Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

Програмний засіб оптичного розпізнавання символів та структуризації текстових документів

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав студент 2 курсу, групи 1КІ-22мс
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма — «Комп'ютерна інженерія»
(шифр і назва напряму підготовки, спеціальності)
Супрун П. С.
(прізвище та ініціали)
Керівник: PhD, стар. викл. каф. ОТ
Обертюх М. Р
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф ПЗ
Коваленко О. О.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
« 11 » червня 2024 р.

ВНТУ 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — бакалавр
Спеціальність — 123 Комп’ютерна інженерія
Освітня програма — Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
яЗавідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
"06" лютого 2024 р.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Супруну Павлу Сергійовичу

1 Тема роботи «Технологія оптичного розпізнавання друкованих символів на основі нейронної мережі» керівник роботи Обертюх Максим Романович к.т.н., доц. каф. ОТ, затверджено наказом вищого навчального закладу від «11» березня 2024 року № 80.

2 Строк подання студентом роботи: 11.06.24.

3 Вихідні дані до роботи: роздільна здатність зображення тексту не менше 300 dpi, модель кольорів для представлення зображення — RGB, кількість градацій яскравості зображення — 256.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз методів та засобів розпізнавання друкованих символів текстових документів, розробка технології та вибір засобів розпізнавання друкованих символів, розробка програмних засобів розпізнавання друкованих символів, тестування та оцінка ефективності функціонування створеної програми.

5 Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень): методи розпізнавання тексту, системи оптичного розпізнавання тексту, структура програми розпізнавання символів, структура програми розпізнавання символів.

м6 Консультанти розділів проекту (роботи)

Таблиця 1— Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к. т. н., доц. Обертюх Максим Романович		

7 Дата видачі завдання 12.03.24

8 Календарний план виконання БДР приведений в таблиці 2.

Таблиця 2— Календарний план

№ з/п	Назва етапів проекту (роботи)	Строк виконання етапів проекту (роботи)		Примітка
1	Пошук та огляд інформаційних джерел	28.03.2024	02.04.2024	виконано
2	Огляд та аналіз існуючих методів і засобів виділення і розпізнавання друкованих символів	03.04.2024	16.04.2024	виконано
3	Результати проведених досліджень та методів виділення і розпізнавання друкованих символів	17.04.2024	28.04.2024	виконано
4	Проектування нейромережевої технології виділення і розпізнавання друкованих символів	29.04.2024	14.05.2024	виконано
5	Перевірка ефективності роботи створення програми виділення і розпізнавання друкованих символів	17.05.2024	19.05.2024	виконано
6	Оформлення пояснювальної записки і презентації	20.05.2024	31.05.2024	виконано
9	Перевірка якості виконання бакалаврської роботи та усунення недоліків	01.05.2024	07.06.2024	виконано

Студент  Павло СУПРУН

Керівник роботи  Максим ОБЕРТЮХ

АНОТАЦІЯ

УДК 004.93

Супрун П. С. Програмний засіб оптичного розпізнавання символів та структуризації текстових документів. Бакалаврська дипломна робота зі спеціальності 123 — Комп’ютерна інженерія, освітня програма — Комп’ютерна інженерія. Вінниця: ВНТУ, 2024.

На укр. мові. Бібліогр.:26 назв; рис.: 8, табл. 1.

У бакалаврській дипломній роботі розроблено програмні засоби оптичного розпізнавання друкованих символів та структуризації тексту у документах. У роботі зроблено аналіз методів оптичного розпізнавання символів, запропоновано здійснювати розпізнавання символів із використанням архітектури нейронної мережі Faster R-CNN, а обробку документів та структуризацію тексту за допомогою засобів NumPy та Pillow. В роботі розроблено послідовність обробки текстового документу для розпізнавання друкованих символів та програмне забезпечення, що його реалізує.

Ключові слова: розпізнавання символів, структуризація текстових документів, згорткова нейронна мережа.

ABSTRACT

Suprun S. Optical character recognition and text document structuring software Bacherlor’s thesis in specialty 123 — Computer Engineering, computer engineering education program. Vinnytsia: VNTU, 2024, 80 p.

In Ukrainian language. Bibliographer: 26 titles, fig: 8, table 1.

In this bachelor’s qualification work described the process of developing an optical character recognition and text document structuring system. The thesis includes an analysis of optical character recognition methods and proposes the use of the Faster R-CNN neural network architecture for character recognition. Document processing and text structuring are implemented using NumPy and Pillow libraries. The work also outlines a sequence for processing a text document to recognize printed characters and develops the software that implements it.

Keywords: character recognition, text document structuring, convolutional neural network

ЗМІСТ

ВСТУП.....8

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ОПТИЧНОГО РОЗПІЗНАВАННЯ СИМВОЛІВ.....11

1.1 Загальний огляд наявних технологій нейронних мереж 11

1.2 Проблеми та виклики 13

1.3 Аналіз засобів розпізнавання текстових документів..... 14

2 РОЗРОБКА ПОСЛІДОВНОСТІ ТА ВИБІР ЗАСОБІВ РОЗПІЗНАВАННЯ СИМВОЛІВ.....21

2.1 Етапи розпізнавання друкованих символів 21

2.2 Розробка послідовності розпізнавання символів..... 27

2.3 Використання згорткової нейронної мережі для розпізнавання..... 32

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РОЗПІЗНАВАННЯ ДРУКОВАНИХ СИМВОЛІВ ТА ЇХ СТРУКТУРИЗАЦІЇ.....41

3.1 Розробка архітектури модуля розпізнавання символів та структуризації .. 41

3.2 Попередня обробка текстових документів 42

3.3 Розробка архітектури нейронної мережі 44

3.4 Тренування нейронної мережі 49

3.5 Створення модуля структуризації документу..... 54

3.6 Тестування програми розпізнавання..... 56

ВИСНОВКИ.....59

ДОДАТОК А Технічне завдання 64

ДОДАТОК Б Лістинг модуля vgg16.py.....68

ДОДАТОК В Лістинг модуля train.py.....70

ДОДАТОК Г Лістинг модуля faster_rcnn.py 77

ДОДАТОК Е Послідовність оптичного розпізнавання та структуризації тексту 82

ДОДАТОК Ж Модель процесу розпізнавання символів.....83

ДОДАТОК И Архітектура згорткової нейронної мережі.....84

ДОДАТОК К Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень.....85

ВСТУП

Системи оптичного розпізнавання символів (Optical Character Recognition, OCR-системи) є одними з найбільш сучасних і затребуваних технологій на основі штучного інтелекту для оптичного розпізнавання об’єктів на зображеннях. За допомогою спеціально розроблених програм система OCR може автоматично розпізнавати зображення друкованого або рукописного тексту та перевести його в формат, який можна використовувати для подальшої обробки текстовими редакторами або текстовими процесорами. OCR іноді означає пристрій для оптичного розпізнавання символів або автоматичного зчитування тексту [1-2].

Використання сучасних технологій обробки та передачі даних значно спрощує доступ до різноманітних інформаційних ресурсів, накопичених людством, за умови, що ці ресурси можна перевести та передати в більш зручний електронний формат. Найбільш швидким і простим способом сканування документів є використання сканерів; після завершення процесу документ отримується у вигляді графічного файлу. Текстове подання даних, з іншого боку, простіше маніпулювати та обробляти. Цей метод дозволяє аналізувати електронні документи та значно знижує витрати на зберігання та передачу даних. Отже, переведення та подання паперових документів у текстові електронні документи є дуже важливим з практичної точки зору.

Тим не менш, розробка нових програмних засобів у цій галузі залишається творчим завданням і вимагає додаткових досліджень через специфічні вимоги до вирішення задач: швидкодії, надійності розпізнавання та обсягів пам’яті [3].

Розмаїття методів вирішення символів у дослідженнях виявлення та розпізнавання символів у друкованих текстах є унікальним. Виявлення та розпізнавання специфічних друкованих символів є традиційними завданнями для цього процесу обробки текстових документів [4].

Завдання автоматизованого розпізнавання друкованих текстів із можливими дефектами стає значно складнішим та проблемним. Це завдання є однією з найбільш **актуальних** і складних задач розпізнавання текстових документів.

Метою дослідження бакалаврської роботи є вдосконалення підходів до оптичного розпізнавання друкованих символів із використанням нейронної мережі.

Задачі дослідження бакалаврської роботи:

— провести аналіз відомих методів оптичного оброблення зображень текстових документів для пошуку й розпізнавання друкованих символів;

— запропонувати поліпшену послідовність обробки зображень текстових документів для виділення й подальшого розпізнавання друкованих символів;

— сформувати послідовність та розробити програму обробки зображення для розпізнавання друкованих символів текстових документів.

Об’єктом дослідження бакалаврської роботи є процес отримання структурованого тексту документу шляхом розпізнавання у виділеній сцені його зображення й розпізнавання друкованих символів.

Предметом дослідження бакалаврської роботи є методи обробки зображень текстових документів для виділення й розпізнавання друкованих символів.

Методи дослідження бакалаврської роботи: метод порівняння та аналогій, аналітичний огляд, метод моделювання та метод об’єктно-орієнтованого програмування. Для створення множини ознак для розпізнавання використано теорію множин, математична статистика для аналізу отриманих результатів розпізнавання друкованих символів. У цій роботі запропонований підхід реалізовано за допомогою принципів об’єктно-орієнтованого програмування.

Новизна отриманих результатів бакалаврської роботи стосується вдосконалених технологій обробки цифрових зображень у текстових документах, які дозволяють виділити, розпізнавати та структурувати друковані символи в текстових документах. Ці методи відрізняються від існуючих методів, які послідовно виконують етапи обробки зображення, використовують нейронні мережі з методом пропонування регіонів інтересу, що підвищує точність розпізнавання, структурують друковані символи в повноцінний макет.

Практичне значення результатів бакалаврської роботи: розроблено цикл обробки цифрових зображень, який зміг виявляти та розпізнавати друковані

символи в текстових документах; розроблено програму обробки зображень, що виявляє та розпізнає друковані символи в текстових документах.

Апробація результатів бакалаврської роботи: зроблено доповідь на Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)».

За результати бакалаврської роботи **опубліковано** 1 наукову працю [5].

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ОПТИЧНОГО РОЗПІЗНАВАННЯ СИМВОЛІВ

У цьому розділі бакалаврської роботи проведено аналіз методів розпізнавання друкованих символів текстових документів та розглянуто засоби, що застосовуються для здійснення дій по розпізнаванню тексту.

1.1 Загальний огляд наявних технологій нейронних мереж

На сьогоднішній день створення нейронних мереж є доволі прогресивною галуззю, за допомогою штучного інтелекту лікарі можуть отримати допомогу у встановленні діагнозу чи виявлення аномалій невидимих для людського ока, у будівництві нейронні мережі можуть оцінити якість ґрунту, розрахувати економічні втрати та прибутки будівництва, також дані технології використовуються при аналізі фінансових ринків, наявних тенденцій та можливих передбачень про майбутній стан економіки.

Наразі одною з найбільш актуальних проблем є оцифрування документообігу, саме вирішення частини даної проблеми є тема даної бакалаврської дипломної роботи. Завдяки оптичному розпізнаванню символів можливо легко оцифровувати великі об’єми текстових документів різних форматів, структуризація ж дозволить зберегти схему документу, що також дозволяє продовжувати подальші маніпуляції над документами, модифікування, додавання або шифрування.

Дана тема бакалаврської роботи є особливо актуальної в умовах війни, адже оцифровані документи можливо зберігати у будь-якій точці світу, що ускладнить фізичне знищення важливої інформації.

Головна задача цієї бакалаврської роботи — оптичне розпізнавання та структуризація тексту документів різних форматів.

При створені даного проекту, було використано хмарний сервіс для віддалених обчислень Google Colab, що робить процес створення продукту швидшим та дешевшим.

У якості архітектури основи застосунку була обрана Faster R-CNN, що дозволяє знаходити необхідні об’єкти на зображеннях, класифікувати їх та отримувати обмежувальні рамки розпізнаних об’єктів. Також дана архітектура вирізняється своєю швидкодією, що позитивно впливає на загальний час обробки отриманого документу.

У сьогоденні двома найбільш розповсюдженими напрямками розробки нейронних мереж є два протилежних підходи. Перший підхід має на меті створювати все більш складні, точні та інформативні нейронні мережі. Дана методика зазвичай застосовується у обробці живої мови (Natural Language Processing), де більша кількість даних та параметрів позитивно впливає на відповідь нейронної мережі, прикладами успішного застосування даного підходу є моделі типу GPT, Gemini, Perplexity і т. д.

Інший же підхід, навпаки, намагається забезпечити максимальну компактність та швидкодію при відносно мізерній втраті у точності, а інколи навіть у прирості якості вихідних результатів. Найбільш показовим прикладом є розвиток покоління моделей типу YOLO та R-CNN, де з кожною новою версією даних архітектур — швидкодія та точність нейронної мережі збільшується.

Серед наявних проблем, що вирішуються нейронними мережами можна виділити такі 5 основних напрямків.

Natural Language Processing, або ж обробка живої мови, даний підхід націлений на обробку різних мов, переклад, класифікацію та генерацію тексту. Найбільш популярними інструментами у вирішенні цієї задачі є моделі типу LSTM (Long-Short Term Memory), Embeddings, Transformer, LLM (Large Language Model) та ін.

Convolutional Neural Network, або конволюційні нейронні мережі, мережі даного типу націлені на обробку зображень та відео, з подальшим розпізнаванням об’єктів або класифікацією зображення. Також популярним методом обробки зображення є сегментація, при якому ми отримуємо точну форму розпізнаного об’єкту завдяки класифікації кожного окремого пікселя на зображенні, проте даний

підхід не завжди є доцільним, адже потребує значних ресурсів, або надлишковим, у випадках коли немає необхідності отримувати точні межі розпізнаних об’єктів.

Регресії, моделі даного типу зазвичай застосовуються до числових або табличних даних. Регресії діляться на лінійні, поліноміальні та логістичні. Лінійні та поліноміальні регресії націлені на знаходження оптимальної функції для n аргументів для отримання кінцевого числового значення $R \in N$. Логістична регресія ж використовується для отримання з певного набору даних результату $R \in [0, 1]$, що відповідає бінарній класифікації, абсолютні значення 0 та 1 відповідають першому та другому класу відповідно, а число R відповідно ступеню впевненості нейронної мережі щодо приналежності вхідного набору даних до першого чи другого класу.

Кластеризація, дані підходи використовуються для визначення подібності певних елементів набору даних. Для цього за певним алгоритмом (PCA, T-SNE, UMAP, і т.д) дані діляться на K кластерів, що відповідають за групи подібних об’єктів. Дана методика дозволяє класифікувати об’єкти без необхідності їх розмітки.

Дерева рішень, мережі даних архітектур є наборами операцій та умовних операторів утворених під час тренування, через які проходить вибірка даних та у результаті отримується результат $R \in N$ (у деяких випадках R також може бути набором символів). R може бути результатом подібним лінійної або логістичної регресії. Найбільш популярними архітектурами дерев рішень є Decision Tree (дерево рішень), Random Forest (Випадковий ліс), XGBoost, CatBoost.

1.2 Проблеми та виклики

Проблеми, що можуть виникнути під час розробки застосунку, можуть виникнути ще на етапі обробки вхідних даних. Необхідно забезпечити уніфіковане читання вхідних текстових документів, найбільш популярними форматами є PDF (Portable Document Format), docx (Формат документів програми Microsoft Word), а також файли зображень у форматах .png, .jpeg, .tiff та інші. Дані файли необхідно трансформувати у матрицю кольорових значень пікселів $h \times w \times c$, де h – висота

зображення, w – ширина зображення, а c – к-сть кольорових каналів зображення, дане значення буде 1, при чорно-білому зображенні, та 3 на стандартному зображенні RGB. Наступним викликом є розробка та створення моделі нейронної мережі. Дана мережа має бути достатньо швидка для обробки великої кількості багатосторінкових документів, точність даної мережі має бути достатньо високою для доцільності використання даного застосунку, в ідеальному випадку вона має прямувати до 100%, але зважаючи на реальні результати наявних конкурентів точність має бути в діапазоні 95-98%. Для цього модель має бути натренована на великій кількості варіативних даних.

До процесу тренування необхідно звернути особливу увагу, адже підбір початкових гіперпараметрів та їх зміна мають критичну важливість для отримання максимальної точності нейронної мережі. Структуризація документу має зважати на макет документу, відступи між символами та рядками, табуляції, пусті лінії і т.д. Також алгоритм трансформування матриці символів у форматований текстовий результат має бути достатньо оптимізованим для забезпечення швидкодії програми.

1.3 Аналіз засобів розпізнавання текстових документів

Наразі доступно широкий спектр програмних систем, які можна використовувати для аналізу, синтезу, редагування та обробки цифрових зображень із текстовими фрагментами. В цьому параграфі розглядається наявне програмне забезпечення, яке може розпізнавати та перетворювати документи з графічного формату в текстовий.

Для розпізнавання текстових символів на теперішній час сформувалася певна послідовність обробки сканованого текстового документу. Ця послідовність включає етапи виділення фрагменту тексту, попередню обробку цієї частини документу і потім приступають до самої важливої роботи по обробці документів — власне розпізнавання із використанням класифікаторів. У системах розпізнавання текстів використовуються такі класифікатори, як растровий, структурний і ознаковий [8]. Растровий класифікатор порівнює символ з набором еталонів, по

черзі накладаючи зображення одне на одне. Ознаковий класифікатор висуває гіпотези, виходячи із ступеня схожості параметрів символу з еталонними значеннями. Структурний класифікатор проводить структурний аналіз символу, розкладаючи останній на елементарні складові (точки, лінії, дуги) і формуючи точну схему аналізованого знаку. Потім отримана схема у вигляді структурного опису символів порівнюється з еталоном. Цей класифікатор працює повільніше растрового і ознакового, зате відрізняється високою точністю.

На теперішній час вказані функції класифікаторів все частіше покладають на нейронні мережі, які залежно від їх структури і налаштування поєднують у собі характеристики цих класифікаторів і дозволяють здійснювати процес розпізнавання символів текстових документів [9]. Тому для створення програмного засобу по розпізнаванню текстових документів було вирішено використати згорткову нейронну мережу. Було вибрано згорткову нейронну мережу, яка дозволяє формувати бажану конфігурацію із трьох типів шарів: згорткового шару, шару підвибірки та вихідного повнозв'язного шару нейронної мережі, а також з механізму отримання та трансформації регіонів на зображенні.

Сучасні системи OCR використовують три рівня розпізнавання: символи, слова та сторінки. Тим не менш, ABBYY додала у FineReader ще один рівень розпізнавання, який відповідає всьому багатосторінковому документу, відповідно до принципів IPA. У першу чергу всі ці елементи були необхідні, щоб правильно подати логічну структуру документа [8].

Для цього була розроблена технологія ADRT (Adaptive Document Recognition Technology), яка аналізує та синтезує документи на логічному рівні. Ця технологія допомагає максимально відтворити FineReader результат ідентичний оригіналу [14].

Звичайно, Abbyy FineReader є найкращою програмою для розпізнавання тексту, але також проаналізовано інші програми, щоб повністю підтвердити це. Таким чином, можливо оцінити стан сучасного ринку програм для розпізнавання тексту та визначити сильні та слабкі сторони кожної з цих програм.

ABBYY FineReader 12 — це професійне програмне забезпечення, яке може розпізнавати текст із графічних файлів і конвертувати його в різні редаговані формати. Програма визначає і переводить текст у новий формат. Це зменшує витрати на обробку, оскільки не потрібно перечитувати текст вручну. Компанії та організації, які часто працюють з паперовими документами, використовують цю програмну систему. Завдяки Abbyy FineReader можна конвертувати файли tiff, jpeg, pdf, doc, xlsx та txt.

