ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

OCR – Optical Character Recognition

ОРС – Оптичне Розпізнавання Символів

XML – Extensible Markup Language

JVM – Java Virtual Machine

ASR – Automatic Speech Recognition

OHA – Open Handset Alliance

JNA – Java Native Access

JDK – Java Development Kit

SDK – Software Deve lopment Kit

XML – Extensible Markup Language

ВСТУП

Об'єкт дослідження – процесс та механізми технологій OCR.

Предмет дослідження – мобільний додаток з використанням технологій OCR.

Мета дослідження – вивчення систем розпізнавання тексту та використання цих технологій для автоматизації роботи з додатками по стеженню за фінансами.

Методи дослідження – навчання нейронної мережі tesseract для подальшого її використання в рамках цієї дипломної роботи.

Сучасне життя швидке та повне подій, як головна трасса у час пік. Та рухатись по цій трасі нам допомагають гроші, це основне паливо для нашого щоденного існування. Тому у цьому наповненому подіями житті необхідно стежити за своїм пальним - за своїм особистим бюджетом, але це не так легко як спочатку здається.

Дякуючи технологіям нам більше не потрібно вести щоденники наших витрат. У кожного з нас є банківські картки які є нашим основним способом оплати. У кожного банку є системи за допомогою яких можна дуже легко простежити за своїм балансом та побачити свої минулі витрати. Ці сервіси дійсно зручні, але в основному, не дають змогу користувачу побачити деталі своїх покупок, тільки сумму, дату, та місце, чого зазвичай не достатньо коли намагаєшься згадати на що ти витратив гроші в останню п'ятницю минулого місяця... Широко використовуються електронні гроші, але дуже багато покупок людина все ще робить за допомогою фізичної валюти. І в цьому випадку єдине підтвердження вашої покупки - чек.

Ще з появою перших мобільних девайсів, однією з їх основних задач було використання програм "органайзерів". Записники, календарі, нагадування - це все вже стало стандартною рутиною для всіх нас завдяки смартфонів та мобільних додатків.

З розвитком технологій приходить ще одна неймовірна річ - автоматизація. В сьогоднішніх реаліях стараються автоматизувати абсолютно усе, аби з кожним

роком людству ставало жити ще легше та комфортніше. З часом автоматизація прийшла і до таких додатків, як “органайзери”. Вона проявляється у різних формах, але одна з основних це - розпізнавання.

Наприклад за допомогою персональних помічників, таких як “Siri” від компанії Apple, можливо голосом (технологія ASR **)** встановити собі будильник, перевірити погоду на завтра, або встановити таймер. За допомогою технології розпізнавання тексту OCR можливо “в реальному часі” перекласти текст просто направивши на нього камеру. Усе це сильно заощаджує наш з вами час.

Додатків, за допомогою яких, можливо слідкувати за своїми витрати дуже багато, але в основному людина в ручну записує свої витрати, вказуючи тему, дату, суму тощо. Витрат за день буває багато, тому багато часу втрачається на ручне “внесення” цієї інформації в додаток. Але як і все інше в нашому житті, процесс роботи з такими додатками можливо автоматизувати.

Під час покупки за готівку, людина отримує чек. В чеку є уся стандартна інформація, що люди хочуть запам'ятати: місце покупки, дата, сумма. Чому би не використати цю вже готову інформацію, щоб не вводити її вручну ще раз в додаток? На допомогу приходять технології оптичного розпізнавання символів (OCR), за допомогою яких можливо “витягнути” всю необхідну інформацію з чеку, використовуючи камеру сматфона.

Мета цієї магістерської роботи полягає в вивченні технологій розпізнавання тексту та їх використанні для автоматизації бізнес процесів.

Основною розробки буде freeware мобільних додатків на android.

1. ДОСЛІДЖЕННЯ СФЕР ЗАСТОСУВАННЯ МЕТОДІВ OCR

Спростити людське життя машинами та навчити їх робити різніпрості речі, такі як считування текстів, звуків, навіть природних ландшафтів - давнє бажання людства. Перші спроби до створення OCR можна віднести до появи новітніх методів сканування текстів. Протягом першої половини дев'ятнадцятого століття було зроблено декілька спроб розробити пристрої для допомоги сліпим людям в повсякденному житті за допомогою методів розпізнавання символів.

Але сучасна версія OCR з'явилася лише в 1940 році, коли було розроблено першиу ЕВМ. Відтоді спонуканням до розвитку стали нові можливості для працевлаштування та розвитку нових галузей взагалому. Через технологічну революцію 1950-х років цифрові дані стали важливим напрямком розвитку та інновацій. Тоді дані вводились перфокартами, тому була потреба в більш економічному методі. Тим часом розвивалася технологія OCR і перші комерційні машини з'явилися, здається, в середині 1950-х. Перша машина OCR була встановлена в Readers Diest у 1954 році. Ця машина використовувалася для форматування звітів про продажі, зроблених на друкарських машинах, у комп'ютерні перфокарти

Машини, що використовулись в бізнесі з 1960 по 1965 роки можна назвати першим комерційним поколінням OCR. Перше покоління машин відрізнялося суворими обмеженнями у формі літер. Символи було розроблено спеціально для машинного читання, і людина не завжди мала можноливість виконувати свою роботу. Через деякий час з'явились багаторядкові машини, котрі могли розгадати цілих десять рядків. Але кількість доступних символів, що піддаються розвізнаванню була обмежена використанням методу розпізнавання зразків, який порівнював символи з наявною цифровою бібліотекою.

Комерційні OCR машини другого покоління почали з'являтись в 1960 – 1970рр. Ці системи можуть розпізнавати стандартні символи блоку, а також мають

мінімальне розпізнавання почерку. Наявна мінімальна можливість обмежувалась набором цифр і декількома літерами, достатніх для, скажімо, самого поштового індексу.

Першим справді популярним пристроєм цього типу стала IBM 1287. Тоді ж Toshiba розробила перший корпус для сортування пошти, який базувався на розпізнаванні поштових індексів. У цей період відбувся сильний розвиток у галузі стандартизації, і були розроблені спеціальні шрифти OCR-A, розроблені американцями, та OCR-B, що розроблений Європейцями. Були спроби створити спільну двухтипову лінію, але натомість на ринку з’явилися машини, які могли негайно розпізнати обидва шрифти. Третє покоління було розроблено для вирішення проблеми з рукописним текстом.

Багато інструментів та програмного забезпечення для полегшення життя вже доступні в Інтернеті, і одним із таких інструментів є інструмент OCR. OCR - це оптичне розпізнавання символів, яке в самій назві розпізнає літери та текст. OCR ставінтегрованим із багатьма програмами та повсякденним життям, але ми навряд чи часто помічаємо його там.

Інструмент OCR використовується для перетворення відсканованих документів або зображень у доступні для пошуку, розпізнавання та машиночитання дані. Стандартно комп'ютер бачить лише відскановані документи та зображення як чорно-білі крапки, але за допомогою інструмента OCR ці тексти можна розглядати так, ніби вони були закодовані з комп'ютера, ідентифікуючи їх як літери, а не крапки. Цей метод також відомий як Машиночитаність.

Ми використовуємо засоби розпізнавання текстів, що призводять до ефективності, доступності, економії грошей та місця. Ми також дізналися, як використовувати один із наших інструментів PDF, на DeftPDF, для перетворення відсканованих файлів у читабельні та доступні для пошуку документи.

Копіювання та зберігання документів

Найпопулярнішим використанням інструмента OCR є перетворення відсканованого файлу в цифрову версію та надання користувачам можливості копіювати та вставляти вміст як текст у буфер обміну. Якщо він не призначений

для OCR, то друковані матеріали без цифрового резервного копіювання потрібно буде зашифрувати та часто записувати на ваш комп’ютер. Багато часу і грошей було витрачено на те, щоб зробити все це, як OCR став доступним, або зробити це ще до того як він став доступним. Звичайно, людський фактор також була проблемою, оскільки переписування стало причиною багатьох неточностей та помилок.

Сканування зашифрованих кодів

Сьогодні мобільні додатки та багато мобільного програмного забезпечення готові продавати пропозиції, які часто ведуть до цієї платформи. Друковані бланки, газетні купони та картки з лояльністю – в минулому, але цей простий рекламний метод все ще використовується у цифрових виданнях. За допомогою інструмента OCR коди можна автоматизувати за допомогою готових до сканування мобільних кодів. Вони були використані та впроваджені значною мірою багатьма компаніями у своїх кампаніях.

Файлова система Office

Завдяки інструменту OCR тепер можливо попрощатися з паперовими документами та зустріти нову еру - цифрових документів. Відскановані елементи можна легко знаходяться на перетворються як машиночитаний файл, в якому можна шукати будь-який текстовий зміст , включаючи ім'я, ключові слова, умовні скорочення, позначення чи фрази. Пошук інформації перестав бути проблемою. Ви також маєте можливість зберігати конфіденційні дані з повишенем рівнем безпеки, використовуючи захист паролем. Уявіть, як виглядали офіси раніше? Вони мали тонни картотечних шаф і потребували нескінченної кількості робочої сили та місця для підтримки порядку, особливо в практикуючих юридичних фірмах.

Автоматичне вилучення даних

Окрім файлової системи, засоби OCR також допомагають усунути ручне написання та зменшити кількість помилок введення даних. Підприємства використовують оптичне розпізнавання символів (OCR) для автоматизації введення даних і автоматичного сортування.Автоматичне вилучення данних стало гарним рішенням для економії часу.

Магазини самообслуговування

Продуктові магазини, торгові центри, кінотеатри та навіть автомати з продажу квитків застововують технологію OCR. Простота придбання та отримання повної інформації про продукт стає швидкою та простою, просто використовуючи мобільний телефон та апарат OCR. Індійські залізничні перевізники - один із прикладів використання торгових автоматів з OCR. Клієнтам необхідно лише придбати квиток через мобільний додаток, отримати доступ до точки OCR для сканування, і машина відразу роздрукує квиток. Оскільки ця система доступна, довгих черг більше не існує.

Витяг із зображень

Копіювання тексту із фотографій та самих фотографій тепер займаж лише одну мить. Після того, як ваше зображення перетворено за допомогою інструментів OCR, весь вміст можна скопіювати та викиростовувати для пошуку в інтернеті, тощо, як машиночитаний документ.

Форматування рукописних листів, старих книжок, рукописів та письмових нотаткок

Музеям та бібліотекам важко зберігати матеріали, які людям потрібно було перечитати. Тепер завдяки технології OCR старовинні книги, рукописи, нотатки та письмові листи тепер доступні для пошуку в Інтернеті. Зміст історичних даних може бути проіндексовано за допомогою OCR та бути широко доступним, не турбуючись про втрату такого важливого матеріалу через неправильні дії людей.

Допомога сліпим

Ви коли-небудь замислювались, як сліпі люди можуть читати і як вони можуть працювати в офісі? Завдяки технологіям інваліди отримують можливість працювати в офісі за допомогою програм перетворення тексту в мову та технології оптичного розпізнавання символів. Вони обидва допомагають читати документи вголос. З розвитком технологій виробництва мобільних пристроїв і спрощення процесу розробки мобільних додатків, OCR-системи стали невід'ємною частиною різноманітних програм: починаючи розважальними, закінчуючи програмами-помічниками систем управляння

1.1 Дослідження методів роботи технології OCR

У вас була необхідність розібрати нерозбірливий почерк коллеги та зрозуміти, про що там йдеться, коли його не було поруч? Тоді вважайте, себе щасливчиком, що ви не працюєте у поштовій службі, яка повинна доставляти та розшифровувати близько мільйона рукописних повідомлень кожного дня!

Допустимо, що життя лечше чим воно є насправді, і в алфавіті ми маємо лише одну букву А, і всі пишуть її однаково. Але навіть з розпізнаванням однієї букви могли бути величезні проблми, оскільки всі пишуть листи по-різному. Друкований текст полегшує завдання, але не сильно, тому що є різні шрифти, все одно будуть проблеми, оскільки текст друкується в різних цифрових форматах. Тому при перегляді принципу роботи існують два основні методи, цими методами є метод розпізнавання зразків, а також метод розпізнавання ознак.

OCR. Нейронні мережі

Дані складаються з вхідних та цільових даних. Одиночна буквено-цифрова орієнтація може бути виконана за допомогою вхідного та вихідного вектора. Довжина вхідного вектора буде точністю введення. в іншому значенні. Якщо вони працюють з четвертою сіткою, то довжина становитиме 77 елементів, кожен рядок в ній йде за іншим. Елементи можуть мати значення 0 або 1. Якщо сітчаста частина зображення, представлена векторним компонентом, охоплює більше 50% характеристик символу, тоді значення векторного елемента дорівнює 1, інакше воно дорівнює 0.

Перевіряючи це для всіх частин мережі, ми отримуємо вектор. Під час первинної нейронної мережі ми маємо 10 чисел. Отже ми матимемо 10 рядків цього типу вектора, кожен рядок представляє число. Це описує вхідну матрицю. Цей тип моделі даних можна очікувати як вхід через Matlab у нейронну мережу. Для мережі з N виходами вихідною матрицею буде матриця розмірності NxN, заповнена нулями та нулями на головній діагоналі. Цей крок розпізнавання виконувався вручну. Друкований текст полегшує завдання, але не сильно, тому що існють інші проблеми, такі як різница шрифтів.

Розпізннання візуального об’єкту

Системи оцінки алфавіту пропонують потенційні переваги, забезпечуючи інтерфейс, що полегшує взаємодію з інтерфейсами людина-машина. Деякі області застосування, де OCR поводиться найбільш критично, складаються з архівування документів, автоматичної перевірки перевірок, введення даних та різноманітних бізнес-орієнтованих додатків. Протягом останніх кількох років були проведені дорогоцінні дослідження в галузі розпізнавання алфавіту, і цій темі було присвячено багато дослідницьких рукописів та новин. Попередня обробка може бути такою ж дискретною, як підтримка та створення подання рукопису, і підходить для введення в механізм OCR. Найважливішими етапами попередньої обробки є:

- усунення білого шуму;
- визначити відхилення;
- забезпечити вдосконалення.

Звук передається за допомогою стратегії візуального сканування в межах зусиль, що призводять до невдалого продуктивного плоттера. Ці недоліки повинні бути усунені рано, щоб не спотворити витвір. Звук може бути представлений на ілюстрації, отримуючи зображення разом із програванням. Звук може бути з колонок окремого типу, таких як гауссовий звук, гамма-звук, релейський звук, експоненціальний звук, уніфікований звук, переривчастий звук тощо. Легко доступна можливість зробити революцію в сферах діяльності, таких як сканування. Розпізнавання та корекція перекосів застаріли для вирівнювання документів за схемою синхронізації сканера. Різноманітні технології розпізнавання відхилень - це проекційний профіль, корелюючий механізм, перетворення Хафа, блок тощо. Перетворіть двійкові файли, ілюстрацію рівня тіні або сірого у подвійне зображення за допомогою порога. Двійкову картину можна отримати за допомогою адаптивного порогу, загального порогу, змінного порогу, методу тощо.

Морфологічні процеси також використовуються при попередній обробці. Розширення та ерозія - це морфологічні процеси, які збільшують або зменшують розмір зображення. Корозія робить об'єкт простішим разом з ерозійними пікселями

на їх кінцях. Усі пікселі, які торкаються фонового пікселя, змінюються на фонові пікселі. Проте розтягування створює статтю, яка приблизно вписує пікселі в її кінці. Усі пікселі, які торкаються пікселя сутності, змінюються на піксель об'єкта. Інші морфологічні процеси відкриваються і закриваються.

Метод оптичного визначення символів був повністю розрахований на відповідність двоступеневим зображенням чорного шляху в білому середовищі. Однак у цій звичайній області зйомки існує ряд конфігурацій зображення через різницю в просторовій забарвленні зображення, підходах програмування, роздільній здатності та форматах стиснення.

Було досліджено звичайну множину, двоступеневі (1 біт на піксель) зображення документів із просторовою роздільною здатністю 200 точок на дюйм (dpi). Ця роздільна здатність не завжди може бути достатньою для якісного OCR. Сучасні механізми розпізнавання символів працюють краще зі сканованими документами із роздільною здатністю 300 точок на дюйм або вище. Інтернет-бібліотеки, які прагнуть практично захистити перевагу оригінальної статті, часто хочуть плавного, точного відтінку сірого або кольорового сканування із роздільною здатністю 600 dpi.

Однак кольорові зображення з високою роздільною здатністю негативно впливають на системне зберігання і можуть стати занадто великими для зберігання у великих кількостях. Розумні цифрові бібліотеки використовували різні системи стиснення як для текстового, так і для нетекстового вмісту. Загальним стисненим зображенням для двоступеневого рукопису є CCITT Fax Group 3 і Group 4, тоді як фактичні зображення людей дуже інтенсивні за допомогою методу JPEG. Нижче ми наводимо опис деяких найпопулярніших форматування та структурування їх застосування до OCR.

TIFF розшифровується як Tagged Image File File. Він ізольований від сучасної та гнучкої більшості поточних форматів польових растрових файлів, що розглядаються для обміну растровими даними. TIFF був розроблений як взаємозамінні між Aldus та Microsoft Corporation. Adobe Systems придбала Aldus, а також є власником патенту на вимоги TIFF. Внутрішній аналіз деталей,

запропонований розробниками засобів друку, моніторів та сканерів, містить дуже багату область основних вимог до даних щодо вирівнювання кольорів та таблиць діапазонів.

За замовчуванням TIFF може переносити зображення з кількома діапазонами (до 64K діапазонів), випадковою кількістю бітів на піксель, кубами інформації та роздвоєними зображеннями на файл, разом із підзортованими ескізами. Підтримувані кольорові простори включають шкалу сірого, псевдо кольори (будь-якого розміру), RGB, YCbCr, CMYK та CIELab. Основна ефективність TIFF полягає в тому, що це дуже гнучкий та автономний макет, який підтримується багатьма програмами розподілу зображень. Найбільш загальними обмеженнями у TIFF є вимоги до будь-яких положень щодо зберігання векторної графіки та текстового опису, але це не представляє суттєвої загрози для OCR, оскільки більшість зусиль механізмів виявлення спрямовані на уточнення растрових зображень.

Навіть якщо його в першу чергу цікавлять цифрово відскановані статті рукописів, метод OCR в даний час стикається з розширеною застосованістю інших засобів масової інформації, таких як цифрові фотографії, відео. Основне завдання там залишається незмінним: пошук тексту на зображеннях; Але формат зображення не обов’язково повинен бути чорно-білим. Працюючи в цій складній галузі, слідчі виконують життєво важливу мету - структурувати метод OCR, який може ідентифікувати будь-який сценарій у будь-якому статичному або анімованому зображенні, запрограмованому в будь-яку конфігурацію. Пізніші підрозділи говорять про звичайні конфігурації зображень OCR і не є звичайними для цього. За допомогою оцінки всіх ізольованих алфавітів за допомогою бази даних заздалегідь визначених алфавітів для їх ідентифікації.

Розділення літери складається з:

- встановлення кольору панелі;
- збільшення контрасту;
- фільтрування об'єктів;

- видалення всіх об’єктів, які мають менше пікселів, ніж очікувалося в алфавіті;
- ізолювання кожного символу.

Поліпшення розпізнавання шаблонів в даний час прискорюється багатьма перспективними пристроями, які є не тільки складними, але до того ж більш обчислювальними, як це очевидно в OCR, наприклад, класифікація тексту, комп'ютерне виявлення, видобуток інформації, виявлення символів, і перевірка біометрії. Область OCR підходить як основний елемент цифрових сканерів статей, і вона використовується в багатьох додатках, таких як поштові процедури, ідентифікація рукописного вводу, банківська справа, безпека (тобто перевірка паспорта, також технологія face recognition) та розпізнавання мови. Скринінг триває у цій галузі більше півстоліття, і результати дивують переможливим коефіцієнтом виявлення письмових символів понад 99%, з помітними вдосконаленнями розпізнавання рукописних функцій, оскільки рівень виявлення перевищує 90%.

