МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О. Є.

«____»_____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: Модуль розпізнавання чеків для програми персонального фінансового обліку

Виконавець: Тхорик В.Б.

Керівник: Глазок О.М.

Нормоконтролер: Тупота Є.В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність (спеціалізація) 123 «Комп'ютерна інженерія»

(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О.Є.

«_____» _____2020 р.

ЗАВДАННЯ

на виконання дипломної роботи

Тхорика Володимира Богдановича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи: Модуль розпізнавання чеків для програми персонального фінансового обліку

затверджена наказом ректора від « 27 » серпня 2020 р. № 1203

2. Термін виконання роботи: з 5 жовтня 2020 р. по 13 грудня 2020 р.

3. Вихідні дані до роботи: Інформація про мову програмування написання програмного продукту; Інструкція по доступних операційним системам, на яких можна використати програмний продукт

4. Зміст пояснювальної записки:

1) OCR-технології та їх реалізації;

2) Проектування модуля розпізнавання чеків;

3) Програмна реалізація модулю розпізнавання

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Порівняння точності методів класифікації

2) Результати прогнозування категорії товару

3) Порівняння точності методів класифікації за категоріями

4) Алгоритм усунення перекосу зображення

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомлення з постановкою завдання дипломної роботи	05.10.2020 – 10.10.2020	
2	Аналіз літератури	10.10.2020 – 14.10.2020	
3	Обґрунтування вибору рішення	14.10.2020 – 28.10.2020	
4	Аналіз існуючих методів вирішення завдання	28.10.2020 – 14.11.2020	
5	Побудова моделей основних процесів	15.11.2020 – 20.11.2020	
6	Розробка програмного забезпечення	21.11.2020 – 02.12.2020	
7	Тестування програмного забезпечення	02.12.2020 – 03.12.2020	
8	Оформлення і друк пояснювальної записки	04.12.2020 – 12.12.2020	
9	Оформлення презентації	13.12.2020	

7. Дата видачі завдання: «05» жовтня 2020 р.

Керівник дипломної роботи (проекту) _____ Глазок О.М.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Тхорик В.Б.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний модуль формування графіку консультацій викладачів університету за допомогою генетичного алгоритму»: 81 сторінка, 23 рисунки, 1 таблиця, 16 літературних джерел, 2 додатки.

АЛГОРИТМ, КЛАСИФІКАЦІЯ, OCR, ПРОДУКТ, МАШИННЕ НАВЧАННЯ, *PYTHON*, ДОСЛІДЖЕННЯ.

Об’єкт дослідження – технології зчитування тексту з фотографії та його подальшої класифікації.

Предмет дослідження – програмний модуль розпізнавання касового чеку та класифікації товарів.

Мета дипломного проекту – проаналізувати сучасний рівень розвитку *OCR*-додатків, розглянути існуючі методи вирішення задач класифікації, обрати та обґрунтувати вибір методу, розробити алгоритм та програмний модуль для класифікації придбаних товарів, що дозволить скоротити час, який витрачається на ведення персонального фінансового обліку.

Методи дослідження – вивчення принципів роботи оптичного розпізнавання символів та методів класифікації, використання об’єктно-орієнтованого програмування при розробці додатку для системи в середовищі програмування *JetBrains PyCharm* за допомогою мови програмування *Python*.

Прогнозні припущення щодо розвитку об’єкта дослідження – розширення можливостей системи шляхом розробки додаткових модулів для масштабування додатку та забезпечення можливості враховувати специфічні нарахування, а також розгортання додатку в якості *Web API*.

У результаті виконання дипломної роботи був проведений аналіз сучасного рівня розвитку додатків в галузі *OCR* та машинного навчання, підтверджено актуальність даної теми та прийнято рішення про створення автоматизованої системи. Був складений список вимог до розроблюваного програмного модуля, вивчена предметна область, обрано і проаналізовано алгоритм, реалізований в системі. На основі цієї предметної області було реалізовано програмний продукт

мовою програмування *Python*. Результатом роботи є створена автоматизована система, що застосовується для автоматичного визначення категорій покупок, використання якого скорочує час, який витрачається на ведення обліку витрат.

Матеріали дипломної роботи рекомендується використовувати для визначення категорій покупок та ведення фінансового обліку.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – *Application Programming Interface*

ML – *Machine Learning*

OCR – *Optical character recognition*

SDK – *Software Development Kit*

ВСТУП

В умовах сьогодення кожній людині для успішного досягнення своїх цілей необхідно передбачати, яким чином вони можуть бути реалізовані, які ресурси для цього потрібні. Фінансова складова часто є першочерговим пунктом, без якого здійснення тих чи інших планів не є можливим. Як наслідок, зважене і раціональне використання коштів сприяє реалізації планів і, відповідно, самоутвердження людини, а безрозсудне відношення до власних грошових ресурсів зводить нанівець усі старання і надії. Першим же кроком до освоєння фінансової грамотності, а отже і стабільності, є вибудова власної політики щодо своїх фінансів. Таким чином, постає проблема обліку наявних у людини ресурсів з ціллю контролю за рухом коштів, характером витрат та подальшим їх аналізом.

Наразі на ринку програмних продуктів є безліч додатків, які дозволяють вирішувати вищеописану проблему. Деякі з них є кросплатформними, що веде до охоплення більшої аудиторії, або мають широкий функціонал, що допомагає користувачу максимально використовувати закладені в програму можливості. Однак використання таких додатків та отримання від цього очікуваної користі вимагає постійного та безперервного ведення своїх витрат – будь-який програмний засіб, навіть із вражаючим функціоналом, чудовою та детальною візуалізацією та простим інтерфейсом, не зможе в повній мірі відобразити характер витрат користувача та звернути увагу на потенційно небажані чи надлишкові витрати, якщо користувач не буде вести облік регулярно та детально, розділяючи свої витрати за категоріями. Тобто фактичне отримання користувачем того, чого він очікує від подібних додатків (ведення контролю над своїми фінансами та відображення супутньої інформації) прямо залежить від того, чи буде він ставитись до цього відповідально. На жаль, практика свідчить, що лише невелика частина користувачів регулярно веде облік у подібних засобах, а не забуває про необхідні дії із своєї сторони через тиждень чи місяць.

Поширеною функцією додатків ведення фінансового обліку, яка частково виправляє такий нюанс, є інтеграція інтернет-банкінгу та, власне, самого додатку –

при проведенні витрати за вказаною користувачем банківською карткою її сума автоматично списується з балансу користувача в програмі, а також додається відповідна транзакція, якій присвоюється певна категорія в залежності від того, який рід діяльності указаний для продавця. Однак такий підхід має все ж ряд недоліків:

– Потрібно надавати третій особі (розробнику програмного забезпечення) свою платіжну інформацію, що потенційно може загрожувати конфіденційності та фінансовій безпеці користувача, через що не кожен буде використовувати такий функціонал;

– Відстежується лише баланс указаної карти, а отже готівкові розрахунки не можуть бути відстежені таким чином;

– Купівля, наприклад, в супермаркеті буде відзначена категорією «Продукти», що не відображає реальний характер витрати – у подібних магазинах може бути придбано що-завгодно, і подібне узагальнення губить фактичну інформацію про те, на що і скільки було витрачено грошей.

Альтернативним рішенням є розміщення в деяких країнах на касовому чеку спеціального *QR*-коду, сканування якого відобразить сторінку на сайті продавця, на якій зберігається даний чек. Розробники знають про такий підхід, тому додатки володіють функціями додавання витрати через *QR*-код, однак в Україні подібних нововведень не відбувається. Окрім того, далеко не кожен продавець чи мережа побачить необхідність у введенні у себе такої послуги. Таким чином, проблема швидкого та якнайбільшого повного автоматичного ведення витрат залишається невирішеною.

Поширена проблема надмірних витрат і необхідність вести облік своїх коштів призвела до розробки великої кількості додатків для відслідковування, однак на даний момент вони все ще не надають для користувача достатніх інструментів для того, щоб з мінімальними зусиллями вести такий облік. Відповідно, було поставлено завдання в процесі дипломного проектування розробити модуль для програми ведення фінансового обліку, який дозволить на основі фотографії касового чеку надавати користувачу інформацію про сумарну ціну кожної із категорій, до яких належать придбані товари.

Метою дипломної роботи є розробка модуля розпізнавання касових чеків для програми персонального фінансового обліку, що дозволить скоротити час та зусилля, що витрачаються на відстеження витрат, та автоматизувати цей процес.

Для успішного досягнення цілі необхідно вирішити такі завдання:

– Проаналізувати наявні технології та програмні засоби, що застосовуються для розпізнавання тексту на зображеннях;

– Дослідити методи та алгоритми машинного навчання при вирішенні задач багатокласової класифікації;

– Розробити модель, що проводитиме розпізнавання тексту із фотографії чеку та здійснюватиме класифікацію даних, вилучених із зображення;

– Підготувати навчальний набір даних для моделей класифікації;

– Розробити програмний модуль на основі оптичного розпізнавання символів та обраних методів класифікації;

– Проаналізувати показники точності категоризації обраних методів та обрати найбільш точний.

При вирішенні поставлених завдань використовувалися методи та алгоритми оптичного розпізнавання символів, машинного навчання, методи попередньої обробки зображень для подальшого розпізнавання і методи об'єктно-орієнтованого програмування.

Об’єкт дослідження – технології зчитування тексту з фотографії та його подальшої класифікації.

Предмет дослідження – програмний модуль розпізнавання касового чеку та класифікації товарів.

Методи дослідження – ознайомлення з наявними засобами вилучення тексту із зображень; вивчення методів та алгоритмів багатокласової класифікації тексту; використання набутих знань для розробки програмного модуля, а також *IDE PyCharm*, бібліотек *Tesseract* та *Sci-Kit Learn* мови програмування *Python* для проведення розпізнавання тексту і його класифікації відповідно.

Практичне значення роботи полягає у використанні розробленого програмного забезпечення у додатках ведення персонального фінансового обліку для полегшення

та пришвидшення обліку витрат. Побудована модель може бути використана не тільки для чеків продуктових магазинів, але і в інших областях, де потрібне проведення категоризації із касового чеку (аптеки, магазини одягу, господарських товарів, заклади громадського харчування тощо), шляхом навчання моделі на наборі даних, що відповідають обраній сфері.

РОЗДІЛ 1
OCR-ТЕХНОЛОГІЇ ТА ЇХ РЕАЛІЗАЦІЇ

1.1 Оптичне розпізнавання символів

Оптичне розпізнавання символів (*OCR*) – це електронне перетворення набраних, рукописних або надрукованих текстових зображень у машинно-закодований текст.

Завдяки *OCR* величезна кількість паперових документів на різних мовах та форматах може бути оцифрована в машино-обролюваний текст, що не тільки полегшує зберігання, але й надає доступ до раніше недоступних даних користувачу за декілька натискань клавіші миші.

Такі зображення та документи можна сканувати як документ, фотографію документа чи фотографії оточення (наприклад, текст на вивісках та рекламних щитах).

Головним викликом роботи *OCR* є вирішення проблеми, яка полягає у тому, що є безліч різних шрифтів та способів написання одного і того ж символу. Окрім того, перш ніж вибирати алгоритм *OCR*, зображення має бути попередньо оброблене, щоб воно було готовим до «читання». Програмне забезпечення *OCR* часто виконує самостійно цей етап, щоб збільшити шанси на розпізнавання. Методи попередньої обробки включають в себе наступні кроки:

1. Усунення перекосу (вирівнювання). Якщо документ не був правильно вирівняний під час сканування або створення фотографії, то для підвищення точності зчитування його потрібно буде нахилити на кілька градусів за або проти годинникової стрілки таким чином, щоб текстові рядки були максимально горизонтально або вертикально розташованими.

2. Усунення шумів. Видаляються позитивні та негативні відносно загальної палітри кольорів плями, згладжуються краї символів. Додатково можуть проводитись такі операції, як відновлення пошкоджених фотографій, наприклад, за допомогою гіпоеліптичної дифузії, а також накладання фільтрів, що усувають розмиті ділянки

фотографії. Загальним приймом на даному етапі є застосування медіанного фільтру, який реалізує мінімізації шуму та згладження гострих країв букв.

3. Бінаризація. Вхідне перетворюється у чорно-біле (називається двійковим, або бінарним зображення, оскільки на ньому залишаються всього два кольори). Завдання бінаризації виконується як простий і точний спосіб відрізнити текст (або будь-який інший необхідний елемент зображення) від фону, що здійснюється за рахунок чіткого фіксування меж букв за рахунок накладання монохромного фільтру. Границю переходу в фільтрі рекомендується встановлювати

4. Видалення лінії. Очищаються несимвольні поля та рядки.

5. Аналіз макету, або зонування. Даний етап визначає стовпці, абзаци, підписи тощо як блоки. Особливо корисний у макетах та таблицях із декількома стовпцями.

6. Розпізнавання рядків і слів. Встановіть форму слова та символу базовою лінією, розділяє слова, коли це потрібно.

7. Розпізнавання почерку. У документах, що містять фрагменти різними мовами, почерк може змінюватись на рівні слова, тому його визначення є вкрай важливим етапом, перш ніж відповідна модель *OCR* може бути використана для роботи із наявним почерком.

8. Розділення символів, або сегментація. Завданням даного кроку є виявлення структурних одиниць тексту – рядків, слів і символів. Спершу визначається середнє значення відстані між двома символами в слові. Далі відбувається процес поділу на зображення на рядки шляхом знаходження відповідних смуг визначеної ширини. Після цього всі смуги діляться на слова, і заключним кроком у послідовності сегментації є розбиття слів на символи. Виділення фрагментів високих рівнів, таких як рядки і слова, може бути виконано на основі аналізу проміжків між темними областями зображення.

9. Нормалізація. Зображення потрібно відредагувати таким чином, щоб отримати оптимальне співвідношення сторін і масштаб. Даний етап відбувається на етапі розпізнавання символів як такого.

Важливим пунктом при роботі із *OCR* є вилучення ознак символів (*feature extraction*). Даний процес може бути здійснено за допомогою наступних основних методів:

1. Алгоритм виявлення ознак визначає символ, оцінюючи його лінії та штрихи.

2. Розпізнавання зразків працює шляхом ідентифікації всього символу в цілому.

Рядок тексту може бути розпізнано шляхом пошуку білих піксельних рядків, які мають чорні пікселі між ними. Так само проводиться розпізнавання того, де символ починається і закінчується.

На рисунках 1.1-1.3 приведено візуалізацію цих методів.

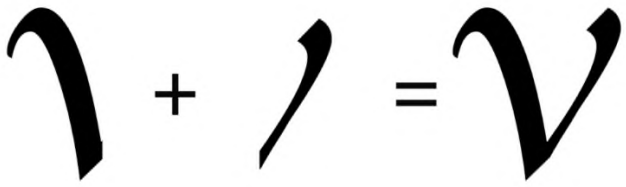

Рис. 1.1. Спосіб 1 – виявлення ознак

Рис. 1.2. Спосіб 2 – розпізнавання шаблону на одному символі.

Далі відбувається перетворення зображення символу у двійкову матрицю, де білі пікселі дорівнюють 1, а чорні пікселі – 0 (оскільки на даному етапі відбувається робота із бінаризованим зображенням, яке часто є інвертованим – світлі ділянки перетворюються в чорні, і навпаки), як показано на наступному зображенні:

Рис. 1.3. Зразок представлення символу у вигляді двійкової матриці

Потім, використовуючи формулу відстані, знаходять відстань від центру матриці до найдальшої 1, після чого створюють коло знайденого радіусу і розділяють його на більш детальні ділянки.

На цьому етапі алгоритм порівнює кожну ділянку (сектор) із набором матриць, що представляють символи з різними шрифтами, щоб статистично ідентифікувати найбільш характерний для нього символ.

Це полегшує перенесення друкованої інформації у цифровий світ, проводячи указані операції із кожним рядком та символом.

Точність *OCR* можна покращити, якщо вихід обмежений лексиконом (переліком слів, дозволених у документі). Наприклад, це можуть бути всі слова англійською мовою або більш технічний словник для певної галузі.

Цей метод може бути менш ефективним, якщо документ містить слова, яких немає в лексиці, як власні іменники.

На щастя, для підвищення точності є Інтернет-бібліотеки, доступні в Інтернеті безкоштовно. Бібліотека *Tesseract* використовує свій словник для управління сегментацією символів.

Вихідним потоком може бути один рядок або файл символів, але більш досконалі системи *OCR* зберігають оригінальну структуру сторінки і, наприклад, створюють *PDF*, що містить як оригінальні сторінки зображень, так і текстове зображення, яке можна шукати.

Метод "Аналіз ближнього сусіда" може використовувати частоти спільного виникнення, щоб виправити помилки, відзначаючи слова, що вживалось разом. Наприклад, в англійській мові словосполучення «*Washington, D.C.*» є більш поширеним, ніж «*Washington DOC*».

Граматика також може допомогти визначити скановану мову, наприклад, чи є слово швидше за все дієсловом або іменником, що забезпечує більшу точність.

При подальшій обробці *OCR* алгоритм відстані Левенштейна часто використовується для подальшої максимізації точності результатів *API OCR.*

Механізми *OCR* розроблені для цілого ряду доменних (галузевих) застосувань *OCR*, включаючи роботи із квитанціями, рахунками, платіжними чеками та юридичними документами.

Іншими випадками використання можуть виступати:

– Введення даних для ділових документів, наприклад, чеків, паспортів, рахунків-фактур, банківських виписок та квитанцій;

– Автоматичне розпізнавання номерного знаку;

– В аеропортах – розпізнавання паспорта та отримання інформації;

– Автоматичне вилучення ключової інформації страхового документа;

– Вилучення інформації про візитну картку до списку контактів;

– Створення цифрової версії друкованого документа, наприклад, відсканованої книги;

– Перетворення електронних зображень друкованих документів у такі, що дозволяють проводити пошук у них;

– Перетворення рукописного вводу в режимі реального часу для управління комп’ютером.

