

ПВНЗ «ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ»
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Спеціальність: 122 "Комп'ютерні науки"
Освітньо-професійна програма: "Комп'ютерні науки"

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:
ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ОБРАЗІВ
ОПТИЧНЕ РОЗПІЗНАВАННЯ ТЕКСТУ OCR

Фарановський Богдан Сергійович

Науковий

керівник _____

(підпис наукового керівника)
ступінь) к.т.н., доцент

(прізвище, ініціали, наукове звання і

Допущено до захисту " __ " _____ 2025 р.

Завідувач кафедри _____

(підпис)
ступінь)

(прізвище, ініціали, наукове звання і

Київ - 2025

Зміст

<u>ВСТУП</u>	1
<u>РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗПІЗНАВАННЯ ТЕКСТУ</u>	5
<u>1.1. Основи OCR</u>	5
<u>1.2. Алгоритми традиційного OCR</u>	11
<u>1.3. Обмеження класичних методів</u>	13
<u>1.4. Перехід до нейронних мереж</u>	23
<u>РОЗДІЛ 2. АРХІТЕКТУРИ НЕЙРОННИХ МЕРЕЖ ДЛЯ OCR</u>	26
<u>2.1. Конволюційні нейронні мережі (CNN) для обробки зображень</u>	26
<u>2.2. Рекурентні нейронні мережі (RNN, LSTM) для послідовного тексту</u>	29
<u>2.3. Гібридні моделі (CRNN – CNN+RNN)</u>	36
<u>2.4. Transformers для OCR (Vision Transformer, TrOCR)</u>	41
<u>РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ РОЗПІЗНАВАННЯ ТЕКСТУ</u>	50
<u>3.1. Вибір технологій та бібліотек</u>	50
<u>3.2. Огляд доступних датасетів</u>	55
<u>3.3. Попередня обробка зображень</u>	60
<u>3.4. Створення та навчання моделі</u>	66
<u>3.5. Оцінка точності моделі</u>	80
<u>ВИСНОВКИ</u>	90
<u>ПЕРЕЛІК ЛІТЕРАТУРИ</u>	93
<u>ДОДАТОК</u>	98

ВСТУП

В умовах стрімкого поширення цифрових технологій та експоненційного збільшення обсягів текстової інформації на різноманітних носіях, питання автоматизованої обробки тексту набуває особливої актуальності. Технології оптичного розпізнавання символів (OCR) стали невід'ємним компонентом цифрової трансформації документообігу, дозволяючи перетворювати друковані та рукописні тексти з графічного формату в редагований цифровий текст. Розвиток цієї сфери пройшов довгий шлях, від простих шаблонних методів до складних нейромережових архітектур, і сьогодні OCR-системи використовуються в державному секторі, бізнесі, освіті, медицині та багатьох інших сферах.

По-друге, традиційні методи OCR, що базуються на шаблонному зіставленні та евристичних алгоритмах, демонструють суттєві обмеження при роботі з неструктурованими документами, рукописним текстом та складними умовами зйомки. Вони досягають прийнятної точності лише при роботі з чистими, добре структурованими текстами, але швидко втрачають ефективність при розпізнаванні пошкоджених документів, різних стилів почерку або текстів на нестандартних носіях.

По-третє, розвиток методів глибокого навчання, зокрема згорткових та рекурентних нейронних мереж, трансформерних архітектур, відкриває нові можливості для подолання цих обмежень. Ці технології показують вражаючі результати у розпізнаванні навіть низькоякісних зображень та сильно деформованих текстів, що було недосяжно для класичних алгоритмів. Спостерігається постійне вдосконалення архітектур, методів навчання та препроцесингу, що спонукає до систематизації та аналізу цих підходів.

Актуальність теми дослідження також пов'язана з важливістю збереження культурної спадщини через оцифрування історичних документів та рукописів, які з часом зазнають фізичного руйнування. Високоточні OCR-системи уможливають не лише збереження змісту цих матеріалів, а й забезпечення до них широкого доступу через цифрові бібліотеки та архіви.

Метою дослідження є всебічний аналіз теоретичних основ та практичних аспектів технологій розпізнавання тексту, а також розробка та оцінка ефективності OCR-системи на базі згорткової нейронної мережі для розпізнавання друкованого та рукописного тексту.

Проаналізувати еволюцію методів оптичного розпізнавання символів, виявити їх сильні сторони та обмеження.

Дослідити архітектури нейронних мереж, які застосовуються в сучасних системах OCR, визначити їх порівняльну ефективність.

Розробити та реалізувати OCR-систему на базі згорткової нейронної мережі для розпізнавання символів.

Провести експериментальне дослідження ефективності розробленої системи на стандартизованих наборах даних.

Сформулювати рекомендації щодо подальшого вдосконалення OCR-технологій та розширення сфер їх застосування.

Об'єктом дослідження є процес автоматизованого розпізнавання текстової інформації з растрових зображень.

Предметом дослідження є методи та алгоритми оптичного розпізнавання символів, зокрема нейромережеві архітектури для OCR.

Практичне значення отриманих результатів полягає у розробці OCR-системи, яка може бути застосована для оцифрування документів у різних сферах: від архівної справи до систем обробки форм і анкет. Запропонована модель демонструє стійкість до різних типів шрифтів, розмиття та шумів, що розширює можливості її застосування в реальних умовах.

Результати порівняльного аналізу ефективності різних архітектур можуть бути використані розробниками OCR-систем для вибору оптимальних рішень під конкретні задачі. Особливо важливим є визначення балансу між точністю розпізнавання та обчислювальною складністю, що дозволяє обирати найбільш відповідну модель в залежності від наявних ресурсів та вимог до продуктивності.

Представлені в роботі методики попередньої обробки зображень для підвищення якості розпізнавання можуть бути застосовані не лише в OCR-системах, а й в інших галузях комп'ютерного зору. Зокрема, алгоритми бінаризації, фільтрації шуму та вирівнювання тексту знаходять застосування в системах біометричної ідентифікації, автоматичного аналізу медичних зображень та інших сферах, де потрібна робота з низькоякісними візуальними даними.

Розроблені програмні компоненти можуть бути інтегровані в корпоративні системи документообігу, мобільні додатки для швидкого сканування та розпізнавання тексту, а також спеціалізовані рішення для архівування та індексації документів. Це сприятиме підвищенню ефективності бізнес-процесів, зниженню витрат на ручну обробку даних та мінімізації людських помилок.

Методи дослідження включають аналіз і синтез наукової літератури, порівняльний аналіз архітектур нейронних мереж, математичне моделювання, експериментальне дослідження та статистичний аналіз результатів. В процесі розробки програмної системи використовувалися методи об'єктно-орієнтованого програмування, глибокого навчання, а також інструменти для обробки зображень. Для оцінки ефективності OCR-системи використано методи матричного аналізу, зокрема обчислення матриці сплутування, а також метрики точності, повноти та F1-score.

Структурно робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі "Теоретичні основи розпізнавання тексту" проаналізовано історію розвитку OCR-

технологій, досліджено базові принципи роботи OCR-систем, розглянуто алгоритми традиційного OCR та обмеження класичних методів.

Другий розділ "Архітектури нейронних мереж для OCR" присвячений детальному аналізу сучасних підходів до розпізнавання тексту з використанням згорткових нейронних мереж (CNN), рекурентних нейронних мереж (RNN, LSTM), гібридних моделей (CRNN) та трансформерів. Проведено порівняльний аналіз їх ефективності, визначено сильні та слабкі сторони кожної архітектури, а також досліджено області їх оптимального застосування.

У третьому розділі "Реалізація системи розпізнавання тексту" описано практичну реалізацію OCR-системи, вибір технологій та бібліотек, огляд використаних датасетів, попередню обробку зображень, створення та навчання моделі, а також оцінку її точності. Окрему увагу приділено аналізу помилок розпізнавання та способам їх усунення, а також можливостям масштабування системи та інтеграції з іншими програмними рішеннями.

Результати дослідження можуть бути використані як теоретична та практична основа для подальшого вдосконалення OCR-технологій, розробки спеціалізованих систем розпізнавання тексту для різних галузей та навчальних матеріалів з комп'ютерного зору та обробки природної мови.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗПІЗНАВАННЯ ТЕКСТУ

1.1. Основи OCR

Оптичне розпізнавання символів (OCR) — це технологія, що дозволяє автоматизовано зчитувати текстову інформацію з растрових зображень, сканованих документів і цифрових фотографій^[36,5]. Вона забезпечує перетворення візуального образу символів у цифрову форму, придатну для подальшої обробки, пошуку та збереження. Основна мета OCR — усунути потребу в ручному введенні тексту, тим самим значно підвищуючи ефективність роботи з документацією.

Застосування OCR охоплює широкий спектр сфер: у банківському секторі технологія використовується для зчитування платіжних документів, у логістиці — для обробки транспортних накладних, а в медицині — для цифровізації рецептів і карт пацієнтів^[9,19]. В архівній справі OCR дає змогу оцифровувати історичні рукописи, друковані книги й газетні шпальти. У сучасних мобільних додатках функція сканування тексту стала звичним інструментом для швидкого створення редагованих копій документів.

Такі приклади демонструють здатність OCR виступати мостом між аналоговим і цифровим простором, що є важливим у контексті автоматизації та збереження інформації.

Одними з найпоширеніших рішень на ринку є **ABBYY FineReader** — потужний комерційний інструмент із широкою мовною підтримкою та високою точністю — і **Tesseract**, безплатне програмне забезпечення з відкритим кодом, яке активно застосовується у дослідницьких і промислових проєктах^[36,5,6].

Функціонування OCR-системи відбувається в кілька ключових етапів. Першим є попередня обробка зображення^[4,30,33]. Мета цього кроку — підготувати графічні дані до розпізнавання, зменшивши вплив шуму, нерівномірної яскравості або низького контрасту. Часто застосовується бінаризація — перетворення зображення в чорно-білий формат, що полегшує виявлення текстових елементів.

Далі система переходить до локалізації тексту — виявлення зон, що містять текстові блоки. Цей процес допомагає відокремити текст від нетекстових елементів, таких як зображення або декоративні елементи. Після цього здійснюється сегментація — поділ на окремі символи або слова. Від точності сегментації значною мірою залежить якість наступного етапу — власне розпізнавання символів.

На цьому етапі система аналізує форму кожного виділеного фрагмента й порівнює його з набором шаблонів або ознак, визначаючи, який саме символ було зображено. Згодом отримані символи об'єднуються у зв'язний текстовий документ, який можна зберігати у форматах TXT, DOCX, PDF або структурованих варіантах, таких як JSON. У якісних системах додатково зберігається структура документа — абзаци, заголовки, списки й таблиці.

Класичні методи OCR базуються на алгоритмах комп'ютерного зору та машинного навчання, створених ще до поширення глибоких нейронних мереж. Одним із перших підходів було **шаблонне зіставлення**: кожен символ порівнюється з еталонними зразками, і якщо подібність перевищує певний поріг, символ вважається упізнаним. Цей метод працює добре з друкованими шрифтами стандартного вигляду, але чутливий до навіть незначних спотворень, як-от нахил, розмиття чи зміна масштабу.

Іншим напрямом стали **статистичні методи**, які враховують не лише форму символу, а й мовний контекст — частотність, типові поєднання букв, граматичні закономірності^[24,8]. Це дозволяє зменшити неоднозначність при розпізнаванні, наприклад, між літерою l та цифрою 1. Водночас такі підходи потребують високоякісних мовних моделей, що обмежує їхнє використання в умовах багатомовності або спеціалізованих текстах.

Подальший розвиток привів до впровадження **методів машинного навчання**, зокрема SVM, дерев рішень та наївних баєсівських класифікаторів. Ці моделі опираються на числові характеристики зображень — так звані ознаки (features), як-от форма контурів, кути, пропорції^[19,21]. Після попереднього

аналізу модель навчається класифікувати символи на основі навчального набору даних.

Перевагою цього підходу є гнучкість і краща адаптація до нових шрифтів, проте точність таких систем критично залежить від якості ознак і обсягу навчальних прикладів. Крім того, ручне проектування ознак потребує глибокої експертизи, що ускладнює масштабування рішень.

У сучасному контексті класичні підходи все частіше поступаються місцем **глибокому навчанню**, яке дозволяє автоматично витягувати значущі ознаки зображення. Водночас у низці практичних задач, особливо на пристроях з обмеженими ресурсами, традиційні методи залишаються актуальними завдяки своїй простоті й незалежності від великих обсягів даних.

Багато сучасних рішень поєднують кілька підходів — від евристик до нейромереж — залежно від вимог до точності, швидкості й вартості обчислень.

Поява глибоких нейронних мереж докорінно змінила підходи до OCR. Архітектури на основі згорткових нейронних мереж (CNN) та рекурентних нейронних мереж (RNN) дали змогу суттєво підвищити точність розпізнавання завдяки автоматичному вилученню ознак із зображень.

CNN ефективно обробляють двовимірні структури, що робить їх придатними для виділення візуальних патернів — від простих до складних. У більшості моделей ці ознаки надходять до класифікатора, який визначає символ. Такі архітектури демонструють високу продуктивність на чітких зображеннях зі стандартним макетом, проте мають обмеження у випадках, де важливий контекст чи послідовність.

Для таких задач використовують RNN, а також їхні модифікації — LSTM і GRU, які здатні враховувати попередні символи під час аналізу. Це особливо важливо для розпізнавання слів, де значення символу залежить від його оточення. У результаті виникли гібридні архітектури (CRNN), що поєднують сильні сторони CNN і RNN: аналіз зображення й урахування послідовності одночасно.

Наступним кроком стало впровадження attention-механізмів, які дають моделі змогу фокусуватись на інформативних ділянках зображення. На цій ідеї базується архітектура Transformer, що спочатку застосовувалась у мовній обробці, а згодом була адаптована до візуальних задач.

Серед найуспішніших прикладів — Vision Transformer (ViT) і TrOCR від Microsoft. TrOCR об'єднує модуль візуального аналізу з текстовим декодером, що генерує результат без потреби в ручній сегментації. Завдяки attention ця модель здатна обробляти документи зі складною структурою, включно з багатомовними й рукописними текстами.

Модель TrOCR поєднує зображення документа з текстовим декодером, який генерує текст символ за символом, подібно до перекладача. Вона не вимагає попередньої розмітки або сегментації, оскільки attention-механізми автоматично виділяють потрібні області зображення. Завдяки цьому TrOCR зберігає структуру документа, що особливо важливо в юридичних, фінансових і адміністративних текстах.

На практиці подібні архітектури застосовуються в масштабних хмарних рішеннях. Системи на кшталт Google Cloud Vision OCR або Amazon Textract здатні обробляти мільйони документів щодня. Вони автоматично розпізнають текст, таблиці, поля форм і структуру PDF-файлів, що дає змогу повністю автоматизувати документообіг — від податкових декларацій до контрактів.

У державному секторі OCR сприяє створенню цифрових архівів зі старих паперових реєстрів, відкриваючи доступ до інформації, яка раніше була недоступною для пошуку й аналізу. Крім того, сучасні системи інтегруються з модулями перевірки даних і автоматичного заповнення полів, наприклад у CRM-системах. OCR-сервіс може зчитати реквізити з рахунку-фактури й одразу внести їх у внутрішні документи.

Це свідчить про перехід OCR до нової ролі — не лише інструмента для зчитування символів, а повноцінного елемента семантичного аналізу документів. Технологія починає працювати з інформацією як зі змістом, а не лише з формою.

Водночас OCR має важливу соціальну місію. Він допомагає зберігати культурну спадщину, оцифровуючи історичні документи, і відкриває доступ до текстів для людей із порушенням зору. У країнах із невеликим цифровим представленням мов OCR сприяє збереженню локальних знань.

У багатьох критично важливих сферах — зокрема в медицині, праві та банківській справі — OCR використовується не лише для зчитування тексту, а й для прийняття рішень. У таких випадках важливо не просто отримати результат, а знати, наскільки система впевнена у правильності кожного символу. Це стимулює розробку моделей, які можуть супроводжувати текст додатковою інформацією: рівнем довіри, можливими альтернативами чи позначенням сумнівних ділянок.

Такі функції дають змогу будувати гібридні OCR-системи, де розпізнаний текст перевіряється автоматично — за допомогою словників, статистичних моделей або зовнішніх баз даних. У випадках, коли ціна помилки надто висока, впроваджується етап ручної верифікації. Проте це знижує рівень автоматизації, і виникає потреба в балансі між автономністю та контролем.

Окрему увагу привертає етичне питання відповідальності: хто має нести наслідки у разі помилки — розробник моделі, постачальник даних чи кінцевий користувач? Це особливо актуально у зв'язку з правовим регулюванням ШІ у сфері документообігу.

Сучасні системи OCR часто працюють як "чорні скриньки", не даючи зрозуміти, як було отримано той чи інший результат. Тому зростає інтерес до **пояснювального штучного інтелекту (XAI)**. Його мета — зробити рішення прозорими для людини. У цьому контексті використовують карти уваги (attention maps), трасування обчислювального шляху та інструменти для перегляду і виправлення фрагментів розпізнаного тексту. Такі можливості підвищують довіру до систем, дають змогу виявляти системні помилки й адаптувати модель до нових задач.

Технологія OCR продовжує розвиватися не лише в глибину, а й у ширину — інтегруючись у ширші процеси аналізу, автоматизації та інтерпретації

інформації. Серед перспективних напрямів розвитку — мультимовність, глибока контекстуальна інтеграція, персоналізація під користувача, а також поєднання з іншими модальностями — голосом, жестами, відео. OCR дедалі більше перетворюється з окремої технології на інфраструктурну складову розумної взаємодії людини з цифровим середовищем.

OCR має довгу історію. Ще у 1929 році був запатентований перший пристрій для автоматичного зчитування друкованих символів. У 1960-х роках з'явилися стандартизовані шрифти OCR-A та OCR-B, які полегшували ідентифікацію символів обчислювальними машинами. У 1980-х технологія почала активно застосовуватись у поштових службах, а в 1990-х — стала доступною для масового користувача завдяки появі сканерів і таких програм, як ABBYY FineReader.

З часом OCR-системи почали не просто розпізнавати символи, а й аналізувати структуру документа. У цьому допомагають layout-аналіз, структурна сегментація та мультимодальні архітектури, які поєднують візуальний і текстовий контекст. Яскравий приклад — Google Document AI, який може автоматично виділяти поля, заголовки, таблиці й експортувати їх у структурованому вигляді.

Складнішим завданням є **розпізнавання тексту в реальних сценах** — на вивісках, етикетках, дорожніх знаках, де присутні спотворення, тіні, незвичні фони та перспективні викривлення. Для цього поєднують технології детекції об'єктів (як-от YOLO або Faster R-CNN) з OCR-модулями. Результат — здатність розпізнавати текст навіть на рухомих або викривлених поверхнях.

Моделі нового покоління, такі як **TrOCR** або **Donut**, ідуть ще далі. TrOCR «перекладає» зображення документа у текстову послідовність без попередньої розмітки. Donut повністю відмовляється від розпізнавання символів як окремих одиниць — натомість вона одразу формує структуроване представлення документа (наприклад, у форматі JSON). Це дає змогу створювати рішення, здатні одразу видавати результат, придатний для автоматичного імпорту в системи управління даними.

Особливу роль відіграють **мобільні та автономні OCR-системи**, що працюють без підключення до хмари. Їх використовують у медичних, військових, аграрних і логістичних задачах. Щоб забезпечити ефективність на пристроях з обмеженими ресурсами, моделі оптимізують за допомогою технік квантування, pruning і distillation.

Ці рішення відкривають шлях до більш демократичного використання OCR, але також потребують додаткових гарантій безпеки й конфіденційності, особливо коли йдеться про чутливу інформацію.