Рукописні шрифти - це завершальна реакція на складний когнітивний та нервово-м’язовий розвиток, а також вплив на функції росту та навчання. Це починається з процесів соціалізації та каракулей. Після цього незріла особа починає використовувати з'єднання методом відбору, тоді як управління його рухом часто буває не дуже ідеальним. Розрахунок та отримані навички допомагають поліпшити моторний контроль і навчання особи разом із просторовими асоціаціями, створюючи таким чином просторову пам’ять або іншу когнітивну карту. Раніше ця інформація була отримана, і рухові навички починають дозрівати, цілком ймовірно, що це в напрямку вибору впорядкованого прогресування цільових точок для реалізації легкого, плавного почерку. На цьому етапі людина готова до опису, виконуючи власний підпис, формування власного почерку. Хоча опис почерку, який є основою розбіжностей між особами, рідко призначається, його виконання постійно змінюється. Часто причиною тривалого блукання виразів є похилий вік, тоді як невелика зміна почерку викликана або психосоматичними причинами, або зовнішніми особливостями, такими як різні ситуації, інструменти сценаріїв,

поверхні підпису або інші незрозумілі причини. Моделювання цієї дисперсії є головною проблемою при розгляді підписів.

1.2 Основні проблеми роботи з OCR

Відкриття оптичного алфавіту, як палаюча галузь галузевих технологій, зазвичай використовується сьогодні для проектів широкого спектру масштабів - від рідкісного сканування документів до створення масивних архівів цифрових документів. Однак це все ще область активних технічних досліджень та креативної інженерії. Ми можемо класифікувати найважливіші події в наступному розслідуванні в останній галузі OCR. Гнучке OCR прагне до активного використання широкого кола зображень статей, надрукованих шляхом обробки:

- багатьох алфавітів та багатомовної ідентифікації;
- сценарії Omni;
- вільний розподіл матеріалів;
- визначте числовий документ.

Розпізнавання рукописної поведінки - це зростаючий метод оптичного розпізнавання символів, який має на меті бути надзвичайно потужним та гнучким. Як правило, залишається відкритою проблемою, яка динамічно досліджується, щоб полегшити рішення корпорацій при їх массовому використанню технологій OCR, наприклад:

- визначення підпису, написаного у документах
- ідентифікація почерку в індивідуальних чеках
- сканування поштових та посилочних адрес
- розпізнавання обличь у портативних та мобільних пристроях.

Оптимізація зображень сценарію передбачає вибір та впровадження фільтрів зображень, придатних для оригінального есе-зображення, щоб полегшити вдосконалений специфічний механізм OCR для ідентифікації алфавітів

Процедура інтелектуального опублікування має велике значення для підвищення точності ідентифікації OCR та створення надійних схем пошуку інформації (IR), які використовують елегантні методи індексації та дискреційного узгодження для зберігання та отримання сценаріїв оглушливих результатів OCR. Мультимедійне оптичне розпізнавання символів (OCR) - це каталітичне вдосконалення, яке є способом визначення оптичного алфавіту в засобах масової інформації, крім письмових, наприклад, зображеннях, відео в Інтернеті.

1.3 Основи системи OCR

На сьогодні доступний широкий спектр систем оптичного розпізнавання алфавітів. Деякі з них беруть участь у попередньому розповсюдженні для ідентифікації статей, а деякі з них цього не роблять, однак ми можемо впевнено припустити, що якийсь алгоритм OCR відокремлює ці два основні елементи:

- витримка за алфавітом;
- категоризатор.

Враховуючи зображення товару, характерні екстрактори отримують характеристику (дескриптори), якою володіє ця деталь. Традиційні особливості попередні як подарунок категоризатору, який вирішує ідентифікований товар, еквівалентний досліджуваній характеристиці. Очікується, що класифікатор також дасть номер рівня впевненості, який показує, що класифікатор стосується розпізнаного елемента. Коротко опишемо деякі класичні методи виявлення оптичного алфавіту. Відповідний шаблон є одиночним, в основному, за звичайними та основними методами класифікації. Він також відомий як відповідність матриці. Набір з усіх моделей малюнків, шаблонів(рис 1.1). Характерний фрагмент використовує окремі пікселі зображення як особливості. Категоризація алфавіту виконується із порівнянням подання природи внеску щодо масиву шаблону. Кожна оцінка в порівняльному обчисленні (заданому відомою функцією "відстань"), що з'єднує розподіл внеску та шаблон

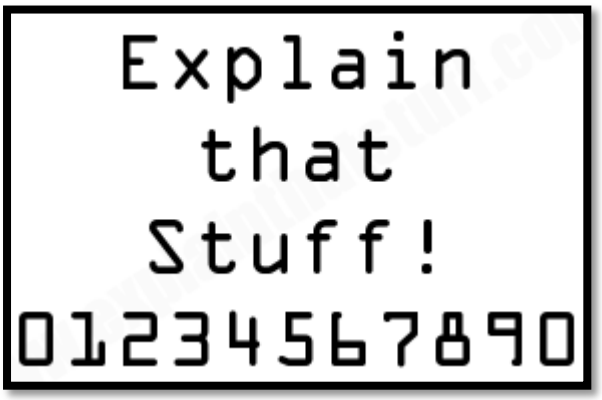

Рис 1.1 – OCR-A.

Подібність збільшується, тоді як піксель в експериментальному алфавіті неможливо відрізнити від ідентичного пікселя на зображенні шаблону (збіг). Коли відповідні пікселі відрізняються (невідповідність), міра подібності може зменшитися. Відповідно, характеристикою алфавіту виділяється із переважно подібним малюнком.

Системи конституційної класифікації займають конституційні описи та висновки щодо класифікації алфавітів. Наприклад, диспетчеризацію "P" можна описати як "вертикальний штрих із криволінійним штрихом, прикріпленим на вищому точному підйомі, складаючи отвір". Для класифікації алфавіту вхідним структурним описом алфавіту є витяг, а організація, що базується на регулюванні, функціональна для обчислення класу алфавіту. Нагадаємо, що вищезазначені методи проілюстровані в їх канонічній структурі; однак вони мають багато процедурних відхилень та методів злиття. Наприклад, узгодження шаблонів не повинно базуватися на пікселях, наприклад можна еквівалентно перетворити «вейвлет» зображень і візерунків. Конституційна категоризація не постійно покладається на норми судових рішень, можна зайняти сусіднього категоризатора знайомств за допомогою відповідної метрики в характерному просторі. Багато сучасних систем OCR базуються на арифметичних формалізмах, які зменшують певну міру неправильної класифікації. Такі розпізнавачі алфавіту, можливо, розвиватимуть піксельний опис або структурні особливості. Згадаймо деякі з них:

Класифікатори дискримінаційних функцій використовують гіперповерхні в багатовимірних просторах ознак, щоб відокремити пояснення особливостей машинопису як різнорідної семантичної програми. Такі класифікатори спрямовані на зменшення середньоквадратичної помилки класифікації.

Помилки неминучі у складних програмах зображень процесора, таких як виявлення візуальної якості; Зі звуком причини цих помилок є серйозним протистоянням процесу вищого струму, що сприяє зусиллям щодо отримання такої інформації.

Помилки класифікації алфавіту запускаються з двох основних основ: зображення частини, що визначає слабку перевагу та недостатню здатність класифікатора. Існує також багато проблем із роботаю OCR пов'язаних с глухихими людьми, і фотографії статті важко відрізнити:

- нероздільне сканування фото, мала роздільна здатність;
- неправильний або недостатній символ перед опрацюванням;
- фрагментація вступних статей у невигідний стан.

З іншого боку, сама процедура розпізнавання символів може вимагати відповідної відповіді на зазначену групу алфавітів, що призводить до помилок класифікації. Вирішити цей тип помилок може бути важко через неадекватний набір інструкцій або недостатні можливості навчання класифікатора. Типові показники розпізнавання письмового шрифту в системі можуть досягати понад 99%, крім рукописних значень повернення, і завжди нижчі, оскільки всі пишуть по-різному. Це поведінка часто виявляється в алфавітних варіаціях в характеристичному просторі, перш за все, у вищих показниках неправильної класифікації. Загальним прикладом «складного» алфавіту є буква «О», яку просто плутають із цифрою «0». Іншим найвищим прикладом може бути незрозуміле "l" із цифрою "1" або неправильне для нероздільного зображення букви "I". Піковий рівень у каталозі активів дефектів складається з

Помилки візуалізації через важкі / незначні неточності написання, арочні базові лінії тощо:

- паралельні піктограми;

- пунктуація через коми, етапи, лапки, виняткові символи;
- покращена обробка зображень виявляється в додатковому фотореалістичному моделюванні процесів друку та сканування.

Адаптивна класифікація алфавітів шляхом зміни класифікатора на існуючий рядок рукопису.

Враховуючи, що більша частина нашого життя наразі являється комп'ютеризованою, наявність зручного способу коммунікації з технологіями є життєво важливим. В основному, для того взаємодії з комп’ютерами, ми повинні використовувати доволі примітивні пристрої, такі як клавіатури та миші, щоб донести до них, чого ми хочемо. Але коли справа доходить до обробки великої кількості інформації, створеної людством за довгі роки, наприклад, перших друкованих книг чи стилізованих почерків, комп’ютери повинні ставати розумнішими. Тут на арену й ввиходить технологія оптичного розпізнавання символів (OCR). Це тип програмного забезпечення, який може проаналізувати друкований текст і перетворювати його у форму, яку комп’ютер зможе обробити. Сфера впливу дуже велика, вона потрібна в дуже різних галузях діяльності: від аналізу рукописів та рукописного тексту до величеизних систем для сортування пошти, які повинні забезпечити те, щоб всі отримувачи цих мільйонів кожноденних листів потрапили саме туди, куди потрубно і щоб отримувачи отримали її без проблем та вчасно. Як точно це та працює? Давайте досліджувати уважніше разом!

Примустимо, коли ви читаєте слова в цій дипломній роботі, не задумаючись про них, ваші очі та розум можуть візуалізуюти букви! Ваш розум працює як вичислювальна машина. Ваші очі розпізнають чорні та білі фігури, з яких складається весь наш алфавіт, а також другі мовні засоби, такі як пунктуація, а ваш мозок використовує цю інформацію, щоб зрозуміти та візуалізувати ж саме що вони означають. Частіше за все вам треба розуміти кожну букву окремо, ваш мозок відразу працює над цілими словами, а то й словосполученнями відрази. Однак, коли ви сфокусуєтесь на одному якомусь прикладі ви почтене розбирати кожну окрему літеру за літерою. Електонним пристроям з цим складніше, вони не мають очей, тому інформацію повинні отримувати зі сканерів, камер, готових цифрових

текстів, та готових цифрових фотографій. Розберемо основні принцими роботи системи.

Виділення та розпізнавання образів

Якби всі писали літеру А однаково, задача ідентифікування такою літери була б легкою. Комп'ютер має просто вирівняти відскановане зображення збереженої копієї букви А, і в разі збігу, нам вдалося розпізнати букву А.

То ж як змусити всіх людей писати однакову літеру? Наприкінца 1950-х років було створено спеціальний шрифт під назвою OCR-A, який міг би використовуватися для таких речей, як банківські чеки, тощо.

Кожна літера була моноширокою(мала однакову ширину), а межі кожної літери були визначені так детально, що кожна літера сильно відрізнялася від всіх ішхих. Більшість чекових принтерів було розроблено з використанням цього шрифту. Завдяки стандартизації шрифтів розпізнавання символів стало доволі простою проблемою для вирішення в сучасному світі. Однак проблема полягала в тому, що, той самий шрифт (OCR-A) – використовували не всі принтери і, також ні в кого не було можливості вручну малювати символи цим шрифтом. Саме через це наступним значимим кроком цього принципу є навчити програми OCR розпізнавати не один, а цілий набір основних шрифтів. Завдяки додатковому навчанню цих програм системи розпізнавання символів наразі можут розпізнати багато типів шрифтів. Але існує шанс того, що деякі складові вашого тексту будут розпізнані неправильно.

1.4 Функція ICR

ICR - інтелектуальне розпізнавання символів, це більш тонкий підхід до розпізнавання. Уявіть, що ви на секунду стали програмою розпізнавання символіві і вам одночасно дають текст написаний різними буквами, різними шрифтами, та ще й з відрізнаяючимся кутом нахилу, як ви розберете всі однакові літери А посередині всього цього безладу? Ми можемо скористатись правилом - якщо ми

бачимо дві діагональні лінії, які з’єднуються вгорі, а в середині є одна горизонтальна лінія, це і є буква А(рис 1.2).

Рис 1.2 – ICR.

Користуючись цим правилом для розшифрування наданого нам тексту, ми зможемо розпізнати більшість букв А, незалежно від її написання.

Тому робота цього принципу полягає у розпізнаванні букв не в цілому, а в розпізнаванні окремих ознак та характеристик (точок, ліній, точок зв’язку) з яких складається текст, який ми розшифровували.

Майже всі сучасні глобальні програми OCR (ті, що можуть розпізнавати надрукований текст, що написано різними шрифтами) працюють, виявляючи характеристики, а не розпізнаючи окремі зразків. Також використовуються нейронні мережі (це програми, які працюють із шаблонами наданого їм тексту і систематизують їх для подальшого використання).

Найскладніше, і знову все через людей – почерки!

Розпізнаваггя лазерним принтером точно набрантх символів доволі легке, зрівнюючи,з людським недосконалим почерком. В цій простій особливості і полягає основна проблема, коли людина завжди перемагає перед машиною: ми всі можемо зробити припущення, приблизне, але дуже вдале, про прихований сенс навіть у якійсь нерівній лінії людського почерку. То ж, я к з цим справлятися? Ми будемо використовувати комбінацію: розпізнавання особливостей і автоматичний шаблон і - найголовніше - знання того, хто саме написав цього листа, а також не останню роль буде відігравати контекстне значення написаного. Словесний контекст впливає на розуміння виразу; звідси норма не цитувати людей поза контекстом. Оскільки більша частина сучасної лінгвістики бере за об'єкти аналізу тексти, дискурси чи бесіди, сучасне вивчення словесного контексту відбувається з

точки зору аналізу дискурсних структур та їх взаємних взаємозв'язків, наприклад, зв'язок узгодженості між реченнями.

Нейролінгвістичний аналіз контексту показав, що взаємодія між співрозмовниками, визначеними як парсери, створює реакцію в мозку, яка відображає прогностичні та інтерпретаційні реакції. Тоді можна сказати, що взаємне знання, співтекст, жанр, оратори, слухачі створюють нейролінгвістичну композицію контексту.

Тому, якщо комп’ютер повинен розпізнати почерк, проблема, як правило, спрощується. У машини для сортування пошти повинна бути можливість розпізнати лише поштовий індекс, а не всю адресу отримувача. Отже, їм потрібно лише вказати порівняно маленьку кількість тексту, що генерується з основних букв та цифр. Зазвичай, від людей потребується уважно писати самі поштові індекси, витримуючи відступи між цифмари, а служби електронної пошти мають продають конверти з місцями, підготовленими для написання поштового індексу, у полях для кожної літери окремо.

Мобільні телефони та планшети часто використовують для розпізнавання рукописного тексту, вони використовують схожий принцип функцій для розпізнавання кожної літери під час її написання. Якщо ви захочете ввести букву А, то сенсорний екран може зрозуміти, що ви вводите одну косу риску, потім другу, а потім горизонтальну лінію, якою вони з’єднуються. Іншими словами, на півдорозі комп'ютер заздалегідь розуміє, що саме ви хочете написати, тому що ви формуєте букву з окремих штрихів, по одному, і це робить розпізнавання набагато простішим, ніж розпізнавання рукописного тексту, вже написаного на папері.

Спростити людське життя на навчити машини виконувати роботу замість людей – давня мрія, така само як навчити ком’ютер читати. Заснування OCR відноситься до появи перших методів оптичного сканування. На початку дев’ятнадцятого століття було зроблено значну кількість спроб розробити пристрої обробки цифрово тексту за допомогою методів розпізнавання символів.

Але сучасна OCR, якою ми її знаємо, з’явилася лише в 1940 році, коли було розроблено першу ЄВМ. Відтоді спонуканням до розвитку стали нові, майже

нескінченні можливості для збереження важливої інформації , а також працевлаштування. У 1950-х роках науково-технічна революція зробила прорив, і цифрові дані стали дуже важливим напрямком розвитку та галуззю для інновацій. Тоді дані вводились за допомогою перфокарт, тому потрібен економніший метод. Оптична технологія розпізнавання символів розвивалася безперестанно, і перші такі комерційні автомати з'явилися в середині 1950-х років. Перший "справжній" автомат OCR було встановлено у Readers Digest у 1954 р. Цей автомат використовувався для перетворення звітів про продажі, зроблені на друкарських машинках, у перфокарти для тогочасних комрьютерів.

Комерційні автомати з 1960 по 1965 роки можна назвати найпершим поколінням OCR. Перше покоління автоматів відрізнялося суворим обмеженням у формі літер. Символи спеціально розроблені для машинного читання, і як завжди, людина не виконує свою роботу ідеально. Але прогресс не стоїть на місто і з часом з’являвились багаторядкові машини, які могли розпізнавати цілих десять рядків одразу . Але кількість доступних шрифтів була обмежена використанням методу розпізнавання зразків, який порівнював символи з наявною бібліотекою прототипування.

Аппарати другого покоління з'явилися в 1960-х роках. Ці системи можуть розпізнавати символи стандартного блоку, а також вміють мінімально розпізнавати почерк. Ця мінімальна можливість обмежувалась набором цифр і декількома буквами, достатніх для, скажімо, самого поштової адреси.

Першим популярним пристроєм цього типу стала IBM 1287. Тоді ж Toshiba розробила перший корпус для сортування пошти, який базувався на розпізнаванні поштових індексів. У цей період відбувся динамічний розвиток у галузі стандартизації, і були розроблені особливі шрифти OCR-A, розроблені американцями, та OCR-B ("природний" шрифт порівняно з американським), розроблений Європою. Були спроби створити спільну лінію виробництва двох типів, але натомість на ринку з’явилися машини, які могли негайно розпізнати обидва. Третє покоління в основному спеціалізується та розроблено для вирішення проблем з рукописними текстами.

1.5 Компоненти OCR-системи

Типова система OCR складається з багатьох компонентів. На рисунку 1.3 показано типичні компоненти цієї системи.

Першим кроком у цьому процесі є заміна паперового документа – цифровим, тобто оцифруванно оного, за допомогою сканера. Коли сканер сканує текстову область, сивмоли в цій області виділяються в процесі хешування. Більше того, витягнуті коди можна обробляти заздалегідь, усуваючи непотрібні шуми та перетворюючи дані на більш зручними для розпізнавання. Ідентичність кожного знаку визначається шляхом порівняння певних ознак з описами класифікованих символів, отриманими на раніше. Нарешті, получана інформація використовується для відновлення слів і цифр оригінальності текстів. У наступному розділі ці кроки та всі задіяні методи будуть описані більш докладно.

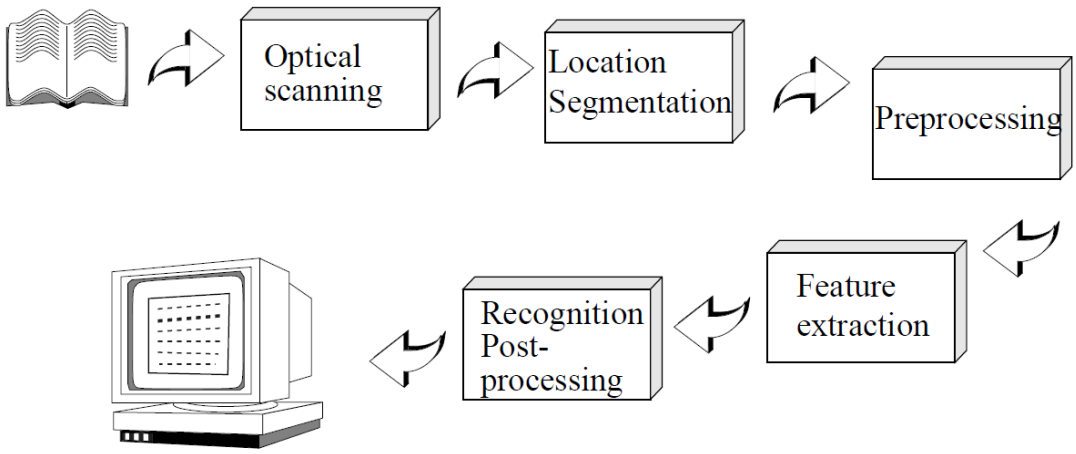

Рис 1.3 – Компоненти системи OCR.

Фотосканування

Сканери для оптичних систем розпізнавання знаків перетворюють щільність багаторівневого зображення в первинну монотонну фігуру(як зображено на рисунку 1.4), яка складається лише з двох кольорів, білого та чорного. При цьому,

якщо ми виконуємо сканування з фізичного документа, ми отримуємо цифрове зображення.

Рис 1.4 – Виправлення кута нахилу.