Результати тестування показують, що ця система дуже добре обробляє та перетворює документи з графічного формату на текстовий. Це програмне забезпечення показало себе як продукт високої якості. Крім того, важливо пам’ятати, що основні поля, з яких у майбутньому будуть витягуватися дані з документа, добре відомі та безпомилкові, що означає, що основні завдання оптимізації та автоматизації можна вже виконати.

Переваги: висока якість розпізнавання тексту; широкий вибір форматів вхідних і вихідних документів; простий і зрозумілий інтерфейс користувача.

Недоліки: програмна система коштує грошей, а вихідний код програми недоступний.

У AbbyFineReader можна використовувати такі системи: SimpleOCR; OmniPage; Reading; and Writing. Free Online Service OCR — безкоштовний онлайн-сервіс для розпізнавання тексту в різних форматах. Gростий у використанні та не вимагає встановлення програми на комп’ютер. Цей продукт підтримує багато мов. Можна зберегти текст у будь-який із трьох форматів: звичайний текст, Microsoft Word і Microsoft Excel. У більшості випадків цих форматів достатньо для швидкого розповсюдження тексту.

Результати показують, що робота даного сервісу є більш ніж задовільною. Отже, він є чудовим інструментом для швидкого розпізнавання та передачі графічних файлів у текстовому форматі. Але через характеристики цього продукту його не можна використовувати для реальної оптимізації бізнес-процесів.

Переваги: багато мов для розпізнавання, швидкий доступ до пристроїв із доступом до Інтернету та простота використання.

Недоліки: цей продукт не містить повних функцій, на відміну від завантаженої програми оптичного розпізнавання символів.

CuneiForm — це програма, яка може розпізнавати текстові документи та перетворювати їх у редаговану форму. Цей продукт можна отримати безкоштовно для будь-якого користувача. Програми можна використовувати для отримання файлу у новому форматі зі зміненим текстом. Дана програма не підтримує деякі з найпоширеніших форматів файлів, таких як Microsoft Word (розширення .doc) і Acrobat Reader DC (розширення .pdf). Виходячи з вищезазначеного цей продукт буде менш зручним та ефективним у використанні, ніж аналог ABBYY FineReader 12.

Результати тестування показують, що OCR CuneiForm є низькоефективним програмним забезпеченням для оптичного розпізнавання тексту. Програма може не розпізнати деякі об’єкти, а розпізнані слова можуть виявитись набором незрозумілих символів. Середня кількість розпізнаних слів трохи менше дев’яносто відсотків.

Переваги: ця програма безкоштовна, а вихідний код програми доступний. Використовуючи його, ви можете запустити програму для аналізу графічних документів, яка використовує код, написаний на C#.

Недоліки: погана якість розпізнавання; аналіз виконується без налаштування активних полів для розпізнавання, тому програми розпізнають текст лише у тих місцях графічного файлу, які визначила програма.

Розпізнавання без коригування має на меті опустити етап ручної попередньої обробки, оскільки користувач не витрачає час на вибір поля.

Можна використовувати ABBYY Screenshot Reader, якщо сканування, експорт і пакетний режим не потрібні. За лічені секунди він захопить будь-яку частину екрана та перетворить її на звичайний текст. Хоча продукт коштує в рази дешевше, ніж його аналоги в сімействі, він має словник і перекладач.

Немає сумніву, що CuneiForm перевершує будь-яку іншу безкоштовну послугу розпізнавання тексту. Продукт трохи простіше, але виконує функції утиліт ABBYY.Readiris Pro і Freemore OCR можна вважати альтернативами FineReader і

CuneiForm для західного ринку. Зображуючи схожість, ви зрозумієте, наскільки програми принципово схожі.

У Readiris є велика перевага: ефективна система для читання рукописного тексту. Acrobat розроблено для інших цілей, але вбудована функція розпізнавання працює добре, навіть якщо вона менш практична, ніж інші рішення в нашому прикладі.

Scanitto Pro і RiDoc можуть швидко створити документ, зображення або аркуш паперу з символами. Утиліта не вимагає багато ресурсів і є ощадливою.

Компанія ABBYY, беззаперечний лідер у галузі оптичного розпізнавання символів, розробила комерційний продукт оптичного розпізнавання символів FineReader. FineReader базується на технології оптичного розпізнавання символів ABBYY OCR, яка має ліцензію від відомих світових компаній, таких як Fujitsu, Panasonic, Xerox і Samsung. Використовуючи цю програму, ви можете конвертувати зображення друкованих документів у безліч редагованих електронних форматів. Версія 12 розпізнає текст на 190 мовах, 48 з яких мають вбудовану перевірку орфографії, може зберігати вихідне форматування документа, має багато функцій попередньої обробки зображень і багато налаштувань параметрів. Тим не менш, його головною перевагою є майже безпомилкова точність визначення. Понад двадцять мільйонів людей у всьому світі зараз використовують дану програму [5].

CognitiveTechnologies розробила систему оптичного розпізнавання текстів під назвою CuneiForm. Наразі CuneiForm є шрифтонезалежною системою для перетворення електронних копій паперових документів і графічних файлів у редаговані. Він може автоматично або напівавтоматично зберігати структуру та гарнітуру шрифтів оригінального документа. Обробка електронних документів здійснюється за допомогою двох додатків у системі [6].

Починаючи з версії 0.4, OCRopus — система оптичного розпізнавання символів, яка використовує власне ядро розпізнавання для перетворення великих кількостей документів у електронний формат. Це пакет алгоритмів розпізнавання тексту з відкритим кодом, доступний за ліцензією Apache 2.0. Додаткові модулі

можна підключати в цю програму для аналізу макета вмісту та покращення попередніх зображень.

Головна ідея розробника полягає в тому, що OCRopus зможе точно визначати текст у графічному файлі та конвертувати його у звичайний текстовий формат для подальшого редагування. Програма може ідентифікувати не тільки друкований текст, але й рукописні матеріали.

OCRopus доступний лише для GNU/Linux. В даний час OCRopus використовує лише командний рядок для прийому вказівок на вхідні зображення з текстом і виводу даних у форматах hOCR або TXT [17].

FreeOCR — це безкоштовна програма, яка може розпізнавати відсканований текст. Вона може працювати з файлами зображень, pdf-файлами та безпосередньо зі сканером. Для сканування вам потрібен лише підключений сканер. Основним плюсом програми є те, що вона повністю автоматизована і не потребує будь-яких налаштувань. Підтримуються такі формати, як PDF, JPEG, GIF, TIFF і BMP. Крім того, існує обмеження на десять зображень на годину. Система здатна розпізнавати більшість східноєвропейських мов, українську в тому числі.

Окремо слід звернути увагу на Tesseract, систему розпізнавання текстів, розроблену Hewlett-Packard. З 2006 року він доступний безкоштовно і зараз використовується Google [3].

Система розпізнавання тексту з відкритим кодом (OCR) Tesseract OCR доступна за ліцензією Apache 2.0. Його можна використовувати безпосередньо для видалення друкованого тексту із зображень або через API для програмістів. Він розуміє багато мов. Хоча Tesseract не має вбудованих графічних інтерфейсів, деякі можна знайти на сторонніх сайтах розробників. За допомогою оболонки, доступної тут, Tesseract сумісний з багатьма мовами програмування та фреймворками. Його можна використовувати для розпізнавання тексту у великих документах за допомогою наявного аналізу макета або за допомогою зовнішніх детекторів тексту для розпізнавання тексту із зображень окремих рядків тексту.

Tressact 3x є процесом, який складається з багатьох етапів, і можна виділити такі етапи, як пошук слів, пошук по рядку та оцінки персонажів. Упорядковані

краплі рядків тексту використовуються для пошуку слів. Крім того, області та рядки скануються, щоб переконатися, що текст має певну висоту або пропорційність. Залежно від відстані між символами рядки тексту поділяються на різні слова. Після цього відбувається акредитація в два етапи. Під час першого проходу послідовно запам’ятовується кожне слово. Кожне слово, що затверджується, надсилається адаптивному класифікатору як навчальні дані. Після цього адаптивний класифікатор краще визначає наступний текст сторінки. Мета оновлення механізму Tesseract полягає в тому, щоб очистити код і додати нову модель LSTM. Для обробки вхідного зображення використовуються вирівняні прямокутники, які вводяться в модель LSTM. Потім виходять результати.Більшість існуючих систем OCR чудово обробляють чорний текст на світлому (білому) фоні, що є перевагою для розпізнавання текстових документів. Але якщо використовувати таку систему розпізнавання для обробки нестандартних зображень наприклад, для паспортів, посвідчень водія або номерних знаків автомобілів, можна зіткнутися з проблемою отримання точні дані без попередньої обробки зображення в текстовому документі. На етапі виконання попередньої обробки зображення багато проблем можна вирішити.

Узагальнюючи, можна сказати, що для методу оптичного розпізнавання тексту в отриманому зображенні потрібно виконати такі вимоги:

— пошук і визначення ділянок тексту на цифровому зображенні;

— виділення окремих рядків та символів на цих ділянках;

розпізнавання цих символів має відбуватися таким чином, щоб відстані між рядками на зображенні були нечутливими та стійкими до методу друку.

У цьому розділі бакалаврської роботи був здійснений огляд відомих методів й систем, що використовуються для оптичного розпізнавання друкованих символів текстових документів, здійснено порівняльний аналіз найбільш поширених засобів розпізнавання, перелічено їх основні недоліки й обґрунтовано необхідність створення нового програмного продукту для розпізнавання друкованих символів.

2 РОЗРОБКА ПОСЛІДОВНОСТІ ТА ВИБІР ЗАСОБІВ РОЗПІЗНАВАННЯ СИМВОЛІВ

У даному розділі бакалаврської роботи розроблена послідовність розпізнавання символів текстових документів та сформована згорткова нейронна мережа для їх розпізнавання.

2.1 Етапи розпізнавання друкованих символів

Засоби та технології штучного інтелекту зараз широко використовуються в різних галузях людської діяльності. Системи оптичного розпізнавання текстів (OCR) є однією з найвідоміших технологій штучного інтелекту. Автоматичне розпізнавання з використанням спеціальних програм цифрових зображень символів друкованого, машинописного або рукописного текстового документа називається системою OCR. У деяких випадках система оптичного розпізнавання символів розуміється як окремий пристрій для оптичного розпізнавання символів тексту [11].

Як класифікатори, системи розпізнавання текстових символів використовують методи, такі як шаблонні або растрові, ознакові та структурні [12]. Для розпізнавання символів шаблонний класифікатор визначає, який із шаблонів слід вибрати з бази. Найбільш простий критерій порівняння — це кількість пікселів, що відрізняються у шаблоні та отриманому зображенні символу. Перевагою шаблонного класифікатора є висока швидкість та простота реалізації. Суттєвим недоліком є необхідність забезпечення підтримки системою різних типів й розмірів шрифтів.

У структурному класифікаторі аналіз виконується лише для набору чисел або об’єктів, обчислених із зображення. Даний метод дозволяє розпізнавати різні дизайни символів, різні почерки, шрифти тощо. Цей спосіб призводить до втрати частини інформації через використання топологічного представлення, яке відображає інформацію про взаємне розташування структурних елементів символу. Отримані дані можна представити в графічній формі. Проте даний підхід

забезпечує незмінність типу та розміру шрифту. Серед недоліків виділяються труднощі з розпізнаванням символів з дефектами та низька швидкодія.

На даний момент сформована певна послідовність розпізнавання тексту. При розгляді методів та способів розпізнавання текстів можливо виділити загальну послідовність їх роботи, що складається із таких етапів.

Перший крок — пошук текстового поля, для його виділення на вихідному зображенні. Дана проблема може бути вирішена за допомогою дескриптора HOG [9], детектора Віоли Джонса, що базується на основі аналізу ознак текстури [10], ознаки Томура [11]. Необхідно зазначити, що сучасні системи оптичного розпізнавання були розроблені для роботи із зображеннями, які містять переважно структурований текст тому їх ефективність для завдання виявлення тексту в довільних зображеннях, у більшості випадків є незадовільною.

Усі системи оптичного розпізнавання символів покращують якість вхідного зображення за допомогою математичних алгоритмів. Серед них наявні спеціальні фільтри використовуються для відновлення пошкоджених зображень, такі як використання еліптичного розсіювання [19], фільтри видалення розмитості [20].

Наступний етап розпізнавання текстів включає в себе виявлення та виділення власне текстової складової у зображенні. Наразі використовуються наступні підходи: виділення країв та кутів [12]; виділення на основі використання інформації про контури (скелетизації) [13]; метод гістограм [16]; аналіз інваріантностей [15]; алгоритми адаптивної бінаризації, такі як Otsu, Niblack, Bernsan [18]; аналіз текстурної інформації; використання штучних нейронних мереж або експертних систем [19].

Щоб збільшити ймовірність точного розпізнавання кожного символу на цьому зображенні, виконується покращення якості вхідного зображення. Для цього виконуються такі дії: проведення операцій з видалення шуму (наприклад медіанний фільтр) і згладжування елементів зображення (для цього можна використати фільтр Гауса). Наступним етапом йде згладжування меж символів. Тут використовуються операції, що розмивають і додатково розширюють вихідне зображення. Отримане зображення перетворюється у карту пікселів у градаціях

сірого, а для тексту на зображенні виконується корекція бічного світла. Для цих завдань може бути використаним алгоритм SSR, як один із модифікованих алгоритмів логарифмічної корекції зображення [19]. Даний алгоритм балансує яскравість зображення, зберігаючи локальний контраст між яскравими та тьмяними ділянками зображення. Алгоритми покращення якості попередньо згладжують зображення, щоб визначити типи присутніх шумів і видалити їх.

Лінійний пошук зазвичай базується на періодичності та регулярності текстових полів і виконується на основі методу Хафа, методу зв'язаних компонент [15] та аналізу горизонтальних, вертикальних та діагональних розмірних графів. Сучасні алгоритми спрямовані на вирішення таких задач: Пошук розташування тексту, визначення структури тексту та кута нахилу рядків тексту. Задача визначення структури документа передбачає вибір однорідних блоків тексту, зображень, графіки, таблиць тощо. Алгоритм, який класифікує блоки на текстові та нетекстові, дозволяє визначити, чи є виділений блок текстом, зображенням, графікою, таблицею тощо.

Даний етап виконує такі задачі: Покращення якості зображення за допомогою фільтрів та інших операцій для покращення якості зображення. Після таких маніпуляцій помилка сканування зображення має бути усунена.

На початку роботи, до отриманого зображення часто застосовують фільтр Гаусса для видалення шумів. Важливу роль відіграють порогові бінарні операції, тобто перетворення зображення з кольорового у чорно-білий формат [17]. Це дозволяє розділити вхідне зображення на текст і фон, що спрощує використання багатьох алгоритмів, а також усуває деякі шуми на зображенні. Цей крок використовує гістограму інтенсивності отриманого текстового зображення. Дана операція передбачає виділення на гістограмі двох екстремумів. Один відповідає білому фону, що є кольором паперу. Темніші пікселі відповідають яскравості текстового символу.

Наступним кроком буде вибір тексту на зображенні як регіону інтересу. На цьому етапі роботи з бінарними зображеннями виділяються області, що містять розпізнаний текст, та видаляються нетекстові елементи на отриманому зображенні

[15, 16]. Ці елементи є кадрами зображень без тексту, точки на папері, які не видаляються під час бінаризації. Наприклад, щоб видалити їх, обирається опорна точка на зображенні, обчислюються геометричні елементи. Опорна точка сприймається як текстова частина або ж класифікується як текстова частина на основі даної точки. Можливе використання машинного навчання або евристичних методів. На цьому етапі текст розділяється або сегментується на компоненти, придатні для аналізу [17]. Найбільш логічною дією на даному етапі є розбиття тексту на окремі рядки (line split), потім розділення рядків на окремі слова (word split), а потім розділення отриманих слів на складові частини (символи).

Крім того, текст стандартизується, а вибрані компоненти зводяться до стандартного формату, щоб зменшити варіативність і спростити процес розпізнавання. Основні дефекти документів тепер вважаються повністю виправленими. Однак розгляд граничних задач створює низку складних проблем, які не дозволяють безпосередньо застосовувати відповідні алгоритми до друкованого тексту. Наприклад, рядки можуть бути не тільки непаралельними, але й надруковані під різними кутами, два окремих рядки, що йдуть один за одним можуть мати замалий або завеликий відступ одне від одного, а текстові елементи, що належать до різних ліній, можуть накладатися.

На цьому етапі виконується сегментація текстового документу до рівня окремих символів. Вона проводиться у три кроки.

Виділення рядка — вихідне зображення тексту обрізається на смуги необхідної ширини. Сегментація слова — у проекції рядка тексту позначається зображення слова.

Сегментація символів — проведення меж символів на проекції слова. Іноді для вибору базової лінії використовують певні алгоритми. Така операція використовується в тих випадках, коли при отриманні текстового документа виник дефект внаслідок деформації під час сканування. Ці методи засновані на ідеї, перетині уявної лінії. У таких методах певне вікно огляду намагається наблизитися до цієї лінії, а потім рядки текстового документу відновлюються відповідно до даного вікна. У цьому випадку перетворення Хафа використовується для вибору

прямих ліній, з невеликою степінню кривизни. Перетворення Хафа застосовано до центроїдів зв’язних компонентів текстових пікселів [19].

Можливість перетину лінійних елементів зі значними дефектами при розпізнаванні сканованих документів є проблемою не тільки для сегментації рядків, а й для розпізнавання тексту, оскільки зміна видимого контура зображення знижує його схожість з іншими подібними елементами, а отже точність розпізнавання значно падає. Перехресні компоненти ускладнюють використання методів пошуку горизонтальної проекції (оскільки вони перебільшують значення профілю проекції в місцях з передбаченим мінімальним значенням) і методів кластеризації, але мало впливають на деякі методи пошуку.

Щоб знайти елементи різних рядків, що перетинаються, використовуються такі функції, як розмір елементів, що з’єднують текст, та перевірка колінеарність з іншими лінійними елементами.

Знайшовши такі проблемні компоненти, необхідно визначити, чи належать вони до одного й того ж певного кластеру, чи ці елементи необхідно сприймати як ті, що належать до різних груп подібності. Вертикальний аналіз є складним завданням у таких випадках. Більш простим рішенням є використання горизонтальних ліній для поділу композиції на окремі частини. Однак також може застосуватися більш детальний підхід, наприклад, виділення окремих штрихів.

Виділені текстові елементи зв'язку, визначені на попередньому етапі системи розпізнавання, в подальшому об'єднуються в слова. Іншим опціональним кроком є нормалізація отриманого зображення. Висока варіативність символів ускладнює можливість їх розпізнавання [19]. Мета нормалізації полягає в трансформації текстових елементів до єдиної стандартної форми без значної втрати інформації, необхідної для наступного етапу розпізнавання. Одним із найдієвіших та популярних методів нормалізації є виправлення градієнта слова вздовж осей і приведення його до певних заданих розмірів. Найбільш примітивний спосіб закріплення слова по горизонталі — це екстримальне підвищення яскравості зображення в певному діапазоні. Існують інші методи нормалізації, такі як зміна

розміру та виділення текстових кадрів, але вони є менш відомими. Також існують методі на основі згладжування та лінійної регресії [20].

Для отримання опису ознак символів використовуються два підходи: перший полягає в аналізі оригінального зображення символу як ознаки, інший аналізує обчислені характеристики, такі як вертикальні хорди [14], приблизні контури та рамки розмітки [15], що класифікуються за певними мітками.

Щоб зменшити розмірність простору ознак, використовуються методи головних компонент і лінійний дискримінантний аналіз. Для розпізнавання символів реалізовано класифікатори на основі регресійних, кластеризаційних та згорткових нейронних мереж [13, 7, 19], які наразі є найбільш популярних підходом.

