Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи розпізнавання зображень для системи персонального обліку витрат
(тема)

Виконав:
студент II курсу, групи СКСм-22-2
Михальчук М.О.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

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

Освітня програма
Спеціалізовані комп'ютерні мережі
(повна назва освітньої програми)

Керівник: доцент каф. АПОТ Рахліс Д.Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри АПОТ _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

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

Освітня програма Спеціалізовані комп'ютерні мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)

"03" вересня 2023р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Михальчуку Максиму Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи розпізнавання зображення для системи персонального обліку витрат

затверджена наказом по університету від "03" листопада 2023 р. № 1282 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 22 січня 2024 р.

3. Вхідні дані до роботи OCR, Оптичне розпізнавання символів, OCR Tesseract, Google Cloud Vision API, Apple Vision API, Abbyy Finereader, ознаковий метод, шаблонний метод, структурний метод, EAST, CRNN, iOS, XCode IDE

4. Перелік питань, що потрібно опрацювати у роботі

1) Проаналізувати послідовність роботи технологій розпізнавання символів
2) Огляд існуючих рішень OCR систем
3) Дослідження алгоритму попередньої обробки зображення
4) Проектування системи розпізнавання символів
5) Програмна реалізація системи
6) Порівняння розробленої системи з існуючими рішеннями

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

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

Календарний план

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Видача теми роботи її узгодження та затвердження	02.09.2023 – 14.09.2023	Виконано
2	Аналіз літератури за напрямом роботи	14.09.2023 – 28.09.2023	Виконано
3	Аналіз існуючих методів вирішення поставленого завдання	28.09.2023 – 17.10.2023	Виконано
4	Проектування системи	17.10.2023 – 06.11.2023	Виконано
5	Розробка програмної реалізація	06.11.2023 – 19.11.2023	Виконано
6	Тестування та порівняння системи з існуючими рішеннями	19.11.2023 – 02.12.2023	Виконано
7	Оформлення пояснювальної записки	02.12.2023 – 13.01.2024	Виконано
8	Перевірка виконаної роботи керівником, допуск до захисту	15.01.2024	Виконано
9	Захист роботи	22.01.2024	

Дата видачі завдання 02 вересня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доцент каф. АПОТ Рахліс Д.Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить 88 сторінок, 44 рисунків, 1 таблиця, 13 джерел за переліком посилань.

ОПТИЧНЕ РОЗПІЗНАВАННЯ ТЕКСТУ, OCR, TESSERACT OCR, ABBYY FINEREADER, GOOGLE CLOUD VISION AI, APPLE VISION API, ШАБЛОННИЙ МЕТОД, ОЗНАКОВИЙ МЕТОД, СТУКТУРНИЙ МЕТОД.

Предметом дослідження є методи розпізнавання тексту на зображені.

В процесі виконання атестаційної роботи було розглянуто технологію OCR, її особливості, сфери використання та перспективи її розвитку у майбутньому. Також було досліджено популярні системи оптичного розпізнавання тексту, визначено їх особливості та виконано порівняння цих систем. Було розглянуто специфіку предметної області, виділено основні фактори, які можуть впливати на досягнення максимального результату. Виконано навчання нейронної мережі, а також програмну реалізацію алгоритму попередньої обробки зображення, проаналізовано отримані результати та порівняно їх з існуючими рішеннями OCR.

ABSTRACT

The explanatory note for the qualification work comprises 88 pages, 44 figures, 1 table, and references to 13 sources.

OPTICAL TEXT RECOGNITION, OCR, TESSERACT OCR, ABBYY FINEREADER, GOOGLE CLOUD VISION AI, APPLE VISION API, TEMPLATE METHOD, TOKEN METHOD, STRUCTURE METHOD.

The subject of the research is methods of text recognition in images.

In the process of completing the certification work, OCR technology was examined, including its features, areas of application, and prospects for future development. Popular optical character recognition systems were also investigated, their features were identified, and a comparison of these systems was carried out. The specificity of the subject area was examined, highlighting the main factors that can influence the achievement of the maximum result. Neural network training was conducted, as well as the software implementation of the image preprocessing algorithm. The obtained results were analyzed, and a comparison was made with existing OCR solutions.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

OCR – optical character recognition;

PDF – portable document format;

НМ – нейронна мережа;

C++ – мова програмування;

SVM – support vector machines;

EAST – ефективний детектор тонкої сцени тексту;

CRNN – конволюційно-рекурентна нейронна мережа;

LSTM – long short-term memory;

IDF – inverse document frequency;

API – application programming interface;

Flow – потік керування;

UI – user interface;

Opensource – програмне забезпечення з відкритим вихідним кодом.

ВСТУП

Розпізнавання – це здатність живих організмів виявляти в потоці інформації, що надходить від органів чуття, певні об'єкти, закономірності, явища. Воно може здійснюватися на основі зорової, слухової, тактильної інформації. Так, людина без зусиль може впізнати іншу знайому йому людини, глянувши на неї або почувши її голос. Деякі тварини активно використовують нюх для впізнавання інших особин і пошуку їжі [1].

Можливість розпізнавання спирається на схожість однотипних об'єктів. Незважаючи на те, що всі предмети і ситуації унікальні в строгому сенсі, між деякими з них завжди можна знайти подібності за тією або іншою ознакою. Звідси виникає поняття класифікації – розбиття всієї множини об'єктів на непересічні підмножини – класи, елементи яких мають деякі схожі властивості, що відрізняють їх від елементів інших класів. І, таким чином, завданням розпізнавання є віднесення розглянутих об'єктів або явищ за їхнім описом до потрібних класів. Тобто поняття розпізнавання можна розширити, якщо говорити про виявлення об'єктів в потоці не тільки чуттєвої, а й будь-якої іншої інформації. Наприклад, можна говорити про розпізнавання хвороби по її симптомах у хворого або про розпізнавання соціальних явищ по статистичній інформації [1].

З самого початку, комп’ютери розроблялися для автоматизації та пришвидшення повсякденних задач з обробки інформації для спеціалізованих державних та військових структур, а завдяки розвитку та поширенню обробки інформації за допомогою комп’ютера, це досить швидко привело до необхідності розробки технологій, що дадуть змогу машинам застосовувати розпізнавання в обробленій ними інформації. Розробка методів машинного [1] розпізнавання дозволила розширити спектр виконаних комп’ютерами завдання та зробило таку переробку інформації більш інтелектуальної. Однією з областей де використовується комп’ютерне

розпізнавання є переведення рукописного та надрукованого на папері тексту в електронний вигляд, а враховуючи всебічну та масову комп’ютеризацію, завдання оцифрування друкованого та рукописного тексту є досить актуальним, як для державних та приватних установ, так і для пересічних громадян, адже дозволяє автоматизувати цей процес, а також збільшує безпекову компоненту, що зменшить кількість помилок при введені даних, які можуть коштувати дуже дорого.

Однією з досить популярних сфер де можна було б використати цю технологію – це системи обліку витрат, адже наявні програмні продукти, які, як заявляється розробником, повинні допомогти автоматизувати облік витрат, насправді мають ряд серйозних недоліків (наприклад, не можна порівняти щомісячну динаміку витрат в окремій категорії), але найголовніший недолік, це те, що використання таких застосунків вимагає від користувача, регулярно та детально вести облік витрачених коштів, розподіляючи в системі свої витрати за категорією [2]. Тобто фактично, головна мета (моніторинг власних фінансів та візуалізація додаткової інформації), заради якої користувач встановлює застосунок такого типу, цілком та повністю, залежить від того наскільки сумлінно та регулярно користувач буде вести облік витрат, що майже ніяк не відрізняється від використання звичайної записної книжки, де можна вносити свої покупки, а це аж ніяк не нагадую вище зазначену “автоматизацію”, а враховуючи що за один день може бути дуже багато витрат, то детальне внесення усіх покупок може займати досить великий проміжок часу, але слід зазначити, що у всіх покупках є одне спільне – це касовий чек, який людина отримує після кожної сплаченої покупки. То чому б не використати чек на якому є вся необхідна людині інформація, щоб не вводити її вручну у застосунок. Саме тут, кожному, хто бажає максимально автоматизовано слідкувати за своїми витратами приходить на допомого технології оптичного розпізнавання символів (OCR), які допоможуть “витягнути” всю необхідну інформації з чеку, використовуючи, камеру телефону [3].

1 ТЕХНОЛОГІЇ ОПТИЧНОГО РОЗПІЗНАВАННЯ СИМВОЛІВ ТА ЇХ РЕАЛІЗАЦІЯ

1.1 Оптичне розпізнавання символів

Оптичне розпізнавання символів (OCR) – це технологія, яка дозволяє комп’ютеру розпізнавати та інтерпретувати текст або символи зображені на фото, PDF-документах або інших типах графічних зображень. OCR використовується для автоматичного перетворення надрукованого або рукописного тексту цифровий формат, що в подальшому може обробляти комп’ютером.

Розпізнавання символів широко використовується в різноманітних областях, включаючи сканування документів та конвертації книг в електронний вигляд, автоматизацію операцій з банківськими чеками, розпізнавання автомобільних номерних знаків, розпізнавання рукописного тексту тощо. Технологія OCR не тільки полегшує зберігання даних, але й дозволяє здійснювати пошук окремих слів та речень, а також редагувати увесь текст або окремі його фрази. Також технології розпізнавання тексту дозволяють проводити аналіз тексту та застосовувати до нього форматування та переклад на інші мови. За допомогою OCR будь-який користувач може отримати доступ до раніше недоступних даних, всього за декілька натискань клавіші миші, а з зростанням потужності обчислювальної систем розвитком машинного навчання, технології OCR стають все точнішими та ширше використовуються [4].

1.2 Проблеми оптичного розпізнавання символів

Зазвичай, зараз галузь оптичних алфавітів використовується у проектах широкого спектру, починаючи від перекладу тексту на зображенні до

створення сканування документів та створення масивних цифрових архівів. Але варто зазначити, що ця технологія все ще є областю активних технічних досліджень. Слід зазначити, що гнучке OCR прагне до активного використання широкого кола зображень статей, надрукованих шляхом обробки:

– багатьох алфавітів та багатомовної ідентифікації;
– сценарії Omni;
– вільний розподіл матеріалів;

Розпізнавання тексту рукописної поведінки – це досить молодий метод OCR, який повинен бути максимально потужним та гнучким. Але, як правило, залигається проблема, яка активно досліджується, щоб полегшити рішення комерційних розробників під час їх розробці та використанні технологій OCR, наприклад:

– ідентифікація почерку;
– визначення підпису людини;
– сканування адрес для поштових відправлень.

1.3 Методи попередньої обробки зображення

Головним викликом для OCR систем є розв’язання задачі, як розпізнати один і той самий символ, який написаний за допомогою різних шрифтів, адже в світі є велика кількість різноманітних шрифтів і кожного дня можуть з’являтися нові й нові шрифти. В додаток до цього, перед тим як розпочати процес “зчитування”, необхідно виконати обробку зображення, цей процес називається перед обробка зображення. Він є необхідним, адже не усі зображення одразу підходять до “зчитування”. Здебільшого OCR системи виконують цей етап самостійно, що дозволяє значно покращити шанс та результат розпізнавання Основні методи попередньої обробки включають наступне.

1. Корекція спотворення. Якщо текст на зображенні або документі має деформації, такі як нахил, скручування або розтягування, для збільшення та покращення точності розпізнавання, нд файлом, що піддається розпізнаванню, здійснюється корекція, яка повинна змінити зображення таким чином, щоб текст на ньому був розташований у максимально горизонтальному або вертикальному положенні (рис 1.1).

Рисунок 1.1 – Виправлення кута нахилу

2. Видалення шумів. Шум може впливати на якість зображення та точність розпізнавання, тому щоб видалити такі “плями”, які можуть бути як позитивними так і негативними у порівнянні з основною гаммою кольорів, застосовують фільтрацію, медіанне видалення або математичні операції, такі як морфологічні операція. Також під час цих операція може бути вирівняно контури символів, усунуто розмиття на зображені [5].

3. Бінаризація. Вхідне зображення (рис. 1.2 (а)) модифікується таким чином, щоб отримати таке саме зображення у чорно-білому форматі, також такі зображення називають бінарними або двійковими, через те, що в ньому використовується тільки два кольори – білий та чорний. Бінаризація дозволяє розділити будь-який елемент, який цікавить користувача, в даному ж випадку це текстові елементи, від фону. Цей процес здійснюється за допомогою застосування однотонового фільтру (рис 1.2 (б)).

4. Видалення фону. Іноді фон може перекривати або впливати на текстові області. Видалення фону допомагає виділити лише текст, шляхом

використання алгоритмів, таких як адаптивне порогування або видалення фону на основі текстури (рис 1.2 (в)).

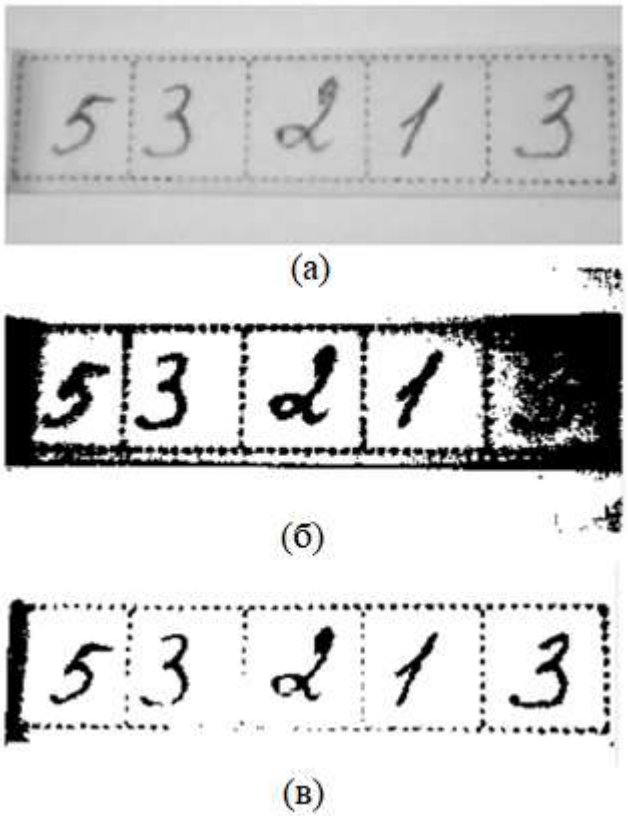


Рисунок 1.2 – Вихідне зображення (а), бінаризоване зображення (б), бінаризоване зображення з підібраним порогом (в)

5. Аналіз зображення. На цьому етапі виконується аналіз зображення для визначення усіх структурних елементів, таких як абзаци, таблиці та підписи і в подальшому віднесення кожного елементу як окремий блок. Зазвичай цей етап є максимально корисним, коли на зображені є таблиці та макети.

6. Визначення рядків і слів. На цьому етапі виконується розділення виявлених на минулому кроці блоків на рядки, окремі слова та символи.

7. Визначення рукописного тексту та аналіз почерку. У документах, в яких є рукописний текст, форма кожного наступного слова може бути дещо відмінною від попереднього, ба більше навість одні й ті самі букви в одному слові написані від руки можуть мати різну форму, саме тому процес

визначення почерку є критично важливим перед тим як конкретна OCR модель буде застосована для розпізнавання тексту з наявним почерком.

8. Поділ тексту на сегменти або інакше кажучи сегментація. Сегментація - це процес, який визначає компоненти зображення. Тобто він знаходить ті області документа, в яких знаходяться необхідні дані (літери, цифри, символи). Він працює таким чином, що спершу знаходиться медіанне значення відстані між двома символами в слові [2]. Після цього розпочинається процес розділення зображення на окремі області, які містять окремі символи або слова. Коректна сегментація дозволяє точно виділити та ізолювати символи, що полегшує процес їх розпізнавання. Цей етап є досить важливим, так як точність подальшого розпізнавання символів залежить від якості сегментація, адже, наприклад, що робити, якщо два символи стикаються один з одним, їх розглядати як один символ чи навпаки взагалі не вважати це за символ. Помилки сегментації можуть призвести до неправильного розпізнавання символів або збільшення кількості помилок [5].

9. Виділення ознак. Цей етап є один з найважчих. Метою вилучення особливостей символів є охоплення основних характеристик зображення. Процес виділення ознак може бути здійснений за допомогою одного з двох основних алгоритмів:

– алгоритм визначення ознак. Він схожий на ICR, про який більш детально буде написано в наступному розділі;

– алгоритм розпізнавання зразків, приклад зображено на рисунку 1.3. Це алгоритм який розпізнає символ цілком;

Рисунок 1.3 – Алгоритм розпізнавання зразків

10. Класифікація символів. Класифікація – це процес під час якого виконується ідентифікація кожного символу та присвоєння йому відповідного класу. Для виконання класифікації існують два методів:

– метод відбору, який заснований на подібності вимірювання відстаней у різних векторах. Цей метод дуже добре працює, коли символи коректно та добре розділені один від одного;

– метод оптимальних статистичних класифікаторів. Цей метод використовує основи ймовірності під час розпізнавання. Сутність цього методу полягає у використанні ідеалізована схема класифікації, що означає, що в середньому вона дає найменший шанс на помилку при класифікації. Такий класифікатор називається “Байєсівським класифікатором”. Представимо собі невідомий символ, який описаний із властивостями вектора, де ймовірність належності до символу розраховується до класу C для всіх класів C = 1… Потім цьому “невідомому” символу призначається клас, який дає символу найбільшу ймовірність.

1.4 ICR функція в OCR технологіях

ICR – це функція інтелектуального розпізнавання символів за допомогою нейронних мереж. Як працює ця функція. На вхід функції подається зображення з текстом, яке розбивається на безліч фрагментів і для кожного з цих фрагментів робить кілька припущень, які потім перевіряються та порівнюються з еталоном. Після цього кожному фрагменту надається оцінка, відповідно ступеню збігу і далі програма, вибираючи припущення з найбільшою оцінкою, таким чином “бачить символ” і виводить його в вбудований текстовий редактор (рис 1.4).

Рисунок 1.4 – Ілюстрація роботи функції ICR

Також слід зазначити, що прототип ICR технології використовується в звичних для усіх людей смартфонах, а саме підчас введення рукописного тексту. Якщо ввести букву А, то сенсорний екран може зрозуміти, що ви вводите одну косу риску, потім другу, а потім горизонтальну лінію, якою вони з’єднуються. Іншими словами, на півдорозі комп'ютер заздалегідь розуміє, що саме ви хочете написати, тому що ви формуєте букву з окремих штрихів, по одному, і це робить розпізнавання набагато простішим, ніж розпізнавання рукописного тексту, вже написаного на папері [3].

Головна перевага OCR з ICR в порівнянні з просто OCR – це компактність, адже основний функціонал виконується за допомогою нейронної мережі, а також безперервне навчання, яке дозволяє ефективно розпізнавати рукописні слова. Але в той же чай є один вагомий недолік. Ця технологія не здатна прочитати один суцільний рукописний текст.

1.5 Методи розпізнавання символів

Припустимо, що в нашому алфавіті є всього лише одна буква А, і всі люди пишуть її однаково. Але навіть в такому гіпотетичному світі з розпізнаванням однієї букви можуть виникнути досить серйозні проблеми, адже кожен текст є унікальним та неповторним. Навіть розпізнавання друкованого тексту не є повністю вирішеною задачею. Адже є купа різноманітних шрифтів та варіацій одного і того самого шрифту, також сюди слід додати різноманітні розміри букв, що також можуть як покращити

розпізнавання так і погіршити його. Так звісно, друкований текст розпізнати дещо простіше, але там також є свої нюанси, про які було згадано раніше.

Отже при розгляді принципу роботи є три основні підходи:

– метод розпізнавання на основі шаблонів;

– метод розпізнавання на основі структури;

– метод розпізнавання на основі ознак.

Методи розпізнавання тексту в OCR системах використовуються для перетворення зображення тексту на машино читабельний формат. Усі перелічені методи базуються на алгоритмах та техніках обробки зображення та машинного навчання [6].

Системи розпізнавання застосовують різноманітні методи, це можуть бути як ознакові, структурні та шаблонні (растрові) так і методи засновані на використанні нейронних мереж тощо (рис. 1.5).