Приклад використання *OCR* по галузях:

Банківська справа

Банківська галузь є значним споживачем *OCR* поряд з іншими галузями економіки, такими як страхування та ринок цінних паперів.

Найпоширенішим використанням *OCR* є належне управління чеками:

1. Рукописний чек сканується;

2. Його деталі перетворюються на цифровий текст;

3. Проводиться перевірка підпису;

4. Чек переводиться у готівку в режимі реального часу.

З використанням *OCR* всі вказані дії проходять без участі людини. Скорочений час обробки чеків – це фінансова перевага для всіх – для платника, банку та одержувача.

Хоча розпізнавання платіжних чеків досягає майже 100% точність за рахунок їх стандартизовної структури (лише перевірка підпису вимагає узгодження з уже існуючою базою даних), повна автономія при роботі із рукописним текстом все ще далека від повної і точної реалізації. Однак із методами поглибленого навчання штучного інтелекту, що застосовуються до розпізнавання *OCR* рукописного вводу, це питання перестає бути таким нерозв'язним, як може здатися.

Юридична

Небагато галузей створюють стільки друкованих документів, як юридична, і тому *OCR* має тут широке застосування. Діджиталізація, зберігання та впорядкуваня оцифрованої інформації, пошук за допомогою найпростіших зчитувачів *OCR* можливі у всіх друкованих документах: письмових показаннях, судових рішеннях, постановах, заявах, заповітах тощо.

Цей підхід також доступний для записів китайською, арабською та іншими мовами, написання яких базується на специфічних символах; таким чином технології *OCR* розширюються до мов, які не використовують латинський алфавіт.

Швидкий доступ до юридичних документів з мільйонів минулих справ, безумовно, має перевагу для галузі, яка значною мірою покладається на минуле.

Охорона здоров'я

Ще одна галузь, яка широко використовує можливості *OCR* – це охорона здоров'я. Всю історію хвороби можна сканувати та зберігати на комп'ютері: звіти, результати проходження рентгенографії, попередні захворювання, особливості лікування або діагностики, аналізи, лікарняні записи, страхові виплати тощо. Із використанням *OCR* всі вони можуть бути доступними в одному місці та передбачають можливість пошуку по них.

Той факт, що вся лікарняна документація зберігається в цифровому вигляді, також є основною перевагою для епідеміології, а також для логістики (ведення відповідних аптечних магазинів, обладнання та інших споживчих товарів).

Такі записи підсумовуються в декількох лікарнях по всьому регіону, що забезпечує величезну базу даних щодо політики охорони здоров'я, постачання тощо на основі даних.

Ланцюг поставок

У харчовій, фармацевтичній та косметичній галузях контроль якості на кожному етапі процесу є критично важливим для дотримання правил безпеки та боротьби з підробкою.

Товари повинні знаходитись у ланцюзі поставок у будь-який визначений момент з чіткою інформаційною документацією про їх походження та місцезнаходження.

Хоча відстеження товару часто розглядається як використання штрих-коду, *OCR* дозволяє зчитувати коди партій, терміни придатності та серійні номери, щоб стежити за товаром на всіх етапах циклу пакування – від маркування упаковки до пакування усієї партії.

Окрім того, *OCR* програму можна побудувати таким чином, щоб проводити порівняння зчитаного тексту з очікуваним рядком, який, в свою чергу, визначено в базі даних, та позначити таким чином відсутні або невідповідні серійні номери.

Штрих-коди та *OCR* часто використовуються разом, щоб максимізувати точність збору інформації.

Переваги *OCR*

Доступність для пошуку. Сучасні *OCR*-системи дозволяють зберегти відсканований файл у форматі *.doc, .rtf, .txt* (найпростіший), *.pdf* тощо після того, як відсканований файл буде відскановано у читабельний текст. У результаті ці файли дозволяють легко проводити пошук по них без прив’язки до конкретної використовуваної операційної системи чи програмного забезпечення.

Можливість редагування. Якщо виникає потреба, наприклад, переглянути або змінити старий контракт, який був підписаний багато років тому, то після

оцифровування цього документа за допомогою *OCR*, з'являється можливість редагувати його за допомогою текстового процесора, а не вводити весь документ заново.

Доступність. Як тільки сканований документ *OCR* потрапляє до загальної бази даних, він стає доступним для всіх, хто має доступ до цієї бази даних. Це особливо корисно для банків, які можуть перевірити попередню кредитну історію клієнта в будь-який час і в будь-якому місці.

Іншим використанням може стати надання державних архівів, щоб будь-який документ, будь то запис про право власності на землю та майно або свідоцтво про народження можна було миттєво знайти з будь-якого місця.

Компактне зберігання. Оцифровування зменшує фізичний простір, необхідний для зберігання архівів документів із однієї або кількох кімнат до декількох гігабайтів на сервері, що забезпечує більшу продуктивність. Крім того, недоцільні сьогодні паперові архіви тепер можна переробити і не накопичувати значну паперову масу.

Резервне копіювання. Замість того, щоб зберігати дорогі паперові дублікати та копії у паперовій формі, цифрові резервні копії можна зробити дешево та у необмеженій кількості. Крім того, *OCR* пропонує набагато більше стабільності в управлінні документацією.

Переклад з однієї мови на іншу. Сучасні *OCR* можуть управляти великою кількістю мов – від арабської до індійської та китайської. Це означає, що документ, написаний однією мовою, можна оцифровувати та перекласти будь-якою іншою мовою. Цей процес вдалось значно спростити за допомогою використання *Unicode Standard* та комп'ютерних програм перекладу на основі машинного навчання (наприклад, *Google Translate* або певні спеціалізовані засоби).

1.2 *Tesseract OCR*

Tesseract – *OCR*-двигун з відкритим кодом, який завоював популярність серед розробників *OCR*. Незважаючи на те, що іноді може бути складно його імплементувати та модифікувати, довгий час на ринку не було достатньо потужних

та безкоштовних альтернатив *OCR*. *Tesseract* починався як дослідницький проект у *HP Labs*, Брістоль.

Він набув популярності і був розроблений компанією *HP* в період з 1984 по 1994 рік. У 2005 році компанія *HP* випустила *Tesseract* як програмне забезпечення з відкритим кодом. З 2006 року розробляється компанією *Google*. Даний механізм розпізнавання тексту з відкритим кодом доступний за ліцензією *Apache* 2.0, що дозволяє безкоштовно використовувати його безпосередньо або (при розробці ПЗ) за допомогою *API* для вилучення друкованого тексту із зображень. Він підтримує широкий спектр мов. Tesseract не має вбудованого графічного інтерфейсу, але є декілька програм від сторонніх розробників, що надають графічний інтерфейс.

Tesseract сумісний з багатьма мовами програмування та фреймворками через спеціальні обгортки. Його можна використовувати у парі з аналізом наявного макета для розпізнавання тексту у великому документі, або можна використовувати разом із зовнішнім детектором тексту для розпізнавання тексту із зображення із мінімумом програмного коду.

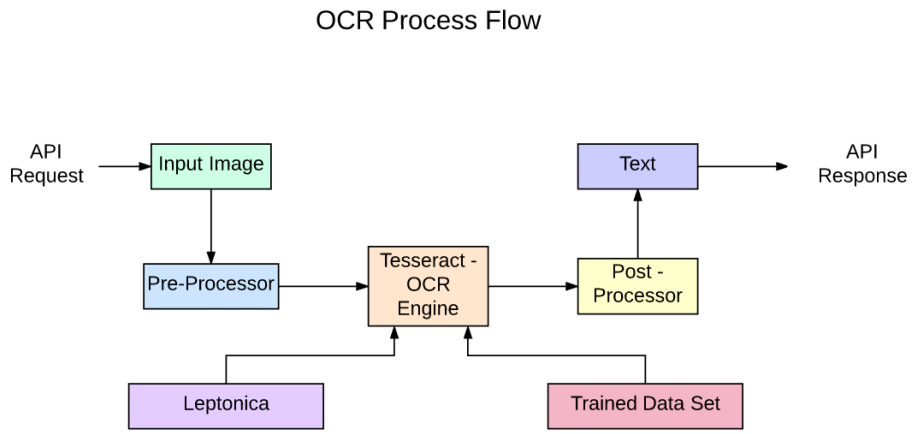

Рис. 1.4. Послідовність робити Tesseract при використанні *API*

Tesseract 4.00 включає нову підсистему нейронної мережі, налаштовану як розпізнавач текстових рядків. Вона бере свій початок від реалізації *LSTM* на базі *Python* від *OCRopus*, але була перероблена для *Tesseract* на мові програмування *C++*. Система нейронних мереж у *Tesseract* передувала бібліотеці *TensorFlow*, але сумісна з нею, оскільки в ній (системі нейронних мереж *Tesseract*) використовується мова

опису мережі, яка називається *Variable Graph Specification Language* (*VGSL*, мова специфікації змінних графів), яка також доступна для *TensorFlow*.

Щоб розпізнати зображення, що містить один символ, ми зазвичай використовуємо згорткову нейронну мережу (*CNN*). Текст довільної довжини - це послідовність символів, і такі проблеми вирішуються за допомогою *RNN*, а *LSTM* є популярною формою *RNN*.

LSTM чудово піддаються навчанню послідовностей, але сильно уповільнюються, коли кількість станів завелика. Є емпіричні результати, які свідчать про те, що *LSTM* вдається краще вивчити довгу послідовність, ніж коротку послідовність багатьох класів. Tesseract розроблений на основі моделі *OCRopus* в Python, яка була відгалуженням *LSTM* в *C++*, що називається *CLSTM*. *CLSTM* – це реалізація рекуррентної моделі нейронної мережі *LSTM* в *C++*, що використовує бібліотеку *Eigen* для чисельних обчислень.

Попередня версія (*Tesseract 3.X*) залежала від багатоетапного процесу, в якому виділяють наступні кроки:

– Пошук слів;

– Знаходження лінії;

– Класифікація символів.

Пошук слів здійснювався шляхом упорядкування текстових рядків у крапки, а рядки та області аналізувались на наявність фіксованого тону або пропорційного тексту. Рядки тексту розбиваються на слова по-різному залежно від виду інтервалів між символами. Потім визнання триває як двопрохідний процес. Під час першого проходу робиться спроба розпізнати кожне слово по черзі. Кожне задовільне слово передається в адаптивний класифікатор як навчальні дані. Тоді адаптивний класифікатор отримує можливість точніше розпізнати текст, розташований далі на сторінці.

Модернізація інструменту *Tesseract* була зусиллям щодо очищення коду та додавання нової моделі *LSTM*. Вхідне зображення обробляється у попередньо виділених прямокутниках, рядок за рядком, подаючи в модель *LSTM* і даючи вихідні дані.

Після додавання нового інструменту для навчання та навчання моделі з великою кількістю даних та шрифтів, *Tesseract* досягає кращих показників, однак, недостатньо високих, щоб працювати над рукописним текстом та нетиповими шрифтами. Можна провести точну настройку або перекваліфікувати верхні шари для експериментів.

Tesseract найкраще працює, коли є чітка сегментація тексту на передньому плані відносно його фону. На практиці гарантувати такі типи налаштування може бути надзвичайно складно. Існує безліч причин, через які *Tesseract* може не отримати якісний вихідний результат, наприклад, якщо зображення має певні шуми на фоні. Чим краща якість зображення (розмір, контраст, освітлення), тим кращий результат розпізнавання буде продемонстровано. Це вимагає певної попередньої обробки для поліпшення результатів *OCR*, зображення потрібно масштабувати належним чином, мати якомога більший контраст зображення, а текст повинен бути вирівняний по горизонталі. Tesseract *OCR* є досить потужним, але має наступні обмеження:

– Tesseract *OCR* не такий точний, як деякі комерційні рішення, доступні на ринку сьогодні;

– Не вдається показати точне розпізнавання символів при роботі із зображеннями, на які впливають артефакти, такі як часткова оклюзію, спотворена перспектива та складний фон;

– Він не здатний розпізнавати почерк;

– Він не може визначити, що вхідні дані є беззмістовними та повідомити про це як вихідний результат *OCR*;

– Якщо документ містить мови, які не вказані в аргументах *-l LANG*, результати можуть бути недостатньо якісними;

– Не завжди вдається аналізувати природний порядок читання документів. Наприклад, *Tesseract* може не розпізнати, що документ містить два стовпці, і в такому випадку спробує об'єднати текст із двох колонок в одну;

– Неякісні скани можуть призвести до низькоякісного розпізнавання;

– Він не надає інформації про те, до якого сімейства шрифтів належить текст.

1.3 *Google Cloud Vision AI*

Google Cloud Vision AI – хмарна програмна платформа, розробена компанією Google, що надає послуги використання комп’ютерного зору.

Google Cloud пропонує два продукти, що реалізують комп’ютерний зір, які використовують машинне навчання, щоб наблизитись до максимально точного аналізу того, що саме знаходиться на зображенні:

– *AutoML Vision* – довзоляє автоматизувати навчання власних моделей машинного навчання. Даний процес побудований наступним чином: користувач завантажує підготований датасет зображеннь та тренує на них власні моделі за допомогою простого у використанні графічного інтерфейсу *AutoML Vision*; користувачу надається можливість оптимізації своїх моделей для підвищення точності, мінімізації затримки та розміру. Після того, як модель надає достовірні результати, користувач може експортувати її до свого хмарного додатку або до масиву *edge*-пристроїв (пристрої, що забезпечують зв'язок локальних мереж із зовнішніми, глобальними або іншими локальними, мережами).

– *Vision API – API*, що пропонує використати за допомогою *REST* та *RPC API* попередньо навчені моделі машинного навчання. Даний застосунок дозволяє призначати ярлики зображенням і класифікувати їх на заздалегідь визначені категорії.

Окрім цього, *Vision API* надає ряд наступних функцій:

– Розпізнавання обличчя – виділяє обличчя та визначає конкретні орієнтири обличчя, такі як очі, вуха, ніс, рот тощо. Також повертаються ймовірності емоцій (радість, смуток, гнів, здивування). Оцінки ймовірностей виражаються одним із 6 значень, від «Невідомий» (не вдалось визначити імовірність) до «Дуже імовірний» (дуже висока імовірність);

– Виявлення орієнтиру – вказує назву орієнтира, оцінку достовірності та обмежувальне поле на зображенні для орієнтира. Надає координати для виявленого місця;

– Виявлення логотипу – надає текстовий опис ідентифікованої сутності, оцінку достовірності та обмежуючий багатокутник для логотипу у файлі;

– Виявлення ярликів – надає узагальнені мітки для зображення. Для кожного ярлика повертається текстовий опис, оцінка впевненості та рейтинг актуальності;

– Розпізнавання тексту – оптичне розпізнавання символів (*OCR*) на зображенні. Ідентифікує та вилучає текст в кодуванні *UTF-8* із зображення. Повертає список слів, ідентифікованих як текст, обмежувальні поля, а також структурну ієрархію тексту, виявленого *OCR*, із наступними рівнями: *TextAnnotation* > Сторінка > Блок > Абзац > Слово > Символ. Кожен структурний компонент від рівня «Сторінка» і далі може мати свої власні властивості, такі як виявлені мови, розриви тощо;

– Виявлення тексту документа (щільний / рукописний текст) – оптичне розпізнавання символів (*OCR*) для файлів (*PDF* / *TIFF*) або зображення щільного тексту. Оптимізовано для роботи з файлами документів (*PDF* / *TIFF*) та зображеннями рукописного тексту;

– Властивості зображення – повертає домінуючі кольори на зображенні. Кожен колір представлений у колірному просторі *RGBA*, має оцінку достовірності та відображає частку пікселів, зайнятих кольором в межах [0, 1];

– Локалізація об'єкта – надає загальні анотації міток та обмежувальних полів для кількох об’єктів, розпізнаних на одному зображенні. Для кожного виявленого об'єкта повертаються такі елементи: текстовий опис, оцінка достовірності та нормалізовані вершини [0,1] для обмежувального багатокутника навколо об'єкта;

– Виявлення відвертого вмісту (безпечний пошук) – Забезпечує рейтинги ймовірності для категорій відвертого вмісту, таких як насильницький, злочинний тощо. Оцінки ймовірностей виражаються як 6 різних значень, від «Невідомий» до «Дуже імовірний».

Розпізнавання тексту з використанням *Vision API* дозволяє визначати цілий ряд мов, які підтримуються даним застосунком. Всього вони поділяються на 3 групи:

– Підтримувані мови – ті, які визначені як пріоритетні та регулярно оновлюються та підтримуються розробниками платформи; дана група містить 60 мов, включаючи українську.

– Експериментальні мови – їх підтримка активно розробляється, але повної підтримки на даний момент не надано; на сьогодні налічує 50 мов.

– Співставлені мови – підтримуються шляхом співставлення їх до іншого мовного коду або до загального розпізнавача символів.

Оскільки *Vision API* виступає хмарним сервісом, то для того, щоб взаємодіяти із ним, потрібне постійне з'єднання з мережею Інтернет. Для роботи із даною платформою передбачено три способи:

– Використання графічного інтерфейсу – здійснивши вхід у свій обліковий запис, користувач може завантажувати зображення для обробки згідно з обраним типом *API* (розпізнавання тексту, локалізація об’єкту тощо) і отримувати відповідь безпосередньо через браузер;

– Засобами командного рядку – при даному підході користувачем через командний рядок відправляється *POST*-запит, в який вкладається *JSON*-файл, у якому міститься інформації про тип обробки зображення, шлях до нього та особистого ключа *API*. У відповідь, яку надсилає *Vision API*, вкладається результат обробки зображення.

– Застосування клієнтських бібліотек конкретних мов програмування – для ряду мов програмування (*C#, Go, Java, PHP, Python, Ruby*, фреймворк *Node.js*) реалізовані бібліотеки *Google Cloud Vision API*, які інкапсулюють в собі нюанси роботи із даною платформою. При такому підході взаємодія із *Vision API* зводиться до завантаження зображення, його відправки та обробки отриманих результатів – усе це з використанням відповідної бібліотеки та обраної мови програмування.

