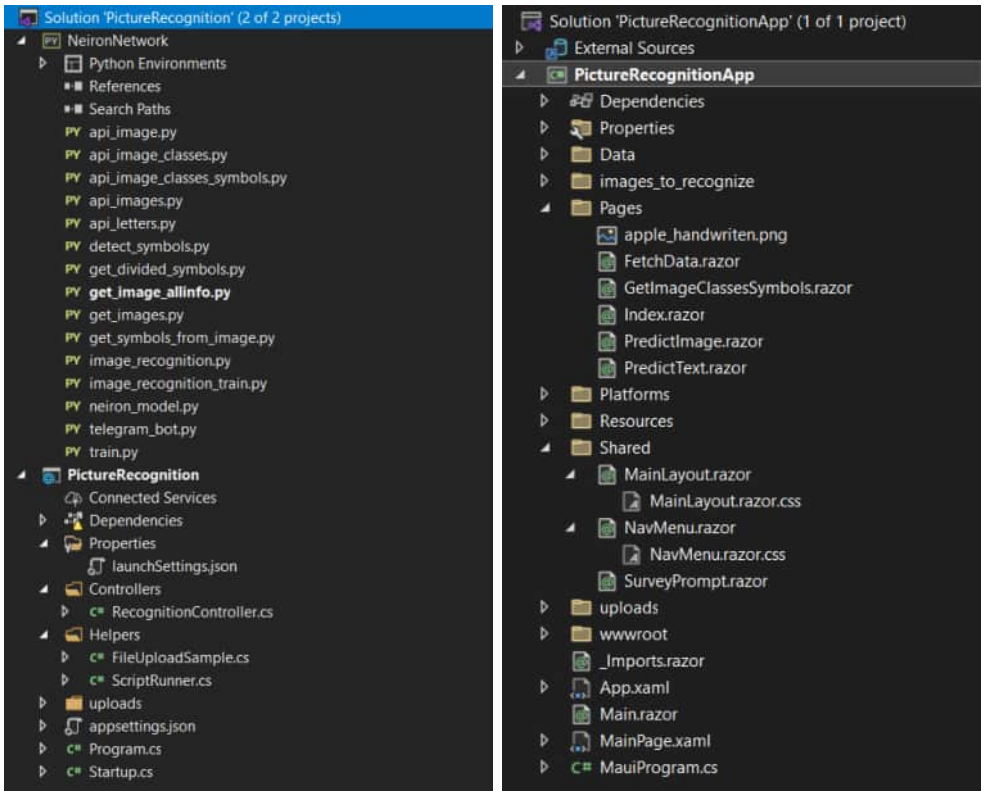

Рис. 3.6.2.

3.7. Опис інтерфейсу програми

У попередній курсовій роботі було створено Веб-API. Програми даного типу не містять жодного користувацького компоненту, оскільки побудовані як набір *http* запитів.
Отже, задля відображення інтерфейсу було створено програмний застосунок використовуючи .NET MAUI Blazor app. У **додатку** В зображено головний

клас та метод для створення застосунків даного типу. Цей клас використовується як вхідна точка для кожної платформи що запускає застосунок.

В результаті було створено кілька сторінок для відображення результатів роботи, які наведені на рис. 3.4.1.

Слід зазначити, що для того щоб працював UI застосунок необхідно щоб був запущена веб-API, тобто існував її IIS процес. Результати тестувань застосунку наведені у наступному розділі.

3.8. Тестування застосунку

Маючи попередньо натреновані нейронні мережі, зв’язок між веб-API і Python програмами та UI застосунок – проведемо тестування.