Словникова перевірка здійснюється на основі стандартних або динамічно створених мовних словників, n-грамів, реалізованих у вигляді списків, дерев або графіків [12, 16].

Для ефективного процесу розпізнавання текстового документу необхідно провести наступні операції:

— Пошук області зображення, що містить текст;

— Покращення якості та бінаризація локалізованої області зображення;

— Отримання структури виділеного блоку тексту на зображенні та визначення порядку читання;

— Сегментація тексту;

— Формування карти ознак кожного окремого символу зображення;

— Розпізнавання та класифікація окремих символів текстового зображення;

— Словникова перевірка текстового фрагменту.

Зображення, що надходить до системи розпізнавання, повинно бути очищено від потенційних помилок і піддано обробці, яка ефективно виділяє та розпізнає окремі символи в тексті.

Програма може розділити зображення на окремі фрагменти тексту, наприклад абзаци, використовуючи функції вирівнювання та розподілу зображення.

Для розпізнавання виділеного тексту зображення має бути розділене на рядки тексту, а потім на слова та символи. Після завершення цього кроку виділення символів, різні системи розпізнавання текстових документів використовують свої унікальні алгоритми.

Можлива обробка виділеного зображення масиву символів як цілий елемент шляхом порівняння з існуючими шаблонами, використання штучних нейронних мереж для розпізнавання або виділення певних характеристик кожного символу для розпізнавання.

Після завершення цього етапу роботи утворюються можливість отримання символу у формі літер, цифр або інших елементів. Засоби розпізнавання, як правило продовжують аналізувати текстовий документ, використовуючи різні методи та підходи, послідовно деталізуючи та уточнюючи результати роботи.

Завершальним кроком може бути використання словника для остаточного уточнення виділеного слова з множиною слів, які знаходяться у вибраному словнику.

2.2 Розробка послідовності розпізнавання символів

Деякі основні етапи процесу отримання зображення для розпізнавання тексту включають визначення області тексту на зображенні, попередню обробку цього фрагмента, сегментацію, визначення ознак, класифікацію та розпізнавання символів тексту та кінцеву обробку, яка називається пост-обробкою. Рисунок 2.1 показує загальну послідовність обробки зображення. Для виявлення та розпізнавання друкованих символів ми покладемо на ці етапи формування технологічної послідовності обробки зображення.

Засіб розпізнавання тексту використовує зображення текстового документа як вхідне зображення. Зображення повинні бути у форматі BMP, JPEG або PNG. Це

зображення можна отримати за допомогою пристрою формування та введення цифрових зображень, наприклад сканера або цифрової камери.

Наступний етап —попередня обробка. Вона складається з багатьох операцій, що виконуються над вхідним зображенням. Дані операції покращують якість зображення.

Бінаризація перетворює чорно-білі зображення на двійкові за допомогою глобального порогового значення. Відбілювання розширює бінаризоване зображення. Для отримання попередньо оброблених зображень, придатних для сегментації, останні два етапи включають масштабування зображення та заповнення отворів.

На етапі опису символьного аналізу дані повинні пройти кілька етапів попередньої обробки. Мета попередньої обробки полягає в тому, щоб отримати дані, необхідні для системи оптичного розпізнавання символів. Очищення шумів, підвищення контрастності та бінарізація зображення є основними завданнями попередньої обробки.

Зображення, утворені під час сканування документів, можуть мати додатковий шум. Низька якість цих зображень впливає на наступний етап обробки документів.

Таким чином, етап попередньої обробки необхідний, щоб підвищити якість зображення перед переходом до наступних етапів обробки документа. Щоб отримати більш точну інформацію, необхідно виключити шум, оскільки він може спричинити хибні виняткові сегменти, довгі рядки та інші помилки. Зображення містять різноманітні види шуму. «Шум солі та перцю», який складається з чорних і білих крапок, розкиданих по всьому зображенню, є одним з дефектів, який зустрічається в більшості документів. Методи зменшення шуму поділяються на дві основні категорії: морфологічна обробка та фільтрація.

Рисунок 2.1 — Схема розв'язання задачі розпізнавання

Фільтрація використовується для зменшення шуму та зайвих точок, які зазвичай виникають через нерівні поверхні запису пристрою збору даних або низьку частоту дискретизації. Як інструмент обробки зображень, морфологічні операції часто використовуються для отримання елементів зображення, корисних для представлення та опису форми області. Операції трансформації можна успішно використовувати для видалення шумів із зображень документів, які можуть бути спричинені сторонніми рухами рук або паперу низької якості. Бінаризація чорно-білих зображень символів є важливим кроком у розпізнаванні символів у позамережевому середовищі. Якість сегментація та розпізнавання символів покращується при використанні правильного алгоритму бінаризації .

Даний процес перетворює вхідне зображення на двійкове. Послідовність символів ділиться на окремі підсторінки під час сегментації зображення. У запропонованій системі попередньо оброблене вхідне зображення ділиться на різні

символи, а кожному символу присвоюється номер за допомогою методу маркування. Позначення показує кількість символів у зображенні. Для сегментації, реєстрації та виявлення об’єктів визначення ребер сегментів є корисним інструментом, оскільки вони характеризують межі об’єктів. Виявлення країв зображення зберігає важливі структурні характеристики зображення, одночасно зменшуючи обсяг даних.

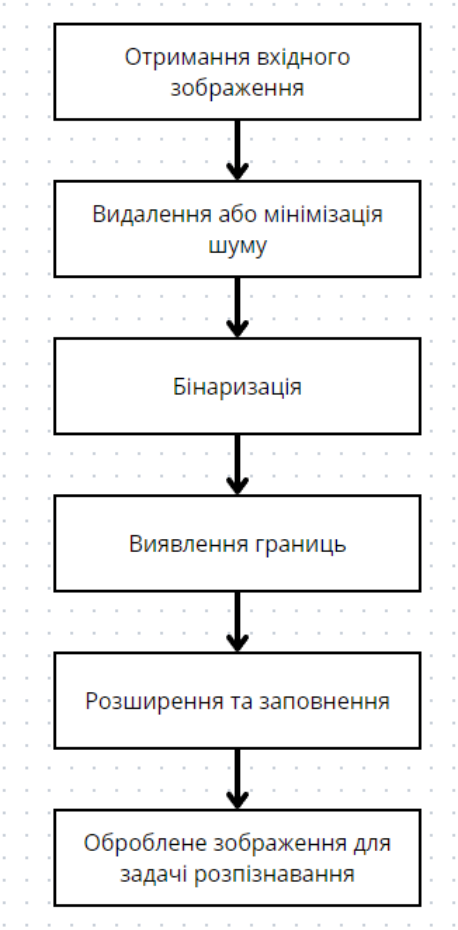

Рисунок 2.2 — Попередня обробки зображення

Краї можна виміряти різними способами. Тим не менш, більшість методів належать до групи Лапласа або градієнтних. Щоб визначити краї, метод градієнта використовує максимальне та мінімальне значення першого похідного зображення. Щоб знайти ребра, метод Лапласа використовує перетин нуля у другому похідному зображенні.

Виявлення спотворень — потенційна асиметрія, виявлена за допомогою сканування. Існує кілька поширених методів ідентифікації зміщення сторінок. Деякі частини залежать від ідентифікації взаємопов’язаних деталей і виявлення

середнього кута, який з’єднує їхні центри. Усунення асиметрії є необхідним, оскільки вона знижує точність розпізнавання документа. При обчисленні кута нахилу проводиться горизонтальна лінія. У методах розпізнавання символів сегментація є найважливішим процесом. Зображення розділяється на окремі символи.

Сегментація поділяється на зовнішню та внутрішню. Розподіл макету сторінки на логічні блоки дозволяє створити зовнішні сегменти. Після цього текст розділяється на різні частини, такі як абзаци, речення та слова. Це найважливіший етап аналізу документів.

Мета аналізу та розпізнавання документів полягає в тому, щоб автоматично отримувати інформацію, яка міститься на папері, та обробляти її відповідно до людського розуміння. Сегментація сторінки є важливим кроком у аналізі макета, проте процес може ускладнитись у випадках компонованих макетів. Аналіз макетів сторінок проводиться у два етапи.

На початку процесу структурного аналізу зображення розділяється на окремі блоки елементів документа, таких як рядки, слова та абзаци.

Другий етап включає функціональний аналіз. За допомогою різних правил розмітки він визначає макет, розмір і функціональний вміст елементів документа.

Згодом сторінка розділяється на частини, щоб знайти місця з білою та чорною текстурами. Мета внутрішньої сегментації — розділити рядок на окремі підфрейми символів.

Процес прийняття рішень у створеній послідовності розпізнавання включає етап класифікації, заснований на виділенні ознак символів. Нейронна мережа використовується для класифікації та розпізнавання друкованих символів. Вхід нейронної мережі отримує сегментоване текстове зображення з останнього кроку.

Запропоноване програмне забезпечення призначено для пошуку та розпізнавання кириличних текстових документів, тому загальна кількість нейронів у вихідному шарі мережі становить 33. Етап післяобробки (пост обробки) є передостаннім етапом запропонованої послідовності розпізнавання. Індекси розпізнаних символів тестового шаблону використовуються для представлення

виявлених і розпізнаних символів у структурованому тексті шляхом визначення відповідних еквівалентних значень у кодовій таблиці ASCII.

Завершальним етапом роботи системи розпізнавання символів є використання словника для уточнення виділеного слова шляхом його порівняння із наявною опорною множиною слів.

2.3 Використання згорткової нейронної мережі для розпізнавання

Для розпізнавання символів виділеного текстового зображення пропонується використати згорткову нейронну мережу.

Згорткові нейронні мережі (CNN) — це клас глибоких штучних нейронних мереж прямого поширення, які застосовуються для аналізу зображень. Ці мережі створюються за аналогією з біологічними процесами, вони є варіантами багатошарових персептронів і вимагають мінімальної попередньої обробки. ЗНМ — це тип багаторівневої нейронної мережі, яка називається «згорткова мережа» від назви операції згортки. ЗНМ складається з вхідного та вихідного шарів, а також n-прихованих шарів.

Зазвичай вхідний шар перетворює 2D-зображення в 3D-матрицю. Таким чином, тривимірна матриця подається на вхід наступного шару, проте розмір тензору може змінюватись на виході шару.

На відміну від загальної моделі багаторівневої нейронної мережі (MNN), приховані шари ЗНМ є спеціалізованими і включають згорткові шари, повністю зв’язані шари, шари зменшення дискретизації та шари вибірки.

Загальна структура ЗНМ формується із багатьох шарів. Зображення подається на вхід мережі та утворюється тензор, що проходить через послідовну серію згорткових шарів, де він поступово згортається й субдискретизується. Карта ознак може бути створена, чергуючи шари. З кожним наступним рівнем розмір цієї карти зменшується одночасно збільшуючи кількість каналів. Після проходження певного кількості шарів карта ознак перетворюється на скалярну або векторну. ЗНМ створює сотні таких карт об’єктів. Після проходження цих згорткових шарів вводяться наступні шари перцептрона (повністю зв’язаної нейронної мережі), яка

використовує вхідні дані, щоб створити остаточну карту ознак. Типова архітектура згорткової нейронної мережі приведен у Додатку И бакалаврської роботи.

Основними компонентами такого типу штучних нейронних мереж є згорткові шари, що застосовують до вхідного зображення операцію згортання (конволюції) й передають результат для опрацювання у наступний шар мережі. Згортання імітує реакцію окремого взятого нейрону на зоровий стимул. Кожний згортковий шар може бути охарактеризований так:

— F — розмір рецептивного поля, відповідає за кількість нейронів у попередньому шарі, що потім будуть зв’язані з одним нейроном у цьому шарі мережі;

— K — вказує на глибину, тобто кількість нейронів у поточному шарі;

— S — крок згортання, тобто кількість окремих значень, що ігноруються рецептивним полем;

— P — розмір нульового доповнення країв матриці, необхідний для отримання стандартизованих вхідних значень матриць.

За потреби, вхід на краях зображення доповнюється нулями. У мережі кожен нейрон згорткового шару є фільтром (ядром). Даний фільтр є матрицею, що має розмір $F_1 \times F_2$, параметри якого змінюються під час навчання мережі. Із використанням цього фільтру формується двовимірна карта активації, що є другою функцією для операції конволюції.

У ЗНМ використовуються спільні ваги у згорткових шарах, тобто для кожного рецептивного поля шару мережі використовується один й той же фільтр у вигляді ваг, це зменшує обсяг необхідної пам'яті для обробки й поліпшує ресурсоефективність мережі .

Над кожною наступною матрицею ознак, що подається на вхід шару, здійснюється операція згортки шляхом обчислення зваженої суми фрагментів аналізованого зображення навколо заданого ядра згортки:

$$(f*g)[m,n] = \sum_{k,l}(f[m-k, n-1] * g[k.l]) \qquad (2.1)$$

,де f —матриця ознак, або вхідне зображення;

g — ядро згортки зображення;

k — номер рядка ядра згортки;

l — номер стовпця ядра згортки.

Архітектура ЗНМ грунтується на послідовному чергуванні шарів згортки (convolution layers) й шарів субдискретизації (subsampling layers, pooling layers). Структура такої нейронної мережі є однонаправленою, зворотні зв'язки відсутні та ефективність забезпечується використанням великої кількості шарів. При навчанні такої ЗНМ застосовується зворотне поширення помилок у шарах мережі.

Робота ЗНМ базується на генералізації аналізованих зображень до більш абстрактних деталей. Як правило, ЗНМ використовуються для формування бажаної ієрархії ознак, опускаючи незмістовні деталі та підкреслюючи важливі.

Функціонування ЗНМ забезпечується двома елементами: фільтром, який виявляє приховані характеристики зображення, та карти ознак, яка описує аналізоване зображення.

Базова модель ЗНМ складається з трьох типів шарів: згортковий шар, шар підвибірки та повноз’єднаний шар нейронної мережі у формі персептрона. Персептрон — це штучний нейрон, що є повноцінною нейронною мережею, де кожен наступний нейрон даного шару пов’язаний з усіма нейронами попереднього рівня, кожен зв’язок має власні вагові коефіцієнти. Невелика вагова матриця, яка переміщується по всьому вхідному зображенню, представленому цими нейронами, виконує функцію згортки в такій нейронній мережі. Після кожного зсуву матриця створює сигнал активації, який активується для наступного нейрона з подібною позицією. Це ядро згортки використовує цю вагову матрицю для різних нейронів у вхідному шарі. Його представлення містить графічне кодування певного елемента зображення. Операція згортання цієї вагової матриці створює наступний шар, який формує карту характеристик, вказуючи на наявність певного об’єкта в обробленому шарі та вказуючи його координати.

ЗНМ має великий набір ваг, що кодують елементи аналізованого зображення (наприклад лінії під різними кутами, або ж дуги). Ядра згортки є унікальними для даної мережі, так як вони формуються методом зворотного поширення помилок.

Набір ознак формується під час ЗНМ за допомогою набору ваг, що робить цю нейронну мережу багатоканальною. Зазвичай під час редагування шару за допомогою вагової матриці шар переміщується на невелику відстань. Наприклад, вагова матриця розміром 5x5 буде зміщена на один або два нейрони. Це необхідно щоб запобігти втраті інформації. Використовуючи операцію згортки, рівень згортки передає результат до наступного рівня. Згорткова мережа базується на наборі вагових коефіцієнтів. У результаті цієї процедури створюється зображення форми на карті об’єктів. Карти функцій виділяють певні характеристики обробленого вхідного зображення залежно від матриці, вибраної для операції згортання. Для оптимізації характеристик вхідного зображення використовується кілька різних ядер згортки, а на виході згорткового рівня мережі формується кілька карт з різними характеристиками. Насправді ядро згортки є вікном або фільтром, який переміщується по попередній карті, щоб вибрати певні характеристики об’єкта.

Під час тренування кожен фільтр масштабується до певного розміру, а також обчислюється скалярний добуток фільтра та вхідних даних, що призводить до створення двовимірної карти збудження фільтра. Це дозволяє мережі знати, які фільтри активуються, коли певний тип функції виявляється в певному просторовому місці входу. Максимізація продуктивності згорткових шарів досягається шляхом збору карт активації кожного фільтра канального виміру.

Таким чином, кожен запис у вихідному відсіку також можна розглядати як вихід нейрона. Такий нейрон реагує на невелику вхідну область і має параметри, пов’язані з іншими нейронами, представленими в одній карті стимулів. Використовуючи операцію згортки, шар передає результат до наступного рівня. У кожному зв’язку згорткової мережі знаходиться набір певних вагових коефіцієнтів. Ця процедура створює карту об’єктів.

Залежно від матриці, вибраної для операції згортання, карти функцій виділяють певні характеристики обробленого вхідного зображення. Для оптимізації характеристик вхідного зображення використовується кілька ядер згортки, а на виході згорткового рівня мережі створюється кілька карт із різними характеристиками. Насправді ядро згортки є вікном або фільтром, що переміщується по попередній карті, щоб створити вибірку певних властивостей об'єкта. Згортка нагадує відповідь одного нейрона на візуальний стимул. Кожен згортковий нейрон обробляє інформацію лише в своєму рецептивному полі. Параметричний рівень складається з набору навчальних фільтрів (ядер), які охоплюють всю глибину вхідного діапазону, незважаючи на невелике сприйнятливе поле. Під час передачі кожен фільтр масштабується відповідно до ширини та висоти вхідного контейнера, а потім відображається шляхом обчислення скалярного добутку фільтра та вхідних даних.

Таким чином, мережа знає, які фільтри активуються, коли певний тип функції виявляється в певному просторовому розташуванні на вході. Агрегування карт збудження кожного фільтру за глибиною забезпечує повну потужність згорткового шару. Таким чином, вхід до вихідного відсіку також можна розглядати як вихід нейрона, який реагує на невелику вхідну область і має параметри, пов'язані з нейронами, які знаходяться в одній карті стимулів.

В мережі рівень підвибірки є операцією нелінійного стиснення (нелінійного перетворення) отриманої карти ознак. З іншого боку, певна група пікселів, яка зазвичай становить 2 на 2 пікселя, стискається в одне значення вихідної карти.

Для отримання цього перетворення зазвичай використовується функція пошуку максимуму. Перетворення використовує прямокутники або квадрати, які не перекриваються. Потім кожен прямокутник замінюється пікселем, який має найбільше значення в сегменті аналізованого зображення.

Субдискретизація дозволяє значно зменшити просторовий розмір аналізованого зображення. Цей процес легко зрозуміти таким чином: якщо певні характеристики зображення були виявлені під час попереднього кроку згортки, вони більше не потрібні для наступного кроку обробки, а обробка відбувається на

нижчому рівні. Крім того, виконання процесу фільтрації непотрібних деталей зображення допомагає мережі уникнути повторного навчання.

Шар зменшення дискретизації (пулінг) змінює розмір карти функцій. Це не тільки полегшує обчислення, але й може запобігти проблемі перетренування. Вважається, що наявність значення важливіше, ніж точне знання координат. Нелінійне перетворення стискає групу пікселів, які зазвичай розміром 2×2, в одну комірку вихідного значення. Таким чином, до кожного фрагмента вхідної матриці застосовується фільтр 2x2, а найбільше число обирається як головне з вікна. З цього випливає напіввимірна матриця [15]. Вплив цього перетворення поширюється на прямокутники чи квадрати, які не перетинаються. Це об’єднання зменшує просторовий об’єм зображення.

Незважаючи на те, що використовується стиснення з функцією максимуму, також використовуються інші функції, такі як середнє значення та нормалізація L2. Однак, як показує практика, найкраще використовувати стиснення з функцією максимуму.

