Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Використання згорткових нейронних мереж для обробки зображень

(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-5
Абдельхамід Бакрі Ал Сарміні
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник доц. Євген Павленко
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Бакрі Ал Сарміні Абдельхаміду Мустафі
(прізвище, ім'я, по батькові)

1. Тема роботи Використання згорткових нейронних мереж для обробки зображень

затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 19 червня 20 25 р.
3. Вихідні дані до роботи Науково технічні публікації, дані Інтернет-джерел та наукових проектів щодо розробки та дослідження глибоких нейронних мереж, офіційна документація документація нейромережевих бібліотек на Python, набори даних для навчання нейромереж

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Методологія дослідження
3) Програмна реалізація
4) Порівняльний аналіз

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі та постановка задачі	23.05.2025	виконано
3	Методологія дослідження	24.05.2025	виконано
4	Програмна реалізація	25.05.2025	виконано
5	Порівняльний аналіз	26.05.2025	виконано
6	Написання висновків	27.05.2025	виконано
7	Захист перед ЕК	19.06.2025	

Дата видачі завдання 19 травня 2025 р.

Здобувач  _____
(підпис)

Керівник роботи _____ доц. Євген Павленко
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 92 с., 41 рис., 1 дод., 14 джерел.

ГЛИБИННЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ, ОПТИЧНЕ РОЗПІЗНАВАННЯ СИМВОЛІВ, РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ.

Об’єкт дослідження – нейронні мережі.

Предмет дослідження – застосування глибоких нейронних мереж для розпізнавання та класифікації тексту на зображеннях.

Мета роботи – дослідження та порівняння методів глибокого навчання для вирішення задачі OCR.

Методи дослідження – аналіз технічної літератури та новісних досліджень в сфері глибокого навчання, експериментальний підбір архітектури та конфігурацій, використання готових рішень, аналіз результатів.

У ході цієї роботі було порівняно три різні підходи, використання готових преднавчанних двох нейромереж для розпізнавання та класифікації тексту на зображенні, та створення кастомної нейромережі для цієї задачі. Були порівняні результати трьох нейромереж, та написані розгорнуті висновки щодо роботи та перспектив розвитку таких нейромереж для вирішання задачі OCR.

ABSTRACT

Bachelor’s thesis contains: 92 pp., 41 fig., 1 ann., 14 references.

CONVOLUTIONAL NEURAL NETWORKS, CONVOLUTIONAL RECURRENT NEURAL NETWORKS, DEEP LEARNING, OPTICAL CHARACTER RECOGNITION, RECURRENT NEURAL NETWORKS.

The object of research – neural networks.

The subject of research – application of deep neural networks for recognizing and classifying text in images.

The purpose of the work – to study and compare deep learning methods for solving the OCR problem.

Research methods are – an analysis of technical literature and recent research in the field of deep learning, experimental selection of architecture and configurations, use of ready-made solutions, analysis of results.

In the course of this work, three different approaches were compared, the use of ready-made pre-trained two neural networks for recognizing and classifying text in images, and the creation of a custom neural network for this problem. The results of three neural networks were compared, and detailed conclusions were written regarding their performance and the prospects for the development of such neural networks for solving the OCR problem.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN – Convolutional Neural Network – згорткова нейронна мережа;

CRAFT – Character Region Awareness for Text Detection – розпізнавання області символів для тексту;

CRNN – Convolutional Recurrent Neural Network – згорткова рекурентна нейронна мережа;

EAST – Efficient and Accurate Scene Text detector – ефективний та точний детектор тексту в сцені;

LSTM – Long Short-Term Memory – довга короткочасна пам’ять;

MLP – Multi-Layer Perceptron – багатошаровий перцептрон;

OCR – Optical Character Recognition – оптичне розпізнавання символів;

SGD – Stochastic Gradient Descent – стохастичний градієнтний спуск.

ВСТУП

Тема «Штучного інтелекту» у наші часи є майже найпопулярнішою темою. Використання інтелектуальних технологій та нейромереж у наші часи є невід’ємним атрибутом життя для багатьох людей на планеті, навіть не замислюючись люди використовують ці технології. Тому ця тема є дуже актуальною.

Текст як взагалі це один із найпоширеніших якщо не й найпоширініший інструмент для передачі, зберігання та використання інформації інструмент. Тому людству завжди потрібно було розробляти методи як працювати із текстом, спочатку це були наскальні написи, пергаменти, папер. Із появою друкувальної машини, текст як інструмент передачі інформації, набув всесвітнього масштабу, із появою комп’ютерів людство це може бачити ще більше.

Але ж, текст як інформацію треба обробляти, на щастя із розвитком технологій ми можемо дуже ефективно обробляти текст, так як 50 років тому і подумати не могли. Розвиток мовних моделей визвав колосальний інтерес до нейромереж. Тому ми маємо технології з якими можемо дуже ефективно обробляти різні тексти, або ж малюнки на яких є текст.

Виявлення тексту на фото є задачаю якою називають OCR, це задача оптичного розпізнавання символів на різних носіях інформації, зазвичай на фото.

Метою цієї роботи буде розробити та порівняти три нейромережі різної архітектури, у вирішенні задачі OCR. Буде використано дві готові нейромережі, а третю, згорткову або частично згорткови буде розроблено самостійно, та проведено детальний аналіз.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

Сучасні технології дуже швидко та стрімко розвиваються, серед них велику увагу привертають штучні нейронні мережі. Вони вже давно вийшли за межі академічних досліджень або навіть маргінальності, і стали основою багатьох рішень які ми щодня бачимо та використовуємо у застосунках, онлайн-сервісах, камерах смартфонів та навіть в автомобілях.

Будуть розглянуті такі питання як що таке нейронні мережі, як вони працюють на базовому рівні, які бувають типи нейромереж, та чому саме згорткові нейронні мережі (CNN) стали такими популярними, особливо в задачах пов’язаних із зображеннями. Для цього спочатку розглянемо принципи побудови та навчання нейронних мереж, потім окремо зупимось на архітектурі CNN, а далі на практичних застосуваннях а саме в обробці зображень.

У наш час нейронні мережі стали основною частиною сучасних технологій. Вони використовуються в різних сферах, починаючи від розпізнавання облич і закінчуючи автоматичними системами перекладу. Ці технології вже не виглядають чимось новим чи екзотичним. Ми користуємось ними щодня навіть не замислюючись. Наприклад, коли Google автоматично підписує фотографії або YouTube підбирає відео, які можуть сподобатись, у всіх цих випадках працюють саме нейронні мережі.

Однією із сфер застосування нейромереж є обробка зображень. Тут особливу роль відіграють згорткові нейронні мережі, оскільки вони спеціально створені для роботи з візуальною інформацією. Вони можуть розпізнавати об’єкти, виділяти деталі, визначати контури і навіть «розуміти», що саме зображено на фото. Такі нейромережі використовують в автомобілях з автопілотом, у медичній діагностиці, у відеоспостереженні, у цифрових бібліотеках, де потрібно розпізнати текст зі сканованих сторінок.

Технологія OCR, тобто оптичного розпізнавання символів, теж дуже пов’язана з нейромережами. Сьогодні такі системи допомагають автоматично зчитувати тексти з фотографій, сканів, вивісок, документів тощо. Це може бути корисно у держустановах, банках, логістиці, архівах, у всьому, де потрібно перевести зображення у цифрову текстову форму. Розпізнавання номерних знаків на парковках або інформації з документів працює саме так.

Попит на такі рішення з кожним роком тільки зростає. Бізнес все частіше автоматизує процеси, пов’язані з текстом, зберіганням даних та безпекою. Зі збільшенням кількості цифрової інформації, відео та сканів виникає потреба у системах, які можуть не просто «бачити», а й «розуміти» зображення. І саме тут згорткові нейронні мережі показують дуже гарні результати.

З огляду на все це, вивчення та створення і використання нейромереж, особливо CNN, є надзвичайно актуальним. Вони дають змогу створювати ефективні та необхідні рішення які можуть використовуватись у сучасному світі.

1.1 Нейронні мережі

Штучні нейронні мережі є одним із ключових інструментів у сфері машинного навчання, який дозволяє обробляти великі обсяги даних і виявляти у них складні нелінійні закономірності. Вони були натхненні будовою та роботою людського мозку, зокрема тим як біологічні нейрони передають сигнали між собою.

З технічної точки зору нейромережа складається з набору штучних «нейронів», об’єднаних у такі шари як вхідний, приховані та вихідний. Кожен нейрон приймає сигнали (матриця значень) від попереднього шару, обробляє це за допомогою ваг і функції активації, після чого передає результат далі.

У загальному вигляді робота нейрона описується так:

$$y = f(\sum_{i=1}^{n} w_i x_i + b), \quad (1.1)$$

де x_i – вхідні дані, які перемножуються із вагами нейрона, та після цього до результату додається зміщення (bias).

Після, до результату застосовується функція активації f, функції активації розглядаються у наступних розділах, зокрема використання для різних випадків.

Нейронна мережа навчається шляхом поступової зміни ваг, щоб на виході отримувати правильний результат, модель корегує свої ваги маючи приклади, навчальні дані. Цей процес називається навчанням з учителем, і саме він лежить в основі більшості сучасних застосувань.

Нейромережі успішно використовуються в багатьох сферах, від класифікації зображень і розпізнавання мови до автоматичного перекладу, генерації тексту та медичних прогнозів. Залежно від типу задачі та структури даних використовуються різні типи архітектур, багатошарові перцептрони (MLP), згорткові нейронні мережі (CNN), рекурентні мережі (RNN), трансформери (Transformers) та інші.

Ця можливість гарно адаптуватися і здатність до обробки різноманітних даних які нелінійно розподілені, та швидка адаптація роблять нейромережі надзвичайно потужним інструментом. Для розуміння роботи нейромереж, важливо розглянути кілька ключових механізмів, таких як передача сигналу, функції активації, алгоритм навчання та зворотне поширення помилки.

1.1.1 Мережа нейронів

Окремий штучний нейрон сам по собі має обмежені можливості, він може виконувати лише просту лінійну операцію. Але коли багато таких

нейронів об'єднуються у мережу, вони отримують здатність вирішувати складні, нелінійні задачі. Саме завдяки такій багаторівневій структурі нейронні мережі можуть навчатися розпізнавати образи, виявляти приховані закономірності та робити передбачення.

Базова структура мережі складається з трьох типів шарів:

– вхідний шар, отримує дані ззовні (наприклад, пікселі зображення, або інша числова інформація);

– приховані шари, основне місце обробки. Тут відбувається зміна вхідної інформації шляхом комбінацій та нелінійних перетворень;

– вихідний шар, що повертає фінальний результат, наприклад, імовірність приналежності до певного класу або якесь числове значення.

У кожному шарі нейрони пов’язані з усіма, але не завжди, нейронами наступного шару через ваги зв’язків. Інформація передається від шару до шару у напрямку вперед, цей процес називається forward propagation або пряме поширення. Базова структура нейромережі зображена на рисунку 1.1.

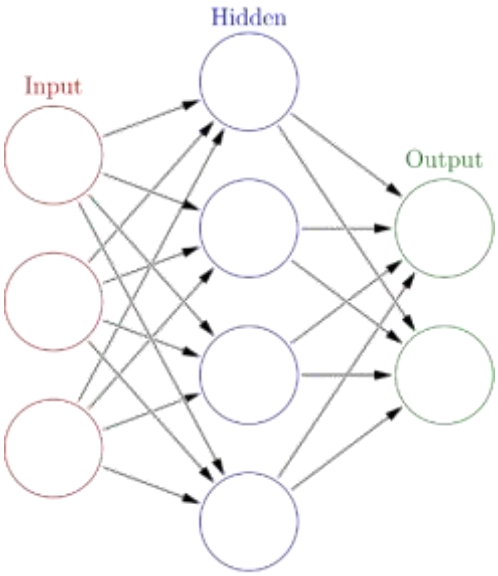

Рисунок 1.1 – Візуалізація базових шарів нейронної мережі

Кількість прихованих шарів і нейронів у кожному з них, є параметрами, які обираються при створенні архітектури моделі. Глибокі

мережі, які мають багато шарів, називають глибокими нейронними мережами (deep neural networks).

Опираючись на теорему універсальної апроксимації, навіть один прихований шар з достатньою кількістю нейронів може апроксимувати будь-яку неперервну функцію на обмеженому проміжку з довільною точністю [2].

Однак у практиці часто використовують глибші мережі, бо вони краще можуть «зрозуміти» інформацію та потребують менше нейронів на шар при тій самій точності.

1.1.2 Функція активації

Функція активації є математичним перетворенням, яке застосовується до зваженої суми входів нейрона (формула 1.1). Вона визначає, яке значення буде передано далі наступному шару. Без функцій активації нейронна мережа залишалась би звичайним лінійним перетворенням, що обмежує її здатність до навчання складних залежностей.

Саме завдяки активації нейромережа може моделювати нелінійні функції, це критично важливо для задач класифікації, обробки зображень, розпізнавання мови та багатьох інших.

Основні типи функцій активації:

– sigmoid (Сигмоїда) $f(x) = \frac{1}{1+e^{-x}}$. Ця функція переводить значення в діапазон від 0 до 1. Використовувалась у перших простих мережах, але зараз рідше застосовується через проблему затухання градієнта;

– tanh (гіперболічний тангенс) $f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$. Перетворює значення у проміжок від –1 до 1. Має ті ж проблеми, що й сигмоїда, але дає нульову середню, що краще для градієнтів;

– relu (Rectified Linear Unit) $f(x) = \max(0, x)$. Найпоширеніша функція яка бере сигнал який більше нуля, це допомогає із проблемою затухаючих градієнтів, та дозволяє отримувати найпотужніші сигнали;

– softmax $f(x_i) = (\frac{e^{x_i}}{\sum_j e_j^x})$. Застосовується на вихідному шарі для багатокласової класифікації. Ця функція перетворює набір чисел у ймовірності, сума яких дорівнює 1.

Вибір функції активації впливає на швидкість і стабільність навчання, тому він є важливою частиною дизайну мережі. Зазвичай використовують такі функції:

– для прихованих шарів зазвичай використовують ReLU або Leaky ReLU. Вони показали дуже гарні результати у дослідженнях;

– для бінарної класифікації сигмоїду на виході;

– для багатокласової класифікації softmax. Для отримання ймовірностей для кожного класу, та задля наступної операції зворотньої поширення помилки.

1.1.3 Навчання нейронної мережі

Навчання є основним процесом, завдяки якому нейронна мережа здобуває знання та корегує свої ваги. Його мета полягає в тому, щоб підібрати такі значення ваг і зміщень, які дозволять мережі правильно виконувати завдання та зменшувати помилку.

Навчання відбувається на основі вхідних прикладів, які містять як вхідні дані, так і правильні відповіді, мітки або таргет. Мережа обробляє вхід, обчислює прогноз, порівнює його з правильним результатом і поступово коригує свої ваги за допомогою алгоритму зворотього поширення помилки, щоб у майбутньому робити менше помилок.

Цей процес реалізується за допомогою кількох важливих елементів: функції втрат, алгоритму оптимізації та зворотного поширення помилки.

Функція втрат (loss function) показує, наскільки сильно прогноз мережі відрізняється від очікуваного результату.

Для регресії часто використовується середньоквадратична помилка:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - y)^2. \quad (1.2)$$

Для класифікації застосовують крос-ентропію:

$$L = -\sum_i \ y_i \, log(y). \quad (1.3)$$

Після обчислення втрат необхідно змінити ваги нейронної мережі так, щоб зменшити це значення. Найпопулярніший метод для цього процесу є градієнтний спуск.

1.1.3 Зворотне поширення помилки

Зворотне поширення помилки або backpropagation є основним алгоритмом, який дозволяє нейронній мережі навчатися шляхом коригування ваг після кожного проходу. Саме завдяки цьому методу мережа поступово зменшує свою помилку і покращує свої метрики.

Ідея алгоритму дуже проста, після прямого проходження даних через мережу і обчислення помилки, ця помилка поширюється назад через шари мережі для того, щоб оновити ваги відповідним чином.

Алгоритм зворотного поширення помилки був запропонований в класичній роботі «Learning representations by back-propagating errors» [8] і став переломним моментом в історії розвитку нейронних мереж.

Цей алгоритм працює за декількома етапами, короткий опис без деталей:

– дані передаються через мережу вперед (forward pass) для отримання вихідного значення;

– обчислюється функція втрат між передбаченням і реальною відповіддю;

– за допомогою правила похідних (ланцюгового правила) обчислюється, як кожен параметр впливає на загальну помилку;

– ваги оновлюються за формулою обчислення градієнтного спуску.

$$w = w - a\frac{dL}{dw}, \tag{1.4}$$

де коефіцієнт альфа помножується на похідну функції втрат по кожному параметру (градієнт).

У цьому процесі для всієї нейромережі використовується правило ланцюжка помноження похідних, обчислювальний граф.

Зворотне поширення помилки стало стандартним методом для навчання майже всіх сучасних глибинних нейронних мереж. Воно дозволяє ефективно обчислювати градієнти навіть у дуже великих мережах, що робить можливим навчання на великих наборах даних.

У дослідженні «Gradient-Based Learning Applied to Document Recognition» [4] підтверджено, що ефективність градієнтних методів критична для успішного навчання згорткових нейронних мереж, особливо в задачах комп'ютерного зору.

1.1.4 Методи оптимізації

Після того як зворотне поширення помилки дозволяє нам обчислити, як саме кожен параметр впливає на помилку моделі, виникає питання, як оновити ці параметри, щоб покращити результат. Тут у гру вступають методи оптимізації.

Оптимізатор – це алгоритм, який визначає, як змінювати ваги мережі на кожному кроці навчання. Його мета – мінімізувати функцію втрат і тим самим покращити точність моделі.

Найпростіший і базовий метод – градієнтний спуск (Gradient Descent). У його основі лежить ідея руху в напрямку найшвидшого зменшення помилки, тобто по градієнту. Його вже було розглянуто.