Цей процес називається "пороговим", і він, в основному, виконується для економії часу, обчислювального простору та зусиль для пристрою. Чим менше інформації, тим вища швидкість обробки (рис 1.4). Порогова операція, як правило, дуже проста, якщо інтенсивність світла менше мінімальної, то замініть його чорним, а якщо більше - білим. Але якщо зображення має змінну контрастність, грубий «первинний» алгоритм уже не дуже підходить, його кадр буде пом’якшено.

Первинне сканування

Отримане зображення є результатом попередніх кроків може містити помарки, неточності, камуфляж та інші недоліки, тому що була порушена орієнтація або кут нахилу літер. Отже, існує етап первинної обробки зображень, який спрямований на зменшення шуму та відновлення цілісності символів.

Виділення особливостей

Метою вилучення ознак є охоплення основних характеристик персонажів, і цей етап є найважчим з усіх. Основним методом є вилучення певних ознак, що характеризують базу символів, і не всі незначні додаткові ознаки враховуються. Технологія вилучення цих властивостей зазвичай поділяється на три основні групи:

- структурний аналіз;
- переходи;
- розподіл балів.

Різні групи можуть оцінюватися по-різному через їх чутливість до шуму та спотворень.

Існує кілька методик передобробки, одна з них, наприклад, - згладжування. Згладжування передбачає наповнення, розпушування та розтягування предметів.

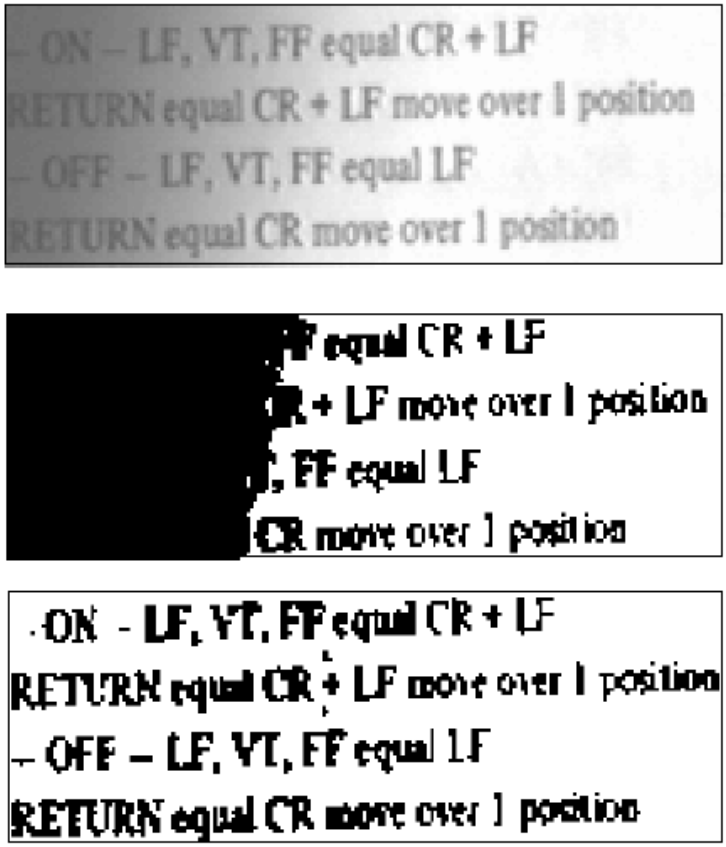

Рис 1.5 – Перетворення у монотонне зображення.

Прокладка усуває незначні розриви, прогалини тощо. Перфорація в цифрових символах, тоді як пом'якшення зменшує явну ширину рядка(рис 1.5).

Окрім згладжування, попередня обробка зазвичай включає в себе процесс нормалізації. Нормалізація використовується для отримання символів однакового розміру, нахилу та обороту. Щоб мати можливість виправити оборот, необхідно знайти кут повороту. Хаф-трансформації звичайно використовують для виявлення відмінювання. Однак точного кута повороту символу наразі знайти неможливо, поки символ не буде розпізнаний. Генерація датасета не завжди може дати дані максимальної якості, але зате їх отримання займає мінімальний час.

Розташування та сегментація символів

Сегментація - це процес, який визначає компоненти зображення. Тобто він знаходить ті області документа, в яких знаходяться необхідні дані (літери, цифри, символи). Наприклад, коли ви сортуєте пошту автоматично, вам потрібно відокремити адресу від інших відомостей у повідомленні, щоб продовжувати з нею працювати. Застосовуючи розділ до тексту, ми виділяємо слова та фрази, а потім розділяємо слова на окремі літери. Це фундаментальний процес ззовні, але з апартеїдом пов’язано безліч проблем, наприклад, якщо символи стикаються один з одним, чи розглядався графічний елемент як символ чи навпаки? Тому цей етап дуже важливий для подального визначення.

Виділення особливостей

Метою вилучення ознак є охоплення основних характеристик персонажів, і цей етап є найважчим з усіх. Основним методом є вилучення певних ознак, що характеризують базу символів, і не всі незначні додаткові ознаки враховуються.

Узгодження та кореляція зразків. Ці методи відрізняються лише одним, тим що що не виділяють жодних ознак, а навпаки, замінюють символи в матриці шаблону та перевіряють, чи є збіг. Результати кожного шаблону опрацьовуються, зберігаються на використовуються за необхідністю, яка потім береться як додаткова основа для подальшої обробки.

Класифікація

Класифікація - це процес ідентифікації кожної літери та присвоєння їй певного класу. Далі описано два різні методи класифікації.

Метод відбору базується на подібності вимірювання відстаней у різних векторах. Цей мінімальний класифікатор визначення відстані символу від символу добре працює, коли сезони корректно розділені.

Оптимальні статистичні класифікатори. Статистична класифікація використовує імовірнісний підхід до розпізнавання. Ідея полягає у використанні ідеалізованої схеми класифікації, що означає, що в середньому вона дає найменший шанс на помилку при класифікації. Класифікатор, що зменшує загальну середню пошибку, називається байєсівським класифікатором. З огляду на невідомий

символ, описаний із властивостями вектора, ймовірність належності до символу обчислюється до класу C для всіх класів C = 1 ... Потім призначається клас, який дає символу максимальну ймовірність.

Точковий розподіл.

Ця група виділяє функції на основі фіксованого розподілу точок (пікселів). Ця група максимально толерантна до спотворень та змін у моделях. Деякі типові методи для цієї групи перелічені нижче.

Перегородка. Основний прямокутник, що описує символ, розділений на кілька областей, що перекриваються або не перекриваються, і щільність точок у цих областях обчислюється та використовується, в данному методі, як характеристика символу.

Перехрестя та відстані. У техніці перетину знаки - це кількість випадків, коли фігура символу перетинає вектор у заданому шляху. Цю технологію зазвичай використовують у рекламних послугах, оскільки вона швидка і вимагає мало складності. При використанні технології відстані певні довжини вимірюються вздовж векторів руху. Наприклад, загальна довжина всіх векторів у межах конкретного символу.

Трансформація

Ця технологія допомагає зменшити розмірну контрастність та спотворення. Можуть викоривуватись різноманітні методи трансформації: Фур'є, Уолш, Хар, Адамард, Кархунен-Люф та Хоув.

Багато з цих переходів базуються на кривій, яка описує контури символів. Це означає, що ці функції дуже відчутливі до шумів, які впливають на контур символу, а не на передбачувані прогалини у контурі чи самому символі.

Структурний аналіз

Під час структурного аналізу виділяються ознаки, що описують геометричні та топологічні структури символу. Ці знаки намагаються описати фізичний склад знака, часто описуючи деякі основні елементи, такі як межі, кінцеві точки та перетини між лініями та петлями. Порівняно з іншими методами, структурний

аналіз відбілює особливості з повишеною толерантністю до шумів та змін структури.

Подальша обробка

Зазвичай, результатом розпізнавання символів у документі є набір роздільних, читаємих символів. Однак самі ці коди зазвичай не мають достатньо інформації. Натомість ми любимо зв'язувати між собою окремі літери, що належать до одного рядка, утворюючи слова, словосполучення, речення та числа. Процес виконання цього зв'язування між символами в рядках звичайно називають групуванням. Групування символів у рядки залежить від розташування символів у документі. Ікони, що прилягають одна до одної, групуються разом.

Для шрифтів із фіксованим кроком процес складання дуже простий, оскільки позиція кожної літери відома заздалегідь. Але лінії без різкості фіксовані, а відстань змінюється.

Однак простір між словами звичайно набагато більший, ніж простір між буквами, тому групування все-таки можливо. Перші складності проблеми виникають, коли розпізнаються рукописні символи або текст перекошується.

Виявлення та виправлення помилок.

Наразі розглядалася можливість групування кожної букви окремо, і звісно, контекст, в якому знаходиться кожна буква, не використовувався. Однак цього недостатньо, оскільки те, що розпізнавання окремих символів не означає, що завдання виконано. Навіть кращі OCR не забезпечують 100% правильної ідентифікації всіх символів в документі, але деякі з цих помилок можна виявити на ранніх етапах або навіть виправити, використовуючи контекст, у якому вони застововуються.

Є два основних підходи, перший полягає у використанні ймовірності появи послідовносі символів, що зустрічаються разом. Це можна зробити за допомогою правил, що визначають синтаксис, і сказати, наприклад, що за періодом повинна йти велика літера.

Також для різних мов можна розрахувати ймовірність появи двох або більше символів у послідовності та використовувати для виявлення помилок. Наприклад,

в англійській мові ймовірність «k» після літери «h» у слові дорівнює нулю, і якщо така комбінація виявлена, передбачається помилка.

Інший метод - використання словників, які виявились найефективнішим способом виявлення та ліквідації помилок. Під час пошуку слова передбачається,що воно не може існували, тобто, що воно може бути помилковим, тому слово шукається у словнику. Якщо слово не було знайдено у словнику, помилку буде виявлено і її можна виправити, змінивши слово на найбільш підходяще, подібне.

Імовірності, отримані за допомогоюкласифікації, допоможуть визначити природу неправильно розпізнаного слова. Наявність слова у словнику, на жаль, це не означає, що помилки не може бути. Можливо, сталася помилка при перетворенні слова з одного відомого слова в інше, і такі помилки не можуть бути виявлені за допомогою цієї процедури. Недоліком словникового методу є те, що пошук та порівняння словника займає доволі багато часу, але зазвичай є опраданний кроком по запобіганню помилок.

1.6 Нейроні мережі та OCR

Існує багато різних способів вирішення проблеми OCR. Один з найбільш популярних і популярних підходів заснований на нейронних мережах, які можна застосовувати до різних завдань, таких як розпізнавання шаблонів, сегментизація, класифікація, кластеризація тощо.

Нейронна мережа (NN) – найкращий інструмент, який може допомогти вирішити всі проблеми з OCR. Звичайно, важливо правильно вибрати класифікатори. Нейронна мережа - це модель обробки інформації, яка повинна працювати як мозок людини, натхненна і неперевершена висота та мрія багатьох науковців, які саме зараз достіджують проблеми та навчають нейронні мережі. Нейронна Мережа - це математична моделей, або їх сукупність, що відображають подібні властивості біологічної нервової системи людини та засновані на адаптивній біологічній аналогії навчання. Основною складовою НМ є топологія.

Сучасна нейронна мережа складається з великої кількості сильно взаємопов'язаних обробних елементів (вузлів), які пов'язані між собою зваженими дугами (посиланнями). Навчання в біологічних системах передбачає кондиціонування синаптичних зв’язків, що існують між нейронами.

Найуспішніші підходи до глибокого навчання перевершуються своєю загальністю. Однак, враховуючи описані вище функції, спеціалізовані мережі можуть бути дуже корисними.

Тут я збираюся вивчити невичерпний зразок деяких визначних підходів, і я перейду до дуже короткого їх резюме статей. Як завжди, кожна стаття відкривається словами. Уважне читання виявить, що ці методи поєднуються з попередніми роботами з глибокого навчання / розпізнавання тексту.

Результати також точно описані, але оскільки існує багато конструктивних відмінностей (включаючи незначні відмінності в наборах даних), фактичне порівняння абсолютно неможливо. Єдиний спосіб дізнатися, як ці методи працюють у вашому завданні, - це отримати їх код (найкраще в гіршому: знайти офіційне репо, знайти неофіційне, але високо оцінене репо та виконати його самостійно) та спробувати його на своїх даних.

Таким чином, ми завжди віддаємо перевагу статтям з хорошим супровідним викупом, якщо це можливо.

EAST

EAST (Ефективний детектор тонкої сцени тексту) - простий, але потужний метод виявлення тексту. Використання спеціалізованої мережі.

На відміну від інших методів, які ми будемо обговорювати, це лише виявлення тексту (а не фактичне розпізнавання), але його сила робить це вартим згадки.

Іншою особливістю є те, що він також був доданий до Відкритої бібліотеки резюме (з версії 4), щоб ви могли легко ним користуватись.

Мережа насправді є версією всім відомого U-Net, який добре підходить для виявлення функцій, які можуть відрізнятися за розміром. Первинний "стовбур" передньої подачі (як сформульовано, див. рисунокк нижче) для цієї мережі може

бути серйозним - у роботі використовується PVANet, проте додаток opencv використовує Resnet. Очевидно, що його також можна навчити заздалегідь (наприклад, використовуючи imagenet). Як і в U-Net, функції витягуються з різних рівнів у мережі(рис 1.6).

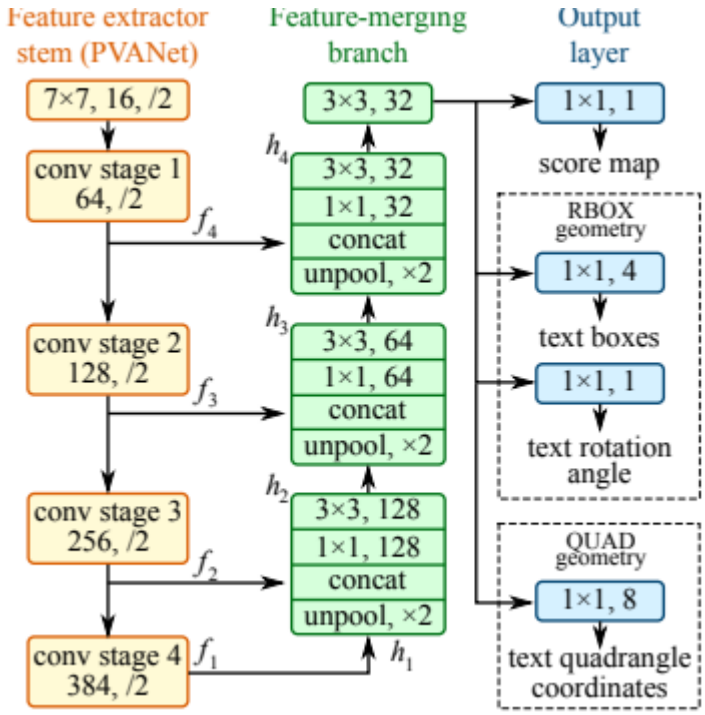

Рис 1.6 – Схема розпізнання ідеального шрифту.

Нарешті, сітка дозволяє виводити два типи периметрів для виводу: або стандартну периметрну коробку з кутом повороту (параметри 2x2 + 1), або “чотирикутник”, який являє собою просто круглий обмежуючий квадрат з координатами всіх вершин.

Якщо результати в реальному житті такі ж, як на зображеннях вище, розпізнавання тексту не потребуватиме великих зусиль. Однак результати реального життя не є ідеальними.

CRNN

Повторювана згорткова нейронна мережа - це стаття 2015 року, яка вказує наскрізну гібридну (або потрійну?) Архітектуру, призначену для захоплення слів, у трикроковий підхід. Ідея така: перший рівень – стандартна згорткова мережа.

Подивіться на зображенні нижче(рис 1.7), як кожна з цих функцій має на меті представити певний розділ у тексті.

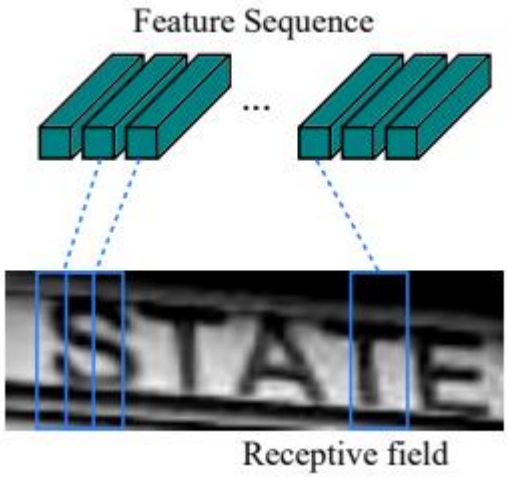

Рис 1.7 – Рецептивне поле.

Подивіться на зображенні нижче(рис 1.7), як кожна з цих функцій має на меті представити певний розділ у тексті.

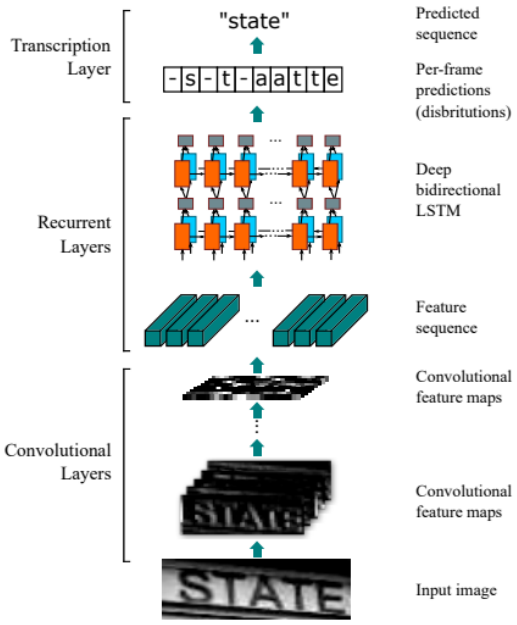

Рис 1.8 – Двонаправленний LSTM

Потім функціональні полюси подаються у глибокий двонаправлений LSTM(рис 1.8), який виробляє послідовність і має на меті знайти взаємовз'яки

між символами.

Нарешті, третя частина - це шар копіювання. Його мета полягає в тому, щоб взяти хаотичні послідовності символів, в яких одні символи є зайвими, а інші порожніми, і використати імовірнісний метод для їх стандартизації та розуміння.

Цей шар можна використовувати з / без попередньо визначеного словникового запасу, що може полегшити передбачення слів.

Це дослідження досягає високих показників точності (> 95%) за допомогою статичного текстового словника та змінних показників успіху без нього.

Машинне навчання зазвичай відбувається за допомогою навчання або розпізнавання великого набору вхідних / вихідних даних, оскільки алгоритм навчання регулює ваги з'єднань. Ваги посилань зберігають знання, необхідні для вирішення конкретних проблем. Нейронні мережі, що виникли наприкінці 1950-х, почали набирати популярність у 1980-х.

Сьогодні НМ в основному використовуються для вирішення складних реальних проблем. Часто вони вирішують проблеми, які є занадто складними для традиційних технік (наприклад, проблеми, які не мають алгоритмічного рішення, або алгоритмічне рішення яких є дуже складним), і часто добре підходять для проблем, які люди можуть легко вирішити, але потребують моделювання або заміни поведінки нога. Це хороші механізми розпізнавання зображень та надійні класифікатори, які можна узагальнити для прийняття рішень на основі неточних вхідних даних. НМ пропонують ідеальне рішення багатьох проблем класифікації, таких як мова, розпізнавання символів та знаків, функціональне передбачення та моделювання систем, в яких фізичні процеси незрозумілі або дуже складні. Перевагою нейронних мереж є їх стійкість до спотворень вхідних даних та здатність до навчання.

Поширений підхід до використання нейронної мережі для вирішення проблеми OCR базується на нейронних мережах прямого живлення з навчанням зворотного розповсюдження. Основна ідея полягає в тому, що нам спочатку потрібно підготувати набір навчальних даних, а потім навчити НН ідентифікувати зразки з цього навчального набору. На етапі навчання ми вчимо мережу реагувати

на заздалегідь відомі «завдання». Для цього кожен зразок навчання представлений двома компонентами: потенційними вхідними даними та бажаними результатами мережі на основі цих вхідних даних. Після завершення етапу навчання ми можемо ввести нові, раніше невідомі вхідні дані, і мережа створить результат, за допомогою якого ми зможемо розпізнати текст, поданий до мережі.

Приклад навчання нейронних мереж OCR з використанням растрових зображень.

