КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

Система автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків з використанням нейронних мереж

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконав:
студент 2 курсу магістратури
групи ТШІ-21

_____Галька Дмитро Романович_____
(ПІБ)

Науковий керівник:

_____Доманецька Ірина Миколаївна_____
(ПІБ)
_____к. т. н., доцент_____
(науковий ступінь, вчене звання)

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № ____ від «____» травня 2020 р.

В.о. зав. кафедри _____ доц. Красовська Г.В.
підпис

Київ 2020

Перелік умовних позначень та скорочень

HTML – Hypertext Markup Language, Мова розмітки гіпертексту.

CSS – Cascading Style Sheets, Каскадні таблиці стилів.

UTF – Unicode Transformation Format, Формат перетворення Unicode.

OCR – Optical character recognition, Оптичне розпізнавання символів.

HP – Hewlett-Packard, технологічна компанія в США.

PDF – Portable Document Format, Формат портативного документа

DPI – Dots per inch, кількість точок на дюйм.

DOM – Document Object Model, Модель об'єктів документа

VGG - Visual Geometry Group, Група візуальної геометрії

GUI – Graphical User Interface, Графічний користувацький інтерфейс

VGT – Visual GUI Testing, візуальне тестування графічних інтерфейсів

РНМ - Рекуре́нтні нейро́нні мере́жі

ДКЧП - До́вга короткоча́сна па́м'ять

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаної літератури із 24 джерел та 1 додатку. Загальний обсяг роботи 63 сторінки. Робота містить 3 таблиці та 27 рисунків.

Актуальність теми полягає в наявності потреби в удосконаленні систем візуального тестування в роботі з текстом зі згладжуванням та доволі стрімкому розповсюдженні нейронних мереж в усіх сферах життєдіяльності.

Об’єктом дослідження є технології автоматизованого візуального тестування інтерфейсів застосунків.

Предметом дослідження є методи візуального тестування інтерфейсів застосунків, що містять текстові компоненти, на основі нейронних мереж.

Метою роботи є створення системи автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків.

Наукова новизна полягає в застосуванні нейронних мереж в системі автоматизованого візуального тестування для розпізнавання текстів відображених зі згладжуванням.

Практична цінність полягає в можливості роботи системи автоматизованого візуального тестування з документами з різним налаштуванням згладжування тексту.

Здійснено порівняльний аналіз різних підходів порівняння зображень як складової запропонованої системи автоматизованого візуального тестування.

Система проводить тестування в декілька етапів: виділення тексту, порівняння зображення без тексту, порівняння тексту та отримання загального результату. Згорткова нейронна мережа дає змогу виділити текст на зображенні та провести розпізнавання виділеного тексту. Ціллю є виділення границь символу та його вилучення з загального зображення.

Реалізація данної концепції була імплементована та протестована на відкритому наборі даних. Виділення тексту окремо від зображення та їх аналіз окремо один від одного показало більш точні результати з меншою кількістю помилок на контурі символів. Подальше дослідження може включати використання більш досконалих моделей для виділення тексту, а також розробка більш складних метрик для оцінки результатів подібних моделей.

Ключові слова: візуальне тестування, розпізнавання тексту, згорткові нейронні мережі, згладжування шрифтів, автоматичне тестування.

ABSTRACT

Work consists of an introduction, 3 sections, conclusions, list of used literature from 24 sources and 1 appendix. Total volume works 63 pages. The work contains 3 tables and 27 figures.

The relevance of the topic lies in the need to improve the systems of visual testing in working with text with smoothing and fairly rapid spread of neural networks in all spheres of life.

The object of research is the technology of automated visual testing of application interfaces.

The subject of research is methods of visual testing of application interfaces containing text components based on neural networks.

The aim of the work is to create a system of automated visual testing of text components of application interfaces.

The scientific novelty is the use of neural networks in the system of automated visual testing to recognize texts displayed with anti-aliasing.

The practical value lies in the ability of the automated visual testing system to work with documents with different text smoothing settings.

A comparative analysis of different approaches to comparing images as part of the proposed system of automated visual testing.

The system performs testing in several stages: text selection, image comparison without text, text comparison and obtaining the overall result. A convolutional neural network allows you to select text in an image and recognize selected text. The goal is to highlight the boundaries of the character and remove it from the overall image.

The implementation of this concept was implemented and tested on an open data set. Selecting text separately from the image and analyzing them separately from each other showed more accurate results with fewer errors in the outline of the

characters. Further research may include the use of more sophisticated models for text selection, as well as the development of more sophisticated metrics to evaluate the results of such models.

Key words: visual testing, text recognition, convolutional neural networks, font smoothing, automatic testing.

ЗМІСТ

ВСТУП

При розробці будь-якого програмного продукту одним із найбільш важливих факторів, які будуть впливати на його конкурентоспроможність, є його інтерфейс. Тому компанії витрачають досить значну частину бюджету проекту для залучення дизайнерів, які зможуть створити «ідеальний» дизайн. Але в процесі розвитку програмного продукту можуть змінюватися вимоги, цілі, а, як результат, і інтерфейс. До того ж, іноді ці зміни значні, наприклад зміна цільової аудиторії чи навіть повна зміна напрямку діяльності, а іноді зміни можуть бути зовсім мізерні, а саме треба змістити логотип на один піксель. Тому добре було б мати який-небудь інструмент, що буде показувати нам ці зміни в інтерфейсі. Тут на допомогу і приходить візуальне тестування. Причому, зараз тільки починають з’являтися готові рішення для автоматизованого візуального тестування, але все ж більшість роботи проводиться вручну.

Таким чином, метою роботи є розробка системи візуального тестування графічних інтерфейсів з урахуванням особливостей опрацювання текстових компонентів. Для досягнення цих цілей вирішені наступні завдання:

1. Провести аналіз особливостей візуального тестування інтерфейсів застосунків, що містять текстові компоненти, та існуючих технологій розпізнавання тексту.
2. Проаналізізувати архітектури штучних нейронних, що вирішують задачу розпізнавання тексту
3. Сформулювати засади технології візуального тестування текстових компонентів інтерфейсів застосунків з використанням нейронних мереж
4. Виконати проектування системи автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків.

5. Розробити систему автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків. Дослідити ефективність та надійність розробленої системи.

РОЗДІЛ 1 АНАЛІЗ ЗАДАЧІ ВІЗУАЛЬНОГО ТЕСТУВАННЯ

1.1 Аналіз сучасного стану проблеми

Тестування програмного забезпечення є важливою частиною процесу розробки програмного забезпечення для забезпечення якості продукції. У процесі тестування програми з користувацьким інтерфейсом, одним з останніх кроків є забезпечення виконання всіх спроектованих функціональних можливостей та відповідність інтерфейсу користувача заданому дизайну. Багато компаній, що розробляють програмне забезпечення, забезпечують це шляхом ручної перевірки «всіх можливих» випадків. Ця виснажлива діяльність дуже трудомістка, а отже, і досить дорога та може містити помилки.

Основна увага в роботі приділена питанням зменшення потреби в ручному тестуванні інтерфейсів користувача шляхом впровадження інструменту, який би автоматизував більшу частину тестування. Мета такого інструменту - розширити можливості системи автоматичного тестування, а не просто замінити ручні тести на візуальне тестування інтерфейсу користувача, та тим самим підвищити ефективність команди з забезпечення якості продукта.

Автоматизація проводиться за допомогою візуального інструменту тестування графічного інтерфейсу, який використовує розпізнавання зображень для пошуку об’єктів в інтерфейсі. На сучасному ринку програмних продуктів вже існують подібні інструменти. В роботі проводиться аналіз трьох таких інструментів – Sikuli, JAutomate та Applitools з метою виділення їх можливотей та особливостей функціонування.

Важливою частиною розпізнавання зображень є розпізнавання тексту. І тестовані інструменти включають цю функціональність, однак точність їх виявилася низькою, що призвело до рішення про вдосконалення, додавши функцію навчання.

1.2 Автоматичне тестування інтерфейсу користувача

Автоматичне тестування інтерфейсу користувача розділяється на 3 покоління:

• Перше покоління: тестування на основі координат.

• Друге покоління: тестування на основі віджетів.

• Третє покоління: візуальне тестування.

У тестових сценаріях на основі координат (перше покоління) тестові сценарії складаються із записаної користувачем взаємодії з системою, наприклад, натискання клавіш та клацання миші. Дії користувача можуть бути відтворені, що імітує взаємодію користувача з системою. Цей підхід залежить від змін макету, оскільки в основі статичні координати x та y.

Підхід другого покоління до тестування інтерфейсу користувача полягає в тому, щоб тестові сценарії безпосередньо взаємодіяли з кодом елементів інтерфейсу системи.

Останнє покоління це візуальне тестування, де вміст екрана аналізується за допомогою технології розпізнавання зображень. Ключові елементи графічного інтерфейсу, такі як кнопки та текстові поля, локалізуються та потім інструмент тестування взаємодіє з ними, щоб запустити додаток. Потім стан інтерфейсу порівнюється з очікуваним результатом за допомогою алгоритмів порівняння зображень.

Візуальне тестування графічного інтерфейсу - це доволі нова методика, хоча серцевина цієї техніки була винайдена на початку дев'яностих років. Однак, завдяки обчислювальній важкості використовуваних алгоритмів розпізнавання зображень, цей підхід не був широко застосований до недавнього часу, поки не було розроблено апаратне та програмне забезпечення, яке було досить потужним, щоб зробити цю техніку придатною для промислових застосувань. Візуальне тестування графічного інтерфейсу використовує розпізнавання зображень та сценарії для взаємодії з верхнім

растровим графіком та вхідним шаром, відображеним користувачеві на моніторі тестованої системи.

Візуальне тестування програми - це намагання з'ясувати її нефункціональні помилки, які з'являються через зміну графічного стану програми.[16]

Типовим прикладом може бути веб-додаток, в якому графічний інтерфейс програмується зазвичай за допомогою комбінації мови розмітки HyperText (HTML) та каскадних таблиць стилів (CSS).

HTML часто використовується для визначення вмісту сторінок веб-програми (наприклад, сторінка містить таблицю, зображення тощо), тоді як CSS визначає структуру та зовнішній вигляд веб-програми (наприклад, колір шрифту, абсолютне розміщення елементів веб-сторінки, і так далі). Отримана веб-програма - це набір правил (CSS та HTML), застосованих до статичного вмісту (наприклад, зображення, відео, текст).

Поєднання правил є вирішальним, і незначна зміна може повністю змінити візуальний стан веб-програми. Такі зміни дуже важко, іноді навіть неможливо виявити за допомогою автоматизованого функціонального тестування програми. Це тому, що функціональні тести перевіряють потрібну функціональність веб-програми та ігнорують такі характеристики веб-сторінок, як червоний колір заголовка, пробіл між двома абзацами тощо. Ось чому має пройти візуальне тестування. Знову ж це робиться або вручну, коли тестер проходить усі випадки використання веб-додатків і перевіряє, що програма не зламалася візуально. Або виконується автоматично, виконуючи скрипти, які підтверджують візуальний стан програми.

Приклад візуальної помилки в додатку Instagram наведений на рисунку 1.1).

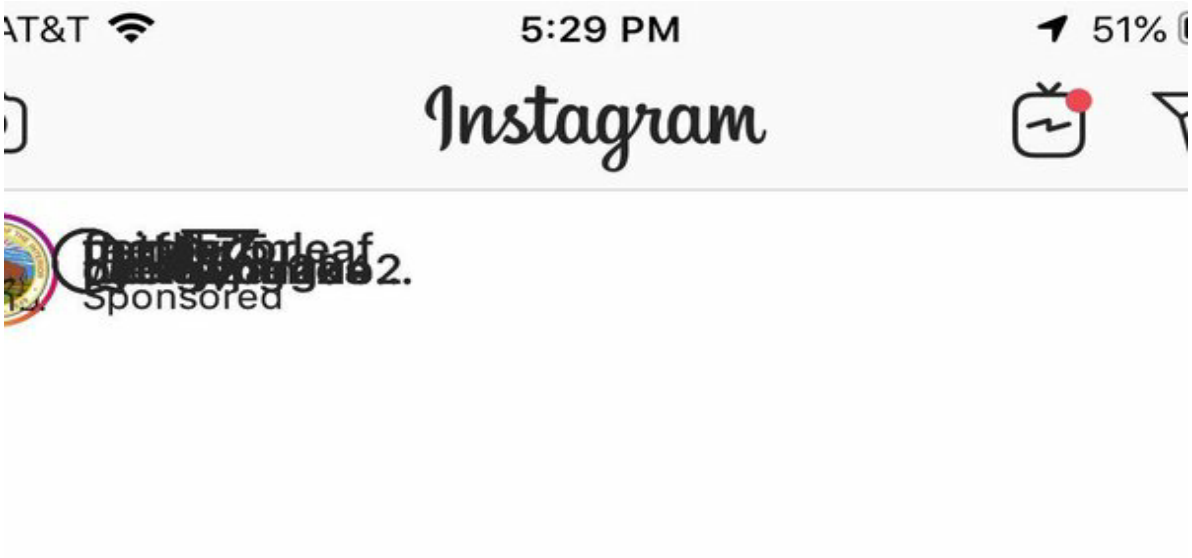


Рисунок 1.1 – Візуальна помилка в додатку Instagram.

Текст і реклама налазять один на одного.

Візуальні помилки трапляються і в інших компаніях: Google, Amazon, Yelp, Southwest, United, Virgin Atlantic, OpenTable. Це не косметичні проблеми. У кожному випадку візуальні помилки блокують дохід. Чому виникають ці візуальні помилки? Вони не роблять функціональне тестування? Вони роблять - але цього недостатньо.

Візуальні помилки викликають проблеми відображення. І перевірка відображення - це не те, для чого призначенні інструменти функціонального тестування. Функціональне тестування оцінює функціональну поведінку.