3.8.1. Тестування на ОС Windows

Протестуємо застосунок використовуючи Windows OS. При запуску відкривається вікно “Головна сторінка”:

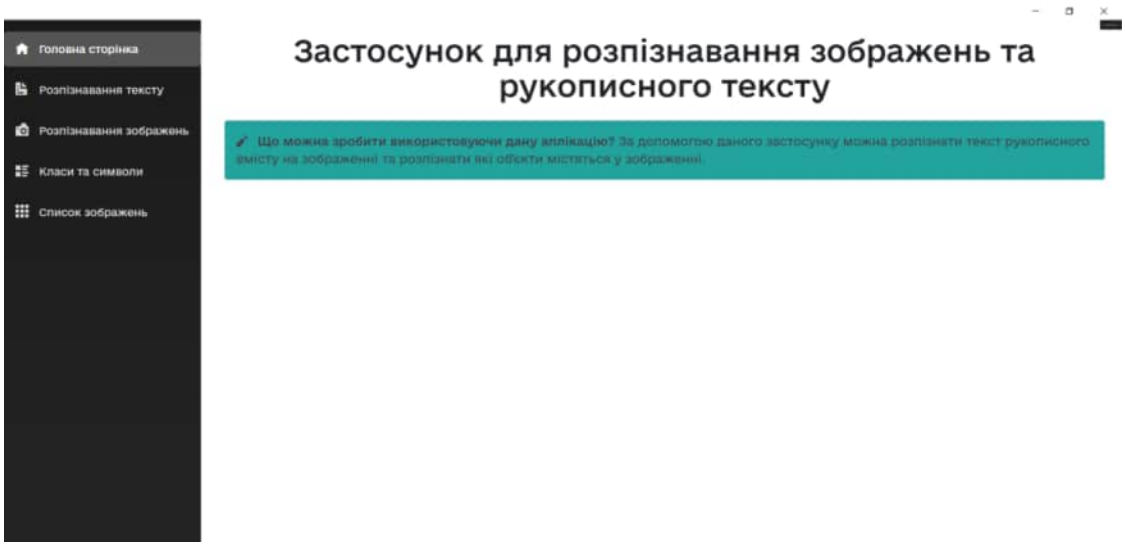

Рис. 3.8.1.

Перейдемо на сторінку “Розпізнавання тексту”:

Рис. 3.8.2.

Завантажимо файл:

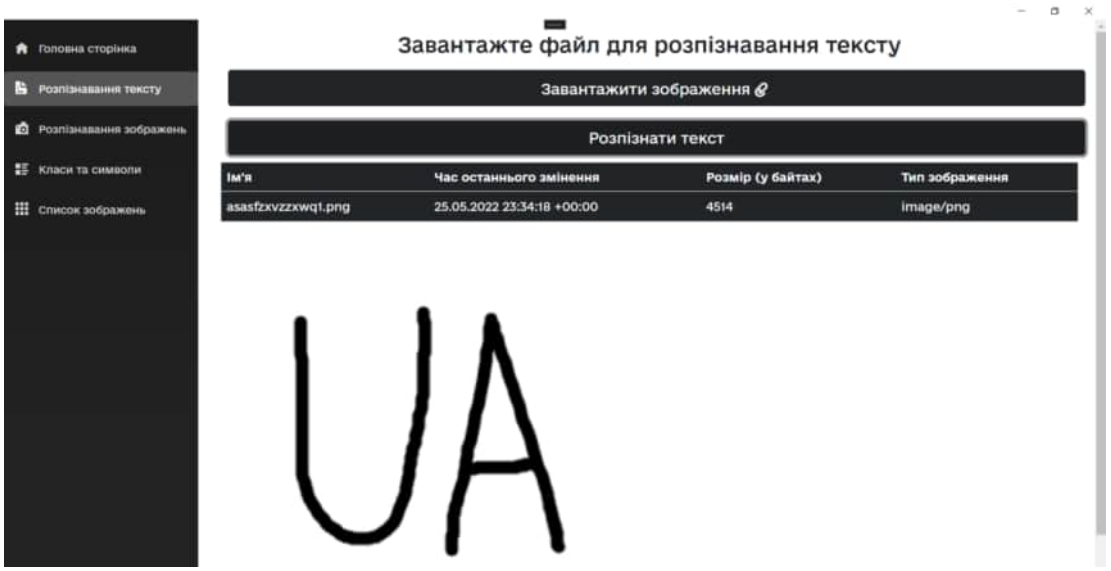

Рис. 3.8.3.