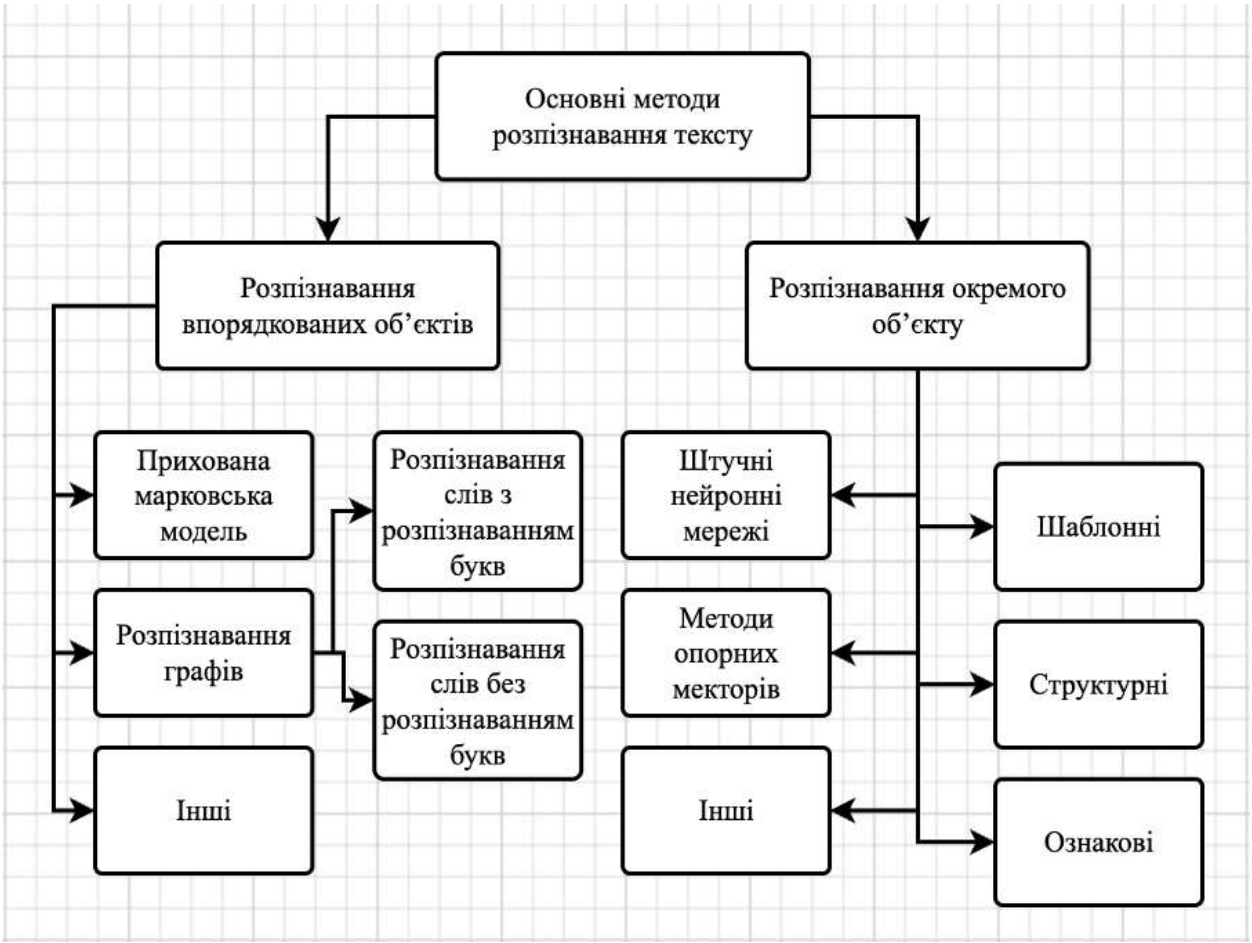


Рисунок 1.5 – Блок-схема основних методів розпізнавання тексту

1.5.1 Шаблонні (растрові) методи розпізнавання тексту

Шаблонні методи є одними з основних методів розпізнавання тексту в OCR. Методи засновані на цьому принципі для розпізнавання символів використовують попередньо визначені шаблони символів. Вони порівнюють вхідне зображення з наявними шаблонами і визначають символ на ньому.

Основна ідея шаблонних методів полягає в тому, що для кожного символу створюється набір шаблонів, які зображують конкретний символ у різних варіаціях, розмірах, або стилістиці. Ці шаблони можуть бути побудовані вручну або отримані з начального набору даних.

Процес розпізнавання символу за допомогою шаблонних методів складається з наступних кроків.

1. Побудова шаблонів. Для кожного символу створюються шаблони, що зображують його в різних варіаціях. Це можуть бути чорно-білі зображення символів або сети ознак, які описують форму та структуру символу.

2. Виділення символів. Зображення розділяється на окремі області, що містять символи за допомогою методів сегментації.

3. Порівняння з шаблонами. Кожен виділений символ порівнюється з усіма шаблонами, що відповідають класу цього символу. Зазвичай використовується метрики схожості, такі як сума або різниця пікселів, для визначення ступеня подібності між символом та шаблоном [7].

4. Визначення символу. Символ призначається класу, який має найбільшу подібність до його шаблону. Це може бути зроблено за допомогою порогового значення для подібності або шляхом використання класифікатора, який віднесе символ до певного класу.

Шаблонні методи мають свої переваги та недоліки. До переваг слід віднести простоту реалізації, швидкість роботи та добре визначені шаблони для всіх символів. А також високу точність розпізнавання дефектних символів. До недоліків можна віднести обмеження у випадках, коли символи

мають велику варіативність в своїх формах або коли не вдалося побудувати достатньо репрезентативні шаблони для всіх символів [7].

На сьогоднішній день шаблонні методи часто комбінують з іншими методами розпізнавання тексту, такими як методи на основі нейронних мереж, для досягнення кращих результатів розпізнавання [8].

1.5.2 Структурні методи розпізнавання тексту

Методи, що для розпізнавання використовують набір ознак, називаються структурними. В своїй основі для виконання процесу розпізнавання вони спираються на інформацію про набір структурних компонентів та їх розташування для кожному з символів.

Процес розпізнавання символу за допомогою структурних методів складається з наступних кроків.

1. Як і для шаблонних методів першим кроком є сегментація тексту.
2. Визначення структури символу. Тобто розбиття символу на складові.
3. Визначення ознак символу. Для кожного символу або слова визначаються ознаки, які характеризують їх вигляд, наприклад форму, розмір, контекстну інформацію тощо. Використовуються різні підходи, включаючи використання геометричних ознак, структурних ознак, ознак текстури та інші.
4. Моделювання структури: Використовуючи зібрані ознаки та синтаксичні правила, структура тексту моделюється у вигляді структурного дерева або графу. Ця модель відображає ієрархічну структуру тексту та зв'язки між його елементами.
5. Вибір інтерпретації: На основі моделі структури тексту можна вибрати найбільш вірогідну інтерпретацію тексту. Це може включати вибір правильного розташування символів, перетворення тексту в послідовності слів або додаткову обробку для поліпшення результатів.

До переваг структурних методів можна віднести, незалежність цих методів від типів та розмірів шрифтів;

Що ж стосується недоліків, то до них, слід віднести складність ідентифікації знаків, що мають дефекти (наприклад, розриви ліній або злиття сусідніх ліній), а також повільна швидкість розпізнавання та великі ресурсні витрати. Також слід зазначити, що ці методи сильніше за інші залежать від якісної сегментації.

1.5.3 Ознакові методи розпізнавання тексту

Ознакові методи є досить популярними для розпізнавання тексту. Для виконання своєї функції ці методи розпізнають текстові символи на основі їх візуальних ознак.

Основна ідея ознакових методів полягає в тому, щоб витягти різноманітні візуальні характеристики символів і використовувати їх для класифікації. Ознаки можуть включати такі аспекти, як форма, текстура, контур, геометричні властивості та інші.

Основні етапи ознакових методів розпізнавання тексту включають.

1. Першим етапом є попередня обробка, як і для більшості методів для розпізнавання тексту, ознакові не стали виключенням.

2. Витягування ознак. На цьому етапі зображення символів аналізується для витягування різних ознак. Це може включати вимірювання геометричних властивостей, аналіз текстури, визначення контурів та інші методи. Ознаки можуть бути числовими векторами, які описують характеристики символів.

3. Класифікація. Після витягування ознак символи класифікуються за допомогою різних алгоритмів. Це можуть бути класифікатори, такі як метод опорних векторів (SVM), настроєні на навчальних даних, або нейронні мережі, які використовуються для розпізнавання тексту.

4. Пост-обробка. Після класифікації можуть застосовуватися алгоритми пост-обробки, такі як корекція помилок або злиття символів у слова, для поліпшення точності розпізнавання [8].

Переваги ознакових методів:

– гнучкість;

– можливість інтерпретації текстової структури;

– простота реалізації та ефективність.

Недоліками таких методів є:

– вразливість до якості ознак;

– вразливість до змін у зображеннях;

– високі вимоги до значної кількості навчальних даних;

– обмежена здатність до розпізнавання складних документів.

У реалізації ознакових методів використовуються різноманітні підходи, але їх успішність залежить від вибору підходів, які найкраще підходять для конкретної задачі OCR і якості використовуваних ознак.

1.6 Технології OCR та нейронні мережі

Як і в будь-якої технології в OCR є свої проблеми, і так само, є багато різних способів їх вирішення. Напевне, один з найбільш популярних – це підходи які в своїй основі містять нейронні мережі, адже вони є “універсальним солдатом“ при вирішені проблем OCR. За допомогою нейронних мереж можна виконувати розпізнавання шаблонів, сегментацію, класифікацію, класифікацію та інші завдання.

Нейронна мережа (НМ) – це, напевне, один з найкращих інструментів, за допомогою якого можна вирішити всі проблеми з OCR, але для цього потрібно буде правильно вибрати класифікатор. НМ – це математична модель, або їх набір, що може імітувати структуру та функціонування біологічної нейронної мережі і її головна мета полягає у вирішені різноманітних завдань. В якості основної складової нейронної мережі

виступає топологія. У своєму складі, сучасні нейронні мережі, містять велику кількість взаємопов’язаних вузлів (елементів) для обробки. Ці елементи з’єднуються за допомогою дуг, які називаються посиланнями. Всі ці компоненти в сукупності називаються графовою структурою.

Нейронні мережі це звичайно чудово, але не слід забувати, що так само, як люди вчиться ходити, НМ треба навчити розпізнавати відповідні класи на зображені І для цього добре підходить процес глибинного навчання. Глибинне навчання – це навчання (його ще називають глибоким структурованим навчанням або диференційованим програмуванням). Цей процес є частиною широкого спектру методів машинного навчання, який в своїй основі використовує штучні нейронні мережі, з представницьким навчанням. Навчання може відбуватись під наглядом, під напівконтролем або взагалі без нагляду [9].

На спеціалізованих підходах глибинного навчання слід зупинитись більш детально.

1.6.1 Нейронна мережа EAST

EAST (Ефективний детектор тонкої сцени тексту) – це простий та потужний метод виявлення тексту, який базується на використанні спеціалізованої мережі. Цей метод, на відміну від інших методів, може лише виявляти текст, тобто не здатний на фактичне розпізнавання, але натомість він має високу надійність. Ще одна перевага цього методу – це те, що його було додано до бібліотеки open-CV (з версії 4), що дозволяє використовувати його всім бажаючим. Насправді, ця мережа є відомою всім мережею U-Net, яка добре підходить для виявлення ознак, які можуть відрізнятися за розмірами. Основний “стовбур”, який зображено на рисунку 1.6, може бути видозмінено в залежності від задач.

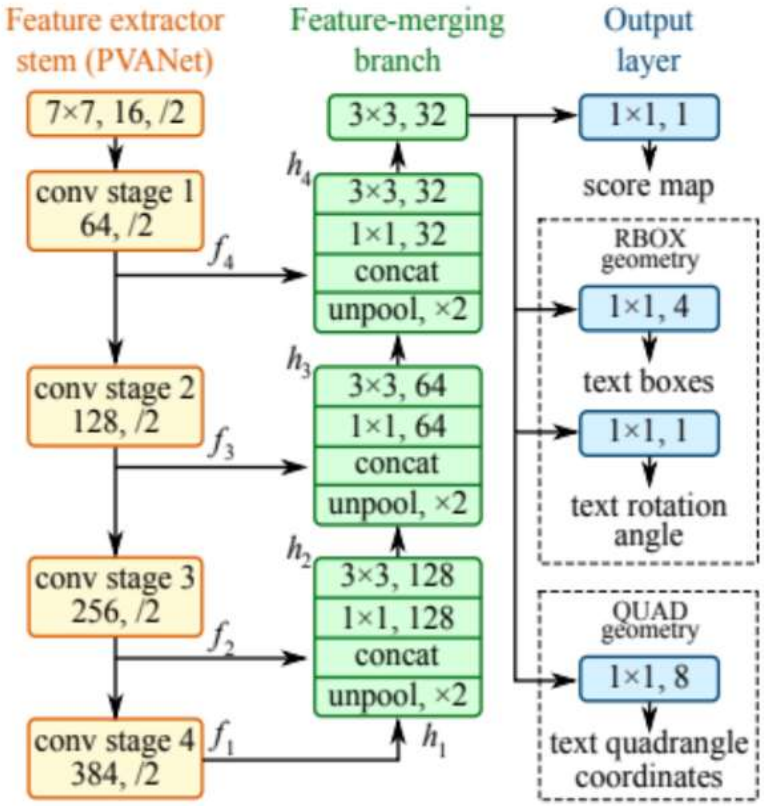


Рисунок 1.6 – Схема розпізнавання тексту з ідеальним шрифтом

В даному випадку на рисунку 1.6 зображено варіацію PVANet. Як і в U-Net ознаки витягуються з різних рівнів мережі.

Нейронна мережа дозволяє використовувати два типи виходів обмежувальних ромок з поворотом:

– стандартна обмежувальна рамка з кутом повороту (2*2+1 параметри);

– обмежувальна рамка типу “чотирикутник”, яка має невеликий кут повороту.

1.6.2 Нейронна мережа CRNN

CRNN – це конволюційно-рекурентна нейронна мережа, яка вперше була описана у 2015 році. Для цієї НМ застосовується гібридна безперервна архітектура, головна мета якої – це захоплення слів за допомогою підходу “тьох кроків”.

Основа ідеї полягає в тому, що на першому рівні знаходиться стандартна згорткова мережа. Натомість останній рівень – це “особливий шар”, який представляє собою велику кількість “стовпців”. Кожен з яких є окремою ознакою і головна мета якого відображати певний розділ в тексті, що зображено на рисунку 1.7.

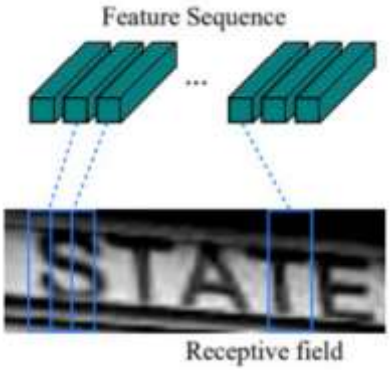


Рисунок 1.7 – Схема представлення тексту стовпцями ознак

По завершенню минулого етапу, усі “стовпці” ознак, щ обули виявлені подаються на функціонал під назвою “двонаправлений LSTM”, який зображено на рисунку 1.8.

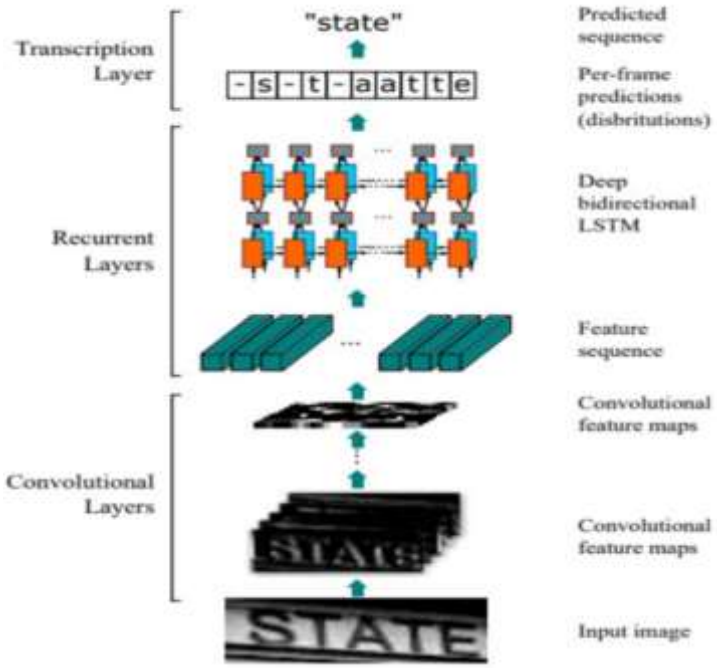


Рисунок 1.8 – Схема двонаправленого LSTM

Цей механізм виводить послідовність, також він застосовується для знаходження зав’язків між символами слова.

І остання, третя частина НМ – це шар транскрипції. ЇЇ мета – це взяти усі наявні символи отримані після перших двох шарів і використовуючи імовірнісний метод уніфікувати їх та осмислити. Цей метод ще можна зустріти під назвою CTC. Головна перевага цього шару, що його можна використовувати як з, так і без попереднього визначення лексикону, що може полегшити передбачення слів.

1.7 Відомі реалізації OCR в сучасному світі

З самого початку технологія OCR розпочала свій розвиток з метою “оцифрувати” фізичні документи (книги, листи, банківські чеки, тощо) і до сьогоднішнього дня головна мета технології – оцифрування фізичної інформації навколо людей, не змінилась, хоча й додалось досить багато другорядних завдань.

Існує багато способів використання OCR, оскільки ця технологія може бути основою основної ідеї та додатковим інструментом для автоматизації будь-чого. Як приклад використання функцій OCR як основної ідеї, в Інтернеті існує багато сайтів для "оцифрування" документів. Однією з цих хороших ідей є сервіс, який розпізнає текст книги, а потім читає його користувачеві за допомогою перетворення тексту в мовлення, ще одна цікава ідея – це музичний OCR SmartScore, головна мета якого – розпізнавати музичні ноти. Як приклад іншого використання, візьмемо Google Translate, головною функцією є перекладач, але для простоти використання Google із його персональним застосуноком OCR він дозволив користувачеві перекладати текст із зображення або камери на будь-який інтелектуальний пристрій “на ходу” з додатковими функціями OCR. І таких “випадків” багато, тому що ви можете звернутися до використання OCR практично в будь-якому застосунку.

OCR дуже популярна технологія і реалізацій безліч, тому у цьому розділі проведемо огляд найпопулярніших рішень, та порівняємо їх.

1.7.1 Tesseract OCR

OCR-система Tesseract – це opensource система, принцип роботи якого зображено на рисунку 1.9, яка займає одне з топових місць по популярності у розробників OCR систем, незважаючи на те, що іноді його може бути складно додати до свого проекту та модифікувати, довгий час на ринку не було альтернатив, які б могли позмагатися з Tesseract OCR у функціоналі, швидкості та якості розпізнавання та при цьому були б безкоштовними.

Розробка цієї системи розпочиналась, як звичайне дослідження, проте з часом він переріс у одне з найпопулярніших opensource рішень у сфері розпізнавання тексту на зображені. Починаючи з 1984 по 94 рік, розробкою системи займалась компанія HP, а у 2005 році проект було відкрито для широких мас. У 2006 році компанія Google викупила права на цю розробки і до сьогоднішнього дня веде розробку цієї OCR системи.

Починаючи з версії 4.0.0 в основі Tesseract використовується нова підсистема нейронної мережі, що налаштована для розпізнавання текстових рядків. Основою цієї системи є нейронна мережа типу LSTM на базі Python від OCRopus, але з деякими змінами, так як Tesseract був розроблений на мові програмування C++.

Зазвичай, щоб розпізнати зображення з одним символом, доцільно використати згорткову нейронну мережу (CNN) [2]. У випадку ж, коли необхідно розпізнати текст довільної довжини, тобто послідовність символів, то для цього використовують RNN, нейронна мережа LSTM, про яку вже було згадано раніше, як раз заснована на RNN.

НМ на базі LSTM добре піддається навчанню символьних послідовностей, але в той же час її швидкість сильно зменшується, коли кількість станів починає зростати. Емпіричним методом було доведено, що LSTM набагато краще може вивчити довгу послідовність, ніж коротку з

багатьма класами. Tesseract версії 4.0.0, розроблений на основі НМ від OCRopus, яка є відгалуженням LSTM C++ і називається CLSTM. CLSTM – це один з варіантів реалізації рекуретної моделі НМ LSTM,яка використовує бібліотеку Eigen, що дозволяє виконувати чисельні обчислення.

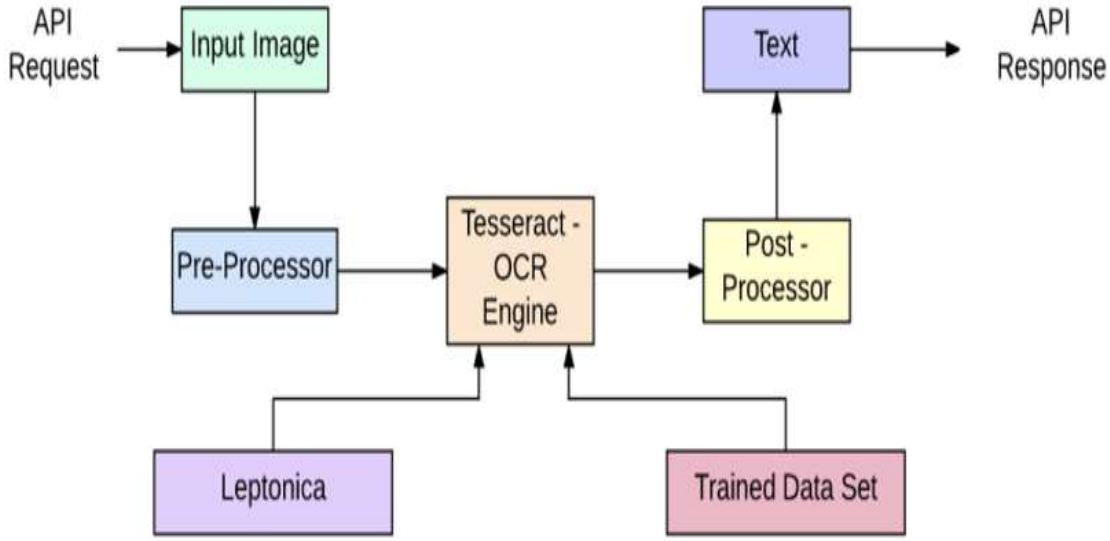


Рисунок 1.9 – Послідовність роботи Tesseract при використанні API

До переваг Tesseract OCR належать.