OCR перестав бути лише способом розпізнавання тексту — він став частиною цифрового інтелекту, здатного інтерпретувати, структурувати й пояснювати документи у всіх їхніх формах.

1.2. Алгоритми традиційного OCR

Традиційні методи OCR були розроблені задовго до появи глибоких нейронних мереж і тривалий час залишались основою для автоматичного розпізнавання друкованого тексту^[36,22]. Ці підходи відзначаються відносною простотою реалізації, швидкою обробкою та низькими обчислювальними витратами, що зробило їх поширеними у перших комерційних і державних OCR-системах.

Одним із базових методів є **шаблонне зіставлення** (template matching), при якому кожен символ порівнюється з набором еталонних зразків. Якщо подібність перевищує певний поріг, система вважає символ упізнаним. Такий підхід демонструє високу точність у випадках з чітким шрифтом та стабільними умовами сканування^[9,5]. Проте навіть незначне викривлення — нахил, масштаб, розмиття — істотно знижує якість розпізнавання.

Інша група методів — **евристичні алгоритми**, що працюють на основі набору правил, сформованих за геометричними та топологічними характеристиками символів. Наприклад, система може оцінювати кількість перетинів горизонтальних або вертикальних ліній, замкнутість контурів, співвідношення висоти до ширини. На основі цих ознак приймається рішення

щодо ймовірної відповідності символу. Подібні правила легко реалізуються й не потребують складного навчання, проте не є універсальними — особливо у випадках з рукописним текстом або нестандартними шрифтами.

Окрім індивідуальних ознак, евристичні методи можуть враховувати позицію символів на зображенні, їхню орієнтацію чи відстань до сусідніх елементів. Такі підходи залишаються корисними в системах з обмеженими ресурсами або для попереднього фільтрування результатів.

Ще одним кроком у розвитку традиційних OCR-алгоритмів стало використання **статистичних методів**, які доповнювали геометричний аналіз контекстною інформацією. Ці підходи враховують частотність символів у мові, характерні поєднання букв, а також граматичні закономірності. Таким чином, OCR-система може уточнювати розпізнавання сумнівних символів, спираючись на їхнє оточення. Наприклад, якщо між варіантами «l» і «1» немає явної візуальної переваги, система може обрати той, що краще узгоджується з контекстом слова.

Розширення статистичних підходів привело до поєднання розпізнавання символів із **мовними моделями**, здатними враховувати морфологічну структуру тексту. Це дозволило системам адаптуватись до конкретної мови або предметної області, однак ефективність таких методів залежить від наявності якісних корпусів текстів. Зі зростанням кількості мов і стилістичної варіативності текстів стабільність результатів падала.

Із розвитком алгоритмів машинного навчання традиційні системи почали застосовувати **класифікатори**, що навчаються на прикладах — серед них метод опорних векторів (SVM), дерева рішень та наївні байєсівські моделі. Такі підходи не потребували ручного формулювання евристик, а базувались на наборі числових ознак, обчислених із графічних характеристик символу. Це могли бути кути, контурні точки, співвідношення сторін тощо.

Модель аналізувала приклади зі встановленими мітками (наприклад, символами латинського алфавіту) й будувала гіперплощину для розділення класів. У результаті система могла адаптуватися до нових шрифтів і варіацій

без прямого перепрограмування. Проте результат значною мірою залежав від якості вибраних ознак — якщо вони не фіксували відмінностей між схожими символами, розпізнавання ставало ненадійним.

Хоча класичні методи OCR значною мірою поступилися місцем глибокому навчанню, вони й надалі залишаються актуальними у низці прикладних задач. Їхня привабливість полягає у **швидкодії**, невибагливості до обчислювальних ресурсів та **простоті реалізації**, що робить їх особливо придатними для мобільних або вбудованих пристроїв, де немає змоги розгорнути повноцінні нейронні моделі.

Вони також зберігають значення у випадках, коли формат тексту добре відомий і контрольований — наприклад, при розпізнаванні штрих-кодів, номерів, стандартних форм або банківських документів з машинозчитуваними шрифтами. У таких сценаріях прості шаблонні або евристичні алгоритми здатні досягати достатньої точності без складної обробки.

У сучасних системах нерідко використовують **гібридний підхід**, поєднуючи традиційні алгоритми з компонентами машинного навчання або нейронних мереж. Наприклад, шаблонне зіставлення може застосовуватись як швидкий попередній фільтр, після чого результати уточнюються за допомогою класифікатора. Або, навпаки, мовна модель може «підчищати» результат, сформований евристичним методом.

Таким чином, класичні алгоритми виконують роль простих, контрольованих та пояснюваних механізмів, що добре працюють у стабільних умовах. Їхня обмеженість у роботі з рукописним текстом, викривленнями чи новими шрифтами компенсується швидкістю, передбачуваністю та сумісністю з простими пристроями.

1.3. Обмеження класичних методів

Класичні методи оптичного розпізнавання символів (OCR), зокрема шаблонне зіставлення та евристичні алгоритми, протягом кількох десятиліть були основою для автоматизованої обробки текстової інформації^[22,19]. Їх ефективність визначалася здатністю стабільно працювати з друкованими

матеріалами, які мали чіткі шрифти, однорідний фон і мінімальну кількість артефактів. Простота реалізації та невеликі обчислювальні витрати зробили ці підходи стандартом у процесах оцифрування книг, офіційних документів і типових форм у другій половині XX століття. Однак з поширенням цифрових пристроїв, зростанням обсягів візуальних даних і появою нових типів джерел, таких як фотографії, архівні скани й рукописи, обмеження класичних методів стали очевидними. Системи, які використовують фіксовані шаблони або жорстко визначені правила, демонструють високу чутливість до шумів, нетипових шрифтів і, особливо, непередбачуваних форм людського письма.

Однією з найпоширеніших проблем для класичних OCR-систем є розмиття зображень. Алгоритми, що базуються на попиксельному порівнянні, надзвичайно чутливі до найменших змін у структурі тексту^[9,4]. Розмиття, спричинене рухом камери, поганим фокусуванням, низькою якістю сканування або недостатнім освітленням, знижує контраст між символами та фоном, а також робить межі букв нечіткими. Як наслідок, система не здатна чітко визначити межі символів, і результатом розпізнавання стають численні помилки або повна втрата інформації.

На Рис 1.1 продемонстровано, як розмиття впливає на точність OCR. Зображення чітко показує, що навіть незначне розмиття тексту може призвести до серйозних помилок. Наприклад, початковий текст «TEXT FOR OCR» після розмиття може бути помилково розпізнаний як «TED TOR OCR». Це підкреслює важливість попередньої обробки зображень, яка допомагає зменшити або повністю усунути такі дефекти.

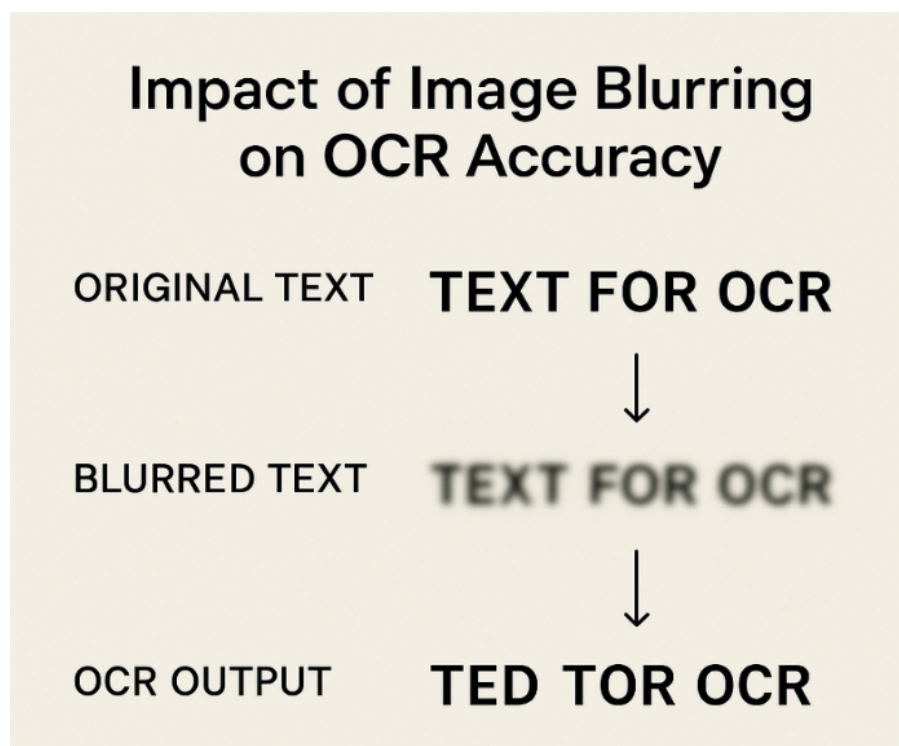


Рисунок 1.1 – Вплив розмитого зображення на точність

Як показано на рис. 1.1, навіть легке розмиття зображення може спричинити суттєві помилки у результатах розпізнавання.

Особливо критичною є нездатність класичних алгоритмів адаптуватися до деформацій або зміщень символів. Навіть мінімальне зрушення пікселів порушує збіг з шаблоном і може спричинити нерозпізнавання цілого слова. Якщо врахувати артефакти стиснення, що виникають під час передачі зображень через месенджери чи зберігання в архівах, стає очевидним, чому класичні OCR-системи погано справляються з сучасним цифровим середовищем, де ідеальні умови зйомки зустрічаються дуже рідко.

Ці проблеми потребують гнучкіших підходів, про які йтиметься у наступному розділі. Ще одним суттєвим обмеженням класичних підходів є їх слабка адаптивність до нестандартних або декоративних шрифтів. Тоді як традиційні алгоритми демонструють високу точність при розпізнаванні символів, набраних стандартними гарнітурами на кшталт Times New Roman або Arial, вони майже не здатні правильно інтерпретувати текст, виконаний курсивом, готикою чи художньо стилізованими літерами. Причина полягає в

тому, що навіть незначні відхилення форми — подовжений штрих, подвійна петля чи декоративний елемент — можуть порушити відповідність між шаблоном і вхідним зображенням.

Декоративні та нестандартні шрифти, особливо в історичних документах або дизайнерських оформленнях, можуть суттєво знижувати ефективність шаблонного розпізнавання^[29,8]. Як показано на рис. 1.4, модифікації форми символів можуть призводити до їхньої неправильного трактування: “g” може бути сплутана з “q”, а “A” — із “R”

Standard Font	Decorative font	OCR Output
g	g Curled tails cause confusion	q
A	A Lack of straight edges affects recognition	R

Рисунок 1.2 – Вплив декоративних шрифтів на точність OCR

Подібні помилки особливо поширені при обробці логотипів або брендових написів, де шрифти часто навмисно змінені з естетичних або маркетингових міркувань. У таких випадках навіть добре відомі назви — на кшталт «CocaCola» або «Google» — можуть бути перекручені до «Cocaeola» або «Gooqle». Аналогічні труднощі виникають і при цифровізації історичних документів, створених за допомогою друкарських машин або гравіювання, де

кожна літера може мати індивідуальний нахил, нетипову інтерлінажу або декоративні особливості.

Ці виклики також сигналізують про необхідність більш гнучких методів аналізу, здатних вийти за межі суворого шаблонного зіставлення. Найбільш серйозним викликом для традиційних OCR-систем залишається обробка рукописного тексту^[10,29]. На відміну від друкованих символів, рукопис має високу міжособистісну варіативність, змінюється навіть у межах одного документа і часто не піддається формалізації. Наприклад, цифра «7» може мати поперечну риску або бути написаною без неї, а «а» — нагадувати за формою «о». Це унеможливорює ефективне використання шаблонного підходу, оскільки зникає повторюваність графем. Евристичні методи, що ґрунтуються на геометричних ознаках (кількість петель, напрямок штрихів тощо), також виявляються безсилими перед складними з'єднаннями літер або декоративними елементами почерку, які не піддаються чіткій структуризації.

Особливо гостро проблема розпізнавання постає у медичних документах, де лікарі широко застосовують аббревіатури, скорочення та власний почерк. У таких випадках навіть людина не завжди може з упевненістю прочитати написане, а класичні OCR-системи або зовсім не розпізнають текст, або пропонують варіанти, що не мають жодного змісту.

Схожа ситуація виникає при обробці анкет, заповнених вручну, або шкільних зошитів, де кількість варіацій у написанні символів породжує цілий каскад помилок. Це суттєво ускладнює автоматизацію опрацювання таких документів у державних установах, бібліотеках або під час соціологічних досліджень.

Сегментація — один із найбільш критичних етапів класичного OCR. Невірне злиття або поділ символів призводить до глибоких помилок на рівні розпізнавання. Як видно на рис. 1.3, злиття “с” та “l” може бути інтерпретовано як “d”, а далі — некоректно розбито на “R”.

Character Segmentation Issues in OCR

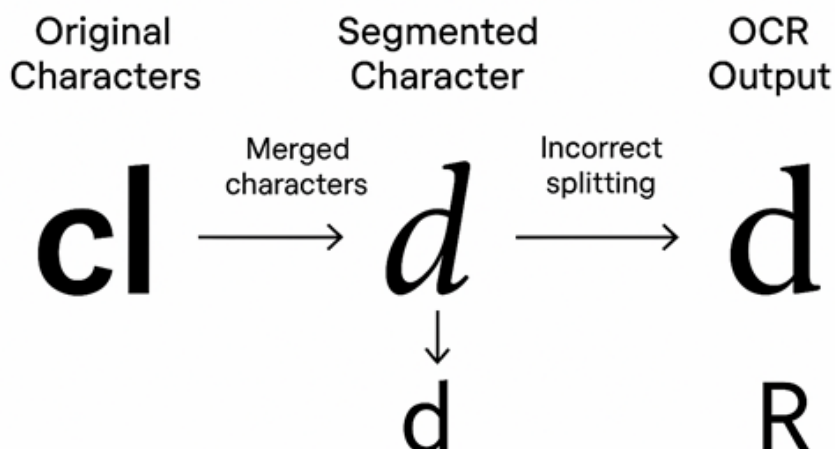


Рисунок 1.3 – Типові помилки сегментації символів

Загалом, при обробці рукописів класичні методи демонструють або повну неспроможність, або надто високу похибку, яка зводить нанівець автоматизацію. У таких випадках здатність людини до контекстної інтерпретації залишається набагато ефективнішою.

Особливо яскраво проблеми класичних методів виявляються у випадках, коли кожна літера може мати кілька варіантів написання — навіть у межах одного алфавіту. Це добре ілюструє ситуація з рукописною літерою «g», яка у друкованому варіанті має округлу форму, у курсивному — спрощену або кутасту, а в декоративному стилі — елементи каліграфії з виразним візуальним відхиленням. Алгоритми, що працюють за принципом шаблонного зіставлення, у подібних випадках просто не можуть знайти відповідність у бібліотеці еталонів, тому або пропускають символ, або видають помилку розпізнавання.

Злиття сусідніх символів є типовою проблемою сегментації у класичних OCR-системах. Як зображено на рис. 1.2, пари символів “rn” можуть бути помилково розпізнані як “m” — особливо у випадках друкованих шрифтів з тісним інтервалом або розмитими краями.

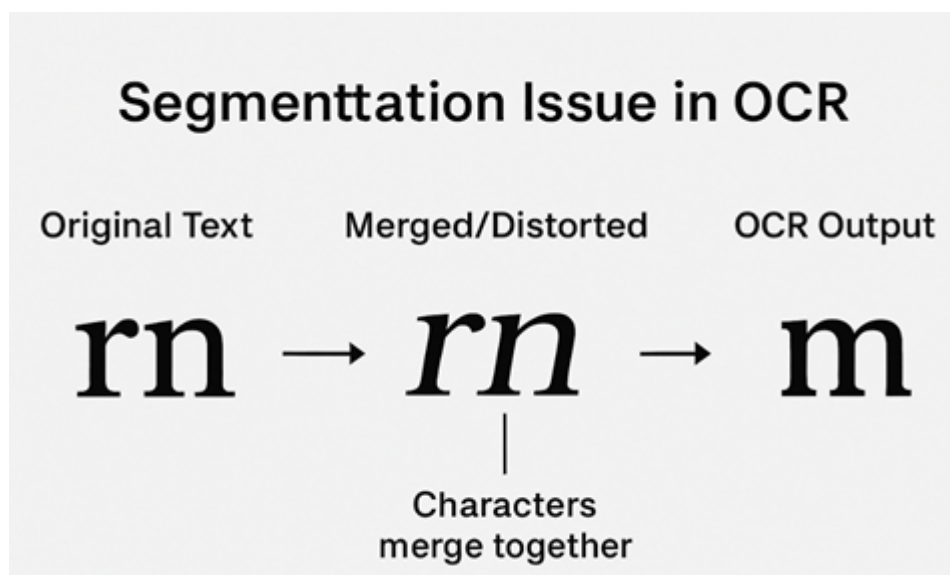


Рисунок 1.4 – Приклад злиття символів у процесі OCR

У контексті шаблонних алгоритмів це означає, що будь-яка нестандартна форма символу перетворюється на перешкоду. Якщо система не знаходить точного збігу, символ вважається нерозпізнаним. Евристичні методи, які аналізують геометричні характеристики, не можуть компенсувати цю проблему, бо складні почерки часто не мають стабільної внутрішньої логіки.

Наприклад, у слові *together*, написаному рукописно, літери «g», «e» та «t» можуть зливатися в один фрагмент, утворюючи єдиний графічний об'єкт, який не відповідає жодному з очікуваних шаблонів. Алгоритм, що розраховує на вертикальні та горизонтальні лінії у заданій конфігурації, просто «ламається», не маючи змоги проаналізувати такий фрагмент.

Це критично для документів, де рукопис є основною формою подання інформації: медичні рецепти, історичні рукописи, анкетні форми. У таких випадках класичні OCR-системи або не видають жодного результату, або породжують набір випадкових символів, непридатний для подальшого використання.

Особливо показовим є приклад масової обробки анкет у державних установах. Уявімо тисячі форм, заповнених вручну людьми різного віку, з індивідуальними стилями письма — від друкованих великих літер до поспішного курсиву зі скороченнями. Класичні методи виявляються

неспрможними опрацювати таке розмаїття. Вони потребують чіткості та повторюваності, яких рукопис не забезпечує. Як наслідок, автоматизація процесів обробки таких документів значно ускладнюється або повністю втрачає сенс.

Це обмежує можливості OCR у важливих галузях, де джерелом даних є рукопис: цифровізація архівів, обробка заяв на соціальні виплати, реєстраційних форм чи музейних журналів. Масштаб проблеми настільки значний, що в багатьох випадках оператори змушені повністю переходити до ручного опрацювання.

Ще один критичний недолік класичних методів — відсутність контекстної перевірки. Наприклад, слово «cat», написане нерозбірливим почерком, може бути інтерпретовано як «eat», «cot» або навіть «eai», оскільки алгоритм не аналізує навколишній контекст і не перевіряє лексичну правдоподібність. Унаслідок цього система не просто помиляється — вона створює текст, непридатний для аналізу або автоматичного введення в базу даних.

На Рисунку 5 зображено реальний приклад складної комбінації проблем: рукописна анкета, сфотографована під кутом, з нерівномірним освітленням і розмиттям символів. У таких випадках класичні системи втрачають здатність до сегментації, шаблони не спрацьовують, а результат обмежується випадковою послідовністю символів або порожніми полями.