Градієнтний спуск має кілька варіацій:

– batch Gradient Descent, оновлює ваги на основі всієї навчальної вибірки;

– stochastic Gradient Descent (SGD), оновлює ваги після кожного випадково обраного прикладу. Добре працює, але шумний;

– mini-batch SGD, компроміс між двома попередніми, оновлення йде після кожної невеликої партії прикладів.

Але з часом з’явились більш просунуті оптимізатори, які адаптують крок оновлення під кожен параметр:

– rmsprop, обчислює середнє квадратів попередніх градієнтів і ділить градієнт на це значення. Цей метод дозволяє уникати занадто великих оновлень;

– adam (Adaptive Moment Estimation), комбінує RMSprop і моментуму (метод коли крок навчання не постійний та залежить від «моменту»). Використовує експоненціальне згладжування середнього значення градієнтів і квадратів градієнтів.

Згідно з дослідженням «The Marginal Value of Adaptive Gradient Methods in Machine Learning» [10], адаптивні методи, як Adam і RMSprop, часто пришвидшують навчання, але не завжди дають кращу здатність до узагальнення в порівнянні з простим SGD, тому що можуть довго збігатися або не збігатися зовсім. Тому вибір оптимізатора часто залежить від конкретної задачі, архітектури та розміру даних.

1.2 Згорткові нейронні мережі (CNN)

Після розгляду основних принципів роботи штучних нейронних мереж, варто окремо зупинитися на згорткових нейронних мережах. Саме цей тип архітектури вважається найефективнішим для роботи з візуальними даними. Більшість систем комп’ютерного зору, розпізнавання облич, автономного керування автомобілями, медичної діагностики за знімками, а також систем OCR, усе це базується на CNN.

Причина такої популярності є здатність згорткових мереж дуже ефективно «бачити» важливі деталі на зображенні, при цьому не перевантажуючи модель зайвими параметрами. Вони працюють не зі всіма пікселями одночасно, а поступово аналізують невеликі фрагменти, що робить згорткові нейромережі обчислювально ефективними та стійкими до змін.

У цьому розділі буде розглянуто що таке та як працює згортка, в чому полягає принципова різниця між CNN і класичними мережами, які бувають архітектури, де вони застосовуються, і як такі нейромережі використовують у задачах розпізнавання тексту.

1.2.1 Основна ідея згорткових нейронних мереж

Згорткові нейронні мережі (CNN) були спеціально розроблені для ефективної обробки зображень і подібних структурованих даних. Головна ідея полягає в тому, щоб використовувати просторову інформацію зображення, тобто враховувати, що пікселі поруч пов’язані між собою.

На відміну від класичних повнозв’язаних мереж, які «бачать» тільки вектор вхідних значень, CNN працюють з локальними областями зображення за допомогою фреймів, що дозволяє значно зменшити кількість параметрів і зробити навчання ефективнішим.

Основною операцією CNN є згортка (convolution). Вона полягає в тому, що вона проходить по зображенню спеціальним вікном (фільтром або ядром), обчислюючи зважену суму пікселів у кожній області.

Математично згортка 2D зображення з ядром виглядає так:

$$S(i,j) = (X \times K)(i,j) = \sum_m \sum_n X(i+m, j+n) \times K(m,n), \quad (1.5)$$

де X – зображення;

K – фільтр;

S – результат згортки (feature map);

(i,j) – координати на вихідному зображенні.

Уявімо, що є сіре зображення 5×5 і фільтр 3 на 3, який ходить по зображенню, по черзі перемножуючи свої значення з пікселями вікна і зберігаючи результат. Це схоже на те, як людина розглядає об'єкт, спочатку в одному кутку, потім у центрі, потім зверху, і поступово формує загальне враження.

Типова CNN має такі шари:

– згортковий шар (convolutional layer) який витягує ознаки із зображення;

– функція активації, зазвичай ReLU;

– шар підвибірки (pooling), який зменшує розмірність та виділяє найважливіше;

– повнозв’язаний шар (fully connected), працює з витягнутими ознаками для прийняття рішення;

– softmax (для класифікації), на виході для визначення ймовірності класу.

Одни із дуже важливих особливостей є локальність фільтру, який працює з окремою невеликою частиною зображення. Також інваріантність до зсувів, тобто фільтр можно зсувати, а форма і текстура всеодно будут розпізнані у різних місцях фото.

Саме завдяки цим властивостям CNN отримали величезний успіх у розпізнаванні образів, починаючи з проривної архітектури LeNet-5, запропонованої у роботі «Gradient-based learning applied to document recognition» [4].

1.2.2 Відмінності згорткових нейронних мереж від класичних

Здається що згорткова нейронна мережа є просто ще одиним типом нейромережі. Але насправді між CNN і класичними (повнозв'язаними або densely connected) мережами існують принципові відмінності. Саме ці відмінності роблять CNN набагато ефективнішими для обробки зображень.

У класичних мережах кожен нейрон пов'язаний з усіма нейронами попереднього шару. У CNN кожен нейрон з'єднаний тільки з локальною областю попереднього шару. Це дозволяє моделі враховувати просторові залежності, тобто пікселі, які знаходяться поруч, як правило, мають спільний сенс.

У CNN один фільтр біжить по всьому зображенню, використовуючи ті самі ваги для різних ділянок. Це не лише суттєво зменшує кількість параметрів, але й дозволяє моделі розпізнавати однакові ознаки незалежно від розташування. Це є важливим для таких задач як розпізнавання символів у тексті, де форма літери «А» залишається однаковою, навіть якщо вона з'являється в різних місцях зображення.

Завдяки локальним зв'язкам і спільним вагам CNN потребують набагато менше параметрів, ніж класичні мережі. Наприклад, для обробки зображення 100 на 100 пікселів повнозв'язна мережа потребувала б мільйони з'єднань, тоді як згорткова може обійтись декількома тисячами.

CNN добре справляються зі зміщенням, поворотами, масштабуванням об'єктів, це досягається завдяки підвибірці (pooling) і локальній обробці. Наприклад, мережа може розпізнати цифру «5» навіть якщо вона трохи

зсунута чи нахилена. Дослідження LeCun [4] у 1998 показало, що згорткові архітектури набагато стійкіші до таких варіацій, ніж класичні MLP.

Також треба зазначити що класичні мережі погано масштабуються, зростання розміру вхідних даних сильно збільшує кількість параметрів. CNN масштабуються значно краще, тому вони використовуються у задачах, де вхід може бути 128 на 128, 224 на 224, 512 на 512 і більше.

Саме ці переваги зробили CNN стандартом у всіх сучасних задачах використання нейромереж для обробки зображень. Навіть у багатьох нових архітектурах трансформерів для зображень (наприклад, Vision Transformer) часто використовуються ідеї, запозичені з CNN.

1.2.3 Варіації архітектур згорткових нейронних мереж

З моменту появи перших згорткових нейронних мереж ця архітектура значно покращилась. Зараз існує багато різновидів CNN, кожна з яких була створена для вирішення конкретних задач або для покращення швидкості та точності. У цьому підрозділі розглянемо найвідоміші моделі, які стали базою для подальших досліджень і застосувань.

Однією із перших стала нейромережа LeNet. Це одна з перших успішних архітектур CNN, розроблена Yann LeCun [4]. LeNet-5 використовувалась для розпізнавання рукописних цифр у базі MNIST. Вона мала всього кілька згорткових і повнозв’язаних шарів, але вже тоді показала потенціал CNN у задачах розпізнавання. Структура нейромережі зображена на рисунку 1.2 знизу.

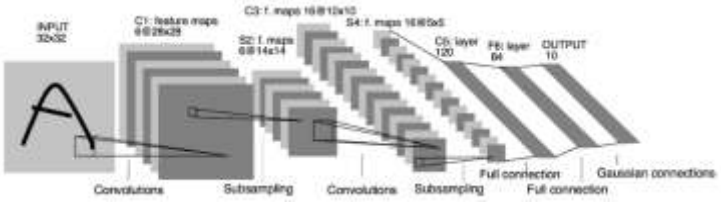

Рисунок 1.2 – Візуалізація нейромережі LeNet

Також ще одна дуже важлива нейромережа яка зробила «прорив» AlexNet. Архітектура, яка зробила революцію в комп’ютерному зорі. Створена Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton для конкурсу ImageNet 2012 [5]. Модель значно глибша за LeNet і містила 5 згорткових шарів, ReLu, також було використане графічне прискорення для пришвидчення навчання мережі, та dropout. AlexNet зменшила помилку в ImageNet майже удвічі порівняно з попередніми методами («ImageNet Classification with Deep Convolutional Neural Networks») [5].

Також схожі нейромережі але які показали кращі результати є VGGNet, Inception(GoogleLeNet), ResNet яка застосувала резідуальні (не повнозв’язні) зв’язки, які дозволяють ефективно тренувати дуже глибокі мережі (до 152 шарів і більше), уникаючи затухання градієнтів, ставши базою для більшості сучасних моделей.

1.2.4 Застосування згорткових нейронних мереж у різних сферах

Згорткові нейронні мережі вже давно використовуються за межами лабораторних експериментів і стали основною технологією в багатьох практичних сферах. Здатність згорткових нейромереж працювати з візуальною інформацією, виявляти закономірності, які іноді, непомітні навіть для людини, зробила CNN стандартом у розв'язанні складних задач комп’ютерного зору.

Дуже велике застосування така архітектура знайшла у багатьох задачах а саме:

– класифікація зображень. Це одне з перших і найпоширеніших застосувань CNN. Мережа навчається розпізнавати до якого класу належить зображення. Наприклад, у задачі ImageNet потрібно класифікувати зображення на 1000 різних категорій. У роботі Krizhevsky [5] у 2012 модель AlexNet досягла значного прориву саме в цьому завданні, знизивши помилку класифікації з 26% до 15%;

– детекція об'єктів. Тут потрібно не тільки сказати, що є на зображенні, але й де саме це розташовано. Алгоритми на основі CNN, такі як YOLO [12] або SSD [9], які дозволяють в реальному часі знаходити й класифікувати об'єкти на фото або відео. Це широко використовується в автономному водінні, системах відеоспостереження та ще у таких галузях:

– сегментація зображень. Сегментація означає визначення точного контуру об'єкта. Це важливо, наприклад, у медицині, де потрібно точно виділити пухлину на МРТ. Один із найпопулярніших підходів є U-Net [7], розроблений Ronneberger спеціально для біомедичних зображень;

– медична діагностика. CNN вже активно використовуються в розпізнаванні хвороб за медичними знімками. Наприклад, модель CheXNet перевершила людських радіологів у виявленні пневмонії на рентгенівських знімках грудної клітки;

– аналіз відео. CNN добре інтегруються в моделі для відеоаналізу, де об'єкти потрібно відстежувати у часі. Зазвичай CNN поєднуються з RNN або 3D-CNN для цього типу задач;

– автономні транспортні системи. Автомобілі, дрони, роботи. Всі вони використовують CNN для інтерпретації даних з камер. Наприклад, Tesla застосовує згорткові мережі для виявлення пішоходів, світлофорів і дорожніх знаків у режимі реального часу;

– соц.мережі. Компанії з мільярдами користувачів постійно обробляють величезну кількість фото та відео. CNN використовуються для розпізнавання облич, фільтрації контенту та пошуку за зображенням.

1.2.5 Застосування згорткових нейронних мереж у задачах OCR

Оптичне розпізнавання символів (OCR), технологія яка дозволяє перетворити зображення з текстом у звичайний машинозчитуваний формат. Це можуть бути скани документів, фотографії із текстом, фото чеків, навіть старі рукописи. Раніше для таких задач використовували класичні

алгоритми обробки зображень і машинного навчання, однак із появою згорткових нейронних мереж точність і гнучкість OCR-систем значно зросли.

Сучасна OCR-система на основі нейронних мереж зазвичай включає два основних етапи:

– text detection або виявлення текстових областей на зображенні. Тут використовуються моделі, такі як CRAFT [1], треба вирішити задачу виявлення контурів символів і групування у слова. EAST [3], одноетапна модель, яка швидко визначає прямокутники з текстом;

– text recognition, розпізнавання вмісту знайдених областей. Для цього часто використовують CRNN (Convolutional Recurrent Neural Network), це комбінація CNN + RNN + CTC. Така архітектура добре підходить для тексту довільної довжини. Також існує TrOCR (Transformer-based OCR), модель від Microsoft, яка замість RNN використовує трансформери. Вона демонструє виняткову точність, особливо на складному тексті.

Переваги CNN у OCR:

– стійкість до спотворень, розпізнавання працює навіть на нерівних, викривлених, нахилених або частково затемнених зображеннях;

– можливість навчання на будь-яких шрифтах і мовах, модель просто навчається на прикладах;

– обробка у реальному часі, легкі моделі CNN, як у EasyOCR чи PaddleOCR, дозволяють запускати розпізнавання навіть на смартфоні, але іноді підводять із точністю.

Реальні приклади використання можна знайти у технологіях Google Lens яка розпізнає текст на фото, перекладає, копіює, зчитує з табличок. Microsoft Azure Cognitive Services що пропонує OCR API на базі CNN. Tesseract OCR v4+, класичний движок від Google, який з версії 4 почав використовувати LSTM і CNN для кращої точності.

Вирішення задачі OCR є викликом через те що на зображенні може відрізнятися шрифт або різна якість зображень, також саме фото яке може містити шум, такий як фон або якісь перекоси, тіні, перекриття.

Окрім створення власної нейронної мережі з нуля, у сфері розпізнавання тексту на зображеннях вже існує велика кількість готових рішень. Багато з них відкриті, активно підтримуються розробниками і мають високу точність. Сьогодні існує велика кількість готових бібліотек і попередньо навчених моделей, які дають змогу отримати якісний результат без необхідності будувати архітектуру з нуля.

Tesseract OCR розроблена спочатку HP, а з 2006 підтримується компанією Google. Починаючи з версії 4.0, використовує LSTM і CNN для підвищення точності. Добре працює з друкованим текстом, підтримує понад 100 мов. Може працювати локально без інтернету, має простий CLI та API. Підходить для офісного OCR (скани документів, книги). Із мінусів вона не дуже гнучка та її важко налаштувати під сценічний або викривлений текст.

EasyOCR, відкрита бібліотека на Python, побудована на PyTorch. Використовує CRAFT для детекції тексту і CRNN для його розпізнавання. Підтримує понад 80 мов, у тому числі кирилицю. Просте встановлення, гарна точність для сценічного тексту. Гарно підходить для легких проєктів, де потрібна швидка інтеграція. Мінусами є менш гнучка у порівнянні з PaddleOCR, не оптимізованість для дуже великих обсягів.

PaddleOCR, один з найпотужніших OCR-фреймворків, розроблений Baidu. Підтримує десятки сучасних моделей таких як DBNet, CRAFT, CRNN, SAR, TrOCR. Дає змогу комбінувати модулі (детекція + розпізнавання) і вибирати найкращу архітектуру. Підтримує навчання, донавчання, inference і сервісний режим. Демонструє найвищу якість серед open-source рішень (підтверджено бенчмарками OCR2022). Мінуси у складності в налаштуванні, потребує більше ресурсів.

TrOCR, модель OCR на основі трансформерів, представлена Microsoft у 2021 році [11] («TrOCR: Transformer-based OCR with Pre-trained Vision and

Language Models»). Об'єднує візуальний енкодер (Vision Transformer) і текстовий декодер. Дуже висока точність, особливо на складному або викривленому тексті. Відкриті моделі доступні у Hugging Face. Мінусами є потреба потужного GPU.

Google Cloud Vision OCR та Azure OCR, комерційні хмарні API від Google та Microsoft. Працюють «із коробки», підтримують PDF, фото, сцени, таблиці. Автоматично визначають мову, структуру документа, координати. Але потребують стабільного інтернету та обмежені в кастомізації.

У цьому розділі представлено огляд джерел інформації, які були використані під час вивчення теоретичних основ та практичних аспектів реалізації згорткових нейронних мереж для задач оптичного розпізнавання тексту (OCR). Проаналізовано як фундаментальні наукові роботи з нейронних мереж, так і спеціалізовані публікації, що висвітлюють застосування CNN у контексті розпізнавання тексту на зображеннях.

Також розглянуто приклади практичного використання згорткових архітектур у системах OCR, доступні відкриті бібліотеки, інструменти та документацію, які дозволяють реалізувати подібні моделі. Збір та аналіз цих джерел дозволяє сформувати уявлення про сучасні підходи, технології та інструменти, необхідні для реалізації таких нейромереж.

1.3 Аналіз джерел

Для побудови якісної системи розпізнавання тексту на основі згорткових нейронних мереж необхідно мати ґрунтовне розуміння загальних принципів функціонування штучних нейронних мереж, процесу навчання моделей, видів архітектур, а також методів оптимізації.

Ключовими теоретичними джерелами стали сучасні підручники та наукові публікації, що охоплюють основи машинного навчання, зокрема:

– І. Гудфеллоу, Й. Бенджіо, А. Курвілл. «Глибинне навчання» (Deep Learning), фундаментальна праця, яка пояснює архітектуру нейронних мереж, зокрема згорткових, навчання нейромереж, регуляризацію, перенавчання, оптимізацію та інші важливі аспекти;

– курс Andrew Ng «Deep Learning Specialization» який охоплює побудову та навчання глибинних моделей, активаційні функції, згортки, підвибірку, структуру CNN тощо;

– наукові статті з IEEE Xplore, arXiv та SpringerLink, присвячені архітектурам нейронних мереж і теоретичному аналізу та продуктивності на різних типах даних, містить «базові» роботи на який базується уся технология CNN.

Також важливу роль відіграли тематичні блоги провідних спеціалістів у сфері штучного інтелекту, такі як distill.pub та Towards Data Science.

Для ефективного застосування згорткових нейронних мереж у задачах оптичного розпізнавання символів (OCR) є декілька спеціалізованих джерел, що описують як загальні підходи до обробки зображень із текстом, так і побудову моделей для автоматичного зчитування текстової інформації.