Наприклад, скажімо, ми хочемо навчити сітку розпізнавати 26 великих символів у вигляді зображень 16 x 16 пікселів. Одним з найбільш очевидних способів перетворення зображення у вхідну частину навчальної вибірки є створення вектора розміром 256 (для нашого випадку), що містить «1» у всіх положеннях, що відповідають пікселю символу, та «0» у всіх положеннях, що відповідають фоновому пікселю. У багатьох навчальних завданнях НМ воліє подавати навчальні схеми "біполярно", із вхідним вектором "0,5" замість "1" та "-0,5" замість "0". Часто таке кодування шаблонів призводить до ще більшого збільшення продуктивності машинного навчання.

Для кожного можливого варіанту з 26 символів нам потрібно створити набір практичних зразків. Для завдань OCR кожен шаблон часто кодується як вектор розміром 26 (оскільки ми маємо 26 різних символів), а вектор позиціонується як "0,5" для позицій, що відповідають шаблону, і "-0,5" для всіх інших випадків.

За допомогою таких зразків, щоб вивчити всі літери, ми можемо почати тренувати нашу мережу. Для вищезазначеного завдання ми можемо використовувати один шар нейронної мережі, який матиме 256 зразків, що відповідають розміру вхідного вектора, і 26 нейронів у шарі, який відповідає розміру вихідного вектора. На кожному навчальному етапі всі зразки з навчального набору надсилаються в мережу і розраховується кінцевий шанс помилки. Коли помилка стає менше зазначеної межі помилок, навчання завершується, і мережа може використовуватися для ідентифікації.

Стандартний підхід глибокого навчання

Як випливає з назви, після виявлення "слів" ми можемо застосувати

стандартні підходи до глибокого навчання, такі як SSD, YOLO та RCNN Mask. Я не буду багато розширювати ці підходи через велику кількість інформації в Інтернеті.

Потрібно сказати, що це моя улюблена техніка, тому що мені подобається глибоке навчання - це філософія "від кінця до кінця", де ви застосовуєте надійну модель з деяким точним налаштуванням, яка вирішить майже кожну проблему. У наступному розділі цього допису ми побачимо, як це насправді працює.

Однак SSD та інші моделі виявлення є складними завданнями, коли мова йде про подібні щільні класи, як розглянуто тут. Я вважаю це трохи парадоксальним, тому що насправді моделям глибокого навчання важче розпізнавати цифри та літери, ніж тим складнішим та детальнішим речам, як собаки, коти чи люди. Вони, як правило, не досягають точної необхідної точності, а отже, спеціалізовані методи процвітають.

Приклад навчання ліній нейронних мереж OCR за допомогою функціональних класифікаторів

Описаний вище підхід працює добре, але обмежений у своїй розширюваності. У рядку та діапазоні є деякі зміни, які загальна нейронна мережа OCR не може впоратись. Подання растрових зображень як вхідних даних для ОМ ОМ дещо неправильне, оскільки люди не бачать символів "піксель за пікселем", і часом важко описати "суть" символу в цілому за допомогою простого растрового зображення. Якщо існують суттєві відмінності у растрових зображеннях у визначенні кожного символу шрифту, найкращим набором вхідних даних для NM буде набір класифікаторів, обчислених на основі растрових зображень, таким чином, що ці класифікатори є універсальними для будь-якої зміни розміру символів та деформації.

Ці класифікатори можуть включати топологічні властивості, такі як число Ейлера, компактність та геометричні властивості, такі як тире та перетини. Звичайно, ці характеристики тепер слід обчислювати з вхідних зображень і надсилати як вхідні дані в систему ОМ OCR. Система повинна бути універсальною для зміни шрифту та розміру, тому характеристики повинні спочатку дізнатися про

тип символу, такий як "a", а вже потім дізнаватися додаткову інформацію, наприклад, розмір шрифту. Як тільки ми знаємо, що таке "a", ми можемо зіставити результат із усіма визначеннями шрифтів "a" у нашій базі даних шрифтів, щоб встановити точний шрифт і розмір.

1.7 Приклад навчання нейронної мережі

Тож після всіх цих розмов, настав час зігріти руки і спробувати продемонструвати деякі моделі для себе. Ми спробуємо вирішити завдання SVHN. Дані SVHN містять три різні набори даних: навчання, тестування тощо. Відмінності не є ясними на 100%, але більший додатковий набір даних (приблизно 500 000 зразків) включає зображення, які якось легко ідентифікувати. Тож для цього ми його використаємо.

Крок 1: Проаналізуйте дані

Подобається це нам чи ні, але немає "золотого" формату для представлення даних у місіях виявлення. Деякі з добре відомих форматів: coco, via, pascal та xml. І там ще. Наприклад, набір даних SVHN пояснюється в. таємничий килимок. На наше щастя, цей вміст надає скрипт read_process_h5 для перетворення файлу .mat у стандартний файл json, і вам доведеться перетворити його у формат pascal, наприклад:

```
def json_to_pascal(json, filename): #filename is the .mat file
    # convert json to pascal and save as csv
    pascal_list = []
    for i in json:
        for j in range(len(i['labels'])):
            pascal_list.append({'fname': i['filename']
            ,'xmin': int(i['left'][j]), 'xmax': int(i['left'][j]+i['width'][j])
            ,'ymin': int(i['top'][j]),  'ymax': int(i['top'][j]+i['height'][j])
            ,'class_id': int(i['labels'][j])})
    df_pascal = pd.DataFrame(pascal_list,dtype='str')
```

```
    df_pascal.to_csv(filename,index=False)
p = read_process_h5(file_path)
json_to_pascal(p, data_folder+'pascal.csv')
```

Тепер у нас повинен бути більш стандартний файл pascal.csv, і він дозволяє нам просуватися вперед. Якщо перетворення відбувається повільно, зверніть увагу, що нам не потрібні всі зразкові дані. ~ 10Kbбуде цілком достатньо.

Крок другий: Подивіться на дані

Перш ніж розпочати процес моделювання, найкраще вивчити дані. Я пропоную лише короткий допис, щоб перевірити розум, але рекомендую зробити додатковий аналіз:

```
def viz_random_image(df):
    file = np.random.choice(df.fname)
    im = skimage.io.imread(data_folder+file)
    annots =  df[df.fname==file].iterrows()
    plt.figure(figsize=(6,6))
    plt.imshow(im)
    current_axis = plt.gca()
    for box in annots:
        label = box[1]['class_id']
        current_axis.add_patch(plt.Rectangle(
            (box[1]['xmin'], box[1]['ymin']), box[1]['xmax']-box[1]['xmin'],
            box[1]['ymax']-box[1]['ymin'], color='blue', fill=False, linewidth=2))
        current_axis.text(box[1]['xmin'], box[1]['ymin'], label, size='x-large',
color='white', bbox={'facecolor':'blue', 'alpha':1.0})
        plt.show()
```

Для наступних кроків я представляю *файл* utils_ssd.py у репо, що полегшує тренування, навантаження вагою тощо. Частина коду взята з репозиторію SSD_Keras, який також широко використовується.

Дуже важливо це відзначити, вибираючи проект для використання.

Крок 3: виберіть стратегію

Як ми вже обговорювали, у нас є кілька можливих підходів до цієї проблеми. У цій статті я буду застосовувати стандартний підхід до глибокого вивчення і використовуватиму модель виявлення SSD. Звідси ми будемо використовувати додаток SSD keras. Це прекрасна реалізація П'єра Луїджі. Хоча у нього менше зірок GitHub порівняно з додатком rykov8, він виглядає більш сучасним та простішим для інтеграції. Дуже важливо це відзначити, вибираючи проект для використання. Іншими хорошими варіантами були б модель YOLO та маска RCNN.

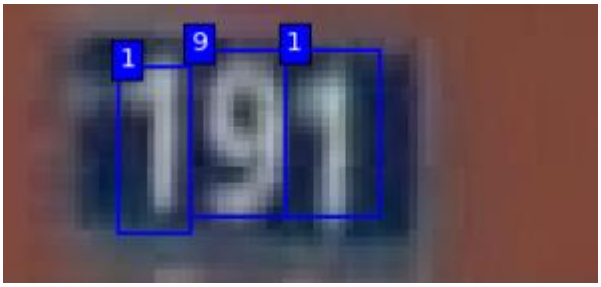

Рис 1.9 – модель виявлення SSD.

Крок 4: Завантажте та навчіть модель SSD

Деякі визначення

Щоб використовувати репозиторій, вам потрібно перевірити, чи маєте ви репозиторій SSD_keras, та заповнити шляхи у файлі json_config.json, щоб блокнот міг знайти шляхи.

Визначте модель, ваги навантаження

Як і в більшості ситуацій глибокого навчання, ми не будемо починати тренування з нуля, а навпаки, завантажуватимемо попередньо навчені ваги. У цьому випадку ми будемо завантажувати ваги моделі SSD, навчені набору даних COCO, який містить 80 категорій. Очевидно, що наша місія має лише 10 класів, тому ми відновимо верхній шар, щоб отримати правильну кількість виходів, після завантаження ваг. Ми робимо це у функції init_weights. Примітка: Правильна кількість виходів у цьому випадку - 44: 4 для кожної категорії (координати обмежуючого поля) та ще 4 для фонового класу / жодного.

Крок 5: Навчання моделі

Тепер, коли шаблон готовий, ми збираємося ввести кілька остаточних визначень навчання та розпочати тренування.

learner.init_training()

history = learner.train(train_generator, val_generator, steps=100,epochs=80)

В якості бонусу я включив зворотний виклик training_plot у навчальний сценарій для візуалізації випадкового зображення після кожної ери. Наприклад, ось знімок прогнозів після шостого етату:

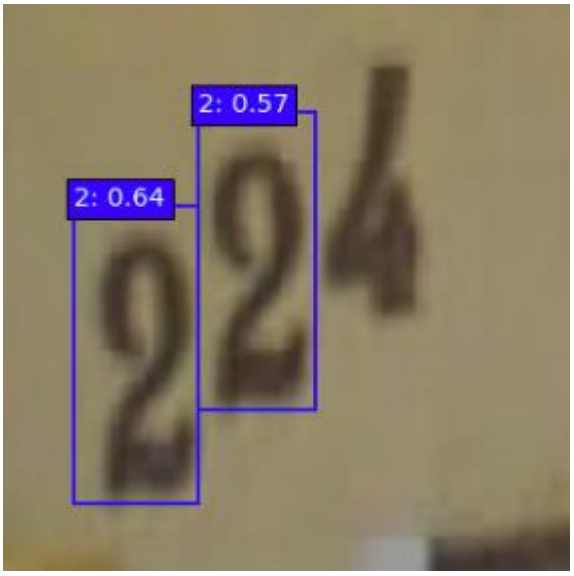

Рис. 1. 10 – шостий етап розпізначчя.

Репозиторій SSD_Keras обробляє збереження форми майже після кожного інтервалу, тому ви можете завантажувати форми пізніше, просто змінивши рядок weights_destination_path на рівний шлях

Якщо ви будете слідувати моїм вказівкам, ви зможете навчити модель. Ssd_keras надає деякі інші функції, такі як прискорювач даних, різні завантажувачі та резидент. Після короткого тренінгу я досяг> 80mAP.

Висновки до розділу 1

OCR - стара та досить проста технологія, що враховує лише базовий корінний алгоритм. Фраза "легко підібрати, важко освоїти" тут дуже доречна, оскільки просту бібліотеку OCR можна написати самостійно, за допомогою одноденного "підручника". Це рішення спрацює і зможе частково розпізнати текст, але якщо всі додаткові параметри знаходяться в ідеальному стані (наприклад, якість зображення, нахил, контраст тощо), але якщо шрифт трохи відрізняється від ідеальної моделі, на якій тренували техніку, Що зображення, що містить текст, розмите або має найменший кут нахилу. Все це, ваша скорочена технологія розпізнавання тексту буде неможливим завданням.

Тому основною суттю технологій OCR є попередня обробка зображень, яка налаштовує вхідні дані на «ідеальний» варіант, та подальша обробка, яка може більш детально визначити текст і збільшити шанс розпізнавання. Тобто суть полягає в принципах та додаткових інструментах та в їх поєднанні. Саме ці групи вирізняють різні сучасні рішення OCR. І межі можливостей тут безмежні, саме тому технології оптичного розпізнавання символів все ще швидко розвиваються, і є нові способи застосування.

2. ВІДОМІ РЕАЛІЗАЦІЇ OCR В СУЧАСНОМУ СВІТІ

З самого початку OCR розпочав свій розвиток з метою "оцифрувати" фізичні документи (книги, листи, банківські чеки тощо) і до сьогоднішнього дня головна мета технології - оцифрування фізичної інформації навколо нас не змінилася.

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

2.1 Відомі реалізації OCR та їх аналіз.

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

Tesseract - це механізм OCR для різних операційних систем. Це безкоштовне програмне забезпечення, випущене за ліцензією Apache. Спочатку розроблений компанією Hewlett-Packard як запатентоване програмне забезпечення у 1980-х роках, він був випущений у відкритому коді в 2005 році і розроблявся за спонсорської підтримки Google з 2006 року.

У 2006 році Tesseract вважався одним із найточніших механізмів OCR з відкритим кодом, доступних на той час

Спочатку Tesseract engine був розроблений як запатентоване програмне забезпечення в лабораторіях Hewlett Packard у Брістолі, Англія та Грілі, Колорадо між 1985 і 1994 роками, з деякими іншими змінами, внесеними в 1996 році для порту на Windows, та деякими переходами з C на C ++ у 1998 році. Значна частина коду була написана на мові C, потім більше написана на мові C ++. З тих пір весь код був перекладений на переклад принаймні за допомогою компілятора C ++. Протягом наступного десятиліття було зроблено чимало роботи. Потім він був випущений у відкритому коді в 2005 році Hewlett Packard та Університетом Невади, Лас-Вегас (UNLV). Tesseract фінансується Google з 2006 року

Tesseract був одним із трьох найкращих механізмів OCR для точності символів у 1995 році. Він доступний для систем Linux, Windows та Mac OS X. Однак через обмежені ресурси він лише ретельно тестується розробниками під Windows і Ubuntu.

Tesseract може обновитись до версії 2 і сприймати в якості вхідних даних лише зображення TIFF простого тексту з одного стовпця. Ці ранні версії не включали аналіз макета, тому введення багатоколонного тексту, зображень або рівнянь дало спотворені результати. Починаючи з версії 3.00, Tesseract підтримує форматування вихідного тексту, аналіз місця інформації hOCR та аналіз макета сторінки. До бібліотеки Leptonica додана підтримка ряду нових форматів зображень. Тессеракт може визначити, чи є текст монорозмірним або пропорційним.

Початкові версії Tesseract можуть розпізнавати лише текст англійською мовою. Tesseract v2 додав шість додаткових західних мов (французька, італійська, німецька, іспанська, бразильська португальська та голландська). Версія 3 значно розширила підтримку мов, включаючи ідеографічні мови (китайську та японську) та мови справа наліво (наприклад, арабську та іврит), на додаток до багатьох інших сценаріїв. Серед нових мов - арабська, болгарська, каталонська, китайська (спрощена та традиційна), хорватська, чеська, датська, німецька (фрактальний текст), грецька, фінська, іврит, хінді, угорська, індонезійська, японська, корейська, латвійська, литовська, норвезька, польська, португальська, румунська, російська,

сербська, словацька (стандартний та фрактальний шрифт), турецька, словенська, шведська . V3.04, випущений в липні 2015 року, додав 39 додаткових груп мов / сценаріїв, що дозволило загальну кількість мов підтримки перевищити 100. Нові мовні коди включали: amh (амхарська), asm (асамська), aze_cyrl (азербайджанська кириличним письмом)), Бод (тибетський), бос (боснійський), цеб (цебуанський), цим (валлійський), дзо (дзонгкха), фас (персидський), гле (ірландський), гудж (гуджараті), капелюх (гаїтянський та гаїтянський креольський), іку (Інуктитут), jav (яванська), kat (грузинська), kat_old (старогрузинська), kaz (казахська), khm (центральна кхмерська), kir (Киргизія), kur (курдська), лаоська (лаоська), латинська (латинська)), Мар (маратхі), Міа (бірманська), Неб (непальська), урі (орія), пан (панджабі), пус (пушту), сан (санскрит), гріх (сингальська), srp_latn (сербська латинським шрифтом), сер (Сирійська), tgk (таджицька), tir (тигриня), uig (уйгури), urd (урду), узбецька (узбецька), uzb_cyrl (узбецька кириличним шрифтом), їд (ідиш).

Крім того, Tesseract можна навчити працювати на інших мовах.

Tesseract може добре обробляти тексти справа наліво, такі як арабська чи іврит, багато текстів хінді на додаток до CJK. Показники точності представлені в цій презентації освітньої програми Tesseract на DAS 2016, Санторіні, автор Рей Сміт.

Tesseract підходить для використання в якості фону і може використовуватися для більш складних завдань OCR, включаючи аналіз макета з інтерфейсом, подібним OCRopus.

Якість виводу Tesseract була б дуже низькою, якби вхідні зображення не були попередньо оброблені, щоб відповідати вимогам: зображення (особливо знімки екрану) повинні бути масштабовані так, щоб висота тексту була не менше 20 пікселів у ширину, будь-яке обертання або перекоси повинні бути виправлені, або текст не буде розпізнаний, Низькочастотні зміни яскравості повинні бути відфільтровані високим проходом, інакше фаза двійкового кодування в Tesseract знищить значну частину сторінки, а темні межі повинні бути видалені вручну, інакше це буде сприйнято неправильно як символи.

Tesseract використовує мовні моделі та словники для розпізнавання тексту певною мовою. Мовна модель містить значення параметрів моделі нейронної мережі та інші навчальні дані. Наприклад, англомовна форма зберігається у файлі eng.traineddata. Користувач може створити власний список слів для Tesseract, щоб Tesseract міг навчитися його розпізнавати.

Tesseract дозволяє розширити стандартний словниковий запас будь-якої підтримуваної мови, додавши власні слова, або навчити мовну модель, повністю замінивши стандартні словникові слова власними словами.

Tesseract використовує спеціальні файли .dawg для різних категорій слів у словнику. Наприклад, файл .word-dawg використовується для словникових слів, а файл freq-dawg використовується для найбільш поширених слів.

Google Cloud Platform - це постачальник обчислювальних ресурсів для розгортання та запуску програм в Інтернеті. Його спеціальність - забезпечити місце для приватних осіб та організацій для побудови та запуску програм, і він використовує Інтернет для спілкування з користувачами цієї програми. Подумайте про десятки тисяч веб-сайтів, що працюють у мережі «надвеликих» центрів обробки даних (дуже великих, але також ділених), і ви зрозумієте основну ідею.

Коли ви запускаєте веб-сайт, додаток або послугу на Google Cloud Platform (GCP), Google відстежує всі ресурси, які вона використовує, зокрема, скільки обчислювальної потужності, сховища даних, запитів до баз даних та підключення до мережі споживає. Замість того, щоб орендувати DNS-сервер або адресу щомісяця (що ви робили б у звичайного постачальника веб-сайтів), ви платите за кожен із цих ресурсів щосекундно (конкуренти беруть плату за хвилину) із знижками, які застосовуються, коли клієнти багато використовують ваші послуги в Інтернеті.

2.2 Особливості платформи Google Cloud

То що ви насправді робите на хмарній платформі, і чому ви хочете це робити на Google? Ви використовуєте хмарну платформу, коли хочете, щоб послуги, які ви

представляєте своїм користувачам, вашим клієнтам або вашим колегам, були додатком, а не веб-сайтом. Можливо, ви хочете допомогти будівельникам будинків оцінити розміри та структуру шаф, необхідних для відбудови кухні. Можливо, ви аналізуєте статистику ефективності спортсменів, які пробують участь у коледжному спортивному клубі, і вам потрібна складна аналітика, щоб повідомити головних тренерів, чиї показники можуть покращитися. Або ви можете сканувати сотні тисяч сторінок архівованої копії газет, і вам потрібно створити сканований індекс, що датується десятиліттями.

Ви використовуєте хмарну платформу, таку як GCP, коли хочете створити та запустити додаток, яке певним чином може використати потужність надмасштабних центрів обробки даних: щоб охопити користувачів у всьому світі, взяти в користування аналітику та складні функції ШІ, скористатися перевагами масового зберігання даних або скористатися економічною ефективністю. Ви платите не за пристрій, а за ресурси, якими користується пристрій.

Вважається, що Google Cloud Platform має певні конкурентні переваги.