Згортковий шар і шар зниження дискретизації чергуються. Мережа трансформується з відомої сітки пікселів високої роздільної здатності в більш абстрактні карти функцій після кількох проходів згортання та стискання зображення за допомогою методу зменшення дискретизації. Кожен наступний шар створює більшу кількість каналів і менше розширення зображення.

Таким чином створюється певний масив вихідних каналів, що зберігають невелику кількість даних або один параметр. У початково проаналізованому зображенні отримуються унікальні числа, які вказують на абстрактні ідеї.

Після кількох повторних чергувань згорткового та підвибіркового шарів використовується один або кілька повнозв’язних шарів нейронів, які є повністю зв’язаними (fully-connected). Як і в звичайних нейронних мережах прямого поширення, нейрони повнозв’язного шару зв’язані з кожним нейроном попереднього шару. Звичайна повнозв’язана нейронна мережа, яка може складатися з кількох шарів, об’єднує та передає ці дані. У цьому випадку

повнозв’язні шари втрачають просторову структуру пікселів і мають меншу розмірність порівняно з кількістю пікселів у початковому зображенні.

Кожен нейрон одного шару з’єднується з нейроном наступного рівня через повністю зв’язаний рівень, який надає мітку високого рівня. Такий тип шару належить до традиційної багатошарової нейронної мережі (БНМ). Використовується повноз’єднаний шар, який визначається на основі ознак, які були отримані. На цьому рівні працює багатошарова нейронна мережа персептрона. Кожен нейрон одного шару зв’язаний з кожним нейроном наступного шару за допомогою повністю з’єднаних шарів. Основною метою рівня злиття ЗНМ є локальне або глобальне злиття та об’єднання виходу кластерів нейронів одного шару з нейронами наступного рівня.

Для нелінійної підвибірки точне розташування об’єкта не є настільки суттєвим, як його приблизне розташування відносно інших об’єктів. Щоб зменшити кількість параметрів і обчислювальних зусиль у мережі та контролювати перенавчання, рівень об’єднання має ефект поступового зменшення просторового розміру представлення. Архітектура ЗНМ зазвичай включає в себе періодичний шар об’єднання між послідовними згортковими шарами. Глобальні операції забезпечують інваріантність іншого типу, ніж паралельна передача. Рівень об’єднання зменшує просторовий розмір і працює незалежно на кожному вхідному зрізі глибини. Максимальне об’єднання використовує максимальну кількість кожного кластера нейронів із попереднього рівня. Він ділить вхідне зображення на набір прямокутників, які не перекриваються, щоб показати максимальне значення для кожної з цих підобластей. Усічений лінійний одиничний рівень (рівень випрямленої лінійної одиниці, ReLU) застосовує функцію передачі ненасиченого ReLU.

Це покращує нелінійні властивості всієї мережі та функції прийняття рішення, не впливаючи на сприйнятливі поля згорткового рівня. Інші функції, такі як сигмоїдна функція та насичений гіперболічний тангенс, використовуються для покращення нелінійності. Зрівняльний шар зменшує розмір входу до каналу. Рівень втрат зазвичай є останнім шаром і показує, як потяг обробляє різницю між

фактичними та прогнозованими мітками класу. Функції втрат можна використовувати для різних цілей. Для прогнозування одного класу серед K взаємовиключних класів застосовуються нормовані експоненціальні втрати, також відомі як softmax. K незалежних значень ймовірності в інтервалі [0, 1] передбачається за допомогою сигмоїдної втрати крос-ентропії. Регресія на мітках із дійсними значеннями в діапазоні (−∞,+∞) використовує евклідові втрати.

ЗНМ має просторову організацію (тривимірну ємність нейронів), локальні зв'язки між нейронами в суміжних шарах для забезпечення просторової локальності та виявлення ознак незалежно від загальних параметрів (ваг) їх положення в полі зору та рівня зв'язку. Це основна відмінність між ЗНМ і БНМ. За допомогою просторового розташування та глобальних параметрів ознаки можна ідентифікувати незалежно від їхнього положення в полі зору та в межах пов'язаного шару. Поєднання цих функцій дозволяє ЗНМ виконувати кращі візуальні завдання. Зважений розподіл значно зменшує кількість вільних параметрів, які мережа вивчає, а також кількість пам'яті, необхідної для роботи мережі. Це також дозволяє навчати більші та потужніші мережі. Традиційне зворотне розповсюдження помилок дозволяє згортковій нейронній мережі навчатися фільтрувати ваги. Метод зворотного поширення помилок (backpropagation) використовується для навчання ЗНМ.

Структура програмного засобу складається із ряду модулів, що послідовно із отриманого зображення текстового документа виділяють сторінки, потім у них виділяють текстовий фрагмент і виконують розпізнавання символів. Процес розпізнавання символів покладається на нейронну мережу. Попередньо нейронну мережу слід налаштувати на розпізнавання символів тексту [4]. Також у складі програмного засобу є ще модуль для навчання згорткової нейронної мережі на виконання процедури розпізнавання друкованих символів і модуль виведення результатів розпізнавання символів.

Розпізнавання текстових символів в даному підході відбувається за допомогою згорткової нейронної мережі, що згортає вихідне зображення розмірності $m_0 \times n_0 \times 3$ до необхідної $m \times n \times c$, де c — кількість каналів, за

допомогою двох основних операцій — конволюції (2-dimensional Convolution) та пулінга (2-dimensional Max Pooling). Додаткові канали утворюються за допомогою накладання фільтрів двовимірною конволюцією на вхідну матрицю. Розмірність зображення зменшується через операцію пулінга, що обирає найбільше значення в певному заданому регіоні, зазвичай розміром 2×2, 3×3 або 4×4 та записує у нову матрицю. Пулінг вирішує дві основні проблеми: зменшує кількість ознак, відмовляючись від тих, що несуть мало інформації, і тим самим запобігає збільшенню кількості математичних операцій та збільшує щільність робочої матриці ознак, відмовляючись від більшості нульових значень та запобігаючи проблемі зникаючого градієнту.

Кожний вихідний канал несе інформацію про певний регіон, що дозволяє перейти від матриці зображення, яка описує кольорові значення, до матриці ознак. Дані ознаки є прихованими і несуть суттєву інформацію для математичної моделі нейронної мережі.

При розпізнаванні вхідного зображення воно розбивається на регіони різного розміру $h_x \times w_x \times c$. Дані регіони проходять певний ланцюжок трансформацій для отримання матриці ознак однакової розмірності $h \times w \times c$. Усі трансформовані регіони об'єднуються у матрицю $n \times h \times w \times c$, де n — кількість отриманих регіонів. Отримана матриця обробляється повноз'єднаним шаром (Dense Layer), вихідне значення якого — матриця $n \times d$, де d — кількість нейронів. Матриця $n \times d$ формує два вихідних значення — матрицю $n \times q$ з нормовано експоненційною активацією (softmax), де q — кількість можливих класів, та $n \times 4$, що є матрицею координат чотирикутників, що описують об'єкт.

У даному розділі описано процедури обробки текстового документа та вимоги до згорткової нейронної мережі, для розпізнавання символів в текстовому документі. У даному розділі бакалаврської роботи сформована послідовність розпізнавання друкованих символів текстових документів. Запропоновано розділити алгоритм розпізнавання на ряд етапів, де послідовно виконується попередня обробка текстового документу, використання нейронної мережі для розпізнавання, та поділ вихідного масиву символів на рядки, слова та абзаци.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РОЗПІЗНАВАННЯ ДРУКОВАНИХ СИМВОЛІВ ТА ЇХ СТРУКТУРИЗАЦІЇ

Процес виділення і розпізнавання друкованих символів сканованих текстових документів виконується за послідовний ряд етапів, реалізація яких у вигляді програми розглянуті у даному розділі бакалаврської роботи.

3.1 Розробка архітектури модуля розпізнавання символів та структуризації

На основі запропонованої послідовності обробки зображення із текстом у структурі програмного засобу мають бути модулі, що послідовно із отриманого зображення текстового документа виділяють сторінки, потім у них виділяють текстовий фрагмент і виконують сегментацію виділеного фрагменту із текстом для розпізнавання символів. У нашому випадку процес розпізнавання символів покладається на нейронну мережу. Таку нейронну мережу необхідно попередньо налаштувати на розпізнавання символів тексту.

Модуль з розпізнавання символів складається з двох основних частин:

— обробка вхідних даних, читання текстових документів, отримання зображень, накладання фільтрів на зображення та перетворення зображень у відповідну матрицю пікселів;

— робота з моделлю, ініціалізація нейронної мережі, тренування, або отримання передбачень, обробка вихідних результатів.

Відповідно до вхідного формату застосунок обирає метод обробки файлу. Якщо дані у форматі .pdf необхідно перевірити чи вони нативні або згенеровані. Нативні .pdf файли — це ті які були створені у спеціальному редакторі, що працює з файлами такого типу, отже усі об’єкти у документі залишаються інтерактивними. Згенеровані ж у свою чергу є сканованими документами, та об’єктами можуть виступати зображення, вміст яких неможливо отримати звичайними методами. У випадку нативних файлів, необхідно провести маніпуляції для отримання записаного тексту (рис. 3.1) та повернути результат. В іншому ж випадку необхідно

перетворити сторінки документу в матрицю пікселів та передати її у нейронну мережу.

При отриманні даних у форматі зображення, необхідно їх зчитати та отримати матрицю пікселів яку згодом можна передати у нейронну мережу.

Лістинг 3.1 – отримання тексту з нативних .pdf документів

```
def get_pdf_for_one_page(page):
    images_area = 0
    for page_data in page.get_images():
        images_area += page_data[2] * page_data[3]
    images_area /= page.rect[2] * page.rect[3]
    if images_area >= 1:
        return None
    words = page.get_text("words")
    return words
```

Також окрім обробки вхідних даних необхідно натренувати модель нейронної мережі. Для цього створюється великий набір даних, що складається з усіх основних алфавітних, номерних та спеціальних символів. Для правильного використання моделі архітектури Faster R-CNN спочатку тренується базова нейронна мережа, що слугує класифікатором. Згодом тренується основна архітектура з натренованою базовою моделлю. Вихідний результат формується з двох складових частин: обмежувальної рамки об’єкту, тобто чотирикутника який обмежує об’єкт так, що жоден піксель не виходить за координати рамки; класу об’єкту, тобто його назви символу присутньому на зображенні.

Наступним етапом після отримання результатів нейронної мережі є структуризація вихідних символів. Використовуючи статистичні, математичні та логічні методи, визначаються поля документу, слова, пусті стрічки, переноси.

3.2 Попередня обробка текстових документів

Забезпечення широкого спектру застосування програми відбувається за допомогою алгоритму перетворення текстових документів у зображення з подальшим викликом нейронної мережі. Для вирішення даної проблеми використаний наступний підхід.

Отриманий вхідний документ необхідно перевести у зображення, так як обрана архітектура нейронної мережі може обробляти лише матриці пікселів. Для перетворення сторінок документу у зображення використовується бібліотека PyMuPDF. Дана бібліотека роботи з файлами розширення .pdf є одною з найбільш продуктивних на ринку, та пропонує широкий вибір інструментів для маніпуляцій над документами даного формату. Для отримання карт пікселів зі сторінок, які можна перетворити у матрицю пікселів (Рисунок 3.2) використовується метод класу сторінки Page.get_pixmap().

Лістинг 3.2 – алгоритм перетворення сторінки документу у матрицю пікселів

```
def pixmap_to_np_array(pix):
    im = np.frombuffer(pix.samples, dtype=np.uint8).reshape(pix.h,
pix.w, pix.n)
    return np.ascontiguousarray(np.copy(im[..., ::-1]))

def get_img_from_page(pdfpath: str):
    with open(pdfpath, 'rb') as f:
        doc = fitz.Document(stream=f.read(), filetype="pdf")
        first_page = doc.load_page(0)
        pixmap = first_page.get_pixmap()
        np_image = pixmap_to_np_array(pixmap)
        np_image_resized = cv2.resize(np_image, (640, 640),
interpolation=cv2.INTER_AREA)
        return np_image_resized
```

Розмір отриманих зображень не є оптимальним для обробки ЗНМ, адже відношення сторін не є 1:1. Зміна розмірів матриці відбувається за допомогою бібліотеки OpenCV (Рисунок 3.3), що дозволяє обрати бажані виміри та метод інтерполяції пікселів, для найкращої якості вихідного зображення. Як еталонні виміри було обрано розмір 640x640, що дозволяє забезпечити баланс між обсягом інформації та обчислювальних потужностей витрачених на обробку такого зображення.

Лістинг 3.3 – алгоритм зміни розміру зображення

```
np_image = pixmap_to_np_array(pixmap)
np_image_resized = cv2.resize(np_image, (640, 640),
interpolation=cv2.INTER_AREA)
return np_image_resized
```

Проте після зміни розміру певна частина інформації викривляється або втрачається. Для покращення деталізації зображення використовується метод increase_image_quality() (Рисунок 3.4), що дозволяє повернути деталізацію об’єктів. Це особливо важливий крок, адже ЗНМ тренується на окремих символах. Наявність неякісних символів на вхідному зображенні може суттєво вплинути на якість розпізнавання.

Лістинг 3.4 – функція покращення якості зображення

```
def basic_image_quality_increase(inputImage):
    grayscaleImage = cv2.cvtColor(inputImage, cv2.COLOR_BGR2GRAY)
    thresh, im_bw = cv2.threshold(grayscaleImage, 210, 230,
cv2.THRESH_BINARY)
    kernel = np.ones((1, 1), np.uint8)
    return cv2.dilate(im_bw, kernel=kernel, iterations=1)
```

3.3 Розробка архітектури нейронної мережі

Тренування Faster R-CNN відбувається у два етапи, першою тренується базова модель класифікатора, у даному випадку VGG-16 (Додаток Б). Структура даної нейронної мережі складається з послідовності згорткових та субдискретизаційних шарів результат яких подається як вхідне значення повноз’єднаного шару. У випадку використання даної моделі як базової для Faster R-CNN, повноз’єднані шари не використовуються, а вихідне значення неповної VGG-16 є матрицею ознак зображення, що згодом використовується для розпізнавання об’єктів мережею пропозиції регіонів (Region Proposal Network RPN).

МПР отримує зображення, матрицю ознак та матрицю якірних точок (anchor points). Матриця ознак проходить через згортковий шар з ініціалізатором з випадковим Гауссівським розподілом, та регуляризатором L^2 тобто Евклідовою нормою (3.1).

$$\left|\left|x\right|\right|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} \tag{3.1}$$

Матриця отримана з цього шару слугує вхідними даними для двох інших згорткових шарів. Перший шар відповідає за класифікацію об’єкту, та має з кількістю фільтрів що дорівнює кількості якірних точок. Другий шар відповідає за

обрамлення об’єкту та має $N * 4$, де N — кількість якірних точок. Результат даних шарів згодом застосовується у методі extract_valid() (Лістинг 3.5) , що повертає якірні точки, які не перетинають межі зображення, класові значення даних якірних точок та дельти регіонів, з яких згодом будуть побудовані реальні регіони інтересу зображення.

Розмірність anchor_valid_map вказує на розмір підвибірки, висоту, ширину та кількімть якоріних точок. Всього якірних точок [N,4], і [N,1] якорів, що не перетинають межі зображення

Лістинг 3.5 – метод _extract_valid()

```
def _extract_valid(self, anchor_map, anchor_valid_map,
objectness_score_map, box_delta_map, allow_edge_proposals):
  height = tf.shape(anchor_valid_map)[1]
  width = tf.shape(anchor_valid_map)[2]
  num_anchors = tf.shape(anchor_valid_map)[3]

  anchors = tf.reshape(anchor_map, shape = (height * width *
num_anchors, 4))                anchors_valid =
tf.reshape(anchor_valid_map, shape = (height * width * num_anchors,
1))  scores = tf.reshape(objectness_score_map, shape = (height *
width * num_anchors, 1))
  box_deltas = tf.reshape(box_delta_map, shape = (height * width *
num_anchors, 4

  anchors_valid = tf.squeeze(anchors_valid)
  scores = tf.squeeze(scores)
```

Надалі фільтруються ті пропозиції, які згенеровані на недійсних якорях. Недійсні якорі – це ті, що перетинають межі зображення і, ці якорі ігноруються під час розрахунку втрат, але їх слід включати при генерації пропозицій. Хороша продуктивність вимагає оцінки великої кількості пропозицій, тому, навіть якщо якорі, що перетинають межі, не сприяють втратам RPN, вони все одно можуть подавати зразки на етап детектора. Тому не рекомендується виключати пропозиції на краях.

Лістинг 3.6 – фільтрація недійсних якорів

```
  if allow_edge_proposals:
    # Use all proposals
    return anchors, scores, box_deltas
  else:
```

Продовження лістингу 3.6

```
        idxs = tf.where(anchors_valid > 0)
    return tf.gather_nd(anchors, indices = idxs),
tf.gather_nd(scores, indices = idxs), tf.gather_nd(box_deltas,
indices = idxs)
```

Наступним кроком за допомогою методу tf_convert_deltas_to_boxes() (Лістинг 3.7) будуються реальні рамки регіонів інтересу з отриманими з попередньої операції параметри.

Лістинг 3.7 – метод tf_convert_deltas_to_boxes()

```
def convert_deltas_to_boxes(box_deltas, anchors, box_delta_means,
box_delta_stds):
  """
  Параметри
  ----------
  box_deltas : np.ndarray
    Дельти рамки з розмірністю (N, 4). Кожен зразок - (ty, tx, th,
tw).
  anchors : np.ndarray
    Відповідні якорі на яких базуються дельти, розмірністю (N, 4).
Кожен зразок -

(center_y, center_x, height, width).
  box_delta_means : np.ndarray
    Коригування дельт відповідно до середнього значення, (4,), що
додається після стандартного відхилення,
    зміни розміру і перед трансформацією в реальні координати рамки
.
  box_delta_stds : np.ndarray
  Стандартне відхилення дельт, (4,). Дельти рамок множаться на дані
числа

  Повертає
  -------
  tf.Tensor
    Координати рамок, (N, 4), кожен зразок якої - (y1, x1, y2, x2).
  """
  box_deltas = box_deltas * box_delta_stds + box_delta_means
  center = anchors[:,2:4] * box_deltas[:,0:2] + anchors[:,0:2]   #
center_x = anchor_width * tx + anchor_center_x, center_y =
anchor_height * ty + anchor_center_y
  size = anchors[:,2:4] * np.exp(box_deltas[:,2:4])              #
width = anchor_width * exp(tw), height = anchor_height * exp(th)
  boxes = np.empty(box_deltas.shape)
  boxes[:,0:2] = center - 0.5 * size                             #
y1, x1
  boxes[:,2:4] = center + 0.5 * size                             #
y2, x2
  return boxes
```

Дані рамки є пропозиціями, у кожної пропозиції є бал «об'єктності», тобто ступінь ймовірності знаходження об'єкту в даному регіоні. Згодом отримані пропозиції сортуються за балом об'єктності та обираються N-найкращих пропозицій, де N — гіперпараметр, що визначається до тренування, та вони обрізаються до розмірів зображення. Далі, рамки регіонів сторони яких менше за F пікселів, де F — гіперпараметр, видаляються. Наступним етам є прігнічення немаксимальних значень (Non-Maximum Supression), при якій з декількох рамок що вказують на один і той же об'єкт обирається та, що перетинає об'єкт найбільше (Лістинг 3.8). Після цього відфільтровані пропозиції повертаються і робота МПР закінчується.