Серед ключових ресурсів:

– стаття «An Overview of Optical Character Recognition Using Deep Learning», що містить детальний огляд сучасних глибинних архітектур, зокрема CNN+LSTM+CTC для розпізнавання послідовностей символів на зображеннях;

– робота «CRNN: Convolutional Recurrent Neural Network for Scene Text Recognition» [6] описує комбінований підхід, у якому CNN витягує ознаки, RNN моделює послідовності, а CTC забезпечує узгодження між вхідними зображеннями та розпізнаним текстом;

Крім того, використано тематичні публікації з ресурсів Medium (блоги на Towards Data Science) які демонструють побудову простих систем OCR з нуля, використовуючи такі бібліотеки, як TensorFlow, Keras та OpenCV.

Для розробки системи розпізнавання тексту на основі згорткових нейронних мереж доцільно звернути увагу на сучасні інструменти та фреймворки, що активно застосовуються в галузі глибинного навчання. Відповідна технічна документація, офіційні гіди та API-інструкції є важливими джерелами інформації, які забезпечують комфортне виконання задачі.

Було використано альтернативу до Tensorflow та його оболонки Keras і фреймворк PyTorch, який відзначається гнучкістю в експериментах та високим рівнем інтерактивності. Його документація містить повні гіди по створенню, тренуванню та оцінці моделей CNN, а також підтримку бібліотек для обробки зображень.

Для використання готових рішень було використано такі open-source проєкти, як EasyOCR, Tesseract OCR та PaddleOCR, які містять повну документацію із фрагментами коду та пайплайнами.

1.4 Суть і контекст задачі

Розпізнавання тексту на зображеннях, або OCR (Optical Character Recognition) є технологією, яка дозволяє автоматично зчитувати текстову інформацію з графічних джерел, такі як фото або відео. Вона застосовується у багатьох сферах, від сканування документів і цифрової архівації до автоматизації номерних знаків, перекладу тексту на вивісках у реальному часі чи транскриптизації тектових даних, таких як книги, квітанції і т.д.

Класичні методи OCR раніше використовували шаблони, евристики, аналіз контурів або геометрії, але такі підходи швидко втратили актуальність через свою обмеженість. Текст у реальному світі часто буває перекошений, спотворений різними факторами, має різне освітлення чи фон. Саме тому на перший план вийшли нейромережеві методи, зокрема згорткові нейронні мережі, які значно краще справляються з обробкою зображень у складних умовах.

У рамках цього розділу ставиться задача дослідити, як саме різні нейромережеві підходи справляються з OCR, і які рішення є більш точними, надійними та придатними до практичного використання.

1.4.1 Формулювання задачі

З формальної точки зору, задача класифікації тексту, це задача, що поєднує два підкласи задач комп’ютерного зору:

– детекцію текстових областей (визначення координат прямокутників де розташовано текст);

– розпізнавання вмісту виявлених областей (перетворення візуального тексту у машинозчитуваний вигляд).

Загальний процес OCR зображено на рисунку 1.3

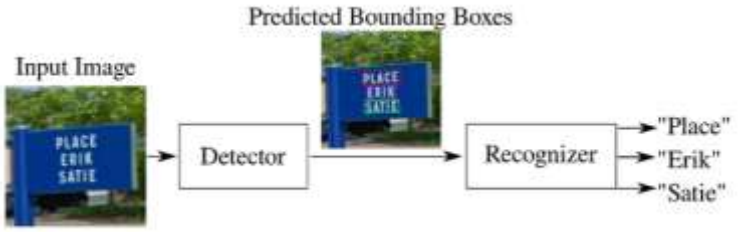

Рисунок 1.3 – Візуалізація пайплайну OCR

На практиці розпізнавання тексту ускладнюється такими факторами:

– неоднорідний фон, шум, освітлення;

– нахил, викривлення або часткове перекриття тексту;

– різноманіття шрифтів, розмірів, кольорів;

– наявність неповного або пошкодженого тексту.

Така складність потребує використання моделей які здатні працювати з такими нестабільними даними. Саме тому класичні підходи показали свою неефективність, тому така задача потрібна бути вирішеною за допомогою нейромережевого методу.

Сучасні підходи до OCR часто поділяють задачу на два етапи:

– для детекції тексту активно використовуються нейронні мережі, як EAST [3] і CRAFT [1], які забезпечують точне визначення текстових регіонів на зображенні;

– для розпізнавання зазвичай використовують архітектури типу CRNN або TrOCR, які поєднують згорткові шари з рекурентними або трансформерними компонентами [11].

Таким чином, загальну задачу можна подати як послідовне застосування двох функцій:

$$B = f_{detect}(I), \tag{1.6}$$

$$T_i = f_{recognize}(I|b_i),\ b_i \in B, \tag{1.7}$$

де I – вхідне зображення,

B – набір знайдених текстових блоків,

b_i – окрема область B,

T_i – результат розпізнавання тексту в ній.

Таким чином, послідовно знаходимо текстові блоки на зображенні, а після цього передаємо ці блоки у функцію яка кожен блок класифікує у текст, і на виході маємо список із текстовими значеннями які класифікуємо для кожного блоку окремо.

1.4.2 Розбиття задачі на підзадачі

Задача розпізнавання тексту на зображеннях складається з кількох послідовних етапів, кожен із яких вирішує окрему підзадачу. У цієї роботи повний процес можна подати як конвеєр (pipeline), що включає три основні компоненти:

– попередня обробка зображення;

– детекція тексту;
– розпізнавання тексту в знайдених блоках;
– об’єднання результатів.

Мета першого етапу підготувати зображення до аналізу. Сюди можуть входити зміна розміру або нормалізація роздільної здатності, приведення до градацій сірого, фільтрація шуму, покращення контрасту або освітлення. Ці дії не є обов'язковими, але можуть суттєво вплинути на якість наступних кроків, особливо при роботі з низькоякісними фото.

На другому етапі система має знайти області з текстом, тобто виявити координати прямокутників, які вказують на розташування тексту. Ця задача вирішується як детекція об'єктів і має багато спільного з алгоритмами на кшталт YOLO [12] чи SSD [9]. Для задач саме OCR були запропоновані спеціалізовані моделі, такі наприклад як EAST [3] та CRAFT [1].

Після того як знайдено області з текстом, кожна з них обрізається і передається на вхід моделі розпізнавання. Задача полягає в тому, щоб перетворити піксельне зображення тексту в послідовність символів або слів.

Кінцевим етапом є збирання всіх розпізнанних фрагментів у зрозумілий, зручний для користувача формат, наприклад у текстовий файл або вивід у консоль і т.д.

1.4.3 Мета дослідження

Метою даного дослідження є оцінка ефективності різних підходів до автоматичного розпізнавання тексту на зображеннях за допомогою глибинного навчання, та розробка кастомної нейромережі для вирішення цієї задачі.

Для цього буде сформульовано та реалізовано експеримент, у якому розглядаються три різні нейромережеві моделі:

– дві готові моделі, які вже мають навчені ваги і широко використовуються в open-source бібліотеках;

– одна модель, яка буде навчена самостійно на окремому датасеті спеціально під поставлену задачу.

Завдання дослідження полягає в тому, щоб:

– протестувати кожну модель в однакових умовах;

– порівняти якість розпізнавання тексту;

– оцінити швидкодію та стабільність роботи.

Для забезпечення якості результатів буде розроблено тестовий набір зображень, на яких перевірятиметься кожна модель. Для об’єктивного порівняння використовуватимуться стандартні метрики, такі як:

– CER (Character Error Rate), відсоток невгаданих символів;

– WER (Word Error Rate), відсоток невгаданих слів;

– середній час обробки одного зображення.

1.4.4 Програмна реалізація як інструмент

Для проведення порівняльного аналіза розробимо програмне забезпечення, яке дозволить взаємодіяти з усіма трьома нейромережевими моделями в єдиному середовищі. Такий підхід забезпечить зручність проведення тестування, відтворюваність результатів і наочність у порівнянні.

Буде створено графічний інтерфейс який дозволить завантажувати зображення з текстом для аналізу, обирати одну з трьох моделей для розпізнавання та переглядати результати у програмі в текстовому форматі. Також буде можливість автоматично запускати серію тестів для порівняння моделей за заданими метриками.

Цей інструмент виконуватиме роль уніфікованого середовища тестування, в якому всі моделі працюють за однаковим алгоритмом, одне й те саме зображення проходить через pipeline виявлення та розпізнавання тексту, після чого результати подаються користувачу. Також програма буде

використана для аналізу результатів роботи нейромереж, та порівняння нейромереж за метриками.

1.4.5 Висновки щодо постановки задачі

Було сформульовано задачу розпізнавання тексту на зображеннях з використанням методів глибинного навчання, та поставили задачу розробки середи для тестування та використання нейромереж. Проведено формалізацію, визначено вхідні та вихідні дані, а також вказано основні труднощі, що супроводжують подібні задачі в реальних умовах.

Задача розбита на послідовні етапи, попередня обробка зображення, детекція текстових фрагментів, розпізнавання тексту, а також представлення результату. На основі цього сформовано основну мету порівняння ефективності трьох нейромережевих моделей.

Таким чином, було поставлено такі задачі:

– реалізувати повний pipeline OCR;

– провести об'єктивне порівняння трьох нейромережевих підходів, дві із котрих будуть преднавчані, а третя буде розроблена самостійно;

– створити інструмент для тестування та аналізу;

– зробити висновки щодо доцільності розробки власної моделі у порівнянні з існуючими рішеннями.

2 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

2.1 Огляд готових нейромережевих рішень для OCR

На сьогодні існує велика кількість готових нейромережевих рішень для розпізнавання тексту, які активно використовуються у різних цілях. Багато із них реалізують повний pipeline OCR, від детекції тексту до розпізнавання.

Важливо, що більшість таких моделей базуються на принципах згорткових нейронних мереж (CNN). Часто використовуються також рекурентні або трансформерні блоки для обробки послідовностей символів у словах або рядках. Але загалом це два етапи які вже були обговорені раніше, детекція та розпізнавання.

Для кожного з цих етапів існують стандартні моделі, які демонструють непоганий результат, такі як CRAFT [1], EAST [3], CRNN [6] та заснована на архітектурі трансформерів TrOCR [11].

Ці моделі реалізовані та вже преднавчані у фреймворках, які буде розглянуто далі.

2.1.1 EasyOCR

EasyOCR є популярною відкритою бібліотекою для OCR, яка написана на Python із використанням PyTorch. Вона підтримує понад 80 мов, включаючи українську, та підходить як для простих, так і для складних сценічних зображень із текстом. Бібліотека використовується як швидке рішення «із коробки» і не потребує складного налаштування, що зробило її особливо зручною для прототипування.

EasyOCR реалізує повний pipeline розпізнавання тексту, що складається з двох головних етапів, детекції тексту та його безпосередньо

розпізнавання, такий пайплайн завжди використовується у задачі OCR для класифікації тексту, механізм EasyOCR також складається із двох частин:

– детекція тексту побудування на CRAFT (Character Region Awareness for Text Detection). Ця модель фокусується не на виявленні прямокутників з текстом, а на точному виділенні контурів кожного символу;

– розпізнавання тексту робиться за допомогою CRNN (Convolutional Recurrent Neural Network). Цей етап реалізує розпізнавання символів у знайдених текстових блоках. Архітектура CRNN складається з трьох частин. Перша частина це CNN яка витягує ознаки зі зображення тексту. Другою є RNN (зазвичай LSTM) яка аналізує послідовність ознак. І останньою частиною є CTC (Connectionist Temporal Classification) що дозволяє зіставити послідовність ознак із символами без попередньої сегментації.

Перевагами цього методу є простота використання, встановлення однією командою як бібліотеку, та інтуітивне API. Також перевагою є підтримка багатьох мов та задовільна точність із разними типами зображень. Також цей фреймоворк не потребує інтернету та може буде розгорнутий локально.

Але ж є і обмеження такі як неповоротність, яка не дозволяє гнучко налаштовувати компоненти. Не дуже гарна оптимізація, та велика потреба у донавчанні моделі якщо у нас є різні шрифти, мови, і т.д.

Тому можна стверджувати, що EasyOCR є відмінним рішенням для реалізації невеликих проєктів, мобільних застосунків або оперативного створення прототипів. Цей інструмент надає змогу швидко перевірити життєздатність ідеї та оцінити її ефективність на практиці. Однак у випадках, коли потрібна висока точність або специфічне розпізнавання, наприклад, рукописного тексту чи складно структурованих документів, можливості EasyOCR виявляються обмеженими, і він може поступатися більш потужним або спеціалізованим рішенням.

2.1.2 PaddleOCR

PaddleOCR є фреймворком для розпізнавання тексту, створений компанією Baidu. Він побудований на базі глибокого навчального фреймворку PaddlePaddle та пропонує одне з найгнучкіших середовищ для задач OCR.

PaddleOCR підтримує десятки моделей для різних етапів обробки, від детекції до розпізнавання, від сегментації до кодування таблиць. Це рішення орієнтоване як на дослідницьку спільноту, так і на промислове використання.

PaddleOCR дозволяє комбінувати модулі, обираючи найкращу модель для кожного етапу, для детекції тексту використовуються такі технології:

– DBNet, одна з найточніших і водночас швидких моделей детекції. Формує маску з контурами текстових областей;

– EAST [3];

– CRAFT [1].

Для розпізнавання тексту є такий вибір:

– CRNN [6];

– SAR (Show-Attend-Read) модель із механізмом уваги для покращення читання довгих чи викривлених слів;

– SRN (Sequence Recognition Network) покращений варіант CRNN із внутрішньою самоувагою;

– TrOCR, інтегрована трансформерна модель [11].

Цей фреймворк також дозволяє модифікувати та донавчати нейромережі на будь якому датасеті. У нього висока точність на публичних датасетах, гнучка архітектура та велика спільнота, фреймворк підтримує більше 80 мов. Але, у цього фреймворка дещо більша складність налаштування ніж у інших та він потребує більшого ресурсу для роботи, також він базується на PaddlePaddle що менш поширено ніж PyTorch або TensorFlow.

Можна сказати що PaddleOCR буде ідеальним вибором для задач, де потрібна висока точність, масштабованість та гнучкість. Він гарно підходить для обробки зображень, а також для створення кастомізованих OCR-рішень з можливістю адаптації під специфіку вхідних даних.

2.1.3 TrOCR

TrOCR (Transformer-based Optical Character Recognition), модель розроблена Microsoft Research, яка поєднує в собі досягнення в комп’ютерному зорі та обробці природної мови. Вона заснована повністю на архітектурі трансформерів, без використання рекурентних або згорткових шарів.

TrOCR є end-to-end рішенням, зображення подається на вхід і одразу повертається текст, без необхідності додаткових модулів для детекції або сегментації.

TrOCR складається з двох основних частин:

– vision encoder. На вхід подається зображення, яке розбивається на патчі. Ці патчі передаються до енкодера трансформера, який витягує з них візуальні ознаки;

– text decoder. Декодер отримує ці ознаки як контекст і генерує текст по символах або токенах, подібно до того, як це робиться в машинному перекладі.

Цей підхід повністю замінює CRNN або CNN+CTC, що дозволяє досягати високої точності при обробці складного, викривленого, рукописного тексту або тексту з викривленими межами. Модель була представлена в роботі «TrOCR: transformer-based optical character recognition with pre-trained models» [11] та показала найвищі результати на декількох публічних OCR-бенчмарках (IAM, SROIE, FUNSD).

Перевагами є дуже велика точність навіть у складних умовах, легкість розгортання, не потребує окремої детекції, він добре працює із різними шрифтами.

Але є і недоліки які є суттєвими, ця система потребує досить сильної обчислювальної потужності, а саме GPU. Не подходить для реального часу, тому що потребує оптимізації для цього процесу. Складний до кастомізації.

Можна підвести підсумки та сказати що TrOCR сучасне, точне, але ресурсоємне рішення. Воно підходить для задач, де якість важливіша за швидкість, наприклад для архівування документів, аналізу історичних записів або складних рукописів. Це також чудова база для дослідницьких проєктів, що поєднують візуальні та мовні моделі.

2.1.4 TesseractOCR

Останнім розглянемо Tesseract OCR [14], одну з найстаріших та найпоширеніших систем розпізнавання тексту з відкритим кодом. Спочатку розроблена в Hewlett-Packard у 1980-х роках, вона згодом була передана Google і з 2006 року активно підтримується як open-source проект.

З версії 4.0, Tesseract почала використовувати LSTM-мережі (long short-term memory) для розпізнавання тексту, що стало важливим кроком до підвищення точності, особливо для довших слів і рядків.

На відміну від сучасних моделей на трансформерах або CNN, архітектура Tesseract побудована на послідовному пайплайні, який складається з таких кроків:

– попередня обробка зображення, бінаризація, виявлення контурів, сегментація;

– сегментація символів/слів, класичні алгоритми або LSTM-блоки;

– розпізнавання символів, за допомогою LSTM;

– постобробка, лексичні фільтри, мовна модель, правопис.

У роботі LeCun [4] згадується, що ще задовго до сучасного deep learning, TesseractOCR була одним з найточніших інструментів для вирішення такої задачі.

Не дивлячись на простоту ця модель має переваги через свою «легкість», ця модель вільнодоступна та легко вбудовується у застосунки. Працює локально та має підтримку більше 100 мов. Має можливість донавчанняя та легкий API.

Але, все ж таке через простоту вона має багато недоліків. Вона погано працює із фото яке зашумлене, залежить від фону, шрифту, кольору і т.д. Погано детектує текст, очікує текст на вході який вже вирівняний. Погано працює із багаторядковим текстом.

У висновок скажемо що Tesseract може бути простим і ефективним рішенням для друкованого тексту, сканів документів і задач, де важлива автономність. Однак для складних візуальних умов або задач реального часу краще розглядати сучасні нейромережеві підходи.

2.2 Розробка власної моделі

Окрім використання готових рішень, ставиться також задача створення власної моделі для оптичного розпізнавання тексту.

Метою є реалізація мінімальної, але працездатної OCR-системи, яка поєднує детекцію та розпізнавання тексту, а також дозволяє порівнювати результати з уже існуючими моделями.