Як вже згадувалось вище, те, як виглядає веб-сторінка, в основному визначається CSS-скриптом. Використовується два способи автоматизованого тестування:

1. затвердження сценарію CSS

2. порівняння фотографій екрану нових та старих версій програми.

Існіючі системи автоматичного візуального тестування:

Sikuli - відкрите крос-платформне візуальне середовище створення сценаріїв-скриптів, яка орієнтована на програмування графічного інтерфейсу за допомогою зображень (скріншотів). Як скриптова мова в Sikuli використовується Jython, тобто в скрипті при бажанні можна використовувати

конструкції з мови Python. У SikuliX з'явилася можливість використовувати для написання скриптів мовою Ruby в реалізації jRuby. Sikuli доступна для роботи в Windows, Mac OS X і Linux.

JAutomate - комерційний інструмент для візуального тестування GUI (VGT), розроблений Innovative Tool Solutions у співпраці зі шведською компанією-тест-консультантом Inceptive AB. Інструмент був розроблений Мішелем Нассом у 2006 році, після визнання потенціалу техніки VGT та відсутності на ринку інструментів, що застосовували цю методику. JAutomate в 2006 році був представлений як концепція провідному продавцю тестових інструментів того часу, Меркурію (сьогодні Hewlet Packard (HP), але компанія не виявила інтересу. У 2011 році перша версія JAutomate була випущена на ринок і з тих пір використовується в декількох промислових проектах, наприклад, у Volvo, Siemens, CompuGroup Medical. [17]

Applitools забезпечує візуальне управління додатками, а також тестування та моніторинг візуального користувацького інтерфейсу веб-додатків і мобільних додатків. На відміну від традиційних інструментів функціонального тестування, які вимагають багато рядків коду для перевірки макета - і при цьому пропускають критичні помилки - Applitools перевіряє весь екран всього одним викликом API. Це економить години роботи інженерів по автоматизації випробувань; розширює охоплення тестами для більшої кількості операційних систем, браузерів і вікон перегляду; покращюэ якысть програмного забезпечення.

Додаток працює з: Selenium (Java, JavaScript, C #, PHP, Python, Ruby), Appium (Java, JavaScript, C #, PHP, Python, Ruby), WebDriverIOProtractorXCUI (Objective-C, Swift), EspressoStorybook (React, Angular, Vue).

Порівняння цих тестових затосунків наведено у таблиці 1.1.

Таблиця 1.1 Порівняння існуючих систем автоматичного візуального тестування

Властивість	JAutomate	Applitools	Sikuli
Мова розробки	Java	C#	Jython
Кількість порівнянь в секунду	Залежить від розмірів зображень	7	5
Підтримка тестових пакетів	+	+	Тільки юніт-тести
Записа та відтворення тесту	+	-	-
Напів-автоматичні тести	+	-	-
Зворотна сумісність	+	+	невизначена

1.3 Особливості візуального тестування текстових блоків програмних інтерфейсів

Текст – основне наповнення будь-якого сайту, в нього вкладається та описується увесь зміст продукту. На більшості сайтів основну інформацію складають тексти. Саме від текстового наповнення залежить інформативність ресурсу, а отже, інтерес до нього з боку користувача.

Інформаційне наповнення — головне, на що звертають увагу користувачі сайтів. Так говорять численні дослідження щодо правил розробки веб-ресурсів. Потрапивши на нову сторінку, відвідувачі перш за все звертаються до тієї її частини, яка містить корисну інформацію, переглядають

заголовки й інші елементи, що вказують на зміст сторінки. І лише після цього вони звертаються до навігації, щоб перейти на іншу сторінку.

Якість текстової інформації безпосередньо впливає на досягнення бажаних результатів. А отже, підготовка текстів потребує серйозного ставлення та детального тестування.

Великою проблема для тестування є локалізація сайтів, адже перевірити усе наповнення сайту декількома мовами для людини є дуже важкою задачею, а інколи й неможливою. В той час, коли для інструменту візуального тестування перевірка текстових блоків в порінянні з макетом для будь-якої кількості мов є частиною стандартної роботи і для комп'ютера важкість задачі від кількості мов не залежить.

Та є інша проблема, яка стосується комп'ютерів і з якою людині впоратись набагато легше. Достатньо точного розпізнавання символів у друкованому тексті на сьогоднішній день можна досягти тільки у випадку чіткого зображення, такого як у друкованих документах.

Сьогодні у світі існує дуже багато різних програм розпізнавання символів, але слід зазначити, що здатність читати не якісний текст, прочитаний людиною, все ще перевищує можливості комп'ютера. Кожен надрукований текст має деяку властивість - шрифт, у який він набраний. А коренем проблеми є відмінності у зглажуванні/рендерінгу шрифтів.

У Apple і Microsoft завжди були розбіжності з приводу того, як показувати шрифти на комп'ютерному екрані. Сьогодні обидві компанії використовують "субпіксельну" візуалізацію, щоб домогтися більш чітко вигляду шрифтів на екранах з традиційно невисокою роздільною здатністю.

Apple вважає, що мета алгоритму - зберегти дизайн шрифту наскільки це можливо, навіть за рахунок невеликої розмитості.

Microsoft вважає, що форма кожної букви повинна бути в межах кордонів, встановлених пікселями, щоб запобігти розмиванню і поліпшити читабельність навіть за рахунок неповної відповідності зображенню.

Шрифти Apple насправді трохи розмиті, зі змазаними краями, але при невеликому розмірі шрифту помітно більше варіацій між різними сімействами шрифтів, тому що їх візуалізація ближче до того, як шрифт виглядав би надрукованим у високій роздільній здатності.

Відмінності виходять через «спадщину» Apple в настільній поліграфії і графічному дизайні. Приємна річ в алгоритмі Apple полягає в тому, що ви створюєте сторінку тексту для друку, і на екрані у вас варіант дуже близький до остаточного продукту. Це особливо корисно, якщо для вас важливо, наскільки темним виглядає блок тексту. Механізм Microsoft по перетворенню шрифтів в пікселі означає, що вони насправді не проти використовувати більш тонкі лінії, щоб уникнути розмитих країв, навіть коли це робить цілий параграф більш «легким», ніж він буде виглядати на друці.

Перевага методу Microsoft полягає в тому, що він працює краще для читання з екрану. Microsoft прагматично вирішила, що чіткий текст на екрані, який зручно читати, більш важливий, ніж задумка дизайнера шрифту про те, наскільки світлим або темним повинен відчуватися цілий блок тексту. Microsoft дійсно розробила шрифти для читання з екрану, наприклад, Georgia і Verdana, з урахуванням меж пікселів. Вони дуже гарні на екрані, але не вражають при друці.

З цього випливає, що одна і та ж сторінка тексту буде виглядати абсолютно по-різному для комп'ютера, якщо макет зроблений шрифтами Apple на Mac комп'ютері, а перевірка і актуальна картинка сторінки тексту зроблена на комп'ютері Windows.

Для того, щоб уникнути хибних результатів візуального тестування при порявнянні сторінок зі згладжуванням тексту, необхідно виокремлюювати на сторінках текст, та порівнювати його за змістом, окремо від візуального контенту. Тому потрібно застосувати технології оптичного розпізнавання тексту, щоб визначити саме зміст текстових блоків інтерфейсів застосунків.

1.4 Аналіз існуючих технологій оптичного розпізнавання тексту

У цьому розділі проведено аналіз існуючих систем оптичного розпізнавання тексту за наступними критеріями: вартість, мова написання, наявність відкритого коду, кількість мов розпізнавання, можливість програмного використання та наявність інформації про структуру розпізнаної системою сторінки, що є корисним для виділення окремих блоків тексту.[2]

1.4.1 Tesseract

Tesseract – система оптичного розпізнавання символів з відкритим кодом. Вона була створена компанією HP в 1984-1995 роки. Підтримується більшістю операційних систем. У 2006 році Google купив її та відкрив початковий код під ліцензією Apache 2.0 для продовження розробки. У цей час програма вже працює з UTF-8, розпізнає багато мов, серед яких і українська. [6]

Систему добре задокументовано. Tesseract написаний на C / C ++. Інструкція з установки досить вичерпна. Tesseract поверне результати у вигляді звичайного тексту, hOCR або у форматі PDF, з текстом, накладеним на вихідне зображення.

Tesseract OCR працює поетапно. Перший крок – бінарізація зображення з використанням адаптивного порогу. Наступним кроком є компонентний аналіз, який використовується для отримання контурів символів. Після цього контури перетворюються на регіони. Регіони розділяються на текстові рядки. Текст поділяється на слова з використанням пробілів. Розпізнавання тексту запускається як двопрохідний процес. Спочатку проходить спроба розпізнавати кожне слово з тексту. Кожне слово, яке було розпізнано, передається адаптивному класифікатору як навчальні дані. Адаптивний класифікатор намагається розпізнати текст більш точним чином. Оскільки адаптивний класифікатор отримав деякі навчальні дані, він навчився чомусь

новому, тому заключний етап використовується для вирішення різних проблем та вилучення тексту з зображень. [21]

Вартість: безкоштовна.

Мова написання: C/C++.

Відкритий код: так.

Мови розпізнавання: більше 100 мов підтримуються бібліотекою, є можливість навчяння новій мові.

Можливість програмного використання: так.

Інформація про структуру розпізнаної сторінки: так.

1.4.2 FineReader

FineReader — програма для оптичного розпізнавання символів, розроблена російською компанією ABBYY [7].

Можливості:

1. Висока точність і швидкість перетворення документів

Програма швидко і з точністю розпізнає відскановані або сфотографовані документи, перетворюючи їх в електронні редаговані формати або PDF з можливістю пошуку. При розпізнаванні якісних документів швидкий режим збільшить швидкість на 40 % без шкоди для точності. А для чорно-білих документів можна використовувати також чорно-білий режим розпізнавання, який прискорить роботу ще на 30 %.

2. Свобода від передруку і переформатування документів

Завдяки технології Adaptive Document Recognition Technology, ABBYY FineReader 12 зберігає вихідну структуру багатосторінкових документів, включаючи розташування тексту, таблиць, колонтитулів, приміток, нумерацію сторінок, змісту, змісту та ін. Задати типи областей (Текст, Картинка, Таблиця і ін.) і вказати їх призначення, можна і вручну.

3. Швидкий доступ до необхідної інформації

FineReader забезпечує миттєвий доступ до сторінок документа, що сканується незалежно від його розміру. Щоб почати працювати з документом, вам не потрібно чекати, поки він розпізнається цілком.

4. Підтримка 190 мов

ABBYY FineReader 12 розпізнає документи на 190 мовах, в будь-яких комбінаціях.

5. Інструменти для поліпшення якості зображень документів

ABBYY FineReader 12 вміє справлятися як з спотвореннями, характерними для цифрових фотографій (трапецієподібні спотворення, викривленння рядка, цифровий шум, і так далі), так і з дефектами зображення, пов'язаними зі станом вихідних паперових документів (пожовклий від часу папір, рукописні позначки, штампи).

6. Перетворення в PDF-файли з можливістю пошуку

Програма перетворює зображення документів і PDF-файли, отримані зі сканера (без текстового шару), в формати, придатні для збереження в електронному архіві з можливістю пошуку: PDF з текстовим шаром або PDF/A.

7. Підтримка широкого спектра форматів збереження результатів

Програма підтримує широкий набір форматів для збереження документів, необхідних вам у роботі. Можна записати результати розпізнавання в файл або відправити їх відразу в додатки Microsoft Word, Excel, PowerPoint, OpenOffice Writer та ін.

8. Створення електронних книг в популярних форматах fb2, ePub

Програма підтримує збереження в найпопулярніші формати електронних книг (fb2 і ePub, також Kindle), це допоможе швидко зробити електронну копію для портативного пристрою — електронної книги, планшета, смартфона, і ін. [8]

Вартість: $200, наявна пробна версія.

Мова написання: C/C++.

Відкритий код: ні.

Мови розпізнавання: 192.

Можливість програмного використання: так.

Інформація про структуру розпізнаної сторінки: ні.

1.4.3 Google Cloud Vision

Хмарні служби Google включають інструмент OCR, Cloud Vision. Cloud Vision найкраще вилучає корисні результати із зображень із низькою роздільною здатністю. Існує кілька кроків, щоб розпочати та запустити систему, але документація їх добре висвітлює. [5]

Це інтерфейс, який дозволяє розробникам аналізувати вміст зображення за допомогою вилучених даних. З цією метою Google використовує моделі машинного навчання, навчені великим набором зображень. Все це доступно за допомогою одного запиту інтерфейсу. Двигун класифікує зображення, виявляє предмети, обличчя людей та розпізнає друковані слова у зображеннях.

Google Cloud Vision – це онлайн система, тож для початку роботи достатньо зареєструватися в програмі, а процесс оборобки зображень не займає ресурси Вашого комп'ютера. Проте, на відміну від офлайн систем, Google Cloud Vision неможливо використовувати без доступу до мережі інтренет, та робота системи може сповільнюватись затримками та пропускною здатністю мережі.

Вартість: 1000 безкоштовних сторінок кожен місяць, $1.50 за кожну наступну тисячу.

Мова написання: невідома.

Відкритий код: ні.

Мови розпізнавання: більше 200 мов.

Можливість програмного використання: так.

Інформація про структуру розпізнаної сторінки: ні.

1.5 Постановка задачі

Виходячи з проведеного аналітичного огляду технологій візуального тестування та технологій розпізнавання тексту було визначено наступні завдання до вирішення:

1. Проаналізізувати архітектури штучних нейронних, що вирішують задачу розпізнавання тексту
2. Сформулювати засади технології візуального тестування текстових компонентів інтерфейсів застосунків з використанням нейронних мереж
3. Виконати проектування системи автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків.
4. Розробити систему автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків. Дослідити ефективність та надійність розробленої системи.