Лістинг 3.8 – функція non_max_supression

```
def non_max_suppression_with_scores(boxes,
                                    scores,
                                    max_output_size,
                                    iou_threshold=0.5,
                                    score_threshold=float('-inf'),
                                    soft_nms_sigma=0.0,
                                    name=None):

  with ops.name_scope(name, 'non_max_suppression_with_scores'):
    iou_threshold = ops.convert_to_tensor(iou_threshold,
name='iou_threshold')
    score_threshold = ops.convert_to_tensor(
        score_threshold, name='score_threshold')
    soft_nms_sigma = ops.convert_to_tensor(
        soft_nms_sigma, name='soft_nms_sigma')
    (selected_indices, selected_scores,
     _) = gen_image_ops.non_max_suppression_v5(
         boxes,
         scores,
         max_output_size,
         iou_threshold,
         score_threshold,
         soft_nms_sigma,
         pad_to_max_output_size=False)
    return selected_indices, selected_scores
```

Наступний компонент Faster R-CNN — модель розпізнавання (Detector Network), дана мережа будується з двох основних шарів. ROIPoolingLayer – шар що відповідає за субдискретизацію пропозицій, даний шар обробляє рамки різного розміру та проектує їх на один розмір. У даній роботі існує два варіанта цього

етапу. Перший — експериментальна імплементація шару описаного в статті [29] за допомогою інструментів фреймворку TensorFlow, проте даний підхід передбачує високі обчислювальні навантаження та був розроблений в навчальних цілях. Другий підхід передбачує спрощений варіант застосування субдискретизації регіонів інтересу, перевагами є простота обчислень та відносно мала втрата в точності (Лістинг 3.9).

Лістинг 3.9 – спрощений алгоритм субдискретизації регіонів інтересу

```
image_height = tf.shape(input_image)[1] # висота
image_width = tf.shape(input_image)[2]  # ширина
rois = proposals / [ image_height, image_width, image_height,
image_width ]

# обрізка, зміна розміру, субдискретизація
num_rois = tf.shape(rois)[0];
region = tf.image.crop_and_resize(image = feature_map, boxes = rois,
box_indices = tf.zeros(num_rois, dtype = tf.int32), crop_size = [14,
14])
pool = tf.nn.max_pool(region, ksize = [1, 2, 2, 1], strides = [1, 2,
2, 1], padding = "SAME")
pool = tf.expand_dims(pool, axis = 0) # (num_rois, 7, 7, 512) -> (1,
num_rois, 7, 7, 512)
```

Наступним ключовим шаром є TimeDistributed, що будується на основі вхідного шару, повноз’єднаного, згорткового та ін. Зазвичай даний шар використовується для виконання операцій над часовими рядами, де другий вимір відповідає за відліки спостереження над об’єктом на різних часових відрізках. Він необхідний у даній архітектурі для того виконання операцій вхідним шаром над кожною пропозицією. Структура моделі розпізнавання приведена у Додатку Ж бакалаврської роботи.

Після отримання результатів, кожен зразок проходить крізь функції втрат, функцією втрат для класифікатора є середнє значення категоріальних перехресних ентропій усіх зразків. В якості функції втрат для регресора виступає L^1 (3.2), або Мангеттенська метрика.

$$L^1 = |x_1 - x_2| + |y_1 - y_2| \tag{3.2}$$

Відповідно до значень метрики нейронна мережа оновлює ваги усіх шарів та продовжує тренування.

3.4 Тренування нейронної мережі

Для ефективного тренування нейронної мережі був створений набір даних з алфавітів англійської та української мови, арабських цифр, спеціальних символів. Варіативність датасету забезпечується використанням великої кількості шрифтів.

За допомогою бібліотеки Pillow утворюється пусте зображення заданого розміру, у даному випадку 100x100 пікселів, із заданим символом. Обмежувальна рамка отримується за допомогою властивостей шрифта, а клас зображення є самим символом. Написана функція, що отримує заданий символ, друкує його вказаними шрифтами та утворює масив зображень (Лістинг 3.10).

Лістинг 3.10 — Функція char_to_image

```
def char_to_image(char, font_file, img_size):
    font_size = int(img_size * 0.9)
    ttfont = TTFont(font_file, fontNumber=0)
    try:
        supported_character = ttfont.getBestCmap().get(ord(char))
        if supported_character:
            img = Image.new('L', (img_size, img_size),
background_color)
            drawer = ImageDraw.Draw(img)
            font = ImageFont.truetype(font_file, font_size)

            # Отримуємо обмежувальну рамку символу
            f_x0, f_y0, f_x, f_y = font.getbbox(char)
            # Отримання довжини та ширини символу
            font_width = f_x - f_x0
            font_height = f_y - f_y0
            # Отримання центральної точки з якої буде друкуватись
символ
            # Таким чином що весь символ буде знаходитись всередині
зображення
            x = (img_size / 2) - (font_width / 2)
            y = (img_size / 2) - (font_height / 2)
            drawer.text((x, y), char, fill_color, font=font)
            return img
        else:
            return
    except AttributeError:
        return
```

Наступним кроком є створення власного класу датасету для оптимальної подачі даних та забезпечення найбільш ефективного використання ресурсів. Для цього використовується клас фреймворку Tensorflow –

tensorflow.keras.utils.Sequence, який буде наслідуваним даним класом. Щоб використати даний клас при тренуванні у ньому необхідно визначити чотири основних методи:

— __getitem__() — основний метод отримання елементів датасету, викликається кожного разу при зверненні до об’єкту датасету;

— __len__() — метод для отримання довжини датасету, у даному випадку це кількість зображення поділена на розмір підвибірки;

— __data_generation — утворення елементів датасету на льоту, це дозволяє не зберігати усі елементи в пам’яті, а створювати їх динамічно по мірі необхідності;

— on_epoch_end() — функція, що викликається після завершення епохи тренування, необхідна для того, щоб перемішати, або змінити дані під час тренування.

Лістинг 3.11 — клас CharDataGenerator

```
class CharDataGenerator(tf.keras.utils.Sequence):
  def __init__(self, alphabets, fonts, batch_size):
    self.fonts = fonts
    self.unified_alphabet = list(itertools.chain(*alphabets))
    self.batch_size = batch_size
  def __len__(self):
    return (len(self.fonts) * len(self.unified_alphabet)) /
self.batch_size
  def __getitem__(self, index):
    if type(index) is not int:
      batches = []
      for i in index:
        char = self.unified_alphabet[i]
        X = self.__data_generation(char)
        batches.append(X)
      batches = np.array(batches)
      return batches
    else:
      char = self.unified_alphabet[index]
      X, y_l, y_b = self.__data_generation(char)
      return X, y_l, y_b

  def __data_generation(self, char):
    X = []
    y_label, y_bbox = [], []
    while len(X) < self.batch_size:
      curr_items_in_batch = len(X)
      font_subset = random.sample(self.fonts, self.batch_size)
      images, labels, bboxes = get_char_images(char, font_subset,
```

Продовження лістингу 3.11

```
100)
      X.extend(images[:self.batch_size - curr_items_in_batch])
      y_label.extend(labels[:self.batch_size - curr_items_in_batch])
      y_bbox.extend(bboxes[:self.batch_size - curr_items_in_batch])
    X = np.expand_dims(np.array(X), axis=-1)
    y_label = np.array(y_label)
    y_bbox = np.array(y_bbox)
    return X, y_label, y_bbox

  def on_epoch_end(self):
    if self.shuffle:
      np.random.shuffle(self.unified_alphabet)
```

Шрифти завантажуються з ресурсу Google Fonts під час запуску програми тренування.

Тренування нейронної мережі забезпечується завдяки модулю train (Додаток В), який має функцію train, що приймає як аргумент зкомпільований об’єкт моделі. На початку функції у консоль виводяться обрані гіперпараметри. Ініціюється два об’єкти датасетів, перший відповідає за тренувальний набір даних, другий — за валідаційний, що є меншим за розміром та містить зразки, жодний з яких не є присутнім у тренувальному наборі, він необхідний для перевірки точності моделі на даних, що відрізняються від тренувальних, а отже визначають реальну приблизну точність в умовах використання вже натренованої моделі.

Використання валідації може також допомогти при відслідковуванні проблеми перенавчання, тобто високої точності на відомих даних, та низької при використанні на даних невідомих моделі. Якщо утворюється перенавчання це унеможливлює ефективне використання нейронної мережі та веде до нестабільних результатів.

Для правильної обробки нейронною мережею отриманих зразків датасету, створено функцію-посередника _convert_training_sample_to_model_input (лістинг 3.11), що має аргументи sample — зразок даних та mode — режим у якому використовується модель, тренувальному або передбачувальному (робочому).

Лістинг 3.12 – функція _convert_training_sample_to_model_input

```
def _convert_training_sample_to_model_input(sample, mode):
    gt_box_corners = np.array([ box.corners for box in
sample.gt_boxes ]).astype(np.float32)
    gt_box_class_idxs = np.array([ box.class_index for box in
sample.gt_boxes ]).astype(np.int32)
    image_data = np.expand_dims(sample.image_data, axis = 0)
    anchor_map = np.expand_dims(sample.anchor_map, axis = 0)
    anchor_valid_map = np.expand_dims(sample.anchor_valid_map, axis
= 0)
    gt_rpn_map = np.expand_dims(sample.gt_rpn_map, axis = 0)
    gt_rpn_object_indices = [ sample.gt_rpn_object_indices ]
    gt_rpn_background_indices = [ sample.gt_rpn_background_indices ]
    gt_box_corners = np.expand_dims(gt_box_corners, axis = 0)
    gt_box_class_idxs = np.expand_dims(gt_box_class_idxs, axis = 0)

    gt_rpn_minibatch_map = _sample_rpn_minibatch(
      rpn_map = gt_rpn_map,
      object_indices = gt_rpn_object_indices,
      background_indices = gt_rpn_background_indices,
      rpn_minibatch_size = 256
    )

    if mode == "train":
      x = [ image_data, anchor_map, anchor_valid_map,
gt_rpn_minibatch_map, gt_box_class_idxs, gt_box_corners ]
    else:
      x = [ image_data, anchor_map, anchor_valid_map ]

    return x, image_data, gt_rpn_minibatch_map
```

Наступним кроком запускається процес тренування за допомогою вбудованого у клас tensorflow.Model методу train_on_batch. На відміну від вбудованого методу tensorflow.Model.fit(), train_on_batch дозволяє провести зворотнє поширення помилок та оновити ваги без стандартних зворотніх викликів (callbacks), що описані в fit(). Даний підхід дозволяє описати власний процес тренування та послідовність викликів додаткових функцій збереження, візуалізації та інші. Після створення функції тренування необхідно отримати об’єкт зкомпільованої моделі та передати його як параметр. Для цього спочатку створюється об’єкт моделі класу FasterRCNNModel (Додаток Г). На цьому етапі створюються три об’єкти нейронних мереж:

— НМ отримання ознак зображення;

— мережа пропозицій регіонів;

— мережа розпізнавання.

У методі call() класу FasterRCNNModel описана логіка послідовного виклику трьох даних об’єктів для отримання вихідного результату.

Після створення об’єкту моделі використовується вбудований метод класу tensorflow.keras.Model.build(). Даний метод використовується для зручності ініціалізації форми вхідних даних. Це дозволяє моделі визначити форму тензорів які вона буде обробляти на різних етапах виклику в будь-якому місці у коді. Особливо зручною ця функція є при умові можливості зміни вхідних форм, наприклад зміна розширення вхідного зображення. Завдяки цьому методу будь-які зміни входу нейронної мережі можливо визначити в одному місці в коді.

Наступним етапом йде компіляція моделі, основним параметром якої є оптимізатор. Існує два варіанта оптимізаторів у цій роботі звичайний стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD) та Adam. Adam є комбінацією SGD та поширення середньоквадратичного (Root Mean Square Propagation, RMSP), головною перевагою якого є використання експоненційного рухомого середнього, що дозволяє відфільтрувати ознаки що мінімізуються експоненційно, та в більшості випадків дозволяє оминути локальні мінімуми та продовжити пошук глобального мінімуму функції.

Повний алгоритм тренування описаний на рисунку 3.2. Після завершення тренування модель нейронної мережі може бути використана за допомогою того ж об’єкту моделі та методу predict().

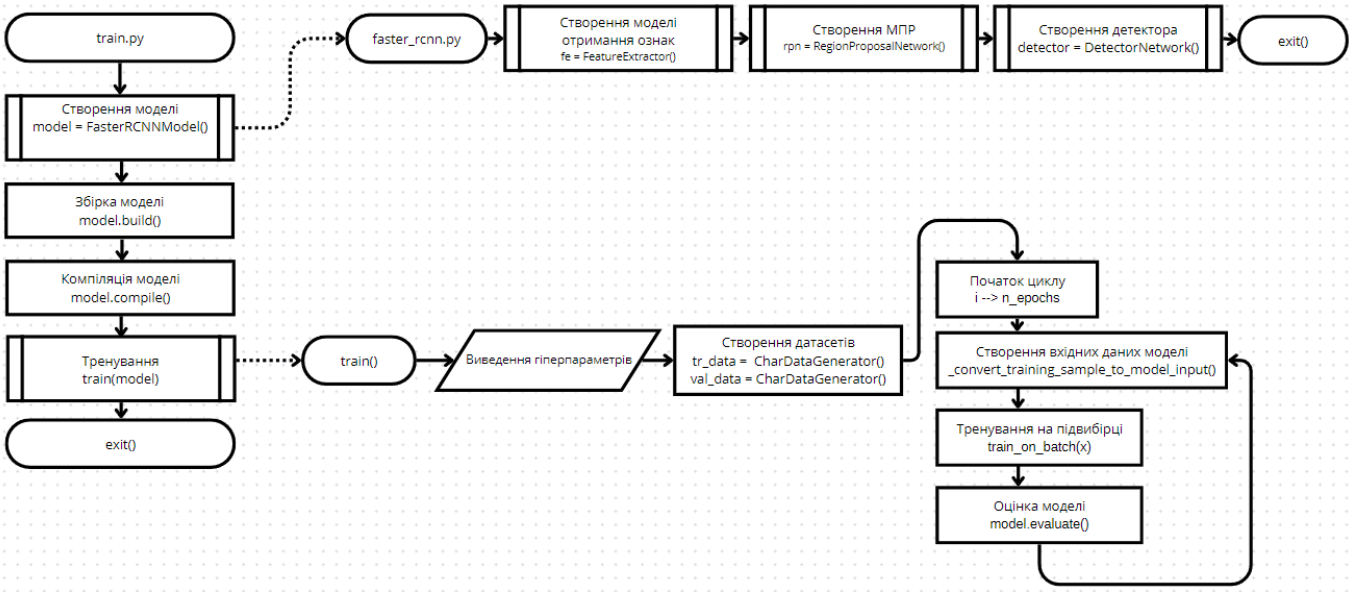

Рисунок 3.2 – Опис алгоритму тренування

3.5 Створення модуля структуризації документу

Модуль структуризації документу виконує дві основних функції — обробку невірно розпізнаних символів та розрахунок відступів між розпізнаними символами.

Першим етапом є розрахунок відступів, завдяки ним можливо виділити поля документу, слова, абзаци, пробіли, символи нової стрічки. Обробка символів відбувається відповідно значенням окремих слів та усього тексту, та має на увазі відфільтрувати та конвертувати символи різних локалізацій. Наприклад англійська літера «A» та українська літера «А» є ідентичними, проте в Unicode вони відрізняються своїм кодом, а отже розпізнавання даного символа є проблемним та потребує подальшої пост-обробки.

Розрахунок відступів відбувається після отримання від нейронної мережі обмежувальних рамок усіх розпізнаних символів, згодом розраховуються поля, тобто місця на документі, що не мають екстремального градієнту. Саме такий підхід розрахунку полів забезпечує ефективну роботу зі сканованими документами з дефектами (підтьоки чорнил, загини і т.д). Після розрахунку полів виділяється регіон документу, що містить інформацію та є опорною точкою для розрахування абзаців.

Наступним етапом є компонування слів з розпізнаних символів, для цього використовується функція get_spaces (Лістинг 3.13), що приймає як аргумент масив обмежувальних рамок символів. Спочатку утворюються рядки документу, для цього перевіряється чи початкова x-координата обмежувальної рамки наступного символа більша за кінцеву попередньої, якщо ні — це означає що наступний символ знаходиться в новому рядку. Після отримання рядків відбувається ітерація та перевіряються відстані між символами, якщо відстань менша за опорну (середню ширину символа з похибкою у -10%) тоді даний символ є частиною одного слова.

Лістинг 3.12 – функція get_spaces

```
def get_spaces(bboxes: np.array, labels: list):
    x_coords = bboxes[:, [0, 3]]
    widths = [x - x0 for x0, x in x_coords]
    x_ref = sum(widths) / len(x_coords)
    lines = []
    line_labels = []
    for x_ind in range(len(x_coords)):
        cur_x = x_coords[x_ind]
        cur_label = labels[x_ind]
        next_x = None
        try:
            next_x = x_coords[x_ind + 1]
            next_label = labels[x_ind + 1]
        except IndexError:
            lines.append([])
        if next_x is None:
            continue
        elif cur_x[0] > next_x[1]:
            lines.append([next_x])
            line_labels.append([next_label])
        else:
            lines[-1].append(cur_x)
            line_labels.append(cur_label)
    text = []
    textboxes = []
```

Розрахунок абзаців відбувається наступним чином, отриманий масив рамок тексту є рядками документу, якщо початкова x-координата першого елементу рядка більша за початкову першого елементу наступного рядка, це означає що даний рядок є абзацом.

3.6 Тестування програми розпізнавання

Для запропонованого підходу оптичного розпізнавання символів на текстових документах було виконано тестування програми. Тестування на працездатність та можливість виявлення друкованих символів створеним програмним забезпеченням було здійснено за наявності різнопланових документів з різним рівнем пошкоджень, шуму та повороту символів.

Для виконання операції перевірки розробленої програми було обрано комп'ютерну систему з високими показниками, що складається із комп'ютера.

Обчислювальна машина характеризується такими параметрами: процесор типу AMD Ryzen 5 3600x з тактовою частотою 3,80 ГГц, оперативна пам'ять комп'ютера 32GB DIMM DDR4, жорсткий диск 1 TB, відеокарта Nvidia GTX 1660 із об'ємом відеопам'яті 6GB. Для відображення зображень використовувався пристрій Acer Nitro VG3 23.8. Перелічені технічні параметри складу комп'ютера є достатніми для виконання тестування розробленого комплексу програм й дозволять сформувати результати для здійснення аналізу коректності роботи створеного комплексу програм.

Обрані програмні засоби для виконання тестування застосунку включають мову програмування Python, модулі TensorFlow, Keras, Pillow та NumPy. Виконана перевірка працездатності розробленого програмного засобу бакалаврської роботи для розпізнавання друкованих символів та структуризації тексту у документах різного розширення.

Для перевірки якості розпізнавання, тестування відбувається в три етапи. У першому етапі перевіряється точність розпізнавання окремих символів (рисунок 3.3) та за допомогою модулю ImageDraw бібліотеки Pillow малюються обмежувальні рамки розпізнаних символів, та за допомогою вбудованого методу зображення формату Pillow.Image.show() дане зображення відкривається у стандартному оглядачі зображень Windows.