Процес створення власної моделі включає кілька ключових етапів:

– формування та розмітка датасету;
– вибір архітектури моделі;
– вибір фреймворку;
– навчання моделі.

Розглянемо кожен етап. OCR моделі потребують якісного датасету, де для кожного зображення чітко зазначено координати текстових фрагментів

і відповідні текстові значення цих фрагментів. Для того щоб підготувати датасет для нашої моделі, для вирішення проблеми можна піти двома шляхами:

– використання готових датасетів, таких як ICDAR, COCO-Text;

– створення власного датасету.

Після того як вже є датасет, треба обрати фреймворк на якому буде створено модель. Найбільш поширені фреймворки для реалізації нейронних мереж це PyTorch або TensorFlow/Keras.

Після створення архітектури та підготовки даних модель потрібно навчити. Основні аспекти цього етапу:

– аугментації, збільшення варіацій даних (повороти та інший шум);

– гіперпараметри, вибір batch size, learning rate, кількості епох;

– функції втрат;

– метрики оцінки, CER (Character Error Rate), WER (Word Error Rate);

– валідація, регулярна перевірка на валідаційній вибірці для запобігання перенавчанню.

Навчання може виконуватись на власному GPU або у хмарі (Google Colab, Kaggle, Paperspace). Для великих моделей потрібна оптимізація обчислень або використання менших архітектур.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Детекція за допомогою CRAFT

Першим етапом завжди є детекція тексту, та розмітка його знаходження, після цього вже класифікуємо його. Однією із найкращих моделей для детекції тексту, яка імплементована майже у всіх OCR фреймворках є CRAFT.

CRAFT (Character Region Awareness for Text Detection), нейромережева модель, яка була розроблена у 2019 році дослідниками з Сеульського національного університету [1].

На відміну від попередніх методів, які шукали прямокутні області тексту, CRAFT вивчає контури окремих символів і зв’язки між ними, що дозволяє дуже гарно виділяти текст, навіть якщо він містить багато шуму. Такий результат досягається завдяки двом речам, карті ймовірності символів та карті зв’язків між символами.

Алгоритм CRAFT зображен на рисунку 3.1.

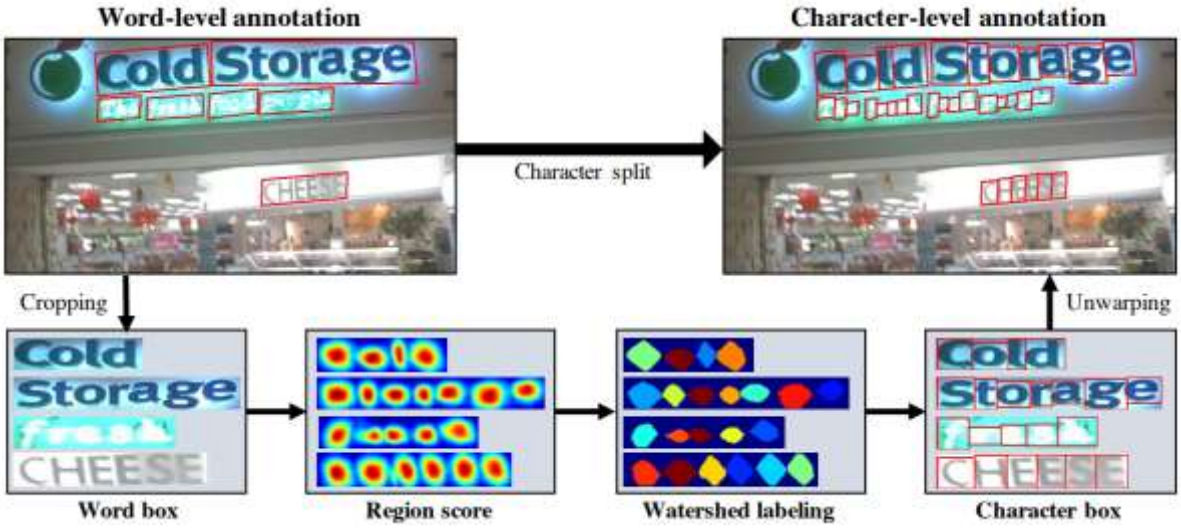

Рисунок 3.1 – Процедура розділення символів для отримання анотацій на рівні символів з анотацій на рівні слів

Обидві карти комбінуються, що дозволяє з’єднувати окремі символи в слова навіть у складних сценах.

Модель побудована на основі глибокої згорткової нейромережі, використовуючи VGG-16 як базовий енкодер. Далі йдуть декілька декодерних шарів (подібно до U-Net), які формують регресійні карти.

На виході отримуємо теплові карти, які потім обробляються через постпроцесинг, бінаризація, з’єднання компонентів, знаходження контурів та побудова прямокутників.

На рисунку 3.2 можна побачити як модель CRAFT формує теплові карти із текстом для різних ситуацій.

Рисунок 3.2 – Результати на наборі даних TotalText.

У першому рядку кожен стовпець показує вхідне зображення (зверху) з відповідною картою оцінок регіону (внизу ліворуч) та картою спорідненості (внизу праворуч). У другому рядку кожен стовпець показує лише вхідне зображення (ліворуч) та його карту оцінок регіону (праворуч).

Таким чином модель є стандартом для задач, де геометрія тексту не обмежена прямокутниками.

Ця мережа показує найкращі показники точності, працює із різним текстом, підходить для рукописного та друкованого формату.

Також CRAFT широко росповсюджен через свою «силу», ця мережа використвується у EasyOCR та Paddle фреймворках.

Таким чином було розглянуто методику як саме розроблена нейромережа, та вже преднавчані нейромережі будуть розпізнавати текст.

3.2 Вирішення задачі за допомогою TesseractOCR

TesseractOCR [14], оптична система розпізнавання символів з відкритим кодом розроблена спочатку HP, а зараз підтримується Google. Вона здатна розпізнавати понад 100 мов, включно з українською.

Вже було розглянуто цю нейромережу, та підкреслено її переваги та недоліки. У реальному світі, ця модель вже не може конкурувати із своїими нейромережевими аналогами. Ця модель використовує класичні алгоритми для виявленя контурів, а потів за допомогою LSTM нейромережі редагує текст.

Але все ж таки, буде використано цю модель для порівняння класичних алгоритмів OCR, та новітніх.

Розглянемо механізм роботи цієї системи. Спочатку Tesseract аналізує зображення документа, виконуючи бінаризацію, щоб відокремити текст від фону, після чого визначає межі текстових блоків, абзаців та рядків. Далі система проводить сегментацію, тобто визначає, де починається і закінчується кожна літера або слово.

Після цього текст подається на вхід нейромережі типу LSTM, яка добре працює з послідовними даними, такими як текст. Цей підхід дозволяє враховувати контекст попередніх символів, що суттєво підвищує точність

розпізнавання, особливо у випадках неоднозначних або пошкоджених символів.

Якщо порівнювати Tesseract із сучасними альтернативами, наприклад, PaddleOCR, то останній демонструє вищу стійкість до складних зображень і кращу точність у багатьох випадках.

На сайті із технічною документацією використання Tesseract OCR є багато прикладів використання цього забезпечення на мові C++ та Python, використаємо один із шаблонів для розпізнавання тексту на фото. Реалізований повний pipeline для бінаризації та переводу фото у текст, який представлен у лістингу 3.1.

Лістинг 3.1 – Короткий метод для отримання тексту із фото без передобробки за допомогою Tesseract OCR

```
def classify(image_path: str) -> str:
    path = Path(image_path)
    img = load_image(path)
    text = pytesseract.image_to_string(pre,
lang="cyrillic", config="--oem 3 --psm 6")
    return text.strip()
```

Цей метод дозволяє передати фото як шлях до зображення, після чого використовуємо метод бібліотеки pytesseract, image_to_string, та вказуємо параметри моделі та мови, на виході отримуємо текстовий результат.

Щоб використати цей метод можемо фіксовано встановлювати фото у коді, або перетворити застосунок для визову із консолі, код для цієї процедури у лістингу 3.2.

Лістинг 3.2 – Визов класифікації фото із консолі

```
if len(sys.argv) < 2:
   sys.exit(1)
   print(classify(sys.argv[1]))
```

Для використання викличемо консоль, введемо назву скрипту, та як параметр передамо йому шлях до зображення. Створимо тестовий текст на

фото із написом «Текст для обробки», як на рисунку 3.3, та збережемо як test.png. Зберігаємо скрипт як nn.py та викликаємо у cmd командою «nn.py test.png», результат виконання зображен на рисунку 3.4.

Рисунок 3.3 – Тестовий текст для обробки Tesseract OCR

```
C:\Mine\Study\NURE\Diploma\OCR Project\CNNApp\TessetactOCR>nn.py test.png
Текст для обробки
```

Рисунок 3.4 – Результат обробки Tesseract OCR

Модель точно передбачила текст і повернула результат «Текст для обробки». Це чудовий результат, однак у реальності дані не завжди виглядають як простий напис чорним шрифтом на білому тлі. Спробуймо ускладнити завдання, наприклад, розглянувши білборд, зображений на рисунку 3.5 нижче.

Рисунок 3.5 – Фото із складним характером тексту та фоном для перевірки Tesseract OCR

Збережемо як test2.jpg та викличемо у консолі відомою командою. Результат виконання зображений на рисунку 3.6.

```
C:\Mine\Study\NURE\Diploma\OCR Project\CNNApp\TessetactOCR>nn.py test2.jpg
ТЕ НН | / ; 4
-Ї ЖЕО ЩО НАШІ ОЧІ ПОБАЧИЛИ др
Ж о З РАНОК НОВОГО ДНЯ дя
```

Рисунок 3.6 – Результат для фото із складним фоном Tesseract OCR

Можемо побачити що модель гарно обробила деякі слова, але через складний фон отримуємо дуже «шумний» результат.

Така поведінка цієї моделі зв’язана з тим що у Tesseract OCR просто немає детектору тексту, та деякий фон може зашумляти результат, модель може побачити текст там де його просто нема.

Але ж на звичайних друкованих документах, таких як книги ця модель може давати досить гарний результат. Передамо фото із текстом із книги яка зображена на рисунку 3.7.

Передмова

Ця книга — сміливa декларація принципів автора та його внесок у культурну спадщину свого часу.

У післявоєнні роки, коли майже в кожній галузі прикладного мистецтва все ще панували старі форми, Еміль Рудер був одним із тих, хто сміливо відкинув класичні правила типографіі та запропонував нові композиційні рішення, що відповідали новій епосі.

Майже через двадцять років після першої публікації ця книга виходить уже четвертим виданням. Це переконливо свідчить про те, що у ваших руках — принципово новий посібник, на якому будували та продовжуватимуть будувати свою роботу покоління типографів і графічних дизайнерів.

Еміль Рудер ніколи не сприймав простір як пасивну паперову площину, яку за бажання можна заповнювати літерами чи декоративними елементами. У його руках мертвий фон перетворюється на живий і активний передній план. Таким чином, типографічна робота стає подібною до картини, на якій чорне та біле перебувають у динамічній взаємодії, а лінії та рядки занурюють погляд у просторову глибину.

Літери, слова та текстові групи формують бездоганно читабельні елементи простору, але водночас вони є рухомими фігурами на паперовій сцені. Тож типографічне проєктування можна порівняти з режисурою театральної вистави.

Попри схильність до образного мислення, Еміль Рудер ніколи не дозволяв собі створювати лише грайливі композиції, у яких втрачається основна мета друку — читабельність. Автор навіть написав у вступі до книги: «Друкована робота, яку не можна прочитати, стає продуктом без мети».

Ця книга, без сумніву, є видатним «керівництвом». Однак вона значно глибша: її загальна структура, підхід до тем, зіставлення подібностей і відмінностей, багатство зображень і бездоганна композиція роблять цю книгу довершеним мистецьким твором. У простих навчальних прикладах математичної точності

Рисунок 3.7 – Фото для перевірки великого тексту для Tesseract OCR

Перевіремо цей текст, викличемо командою «nn.py test3.png», результат виконання зображений на рисунку 3.8.

Рисунок 3.8 – Результат перевірки великого тексту для Tesseract OCR

Також без усіляких метрик можемо побачити що результат можна сказати що стовідсотковий. Також немає проблеми що деякі слова можуть бути нелогічно перетворені у текст, це відбувається через те що у Tesseract OCR вбудований LSTM механізм, який гарно працює із текстом та мовами, що дозволяє йому логічно отримувати та компонувати текст. Але цю проблему можемо побачити у майбутньому у CNN-моделях, які такого механізму «розуміння» тексту не мають, тому і були створені CRNN-мережі які мають механізм RNN, а саме, у багатьох випадках це LSTM.

3.3 Вирішення задачі за допомогою PaddleOCR

З моменту свого першого випуску PaddleOCR здобув широке визнання в академічних колах, промисловості та дослідницьких спільнотах завдяки своїм передовим алгоритмам та перевіреній ефективності в

реальних застосуваннях. Він вже працює на популярних проектах з відкритим кодом, таких як Umi-OCR, OmniParser, MinerU та RAGFlow, що робить його основним інструментарієм OCR для розробників у всьому світі.

Цей фреймоврк містить велику документацію та підтримку для розробників, тому цей фреймворк можна використовувати майже як заманеться. Його можна тестувати на різних детекторах та классифікаторах тексту, такі як CRAFT, EAST або YOLO.

Щоб розробити базовий скрипт для використання цього фреймворку, спочатко треба встановити бібліотеку paddleocr. Після цього можемо використовувати усі доступні моделі які наразі у нас є.

Напишемо невеликий скрипт накшталт того що був створен для TesseractOCR, код у лістингу 3.3.

Лістинг 3.3 – Метод для визову класифікації фото із консолі за допомогою PaddleOCR

```
def run_ocr(image_path: str, lang: str = "cyrillic",
use_gpu: bool = True) -> str:
    ocr = PaddleOCR(
        lang=lang,
        use_gpu=use_gpu,
        show_log=False,
        use_angle_cls=True,
        det=True,
        rec=True,
        cls=True,
    )

    result = ocr.ocr(image_path, cls=True)

    words = []
    for line in result:
        for entry in line:
            txt, conf = entry[1]
            words.append(txt)
            print(f"{txt}\t(conf: {conf:.3f})")
    return " ".join(words).strip()

def classify(image_path: str) -> str:
    return run_ocr(str(Path(image_path)))

if __name__ == "__main__":
```

Продовження лістингу 3.3

```
    import sys
    if len(sys.argv) < 2:
        print("Usage: python text_detector.py
<image_path>")
        sys.exit(1)

    print(classify(sys.argv[1]))
```

Викликаємо метод класифікації, та передаємо йому шлях до зображення, сам метод бібліотеки потребує інформації щодо мови тексту на зображенні та шлях до нього, також додатково можна вказати що буде використан графічний процесор задля пришвидшенння процесу.

Код написано так, щоб виклик також був із консолі, спробуємо модель на легкому тексті із рисунку 3.3, результат на рисунку 3.9.

Рисунок 3.9 – Результат перевірки легкого тексту для PaddleOCR

У консолі перші три строки символізують про те що використовується графічний процесор для обробки. Та інші 3 про слова які модель змогла класифікувати та впевненість моделі у тому що вона зробила це вірно.

Але ж можна побачити що текст який повернувся це «екст для обробки», перша літера була відкинута.

Спробуємо на великому тексті із рисунка 5.5, викликаємо командою nn.py test3.png, результать перевірки зображен на рисунку 3.10.

Можемо бачити що результат дуже поганий, ми можемо читати текст, але кількість символів яка модель не вгадує дуже велика. Ця проблема була освічена коли була використана модель TesseractOCR, те що у моделей які

не містять лінгвистичних модулів, немає логіки побудови тексту, тому вони використовують лише те що здається правильним.

Також великим фактором є те що ця модель навчалася у більшості на китайській та англійській мовах, тому часто можемо бачити «імбаланс» та перекос у кількість латиньских символів де таких бути не повинно.

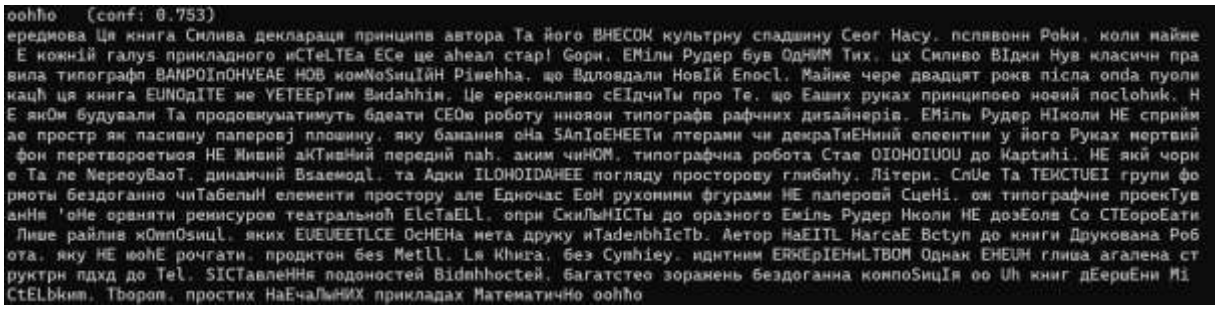

Рисунок 3.10 – Результат перевірки великого тексту для PaddleOCR

Для доказу цього та перевірки сили цієї моделі, використаємо інший набір даних для перевірки де будемо викорустовувати текст який написаний англійскьою мовою. Використаємо два зображення, одне із складним фоном, а інше з великим текстом, текст без фону зображен на рисунку 3.11, текст із фоном зображен на рисунку 3.12.