В результаті виконання поставлених задач буде розроблена система автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків з консольним інтерфейсом.

1.6 Визначення вимог та завдань

Основними вимогами до програми є:

- Система має приймати на вхід файли зображень у растрових форматах (jpeg, png, gif, bmp або tiff)

- Система повинна визначати різницю між вхідними файлами в візуальних сегментах сторінки (кількість відмінних пікселів) та в текстових сегментах (кількість відмінних символів).
- Система повинна записувати в вихідний файл попіксельну різницю між вхіжними зображеннями.
- Система повинна забезпечувати якість обслуговування користувачеві.
- Система повинна забезпечувати точність розпізнавання не нижче 90%.
- Система повинна видавати результат в обмежений проміжок часу (до 1 секунди), оскільки при автоматичному візуальному тестуванні необхідно порівнювати велику кількість сторінок.
- Система повинна мати консольний інтерфейс з можливістю збереження результатів в файл для ручного візуального тестування, а також програмний інтерфейс для інтеграції з існуючими системами автоматичного візуального тестування.

РОЗДІЛ 2 АРХІТЕКТУРА СИСТЕМИ

2.1 Узагальнена структура системи візуального тестування текстових компонентів інтерфейсів застосунків з використанням нейронних мереж

Система повинна приймати на вхід два зображення, та розділити їх на текстові та нетекстові блоки для окремого порівняння. Нетекстові частини зображення порівнюються попіксельно. Текстові блоки порівнюються за змістом за допомогою системи розпізнавання тексту. Таким чином, систему можна розділити на такі основні компоненти: зчитуваня зображень, поділ на текстові та візуальні блоки, порівняння візуальних блоків, розпізнавання тексту за допомогою нейромереж, порівняння тексту, виведення результату. Узагальнена структура системи представлена на рисунку 2.1.

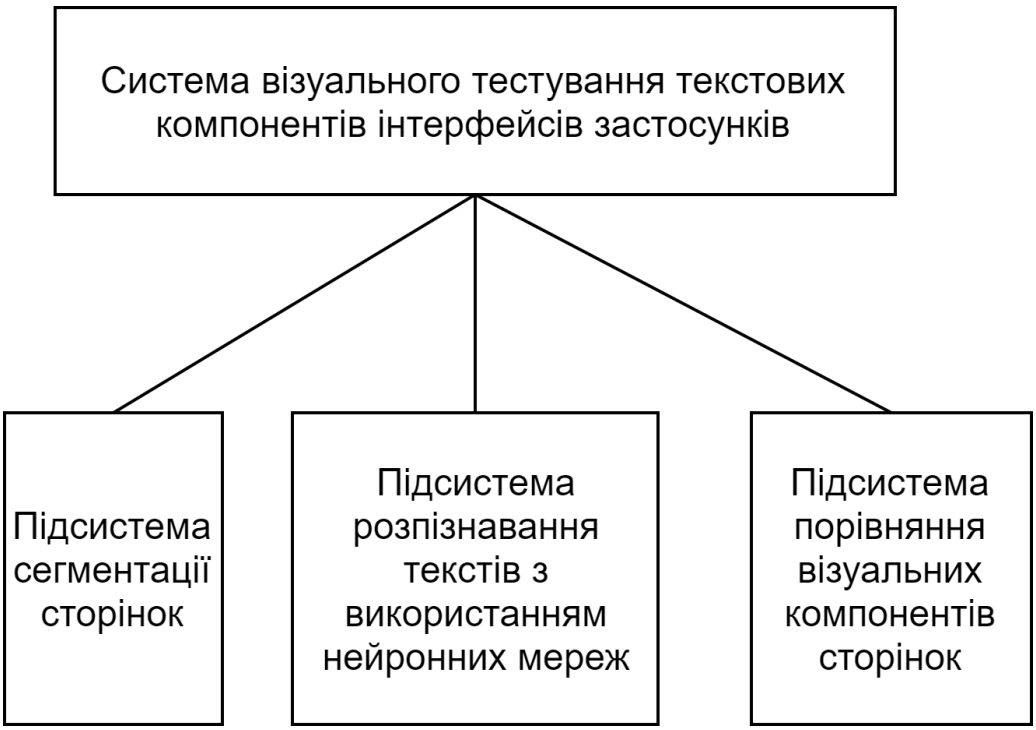


Рисунок 2.1 Узагальнена структура системи

Головний компонент сисеми - це блок розпізнавання тексту, адже саме завдяки цьому блоку система здатна порівнювати зображення з різними налаштуваннями згладжування тексту.

Система повинна показувати повідомлення про помилку користувачеві, коли вхідні дані не в потрібному форматі.

2.2 Алгоритм розпізнавання тексту

У цьому підрозділі буде розглянуто основні складові процесу розпізнавання тексту. Загальна схема зазначена на рисунку 2.2.

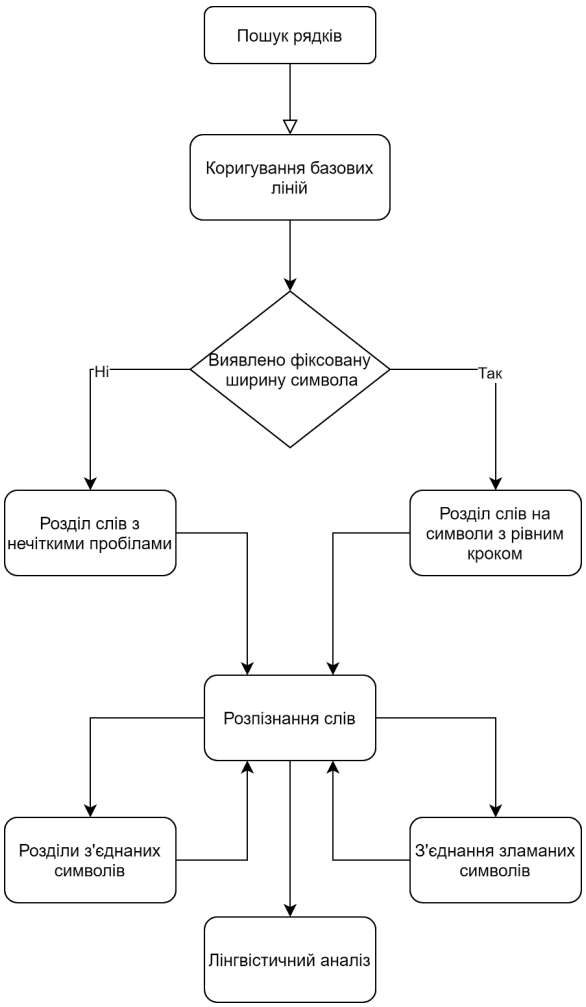


Рисунок 2.2 – Загальний алгоритм розпізнавання тексту

Пошук рядків

Пошук рядків виконується таким чином, що нахилена сторінка може бути розпізнана без необхідності вирівнювання, тим самим не витрачаючи якості зображення. Ключовими частинами процесу є фільтрування сегментів та побудова ліній. Якщо припустити, що аналіз макетів сторінки вже надав текстові області приблизно однакового розміру тексту, простий фільтр по висоті видаляє літери, що випадають і вертикально торкаються символів. Середня висота рядка в регіоні наближається до розміру тексту, тому безпечно відфільтрувати крапки, менші за деяку частку середньої висоти, найімовірніше, це розділові знаки, діакритичні позначки та шум. Відфільтровані сегменти краще підходять для моделі непересічних, паралельних, але похилих ліній. Сортування та обробка точок за x-координатою дає можливість знайти точки в унікальному текстовому рядку, відстежуючи нахил по всій сторінці, значно зменшуючи небезпеку присвоєння невірного рядка тексту з нахилом. Після того, як відфільтровані сегменти будуть співставлені рядкам та знайдені базові лінії, відфільтровані сегменти вставляються у відповідні рядки. Заключний крок процесу створення рядків об’єднує сегменти, які перекриваються принаймні наполовину по горизонталі, ставлячи діакритичні позначки разом з правильною базовою лінією та правильно пов’язуючи частини деяких зламаних символів. [7]

Коригування базових ліній

Після того як текстові рядки знайдені, базові лінії встановлюються більш точно, використовуючи квадратичний сплайн. Базові лінії встановлюються шляхом розподілу сегментів на групи з відносно лінійним зміщенням для початкової прямої базової лінії. Квадратний сплайн застосовується до найбільш густоподібної частини тексту. Перевага квадратичного сплайна полягає в тому, що це обчислення є досить стабільним, але можуть виникати розриви сегментів сплайну. Більш традиційний кубічний сплайн може працювати краще.[14]

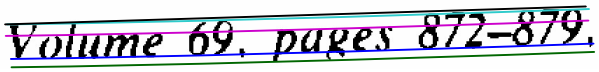


Рис. 2.3 - Приклад вигнутої встановленої базової лінії.

На рис.2.3 показаний приклад рядка тексту з відкорегованою базовою лінією, нижньою лінією, середньою лінією та верхньою лінією. Усі ці лінії є «паралельними» (у-поділ є постійним по всій довжині) і трохи вигнуті. Верхня лінія руху – зелено-блакитна, а чорна лінія над нею насправді пряма. При ретельному огляді видно, що зелено-блакитна лінія вигнута відносно прямої чорної лінії над нею.

Виявлення тексту з фіксовано шириною символа

Необхідно перевірити текстові рядки, щоб визначити, чи є вони з фіксованим кроком між буквами. Якщо знайдено текст з фіксованою шириною символа, необхідно розбити слова на символи, використовуючи знайдений крок, і пропустити фазу розбиття на символи для цих слів. На рис. 2.4 показаний типовий приклад слова з фіксованим кроком.

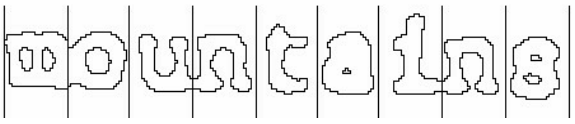


Рис 2.4 - Слово з фіксованим кроком.

Пропорційний пошук слів

Нефіксований або пропорційний інтервал між текстами є дуже нетривіальним завданням. Рисунок 2.5 ілюструє деякі типові проблеми. Розрив між цілими та десятковими цифрами в "11,9%" є розміром аналогічним загальному пробілу і, безумовно, більший, ніж пробіл між "erated" та "junk". Не існує горизонтального проміжку між обмежувальними полями "of" та "financial". Tesseract вирішує більшість цих проблем, вимірюючи зазори в обмеженому вертикальному діапазоні між базовою лінією та середньою

лінією. Проміжки, близькі до порогу на цьому етапі, стають нечіткими, так що остаточне рішення може бути прийняте після розпізнавання слова.

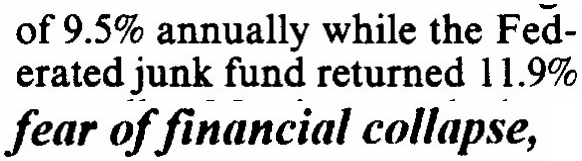


Рис. 2.5 - Нефіксований інтервал слів.

Розпізнавання слів

Частиною процесу розпізнавання є визначення того, як слово має бути сегментоване на символи. Початковий результат сегментації рядків класифікується першим. Решта кроків розпізнавання слів стосується лише тексту з фіксованою шириною тексту.

Поділ з'єднанних символів

Поки результат розпізнавання слова незадовільний, система намагається покращити результат, розділяючи сегменти з найгіршим результатом від класифікатора символів. Точки розділу знаходять із увігнутих вершин багатокутного наближення контуру і можуть мати іншу увігнуту вершину, протилежну, або відрізок лінії. Щоб успішно відокремити об'єднані символи з набору ASCII, може знадобитися до 3 пар точок відсікання.

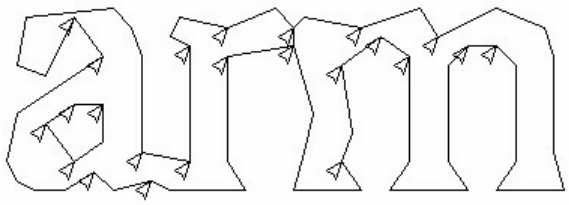


Рис. 2.6 - Можливі точки подрібнення.

На рис. 2.6 показаний набір кандидатів зі стрілками та обраний поділ у вигляді лінії по контуру, де літера 'r' торкається 'm'.

Відсікання виконується за пріоритетом. Будь-яке відсікання, яке не покращує впевненість у результаті, скасовується, але не повністю відкидається, так що відрізок може бути використаний пізніше асоціатором, якщо потрібно.

Пов’язування зламаних символів

Коли потенційні розділи вичерпані, якщо слово все ще не розпізнано, воно надається асоціатору. Асоціатор здійснює пошук A* на графі сегментації можливих комбінацій максимально нарізаних сегментів на символи-кандидати. Це робиться без фактичної побудови графу сегментації, але натомість використовується хеш-таблиця відвідуваних вершин. Пошук A* виконується шляхом витягування нових вершин-кандидатів із черги пріоритетності та оцінки некласифікованих комбінацій фрагментів.

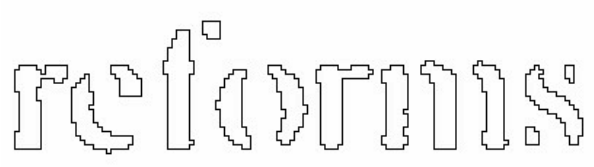


Рис. 2.7 - Типовий приклад зламаних символів.