1. Як вже було зазначено ця OCR система є відкритим програмним забезпеченням, розповсюджується під ліцензією Apache 2.0. Це означає, що кожен може використовувати цю розробку, модифікувати та розповсюджувати її безкоштовно.

2. Підтримка великої кількості мов. Tesseract підтримує понад 100 різних мов та має механізм для додавання підтримки нових мов.

3. Також ця система може використовуватись для розпізнання тексту з різноманітних джерел, таких як, JPEG, PNG, TIFF та інших. Він також може обробляти PDF-файли, скани та інші формати документів, що містять текстову інформацію.

4. Висока точність розпізнавання. Tesseract використовує потужні алгоритми для розпізнавання тексту, що дозволяє досягти високої точності.

Залежно від якості зображень та мови, ця OCR система може досягнути вражаючих результатів. Це можливе за рахунок використання комбінованих методів розпізнавання тексту.

5. Налаштування параметрів. Tesseract надає розширені можливості налаштування через використання параметрів. Це дозволяє користувачам визначати різні параметри розпізнавання, такі як мова, режими обробки, шрифти та інші.

6. Зворотня сумісність. Tesseract зберігає зворотню сумісність зі своїми попередніми версіями, що дозволяє старим проектам та застосункам продовжувати використовувати Tesseract без необхідності внесення значних змін в код.

До недоліків Tesseract OCR слід віднести наступні пункти.

1. Необроблені складні зображення. Tesseract може мати складнощі з розпізнаванням тексту на зображеннях з поганою якістю, незвичайними шрифтами або сильним спотворенням. Він чутливий до шуму, розмитості та інших артефактів, що можуть вплинути на якість розпізнавання.

2. Потреба в налаштуванні. Можливість налаштувати це як плюс так і мінус даної системи. Оптимальна робота Tesseract може вимагати деякого налаштування параметрів, щоб досягти кращої точності та результатів. Це може бути викликом для новачків, які не мають досвіду зі штучним інтелектом або OCR.

3. Обмеження на розпізнавання різних об'єктів. Ця OCR спеціалізується на розпізнаванні тексту і не підтримує автоматичне розпізнавання інших об'єктів, таких як карти, таблиці, графіки тощо.

Але незважаючи на ці недоліки, Tesseract OCR залишається одним з найпопулярніших та потужних рішень для оптичного розпізнавання тексту.

1.7.2 ABBYY FineReader

ABBYY FineReader – це один з найпопулярніших та потужних комерційних продуктів для оптичного розпізнавання тексту та конвертації

документів у текстовий формат. До особливостей цієї системи можна віднести наступні 6 пунктів.

1. Висока точність розпізнавання. ABBYY FineReader використовує потужні алгоритми розпізнавання, що дозволяють досягати високої точності навіть при розпізнаванні складних документів з різними шрифтами, розмірами та форматами;

2. Багатомовна підтримка. ABBYY FineReader підтримує приблизно 210 різноманітних мов. Це дозволяє цій системи розпізнавати тексти з багатьох мов, включаючи мови з кириличними, латинськими, китайськими, японськими та іншими алфавітами.

3. Розпізнавання різних типів документів. Ця OCR може обробляти різні типи документів, включаючи текстові файли, скановані зображення, PDF-файли, факси, фотографії документів та інші формати. Він вміє розпізнавати текст зі структурованих та неструктурованих документів.

4. Збереження форматування. Під час процесу OCR ABBYY FineReader намагається зберегти оригінальне форматування документів, включаючи стилі, таблиці, списки, заголовки, шрифти та інші атрибути. Це допомагає зберегти вигляд та структуру оригінального документа.

5. Редагування та експорт. ABBYY FineReader надає можливість редагувати розпізнаний текст безпосередньо в програмі. Крім того, ви можете експортувати розпізнаний текст у різні формати, такі як Microsoft Word, Excel, PDF, HTML, TXT та інші.

6. Інтеграція з іншими програмами. ABBYY FineReader може бути інтегрований з іншими програмами і сервісами, такими як Microsoft Office, Adobe Acrobat, Dropbox, Google Drive тощо.

До переваг даної OCR можна віднести наступні пункти.

1. Висока точність розпізнавання тексту, що дозволяє отримувати надійні результати.

2. Зручний та інтуїтивно зрозумілий інтерфейс користувача.

3. Широкий спектр функцій для редагування, форматування та експорту документів.Підтримка розпізнавання тексту з різних джерел та форматів документів.

4. Добре розроблені інструменти для роботи з багатосторінковими документами.

До недоліків ABBYY FineReader слід віднести наступне.

1. Вартість: ABBYY FineReader є комерційним продуктом і вимагає придбання ліцензії. Це може бути витратною опцією для окремих користувачів чи невеликих компаній.

2. Обмежена підтримка безкоштовної версії: Безкоштовна версія ABBYY FineReader має обмежені можливості та функціонал порівняно з повною комерційною версією.

ABBYY FineReader як і Tesseract OCR поєднує різі методи розпізнавання тексту, серед них методи машинного навчання (нейронні мережі, глибокі нейронні мережі та інші методи),статистичні методи, ознакові методи, структурні методи.

Ці методи використовуються в поєднанні один з одним для досягнення найкращих результатів розпізнавання тексту в ABBYY FineReader. Використання комплексного підходу та поєднання різних методів дозволяє досягнути високої точності та якості розпізнавання навіть для складних документів з різними шрифтами, розмірами та форматами.

ABBYY FineReader є потужним інструментом для оптичного розпізнавання символів та конвертації документів у текстовий формат, який широко використовується в багатьох галузях, включаючи офісні роботи, архівування документів, дослідження та багато іншого.

1.7.3 Google Cloud Vision AI

Google Cloud Vision AI – це комплексна система від компанії Google, яка надає потужні можливості у сфері розпізнавання та аналізу зображень і відео. Вона використовує набір передових моделей машинного начання для

розпізнавання об’єктів, сцен, тексту, емоцій та багато іншого на зображеннях.

Основні особливості Google Cloud Vision AI можна віднести наступне.

1. Розпізнавання об'єктів. Google Cloud Vision AI може ідентифікувати та класифікувати об'єкти на зображеннях, включаючи людей, тварин, транспортні засоби, будівлі, продукти та інше. Він надає детальну інформацію про виявлені об'єкти, їхнє місцезнаходження та контекстні дані.

2. Розпізнавання тексту. Google Cloud Vision AI може витягувати текст з зображень та розпізнавати його. Він підтримує розпізнавання тексту з різних мов та може виділяти окремі слова, речення та блоки тексту. Крім того, він може розпізнавати печатний текст, рукописний текст та текст на табличках чи навіть в реальному часі з відео.

3. Аналіз зображень: Google Cloud Vision AI надає аналіз зображень, що допомагає розуміти зміст та контекст зображень. Він може розпізнавати сцени, ландшафти, архітектуру, настрої, наявність небажаного контенту та інше.

4. Інтеграція з іншими сервісами: Google Cloud Vision AI може бути легко інтегрований з іншими послугами Google Cloud Platform, такими як Google Cloud Storage, Google Cloud Pub/Sub, Google Cloud Functions та інші. Він також надає різні API та SDK для розробників для зручного використання та інтеграції з власними застосунками [10].

Переваги Google Cloud Vision AI включають наступне.

1. Висока точність розпізнавання та аналізу зображень завдяки використанню передових моделей машинного навчання.

2. Широкий функціонал, що охоплює різні аспекти розпізнавання зображень та тексту.

3. Масштабованість та надійність завдяки хмарній інфраструктурі Google Cloud Platform.

4. Легка інтеграція з іншими сервісами та застосунками.

Недоліки Google Cloud Vision AI є наступне.

1. Вартість використання. Використання Google Cloud Vision AI вимагає плати відповідно до тарифних планів Google Cloud Platform.

2. Залежність від хмарної інфраструктури. Використання Google Cloud Vision AI передбачає доступ до Інтернету та залежність від доступності хмарних сервісів Google.

Загалом, Google Cloud Vision AI є потужним інструментом для розпізнавання та аналізу зображень, який знайшов застосування у багатьох галузях, включаючи медіа, рекламу, безпеку, робототехніку та інші. Його потужність, точність та широкий функціонал роблять його одним із лідерів у сфері обробки зображень за допомогою штучного інтелекту [10].

1.7.4 Apple Vision API

Apple Vision API – це досить молода розробка від Apple, яка дозволяє розробникам під iOS платформу, відмовитись від використання сторонніх фреймворків та додати до своїх застосунків нативний функціонал з розпізнавання тексту та предметів на зображеннях, а також для аналізу зображення.

До переваг Apple Vision API можна віднести наступне.

1. Можливість розпізнавання багатомовних текстів. Це можливо завдяки підтримці досить великої кількості мов.

2. Інтеграція з екосистемою Apple. Vision API надає розробникам зручний спосіб використання можливостей обробки зображень в застосунках для iOS та macOS. Це означає, що кожен може легко інтегрувати розпізнавання об'єктів, тексту та обличь у свої мобільні застосунки, що спрощує розробку та використання функцій обробки зображень [11].

3. Розширений функціонал. Apple Vision API надає розширений функціонал для обробки зображень, включаючи розпізнавання об'єктів, тексту та облич, а також аналіз зображень та перетворення зображень. Це дозволяє розробникам створювати потужні функції обробки зображень у своїх застосунках.

4. Висока продуктивність. Apple Vision API має високу продуктивність, оскільки він оптимізований для роботи на пристроях Apple. Він використовує апаратне прискорення та оптимізовані алгоритми для швидкого та ефективного розпізнавання зображень без значного впливу на продуктивність пристрою.

5. Незалежність від наявності інтернет мережі.

До мінусів Apple Vision API можна віднести такі пункти.

1. Обмеженість платформи. Apple Vision API доступний лише для розробників iOS та macOS, що обмежує його використання до цих платформ.

2. Обмежені можливості порівняно з іншими рішеннями: У порівнянні з іншими масштабними облачними платформами, Apple Vision API може мати обмеженіші можливості та меншу гнучкість. Це може вплинути на певні розширені сценарії розпізнавання зображень. Хоча, враховуючи що компанія Apple постійно вдосконалює свої розробки, цей недолік може вже бути неактуальним через деякий час [11].

3. Залежність від пристрою. Apple Vision API використовує апаратне прискорення пристрою для досягнення високої продуктивності. Оскільки це функціонал стає доступним з конкретної версії операційної системи, то не всі пристрої підтримують даний функціонал.

Також слід зазначити, що в Apple Vision API реалізовано два режим роботи.

1. Швидке розпізнавання тексту. Коли для розпізнавання тексту використовується можливості платформи по виявленню символів для пошуку окремих символів, а потім використовується модель машинного навчання для розпізнавання окремих символів та слів. Цей підхід є аналогічним до традиційного оптичного розпізнавання символів (рис. 1.10) [11].

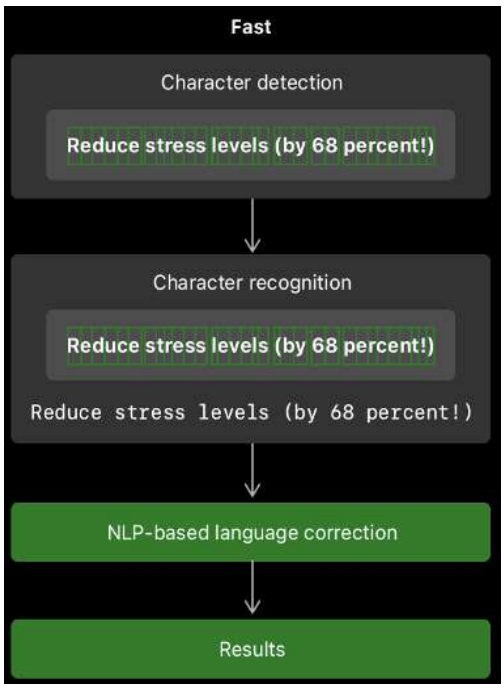


Рисунок 1.10 – Візуалізація роботи швидкого методу розпізнавання текст

2. Розпізнавання тексту з підвищеною точністю. Для цього режиму роботи використовується нейронна мережа, що виконує пошук тексту у вигляді рядків та ліній, а потім проводиться, подальший аналіз для виявлення окремих слів та речень. Цей підхід більш схожий з тим, як люди читають текст (рис. 1.11) [11].

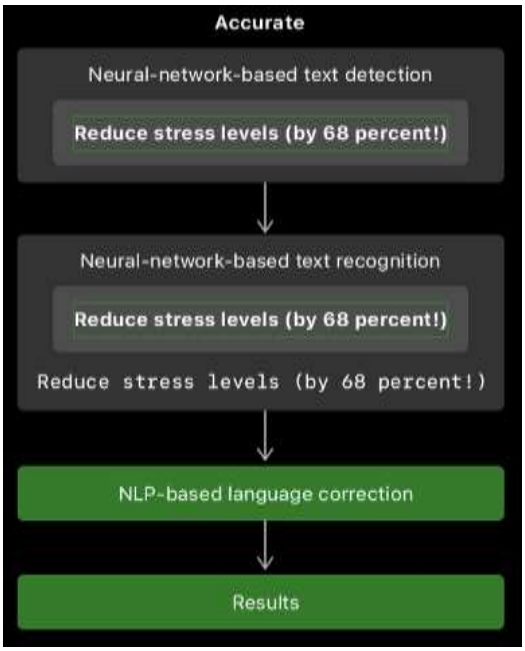


Рисунок 1.11 – Візуалізація роботи точного методу розпізнавання текст

1.8 Порівняння розглянутих реалізацій OCR систем

В таблиці 1.1 наведені деякі з показників, що демонструють на скільки часто оновлюються OCR системи, охоплення мов, можливість інтеграції з іншими системами тощо.

Усі з розглянутих систем мають підтримку української мови та багатьох інших мов. Також вони дозволяють налаштувати свою реалізацію в залежності від потреб користувача.

Таблиця 1.1 – Порівняння розглянутих OCR систем

	Tesseract	ABBYY FineReader	Google Cloud Vision API	Apple Vision API
Дата останньої стабільної версії	01.04.23	10.11.22	16.12.22	15.10.20
Цінова політика	Безкоштовно	Платно, за виконанні операції	Платно, за виконанні операції	Безкоштовно
Робота офлайн	Так	Ні	Ні	Так
Кількість підтримуваних мов	100+	210	110	100+
Підтримка української мови	+	+	+	+
Мови програмування	Python, JS, TypeScript, C, C++, C#, PHP, Java, Swift, Kotlin, Go, Ruby	C, C++, .NET, Delphi, Java, JS	Python, C#, PHP, Java, Swift, Kotlin, Go, Ruby, Node.js	Swift, ObjC
Формат результату	Текст, TXT, TSV, PDF	Текст	Текст	Текст, PDF
Інструменти перед-обробки	Так	Так	Так	Так
Здатність пристосовуватись до конкретного сценарію використання	Так	Так	Так	Так

Розглянуті системи, окрім ABBYY FineReader, мають можливість виводити результат розпізнавання в різних форматах. Але без недоліків не обійшлося – за усі ці плюси, переваги та гарну ефективністю, призначається висока плата, яка унеможливлює використання таких систем на постійній основі. В додаток до усього вище сказаного, важливим моментом є можливість виконувати обробку та розпізнавання тексту на зображенні, не маючи інтернет з'єднання, що є неможливим для хмарних застосунків.

У зв'язку з високою ціною, та через залежність від наявності інтернет з'єднання використання в цій роботі Cloud Vision API та ABBYY FineReader є не доречним

Через високу вартість та залежність від інтернет з'єднання Google Cloud Vision API та ABBYY FineReader їх використання в даній роботі не є доречним. Модуль обліку персональних витрат буде розроблятися для платформи iOS, тому було вирішено використати Apple Vision API та порівняти "нативну" реалізації OCR від Apple з OCR Tesseract.

1.9 Постановка мети та задачі дослідження

OCR технологія – це насамперед, досить просто та стара технологія, розвиток якої почався ще в 40-их роках двадцятого століття, яка враховує тільки базовий алгоритм. Слід зазначити, що найпростішу OCR бібліотеку цілком реально написати самостійно, використовуючи підручники-довідники, і вона, навіть, частково буде працювати, але тільки в ідеальних умовах, де чудова якість фотографії, ідеальний нахил символів і т.д. При зміні значення, хоча б одного, додаткового параметру, наша міні OCR вже не зможе розпізнати текст.

Саме тому величезне значення в OCR має попередня обробка зображення, за допомогою якої виконується "ідеальне налаштування" та подальша обробка, яка допомагає підвищити шанс розпізнавання.

Таким чином, метою кваліфікаційної роботи є дослідження технології OCR Tesseract на базі нейронної мережі LSTM для побудови системи персонального обліку витрат.

Для досягання поставленої мети необхідно вирішити наступні задачі.

1. Проаналізувати існуючі алгоритми розпізнавання тексту та визначити їх недоліки.
2. Розглянути IRC функції, які застосовують в OCR технологіях.
3. Дослідити поєднання OCR технологій та нейронних мереж.
4. Проаналізувати існуючи системи персонального обліку.
5. Обрати оптимальний метод розпізнавання тексту для реалізації системи персонального обліку.

2 ПРОЕКТУВАННЯ СИСТЕМИ ПЕРСОНАЛЬНОГО ОБЛІКУ ВИТРАТ

2.1 Огляд існуючих систем персонального обліку витрат

В умовах сьогодення для успішного досягнення своїх цілей необхідно визначити, яким чином вони можуть бути реалізовані та які ресурси для цього потрібні. Фінансова складова часто є першочерговим пунктом, без якого здійснення тих чи інших планів не є можливим. Саме тому більшість людей бажають витрачати менше та економити більше, і в цьому їм допомагають системи для обліку витрат.

Наразі на ринку програмних продуктів є безліч застосунків, які дозволяють вести облік фінансів [13]. Розглянемо декілька з них.

1. Money lover – це один популярних мобільних застосунків, яким я користувався, для ведення обліку витрат. Його функціонал дозволяє записувати усі витрати та надходження, встановлюючи для кожного запису відповідну категорію, а також вести бюджет. Але в цьому застосунку немає можливості вносити записи через фотографію чеку, тобто використовувати OCR технологію. А також головним його недоліком є те що весь функціонал максимально обмежений в безкоштовній версії, а місячна підписка коштує досить дорого.

2. Expensify - Expense Tracker – це застосунок, який схожий на Money lover, але з одною великою відмінність – це можливість вносити витрати та автоматично визначати категорію за допомогою сканування фотографії чеку, але цей функціонал стає доступним користувачу, тільки після оформлення преміум підписи, а також звернувши увагу на відгуки в магазині застосунків, можна зробити висновок, що застосунок працює досить погано.

3. Wallet – це стандартний застосунок для ведення персонального обліку, що дозволяє створювати бюджет, вести облік витрат та доходів, але

він також, як і Money lover не має функціоналу вносити витрати за допомогою фотографії чеку, хоча й дозволяє прикріпити до запису чеку.

Проаналізувавши описані вище застосунки можна зробити висновок, що досить мала їх кількість має реалізацію OCR функціоналу, а ті застосунки, в яких він є, відкривають доступ до нього тільки після купівлі підписки. Також слід зазначити, що майже всі застосунки для ведення обліку витрат, мають обмежений функціонал для користувачів, які не купують підписку.

2.2 Модель системи персонального обліку витрат

Модуль системи персонального обліку витрат буде приймати на вході фотографію товарного чеку з магазинів-супермаркетів, в асортименті яких, є товари різноманітних категорій – починаючи від різноманітних хлібобулочних товарів та напоїв і закінчуючи товарами побутового призначення (рис. 2.1).

Рисунок 2.1 – Приклад товарного чеку

Вибраний мною напрям буде реалізований за допомогою створеного навчального набору даних, який буде включати список товарів. Великим плюсом такої реалізації є те, що в будь-який момент можна змінити сферу, з

якої буде скануватись чек. Наприклад, щоб модуль розпізнав текст з касового чеку аптеки, ресторану або кафе, потрібно буде змінити набір навчальних даних, або ж доповнити існуючи набір, що дозволить використовувати модуль в різноманітних сферах, хоча це може збільшити відсоток невірних розпізнавань.

На вхід модуль буде приймати фотографію чеку, а вихідні дані будуть містити список товарів та ціною (рис. 2.2).