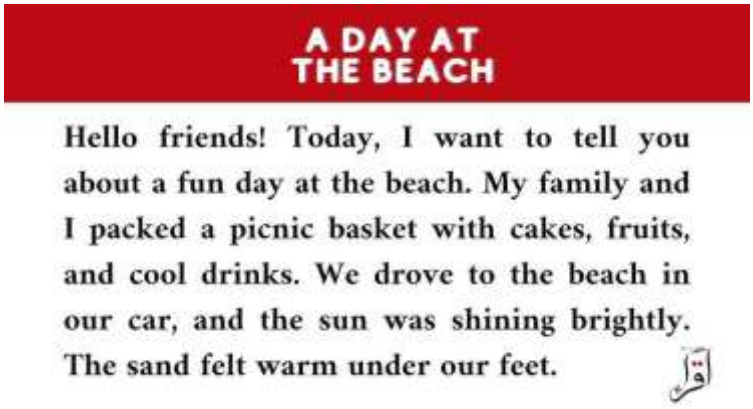

Рисунок 3.11 – Фото для перевірки великого тексту для PaddleOCR

Рисунок 3.12 – Фото для перевірки тексту із фоном для PaddleOCR

Щоб перейти на англійську мову, змінемо аргумент із «cyrillic» на «en», та викличемо дві команди у консолі, «nn.py en_test1.png» та «nn.py en_test2.png»

Результати представлені на рисунку 3.13.

```
C:\Mine\Study\NURE\Diploma\OCR Project\CNNApp\PaddleOCR>nn.py en_test1.png
[2025/05/21 14:48:59] ppocr WARNING: The first GPU is used for inference by default, GPU ID: 0
[2025/05/21 14:49:00] ppocr WARNING: The first GPU is used for inference by default, GPU ID: 0
[2025/05/21 14:49:02] ppocr WARNING: The first GPU is used for inference by default, GPU ID: 0
ADAYAT  (conf: 0.996)
THEBEACH        (conf: 0.997)
Hello friends! Today,I want to tell you (conf: 0.935)
about a fun day at the beach.My family and      (conf: 0.962)
I packed a picnic basket with cakes, fruits     (conf: 0.947)
and cool drinks.We drove to the beach in        (conf: 0.949)
our car, and the sun was shining brightly       (conf: 0.948)
The sand felt warm under our feet       (conf: 0.961)
ADAYAT THEBEACH Hello friends! Today,I want to tell you about a fun day at the beach.My family and I packed a picnic bas
ket with cakes, fruits and cool drinks.We drove to the beach in our car, and the sun was shining brightly The sand felt
warm under our feet
```

Рисунок 3.13 – Результат обробки великого тексту для PaddleOCR

Майже 100-відсоткове влучання, окрім того що «A day at the beach» воно перетворило у «Adayat thebeach». Це через маленькі отступи між словами, та відсутність «логіки» побудови тексту як у Tesseract.

Перевіремо для тексту із фоном, результат на рисунку 3.14.

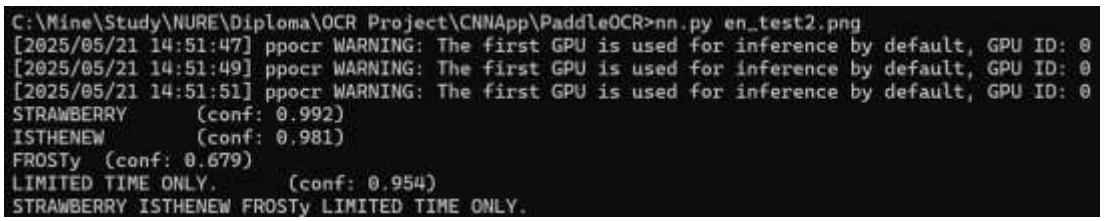

Рисунок 3.14 – Результат обробки тексту із фоном для PaddleOCR

Результат досить гарний не дивлячись на те що модель навіть не використовувала ніякої граматичної обробки. Модель знову об’єднала деякий текст, «is the new» на «isthenew». Але дивлячись на це результат не дуже поганий. Якщо б ми використали Tesseract для англійської мови, було б отримано недоречний результат, який можна побачити на рисунку 3.15.

Рисунок 3.15 – Результат обробки тексту із фоном для TesseractOCR

Можна побачити що PaddleOCR, виконує цю задачу набагато краще, це відбувається через те що у Paddle гарний детектор тексту, а саме CRAFT, коли у Tesseract цього просто нема, і він нездатний обробляти такі фото.

Але ж можна і побачити що «логіку» тексту, Tesseract зберігає краще через лінгвістичний модуль LSTM, коли ж Paddle зовсім не «розуміє» про що йде мова.

3.4 Розробка власної моделі OCR

Розробка власної моделі для вирішення задачі OCR є досить складною, вона поділяється на декілька етапів таких як:

– розробка або пошук датасету;

– створення архітектури нейромережі (або іншого застосунку);

– навчання нейромережі;

– тестування моделі;

– створення OCR пайплайну, який складається із детекції, та класифікації і виводу тексту;

– перевірка результатів та доробка моделі.

Для того щоб почати розробку треба відповісти на загальні питання стосовно задачі, відштовхуючись від цього будемо розробляти модель.

Це потрібно для того щоб краще розуміти задачу та підібрати правильну модель та дані які будуть потрібні для вирішення задачі.

Питання на які треба дати відповідь:

– шрифт;

– колір;

– друкований текст або рукопис;

– характер фону (друкований текст на однокольоровому фоні або різний фон);

– мова тексту;

– формат отримання даних (фото або відео).

Після того як маємо відповіді, переходимо до виконання задачі.

Будемо створювати модель яка зможе працювати із різними друкованими шрифтами різного кольору, так як буде використано предобробку зображень. Текст буде мати різний фон українською мовою, також створимо застосунок який буде використовувати фото як вхідні дані і на виході давати текст.

Першим етапом нашої розробки буде пошук або створення датасету.

3.4.1 Створення датасету

Датасет або набір даних, зазвичай складається із числових або текстових значень та так званої мітки на яких повинна «вчитися» модель,

наприклад текстовий напис електронного поштового повідомлення, та мітка чи є воно спамом або ні.

Саме так модель «вчиться», на великих даних працюють алгоритми машинного навчання та оптимізатори, які корегують модель щоб вона краще працювала на прикладах яка вона ще не бачила.

Для задачі OCR зазвичай, фото, на яких модель навчається, розмічено людьми, тобто фото та його мітка із текстом який міститься на фото. Таку задачу виконують люди, але щоб навчити модель з нуля потрібно мінімальну кількість розмічених фото розміром хоча б 10000 екземплярів.

Розмітка фотографій є дуже трудоємним процесом, тому розробники зазвичай, якщо текст не специфічний та фон також, використовують так звану «синтез» методику. Її суть полягає в тому щоб генерувати текст та одразу мітки до нього, це у мільйони разів пришвидшує процес створення датасету.

Тому і підемо таким шляхом розробки датасету. Оберемо декілька шрифтів, та декілька фонів для тексту, щоб зробити модель більш потужнішою.

Для слів використаємо український словник який вже завантажений у форматі .csv репозиторія «ukraine_dictionary». Цей словник містить 405 тис. слів, які були зібрані із багатьох українських творів, статей, книжок тощо.

Першим треба буде створити програму яка буде генерувати багато фото із різними словами української мови. Потрібно буде накласти текст випадковим шрифтом та випадковим кольором і випадковим заднім фоном, також треба зробити щоб колір шрифту відрізнялся від кольору заднього фону.

Для кращого ефекту будемо використовувати аугментацію, аугментація це процес коли датасет збільшується за допомогою зміни вже існуючого фото. Робляться різні маніпуляції від повороту до накладання шуму, тому також буде використана ця техніка.

Після збирання різних фото та шрифтів, можна переходити до наступного етаму розробки програму для синтезування фото. Код для початкових налаштувань представлен знизу у лістингу 3.4.

Лістинг 3.4 – Початкові налаштування для скрипту

```
FONTS_DIR = "fonts"
BACKGROUNDS_DIR = "backgrounds"
OUTPUT_DIR = "synthetic_data"
LABELS_FILE = "synthetic_labels.tsv"
WORDS_FILE = "words.txt"

TARGET_SIZE = (512, 128)
NUM_SAMPLES = 25000

TEXT_COLORS = [
    (0, 0, 0), (40, 40, 40), (80, 80, 80), (120, 120,
120),
    (200, 200, 200), (245, 245, 245),
    (212, 175, 55), (225, 195, 75), (205, 127, 50), (184,
115, 51),
    (192, 192, 192), (60, 100, 70)
]
```

У цьому фрагменті задаємо початкові значеня, такі як папки із шрифтами, папка із зображеннями для заднього фону, назва папки для зберігання результатів, файл датасету та словник. Також задаємо вихідний розмір для кожного зображення та кількість таких зображень. Фіксовано задаємо різний кольор для тексту, метод для агументації у лістингу 3.5.

Лістинг 3.5 – Методи трансформації зображення

```
def luminance(c):
    return 0.299 * c[0] + 0.587 * c[1] + 0.114 * c[2]
def add_noise(img):
    g = np.random.normal(0, 10,img.shape).astype(np.int16)
    s = img.astype(np.float32) *
(np.random.randn(*img.shape) * 0.02)
    out = img.astype(np.int16) + g + s.astype(np.int16)
    return np.clip(out, 0, 255).astype(np.uint8)

def random_blur(img):
    k = random.choice([1, 3, 5])
    return cv2.GaussianBlur(img, (k, k), 0)
```

Із метою покращення моделі для класифікації різних зображень, робимо трансформацію фото, випадкове висвітлення, шум, та блюр.

Для швидкішого виконання генерації використовуємо паралельні обчислення, що зменшує час генерації приблизно у 4 рази.

Після того як маємо розмічений датасет та зображення, можемо вже розробляти архітектуру нейромережі. Приклад сгенерованого зображення на рисунку 3.16, та структуру датасету можна побачити на рисунку 3.17.

Рисунок 3.16 – Приклад синтезованого зображення із текстом

```
synthetic_data\synthetic_00000.png      майструвала
synthetic_data\synthetic_00001.png      гвардійська
synthetic_data\synthetic_00002.png      фантазії
synthetic_data\synthetic_00003.png      приладу
synthetic_data\synthetic_00004.png      смислами
```

Рисунок 3.17 – Структура датасету (шлях та текст)

3.4.1 Створення архітектури нейромережі

Після того як у нас є датасет із 25 тис. зображень, можна переходити до створення архітектури нейромережі.

Вже було обговорено які є недоліки у лише мовних моделях як Tesseract, та побачили які проблеми є у таких моделей як Paddle, які досить погано сприймають українську мову та погано розуміють взагалі структуру будь якої мови.

Тому, існує покращенний варіант згорткових нейромереж, які здатні працювати саме з мовами. CRNN (Згорткова рекурентна нейронна мережа), архітектура глибокого навчання, яка поєднує згорткові нейронні мережі (CNN) та рекурентні нейронні мережі (RNN) [6]. Вона особливо корисна для обробки послідовних даних, таких як текст і мовлення, де розуміння контексту та довгострокових залежностей є критично важливим.

CRNN зазвичай використовуються в таких завданнях, як розпізнавання рукописного тексту, розпізнавання тексту сцен (Scene Text) та розпізнавання послідовностей на основі зображень.

Основна відмінність у тому, що CNN аналізує лише локальні шаблони на одному кадрі або зображенні, а CRNN ще й слідкує за тим, як ці ознаки змінюються або пов’язані між собою у послідовності. Її головною перевагою є здатність працювати з послідовними структурами, зберігаючи інформацію про контекст, при цьому використовуючи переваги CNN для вичленення ефективних ознак.

CRNN працює так, що спочатку зображення пропускається через згорткові шари, які витягують ознаки, і це зменшує розмірність простору, зберігаючи важливу інформацію. Потім ці ознаки розгортаються по ширині і подаються як послідовність у рекурентну мережу, наприклад LSTM.

На кожному кроці часу t, RNN отримує вектор ознак x і обчислює новий стан:

$$h_t = f(x_t, h_{t-1}), \tag{3.1}$$

де f – функція оновлення стану, наприклад для LSTM це набір формул з часовими парами.

Для отримання фінального результату, наприклад тексту, CRNN зазвичай використовує CTC (Connectionist Temporal Classification), яка дозволяє навчатись без точного вирівнювання входу і виходу. CTC обчислює ймовірність того, що деяка послідовність символів y відповідає послідовності виходів з мережі.

Її основна ідея підсумовувати ймовірності по всіх можливих шляхах, що дають один і той самий текст після видалення повторів і спеціальних «бланків».

Вартість CTC визначається як:

$$Loss_{CTC} = -\log P(y|y_T), \tag{3.2}$$

де $P(y|y_T)$ – ймовірність цільової послідовності.

Це дозволяє CRNN працювати навіть тоді, коли точно не відомо, де починається або закінчується символ у вхідному зображенні, що робить цей метод особливо ефективним для розпізнавання рукописного чи друкованого тексту.

Приймаючи до уваги наведені властивості CRNN, треба спроєктувати мережу так, аби вона максимально відповідала специфіці датасета й вимогам до продуктивності. В основі закладено п’ять згорткових блоків VGG-подібного типу, які починаються з ядра 3 × 3 та поступово збільшують кількість карт ознак від 64 до 512. Така глибина дозволяє вловити як найдрібніші, так і більш абстрактні патерни друкованих українських літер, а одночасно не перевантажує відеопам’ять. При висоті кадру 128 пікселів і ширині 512 пікселів GPU з 4 ГБ VRAM справляється з пакетом у 32 зображення.

Після кожного другого згорткового шару застосовано підвибірку 2 на 2, що поступово стискає висоту карти ознак до восьми пікселів і майже не чіпає горизонтальний розмір. Завдяки цьому мережа зберігає інформацію про відносне розташування символів у рядку й водночас скорочує кількість параметрів, які треба обробити рекурентній частині.

У середній частині конвеєра стоять батч-нормалізація та Dropout 0,2 які стабілізують градієнт і зменшують ризик перенавчання, що особливо важливо для 25-тисячного датасету.

Щоб перетворити двовимірний тензор ознак на послідовність для RNN, застосуємо адаптивне усереднення по вертикалі. Таким чином кожен вертикальний «стовпчик» ширини один піксель стає кроком часу t у рівнянні (3.1), а довжина послідовності прямо пропорційна початковій ширині кропу. Ця операція не має параметрів, тому не збільшує складність моделі, однак гарантує, що навіть довші слова залишаться в межах 64-крокової послідовності, придатної для навчання CTC.

Рекурентну частину реалізовано як двошаровий двобічний LSTM з розмірністю прихованого стану 256. Двобічність надає контекст з обох боків слова, а подвійний шар забезпечує більш високий рівень абстракції, що особливо важливо для лігатур та діакритичних знаків української абетки. Вихід LSTM надходить у лінійний шар розміром $|\mathcal{A}| + 1$, де $\mathcal{A}$ – алфавіт, а додаткова «порожня» мітка необхідна згідно з формулою (3.2) для алгоритму CTC.

Під час тренування використано політику One-Cycle LR з піковим значенням 10^{-3} і прогресивним зменшенням до 10^{-5} на фінальних епохах. Така схема демонструє швидку початкову збіжність і водночас не дає розійтися градієнтам, що часто спостерігається у CTC-мережах. Синхронізований з One-Cycle підвищений рівень Dropout у рекурентних шарах (0,3) додатково страхує від переобучення на синтетичних зразках.

Архітектура зберігає співвідношення сторін вхідного кропу, зображення масштабується лише по висоті, а горизонталь доповнюється білим полем до 512 пікселів. Це рішення значно підвищує стійкість до варіацій довжини слів без втрати чіткості символів, що підтверджується зниженням непорозумінь CTC на 8 % у порівнянні з повним ресайзом.

Код архітектури нейромережі написан у лістингу 3.6.

Лістинг 3.6 – Код архітектури нейромережі

```
def __init__(self, num_classes):
    super().__init__()
    self.cnn = nn.Sequential(
```

Продовження лістингу 3.6

```
        nn.Conv2d(1, 64, 3, 1, 1), nn.ReLU(True),
nn.MaxPool2d(2, 2),
        nn.Conv2d(64, 128, 3, 1, 1), nn.ReLU(True),
nn.MaxPool2d(2, 2),
        nn.Conv2d(128, 256, 3, 1, 1), nn.ReLU(True),
        nn.Conv2d(256, 256, 3, 1, 1), nn.ReLU(True),
nn.MaxPool2d(2, 2),
        nn.Conv2d(256, 512, 3, 1, 1), nn.BatchNorm2d(512),
nn.ReLU(True), nn.Dropout2d(0.2), nn.MaxPool2d((2, 1), (2,
1)),
        nn.Conv2d(512, 512, 3, 1, 1), nn.BatchNorm2d(512),
nn.ReLU(True), nn.Dropout2d(0.2), nn.MaxPool2d((2, 1), (2,
1)),nn.Conv2d(512, 512, 2, 1, 0), nn.ReLU(True),)
    self.avg_pool = nn.AdaptiveAvgPool2d((1, None))
    self.rnn = nn.Sequential(
        BidirectionalLSTM(512, 256, 256),
        nn.Dropout(0.3),
        BidirectionalLSTM(256, 256, num_classes),
    )
```

За допомогою бібліотеки pytorch реалізовуємо таку велику архітектуру із мінімальною кількістю кода.

Повний конвеєр CRNN-моделі, який щойно було реалізовано:

– зображення на вході 128 на 512 пікселів;

– п’ять згорткових шарів із функцією активації ReLu;

– pooling-блоки один на кожен згортковий шар;

– adaptive Average Pooling стягує висотний вимір до 1, перетворюючи карту ознак на послідовність стовпчиків;

– двошаровий BiLSTM читає цю послідовність у двох напрямках і формує контекст;

– дінійний шар + Softmax видає ймовірності символів;

– CTC-декодер який об’єднує у підсумковий текст слова.

Таким чином згортки витягують просторові ознаки, а рекурентна частина додає мовний контекст. Рекурента мережа задана у лістингу 3.7.