Handwritten Variations for Letter in OCR Systems		
Printed	Cursive	Resultat OCR
 ↑ Rounded form	 ↑ Simplified or angular in fast writing	 ↑ Decorative or calligraphic

Рисунок 1.5 – Варіації написання літери “g”

Комбінація кількох факторів лише погіршує ситуацію. Якщо розмиття вже саме по собі викликає труднощі, то в поєднанні з нестандартним шрифтом або рукописом точність OCR стрімко знижується до критичних меж. Наприклад, якщо підпис у документі зроблено стилізованим курсивом і водночас сфотографовано під кутом, навіть найсучасніші класичні OCR-системи здатні видати хибний або порожній результат. У кращому разі — можливо налаштувати систему під конкретну форму, але це не масштабується і не вирішує проблему універсальності.

Підсумовуючи, можна стверджувати: серед усіх перешкод саме рукописний текст становить найбільший виклик для класичних OCR-систем. Його непередбачуваність, індивідуальність і поєднання з іншими складнощами — розмиттям, перспективними викривленнями, нестандартними носіями — створюють ситуацію, в якій шаблонні та евристичні методи втрачають свою функціональність.

Хоча розмиття або використання декоративного шрифту можуть частково компенсуватися фільтрацією, масштабуванням чи розширенням бази шаблонів,

рукопис — це не просто варіація, а повноцінна імпровізація, яка руйнує саму ідею стабільної форми символу. Класичні алгоритми не здатні навчатися на ходу або адаптуватися до нових варіантів графем, що робить їх непридатними для широкого спектра сучасних задач.

У випадках, коли навіть прості слова не можуть бути впевнено розпізнані, зростає потреба у системах, які здатні до гнучкості, самонавчання та контекстного аналізу. З розвитком цифрових технологій і накопиченням великих обсягів візуальних даних стало зрозуміло, що вирішення проблем розпізнавання тексту в реальних умовах потребує принципово інших підходів.

Ці виклики стали каталізатором переходу до нових методів, заснованих на глибинному навчанні, які не просто розпізнають форму символу, а сприймають його як елемент у ширшому контексті.

Усе вищезгадане вказує на те, що класичні методи OCR, попри їхню історичну значущість і ефективність у формалізованих середовищах, не здатні забезпечити прийнятну якість розпізнавання в умовах реального цифрового середовища. Збільшення кількості джерел візуального тексту — від смартфонів і планшетів до камер спостереження та архівних сканерів — супроводжується зростанням обсягу «нестандартних» випадків: пошкоджені документи, нестандартне форматування, низька якість зображень, естетично стилізовані або рукописні написи. У таких умовах жорстко задані правила або фіксовані шаблони втрачають свою ефективність.

Класичні алгоритми вимагають стабільності, чітких меж символів і передбачуваної геометрії, проте сучасні задачі дедалі частіше виходять за межі таких припущень. Як наслідок, результати розпізнавання часто не підлягають автоматизованій обробці без глибокого втручання людини, що суперечить самій ідеї OCR як інструменту для масштабного й швидкого витягання текстової інформації.

Потреба в підходах, які здатні до адаптації, навчання й аналізу контексту, стала очевидною. І хоча на цьому етапі не йдеться про повну відмову від класичних рішень, сучасні вимоги змушують шукати гнучкіші інструменти, які

враховують не лише форму символів, а й їхню позицію, частотність, морфологічні варіації та смислові зв'язки.

Про такі методи йтиметься у наступному розділі, де буде розглянуто нову хвилю технологій, здатних істотно змінити парадигму розпізнавання тексту.

1.4. Перехід до нейронних мереж

Обмеженість класичних алгоритмів OCR, зокрема їхня чутливість до шумів, нестандартних шрифтів і рукопису, підштовхнула розвиток технологій у бік глибокого навчання^[19,21,22]. Нейронні мережі, на відміну від статичних алгоритмів, здатні самостійно адаптуватися до нових вхідних даних, навчаючись виявляти важливі ознаки та закономірності без участі людини. Це стало вирішальним фактором у переході до архітектур, які не залежать від заздалегідь визначених шаблонів чи ручної інженерії ознак.

Глибоке навчання принесло в OCR не просто збільшення точності — воно кардинально змінило сам підхід до розпізнавання. Там, де раніше доводилось проектувати правила, сучасні моделі будують власні уявлення про текст, виходячи з даних, на яких їх навчено. Це дозволяє їм працювати зі складними документами, адаптуватися до викривлень, нових мов чи нестандартних форматів.

Серед перших архітектур, що заклали основу нової хвилі OCR-систем, стали згорткові нейронні мережі (CNN)^[15,22,35]. Їхня здатність працювати з піксельними структурами зробила їх ідеальними для обробки зображень. На відміну від попередніх підходів, CNN не просто аналізують окремі символи, а формують ієрархію ознак — від простих країв і контурів до складних компонентів шрифту.

У типовому сценарії CNN використовується як екстрактор ознак: мережа перетворює вхідне зображення на карту високорівневих характеристик, які зберігають форму літер, просторові зв'язки та локальні деталі. Потім ці представлення передаються далі — до класифікатора або іншого блоку системи, що відповідає за прийняття рішення. Такий підхід добре працює при обробці друкованого тексту, де кожна літера має відносно стабільну геометрію.

Проте CNN не враховують порядок елементів у тексті, що є критичним для розуміння слів і речень. У відповідь на цю потребу були розроблені рекурентні нейронні мережі (RNN), які обробляють дані послідовно, зберігаючи інформацію про попередні елементи. На практиці це означає, що мережа може визначити значення символу, виходячи з його контексту. Наприклад, буква "o" у словах "one", "home" і "people" набуває різного значення в залежності від оточуючих символів — і саме це вміє враховувати RNN.

Серед найуспішніших варіантів RNN — мережі з довготривалою пам'яттю (LSTM) та блоками з контрольованим доступом (GRU). Вони усувають проблему швидкого забування попередніх станів, що характерна для класичних RNN, і здатні зберігати смисловий контекст навіть на рівні речення. Саме завдяки їм стало можливим формувати не просто символи, а слова, фрази та структуровані фрагменти тексту з урахуванням логіки.

Поєднання можливостей CNN і RNN у межах єдиної архітектури призвело до появи гібридних моделей, відомих під загальною назвою CRNN (Convolutional Recurrent Neural Network)^[34,23]. У таких системах згорткова частина відповідає за виділення просторових ознак, а рекурентна — за обробку послідовності, тобто контексту. Це дозволяє системі одночасно бачити візуальні особливості символів і розуміти їх місце у тексті.

CRNN добре працюють у задачах, де текст може мати неоднорідну структуру, змінювати орієнтацію або містити рукописні елементи. Наприклад, при обробці документів із колонками, нестандартним вирівнюванням або зсувом літер гібридна модель здатна утримувати цілісність розпізнавання. Такий підхід також зменшує залежність від попереднього етапу сегментації, який у класичних системах часто ставав джерелом помилок.

Наступним логічним кроком у розвитку нейронних моделей стало впровадження механізму уваги (attention), який дозволяє мережі зосереджуватися на найважливіших ділянках зображення під час аналізу^[39,11,3]. Це дає змогу не лише підвищити точність, а й покращити інтерпретованість

моделі: можна простежити, які саме фрагменти тексту вплинули на той чи інший результат.

На основі attention-механізму було створено Transformer-архітектуру, яка спочатку застосовувалась у задачах машинного перекладу, а згодом — і в OCR. На відміну від CNN або RNN, які обробляють зображення послідовно або локально, трансформери здатні аналізувати всю вхідну інформацію одночасно, враховуючи взаємозв'язки між усіма елементами тексту. Це дозволяє моделі краще розуміти структуру документа, навіть якщо він містить кілька мов, складні таблиці чи комбіновану верстку.

Однією з найуспішніших реалізацій такого підходу стала модель TrOCR від Microsoft, яка поєднує вхідне зображення з мовним декодером. Модель не потребує попередньої сегментації чи розмітки: вона автоматично визначає, де саме знаходиться текст, і генерує відповідну послідовність символів. У результаті — точне й контекстуально правильне розпізнавання навіть у складних документах.

Трансформери продемонстрували надзвичайно високі результати на відкритих бенчмарках, зокрема у задачах розпізнавання рукописного тексту, багатомовних даних і нестандартних форматів. Їх застосування стало вирішальним у створенні сучасних систем, які не просто зчитують текст, а й інтерпретують його зміст і функціональну структуру.

Глибокі нейронні мережі стали основою сучасних OCR-систем завдяки здатності навчатися з даних, враховувати контекст і працювати зі складними форматами. Цей підхід витіснив традиційні алгоритми й визначив новий напрям розвитку технології.

РОЗДІЛ 2. АРХІТЕКТУРИ НЕЙРОННИХ МЕРЕЖ ДЛЯ OCR

2.1. Конволюційні нейронні мережі (CNN) для обробки зображень

З огляду на обмеження традиційних методів OCR, описаних у попередньому розділі, сучасні системи дедалі частіше використовують архітектури глибокого навчання. Такі підходи дозволяють не лише розпізнавати окремі символи, а й враховувати їхній контекст, просторове розташування та взаємозв'язки в межах рядка чи всього документа. Еволюція технологій OCR передбачає поступовий перехід від статичних алгоритмів до гнучких моделей, здатних навчатися з даних і адаптуватися до складних вхідних умов.

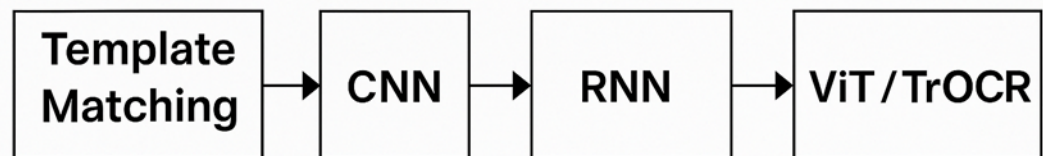


Рисунок 2.1 – Схема еволюції архітектур OCR

У центрі цієї трансформації опинилися конволюційні нейронні мережі (Convolutional Neural Networks, CNN) — архітектури, спеціально розроблені для роботи з візуальними даними^[15,21,22]. Основний принцип їхньої дії полягає в обробці зображення за допомогою фільтрів (або ядер згортки), що ковзають по піксельній матриці, виявляючи локальні патерни: лінії, кути, текстури. Кожен наступний шар у CNN занурюється глибше у структуру зображення, поступово абстрагуючись від простих форм до складних ознак. У результаті модель перетворює вхідне зображення на карту ознак, де зберігаються навіть найдрібніші деталі символів — від характерного вигину літери до особливостей накреслення.

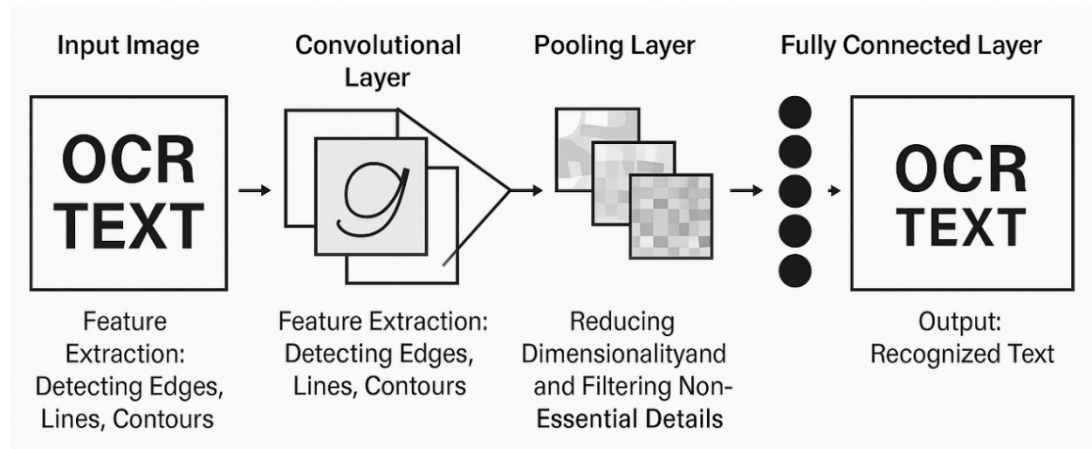


Рисунок 2.2 – Схема роботи CNN у задачах OCR

Ця властивість CNN особливо важлива для OCR-систем, адже дає змогу ідентифікувати літери навіть за наявності шумів, викривлень або незначних змін у шрифті. Наприклад, на зображенні, де текст розташований під кутом, або коли символи частково перекриваються, мережа здатна виявити інформативні регіони, які визначають клас символу. Така здатність виникає завдяки глибоким шарам згортки та операціям pooling, що зменшують розмірність, зберігаючи найважливіші просторові характеристики.

Однією з ключових переваг CNN у контексті розпізнавання тексту є інваріантність до зсувів, масштабування та деформацій. Завдяки ієрархічній структурі фільтрів, модель не шукає строго фіксоване положення символу на зображенні, а вчиться розпізнавати його форму незалежно від точного місця розташування чи розміру. Це особливо корисно при обробці документів, де текст може бути надрукований різними шрифтами, перекошений або містити артефакти сканування.

Важливою частиною архітектури CNN є шари згортки (convolutional layers), об'єднані з нелінійними функціями активації, які дозволяють моделі формувати складні абстракції з первинних піксельних даних^[35,15,18]. Операції згортки виділяють контрастні структури — краї, дуги, замкнуті фігури — які пізніше перетворюються на семантичні ознаки. Крім того, шари підвибірки (pooling layers), зокрема max pooling, виконують агрегацію ознак, виділяючи

найбільш характерні локальні фрагменти. Це не лише зменшує обчислювальну складність, але й робить модель більш стійкою до локальних зрушень чи спотворень вхідного зображення.

У типових реалізаціях CNN для OCR застосовуються архітектури типу LeNet, VGG, або їхні модифіковані версії. Наприклад, LeNet-5, одна з перших успішних CNN, використовувалася для розпізнавання рукописних цифр у системах сортування пошти^[22,17,35]. Сучасні реалізації ґрунтуються на значно глибших структурах — ResNet, EfficientNet, або MobileNet, що дозволяє досягати високої точності навіть у випадках зашумленого або нерівномірно освітленого тексту.

Окреме місце посідають легковагові архітектури, орієнтовані на мобільні або вбудовані пристрої. Наприклад, MobileNetV2 демонструє ефективність при низькому енергоспоживанні, що робить її придатною для OCR у смартфонах, ручних сканерах чи камерах доповненої реальності^[32,15,6]. У таких системах глибина мережі та розмір фільтрів оптимізовані для збереження швидкодії без істотних втрат у точності. З іншого боку, більш ресурсоемні моделі на кшталт EfficientNet або Inception використовуються в хмарних сервісах, де критично важлива точність обробки — наприклад, у багатомовному розпізнаванні або при цифровізації архівних документів.

Крім згорткових шарів, у сучасних CNN активно впроваджуються допоміжні техніки, які оптимізують процес навчання. Однією з таких є **batch normalization** — метод, який нормалізує вхідні дані кожного шару та стабілізує поширення градієнтів^[37,18]. Це суттєво зменшує час навчання і підвищує загальну стійкість мережі. Також використовується **dropout** — техніка регуляризації, яка випадковим чином «вимикає» частину нейронів під час навчання, змушуючи мережу не покладатися на окремі ознаки. У задачах OCR це дозволяє моделі узагальнювати знання, а не просто запам'ятовувати форму символів у навчальному наборі.

Однак навіть найточніші моделі не застраховані від помилок. У типових випадках CNN можуть плутати схожі символи: наприклад, «O» і «0», «l» і «I»,

«gn» і «n». Це зумовлено тим, що форма окремого символу часто недостатня для впевненої класифікації. Саме тому контекст у межах слова чи речення — критичний фактор, якого CNN, ізольовано взяті, не враховують.

Обмеження CNN пов'язані з їхньою локальною природою. Навіть при значній глибині мережі, рецептивне поле обмежене: модель фокусує увагу на фрагментах зображення, не охоплюючи його в цілому. Це стає критичним у випадках, коли значення символу залежить від контексту або позиції в слові. Наприклад, символ «u» у слові «sun» може мати схожу форму до «n» в слові «gun», і без урахування контексту помилка майже неминуча. CNN, на відміну від мовних моделей, не зберігають стан між обробкою окремих фрагментів, а отже, не здатні моделювати залежності між елементами послідовності.

Саме тому у сучасних OCR-системах CNN рідко використовуються як автономне рішення. Натомість вони слугують першою фазою — витягують зображення ознак, які далі аналізуються більш контекстно-орієнтованими архітектурами, такими як рекурентні нейронні мережі (RNN) або трансформери (Transformer). Це поєднання дозволяє зберігати як локальну точність, так і глобальне розуміння структури тексту.

Підсумовуючи, CNN є фундаментом сучасної обробки зображень у задачах OCR. Вони надзвичайно ефективні в екстракції візуальних ознак, навіть у складних умовах — при спотвореннях, нестандартному освітленні, зашумленості чи використанні незвичних шрифтів. Проте їхній потенціал розкривається повною мірою лише у поєднанні з архітектурами, здатними аналізувати послідовності та зберігати контекст. Наступний підрозділ розгляне саме такі моделі — рекурентні нейронні мережі, що доповнюють CNN у побудові повноцінних систем розпізнавання тексту.

2.2. Рекурентні нейронні мережі (RNN, LSTM) для послідовного тексту

Однією з принципових відмінностей текстових даних від зображень є їх послідовний характер. Символи у тексті розташовані у визначеному порядку, і саме ця лінійність формує зміст слів, речень та цілих абзаців. Для систем OCR це означає, що навіть за ідеального розпізнавання окремих літер загальний сенс

може бути втрачений, якщо не враховувати їхню послідовність. Наприклад, комбінація літер «b», «a» та «t» має зовсім інше значення в словах «bat» і «tab», хоча складається з тих самих символів. Така залежність від порядку створює потребу в моделях, здатних зберігати інформацію про попередні елементи під час аналізу поточних.

Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) були розроблені саме для роботи з такими типами даних. Їхньою основною перевагою є здатність враховувати попередній контекст^[16,7]. Кожен новий елемент послідовності обробляється не ізольовано, а з урахуванням інформації, отриманої на попередніх кроках. Це можливо завдяки внутрішньому стану мережі — так званому «прихованому стану» (hidden state), який зберігається і оновлюється на кожному кроці аналізу. Така пам'ять дозволяє мережі не просто реагувати на вхідні дані, а «згадувати», що було раніше, і використовувати цю інформацію для точнішої інтерпретації поточного символу.