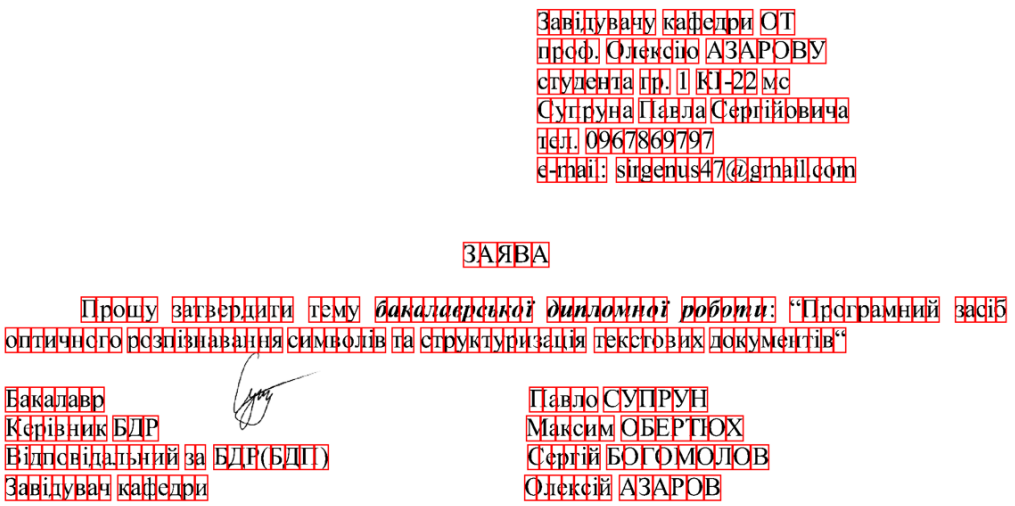

Рисунок 3.3 – тестування точності розпізнавання символів

Наступним був протестований модуль структуризації, що за допомогою математичних операцій виділяє слова (рисунок 3.4) та абзаци (рисунок 3.5) документу. Відображення вихідних зображень відбувається за алгоритмом попереднього етапу тестування.

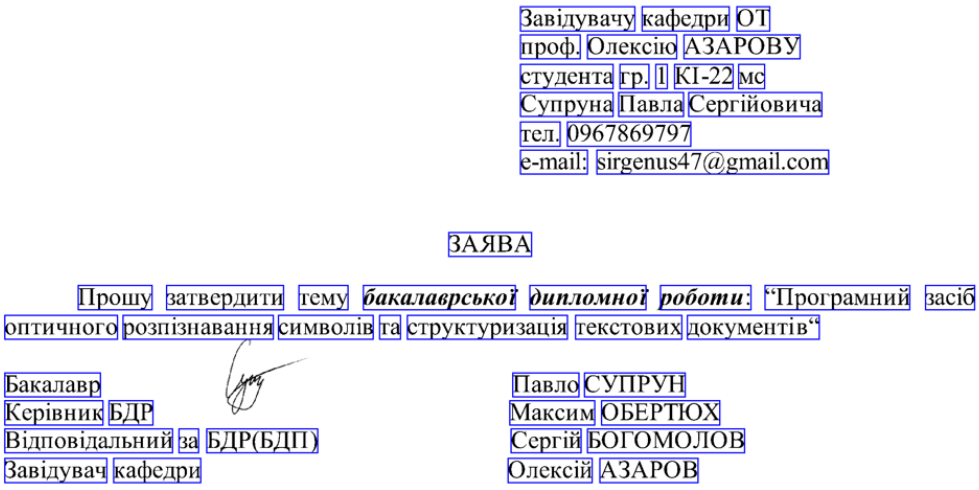

Рисунок 3.3 – Тестування точності розпізнавання слів

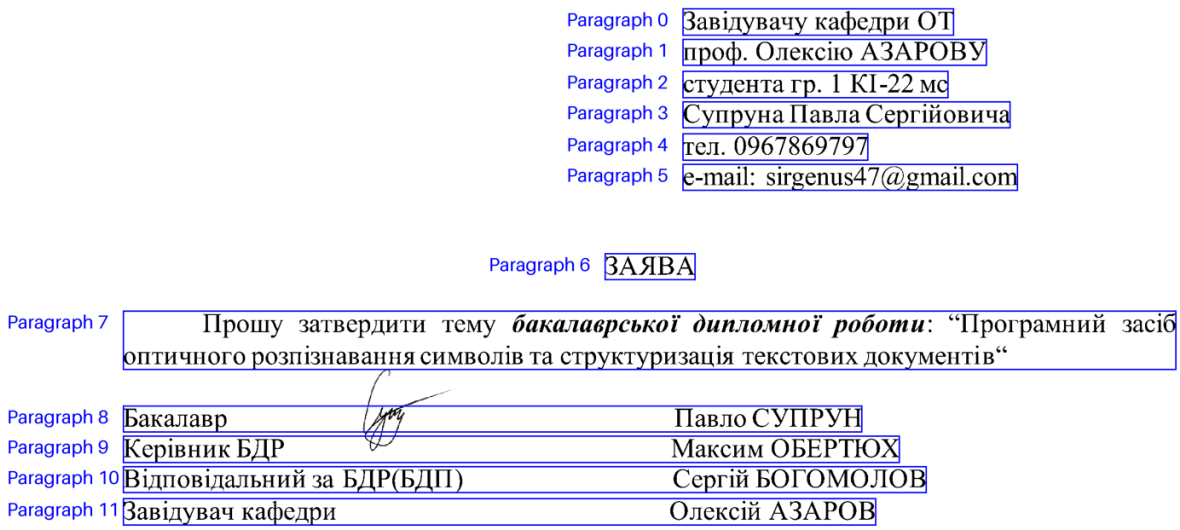

Рисунок 3.4 — Тестування точності розпізнавання абзаців

Результати проведеного тестування є відповідними заданим цілям та мають потенціал до покращення. Отримане програмне забезпечення може бути покращене та використане як основа для повноцінного застосунку для обробки документів.

Цей розділ роботи був присвячений розробці та опису структури програми для розпізнавання друкованих символів текстових документів, розроблена програма розпізнавання символів та виконано процес навчання нейронної мережі на сформованому тестовому наборі символів. Також був створений модуль структуризації вихідних символів у повноцінний текстовий документ.

ВИСНОВКИ

У сьогоденні цифровий документообіг стає все більш нагальною потребою. Переведення текстових документів на паперових носіях у електронний формат стало одним із головних рушіїв до прогресу у різних сферах людського життя. Проте при розпізнаванні документів можуть виникнути помилки, адже сканування пошкодженних паперових носіїв може значно спотворити вихідне зображення документу. Вирішення деяких аспектів цієї проблеми із використанням нейронної мережі розглядається у даній бакалаврській роботі.

У першому розділі бакалаврської роботи був здійснений огляд відомих методів й систем, що використовуються для оптичного розпізнавання друкованих символів текстових документів, здійснено порівняльний аналіз найбільш поширених засобів розпізнавання, перелічено їх основні недоліки й обґрунтовано необхідність створення нового програмного продукту для розпізнавання друкованих символів. Аналіз розглянутих методів й засобів розпізнавання текстових документів дозволив виявити деякі проблеми, що існують у цій області, виявити переваги й недоліки наявних систем вирішення задач з виявлення та розпізнавання символів, у томі числі із деякими спотвореннями.

У другому розділі бакалаврської роботи сформована послідовність розпізнавання друкованих символів текстових документів. Запропоновано розділити алгоритм розпізнавання на ряд етапів, де послідовно виконується попередня обробка текстового документу, використання нейронної мережі для розпізнавання, та поділ вихідного масиву символів на рядки, слова, та абзаци.

Третій розділ роботи був присвячений розробці та опису структури програмного продукту для розпізнавання друкованих символів текстових документів, розроблена програма розпізнавання символів та виконано процес навчання нейронної мережі на сформованому тестовому наборі символів. Також був створений модуль структуризації вихідних символів у повноцінний текст документу.

Розроблений у бакалаврській роботі програмний продукт може бути використаний у повноцінних системах розпізнавання документів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Chaudhuri A. Optical character recognition systems for different languages with soft computing. / A. Chaudhuri, K. Mandaviya, P. Badelia, S. K. Ghosh // Springler, 2017. — 260 p. https://doi.org/10.1007/978-3-319-50252-6_1.

2. Оптичне розпізнавання символів (OCR) [Електронний ресурс] — Режим доступу: https://hashdork.com/uk/optical-character-recognition/.

3. FineReader [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/ABBYY_FineReader

4. Субботін С. О. Нейронні мережі : теорія та практика: навч. посіб. / С. О. Субботін. — Житомир : Вид. О. О. Євенок, 2020. — 184 с.

5. Тимченко О. В. Нейромережеві методи розпізнавання зображень текстів /О. В. Тимченко, Б. М. Гавриш, Б. В. Дурняк // Поліграфія і видавнича справа. 2021, № 1 (81) — С.72 — 88.

6. Жихаревич В. В. Аналіз методів розпізнавання символів тексту / В. В. Жихаревич, С. Е. Остапов, І. В. Миронів // Радіоелектронні і комп'ютерні системи. —2016. — № 5. — С. 137—142.

7. Супрун П. С. Розпізнавання символів текстових документів із використанням нейронної мережі П. С. Супрун, М. А. Очкуров // Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)». — Електрон. текст. дані. — Вінниця, 2024. — Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21297.

8. Згорткова нейронна мережа. [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/Згорткова_нейронна_мережа.

9. Shi B. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition / B. Shi, X. Bai, C. Yao // IEEE transactions on pattern analysis and machine intelligence. —2017, vol. 39, №. 11, pp. 2298 — 2304.

10. Busta M. Deep text spotter: an end-to-end trainable scene text localization and recognition framework / M. Busta, L. Neumann, J. Matas // Proceedings of the IEEE International Conference on Computer Vision. 2017, pp. 2204-2212.

11. Ye Q. Text detection and recognition in imagery: A survey / Q. Ye, D. Doermann // IEEE transactions on pattern analysis and machine intelligence. 2015, vol. 37, № 7, pp. 1480-1500.

12. Uchida S. Statistical Deformation Model for Handwritten Character Recognition // Advances in Digital Document Processing and Retrieval. 2014, pp. 157-174.

13. Тимченко О. В. Алгоритми та функції інформаційної системи розпізнавання символів на основі методів поліпшення зображень / О. В. Тимченко, І. О. Кульчицька, О. О. Тимченко // Моделювання та інформаційні технології. Зб. наук. пр. ІПМЕ НАН України. – Вип.69. – К.: 2013. – С.167-173.

14. Zahra Elhamraoui. Introduction to convolutional neural network. *https://medium*.com/analytics-vidhya/introduction-to-convolutional-neural-network-6942c189a723, May 28, 2020.

15. Huang Gao. Densely Connected Convolutional Networks / Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger // NASA ADS. — 2016. — Pp. 21–29.

16. Що таке ABBYY FineReader [Електронний ресурс] – Режим доступу: https://help.abbyy.com/uk-ua/finereader/12/overview.

17. OCR CuneiForm для Windows [Електронний ресурс] – Режим доступу: http://softobase.com/ru/ocr-cuneiform.

18. Програма Readiris Pro [Електронний ресурс] – Режим доступу: *https://readiris*.ru.uptodown.com/windows

19. LeCun Y. LeNet-5, convolutional neural networks. [Електронний ресурс] – Режим доступу: URL: *http://yann*.lecun.com/exdb/lenet/.

20. Comparison of optical character recognition software. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/ Comparison_of_optical_character_ recognition_software.

21. Smith, R.W. History of the Tesseract OCR engine: what worked and what didn’t / R.W. Smith // Proceedings of SPIE. – 2013. – Vol. 8658. – pp. 186-195.

22. InftyReader. Офіційий сайт. [Електронний ресурс] – Режим доступу: URL: *http://www*.inftyreader.org/.

23. Keras. Офіційна документація. [Електронний ресурс] – Режим доступу: URL: *https://keras*.io/.

24. OpenCV library [Електронний ресурс] – Режим доступу: https://opencv.org/.

25. Aforge.NET: Framework [Електронний ресурс]. – Режим доступу: http://www.aforgenet.com/framework/.

26. The EMNIST dataset. [Електронний ресурс]. – Режим доступу: https://www.nist.gov/itl/iad/image-group/emnist-dataset.

27. Shaoqing R. et al. Faster r-cnn: Towards real-time object detection with region proposal networks // Advances in neural information processing systems. 2016, pp. 3-6.

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

професор, д.т.н. Азаров О. Д.

"06" травня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської дипломної роботи

«Програмний засіб оптичного розпізнавання символів та структуризації текстових документів»

08-54.БДР.023.00.000 ПЗ

Науковий керівник: PhD, стар. викл.

_____Обертюх М.Р.

Студент групи 1КІ-22мс

_____ Супрун П.С.

м. Вінниця 2024

1 Підстава для виконання бакалаврської кваліфікаційної роботи (БДР)

1.1 Актуальність даного дослідження визначається необхідністю вирішення основних проблем існуючих засобів оптичного розпізнавання друкованих символів. Актуальним дане дослідження є у зв’язку з тенденціями переходу до електронного документообігу та застосування подібних засобів у різноманітних системах розпізнавання документів.

1.2 Наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР

2.1 Мета роботи полягає у підвищенні ефективності оптичного розпізнавання символів на текстових документах. Підвищення ефективності оптичного розпізнавання друкованих символів заключається у покращенні точності та швидкості розпізнавання.

2.2 Призначення розробки — виконання бакалаврської дипломної роботи із подальшим впровадженням та розвитком продукту.

3 Вихідні дані для виконання БДР

3.1 Запропонувати нові підходи для реалізації оптичного розпізнавання друкованих символів та структуризації текстових документів.

3.2 Текстовий документ у форматі .pdf або зображення з розширенням не менше за 595 пікселів по горизонталі та 842 пікселів по вертикалі.

4 Вимоги до виконання БДР

Запропонувати нові підходи для реалізації оптичного розпізнавання друкованих символів та структуризації текстових документів.

Розробити послідовність для попередньої обробки вхідних даних, розпізнавання символів за допомогою нейронної мережі та структуризації вихідних символів.

Виконати розробку програмного забезпечення для реалізації оптичного розпізнавання друкованих символів та структуризації текстових документів, провести його тестування. Скласти схему послідовності обробки та розпізнавання документа та додати лістинги програми представити в додатках до роботи.

5 Етапи БДР та очікувані результати

Етапи БДР та очікувані результати приведені у таблиці А.1.

Таблиця А.1 — Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз завдання. Вступ	25.03.24	30.03.24	Вступ
2	Аналіз предметної області	01.04.24	10.04.24	розділ 1
3	Розробка технології та вибір засобів розпізнавання друкованих символів	11.04.24	21.04.24	Розділ 2, розробка способу
4	Розробка програмних засобів розпізнавання друкованих символів та структуризації	22.04.24	14.05.24	Розділ 3, розробка програми
5	Практична реалізація, результати	15.05.24	30.05.24	Розділ 3
7	Оформлення пояснювальної записки	01.06.24	07.06.24	ПЗ, презентація

6 Матеріали, що подаються до захисту БДР:

пояснювальна записка БДР, графічні і ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відзив наукового керівника, відзив рецензента, протоколи складання державних екзаменів, анотації до БДР українською та іноземною мовами.

7 Порядок контролю виконання та захисту БДР

Виконання етапів розрахункової та графічної документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист БДР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення БДР

8.1 При оформлювання БДР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами, на які посилаються у вище вказаних.

8.2 Порядок виконання БДР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

Лістинг модуля vgg16.py

```
import tensorflow as tf
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.initializers import glorot_normal


class FeatureExtractor(tf.keras.Model):
  def __init__(self, l2 = 0):
    super().__init__()

    initial_weights = glorot_normal()
    regularizer = tf.keras.regularizers.l2(l2)
    input_shape = (None, None, 1)

    # First two convolutional blocks are frozen (not trainable)
    self._block1_conv1 = Conv2D(name = "block1_conv1", input_shape = input_shape,
kernel_size = (3,3), strides = 1, filters = 64, padding = "same", activation =
"relu", kernel_initializer = initial_weights, trainable = False)
    self._block1_conv2 = Conv2D(name = "block1_conv2", kernel_size = (3,3),
strides = 1, filters = 64, padding = "same", activation = "relu",
kernel_initializer = initial_weights, trainable = False)
    self._block1_maxpool = MaxPooling2D(pool_size = 2, strides = 2)

    self._block2_conv1 = Conv2D(name = "block2_conv1", kernel_size = (3,3),
strides = 1, filters = 128, padding = "same", activation = "relu",
kernel_initializer = initial_weights, trainable = False)
    self._block2_conv2 = Conv2D(name = "block2_conv2", kernel_size = (3,3),
strides = 1, filters = 128, padding = "same", activation = "relu",
kernel_initializer = initial_weights, trainable = False)
    self._block2_maxpool = MaxPooling2D(pool_size = 2, strides = 2)

    self._block3_conv1 = Conv2D(name = "block3_conv1", kernel_size = (3,3),
strides = 1, filters = 256, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
    self._block3_conv2 = Conv2D(name = "block3_conv2", kernel_size = (3,3),
strides = 1, filters = 256, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
    self._block3_conv3 = Conv2D(name = "block3_conv3", kernel_size = (3,3),
strides = 1, filters = 256, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
    self._block3_maxpool = MaxPooling2D(pool_size = 2, strides = 2)

    self._block4_conv1 = Conv2D(name = "block4_conv1", kernel_size = (3,3),
strides = 1, filters = 512, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
    self._block4_conv2 = Conv2D(name = "block4_conv2", kernel_size = (3,3),
strides = 1, filters = 512, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
    self._block4_conv3 = Conv2D(name = "block4_conv3", kernel_size = (3,3),
strides = 1, filters = 512, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
    self._block4_maxpool = MaxPooling2D(pool_size = 2, strides = 2)

    self._block5_conv1 = Conv2D(name = "block5_conv1", kernel_size = (3,3),
strides = 1, filters = 512, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
    self._block5_conv2 = Conv2D(name = "block5_conv2", kernel_size = (3,3),
strides = 1, filters = 512, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)
```

```
    self._block5_conv3 = Conv2D(name = "block5_conv3", kernel_size = (3,3),
strides = 1, filters = 512, padding = "same", activation = "relu",
kernel_initializer = initial_weights, kernel_regularizer = regularizer)

  def call(self, input_image):
    y = self._block1_conv1(input_image)
    y = self._block1_conv2(y)
    y = self._block1_maxpool(y)

    y = self._block2_conv1(y)
    y = self._block2_conv2(y)
    y = self._block2_maxpool(y)

    y = self._block3_conv1(y)
    y = self._block3_conv2(y)
    y = self._block3_conv3(y)
    y = self._block3_maxpool(y)

    y = self._block4_conv1(y)
    y = self._block4_conv2(y)
    y = self._block4_conv3(y)
    y = self._block4_maxpool(y)

    y = self._block5_conv1(y)
    y = self._block5_conv2(y)
    y = self._block5_conv3(y)

    return y
```