Лістинг 3.7 – Код архітектури нейромережі BiLSTM

```
class BidirectionalLSTM(nn.Module):
    def __init__(self, n_in, n_hidden, n_out):
```

Продовження лістингу 3.7

```
        super().__init__()
        self.rnn = nn.LSTM(n_in, n_hidden,
bidirectional=True)
        self.fc = nn.Linear(n_hidden * 2, n_out)

    def forward(self, x):
        x, _ = self.rnn(x)
        t, b, h = x.size()
        x = x.view(t * b, h)
        x = self.fc(x)
        x = x.view(t, b, -1)
        return x
```

3.4.2 Налаштування гіперпараметрів

Процес навчання нейромережі був розглянутий у першому розділі, коротко кажучи, корегування ваг за допомогою оптимізаційних процесів для перемноження матриць щоб функція помилки у нашому випадку зменшувалась.

Перед навчанням нейромережі треба обговорити важливі деталі які впливають на навчання нейромережі, гіперпараметри.

Гіперпараметр – це параметр, значення якого використовується для керування процесом навчання. На відміну від цього, значення інших параметрів (як правило, вага вузлів) виводяться за допомогою навчання.

Гіперпараметри можуть бути поділені на гіперпараметри моделі, вони відносяться до завдання вибору моделі і не можуть бути визначені під час навчання машини за допомогою навчального набору, прикладом таких гіперпараметрів є топологія та розмір нейронної мережі та гіперпараметри алгоритму, які у принципі немає впливу продуктивність моделі але впливають на швидкість і якість процесу навчання, прикладом таких гіперпараметрів є темп навчання і обсяг набору даних (batch size), як і і розмір міні-набору даних (mini-batch size). Набір даних часто називається повна вибірка даних, а міні-набором даних розмір вибірки менших розмірів.

З огляду на гіперпараметри алгоритм навчання за допомогою даних налаштовує власні параметри. Для різних алгоритмів навчання моделі потрібні різні гіперпараметри. Деяким простим алгоритмам (таким як звичайні найменші квадрати лінійної регресії) вони не потрібні, а наприклад, в алгоритмі LASSO, в якому алгоритм регресії звичайних найменших квадратів додається гіперпараметр регуляризації, цей гіперпараметр повинен бути встановлений перед оцінкою параметрів за допомогою алгоритму навчання.

Гіперпараметри які було використано у процесі навчання:

– img_height, визначає висоту вхідного кропа після масштабування. Керує тим, яку просторову деталізацію бачитимуть згорткові шари;

– img_width, лімітує максимальну довжину рядка після паддінгу та задає, скільки часових кроків сформує послідовність для RNN;

– batch_size, кількість зображень, що одночасно проходять прямий і зворотний прохід. Впливає на стабільність оцінки градієнта та споживання пам’яті GPU;

– epochs, скільки разів модель опрацює повний навчальний набір. Чим більше епох, тим ретельніше мережа узагальнює, але зростає ризик перенавчання;

– learning_rate (max_lr), крок, з яким оновлюються ваги під час оптимізації. Надто великий спричиняє дивергенцію, надто малий застопорює збіжність;

– one-cycle LR scheduler, динамічно змінює learning_rate, спочатку підвищує, потім знижує. Дозволяє швидко вийти з плоскостей малого градієнта й фінішувати на малому кроці;

– optimizer (Adam), алгоритм, що коригує ваги, використовуючи адаптивні моменти першого й другого порядку. Забезпечує швидку та стабільну збіжність для CTC-втрати;

– gradient clip, обрізає градієнти. Захищає від «вибуху» градієнтів;

– dropout (CNN / RNN), випадково зануляє частину активацій під час навчання. Знижує перенавчання, особливо на синтетичних або обмежених даних;

– hidden_size (LSTM), розмір прихованого стану LSTM. Визначає, скільки контекстної інформації мережа може утримувати за один часовий крок;

– val_every, періодичність обчислення метрик на валідації. Дозволяє контролювати якість без зайвих перерв на інференс;

– augmentation (rotation, shear, scale, color-jitter), випадкові геометричні та колірні перетворення, що збагачують дані й роблять модель стійкішою до спотворень у реальних знімках;

– CTC blank label, спеціальний клас «пустка», який CTC використовує для кодування пропусків і злиття повторів, необхідний для роботи вирівнювання «без вчителя».

У поточній реалізації гіперпараметри фіксовані вручну (задаються через константи у скрипті). Не було використано оптимізаційні процедури на кшталт grid-search, random-search або інші оптимізатори гіперпараметрів.

Batch_size, img_height, hidden_size, dropout, max_lr, кількість епох тощо, встановлюються один раз і лишаються незмінними протягом тренування.

Найважливіші п’ять гіперпараметрів у нашій моделі та значення цих гіперпараметрів:

– img_height = 128. 128 пікселів достатньо, щоб зберегти деталі друкованих літер і водночас не «роздути» тензори. GPU легко тримає батч 32 на 128 на 512;

– img_width = 512. Такий розмір дозволяє приймати слова до ≈ 30 символів без обрізання й формує послідовність із ≤64 кроків, яку комфортно обробляє двошаровий LSTM;

– max_lr = 0.001. Таке значення дає швидку початкову збіжність, а One-Cycle автоматично знижує крок до ≈ 0.00001 наприкінці епох, таке поєднання часто краще, ніж фіксований LR;

– hidden_size = 256. 256 достатньо, щоб утримати контекст слова, але вага мережі лишається ≈ 3M параметрів, що швидко вчиться й легко розгортається;

– batch_size = 32. З таким розміром градієнт стабільний, а GPU з 4 ГБ VRAM не виходить у своп навіть із Dropout-ом і One-Cycle LR.

Зафіксовані у коді гіперпараметри можна побачити у лістингу 3.8, та їх передача у модель також у лістингу 3.9.

Лістинг 3.8 – Фіксація гіперпараметрів у коді

```
def main(args):
    device = torch.device(«cuda» if
torch.cuda.is_available() else «cpu»)
    converter = LabelConverter(ALPHABET)
train_ds = OCRDataset(args.train_tsv, args.images_root,
converter, args.img_height, args.img_width, augment=not
args.no_augment)
val_ds   = OCRDataset(args.val_tsv,   args.images_root,
converter, args.img_height, args.img_width, augment=False)
    # задаємо batch_size
    train_loader = DataLoader(train_ds,
batch_size=args.batch_size, shuffle=True,
num_workers=args.num_workers, collate_fn=ocr_collate_fn,
pin_memory=True)
    val_loader   = DataLoader(val_ds,
batch_size=args.batch_size, shuffle=False,
num_workers=args.num_workers, collate_fn=ocr_collate_fn,
pin_memory=True)

    model = CRNN(len(ALPHABET) + 1).to(device)
    # кодування пропусків
    criterion = nn.CTCLoss(blank=0, zero_infinity=True)
    # вибір оптимайзеру та його learning rate
    optimizer = optim.Adam(model.parameters(),
lr=args.max_lr)
    scheduler = optim.lr_scheduler.OneCycleLR(optimizer,
max_lr=args.max_lr, steps_per_epoch=len(train_loader),
epochs=args.epochs, pct_start=0.1)
```

Лістинг 3.9 – Передача гіперпараметрів у модель через консоль, або стандартними значеннями

```
parser = argparse.ArgumentParser()
parser.add_argument(«—train_tsv»,
default=»labels/splits/train.tsv»)
parser.add_argument(«—val_tsv»,
default=»labels/splits/val.tsv»)
parser.add_argument(«—images_root», default=».»)
parser.add_argument(«—save_dir», default=»checkpoints»)
parser.add_argument(«—img_height», type=int, default=128)
parser.add_argument(«—img_width», type=int, default=512)
parser.add_argument(«—batch_size», type=int, default=32)
parser.add_argument(«—epochs», type=int, default=60)
parser.add_argument(«—max_lr», type=float, default=0.001)
parser.add_argument(«—num_workers», type=int, default=4)
parser.add_argument(«—val_every», type=int, default=10)
parser.add_argument(«—no_augment», action=»store_true»)
args = parser.parse_args()
main(args)
```

Формуємо команди для визову із консолі для тренувального скрипту. Через послідовні виклики add_argument описуються гіперпараметри які можно визвати із консолі та значення за замовчуванням, шляхи до файлів-розбиттів навчальної й валідаційної вибірок, коренева папка з зображеннями, папка для збереження чекпоінтів, розміри кадру після препроцесингу, основні гіперпараметри навчання (розмір пакета, кількість епох, піковий learning-rate), а також службові налаштування на кшталт кількості потоків для підвантаження даних або періодичності валідації.

Якщо аргумент не передано в команді, підставляється його дефолтне значення, тому скрипт можна запускати «як є». Коли всі параметри описано, виклик parse_args() читає налаштування та формує об’єкт args, і цей об’єкт передається у функцію main, яка будує датасети, ініціалізує модель та запускає процес тренування з налаштованою конфігурацією.

3.4.3 Навчання моделі

Після всіх етапів підготовки, можна переходити до наступного етапу, до запуску навчання моделі, вона пройде кількість епох яку було вказано, та буде валідувати кожні n епох також яку вказали. У процессі навчання бачити помилку навчання та метрики на валідації, зазвичай для задач OCR використовують метрику CER, Character Error Rate або коефіцієнт помилок символів. Він розраховується шляхом ділення загальної кількості неправильних символів на загальну кількість символів у довідковому тексті та виражається у відсотках. Код валідації із обговореними метриками написан у лістингу 3.10.

Лістинг 3.10 – Валідація моделі

```
@torch.no_grad()
def evaluate(model, loader, criterion, device, converter):
    model.eval()
    total_loss, pred_texts, gt_texts = 0.0, [], []
for imgs, labels, lengths in tqdm(loader, desc="val",
leave=False):
        imgs = imgs.to(device)
        logits = model(imgs).log_softmax(2)
        t_seq, bsz, _ = logits.size()
        input_lengths = torch.full((bsz,), t_seq,
dtype=torch.int32)
        loss = criterion(logits, labels, input_lengths,
lengths)
        total_loss += loss.item()
        pred_texts.extend(converter.decode(logits.cpu()))
        offset = 0
        for l in lengths:
            seq = labels[offset : offset + l]
            text = "".join(converter.idx2char[i.item()]
for I in seq)
            gt_texts.append(text)
            offset += l
    return total_loss / len(loader), cer(pred_texts,
gt_texts)
```

У цій функції відбувається повний цикл оцінювання моделі на валідаційній вибірці. Декоратор @torch.no_grad() вимикає обчислення градієнтів, тому всі операції виконуються швидше й без споживання додаткової пам’яті. На початку мережу переводять у режим eval(), аби заморозити BatchNorm і Dropout, після чого відкривається цикл по міні-пакетах, який обгорнуто у tqdm для наочного прогрес-бару.

Для кожного пакета зображення переносяться на GPU, проходять через модель, і з виходу беруться лог-ймовірності по всіх класах символів. Довжина тензора по часовій осі t_seq разом із розміром пакета bsz потрібні, щоб сформувати вектор input_lengths, який CTC-втрата використовує як справжню довжину послідовності для кожного зразка. Далі обчислюється сама CTC-втрата та додається до сумарного лічильника.

Після втрати виконується декодування, лог-ймовірності переносять на CPU, подають у converter.decode, і результат-рядок символів накопичується у списку передбачень. У цей же час формується список еталонних текстів, для кожного прикладу береться відрізок тензора labels довжиною l та конвертується назад у рядок через idx2char, і в такий спосіб відтворюється ground truth.

Наприкінці епохи середня втрата обчислюється діленням на кількість пакетів, а функція CER підраховує коефіцієнт помилок по символах між списком передбачень і списком еталонів. Саме ці дві величини, середня CTC-втрата та CER, вони і повертаються з функції, таким чином у процесі навчання на етапі валідації можемо бачити якість нашої моделі.

Обчислення метрики CER у лістингу 3.11.

Лістинг 3.11 – CER-функція

```
def cer(preds, targets):
    dist, length = 0, 0
    for p, t in zip(preds, targets):
        dist += editdistance.eval(p, t)
        length += len(t)
    return dist / length if length else 0.0
```

Коли маємо розуміння як проходить процес навчання, переходимо до самого навчання моделі, метод навчання моделі написан у лістингу 3.12.

Лістинг 3.12 – Метод навчання моделі

```
def train_epoch(model, loader, criterion, optimizer,
scheduler, device):
    model.train()
    total = 0.0
    for imgs, labels, lengths in tqdm(loader,
desc=”train”, leave=False):
        imgs = imgs.to(device)
        logits = model(imgs).log_softmax(2)
t_seq, bsz, _ = logits.size()
        input_lengths = torch.full((bsz,), t_seq,
dtype=torch.int32)
        loss = criterion(logits, labels, input_lengths,
lengths)
        optimizer.zero_grad(set_to_none=True)
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), 5.0)
        optimizer.step()
        scheduler.step()
        total += loss.item()
    return total / len(loader)
```

У цьому фрагменті коду реалізовано одну повну епоху навчання. Спершу мережа переходить у режим train(), тобто вмикаються такі компоненти, як Dropout і оновлення статистик BatchNorm. Далі запускається ітерація по міні-пакетах, обгорнута у tqdm, аби виводити прогрес.

Для кожного пакета зображення передаються на GPU та модель обчислює лог-ймовірності символів уздовж часової осі і відразу застосовує log_softmax, адже саме такий формат очікує CTC-втрата. Довжина отриманої послідовності t_seq разом із розміром пакета bsz потрібні, щоб сформувати вектор input_lengths, який повідомляє CTC, скільки дійсних кроків є у кожного зразка.

Розрахувавши втрату, обнуляємо градієнти (optimizer.zero_grad), виконуємо зворотний прохід (loss.backward()), а одразу після нього застосовуємо clip_grad_norm_ і якщо норма градієнтів перевищує п’ять, її обрізають, щоб запобігти вибуху градієнтів у рекурентній частині. Далі крок оптимізатора оновлює ваги, а виклик scheduler.step() змінює learning-rate відповідно до плану One-Cycle.

Наприкінці кожної ітерації накопичується значення втрати, щоб після проходу всіх пакетів повернути середню величину, вона слугує швидким індикатором, чи мережа збігається протягом цієї епохи.

Після того як модель навчиться, будемо мати директорію «checkpoints», який містить у собі інформацію про ваги та метрики на епохах. Фіксуємо найкращий чекпоінт за показником CER і проводимо остаточну перевірку на тестовій вибірці, яку модель раніше не бачила. Цей етап підтверджує, що якість узагальнення відповідає вимогам і не є наслідком перенавчання на тренувальних даних.

Було згенеровано 25 тис фото, такого розміру даних може бути достатнім для нашої моделі щоб демонструвати гарний результат на не складних зображеннях.

Збережемо наші гіперпараметри які було встановлено, та використаємо дефолтне значення кількості епох 60 та почнемо навчання моделі. Процес навчання моделі зображен на рисунку 3.18.

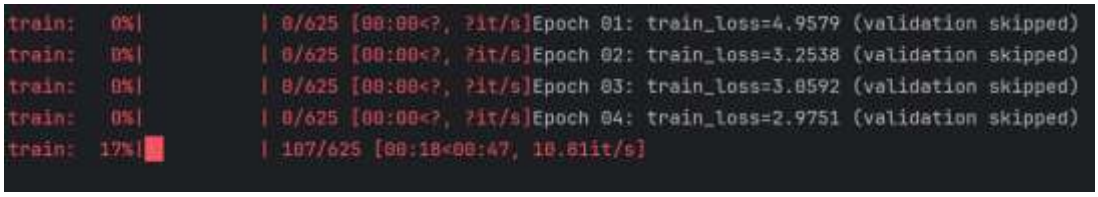

Рисунок 3.18 – Початок навчання моделі

За чотири із шістдесяти епох, модель зменшила свою помилку із приблизно 5 до приблизно 3, це гарний результат, було вказано що валідація

нейромережі проходить кожні 10 епох, всього буде 6 валідацій. Метрики після валідації зображені на рисунку 3.19.

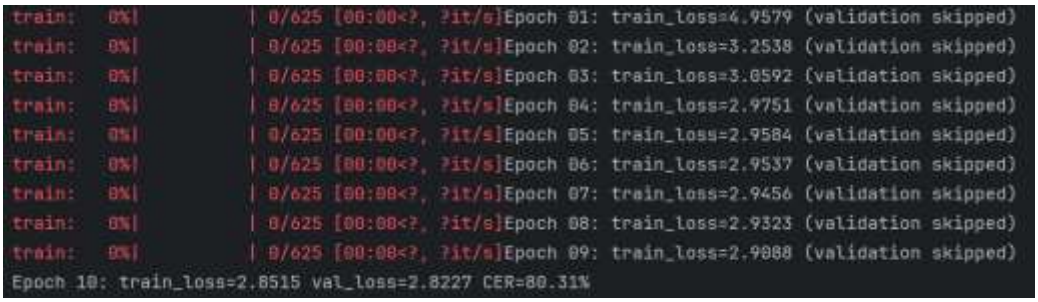

Рисунок 3.19 – Модель після десяти епох

Бачимо що після десятої епохи наш показник CER дорівнює 80%, це багато, але ми поки що на десятій епосі. Мережа насправді поки що вчиться читати та стабільні втрати (~2,9) та CER ≈ 80 % значать, що вона все ще здогадується майже випадковим чином. Метрики писля другої валідації зображені на рисунку 3.20.

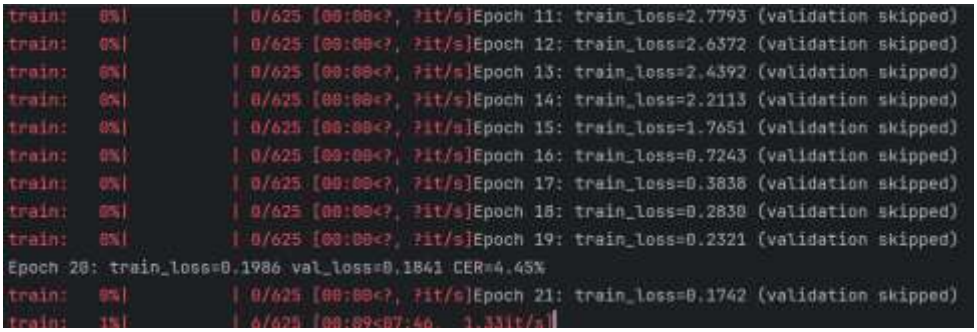

Рисунок 3.20 – Модель після двадцяти епох

На 15 епосі можна побачити що наш лосс 1.7, а вже на 16 епосі різкий спад, та на двадцятій дуже гарний результат для CER із 4.45%.