Автоматизоване розгортання сучасних додатків. Додаток складається з багатьох мобільних частин, саме тому деякі розробники вважають за краще будувати свої програми в хмарі для початку ("рідна хмара"). Google є творцем Kubernetes, який є багатокомпонентним координатором додатків. На початку Google застосовував ініціативний підхід до автоматизації розгортання цих багатогранних додатків у хмарі: наприклад, він відкрився для Kubo, платформи автоматизації, спочатку створеної, щоб допомогти розробникам, які використовують Cloud Foundry, поширювати свої програми з платформ розробки в хмару.

Творчий контроль витрат. Як ви побачите пізніше, замість того, щоб бути недорогим лідером, стратегія Google із GCP полягає у забезпеченні конкурентоспроможності за певними «солодкими» сценаріями. Наприклад, Google пропонує менеджер життєвого циклу для зберігання даних об’єктів, що дозволяє вивантажувати або видаляти об’єкти, які не використовувались протягом 30 днів і більше.

Набагато простіше тримати в руці користувачів, які вперше користуються послугами. Платформа хмарних сервісів може бути переважною концепцією для розуміння новачка. Подібно до того, як багатьом споживачам не було зрозуміло, в чому полягає мета мікрокомп'ютера, публічна хмара - це новий, чужий звір для людей, які звикли бачити та торкатися машини, якою вони користуються. GCP надає покрокові приклади для виконання багатьох найпоширеніших завдань - наприклад, запуску віртуальної машини на базі Linux, що нагадує заявку та налаштування вашого нового комп’ютера з повітря.

Cloud Vision дозволяє розробникам легко інтегрувати функції розпізнавання зору в додатки, включаючи тегування зображень, виявлення обличчя та орієнтирів, оптичне розпізнавання символів (OCR) та явне позначення вмісту.

Цей фреймворк Google Cloud vision містить модулі:

Label Detection - Ми можемо аналізувати зображення, виявляти та витягувати інформацію про різні істоти та сутності на зображенні. За допомогою функції виявлення міток ми можемо визначати загальнодоступні організми, місця, види діяльності, види тварин, продукти тощо.

OCR – модуль оптичного розпізнавання символів, саме про цей модуль йдеться в данній роботі.

Explicit Content Detection - Функція безпечного пошуку для фільтрації невідповідного вмісту.

Чи знали ви, що налаштування Google Safe Search забезпечується функцією SafeSearch Discovery? Функція виявлення безпечного пошуку сканує зображення та визначає, чи потрапляє воно в одну з п’яти категорій: доросле, пародія, медичне обслуговування, насилля та расизм.

Facial Detection - Це технологія, здатна зіставити людське обличчя з цифрового кадру зображення чи відеозапису з базою даних обличчя, яка зазвичай використовується для автентифікації користувачів за допомогою служб перевірки особи, і вона працює шляхом виявлення та вимірювання рис обличчя за певним зображенням.

Хоча спочатку це була форма комп’ютерних додатків, останнім часом системи розпізнавання обличчя стали широко застосовуватися на смартфонах та інших видах технологій, таких як роботи. Оскільки комп’ютеризоване розпізнавання обличчя передбачає вимірювання фізіологічних характеристик людини, системи розпізнавання обличчя класифікуються як біометричні. Хоча системи розпізнавання обличчя мають меншу точність як біометрична технологія, ніж розпізнавання райдужки та розпізнавання відбитків пальців, вони широко застосовуються завдяки їх безконтактному та неінвазивному процесу. Системи розпізнавання обличчя розгорнуті в системі вдосконаленої взаємодії людина-комп’ютер, відеоспостереження та автоматичного індексування зображень.

Соц.медіа

Після успіху краудфандингу Looksery було запущено в жовтні 2014 р. Додаток дозволяє відеочат в чаті за допомогою спеціального фільтра обличчя, який змінює зовнішній вигляд користувачів. Програми для збільшення зображень, які вже є на ринку, такі як FaceTune та Perfect365, обмежувались нерухомими зображеннями, тоді як доповнена реальність Looksery дозволяла відео в реальному часі. Наприкінці 2015 року SnapChat придбав Looksery, який згодом став функцією преміум-об'єктива. Програми фільтрів Snapchat використовують технологію розпізнавання обличчя, і на основі особливостей обличчя, визначених на зображенні, на обличчя наноситься 3D-маска сітки.

DeepFace - це система глибокого розпізнавання обличчя, створена дослідницькою групою у Facebook. Ідентифікує людські обличчя на цифрових фотографіях. У ньому працює дев'ятирівнева нейронна мережа з понад 120 мільйонами з'єднань, і вона пройшла навчання з чотирьох мільйонів фотографій, завантажених користувачами Facebook. Кажуть, що система є точною на 97% порівняно з 85% для системи ідентифікації наступного покоління ФБР.

Перевірка особи

Новим способом розпізнавання обличчя є використання служб підтвердження особи. Зараз на ринку працюють багато компаній та інші, які надають ці послуги банкам, ICO та іншому електронному бізнесу. Розпізнавання

обличчя було використано як форму біометричної аутентифікації для багатьох обчислювальних платформ і пристроїв; Android 4.0 «Ice Cream Sandwich» додав розпізнавання обличчя за допомогою фронтальної камери смартфона як спосіб розблокування пристроїв, тоді як Microsoft представила вхід для розпізнавання обличчя на відеоігрову консоль Xbox 360 через аксесуар Kinect та Windows 10 через платформу "Windows Hello" (для якої потрібна інфрачервона камера з підсвічуванням). У 2017 році смартфон Apple iPhone X представив функцію розпізнавання обличчя на виробничій лінії через платформу «Face ID», яка використовує систему інфрачервоного підсвічування.

Ідентифікатор обличчя

Apple представила Face ID на своєму флагманському iPhone X як наступника біометричної аутентифікації Touch ID, яка є системою, що базується на відбитках пальців. Ідентифікатор обличчя містить датчик розпізнавання обличчя, який складається з двох частин: модуля «Ромео», який відображає понад 30000 інфрачервоних точок на обличчі користувача, та модуля «Джульєтта», який зчитує візерунок. Форма надсилається в локальну "безпечну зону" центрального процесора пристрою, щоб підтвердити збіг з обличчям власника телефону.

Apple не може отримати доступ до стилю обличчя. Система не працюватиме із закритими очима, намагаючись запобігти несанкціонованому доступу. Технологія вчиться на змінах зовнішнього вигляду користувача і, отже, працює з шапками, шарфами, окулярами, багатьма сонцезахисними окулярами, бородою та макіяжем. Це також працює в темряві. Це робиться за допомогою "Flood Illuminator", який є виділеним інфрачервоним спалахом, який відливає невидиме інфрачервоне світло на обличчя користувача, щоб правильно прочитати 30000 точок обличчя.

Landmark Detection - Класифікуючи зображення за локацією, ми тренуємо нейронну мережу виявляти об'єкти, а потім визначати їх розташування, передбачаючи координати обмежувального вікна навколо них. Працюючи з проектами комп’ютерного зору, наших консультантів з машинного навчання часто запитують про відкриття пам’ятки. У цій статті ми пояснимо, як виявлення функцій

працює при глибокому навчанні для програм комп’ютерного зору. Дивлячись на вхідні зображення, це буде вихід такого алгоритму:

- ймовірність не знайти нічого;
- якщо об'єкт існує, то координати обмежувального вікна (bx, by, bh, bw) навколо нього.

У багатьох додатках комп'ютерного зору нейронній мережі часто потрібно розпізнавати основні цікаві місця (з обмежувального вікна) у вхідному зображенні. Ми називаємо ці пункти визначними пам'ятками. У таких додатках ми хочемо, щоб нейронна мережа виводила (x, y) координати точок об'єкта замість координат для обмежуючих квадратів.

Давайте розглянемо конкретний приклад алгоритму розпізнавання облич. Уявіть, що ми хочемо, щоб нейронна мережа вивчала двокутові позиції людського ока (тобто чотири орієнтири) і виводила вісім чисел, як показано нижче:

- (l1x, l1y);
- (l2x, l2y);
- (l3x, l3y);
- (l4x, l4y).

Але що, якби ми хотіли знайти десятки орієнтирів вздовж верхньої та нижньої оболонок ока та рота разом з деякими іншими важливими рисами обличчя на обличчі.

(l1x, l1y), (l2x, l2y), ..., (lNx, lNy)

Для цього нам потрібно буде спочатку визначити місце розташування об’єктів, а потім назвати навчальні зображення з функціями. Нанесення орієнтирів на навчальні зображення може бути непростим завданням, якщо орієнтирів та навчальних зображень у великій кількості. Важливо зазначити, що послідовність орієнтирів повинна бути однаковою для всіх навчальних зображень. Більш конкретно, якщо перший вчитель знаходиться під прямим кутом правого ока, це має бути так у всіх класифікованих прикладах у навчальному наборі.

Щоб навчити наш алгоритм, ми передамо навчальні приклади до мережі Convolutional, щоб ви могли вивчити функції, а потім ввести їх у повний зв’язок

(FC). В кінцевому підсумку FC отримає вихідні одиниці 1 + 2N; 1 двійкова одиниця (людина чи ні) та 2N одиниць для N із виділених (l1x, l1y), (l2x, l2y), ..., (lNx, lNy)

Визначення орієнтирів є одним із основних елементів програм комп'ютерного зору, таких як розпізнавання обличчя, розпізнавання позицій, розпізнавання емоцій, розпізнавання голови для розміщення на ньому корони та багато іншого в AR.

Logo Detection - це функція, яка використовує зворотний пошук зображень і дозволяє користувачам шукати пов'язані зображення, просто завантажуючи зображення або URL-адресу зображення. Google досягає цього шляхом аналізу зображеного зображення та побудови його математичної моделі за допомогою вдосконалених алгоритмів. Потім його порівнюють з мільярдами інших зображень у базах даних Google, перш ніж повертати однакові та подібні результати. Коли вони доступні, Google також використовує метадані про зображення, такі як опис.

OCR - Optical Charecter Recognition – Тут-то ми й доходимо до самого цікавого, в рамках цієї димпломної роботи, нам треба дізнатися про спроможність гугла розпізнавами тексти. У квітні 2008 року Google оголосив про App Engine, платформу для розробки та розміщення веб-додатків у центрах обробки даних Google. Це була перша хмарна послуга компанії. Взагалі кажучи, послуга стала доступною в грудні 2011 р. З моменту запуску App Engine, Google зумів додати кілька хмарних сервісів на свою платформу.

Дуже багато користувачів у цього сетменту, а, відповідно, було створено дуже багато різноматніх реалізацій технології OCR, а саме:

- розпізнання книг;
- цінників;
- табличок;
- автономерів.

Google Cloud Platform - Група хмарних сервісів, що надаються Google, що працюють на тій самій інфраструктурі, що Google використовує для продуктів кінцевих користувачів, таких як Пошук Google та YouTube. Окрім інструментів управління, надається також ряд модульних хмарних послуг, таких як хмарні

обчислення, зберігання даних, аналіз даних та машинне навчання. Для реєстрації у вас повинна бути банківська картка або банківський рахунок.

Google Cloud Platform надає такі послуги, як інфраструктура як послуга, платформа як послуга та безсерверні обчислення.

У квітні 2008 року Google оголосив про App Engine, платформу для розробки та розміщення веб-додатків у центрах обробки даних Google. Це був перший хмарний сервіс компанії. Послуга стала доступною для громадськості в грудні 2011 р. З моменту оголошення App Engine, Google додав кілька хмарних сервісів до своєї платформи.

Google Cloud Platform є частиною Google Cloud, яка також включає G Suite, корпоративні версії ОС Android і Chrome OS, API машинного навчання та Карти Google.

У січні 2019 року Google запустив чотири нові програми сертифікації для хмарних розробників та інженерів: Professional Cloud Developer, Professional Cloud Network Engineer (бета-версія), Professional Cloud Security Engineer (бета-версія) та G Suite. Ви можете пройти навчання на платформі Coursera та від інших партнерів компанії.

Підсумовіючи, то Google Cloud Platform це розвинута платформа яка дозволяє робити усе і відразу але, це і є її мінус, вона не підходить для маленьких специфічних задач.

Наразі поговоримо про існуючі рішення більш конкретно, саме про ті, які буде виконувати і моя програма.

Програмне забезпечення OCR (оптичне розпізнавання символів) надає вам можливість використовувати сканування документів для сканування рахунків-фактур, текстів та інших файлів у цифрових форматах - особливо PDF -, щоб полегшити управління всіма документами. А саме:

Adobe Acrobat Pro

Основна функція Adobe Acrobat - створювати, переглядати та редагувати документи PDF. Він може імпортувати загальні формати документів та зображень

та зберігати у форматі PDF. Також можна імпортувати вихідні дані сканера, веб-сайту або вмісту буфера обміну Windows.

Доступний лише для Microsoft Windows та macOS, Commercial Acrobat може створювати, редагувати, конвертувати, цифрово підписувати, шифрувати, експортувати та публікувати файли PDF.

Однак через природу PDF, як тільки PDF-документ створено, його природну організацію та структуру не можна корисно змінити. Іншими словами, Adobe Acrobat може змінювати вміст абзаців та зображень, але це не перенумерує весь документ, щоб вмістити довший або коротший документ. Acrobat може обрізати сторінки PDF, змінювати їх порядок, працювати з гіперпосиланнями, цифровим підписом PDF, додавати коментарі та редагувати певні частини PDF, а також гарантувати, що вони відповідають таким стандартам, як PDF / A.

OmniPage

Kofax є провідним постачальником рішень для автоматизації та підвищення продуктивності для цифрового перетворення людських та інформаційних процесів у різних організаціях. Кофакс, заснований у 1985 році, є світовим бізнесом, який базується в США.

У 2019 році Kofax придбала Nuance Document Imaging, що включає Power PDF, PaperPort, OmniPage та інші засоби обробки зображень.

Маючи понад мільйон клієнтів на підприємствах, малих підприємствах та приватних особах, OmniPage пропонує провідне в галузі оптичне розпізнавання символів (OCR) для швидкого та простого точного перетворення документів. Миттєво перетворіть паперові та цифрові документи у файли, які можна безпечно редагувати, шукати та ділитися ними.

ABBYY FineReader PDF - програма оптичного розпізнавання символів (OCR), розроблена ABBYY, з підтримкою редагування PDF-файлу з версії 15. Програма працює під управлінням Microsoft Windows v7 або новішої версії та Apple macOS 10.12 Sierra або пізнішої версії (без редагування PDF) .

Програма дозволяє перетворювати графічні документи (зображення, скани, файли PDF) у редаговані електронні формати. Зокрема, Microsoft Word, Microsoft

Excel, Microsoft PowerPoint, Rich Text Format, файли HTML, PDF / A, файли PDF, CSV та txt (звичайний текст) для пошуку. З версії 11 файли можна зберігати у форматі DjVu. Версія 15 підтримує розпізнавання тексту 192 мовами та має вбудовану перевірку правопису для 48 з них.

У всьому світі понад 20 мільйонів користувачів ABBYY FineReader. На основі оптичного розпізнавання символів FineReader ABBYY ліцензує цю технологію декільком компаніям, таким як Fujitsu, Panasonic, Xerox, Samsung та ін. Журнал PC Magazine версію 12 програми оцінив як "Відмінну"

OCRopus - Це система безкоштовного аналізу документів та оптичного розпізнавання символів (OCR), випущена під ліцензією Apache v2.0 з дуже модульним дизайном із використанням інтерфейсів командного рядка.

OCRopus був розроблений під керівництвом Томаса Бреуеля з Німецького дослідницького центру штучного інтелекту в Кайзерслаутерні, Німеччина, за підтримки Google.

OCRopus спеціально розроблений для використання у великих проектах оцифрування книг, таких як Google Books, Інтернет-архів чи бібліотеки. Повинна підтримуватися велика кількість мов та шрифтів. Однак його також можна використовувати для настільних та офісних додатків або для додатків для людей із вадами зору.

Основні компоненти OCRopus складаються з:

- аналіз верстки документа;
- OCR;
- використання статистичних мовних моделей.

Для цих компонентів доступні один або кілька сценаріїв. Модульний підхід дозволяє використовувати окремі робочі процеси та обмінюватися окремими етапами.

За замовчуванням OCRopus постачається з шаблоном для англійського тексту та шаблоном для тексту у Fraktur. Ці форми стосуються тексту і значною мірою не залежать від фактичної мови. Нові символи або лінгвістичні варіанти можуть навчатися як новим, так і додатково до них.

Нещодавно розпізнавання тексту спирається на періодичні нейронні мережі (LSTM) і не вимагає мовної моделі. Це дає можливість навчати незалежним від мови моделям, в яких одночасно були представлені результати визнання англійської, німецької та французької мов. На додаток до латинського тексту є результати інших текстів, таких як санскрит, урду, деванагарі та грецька.

Дуже хороших показників виявлення можна досягти за допомогою відповідної підготовки. Ці додаткові зусилля особливо корисні для складних документів чи текстів, які сьогодні вже не популярні і які не є в центрі інших програм OCR.

Rossum Відстеження товарів, які ви відправляєте, - це дорогий, трудомісткий процес. Збір, архівування та пошук даних з таких важливих записів, як рахунки-фактури, митні документи та коносаменти, можуть створити неефективність, якої можна уникнути.

Rossum використовує потужну навчальну технологію штучного інтелекту, щоб допомогти вам замінити ручний введення даних на автоматизований збір даних. Навіть масивні таблиці - це дитяча гра для платформи, яка дозволяє витягувати та експортувати табличні дані за лічені хвилини.

Збирайте та отримуйте доступ до інформації про доставку швидко та надійно. Допоможіть підвищити ефективність та спростити дотримання митних правил. Впорядкуйте свої логістичні операції за допомогою точного, економічного збору даних Rossum.

Висновки до розділу 2

Вибираючи рішення OCR, ви повинні розуміти, для якої мети вам потрібна ця функціональність, і відповідно до вимог щодо ідентифікації рішення. Майже всі готові рішення розпізнавання працюють за одним і тим же алгоритмом розпізнавання, але широко відрізняються між собою методами та принципами попередньої обробки та подальшої обробки зображень, а також набором додаткових фільтрів, що дозволяють краще підготувати та модифікувати зображення для розпізнавання.

Отже, якщо зображення, наприклад, з «реального» світу (наприклад, фото вивіски магазину на вулиці), то вам потрібно зафіксувати «громіздку» систему OCR з великою кількістю фільтрів, оскільки вам потрібно набагато більше фільтрувати зображення Чорно-білого розпізнавання зображень. І якщо зображення, представлене для системного введення, буде завжди в одному форматі, без спотворень і в монотонному кольорі, то немає сенсу брати «важку» систему, оскільки вхідні дані вже будуть близькі до формату, необхідного для розпізнавання.

На основі досліджень було вирішено, що найкращим варіантом реалізації розробленого мобільного тестового додатку є використання технології Tesseract OCR. Він простий і швидкий у використанні, має хорошу оболонку Java та порівняно простий у вивченні у нових шрифтах та форматах. Крім того, він підтримує українську мову з невеликим базовим набором шрифтів "нестандартно".

3. РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

Враховуючи, що розробляется мобільний додаток, то спочатку ми маємо вибрати платформу. Вибір не дуже різноманітний, залишились лиш гіганти IOS та Android, ще є Windows Platform але я не думаю, шо є якийсь сенс про нього говорити. Tessaract має оболонки на будь-які мови, відновідно, ми же сковані у виборі платформи, можемо обрати вільно. Основною моєю мовою програмування буде обрана Java. Це буде логічно тому что в мене найбільше досвіду з роботою на цій мові. Також, немає можливості розробляти на тестувати програмне забезпечення на девайсах від Apple, тому вибір IOS відпадає сам по собі. Хоча це не повинно зупиняти нас від подальшого дослідження і цієї платфоми також.

Розробка програмного забезпечення Android - це процес, за допомогою якого створюються додатки для пристроїв Android. Google заявляє "Програми для Android можна писати за допомогою мов Kotlin, Java та C ++", використовуючи Android SDK, тоді як також можна використовувати інші мови. Усі мови, що не належать до JVM, такі як Go, JavaScript, C, C ++ або Assembly, потребують допомоги від мовного коду JVM, який може надаватися інструментами, можливо, з обмеженою підтримкою API. Деякі мови програмування та інструменти дозволяють підтримувати програми на різних платформах (наприклад, як для Android, так і для iOS). Сторонні інструменти, середовища розробки та підтримка мови також продовжували розвиватися та розширюватися з моменту первинного випуску SDK у 2008 році. Офіційним механізмом розповсюдження додатків Android для кінцевих користувачів є Google Play; Це також дозволяє здійснювати поетапний випуск програми, а також розповсюджувати попередні версії програми тестувальникам.