Як і ряд своїх аналогів, *Google Cloud Vision API* надає свої послуги на платній основі. Вартість залежить від об’ємів використання обраної функції: наприклад, розпізнавання тексту для першої 1000 зображень в місяць проводиться безкоштовно, при перевищенні цієї позначки вартість обробки складає 1,50 доларів США (при кількості оброблюваних зображень в межах 1 001 – 5 000 000) за кожну тисячу зображень або 0,60 доларів США за кожну тисячу операцій (при кількості зображень понад 5 000 000).

Варто відзначити, що, в цілому, дана *OCR* отримує схвальні відгуки серед спільноти фахівців, що працюють із *OCR*-додатками. Даний програмний інтерфейс

надає можливість розпізнавати текст як в режимі реального часу, так і з уже готових зображень текстових документів.

До переваг даної *OCR* можна віднести наступне:

– Простота налаштування без потреб надлишкового налаштування та адаптації;

– Наявність вбудованих алгоритмів комп'ютерного зору та модулів обробки відео, що спрощує обробку фото- та відео-матеріалів;

– Швидкість – даний *API* виділяється на фоні своїх аналогів за рахунок високої швидкості обробки зображень;

– Масштабованість;

– Високі показники точності розпізнавання символів;

– Можливість розпізнавання тексту в реальному часі.

Попри наявність ключових переваг, було виявлено наступні недоліки:

– Вартість обслуговування – при наявності значного об'єму оброблюваних зображень ціна за виконання операцій обробки може бути недосяжною (якщо розроблюваний проект є некомерційним або продуктом для особистих цілей);

– Потреба постійного з'єднання з мережею Інтернет – обробка зображень в офлайн-режимі все ще не реалізована;

– Документація покриває не усі поширені сценарії та проблеми при роботі із плафтормою;

– Навіть при обробці тексту, який містить мову, що підтримується, результати розпізнавання все рівно можуть бути далекими від прийнятних навіть при чіткому вхідному зображенні.

1.4 *ABBYY FineReader Engine*

ABBYY FineReader Engine (скорочено *FRE*) – багатофункціональний інструментарій розробника для вбудовування передових технологій розпізнавання, конвертації і класифікації документів: оптичного розпізнавання символів (*OCR*), інтелектуального розпізнавання рукописних символів (*ICR*), оптичного

розпізнавання міток (*OMR*), читання штрих-кодів, обробки зображень документів, конвертації *PDF*-файлів, автоматичної класифікації і сортування документів.

Технології розпізнавання від *ABBYY* широко використовуються постачальниками рішень і продуктів для вирішення завдань по введенню і перетворенню зображень в редаговані формати в наступних сценаріях:

– Обробка документів в системах електронного документообігу;

– Створення електронних архівів документів;

– Організація повнотекстового пошуку в системах управління контентом і аналізу великих обсягів невизначено структурируемой інформації (системи *Big Data*);

– Створення електронних медичних карт – джерел даних про пацієнтів і численних записів про медичне обслуговування;

– Введення бухгалтерських і фінансових документів;

– Мобільне введення даних і документів з використанням фотокамер мобільних пристроїв;

– Рішення з читання тексту для людей з обмеженими можливостями.

При роботі із *FRE* можна виділити такіе етапи обробки зображень:

1. Імпорт зображень. *ABBYY FineReader Engine* підтримує роботу із зображеннями, отриманими з різних джерел: з сканера або безпосередньо з пам'яті, - а також завантаження будь-яких файлів 15 підтримуваних форматів (*TIFF, PNG, PDF, JPEG, BMP* тощо);

2. Попередня обробка зображень. Інструментарій розробника надає доступ до бібліотеці функцій попередньої обробки зображень. Обробка допомагає досягти істотного підвищення якості розпізнавання зображення;

3. Аналіз структури документа. Технологія аналізу структури документа включає в себе наступні можливості:

– автоматична конвертація документа з збереженням його оригінальної структури;

– режим пошуку всіх текстових полів на документі;

– підтримка ручної вказівки областей розпізнавання;

– спеціальні настройки для обробки бухгалтерської документації тощо.

4. Розпізнавання документів. Технології *FRE* забезпечують максимально точне розпізнавання друкованих текстів (*OCR*) на 210 мовах, рукописних текстів (*ICR*) на 135 мовах, оптичного розпізнавання міток (*OMR*) і читання штрих-кодів. *API* продукту пропонує можливості навчання на основі шаблонів користувача і створення унікальних користувацьких мов.

5. Експорт результатів розпізнавання. *ABBYY FineReader Engine* пропонує безліч варіантів експорту результатів розпізнавання, включаючи можливість настройки таких параметрів експорту, як роздільна здатність зображення, вибір ступеня відтворення структури документа та інших.

Засоби розробки *ABBYY FineReader Engine* надають розробникам можливість налаштувати своє рішення для досягнення максимально можливої якості і швидкості розпізнавання завдяки наступним особливостям продукту:

– Контроль над настройками на всіх етапах процесу обробки документів - від імпорту до експорту завдяки функціям розширеного *API*;

– Проста масштабованість рішення завдяки вбудованим засобам розпаралелювання завдань і підтримки багатоядерної архітектури;

– Оптимізація часових та ресурсних витрат на розробку за допомогою заздалегідь налаштованих профілів з попередньо налаштованими значеннями параметрів розпізнавання для вирішення типових завдань, розгорнутої бібліотеки прикладів коду і готового набору візуальних компонент для створення інтерфейсу користувача;

– Штучний інтелект та машинне навчання. Технології на основі *AI* та *ML* у поєднанні з *ADRT* (*ABBYY Adaptive Document Recognition Technology* – технологія адаптивного розпізнавання документу *ABBYY*) та іншими технологіями *ABBYY* виявляють логічну структуру документа та визначають його елементи форматування, такі як зміст, заголовки, колонтитули, підписи до рисунків (див. рис. 1.5.) шрифти та стилі шрифтів, щоб точно відтворити оригінальний документ;

– Підтримка хмарних та віртуальних середовищ. Додатково до локального розгортання, розробники програмного забезпечення мають можливість розгорнути свої програми з використанням *ABBYY FRE* у віртуальному середовищі або

розмістити свої рішення на хмарних платформах, таких як *Microsoft Azure* або *Amazon Web Services*.

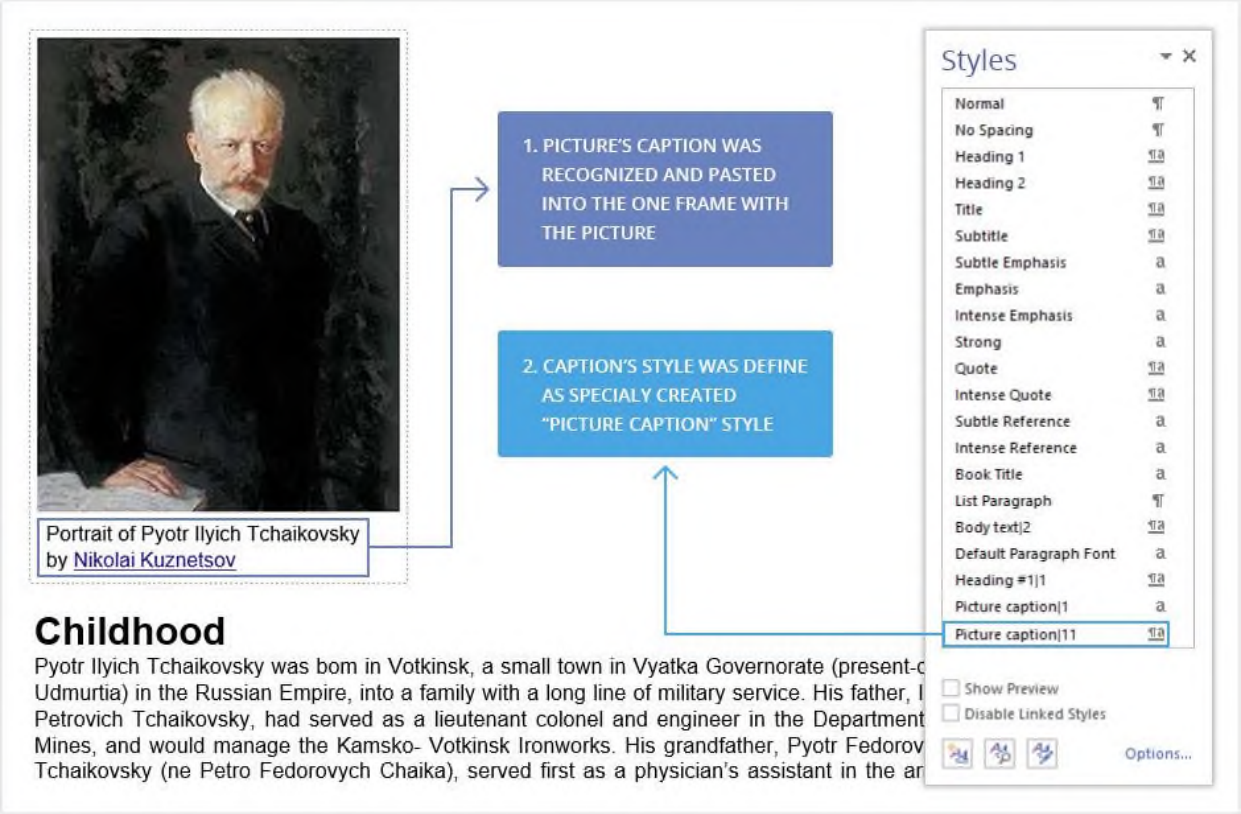

Рис. 1.5. Результат розпізнавання елементу «Підпис рисунку»

Можливості:

– Передові технології розпізнавання тексту і конвертації документів. Компанія *ABBYY* є визнаним світовим лідером з розробки та застосування технологій розпізнавання друкованих текстів (*OCR*), рукописних символів (*ICR*), різних видів штрихкодів, міток (*OMR*) і зонального розпізнавання для перетворення відсканованих або сфотографованих зображень в редаговані текстові файли.

– Профілі розпізнавання. У складі продукту є заздалегідь налаштовані профілі з оптимальними значеннями параметрів обробки для вирішення типових задач розпізнавання і перетворення документів. Тепер, вибравши відповідний профіль, розробник отримує оптимальні настройки для вирішення конкретного завдання, які, однак, розробник має можливість відкорегувати за бажанням.

– Попередня обробка зображень. *FRE* пропонує широкі можливості для попередньої обробки зображень – усунення перекосів, поворот, виправлення

трапецієподібних перекручувань, випрямлення ліній тексту, поділ подвійних сторінок, адаптивна бінаризація та інші.

– Збереження структури документа. *ABBYY FineReader Engine* забезпечує набір можливостей для доступу до елементів структури документа, таким як заголовки, розділи документа, номера сторінок, виноски і ін. Функції даного *API* дозволяють точно і акуратно внести зміну в документ, зберігаючи при цьому його оригінальну структуру.

– Підтримка паралельної обробки на декількох ядрах. Підтримка багатопроцесорної архітектури забезпечує масштабованість рішення і збільшує швидкість розпізнавання за рахунок поділу багатосторінкового документа на частини і одночасної обробки цих частин на кількох ядрах процесора. Таким чином, загальна продуктивність рішення збільшується з кожним додатковим мікропроцесорним ядром.

Таким чином, до переваг *ABBYY FRE* можна віднести наступні особливості:

– Висока точність розпізнавання. Технології *ABBYY* отримали визнання у всьому світі завдяки високій точності розпізнавання, яка є ключовим параметром *OCR*-технологій. *ABBYY FineReader Engine* розпізнає з високою точністю як європейські, так і азіатські (китайська, японська, тайська, іврит) мови.

– Краще співвідношення швидкості до якості розпізнавання. *ABBYY FineReader Engine* за замовчуванням пропонує оптимальне поєднання швидкості і якості розпізнавання, а також кілька додаткових режимів для підвищення швидкості обробки. Суттєвого приросту швидкості можна також досягти, використовуючи розпаралелювання задач в режимі багатоядерної обробки.

– Передова технологія обробки фотографій документів *Camera OCR*. Оцифровка документів за допомогою цифрових фотокамер з кожним роком набуває все більшого поширення. Відповідаючи на цей тренд, *ABBYY* розробила спеціалізовану технологію *ABBYY Camera OCR* – набір інструментів для обробки фотографій документів, за допомогою яких можна підвищити якість їх розпізнавання.

– Світовий лідер за кількістю підтримуваних мов розпізнавання. *ABBYY FineReader Engine* підтримує розпізнавання друкованих текстів на 210 мовах, а

рукописних – на 135. Таким чином, інструментарій розробника дозволяє створювати додатки для ринків будь-яких країн.

Важливим нюансом є те, що *ABBYY FineReader Engine*, розгорнутий на платформі Windows підтримує *COM* стандарт, і може використовуватись лише у мовах програмуваннях та інструментах, що підтримують цей стандарт, таких як *C / C++*, мовах програмування платформи *.NET, Delphi, Java*, а також скриптових мовах (*JS* або *Perl*). Для операційних систем *Linux FRE* надається *API* для мов *C / C++* і обгортка для використання *Java*, а в *MacOS* можливе використання лише використання нативного *API* для *C / C++*. Цей фактор накладає значне обмеження на вибір інструментарію для роботи із даним застосунком, тому що обгорток для, наприклад, таких поширених мов, як *Python* або *PHP*, не існує.

Компанія *ABBYY* була і залишається флагманом на ринку *OCR*-технологій, а її рівень її продуктів задає темп розвитку усієї галузі. Застосунки *ABBYY* зайняли цю нішу на усіх рівнях – від особистого і домашнього використання до застосування в бізнесі і розробці програмних інструментів. Ціною за використання таких передових технологій є, власне, їх вартість. Інтеграція *ABBYY FineReader Engine* у розроблюваний проект потребує підписання юридичного контракту та індивідуального обговорення фінансового аспекту (мінімальна ціна вимірюється в сотнях доларів США за відносно незначні об’єми розпізнавання). Цей фактор може унеможливити використання таких технологій в невеликих або некомерційних проектах. В якості альтернативи може виступити інший продукт компанії *ABBYY* – *Cloud OCR SDK*, хмарне рішення на основі *Microsoft Azure*, яке надає аналогічний функціонал при нижчій вартості (100 доларів США за 2 000 сторінок в місяць) і не потребує підписання контракту. Окрім того, при дотриманні ряду умов може бути отримано доступ до цього застосунку на безкоштовній основі (виключно для освітніх некомерційних проектів).

1.5 Порівняння розглянутих *OCR*-додатків

В таблиці 1.1 приведені деякі із показників, що демонструють оновлюваність *OCR*-додатків, охоплення мов, можливість інтеграції із проектами тощо.

Таблиця 1.1

Порівняння розглянутих *OCR*-додатків

	Tesseract	*Google Cloud Vision API*	*ABBYY FineReader Engine*
Дата останньої стабільної версії	26/12/2019	18/09/2019	08/09/2020
Цінова політика	Безкоштовно	Платно, за виконані операції	Платно, вартість індивідуальна
Робота офлайн	Так	Ні	Так
Кількість мов	100+	110	210
Підтримка української мови	Так	Так	Так
Мови програмування, фреймворки	*Python*, *JS*, *TypeScript*, *C*, *C++*, *C#*, *PHP*, *Java*, *Ruby*, *Go*, *Swift*	*C#*, *Go*, *Java*, *PHP*, *Python*, *Ruby*, *Node.js*	*C*, *C++*, *.NET*, *Delphi*, *Java*, *JS*
Формат результату розпізнавання	Текст, *PDF*, *TXT*, *HOCR*, *TSV*	Текст	*TXT*, *PDF*, *CSV* та 13 інших
Інструменти передобробки	Ні	Так	Так
Можливість адаптації до специфічних сценаріїв	Так	Так	Так

Усі із розглянутих додатків підтримують значну кількість мов, в тому числі і українську, дозволяють адаптувати їх реалізацію до особливих предметних областей та надають вибір із форматів виведення результатів. Разом із тим, за високу

ефективність роботи *OCR*, яка властива платним рішенням, встановлюється немала ціна за їх використання. Також важливим аспектом є можливість обробляти зображення без підключення до мережі Інтернет, якої немає у хмарних додатків.

Через високу вартість *ABBYY FineReader Engine*, а також обмежену кількість підтримуваних мов програмування його використання в даній роботі не є можливим. *Google Cloud Vision API*, в свою чергу, накладає обмеження у вигляді функціонування при умові підключення до мережі Інтернет. Тому для програмної реалізації дипломної роботи було обрано *OCR*-додаток *Tesseract,* що поширюється безкоштовно, надає можливість використання функціоналу в режимі офлайн і вільного вибору мови програмування, що є вкрай важливим фактором, оскільки окрім розпізнавання тексту потрібно проводити ряд інших операцій, таких як робота із зображеннями та алгоритмами машинного навчання, що потребує використання сторонніх бібліотек, які мають надавати потужний та зручний функціонал. Відповідно, чим більший вибір мови програмування – тим більше можливостей коректної та надійної інтеграції із іншими модулями програми.

1.6 Висновки до розділу

В ході написання даного розділу були виконані наступні пункти:

– Описано призначення *OCR*-технологій та їх проблематику;

– Визначено послідовність роботи із розпізнаванням символів;

– Виділено низку переваг та можливостей застосування оптичного розпізнавання символів у різних сферах бізнесу;

– Розглянуто ряд сучасних *OCR*-систем, їх можливості, функціонал та переваги;

– Проведено аналіз можливостей додатків та здійснено вибір *OCR*-додатку в розроблюваному програмному модулі.