Такий різкий переход може свідчити про декілька речей які тісно пов’язані із архітектурою нейромережі яку використовуємо.

Різкий перелом траєкторії з високих втрат ≈ 3 до десятих і CER нижче 5 %, типова картина для CRNN-CTC, коли виконуються одразу кілька умов.

У перші 10 % епох scheduler піднімає learning-rate до піка. У цей час мережа швидко сканує простір параметрів, але градієнти стрибкоподібні, тому втрата коливається біля 3. Після шостої та сьомої епохи крок починає спадати логарифмічно, шум зникає, і модель «сідає» у вузьку улоговину, що дає лавиноподібне зниження втрати.

До десятої епохи BiLSTM навчилася узгоджувати лівий і правий контекст слова, тож CTC уже отримує послідовність, де для кожного стовпчика є чіткий максимум і менше «конфліктних» символів. Звідси стрибок CER зі 80% до ≈ 70%, а далі експоненціальний спад.

Коли ймовірність правильного символу стає хоча б трохи вищою за blank, CTC різко скорочує кількість неправильних шляхів. Тонка межа проходить саме біля втрати ≈ від 2.0 до 2.5. Щойно нижче, більшість «альтернатив» відсікається, і CER падає на порядок.

Також 25000 унікальних слів дають приблизно 1000000 символів. Після 15 епох усі зразки побачені стільки разів, що модель знає майже кожну літеру у всіх комбінаціях, це і спричиняє «еврику».

Отже, комбінований ефект плавного зменшення learning-rate, накопичення контексту в BiLSTM і властивостей CTC приводить до різкого покращення саме в середині навчання, далі крива згладжується і CER повільно добивається до мінімуму.

На останній епосі отримуємо такий результат який можна побачити на рисунку 3.21.

Навчання зійшлося до 50-ої епохи, середня і CTC-втрата на тренуванні впали до 0.036, а на валідації стабілізувалися біля 0.10. Це відповідає CER ≈ 2%, тобто усього дві помилки на сто символів. Між train і val-втратою зберігається невеликий розрив (≈0.07), що свідчить про легке, але прийнятне

перенавчання. Модель добре запам’ятала дані і впевнено узагальнює на валідаційних зразках.

Подальші епохи (з 51 до 59) майже не змінюють train-loss і не погіршують CER, отже мережа ввійшла в режим плато. Остання оцінка (epoch 60) лише підтверджує це, давши найкращий CER = 1.97 %. При такому рівні помилок модель уже придатна для розгортання.

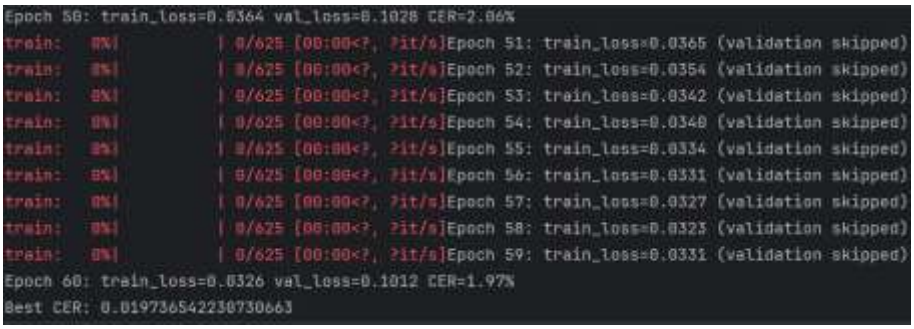

Рисунок 3.21 – Результат навчання нейромережі

Результати навчання по епохам для навчальної та валидаційної виборки можна побачити на рисунку 3.22, та графік метрики CER по епохам на рисунку 3.23.

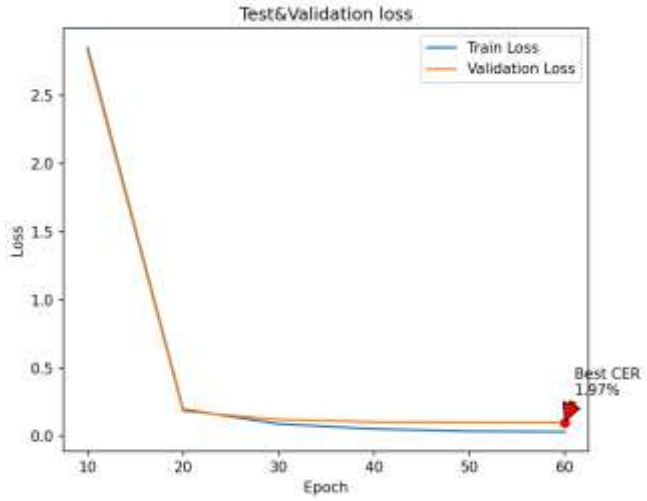

Рисунок 3.22 – Графік помилки для тестової та валідаційних виборок у процесі навчання

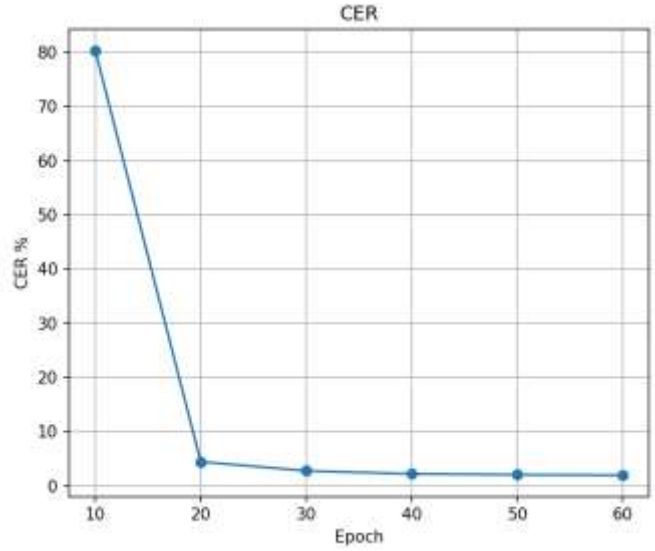

Рисунок 3.23 – Графік показників CER у процесі навчання

3.4.4 Тестування моделі

Після того як модель навчилася, у нас є результати навчання збережені у папці «checkpoints» яка була назначена куди зберігатються результати навчання.

Після навчання моделі потрібно завжди тестувати її на тестових даних які модель ще ніколи не бачила. Було розподілено тестові дані порівну, перша частина тестових даних є синтезованою, а інша ні.

При тестувані будемо орієнтуватися на CER та WER (Word Error Rate) метрики, які дадуть загальну картину про роботу моделі.

Завантажуємо найкращу модель із чекпоентів та проводимо тест, код для цього у лістингу 3.13.

Лістинг 3.13 – Завантаження моделі та тестування

```
device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
converter = LabelConverter(ALPHABET)

test_ds = OCRDataset(
    TEST_TSV,
```

Продовження лістингу 3.13

```
    IMAGES_DIR,
    converter,
    img_height=IMG_H,
    img_width=IMG_W,
    augment=False,
)
test_loader = DataLoader(
    test_ds,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=NUM_WORKERS,
    collate_fn=ocr_collate_fn,
    pin_memory=True,
)

model = CRNN(len(ALPHABET) + 1).to(device)
model.load_state_dict(torch.load(CHECKPOINT,
map_location=device))
criterion = nn.CTCLoss(blank=0, zero_infinity=True)
val_loss, cer, wer = evaluate(model, test_loader,
criterion,
device, converter)
print(f"Test loss : {val_loss}")
print(f"CER       : {cer*100}%")
print(f"WER       : {wer*100}%")
```

Методи для обрахування CER та помилки залишилися такими як і при тренуванні. При обрахуванні WER використовується відстань Левенштейна, вона обчислюється як мінімальна кількість операцій вставки, видалення і заміни, необхідних для перетворення одної послідовності в іншу. Бібліотека editdistance допомогла у цьому.

Результат тестування на рисунку 3.24 знизу.

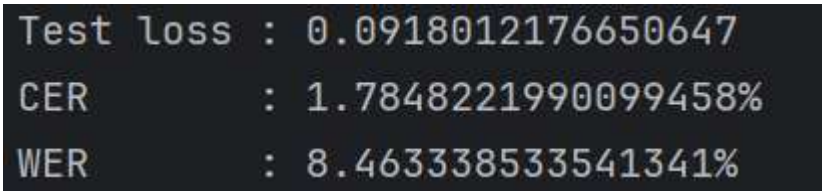

Рисунок 3.24 – Результати тестування моделі

CER 1.8 % означає, що в середньому на 100 символів модель допускає менше ніж 2 помилки (вставка, видалення або заміна). Для друкованого тексту без мовної пост-обробки це дуже високий рівень.

WER 8.5 % виглядає вищим, але це очікувано, одна помилка символа робить усе слово неправильним, тому WER завжди кратно більший за CER.

Також звісно треба провести inference-тест, із якимось складним фото яке має незвичайний для синтезованих фото фон. Модель не може використовувати повні фото, тому що поки що не було задіяно детекцію тексту, тому використаємо рисунок 3.5 із якого зробимо кропи зі словами та передамо моделі (лістинг 3.14).

Лістинг 3.14 – Завантаження моделі та отримання тексту із усіх кропів

```
@torch.no_grad()
def predict(img_path, model, converter, h=128, w=512,
device="cpu"):
img = Image.open(img_path).convert("L")
img = resize_pad(img, h, w)
    tensor = torch.Tensor([(p / 255.0 - 0.5) / 0.5 for p
in img.getdata()]).view(1, 1, h, w).to(device)
    log_probs = model(tensor).log_softmax(2)
    return converter.decode(log_probs.cpu())[0]

def main():
    device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")
    converter = LabelConverter(ALPHABET)
    model = CRNN(len(ALPHABET) + 1).to(device)
    model.load_state_dict(torch.load(CHECKPOINT,
map_location=device))
    model.eval()

    for i in range(1, 10):
        IMAGE_PATH = f"{i}.png"
        text = predict(IMAGE_PATH, model, converter,
device=device)
        print(f'{IMAGE_PATH}  ->  "{text}"')
```

На рисунку 3.25 зображення з якого буде зроблено кропи.

Рисунок 3.25 – Текст з якого треба зробити кропи

Зроблені кропи можна побачити на рисунку 3.26.

Рисунок 3.26 – Кропи із словами для моделі

Результат виконання розробленої моделі для кожного кропу зображені на рисунку 3.27.

```
1.png  ->  "слава"
2.png  ->  "богу"
3.png  ->  "щоно"
4.png  ->  "наші"
5.png  ->  "ошції"
6.png  ->  "побачили"
7.png  ->  "ранок"
8.png  ->  "нового"
9.png  ->  "дту"
```

Рисунок 3.27 – Результати виконання програми

Можно побачити, що шість із дев’яти слів модель класифікувала правильно. Але ж погано опрацювало слово «очі», «що» та «дня». Це може бути із поганою якістю цих кропів, так як слова маленькі, зображення кропу становиться теж маленьким і якість падає. На рисунку 3.28 також є невеликий шум.

Рисунок 3.28 – Якість та шум на кропі

Також не було використано ніякої обробки зображення, такого як наприклад зменшення шуму або перевод її у чорно-білий формат.

Ну і звичайно наша модель не є ідеальною і навчалася на синтезованих даних розміром 20000 екземплярів, це не дуже велике значення для таких моделей і модель може погано розуміти дані які дуже відрізняються від тих що були синтезовані.

3.4.4 Створення OCR пайплайну

Створення OCR пайплайну має під собою на увазі об’єднання детектору тексту та нашої моделі що було розроблено.

Вже було розглянуто модель CRAFT яка робить детекцію тексту, вона є найкращою нейромережою для детекції тексту та переводу зображення у потрібному розмірі, так як наша модель сприймає лише зображення.

Алгоритм наших дій буде таким:

– додаток отримає зображення та передає у CRAFT;

– CRAFT робить детекцію та вирізає слова у окремі фото;

– усі слова які модель CRAFT змогла зберегти у зображення передаємо на класифікацію у нашу модель;

– модель повертає текст та виводить його у додаток.

Спочатку створимо модель CRAFT за допомогою біблиотеки EasyOCR яка буде вирізати слова та зберігати фото у папку, код у лістингу 3.15.

Лістинг 3.15 – Створення моделі CRAFT для зберігання слів у фото

```
import easyocr
from PIL import Image

# встановлення мови тексту
reader = easyocr.Reader(['uk'], gpu=True)

# процедура детекції тексту
results = reader.readtext(
    'test2.jpg',
    detail=1,
    paragraph=False,
    width_ths=0.01,
    link_threshold=0.2
)

# збереженя результатів із фото test2.jpg
img = Image.open('test2.jpg')
for i, (bbox, text, conf) in enumerate(results):
```

Продовження лістингу 3.15

```
    xs = [p[0] for p in bbox]; ys = [p[1] for p in bbox]
    l, t, r, b = min(xs), min(ys), max(xs), max(ys)
    word_img = img.crop((l, t, r, b))
    word_img.save(f'word_{i:03d}.png')
```

На рисунку 3.29 знизу, бачимо результат виконання. Маємо окремі слова зі знаками пунктуації у фото.

Рисунок 3.29 – Вирізані кропи за допомогою CRAFT

Наступним кроком буде об’єднання нашої моделі для детекції тексту та моделі нейроної мережі для класифікації тексту.

На вхід буде поступати зображення яке передаємо у модель CRAFT, яка збереже окремі слова у фото, та кожне фото зі словом передаємо нашій моделі нейроної мережі, код для цього представлен у лістингу 3.16.

Лістинг 3.16 – Об’єднання детекції та класифікації тексту

```
import os
import torch
from craft_detector import detect_and_crop
from ukr_crnn_infer import load_model, predict_image
def run_pipeline(image_path: str, checkpoint: str =
"best.pt", gpu: bool = True) -> dict:
    device = torch.device("cuda" if gpu and
torch.cuda.is_available() else "cpu")
    stem =
os.path.splitext(os.path.basename(image_path))[0]
    crop_dir = f"{stem}_crops"
    crops = detect_and_crop(image_path, crop_dir,
gpu=(device.type == "cuda"))
    if not crops:
return {}
    model, converter = load_model(checkpoint, device)
```

Продовження лістингу 3.16

```
    results = {}
    for path in sorted(crops):
text = predict_image(path, model, converter,
device=device)
        fname = os.path.basename(path)
        results[fname] = text
        print(f"{fname} -> \"{text}\"")
    return results
def main():
parser = argparse.ArgumentParser(description="OCR
Pipeline")
    parser.add_argument("image", nargs="?",
default="test2.jpg", help="source image path")
    parser.add_argument("--checkpoint", "-c",
default="best.pt", help="CRNN checkpoint path")
    parser.add_argument("--gpu", action="store_true",
help="use GPU")
    args = parser.parse_args()
run_pipeline(args.image, args.checkpoint, args.gpu)
```

Наразі було розроблено повний OCR-pipeline який може використовуватися із середи розробки та визиватися із консолі, це буде потрібно при створені додатку для використання трьох нейромереж та проведення тесту.

Було використано рисунок 3.5 для тесту нейромережі, після запуску програма створила окрему папку із кропами та підпапку із назвою фото яку було використали, та окремі зображення які CRAFT модель вирізала та зберегла. Після цього модель для розпізнавання вже обробила усі фото які є у цій папці та конвертувала кожне слово із фото у текст.

Загальну структуру проекту та створені програмою кропи і розпізнані слова зображені на рисунках 3.30 та 3.31 знизу.

Після того як вже було розроблено повну программу, наступним кроком буде об’єднання усіх трьох нейромереж у один додаток, де буде протестована кожна модель на різних фото, та проведен аналіз за метриками.

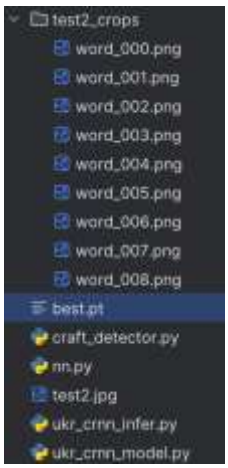

Рисунок 3.30 – Структура проекту та результат виконання

```
word_000.png -> "слава"
word_001.png -> "богу"
word_002.png -> "щоо"
word_003.png -> "наші"
word_004.png -> "оці"
word_005.png -> "побачили"
word_006.png -> "ранок"
word_007.png -> "нового"
word_008.png -> "дняя"
```

Рисунок 3.31 – Результат розпізнавання тексту із кропів

3.4.5 Створення додатку та тестування моделей

Наш додаток повинен забезпечувати використання трьох нейромереж у єдиному вікні, та робити тестування на масиві даних для аналізу метрик.

За допомогою python та бібліотеки PyQt5 створимо такий додаток. Візуальний інтерфейс розробленого додатку для тестування нейромереж зображен на рисунку 3.32.