3.1 Android SDK

Комплект розробки програмного забезпечення для Android (SDK) включає в себе

Комплект розробки програмного забезпечення для Android (SDK) включає широкий набір засобів розробки. Це включає налагоджувач, бібліотеки та емулятор телефону на основі QEMU, документацію, зразок коду та навчальні посібники. На даний момент підтримувані платформи розробки включають комп'ютери Linux (тобто сучасний настільний дистрибутив Linux), Mac OS X 10.5.8 або новішої версії та Windows 7 або новішої версії. Станом на березень 2015 року SDK недоступний для самого Android, але розробка програмного забезпечення можлива за допомогою спеціалізованих програм для Android.

Приблизно до кінця 2014 року офіційно підтримуваним інтегрованим середовищем розробки (IDE) було Eclipse з використанням компонента Android Development Tools (ADT), хоча IntelliJ IDEA IDE (усі версії) повністю підтримує розробку Android та підтримує IDE NetBeans. Також розробка Android за допомогою плагіна. Починаючи з 2015 року, Android Studio, від Google і на базі IntelliJ, є офіційною IDE; Однак розробники можуть вільно використовувати інші, але Google уточнив, що ADT офіційно припинено з кінця 2015 року, щоб зосередитись на Android Studio як офіційній IDE для Android. Крім того, розробники можуть використовувати будь-який текстовий редактор для редагування файлів Java та XML, а потім використовувати інструменти командного рядка (необхідний Java Development Kit та Apache Ant) для створення, створення та налагодження програм Android, а також контролювати підключені пристрої Android (наприклад, запустити перезапуск - Віддалено встановіть програмний пакет (пакети).

Удосконалення Android SDK йдуть паралельно із загальним розвитком платформи Android. SDK також підтримує старіші версії платформи Android на випадок, якщо розробники хочуть націлити свої програми на старі пристрої. Інструменти розробки - це завантажувані компоненти, тому після завантаження останньої версії та платформи старі платформи та інструменти також можна завантажити для перевірки сумісності.

Додатки Android зібрані у форматі apk. І збережіть його в папці / data / app в ОС Android (лише користувач root може отримати доступ до папки з міркувань

безпеки). Пакет APK містить файли dex. (Файли масового байтового коду, що називаються виконуваними файлами Dalvik), файли ресурсів тощо.

Інструменти платформи Android SDK є окремим завантажуваним підмножиною повного SDK, що складається з інструментів командного рядка, таких як adb та fastboot.

Міст налагодження Android (ADB)

Android Debug Bridge (ADB) - це інструмент для запуску команд на підключеному пристрої Android. Демон adbd працює на пристрої, клієнт adb запускає сервер у фоновому режимі для команд мультиплексування, що надсилаються на пристрої. На додаток до інтерфейсу командного рядка існує кілька графічних інтерфейсів користувача для управління adb.

Формат виконання команд зазвичай такий:

adb [-d|-e|-s <serialNumber>] <command>

where -d is the option for specifying the single USB-attached device,

-e for the single running Android emulator on the computer,

-s for specifying a USB-attached device by its unique serial number.

Наприклад, програми для Android можна зберегти, створивши резервну копію команди у файлі з іменем backup.ab за замовчуванням.

У проблемі безпеки, про яку повідомлялося в березні 2011 р., АБР був призначений вектором для спроби встановити руткіт на підключені телефони за допомогою "атаки вичерпання ресурсів".

Android NDK

Код, написаний на C / C ++, можна скомпілювати у власний код ARM або x86 (або його 64-розрядні варіанти) за допомогою Android Native Development Kit (NDK). NDK використовує компілятор Clang для компіляції C / C ++. GCC був включений до NDK r17, але вилучений в r18 в 2018 році.

Власні бібліотеки можна викликати з коду Java, що працює під час виконання Android, за допомогою System.loadLibrary, яка є частиною стандартних класів Java на Java.

Інструменти командного рядка можна компілювати за допомогою NDK та встановлювати за допомогою adb.

Він використовує Android Bionic як свою бібліотеку C, а LLVM libc ++ як свою стандартну бібліотеку C ++. NDK також включає в себе безліч інших API: стиснення zlib, графіку OpenGL ES або Vulkan, аудіо OpenSL ES та кілька Android API для таких речей, як запис, доступ до камер або пришвидшення нейронних мереж.

NDK включає підтримку CMake та власну збірку ndk (на основі GNU Make). Android Studio підтримує будь-яку з них від Gradle. Інші сторонні інструменти дозволяють інтегрувати NDK в Eclipse та Visual Studio.

Плюси використання андроід

Для характеристики процесора NDK також включає simpleperf , схожий на інструмент перфузії Linux, але з кращою підтримкою для Android та, зокрема, для гібридних кластерів Java / C ++.

Початкова розробка для пристроїв Android складалася з цього базового набору інструментів: SDK, IDE, мов програмування, бібліотек та плагінів. Давайте подивимося, з якими продуктами ви можете створити свій технічний пакет.

Інструменти / Набір для розробки програмного забезпечення. SDK - це набір інструментів, що включає виконувану програму. Це включає документацію, коректори помилок, симулятори, фреймворки, бібліотеки, засоби ідентифікації тощо. Пакет SDK для Android вже включений в Android Studio, але якщо ви хочете використовувати іншу IDE, ви можете завантажити його окремо внизу сторінки, на яку пов’язано посилання.

Редактори та IDE. Теоретично ви можете писати програми для Android у звичайному текстовому редакторі або командному рядку, але загальним підходом є використання IDE. Цей інструмент інтегрує всі інструменти SDK та допомагає керувати ними простіше та зручніше. Android Studio є офіційною IDE для Android, але інші варіанти також популярні. Eclipse - попередник Studio, який може використовувати плагіни для розширення коду на більше мов. IntelliJ IDEA - платна опція, але її можна легко налаштувати.

Мови програмування. Java та Kotlin вказані як офіційні мови програмування Android, але існують альтернативні варіанти. Ви також можете використовувати C і C ++ з Android Native Development Kit - інструментом для реалізації частин програми, попередньо записаних у власному коді. Існують також зовнішні інструменти, які дозволяють створювати власні програми для Android, використовуючи ваші улюблені мови, такі як Ruboto (Ruby) або Kivy (Python). Тільки пам’ятайте, що будь-які неофіційні рішення будуть відставати в плані оновлення.

Бібліотеки. Розробники програмного забезпечення використовують бібліотеки для всіх видів завдань. Вони є фрагментами попередньо написаного коду, які автоматизують завдання програміста і позбавляють від необхідності винаходити колесо. Спільнота Android щедро ставилася до таких безкоштовних рішень. Найпопулярніші - GSON для серіалізації та десеріалізації Java-об'єктів для зв'язку з API, модернізація для API та EventBus для зручного зв'язку між різними елементами програми. Щоб налаштувати всі ці параметри та впорядкувати процес додавання зовнішніх бібліотек, розробники використовують інструмент Gradle.

Доповнення. Хоча бібліотеки використовуються для автоматизації завдань проекту, плагіни створюються для розширення кожного програмного інструменту, IDE у нашому випадку. Можливо, ми не зможемо перерахувати їх усі, тож ознайомтесь із цією колекцією плагінів Android Studio та куратором списків плагінів Eclipse на BestPlugins.com.

Апаратна незалежність

Серед речей, необхідних для розробки додатків для Android, найменше турбують пристрої. Розробка Android здійснюється на Java, що робить процес крос-платформним. Android Studio, Eclipse, IntelliJ IDEA, Fabric та багато інших інструментів розробки Android можна використовувати та завантажувати в Windows, Mac OS та Linux. Створення програм для iOS вимагає використання Mac або віртуальної машини. Інші сторонні інструменти дозволяють інтегрувати NDK в Eclipse та Visual Studio.

Java і Kotlin як мови програмування

Android має дві офіційно підтримувані мови програмування - Java та Kotlin. Перша мова стала улюбленцем вже двох десятиліть і посіла п’яте місце серед найпопулярніших технологій у 2018 році. Java - це об’єктно-орієнтована, кроссплатформенна мова, яка використовується скрізь, починаючи від fintech-стартапів і закінчуючи ініціативами аналізу даних. Інтернет, настільні, мобільні та Інтернет-речі - все це може працювати на Java, що є перевагами та недоліками, які ми нещодавно виявили в окремому дописі в блозі. Популярність Java дозволяє легко знайти кваліфікованих розробників Android на ринку або навіть у вашій технічній команді.

Що стосується Котліна, це ще одна високо оцінена технологія. Це не зовсім сама по собі мова, а скоріше схожа на новий стиль кодування на основі Java. Це бере на себе всю складність і багатослів'я від Java і робить весь процес написання програми швидшим та приємнішим. Будь-який розробник Java може в короткий час освоїти Kotlin, але це не потрібно, якщо у вас немає трохи зайвого часу, щоб інвестувати в перехід до нового процесу. Ще одна хороша річ полягає в тому, що Джотлін і Кава (бачите, що ми тут зробили?) Повністю взаємодіючі і можуть вільно використовуватися одночасно.

Має сенс стверджувати, що Java і Kotlin - не єдині - хоча і офіційні - варіанти. Наприклад, Android Studio також підтримує C і C ++. Обидва вони складніші, ніж Java, але можуть бути корисними в певних ситуаціях, таких як ігрові програми. До того ж, якщо взяти до уваги багатоплатформні інструменти, набір мов ще більший - у Xamarin є C #, у PhoneGap - JavaScript, а в Flutter - Dart.

SDK також підтримує старіші версії платформи Android на випадок, якщо розробники хочуть націлити свої програми на старі пристрої.

Навчальні ресурси

Якщо вам доводиться мати справу з будь-якими документами Google, ви знаєте, про який рівень ми говоримо. Окрім величезної кількості інформації в Інтернеті, Google пропонує власний тренінг з розробки Android для початківців, досвідчених інженерів та навіть особливі випадки, що вимагають глибших знань.

Матеріали добре організовані за допомогою графіки, анімації та пояснювальних відео, а також деякі інтерактивні вправи.

На даний момент тренінг включає три потоки: Android Developer Essentials, Kotlin Boot Camp та власні програми з Flutter.

Більше того, програмісти можуть отримати офіційну сертифікацію розробників Google. Займає 8 годин, і перевіряє робочі знання людини на основі матеріалів для самостійного вивчення, наданих Google.

Flutter

Flutter - це новий безкоштовний SDK від Google, який дозволяє писати власні програми для Android та iOS за допомогою єдиної кодової бази. Незважаючи на те, що крос-платформні інструменти популярні на ринку вже давно, Google висловлює свою точку зору на зростаючий попит на його одноразовий метод програмування де завгодно.

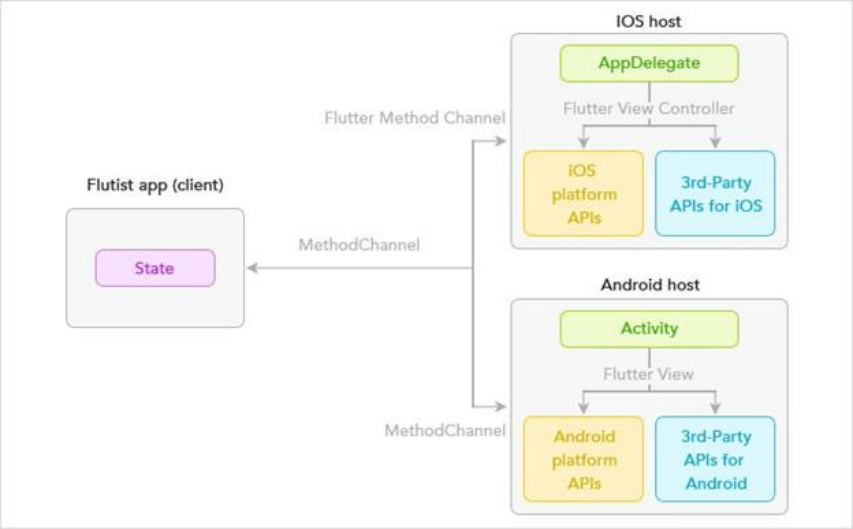

Рисунок 3.1 – Метод роботи Flutter.

Миттєві програми. Конференція розробників Android 2017 року була насичена новинками.

Подібно до прогресивних веб-програм, це швидші, легкодоступні та привабливі програми, які можуть надати доступ до обмеженої функціональності повного сервісу або надати користувачам попередній перегляд платного додатка.

До переваг використання Android відноситься.

Рис 3.2 – Ілюстрація переваг використання Android.

Мінуси використання Андроід

Проблеми безпеки. Хоча природа Android з відкритим кодом є благом для розробників, це також може бути великою проблемою.

Хоча природа Android з відкритим кодом є благом для розробників, це також може бути прокляттям. Хоча зловмисне програмне забезпечення та хакери націлені на мільйони користувачів Android майже щотижня, Google швидко випускає виправлення безпеки. На жаль, більшість людей не можуть регулярно оновлювати свої телефони. Це означає, що розробникам програм часто доводиться самостійно піклуватися про дані користувачів, виконуючи складне шифрування, включаючи додаткові функції безпеки, або взагалі уникаючи введення особистих даних.

Вибір мови програмування

Чи знаєте ви, скільки разів перевіряєте свій смартфон за день?

Ні? Ну, за даними журналу Accounting Magazine, це приголомшливі 86 разів на день (принаймні для нас, тисячоліть ... решта населення трохи більш осудна). Причиною нашої залежності від наших телефонів є захоплюючий сенсорний досвід, який вони вносять у наше повсякденне життя. Як це, запитаєте ви? Facebook, Instagram, Twitter, YouTube, WhatsApp

Яку мову вибрати?

Хоча Java є офіційною мовою Android, існує багато інших мов, які можна використовувати для розробки програм для Android. Детальна інформація про них наведена нижче, щоб допомогти прийняти обґрунтоване рішення.

3.2 Вибір середовища розробки

Java є офіційною мовою для розробки додатків для Android, і, отже, вона також є найбільш часто використовуваною мовою. Багато програм у магазині Play створено за допомогою Java, яка також є найбільш підтримуваною мовою Google. На додаток до всього, Java має чудове інтернет-співтовариство для підтримки у разі виникнення проблем (і повірте, будуть проблеми!).

Однак Java - це складна мова для початківців, оскільки вона містить складні теми, такі як конструктори, винятки з нульовим покажчиком, синхронізація, конкретні винятки тощо. Крім того, Android Software Development Kit (SDK) збільшує складність на новий рівень!

Загалом, Java - чудова мова, щоб випробувати всі задоволення від розробки додатків для Android. Однак це може трохи ускладнити початківців, які вважають за краще починати з чогось простішого, а потім повертатися до цього.

Kotlin

Kotlin - це міжплатформна мова програмування, яка може використовуватися як альтернатива Java для розробки програм для Android. Він також був представлений як "офіційна" вторинна мова Java у 2017 році. Kotlin може працювати з Java і працювати на віртуальній машині Java.

Єдина велика різниця полягає в тому, що Kotlin видаляє непотрібні функції Java, як виключення порожнього вказівника. Це також усуває необхідність закінчувати кожен рядок крапкою з комою. Коротше кажучи, Kotlin набагато простіший для початківців у порівнянні з Java, і його також можна використовувати як "точку входу" для розробки додатків для Android.

C ++

C ++ можна використовувати для розробки програм для Android за допомогою Android Native Development Kit (NDK). Однак додаток не може бути повністю побудований за допомогою C ++, а NDK використовується для реалізації частин програми в рідному коді C ++. Це допомагає використовувати бібліотеки коду C ++ програми за необхідності.

Хоча C ++ в деяких випадках корисний для розробки програм для Android, його важче налаштувати і набагато менш гнучким. Це також може призвести до більшої кількості помилок через підвищену складність. Тому краще використовувати Java порівняно з C ++, оскільки вона не дає достатнього прибутку, щоб компенсувати необхідні зусилля.

C #

C # дуже схожий на Java, тому ідеально підходить для розробки програм для Android. Як і Java, C # також виконує збір сміття, тому ймовірність витоків пам'яті зведена до мінімуму. C # також має чистішу та простішу архітектуру, ніж Java, що робить кодування в ній відносно простішим.

Раніше найбільшим недоліком C # було те, що він може працювати лише в системах Windows, оскільки вони використовують .NET Framework. Однак цією проблемою займався Xamarin.Android (раніше відомий як Mono для Android), який є крос-платформною реалізацією публічної інфраструктури мови. Тепер інструменти Xamarin.Android можна використовувати для написання власних програм для Android та обміну кодами на багатьох платформах.

Python

Python можна використовувати для розробки програм для Android, навіть якщо Android не підтримує власну розробку Python. Це можна зробити за

допомогою різних інструментів, які перетворюють програми Python в пакети Android, які можуть працювати на пристроях Android.

Прикладом цього є Kivy - бібліотека Python з відкритим кодом, яка використовується для розробки мобільних додатків. Він підтримує Android, а також заохочує швидку розробку додатків (а це безпрограшний варіант для мене!). Однак недоліком цього є те, що оригінальних переваг для Kivy не буде, оскільки він не підтримується локально.

Corona

Corona - це інструментарій для розробки програмного забезпечення, який можна використовувати для розробки додатків Android за допомогою Lua. Він має два режими роботи, Corona Simulator та Corona Native. Corona Simulator використовується для безпосереднього створення додатків, тоді як Corona Native використовується для інтеграції коду Lua з проектом Android Studio для створення програми з використанням власних функцій.

Хоча Lua трохи обмежений у порівнянні з Java, він також набагато простіший і має легшу криву навчання. Більше того, є вбудовані функції монетизації, а також різні активи та плагіни, які збагачують досвід розробки додатків. Corona в основному використовується для створення графічних додатків та ігор, але аж ніяк не обмежується цим.

HTML, CSS та JavaScript

Додатки Android можна створювати за допомогою HTML, CSS та JavaScript, використовуючи фреймворк Adobe PhoneGap на базі Apache Cordova. Фреймворк PhoneGap в основному дозволяє використовувати навички веб-розробки для створення змішаних програм, які відображаються через WebView, але групуються як додаток.

Хоча фреймворку Adobe PhoneGap достатньо для виконання основних завдань у галузі розробки додатків Android, для нього не потрібно багато програмування, крім JavaScript. І оскільки для створення гідної програми потрібно багато працювати, найкраще використовувати інші мови у цьому списку, якщо ви хочете, щоб вас називали справжнім розробником Android

Отже, існує багато додатків, таких як Chat messenger. Музичні та ігрові програвачі. Калькулятори. І т. Д., Які можна створити, використовуючи мови, згадані вище. Немає мови, яку можна назвати "правильною мовою" для розробки програм для Android. Тому ви несете відповідальність за правильний вибір мови на основі ваших цілей та уподобань для кожного окремого проекту.

А теперь детальніше про Java.

Мобільна версія Java називається Java ME. Java ME заснована на Java SE і підтримується більшістю смартфонів та планшетів. Java Platform Micro Edition (Java ME) забезпечує гнучке та безпечне середовище для створення та реалізації програм, орієнтованих на вбудовані пристрої та мобільні пристрої. Програми, побудовані на Java ME, портативні, безпечні та можуть використовувати переваги власних можливостей пристрою. Java ME розглядає обмеження, пов'язані зі створенням програм, орієнтованих на мобільні пристрої. По суті, Java ME вирішує проблему запуску програм на пристроях з малою пам’яттю, екраном та доступною потужністю.

Існують різні способи створення додатків для пристроїв Android, але рекомендованим є використання мови програмування Java та Android SDK.

Переваги використання Java.

Це добре працює як для локальних, так і для кроссплатформенних додатків. Оскільки сам Android побудований на Java, існує безліч бібліотек Java, які допоможуть вам. Крім того, Java має величезну екосистему з відкритим кодом. Програми Java легші та компактніші, навіть у порівнянні з програмами Kotlin, що призводить до швидшого використання програм. Java також пришвидшує процес створення, дозволяючи отримати більше коду за менший час. Завдяки прискореній компіляції з Gradle збирання великих проектів стає простішим у Java.

3.3 Додаткові інструменти

Маленькі проекти, які не користуються будь-якими зовнішніми бібліотеками, легко було б побудувати не прибігаючи до будь-яких зовнішніх інстументів збірки,

але коли з’являться медіа або текстові файли у програміста виникнуть проблеми з підтримкою всіх цих речей, від яких залежить програма. Заради поліпшення якості вихідного продукту, також, використання зовнішніх систем для збірки допоможе виконати цей процес у правильній послідовності, зберігаючи час та моральний стан розробника в нормі. В нашому випадку інструментом автоматизованої збірки виступатиме Gragle.