ДОДАТОК В

Лістинг модуля train.py

```
import itertools
import faster_rcnn
alphabet_capital = list('АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ')
alphabet_small = list("абвгґдеєжзиіїйклмнопрстуфхцчшщьюя")
alphabet_L_capital = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
alphabet_L_small = list("abcdefghijklmnopqrstuvwxyz")
numbers = list("1234567890")
special = list(r"~`!@#$%^&*()_-+=/<>,.:;'\"")
unified = [alphabet_capital, alphabet_small, alphabet_L_capital, alphabet_L_small,
numbers, special]
chain = list(itertools.chain(*unified))
rcnn = faster_rcnn.FasterRCNNModel(num_classes=len(chain),
allow_edge_proposals=False, custom_roi_pool=False, activate_class_outputs=True)
import argparse
import numpy as np
import os
import random
from tqdm import tqdm
from keras.optimizers import SGD
from keras.optimizers import Adam
import tensorflow as tf

from .statistics import TrainingStatistics
from .statistics import PrecisionRecallCurveCalculator
import faster_rcnn
import anchors
import make_dataset
import visualize


def render_anchors():
  training_data = make_dataset.CharDataGenerator(alphabets=unified,
fonts=dowloaded_fonts, batch_size=64)
  if not os.path.exists(options.dump_anchors):
    os.makedirs(options.dump_anchors)
  print("Rendering anchors from '%s' to set '%s'..." % (options.train_split,
options.dump_anchors))
  for sample in iter(training_data):
    output_path = os.path.join(options.dump_anchors, "anchors_" +
os.path.basename(sample.filepath) + ".png")
    visualize.show_anchors(
      output_path = output_path,
      image = sample.image,
      anchor_map = sample.anchor_map,
      anchor_valid_map = sample.anchor_valid_map,
      gt_rpn_map = sample.gt_rpn_map,
      gt_boxes = sample.gt_boxes
    )

def _sample_rpn_minibatch(rpn_map, object_indices, background_indices,
rpn_minibatch_size):
  assert rpn_map.shape[0] == 1, "Batch size must be 1"
  assert len(object_indices) == 1, "Batch size must be 1"
  assert len(background_indices) == 1, "Batch size must be 1"
  positive_anchors = object_indices[0]
  negative_anchors = background_indices[0]
  assert len(positive_anchors) + len(negative_anchors) >= rpn_minibatch_size
  assert len(positive_anchors) > 0,  assert rpn_minibatch_size % 2 == 0,
```

```
  num_positive_anchors = len(positive_anchors)
  num_negative_anchors = len(negative_anchors)
  num_positive_samples = min(rpn_minibatch_size // 2, num_positive_anchors)
  num_negative_samples = rpn_minibatch_size - num_positive_samples
  positive_anchor_idxs = random.sample(range(num_positive_anchors),
num_positive_samples)
  negative_anchor_idxs = random.sample(range(num_negative_anchors),
num_negative_samples)

  positive_anchors = positive_anchors[positive_anchor_idxs]
  negative_anchors = negative_anchors[negative_anchor_idxs]
  trainable_anchors = np.concatenate([ positive_anchors, negative_anchors ])
  batch_idxs = np.zeros(len(trainable_anchors), dtype = int)
  trainable_idxs = (batch_idxs, trainable_anchors[:,0], trainable_anchors[:,1],
trainable_anchors[:,2], 0)

  rpn_minibatch_map = rpn_map.copy()
  rpn_minibatch_map[:,:,:,:,0] = 0
  rpn_minibatch_map[trainable_idxs] = 1

  return rpn_minibatch_map

def _convert_training_sample_to_model_input(sample, mode):
    gt_box_corners = np.array([ box.corners for box in sample.gt_boxes
]).astype(np.float32)        # (num_boxes,4), where each row is (y1,x1,y2,x2)
    gt_box_class_idxs = np.array([ box.class_index for box in sample.gt_boxes
]).astype(np.int32)    # (num_boxes,), where each is an index [1,num_classes)

    image_data = np.expand_dims(sample.image_data, axis = 0)
    anchor_map = np.expand_dims(sample.anchor_map, axis = 0)
    anchor_valid_map = np.expand_dims(sample.anchor_valid_map, axis = 0)
    gt_rpn_map = np.expand_dims(sample.gt_rpn_map, axis = 0)
    gt_rpn_object_indices = [ sample.gt_rpn_object_indices ]
    gt_rpn_background_indices = [ sample.gt_rpn_background_indices ]
    gt_box_corners = np.expand_dims(gt_box_corners, axis = 0)
    gt_box_class_idxs = np.expand_dims(gt_box_class_idxs, axis = 0)

    gt_rpn_minibatch_map = _sample_rpn_minibatch(
      rpn_map = gt_rpn_map,
      object_indices = gt_rpn_object_indices,
      background_indices = gt_rpn_background_indices,
      rpn_minibatch_size = 256
    )

    if mode == "train":
      x = [ image_data, anchor_map, anchor_valid_map, gt_rpn_minibatch_map,
gt_box_class_idxs, gt_box_corners ]
    else:
      x = [ image_data, anchor_map, anchor_valid_map ]

    return x, image_data, gt_rpn_minibatch_map

def evaluate(model, eval_data = None, num_samples = None, plot = False,
print_average_precisions = False):
  if eval_data is None:
    eval_data = voc.Dataset(dir = options.dataset_dir, split = options.eval_split,
augment = False, shuffle = False)
  if num_samples is None:
    num_samples = eval_data.num_samples
  precision_recall_curve = PrecisionRecallCurveCalculator()
  i = 0
  print("Evaluating '%s'..." % eval_data.split)
  for sample in tqdm(iterable = iter(eval_data), total = num_samples):
    x, image_data, _ = _convert_training_sample_to_model_input(sample = sample,
```

```
mode = "infer")
    scored_boxes_by_class_index = model.predict_on_batch(x = x, score_threshold =
0.05)
    precision_recall_curve.add_image_results(
      scored_boxes_by_class_index = scored_boxes_by_class_index,
      gt_boxes = sample.gt_boxes
    )
    i += 1
    if i >= num_samples:
      break
  mean_average_precision = 100.0 *
precision_recall_curve.compute_mean_average_precision()
  print("Mean Average Precision = %1.2f%%" % mean_average_precision)e)
  return mean_average_precision

def train(model):
  print("Training Parameters")
  print("-------------------")
  print("Initial weights           : %s" % (options.load_from if options.load_from
else "Keras VGG-16 ImageNet weights"))
  print("Dataset                   : %s" % options.dataset_dir)
  print("Training split            : %s" % options.train_split)
  print("Evaluation split          : %s" % options.eval_split)
  print("Epochs                    : %d" % options.epochs)
  print("Optimizer                 : %s" % options.optimizer)
  print("Learning rate             : %f" % options.learning_rate)
  print("Gradient norm clipping    : %s" % ("disabled" if options.clipnorm <= 0
else ("%f" % options.clipnorm)))
  print("SGD momentum              : %f" % options.momentum)
  print("Adam Beta-1               : %f" % options.beta1)
  print("Adam Beta-2               : %f" % options.beta2)
  print("Weight decay              : %f" % options.weight_decay)
  print("Dropout                   : %f" % options.dropout)
  print("RoI pooling implementation: %s" % ("custom" if options.custom_roi_pool
else "crop-and-resize w/ max pool"))
  print("Detector output           : %s" % ("logits" if options.detector_logits
else "probabilities"))
  print("Augmentation              : %s" % ("disabled" if options.no_augment else
"enabled"))
  print("Edge proposals            : %s" % ("excluded" if
options.exclude_edge_proposals else "included"))
  print("CSV log                   : %s" % ("none" if not options.log_csv else
options.log_csv))
  print("Checkpoints               : %s" % ("disabled" if not
options.checkpoint_dir else options.checkpoint_dir))
  print("Final weights file        : %s" % ("none" if not options.save_to else
options.save_to))
  print("Best weights file         : %s" % ("none" if not options.save_best_to
else options.save_best_to))
  training_data = make_dataset.CharDataGenerator(alphabets=unified,
fonts=downloaded_fonts, batch_size=64)
  eval_data = make_dataset.CharDataGenerator(alphabets=unified,
fonts=downloaded_fonts, batch_size=64)
  if options.checkpoint_dir and not os.path.exists(options.checkpoint_dir):
    os.makedirs(options.checkpoint_dir)
  for epoch in range(1, 1 + options.epochs):
    print("Epoch %d/%d" % (epoch, options.epochs))
    stats = TrainingStatistics()
    progbar = tqdm(iterable = iter(training_data), total =
training_data.num_samples, postfix = stats.get_progbar_postfix())
    for sample in progbar:
      x, image_data, gt_rpn_minibatch_map =
_convert_training_sample_to_model_input(sample = sample, mode = "train")
      losses = model.train_on_batch(x = x, y = gt_rpn_minibatch_map, return_dict =
```

```
True)
      stats.on_training_step(losses = losses)
      progbar.set_postfix(stats.get_progbar_postfix())
    mean_average_precision = evaluate(
      model = model,
      eval_data = eval_data,
      num_samples = options.periodic_eval_samples,
      plot = False,
      print_average_precisions = False
    )
    if options.checkpoint_dir:
      checkpoint_file = os.path.join(options.checkpoint_dir, "checkpoint-epoch-%d-
mAP-%1.1f.h5" % (epoch, mean_average_precision))
      model.save_weights(filepath = checkpoint_file, overwrite = True, save_format
= "h5")
      print("Saved model checkpoint to '%s'" % checkpoint_file)
    if options.log_csv:
      log_items = {
        "epoch": epoch,
        "learning_rate": options.learning_rate,
        "clipnorm": options.clipnorm,
        "momentum": options.momentum,
        "beta1": options.beta1,
        "beta2": options.beta2,
        "weight_decay": options.weight_decay,
        "dropout": options.dropout,
        "mAP": mean_average_precision
      }
      log_items.update(stats.get_progbar_postfix())
  if options.save_to:
    model.save_weights(filepath = options.save_to, overwrite = True, save_format =
"h5")
    print("Saved final model weights to '%s'" % options.save_to)
  print("Evaluating %s model on all samples in '%s'..." % (("best" if
options.save_best_to else "final"), options.eval_split))
  evaluate(
    model = model,
    eval_data = eval_data,
    num_samples = eval_data.num_samples,
    plot = options.plot,
    print_average_precisions = True
  )

def _predict(model, image_data, image, show_image, output_path):
  anchor_map, anchor_valid_map = anchors.generate_anchor_maps(image_shape =
image_data.shape, feature_pixels = 16)
  anchor_map = np.expand_dims(anchor_map, axis = 0)
  anchor_valid_map = np.expand_dims(anchor_valid_map, axis = 0)
  image_data = np.expand_dims(image_data, axis = 0)
  x = [ image_data, anchor_map, anchor_valid_map ]
  scored_boxes_by_class_index = model.predict_on_batch(x = x, score_threshold =
0.7)
  visualize.show_detections(
    output_path = output_path,
    show_image = show_image,
    image = image,
    scored_boxes_by_class_index = scored_boxes_by_class_index,
    class_index_to_name = voc.Dataset.class_index_to_name
  )

def predict_one(model, url, show_image, output_path):
  from .datasets.image import load_image
  image_data, image, _, _ = load_image(url = url, min_dimension_pixels = 600)
  _predict(model = model, image_data = image_data, image = image, show_image =
```

```
show_image, output_path = output_path)

def predict_all(model, split):
  dirname = "predictions_" + split
  if not os.path.exists(dirname):
    os.makedirs(dirname)
  print("Rendering predictions from '%s' set to '%s'..." % (split, dirname))
  dataset = voc.Dataset(dir = options.dataset_dir, split = split, augment = False,
shuffle = False)
  for sample in iter(dataset):
    output_path = os.path.join(dirname,
os.path.splitext(os.path.basename(sample.filepath))[0] + ".png")
    _predict(model = model, image_data = sample.image_data, image = sample.image,
show_image = False, output_path = output_path)

def create_optimizer():
  kwargs = {}
  if options.clipnorm > 0:
    kwargs = { "clipnorm": options.clipnorm }
  if options.optimizer == "sgd":
    optimizer = SGD(learning_rate = options.learning_rate, momentum =
options.momentum, **kwargs)
  elif options.optimizer == "adam":
    optimizer = Adam(learning_rate = options.learning_rate, beta_1 =
options.beta1, beta_2 = options.beta2, **kwargs)
  else:
    raise ValueError("Optimizer must be 'sgd' for stochastic gradient descent or
'adam' for Adam")
  return optimizer


if __name__ == "__main__":
  parser = argparse.ArgumentParser("FasterRCNN")
  group = parser.add_mutually_exclusive_group()
  group.add_argument("--train", action = "store_true", help = "Train model")
  group.add_argument("--eval", action = "store_true", help = "Evaluate model")
  group.add_argument("--predict", metavar = "url", action = "store", type = str,
help = "Run inference on image and display detected boxes")
  group.add_argument("--predict-to-file", metavar = "url", action = "store", type
= str, help = "Run inference on image and render detected boxes to
'predictions.png'")
  group.add_argument("--predict-all", metavar = "name", action = "store", type =
str, help = "Run inference on all images in the specified dataset split and write
to directory 'predictions_${split}'")
  parser.add_argument("--load-from", metavar = "file", action = "store", help =
"Load initial model weights from file")
  parser.add_argument("--save-to", metavar = "file", action = "store", help =
"Save final trained weights to file")
  parser.add_argument("--save-best-to", metavar = "file", action = "store", help =
"Save best weights (highest mean average precision) to file")
  parser.add_argument("--dataset-dir", metavar = "dir", action = "store", default
= "VOCdevkit/VOC2007", help = "VOC dataset directory")
  parser.add_argument("--train-split", metavar = "name", action = "store", default
= "trainval", help = "Dataset split to use for training")
  parser.add_argument("--eval-split", metavar = "name", action = "store", default
= "test", help = "Dataset split to use for evaluation")
  parser.add_argument("--cache-images", action = "store_true", help = "Cache
images during training (requires ample CPU memory)")
  parser.add_argument("--periodic-eval-samples", metavar = "count", action =
"store", default = 1000, help = "Number of samples to use during evaluation after
each epoch")
  parser.add_argument("--checkpoint-dir", metavar = "dir", action = "store", help
= "Save checkpoints after each epoch to the given directory")
  parser.add_argument("--plot", action = "store_true", help = "Plots the average
```

```
precision of each class after evaluation (use with --train or --eval)")
  parser.add_argument("--log-csv", metavar = "file", action = "store", help = "Log
training metrics to CSV file")
  parser.add_argument("--epochs", metavar = "count", type = int, action = "store",
default = 1, help = "Number of epochs to train for")
  parser.add_argument("--optimizer", metavar = "name", type = str, action =
"store", default = "sgd", help = "Optimizer to use (\"sgd\" or \"adam\")")
  parser.add_argument("--learning-rate", metavar = "value", type = float, action =
"store", default = 1e-3, help = "Learning rate")
  parser.add_argument("--clipnorm", metavar = "value", type = float, action =
"store", default = 0.0, help = "Gradient norm clipping (use 0 for none)")
  parser.add_argument("--momentum", metavar = "value", type = float, action =
"store", default = 0.9, help = "SGD momentum")
  parser.add_argument("--beta1", metavar = "value", type = float, action =
"store", default = 0.9, help = "Adam beta1 parameter (decay rate for 1st moment
estimates)")
  parser.add_argument("--beta2", metavar = "value", type = float, action =
"store", default = 0.999, help = "Adam beta2 parameter (decay rate for 2nd moment
estimates)")
  parser.add_argument("--weight-decay", metavar = "value", type = float, action =
"store", default = 5e-4, help = "Weight decay")
  parser.add_argument("--dropout", metavar = "probability", type = float, action =
"store", default = 0.0, help = "Dropout probability after each of the two fully-
connected detector layers")
  parser.add_argument("--custom-roi-pool", action = "store_true", help = "Use
custom RoI pool implementation instead of TensorFlow crop-and-resize with max-pool
(much slower)")
  parser.add_argument("--detector-logits", action = "store_true", help = "Do not
apply softmax to detector class output and compute loss from logits directly")
  parser.add_argument("--no-augment", action = "store_true", help = "Disable image
augmentation (random horizontal flips) during training")
  parser.add_argument("--exclude-edge-proposals", action = "store_true", help =
"Exclude proposals generated at anchors spanning image edges from being passed to
detector stage")
  parser.add_argument("--dump-anchors", metavar = "dir", action = "store", help =
"Render out all object anchors and ground truth boxes from the training set to a
directory")
  parser.add_argument("--debug-dir", metavar = "dir", action = "store", help =
"Enable full TensorFlow Debugger V2 logging to specified directory")
  options = parser.parse_args()

  # Run-time environment
  cuda_available = tf.test.is_built_with_cuda()
  gpu_available = tf.test.is_gpu_available(cuda_only = False,
min_cuda_compute_capability = None)
  print("CUDA Available : %s" % ("yes" if cuda_available else "no"))
  print("GPU Available  : %s" % ("yes" if gpu_available else "no"))
  print("Eager Execution: %s" % ("yes" if tf.executing_eagerly() else "no"))

  # Perform optional procedures
  if options.dump_anchors:
    render_anchors()

  # Debug logging
  if options.debug_dir:
    tf.debugging.experimental.enable_dump_debug_info(options.debug_dir,
tensor_debug_mode = "FULL_HEALTH", circular_buffer_size = -1)

  # Construct model and load initial weights
  model = faster_rcnn.FasterRCNNModel(
    num_classes = voc.Dataset.num_classes,
    allow_edge_proposals = not options.exclude_edge_proposals,
    custom_roi_pool = options.custom_roi_pool,
    activate_class_outputs = not options.detector_logits,
```

```
    l2 = 0.5 * options.weight_decay,
    dropout_probability = options.dropout
  )
  model.build(
    input_shape = [
      (1, None, None, 3),      # input_image: (1, height_pixels, width_pixels, 3)
      (1, None, None, 9 * 4), # anchor_map: (1, height, width, num_anchors * 4)
      (1, None, None, 9),      # anchor_valid_map: (1, height, width, num_anchors)
      (1, None, None, 9, 6),   # gt_rpn_map: (1, height, width, num_anchors, 6)
      (1, None),               # gt_box_class_idxs_map: (1, num_gt_boxes)
      (1, None, 4)             # gt_box_corners_map: (1, num_gt_boxes, 4)
    ]
  )
  model.compile(optimizer = create_optimizer()) #
  if options.load_from:
    model.load_weights(filepath = options.load_from, by_name = True)
    print("Loaded initial weights from '%s'" % options.load_from)
  else:
    model.load_imagenet_weights()
    print("Initialized VGG-16 layers to Keras ImageNet weights")

   if options.train:
    train(model = model)
  elif options.eval:
    evaluate(model = model, plot = options.plot, print_average_precisions = True)
  elif options.predict:
    predict_one(model = model, url = options.predict, show_image = True,
output_path = None)
  elif options.predict_to_file:
    predict_one(model = model, url = options.predict_to_file, show_image = False,
output_path = "predictions.png")
  elif options.predict_all:
    predict_all(model = model, split = options.predict_all)
  elif not options.dump_anchors:
    print("Nothing to do. Did you mean to use --train or --predict?")
```