Було встановлено, що *OCR* має ряд переваг, таких як рух до діджиталізації. У бізнесі часто існує дуже великий обсяг даних та документів, контрактів, державних бланків, ліцензій тощо.

Використання ж *OCR* при оцифровуванні документів робить їх доступними для сучасного аналізу, який може закласти основу для довгострокових вдосконалень у бізнесі. Це дозволяє виявити збиткові напрямки підприємства, перевіряти наявність відхилень від початкових положень та умов за підписаним контрактом. Так само *OCR* довзоляє перевірити платіжні документи, порівняти рахунки-фактури тощо.

Визначено, що діджиталізація має цілий ряд переваг, і *OCR* є найважливішим першим етапом трансформації паперових та аналогових носіїв у цифрові записи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ МОДУЛЯ РОЗПІЗНАВАННЯ ЧЕКІВ

2.1 Огляд предметної області

Завданням розроблюваного проекту є розпізнавання товарів із фотографії касового чеку (його зразок приведено на рис. 2.1) та встановлення відповідної категорії для кожного із товарів.

Рис. 2.1. Фотографія касового чеку

Розроблюваний продукт оброблятиме касові чеки із магазинів загального призначення, які поширюють товари ряду категорій – від напоїв та бакалії до товарів побутового характеру. Вибір даного напрямку реалізується за рахунок підготовки навчального датасету, який міститиме перелік товарів та відповідних їм категорій.

Прямої залежності від обраного виду чеків немає – для того, щоб змінити сферу, наприклад, на чеки аптек, магазинів одягу чи закладів громадського харчування, потрібно замінити або доповнити датасет відповідними товарами, після чого заново провести навчання моделі.

Вхідними даними в цьому випадку виступає фотографія чеку, а результатом роботи (вихідними даними) є перелік категорій, до яких належать товари, вказані на чеку, із їх сумарною вартістю.

Завдання роботи зводиться до роботи виключно із тими даними, які відображають характер та об’єм покупок, однак на чеку також присутні і інші текстові поля, які у даному контексті не несуть змістовного навантаження. Використання обхідних рішень, таких як обрізання фотографії до тієї частини, де перелічені товари, не є універсальним і правильним, тому що робота розроблюваного модуля повинна бути універсальною. Тому потрібно провести фільтрацію – ті записи, що не є товарами, повинні відсіятись. Цього можна досягти за рахунок співставлення розташування виявленого текстового блоку та координат блоку ціни, що для товарів розташовується на такому ж горизонтальному рівні із ними. Цінові блоки, в свою чергу, можуть бути визначені за допомогою використання регулярних виразів.

Виконання даного завдання включає в себе наступну послідовність кроків:

– Усунення перекосу (вирівнювання зображення таким чином, щоб чек перебував у вертикальному положенні);

– Попередня підготовка зображення (для підвищення точності розпізнавання тексту відповідним чином покращити якість зображення, провести розширення тексту та виділити його контури);

– Розбиття отриманих контурів на контури окремих товарів;

– Розпізнавання тексту;

– Класифікація товарів в рамках попередньо визначених категорій;

– Обчислення сумарних вартостей для кожної встановленої категорії.

2.2 Опис послідовності роботи модуля

2.2.1. Усунення перекосу.

При роботі із зображенням чеку допускається, що на фотографії чек розміщено під певним нахилом відносно вертикальної осі в межах від -90° до 90°. При розпізнаванні тексту, що є одним із наступних етапів роботи модуля, найкращі результати досягаються тоді, коли строки будуть розташовані максимально горизонтально. Для цього потрібно усунути перекіс – повернути зображення таким чином, щоб чек на фотографії знаходився вертикально.

Завдання усунення перекосу чека виконується за допомогою використання алгоритму Кенні та перетворення Хафа.

Алгоритм Кенні – багатоступінчастий алгоритм для виявлення меж (границь) на цифровому зображенні, розроблений Джоном Кенні в 1986 році.

Кенні ввів таке поняття як заглушення «немаксимумів» (*non-maximum suppression*), що означає, що пікселями границь є тільки ті пікселі, в яких досягається локальний максимум градієнта в напрямку вектора градієнта. Решта ж значень обнуляються. Даний крок багатоступінчастого оператора призводить до більш чіткого результату.

Крім окремих специфічних випадків важко знайти детектор границь, який працював би значно краще, ніж детектор Кенні. Завданням Кенні була розробка оптимального алгоритму виділення кордонів, який би задовольняв наступним вимогам:

– Точне виявлення кордонів (підвищення відношення сигнал / шум);

– Влучна локалізація (точне визначення положення межі);

– Один відгук на одну границю.

З цих вимог будувалася цільова функція вартості помилок, мінімізацією якої знаходиться оптимальний лінійний оператор для згортки із зображенням.

Алгоритм детектора меж Кенні виконує не тільки обчислення градієнта згладженого фільтром Гаусса зображення. У контурі границі видаляються локально не максимальні точки, що знаходяться поруч з нею. Тут також використовується інформація про напрямок границі для того, щоб видаляти точки саме поруч з нею і не розривати саму межу поблизу локальних максимумів градієнта. Після цього з

використанням порогової фільтрації видаляються слабкі границі. Фрагмент межі при цьому обробляється як ціле. Якщо значення градієнта десь на відстежуваному фрагменті перевищить верхній поріг, то цей фрагмент залишається також «допустимою» границею і в тих місцях, де значення градієнта падає нижче цього порогу, до тих пір, поки вона не стане нижче нижнього порога. Якщо ж на встому фрагменті немає жодної точки з значенням великим верхнього порогу, то він видаляється. Такий гістерезис дозволяє знизити число розривів в вихідних межах. Включення в алгоритм Кенні шумозаглушення з одного боку підвищує стійкість результатів, а з іншого – збільшує обчислювальні витрати і призводить до спотворення і навіть втрати подробиць кордонів. Так, наприклад, таким алгоритмом можуть заокруглюватись кути об'єктів і руйнуватись границі в точках з'єднань.

Алгоритм Кенні розділяють на наступні основні пункти:

1. Переведення кольорового зображення в градації сірого;

2. Згладжування зображення. Проводиться шляхом накладання фільтра Гаусса на зображення з попереднього кроку;

3. Пошук градієнтів. Межі на зображенні можуть перебувати в різних напрямках, тому алгоритм Кенні використовує чотири фільтри для виявлення горизонтальних, вертикальних і діагональних границь. Скориставшись оператором виявлення кордонів (Наприклад, оператором Собеля або Щарри), отримують значення для першої похідної в горизонтальному і вертикальному напрямку. З отриманих значень обчислюють кут напрямку границі, який округляється до одного з чотирьох кутів, які представляють вертикаль, горизонталь і дві діагоналі (див. рис. 2.2).

4. Заглушення немаксимумів. Головним завданням цього кроку є переведення із «згладжених» граней зображення величин градієнта в «чіткі» межі. Простіше кажучи, це реалізується за допомогою збереження всіх локальних максимумів градієнта зображення, всі інші значення прирівнюють до нуля. Кроки алгоритму для кожного пікселя зображення градієнта наступні:

– порівняти силу границі поточного пікселя з силою границі пікселів як в позитивному, так і негативному напрямку градієнта;

– якщо сила границі поточного пікселя більше сил границь сусідніх пікселів, тоді залишаємо значення, інакше робимо значення пікселя рівним нулю.

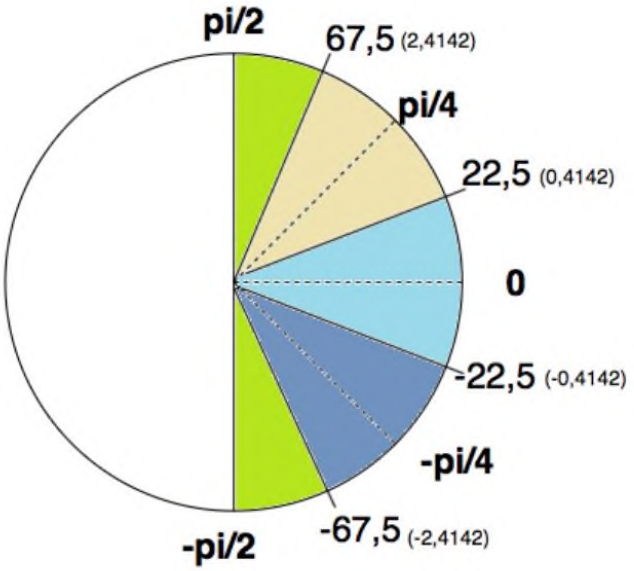

Рис. 2.2. Округлення кута градієнта

5. Подвійна порогова фільтрація. Потенційні межі визначаються порогами. Після виконання попереднього кроку, граничні пікселі і їх сили границь будуть ідентифіковані. Серед них більшість буде є реальними гранями зображення, але деякі не будуть такими.

Найпростіший спосіб розрізнити два типи пікселів – використання порогу. Алгоритм пошуку границь Кенні використовує подвійну порогову фільтрацію, в якій граничні пікселі зі значенням сили більшим за високий поріг ідентифікується як «сильні», граничні пікселі, що «слабші» низького порогу видаляються; пікселі, що знаходяться між двома порогами, позначаються як «слабкі».

Перетворення Хафа – алгоритм, що застосовується для параметричної ідентифікації геометричних елементів (прямих, кругів, еліпсів тощо) на зображенні. Його призначення – вирішити проблему групування границчних точок шляхом застосування визначеної процедури голосування до набору параметризованих об’єктів зображення.

Пряма на площині описується рівнянням $y = kx + b$ і може бути задана парою неспівпадаючих точок. Однак зручніше представити пряму за допомогою двох інших параметрів – ρ і θ. Параметр ρ – це довжина перпендикуляра, проведеного до прямої

з початку координат, а θ – це кут між даними перпендикуляром і віссю *Ox* (див. рис. 2.3). Площину (ρ, θ) іноді називають простором Хафа (*Hough space*) для набору прямих в 2-вимірному випадку, або фазовим простором.

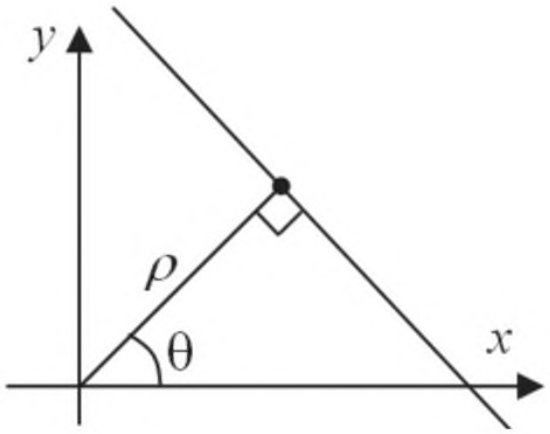

Рис. 2.3. Задання прямої на площині параметрами ρ і θ

Через одну точку в декартовій площині можна провести нескінченне число прямих. Якщо ця точка має координати (x_0, y_0) на зображенні, то всі прямі, що проходять через неї, відповідають наступному рівнянню

$$\rho(\theta) = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$

Це відповідає синусоїдальній кривій в просторі (ρ, θ). В свою чергу, кожній точці простору (ρ, θ) відповідає набір точок (x_0, y_0) на зображенні, що утворює пряму.

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 (ρ_0, θ_0) для цієї лінії. Тому, проблема виявлення колінеарних точок може бути зведена до проблеми виявлення кривих, що перетинаються.

Кожній точці (ρ_0, θ_0) простору (ρ, θ) можна співставити лічильник, що відповідає кількості точок (x, y), що лежать на прямій $\rho_0 = x \cdot \cos\theta + y \cdot \sin\theta$.

Таким чином, достатньо вибрати на фазовому просторі точки, які мають найбільше пересічень, отримавши тим самим параметров відповідної прямої.

Приклад поєднання вищеописаних методів приведено на рис. 2.4-2.6.

Рис. 2.4. Вхідне зображення

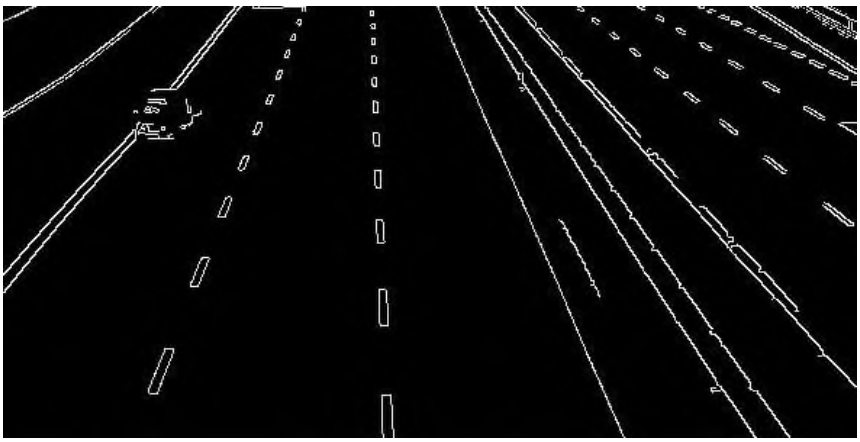

Рис. 2.5. Результат обробки зображення алгоритмом Кенні

Рис. 2.6. Результат виділення ліній на вхідному зображенні перетворення Хафа

В результаті застосування алгоритму Кенні та перетворення Хафа будуть встановлені лінії, що виступають межами чеку. Після цього вираховуються кути

нахилу даних ліній, на основі чого буде здійснено поворот зображення в напрямку, необхідному для горизонтального розташування тексту.

Для коректної роботи модуля потрібно, щоб чек займав максимальну ділянку зображення, а фон був однорідним і відділявся від чеку (що дозволить правильно бінаризувати зображення для розпізнавання тексту). В такому випадку шанси на коректну подальшу обробку фотографії зростають. Алгоритм усунення перекому приведений на рис. 2.7.

2.2.2. Попередня підготовка зображення

Попередня підготовка зображення проводиться для підвищення якості розпізнавання зображення. Оскільки текст має бути максимально виділеним на фоні всього іншого зображення, то для цього проводять бінаризацію зображення, яка полягає в виконанні наступних кроків:

1. Перетворити вхідне зображення у напівтонове (в градації сірого);

2. Провести бінаризацію зображення з накладанням порогу.

Одним із найбільш використовуваних методів для бінаризації зображень є метод Оцу – алгоритм для виконання порогової бінаризації напівтонових зображень, який базується на наявності в зображенні двох класів пікселів (корисних і фонових) і виконує завдання пошуку оптимального порогу, що розділятиме два класи таким чином, щоб внутрішньокласова дисперсія була мінімальною (було доведено, що мінімізація дисперсії всередині класів рівносильна максимізації дисперсії між класами, що дозволяє чітко бінаризувати зображення). Існує також покращена версія початкового методу для підтримки багаторівневих порогів, що називається мульти-Оцу метод.

Пошук порогу методом Оцу заснований на обчисленні суми внутрішньокласових дисперсій обох класів, що проводиться за формулою:

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

де ω_i – вагові коефіцієнти, що виражають імовірності класів, розділених порогом t;

σ_i^2 – дисперсії цих класів.

Рис. 2.4. Алгоритм усунення перекосу зображення

Точність розділення порогом напряму залежить від імовірностей класів – чим більш явно виділяються класи, тим краще спрацьовує метод Оцу (див. рис. 2.5.)

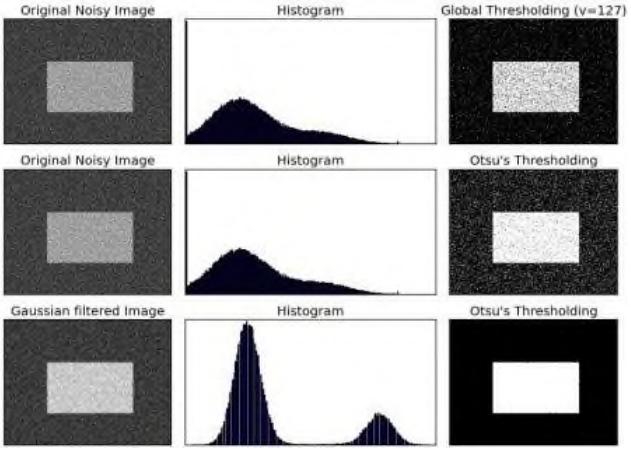

Рис. 2.5. Приклад бінаризації зображення з визначенням порогу методом Оцу

Перед передачею на вхід *OCR*-додатку зображення необхідно провести модифікацію зображення таким чином, щоб текст зміг бути чітко виділений. Для цього потрібно провести попередню сегментацію зображення, що включає в себе наступні кроки:

– Виділити матрицю згортки (*kernel*), що використовується як фільтр при розмитті, підвищенні різкості, виділенні границь і інших операціях при роботі з зображенням. Вона являє собою квадратну матрицю переважно невеликих розмірів, і використовується в процесі визначення нового значення кожного пікселя зображення. Для матриці згортки, яку також часто називають ядром, визначається опорна точка (зазвичай нею виступає центральний елемент);

– Провести операцію розширення. Ядро поелементно рухається по вхідному зображенню, накладаючись на нього і обчислює локальний максимум серед пікселів, «покритих» ядром і заміняє значення пікселя, що відповідає опорній точці, значенням знайденого максимуму. Тобто на вхідне бінаризоване зображення, що являє собою двовимірну матрицю, елементи якої дорівнюють 0 або 1, накладається фільтр у вигляді ядра, а саме – ті його елементи, що дорівнюють 1. Якщо у зоні, яку покривають одиничні елементи ядра, є хоча б один піксель, який дорівнює одиниці, то елемент вхідного зображення, що відповідає опорній точці, приймає значення 1. Таким чином, така операція максимізує світлі ділянки, розширюючи їх розмір. Приклад роботи операції розширення приведений на рис. 2.6.