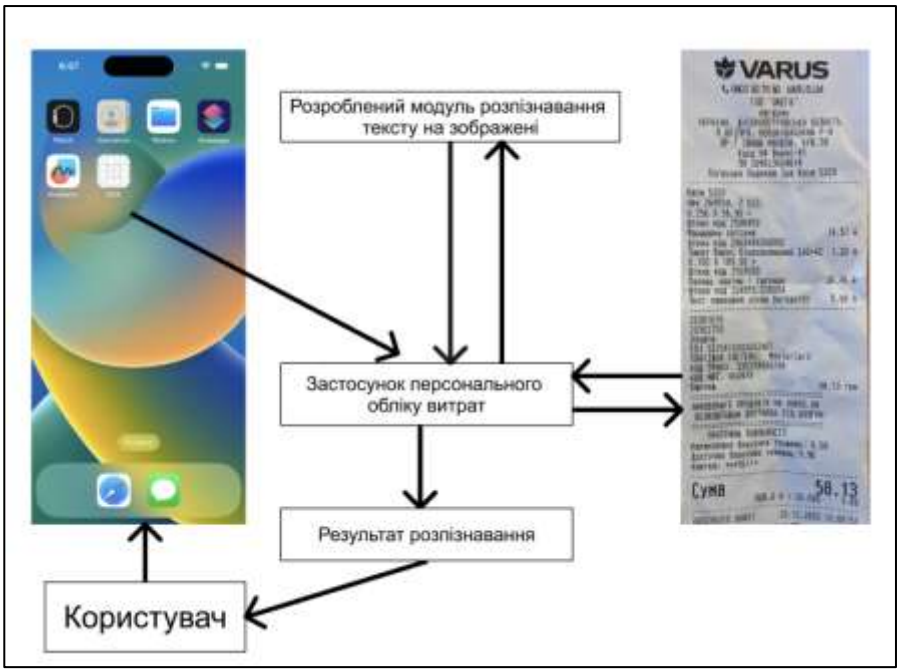


Рисунок 2.2 – Модель системи

Так як на чеку присутня інформацію не тільки про куплений товар та ціну, а ще й інформація про магазин, ім’я касира та номер каси, а також спосіб оплати та багато іншої непотрібної, в даному випадку, інформації.

Для вирішення цієї проблеми можна обрізати фотографію чеку таким чином, щоб залишалась тільки інформація про куплений товар та ціна, але це може зменшити можливість універсального застосування модуля. Іншим варіантом, є проведення фільтрації, що буде відсікати записи, що не є товарами, але цей варіант також не дає сто відсоткової надійності, бо якщо під час розпізнавання система не зможе розпізнати ціну товару, по якій буде

виконуватись фільтрація, то відповідно товар буде відсіяно і він не потрапить до результуючої таблиці.

Щоб виконати дане завдання, необхідно реалізувати наступні кроки.

1. Виконати алгоритм видалення перекосу (модифікувати зображення, так щоб фотографія чеку була максимально у вертикальній орієнтації.
2. Щоб покращення точності розпізнавання, слід здійснити попередню обробку зображення, що включатиме підвищення якості.
3. Виконання процесу розпізнаванняя тексту [2].
4. Виконати сортування товарів, за раніше визначеними категоріями.
5. Розрахунок загальної вартості для знайдених категорій.

2.3 Опис послідовності функціонування системи

2.3.1 Алгоритм видалення перекосу зображення

При опрацюванні фотографії чеку може виникнути ситуація, коли чек буде розташований не чітко вертикально, а в межах від $-\frac{\pi}{2}$ до $\frac{\pi}{2}$. Будь-який OCR модулю буду показувати найкращі результати в розпізнаванні тексту тільки тоді, коли рядки розташовані в максимально горизонтальному положенні. Для досягнення максимальних результатів у розпізнаванні, необхідно виправити нахил – а саме повернути зображення, так, щоб зображення чеку було розташоване вертикально.

Для вирішення завдання корекції нахилу чеку можна використати алгоритм Кенні, а також перетворення Хафа. Розглянемо їх більш детально.

Алгоритм Кенні представляє собою багатоступінчатий метод знаходження межі на зображені, який був розроблений у 1986 році Джоном Кенні.

Кенні першим впровадив поняття “не максимумів” (англійською non-maximum). Що ж мається на увазі під цим поняттям? Це означає, що лише ті пікселі де досягається локальне максимальне значення градієнта в порівнянні з вектором градієнта, а решта значень прирівнюються до нуля. Цей

призводить до отримання чіткішого результату. Слід зазначити, що, за винятком конкретних ситуацій, складно знайти алгоритм визначення границь, який показував би більш ефективно, ніж алгоритм Кенні. Головною метою Кенні було створення алгоритму для виділення контурів зображення, який відповідав б певним вимогам, які наведені нижче.

1. Максимально ефективне визначенні границь (збільшення співвідношення сигнал-шум).
2. Визначення межі з високою точністю.
3. Для кожної межі повинен бути тільки один відгук.

На основі цих вимог і проектувалась функція, яка використовується для пошуку оптимального лінійного оператора для згортки зображенням.

Розроблений Джоном Кенні алгоритм, не обмежується лише обчисленням градієнта для зображення, яке обробленого фільтром Гаусса. Він також локально виділяє не максимальні точки, які знаходяться поруч з межею.

Алгоритм Кенні для виявлення меж об’єкту використовує два етапи: на першому виконується обчислення градієнта зображення та видалення локально не максимальних точок. На другому етапі алгоритм також використовує інформацію про напрям границі для видалення точок, які знаходяться поруч із межею, але не є її частиною. Після цього алгоритм використовує порогову фільтрацію для видалення слабких меж, при цьому частина межі опрацьовується як цілий об’єкт. Якщо показник градієнту на певному фрагменті буде більше ніж верхній поріг, то ця частина межі вважається "допустимою" границею. А відповідно, в тих випадках, коли значення градієнта падає нижче верхнього порогу, фрагмент вважається границею до тих пір, поки не досягне нижнього порогу. У випадку, якщо на фрагменті немає точок із значенням, що перевищує верхній поріг, фрагмент вилучається. Використання такого гістерезису сприяє зменшенню кількості розривів у вихідних межах. Інтеграція методу видалення шумів в цей алгоритм, збільшує стійкість результату, але в той же час підвищує затрати

на обчислення і може викликати спотворення та втрату деталей кордонів. Наприклад, цей алгоритм може спричиняти заокруглення кутів об'єктів та руйнування границь у точках їхніх з'єднань.

Алгоритм розроблений Джоном Кенні поділяється на наступні ключеві етапи.

1. Конвертування кольорового зображення у чорно-біле зображення.
2. Згладжування зображення. Цей процес проводиться за допомогою фільтра Гаусса. Фільтр Гауса зменшує контраст між сусідніми пікселями, що робить зображення більш плавним
3. Пошук градієнтів [2]. Для пошуку градієнтів на зображенні в різних напрямах, алгоритм Кенні використовує чотири різноманітні фільтри, завданням яких є знаходження границь по горизонталі, вертикалі та діагоналі. Далі використовується оператор, що допомагає виявити межі, наприклад оператор Собеля або Шарра. Наприклад, за допомогою оператора Собеля алгоритм визначає значення градієнта в горизонтальному та вертикальному напрямах. З отриманих значень обчислюється кут напрямку границі. Округливши кут до одного з чотирьох можливих значень можна визначити в якому напрямку проходить границя (рис 2.3).

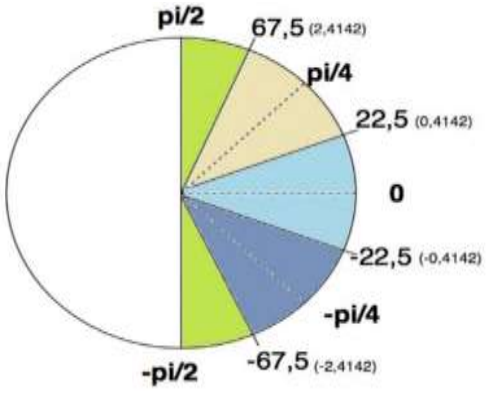


Рисунок 2.3 – Апроксимація кута градієнта

4. Заглушення немаксимумів [2]. Основною метою цього етапу є перетворення “згладжених” контурів зображення, отриманих з величин градієнту, в чіткіші межі. Щоб досягти цього, використовується метод

збереження всіх локальних максимумів градієнта на зображенні, при цьому всі інші значення прирівнюються до нуля. Кожен піксель зображення градієнта обробляється згідно з наступними кроком алгоритму:

1) Для визначення, чи є піксель границею, алгоритм Кенні порівнює значення градієнта з значенням градієнта пікселів в його околиці, в позитивному та негативному напрямку.

2) Якщо значення градієнта поточного пікселя більше значення сусідніх пікселів, то залишаємо поточне значення, в інакшому випадку присвоюємо пікселю значення нуль.

5. Фільтрація з подвійним порогом. Визначення потенційних меж відбувається за допомогою порогів.

По завершенню минулого етапу, тобто після того як були виявлені усі граничні точки, їх потрібно оцінити, щоб визначити, чи є вони справжніми границями зображення. Більшість граничних пікселів будуть реальними границями, але деякі можуть бути помилковими.

Для того, щоб розрізняти граничні пікселі на “сильні” та “слабкі”, алгоритм Кенні використовую фільтрацію з подвійним порогом. Пікселі, у яких значення “сили” перевищує високий поріг, ідентифікуються як “сильні”, граничні пікселі зі значенням сили меншим за низький поріг видаляються, а граничні пікселі, що за показником сили знаходяться між нижнім та верхнім порогом вважаються як “слабкі”.

Перетворення Хафа – це алгоритм, який використовується для виявлення геометричних елементів, за допомогою параметрів, на зображенні, наприклад, таких як кола, еліпси та прямі та інші. Він працює створюючи сітку точок на зображенні, а потім “голосуючи” за точки, які більш імовірно належать до конкретного геометричного елементу, відповідно точки, що отримають найбільше “голосів” будуть вважатися точками цього геометричного елементу. Головна мета цього алгоритму – це вирішення проблеми групування граничних точок.

Положення прямої на площині можна описати рівнянням $y = kx + b$, яке може бути визначене за допомогою пари точок, які мають різні координати. Хоча пряма і можна описати рівнянням $y = kx + b$, на практиці зручніше використовувати два інші параметри: p і θ. Параметр p – це довжина перпендикулярної лінії, проведеної до прямої з початку координат, а θ – кут між цим перпендикуляром та віссю абсцис, або інакше кажучи, кут між перпендикуляром та віссю Ox (рис. 2.4). Площина, на якій задані параметри p і θ, які описують пряму в двох вимірному просторі, називають, або фазовим простором, або простором Хафа.

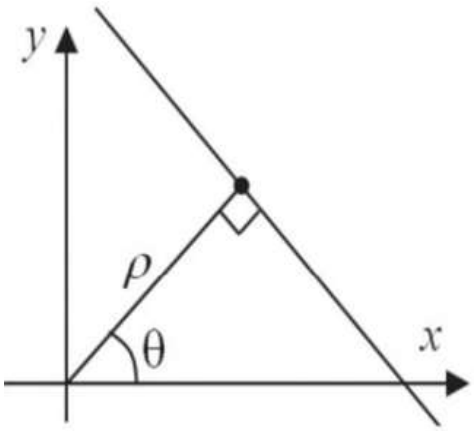


Рисунок 2.4 – Опис прямої на площині за допомогою параметрів p і θ

На декартовій площині, через одну точку, можна провести безкінечну кількість прямих. Якщо представити, що точка, яка зображена на рисунку 2.4 моє координати x_0, y_0, то усі прямі, що будуть проходити через неї можна буде описати рівнянням

$$p(\theta) = x0\ cos + y0\ sin.$$

Для кожної точки на зображенні, що відповідає прямій, існує синусоїдальна крива в просторі (p, θ). Ця крива показує, як змінюється відстань від точки з координатами (x_0, y_0) до прямої в залежності від кута нахилу прямої.

Якщо об’єднати синусоїди, які відповідають двом точкам на декартовій площині, то точка перетину цих синусоїд, в просторі Хафа, буде відповідати характеристикам прямої, що містить ці дві точки. Як підсумок, набір точок, за допомогою яких формується пряма лінія, може бути описаний синусоїдами, які перетинаються в точці с координатами (p_0, θ_0) для відповідної прямої. Таким чином, задачу виявлення колінеарних точок можна зводити до задачі виявлення перетину синосуїд

Якщо синусоїди, що відповідають двом точкам декартової площини, накласти одну на одну, то точка (в просторі Хафа), де вони пересічуться, буде відповідати параметрам прямої, що проходить через обидві ці точки. Таким чином, ряд точок, які формують пряму лінію, визначають синусоїди, що перетинаються в точці параметрів (p_0, θ_0) для цієї лінії. Тому, проблема виявлення колінеарних точок може бути зведена до проблеми виявлення кривих, що перетинаються [2].

Відповідно для кожної точки з координатами (p_0, θ_0) в просторі (p, θ) знайти лічильник, який буде відповідати набору точок, по координатах, (x, y), які знаходяться на прямій.

$$p_0 = xcos + ysin.$$

Отже, для визначення параметрів конкретної прямої, на фазовому просторі точки з найбільшою концентрацією збігів. Приклад застосування комбінації вищеописаних методів наведено на рисунку 2.5 – 2.7.

Рисунок 2.5 – Початкове зображення

Рисунок 2.6 – Вигляд початкового зображення після виділення ліній перетворенням Хафа

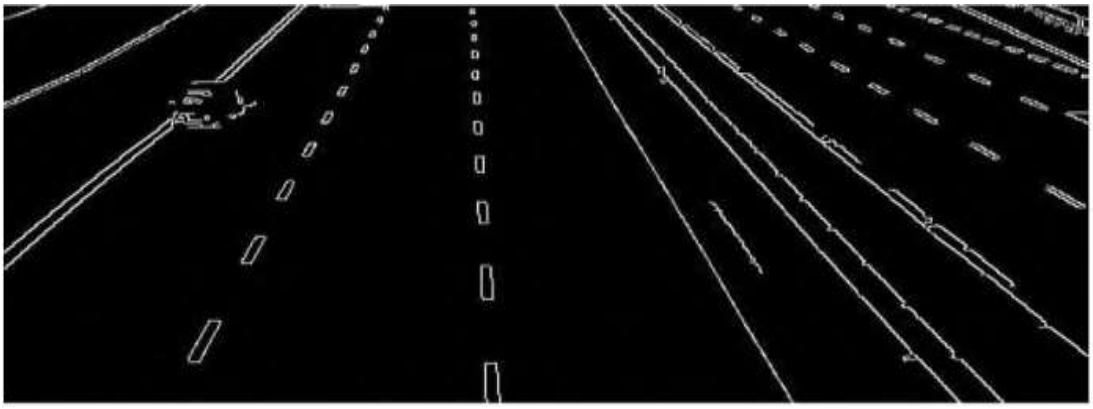


Рисунок 2.7 – Вигляд початкового зображення після обробки алгоритмом Кенні

В наслідок застосування перетворення Хафа та алгоритму Кенні буде визначені лінії, які визначають межі чеку. Наступним кроком буде виконуватись обчислення кута нахилу отриманих ліній, на основі яких, буде виконуватись корекція нахилу зображення, щоб отримати горизонтальне розташування тексту.

Дві головні умови, для ефективної роботи модуля, необхідно, щоб чек займав максимум місця на зображенні, а також, щоб фон зображення (фон на якому зображений чек), був однорідним і чітко відокремлювався від чеку, це необхідно, щоб правильно провести бінаризація зображення для подальшого розпізнавання. В такому випадку збільшується можливість успішної

подальшої обробки фотографії. Алгоритм усунення перекосу зображено на рисунку 2.8.

Рисунок 2.8 – Алгоритм усунення перекосу зображення

2.3.2 Підготовка зображення перед подальшим використанням

Перед розпізнаванням, зображення піддають попередній обробці, щоб покращити його якість. Так як текст має бути максимально добре видно на фоні зображення, його виділяють за допомогою бінаризації. Бінаризація – це процес модифікації зображення у чорно-біле за допомогою порогу. Цей процес виконується в два етапи:

– модифікація початкового зображення у напівтонове (або інакше кажучи в градацію сірого);

– виконати бінаризацію з накладанням порогу.

Метод Оцу є одним з найбільш популярним алгоритмом для виконання бінаризації. Алгоритм Оцу – це метод, який призначений для бінаризації

напівтонових зображень за допомогою порогу. В своїй основі він використовує класифікатор, який розділяє пікселі на два типи: фонові та корисні. Алгоритм знаходить оптимальне значення порогу, таким чином, щоб мінімізувати внутрішньокласову дисперсію. Також слід зазначити, що було доведено, що чим меншою буде дисперсії в середині класів, тим більшою буде дисперсія між класами, це дозволяє збільшити точність бінаризації. Також слід зазначити, що є покращена версія методу Оцу, яка називається мульти-Оцу, різниця в тому, що покращений алгоритм підтримує багаторівневу бінаризація зображення.

Для того аби знайти оптимальне значення порогу, метод Оцу виконує обчислення суми внутрішньокласових дисперсій. Для цього використовується формула, що наведена нижче:

$$\sigma_w{}^2(t) = \omega_1(t)\sigma_1{}^2(t) + \omega_2(t)\sigma_2{}^2(t)$$

де ω_i – це вагові коефіцієнти, що виражають імовірність класів, різділених порогом t, $\sigma_i{}^2$ – дисперсія цих класів [2].

Точна роздільність порогового методу прямо пропорційна ймовірностям класів. Тобто, чим різкіше розмежовані класи, тим краще спрацьовує метод-Оцу. Приклад зображено на рисунку 2.9.

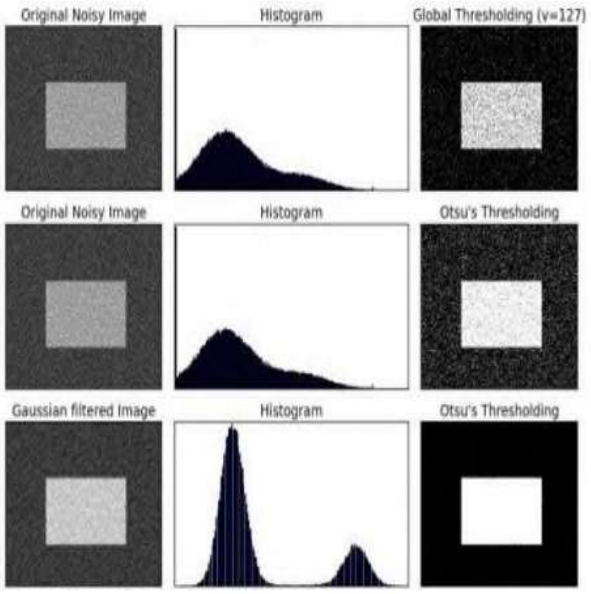


Рисунок 2.9 – Бінаризації зображення з визначенням порогу методу-Оцу

Перед тим як передати зображення до OCR-модуля, його необхідно підготувати, щоб текст був добре та чітко видимим для OCR-системи. Для цього над зображенням проводиться попередня сегментація, що містить наступні етапи.

1. Визначити ядро згортки, яке застосовується в якості фільтру для розмиття, виділення границь, збільшення різкості та інших операцій обробки зображення. Ядро (англ. kernel) – це є квадратна матриця, зазвичай невеликого розміру, яка використовується для обчислення для кожного пікселю нового значення. Опорна точка ядра, (зазвичай – це центральний елемент матриці) визначає, який вплив матиме кожний елемент ядра на нове значення пікселя.

2. Далі необхідно виконати операцію розширення. Алгоритм проходиться циклом по всіх елементах, тобто пікселях, зображення і на кожен з них накладається ядро та обчислюється локальний максимум, для всіх пікселів, які “покриті” ядром, після цього відбувається заміна значення для пікселів, які є опорними точками на значення локального максимуму. Тобто, іншими словами на вході алгоритм отримує бінаризоване зображення, яке являє собою 2D-матрицю, значення елементів якої може бути, або один, або нуль, після цього накладається ядро-фільтр. Якщо в області, яку накриває елемент ядра зі значення один, є хоча б один піксель, що має значення одиниці, то елемент початкового зображення, що відповідає опорній точці, отримує значення один. Це процес допомагає збільшити світлі ділянки, на зображенні, а також в результаті розширення рядки тексту перетворюються у блоки прямокутної форми, що зображено на рисунках 2.10 – 2.11.

3. Останній етап сегментації зображення – це виділення контурів. Що таке контур? Це крива, яка з’єднує всі неперервні точки, що проходять поряд з межею, що має однаковий колір. Розпізнавання контурів, є одним з найефективніших механізмів, підчас роботи з зображенням у двійковому вигляді, саме для того щоб було простіше отримати координати контурів

об’єктів і було виконано минулі кроки. Після цього кроку початкове зображення можна розбити на набір виділених фрагментів.