Статичний класифікатор символів

Характеристики символів у нерозпізнаному тексті не повинні бути такими ж, як характеристики у навчальних даних. Під час навчання сегменти полігонального наближення використовуються як характеристики, але під час розпізнавання ознаки невеликої фіксованої довжини (у нормованих одиницях) витягуються з контуру та співставляються з кластеризованими ознаками прототипу даних від навчання. На рис. 2.8 короткі товсті лінії є характеристиками, взятими з нерозпізнаного тексту, а тонкі довші лінії - це згруповані сегменти полігонального наближення, які використовуються як прототипи. Один прототип, який з'єднує частини літери, абсолютно не співпадає. Але більшість характеристик добре узгоджуються. Цей приклад

показує, що цей процес коротких характеристик, що відповідають довшим прототипам, легко впорається з розпізнаванням пошкоджених зображень. Основна його проблема полягає в тому, що обчислювальна вартість обчислення відстані між невідомим та прототипом дуже висока.

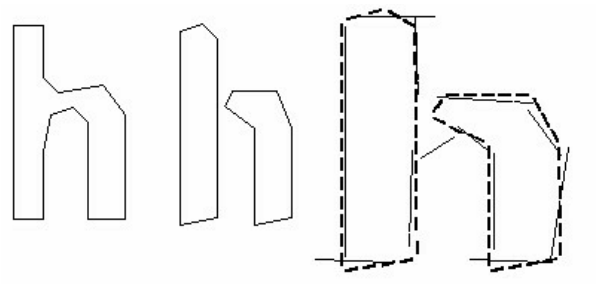


Рис. 2.8 - Характеристики символу ‘h’

Таким чином, характеристики, витягнуті з невідомого, є тривимірними, (x, y положення, кут), зазвичай приблизно з 50100 ознаками в символі, а ознаки прототипу - 4-мірні (x, y, положення, кут, довжина), зазвичай 10-20 характеристик у конфігурації прототипу.

Класифікація виконується в два етапи. На першому кроці створюється список класів символів, яким невідомі можуть відповідати. Кожна функція отримує з грубо квантованої тривимірної таблиці пошуку, бітовий вектор класів, який може відповідати, і бітові вектори підсумовуються за всіма характеристиками. Класи з найбільшою кількістю (після виправлення очікуваної кількості функцій) стають списком для наступного кроку.

Кожна характеристика невідомого символа шукає бітовий вектор прототипів даного класу, якому вона може відповідати, а потім обчислюється фактична схожість між ними. Кожен клас символів прототипу представлений логічним виразом суми добутків з кожним символом, який називається конфігурацією, тому процес обчислення відстані зберігає запис про загальну доказову схожість кожної характеристики у кожній конфігурації, а також кожного прототипу. Найкраща комбінована відстань, яка обчислюється за

підсумками ознак та прототипів, є найкращою серед усіх збережених конфігурацій класу.

Тренувальні дані.

Оскільки класифікатор може легко розпізнати пошкоджені символи, класифікатор не навчається пошкодженим символам. Класифікатор підготовлюється на 20 зразках із 94 символів із 8 шрифтів одного розміру, але з 4-ма атрибутами (звичайний, жирний, курсив, жирний курсив), обробивши загалом 60160 навчальних символів.

Лінгвістичний аналіз

Щоразу, коли модуль розпізнавання слів розглядає нову сегментацію, лінгвістичний модуль вибирає найкращий рядок слів за оцінкою класифікатора. Остаточне рішення для даної сегментації - це слово з найнижчою загальною оцінкою похибки, де кожна з перерахованих вище категорій множиться на різну константу.

Слова з різних сегментацій можуть містити в них різну кількість символів. Важко порівняти ці слова безпосередньо, навіть коли класифікатор стверджує, що оцінює ймовірності. Ця проблема вирішується шляхом генерації двох чисел для кожної класифікації символів. Перший, що називається впевненістю, - це від'ємна нормована похибка від прототипу. Другий, який називається рейтингом, помножує нормоване відстань від прототипу на загальну довжину контуру в невідомому символі. Рейтинги символів у слові можуть бути підсумовані суттєво, оскільки загальна довжина контуру для всіх символів у слові завжди однакова.

Адаптивний класифікатор

Було запропоновано та продемонстровано, що системи OCR можуть отримати користь від використання адаптивного класифікатора. Оскільки статичний класифікатор повинен добре узагальнювати будь-який тип шрифту, його здатність розрізняти різні символи або символами та несимволи послаблюється. Тому більш чутливий до шрифту адаптивний класифікатор, який навчається результатами статичного класифікатора, зазвичай

використовується для отримання більшої точності в кожному документі, де кількість шрифтів обмежена.

Єдина суттєва відмінність статичного класифікатора від адаптивного класифікатора, крім навчальних даних, полягає в тому, що адаптивний класифікатор використовує ізотропну нормалізацію базової лінії / висоти, тоді як статичний класифікатор нормалізує символи центроїдом для позиції та моменти для анізотропної нормалізації розмірів.

Нормалізація базової лінії / висоти полегшує розрізнення символів верхнього та нижнього регістрів, а також покращує імунітет до пошкоджень. Основна перевага нормалізації мовного символу - це усунення співвідношення розмірів шрифту та деякого ступеня ширини шрифту. [6]

2.3 Проектування системи

Основні компоненти системи: зчитуваня зображень, поділ на текстові та візуальні блоки, порівняння візуальних блоків, розпізнавання тексту за допомогою нейромереж, порівняння тексту, виведення результату. Загальна блок-схема зазначена на рисунку 2.9.

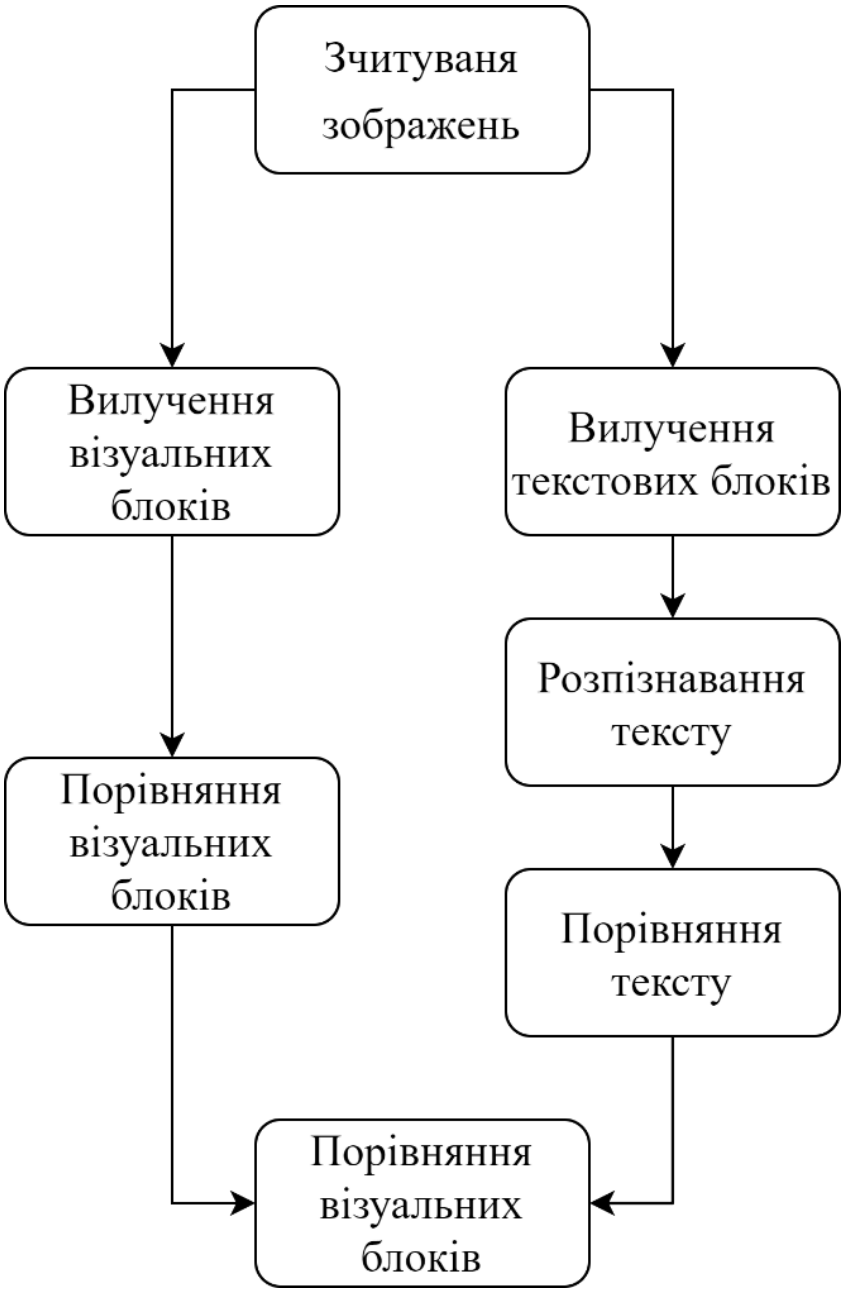


Рисунок 2.9 Основні компоненти системи.

Зчитування зображень може відбуватися в двох режимах: ручному та автоматичному. В ручному режимі, користувач вказує шлях до зображень для порівняння в консольному інтерфейсі. В автоматичному режимі система використовується як блок порівняння зображень у сторонній програмі автоматичного візуального тестування.

Порівняння візуальних блоків виконується за допомогою попіксельного поріняння, кожний піксель розкладається на компоненти кольорів R, G, B.

Кожний компонент порівнюється окремо та при виявленні розбіжності система виводить різницю зображень та кількість неспівпадаючих пікселів.

Робота з текстовими блоками починається з попередньої обробки зображень. Далі виконується поділ зображення на блоки, розпізнавання текстових блоків за допомогою нейронних мереж та порівняння розпізнаного змісту.

Етап попередньої обробки працює над вхідним зображенням, щоб зробити його готовим до розпізнавання. Слід зазначити, що між обробкою та точністю існує сильний зв'язок. Чим більше часу ви витрачаєте на попередню обробку, тим вища точність, але це також збільшує час виконання.

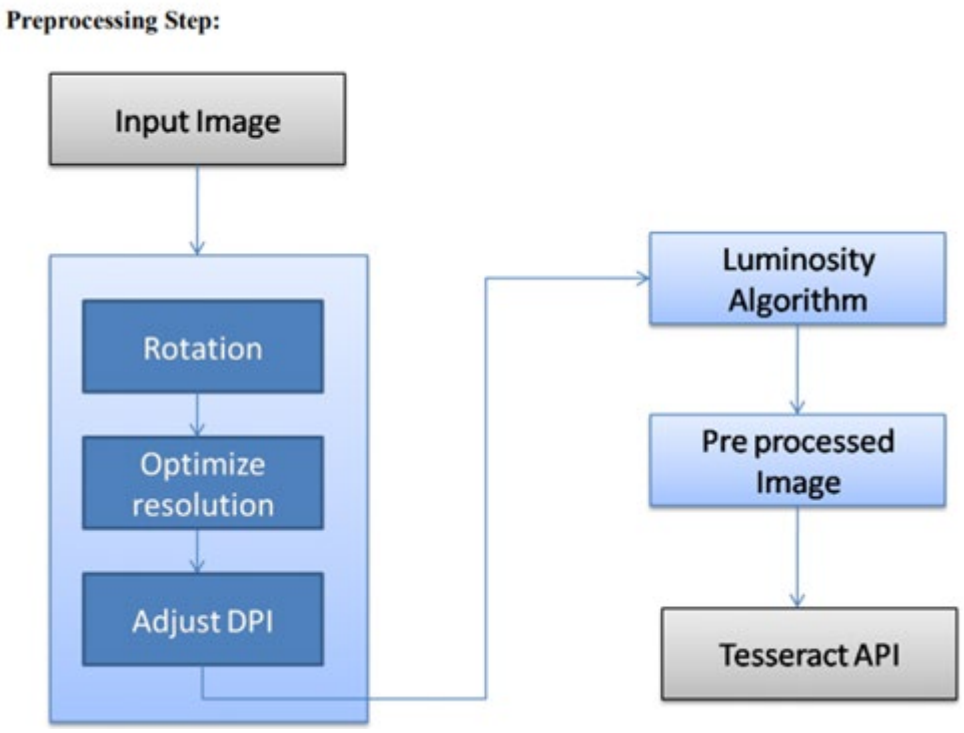


Рисунок 2.10 - Попередня обробка зображення.

Рисунок 2.10 пояснює підсистему попередньої обробки зображення. Етапи попередньої обробки: обертання, оптимізація роздільної здатності, коригування DPI і, нарешті, кольорові зображення перетворюються в чорно-

білі з використанням техніки освітленності. Потім оброблене зображення подається в Tesseract як вхід.

DPI (Dots per inch) - показник роздільної здатності при введені чи виведені зображальної інформації на/в носій для подальшої її виведення, обробки чи зберігання [4].

Крок обертання обертає зображення, якщо під час зйомки камера не знаходилася під кутом 0 градусів. Оптимізація роздільної здатності є порівняно малим кроком, який стискає великі зображення до найкращої роздільної здатності для Tesseract. Більшу частину часу витрачається на алгоритм налаштування та розширення масштабів у DPI. Вони є основними кроками, завдяки чому зображення готове до Tesseract.

2.2.1 Техніка освітленості

Техніка - це спосіб перетворення зображення в градації сірого, зі збереженням деякої інтенсивності кольорів. Техніка освітленості майже подібна до методу середнього кольору, але більш досконала для врахування сприйняття кольором людини.

Людське око чутливіше до певного кольору. Наприклад, воно найбільш чутливе до зеленого і найменш чутливийедо синього [3].

$Y = (0.2126*R + 0.7152*G + 0.0722*B)$,

де Y – інтенсивність сірого,

R – інтенсивність червоного,

G – інтенсивність зеленого,

B – інтенсивність синього.