– Виділити контури на зображенні. Контур – це крива, що об'єднує усі неперервні точки уздовж межі, що, мають однаковий колір. При розпізнаванні об'єктів використання контурів дає найкращі результати при роботі з двійковими зображеннями; саме тому в попередніх кроках було приведено вхідне зображення до чорно-білого вигляду, а також розширено ділянки із текстом – для того, щоб в результаті отримати координати контурів об'єктів із фотографії (в результаті розширення рядки тексту перетворюються в прямокутні блоки – див. рис. 2.7). Після цього вхідне зображення розбивається на масив виділених фрагментів.

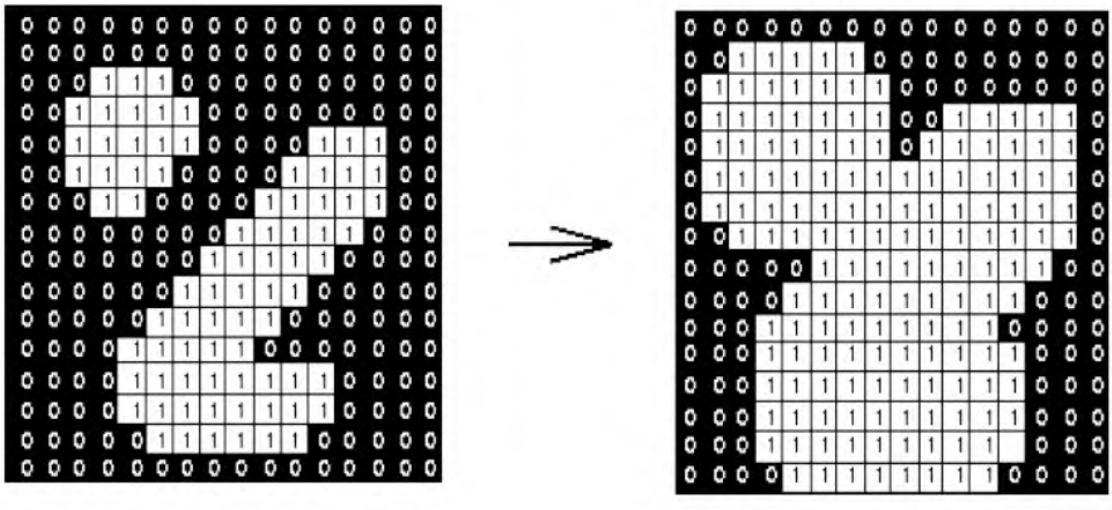

Рис. 2.6. Результат операції розширення двійкового зображення ядром 3*x*3

Рис. 2.7. Результат операції розширення на фотографії касового чеку

2.2.3. Співставлення товару та його ціни

На касових чеках міжрядковий інтервал однаковий і між товарами, і в них самих, і тому візуально ніяким чином не відображається, де закінчилась назва одного товару і розпочалась назва іншого. Через відсутність розділяючих знаків виділення контурів видає усі товари в рамках одного блоку (див. рис. 2.8), тому потрібно спеціальним чином розділити цей блок на частини, кожною з яких буде окремий товар.

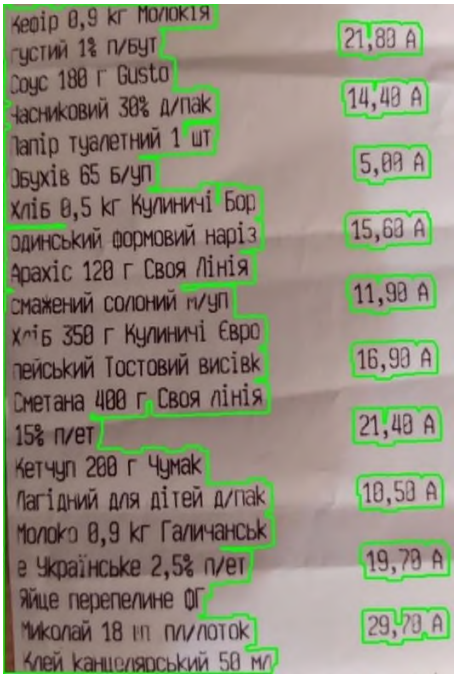

Рис. 2.8. Виділення контурів тексту на чеку

В даному випадку орієнтиром може послужити розташування блоків із ціною товарів – відношення висоти тексту у них до відстані між сусідніми блоками достатнє, щоб віднести їх до різних контурів, і кожен товар гарантовано має ціну, а отже і орієнтир.

Для того, щоб коректно розділити загальний блок товарів, потрібно для кожного блоку ціни (див. рис. 2.9) створити новий контур на основі наявного загального блоку, якщо його нижній край знаходиться нижче, ніж поточна ціна. В такому випадку після створення нового контуру потрібно відсікти його із загального блоку.

Після цього почергово виділені блоки (і товарів, і цін) передаються на вхід *OCR*, що дозволяє проводити розпізнавання лише тих областей, де було виділено текст, в свою чергу дає точніші результати та є менш ресурсоємним варіантом.

2.2.4. Розпізнавання тексту

На даному етапі обробляються та розпізнаються зображення, що було створені у попередньому кроці. *OCR*-додаток *Tesseract*, який було обрано для проведення розпізнавання, навіть при відсутності користувацьких налаштувань у стартовій конфігурації дозволяє провести дану операцію із мінімальними зусиллями – він

приймає на вхід зображення, з яким буде проводитись операція розпізнавання, і повертає текст, вилучений із вхідних даних.

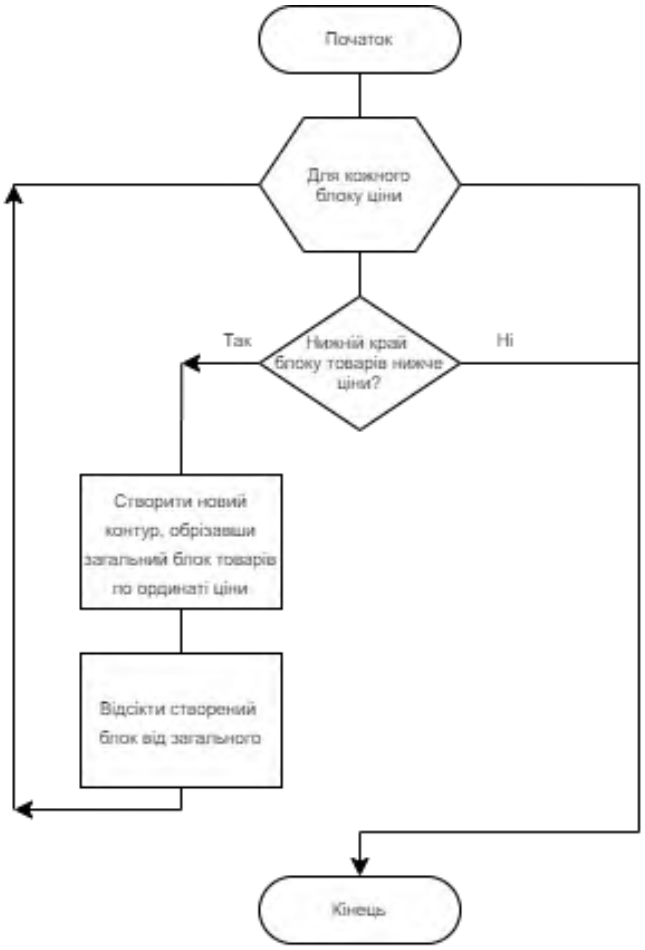

Рис. 2.9. Алгоритм розбиття на товари відносно блоків ціни

2.2.5. Класифікація товарів в рамках попередньо визначених категорій

Даний етап потребує реалізації алгоритмів класифікації, точніше, її конкретного виду – багатокласової класифікації.

В рамках предметної області вхідними даними для даної задачі виступають назви товарів у їх текстовому поданні, а результатом повинен бути клас (категорія), до якого даний товар належить. Отже, це задача, в якій є множина об'єктів, розділених деяким чином на класи. Задано кінцеву множину об'єктів, для яких відомо, до яких класів вони належать. Ця множина називається вибіркою (навчальною). Класова приналежність інших об'єктів невідома. Потрібно обрати алгоритм, здатний класифікувати довільний об'єкт з початкової множини.

Окрім цього, потрібно реалізувати перетворення текстової інформації в числову, оскільки алгоритми класифікації потребують подання даних у числовому форматі, і не в будь-якому іншому.

Для векторизації тексту використовують метод *TF-IDF* (англ. *TF – term frequency*, частота терміну, *IDF – inverse document frequency*, зворотня частота документу).

Багатокласова класифікація реалізується рядом методів, кожен з яких володіє певними перевагами та недоліками. Для того, щоб продемонструвати ефективність та доцільність використання в даній предметній області того чи іншого алгоритму, в розроблюваному модулі класифікація проводиться 4 поширеними методами:

– Логістична регресія;

– Випадковий ліс;

– Мультиноміальний баєсів класифікатор;

– Метод опорних векторів.

2.2.6. Обчислення сумарних вартостей для кожної встановленої категорії

На даному етапі на вхід поступає перелік встановлених категорій та цін для кожного товару. Елементи вхідного масиву перевіряються один за одним на предмет присутності у результуючому словнику, що зберігає пари «ключ-значення», де ключем є назва категорії, а значенням – сумарна вартість її товарів. Вибір такої структури даних для досягнення даної цілі дозволить забезпечити унікальність ключів, що в свою чергу не дозволить дублювати категорії і цілісно зберігати дані.

Цей крок є завершальним при обробці зображення, і результати, отримані на ньому, передаються далі – в потрібний модуль програми або напряму користувачу.

2.3 Формування датасету для категоризації

Алгоритми машинного навчання потребують даних, на яких вони зможуть натренувати модель та підібрати вагові коефіцієнти для того, щоб при подачі на їх вхід назви товару коректно спрогнозувати категорію, до якої він належить.

Відповідно до обраного при розгляді предметної області профілю закладів, що надають чеки (продуктові магазини та супермаркети), датасет потрібно заповнити товарами, що відносяться до найбільш поширених категорій. Було виділено ряд наступних категорій, до яких програмний продукт буде відносити товари:

– Бакалія;

– Гігієна і косметика;

– Молочні продукти;

– М'ясо та яйця;

– Овочі та фрукти;

– Побутові і непродовольчі товари;

– Риба і морепродукти;

– Хлібобулочні вироби.

Збір даних було вирішено проводити на основі асортименту мережі супермаркетів АТБ, оскільки в своїх чеках вона вказує назви товарів максимально наближеними до тих, які опубліковані на сайті мережі. Це якісно виділяє її, оскільки поширеним є підхід, коли назва товару у чеку значно скорочується, слова не розділяються окремо заради зменшення його розміру та відповідних витрат Наприклад, мережа «Сільпо» товар «Хліб Кулиничі Європейський тостовий висівковий» на чеках відображає як «ХлібКулиниВисів350г». Такий спосіб подання інформації не дозволить моделі побудувати коректну та доречну залежність між назвою товару та його категорією, оскільки перед безпосередньою класифікацією назви потрібно перевести її в числовий вигляд. Усі методи, які дозволяють провести подібне перетворення, базуються на використанні такої одиниці, як слово, та їх поєднаннях. В тому випадку, коли уся назва представлена одним-єдиним словом, результати векторизації не дозволять алгоритмам машинного навчання встановити відповідність між назвою товару та категорією.

Сформований датасет містить два поля – назву товару та його категорію. Для кожної із перелічених вище категорій було підібрано ряд товарів (див. рис. 2.10), які в сукупності її репрезентують.

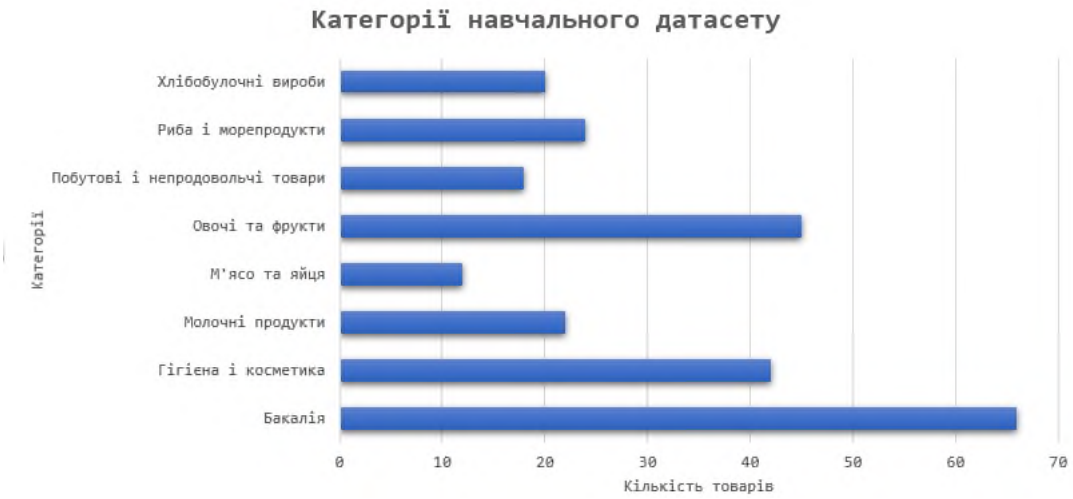

Рис. 2.10. Гістограма із кількістю представлених товарів для обраних категорій

Як видно із рис. 2.10, категорії представлені нерівномірно, що може вплинути на хибну оцінку точності прогнозування – наприклад, товари групи «Бакалія» будуть отримувати достатньо точні результати за рахунок того, що для моделі було достатньо навчальних даних, однак група «М’ясо та яйця» потенційно може бути хибно спрогнозована. За рахунок великої кількості точних результатів групи «Бакалія» оцінка точності може показати хороші результати, в незначній мірі враховуючи декілька помилок відносно «вузьких» категорій, які, однак, на фоні незначного кількісного представлення можуть свідчити про неготовність моделі правильно прогнозувати певну категорію товарів.

Для об’єктивного та повного подання інформації про успішність класифікації, цей аспект потрібно врахувати при виборі моделі оцінювання точності результатів прогнозування.

2.4 Векторизація назв товарів методом *TF-IDF*

TF-IDF (англ. *TF – term frequency*, частота терміну, *IDF – inverse document frequency*, зворотня частота документу) – це статистичний показник, що оцінює, наскільки слово є релевантним для даного документу в колекції документів. Це робиться шляхом множення двох показників: скільки разів слово з’являється в документі, і зворотна частота документа у цьому документі.

Цей метод має багато застосувань, найголовніше з яких – в автоматизованому аналізі тексту; окрім того, він корисний для визначення важливості слів в алгоритмах машинного навчання для обробки природних мов (*NLP – Natural Language Processing*).

TF-IDF (дослівно «частота теріну – зворотна частота документа») був винайдений для пошуку документів та виоучення інформації. Це працює за рахунок збільшення показника пропорційно кількості випадків, коли слово з'являється в документі, але компенсується загальною кількістю документів, що містять це слово. Отже, слова, поширені в кожному документі, такому, як поточний, мають низький рейтинг (показник), хоча вони можуть з'являтися багато разів – оскільки вони не відіграють значної ролі у цьому документі.

Однак, якщо, наприклад, слово «Збій» багато разів зустрічається в документі, але не часто зустрічається в інших, це, ймовірно, означає, що воно дуже значиме для даного документа. Наприклад, якщо проводиться аналіз опитування користувачів, на основі якого потрібно співставити відгук та його тематику, то слово «Збій», ймовірно, в кінцевому підсумку буде пов'язане із темою «Надійність», оскільки більшість відповідей, що містять це слово, стосуються саме цієї теми.

TF-IDF для слова в документі обчислюється множенням двох різних показників:

- Частота слова в документі (*TF*);
- Зворотня частота документу (*IDF*).

Частота слова в документі – це відношення кількості входження конкретного терміну (слова) до сумарного набору слів в досліджуваному тексті (документі). Цей показник відображає важливість (вагомість) слова в рамках певної статті, публікації тощо. Вираховується вона наступним чином:

$$tf(t,d) = \frac{n_t}{n},$$

де t – задане слово;

d – документ, в якому знаходиться слово;

n_t – кількість входжень слова t в документ d;

n – загальна кількість слів документа d.

Зворотна частота документу – це інверсія частоти, з якої певне слово фігурує в колекції текстів (документів). Завдяки даним показником можна знизити вагомість найбільш широко використовуваних слів (прийменників, загальних термінів і понять). Для кожного терміна в рамках певної бази текстів передбачається лише одне єдине значення *IDF*. Таким чином визначається, наскільки поширеним або рідкісним слово є у всьому наборі документів. Чим ближче цей показник до 0, тим більше поширене слово. Цю метрику можна обчислити, взявши логарифм від ділення загальної кількості документів на кількість документів, що містять слово:

$$idf(t, D) = \log_{10} \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|},$$

де $|D|$ – число документів у колекції;

$|\{d_i \in D \mid t \in d_i\}|$ – кількість документів з колекції D, в яких зустрічається слово t.

Отже, якщо слово дуже поширене і зустрічається у багатьох документах, то його показник *IDF* наближається до 0. В іншому випадку воно наблизиться до 1.

Множення цих двох чисел призводить до оцінки *TF-IDF* слова в документі. Чим вищий бал, тим актуальніше це слово у цьому конкретному документі.

Таким чином, оцінка *TF-IDF* для слова t у документі d із набору документів D обчислюється наступним чином:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

Вищий показник *TF-IDF* отримають слова з високою частотою потрапляння в рамках конкретного документу і з низькою частотою вживань в інших документах.

Машинне навчання при роботі природною мовою стикається з однією значною перешкодою – її алгоритми зазвичай мають справу з числами, а природна мова – це, звичайно, текст. Тому виникає потреба перетворити цей текст на числа – такий процес називають векторизацією тексту. Це фундаментальний крок у процесі машинного навчання для аналізу даних, і різні алгоритми векторизації суттєво впливають на кінцеві результати.

Після того, як слова перетворені в числа таким чином, щоб це могли прийняти алгоритми машинного навчання, оцінка *TF-IDF* може передаватися таким

алгоритмам, як наївний баєсів класифікатор та опорно-векторні машини, що значно покращує їх результати.