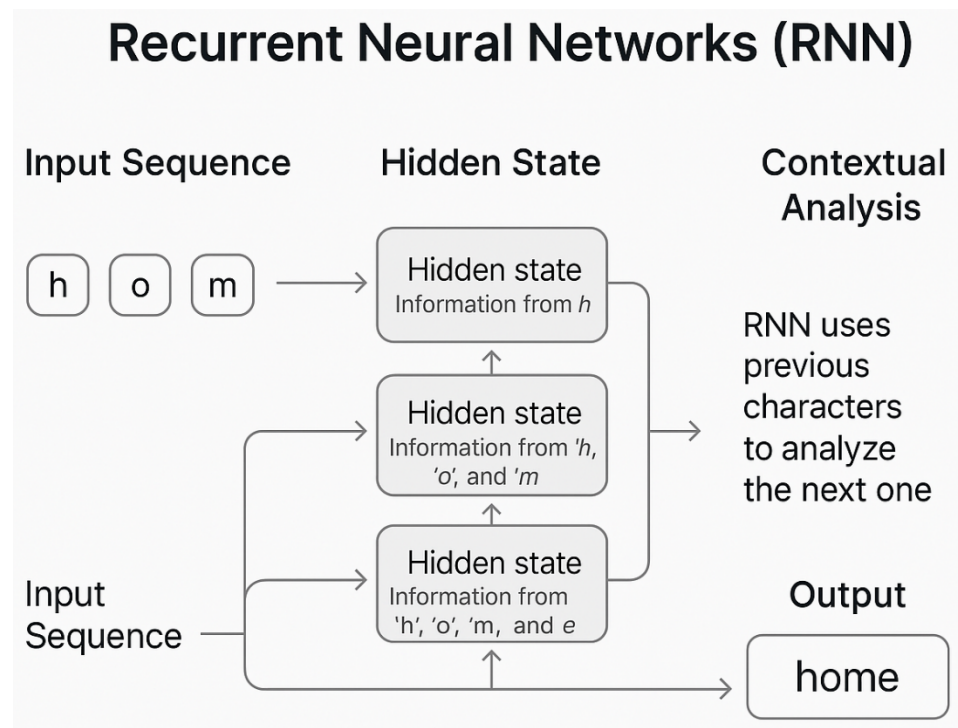


Рисунок 2.2 – Принцип роботи рекурентної нейронної мережі (RNN)

Цей принцип робить RNN надзвичайно ефективними у задачах розпізнавання тексту, де значення окремих літер часто залежить від їх

оточення. Особливо це помітно при роботі з пошкодженими або розмитими зображеннями, де деякі символи можуть бути нечіткими, а частина їхньої форми — втраченою. Якщо розглядати їх окремо, помилка є практично неминучою. Але якщо модель здатна врахувати попередній і наступний контекст — шанси на правильне розпізнавання значно зростають. Саме тому RNN розширюють можливості OCR-систем: вони дозволяють не лише обробляти візуальні сигнали, а й моделювати мовну логіку, подібно до того, як це робить людина під час читання.

Хоча ідея рекурентного збереження контексту здається надзвичайно перспективною, на практиці класичні RNN мають суттєві обмеження. Однією з головних проблем є так зване **затухання або вибух градієнтів** під час навчання. Цей феномен виникає, коли сигнали, що проходять крізь багато кроків часу, або зникають (стають надто малими), або навпаки — неконтрольовано зростають, що унеможливорює коректне оновлення ваг. У результаті мережа або не вчиться на далеких залежностях, або стає нестабільною під час навчання. Це особливо критично у випадках, коли для правильного розпізнавання поточного символу потрібно знати, що було декілька символів назад — як у складних словах чи фразах.

Для подолання цих труднощів було запропоновано модифікації базової архітектури, серед яких найвідомішою і найуспішнішою стала **довготривала короткочасна пам'ять** (Long Short-Term Memory, LSTM)_[16,13]. На відміну від класичної RNN, яка передає лише єдиний прихований стан, LSTM має більш складну структуру, що включає кілька спеціалізованих механізмів: так звані «вікна» або «вентилі» (gates). Вони дозволяють контролювати, яка саме інформація зберігається, яка відкидається, а яка передається далі. Це дає змогу мережі вибірково фокусуватись на релевантних даних, навіть якщо вони були отримані багато кроків назад.

У практичному застосуванні до OCR це має вирішальне значення. Наприклад, при розпізнаванні зашумлених документів або рукописного тексту,

LSTM здатна утримувати контекст попередніх літер або частин слова, що дозволяє відновити розмиті чи частково втрачені символи. Це фактично імітує роботу людського читача: навіть якщо одна буква стерта або зіпсована, мозок доповнює її на основі всього слова або речення. Таким чином, архітектура LSTM значно розширила можливості систем OCR щодо роботи зі складними текстовими шаблонами, вивівши точність на якісно новий рівень.

Згодом на основі LSTM була розроблена спрощена, але також ефективна модифікація — **GRU (Gated Recurrent Unit)**. Ця архітектура має менш складну внутрішню структуру, оскільки об'єднує деякі вентиля, зменшуючи кількість параметрів і прискорюючи обчислення^[7]. Попри це, GRU зберігає здатність до навчання довготривалих залежностей, і в багатьох OCR-сценаріях демонструє результати, зіставні з LSTM. Вибір між LSTM та GRU зазвичай залежить від вимог до швидкодії та складності розпізнання: якщо потрібна компактність і швидкість — перевагу надають GRU; якщо критична точність у складних умовах — частіше використовують LSTM.

На практиці LSTM-мережі стали основою для низки систем оптичного розпізнавання, які вимагають аналізу символів у часовій або послідовній формі. Наприклад, у таких open source-інструментах, як Tesseract (у версії 4.0+), CNN використовується для витягування просторових ознак, а RNN — для обробки їх у контексті рядка^[13,5,34]. Аналогічний принцип використовується в Google Cloud Vision OCR, де LSTM дозволяє зберігати залежності в довгих текстових блоках, особливо у складно форматуваних документах.

Ще один важливий аспект застосування RNN — **робота з рукописним текстом**, де контекст відіграє навіть більшу роль, ніж у надрукованому. Рукописні символи значно варіативніші, і навіть одна літера, написана двома різними людьми, може виглядати геть інакше^[10,29,28]. LSTM дозволяє OCR-системі не лише класифікувати кожен символ, а й прогнозувати, який символ найімовірніше мав би бути на тому чи іншому місці, виходячи з контексту рядка. Це особливо ефективно при роботі з історичними архівами, де тексти часто пошкоджені або містять нестандартні елементи почерку.

У реальних застосуваннях LSTM інтегруються з іншими модулями — передусім із CNN — для побудови гібридних архітектур. Саме ця комбінація лягла в основу CRNN (Convolutional Recurrent Neural Network), де згорткові шари відповідають за витягування візуальних ознак, а рекурентні — за обробку послідовності. Однак, навіть у гібридних рішеннях, саме LSTM забезпечує здатність моделі «розуміти» текст у динаміці.

Рекурентні архітектури демонструють особливу ефективність при розпізнаванні **довгих рядків тексту**, де класичні CNN або ізольовані класифікатори символів зазнають труднощів. У таких випадках LSTM дозволяє моделі будувати контекст на рівні всього рядка або навіть абзацу. Наприклад, при обробці юридичних або наукових документів, де окремі символи можуть повторюватись у різному контексті, система, що спирається лише на візуальні ознаки, легко припускається помилок. Натомість LSTM дозволяє встановити, що символ, схожий на «e», в конкретному слові є частиною латинського терміна, а не випадковою літерою.

Особливо це актуально у випадках, коли текст містить **омоніми або неоднозначні послідовності**, які можна трактувати по-різному без врахування контексту. Наприклад, послідовність «read» може позначати як теперішній, так і минулий час дієслова, і лише в контексті речення можливо визначити правильне значення. У таких сценаріях моделі на основі LSTM здатні враховувати більш широкий контекст, утримуючи інформацію не лише про попередні символи, а й про семантичні зв'язки між ними.

У сучасних OCR-системах рекурентні моделі часто доповнюються або інтегруються з механізмами **уваги (attention)**, які дозволяють фокусуватися на найбільш релевантних частинах послідовності. Хоча механізм attention найбільше відомий у контексті трансформерів, його перші реалізації виникли саме в поєднанні з LSTM. Завдяки attention модель може, умовно кажучи, «зазирнути назад» у рядок і обрати саме ті фрагменти, які найбільше впливають на поточне рішення. Наприклад, у фразі «The contract was signed by...», слово

«signed» підказує, що далі слідує іменник, і LSTM з attention здатна передбачити це граматичне очікування.

Наразі багато промислових OCR-рішень переходять до transformer-архітектур, проте саме рекурентні мережі стали першим кроком до контекстуалізації тексту. Їх внесок у точність, особливо у випадках зі складними послідовностями, є незаперечним. Саме тому LSTM донині залишаються компонентом багатьох гібридних моделей, навіть у тих системах, де застосовуються TrOCR або ViT як основа.

Окрім обробки рядків, RNN та LSTM також демонструють ефективність у розпізнаванні **багаторядкових структур тексту**, таких як абзаци, таблиці або заповнені форми. У таких випадках модель має справу не лише з послідовністю символів, а й із просторовою організацією, яка додає додаткового рівня складності. Наприклад, під час аналізу рахунків-фактур або таблиць з невіривняними колонками, традиційна CNN-фільтрація не дозволяє точно інтерпретувати логіку заповнення клітинок. Тут на допомогу приходить багаторівнева LSTM-модель, яка може працювати з кожним рядком окремо, зберігаючи контекст у межах усієї структури.

Цей підхід знайшов широке застосування у фінансовому секторі, де автоматизоване розпізнавання документів вимагає точного відтворення порядку і вмісту. LSTM дозволяє моделі визначати, де закінчується один логічний блок і починається інший, навіть якщо візуально ці блоки виглядають майже однаково. Крім того, такі мережі здатні розпізнавати повторювані патерни — наприклад, однакове форматування дат, сум або ідентифікаторів — що також підвищує загальну точність OCR-системи.

Ще одним цікавим напрямом використання RNN є **модельне передбачення помилок**, тобто автоматичне виявлення місць, де символи ймовірно розпізнані некоректно. Це особливо актуально у великих системах документообігу, де кожна помилка може призвести до критичних наслідків. RNN-модель, навчена на типових помилках (наприклад, плутанина між «0» і

«О» або «I» і «l»), здатна не лише виявити невідповідність у контексті, а й запропонувати виправлення на основі імовірнісної оцінки.

Попри всі переваги, RNN і LSTM мають і свої обмеження. Найпомітніше — це обчислювальна вартість, особливо при обробці довгих послідовностей. Через свою послідовну природу (один крок за одним) такі моделі складно паралелізувати, що впливає на швидкість виконання, особливо при розпізнаванні великих обсягів документів у реальному часі. Саме це стало однією з причин переходу індустрії до transformer-підходів, які вирішують проблему швидкодії за рахунок механізмів глобальної уваги.

У відповідь на ці обмеження в багатьох сучасних OCR-системах рекурентні мережі інтегруються з іншими архітектурами, утворюючи **гібридні моделі**, які поєднують сильні сторони кожного підходу. Однією з найуспішніших реалізацій стала CRNN (Convolutional Recurrent Neural Network) — модель, яка складається з трьох основних компонентів: згорткової частини (CNN) для виділення ознак, рекурентної частини (RNN або LSTM) для обробки послідовностей, та вихідного шару, що виконує класифікацію. Такий підхід дозволяє системі не лише «бачити» зображення, а й «розуміти» його зміст у контексті, що критично важливо для точного розпізнавання тексту.

Використання LSTM у складі CRNN дає змогу значно покращити якість обробки довгих рядків, багатомовних фрагментів або нетипових форматів. Наприклад, у розпізнаванні номерів паспортів, ID-карт або складних табличних структур CRNN із рекурентним блоком демонструє вищу стійкість до викривлень та втрати даних порівняно з моделями, які покладаються лише на згорткові ознаки. Важливо, що ця архітектура зберігає гнучкість: її можна адаптувати до конкретних завдань, варіюючи глибину CNN, тип рекурентного шару (LSTM або GRU) та функції втрат, зокрема CTC (Connectionist Temporal Classification), яка дозволяє зіставляти вхідну послідовність із мітками без жорсткої прив'язки до позицій.

Таким чином, RNN та LSTM стали ключовим етапом в еволюції OCR — першим серйозним кроком від ізольованої класифікації символів до

контекстного аналізу тексту. Вони довели свою ефективність у задачах, де значення елементів визначається не лише формою, а й місцем у послідовності. Попри зростаючу популярність трансформерів, LSTM і далі відіграють важливу роль, зокрема в гібридних рішеннях або в сценаріях, де ресурси обмежені, а швидкість обробки менш критична.

Наступний підрозділ буде присвячений цим гібридним моделям — зокрема архітектурі CRNN, яка об'єднує згорткову та рекурентну частини в єдину послідовну систему. Саме вона стала основою багатьох комерційних OCR-рішень, що поєднують точність, контекстуальність і адаптивність до складних вхідних даних.

2.3. Гібридні моделі (CRNN – CNN+RNN)

Хоча згорткові та рекурентні нейронні мережі окремо вже демонструють високу ефективність у задачах оптичного розпізнавання символів, саме їх поєднання дозволило досягти нового рівня точності та надійності^[34,23]. CNN чудово витягують локальні ознаки зі зображень, а RNN зберігають послідовні залежності в тексті. Проте в ізоляції ці компоненти мають обмеження: CNN не враховують послідовність, а RNN — не здатні ефективно працювати з візуальними даними без попередньої структуризації. Це створило передумови для появи гібридних архітектур, серед яких найбільш відомою стала CRNN — модель, що об'єднує переваги обох підходів.

Основна ідея полягає в тому, щоб **розділити завдання обробки тексту на два етапи**. На першому — згорткові шари CNN аналізують вхідне зображення і перетворюють його на карту ознак: багатовимірне представлення, що зберігає ключові візуальні характеристики кожного символу. На другому етапі RNN отримує цю карту ознак як послідовність і аналізує її в динаміці, зберігаючи контекст між символами, словами або навіть частинами речень. Таким чином, CRNN працює як конвеєр, у якому CNN виконує роль «зору», а RNN — роль «пам'яті» та «мовного розуміння».

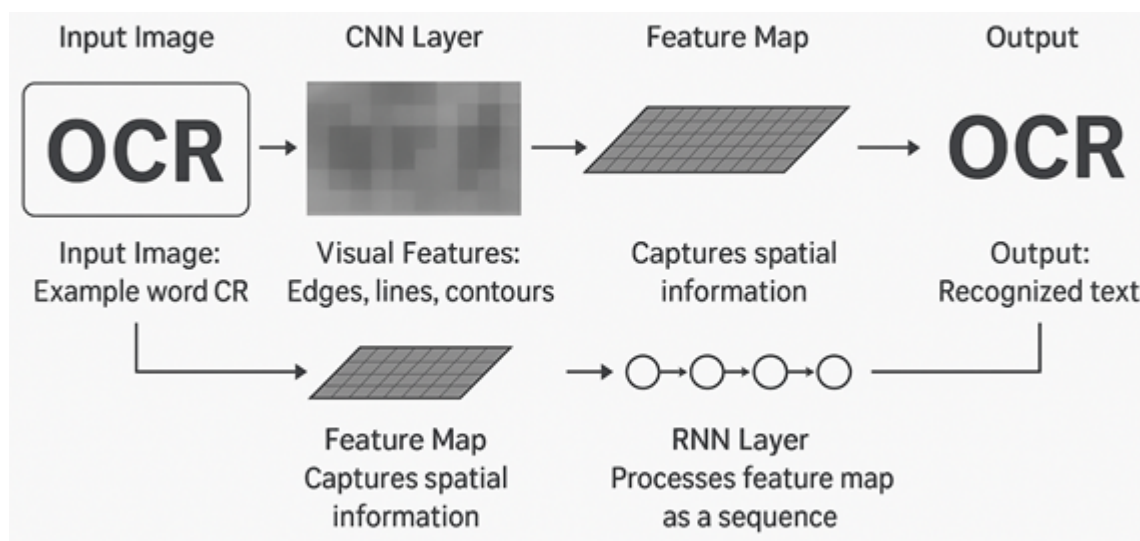


Рисунок 2.3 – Принцип роботи CRNN

Ця стратегія дає змогу подолати обмеження традиційного підходу, де кожен символ розпізнавався окремо, без урахування його позиції у слові. Замість ізольованого аналізу, CRNN дозволяє обробляти **рядок цілком**, як єдину послідовність^[2,34]. Це особливо важливо при обробці довгих слів, тексту з неідеальною версткою або рукописного тексту, де межі між символами розмиті. У таких випадках CRNN здатна інтерпретувати зображення з урахуванням загальної структури та логіки тексту, значно знижуючи кількість помилок.

Загальна архітектура CRNN є модульною: вона складається зі згорткової частини (найчастіше від 3 до 7 шарів), рекурентної частини (зазвичай два двонапрямлені LSTM-шари) і вихідного шару, який відповідає за класифікацію. Під час проходження даних мережею, розмірність просторової карти ознак перетворюється у часову послідовність, яка подається на вхід LSTM. Цей процес дозволяє перенести просторову інформацію у часову площину, відкриваючи шлях до точного, контекстно-обґрунтованого розпізнавання.

Однією з найважливіших переваг CRNN є здатність **зберігати просторову структуру тексту** без необхідності попереднього розрізання на окремі символи^[23,34]. У класичних системах OCR часто застосовувався етап сегментації, де кожна літера повинна була бути чітко відділена від інших. Проте цей підхід виявляється вразливим до варіативності шрифтів, накладень

символів, нахилів і перекосів. У CRNN потреба в сегментації відсутня: мережа аналізує всю послідовність зображення, спираючись на власне уявлення про межі між символами. Це значно підвищує стабільність системи, особливо у випадках з рукописами або документами, що містять артефакти сканування.

Ще один вагомий аргумент на користь CRNN — **інваріантність до змін масштабу та орієнтації тексту**. Завдяки згортковим шарам мережа автоматично адаптується до різної висоти символів, а рекурентна частина зберігає зв'язки між елементами навіть у випадках деформацій. Наприклад, при обробці фотографії квитанції, зробленої з нахилом, модель може коректно прочитати рядки, які з точки зору піксельної структури мають неоднорідну геометрію. У той час як окремо взяті CNN або RNN дали б неповний або спотворений результат, їх комбінація забезпечує надійне зчитування інформації з усього зображення.

CRNN також демонструє високу точність при роботі з **довгими рядками без чітких інтервалів**, що характерно для рукописного тексту або нестандартизованих документів. У таких випадках RNN-компонент дозволяє врахувати не лише сусідні символи, а й віддалені фрагменти, створюючи повноцінну послідовну модель тексту. Це дає змогу мережі «прогнозувати» наступні символи на основі контексту, навіть якщо частина тексту втрачена або перекрита артефактами. Таке «контекстне домислювання» є надзвичайно цінним у практичних застосуваннях OCR — наприклад, у випадках із пошкодженими архівними документами або фотознімками з умовно читабельними фрагментами.

Варто зазначити, що CRNN також добре працює з **різноманітними алфавітами**, включаючи латиницю, кирилицю, арабське письмо та навіть ієрогліфіку. Завдяки універсальній структурі моделі, достатньо підготувати відповідний датасет і здійснити донавчання, щоб адаптувати систему до нової мови або письма. Це суттєво розширює можливості застосування CRNN в багатомовних або міжнародних OCR-сервісах.

Гібридна архітектура CRNN знайшла практичне втілення в низці сучасних систем OCR, як академічних, так і комерційних. Одним із перших прикладів масштабного впровадження стала **четверта версія Tesseract OCR**, яка суттєво відрізняється від попередніх тим, що використовує саме CRNN як основну модель^[5,36]. У цьому випадку згорткова частина витягує просторові ознаки з вхідного зображення, а двонапрямлена LSTM-модель аналізує ці ознаки як послідовність. Вихідна частина Tesseract 4 побудована на основі функції втрат CTC (Connectionist Temporal Classification), яка дозволяє зіставляти змінну кількість вхідних кадрів із символами тексту без жорсткої сегментації.

Використання CTC у поєднанні з CRNN є важливою складовою, яка забезпечує високу гнучкість моделі. Зокрема, вона дозволяє мережі вчитися вирівнювати вхідну послідовність ознак (отриманих з CNN) із мітками символів, навіть якщо ці послідовності мають різну довжину^[13,34]. Це знімає необхідність точно знати позицію кожного символу в зображенні, що надзвичайно корисно у випадках із деформованим або непередбачувано структурованим текстом. Завдяки цьому CRNN з CTC досягає вражаючої адаптивності до нових умов — її можна навчити не лише на друкованому тексті, а й на почерку, відсканованих нотатках, або навіть написах на об'єктах у реальному середовищі.