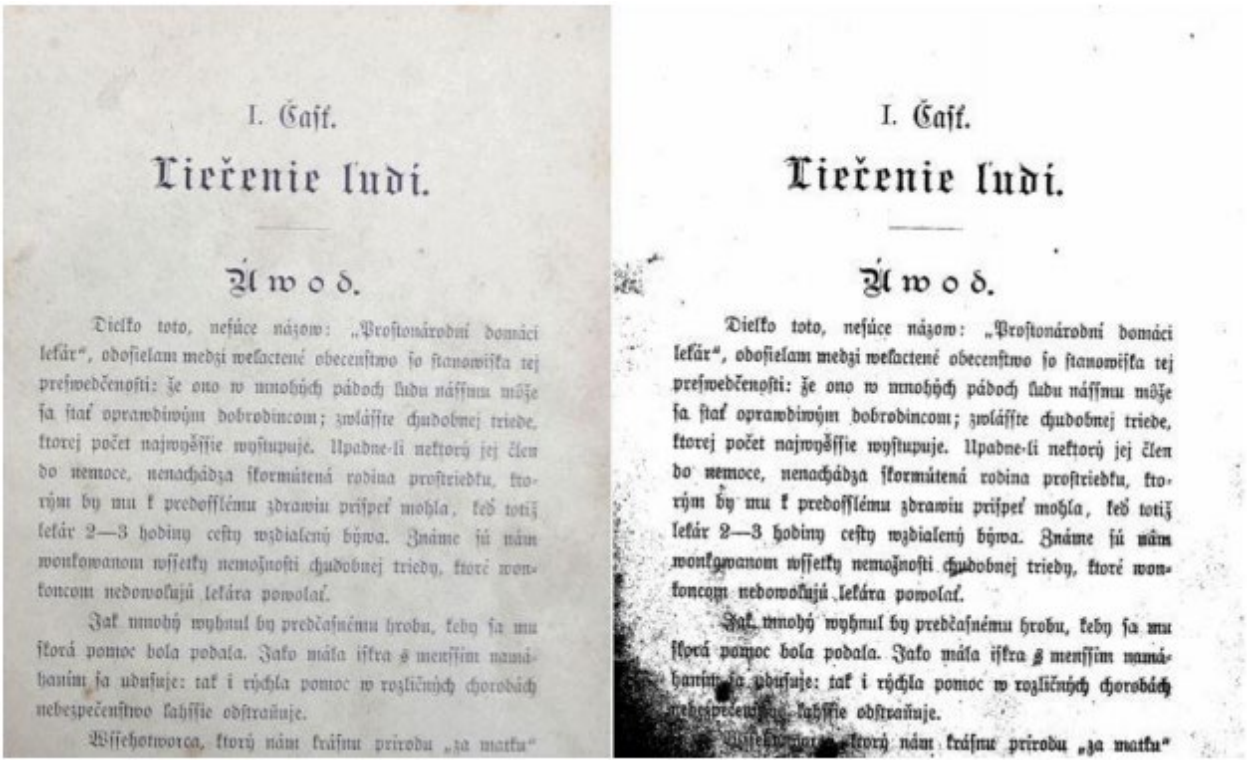
Кожен піксель перетворюється за допомогою наведеної вище формули. Нижче наведено зображення перетворених зображень з використанням обох методів масштабування сірого кольору.

Рисунок 2.11 - Вхідне зображення

Рисунок 2.12 - Вихідне зображення після техніки освітлення, порівнюючи з простим методом середнього кольору

2.2.2 Підвищення DPI

Щоб отримати найкращі результати на зображенні, також потрібно виправити DPI [4]. Тільки масштабування сірого спрацює лише в тому випадку, коли в зображенні немає спотворень, світлових ефектів [5]. Кроки, які потрібно врахувати при вдосконаленні DPI, - це: виправити DPI (за потреби), 300 DPI - мінімально прийнятний для Tesseract. Кращий діапазон DPI призводить до кращого процесу вилучення. На малюнку нижче описаний ефект посилення DPI (рисунок 2.13)



I. Časť.

Liečenie ľudi.

Úvod.

Dielko toto, nesúce názov: „Prostonárodní domáci lekár“, odosielam medzi vďačné obecenstvo so stanoviska tej presvedčenosti: že ono v mnohých pádoch ľudu nášmu môže sa stať opravdivým dobrodincom; zvlášte chudobnej triede, ktorej počet najvyššie vystupuje. Upadne-li niektorý jej člen do nemoce, nenachádza sformútená rodina prostriedku, ktorým by mu k predošlému zdraviu prispieť mohla, keď totiž lekár 2—3 hodiny cesty vzdialený býva. Známe sú nám vonkovanom všetky nemožnosti chudobnej triedy, ktoré vonkoncom nedovoľujú lekára povolať.

Jak mnohý vyhnul by predčasnému hrobu, keby sa mu skorá pomoc bola podala. Jako mála iskra s menším namáhaním sa udusuje: tak i rýchla pomoc v rozličných chorobách nebezpečenstvo ľahšie odstraňuje.

Všehotvorca, ktorý nám krásnu prírodu „za matku“

Рисунок 2.13 - Підвищення DPI

2.2.3 Поділ зображень на блоки (сегментація)

Оцінка результатів сегментації складніше, ніж оцінка результатів розпізнавання тексту. Перший підхід використовує звичайну основну правду тексту, в якій текст упорядковується відповідно до порядку читання. Потім програмне забезпечення OCR виконує сегментацію, виявлення порядку читання та нарешті розпізнавання символів. [22]

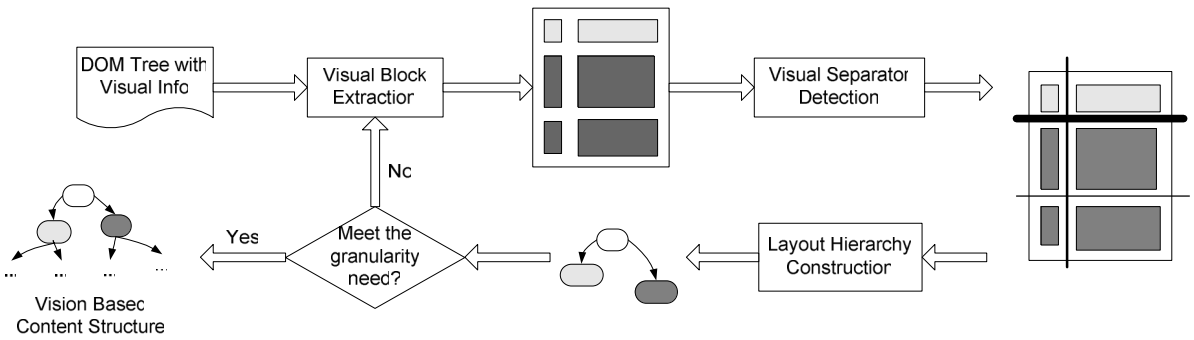
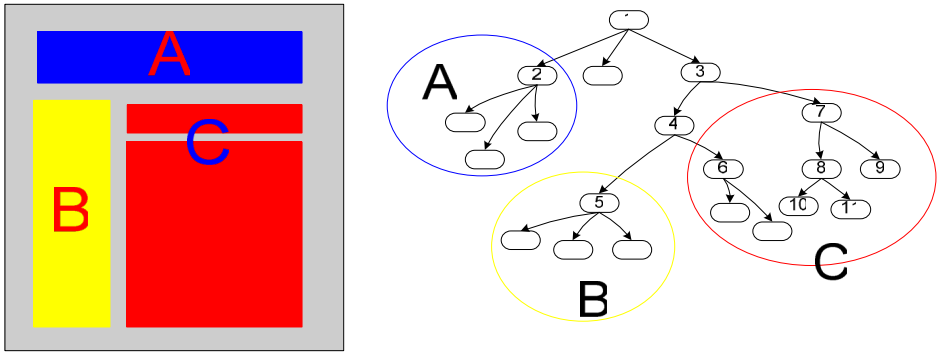


Рисунок 2.14 – Алгоритм сегментації сторінки [22]

Структура вмісту сторінки отримується шляхом поєднання структури DOM та візуальних підказок. Процес сегментації проілюстрований на рисунку 2.14. Він має три етапи: вилучення блоку, виявлення сепараторів та побудова структури контенту. Ці три етапи в цілому розглядаються як раунд. Алгоритм працює зверху вниз. Веб-сторінка спочатку сегментується на кілька великих блоків, і ієрархічна структура цього рівня записується. Для кожного великого блоку той же процес сегментації здійснюється рекурсивно, поки ми не отримаємо достатньо невеликі блоки.



(a) **(b)**

Рисунок 2.15 Деревоподібна структура сторінки [22]

Для кожного раунду дерево DOM з його візуальною інформацією (сторінка для першого раунду) отримується з веб-браузера, як ми показуємо на рисунку 2.15. Процес вилучення блоків починається з вилучення блоків з кореневих вузлів дерева DOM на основі візуальних підказок. Кожен вузол DOM перевіряється, чи утворює він єдиний блок чи ні. Якщо ні, його дочірні вузли будуть оброблятися аналогічно.

Коли всі блоки поточного раунду на поточній сторінці або підсторінці витягнуті, вони зберігаються. Розділювачі серед цих блоків ідентифікуються, а вага розділювача встановлюється на основі властивостей сусідніх блоків. Ієрархія компонування була побудована на основі цих розділювач. Після побудови ієрархії компонування поточного раунду кожен вузол сторінки структури вмісту перевіряється, чи відповідає він вимогам деталізації. Якщо

ні, цей вузол сторінки буде розглядатися як окрема сторінка і далі буде сегментований аналогічно. Наприклад, якщо блок C не відповідає вимозі, ми розглядаємо цей блок як окрему сторінку, і він буде додатково сегментований на дві частини, C1 і C2.

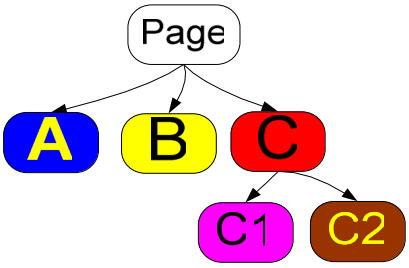


Рисунок 2.16 – Приклад структури сторінки

Після обробки всіх блоків виводиться остаточна структура вмісту веб-сторінки на основі бачення. У наведеному вище прикладі ми нарешті отримуємо структуру дерева вмісту, як показано на рисунку 2.16.

2.2.4 Розпізнавання тексту

Оцінка розпізнавання тексту зазвичай проводиться шляхом вирівнювання вихідного сигналу OCR із «основною правдою» та обчислення відстані Левенштейна, іноді його також називають дистанцією редагування, яка є мінімальною кількістю вставок, видалень та підстановок, необхідних для того, щоб обидва тексти були рівними, тобто кількість помилок. «Основна правда» — термін, що використовується в різних сферах для позначення інформації, що підтверджується шляхом безпосереднього спостереження (тобто емпіричних доказів) на відміну від інформації, наданої припущенням [1]. Тоді:

- Коефіцієнт помилок розпізнавання символів - це відношення кількості помилок до кількості всіх символів у тексті «основної істини».
- Точність розпізнавання символів - це відношення кількості правильно

розпізнаних символів до кількості всіх символів у тексті «основної істини».

Слово правильно розпізнається, якщо всі його букви були правильно розпізнані. При оцінці розпізнавання слів ігноруються пунктуація, цифри, інші спеціальні символи, а іноді навіть регістр. Точність слова є важливою метрикою для декількох випадків використання, наприклад, створення бази даних для системи пошуку інформації, де для отримання інформації, яка потрібна, потрібно ввести ціле слово. Ці системи часто не індексують все слово в тексті, натомість вони пропускають стоп-слова, які є дуже корисними для пошуку інформації, наприклад, артиклі ("a", "the"), прийменники ("of", "to") або сполучники ("and", "or").

- Точність розпізнавання без стоп-слів - це відношення кількості правильно розпізнаних слів, які не є стоп-словами, до кількості всіх слів у тексті основної істини, які не є стоп-словами.

Якщо дві системи мають аналогічну точність розпізнавання символів, але одна має більш високу точність розпізнавання слів, це означає, що помилки є більш сконцентрованими та їх легше виправити. Точність розпізнавання фраз для заданих довжин фрази і вказуватиме наскільки сконцентровані чи поширені помилки. Точність розпізнавання слів також можна наблизити методом bag-ofwords. Цей метод ігнорує порядок слів і записує лише кількість зустрічей кожного слова.

2.4 Нейронні мережі для розпізнавання текстів

Рекуре́нтні нейро́нні мере́жі (РНМ) — це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф орієнтований у часі. Це створює внутрішній стан мережі, що дозволяє їй проявляти динамічну

поведінку в часі. На відміну від нейронних мереж прямого поширення, РНМ можуть використовувати свою внутрішню пам'ять для обробки довільних послідовностей входів. Це робить їх застосовними до таких задач, як розпізнавання несегментованого неперервного рукописного тексту та розпізнавання мовлення. [3]

Рекурентні нейронні мережі насправді є рекурсивними нейронними мережами з певною структурою: такою, як в лінійного ланцюжка. В той час як рекурсивні нейронні мережі працюють на будь-якій ієрархічній структурі, поєднуючи дочірні представлення в батьківські, рекурентні нейронні мережі діють на лінійній послідовності часу, поєднуючи попередній такт і приховане представлення в представлення поточного такту.