Це працює за рахунок того, що вектор слова представляє документ як перелік чисел, одне для кожного можливого слова корпусу (колекції документів). Векторизація документа – це взяття тексту та створення одного з цих векторів, номери яких певним чином представляють зміст тексту. *TF-IDF* дає можливість зв'язати кожне слово в документі з числом, що відображає наскільки кожне слово релевантне в цьому документі. Тоді документи з подібними, відповідними словами матимуть подібні вектори – саме це ми потрібно в алгоритмах машинного навчання.

Використання *TF-IDF* для визначення того, наскільки слово відповідає документу, корисно для цілого ряду задач, таких як:

– Пошук інформації. *TF-IDF* був винайдений для пошуку в документах і може використовуватися для отримання результатів, найбільш відповідних тому, що було задано у пошуковому запиті. Наприклад, засобами пошукової системи користувач намагається знайти інформацію про улюбленого актора; результати відображатимуться в порядку відповідності. Це означає, що найрелевантніші статті будуть вищі в рейтингу, оскільки *TF-IDF* дає прізвищу актора вищий бал.

– Вилучення ключових слів. *TF-IDF* також корисний для вилучення ключових слів із тексту – слова з найвищим балом документа є найбільш відповідними для цього документа, і тому їх можна вважати ключовими словами для нього.

У контексті заданої предметної області та підготованого датасету документом виступатиме назва товару, а корпусом документів – перелік усіх наявних товарів. Таким чином слова, що рідко використовуються в назвах товарів, будуть мати більшу вагу, а поширені слова – меншу.

2.5 Вибір моделі оцінки точності класифікації

Оскільки вхідні навчальні дані для моделей класифікації не є збалансованими, критично важливим є підбір правильного підходу оцінювання результатів

класифікації товарів. Загальна точність не є чутливою до повноти представлення кожного із класів, тому вона не є демонстративним показником.

Тому для оцінювання точності роботи моделі буде використовуватись сукупність наступних характеристик:

– Влучність (*precision*);

– Повнота, або ж чутливість (*recall*);

– *F1*-міра (*F1-score*);

– Кількість входжень класу в фактичному (істинному) наборі даних (*support*);

– Точність (*accuracy*).

Ці параметри використовують таке поняття, як матриця невідповідностей (*confusion matrix*), якою в задачах класифікації називаються матрицю (див. рис. 2.11), в якій для кожного класу приводиться кількість результатів прогнозування, віднесених моделлю до того чи іншого класу.

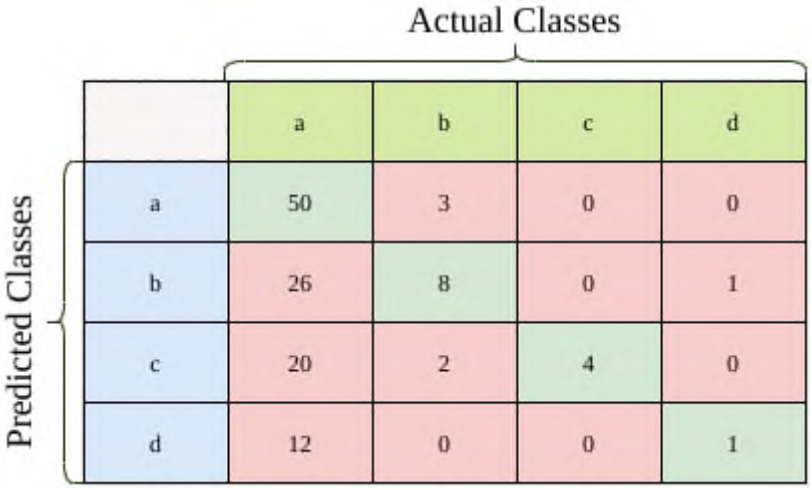

Рис. 2.11. Приклад матриці невідповідностей

Матриця невідповідностей заповнюється наступними показниками:

– Істинно позитивні випадки, ІП (*True Positive, TP*) – кількість правильних класифікацій для певного класу. Заповнюються по головній діагоналі;

– Хибно позитивні випадки, ХП (*False Positive, FP*) – кількість класифікацій об’єктів конкретного класу, коли моделлю об’єкт було віднесено до класу, до якого він насправді відноситься – об’єкт фактично належить до класу Б, однак модель класифікувала його як клас А. Іншими словами, кількісна величина, що показує,

скільки разів модель повідомила «хибну тривогу», розпізнавши певний клас на об’єктах, які до нього не належать ;

– Хибно негативні випадки, ХН (*False Negative, FN*) – кількість класифікацій, коли модель не змогла в об’єкті конкретного класу його розпізнати і віднесла до іншої категорії – так би мовити, «проспала» правильне його виділення.

На основі вищевказаних показників визначаються характеристики, до яких відносять влучність, чутливість, точність та *F*-міру.

Влучність – міра, яка відображає здатність класифікатора ідентифікувати тільки коректні об’єкти кожного класу. Визначається як відношення кількості коректно визначених позитивних об’єктів певного класу до загальної кількості класифікованих до певного класу об’єктів (як істинних, так і хибних):

$$precision = \frac{TP}{TP+FP}$$

Фактично у знаменнику записується кількість розпізнавань, що визначається як сума всіх клітинок рядка, який відповідає класифікованим до заданого класу об’єктам, у матриці невідповідностей.

Повнота – міра, яка відображає відношення кількості коректно визначених об’єктів певного класу до кількості його фактичних об’єктів:

$$recall = \frac{TP}{TP+FN}$$

У знаменнику записується кількість реальних об’єктів заданого класу, і визначається вона як сума клітинок стовпця у матриці невідповідностей, що відображає відповідний клас.

Влучність і повнота вираховуються подібним чином, однак несуть в собі різне значення. Простіше кажучи, суть влучність можна умовно виразити запитанням «Модель спрогнозувала 100 об’єктів класу А. Скільки із них фактично є об’єктами класу А?», в той час як повнота дає відповідь на питання «В наборі даних фактично істинними є 100 об’єктів класу А. Скільки із них модель змогла правильно класифікувати до цього класу?». Таким чином, ці дві міри виражають коректність роботи моделі із різних відношень.

Найбільш інтуїтивно зрозумілою мірою коректності роботи класифікатора є точність – відношення кількості істинних позитивних передбачень до загальної кількості передбачень:

$$accuracy = \frac{TP}{N_{total}}$$

Однак такий підхід до визначення точності вкрай нерепрезентативний у тому випадку, коли дані будуть незбалансовами. Якщо клас А буде домінуючим у наборі даних і отримуватиме точні передбачення в той час, коли клас Б матиме низькі показники, то обчислення точності не врахує подібну незбалансованість даних, і надасть достатньо високі показники, які не будуть релевантними для такого сценарію. Окрім того, самі по собі влучність і повнота не дають в повній мірі розуміння, наскільки точною є модель. Тому в статистичному аналізі використовують їх поєднання – *F1*-міру, яка поєднює влучність та та повноту, обчислюючись як їхнє середнє гармонійне:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Як було згадано вище, незбалансованість даних може впливати на результати обчислень статистичних мір, які будуть надавати інформацію, відмінну від реальної. Традиційно влучність, повнота та *F1*-міра обчислювались для кожного окремого класу; для того ж, щоб обчислити ці показники для моделі загалом, урахувавши результати класифікації усіх класів, виділяють три наступних підходи:

– Мікро-середнє (*micro average*) – обчислити відношення суми всіх ІП класифікованих об’єктів до суми ІП та ХП (у випадку влучності) або ХН (у випадку обчислення повноти);

– Макро-середнє (*macro average*) – обчислити потрібний показник для кожного класу, після чого знайти їх середнє арифметичне – без урахування пропорцій представленя класів у наборі даних;

– Зважене середнє (*weighted average*) – обчислити потрібний показник для кожного класу, після чого знайти їх зважену суму (суму добутків показника та величини, яка відображає пропорційне представлення у наборі даних).

2.6 Висновки до розділу

У даному розділі було розглянуто предметну область та встановлено її обмеження. Визначено підхід до відсіювання надлишкової інформації з чеку.

Було сформовано послідовність роботи модуля та визначено порядок обробки вхідного зображення.

Розібрано наступні методи для підготовки зображення перед розпізнавання на ньому тексту:

– алгоритм Кенні для виявлення границь на фотографії, і перетворення Хафа для знаходження на ній ліній – ці методи дозволяють усунути проблему перекосу зображення;

– метод Оцу для бінаризації зображення, яка необхідна для чіткого виділення тексту на його фоні.

Проведено аналіз особливостей їх застосування та математичної основи.

Для коректного пов'язування товару та ціни було встановлено принцип їх співставлення.

Обрано метод векторизації тексту в числову інтерпретацію, яка може бути опрацьована методами класифікації. Для їх тренування було підготовано набір навчальних даних, який містить перелік товарів основних категорій. Підібрано модель оцінки точності класифікації товарів за категоріями задля урахування незбалансованості даних.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЮ РОЗПІЗНАВАННЯ

3.1 Мова програмування *Python*

Python – це інтерпретована, об'єктно-орієнтована, Тюринг-повна мова програмування високого рівня, призначена для вирішення найширшого кола задач. З її допомогою можна обробляти числову і текстову інформацію, створювати зображення, працювати з базами даних, розробляти веб-сайти і додатки з графічним інтерфейсом. *Python* – це багатоплатформенна мова, що дозволяє створювати програми, які будуть працювати у цілому ряді операційних систем.

Деякі помітні особливості *Python*:

– Використовує елегантний синтаксис, що полегшує читання створюваних програм;

– Проста у використанні мова, яка спрощує роботу вашої програми. Це робить Python ідеальним для розробки прототипів та інших спеціальних завдань програмування без шкоди для надійності експлуатації;

– Поставляється з великою стандартною бібліотекою, яка підтримує багато загальних завдань програмування, таких як підключення до веб-серверів, пошук тексту за допомогою регулярних виразів, читання та зміна файлів;

– Інтерактивний режим *Python* дозволяє легко тестувати короткі фрагменти коду. Існує також комплексне середовище розробки під назвою *IDLE*;

– Працює де завгодно, включаючи *MacOS* , *Windows*, *Linux* та *Unix*, неофіційні збірки також доступні для *Android* та *iOS*;

Python є потужним інструментом за рахунок наступних пунктів:

– Доступні різноманітні основні типи даних: числа (з плаваючою точкою, складні та довгі цілі числа необмеженої довжини), рядки (як *ASCII*, так і *Unicode*), списки та словники.

– *Python* підтримує об'єктно-орієнтоване програмування з класами та множинним наслідуванням;

– Код можна згрупувати в модулі та пакети;

– Мова підтримує створення та перехоплення винятків, що призводить до більш організованої обробки помилок;

– Типи даних підбирається динамічно. Змішування несумісних типів (наприклад, спроба додати рядок і число) призводить до виникнення винятку, тому помилки виявляються раніше;

– Автоматичне управління пам’яттю *Python* позбавляє розробника від необхідності розподіляти та звільняти пам’ять у коді вручну.

Програма на мові *Python* являє собою звичайний текстовий файл з розширенням *.py* (консольна програма) або *.pyw* (програма з графічним інтерфейсом). Всі інструкції з цього файлу виконуються інтерпретатором поелементно (рядок за рядком). Для прискорення роботи при початковому імпорті модуля створюється проміжний байт-код, який зберігається в однойменному файлі з розширенням *.pyc*. При наступних запусках, якщо модуль не був змінений, виконується той самий байт-код. Для виконання низькорівневих операцій і завдань, що вимагають високої швидкості роботи, цей модуль може бути написаний мовою програмування *C* або *C++*, після чого його потрібно скомпілювати, і підключити до основної програми.

Python відноситься до категорії мов об'єктно-орієнтованих. Це означає, що практично всі дані в ньому є об'єктами, навіть значення, які відносяться до елементарних типів, на зразок чисел і рядків, а також самі типи даних. В змінній завжди зберігається тільки посилання на об'єкт, а не сам об'єкт. Наприклад, можна створити функцію, зберегти посилання на неї в змінну, а потім викликати функцію через цю змінну.

Така обставина робить мову *Python* чудовим інструментом для створення програм, використовують функції зворотного виклику, наприклад, при розробці графічного інтерфейсу. Той факт, що мова є об'єктно-орієнтованою, аж ніяк не означає, що і об'єктно-орієнтований стиль програмування (ООП) є при його використанні обов'язковим. Мовою *Python* можна писати програми як в стилі ООП, так і в процедурному стилі – як того вимагає конкретна ситуація або як вважає за краще сам розробник.

Python не допускає двоякого написання коду. Так, для прикладу мові *Perl* притаманні залежність від контексту і множинність синтаксису, і часто два програмісти, що пишуть код на *Perl*, просто не розуміють напрацювання іншого спеціаліста. В *Python* ж код можна написати тільки одним способом. У ньому відсутні зайві конструкції. Усі розробники, які використовують дану мову програмування, повинні дотримуватися стандарту *PEP-8*, що довзоляє *Python* бути однією із найбільш читабельних мов програмування.

Синтаксис мови *Python* викликає ряд нарікань у програмістів, знайомих з іншим мовами програмування. На перший погляд може здатися, що відсутність обмежувальних символів (фігурних дужок, конструкцій *begin ... end*, розділення рядків за допомогою крапки з комою) для виділення блоків і обов'язкова вставка порожніх рядків перед командами можуть призводити до помилок. Однак це тільки перше і неправильне враження.

Відповідно до стандарту, для виділення блоків необхідно використовувати чотири пробіли. Якщо кількість пробілів всередині блоку буде різна, то інтерпретатор виведе повідомлення про фатальну помилку, і програма буде зупинена. Таким чином, мова *Python* привчає програмістів писати чистий і зрозумілий код.

Оскільки програма на мові *Python* являє собою звичайний текстовий файл, його можна редагувати за допомогою будь-якого текстового редактора. Однак, розробники ПЗ зазвичай все ж надають перевагу спеціальним середовищам розробки, що не тільки відповідним чином виділяють код, але також надають різні підказки і дозволяють виконувати відладку програми.

Python використовується у таких сферах:

– Web-розробка. На *Python* можна реалізувати весь backend інтернет-ресурсу, який буде виконуватися на сервері. Робиться це за допомогою спеціальних фреймворків (Django і Flask), написаних на цій мові. З їх допомогою спрощується процес обробки адрес, звернення до баз даний та створення HTML, відображаються на призначених для користувача сторінках;

– Графічний інтерфейс. Якщо говорити про візуальну складову в сфері *IT*, то і тут *Python* може показати себе як цілком ефективний інструмент, який вирішує

широкий спектр завдань. Створюючи сучасні графічні інтерфейси на Python, можна легко підлаштуватися під стилістику операційної системи, в середовищі якої створюється додаток. Спеціально для цих цілей були створені додаткові бібліотеки для побудови інтерфейсу - *PythonCard* і *Dabo*, що полегшують процес роботи;

– Бази даних. Розробники сучасної версії *Python* створили максимально простий і зрозумілий доступ практично до будь-яких баз даних. Так, на сьогоднішній день, в робочому середовищі мови знаходиться програмний інтерфейс, який дозволяє користуватися базами прямо зі скрипта за допомогою запитів *SQL*. Також, код, написаний на *Python*, може з мінімальними доробками використовуватися для баз даних *MySQL* і *Oracle*;

– Системне програмування. Інтерфейси мови дозволяють управляти службами операційних систем Windows, Linux і ін. *Python* також застосовується для створення програмного забезпечення, що використовуються системними адміністраторами. Таким чином, *Python* прискорює пошук і відкриття файлів, запуск програм, полегшує обчислення тощо;

– Складні обчислювальні процеси. *Python* в даній сфері може позмагатись в своїх можливостях з C ++. Спеціальне розширення *NumPy*, написане для математичних розрахунків, прекрасно працює з масивами, інтерфейсами рівнянь і іншими даними. Як тільки розширення встановлюється на комп'ютер, *Python* проходить інтеграцію з бібліотеками формул. Однак *NumPy* призначений не тільки для обчислень. Крім свого основного завдання, з його допомогою можна створювати анімовані елементи і прорисовувати об'єкти в середовищі *3D*, застосовуючи при цьому паралельні обчислення. Окрім того, популярне доповнення *ScientificPython* може похвалитися власними бібліотеками, які створені для обчислювальних процесів в сфері науки. Крім розрахунків, *Python* дозволяє візуалізувати отримані дані, що досить зручно;

– Машинне навчання. Крім основного інструментарію, у *Python* є додаткові бібліотеки і фреймворки, що дозволяють працювати в області машинного навчання. Особливою популярністю користуються *scikit-learn* і *TensorFlow*. *Scikit-learn* відрізняється тим, що в нього вже вбудовані найпоширеніші алгоритми навчання.

TensorFlow, в свою чергу - це низькорівнева бібліотека, яка відкриває можливості для створення користувачем своїх власних алгоритмів. Процеси машинного навчання, засновані на мові програмування Python, допомагають реалізовувати системи розпізнавання лиця і голосу, створювати нейронні мережі, проводити глибоке навчання тощо;

– Автоматизація процесів. Сьогодні одним з найбільш затребуваних способів використання мови *Python* є створення невеликих скриптів, що автоматизують деякі робочі процеси. Наприклад, можна написати цілком простий код, який буде «самостійно» працювати з листами на електронну пошту. *Python* чудово підходиться для цього з двох причин? По-перше, він відрізняється цілком простим синтаксисом, який дозволяє з легкістю розробляти сценарії. А по-друге, сам код не проходить компіляцію перед запуском, що помітно полегшує процес налагодження.