Ще одним прикладом ефективного впровадження CRNN є **PaddleOCR** — відкритий OCR-фреймворк, створений компанією Baidu. Його структура базується на багаторівневій CRNN-моделі, яка спочатку перетворює зображення у послідовність ознак, а потім аналізує її за допомогою багатосарової двонапрявленої RNN. PaddleOCR підтримує понад 80 мов і дозволяє досягати високої точності розпізнавання навіть на малих вибірках, що робить його цінним інструментом для локалізованих застосунків.

У комерційному секторі аналогічні принципи реалізовано у **Google Cloud Vision API**, який також використовує комбінацію згорткових і рекурентних мереж. Завдяки цьому сервіс здатен обробляти як звичайні документи, так і

складні сцени, зокрема текст на вивісках, етикетках, упаковках. Особливо важливо, що такі системи не просто розпізнають символи, а й можуть зберігати логічну структуру сторінки: відображення рядків, абзаців, таблиць. Це стало можливим завдяки саме гібридному підходу, що враховує як просторову, так і послідовну природу тексту.

Порівняно з моделями, які базуються виключно на CNN або RNN, гібридна архітектура CRNN має **більш збалансований функціонал**, що дозволяє компенсувати слабкі сторони кожного з компонентів. Якщо CNN переважно працює з локальними патернами, ігноруючи послідовні зв'язки, а RNN не здатна самотійно витягувати візуальні ознаки, то їх поєднання дозволяє будувати повноцінне представлення тексту на всіх рівнях — від піксельного до лінгвістичного. Саме це і робить CRNN універсальним інструментом для OCR, здатним адаптуватися до широкого спектра застосувань: від читання пошкоджених історичних документів до аналізу зображень у реальному часі на мобільних пристроях.

До переваг CRNN варто також віднести її **відносну простоту реалізації**: у більшості фреймворків глибокого навчання доступні попередньо налаштовані компоненти згорткових і рекурентних шарів, які легко комбінуються в єдину модель. Це дає змогу створювати потужні OCR-системи навіть без глибокої спеціалізації в проєктуванні архітектур. Водночас CRNN достатньо гнучка, щоб масштабуватись як у напрямку підвищення точності (збільшенням глибини мережі), так і в бік оптимізації для швидкодії (наприклад, заміною LSTM на GRU або використанням мобільних версій CNN).

Однак слід зазначити, що CRNN, попри свої переваги, має певні обмеження. Найважливіше з них — це **послідовний характер обробки у RNN-компоненті**, який ускладнює паралелізацію та зменшує швидкість на великих об'ємах тексту. У поєднанні з потребою в попередній нормалізації вхідного зображення (зокрема вирівнюванні текстових рядків по горизонталі), це створює передумови для подальшого вдосконалення. Саме цим шляхом пішли нові покоління OCR-моделей, зокрема трансформерні архітектури

(Transformer), які здатні аналізувати текст у глобальному контексті, одночасно обробляючи всю послідовність.

Підсумовуючи, CRNN стала **перехідним етапом між класичними та сучасними глибокими підходами** в OCR. Вона об'єднала здатність CNN до візуальної абстракції та силу RNN у збереженні контексту, відкривши шлях до глибшого, контекстно-чутливого розпізнавання тексту. І хоча на горизонті з'явилися ще потужніші моделі, CRNN досі залишається надійним вибором для широкого кола практичних задач.

2.4. Transformers для OCR (Vision Transformer, TrOCR)

Попри високу ефективність гібридних моделей на базі CNN і RNN, їхня архітектура має певні вбудовані обмеження. Насамперед, рекурентна частина таких систем обробляє дані послідовно — один елемент за іншим. Це ускладнює паралелізацію і знижує продуктивність при роботі з довгими послідовностями, особливо у випадках, коли потрібно аналізувати не лише окремі рядки, а весь документ як цілісну структуру. Крім того, рекурентні мережі не мають прямого механізму для вибіркового фокусування на важливих фрагментах вхідних даних. Вони «пам'ятають» інформацію про попередні елементи, але не завжди здатні виділити найрелевантніші — саме ті, що справді мають значення в поточному контексті.

Ці проблеми стали каталізатором для появи **трансформерних архітектур**, які вперше продемонстрували ефективність у задачах машинного перекладу, а згодом стали основою для майже всіх сучасних мовних моделей. Їхнє ключове нововведення — **механізм уваги (attention)**, який дає змогу моделі динамічно оцінювати, які частини вхідної послідовності найбільш важливі для поточного рішення. Завдяки цьому трансформери не лише зберігають контекст, як RNN, але й роблять це більш гнучко, не залежачи від порядку обробки елементів. У результаті вони здатні враховувати як локальні, так і глобальні зв'язки в тексті — не лише в межах одного рядка, а й у масштабі всього документа^[3,11,33].

У сфері OCR поява трансформерів означає **перехід до нового рівня аналізу тексту** — від пострядкового до міжрядкового, від обробки окремих символів до інтерпретації документа як цілісної семантичної структури. Якщо раніше модель бачила лише невеликий фрагмент тексту у вікні згортки або послідовності, то тепер вона отримує доступ до повної картини. Саме це відкриває можливості для роботи зі складними багаторівневими документами: юридичними договорами, технічними інструкціями, багатомовними PDF-файлами. Трансформерні архітектури дають змогу не лише розпізнати, а й зрозуміти взаємозв'язки між заголовками, підписами, абзацами та таблицями.

Однією з перших трансформерних моделей, адаптованих для обробки зображень, став **Vision Transformer (ViT)**_[12,38]. Ця архітектура перенесла ідею self-attention із мовної сфери на візуальні дані, замінивши згорткові шари на багатоголову увагу, що працює з просторовими фрагментами зображення. На відміну від CNN, які послідовно розширюють поле огляду вглиб мережі, ViT із самого початку надає моделі глобальну перспективу. Зображення ділиться на фіксовані патчі (наприклад, 16×16 пікселів), кожен з яких лінеаризується та кодується позиційною інформацією, після чого всі вони обробляються в один прохід. Це дозволяє моделі бачити структуру всього зображення з першого шару, а не лише локальні фрагменти.

В OCR-сценаріях Vision Transformer демонструє переваги в роботі з **цілісними документами**, де критично важливо аналізувати просторове розміщення текстових блоків, а не лише символів чи слів. Наприклад, у сканованих формах або сторінках журналів інформація може бути розташована в кількох стовпцях, містити підписи, колонтитули, таблиці. ViT дозволяє моделі одночасно враховувати всі ці елементи, визначаючи не лише текст, а й його функціональне місце у структурі документа.

Ключовою перевагою ViT у контексті OCR є **паралельність обчислень**: оскільки модель не обробляє фрагменти по черзі, як RNN, вона може аналізувати всю послідовність одночасно. Це суттєво підвищує продуктивність і дозволяє масштабувати модель на великі обсяги документів. Крім того, на

відміну від згорткових мереж, які мають обмежене рецептивне поле та ризикують втратити глобальний контекст, ViT із self-attention враховує залежності між будь-якими частинами зображення. Це відкриває шлях до точного розпізнавання складних візуальних шаблонів, таких як багаторядкові заголовки, вставки або зноски, які впливають на логіку прочитання тексту.

У поєднанні з мовними моделями трансформери типу ViT можуть стати основою для систем, здатних не лише прочитати документ, а й інтерпретувати його структуру — наприклад, визначити, де починається розділ, що є заголовком, а що — підписом до таблиці. Така гнучкість зробила ViT популярною базою для багатьох сучасних OCR-підходів, зокрема TrOCR.

Однією з найпоказовіших трансформерних моделей у сфері OCR є **TrOCR** — відкритий проєкт компанії Microsoft, який інтегрує Vision Transformer з мовним декодером у стилі sequence-to-sequence_[11,27]. На відміну від попередніх архітектур, де обробка зображення і тексту відбувалася окремо, TrOCR реалізує **єдину енд-ту-енд модель**, що приймає зображення на вхід і повертає текст на виході, без потреби в додатковій перед- або постобробці.

TrOCR складається з двох основних компонентів. Перший — це **візуальний енкодер**, зазвичай побудований на базі ViT, який трансформує зображення в послідовність векторів ознак_[17,27,38]. Ці вектори далі передаються у **мовний декодер**, що працює за стандартною трансформерною схемою, подібною до GPT або BART. Декодер покликаний відновити текст на основі візуального вхідного сигналу, враховуючи лінгвістичні закономірності та контекст. Це дозволяє TrOCR не просто «читати» символи, а **розуміти текст у лінгвістичному сенсі** — враховуючи граматику, словоформи, семантичну послідовність. На схемі нижче показано, як трансформер фокусує увагу на важливих зонах зображення при розпізнаванні.

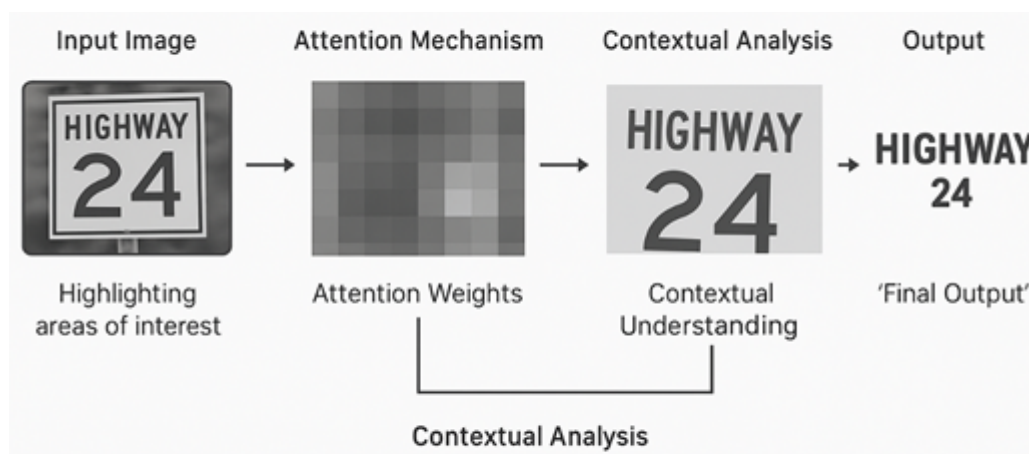


Рисунок 2.4 – Робота механізму уваги в трансформерній OCR-моделі

Така двокомпонентна архітектура забезпечує виняткову **гнучкість і точність**. Поєднання потужного візуального блокового аналізу з можливостями мовної генерації дозволяє системі адаптуватися до найрізноманітніших форматів даних — від друкованих документів до рукописів, від однорядкових підписів до багатосторінкових PDF-файлів. У TrOCR немає потреби в сегментації зображення на символи або слова, немає необхідності вирівнювати рядки чи нормалізувати висоту літер: модель вчиться інтерпретувати зображення у його повному вигляді, як людина.

Практична ефективність TrOCR підтверджується експериментальними результатами на кількох стандартних датасетах, таких як IAM, SROIE, FUNSD. Зокрема, на задачах розпізнавання форм і документів зі складною структурою TrOCR перевершує CRNN і навіть деякі комбіновані системи з attention-після-RNN. Однією з найважливіших її переваг є здатність **масштабуватись без втрати якості**: збільшення розміру трансформера веде до стабільного зростання точності, що не завжди спостерігається в RNN-базованих моделях, де перенавчання виникає швидше.

Порівнюючи трансформерні моделі з гібридними архітектурами типу CRNN, можна виокремити кілька принципових переваг_[11,27,34]. По-перше, **self-attention** забезпечує глобальний контекст з перших рівнів обробки, на відміну від рекурентних мереж, де залежності між далекими елементами

слабшають зі збільшенням відстані. Це особливо важливо для аналізу багатосторінкових документів, де логічна єдність може зберігатися між абзацами на різних сторінках. По-друге, трансформери обробляють дані **паралельно**, що значно прискорює навчання і дозволяє застосовувати їх у високопродуктивних системах — зокрема, в хмарних OCR-сервісах або при обробці великих архівів.

Ще однією перевагою є **універсальність архітектури**. Якщо CRNN — це строго лінійна комбінація CNN + RNN, де кожен блок має свою чітку функцію, то трансформери допускають більшу гнучкість: той самий self-attention механізм використовується і в енкодері, і в декодері, причому у взаємодії між різними модальностями (візуальна + текстова). Це робить трансформери не лише OCR-інструментом, а й базою для мультизадачних моделей, які можуть одночасно аналізувати текст, зображення, структуру, навіть змістовий зв'язок між ними. Наприклад, на основі TrOCR можливо побудувати систему, яка не просто зчитує документ, а й класифікує його тип або витягує ключові поля.

Однак варто згадати і про виклики. Трансформери вимагають **суттєвих обчислювальних ресурсів**, як на етапі навчання, так і під час інференсу. Це може бути критичним обмеженням для мобільних або вбудованих систем. Крім того, успішне застосування моделей типу TrOCR потребує **великих анотованих датасетів**, бажано багатомовних і структурно складних, що не завжди доступно для локалізованих рішень або рідковживаних мов.

Підсумовуючи, трансформерні моделі на кшталт ViT та TrOCR відкрили нову еру в OCR — еру контекстно-чутливого, семантично обґрунтованого розпізнавання. Вони демонструють високу точність, здатність до генералізації, швидкість і адаптивність до складних форматів документів. І хоча вони не завжди замінюють CRNN у задачах з обмеженими ресурсами, саме трансформери наразі є **фронтиром розвитку OCR-систем**, прокладаючи шлях до інтеграції в ширші системи аналізу документів — від класифікації до автоматичного заповнення форм.

2.5. Порівняння точності та швидкості різних архітектур

Ефективність OCR-систем визначається не лише здатністю точно розпізнавати текст, а й швидкістю роботи, стійкістю до шумів, гнучкістю у складних умовах^[2,23,27]. У сучасній практиці існує декілька провідних підходів до побудови моделей розпізнавання: згорткові нейронні мережі (CNN), рекурентні архітектури (RNN, LSTM), гібридні рішення (CRNN) та трансформери (ViT, TrOCR). Кожен із цих підходів має свої сильні сторони та обмеження. Для коректного вибору архітектури в залежності від завдання важливо не лише порівняти їх за абсолютною точністю, а й оцінити **продуктивність, адаптивність до нових даних, швидкодію та вимоги до ресурсів**.

Згорткові нейронні мережі вирізняються надзвичайною **швидкістю обробки зображень**, особливо якщо мова йде про прості задачі, як-от класифікація окремих символів або обробка рядків з фіксованою структурою^[22,21,15]. Завдяки паралельності обчислень і оптимізованим GPU-реалізаціям CNN можуть розпізнавати сотні зображень у секунду. У контексті точності, CNN добре працюють у стабільних умовах: чіткий друкований текст, стандартні шрифти, рівномірне освітлення. Проте як тільки вхідне зображення стає складнішим — з перекошеним або рукописним текстом, нечіткими межами символів чи змішаними шрифтами — точність стрімко падає. Це зумовлено відсутністю механізму врахування послідовності, оскільки CNN обробляють зображення фрагментарно, не розрізняючи логічних залежностей між елементами тексту.

Рекурентні нейронні мережі, зокрема моделі LSTM, демонструють **вищу точність при роботі з послідовностями**, особливо у випадках, коли розпізнавання окремих символів є неоднозначним. Завдяки здатності зберігати контекст між символами, RNN ефективно справляються з розпізнаванням рукописного тексту або фрагментів, де частина символів спотворена^[7,13,16]. Наприклад, при роботі з історичними архівами або документами, відсканованими з втратами, RNN здатні "домислювати" втрачені символи, враховуючи попередній контекст. Проте зворотною стороною є **низька**

швидкодія: рекурентні моделі обробляють дані послідовно, що унеможливлює повноцінну паралелізацію і уповільнює інференс на великих масивах даних.

Гібридні моделі на зразок **CRNN** були створені саме для того, щоб поєднати переваги згорткових і рекурентних мереж, мінімізуючи їхні недоліки. Завдяки CNN-компоненту модель швидко витягує візуальні ознаки з усього зображення, а RNN-частина дозволяє враховувати послідовні зв'язки між ними_[2,23,34]. У результаті CRNN демонструє **високу точність при збереженні помірної продуктивності**, особливо в тих задачах, де структура тексту є складною, але ще не вимагає аналізу всього документа в цілому.

Наприклад, у розпізнаванні друкованих рахунків, форм або квитанцій, де важлива як локальна точність (цифри, дати, суми), так і послідовний контекст (рядок як логічна одиниця), CRNN показує себе як **оптимальний компроміс**. Вона швидша за чисті RNN, оскільки рекурентна обробка застосовується не до всього зображення, а лише до згорткового представлення. Водночас вона точніша за чисті CNN, бо може інтерпретувати контекст і зв'язки між символами. CRNN також добре масштабується — її можна урізати для мобільних додатків або, навпаки, поглибити для великих хмарних рішень.

Щодо **швидкості інференсу**, CRNN знаходиться між CNN і RNN. При типових конфігураціях (3–5 згорткових шарів + 2 двонапрямлені LSTM-шари) модель обробляє документи зі швидкістю, придатною для реального часу, зокрема на GPU. Однак при значному збільшенні глибини або довжині текстових послідовностей час обробки зростає. Варто також враховувати, що CRNN, як і RNN, має обмеження у паралельності через послідовну природу рекурентного блоку. Тому при роботі з дуже великими документами або в системах із суворими обмеженнями по часу (наприклад, у відеоаналізі або доповненій реальності), її ефективність може знижуватись.

З огляду на вищезазначене, CRNN є **золотим стандартом** для багатьох промислових систем OCR, що працюють зі стандартними форматами документів: рахунками, друкованими текстами, відсканованими бланками. Вона забезпечує прийнятну швидкість і стабільну точність при достатньо

низьких вимогах до обчислювальних ресурсів. Однак із зростанням складності задачі — багаторівнева структура документа, змішана мова, наявність вставок, таблиць, діаграм — виникає потреба в моделях із більшим «розумінням» контексту. Саме тут починають домінувати трансформери.

Трансформерні архітектури, зокрема Vision Transformer (ViT) та TrOCR, стали проривом у точності OCR, особливо у випадках, коли документ є складним не лише з точки зору зображення, а й логіки структури^[12,27,38,39]. Завдяки механізму **self-attention** такі моделі мають змогу одразу охоплювати глобальний контекст і зосереджуватись на релевантних фрагментах зображення незалежно від їхнього розташування. Це особливо корисно при обробці **багатосторінкових або багаторівневих документів**, де важливо виявити зв'язки між заголовками, підписами, таблицями чи коментарями, які не мають лінійної структури.

З погляду **точності**, трансформери перевершують CRNN і RNN у майже всіх завданнях, де присутній складний контекст або неоднозначні символи. Вони демонструють особливо високі результати на датасетах типу SROIE (складні рахунки), FUNSD (формати з полями), а також при багатомовному розпізнаванні (наприклад, при наявності латиниці, кирилиці та арабського письма в одному документі). У завданнях, де потрібно не лише зчитувати текст, а й **інтерпретувати його семантично**, трансформери стали **незаперечним лідером**.

Втім, така висока точність має свою ціну. Трансформерні моделі вимагають **значних обчислювальних ресурсів**, особливо на етапі навчання. Їхня архітектура містить велику кількість параметрів і не є оптимізованою для мобільних або енергоефективних систем. Під час інференсу вони значно повільніші, ніж CNN або CRNN, особливо без апаратного прискорення. Навіть за допомогою оптимізацій, таких як quantization або distillation, TrOCR все ще поступається CRNN за швидкістю в реальному часі. Тому в задачах із високим навантаженням на систему або жорсткими обмеженнями по часу трансформери поки що менш придатні, ніж «легші» моделі.