Рекурентні нейронні мережі, зокрема, можна представляти як нелінійні версії фільтрів зі скінченною та нескінченною імпульсною характеристикою, а також як нелінійну авторегресійну екзогенну модель (англ. nonlinear autoregressive exogenous model, NARX).[11]

Дóвга короткочáсна пáм'ять (ДКЧП) — це архітектура рекурентних нейронних мереж (РНМ, штучна нейронна мережа), запропонована 1997 року Зеппом Хохрайтером та Юргеном Шмідгубером. Як і більшість РНМ, мережа ДКЧП є універсальною в тому сенсі, що за достатньої кількості вузлів мережі вона може обчислювати будь-що, що може обчислювати звичайний комп'ютер, за умови, що вона має належну матрицю вагових коефіцієнтів, що може розглядатися як її програма. На відміну від традиційних РНМ, мережа ДКЧП добре підходить для навчання з досвіду з метою класифікації, обробки або передбачення часових рядів в умовах, коли між важливими подіями існують часові затримки невідомої тривалості. Відносна нечутливість до довжини прогалин дає ДКЧП перевагу в численних застосуваннях над альтернативними РНМ, прихованими марковськими моделями та іншими методами навчання послідовностей. Серед інших успіхів, ДКЧП досягла найкращих з відомих результатів у стисненні тексту природною мовою, розпізнаванні несегментованого неперервного рукописного тексту, і 2009 року

виграла змагання з розпізнавання рукописного тексту ICDAR. Мережі ДКЧП також застосовувалися до автоматичного розпізнавання мовлення, і були головною складовою мережі, яка 2003 року досягла рекордного 17.7-відсоткового рівня фонемних похибок на класичному наборі даних природного мовлення TIMIT. Станом на 2016 рік основні технологічні компанії, включно з Google, Apple, Microsoft та Baidu, використовують мережі ДКЧП як основні складові нових продуктів. [9]

Мережа ДКЧП є штучною нейронною мережею, яка містить вузли ДКЧП замість, або на додачу, до інших вузлів мережі. Вузол ДКЧП — це вузол рекурентної нейронної мережі, який виділяється запам'ятовуванням значень для довгих, або коротких проміжків часу. Ключем до цієї здатності є те, що він не використовує функції активації в межах своїх рекурентних складових. Таким чином, значення, що зберігається, не розплющується ітеративно з плином часу, і член градієнту або вини не має схильності розмиватися, коли для його тренування застосовується зворотне поширення в часі.

Вузли ДКЧП часто втілюють у «блоках», які містять декілька вузлів ДКЧП. Така конструкція є типовою для «глибинних» багатошарових нейронних мереж, і сприяє реалізаціям на паралельному апаратному забезпеченні.

Блоки ДКЧП містять три або чотири «вентилі», які вони використовують для керування плином інформації до або з їхньої пам'яті. Ці вентилі реалізують із застосуванням логістичної функції для обчислення значень між 0 та 1. Для часткового дозволяння або заборони плину інформації до або з цієї пам'яті застосовується множення на це значення. Наприклад, «входовий вентиль» керує мірою, до якої нове значення входить до пам'яті. «Забувальний вентиль» керує мірою, до якої значення залишається в пам'яті. А «вихідний вентиль» керує мірою, до якої значення в пам'яті використовується для обчислення активування виходу блоку. (В деяких втіленнях входовий та забувальний вентилі об'єднують в один. Ідея їхнього об'єднання полягає в тому, що час забувати настає тоді, коли з'являється нове значення, варте запам'ятовування.)

Єдині ваги, що є в блоці ДКЧП (W та U), використовуються для спрямовування дії вентилів. Ці ваги застосовуються між значеннями, які надходять до блоку (включно з входовим вектором x_t та виходом з попереднього моменту часу h_{t-1}) та кожним із вентилів. Отже, блок ДКЧП визначає, яким чином підтримувати свою пам'ять як функцію від цих значень, і тренування ваг блока ДКЧП спричиняє його навчання такої функції, яка мінімізує втрати. Блоки ДКЧП зазвичай тренують за допомогою зворотного поширення в часі.[12]

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір мови/середовища програмування

Для розобки системи обрана мова програмування Python в першу чергу через наявну велику кількість бібліотек для аналізу та розпізнавання тексту, наприклад pytesseract.

Серед інших переваг Python:

- простий і зрозумілий синтаксис
- об'єктно-орієнтоване програмування
- підтримує імперативне та функціональне програмування
- велика стандартна бібліотека та багато сторонніх бібліотек
- підтримує декілька платформ (веб та мобільні обчислення)
- Python легко розширюється за допомогою коду C / C ++ / Java
- відкритий код та підтримка

Але є і недоліки:

- Python повільний (не має великого значення, оскільки основні обчислення проводяться в C++ бібліотеках)
- має обмеження в доступі до бази даних
- оскільки Python динамічний, під час виконання виявляється більше помилок

Середовищем розробки обраний PyCharm.

Pycharm - це інтегроване середовище розробки, розроблене JetBrains, використовується для програмування в Python. Воно відрізняється від конкурентів завдяки своїм інструментам підвищення продуктивності. Доступний у трьох версіях, серед яких є безкоштовна.

Безкоштовна версія має функції підсвічування синтаксису, автоматичного доповнення та перевірки коду в реальному часі. Платна версія, очевидно, має більш вдосконалені функції, такі як повне управління базами

даних і безліч бібліотек, таких як Django, Flask, Google App, Engine, Pyramid та web2py.

Переваги:

- активна підтримка
- перевірка коду та підсвічування синтаксису в реальному часі
- виконує правки та налагодження коду Python без будь-яких зовнішніх вимог

Серед недоліків повільний час завантаження.

3.2 Опис даних для навчання

Створення набору даних - непросте завдання. Існує два різних способи створення наборів даних - використання реальних даних та генерування синтетичних даних.

Перший підхід, як правило, передбачає набагато більше ручної роботи, другий вимагає багато розуміння, щоб створити точну модель для генерації даних. Часто ці підходи змішуються разом, щоб досягти гарного балансу, наприклад, реальні дані використовуються як основа для різних перетворень генерування нових даних або OCR використовується для отримання першої версії тексту. Процес складається з таких завдань:

1. Першою частиною набору даних є збір зображень документів. Для вибору даних слід враховувати два основні аспекти - реалістичність та репрезентативність проблемної області. Бути реалістичним означає, що зображення повинні бути максимально наближені до реальних зображень, тоді як репрезентативність означає, що набір даних повинен бути побудований з усіх класів документів, що належать до цієї проблемної області, збалансовано. Наприклад, набір даних для

розпізнавання символів повинен містити документи, що використовують різні комбінації типів шрифтів, розміри сценарію та інші параметри форматування, тоді як набір даних для тестування можливостей сегментації пакету OCR повинен складатися з документа з різними розмірами текстових областей різного розміру, таблиць та графіка. Для досягнення реалістичного та репрезентативного набору даних необхідно, щоб зразки містили спотворення, шум та інші погіршення зображень різного рівня, як і в сценаріях реального використання. Після визначення того, які типи зображень документа потрібно вибрати, збирається достатньо великий набір, де достатньо великий зв'язаний з представницькою ознакою, обговореною раніше.

2. Визначення «основної правди» та анотація / визначення - друга частина набору даних складається з «основної правди», пов'язаної з даними зображення. Характер даних «основної правди» дуже специфічний для даної області. Наприклад, текстові рядки потрібні для оцінки розпізнавання тексту, обмежувальні поля та багатокутники зазвичай використовуються в сегментації зон, кути нахилу для тестів автовиведення, передні пікселі для бінаризації тощо. Після цього всі картинки повинні бути помічені з даними «основної правди». Це дуже трудомістко і нудно, якщо робити це вручну, особливо якщо «основна правда» є складною. Ця проблема усувається, якщо використовувати синтетичні дані.
3. Організація та структурування набору даних - у кінцевому результаті зображення та дані «основної правди» збираються разом. Набір даних може бути відсортований / анотований відповідно до різних критеріїв, таких як мова, тип документа тощо.

3.2.1 Синтетичні та реальні дані

Існує два способи збору даних - збір реальних даних або генерування синтетичних даних. Використання реальних даних тривіально задовольняє

вимогу "бути реалістичним", але має ряд недоліків. По-перше, може бути важко зібрати достатню кількість реальних зображень, особливо якщо потрібно розглянути питання щодо ліцензування, авторських прав та конфіденційності, щоб набір даних був оприлюднений для загального користування. По-друге, як уже було сказано, анотація великих наборів - це дуже дорогий процес з точки зору людських зусиль. Крім того, він схильний до людських помилок. Щоб полегшити анотацію, було розроблено кілька інтерактивних та спільних інструментів.

Генерація синтетичних даних дозволяє генерувати довільну кількість даних разом з усіма необхідними принципами правдивості, що вимагає невеликих зусиль вручну. Однак розробка моделей для реалістичної та репрезентативної автоматичної генерації є дуже важкою проблемою, якщо домен широкий, і потрібно враховувати безліч факторів. Запропоновано декілька теоретичних моделей деградації. Невеликий прогрес у розробці методів генерації документів із різноманітними та реалістичними формами та форматуванням.

В даній роботі в якості датасету було обрано MJSynth, який складається з 9 мільйонів зображень з більш ніж 90 тис. англомовних слів на них.

Зображення в датасеті створені штучно в декілька етапів, які можна побачити на рисунку 3.1.

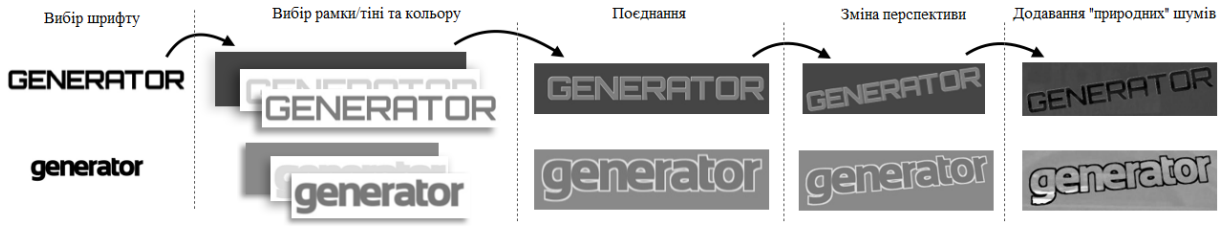


Рисунок 3.1 – Етапи створення елементів датасету

3.3 Згорткові як альтернативний інструмент розпізнавання текстів

Класичні моделі згорткових мереж є надто складними для навчання на персональному комп’ютері. Навіть при доволі високих параметрах компонент, спроба навчання архітектури VGG-16 на 100 епохах, з потрібних 5000, тривала більше 2 годин, а точність була в межах 10-13% [1]. Ця система виявилась не дієздатною для вибірки малого розміру, тому було вирішено за основу взяти VGG-11 – спрощену 11-шарову версію, зазначеної вище VGG-16 та звичайні згорткові мережі з різною кількістю та типами шарів, та різними параметрами.[14]

Таким чином, у вибірці взяли участь 3 різні архітектури системи:

1. Перша архітектура - 5 пар послідовних згорткових і агрегуючих шарів, після останньої агрегації застосовується функція виключення нейронів. Останній шар використовує функцію soft-max.
2. Друга архітектура - відповідає стандарту VGG-11, що можна побачити на рисунку 3.2.
3. Третя архітектура - це суміш попередніх архітектур, має п'ять блоків згорткових і агрегаційних шарів - перші три мають лише один згортковий шар, два інші по два. В архітектурі є два повнозв’яні шари. Це було зроблено для прискорення роботи системи.

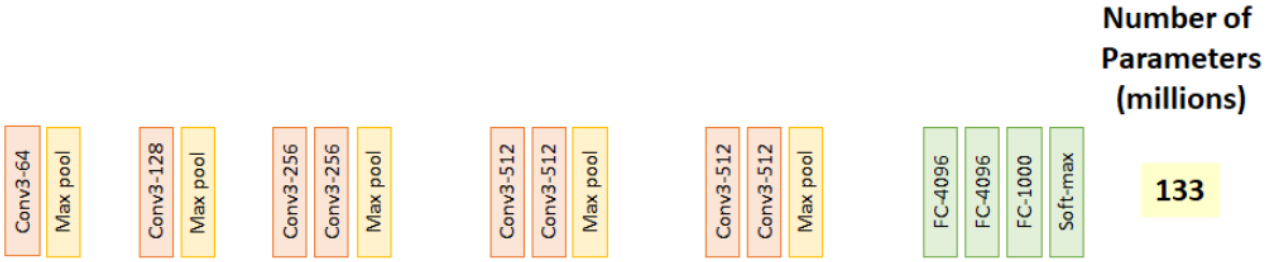


Рисунок 3.2 – Відображення архітектури VGG-11 [23]

Після проведення навчання протягом 5000 епох та валідації на тестовому зразку зображень було отримано порівняльну таблицю роботи, яка викладена нижче. Клітини показують час роботи останніх 100 епох і середню точність нейронної мережі на 100 епохах тренувань. (Таблиця 3.1)

Таблиця 3.1 – Порівняльна таблиця роботи структур нейронних мереж

Номер епохи	VGG-11	Архітектура №1	Архітектура №3
1	2.67s – 0.1246	2.35s – 0.0952	3.62s - 0.1182
100	86.7s – 0.3045	27.2s - 0.7969	33.1s - 0.7927
200	84.1s – 0.4931	22.7s – 0.8906	23.2s - 0.9794
500	89.7s – 0.6378	25.4s – 1.000	25.7s - 0.9817
1000	73.1s – 0.9175	38.0s – 1.000	26.7s – 1.000
1500	76.9s - 0.9252	21.6s – 1.000	21.7s – 0.9912
2000	74.8s - 0.9856	24.4s – 1.000	23.7s – 0.9437
2500	70.6s - 0.9856	26.25s – 1.000	20.6s – 0.9972
3000	73.4s – 1.000	24.5s – 1.000	22.4s – 1.000
3500	78.46s – 1.000	26.86s – 1.000	24.9s – 1.000
4000	83.4s – 0.9781	27.2s – 1.000	30.1s – 1.000
5000	73.7s – 1.000	26.34s – 1.000	25.3s – 1.000
Тестування	0.95723	**0.96972**	0.96345

Тому, залежно від даних, отриманих із таблиці, виберемо найбільш точну, тобто першу архітектуру, яка використовує функцію вилучення нейронів. Ця нейромережева структура показала один з найшвидших результатів навчання при меншому витраченому часу. Ця таблиця показує, що більш складні системи займають більше часу, а навчання відбувається повільніше. (Таблиця 3.2)

Перевіримо вибрану нейронну мережу натреновану на 1000, 2000, 5000 та 7000 епох, оскільки кількість епох може змінити точність системи. Ця мінливість дозволяє визначити, коли мережі потрібна додаткова підготовка та коли почати перекваліфікацію, тобто перекваліфікацію, як навчання. Результати навчання такі:

Таблиця 3.2 – Вплив кількості епох навчання на точність мережі

Кількість епох	1000	2000	5000	7000
Точність	0.95341	0.96234	**0.96972**	0.95468

Нові дані показують, що найбільш точна система з'являється в середньому значенні епохи навчання. На 1000 циклів нейронна мережа залишається недонавчанною, і з 7000 циклів точність знижується, коли система починає перенавчатися. Тому беремо значення 5000 як стандартне і побудуємо остаточну систему. (Таблиця 3.3)

При вивченні нейронних мереж важливий вибір оптимізатора. Оптимізатор поширює помилки по всій мережі та змінює значення параметрів під час кроку навчання. Бібліотека Tensorflow забезпечує готову реалізацію оптимізатора для використання в системі. Тому було вирішено перевірити працездатність різних оптимізаторів на конкретній структурі.