Після завантаження отримуємо інформацію про зображення, а саме: ім’я, час останньої зміни, розмір у байтах та тип зображення. Перевіримо розпізнавання тексту.

Після того як текст був розпізнаний користувач отримує результат:

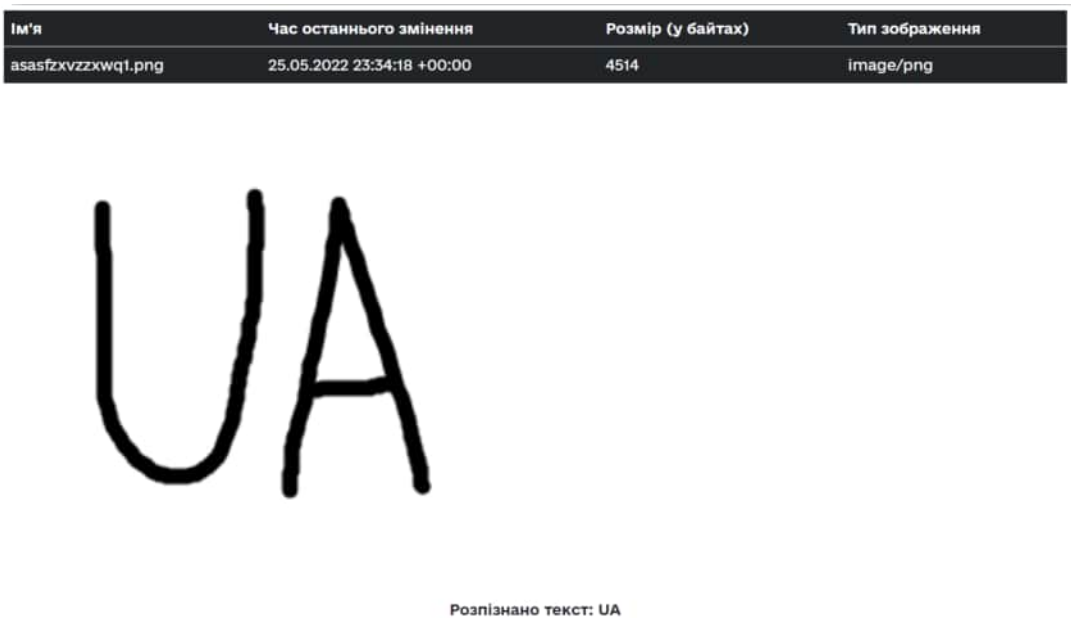

Рис. 3.8.4.

Як бачимо в результаті текст що міститься у файлі розпізнано правильно.

Отже, тепер перейдемо до сторінки “Розпізнавання зображень”:

Рис. 3.8.5.

Отримуємо схоже вікно, завантажуємо файл то розпізнаємо зображення:

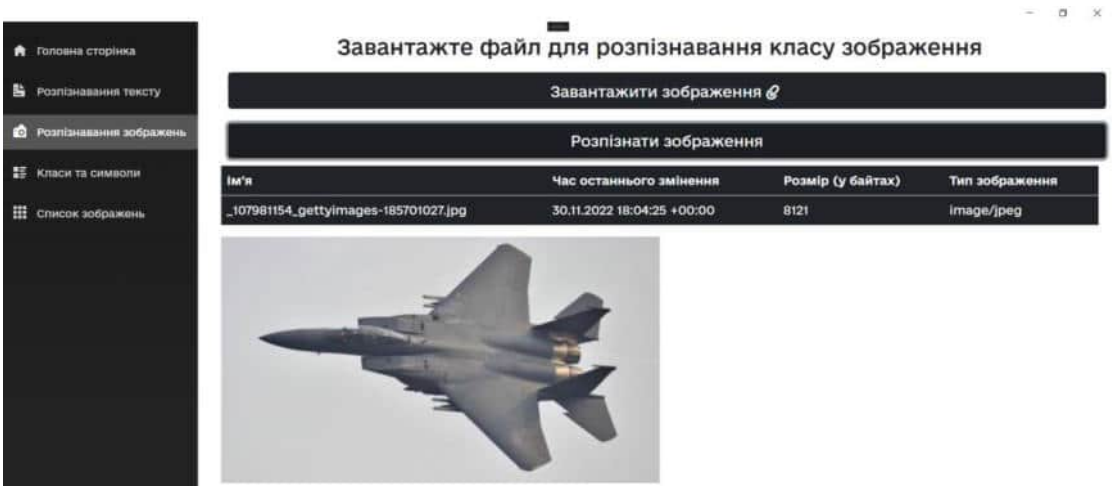

Рис. 3.8.6.

Коли користувач натискає кнопку відбувається надсилання запиту до веб-API та звідти запуск python модуля що розпізнає зображення. Це може зайняти якийсь час, тому було додати інтерфейс який вказує на те, що відбувається розпізнавання:

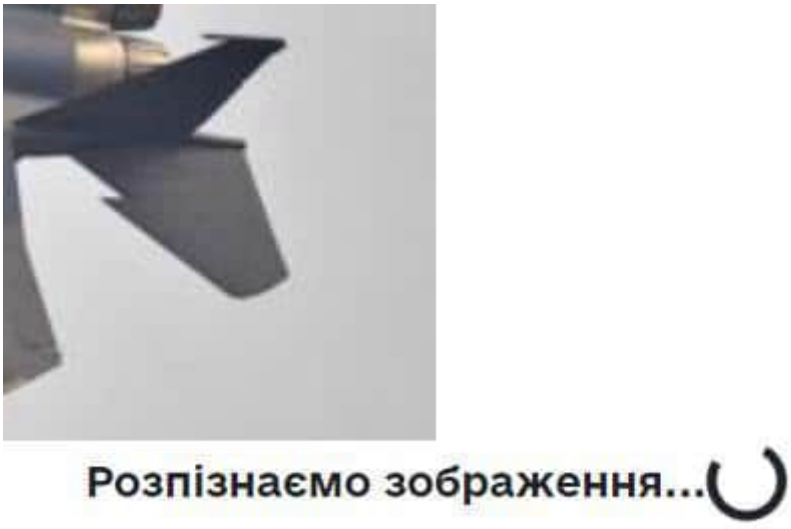

Рис. 3.8.7

В результаті отримуємо до якого класу зображення належить.