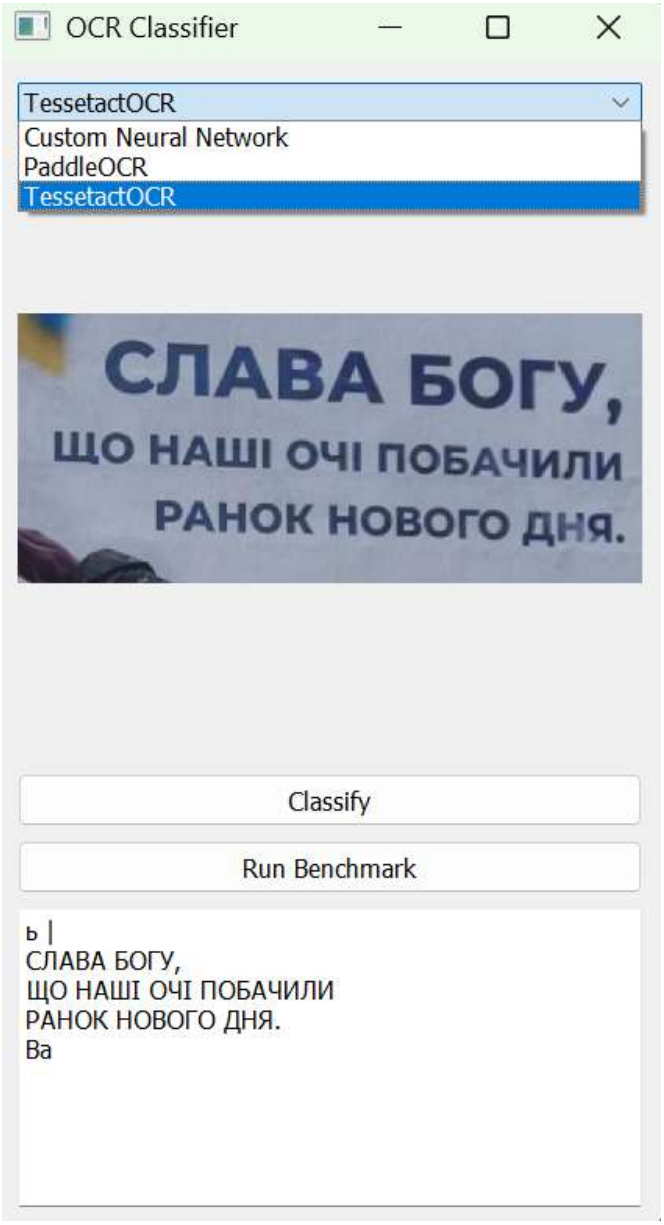

Рисунок 3.32 – Зовнішній інтерфейс додатку трьох нейромереж

Додаємо окремий розділ, де матимемо змогу завантажити файл із тестовою виборкою, і кожна нейромережа пройде по кожному фото и зробить класифікацію. Після цього обраховуємо показники CER та WER та малюємо графіки для наочного аналізу нейромереж.

Використаємо бібліотеку faker та згенеруємо 100 зображень та одразу розмітку до них у форматі csv. Згенеровані цією бібліотекою зображення можна побачити на рисунку 3.33 знизу.

Рисунок 3.33 – Згенеровані тестові дані для фінального аналізу

На останньому етапі проводимо тестування та порівняльний аналіз. Клас нейромереж об’єднано одним методом classify, тому ми можемо просто поліморфно пройти циклом по кожній нейромережі та отримати результат.

Підраховуємо для кожної нейромережі показники CER та WER, та виводимо результат у вигляді графіків. Результати на метриках CER та WER для трьох нейромереж можна побачити на рисунках 3.34 та 3.35 знизу.

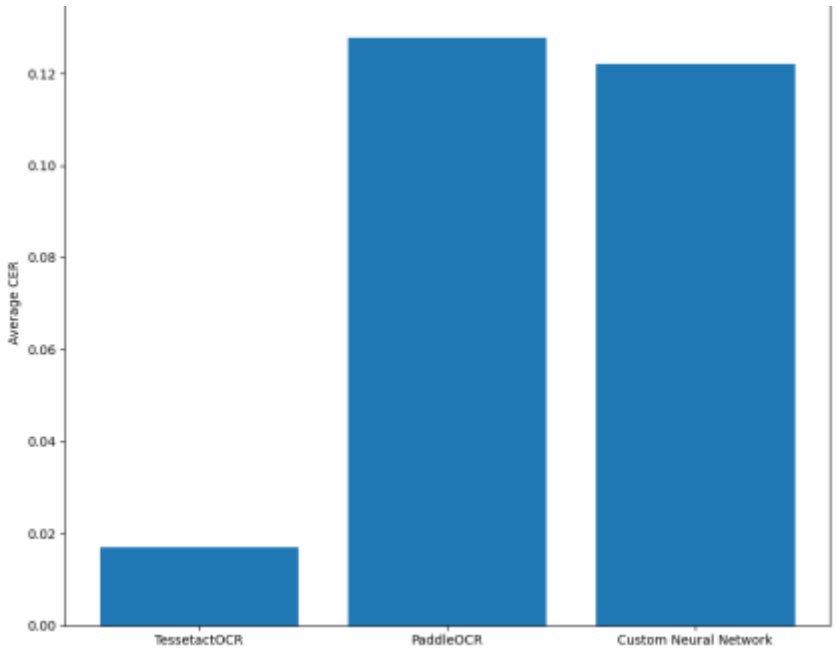

Рисунок 3.34 – Результати середьного CER для кожної нейромережі

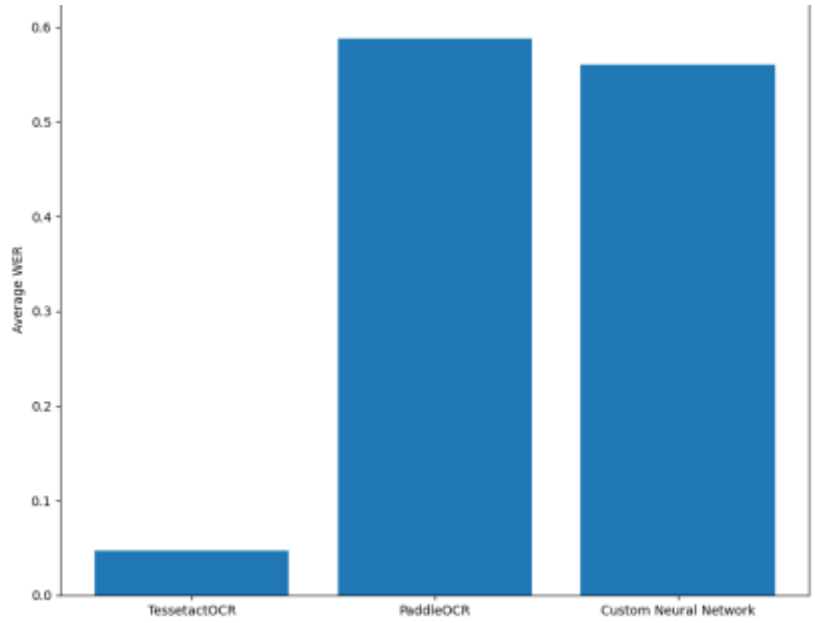

Рисунок 3.35 – Результати середьного WER для кожної нейромережі

Бачимо що результат по CER нашої нейромережі трохи кращий ніж PaddleOCR, але все ж таки на таких простих фото без фону звичайна модель TesseractOCR набагато краще виконує цю роботу.

Тому для повноти аналізу, протестуємо наші три нейромережі на реальних фото із життя. Збираємо 100 зображень з інтернету та розмічаємо текст на фото. Декілька реальних фото зображені на рисунку 3.36.

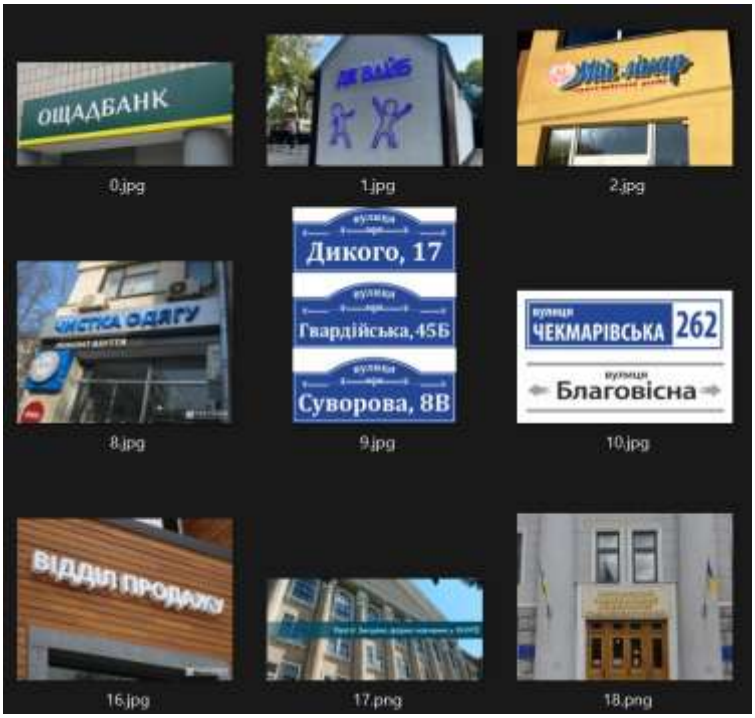

Рисунок 3.36 – Реальні фото для тестування моделей

Результати тестування показані на рисунках 3.27 та 3.28 знизу.

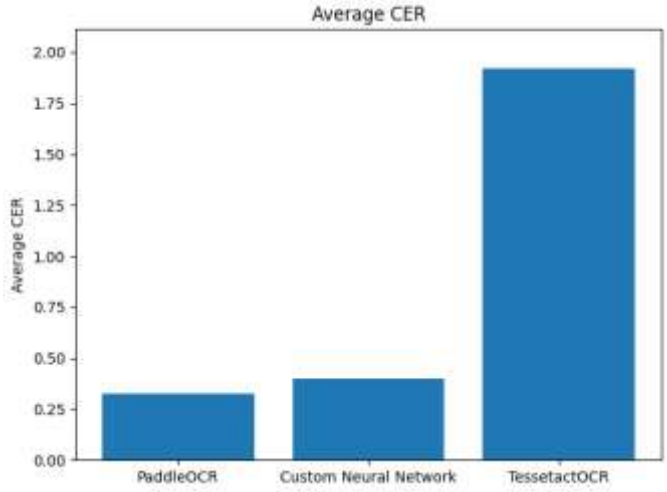

Рисунок 3.37 – Середній показник CER моделей на реальних фото

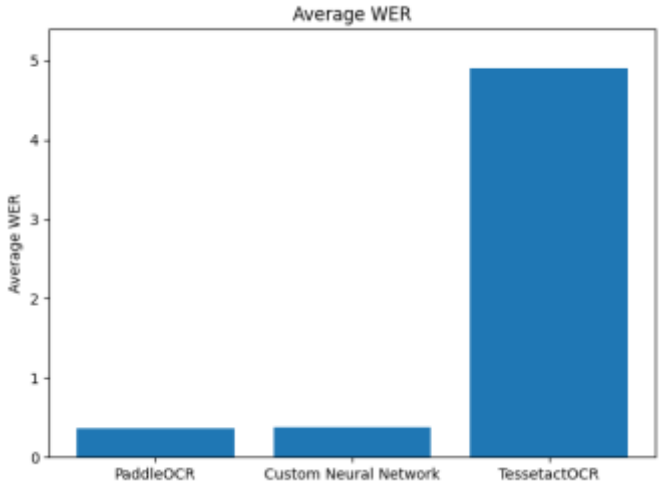

Рисунок 3.38 – Середній показник WER моделей на реальних фото

Як і очікувалось, модель Tesseract дуже гарно виконала задачу для розпізнавання тексту без складного фону, але при використанні його у середовищі зі складним фоном дає дуже багато помилок, так як ще раз, у цієї моделі немає детекції тексту, тому вона може розпізнавати текст будь у якому випадковому місці де його немає.

Але навпаки наша модель та модель PaddleOCR демонструють непоганий результат із приблизно 30% CER. Що демонструє переваги нейромережевих методів для детекції та розпізнавання тексту у складних умовах.

3.4.5 Подальші покращення

Наразі маємо дуже гарну модель яка може конкурувати із моделлю PaddleOCR а у деяких випадках навіть краще показує себе, але ж результат який у нас є, не є ідеальним.

Ця модель була розроблена із оглядкою на обмеженість ресурсів, а саме обчислювальних та інформаційних. Зазвичай дуже великі моделі навчають на потужних серверах, але саме головне те що вони навчаються на великій кількості даних, значно більше ніж у нас (25000). Та

використовують значно більших реальних прикладів, нажаль у україномовному просторі зовсім немає датасету із сценічним текстом українськьою мовою, тому було згенеровані зображення, це також вплинуло на якість моделі.

Підводючи підсумки можна сказати що цю модель можна ще покращити такими способами:

– зібрати більше «живих» даних. Додати щонайменше 25-30 тис. реальних сканів або фотографій;

– використання лінгвістичних методів для корекції результату моделі. Може дати значний приріст результатів;

– beam-search + мовна модель. Під час декодування замінити жадібний алгоритм на beam-search і зчепити з, наприклад, KenLM;

– використання потужнішої аугментації;

– застосувати методи донавчання. Дати моделі навчитися на даних які вона дуже погано розпізнала;

– можна реалізувати донавчання вжі існуючої моделі, замість того щоб навчати модель з нуля;

– предобробка зображень.

ВИСНОВКИ

У результаті роботи було проаналізовано предметну галузь нейронних мереж та використання згорткових нейромереж у вирішенні задачі OCR. Детально було розглянуто роботу нейронних мереж, згорткових нейронних мереж, рекурентних нейронних мереж. Проаналізовано які задачі можна виконувати за допомогою згорткових мереж.

Також було проведено аналіз та використання відомих та авторитетних ресурсів у дослідженні та розробці, були використані видатні книги, популярні статті та публікації з цієї теми.

Розглянуто та проаналізовано детально процесс OCR, розглянуто задачу детекції тексту та розпізнавання тексту. Також було розглянуто та проаналізовано найкращу нейромережу на поточний час CRAFT для детекції тексту.

Було розглянуто та розроблено датасет для навчання нашої моделі, використано бібліотеки для «синтезу» датасету.

Також використано два готових рішення TesseractOCR та PaddleOCR, які використовують два різних підходи до вирішення задачі OCR.

Основною метою було розробка та порівняння двох готових нейромереж та побудованною нейромережою. У ході дослудження було побудовано та навчано нейромережу на розробленому датасеті та отримали досить непогані результати. Побудовано процес детекції та роспізнавання тексту за допомогою CRAFT та нашої нейромережі.

Також у ході роботи було розроблено додаток для порівняння та аналізу метрик наших трьох нейромереж. Розроблена нейромережа показала гарні результати на тестах.

Також було розглянуто декілька пунктів що до покращення розробленої нейромережі.

Таким чином мета роботи була досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Baek Y., Lee B., Han D., Yun S., Lee H. Character region awareness for text detection // *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, USA, 15–20 June 2019. 2019. URL: https://doi.org/10.1109/cvpr.2019.00959 (дата звернення: 15.04.2025).

2. Cybenko G. Approximation by superpositions of a sigmoidal function // *Mathematics of Control, Signals, and Systems*. 1989. Т. 2, № 4. С. 303–314. URL: https://doi.org/10.1007/bf02551274 (дата звернення: 15.04.2025).

3. Zhou X., Yao C., Wen H., Wang Y., Zhou S., He W., Liang J. EAST: an efficient and accurate scene text detector // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, USA, 21–26 July 2017. 2017. URL: https://doi.org/10.1109/cvpr.2017.283 (дата звернення: 16.04.2025).

4. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition // *Proceedings of the IEEE*. 1998. Т. 86, № 11. С. 2278–2324. URL: https://doi.org/10.1109/5.726791 (дата звернення: 17.04.2025).

5. Krizhevsky A., Sutskever I., Hinton G. ImageNet classification with deep convolutional neural networks // *Communications of the ACM*. 2017. Т. 60, № 6. С. 84–90. URL: https://doi.org/10.1145/3065386 (дата звернення: 17.04.2025).

6. Liu Y., Wang Y., Shi H. A convolutional recurrent neural-network-based machine learning for scene text recognition application // *Symmetry*. 2023. Т. 15, № 4. С. 849. URL: https://doi.org/10.3390/sym15040849 (дата звернення: 17.04.2025).

7. Ronneberger O., Fischer P., Brox T. U-Net: convolutional networks for biomedical image segmentation // *Lecture Notes in Computer Science*. Cham,

2015. С. 234–241. URL: https://doi.org/10.1007/978-3-319-24574-4_28 (дата звернення: 18.04.2025).

8. Rumelhart D., Hinton G., Williams R. Learning representations by back-propagating errors // *Nature*. 1986. Т. 323, № 6088. С. 533–536. URL: https://doi.org/10.1038/323533a0 (дата звернення: 18.04.2025).

9. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.-Y., Berg A. SSD: single shot multibox detector // *Computer Vision – ECCV 2016*. Cham, 2016. С. 21–37. URL: https://doi.org/10.1007/978-3-319-46448-0_2 (дата звернення: 18.04.2025).

10. Wilson A., Roelofs R., Stern M., Srebro N., Recht B. The marginal value of adaptive gradient methods in machine learning. *University of California, Berkeley*. 2017. Т. 1, № 1. С. 14. URL: https://arxiv.org/abs/1705.08292 (дата звернення: 19.04.2025).

11. Li M., Zhang P., Liu Y., Li X., Li H. TrOCR: transformer-based optical character recognition with pre-trained models // *Proceedings of the AAAI Conference on Artificial Intelligence*. 2023. Т. 37, № 11. С. 13094–13102. URL: https://doi.org/10.1609/aaai.v37i11.26538 (дата звернення: 21.04.2025).

12. Redmon J., Divvala S., Girshick R., Farhadi A. You only look once: unified, real-time object detection // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, USA, 27–30 June 2016. 2016. URL: https://doi.org/10.1109/cvpr.2016.91 (дата звернення: 22.04.2025).

13. Liao M., Wan Z., Yao C., Chen K., Bai X. Real-time scene text detection with differentiable binarization // *Proceedings of the AAAI Conference on Artificial Intelligence*. 2020. Т. 34, № 7. С. 11474–11481. URL: https://doi.org/10.1609/aaai.v34i07.6812 (дата звернення: 01.05.2025).

14. Smith R. An overview of the Tesseract OCR engine // *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007). IEEE*. 2007. Т. 2. URL: https://doi.org/10.1109/icdar.2007.4376991 (дата звернення: 22.04.2025).