Таблиця 3.3 – Вплив вибору оптимізатора на точність мережі

Оптимізатор	Adamax	Momentum	SMORMS3	Nesterov momentum
Точність	**0.97301**	0.83803	0.90432	0.92387

Як видно з таблиці результатів, AdamaxOptimizer забезпечує найкращу точність. Завдання оптимізатора в системі - мінімізувати функцію помилок на конкретному етапі навчання, вказаному як параметр. Щоб уникнути паралічу системи, рекомендується з часом зменшити значення кроку навчання.

AdamaxOptimizer зберігає тренувальний крок для кожного вагового параметра та експоненціального середнього попереднього схилу. [15] Зараз оптимізатор є одним з найпопулярніших оптимізаторів нейронної мережі (рис. 3.3).

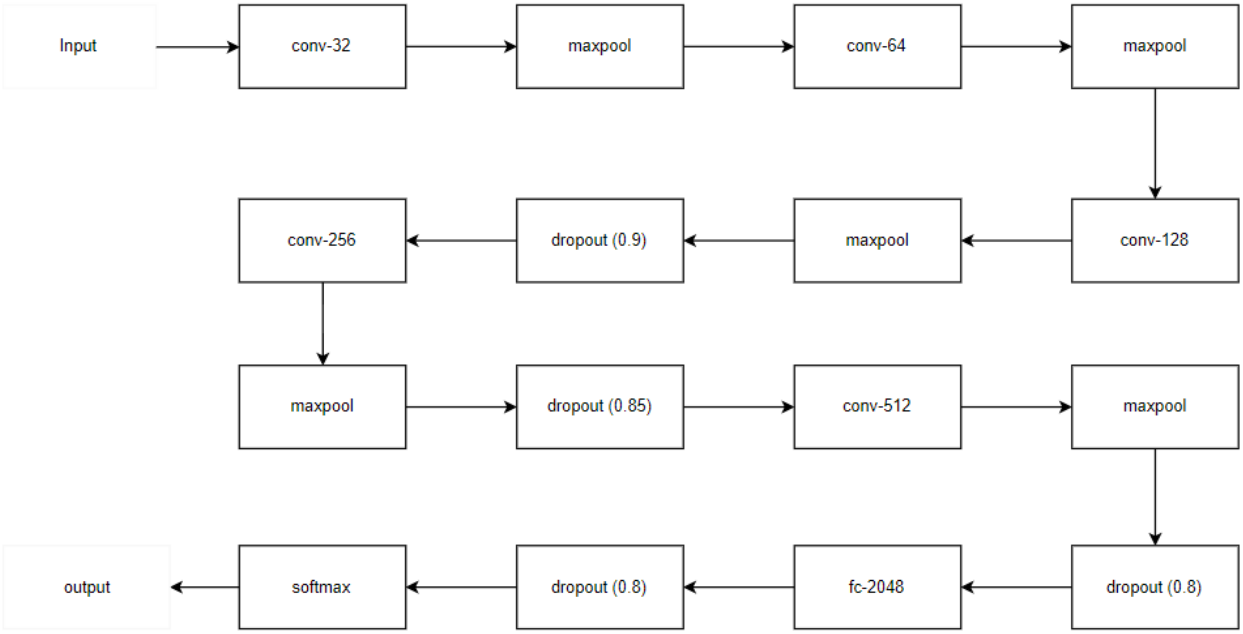


Рисунок 3.3 – Отримана структура системи VGG-11

Отже, отримана нейронна мережа має 7-шарову структуру, має величин навчання 5000 епох, використовуючи оптимізатор AdamaxOptimizer та дозволяє отримати точність 97.301% на тестовій вибірці.

3.4 Засоби python-tesseract

Python-tesseract - це інструмент оптичного розпізнавання символів (OCR) для python. Тобто він розпізнає та «прочитає» текст, вбудований у зображення. [18]

Python-tesseract - це обгортка для Google Tesseract-OCR Engine. Він також корисний як автономний скрипт, оскільки він може читати всі типи зображень, підтримувані бібліотеками зображень Pillow та Leptonica, включаючи jpeg, png, gif, bmp, tiff та інші. Крім того, якщо використовується

як скрипт, Python-tesseract виводить розпізнаний текст замість того, щоб записувати його у файл. [24]

Tesseract – система оптичного розпізнавання символів з відкритим кодом. Вона була створена компанією HP в 1984-1995 роки. Підтримується більшістю операційних систем. У 2006 році Google купив її та відкрив початковий код під ліцензією Apache 2.0 для продовження розробки. У цей час програма вже працює з UTF-8, розпізнає багато мов, серед яких і українська. [19]

Систему добре задокументовано. Tesseract написаний на C / C ++. Інструкція з установки досить вичерпна. Tesseract поверне результати у вигляді звичайного тексту, hOCR або у форматі PDF, з текстом, накладеним на вихідне зображення.

Основні функції бібліотеки:

get_tesseract_version: Повертає версію Tesseract, встановлену в системі.

image_to_string: Повертає результат запуску Тессеракт OCR на зображенні.

image_to_boxes: Повертає результат, що містить розпізнані символи та їх межі на зображенні.

image_to_data: Повертає результат, що містить межі поля, конфіденційність та іншу інформацію. Потрібна Tesseract версії 3.05+.

image_to_osd: Повертає результат, що містить інформацію про орієнтацію та виявлення тексту. [20]

Python-tesseract є ефективною бібліотекою, серед головних переваг якої є натренована модель для позпізнавання текстів, швидкість роботи, відкритий код, легкість розробки на мові python, та висока точність. Саме тому для розробки системи автоматизованого візуального тестування була обрана бібліотека python-tesseract.

3.5 Опис роботи системи

Для запуску системи в ручному режимі необхідно вказати назви двох файлів зображень та шлях до зображення, куди буде записана візуальна різниця.

Спочатку, протестуємо систему на двох однакових зображеннях, передавши одне зображення (рисунок 3.4) двічі:

Theory

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensity denotes valleys. You start filling every isolated valleys (local minima) with different colored water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colors will start to merge. To avoid that, you build barriers in the locations where water merges. You continue the work of filling water and building barriers until all the peaks are under water. Then the barriers you created gives you the segmentation result. This is the "philosophy" behind the watershed. You can visit the CMM webpage on watershed to understand it with the help of some animations.

But this approach gives you oversegmented result due to noise or any other irregularities in the image. So OpenCV implemented a marker-based watershed algorithm where you specify which are all valley points are to be merged and which are not. It is an interactive image segmentation. What we do is to give different labels for our object we know. Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm. Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

Code

Below we will see an example on how to use the Distance Transform along with watershed to segment mutually touching objects.

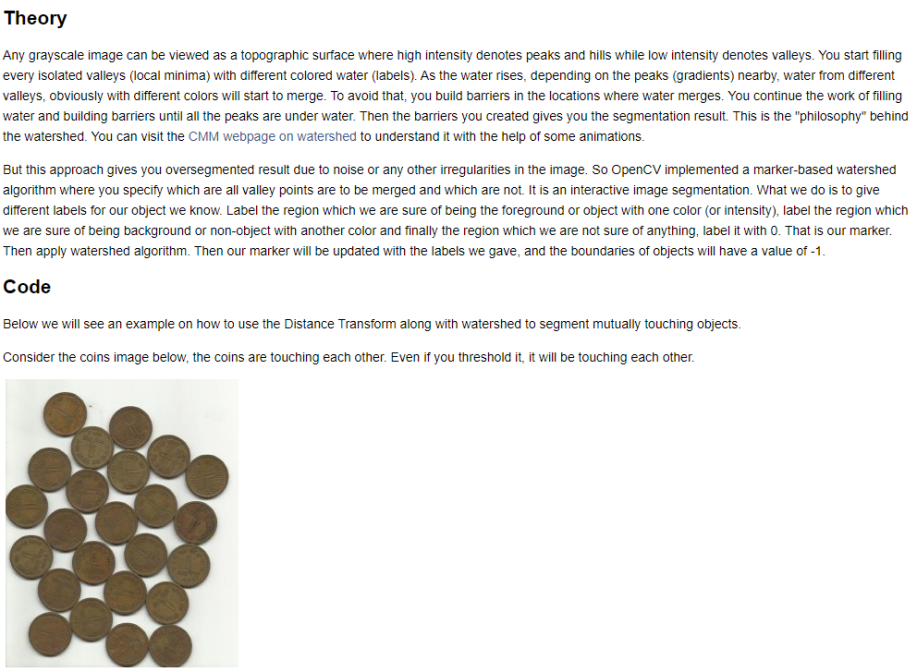
Consider the coins image below, the coins are touching each other. Even if you threshold it, it will be touching each other.

Рисунок 3.4 – файл expected.png

Як і очікувалось, система показала 0 відмінностей між зображеннями як в текстовому змісті, так і в нетекстових сегментах. Вивід програми можна побачити на рисунку 3.5. Зображення відмінностей result.png в цьому випадку залишилось пустим.

Рисунок 3.5 – Результат роботи програми на двох однакових зображеннях.

Спробуємо перевирити систему в роботі з візуальними відмінностями в зображеннях. Для цього, подамо на вхід сторінку, зі зміщенним зображенням як на рисунку 3.6.



Theory

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensity denotes valleys. You start filling every isolated valleys (local minima) with different colored water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colors will start to merge. To avoid that, you build barriers in the locations where water merges. You continue the work of filling water and building barriers until all the peaks are under water. Then the barriers you created gives you the segmentation result. This is the "philosophy" behind the watershed. You can visit the CMM webpage on watershed to understand it with the help of some animations.

But this approach gives you oversegmented result due to noise or any other irregularities in the image. So OpenCV implemented a marker-based watershed algorithm where you specify which are all valley points are to be merged and which are not. It is an interactive image segmentation. What we do is to give different labels for our object we know. Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm. Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

Code

Below we will see an example on how to use the Distance Transform along with watershed to segment mutually touching objects.

Consider the coins image below, the coins are touching each other. Even if you threshold it, it will be touching each other.

Рисунок 3.6 – Зміщене зображення в тестовому файлі.

В цьому вирадку система виводить кількість кількість відмінних пікселей (рисунок 3.7) як результат, тож ми можемо з впевненістю стверджувати, що вхідні зображення не є еквівалентними.

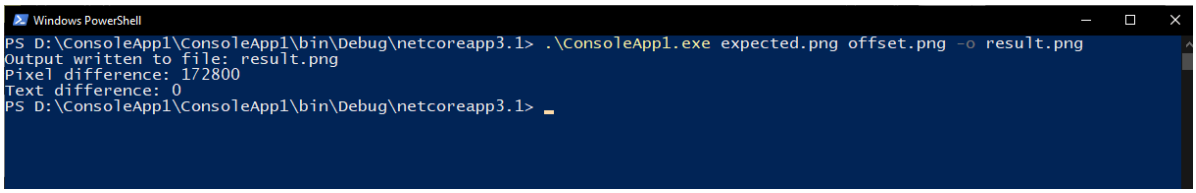


Рисунок 3.7 – вивід програми для файлу зі зміщенням.

У вихідному файлі result.png можемо перевірити попіксельну різницю між двома вхідними файлами, що допомагає швидко визначити зміни в інтерфейсі тестованого додатку (рисунок 3.8).