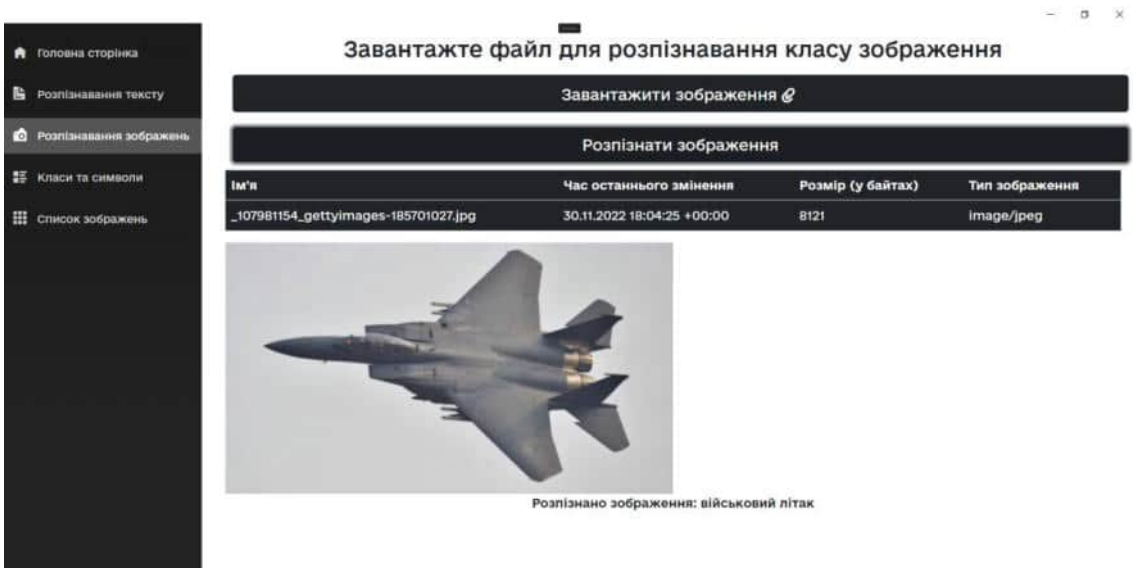

Рис. 3.8.8.

Як бачимо в результаті зображення розпізнано правильно.

Отже, тепер перейдемо до сторінки “Класи та символи” та завантажимо файл, що містить зображення та символи:

Рис. 3.8.9.

В результаті було отримано список розпізнаних символів. Для того щоб дізнатися чи правильно вони були розпізнані проаналізуємо зображення у файловій системі:

Рис. 3.8.10.

В результаті виконання програми можна побачити що :

- зображення 0 це символ “A” – вірно.
- зображення 1 це військовий танк – вірно.

Також перейдемо на сторінку “Список Зображень” та отримаємо дані для кожного з зображень яке було використане у дані програмі:

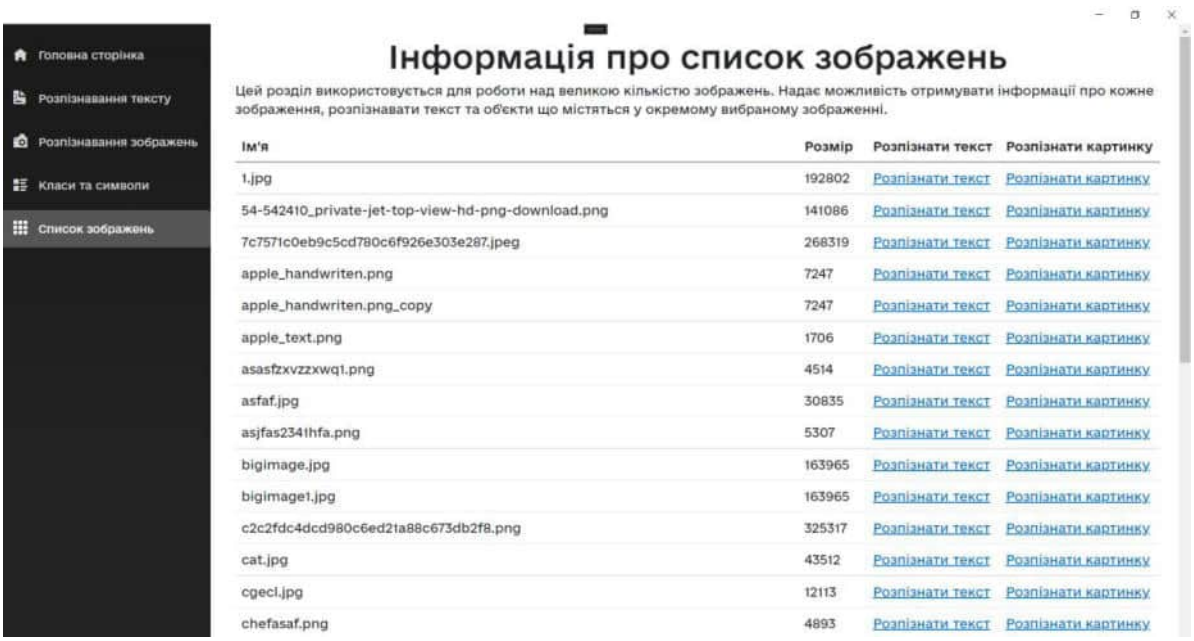

Рис. 3.8.11.