Gradle - це інструмент автоматизації будівель для розробки багатомовного програмного забезпечення. Контролює процес розробки завдання збірки та упаковки для тестування, розгортання та розгортання. Підтримувані мови включають Java (Kotlin, Groovy та Scala), C / C ++ та JavaScript.

Gradle спирається на концепції Apache Ant та Apache Maven та вводить специфічну для домену мову на основі Groovy та Kotlin, що контрастує з конфігурацією проекту на основі XML, що використовується Maven. Забезпечуючи управління залежностями, Gradle використовує векторизований періодичний графік, щоб визначити порядок виконання завдань.

Gradle призначений для багатопроектних будівель, які можуть перерости у великі. Він діє на основі ряду будівельних завдань, які можна виконувати як послідовно, так і паралельно. Додаткові конструкції підтримуються вказівкою вже модернізованих частин дерев будівель; Будь-яке завдання, яке покладається лише на ці частини, не потрібно повторювати. Він також підтримує кешування компонентів збірки, швидше за все, через спільну мережу за допомогою Gradle Build Cache. Він виробляє веб-візуалізацію будівель під назвою Gradle Build Scans. Програма розширюється для нових функцій та мов програмування за допомогою підсистеми плагіна.

Gradle поширюється як програмне забезпечення з відкритим кодом за ліцензією Apache 2.0 і вперше вийшов у 2007 році.

Запуск завдання збірки (gradle) призводить до консольного реєстру:

> gradle build

:compileJava

:processResources

:classes

:jar

:assemble

:compileTestJava

:processTestResources

:testClasses

:test

:check

:build

BUILD SUCCESSFUL

Плагін Java імітує кілька передбачуваних життєвих циклів Maven як завдання у неперіодичному графіку, орієнтованому на залежність, для вхідних та вихідних даних кожного завдання. У цьому простому випадку завдання побудови залежить від результату завдань перевірки та складання.

3.4 Розробка тестового додатку

3.4.1 Базові функції

Починаючи розробку мобільного додатку, для початку було б зручно розробити стартовий “workspace” додатку. Тобто створити стартову сцену, де б в разі підключення камери ми отримали всі необхідні доступи, потрібно, також підключити галерею як ще один, додатковий, інформаційний ресурс.

В нашому випадку алгоритм роботи з програмою буде наступний:

Встанолення додатку –> підтвердження надання всіх необхідних дозвілів –> Вибір опції сканування чеку –> Потім потрібно зробити фотографію, або відкрити вже існуюче зображення з галереї –> Додатково оптимізувати фотографію –> Переглянути результат.

Я думаю, що це буде найбільш раціанальним способ використання ниших можливостой щодо проектування додатку по скануванню чеків.

Створення кнопки, що відповідає за функцію сканування та підключаємо перевірку всіх дозвілів до неї, надаємо користувачу можливість знайти на обрізати, як йому треба, вхідне зображення. (код в Додатку А)

На рисунку 3.3 продемонстроване виконная перших етапів розробки нашої програми.

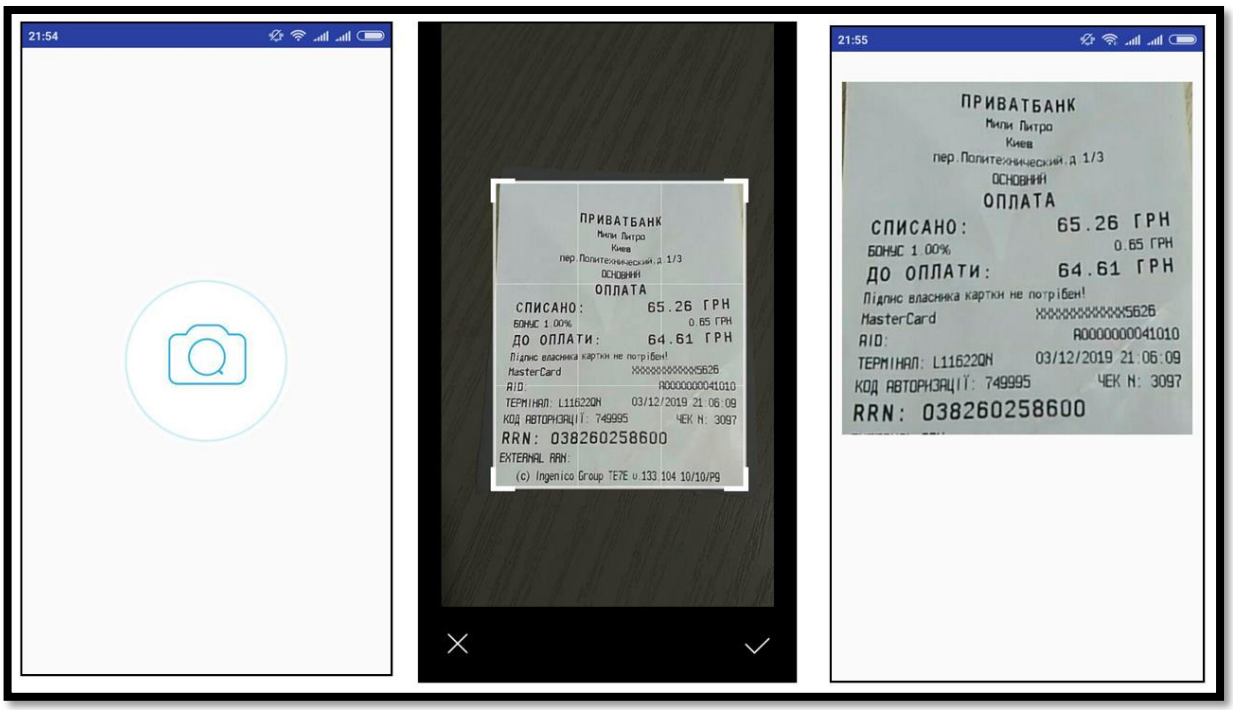

Рис 3.3 Стартове вікно/оптимізація/результат

3.4.2 Підключення Tesseract

Усе готово для підключення бібліотеки Tesseract. Оригінально Tesseract написаний на C/C++ але ізза своєї популярності, має «обгортки» навколо себе на усі популярні мови програмування. Так як я обрав Java як мову під Android платформу, тому я буду використовувати обгортку (JNA)

Tesseract під Java, а саме Tess4J.

Підключаємо Tess4J через Gradle. Він бере на вхід зображення та на вихід видає рохпізнаний текст, який виводимо на фінальне вікно. Спробуємо розпізнати звичайний текст.

На рисунку 3.4 буде продемонстровано, що тест пройдений успішно, і Tesseract впорався зі своєю роботою хоч і не ідеально, але добре. Протестуємо ще раз на інакшому текстовому форматі. Цього разу, хоч їх і не багато, такі помилки здаються мені доволі безглуздими. Я також перевірив цей текст через сервіс Google Vision, він використовує досконально інший підхід до розпізнавання.

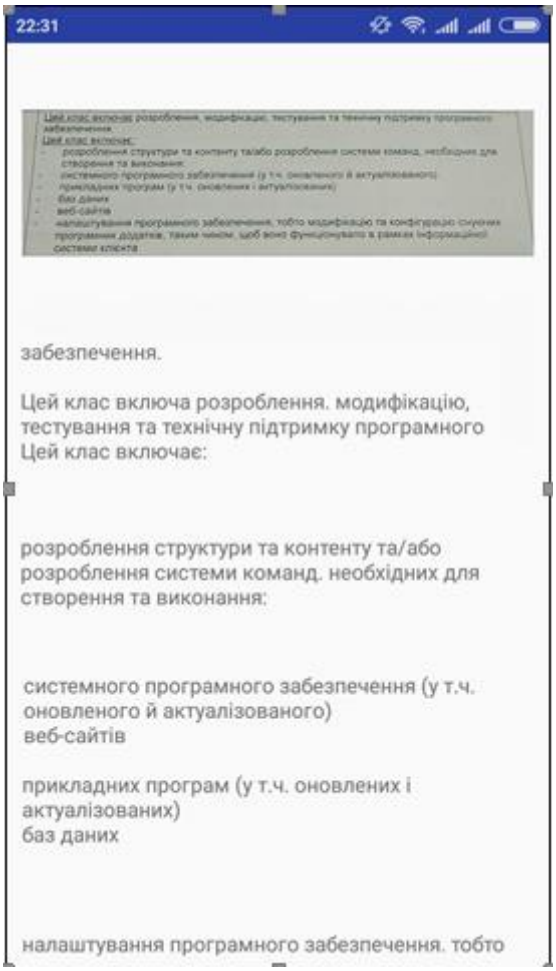

Рис 3.4 – обробка тестового зображення.

Але не дивлячись на це, помилки залишились і там. Задопомогою цього дослідження ми тепер можемо зробити висновок, що в готових бібліотеках україньсої мови алгоритми розпізнання працюють тільки на базі базового, основного, набору шрифтів, а через те, що людям найчастіше треба розпізнавати текст, написаний англійскою, його розпізнавання є найбільш розвиненим і точним, з підтримкою багатьох типів шрифтів. Але в випадку української мови (або будь-якої іншою не такої популярної мови), яка є не найбільш популярною мовою, має

місце меньший шанс розпізнавання, та меньший базовий набір шрифтів. Зрозумівши, що проблема помилок в розпізнаванні не пов’язана з вибором бібліотеки, а з самими алгоритмами розпізнавання. Давайте тепер протестуємо на чеках(рис3.5).

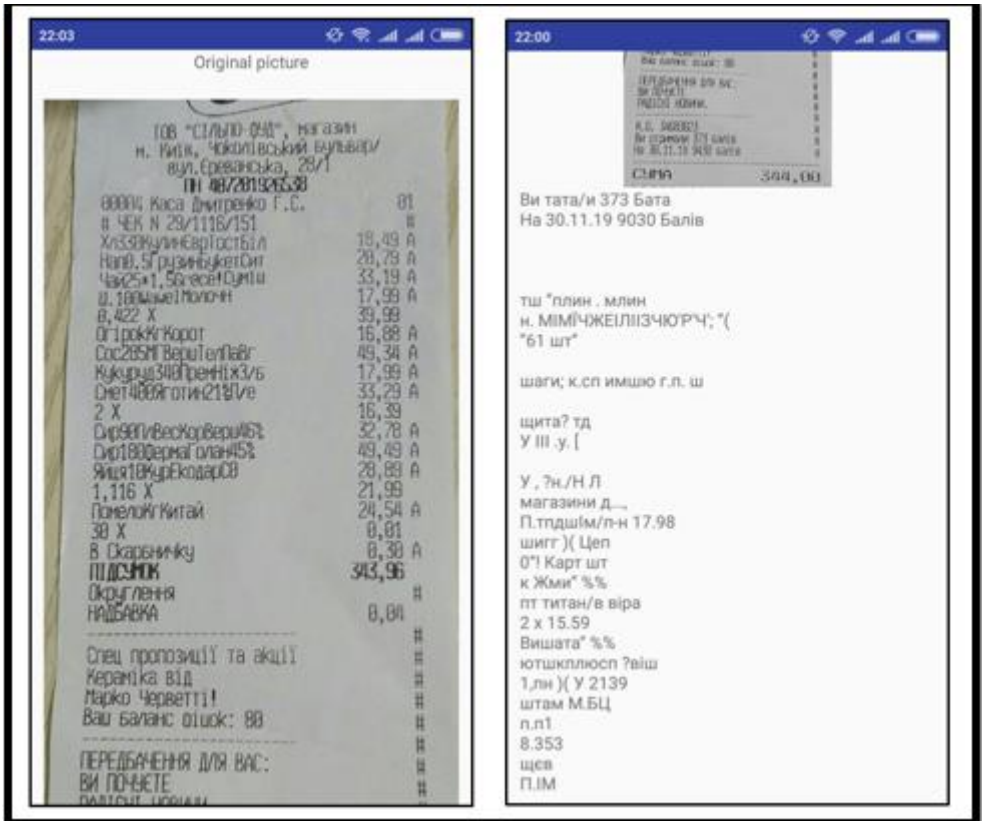

Рис 3.5 – Розпізнавання тексту з чека

За результатом тесту можна побачити, що цей формат чеку залишився майже не розпізнаним. Другий формат чеку(рис 3.6) розпізнається набаго краще, що дає надію на повне його розширування, але наразі, тільки посимвольно, нема жодного слова яке б розпізналося повністю. Це точно пов’язано з тим, що шрифт, на продемонстрованих чеках, вкрай відрізняється від типових шрифтів, що ми з вами звкили бачили. Ці шрифти є мінімалістичними, і через, літерам в них, бракує особливостей. Також в цих шрифтах значни менша відстань між символами, що також шкодить нормальному їз розпізнаванню.

Враховуючи, що другий формат краще розпізнаеються базовиим алгоритмами, було б нерозумно не взяти його як основу для подальшох роботи, і

будемо покращувати відсоток розпізнавання навчаючи Tesseract новим шрифтам, та, звичайно, додатковій обробці, фільтрації зображення перед його розпізнаванням.

Рис 3.6 – Виявлення кращого шрифта для подальшого позпізнавання.

3.4.3 Тренування Tesseract та Grayscale фільтр

Попередня обробка зображення є важливою складовою систем розпізнавання символів, бо яка різниця скільки мов і шрифтів може розпізнати ваша система, якщо вона не зможе розпізнавати текст, що надрукований на частково підгорілому папері, або якщо серед тексту буде якась інша проблема, наприклад, ви пролили воду на документ.

Тому для оптимізації розпізнавання необхідно додати фільтр. Я додав Grayscale фільтр, який переводить кольорове зображення у чорно-білу палітру кольорів(рис 3.7), з додатковим затемненням або навпаки висвітлюванням проблемних зон.

Наступним кроком навчання Tesseract’а шрифтам, що широко використовуються в Україні для друка чеків, мені здається, шрифт

Courier new для Української мови підійде ідеально.

Рис 3.7 – чорно-біла палітра Grayscale

Цей процес є монотонним і є доволі часозатратним.

Для початку нам необхідно створити тестове зображення. Бажано щоб цей шаблон того тексту, який ми потім будемо змушені розпізнавати. Важливим є той момент, щоб кожен символ відсканового тексту зустрічався неодноразово, а краще – разів 25. Формат цього файлу має бути .tiff, без зжимання, також бажано щоб цей файл не був багатосторінковим. Кожен символ має бути чітко відокремлений від іншого. Кладемо наше зображення в окрему директорію і називаємо у вигляді <назва шрифту>. exp <номер> .<код мови>.tif. Зображення має бути не одно, але відрізнятись мають тільки номером у названому файлі. Формат найменувать є дуже важдивим, тощу що файли з невірними назвами утіліти, котрі ми могли б використати, не будуть працювати. Назва шрифту – couriernew і код мови – ukr, отже файл названо couriernew.exp0.ukr.tiff .

Наступним кроком буде створення box-файлів щоб мати можливість точно визначити символи на зображенні та задати їх відповідність в utf-8. Вони являються звичайними текстовими файлами, в яких кожному символу відповідає рядок з символом і координатами прямокутника в пікселях.

Для початку генеруємо файл утілітою з пакета tesseract:

tesseract.couriernew.exp0.ukr.tiff couriernew.exp0.ukr.tiff batch.nochop.makebox

Після цього ми отримаємо файл *.couriernew.exp0.ukr.tiff.box* в поточній директорії. Перевіримо результати. В разі, якщо символи повністю відповідають символам в файлі - то тренувати нашу нейронку більшене потрібно, роботу виконано. В нашому ж випадку не збігатиметься ні суть документу, ні навіть кількість літер в ньому. Тобто tesseract зі стандартним словником не тільки не розпізнав символи, ще й порахував деякі з них за два або навіть більше. Ще може бути таке, що деякі символи будуть розпізнані як один. Всі ці помилки потрібно виправити перед наступним кроком. Робота є монотонною і часозатратною, але нам пощастило, і для цього можна використати сторонній софт. Я використаю jTessBoxEditor (рис 3.8). Відкриваємо за допомогою нього box-file та виправляємо всі помилки.

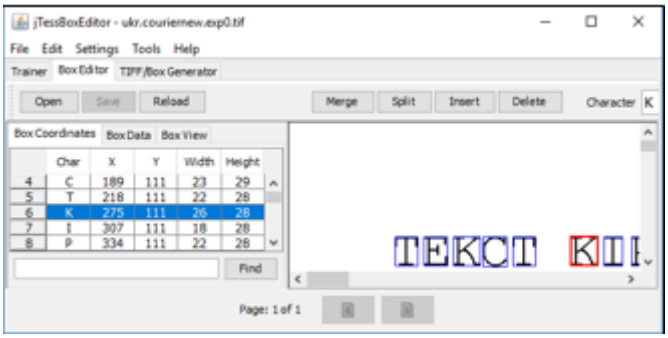

Рис 3.8 – jTessBoxEditor.

Через деякий, доволі довгий, проміжуток часу ми маємо корректний box-file. Ми готові до роботи на наступному етапі.

Тепер тренуємо ним Tesseract:

tesseract.couriernew.exp0.ukr.tiff couriernew.exp0.ukr.tiff batch.nochop.boxtrain

Почнемо пеший етап тренування.

Отримуємо доволі багато помилок, але наша увага прикута до "Found 105 good blobs". Якщо ця цифра істотно більша числа досліджвуваних символів, то цей етап тренування можна вважати вдалим. Якщо ні – повертоємося на початок і навчаємо нашу нейронку знову. По завершенню цього кроку у нас буде файл, *couriernew.exp0.ukr.tr*

Витягуємо звідти наш набір символів:

unicharset_extractor couriernew.exp0.ukr.box

Отримуємо набір символів у вигляді файлу unicharset в поточній директорії, де кожен символ і його ознаки розташовуються в окремому рядку. Зараз нашим завданням буде перевірити і поправити характеристики символів. Для маленьких букв алфавіту ставимо 3, для великих 5, для розділових знаків 10 для чисел 8, все інше помічаємо 0.

Описуємо стиль шрифту

Створюємо файл ukr.font_properties з єдиною рядком: couriernew 0 0 0 0 0. Тут спочатку пишемо ім'я шрифту, потім числом 1 або 0 помічаємо наявність у символів стилю.

Для подальшого навчання нам знадобитися виконати ще три операції.

shapeclustering -F ukr.font_properties -U unicharset couriernew.exp0.ukr.tr

з'явиться файл shapetable

а потім:

mftraining -F ukr.font_properties -U unicharset -O ukr.unicharset unicharset.couriernew.exp0.ukr

отримаємо файли ccc.unicharset, inttemp, pffmtable

і нарешті:

cntraining ukr.couriernew.exp0.tr

отримаємо файл normproto.

В подальшому нам будуть потрібні ці файли.

В результаті отримуємо файл ukr.traineddata, який і дозволить нам краще розпізнавати текст. Додаємо його в список traineddata Tess4J.

В результаті отримуємо файл ukr.traineddata, який і дозволить нам краще розпізнавати текст. Додаємо його в список traineddata Tess4J.

Тепер можна спробувати протестувати додаток ще раз.

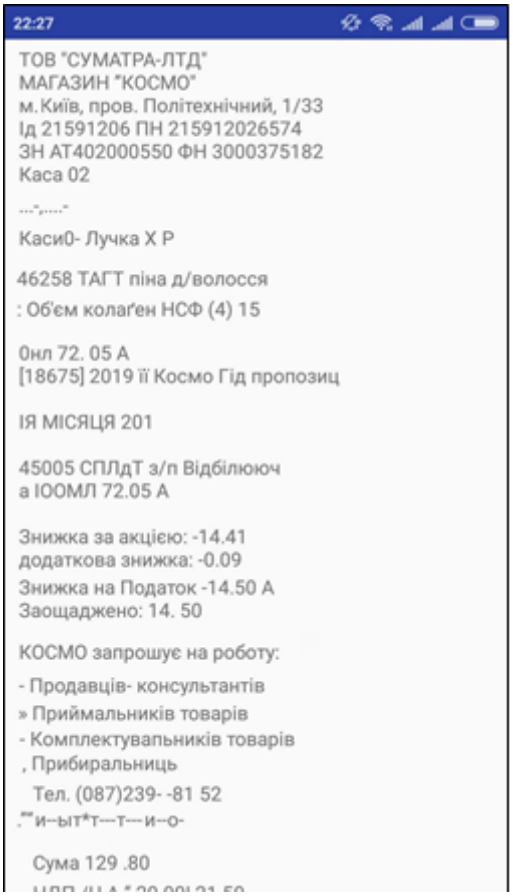

Рис 3.9 – Сканування чеку після тренування HP.