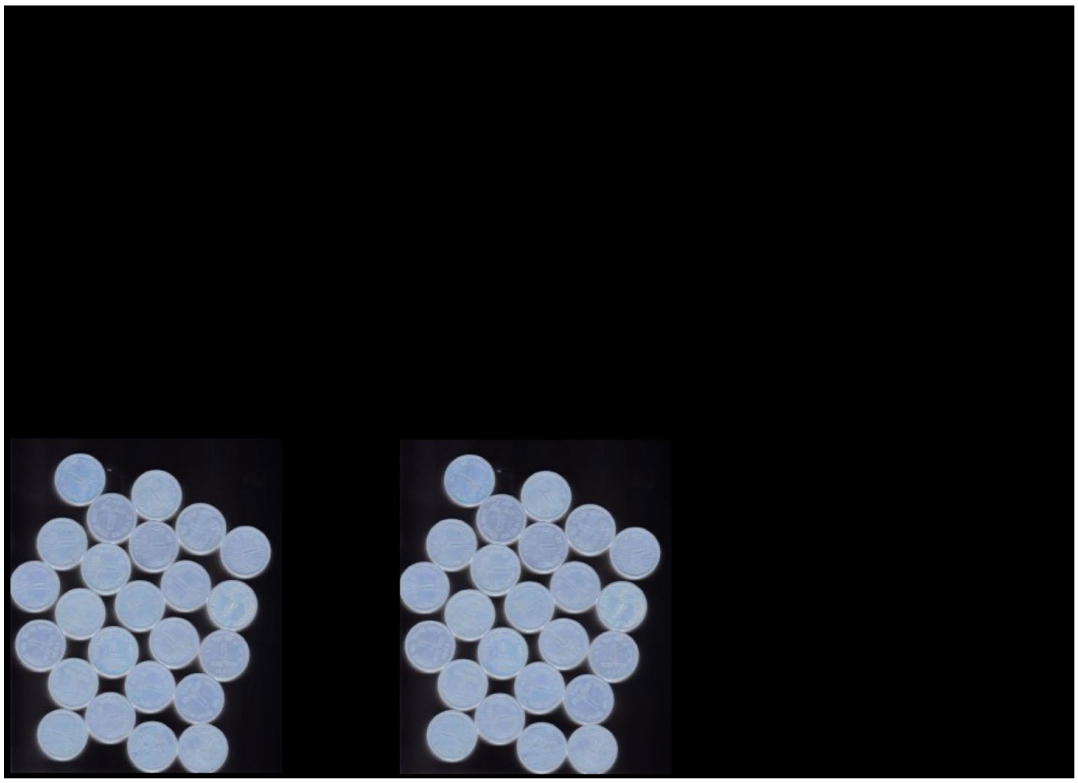
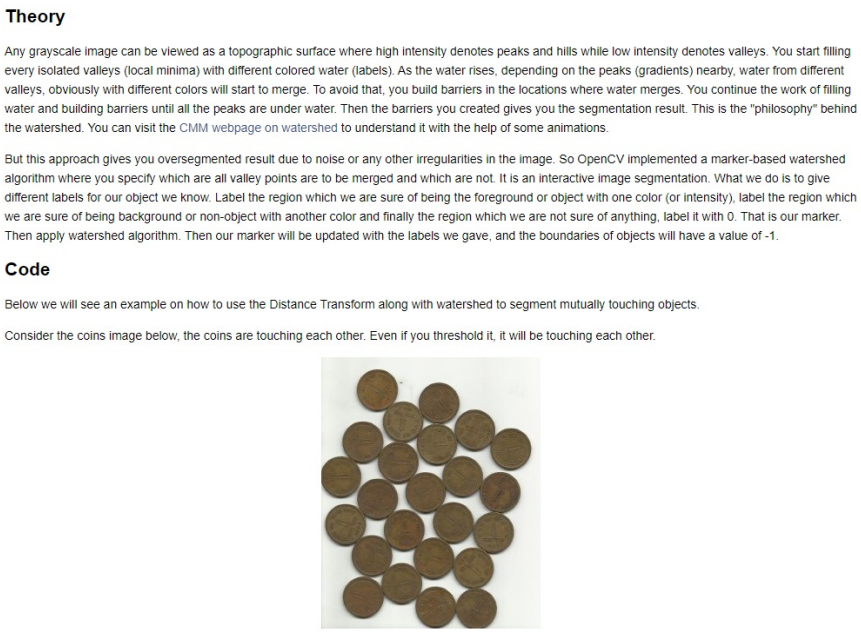


Рисунок 3.8 – файл-результат роботи програми.

Перевіримо головний функціонал програми – порівнянння зображення з відмінними налаштуваннями згладжування шрифтів. Для цього візьмемо зображення з увімкненим згладжуванням (рисунок 3.9) .



Theory

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensity denotes valleys. You start filling every isolated valleys (local minima) with different colored water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colors will start to merge. To avoid that, you build barriers in the locations where water merges. You continue the work of filling water and building barriers until all the peaks are under water. Then the barriers you created gives you the segmentation result. This is the "philosophy" behind the watershed. You can visit the CMM webpage on watershed to understand it with the help of some animations.

But this approach gives you oversegmented result due to noise or any other irregularities in the image. So OpenCV implemented a marker-based watershed algorithm where you specify which are all valley points are to be merged and which are not. It is an interactive image segmentation. What we do is to give different labels for our object we know. Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm. Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

Code

Below we will see an example on how to use the Distance Transform along with watershed to segment mutually touching objects.

Consider the coins image below, the coins are touching each other. Even if you threshold it, it will be touching each other.

Рисунок 3.9 – зображення з увімкненим згладжуванням тексту.

Завдяки тому, що текстові сегменти порівнюються виключно за змістом, не враховуючи різницю кольорів пікселів, та наявний текст повністю співпадає на обох зображеннях, програма вивела корректний результат, що можна вважати ці зображення еквівалентними.

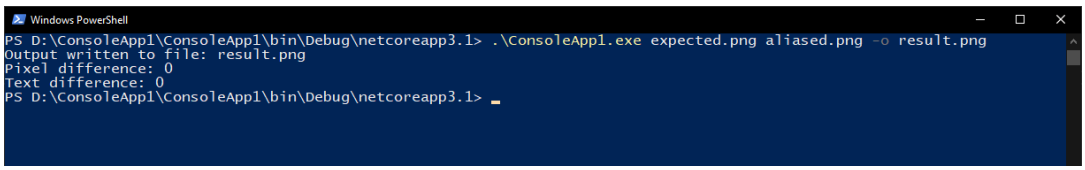


Рисунок 3.10 – результат порівняння зображення зі згладжуванням

Одним із головних механізмів системи є виділення текстових блоків на зображеннях. Саме завдяки ньому, система здатна порівнювати окремо текстові та нетекстові частини документу для окремого порівняння. На рисунку 3.11 можна бачити, що система корректно виконала поділ зображення на блоки.

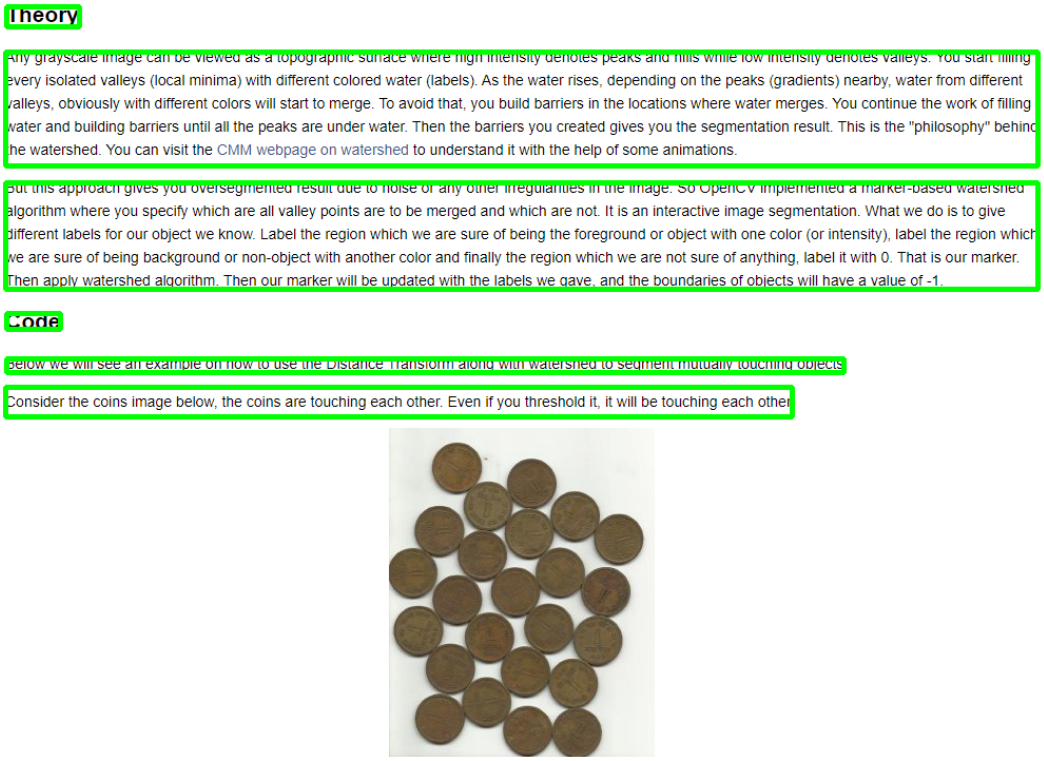


Рисунок 3.11 Результат виділення текстових блоків на зображенні

ВИСНОВКИ

Робота присвячена задачі автоматизованого візуального тестування.

Актуальність задачі полягає в наявності потреби в удосконаленні систем візуального тестування в роботі з текстом зі згладжуванням та доволі стрімкому розповсюдженні нейронних мереж в усіх сферах життєдіяльності.

Метою дипломної роботи було створення системи автоматизованого візуального тестування текстових компонентів інтерфейсів застосунків.

У першому розділі була розглянута загальна теорія розпізнавання зображень. Був наведений аналіз актуальності задачі розпізнавання та порівняння існуючих підходів та обрано метод, який найкраще підходить для нашої задачі. Також були проаналізовані сучасні бібліотеки для роботи з нейронними мережами, одна з яких була використана у даній дипломній роботі. Приклади використання систем розпізнавання тексту охопили сферу безпеки, освіти та соціальних мереж.

У другому розділі більш детально було розглянуто поставлену задачу. Було приведено критерії оцінювання якості роботи, серед них найпопулярніший – точність нейронної мережі. Було розглянуто різні алгоритми розпізнавання тексту та їх етапи, один з яких і представлений в даній роботі.

Третій розділ присвячений практичній реалізації програмного продукту. Було проведено глибокий порівняльний аналіз роботи різних архітектур нейронної мережі, поміж них вибрана найкраща для подальшої роботи. Також проаналізовано точність розпізнавання при навчанні мережі на різній кількості епох та при використанні різних представлених у бібліотеці оптимізаторів. Після чого наведено схеми роботи готового програмного продукту та результати його роботи.

Система здатна проводити порівняння зображень з еталоном з високою точністю. У подальшому модуль можна розширити на більшу кількість шрифтів для тексту та реалізувати паралельну обробку символів чи тексту на одному зображенні.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ground truth [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Ground_truth
2. Text Recognition Pane [Електронний ресурс] – Режим доступу до ресурсу: https://admhelp.microfocus.com/uft/en/15.0-15.0.1/UFT_Help/Content/User_Guide/Text_Recognition_Options.htm
3. Sav.(2012) [Електронний ресурс] – Режим доступу до ресурсу: http://www.savthecoder.com/blog/index.php?page=9
4. Dots per inch [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Dots_per_inch
5. GOOGLE. Google Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.google.com/p/tesseractocr/wiki/ImproveQuality
6. Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/235956427_Optical_Character_Recognition_by_Open_source_OCR_Tool_Tesseract_A_Case_Study
7. Our Search for the Best OCR Tool, and What We Found [Електронний ресурс] – Режим доступу до ресурсу: https://source.opennews.org/articles/so-many-ocr-options/
8. ABBYY FineReader [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/ABBYY_FineReader
9. Michael A. Nielsen. Neural Networks and Deep Learning. URL: http://neuralnetworksanddeeplearning.com
10. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. Москва: Мир, 1992. 184 с.
11. Іван Гудфелов, Йоша Бенгіо, Арон Коурвілле. «Машинне навчання» [Електронний ресурс] – Режим доступу до ресурсу: : http://www.deeplearningbook.org.

12. Кононюк А.Ю. Нейронні мережі і генетичні алгоритми. Київ: Корнійчук, 2008. 446 с.
13. Савчук Т.О., Ярема Є.О. Використання нейронних мереж для розпізнавання символів. Вінниця: Вінницький національний технічний університет, 2005. 84 с.
14. Zaccone G. Deep Learning with TensorFlow. Birmingham: Packt Publishing, 2017. 300p.
15. Местецький Л.М. Математичні методи розпізнавання образів. Москва: МГУ, 2004. 239 с.
16. Apply computer vision in GUI automation for industrial applications, Yung-Pin Cheng, Ching-Wei Li and Yi-Cheng Chen, 2019
17. Automated system tests with image recognition, Moa Eriksson, Oskar Olsson, 2019
18. Improving the efficiency of tesseract ocr engine, Sahil Badla, 2014
19. Suitability of OCR Engines in Information Extraction Systems, Zacharias Erlandsson, 2019
20. Optical Character and Symbol Recognition using Tesseract, Victor Ohlsson, 2016
21. Evaluation of off-the-shelf OCR technologies, Martin Tomaschek, 2017
22. Improving Pseudo-Relevance Feedback in Web Information Retrieval Using Web Page Segmentation [Електронний ресурс] – Режим доступу до ресурсу: http://www2003.org/cdrom/papers/refereed/p300/p300-Yu.html
23. Review: VGGNet — 1st Runner-Up (Image Classification), Winner (Localization) in ILSVRC 2014 [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11
24. An Overview of the Tesseract OCR Engine [Електронний ресурс] – Режим доступу до ресурсу: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf

ДОДАТОК

Додаток А. Лістинг програми

```
try:
    from PIL import Image
except ImportError:
    import Image
import pytesseract
import cv2

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

print(pytesseract.get_tesseract_version())

image_path = r"C:\Users\Dima\Desktop\Untitled.png"
image_path_result = r"C:\Users\Dima\Desktop\Untitled2.png"
test = pytesseract.image_to_data(image_path)

image = cv2.imread(image_path)


blocks = {}

for line in test.split('\n')[1:]:
    fields = line.split('\t')
    field = lambda i: int(fields[i])
    block = field(2)
    left = field(6)
    top = field(7)
```

```
    if left == 0 and top == 0:
        continue

    width = field(8)
    height = field(9)
    right = left + width
    bottom = top + height
    if block not in blocks:
        blocks[block] = left, top, right, bottom
        continue

    current_left, current_top, current_right, current_bottom = blocks[block]
    if left < current_left:
        current_left = left
    if right > current_right:
        current_right = right
    if top < current_top:
        current_top = top
    if bottom > current_bottom:
        current_bottom = bottom

    blocks[block] = current_left, current_top, current_right, current_bottom

for left, top, right, bottom in blocks.values():
    cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 3)

cv2.imwrite(image_path_result, image)

print(test)
```