При натисканні на посилання користувач перенаправляється на сторінку де може виконати подальші дії.

3.9. Результати тестування

Проаналізувавши кожен з методів окремо можемо побачити що програма працює правильно і для кожного з тестових прикладів спрацювала вірно. Проте слід зазначити що даний програмний застосунок навчався на даних, які не дають можливості розпізнавати кожен з символів та зображень. Оскільки для розпізнавання символів використовувалися дані які містять рукописні англійські літери та числа і для розпізнавання зображень дані, які містять лише 10 класів.

Програма може приймати формати зображення такі як JPEG, BMP, PNG. Також зазначимо, що програма може видавати неправильний результат при роботі з певними шрифтами друкованих символів та інших класів, які не відносяться до тиз що визначені у даних для розпізнавання зображень.

Висновки

У даній магістерській роботі розроблено десктопний програмний застосунок, який дає змогу користувачу розпізнавати рукописний текст та класифікувати зображення що міститься у файлі. Дана програма розроблена за допомогою нейронних мереж, а саме згорткових, технологій комп’ютерного бачення та технологій .NET 6.0. В процесі даної роботи було висвітлено принципи роботи нейромереж, їх створення та навчання. Також були розглянуті процеси обробки зображення та створення веб-API на базі .NET, що містить бізнес-логіку у Python програмах. Наведено процес розробки десктопного застосунку за допомогою MAUI та тестування на операційній системі Windows. Процес навчання відбувався на великій базі даних рукописних символів англійського алфавіту та даних що містять військові та цивільні транспортні засобі такі як військові вантажівки, танки, військові та цивільні літаки, військові вертольоти, цивільні автомобілі.

Список використаних джерел

1. Microsoft: .NET Multi-platform App UI documentation – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/maui/
2. Gupta, Priyanka; Pareek, Bhavya; Singal, Gaurav; Rao, D Vijay (2021), "Military and Civilian Vehicles Classification", Mendeley Data, V1, doi: 10.17632/njdjkbxdpn.1 – [Електронний ресурс] – Режим доступу: https://data.mendeley.com/datasets/njdjkbxdpn/1
3. Mind: Майже 23 млн українців регулярно користуються Інтернетом – дослідження – [Електронний ресурс] – Режим доступу: https://mind.ua/news/20204323-majzhe-23-mln-ukrayinciv-regulyarno-koristuyutsya-internetom-doslidzhennya
4. Pewresearch: Mobile Fact Sheet – [Електронний ресурс] – Режим доступу: https://www.pewresearch.org/internet/fact-sheet/mobile/
5. TensorFlow. [Електронний ресурс] – Режим доступу: https://www.tensorflow.org/
6. Keras: Why choose Keras? [Електронний ресурс] – Режим доступу: https://keras.io/why_keras/
7. Tutorialsteacher: What is Web API? – [Електронний ресурс] – Режим доступу: https://www.tutorialsteacher.com/webapi/what-is-web-api
8. Класифікація зображень за допомогою ЗНМ – 2018 – [Електронний ресурс] – Режим доступу: https://codeguida.com/post/1400
9. The EMNIST Dataset: NIST – [Електронний ресурс] – Режим доступу: https://www.nist.gov/itl/products-and-services/emnist-dataset
10. OpenCV modules: Contour Features — [Електронний ресурс] – Режим доступу: https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html
11. Keras Conv2D and Convolutional Layers / Adrian Rosebrock – 2018 – [Електронний ресурс] – Режим доступу:

https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

12. Keras 1.2.2 Documentation: Convolutional Layers — [Електронний ресурс] – Режим доступу: http://faroit.com/keras-docs/1.2.2/layers/convolutional