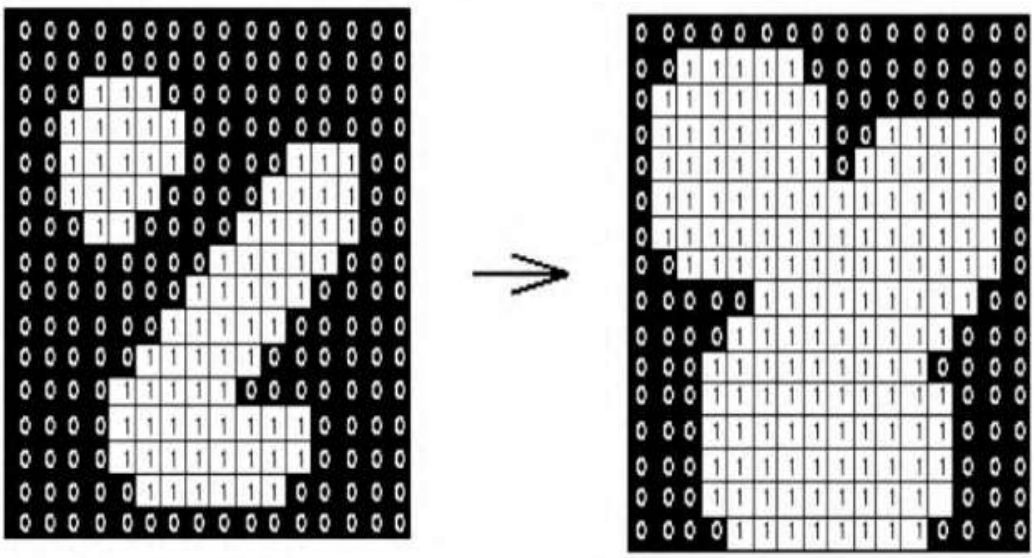


Рисунок 2.10 – Результат виконання операції розширення зображення у 2D-вигляді ядром 3x3

Рисунок 2.11 – Результат операції розширення на прикладі фотографії чеку

2.3.3 Співставлення назви товару та ціни

Одна з проблем при розпізнаванні тексту на касових чеках полягає в тому, що інтервал між рядками, між товарами та між словами однаковий, що робить майже неможливим візуально розрізнити, де закінчилась назва одного товару та почалась назва іншого. Через те, що на чеках відсутні будь-які роздільні знаки, процес виділення контурів повертає усі товари у вигляді одного блоку, що зображено на рисунку 2.12. Саме тому необхідно спеціальним чином виконати розділення цього блоку на частини, тобто окремі товари.

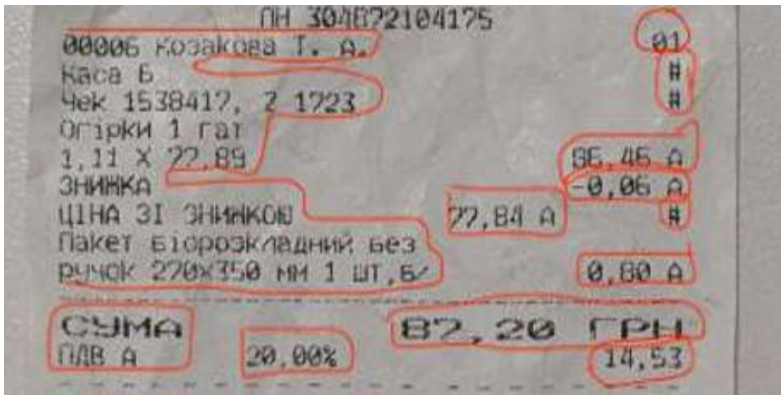


Рисунок 2.12 – Виділення контурів тексту на чеку

В нашому випадку, за орієнтир можна спробувати взяти розташування блоку ціни товару, адже у них відношення між висотою тексту та відстанню до сусідніх блоків досить велике, щоб спробувати віднести їх до різних контурів, а також, враховуючи, що кожен товар гарантовано має ціну, відповідно має орієнтир.

Щоб правильно розділити блок товарів, потрібно, після розпізнавання тексту виконати декілька операцій.

По-перше, необхідно знайти блок, що відповідає номеру чека, та видалити все, що знаходиться вище цього блоку. По-друге, треба знайти усі блоки, що є ціною, або містять ціну, відфільтрувати їх по вісі *Oy* та також видалити їх з основного блоку об’єктів. І останнім етапом, потрібно для кожного блоку, що залишився в початковому масиві перевірити умову, що

нижні координати блоку верхньої ціни знаходяться вище, ніж верхні координати блок товару, а також, що нижні координати товару розміщені не нижче ніж блок ціни. Алгоритм цього процесу зображено на рисунку 2.13.

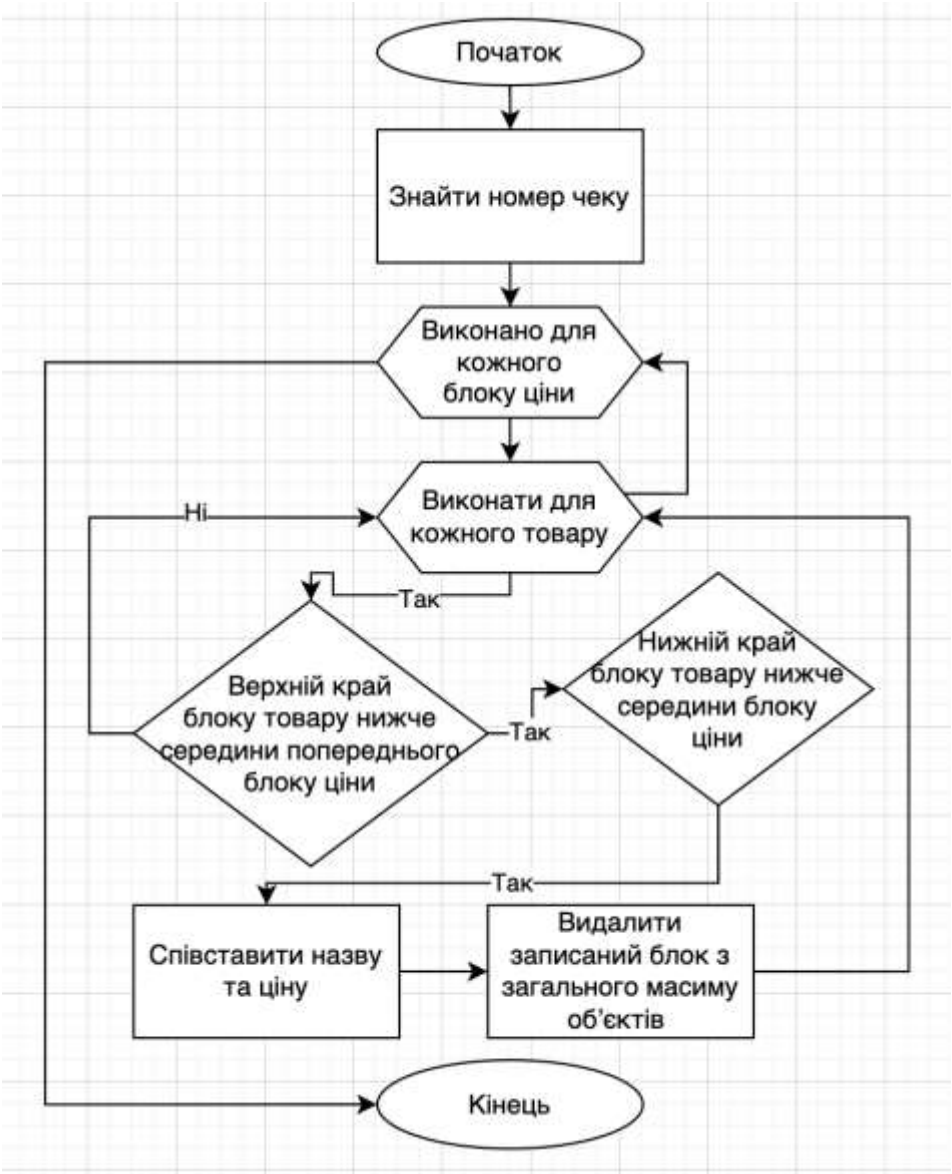


Рисунок 2.13 – Алгоритм співставлення назви товару та ціни

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір платформи

Беручи до уваги, що застосунки для ведення персонального обліку витрат, зазвичай, використовуються на мобільних пристроях, таких як смартфони або планшети, то логічним буде розробляти таку систему саме під ці девайси.

Далі слід визначитися з платформою, під яку буде виконуватись розробка. На даний момент, на ринку представлено тільки дві платформи: Android та iOS/iPadOS, раніше були ще пристрої на Windows Mobail, але їх давно не випускають і ця платформа є “мертвою”. Враховуючи, що Tesseract має API під усі популярні платформи, і те що я працюю розробником застосунків під iOS/iPadOS, а також те що раніше в минулих розділах зазначалося, що буде проводитись порівняння “нативної” реалізації OCR з opensource рішенням у вигляді Tesseract, саме тому вибір було зроблено в бік платформи Apple.

3.2 Вибір мови програмування

Для розробки мобільних застосунків під платформу iOS/iPadOS є досить багато різноманітних мов програмування та фреймворків, навіть незважаючи на те, що усі операційні системи від Apple мають закриту та досить обмежену у доступі для розробників архітектуру.

Мови за допомогою яких можна розробляти застосунки під iOS/iPadOS:

– Objective-C – це перша мова програмування, яка використовувалась для розробки під Apple. Це високорівнева, об’єктно-орієнтована мова програмування, яка в своїй основі має суміш мов програмування C та

Smalltalk. Вона була представлена у 1986 році. Ця мова програмування має специфічний синтаксис і досить великий поріг входу. Активно використовувалась, приблизно, до 2015 – 2016 року.

– Swift – мова програмування, яка прийшла на заміну Objective-C. Є наймолодшою мовою програмування в цьому списку. Вона була представлена компанією Apple у 2014 році і підходить для розробки під усі наявні платформи Apple. Swift – це, також як і Objective-C, високорівнева, об'єктно-орієнтована, компільована (використовую LLVM компілятор) мова для написання мобільних додатків. Swift поєднав у себе краще, що є в C та Objective-C.

– C# (Xamarin) – це фреймворк для розробки крос-платформених застосунків під Android та iOS з використанням мови програмування C#. Цей фреймворк мав досить гарну продуктивність, яка була близька до нативної розробки, також ще одним плюсом було те, що під час розробки використовувались нативні UI-елементи під кожну з платформ, але з недоліків, те, що для розробки з використанням Xamarin все одно потрібно знати нативні технології для кожної з платформ, також мінусом є значно більший розмір застосунку в порівнянні з нативною розробкою. Можливо саме тому, цей фреймворк так і не отримав значної популярності. [12]

– JavaScript (React Native) – це ще один фреймворк для розробки крос-платформених застосунків під Android та iOS з використанням мови програмування JavaScript. Цей фреймворк розробила у 2015 році компанія Facebook у якості проекту з відкритим вихідним кодом. Для розробки використовувались власні UI-елементи, що з одного боку дозволяло веб-розробникам швидко змінити спеціалізацію, але в той же час цей фреймворк не могі надати тієї ж плавності та анімацій в мобільних застосунках як нативні рішення. Також з недоліків слід зазначити гіршу в порівнянні з нативною розробкою продуктивність застосунків та більшу складність розробки.

– Dart (Flutter) – це, напевне, одне з найуспішніших крос-платформених рішень на даний момент. Dart – це мова програмування, представлена у 2011 році, яка досить сильно схожа на Java та JavaScript. Застосунки написані на Flutter мають близьку до нативних застосунків продуктивність та швидкість роботи. В якості UI використовуються власні компоненти фреймворку, це несе ті ж самі плюси та міни що й у JavaScript.

Проаналізувавши всі наявні варіанти мов програмування під платформу Apple, було визначено, що найкращим рішенням буду використати мову програмування Swift, так як вона є досить новою, є “рідною” мовою для обраної платформи і надає увесь необхідний функціонал для реалізацію поставлених завдань.

3.3 Додаткові інструменти для реалізації вибраних алгоритмів

OCR Tesseract в деяких випадках може розпізнавати текст не виконуючи попередню підготовку зображення, але це можливо тільки в ідеальних умовах та зі стандартними шрифтами, а також обмеженою кількістю мов, а враховуючи що Tesseract буде використовуватись для розпізнавання тексту на касових чеках, в яких зазвичай використовується шрифт Courier new та українська мова, а якість друку бажає бути кращою, то без попередньої підготовки та покращення зображення не обійтись.

Для попередню підготовку зображення буде використано opensource бібліотека GPUImage написана під платформу iOS/iPadOS. Це рішення дозволяє використовувати значний набір алгоритмів та фільтрів для обробки та модифікації зображення або відео. Фільтри, які будуть використані в цій роботі наведені нижче:

– GPUImageAdaptiveThresholdFilter – фільтр для переведення зображення у градацію сірог;

– GPUImageCannyEdgeDetectionFilter – алгоритм Кенні;

– HoughTransformLineDetector – перетворення Хафа;

– GPUImageLuminanceThresholdFilter – фільтр для бінаризації зображення за допомогою порогу Оцу;

– GPUImageRGBDilationFilter – операція розширення зображення;

– GPUImage3x3ConvolutionFilter – фільтр, ща накладає матрицю згортки на зображення.

Це невеликий перелік алгоритмів та фільтрів, які доступні в цій бібліотеці.

Також ця бібліотека, може бути корисною та підвищити ефективність роботи під час попередньої обробки зображення у парі з “нативною” OCR від Apple, що також буде перевірено на практиці.

Слід зазначити, що як запевняє автор, в деяких випадках ця бібліотека працює в два-три рази швидше ніж стандартне рішення від Apple у вигляди CoreImage.

3.4 Розробка системи персонального обліку витрат

Для написання мобільного застосунку було використано IDE XCode. Так як ціль моєї роботи – це порівняння OCR Tesseract з стандартною реалізацією OCR від Apple, а також реалізувати та дослідити алгоритм описаний раніше, відповідно досконала розробка графічного інтерфейсу має другорядне значення і не призначена для широкого використання. В подальшому це направлення можна досить швидко вдосконалити.

3.4.1 UI системи обліку витрат

Враховую те, що розроблений модуль мобільного застосунку призначений тільки для дослідження, описаного раніше, алгоритму, демонстрації працездатності систем та для порівняння написаного алгоритму та нативної OCR системи від Apple, відповідно блок схема алгоритму роботи застосунку буде виглядити так як наведено на рисунку 3.1.

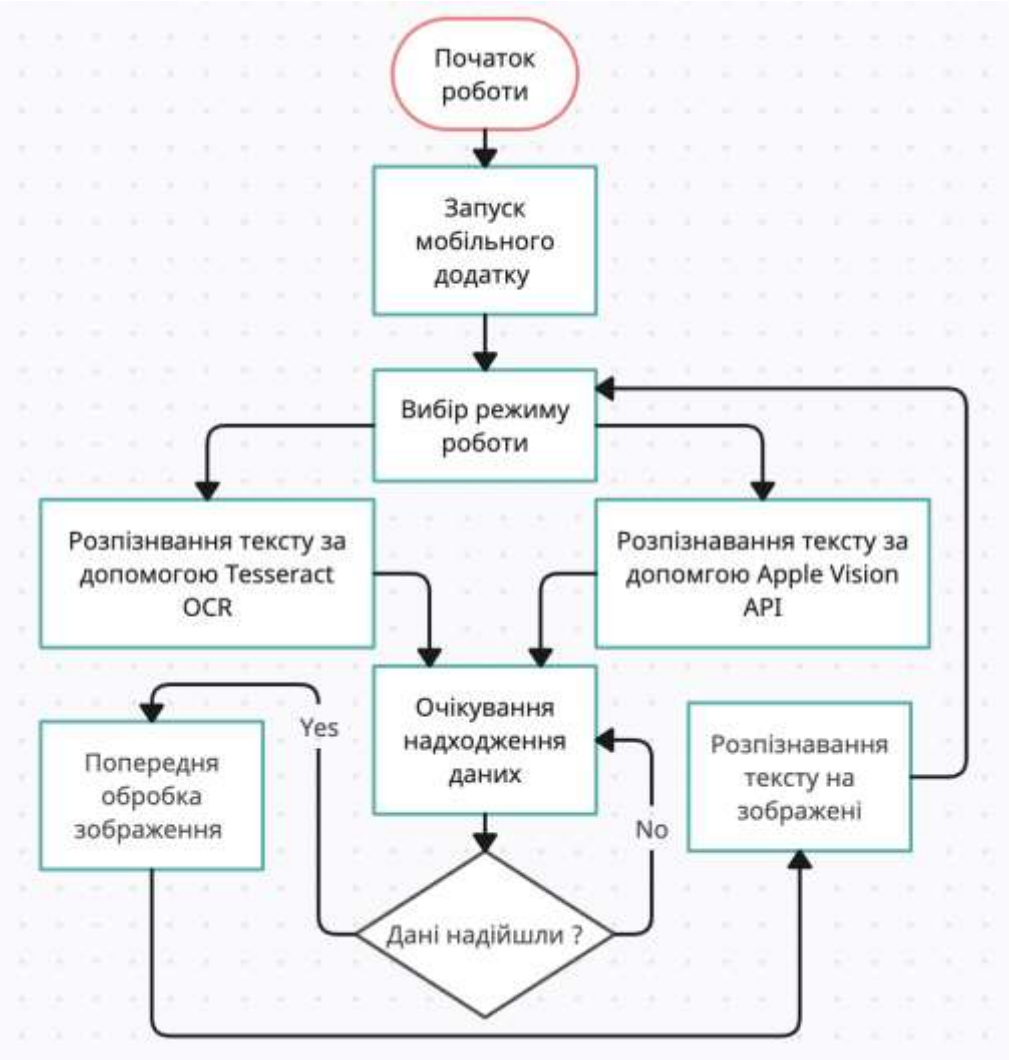
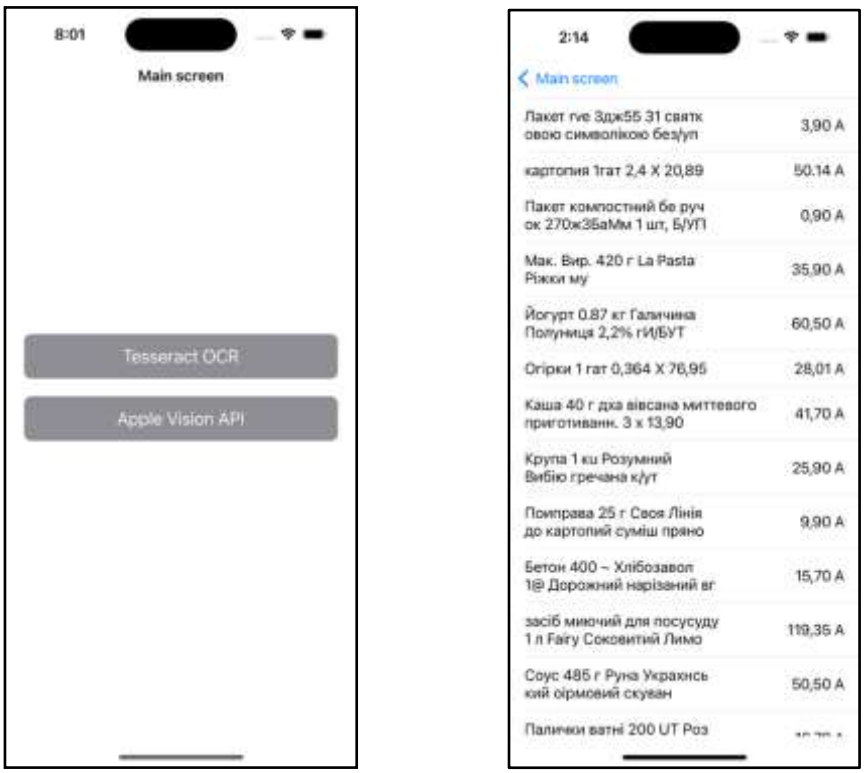


Рисунок 3.1 – Блок схема алгоритму роботи тестового застосунку

Мобільний застосунок буде складатись з двох екранів.

Перший екран, який зображено на рисунку 3.2(а) – це головний екран застосунку, який буде використовуватись при відкритті тестового застосунку. На цьому екрані є дві кнопки, при натисканні на які, користувачеві буде запропоновано вибрати фотографію чеку з галереї, або ж зробити знімок за допомогою камери. Далі відповідно від вибраного процесу розпізнавання запускається відповідний потік (flow). На рисунку 3.2(б) зображено екран результату, на якому буде виведено розпізнаний текст, який буде включати в себе назву товару та ціну. Увесь програмний код наведено у додатку А



(а) (б)

Рисунок 3.2 – Вигляд UI тестового застосунка:
(а) – головний екран; (б) – екран результату розпізнавання

3.4.2 Реалізація алгоритму Tesseract на мові Swift

Після того як було створено мінімальний користувацький інтерфейс, можна перейти наступного етапу розробки, а саме, до імплементації OCR Tesseract. Для цього використаємо менеджер сторонніх бібліотек CocoaPods.

В папці проекту створюємо текстовий документ з назвою Podfile, через термінал, використовуючи, команду pod init, щоб згенерувати необхідні файли для проекту. Зміст відредагованого файлу наведено на рисунку 3.3. Далі знову відкриваємо термінал, переходимо в директорію проекту та викликаємо команду “pod install”. Після цього в наш проект буде додано три сторонні бібліотеки TesseractOCRiOS, яка використовує Tesseract версії 3.0.3, SwiftyTesseract, яка використовує версію 4.0.0, а також GPUImage, про яку вже було написано раніше.