3.2 Реалізація побудованого алгоритму

Першим кроком при роботі модуля є усунення перекосу зображення, який реалізується використанням алгоритму Кенні та перетворенням Хафа. Таким чином, здійснюються наступні послідовності команд:

```
img_before = cv2.imread('rotate_me_flip.jpg')
img_gray = cv2.cvtColor(img_before, cv2.COLOR_BGR2GRAY)
img_edges = cv2.Canny(img_gray, 100, 150, apertureSize=3)
```

В змінну *img_before* завантажується вказане зображення, яке після цього передається методу *cv2.cvtColor()* з вказівкою, що дане зображення має бути переведено у градації сірого. Виклик наступного методу (*cv2.Canny()*) за допомогою алгоритма Кенні визначає на вхідному зображенні границі, і повертає результат в *img_edges*. Параметри 100 і 150 задають нижній та верхній поріг при фільтруванні, що дозволяє відсіяти небажані лінії з зображення.

Наступний крок (застосування перетворення Хафа) проводиться викликом наступного методу:

lines = cv2.HoughLinesP(img_edges3, 1, math.pi / 180.0, 100, minLineLength=100, maxLineGap=2)

Параметри даного методу відповідно задають:

– Вхідне зображення;

– Параметр ρ;

– Параметр θ;

– Поріг – порогове значення для акумулятора;

– Мінімальну довжину лінії;

– Максимум прогалин, які виникають між точками вздовж лінії, що їх з'єднує.

Надалі потрібно вирахувати нахил знайдених ліній, щоб провести поворот зображення:

```
angles = []
for [[x1, y1, x2, y2]] in lines:
    cv2.line(img_before, (x1, y1), (x2, y2), (255, 0, 0), 3)
    angle = math.degrees(math.atan2(y2 - y1, x2 - x1))
    angles.append(angle)
```

Таким чином у список angles вноситься значення змінної *angle*, яке визначається через тангенс різниці координат кінцевих точок лінії.

Після цього потрібно вирахувати медіанне значення нахилу, в повернути зображення на відповідний кут:

```
median_angle = statistics.median(angles)
img_rotated = ndimage.rotate(img_before, median_angle - np.sign(median_angle)*90)
```

Таким чином вхідне зображення буде вирівняно по вертикальній осі відносно чеку. На рис. 3.1 приведений зразок такого перетворення (в ілюстративних цілях на вихідному зображенні синім кольором наведені лінії, виявлені перетворення Хафа).

Рис. 3.1. Вхідне зображення та результат усунення перекосу

Далі обернене зображення попередньо обробляється перед подачею на розпізнавання. Відбувається це наступним чином:

gray = cv2.cvtColor(img_rotated, cv2.COLOR_BGR2GRAY)

_, thresh1 = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)

Отримане зображення (таке ж саме, як і вхідне, але вирівняне) зводиться до градацій сірого, після чого на нього накладається бінаризація методом Оцу.

Наступний крок – укрупнити на бінаризованому зображенні текст, щоб його легше було виділити.

rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (28,28))

dilation = cv2.dilate(thresh1, rect_kernel, iterations=1)

contours, hierarchy = cv2.findContours(dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

Змінній *rect_kernel* присвоюється матриця згортки розміром 28*x*28 шляхом виклику *cv2.getStructuringElement()*. Після цього відбувається операція розширення, яка розширить ділянки тексту, накладаючи на нього матрицю згортки та обчислюючи нові значення пікселів зображення. По завершенні розширення запускається процедура пошуку контурів на обробленому зображенні через метод *cv2.findContours()*, який присвоює змінним *contours, hierarchy* об’єкти контурів та їх ієрархію відповідно.

Після цього наступає етап розпізнавання як такого – кожен розпізнаний контур почергово передається на вхід *OCR*.

pytesseract.pytesseract.tesseract_cmd = r'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'

for cnt in contours:

x, y, w, h = cv2.boundingRect(cnt)

cropped = thresh1 [y:y + h, x:x + w]

text = pytesseract.image_to_string(cropped, lang="ukr")

Для запуску *Tesseract* необхідно вказати йому розташування на комп’ютері виконуваного *.exe*-файлу, якийі організовуватиме процес розпізнавання.

Після цього кожен із виділених контурів приймає участь у наступній послідовності:

1. Передати координати верхньої лівої точки, а також ширину та довжину контура відповідним змінним шляхом виклику методу *cv2.boundingRect()*, який підбирає найменший зовнішній прямокутник, яким можна покрити виділений контур;

2. Створити нове зображення, обрізавши вхідне зображення до розміру контура;

3. Вичитати текст із контура у відповідну змінну викликом методу *pytesseract.image_to_string()*, вказавши параметрами вхідне зображення та мову тексту, який буде розпізнаватись.

В результаті у змінній *text* буде збережено текст, який вдалось розпізнати із зображення.

Класифікація ж відбувається наступним чином. Модель потрібно спершу натренувати на навчальному датасеті, щоб вона змогла підібрати коректні вагові коефіцієнти. Окрім того, також потрібно перевести текстові дані із датасету в числовий формат.

У датасеті містяться два значення – товар (*Item*) і категорія (*Category*). Остання задається у текстовому вигляді, однак для моделей класифікації її потрібно перевести в число, яке буде відповідати певній категорії.

df = pd.read_csv(r'C:\\Users\\user\\Desktop\\project\\data_upd.csv', delimiter=';')

df['Category_ID'] = df['Category'].factorize()[0]

Після того, як команди будуть виконані, у датафреймі *df* буде додана ще одне значення – *Category_ID*, яке відповідатиме порядковому номеру категорії.

Тепер можжна переходити до векторизації назв товарів.

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', ngram_range=(1, 2))

features = tfidf.fit_transform(df.Item).toarray()

labels = df.Category_ID

В змінній *tfidf* зберігається векторизатор, що буде обчислювати числові представлення тексту. Викликом методу *tfidf.fit_transform()* назви товарів буде перетворено в числовий еквівалент згідно методу *TF-IDF*. Окрім цього, у тому ж порядку, в якому представлені товари, в змінну *labels* передаються номери відповідних їм категорій.

Після того, як було перетворено назви товарів в числове представлення, вилучено відповідні ярлики (класи), постає питання реалізації навчання моделей машинного навчання. В даному аспекті потрібно врахувати вкрай важливий для точної класифікаціх момент.

Навчання та встановлення параметрів функції прогнозування та тестування їх на одних і тих самих даних є методологічною помилкою: модель, яка б просто повторювала мітки зразків, які вона щойно бачила, мала б ідеальну оцінку, але не змогла б передбачити щось точне на даних, з якими вона не працювала раніше. Така ситуація називається перенавчанням (*overfitting*). Щоб уникнути цього, загальноприйнятою практикою при проведенні експерименту машинного навчання з учителем є надання частини доступних даних як тестового набору в двох частинах – вхідні дані (X_{test}) та значення фактичної принадлежності до певного класу (y_{test}).

У бібліотеці *scikit-learn* випадковий розподіл на навчальні та тестові набори можна швидко обчислити за допомогою допоміжної функції *train_test_split()* за допомогою вказання параметра *test_size* в межах від 0 до 1.

При обчисленні різних параметрів моделі, які також іменуються гіперпараметрами, для її оцінювачів (часто їх необхідно встановити вручну), все ще існує ризик перенавчання тестового набору, оскільки ці параметри можна відрегулювати, доки оцінювач не працюватиме оптимально. Таким чином, інформація про тестовий набір може «просочитися» в модель, і тоді показники оцінки більше не звітуватимуть про загальну точність результатів. Для вирішення цієї проблеми ще одна частина набору даних може бути виділена як так званий «набір перевірки» (*validation set*): навчання триває на навчальному наборі, після чого проводиться оцінка на наборі перевірки, і коли експеримент набуває достатньо успішних результатів, остаточне оцінювання проводиться на тестовому наборі.

Однак, розділивши наявні дані на три набори, різко зменшується кількість зразків, які можна використовувати для вивчення моделі, і результати можуть залежати від конкретного випадкового вибору для пари наборів (тренувального, перевірочного).

Рішенням цієї проблеми є процедура, яка називається перехресною валідацією (*cross-validation*, скорочено *CV*). Для остаточного оцінювання все ще слід виділяти тестовий набір, але при проведенні перехресної валідації набір перевірки більше не потрібен. У базовому підході, який називається *k*-кратною *CV*, навчальний набір розбивається на *k* менших наборів (є ряд модифікованих підходів, але, як правило,

загалом дотримуються тих самих принципів). Для кожної з *k* підгруп (*folds*) виконується наступна процедура (рис. 3.2):

– Модель тренується з використанням *k*-1 підгруп як навчальних даних;

– Отримана модель перевіряється на решті даних (тобто вона використовується як тестовий набір для обчислення показника ефективності, такого як точність, чутливість або будь-який інший обраний показник).

Показник ефективності, визначений за допомогою *k*-кратної перехресної валідації, в такому випадку є середнім значенням, що обчислюється у циклі. Цей підхід може бути витратним в контексті кількості обчислювальних операцій, але не витрачає занадто багато даних (як це має місце при фіксації довільного набору перевірки), що є основною перевагою в таких задачах, як зворотний висновок, коли кількість вибірок дуже мала.

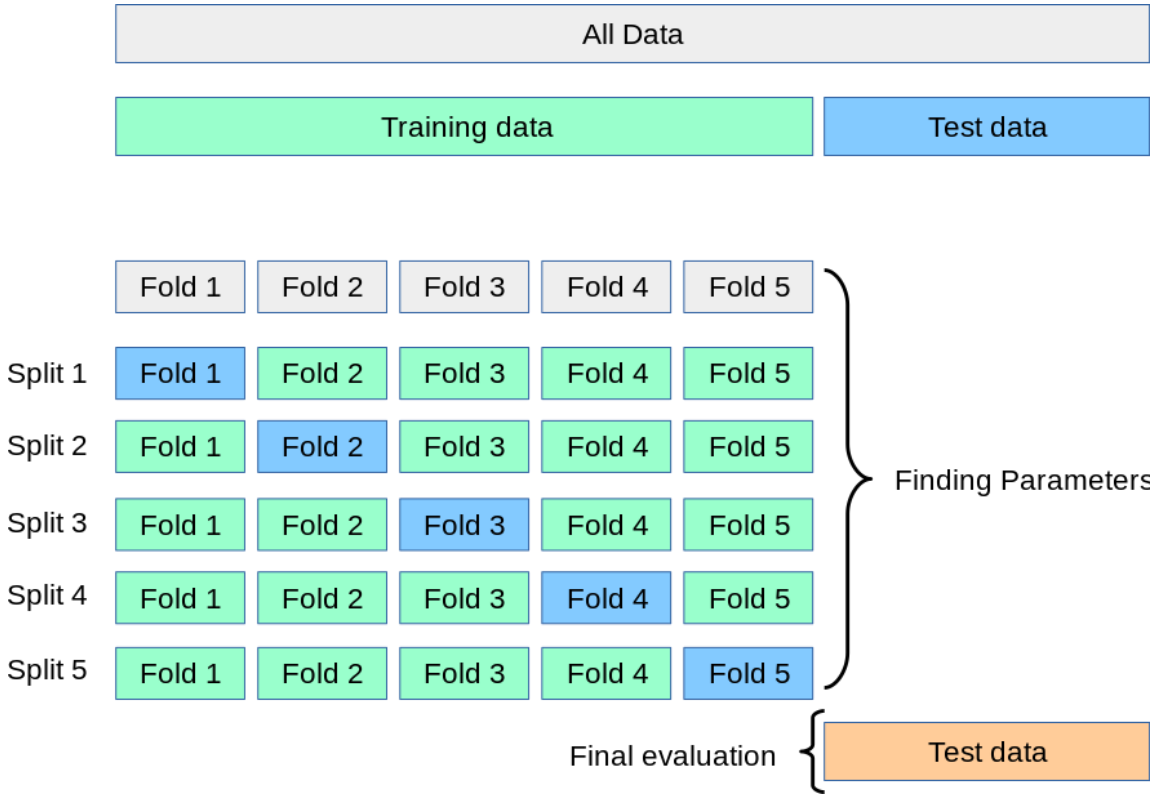

Рис. 3.2. Ілюстрація методу *k*-кратної перехресної валідації

Таким чином, використання методу перехресної валідації дозволяє уникнути проблеми перенавчання, та адаптує модель до роботи із даними, з якими вона раніше не працювала.

Традиційно кількість груп, на які розбивається навчальний датасет, дрівнює п’яти. Саме таке значення і було використано у розроблюваному модулі.

Згідно із проведеним проектуванням, класифікація товарів відбувається 4 методами. Програмно їх моделі оголошуються наступним чином:

```
models = [
    RandomForestClassifier(n_estimators=200, max_depth=3),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(),
]
```

Для того, щоб з’ясувати коректність роботи кожної з моделей, потрібно оцінити, наскільки точні результати були отримані в результаті роботи із підготованим датасетом. Кожна з моделей повинна отримати свою оцінку при використанні методу перехресної валідації.

```
CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))
entries = []
for model in models:
  model_name = model.__class__.__name__
  accuracies = cross_val_score(model, features, labels, scoring='accuracy', cv=CV)
  for kfold_id, accuracy in enumerate(accuracies):
    entries.append((model_name, kfold_id, accuracy))
cv_df = pd.DataFrame(entries, columns=['model_name', 'kfold_id', 'accuracy'])
```

Відносно обраного значення поділу навчального датасету у змінній *CV* було підготовано датафрейм (змінна *cv_df*), що міститиме оцінки точності класифікації кожним методом на кожній ітерації перехресної валідації. Під час поелементного проходження по масиву моделей обчислюється точність класифікації товарів поточною моделлю на кожній підгрупі перехресної валідації, яка в результаті накопичується в оголошеному датафреймі з указанням назви моделі, номеру підгрупи та самої точності.

Для наочного представлення продемонстрованих моделями класифікації результатів було обрано їх подання за допомогою діаграми розмаху – засобу візуалізації в описовій статистиці груп числових даних через їх квантилі. Діаграма розмаху дає можливість візуально представити наступні дані:

– медіана (*Q2*) – середнє значення набору даних, яке порівну розділяє його на рівну кількість більших та менших елементів;

– перший квартиль (*Q1*) – середнє число між найменшим числом (не "мінімальним") та медіаною набору даних;

– третій квартиль (*Q3*) – середнє значення між медіаною та найвищим значенням (не "максимальним") набору даних;

– міжквантильний діапазон (*IQR*) – діапазон від першого до третього квартиля;

– вуса – задають краї статистично значимої вибірки;

– викиди – результат вимірювання, що виділяється із загальної статистичної вибірки.

```
sns.boxplot(x='model_name', y='accuracy', data=cv_df)
sns.stripplot(x='model_name', y='accuracy', data=cv_df, size=12, jitter=True, edgecolor="gray", linewidth=2)
cv_df.groupby('model_name').accuracy.mean()
```

Діаграма розмаху створюється викликом методу *sns.boxplot()*, в який передаються назви змінних, що мають бути враховані, а також джерело даних (в даному випадку – заповнений показниками точності моделей датафрейм). Методом *sns.stripplot()* відбувається налаштування його візуального подання, а також побудова графіка, де одна із змінних є категоріальною (визначною).

Ще одним способом продемонструвати результати роботи моделі класифікації є відображення матриці невідповідностей.

```
conf_mat = confusion_matrix(y_test, y_pred)
_, _ = plt.subplots(figsize=(10,10))
sns.heatmap(conf_mat, annot=True, fmt='d',
            xticklabels=category_id_df.Category.values,
            yticklabels=category_id_df.Category.values)
```

plt.ylabel('Фактично')

plt.xlabel('Передбачено')

plt.show()

За допомогою методу *confusion_matrix()* формується матриця невідповідностей як така, яка порівнює фактичні і спрогнозовані значення. Метод *plt.subplots()* встановлює розмір клітинки, в якій буде відображатись інформація про кількість класифікацій. Після цього сформована матриця невідповідностей передається в метод *sns.heatmap(),* в якому також задається формат відображення даних, наявність текстових анотацій, співставлення значень осей). Після того, як задано підписи осей, матриця готова для виведення на екран викликом методу *plt.show()*.

3.3 Аналіз отриманих результатів

На етапі попереднього навчання моделікласифікація товару проводилась 4 методами:

– Випадковий ліс;

– Метод опорних векторів;

– Мультиноміальний баєсів класифікатор;

– Логістична регресія.

Кожен із заданих методів застосовувався для класифікації товарів, і, відповідно, кожен із них отримав свої показники точності (рис. 3.3).

З діаграми розмаху видно, що метод опорних векторів (*Linear Support Vector Classifier, LinearSVC*) надає найбільшу точність класифікації – її медіанне значення складає 83%.

З огляду на це, саме метод опорних векторів буде використано надалі.

Як наслідок, було обрано саме метод опорних векторів як основну модель класифікації товарів в рамках розроблюваного модулю. В результаті для неї було отримано характеристики, зображені на рис. 3.4-3.5.

З наведених статистичних показників та матриці невідповідностей можемо побачити, що найбільше помилок було спричинено при роботі із категорією

«Бакалія», якої була представлена найширше у датасеті – її однаково хибно розпізнавали або ж навпаки – не розпізнавали, що викликало низький *F1*-показник.