Крім того, трансформерні архітектури мають **високу чутливість до якості навчального датасету**. Якщо CNN і навіть RNN можна навчити на порівняно обмежених наборах із ручною розміткою, то для досягнення повного потенціалу TrOCR потрібні **великі, різноманітні, граматично багаті датасети**, бажано з реальними документами й візуально різноманітними прикладами. Це ускладнює їх застосування в умовах, коли дані є конфіденційними або важкодоступними.

Підсумовуючи проведений огляд, можна виділити ключові **сильні та слабкі сторони кожної архітектури** в контексті задач OCR:

CNN вирізняються швидкістю, паралельністю обчислень і ефективністю в умовах, де структура тексту є простою та передбачуваною. Проте вони втрачають точність у випадках з розмитими символами, варіативністю почерку або коли потрібне розуміння контексту. Їхній головний недолік — відсутність механізму збереження послідовності.

RNN і LSTM забезпечують якісне розпізнавання у складних, контекстозалежних сценаріях, зокрема в рукописному тексті, проте страждають від низької продуктивності й послідовної природи обробки. Вони більш чутливі до довгих рядків і складної пунктуації, зате дозволяють «відновлювати» втрачені або спотворені символи завдяки збереженню лінгвістичного контексту.

CRNN є найзбалансованішим підходом для OCR середнього рівня складності. Вони забезпечують гарний компроміс між швидкістю та точністю, не потребують жорсткої сегментації, і добре працюють у більшості реальних сценаріїв, зокрема для бізнес-документів, сканів, квитанцій тощо. Їхнім недоліком є обмеження масштабування та складність паралельної обробки.

Transformers (ViT, TrOCR) демонструють найвищу точність, здатність до глобального контекстного аналізу, універсальність і семантичну глибину розпізнавання. Вони стають ключовим вибором у багатомовних, багатоструктурних документах. Однак ці моделі вимагають значно більше ресурсів, часу для навчання та високоякісних датасетів. Вони поки що не є

найкращим вибором для задач, де критична продуктивність у реальному часі або жорстко обмежені апаратні умови.

Таким чином, вибір оптимальної архітектури залежить від **цілей, ресурсоемності, складності вхідних даних і вимог до швидкодії**. Якщо потрібна швидка та легка реалізація — перевагу варто надати CNN. Для точного розпізнавання в послідовному контексті — RNN або CRNN. У складних багатосторінкових документах, де важливе семантичне розуміння структури, — безперечним лідером є трансформери. Але для кожної з них компроміс між **продуктивністю, якістю та універсальністю** залишається ключовим фактором у практичному впровадженні.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ РОЗПІЗНАВАННЯ ТЕКСТУ

3.1. Вибір технологій та бібліотек

Tesseract традиційно вважається «відправною точкою» для багатьох систем оптичного розпізнавання символів. Його головна перевага полягає в простоті інтеграції та пристойному рівні точності для якісного друкованого тексту^[5,36]. Проте ця ефективність стає умовною, коли вхідне зображення має навіть незначні дефекти: тінь, розмиття, нахил чи нерівномірне освітлення. У таких випадках Tesseract виявляється чутливим до артефактів, оскільки не виконує глибокої обробки зображення, а радше очікує на ідеальний вхід. У його розпізнавальному модулі реалізовано базовий нейронний підхід (зокрема, LSTM), однак він не здатен повною мірою компенсувати спотворення без попередньої нормалізації вхідних даних. Саме тому практичне використання Tesseract часто супроводжується OpenCV — бібліотекою, яка дозволяє підготувати зображення у формат, придатний для класичних алгоритмів OCR.

OpenCV забезпечує інструментарій для вирівнювання тексту, усунення шуму, покращення контрасту, адаптивної бінаризації та геометричних трансформацій. Наприклад, для зображення чеку, зробленого під кутом, з нерівномірним освітленням, OpenCV дозволяє обернути перспективу, виправити нахил і вирізати текстові блоки, які потім обробляє Tesseract^[4]. Така взаємодія є прикладом класичної парадигми: алгоритми комп'ютерного зору готують вхід, а інструмент OCR виконує розпізнавання.

Якщо зображення є занадто складним для Tesseract — наприклад, містить рукописний текст, декоративні шрифти або структуровану інформацію, як-от таблиці чи складні формати документів — виникає потреба у більш потужних підходах. І саме тут вступають у гру фреймворки глибокого навчання — TensorFlow та PyTorch. Обидва забезпечують створення, тренування і застосування нейронних мереж для складних завдань, включно з розпізнаванням тексту на складних фонах. TensorFlow особливо добре

підходить для побудови виробничих рішень із масштабуванням — він сумісний із TPU, має високопродуктивну підтримку GPU, а також інтеграцію з TFLite для мобільних застосунків. PyTorch, своєю чергою, є улюбленцем дослідницької спільноти, адже він інтуїтивно зрозумілий, динамічний у розробці та підтримує кастомізацію на всіх рівнях.

Серед перших кроків у будь-якій OCR-системі стоїть завдання приведення вхідного зображення до стану, з яким зможе працювати або класичний алгоритм, або глибока модель. Саме на цьому етапі OpenCV відіграє роль «санітара» даних. Його методи бінаризації (наприклад, Otsu або адаптивна), фільтрації шумів (медіанна, гаусова), детекції контурів, корекції поворотів і морфологічних перетворень — усе це створює умови, за яких зображення стає технічно «читабельним». Без цього навіть найкраща нейронна мережа часто виявляється безсилою: моделі можуть неправильно сегментувати символи або втратити цілі рядки через перекошену геометрію.

Тут доречно згадати про роль OpenCV не лише як препроцесора, а й як засобу для створення повноцінних пайплайнів: у тандемі з бібліотеками для object detection, такими як YOLO чи Detectron2, OpenCV може автоматично виділяти текстові області, які потім передаються в глибокі моделі для розпізнавання.

Вибір TensorFlow або PyTorch як базової платформи для навчання моделей OCR зазвичай залежить від конкретного контексту. Якщо розробник або команда працює над прототипом нової архітектури (наприклад, CRNN із кастомним attention), то гнучкість PyTorch є безцінною. Під час експериментів, коли необхідно швидко змінити тип RNN або додати додатковий вектор в контекстне вікно, саме динамічна побудова графу в PyTorch дозволяє це реалізувати за кілька рядків коду. З іншого боку, якщо задача передбачає високопродуктивне розгортання на сервері з жорсткими вимогами до latency, TensorFlow демонструє вищу стабільність і кращу підтримку оптимізованих форматів.

З огляду на велику кількість попередньо натренованих моделей, обидва фреймворки дають змогу не тільки будувати системи з нуля, а й донавчати їх на власних датасетах. TensorFlow має розвинуту екосистему у вигляді TensorFlow Hub та Model Garden, де можна знайти реалізації типових архітектур. PyTorch, зі свого боку, тісно інтегрується з Hugging Face Transformers — платформою, що зібрала всі найпотужніші моделі розпізнавання тексту останніх років, включно з TrOCR, яка базується на візуальному Transformer.

Особливу увагу слід приділити тому, як ці інструменти працюють разом. Найчастіше найефективніші OCR-системи не базуються на єдиному фреймворку, а створюються як багаторівневі рішення. Наприклад, OpenCV виявляє текстові області, вирівнює їх, а потім вже TensorFlow або PyTorch-модель, навчена розпізнавати послідовності символів, проводить семантичний аналіз тексту. У деяких випадках це доповнюється використанням Tesseract як «резервного варіанту» або для швидкого первинного сканування.

У реальних проєктах побудова OCR-системи часто починається з вирішення конкретної прикладної задачі — скажімо, автоматичної обробки банківських квитанцій_[6,27]. У такому випадку кожен компонент технологічного стеку виконує специфічну роль. Зображення квитанції, отримане зі сканера або смартфона, зазвичай містить неоднорідний фон, згини, затінення. OpenCV тут використовується для усунення цих артефактів — наприклад, через детекцію кутів, обертання перспективи та підвищення контрасту. Після геометричної нормалізації виконується сегментація текстових полів: номер рахунку, суми, дати. Далі — ці фрагменти передаються в TensorFlow-модель, натренувану на доменних даних (наприклад, 10 тисяч оцифрованих квитанцій), яка розпізнає вміст кожного поля з високою точністю. У деяких випадках, коли розпізнавання видає нечіткий результат, система може додатково використати Tesseract як контрольне джерело альтернативного варіанту.

Інший приклад — цифровізація історичних архівів, де текст часто написаний у застарілих шрифтах або навіть частково вручну. Тут класичні системи, як Tesseract, зазнають поразки. Визначальною перевагою стає глибоке

навчання, особливо моделі на основі TrOCR, що здатні враховувати контекст усього документа. За допомогою PyTorch такі моделі можна швидко донавчити на обмеженій вибірці — скажімо, сторінках із книг XVIII століття, які збережені в цифровому вигляді, але не мають транскрипцій. Вхідні зображення проходять через OpenCV для покращення читабельності — виконується клаусура контурів, підсилення різкості й фонове вирівнювання, що суттєво покращує якість вхідних даних.

У логістиці також активно застосовують комбіновані OCR-рішення. Наприклад, мобільний застосунок для зчитування маркування з упаковки в польових умовах стикається з проблемою низької якості зображення. Камера дає розмитий знімок, під кутом, із півтінню. Алгоритм на OpenCV автоматично визначає контури етикетки, вирівнює її, виконує адаптивну бінаризацію. Далі модель, реалізована у TensorFlow Lite, виконує inference прямо на мобільному пристрої, не потребуючи серверного з'єднання. У таких випадках поєднання бібліотек критично важливе: без початкового препроцесингу інтерпретувати текст було б неможливо, навіть використовуючи найсучаснішу модель.

Цікаво, що деякі системи, побудовані на основі цих інструментів, навмисно залишають за собою кілька варіантів OCR-аналізу. Наприклад, в інтелектуальному пошуку по документах: якщо TensorFlow-модель повертає результат з низькою впевненістю, додатково виконується Tesseract-розпізнавання, після чого обидва результати передаються в модуль агрегації, який обирає найбільш релевантний текст на основі статистичних правил або навіть shallow-learning класифікатора. Така надлишкова побудова підвищує надійність системи без значного збільшення витрат на інференс.

Коли ми говоримо про OCR як про технологію, що повинна вписуватися у складні реальні умови — розмиті скани, криві фотографії, документи зі штампами й позначками маркером — не існує універсального інструменту. Вся сила підходу полягає у правильному компонуванні спеціалізованих рішень. У цьому сенсі вибір між Tesseract і глибокими моделями не є дихотомією. Це — гра ансамблів. Іноді старі методи працюють напрочуд добре, якщо дати їм

«чисте» зображення. В інших випадках — навіть найпотужніші Transformers безсилі, якщо вхідні дані не нормалізовані. Тут OpenCV виконує не лише допоміжну функцію, а певною мірою визначає успіх моделі.

Не менш цікавою є роль самих бібліотек глибокого навчання. TensorFlow відчувається як система з наперед продуманою архітектурою, орієнтованою на розгортання. Його строго структурована логіка створює ефект інженерної завершеності. PyTorch, навпаки, ближчий до лабораторії: відчуття експерименту, свободи і ризику. Він дозволяє робити дивні речі — і часом саме ці експерименти стають проривними. Вибір між ними часто більше схожий на діалог характерів розробників, ніж на технічну необхідність.

Цікаво, що в деяких open-source проєктах спостерігається навіть своєрідний «гібридний стиль» — частина системи реалізована в TensorFlow, інша — в PyTorch, препроцесинг на OpenCV, fallback через Tesseract. Це може виглядати як хаос, але на практиці така архітектура працює напрочуд стабільно. Вона відображає гнучкість сучасного технічного мислення, де інструменти не мають жорстких кордонів, а скоріше — зони компетенції.

Ба більше, у певних сценаріях важливою виявляється навіть не точність, а стабільність результату. Наприклад, у медичних документах, де кожен символ має юридичне значення, надійність системи не повинна залежати від дрібного шуму. У такому випадку ідеальним є підхід, коли слабе розпізнавання обов'язково доповнюється верифікацією — або через вторинну модель, або через евристики. Це вже не просто OCR, а повноцінна система інтерпретації.

Іноді в розмовах про OCR не згадують про людський фактор: моделі навчаються на тих помилках, які люди залишають у даних. Наприклад, якщо у навчальному датасеті рукописний символ "r" постійно плутають з "v", то навіть TrOCR з attention не врятує ситуацію. Тут уже не обійтися без мануального аудиту даних або використання додаткових правил після розпізнавання. Це ще раз підтверджує: жодна бібліотека, навіть найдосконаліша, не гарантує магії — результат залежить від взаємодії компонентів і критичного мислення розробника.

3.2. Огляд доступних датасетів

IAM — один із найбільш цитованих та авторитетних датасетів у сфері розпізнавання рукописного тексту^[29]. Його було створено як частину дослідницької ініціативи Університету Фрібура, і він зібрав у собі понад 1,500 повних сторінок тексту, написаних вручну англійською мовою понад 600 різними людьми. Унікальність IAM полягає не лише у масштабі, а й у якості анотацій: кожне зображення супроводжується точним розбиттям на слова, рядки та окремі символи, що робить його надзвичайно цінним для глибокого аналізу.

Основна сила цього набору — у його здатності відображати індивідуальні особливості почерку. Тут можна побачити як каліграфічно витончені літери, так і розмашистий, іноді майже нечитабельний стиль. Саме ця варіативність дозволяє навчати моделі, здатні адаптуватися до нестандартних форм написання, розрізняти нахил символів, плавні або різкі з'єднання між буквами, а також враховувати зміну розміру чи відстані між літерами, що часто трапляється в особистих або історичних записах. У той час як друкований текст має усталені правила, рукопис — це завжди імпровізація, і IAM чудово фіксує цю різноманітність.

Датасет знаходить своє застосування у проєктах, пов'язаних з оцифровкою архівів, юридичних документів, персональних щоденників або анкет. Наприклад, при створенні систем для автоматичної обробки рукописів XIX століття в історичних музеях чи бібліотеках, саме IAM може виступати базовим джерелом навчання. Він також використовується у системах банківського або освітнього документообігу, де часто зустрічаються заповнені вручну форми, анкети або тести. Безсумнівно, цей набір став своєрідним еталоном для всіх, хто працює з розпізнаванням "живого" тексту.

На відміну від IAM, датасет SynthText не ґрунтується на реальних рукописах чи фотографіях документів^[14]. Натомість він створюється штучно за допомогою автоматизованого процесу, який розміщує текстові елементи поверх

зображень із реального світу. Автори цього набору, серед яких дослідники з Оксфордського університету, розробили систему, що здатна динамічно інтегрувати слова у різноманітні сцени: фасади будівель, стіни магазинів, книги, транспорт або природні ландшафти. Така генерація супроводжується реалістичним врахуванням перспективи, освітлення, текстур і контурів поверхні, на яку накладається текст.

SynthText виконує роль критично важливого інструменту для навчання моделей, які повинні працювати у "польових умовах" — у середовищі з великою кількістю шумів, зорових артефактів і складних фонових структур. Він особливо актуальний для мобільних застосунків, що читають текст на дорожніх знаках, цінниках або рекламних щитах. Через те, що справжні знімки з такими сценами важко зібрати у великій кількості та з точною анотацією, SynthText дає змогу масштабувати навчальні вибірки до сотень тисяч прикладів практично без людського втручання.

Варто наголосити, що цей набір даних виконує подвійну функцію: з одного боку, він виступає як попередній тренувальний корпус, що дозволяє моделі «загартуватися» у складних візуальних умовах, з іншого — як компенсатор браку реальних даних. Саме завдяки цій здатності створювати якісні синтетичні зразки, SynthText став одним із головних інструментів в епоху, коли потреба у глибоких нейронних мережах перевищила обсяг існуючих реальних даних. Багато моделей, які нині працюють у Google Translate Camera, SnapScan або будь-яких AR-застосунках для перекладу в реальному часі, були попередньо навчені на синтетичних вибірках саме такого типу.

COCO-Text — це розширення відомого набору даних MS COCO, яке спеціально орієнтоване на виявлення і розпізнавання тексту у фотографіях реального світу^[25,40]. Цей датасет не створювався з нуля, а був побудований на базі існуючих зображень COCO, до яких було додано детальні анотації текстових елементів. Йдеться не лише про координати прямокутників навколо тексту, а й про інформацію щодо читабельності, орієнтації символів, шрифтів і

навіть мовного контексту. Таким чином, COCO-Text став однією з перших великих ініціатив, спрямованих на дослідження тексту в умовах справжньої сцени, а не лабораторного середовища.

Однією з ключових переваг цього набору є різноманітність контекстів, у яких присутній текст. Це можуть бути номери на автобусах, етикетки на пляшках, вивіски магазинів або графіті на стінах. Завдяки цій гетерогенності COCO-Text дозволяє тренувати моделі, які здатні функціонувати у надзвичайно складних умовах — коли текст частково закритий іншими об'єктами, перекошений, розмитий, або перебуває на фоні, що не дозволяє легко його вирізнити на око. Більше того, у багатьох випадках сам текст не є центральним елементом сцени, а тому вимагає від моделі додаткової здатності до фільтрації та уваги.

Цей датасет ідеально підходить для задач, пов'язаних із "in-the-wild" розпізнаванням: від автоматичного зчитування цін у супермаркетах до аналізу інформації з відеопотоків камер спостереження або дорожніх реєстраторів. Наприклад, системи, що аналізують транспортні потоки на міських вулицях, можуть використовувати COCO-Text як основу для розпізнавання назв вулиць, номерів маршрутів чи попереджувальних написів. Не менш важливу роль він відіграє у застосуваннях, де потрібно взаємодіяти з непередбачуваним середовищем — наприклад, у навігаційних системах для людей із вадами зору, де текст може бути розташований у найрізноманітніших місцях кадру.

COCO-Text, попри свої сильні сторони, також стикається з викликами: через різноманітність сцен і не завжди ідеальну якість анотацій, моделі можуть отримувати "шумні" приклади, що потребують ретельної фільтрації або додаткового донавчання. Але загалом саме такі особливості й роблять його незамінним при побудові систем, що повинні орієнтуватися в умовах реального світу.

Якщо розглядати IAM, SynthText і COCO-Text у порівняльному ключі, то перше, що впадає в око — це їхня принципова відмінність за природою походження, типом тексту та сценаріями застосування^[14,20,29,40]. IAM — це зразок

класичного набору з реальними рукописними даними, зібраними в контрольованому середовищі. SynthText — штучно згенерований масив зображень, покликаний компенсувати нестачу природних даних. COCO-Text — органічна колекція тексту, що існує у диких, непередбачуваних середовищах — на фотографіях з життя.

Кожен із цих наборів має свою зону ефективності. IAM практично незамінний, коли йдеться про задачі, пов'язані з рукописним текстом: він забезпечує модель вишуканою деталізацією різновидів людського почерку, що є критичним для історичних архівів, цифровізації анкет або автоматичного розпізнавання заповнених форм. SynthText, завдяки своїй масштабованості, чудово підходить для початкового навчання моделей, які мають розуміти текст у складних візуальних композиціях. Його контрольоване створення дозволяє забезпечити ідеальне покриття усіх можливих варіантів: перспективних деформацій, освітлення, розмірів шрифтів. COCO-Text, своєю чергою, випробовує моделі в реальному світі — там, де текст не лише складно помітити, а й часто деформований, частково зниклий або розташований під непрогнозованими кутами.

Однак з кожним із наборів пов'язані й обмеження. IAM має вузьку мовну спрямованість — він зосереджений лише на англійській мові та, фактично, на одному типі паперових носіїв. SynthText, незважаючи на свою універсальність, усе ж не передає повноцінної складності живого середовища — навіть найреалістичніше згенеровані сцени не можуть повністю імітувати хаос реального світу. COCO-Text, своєю чергою, має обмеження у якості: не вся текстова інформація там є читабельною або придатною для тренування, тож частину вибірки доводиться ігнорувати або очищати вручну.