ДОДАТОК Г

Лістинг модуля faster_rcnn.py

```
import numpy as np
import tensorflow as tf


import vgg16
import rpn
import detector
import math_utils


class FasterRCNNModel(tf.keras.Model):
  def __init__(self, num_classes, allow_edge_proposals, custom_roi_pool,
activate_class_outputs, l2 = 0, dropout_probability = 0):
    super().__init__()
    self._num_classes = num_classes
    self._activate_class_outputs = activate_class_outputs
    self._stage1_feature_extractor = vgg16.FeatureExtractor(l2 = l2)
    self._stage2_region_proposal_network = rpn.RegionProposalNetwork(
      max_proposals_pre_nms_train = 12000,
      max_proposals_post_nms_train = 2000,
      max_proposals_pre_nms_infer = 6000,
      max_proposals_post_nms_infer = 300,
      l2 = l2,
      allow_edge_proposals = allow_edge_proposals
    )
    self._stage3_detector_network = detector.DetectorNetwork(
      num_classes = num_classes,
      custom_roi_pool = custom_roi_pool,
      activate_class_outputs = activate_class_outputs,
      l2 = l2,
      dropout_probability = dropout_probability
    )

  def call(self, inputs, training=False):

    input_image = inputs[0]              # (1, height_pixels, width_pixels, 3)
    anchor_map = inputs[1]               # (1, height, width, num_anchors * 4)
    anchor_valid_map = inputs[2]         # (1, height, width, num_anchors)
    if training:
      gt_rpn_map = inputs[3]             # (1, height, width, num_anchors, 6)
      gt_box_class_idxs_map = inputs[4] # (1, num_gt_boxes)
      gt_box_corners_map = inputs[5]     # (1, num_gt_boxes, 4)


    feature_map = self._stage1_feature_extractor(input_image = input_image,
training = training)


    rpn_scores, rpn_box_deltas, proposals = self._stage2_region_proposal_network(
      inputs = [
        input_image,
        feature_map,
        anchor_map,
        anchor_valid_map
      ],
      training = training
    )


    if training:
```

```
      # Assign labels to proposals and take random sample (for detector training)
      proposals, gt_classes, gt_box_deltas = self._label_proposals(
        proposals = proposals,
        gt_box_class_idxs = gt_box_class_idxs_map[0], # for now, batch size of 1
        gt_box_corners = gt_box_corners_map[0],
        min_background_iou_threshold = 0.0,
        min_object_iou_threshold = 0.5
      )
      proposals, gt_classes, gt_box_deltas = self._sample_proposals(
        proposals = proposals,
        gt_classes = gt_classes,
        gt_box_deltas = gt_box_deltas,
        max_proposals = 128,
        positive_fraction = 0.25
      )
      gt_classes = tf.expand_dims(gt_classes, axis = 0)           #
(N,num_classes) -> (1,N,num_classes) (as expected by loss function)
      gt_box_deltas = tf.expand_dims(gt_box_deltas, axis = 0)    #
(N,2,(num_classes-1)*4) -> (1,N,2,(num_classes-1)*4)

      # Ensure proposals are treated as constants and do not propagate gradients
      proposals = tf.stop_gradient(proposals)
      gt_classes = tf.stop_gradient(gt_classes)
      gt_box_deltas = tf.stop_gradient(gt_box_deltas)

    # Stage 3: Detector
    detector_classes, detector_box_deltas = self._stage3_detector_network(
      inputs = [
        input_image,
        feature_map,
        proposals
      ],
      training = training
    )

    # Losses
    if training:
      rpn_class_loss = self._stage2_region_proposal_network.class_loss(y_predicted
= rpn_scores, gt_rpn_map = gt_rpn_map)
      rpn_regression_loss =
self._stage2_region_proposal_network.regression_loss(y_predicted = rpn_box_deltas,
gt_rpn_map = gt_rpn_map)
      detector_class_loss = self._stage3_detector_network.class_loss(y_predicted =
detector_classes, y_true = gt_classes, from_logits = not
self._activate_class_outputs)
      detector_regression_loss =
self._stage3_detector_network.regression_loss(y_predicted = detector_box_deltas,
y_true = gt_box_deltas)
      self.add_loss(rpn_class_loss)
      self.add_loss(rpn_regression_loss)
      self.add_loss(detector_class_loss)
      self.add_loss(detector_regression_loss)
      self.add_metric(rpn_class_loss, name = "rpn_class_loss")
      self.add_metric(rpn_regression_loss, name = "rpn_regression_loss")
      self.add_metric(detector_class_loss, name = "detector_class_loss")
      self.add_metric(detector_regression_loss, name = "detector_regression_loss")
    else:
      # Losses cannot be computed during inference and should be ignored
      rpn_class_loss = float("inf")
      rpn_regression_loss = float("inf")
      detector_class_loss = float("inf")
      detector_regression_loss = float("inf")

    # Return outputs
```

```
    return [
      rpn_scores,
      rpn_box_deltas,
      detector_classes,
      detector_box_deltas,
      proposals,
      rpn_class_loss,
      rpn_regression_loss,
      detector_class_loss,
      detector_regression_loss
    ]

  def predict_on_batch(self, x, score_threshold):
    _, _, detector_classes, detector_box_deltas, proposals, _, _, _, _ =
super().predict_on_batch(x = x)
    scored_boxes_by_class_index = self._predictions_to_scored_boxes(
      input_image = x[0],
      classes = detector_classes,
      box_deltas = detector_box_deltas,
      proposals = proposals,
      score_threshold = score_threshold
    )
    return scored_boxes_by_class_index

  def load_imagenet_weights(self):

    keras_model = tf.keras.applications.VGG16(weights = "imagenet")
    for keras_layer in keras_model.layers:
      weights = keras_layer.get_weights()
      if len(weights) > 0:
        vgg16_layers = self._stage1_feature_extractor.layers +
self._stage3_detector_network.layers
        our_layer = [ layer for layer in vgg16_layers if layer.name ==
keras_layer.name ]
        if len(our_layer) > 0:
          print("Loading VGG-16 ImageNet weights into layer: %s" %
our_layer[0].name)
          our_layer[0].set_weights(weights)

  def _predictions_to_scored_boxes(self, input_image, classes, box_deltas,
proposals, score_threshold):
    input_image = np.squeeze(input_image, axis = 0)
    classes = np.squeeze(classes, axis = 0)
    box_deltas = np.squeeze(box_deltas, axis = 0)


    if not self._activate_class_outputs:
      classes = tf.nn.softmax(classes, axis = 1).numpy()

        proposal_anchors = np.empty(proposals.shape)
    proposal_anchors[:,0] = 0.5 * (proposals[:,0] + proposals[:,2]) # center_y
    proposal_anchors[:,1] = 0.5 * (proposals[:,1] + proposals[:,3]) # center_x
    proposal_anchors[:,2:4] = proposals[:,2:4] - proposals[:,0:2]   # height,
width


    boxes_and_scores_by_class_idx = {}
    for class_idx in range(1, classes.shape[1]):
      box_delta_idx = (class_idx - 1) * 4
      box_delta_params = box_deltas[:, (box_delta_idx + 0) : (box_delta_idx + 4)]
# (N, 4)
      proposal_boxes_this_class = math_utils.convert_deltas_to_boxes(
        box_deltas = box_delta_params,
        anchors = proposal_anchors,
```

```
        box_delta_means = [0.0, 0.0, 0.0, 0.0],
        box_delta_stds = [0.1, 0.1, 0.2, 0.2]
      )


      proposal_boxes_this_class[:,0::2] =
np.clip(proposal_boxes_this_class[:,0::2], 0, input_image.shape[0] - 1)
      proposal_boxes_this_class[:,1::2] =
np.clip(proposal_boxes_this_class[:,1::2], 0, input_image.shape[1] - 1)
      scores_this_class = classes[:,class_idx]
      sufficiently_scoring_idxs = np.where(scores_this_class > score_threshold)[0]
      proposal_boxes_this_class =
proposal_boxes_this_class[sufficiently_scoring_idxs]
      scores_this_class = scores_this_class[sufficiently_scoring_idxs]
      boxes_and_scores_by_class_idx[class_idx] = (proposal_boxes_this_class,
scores_this_class)

    scored_boxes_by_class_idx = {}
    for class_idx, (boxes, scores) in boxes_and_scores_by_class_idx.items():
      idxs = tf.image.non_max_suppression(
        boxes = boxes,
        scores = scores,
        max_output_size = proposals.shape[0],
        iou_threshold = 0.3
      )
      idxs = idxs.numpy()
      boxes = boxes[idxs]
      scores = np.expand_dims(scores[idxs], axis = 0) # (N,) -> (N,1)
      scored_boxes = np.hstack([ boxes, scores.T ])
      scored_boxes_by_class_idx[class_idx] = scored_boxes

    return scored_boxes_by_class_idx

  def _label_proposals(self, proposals, gt_box_class_idxs, gt_box_corners,
min_background_iou_threshold, min_object_iou_threshold):

    proposals = tf.concat([ proposals, gt_box_corners ], axis = 0)
    ious = math_utils.tf_intersection_over_union(boxes1 = proposals, boxes2 =
gt_box_corners)


    best_ious = tf.math.reduce_max(ious, axis = 1)
    box_idxs = tf.math.argmax(ious, axis = 1)
    gt_box_class_idxs = tf.gather(gt_box_class_idxs, indices = box_idxs)
    gt_box_corners = tf.gather(gt_box_corners, indices = box_idxs)

    idxs = tf.where(best_ious >= min_background_iou_threshold)
    proposals = tf.gather_nd(proposals, indices = idxs)
    best_ious = tf.gather_nd(best_ious, indices = idxs)
    gt_box_class_idxs = tf.gather_nd(gt_box_class_idxs, indices = idxs)
    gt_box_corners = tf.gather_nd(gt_box_corners, indices = idxs)


    retain_mask = tf.cast(best_ious >= min_object_iou_threshold, dtype =
gt_box_class_idxs.dtype)
    gt_box_class_idxs = gt_box_class_idxs * retain_mask

        num_classes = self._num_classes
    gt_classes = tf.one_hot(indices = gt_box_class_idxs, depth = num_classes) #
(N,num_classes)
    proposal_centers = 0.5 * (proposals[:,0:2] + proposals[:,2:4])          #
center_y, center_x
    proposal_sides = proposals[:,2:4] - proposals[:,0:2]                    #
height, width
```

```
    gt_box_centers = 0.5 * (gt_box_corners[:,0:2] + gt_box_corners[:,2:4])  #
center_y, center_x
    gt_box_sides = gt_box_corners[:,2:4] - gt_box_corners[:,0:2]            #
height, width


    detector_box_delta_means = tf.constant([0, 0, 0, 0], dtype = tf.float32)
    detector_box_delta_stds = tf.constant([0.1, 0.1, 0.2, 0.2], dtype =
tf.float32)
    tyx = (gt_box_centers - proposal_centers) / proposal_sides
    thw = tf.math.log(gt_box_sides / proposal_sides)
    box_delta_targets = tf.concat([ tyx, thw ], axis = 1)
    box_delta_targets = (box_delta_targets - detector_box_delta_means) /
detector_box_delta_stds


    gt_box_deltas_mask = tf.repeat(gt_classes, repeats = 4, axis = 1)[:,4:]
gt_box_deltas_values = tf.tile(box_delta_targets, multiples = [1, num_classes -
1])
    gt_box_deltas_mask = tf.expand_dims(gt_box_deltas_mask, axis = 0)      #
(N,4*(C-1)) -> (1,N,4*(C-1))
    gt_box_deltas_values = tf.expand_dims(gt_box_deltas_values, axis = 0) #
(N,4*(C-1)) -> (1,N,4*(C-1))
    gt_box_deltas = tf.concat([ gt_box_deltas_mask, gt_box_deltas_values ], axis =
0) # (2,N,4*(C-1))
    gt_box_deltas = tf.transpose(gt_box_deltas, perm = [ 1, 0, 2])         #
(N,2,4*(C-1))

    return proposals, gt_classes, gt_box_deltas

  def _sample_proposals(self, proposals, gt_classes, gt_box_deltas, max_proposals,
positive_fraction):
    if max_proposals <= 0:
      return proposals, gt_classes, gt_box_deltas


    class_indices = tf.argmax(gt_classes, axis = 1)
    positive_indices = tf.squeeze(tf.where(class_indices > 0), axis = 1)
    negative_indices = tf.squeeze(tf.where(class_indices <= 0), axis = 1)
    num_positive_proposals = tf.size(positive_indices)
    num_negative_proposals = tf.size(negative_indices)

    num_samples = tf.minimum(max_proposals, tf.size(class_indices))
    num_positive_samples = tf.minimum(tf.cast(tf.math.round(tf.cast(num_samples,
dtype = float) * positive_fraction), dtype = num_samples.dtype),
num_positive_proposals)
    num_negative_samples = tf.minimum(num_samples - num_positive_samples,
num_negative_proposals)


    positive_sample_indices =
tf.random.shuffle(positive_indices)[:num_positive_samples]
    negative_sample_indices =
tf.random.shuffle(negative_indices)[:num_negative_samples]
    indices = tf.concat([ positive_sample_indices, negative_sample_indices ], axis
= 0)

    return tf.gather(proposals, indices = indices), tf.gather(gt_classes, indices
= indices), tf.gather(gt_box_deltas, indices = indices)
```

ДОДАТОК Е

Послідовність оптичного розпізнавання та структуризації тексту

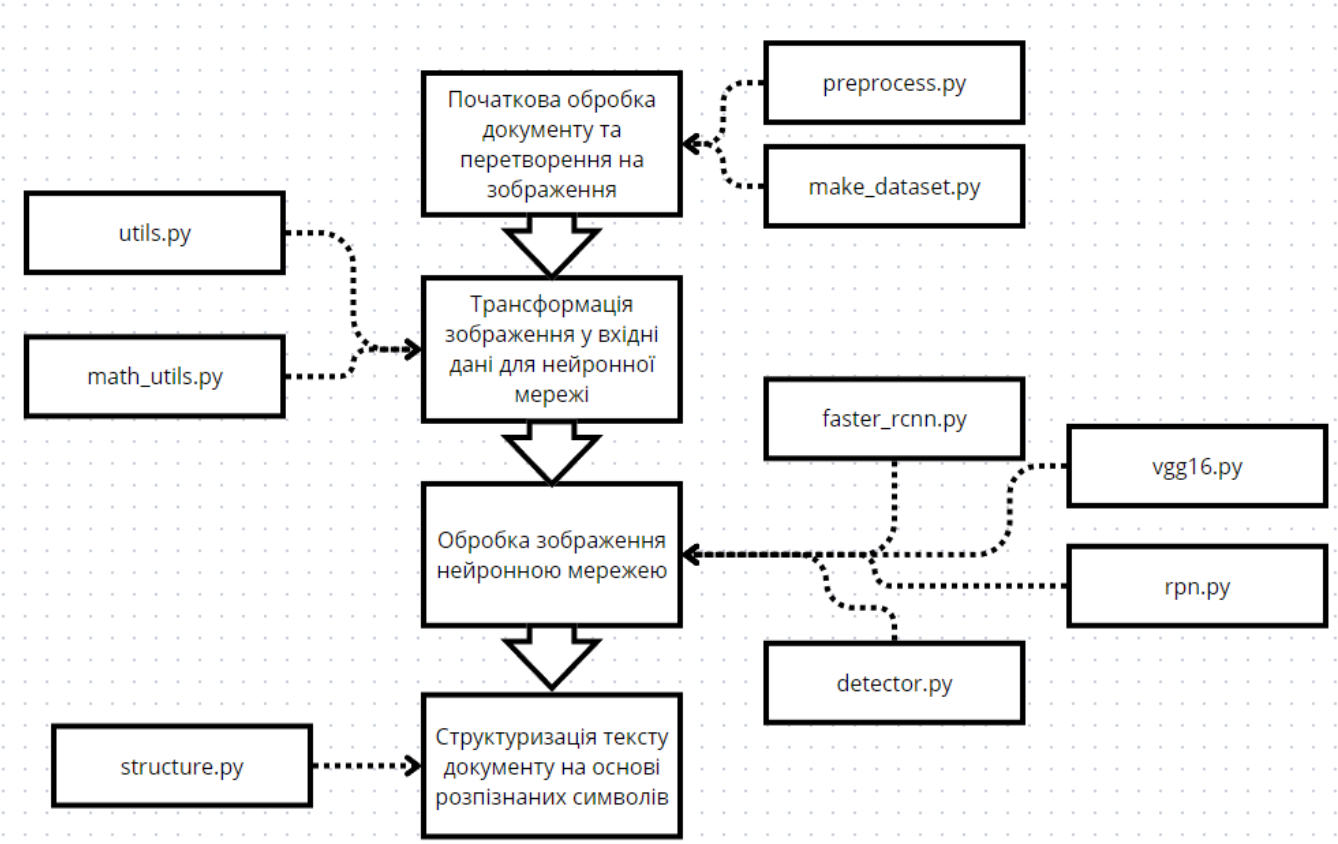

Рисунок Е.1 — Послідовність оптичного розпізнавання та структуризації тексту

ДОДАТОК Ж

Модель процесу розпізнавання символів

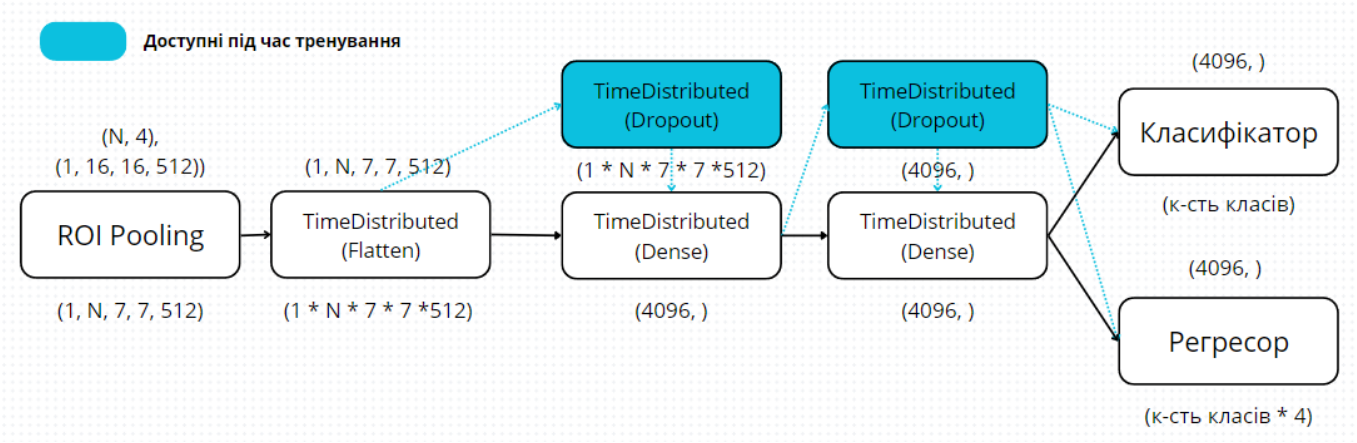

Рисунок Ж.1 — Модель процесу розпізнавання символів

ДОДАТОК И

Архітектура згорткової нейронної мережі

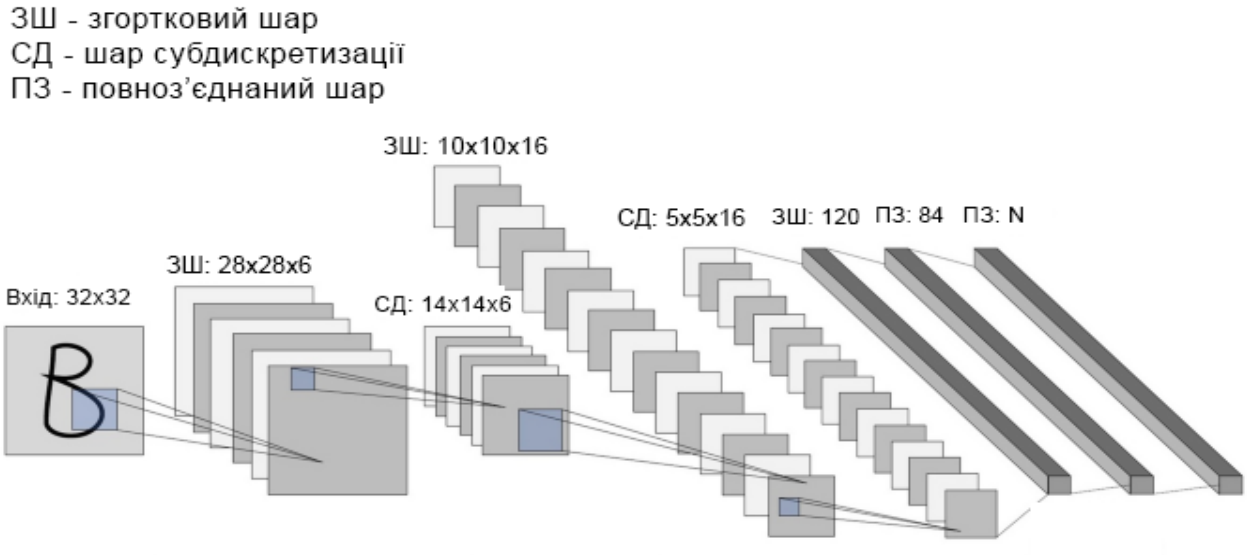

Рисунок И.1 — Архітектура згорткової нейронної мережі

ДОДАТОК К

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний засіб оптичного розпізнавання символів та структуризації текстових документів

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 99% Схожість 1%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Супрун П. С.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Обертюх М. Р.
(підпис) (прізвище, ініціали)