Podfile

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '12.1'

target 'OCR' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for OCR
  pod 'SwiftyTesseract'
  pod 'TesseractOCRiOS'
  pod 'GPUImage'

end

#post_install do |installer|
#  installer.pods_project.build_configurations.each do |config|
#    config.build_settings["EXCLUDED_ARCHS[sdk=iphonesimulator*]"] = "arm64"
#  end
#end
```

Рисунок 3.3 – Зміст Podfile

3.4.3 Порівняння Tesseract версії 3.0.3 та Tesseract версії 4.0.0

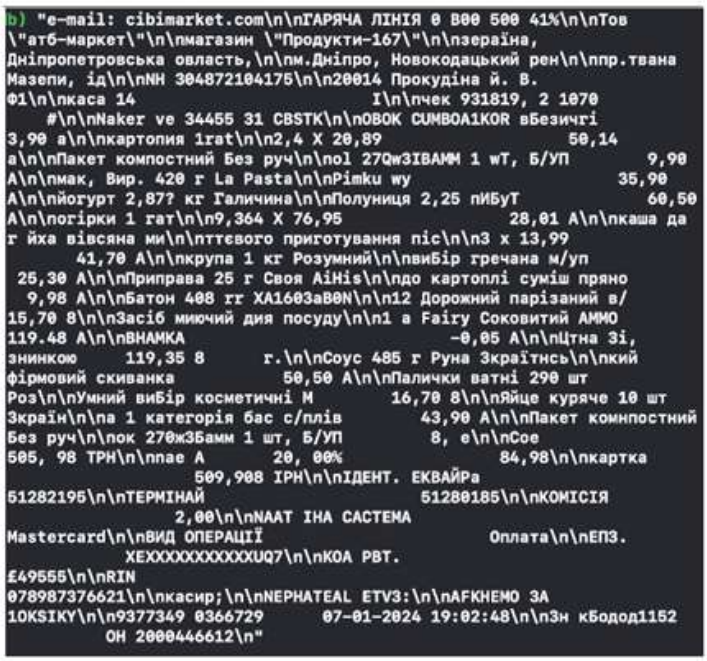
Для першого запуску візьмемо навчальні данні для української та англійської мов з репозиторіїв для кожної з версій фреймворку відповідно, та перевіримо робото спроможність двох бібліотек, використовуючи чек, зображеній на рисунку 3.4.

Рисунок 3.4 – Зображення чеку для тестового розпізнавання

Для порівняння різниці між версією 3.0.3 та 4.0.0 до тестового проекту було додано дві сторонні бібліотеки TesseractOCRiOS та SwiftyTesseract, кожна з цих бібліотек має в своїй основі відповідну версію Tesseract OCR.

Третя версія бібліотеки виконує розпізнавання тексту за допомогою шаблонів з навчальних даних, а четверта версія розпізнає текст використовуючи нейронну мережу, заснованої на LSTM. Результат першого запуску зображено на рисунку 3.5.

(а)



(б)

Рисунок 3.5 – Результати розпізнавання чеку версіями 4.0.0 (а)та 3.0.3 (зверху вниз)

Як видно з результатів, версія 3.0.3 майже не змогла розпізнати жодного слова, в той час як, версія 4.0.0 змогла розпізнати деякі слова, хоча й допустивши в них помилки.

Звісно треба брати до уваги, що навчальні данні були взяти стандартні, тобто без урахування шрифту, а також і те що перше розпізнавання було виконано без будь-якої попередньої підготовки зображення. Але все ж таки, цей тест наочно продемонстрував, перевагу OCR, в основі яких лежить нейронні мережі над тими системами, що засновані на шаблонному методі розпізнавання. Також це порівняння продемонструвало, що використання TesseractOCR версії 3.0.3 є недоцільним, тому в подальших тестах ця версія брати участь не буде.

3.4.4 Навчання Tesseract

Після того як було перевірено, що підключення Tesseract пройшло успішно і немає ніяких проблем з його роботою, можна спробувати навчити його необхідному шрифту, в моєму випадку це буде шрифт Courier new, який використовуються на касових чеках, а для навчання будемо використовуючи дані з реальних чеків.

Слід зауважити, що для того, щоб добре навчити нейронну мережу, необхідно, щоб кожен символ зустрічався мінімум п’ять разів, а краще в рази більше. Для навчання Tesseract буду використано jTessBoxEditor – сторонню утиліту-редактор, яка зображена на рисунку 3.6.

Тож перейдемо до процесу навчання. Для цього необхідно, або фото чеку, або текстовий документ з навчальними даними. Якщо використовувати фотографію з навчальними даними, то за допомогою jTessBoxEditor можна сформувати зображення у форматі tif.

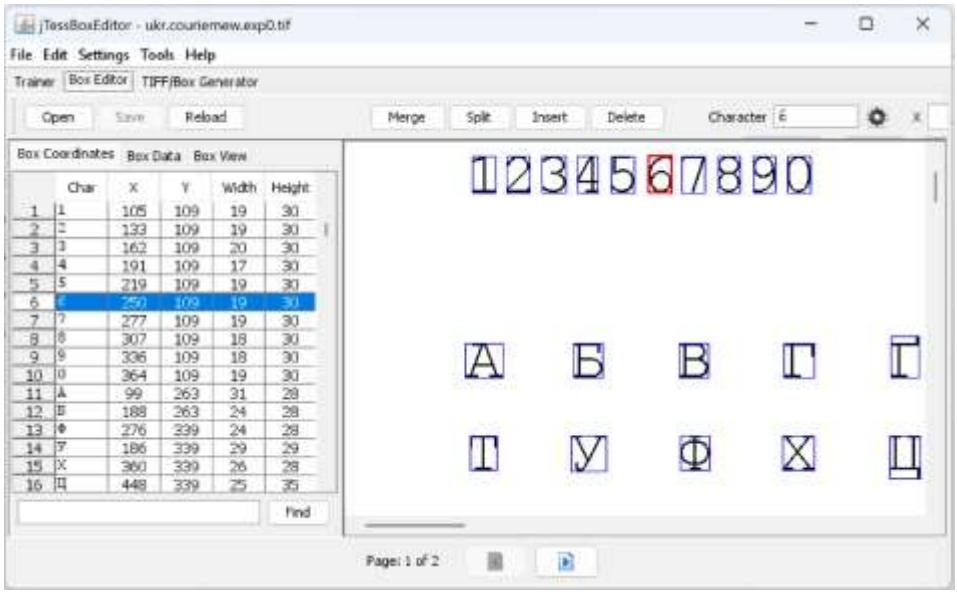


Рисунок 3.6 – Зовнішній вигляд редактору jTessBoxEditor

Якщо ж використовуються навчальні дані у вигляді текстових документів, то для цього також можна використати, раніше згаданий редактор та розділ TIFF/BOX Generator, що зображений на рисунку 3.7 Також, слід зауважити, що згенероване зображення повинно мати спеціальний формат назви у вигляді

{код мови}.{ Назва шрифту}.exp {номер}.tif.

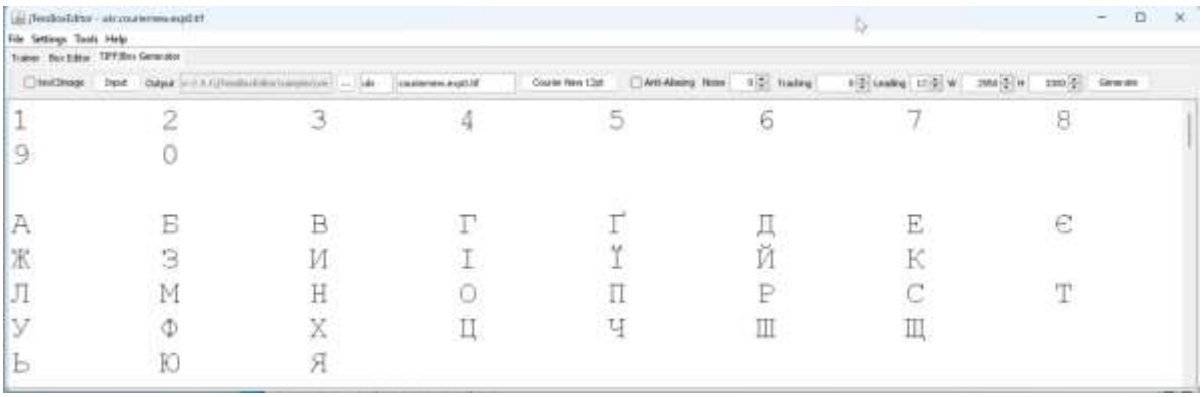


Рисунок 3.7 – Генерація tif зображення з текстового файлу за допомогою jTessBoxEditor

Після того, як було отримано зображення навчальних даних у tif форматі, потрібно створити файл “відповідність” box формату, в якому

будуть вказані символ, та координати його прямокутника на зображені, для цього необхідно в терміналі викликати команду:

tesseract ukr.couriernew.exp1.tif ukr.couriernew.exp1 batch.nochop makebox,

де ukr.couriernew.exp1.tif – це назва нашого зображення, а ukr.couriernew.exp1 – це назва файлу який було створено командою, що була написана вище.

Отримавши файл-відповідностей, відкриваємо наше зображення в редакторі та перевіряємо чи правильно вказані символи та їх координати, що зображено на рисунку 3.8.

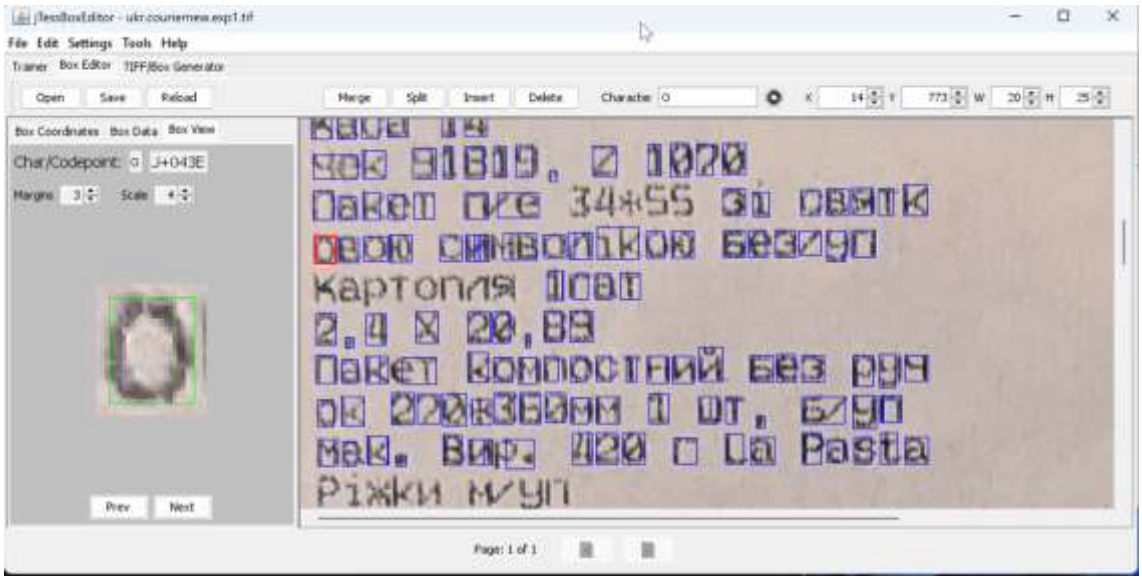


Рисунок 3.8 – Зображення чеку з координатами символів з box-файлу

На зображені також видно, що деякі символи не обведені своїм прямокутником, це означає, що tesseract не зміг розпізнати їх і відповідно згенерований файл не містить відповідних координат. Це легко виправити, якщо відкрити box-файл як текстовий документ, та додати нову строку у такому ж форматі як і інші строки. Після цього перевіряємо кожен символ за виправляємо помилки в розпізнаванні.

Далі виконуємо команду:

tesseract ukr.couriernew.exp1.tif ukr.couriernew.exp1 nobatch box.train

Ця команда створить файл ukr.couriernew.exp1 з tr розширенням.

Після цього потрібно створити файл, в якому буде набір символів та їх характеристики, для цього виконуємо команду:

unicharset_extractor ukr.couriernew.exp1.box,

Тепер необхідно створити текстовий документ, в якому буде міститися інформація про шрифт, а саме назва шрифту та параметри у вигляді 0 або 1, ці параметри вказують наявність у символів відповідного стилю (italic bold fixed serif fraktur), яким буде навчено Tesseract. В даному випадку це буде текст couriernew 0 0 1 0 0. Отриманий файл має назву ukr.font_properties.

Далі виконуємо наступні дві команди:

shapeclustering -F ukr.font_properties -U unicharset ukr.couriernew.exp1.tr

mftraining -F ukr.font_properties -U unicharset -O ukr.unicharset ukr.couriernew.exp1.tr

Перша команда створить файл shapetable, а друга – файли inttemp, pffmtable, normproto.

Після цього необхідно перейменувати файли, як зображено на рисунку 3.9.

```
ukr.couriernew.exp1.box -> ukr.box
inttemp ->                 ukr.inttemp
pffmtable ->               ukr.pffmtable
ukr.couriernew.exp1.tif -> ukr.tiff
normproto ->               ukr.normproto
shapetable ->              ukr.shapetable
ukr.couriernew.exp1.tr ->  ukr.tr
unicharset ->              ukr.unicharset
```

Рисунок 3.9 – Старі та нові назви файлів

Після виконання усіх підготовчих процесів, можна перейти до команди навчання Tesseract:

combine_tessdata ukr,

де ukr – це назва натренованої моделі. Після виконання цієї команди буде отримано файл ukr.traineddata, що повинен покращити результат розпізнавання тексту.

3.4.5 Реалізація алгоритму попередньої обробки зображення

Тепер випробуємо робото спроможність Tesseract з зображення, яке було попередньо оброблене за допомогою раніше описаного алгоритму.

Але перед цим необхідно виконати “нульовий” етап, а саме масштабувати зображення, щоб зображення правильно потрапило в “поле зору” Tesseract. Алгоритм масштабування зображено на лістингу 3.1.

Лістинг 3.1 – Алгоритм масштабування зображення

```
     func scaledImage(_ maxDimension: CGFloat) -> UIImage? {
         var scaledSize = CGSize(width: maxDimension, height:
maxDimension)
         if size.width > size.height {
             scaledSize.height = size.height / size.width *
scaledSize.width
         } else {
             scaledSize.width = size.width / size.height *
scaledSize.height
     }
          UIGraphicsBeginImageContext(scaledSize)
          draw(in: CGRect(origin: .zero, size: scaledSize))
          let scaledImage =
UIGraphicsGetImageFromCurrentImageContext()
       UIGraphicsEndImageContext()

       return scaledImage
       }
```

Метод масштабування зображення на вхід приймає максимальний розмір (висоту або ширину) зображення. Наступним кроком буде виконано розрахунок меного розміру зображення при якому зберігається відношення сторін початкового зображення. Далі строюється “контекст зображення” після чого виконується виклик метода draw, щоб намалювати початкове зображення з розрахованим раніше розміром. І останнім кроком метод повертає отримане зображення.

Тепер можна перейти до налаштування Tesseract. Для цього спершу створимо об’єкт типу Tesseract з вказанням, мови, яку буде розпізнавати система. Цей код наведено на лістингу 3.2.

Лістинг 3.2 – Створення об’єкту Tesseract та його налаштування

```
let tesseract = Tesseract(languages: [.ukrainian])
tesseract.configure {
     set(.preserveInterwordSpaces, value: "1")
}
```

Також, як видно на лістингу 3.2 виконується викликлик замикання, в якому встановлюється змінній preserveInterwordSpaces значення один, тобто true. Ця змінна, вказує чи буде Tesseract зберігати такі ж відступи між словами як на зображені, яке буде розпізнаватись.

Далі буде створено метод preprocessedImage, в якому виконується попередня обробка зображення.

Першим кроком, необхідно застосувати алгоритм Кенні, щоб визначити границі чеку та виконати вирівнювання зображення. Цей код наведено на лістингу 3.3.

Лістинг 3.3 – Виконання Алгоритму Кенні та методу Хафа

```
     let grayscaleImageFilter =
GPUImageAdaptiveThresholdFilter()
     rayscaleImageFilter.blurRadiusInPixels = 5
     let grayscaleImage =
grayscaleImageFilter.image(byFilteringImage: self)
     let cannyFilter = GPUImageCannyEdgeDetectionFilter()
     cannyFilter.texelHeight = 6
     cannyFilter.texelWidth = 4
     cannyFilter.blurRadiusInPixels = 1.0
     cannyFilter.blurTexelSpacingMultiplier = 0.35
     cannyFilter.upperThreshold = 0.2
     cannyFilter.lowerThreshold = 0.15
     let imageWithCannyWilter =
cannyFilter.image(byFilteringImage: grayscaleImage)
     let houghFilter = GPUImageHoughTransformLineDetector()
     houghFilter.edgeThreshold = 0.2
     houghFilter.lineDetectionThreshold = 0.26
```

В першу чергу, необхідно використати фільтр AdaptiveThresholdFilter, щоб отримати зображення у градації сірого. У AdaptiveThresholdFilter є один параметр blurRadiusInPixels – це множник, за допомогою якого встановлюється середнє значення розмиття фону в пікселях. Емпіричним методом було встановлено, що значення цього параметру необхідно встановити на рівень від 3.8 до 5, щоб отримати зображення у градації сірого в нормальній якості. Далі створюємо фільтр заснований на алгоритмі Кенні та налаштовуємо його параметри:

– параметри texelHeight та texelWidth впливають видимість виявлених країв;

– blurRadiusInPixels – це параметр базового радіусу розмиття Гауса;

– blurTexelSpacingMultiplier – цей параметр вказує множник для інтервалів між тексилями розмиття;

– параметри-пороги upperThreshold та lowerThreshold необхідні, щоб визначити краї об’єктів. Будь-який край, з величиною градієнту більше за поріг буде відображений в кінцевому результаті.

Після цього створюємо фільтр заснований на методі Хафа та застосовуємо його до отриманого на попередньому етапі зображення.

Далі виконуємо бінаризацію зображення за допомогою алгоритму Оцу за допомогою порогу. Виконання цього алгоритму наведено на лістингу 3.4.

Лістинг 3.4 – Виконання бінаризації алгоритмом Оцу за допомгою порогу

```
     let grayScaleImageFilterWithOtsu =
GPUImageLuminanceThresholdFilter()
     grayScaleImageFilterWithOtsu.threshold = 0.38
     let grayScaleImageWithOtsu =
     grayScaleImageFilterWithOtsu.image(byFilteringImage:
     imageWithCannyWilter)
```

На лістингу 3.4 зображено створення фільтру бінаризації та його налаштування, в даному випадку було встановлено, що найркащий результат отримується з порогом на рівні 0.38.

Наступним кроком необхідно виконати розширення для бінаризованого зображення. Цей пункт буде виконано за допомогою фільтру RGBDilationFilter, що зображено на лістингу 3.5.

Лістинг 3.5 – Розширення зображення з використанням матриці згортки

```
    let convolutionFilter = GPUImage3x3ConvolutionFilter()
    let convolutionImage =
convolutionFilter.image(byFilteringImage:
grayScaleImageWithOtsu)
    let dilationFilter = GPUImageRGBDilationFilter(radius: 1)
    dilationFilter?.currentlyReceivingMonochromeInput = true
    dilationFilter?.verticalTexelSpacing = 0.1
    dilationFilter?.horizontalTexelSpacing = 0.1
    let resultImage = dilationFilter?.image(byFilteringImage:
convolutionImage)
```

Для виконання розширення необхідно встановити радіус прямокутної області, щоб регулювати ступінь розширення. Також необхідно викликати параметр currentlyReceivingMonochromeInput, та встановити йому значення true, щоб фільтр розумів, що на вході у нього буде бінаризоване зображення. В додаток до цих налаштувань встановимо значення 0.1 для параметрів verticalTexelSpacing та horizontalTexelSpacing, корегують відстань між тексилями по вертикалі та горизонталі відповідно.

Після цього можна розпочинати розпізнавання. Модифіковане зображення конвертуємо у об’єкт Data та передаємо на вхід до Tesseract.

Після того як було отримано результат розпізнавання, необхідно перевірити його на наявність помилок, що й зображено на лістингу 3.6. Також на лістингу 3.6 зображено, що Tesseract буде розпізнавати текст рядками.

Лістинг 3.6 – Запуск розпізнавання тексту за допомогою Tesseract