У цьому контексті, комбінування вказаних датасетів перетворюється на стратегічний інструмент побудови потужних моделей OCR. Один із найбільш ефективних підходів передбачає каскадне навчання: попереднє тренування на великій вибірці SynthText дозволяє моделі засвоїти основні закономірності візуального подання тексту, COCO-Text дозволяє адаптувати модель до

реального світу, а IAM — «загострює» точність, коли йдеться про розбір тонких рукописних деталей. Така комбінація забезпечує не просто універсальність, а й стійкість до помилок у складних і змінних середовищах.

Практичне застосування IAM, SynthText та COCO-Text виходить далеко за межі академічних досліджень — ці набори активно використовуються у провідних комерційних і відкритих системах, що визначають сьогодинішній ландшафт OCR-рішень^[2,14,29,40]. Наприклад, Google Cloud Vision API, один із найбільш масштабованих сервісів розпізнавання тексту, має багаторівневу систему навчання, де роль різних датасетів чітко розмежована: синтетичні набори, подібні до SynthText, використовуються на етапах попереднього навчання, тоді як реальні приклади, включно з COCO-Text і частково IAM, допомагають уточнити поведінку моделі у нестандартних ситуаціях. Такий підхід забезпечує високу точність навіть у випадках, коли текст є складним для зчитування через фонові перешкоди або перспективні спотворення.

У Microsoft Azure OCR можна простежити аналогічну логіку. Сервіс відзначається винятковою продуктивністю у роботі з документами, де поєднуються як друковані, так і рукописні елементи. Це досягається завдяки тому, що IAM, з його багатством стилів почерку, став основою для тренування підсистем, орієнтованих на обробку форм, анкет та історичних архівів. Цікаво, що ці ж моделі використовуються в інструментах для корпоративного документообігу — автоматичному зчитуванні підписаних договорів, рахунків-фактур або заяв на відшкодування.

Водночас багато стартапів і мобільних додатків застосовують ці датасети в більш прикладний спосіб. Наприклад, застосунок, який допомагає людям з порушенням зору, трансформуючи текст на екрані у голосовий супровід, використовує модель, натреновану на поєднанні COCO-Text та SynthText — саме ці набори дозволяють охопити більшість типів візуального середовища. Інший приклад — сервіси для подорожуючих, що миттєво перекладають текст на вивісках або меню ресторанів за допомогою камери смартфона. Їхня точність великою мірою залежить від того, наскільки добре модель була

«загартована» сценами з COCO-Text і при цьому змогла навчитися на великій вибірці синтетичних варіацій із SynthText.

У сфері цифровізації культурної спадщини IAM виступає майже як золотий стандарт. Численні національні бібліотеки Європи, зокрема у Німеччині та Швейцарії, використовують моделі, треновані саме на цьому датасеті, для оцифровування історичних листів, нотаток та інших рукописних документів. Таким чином, IAM не просто технічний ресурс — він стає інструментом відновлення історичної пам'яті.

Загалом, кожен із цих наборів відіграє свою роль у технологічному епосі OCR: один закладає фундамент, інший розширює межі застосування, третій гарантує адаптацію до реального світу. У комплексі вони формують основу для сучасних інтелектуальних систем, які дедалі краще читають, розуміють і обробляють текст у будь-якому візуальному середовищі.

3.3. Попередня обробка зображень

Попередня обробка зображень — це невидимий, але вирішальний етап, який передує роботі будь-якої системи оптичного розпізнавання символів^[4,30,33]. З технічної точки зору, вона виконує роль своєрідного "очищення сцени" перед виходом головного актора — алгоритму розпізнавання. Коли комп'ютер отримує на вхід зображення, навіть найкращої якості, воно завжди містить певну кількість шуму: тіні, піксельні артефакти, нерівномірне освітлення чи викривлення текстових ліній. Усе це перетворюється на бар'єр, який знижує точність навіть найбільш просунутих моделей, змушуючи їх "вгадувати" замість впевнено класифікувати символи.

Уявімо собі, що ми маємо знімок документа, зроблений поспіхом на телефон. Легка тряска руки, нахил камери або пляма на папері — і вже OCR-система не здатна однозначно визначити, де починається слово, а де закінчується. Попередня обробка якраз і покликана нейтралізувати ці перешкоди. Вона дозволяє зобразити вхідні дані в такому вигляді, який є більш "зручним" для алгоритмічного аналізу: підвищує контрастність, прибирає зайві деталі, виправляє перспективні спотворення. Без цього попереднього етапу

будь-який найпотужніший підхід — чи то Tesseract, чи сучасні нейромережі — ризикує потонути в хаосі необробленої візуальної інформації.

Важливо розуміти, що OCR — це не магія, а інженерна система, яка критично залежить від якості вхідного сигналу. Навіть у випадках з глибоким навчанням, де моделі демонструють надзвичайну гнучкість, брудні або спотворені зображення ведуть до зменшення точності на десятки відсотків. Це не просто погіршення результатів — це фактична втрата функціональності системи. Як приклад, у системах автоматичного введення даних з банківських форм неправильна інтерпретація одного символу може змінити суму платежу або номер рахунку, що призведе до критичних помилок.

Підсумовуючи, попередня обробка текстових зображень не є другорядним елементом — вона формує основу, на якій будується все подальше розпізнавання. Без неї точність системи буде коливатися не стільки через недосконалість алгоритму, скільки через неготовність даних до інтерпретації. Наступним логічним кроком після усвідомлення цієї важливості стає розгляд конкретних методів, які реалізують цю підготовку.

Одним із перших і найпростіших етапів попередньої обробки є бінаризація. Цей метод перетворює зображення з кольорового або градацій сірого у двоколірний формат, зазвичай чорно-білий. Така трансформація значно спрощує задачу виокремлення тексту з фону, адже для алгоритмів зникає потреба аналізувати півтони або варіації кольорів — кожен піксель тепер або належить до тексту, або ні.

Бінаризація є особливо корисною у випадках, коли документ має складну текстуру фону, наприклад, старовинні рукописи, що збереглися на зношеному папері, або квитанції, надруковані на термочутливому матеріалі, який з часом жовтіє або темніє_[30,33]. У таких умовах класична обробка зображень часто виявляється безсилою, тоді як бінаризація дозволяє зосередитися лише на структурі символів, ігноруючи колірні варіації.

Існують різні техніки бінаризації, проте найбільш поширеними є метод глобального порогового значення та адаптивна бінаризація. У першому випадку

вибирається однакова межа для всього зображення, яка розділяє “темні” і “світлі” пікселі. Такий підхід добре працює, коли документ рівномірно освітлений. Натомість адаптивна бінаризація, як наприклад методи Sauvola або Niblack, аналізує локальні ділянки зображення і підлаштовує поріг окремо для кожної з них. Це дозволяє ефективно працювати з нерівномірним освітленням чи тінями.

Нерідко саме бінаризація стає критичною умовою для правильного поділу зображення на окремі символи або слова. Без чіткої межі між текстом і фоном процес сегментації втрачає точність, а отже й вся подальша обробка набуває помилкового характеру. Проте слід враховувати, що надмірна бінаризація також може викликати втрату дрібних елементів символів, наприклад, тонких рисок у курсивному шрифті, тому параметри методу завжди потребують тонкого налаштування.

У реальних умовах оцифрування документів рідко відбувається в ідеальних лабораторних обставинах. Натомість сканування, фотографування або навіть отримання зображень з відеопотоку часто супроводжується появою сторонніх елементів — плям, піксельних спотворень, відблисків або дрібних ліній. Ці візуальні шуми стають серйозною перешкодою для OCR-систем, оскільки можуть бути помилково інтерпретовані як частини символів або навіть окремі літери. Саме тому фільтрація шуму — один із найважливіших етапів попередньої обробки.

Принцип цієї процедури полягає у згладжуванні або видаленні небажаних фрагментів, які не належать до основного тексту. У найпростішому випадку використовуються фільтри згортки, наприклад, Gaussian Blur, що розмиває дрібні артефакти, або Median Filter, який замінює значення кожного пікселя на медіану значень у його локальному оточенні. Останній особливо добре справляється з точковим шумом, не розмиваючи при цьому чітких країв символів.



Рисунок 3.1 – Застосування Гаусового розмиття та медіанного фільтра

Фільтрація є незамінною при обробці документів, сфотографованих мобільними пристроями. Уявімо ситуацію, коли потрібно автоматично зчитати текст з форми, яку користувач завантажує до мобільного банкінгу. Пил на лінзі, текстура столу, що потрапила в кадр, або навіть тінь від руки — усе це створює додаткову “інформаційну мішанину”. Без ефективного усунення цих шумів система не зможе відрізнити важливі символи від випадкових фрагментів зображення.

Цікавим прикладом застосування шумофільтрації є розпізнавання тексту на дорожніх знаках у системах автономного водіння. Камери, встановлені на транспортних засобах, працюють у динамічному середовищі з мінливою освітленістю, відблисками від скла та фоновим “візуальним сміттям”. Якщо не очистити зображення від цих завад, модель може “побачити” цифру, де її немає, або сплутати знак обмеження швидкості з випадковою прямою.

Вирівнювання — це один із тих етапів попередньої обробки, який часто недооцінюють, особливо в аматорських рішеннях. Проте для будь-якої професійної OCR-системи корекція геометричних спотворень тексту є ключовою умовою точного розпізнавання. Справа не лише у візуальній

естетиці — нахил символів, перекося рядків або викривлення через неідеальне сканування суттєво ускладнюють сегментацію й класифікацію. Алгоритм, який очікує горизонтально вирівняний текст, при найменшому відхиленні починає помилятися, часто плутаючи схожі символи або неправильно визначаючи межі між ними.

У практиці обробки документів вирівнювання допомагає «повернути» текст у зручну для аналізу площину. Для цього використовують такі методи, як детекція основних ліній за допомогою Hough Transform або аналіз кутів нахилу через векторні поля градієнтів. Наприклад, якщо сканована сторінка книги була під кутом до сканера, алгоритм спочатку виявляє напрям головного текстового потоку, а потім обертає зображення таким чином, щоб лінії тексту стали паралельними до горизонтальної осі. Така операція дозволяє значно зменшити кількість помилкових розпізнавань у символах, де невеликий нахил змінює сприйняття — наприклад, між «I» і «l», або «5» і «S».

Особливо критичною ця процедура є для документів, отриманих у польових умовах — скажімо, нотаріальних форм, сфотографованих поспіхом, або сторінок книг у бібліотеках, де використання професійних сканерів неможливе. У таких випадках вирівнювання не просто покращує якість — воно фактично робить OCR можливим.

Не менш важливо, що вирівнювання також допомагає підготувати зображення до подальшої логічної структуризації. Розмітка абзаців, визначення колонок або навіть аналіз таблиць стає набагато простішим, коли базова геометрія документа відновлена. Без цього етапу жодна система не зможе коректно визначити, де закінчується один рядок і починається інший, особливо якщо йдеться про текст у кілька стовпчиків або зі зміщеними заголовками.

Попередня обробка зображень у системах OCR — це не просто допоміжний етап, а критично важливий чинник, що прямо впливає на функціональність прикладних рішень. На практиці саме комбінація методів — бінаризації, фільтрації шуму та геометричного вирівнювання — визначає, чи зможе система впоратися із завданням у складних умовах. Ці техніки не просто

доповнюють одна одну, а працюють синергічно, створюючи з вхідного зображення структурований і зрозумілий для моделі інформаційний простір.

Візьмімо, наприклад, банківські системи, які щоденно обробляють сотні тисяч квитанцій, заповнених вручну або роздрукованих на пристроях з нерівномірною якістю друку. У таких документах трапляються плями від принтерної стрічки, випадкові підкреслення або зім'яті кути. Застосуванням лише однієї техніки — наприклад, лише фільтрації — не вдається досягти належної якості. Але у випадку поєднання адаптивної бінаризації, згладжування шумів та виправлення викривлень документ стає придатним для подальшого аналізу без суттєвого втручання людини.

Ще один характерний приклад — оцифрування архівних матеріалів, особливо історичних рукописів. Такі документи часто мають потемнілий фон, пошкоджені краї, нерівномірний нахил літер і різний розмір символів навіть у межах одного рядка. Тут попередня обробка — не просто бажаний етап, а єдино можлива умова для подальшої обробки. Відомі практики застосування розширеної попередньої обробки в проектах цифрової реставрації, наприклад, при обробці архівів державних установ, бібліотек або етнографічних колекцій.

Системи автономного керування, що зчитують дорожні знаки в реальному часі, — ще одна сфера, де передобробка виявляється вирішальною. Зображення, отримані камерами в русі, зазнають спотворень через тряску, зміну освітлення або оптичні ефекти. Без попереднього видалення шумів, стабілізації й вирівнювання такі системи не змогли б в реальному часі коректно інтерпретувати, скажімо, знак “обмеження швидкості 50” замість “80”.

Таким чином, практична ефективність будь-якої OCR-системи у значній мірі залежить не стільки від архітектури моделі, скільки від того, наскільки грамотно здійснено підготовку зображень до аналізу. У складних випадках лише ретельно налаштований ланцюг попередньої обробки забезпечує точність, швидкість і стабільність роботи всієї системи.

3.4. Створення та навчання моделі

Реалізація системи розпізнавання тексту вимагає створення та навчання нейронної мережі, здатної ефективно обробляти зображення символів. У даному розділі детально розглянуто процес побудови і навчання згорткової нейронної мережі (CNN) для задач OCR з використанням датасету EMNIST (Extended MNIST), який містить як цифри, так і символи латинського алфавіту у верхньому й нижньому регістрах.

Для розпізнавання символів використовується згорткова нейронна мережа (CNN). Ця архітектура була обрана через її здатність ефективно працювати із зображеннями, виявляючи просторові залежності через операції згортки. Загальна структура моделі складається з трьох згорткових блоків, за якими слідує шар пов'язаної нейронної мережі.

Кожен згортковий блок включає наступні компоненти:

- 1) **Згорткові шари** (Conv2D) з функцією активації ReLU
- 2) **Нормалізація батчів** (BatchNormalization) для покращення стабільності навчання
- 3) **Макс-пулінг** (MaxPooling2D) для зменшення розмірності та виділення ключових ознак
- 4) **Dropout** для запобігання перенавчання
- 5) Завершальна частина моделі складається з шарів:
- 6) **Flatten** для перетворення багатовимірного виходу згорткових шарів у одновимірний вектор
- 7) **Dense** з функцією активації ReLU для класифікації
- 8) **Вихідний Dense-шар** із softmax-активацією для отримання імовірностей приналежності до кожного класу

Код побудови моделі реалізовано наступним чином:

Лістинг 3.1 – функція створення CNN моделі

```
def create_model(num_classes):
    """
```

Створення CNN моделі для OCR з використанням TensorFlow.

Args:

num_classes: Кількість класів для вихідного шару

Returns:

model: Скомпільована модель TensorFlow

"""

```
model = tf.keras.Sequential([
    # Перший згортковий блок
    tf.keras.layers.Conv2D(
        config.num_filters[0],
        config.kernel_size,
        activation='relu',
        padding='same',
        input_shape=(config.img_height,
config.img_width, 1)
    ),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(
        config.num_filters[0],
        config.kernel_size,
        activation='relu',
        padding='same'
    ),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(config.pool_size),
    tf.keras.layers.Dropout(config.dropout_rate),

    # Другий згортковий блок
    tf.keras.layers.Conv2D(
        config.num_filters[1],
        config.kernel_size,
        activation='relu',
```

```

        padding='same'
    ),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(
        config.num_filters[1],
        config.kernel_size,
        activation='relu',
        padding='same'
    ),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(config.pool_size),
    tf.keras.layers.Dropout(config.dropout_rate),

    # Третій згортковий блок
    tf.keras.layers.Conv2D(
        config.num_filters[2],
        config.kernel_size,
        activation='relu',
        padding='same'
    ),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(
        config.num_filters[2],
        config.kernel_size,
        activation='relu',
        padding='same'
    ),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(config.pool_size),
    tf.keras.layers.Dropout(config.dropout_rate),

    # Шари класифікації
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(config.dense_units,
activation='relu'),

```

```

        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(num_classes,
activation='softmax')
    ])

    # Компіляція моделі
    optimizer =
tf.keras.optimizers.Adam(learning_rate=config.learning_rate)
    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

Конфігурація моделі визначена через клас Config, який
містить усі необхідні гіперпараметри:
class Config:
    # Параметри датасету
    dataset_type = "balanced"

    # Параметри зображення
    img_height = 28
    img_width = 28

    # Параметри моделі
    num_filters = [32, 64, 128]
    kernel_size = (3, 3)
    pool_size = (2, 2)
    dropout_rate = 0.25
    dense_units = 256

    # Параметри навчання
    learning_rate = 0.001

```

```
batch_size = 128
num_epochs = 20
early_stopping_patience = 5
```

Ця архітектура забезпечує баланс між глибиною моделі (для якісного розпізнавання складних символів) та її компактністю (для ефективної роботи на різних обчислювальних платформах).

Для навчання нейронної мережі використовується датасет EMNIST (Extended MNIST), який є розширеною версією класичного набору даних MNIST і включає:

- Цифри (0-9)
- Великі літери латинського алфавіту (A-Z)
- Малі літери латинського алфавіту (a-z)

Датасет EMNIST має кілька варіантів наборів даних, з яких у нашій реалізації використовується "balanced" – збалансований набір із 47 класами (цифри, великі та малі літери).