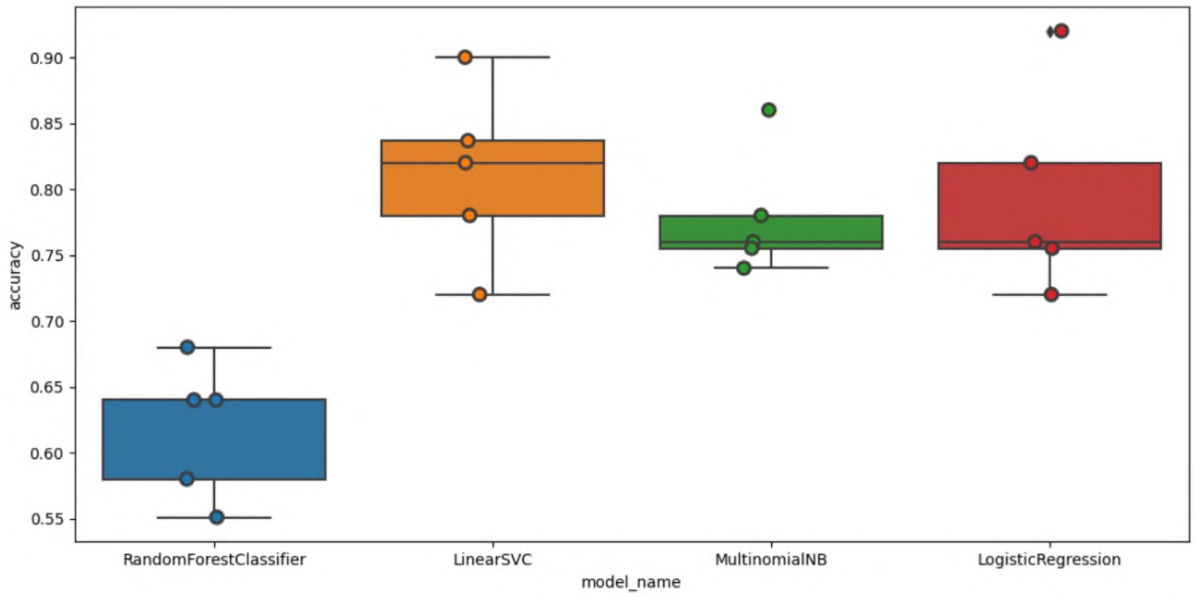

Рис. 3.3. Діаграма розмаху результатів класифікації

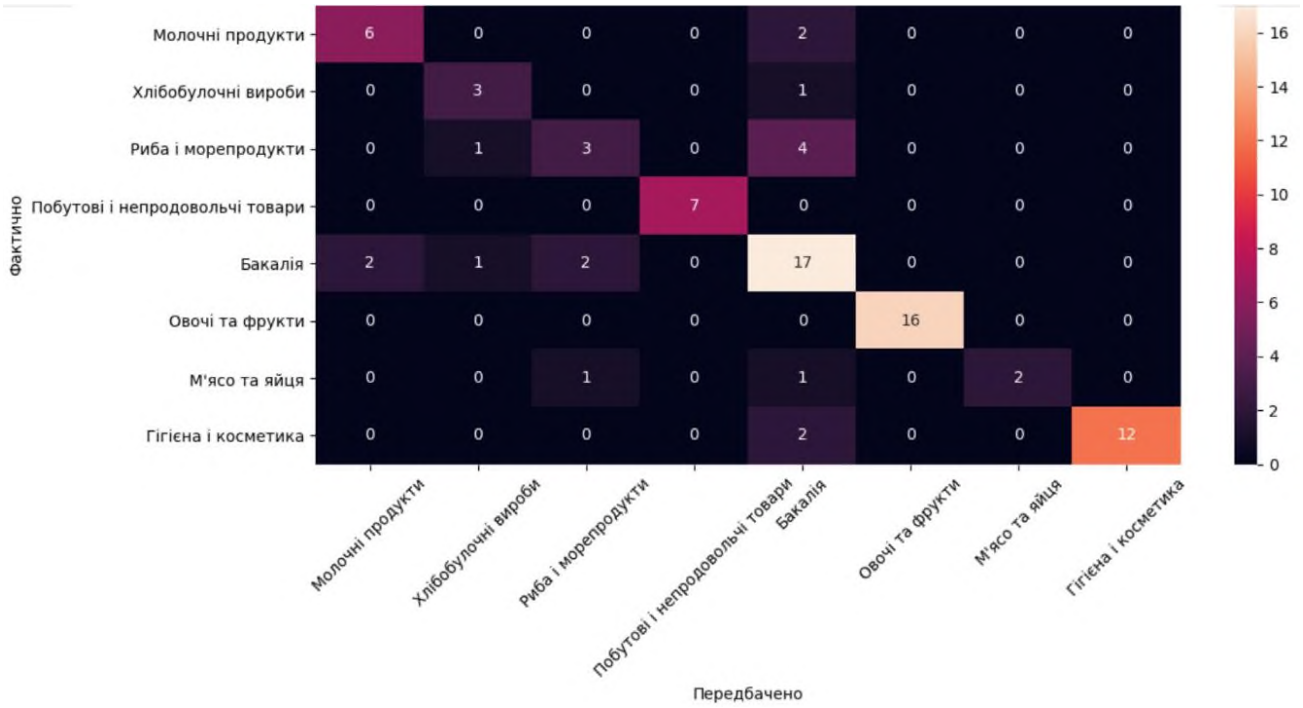

Рис. 3.4. Матриця невідповідностей класифікації методом опорних векторів

Симетрично подібні проблеми спостерігаються і при розпізнаванні категорії «М'ясо та яйця», яка має найменшу кількість товарів у датасеті.

Це свідчить про те, що хоч загальна точність і дорівнює 80%, набір даних все ж повинен бути збалансованим.

	precision	recall	f1-score	support
Молочні продукти	0.75	0.75	0.75	8
Хлібобулочні вироби	0.60	0.75	0.67	4
Риба і морепродукти	0.50	0.38	0.43	8
Побутові і непродовольчі товари	1.00	1.00	1.00	7
Бакалія	0.63	0.77	0.69	22
Овочі та фрукти	1.00	1.00	1.00	16
М'ясо та яйця	1.00	0.50	0.67	4
Гігієна і косметика	1.00	0.86	0.92	14
accuracy			0.80	83
macro avg	0.81	0.75	0.77	83
weighted avg	0.81	0.80	0.79	83

Рис. 3.5. Статистичні показники класифікації методом опорних векторів

Приклад роботи кінцевого варіанту модуля можна представити за допомогою обробки тестового зображення (рис. 3.6). На ньому вказані 6 товарів, які належать до наступних категорій:

– Гігієна і косметика (1 товар);

– Риба і морепродукти (1 товар);

– Бакалія (1 товар);

– Хлібобулочні вироби (1 товар);

– Молочні продукти (2 товари).

Модуль повинен обробити вхідне зображення, підготувавши його до обробки, виділити текст, відсіяти надлишкові поля, залишивши лише пари значень «товар-ціна», віднести товар до заданої категорії та обчислити сумарну вартість кожної із знайдених категорій.

Рис. 3.6. Зображення чеку для перевірки роботи модуля

В результаті виконання даних операцій було отримано вихідну інформацію, представлену на рис. 3.7.

```
Встановлені категорії:
 Гігієна і косметика | 22.10
 Риба і морепродукти | 19.90
             Бакалія | 12.08
 Хлібобулочні вироби | 13.98
    Молочні продукти | 22.98
             Бакалія | 19.30


Загальні витрати:
 Гігієна і косметика | 22.10
 Риба і морепродукти | 19.90
             Бакалія | 31.38
 Хлібобулочні вироби | 13.98
    Молочні продукти | 22.98
```

Рис. 3.7. Результат розпізнавання чеку

Модуль коректно розпізнав категорії 5 із 6 товарів, помилково класифікуваши товар категорії «Молочні продукти» як «Бакалію». Були, однак, допущені також

помилки розпізнавання цін товарів – при чому масштаб похибки варіюється від декількох копійок до гривень, як результат – істинна сума чеку складає 115,90 гривень, а сума розпізнаних цін складає 110,34 гривень. Такий недолік відбувся через не точне розпізнавання символів OCR-додатком.

Для забезпечення коректнішої роботи можуть бути виконані наступні дії:

– Розширити навчальний датасет для представлення у ньому більшої кількості товарів та категорій;

– Провести збалансування навчальних даних, щоб уникнути помилкового віднесення товарів до найбільш поширених категорій;

– Для підвищення точності розпізнавання цін товарів можливим рішенням є використання OCR-систем на комерційній основі, які надають точніші інструменти розпізнавання тексту.

3.4 Висновки до розділу

В даному розділі було розглянуто особливості та призначення мови програмування *Python,* її можливості та призначення. Було розглянуто ряд бібліотек, які використовувались під час розробки програмного модуля, функціонал та сфери їх застосування.

Було розроблено програмний продукт, що, використовуючи передане йому зображення касового чеку, проводить розпізнавання на ньому товарів та категорій. Приведено програмну реалізацію ключових кроків його алгоритму, таких як підготовки зображення, розпізнавання тексту, навчання моделі класифікації та виведення їх результатів.

Проаналізовано показники точності класифікації рядом методів, та на їх основі обрано той, який використовується в фінальній версії програмного модуля. Приведено зразок роботи модулю, проаналізовано його помилки, їх причини та способи вирішення.

ВИСНОВКИ

Метою дипломної роботи була розробка модуля розпізнавання касових чеків для програми персонального фінансового обліку, що дозволить скоротити час та зусилля, що витрачаються на відстеження витрат, та автоматизувати цей процес.

Досягнення фінансової стабільності є вкрай важливим для кожної людини. Ключовим фактором для цього є контроль та відстеження здійснюваних витрат задля збереження персонального бюджету. Програмні засоби можуть виступати помічником у цьому процесі, надаючи широкий функціонал обліку фінансових витрат та доходів. Розширення можливостей таких додатків дозволяє користувачам доцільніше стежити за своїм фінансовим благополуччям, а спрощення ведення обліку, реалізація нових підходів до цього надає нам доречно будувати свою фінансову політику.

Для досягнення визначеного результату був проведений аналіз актуальності даного питання та можливі шляхи його вирішення. Було розглянуто технологію *OCR*, її особливості, сфери використання та перспективи. Визначено послідовність кроків, які необхідно виконати для підготовки вхідного зображення, що дозволить підвищити точність розпізнавання. Розглянуто ряд популярних *OCR*-додатків, які дозволяють провести зчитування тексту із зображення, встановлено їх особливості, функціонал, переваги та недоліки. В результаті їх аналізу було здійснено вибір *OCR*-додатку для використання в дипломній роботі – ним було обрано *Tesseract* за рахунок його наступних переваг:

– Високий показник точності розпізнавання;

– Значна кількість підтримуваних мов (включаючи українську);

– Можливість використання в режимі офлайн (без підключення до мережі Інтернет);

– Відкритий формат поширення;

– Можливість використання словників мов для перевірки правильності розпізнавання дозволяє підвищувати його точність;

– Наявність значної кількості мов програмування, що містять реалізації даної бібліотеки.

Після огляду інструментів для оптичного розпізнавання символів було проведено проектування модулю. Розглянуто специфіку предметної області, виділено основні фактори, які впливають на досягнення поставленої цілі. Проведено аналіз подання товарів у касових чеках з метою вибору максимально змістовного відображення інформації, яке може бути коректно сприйнято і оброблено алгоритмами машинного навчання. Побудовано алгоритм роботи модулю, встановлено призначення кожного з його кроків. Розглянуто ключові методи підготовки зображення до розпізнавання, такі як алгоритм Кенні, перетворення Хафа, метод Оцу.

Розглянуто та обрано метод *TF-IDF* для перетворення текстових даних у числові, які можуть бути оброблені моделями класифікації, проаналізовано його застосування, принцип роботи та математичну основу. Підготовано навчальний та тестувальний набір даних для навчання моделей класифікації, у якому підібрано товари 7 основних категорій товарів. Сформовано модель оцінки точності класифікації з метою відображення фактичних її показників для кожного з класів, а також при роботі з незбалансованими класами.

На основі сформульованого алгоритму було реалізовано програмний продукт, що розпізнає товари на фотографії чеку та співвідносить їх до відповідних категорій, вираховуючи сумарну вартість кожної із них. Програму реалізовано мовою програмування *Python*.

На основі отриманих даних після проведення категоризації товарів чеку в порівнянні з ручним методом ведення витрат, можна зробити висновок про досягнення поставленої мети: час, витрачений на розпізнавання та класифікацію чеку розробленим модулем значно менше часу, що витрачається на ручне розбиття товарів за категоріями та обчислення їх сумарної вартості.

Результати роботи модулю передаються у формі, придатній для зручного подальшого використання та оброблення. Закладена можливість подальшого

удосконалення визначеного алгоритму та розширення охоплення ним різних груп товарів.

Подальшим розвитком програмного модулю може бути його інтеграція із повноцінними додатками для ведення персонального фінансового обліку з метою полегшення та пришвидшення обліку витрат. За рахунок зміни чи доповнення набору даних, що відповідають обраній сфері, реалізований модуль може бути використаний не тільки для чеків продуктових магазинів, але і в інших областях, де потрібне проведення категоризації із касового чеку шляхом повторного навчання моделі. За допомогою уточнення процедури класифікації, окрім назв товарів та їх цін, може бути реалізовано вилучення додаткових полів, таких як назва магазину, його адреса, дата та час здійснення покупки тощо.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Highly corrupted image inpainting through hypoelliptic diffusion / U. Boscain, R. A. Chertovskih, J. P. Gauthier et al. // Journal of Mathematical Imaging and Vision. — 2018. — Vol. 60, no. 8. — P. 1231–1245.*

2. *Dipanjan Sarkar. Text Analytics with Python. — Berkeley, CA: Apress, 2019.*

3. *Detect text in images*. – [Електронний ресурс]. – Режим доступу: *https://cloud.google.com/vision/docs/ocr* .

4. *ABBYY FineReader Engine*. – [Електронний ресурс]. – Режим доступу: *https://www.abbyy.com/ocr-sdk/*.

5. *Building Custom Deep Learning Based OCR models*. – [Електронний ресурс]. – Режим доступу: *https://nanonets.com/blog/attention-ocr-for-text-recogntion/.*

6. Распознавание текстовых изображений при помощи нейронных сетей – [Електронний ресурс]. – Режим доступу: *https://rep.bntu.by/bitstream/handle/data/27276/Raspoznavanie_tekstovyh_izobrazhenij_pri_pomoshchi_nejronnyh_setej.pdf*

7. Математическое моделирование систем распознавания изображений, содержащих текстовую информацию, на основе нейронных сетей – [Електронний ресурс]. – Режим доступу: *https://moluch.ru/archive/98/21912/.*

8. Обзор библиотек обучения нейронных сетей на языке Python – [Електронний ресурс]. – Режим доступу: *https://moluch.ru/archive/336/75084/*

9. *The Math behind Linear SVC Classifier*. – [Електронний ресурс]. – Режим доступу: *https://www.kaggle.com/xingewang/the-math-behind-linear-svc-classifier* .

10. *Vectorization, Multinomial Naive Bayes Classifier and Evaluation*. – [Електронний ресурс]. – Режим доступу: *https://www.ritchieng.com/machine-learning-multinomial-naive-bayes-vectorization/.*

11. *OpenCV Tutorial – Erosion and Dilation of Image*. – [Електронний ресурс]. – Режим доступу: *https://machinelearningknowledge.ai/opencv-tutorial-erosion-and-dilation-of-image/#Dilation.*

12. *Applying Machine Learning to Product Categorization*. – [Електронний ресурс]. – Режим доступу: *http://cs229.stanford.edu/proj2011/LinShankar-Applying%20Machine%20Learning%20to%20Product%20Categorization.pdf*.

13. ГОСТ 19.701–90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.

14. ДСТУ ГОСТ 7.1:2006. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання.

15. ГОСТ 2.106–96 ЕСКД “Текстовые документы”.

16. ДСТУ 3008–95 “Документація. Звіти у сфері науки і техніки. Структура і правила оформлення”.

Додаток А

Лістинг коду модулю розпізнавання тексту

```
def recognize_text():
    import cv2
    import pytesseract

    pytesseract.pytesseract.tesseract_cmd = r'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'

    img = cv2.imread("recept2.jpg")
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    ret, thresh1 = cv2.threshold(gray, 000, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
    rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (28,28))
    dilation = cv2.dilate(thresh1, rect_kernel, iterations=1)
    cv2.imshow("dilation", dilation)
    cv2.waitKey(0)
    contours, hierarchy = cv2.findContours(dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    im2 = img.copy()
    cv2.drawContours(img, contours, -1, (0, 255, 0), 3)
    filename = 'savedImage.jpg'

    cv2.imwrite(filename, img)

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
```

```
rect = cv2.rectangle(im2, (x, y), (x + w, y + h), (0, 255, 0), 2)
cropped = im2[y:y + h, x:x + w]
text = pytesseract.image_to_string(cropped, lang="ukr")
print(text)
print("\n")
```

Додаток Б

Лістинг коду модулю усунення перекосу

```
import numpy as np
import cv2
import math
import statistics
from scipy import ndimage
from PIL import Image

def deskew():
    img_before = cv2.imread('input_img.jpg')

    img_gray = cv2.cvtColor(img_before, cv2.COLOR_BGR2GRAY)
    img_edges = cv2.Canny(img_gray, 100, 100, apertureSize=3)
    img_edges2 = cv2.Canny(img_gray, 100, 150, apertureSize=3)
    img_edges3 = cv2.Canny(img_gray, 100, 250, apertureSize=3)

    #cv2.imshow("with canny default", img_edges)
    #cv2.imshow("with canny 150", img_edges2)
    cv2.imshow("with canny 250", img_edges3)
    key = cv2.waitKey(0)

    lines = cv2.HoughLinesP(img_edges3, 1, math.pi / 180.0, 100,
minLineLength=100, maxLineGap=2)

    angles = []

    for [[x1, y1, x2, y2]] in lines:
        cv2.line(img_before, (x1, y1), (x2, y2), (255, 0, 0), 3)
```

```
            angle = math.degrees(math.atan2(y2 - y1, x2 - x1))
            angles.append(angle)

        cv2.imshow("Detected lines", img_before)
        key = cv2.waitKey(0)

        median_angle = statistics.median(angles)
        img_rotated = ndimage.rotate(img_before, median_angle -
np.sign(median_angle)*90)

        print(f"Angle is {median_angle:.04f}")
        cv2.imshow("After", img_rotated)
        cv2.imwrite('rotated.jpg', img_rotated)
        key = cv2.waitKey(0)
```