```
    if let data = image.pngData() {
        let result = tesseract.recognizedBlocks(from: data,
for: [.textline])
            switch result {
            case .success(let result):
                    ...
            case .failure(let error):
                print("Fail: \(error.localizedDescription)")
                print(error.message)
            }
    }
```

Приклад розпізнаного тексту та його формату зображено на рисунку 3.10.

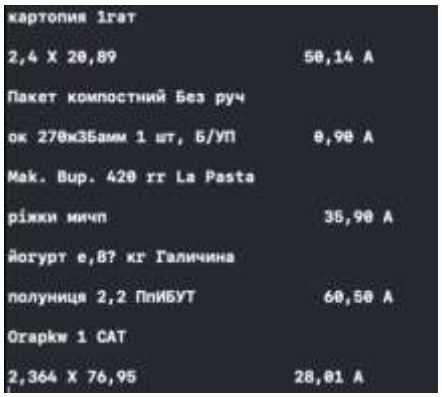


Рисунок 3.10 – Приклад розпізнаного тексту

Вже розпізнаний текст конвертується у об’єкт TesseractTextModel, що містить в собі змінні для тексту та координат цього тексту.

Враховуючи, що Tesseract розпізнавав тест по рядках, то відповідно дані були автоматично розбиті так як це зроблено в чеку, тому необхідно об’єднати інформацію про товар в один рядок, якщо вона була розділена на дві частини. Для того щоб проводити подальші модифікації отриманих даних, необхідно конвертувати їх у об’єкт TesseractTextModel, що містить в собі змінні для тексту та його координат, а також створимо їх дублікат в змінну tmpTesseractObjects.

Для того, щоб співставити дві частини назви товару ми, в першу чергу, в масиві об’єктів, знаходимо інформацію про номер чеку, яка має стандартизований формат – ЧЕК {число у 3 і більше символів}, Z {число довжиною у 2 і більше символів}. Увесь процес знаходження номеру чека зображено на лістингу 3.7.

Лістинг 3.7 – Знаходження номеру чеку

```
    var checkNumber: TesseractTextModel?
    for item in tmpTesseractObjects {
        if item.text.range(of: #"[Чек|чек|ЧЕК] \d+,|Z \d+$"#,
options: .regularExpression) != nil {
            checkNumber = TesseractTextModel(text: item.text,
boundingBox: item.boundingBox)
            tesseractObjects.removeAll(where: {
$0.boundingBox.midY == item. boundingBox.midY &&
$0.boundingBox.midX == item. boundingBox.midX})
            break
             } else { tesseractObjects.removeFirst() }
    }
    tmpTesseractObjects = tesseractObjects
```

Також під-час знаходження номеру, потрібно видалити увесь, непотрібний текст, що знаходяться вище номеру чека.

Наступним кроком, необхідно знайти інформацію про загальну суму чеку та видалити все що знаходиться нижче цього рядка. Цей процес зображено на лістингу 3.8.

Лістинг 3.8 – Знаходження інформації про суму чеку та видалення непотрібних даних

```
    var mainSum: TesseractTextModel?
    for item in tmpTesseractObjects.reversed() {
        if item.text.capitalize.findSubstring(with:
"(КАРТКА|\\d+[.,]\\d+ ГРН)", options: [.caseInsensitive]) {
            mainSum = TesseractTextModel(text: item.text,
boundingBox: item.boundingBox)
            tesseractObjects.removeLast()
            break
        } else { tesseractObjects.removeLast() }
    }
```

І останній етап, перед тим, як почати об’єднувати інформацію про товар – це потрібно знайти всі рядки в яких є ціна, перемістити їх в окремий масив, щоб уникнути дублювання даних, та видалити їх з основного масиву. Цей процес зображено на лістингу 3.9 Для цього етапу можна також використати регулярний вираз, так як усі ціни мають стандартизований формат у вигляді {один або більше числових символів}{крапка або кома}{два числа}{пробіл}{буква А}.

Лістинг 3.9 – Процес знаходження всіх рядків, де є ціна

```
    var priceObjects: [TesseractTextModel] = []
    for item in tmpTesseractObjects {
        if item.text.findSubstring(with: "\\d+[,.]\\d{2} [AA]",
options: []) {
            priceObjects.append(item)
            tesseractObjects.removeAll(where: {
$0.boundingBox.midY == item.boundingBox.midY &&
$0.boundingBox.midX == item.boundingBox.midX })
            }
        }
        tmpTesseractObjects = tesseractObjects
        priceObjects = priceObjects.sorted { $0.boundingBox.maxY
< $1.boundingBox.maxY }
```

Також в кінці потрібно виконати фільтрацію, щоб унеможливити ситуацію, коли ціни будуть переплутані місцями.

Тепер можна спробувати об’єднати дві половини інформації про товар. Алгоритм співставлення ціни та назви товару було зображено на рисунку 2.12. Реалізація цього алгоритму зображена на лістингу 3.10.

Лістинг 3.10 – Реалізація алгоритму співставлення ціни та назви товару

```
    var filteredProductItem: [[TesseractTextModel]] = []
    for priceItem in priceObjects.enumerated() {
        let topItemCoordinate = priceItem.offset == .zero
        ? (checkNumber?.boundingBox ?? .zero)
        : priceObjects[priceItem.offset - 1].boundingBox
        var tmpFilteredProductItem: [TesseractTextModel] = []
        for productItem in tmpTesseractObjects {
            if productItem.boundingBox.minY >
topItemCoordinate.midY && productItem.boundingBox.maxY <
priceItem.element.boundingBox.midY {
```

```
                    tmpFilteredProductItem.append(productItem)
                    tesseractObjects.removeAll(where: {
                        $0.boundingBox.midY ==
productItem.boundingBox.midY && $0.boundingBox.midX ==
productItem.boundingBox.midX
                    })
                }
            }
            if !tmpFilteredProductItem.isEmpty {
                filteredProductItem.append(tmpFilteredProductItem)
            }
        }
```

Зовнішній вигляд чеку, з доданими фільтрами, зображене на рисунку 3.11.

Рисунок 3.11 – Зображення чеку після застосування фільтрів

Застосувавши ці операції до початкового зображення, запускаємо розпізнавання, результат якого зображено на рисунку 3.12.

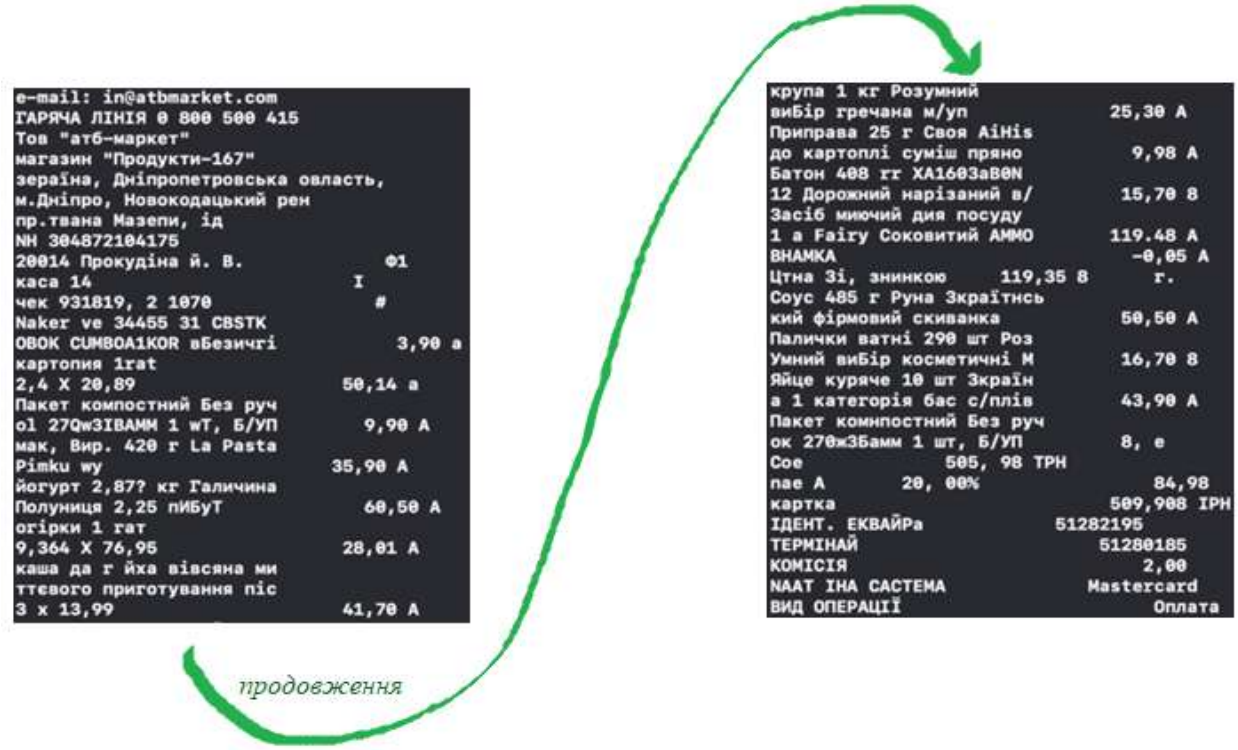


Рисунок 3.12 – Результат розпізнавання чеку з попередньою обробкою

Розроблена система коректно розпізнала більшу частину інформації, що була на чеку. Проте були допущені, як незначні помилки, наприклад в назві “Соус 458 г Руна Український …”, так і досить серйозні, наприклад ціна товару “яйця курячі…” була розпізнана з помилкою у шість гривень. Більшість помилок у розпізнаванні зумовлені тим, що OCR модуль неправильно розпізнав символи.

Для покращення роботи модулю можна виконати наступні дії:

– збільшити набір даних для навчання;

– додати інші шрифти з сімейства Courier new;

– покращити процес попередньої обробки зображення;

– залучення OCR систем, що надають більш точні рішення, на комерційній основі, для покращення точності розпізнавання ціни товару.

3.4.6 Інтеграція Apple Vision API

Тепер перейдемо до інтеграції нативного рішення від Apple – Apple Visual API, а саме VNRecognizeTextRequest. Для того, щоб додати цей

фреймворк в проект, не потрібно використовувати ніякі менеджери сторонніх бібліотек, як у випадку з TesseractOCR, достатньо лише, щоб проект підтримував мінімум тринадцяту версію операційної системи і все. Далі в будь якому місці проекту, де це доцільно додається код – “import Vision” і все можна писати логіку для розпізнавання тексту або будь-яких інших об’єктів, які можуть бути зображені на фотографії.

Також в порівняні з Tesseract OCR для налаштування Vision API не потрібно додавати набір навчальних даних та виконувати попередню обробку зображення перед запуском процесу розпізнавання, достатньо лише перевести зображення до типу cgImage (core graphics image) – це спеціальний тип зображення, який можна використовувати під час розробки під iOS.

3.4.7 Етапи реалізації функціоналу розпізнавання тексту за допомогою Vision API

Як вже зазначалось раніше, щоб отримати доступ до функціоналу Vision API необхідно зробити “import Vision” в клас/модуль, далі можна приступати до написання функціоналу розпізнавання тексту.

В першу чергу необхідно реалізувати запит, або інакше кажучи request, за допомогою якого запит буде відправлено на обробку до Vision. Під час реалізації запиту потрібно встановити деякі налаштування, щоб розпізнавання було більш точним та швидким. До таких налаштування відноситься встановлення масиву пріоритетних мов, для розпізнавання. Також слід визначити, чи буде запит віддавати пріоритет точності, що займає більше часу, чи навпаки буду виконувати більш швидке розпізнавання, але з меншою точністю. Код налаштування запиту наведено на лістинг 3.11.

Лістинг 3.11 – Налаштування запиту на розпізнавання тексту для Vision API

```
     let request = VNRecognizeTextRequest { [weak self] request,
error in
          //...
     }
          request.recognitionLanguages = ["ukr", "en"]
      request.recognitionLevel = .accurate
      request.usesLanguageCorrection = true
```

Після того як було створено та налаштовано запит, необхідно створити обробника, за допомогою якого буде виконано запит на розпізнавання. Для цього потрібно реалізувати об’єкт типу VNImageRequestHandler в який буде передано зображення, після чого можна запустити запит на розпізнавання тексту. Приклад реалізації цього функціоналу наведено на лістингу 3.12.

Лістинг 3.12 – Реалізація обробника для Vision API

```
guard let cgImage = image.cgImage else {
     return
}
let handler = VNImageRequestHandler(cgImage: cgImage)

do {
     try handler.perform([request])
} catch let error {
    print("Error: \(error.localizedDescription)")
}
```

І останній крок це – отримати текст та обробити його, для цього програма викликає метод perform і передає туди запит на розпізнавання, після чого результат буде отримано в completionHandler запиту. Там буде отримано масив результатів, який необхідно перевести до масиву об’єктів з типом VNRecognizedTextObservation та у кожного об’єкту масиву виконується виклик функції topCandidates, яка повертає найкращій варіант розпізнавання. Отримавши найкращі варіанти тексту система формує один єдиний текст. Код даного функціоналу наведено на лістингу 3.13

Лістинг 3.13 – Отримання та обробка тексту

```
     guard let observations = request.results as?
     [VNRecognizedTextObservation], error == nil else {
          return
          }
     let text = observations.compactMap {
          $0.topCandidates(1).first?.string
     }.joined(separator: "\n")
print(text)
```

3.4.8 Тестування Vision API

Після того як було описано функціонал та складові частини для реалізації Vision API слід провести тестування. В якості тестового зображення будемо використовувати чек, зображений на рисунку 3.4. Результат розпізнавання наведено на рисунку 3.13.

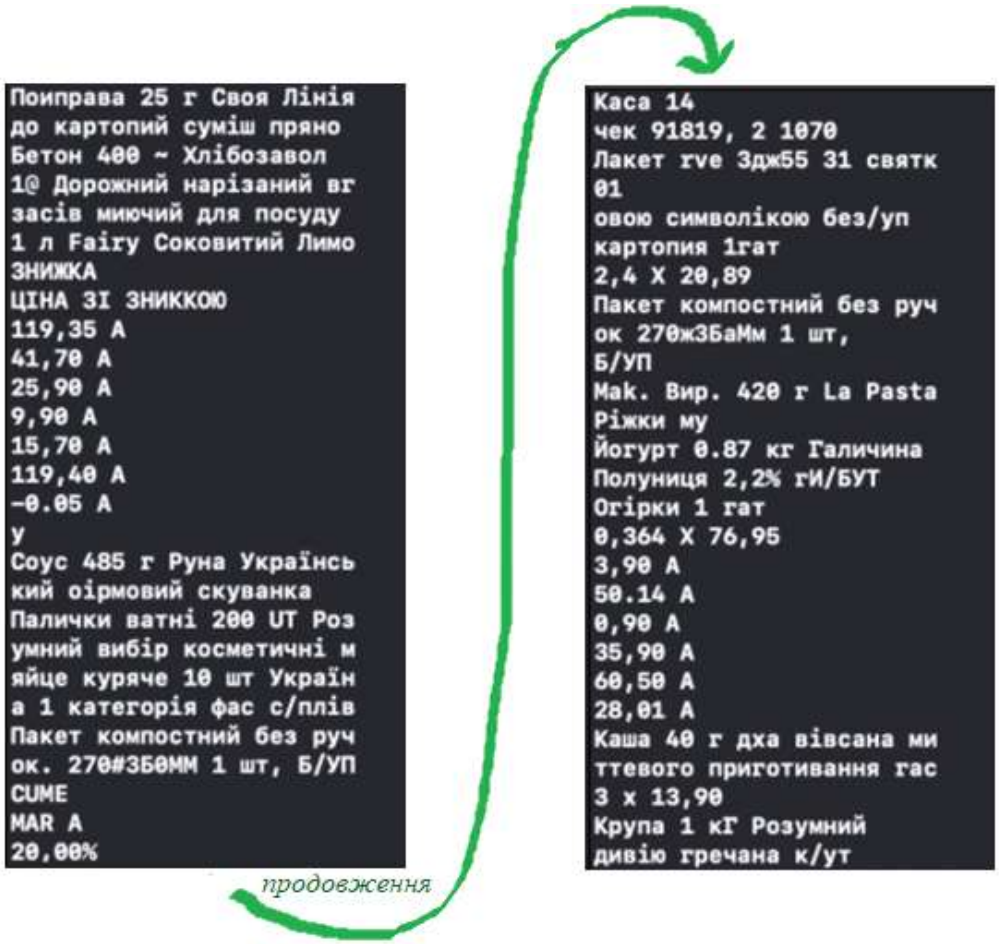


Рисунок 3.13 – Результат розпізнавання тексту за допомогою Vision API

Результат розпізнавання тексту, зображений на рисунку 3.13 є задовільним. Vision API правильно розпізнав усі блоки цін, що було проблемою для Tesseract, але також допустив помилки під-час розпізнавання назв товарів, але в його випадку покращити результат за допомогою до тренування моделі не є можливим, тому для отримання гарного результату потрібно використовувати фотографію максимально гарної якості. Однією з проблем при роботі з Vision API є те, що ця OCR система розпізнає текст як окремі блоки, якщо між ними є великий відступ, тому в більшості випадків ціна товару розпізнається к окремий запис і не співпадає з назвою товару.

3.4.9 Реалізація алгоритму співставлення товару та ціни

Для вирішення проблеми з розпізнаванням ціни як окремого блоку, потрібно реалізувати алгоритм співставлення ціни та товару, як для Tesseract OCR, але з деякими змінами, на останньому етапі – співставлення назви товару та ціни. Код алгоритму останнього етапу наведено на лістингу 3.14.

Лістинг 3.14 – Співставлення назви продукту та ціни