Додаток А. Клас ImageService

```
public class ImageService
{

    3 references
    public async Task<string> sendRequest(string _url, string image_location, string image_name)
    {
        using var client = new HttpClient();
        string url = _url;

        var multipartFormContent = new MultipartFormDataContent();

        File.Copy(image_location, image_location + "_copy", true);

        using (var fs = new FileStream(image_location + "_copy", FileMode.Open, FileAccess.Read))
        using (var fileStreamContent = new StreamContent(fs))
        {
            fileStreamContent.Headers.ContentType = new MediaTypeHeaderValue("image/png");

            multipartFormContent.Add(fileStreamContent, name: "file", fileName: image_name);
            var response = await client.PostAsync(url, multipartFormContent);

            return await response.Content.ReadAsStringAsync();
        }
    }
```

Додаток Б. Методи розпізнавання

```
1 reference
public async Task<string> PredictText(string image_location, string image_name)
{
    var url = @$"{apiUrl}/PredictText";
    return await sendRequest(url, image_location, image_name);
}

1 reference
public async Task<string> PredictImage(string image_location, string image_name)
{
    var url = @$"{apiUrl}/PredictImage";
    return await sendRequest(url, image_location, image_name);
}
1 reference
public async Task<string> GetImageClassesSymbols(string image_location, string image_name)
{
    var url = @$"{apiUrl}/GetImageClassesSymbols";
    return await sendRequest(url, image_location, image_name);
}
```

Додаток В. MAUI Program

```
public static class MauiProgram
{
    4 references
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("e-Ukraine-Bold.otf", "e-Ukraine-Bold");
                fonts.AddFont("e-Ukraine-Medium.otf", "e-Ukraine-Medium");
                fonts.AddFont("e-Ukraine-Regular.otf", "e-Ukraine-Regular");
            });

        builder.Services.AddMauiBlazorWebView();
#if DEBUG
        builder.Services.AddBlazorWebViewDeveloperTools();
#endif

        builder.Services.AddSingleton<ImageService>();

        return builder.Build();
    }
}
```

Додаток Г. Модель нейронної мережі для військових та цивільних транспортних засобів

```
model = keras.models.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape = [image_rows, image_cols,3]),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(64, (2, 2), activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(64, (2, 2), activation='relu'),
    keras.layers.Flatten(),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(len(classes), activation ='softmax')
])
model.compile(optimizer='adam',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])
```

Додаток Д. Тренування нейронної мережі для розпізнавання військових та цивільних об’єктів

```
Epoch 22/30
251/251 [==============================] - 304s 1s/step - loss: 0.0225 - accuracy: 0.9661 - val_loss: 0.4490 - val_accuracy: 0.
7579
Epoch 23/30
251/251 [==============================] - 300s 1s/step - loss: 0.0256 - accuracy: 0.9650 - val_loss: 0.4259 - val_accuracy: 0.
7500
Epoch 24/30
251/251 [==============================] - 292s 1s/step - loss: 0.0235 - accuracy: 0.9674 - val_loss: 0.4365 - val_accuracy: 0.
7504
Epoch 25/30
251/251 [==============================] - 299s 1s/step - loss: 0.0231 - accuracy: 0.9685 - val_loss: 0.4122 - val_accuracy: 0.
7564
Epoch 26/30
251/251 [==============================] - 311s 1s/step - loss: 0.0213 - accuracy: 0.9679 - val_loss: 0.4437 - val_accuracy: 0.
7594
Epoch 27/30
251/251 [==============================] - 299s 1s/step - loss: 0.0202 - accuracy: 0.9701 - val_loss: 0.4326 - val_accuracy: 0.
7583
Epoch 28/30
251/251 [==============================] - 302s 1s/step - loss: 0.0220 - accuracy: 0.9670 - val_loss: 0.4721 - val_accuracy: 0.
7572
Epoch 29/30
251/251 [==============================] - 272s 1s/step - loss: 0.0198 - accuracy: 0.9688 - val_loss: 0.4213 - val_accuracy: 0.
7628
Epoch 30/30
251/251 [==============================] - 272s 1s/step - loss: 0.0204 - accuracy: 0.9681 - val_loss: 0.4903 - val_accuracy: 0.
7617
```