Як видно на рисунку 3.9, тепер наша програма працює як треба. Помилок не було виявлено і ми може сказати, що впоралился з поставленною задачею.

Висновки до розділу 3

В процесі розробки мобільного додатку виникли специфічні проблеми при навчанні використовуємої нейронної мережі, а також розпізнаванні різних шрифтів.

Після тренування Tesseract новими шрифтами та після додавання додаткового фільтру, вдалось реалізувати розпізнавання тексту на чеках та виділення з нього основної інформацію, що і було основною ціллю тестового додатка. Нажаль, успішно розпізнати всі чеки на нинішньому етапі розробки тестового додатку не можливо, бо часто чеки дуже відрізняються за шрифтом та форматом друку.

Розроблений тестовий додаток може бути покращеним шляхом навчання Tesseract OCR новим шрифтам української мови.

ВИСНОВКИ

Метою даної магістерської роботи є вивчення OCR систем та пристосування таких систем для використання у мобільному додатку для моніторингу особистого бюджету.

У першому розділі було описано проблему яку має вирішити розроблений програмний продукт. Було проведено детальний огляд та розібрано основний принцип роботи OCR систем та їх компоненти.

У другому розділі було детально розібрано готові OCR рішення, їх відмінності та особливості. Наведені приклади використання OCR систем для різного виду програм та функціоналу.

У третьому розділі було описано технології та інструменти які будуть використовуватися. Було проведено розробку тестового додатку для сканування чеків на базі Tesseract OCR. Додаток розроблявся на платформу Android з використанням мови програмування Java. Було детально розібрано проблеми, що виникнули у ході розробки, та знайдені способи їх вирішення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ExplainThatStuff [Електронний ресурс] – Режим доступу до ресурсу: https://www.explainthatstuff.com/how-ocr-works.html.

2. Line Eikvil “Optical Character Recognition” – 54 с.

3. Порівняння програмного забезпечення для оптичного розпізнавання символів [Електронний ресурс] – Режим доступу до ресурсу: https://source.opennews.org/articles/so-many-ocr-options/

4. Ray Smith, Google Inc. “An Overview of the Tesseract OCR Engine” – 136-138с.

5. Tutorialspoint: Android - Overview [Електронний ресурс] – Режим доступу до ресурсу : https://www.tutorialspoint.com/android/android_overview.html

6. Gradle [Електронний ресурс] – Режим доступу до ресурсу: https://pspdfkit.com/blog/2019/understand-gradle-build-system

7. Регулярний вираз [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-reference

ДОДАТОК А
ЛІСТИНГ СИСТЕМИ

Build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28

    defaultConfig {
        applicationId "com.wt.ocr"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 2
        versionName "1.1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('androidx.test.espresso:espresso-core:3.1.0', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })

    implementation 'androidx.appcompat:appcompat:1.1.0'
    testImplementation 'junit:junit:4.12'
    implementation 'com.edmodo:cropper:1.0.1'
    implementation 'com.rmtheis:tess-two:6.1.1'
```

```
    implementation 'com.google.android.gms:play-services-analytics:17.0.0'
}
apply plugin: 'com.google.gms.google-services'
```

App.java

```
package com.wt.ocr;

import android.app.Application;
import com.google.android.gms.analytics.GoogleAnalytics;
import com.google.android.gms.analytics.Tracker;

public class App extends Application {
    private static GoogleAnalytics sAnalytics;
    private static Tracker sTracker;

    @Override
    public void onCreate() {
        super.onCreate();
        sAnalytics = GoogleAnalytics.getInstance(this);
    }

    synchronized public Tracker getDefaultTracker() {
        if (sTracker == null) {
            sTracker = sAnalytics.newTracker(R.xml.global_tracker);
        }
        return sTracker;
    }
}
```

BaseActivity.java

```
public class BaseActivity extends AppCompatActivity {
    private Tracker mTracker;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        App application = (App) getApplication();
        mTracker = application.getDefaultTracker();
    }

    public Tracker getTracker() {
        return mTracker;
    }
}
```

MainActivity.java

```
package com.wt.ocr;

public class MainActivity extends BaseActivity implements View.OnClickListener {
    private      Context context;
    private ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        context = this;
        imageView = findViewById(R.id.btn_camera);
        imageView.setOnClickListener(this);

        new Thread(new Runnable() {
            @Override
            public void run() {
                deepFile("tessdata");
            }
        }).start();
        sendScreenImageName();
    }

    private static final int PERMISSIONS_REQUEST_CAMERA        = 454;
    private static final int PERMISSIONS_REQUEST_WRITE_STORAGE = 455;
```

```
    static final String  PERMISSION_CAMERA       = Manifest.permission.CAMERA;
    static final String PERMISSION_WRITE_STORAGE = Manifest.permission.WRITE_EXTERNAL_STORAGE;

    public void onClick(View view) {
        if (view.getId() == R.id.btn_camera) {
            checkSelfPermission();
            getTracker().send(new HitBuilders.EventBuilder()
                    .setCategory("Action")
                    .setAction("Take a photo")
                    .build());
        }
    }

    void checkSelfPermission() {
        if (ContextCompat.checkSelfPermission(this, PERMISSION_CAMERA) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new String[]{PERMISSION_CAMERA}, PERMISSIONS_REQUEST_CAMERA);
        } else if (ContextCompat.checkSelfPermission(this, PERMISSION_WRITE_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new String[]{PERMISSION_WRITE_STORAGE}, PERMISSIONS_REQUEST_WRITE_STORAGE);
        } else {
            Intent intent = new Intent(context, TakePhoteActivity.class);
            startActivity(intent);
        }
    }

    public void deepFile(String path) {
        String newPath = getExternalFilesDir(null) + "/";
        try {
            String str[] = getAssets().list(path);
            if (str.length > 0) {
                File file = new File(newPath + path);
```

```
                file.mkdirs();
                for (String string : str) {
                    path = path + "/" + string;
                    deepFile(path);
                    path = path.substring(0, path.lastIndexOf('/'));
                }
            } else {
                InputStream is = getAssets().open(path);
                FileOutputStream fos = new FileOutputStream(new File(newPath + path));
                byte[] buffer = new byte[1024];
                while (true) {
                    int len = is.read(buffer);
                    if (len == -1) {
                        break;
                    }
                    fos.write(buffer, 0, len);
                }
                is.close();
                fos.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void sendScreenImageName() {
        getTracker().setScreenName("Activity-" + "Home");
        getTracker().send(new HitBuilders.ScreenViewBuilder().build());
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
```

```
        if (requestCode == PERMISSIONS_REQUEST_CAMERA || requestCode ==
PERMISSIONS_REQUEST_WRITE_STORAGE) {
            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                Intent intent = new Intent(context, TakePhoteActivity.class);
                startActivity(intent);
            } else {
                Toast.makeText(context, "Please open the permission",
Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

TakePhotoActivity.java

```
public class TakePhotoActivity extends AppCompatActivity implements
CameraPreview.OnCameraStatusListener, SensorEventListener {

    private Context context;
    public static final boolean isTransverse = true;
    private static final String TAG      = "TakePhoteActivity";
    public static final Uri IMAGE_URI =
MediaStore.Images.Media.EXTERNAL_CONTENT_URI;
    private String PATH;
    private CameraPreview  mCameraPreview;
    private CropImageView  mCropImageView;
    private RelativeLayout mTakePhotoLayout;
    private LinearLayout   mCropperLayout;
    private ImageView      btnClose;
    private ImageView      btnShutter;
    private Button         btnAlbum;
```

```
        private ImageView      btnStartCropper;
        private ImageView      btnCloseCropper;
        private boolean isRotated = false;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            requestWindowFeature(Window.FEATURE_NO_TITLE);
            setContentView(R.layout.activity_take_phote);
            getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

            context = this;
            PATH = getExternalCacheDir() + "/AndroidMedia/";

            btnClose = findViewById(R.id.btn_close);
            btnClose.setOnClickListener(onClickListener);
            btnShutter = findViewById(R.id.btn_shutter);
            btnShutter.setOnClickListener(onClickListener);
            btnAlbum = findViewById(R.id.btn_album);
            btnAlbum.setOnClickListener(onClickListener);

            btnStartCropper = findViewById(R.id.btn_startcropper);
            btnStartCropper.setOnClickListener(cropcper);
            btnCloseCropper = findViewById(R.id.btn_closecropper);
            btnCloseCropper.setOnClickListener(cropcper);

            mTakePhotoLayout = findViewById(R.id.take_photo_layout);
            mCameraPreview = findViewById(R.id.cameraPreview);
            FocusView focusView = findViewById(R.id.view_focus);

            mCropperLayout = findViewById(R.id.cropper_layout);
            mCropImageView = findViewById(R.id.CropImageView);
            mCropImageView.setGuidelines(2);

            mCameraPreview.setFocusView(focusView);
```

```
    mCameraPreview.setOnCameraStatusListener(this);

    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mAccel = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

}

@Override
protected void onResume() {
    super.onResume();
    if (isTransverse) {
        if (!isRotated) {
            TextView tvHint = findViewById(R.id.hint);
            ObjectAnimator animator = ObjectAnimator.ofFloat(tvHint, "rotation", 0f, 90f);
            animator.setStartDelay(800);
            animator.setDuration(500);
            animator.setInterpolator(new LinearInterpolator());
            animator.start();

            ImageView btnShutter = findViewById(R.id.btn_shutter);
            ObjectAnimator animator1 = ObjectAnimator.ofFloat(btnShutter, "rotation", 0f, 90f);
            animator1.setStartDelay(800);
            animator1.setDuration(500);
            animator1.setInterpolator(new LinearInterpolator());
            animator1.start();

            View view = findViewById(R.id.crop_hint);
            AnimatorSet animSet = new AnimatorSet();
            ObjectAnimator animator2 = ObjectAnimator.ofFloat(view, "rotation", 0f, 90f);
            ObjectAnimator moveIn = ObjectAnimator.ofFloat(view, "translationX", 0f, -50f);
            animSet.play(animator2).before(moveIn);
            animSet.setDuration(10);
            animSet.start();

            ObjectAnimator animator3 = ObjectAnimator.ofFloat(btnAlbum, "rotation", 0f, 90f);
```

```
            animator3.setStartDelay(800);
            animator3.setDuration(500);
            animator3.setInterpolator(new LinearInterpolator());
            animator3.start();
            isRotated = true;
        }
    } else {
        if (!isRotated) {
            View view = findViewById(R.id.crop_hint);
            AnimatorSet animSet = new AnimatorSet();
            ObjectAnimator animator2 = ObjectAnimator.ofFloat(view, "rotation", 0f, 90f);
            ObjectAnimator moveIn = ObjectAnimator.ofFloat(view, "translationX", 0f, -50f);
            animSet.play(animator2).before(moveIn);
            animSet.setDuration(10);
            animSet.start();
            isRotated = true;
        }
    }
    mSensorManager.registerListener(this, mAccel, SensorManager.SENSOR_DELAY_UI);
}

@Override
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    Log.e(TAG, "onConfigurationChanged");
    super.onConfigurationChanged(newConfig);
}

private View.OnClickListener onClickListener = new View.OnClickListener() {
    @Override
```

```
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.btn_close:
                finish();
                break;
            case R.id.btn_shutter:
                if (mCameraPreview != null) {
                    mCameraPreview.takePicture();
                }
                break;
            case R.id.btn_album:
                Intent intent = new Intent();
                intent.setType("image/*");
                intent.setAction(Intent.ACTION_GET_CONTENT);
                startActivityForResult(intent, 1);
                break;
        }
    }
};

private View.OnClickListener cropcper = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.btn_closecropper:
                showTakePhotoLayout();
                break;
            case R.id.btn_startcropper:
                Bitmap cropperBitmap = mCropImageView.getCroppedImage();

                Bitmap bitmap;
                bitmap = Utils.rotate(cropperBitmap, -90);

                long dateTaken = System.currentTimeMillis();
```

```
                String filename = DateFormat.format("yyyy-MM-dd kk.mm.ss", dateTaken).toString() + ".jpg";
                Uri uri = insertImage(getContentResolver(), filename, dateTaken, PATH, filename, bitmap, null);

                Intent intent = new Intent(context, ShowCropperedActivity.class);
                intent.setData(uri);
                intent.putExtra("path", PATH + filename);
                intent.putExtra("width", bitmap.getWidth());
                intent.putExtra("height", bitmap.getHeight());
                startActivity(intent);
                bitmap.recycle();
                finish();
                break;
            }
        }
    };

    @Override
    public void onCameraStopped(byte[] data) {
        Log.i("TAG", "==onCameraStopped==");
        Bitmap bitmap = BitmapFactory.decodeByteArray(data, 0, data.length);
        if (!isTransverse) {
            bitmap = Utils.rotate(bitmap, 90);
        }
        long dateTaken = System.currentTimeMillis();
        String filename = DateFormat.format("yyyy-MM-dd kk.mm.ss", dateTaken).toString() + ".jpg";
        Uri source = insertImage(getContentResolver(), filename, dateTaken, PATH, filename, bitmap, data);

        bitmap = Utils.rotate(bitmap, 90);
        mCropImageView.setImageBitmap(bitmap);
        showCropperLayout();
    }
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK) {
        Uri uri = data.getData();
        Log.e("uri", uri.toString());
        ContentResolver cr = this.getContentResolver();
        try {
            Bitmap bitmap = BitmapFactory.decodeStream(cr.openInputStream(uri));
            bitmap = Utils.rotate(bitmap, 90);
            mCropImageView.setImageBitmap(bitmap);
        } catch (Exception e) {
            Log.e("Exception", e.getMessage(), e);
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
    showCropperLayout();
}

private Uri insertImage(ContentResolver cr, String name, long dateTaken,
 String directory, String filename, Bitmap source, byte[] jpegData) {
    OutputStream outputStream = null;
    String filePath = directory + filename;

    try {
        File dir = new File(directory);
        if (!dir.exists()) {
            dir.mkdirs();
        }
        File file = new File(directory, filename);
        if (file.createNewFile()) {
            outputStream = new FileOutputStream(file);
            if (source != null) {
                source.compress(Bitmap.CompressFormat.JPEG, 100, outputStream);
            } else {
```

```
                outputStream.write(jpegData);
            }
        }
    } catch (IOException e) {
        Log.e(TAG, e.getMessage());
        return null;
    } finally {
        if (outputStream != null) {
            try {
                outputStream.close();
            } catch (Throwable t) {
            }
        }
    }

    ContentValues values = new ContentValues(7);
    values.put(MediaStore.Images.Media.TITLE, name);
    values.put(MediaStore.Images.Media.DISPLAY_NAME, filename);
    values.put(MediaStore.Images.Media.DATE_TAKEN, dateTaken);
    values.put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg");
    values.put(MediaStore.Images.Media.DATA, filePath);
    return cr.insert(IMAGE_URI, values);
}

private void showTakePhotoLayout() {
    mTakePhotoLayout.setVisibility(View.VISIBLE);
    mCropperLayout.setVisibility(View.GONE);
}

private void showCropperLayout() {
    mTakePhotoLayout.setVisibility(View.GONE);
    mCropperLayout.setVisibility(View.VISIBLE);
    mCameraPreview.start();
}
```

```
private float        mLastX       = 0;
private float        mLastY       = 0;
private float        mLastZ       = 0;
private boolean      mInitialized = false;
private SensorManager mSensorManager;
private Sensor       mAccel;

@Override
public void onSensorChanged(SensorEvent event) {

   float x = event.values[0];
   float y = event.values[1];
   float z = event.values[2];
   if (!mInitialized) {
      mLastX = x;
      mLastY = y;
      mLastZ = z;
      mInitialized = true;
   }
   float deltaX = Math.abs(mLastX - x);
   float deltaY = Math.abs(mLastY - y);
   float deltaZ = Math.abs(mLastZ - z);

   if (deltaX > 0.8 || deltaY > 0.8 || deltaZ > 0.8) {
      mCameraPreview.setFocus();
   }
   mLastX = x;
   mLastY = y;
   mLastZ = z;
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
```

```
}
```

ShowCroppedActiviry.java

```
public class ShowCropperedActivity extends AppCompatActivity {

    private             Context context;

    private static      String  LANGUAGE_PATH = "";

    private static final String  LANGUAGE     = "ukr2+eng";

    private static final String    TAG = "ShowCropperedActivity";
    private             ImageView imageView;
    private             ImageView imageView2;
    private             TextView  textView;

    private int    width;
    private int    height;
    private Uri    uri;
    private String result;

    private TessBaseAPI    baseApi = new TessBaseAPI();
    private Handler        handler = new Handler();
    private ProgressDialog dialog;

    private ColorMatrix colorMatrix;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_show_croppered);
        context = this;
        LANGUAGE_PATH = getExternalFilesDir("") + "/";
        Log.e("---------", LANGUAGE_PATH);

        width = getIntent().getIntExtra("width", 0);
        height = getIntent().getIntExtra("height", 0);
        uri = getIntent().getData();

        initView();
        initTess();
    }

    private void initView() {
        imageView = findViewById(R.id.image);
        imageView2 = findViewById(R.id.image2);
        textView = findViewById(R.id.text);

        dialog = new ProgressDialog(context);
        dialog.setMessage("Recognizing...");
        dialog.setCancelable(false);
        dialog.show();

        if (width != 0 && height != 0) {
            int screenWidth = Utils.getWidthInPx(this);
            float scale = (float) screenWidth / (float) width;
            final ViewGroup.LayoutParams lp = imageView.getLayoutParams();
            int imgHeight = (int) (scale * height);
            lp.height = imgHeight;
            imageView.setLayoutParams(lp);
            Log.e(TAG, "imageView.getLayoutParams().width:" +
imageView.getLayoutParams().width);
        }
        imageView.setImageURI(uri);
    }
```

```
    private void initTess() {
        baseApi.init(LANGUAGE_PATH, LANGUAGE);
        baseApi.setPageSegMode(TessBaseAPI.PageSegMode.PSM_AUTO);
        Thread myThread = new Thread(runnable);
        myThread.start();
    }

    private Bitmap getBitmapFromUri(Uri uri) {
        try {
            return MediaStore.Images.Media.getBitmap(this.getContentResolver(), uri);
        } catch (Exception e) {
            Log.e("[Android]", e.getMessage());
            Log.e("[Android]", "Directory is : " + uri);
            e.printStackTrace();
            return null;
        }
    }

    public Bitmap convertGray(Bitmap bitmap3) {
        colorMatrix = new ColorMatrix();
        colorMatrix.setSaturation(0);
        ColorMatrixColorFilter filter = new ColorMatrixColorFilter(colorMatrix);

        Paint paint = new Paint();
        paint.setColorFilter(filter);
        Bitmap result = Bitmap.createBitmap(bitmap3.getWidth(), bitmap3.getHeight(),
Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(result);

        canvas.drawBitmap(bitmap3, 0, 0, paint);
        return result;
    }

    private Bitmap binaryzation(Bitmap bitmap22, int tmp) {
```

```
int width = bitmap22.getWidth();
int height = bitmap22.getHeight();
Bitmap bitmap;
bitmap = bitmap22.copy(Bitmap.Config.ARGB_8888, true);
for (int i = 0; i < width; i++) {
    for (int j = 0; j < height; j++) {
        int pixel = bitmap.getPixel(i, j);
        int alpha = pixel & 0xFF000000;
        int red = (pixel & 0x00FF0000) >> 16;
        int green = (pixel & 0x0000FF00) >> 8;
        int blue = pixel & 0x000000FF;
        if (red > tmp) {
            red = 255;
        } else {
            red = 0;
        }
        if (blue > tmp) {
            blue = 255;
        } else {
            blue = 0;
        }
        if (green > tmp) {
            green = 255;
        } else {
            green = 0;
        }

        int gray = (int) ((float) red * 0.3 + (float) green * 0.59 + (float) blue * 0.11);
        if (gray <= 95) {
            gray = 0;
        } else {
            gray = 255;
        }
        int newPiexl = alpha | (gray << 16) | (gray << 8) | gray;
        bitmap.setPixel(i, j, newPiexl);
```

```
            }
        }
        return bitmap;
    }

    private Runnable runnable = new Runnable() {
        @Override
        public void run() {
            final Bitmap bitmap_1 = convertGray(getBitmapFromUri(uri));
            baseApi.setImage(bitmap_1);
            result = baseApi.getUTF8Text();
            baseApi.end();
            handler.post(new Runnable() {
                @Override
                public void run() {
                    imageView2.setImageBitmap(bitmap_1);
                    textView.setText(result);
                    dialog.dismiss();
                }
            });
        }
    };
}
```

ДОДАТОК Б
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