```
for priceItem in priceObjects.enumerated() {
    let topItemCoordinate = priceItem.offset == .zero
    ? (checkNumberObjects.first?.boundingBox ?? .zero)
    : priceObjects[priceItem.offset - 1].boundingBox
    var tmpFilteredProductItem: [TextModel] = []
    for productItem in observationsResult {
        if priceItem.element.boundingBox.minX < 0.25 {
            tmpFilteredProductItem.append(priceItem.element)
            break
        }
        if productItem.boundingBox.maxY <
topItemCoordinate.midY &&
Double(Int(productItem.boundingBox.minY * 1000)) / 1000 >=
Double(Int(priceItem.element.boundingBox.minY * 100)) / 100 {
            tmpFilteredProductItem.append(productItem)
        }
    }
    if !tmpFilteredProductItem.isEmpty {
        filteredProductItem.append(tmpFilteredProductItem)
    }
}
```

Головна відмінність реалізації цього алгоритму для Vision API та Tesseract OCR полягає в тому, що перший використовую Декартову систему координат в якій точка (0, 0) розміщена в лівому верхньому куті, в той час як в Vision API ця точка розміщена в лівому нижньому куті. Ще однією відмінність є те, що у випадках, коли назва товару досить велика та відстань між нею та ціною цього товару мала, то OCR від Apple розпізнає такий блок як одне ціле, для обробки такого результату було додано блок if в якому перевіряється, де розташовані координати ціни по осі *Ox*. Результат співставлення цін та назви товару зображено на рисунку 3.14.

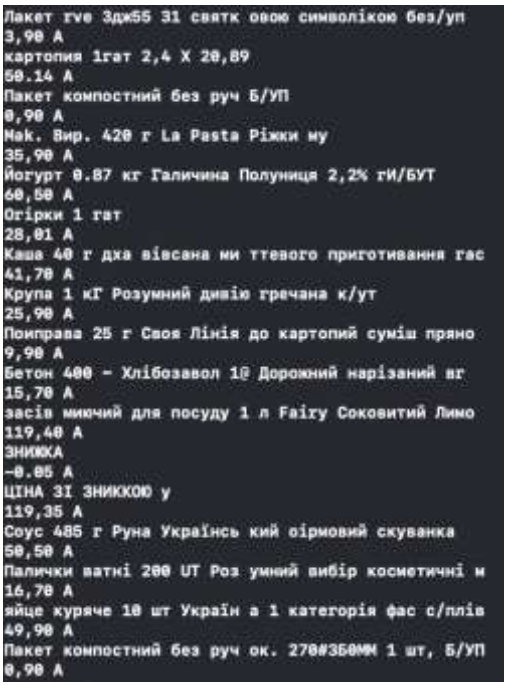


Рисунок 3.14 – Результат співставлення ціни та назви товару

3.5 Порівняння Tesseract OCR та Vision API

Отримавши результати розпізнавання тексту, з використанням Tesseract OCR та Apple Vision API, перейдемо до їх порівняння.

Порівнювати ці дві OCR системи будемо по двох категоріях:

– швидкість розпізнавання тексту;

– точність розпізнавання символів. Порівняння цю характеристику будемо використовуючи співвідношення кількість символів на зображені та кількості символів, які були не правильно розпізнані кожною з систем. Таким чином буде визначено відсоток помилок.

Ці характеристики є вкрай важливими, адже безпосередньо пливають на досвід користувача під час використання розробленого та йому подібних модулів.

Але спочатку порівняємо різні режими роботи Tesseract OCR, які наведені нижче:

– lstmOnly – це нова версія двигуна, що використовує нейронну мережу засновану на LSTM;

– tesseractLstmCombined – це режим роботи, що поєднує стару та нову версію двигуна Tesseract.

В якості тестового зображення буде використано чек, приклад, якого було зображений на рисунку 3.4. Цей чек містить 914 символів.

3.5.1 Порівняння точності розпізнавання різних двигунів Tesseract

В першу чергу, порівняємо точність розпізнавання у відсотках. Для цього виконаємо по десять ітерацій для кожного з варіантів двигуна.

Як вже зазначалось, порівнювати цей параметр будемо використовуючи відношення кількості символів, що зображені на рисунку та кількістю символів, які не вдалося розпізнати. Результат порівняння наведені на рисунку 3.15.

Проаналізувавши діаграму, чітко видно, що найкращій результат розпізнавання досягається при використанні двигуна lstmOnly. Тому в наступних тестах буде використано саме його.

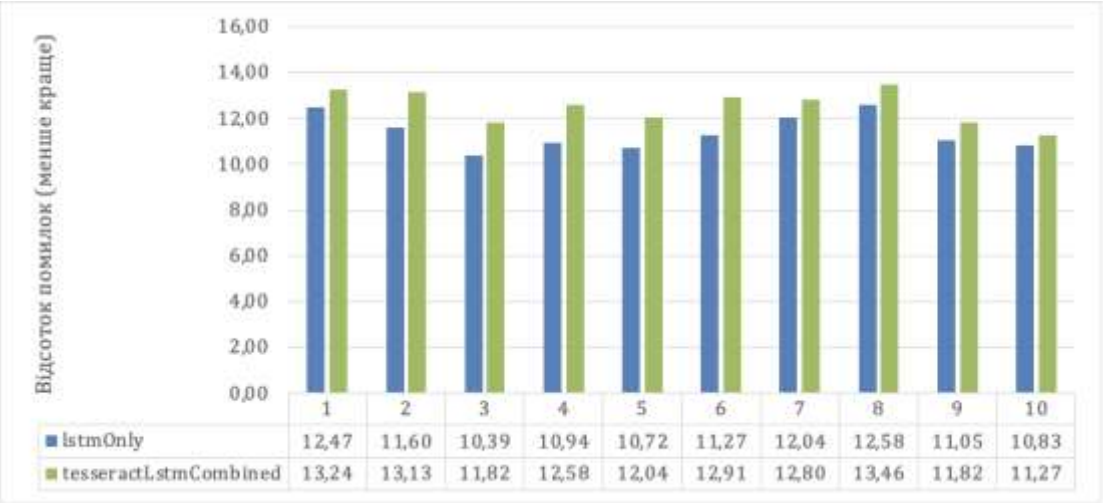


Рисунок 3.15 – Діаграма порівняння режимів роботи

3.5.2 Порівняння швидкості роботи двигунів Tesseract

Тепер порівняємо швидкість виконання операції розпізнавання. Для проведення цього тесту, буде використано стандартний об’єкт часу, що застосовуються під час розробки iOS застосунків.

Додаймо змінну типу Date перед початком попередньої обробки зображення та в кінці, коли результат вже розпізнавання відомий, щоб отримати різницю часу між цими двома об’єктами. Результат порівняння буде зображено на рисунку 3.16.

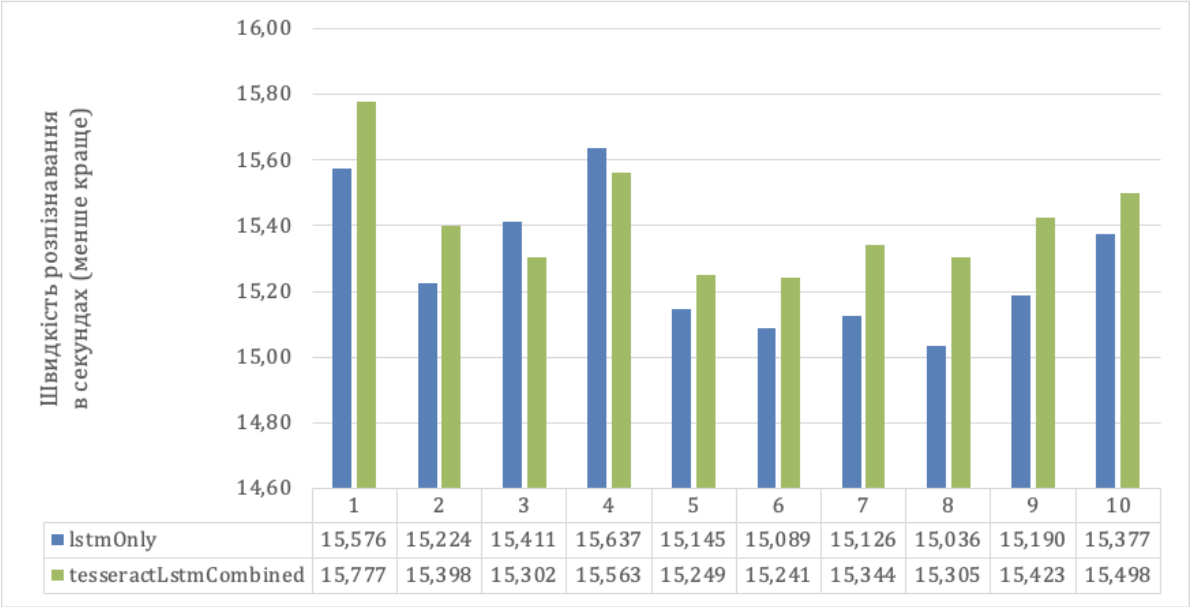
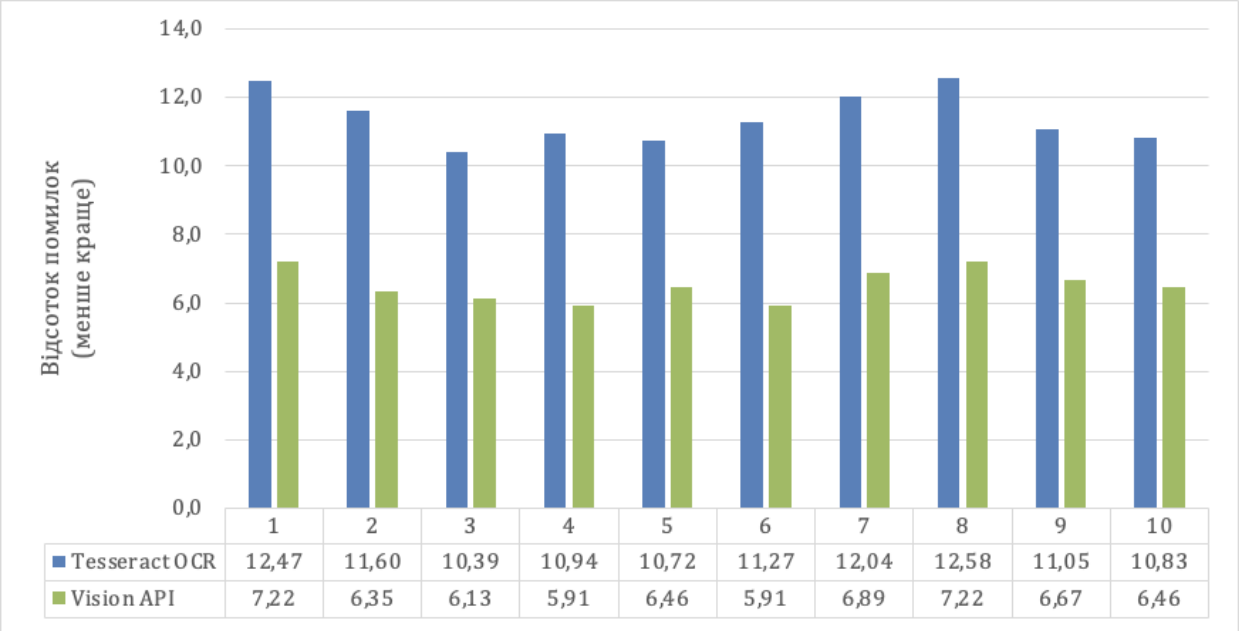


Рисунок 3.16 – Діаграма швидкості розпізнавання тексту

Проаналізувавши дані, що зображені на рисунку вище, можна прийти до висновку, що двигун заснований на нейронній мережі LSTM виконую операцію розпізнавання швидке ніж комбінація нового та старого двигунів.

3.5.3 Порівняння точності розпізнавання Tesseract та Vision API

Враховуючи, що в попередніх тестах було з’ясовано, що використання двигуна заснованого на комбінації нових та старих підходів для Tesseract є недоцільним, тому для цього та наступних тестів буде взято виміри саме для двигуна lstmOnly. Тепер виконаємо заміри для Vision API. Результати тестування наведено на рисунку 3.17.



	1	2	3	4	5	6	7	8	9	10
Tesseract OCR	12,47	11,60	10,39	10,94	10,72	11,27	12,04	12,58	11,05	10,83
Vision API	7,22	6,35	6,13	5,91	6,46	5,91	6,89	7,22	6,67	6,46

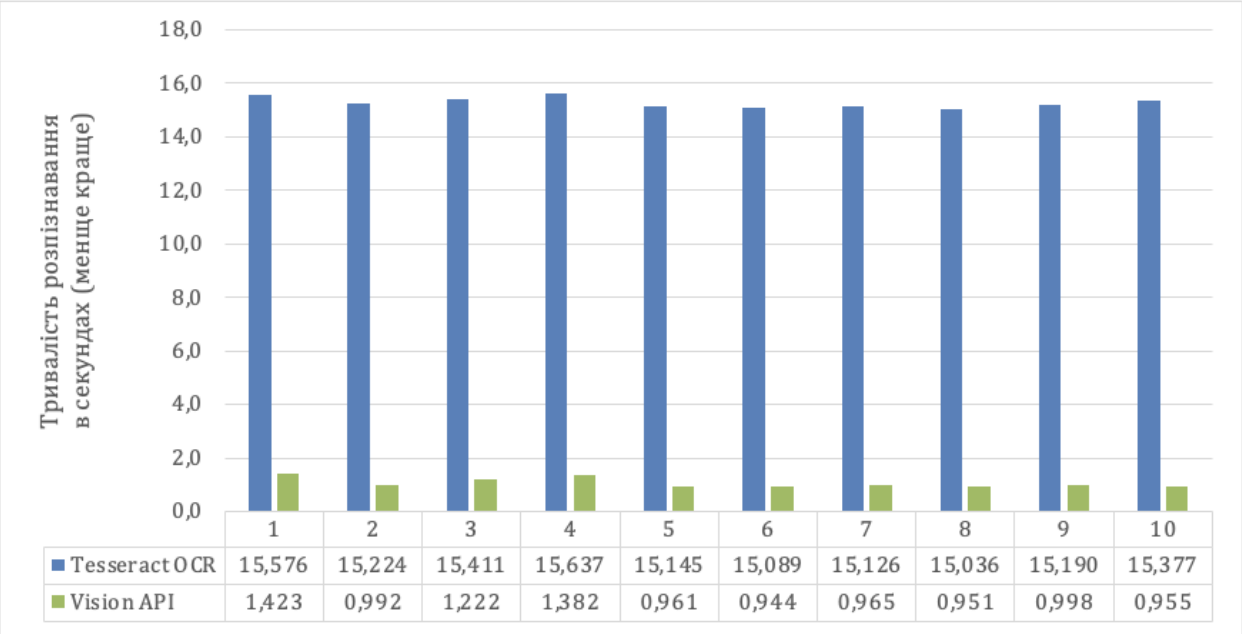
Рисунок 3.17 – Порівняння точності розпізнавання Tesseract OCR та Vision API

Аналізуючи діаграму, наведену вище, можна зробити висновок що Vision API робить помилки майже в два рази менше ніж Tesseract OCR. Цей результат був досить очевидним, тому що. Розробка Apple для розпізнавання використовую не тільки машинний зір, а й фізичний нейронний процесор, що й дозволяє отримувати такі результати. Хоча й результат Tesseract досить

непоганий, в додаток до цього його ще можна спробувати покращити за допомогою доповнення датасету.

3.5.4 Порівняння Tesseract OCR та Vision API по швидкості розпізнавання

Враховуючи, що тестові дані для Tesseract вже були отримані на минулих тестах, то відповідно залишається тільки отримати результати тестування для Vision API та порівняти їх. Результати тестування наведено на рисунку 3.18.



	1	2	3	4	5	6	7	8	9	10
Tesseract OCR	15,576	15,224	15,411	15,637	15,145	15,089	15,126	15,036	15,190	15,377
Vision API	1,423	0,992	1,222	1,382	0,961	0,944	0,965	0,951	0,998	0,955

Рисунок 3.18 – Порівняння швидкості розпізнавання тексту Tesseract OCR та Vision API (чек №1)

Результати, наведені на рисунку 3.18 показують, що реалізація OCR технології від Apple, в середньому, в 15 разів швидше розпізнає текст, ніж Tesseract. Як вже зазначалось, це зумовлене тим, що Vision API може використовувати окремий нейронний процесор під час своєї роботи.

Тепер спробуємо узяти чек, з меншою кількістю інформації, та провести тестування швидкості. Чек №2 зображено на рисунку 3.19.

Результати вимірювання швидкості з використанням чеку №2 зображено на рисунку 3.20.

Рисунок 3.19 – Приклад чеку під номером два

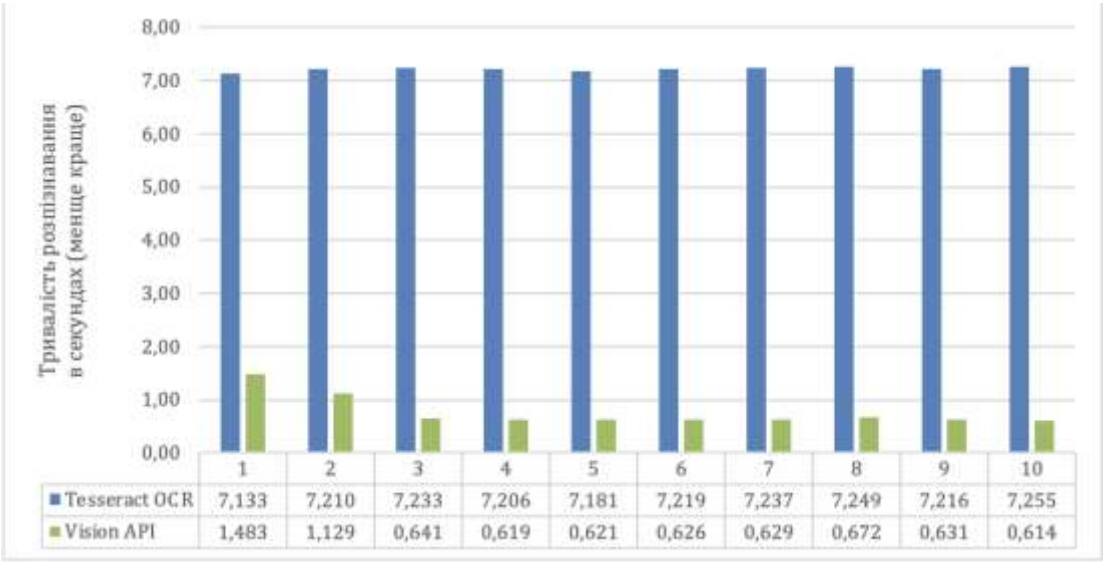


Рисунок 3.20 – Порівняння швидкості розпізнавання тексту Tesseract OCR та Vision API (чек №2)

Результати наведені на рисунку 3.20 показують, що об’єм тексту на зображенні істотно впливає на швидкість розпізнавання Tesseract OCR. Розпізнавання тексту на другому чеку, приблизно, в два рази швидше ніж для першого. В той час як швидкості розпізнавання для Apple Vision API збільшилася приблизно 30 % між першим та другим чеком. Цей тест доводить, те, що нейронні мережі засновані на LSTM погано справляються з розпізнаванням тексту з великою кількістю класів.

3.6 Висновки до розділу

У даному розділі було розглянуто платформу Apple, мови програмування, які використовуються для розробки під неї та аргументовано використання мови Swift для розробки застосунку представленого в цій роботі.

Також було розглянуто декілька бібліотек, що використовувались під час розробки, а саме GPUImage, для виконання попередньої обробки зображення, SwiftyTesseract та Vision API для розпізнавання тексту.

Також було описано процес навчання Tesseract та реалізація алгоритму, який було представлено в попередньому розділі та наведено приклади розпізнавання тексту та зображення чеку після попередньої обробки.

В додаток до цього було реалізовано розпізнавання та співставлення товару та ціни за допомогою Apple Vision API. Та проведено порівняння цих двох технологій за швидкістю та точністю розпізнавання.

ВИСНОВОК

Контроль та відстеження витрат є вирішальним для досягнення фінансової стабільності, що є надзвичайно важливим для кожної людини. У цьому процесі програмні рішення можуть виступати в якості надійного помічника, надаючи широкий функціонал для обліку фінансових витрат і доходів.

Аналіз існуючих систем, показав, що застосування OCR технологій для ведення персонального обліку витрат є не досить розповсюдженим, а ті застосунки, що мають такий функціонал відкривають до нього доступ тільки після купівлі дороговартісної підписки, саме тому було вирішено розробити модуль розпізнавання тексту на касових чеках, який в подальшому можна буде інтегрувати в повноцінний застосунок персонального обліку.

Метою кваліфікаційної роботи було дослідження методів розпізнавання зображень для системи персонального обліку витрат. Для досягнення мети кваліфікаційної роботи було розглянути технологію OCR, її особливості, сфери використання та перспективи у майбутньому. Також було досліджено популярні системи оптичного розпізнавання тексту, визначено їх особливості та виконано порівняння цих систем. Далі було розібрано процес навчання Tesseract та реалізовано алгоритм попередньої обробки на мові Swift. Розроблений модуль було протестовано в складі тестового застосунку на реальних даних. Ефективність розробленого алгоритму була порівняна з іншими OCR системами, на прикладі Apple Vision API. Подальшим розвитком даної роботи може бути доведення поточної реалізації розробленого модуля до рівня комерційних застосунків, для цього потрібно оптимізувати алгоритм, провести додаткове навчання нейронної мережі та її налагодження, що дозволить використовувати розроблену систему не тільки для сканування касових чеків продуктових супермаркетів, а й будь-яких інших магазинів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Жихаревич В. Аналіз методів розпізнавання символів тексту / В. Жихаревич, С. Остапов, І. Миронів // Радіоелектронні і комп'ютерні системи. – 2016. – № 5. – С. 137–140.

2. Тхорик В. Модуль розпізнавання чеків для програми персонального обліку витрат/ В. Тхорик // Репозитарій Національного Авіаційного Університету: Home. – Режим доступу: https://er.nau.edu.ua/bitstream/NAU/50800/1/ФККПІ_2020_123_ТхорикВБ.pdf (дата звернення: 25.10.2023). – Назва з екрана.

3. Головна: Державний університет інформаційно-комунікаційних технологій. – Режим доступу: https://duikt.edu.ua /repozitorii/ipz/2020/Гаращук%20Вадим%20Чергійович.pdf (дата звернення: 25.10.2023). – Назва з екрана.

4. Що таке оптичне розпізнавання символів (OCR)? [Електронний ресурс] // Go travels. – Режим доступу: https://uk.go-travels.com/74733-optical-character-recognition-4158322-8335115 (дата звернення: 04.06.2023). – Назва з екрану.

5. Survey on Image Preprocessing Techniques to Improve OCR Accuracy [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/ technovators/survey-on-image-preprocessing-techniques-to-improve-ocr-accuracy-616ddb931b76 (дата звернення: 04.06.2023). – Назва з екрану.

6. Методи розпізнавання тексту [Електронний ресурс] // wik.uk-ua.nina.az. – Режим доступу: http://surl.li/hqdek (дата звернення: 04.06.2023). – Назва з екрану.

7. CORE – Aggregating the world's open access research papers [Електронний ресурс]. – Режим доступу: https://core.ac.uk/download/pdf/53065993.pdf (дата звернення: 04.06.2023). – Назва з екрану.

8. Paramud Y. Method of image symbol recognition on the basic of convolution neural network [Electronic resource] / Yaroclav Paramud, Volodymyr Yarkun // Computer systems and network. – 2018. – Vol. 1, no. 905. – P. 96-106. Mode of aaccess https://doi.org/10.23939/csn2018.905.096 (date of access: 04.06.2023). – Title from screen.

9. Холявкіна Т. В. Deep learning based image recognition system with neural network architecture. [Electronic resource] / Тетяна Володимирівна Холявкіна, Ярослав Олегович Резаєв, Олексій Олексійович Харченко // Science-based technologies. – 2020. – Vol. 45, no. 1. – Mode of access: https://doi.org/10.18372/2310-5461.45.14572 (date of access: 25.10.2023). – Title from screen.

10. Антон М. Google cloud Vision API. Будущее Computer Vision as a service настало? [Електронний ресурс] / Мальцев Антон // Habr. – Режим доступу: https://habr.com/ru/articles/312714/ (дата звернення: 04.06.2023). – Назва з екрану.

11. Recognizing Text in Image | Apple Developer Documentation [Електронний ресурс] // Apple Developer Documentation. – Режим доступу: https://developer.apple.com/documentation/vision/recognizing_text_in_images (дата звернення: 04.06.2023). – Назва з екрану.

12. What is Xamarin? - Xamarin. *Microsoft Learn: Build skills that open doors in your career*. [Електронний ресурс] // Режим доступу: https://learn.microsoft.com/uk-ua/xamarin/get-started/what-is-xamarin (дата звернення: 13.01.2024). – Назва з екрану.

13. Михальчук М. О. Система обліку персональних витрат / М. О. Михальчук // Радіоелектроніка та молодь у XXI столітті : матеріали 27-го Міжнар. молодіж. форуму, 10–12 травня 2023 р. – Харків : ХНУРЕ, 2023. – Т. 5. – С. 5–6. (дата звернення 13.01.2024). – Назва з екрану.