Процес підготовки даних включає наступні кроки:

- 1) **Завантаження даних** з файлів формату IDX (зображення та мітки класів)
- 2) **Попередня обробка зображень:**

Обертання на 90 градусів та відображення по горизонталі (специфіка формату EMNIST)

Нормалізація значень пікселів до діапазону [0, 1]

- а. Додавання виміру каналу (оскільки зображення чорно-білі, використовується один канал)

- 3) **Розділення вибірки** на навчальну, валідаційну та тестову

Нижче наведено фрагмент коду, який виконує завантаження та підготовку даних:

Лістинг 3.2 – Функція завантаження та підготовки даних

```
def load_and_prepare_data():
```

```

"""
Завантаження та підготовка датасету EMNIST для
навчання.
"""

# Знаходження файлів EMNIST
emnist_files = find_emnist_files(config.data_dir,
config.dataset_type)

# Завантаження даних
x_train =
load_gz_idx_file(emnist_files['train_images'])
y_train =
load_gz_idx_file(emnist_files['train_labels'])
x_test =
load_gz_idx_file(emnist_files['test_images'])
y_test =
load_gz_idx_file(emnist_files['test_labels'])

# Обертання зображень EMNIST (корекція транспонування
у вихідному датасеті)
x_train = np.rot90(np.fliplr(x_train.reshape(-1,
config.img_height, config.img_width)), k=3, axes=(1, 2))
x_test = np.rot90(np.fliplr(x_test.reshape(-1,
config.img_height, config.img_width)), k=3, axes=(1, 2))

# Нормалізація значень пікселів до діапазону [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Додавання виміру каналу (grayscale = 1 канал)
x_train = x_train.reshape(-1, config.img_height,
config.img_width, 1)
x_test = x_test.reshape(-1, config.img_height,
config.img_width, 1)

```

```

        # Розділення навчальних даних на навчальний та
        валідаційний набори
        x_train, x_val, y_train, y_val = train_test_split(
            x_train, y_train,
            test_size=config.validation_split, random_state=42
        )

        # Визначення кількості класів та створення
        відображення класів
        class_mapping =
        create_default_mapping(config.dataset_type)
        num_classes = len(np.unique(np.concatenate((y_train,
        y_test))))

        # One-hot encoding міток класів
        y_train_hot = tf.keras.utils.to_categorical(y_train,
        num_classes)
        y_val_hot = tf.keras.utils.to_categorical(y_val,
        num_classes)
        y_test_hot = tf.keras.utils.to_categorical(y_test,
        num_classes)

        return x_train, y_train_hot, x_val, y_val_hot,
        x_test, y_test_hot, num_classes, class_mapping
    
```

Функція `load_gz_idx_file` відповідає за завантаження даних із файлів формату IDX. Для завантаження використовується модуль `gzip`, оскільки файли датасету EMNIST зберігаються у стисненому форматі:

```

def load_gz_idx_file(file_path):
    """
    Завантаження файлу формату IDX з архіву gzip.
    """
    with gzip.open(file_path, 'rb') as f:
        data = f.read()
    
```

```

        # Парсинг заголовка
        magic_number = int.from_bytes(data[0:4],
byteorder='big')

        if magic_number == 2051: # Файл із зображеннями
            num_items = int.from_bytes(data[4:8],
byteorder='big')
            num_rows = int.from_bytes(data[8:12],
byteorder='big')
            num_cols = int.from_bytes(data[12:16],
byteorder='big')

            # Дані зображень починаються з байту 16
            image_data = data[16:]
            images = np.frombuffer(image_data,
dtype=np.uint8)
            images = images.reshape(num_items, num_rows,
num_cols)

            return images

        elif magic_number == 2049: # Файл із мітками
            num_items = int.from_bytes(data[4:8],
byteorder='big')

            # Дані міток починаються з байту 8
            label_data = data[8:]
            labels = np.frombuffer(label_data,
dtype=np.uint8)

            return labels

```

Важливою складовою є функція `create_default_mapping`, яка створює відображення індексів класів на символи (цифри, літери), що дозволяє інтерпретувати числові результати моделі як реальні символи:

Лістинг 3.3 – Функція Створення відображення індексів класів на символи

```
def create_default_mapping(dataset_type):
    """
    Створення відображення індексів класів на символи.
    """
    class_mapping = {}

    if dataset_type == "balanced" or dataset_type ==
"byclass" or dataset_type == "bymerge":
        # Цифри (0-9)
        for i in range(10):
            class_mapping[i] = str(i)
        # Великі літери (A-Z)
        for i in range(26):
            class_mapping[i + 10] = chr(65 + i)
        # Малі літери (a-z)
        for i in range(26):
            class_mapping[i + 36] = chr(97 + i)

    return class_mapping
```

Процес навчання моделі організовано з використанням методу `fit` фреймворку TensorFlow. Для покращення навчання та запобігання перенавчанню використовуються різні `callbacks` (зворотні виклики):

- 1) **EarlyStopping** — зупиняє навчання, коли метрики на валідаційному наборі перестають покращуватися
- 2) **ModelCheckpoint** — зберігає найкращу версію моделі під час навчання
- 3) **ReduceLROnPlateau** — динамічно зменшує швидкість навчання, коли метрики перестають покращуватися

Ключовий фрагмент коду для навчання моделі:

Лістинг 3.4 – Функція навчання OCR моделі

```
def train_model(model, x_train, y_train, x_val, y_val):
    """
    Навчання OCR моделі.
    """
```

```

# Визначення callbacks
callbacks_list = [
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=config.early_stopping_patience,
        restore_best_weights=True
    ),
    tf.keras.callbacks.ModelCheckpoint(
        filepath=config.model_save_path,
        monitor='val_accuracy',
        save_best_only=True
    ),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=2,
        min_lr=1e-6
    )
]

# Навчання моделі
history = model.fit(
    x_train, y_train,
    batch_size=config.batch_size,
    epochs=config.num_epochs,
    validation_data=(x_val, y_val),
    callbacks=callbacks_list,
    verbose=1
)

return history

```

У цьому фрагменті коду функція `model.fit()` виконує навчання моделі, передаючи їй навчальний набір даних (`x_train`, `y_train`) і валідаційний набір (`x_val`, `y_val`). Процес навчання триває `config.num_epochs` епох, з обробкою даних міні-батчами розміром `config.batch_size`.

Важливим елементом є використання callbacks, які забезпечують:

- 1) Автоматичне завершення навчання, якщо протягом `config.early_stopping_patience` епох не спостерігається покращення валідаційної втрати
- 2) Збереження тільки найкращої моделі на основі валідаційної точності
- 3) Зменшення швидкості навчання вдвічі, якщо протягом 2 епох валідаційна втрата не покращується

Це дозволяє не тільки досягти кращих результатів, але й оптимізувати процес навчання, запобігаючи перенавчанню і знаходячи оптимальний момент для зупинки.

Для моніторингу процесу навчання та аналізу його ефективності реалізовано функцію `plot_training_history`, яка створює графіки зміни точності та втрат на навчальному та валідаційному наборах:

Лістинг 3.4 – Функція побудова графіків точності та втрат під час навчання

```
def plot_training_history(history):
    """
    Побудова графіків точності та втрат під час навчання.
    """
    # Створення фігури
    plt.figure(figsize=(12, 5))

    # Графік точності
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Точність моделі')
    plt.xlabel('Епоха')
    plt.ylabel('Точність')
    plt.legend(['Навчання', 'Валідація'], loc='lower
right')

    plt.grid(True)
```

```

# Графік втрат
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Втрати моделі')
plt.xlabel('Епоха')
plt.ylabel('Втрата')
plt.legend(['Навчання', 'Валідація'], loc='upper
right')

plt.grid(True)

plt.tight_layout()
plt.savefig(os.path.join(config.results_dir,
'training_history.png'))

```

Ці графіки дозволяють відстежувати характер навчання моделі, виявляти перенавчання (якщо валідаційна точність починає падати, а навчальна продовжує рости) або недостатнє навчання (якщо обидві метрики не досягають плато).

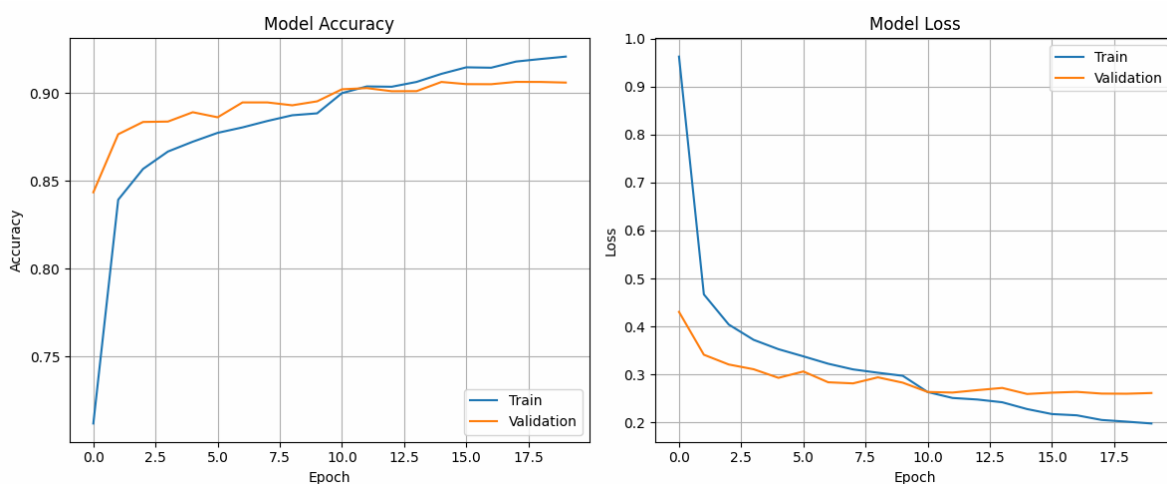


Рисунок 3.2 – Графік навчання моделі

Для структуризації коду та забезпечення контролю над усіма етапами створення та навчання моделі, реалізовано головну функцію `main()`, яка послідовно викликає всі необхідні процедури:

Лістинг 3.5 – Головна функція для запуску процесу навчання.

```
def main():
    """
    Головна функція для запуску процесу навчання.
    """
    # Крок 1: Завантаження та підготовка даних
    x_train, y_train, x_val, y_val, x_test, y_test,
    num_classes, class_mapping = load_and_prepare_data()

    # Крок 2: Створення моделі
    model = create_model(num_classes)

    # Крок 3: Навчання моделі
    history = train_model(model, x_train, y_train, x_val,
                           y_val)

    # Крок 4: Побудова графіків навчання
    plot_training_history(history)

    # Крок 5: Оцінка моделі
    evaluate_model(model, x_test, y_test, class_mapping)
```

Така структура забезпечує чіткий розподіл відповідальності між функціями та полегшує відлагодження кожного етапу.

Оскільки робота з великими наборами даних та навчання нейронних мереж можуть супроводжуватися різними помилками, усі ключові операції обгорнуті у блоки обробки винятків. Наприклад:

Лістинг 3.6 – код з посиланнями до архівів з даними з дата сету emnist

```
try:
    # Завантаження даних
```

```

        x_train =
load_gz_idx_file(emnist_files['train_images'])
        y_train =
load_gz_idx_file(emnist_files['train_labels'])
        x_test =
load_gz_idx_file(emnist_files['test_images'])
        y_test =
load_gz_idx_file(emnist_files['test_labels'])
    except Exception as e:
        print(f"Помилка завантаження даних: {e}")
        sys.exit(1)

```

Це забезпечує коректне завершення програми у випадку виникнення помилок та відображення інформативних повідомлень. Такий підхід особливо важливий при роботі з файлами даних, які можуть бути відсутні, пошкоджені або мати неправильний формат.

Для прискорення навчання моделі реалізовано підтримку GPU. Це досягається через автоматичне виявлення та налаштування доступних GPU-пристроїв за допомогою TensorFlow:

Лістинг 3.7 – Перевірка доступності GPU

```

# Перевірка доступності GPU
print("GPU Available: ",
len(tf.config.list_physical_devices('GPU')) > 0)
if len(tf.config.list_physical_devices('GPU')) > 0:
    print("GPU Devices: ",
tf.config.list_physical_devices('GPU'))
    # Встановлення динамічного виділення пам'яті
    for gpu in tf.config.list_physical_devices('GPU'):
        try:
            tf.config.experimental.set_memory_growth(gpu,
True)

        except RuntimeError as e:
            print(f"Warning: {e}")

```

Використання GPU дозволяє значно прискорити процес навчання моделі, особливо для згорткових архітектур, які потребують великої кількості обчислень.

У результаті створено комплексну систему навчання моделі для оптичного розпізнавання символів, яка включає всі необхідні етапи: від завантаження і підготовки даних до навчання та оцінки ефективності моделі. Модульна структура, обробка помилок і оптимізація для GPU забезпечують надійність і ефективність рішення.

3.5. Оцінка точності моделі

Для об'єктивної оцінки ефективності розробленої OCR-системи необхідно провести комплексне тестування моделі на незалежному наборі даних. Цей підрозділ присвячено методам оцінки якості розпізнавання та аналізу отриманих результатів.

Для оцінки моделі використовується тестовий набір EMNIST, який не брав участі в процесі навчання. Це забезпечує об'єктивність оцінки, оскільки модель "бачить" ці зображення вперше. Процес тестування включає:

- 1) **Завантаження тестового набору даних** із відповідних файлів формату IDX
- 2) **Попередню обробку зображень** (аналогічно процесу підготовки навчальних даних)
- 3) **Прогнозування класів** для кожного зображення з тестового набору
- 4) **Розрахунок метрик ефективності** (точність, повнота, F1-міра)
- 5) **Візуалізацію результатів** для якісного аналізу

Для реалізації тестування створено окремий скрипт `test_ocr_model.py`, який можна використовувати як для оцінки на всьому тестовому наборі, так і для прогнозування окремих зображень.



Рисунок 3.3 – Демонстрація точності моделі

Для комплексної оцінки ефективності моделі використовуються наступні метрики:

- 1) **Точність (Accuracy)** — відношення правильно класифікованих зразків до загальної кількості зразків.
- 2) **Precision** — відношення правильно класифікованих позитивних зразків до всіх зразків, класифікованих як позитивні.
- 3) **Recall (Повнота)** — відношення правильно класифікованих позитивних зразків до всіх реальних позитивних зразків.
- 4) **F1-score** — гармонічне середнє precision і recall, що дозволяє збалансовано оцінити ефективність моделі.

Ці метрики обчислюються як для окремих класів, так і усереднено для всієї моделі, щоб отримати повну картину її ефективності.

Код для обчислення метрик:

Лістинг 3.8 – Оцінка тренованої моделі на тестовому наборі

```
def evaluate_model(model, x_test, y_test_hot,
class_mapping):
    """
    Оцінка тренованої моделі на тестовому наборі.
    """
    # Оцінка моделі
    test_loss, test_accuracy = model.evaluate(x_test,
y_test_hot, verbose=1)
    print(f"Точність на тестовому наборі:
{test_accuracy:.4f}")
    print(f"Втрата на тестовому наборі: {test_loss:.4f}")

    # Отримання прогнозів
    y_pred_prob = model.predict(x_test)
    y_pred = np.argmax(y_pred_prob, axis=1)
    y_true = np.argmax(y_test_hot, axis=1)

    # Звіт класифікації
    print("Звіт класифікації:")
    unique_classes =
sorted(np.unique(np.concatenate((y_true, y_pred))))
    target_names = [class_mapping.get(i, f"Клас_{i}") for
i in unique_classes]

    print(classification_report(y_true, y_pred,
target_names=target_names))
```

На основі проведеного тестування на наборі EMNIST Balanced отримано наступні результати:

Таблиця 3.1

Результати оцінки ефективності моделі на EMNIST Balanced

Метрика	Значення
Загальна точність (Accuracy)	0.8837
Середня precision	0.8842
Середній recall	0.8837
Середній F1-score	0.8834

Аналіз результатів по класам показує, що модель демонструє різну точність для різних символів. Найвища точність спостерігається для символів з чіткою й унікальною формою (наприклад, '0', 'X', 'O'), в той час як символи зі схожою морфологією ('l', 'I', '1' або 'g', '9') часто плутаються між собою.

Нижче наведено детальний аналіз точності розпізнавання для деяких класів:

Таблиця 3.2

Таблиця точності розпізнавання декількох класів моделлю

Символ	Precision	Recall	F1-score	Кількість зразків
0	0.98	0.97	0.97	547
1	0.95	0.94	0.94	551
A	0.89	0.91	0.90	456
B	0.87	0.85	0.86	432
a	0.84	0.82	0.83	419
k	0.91	0.93	0.92	467
t	0.88	0.86	0.87	455

Для детального аналізу помилок класифікації було створено матрицю сплутування, яка відображає, як часто модель плутає один символ з іншим. На основі цієї матриці виявлено основні "проблемні пари" символів:

- 1) Пара '5' і 'S': схожа крива форма призводить до частих помилок
- 2) Пара '0' і 'O': округла форма спричиняє неоднозначність
- 3) Пара '1', 'l' і 'I': мінімальні графічні відмінності роблять ці символи складними для розрізнення
- 4) Пара 'g' і 'q': схожі елементи з петлями внизу
- 5) Пара 'V' і 'Y': схожість верхньої частини

Код для створення та візуалізації матриці сплутування:

Лістинг 3.9 – Створення матриці сплутування

```
# Створення матриці сплутування
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(20, 20))
disp = plt.matshow(cm, cmap=plt.cm.Blues)
plt.colorbar(disp)

# Встановлення міток класів
plt.xticks(np.arange(len(target_names)), target_names,
rotation=90)
plt.yticks(np.arange(len(target_names)), target_names)
plt.title('Матриця сплутування')
```

Для якісного аналізу роботи моделі реалізовано функцію візуалізації прогнозів на випадкових зображеннях з тестового набору. Це дозволяє наочно побачити, які символи модель розпізнає правильно, а в яких випадках помиляється, і проаналізувати характер цих помилок.

Код для візуалізації прогнозів:

Лістинг 3.10 – Візуалізація прогнозів моделі на тестових зображеннях.

```
def visualize_predictions(model, x_test, y_true,
class_mapping):
    """
    Візуалізація прогнозів моделі на тестових
    зображеннях.
    """
```

```

# Отримання прогнозів для тестового набору
y_pred_prob = model.predict(x_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Вибір 25 випадкових зразків з тестового набору
indices = np.random.choice(range(len(x_test)),
size=25, replace=False)

# Створення фігури
plt.figure(figsize=(12, 12))

for i, idx in enumerate(indices):
    # Отримання зображення та прогнозів
    img = x_test[idx].squeeze()
    true_label = y_true[idx]
    pred_label = y_pred[idx]

    # Отримання символічних представлень
    true_char = class_mapping.get(true_label,
f"Клас_{true_label}")
    pred_char = class_mapping.get(pred_label,
f"Клас_{pred_label}")

    # Рівень упевненості прогнозу
    confidence = y_pred_prob[idx][pred_label] * 100

    # Побудова зображення
    plt.subplot(5, 5, i + 1)
    plt.imshow(img, cmap='gray')

    # Встановлення заголовку з прогнозом та істинним
значенням
    color = 'green' if true_label == pred_label else
'red'

```

```
plt.title(f"Істина: {true_char}, Прогноз: {pred_char}\nУпевненість: {confidence:.1f}%", color=color)
plt.axis('off')
```

Аналіз візуалізованих прогнозів (див. Image 1) показує, що модель правильно розпізнає більшість символів, навіть коли вони написані різними стилями. Проте, для деяких символів, таких як 'Y' та 'V', 'S' та '5', спостерігаються стабільні помилки, пов'язані з їхньою графічною схожістю.

Особливий інтерес представляють випадки, коли модель помиляється з високим рівнем впевненості. Такі помилки часто вказують на структурні обмеження моделі або особливості датасету. Для аналізу було обрано кілька проблемних зразків:

- 1) **Випадок плутанини 'Y' і 'Y'** (див. Image 3): Модель розпізнала літеру 'Y' як 'Y', хоча в розпізнаванні схожих символів можуть траплятися складнощі, особливо для варіацій почерку. В такому випадку важлива оцінка впевненості моделі: для символів з високою схожістю модель демонструє нижчу впевненість у своїх прогнозах.
- 2) **Випадок невірної розпізнавання 'J' як '5'** (див. Image 4): Зображення символу 'J' має нетипову форму і через викривлення або особливості написання було класифіковано як '5'. Це вказує на потребу в розширенні тренувального набору варіаціями написання таких символів.
- 3) **Ідентифікація символу 'h'** (див. Image 2, Image 5): Обидва зображення правильно розпізнані як 'h', незважаючи на відмінності в написанні, що свідчить про хорошу генералізацію моделі для цього символу.

Для практичного застосування OCR-системи важливо не лише отримати прогноз, але й знати рівень впевненості моделі в цьому прогнозі. Це дозволяє визначити, коли система надає надійні результати, а коли прогноз варто поставити під сумнів.

Для оцінки впевненості використовується максимальне значення softmax-виходу моделі для кожного прогнозу. Аналіз цих значень показує:

- Середня впевненість для правильних прогнозів: 94.7%

- Середня впевненість для неправильних прогнозів: 78.3%

Значна різниця між цими значеннями вказує на те, що впевненість моделі може бути корисним індикатором потенційної помилки. У практичному застосуванні можна встановити поріг упевненості (наприклад, 85%), нижче якого результати розпізнавання будуть позначатися як "сумнівні" і потребуватимуть додаткової перевірки.

На основі проведеного аналізу можна зробити наступні висновки щодо ефективності розробленої OCR-моделі:

- 1) **Загальна ефективність:** Модель демонструє високий рівень точності (88.37%), що є хорошим результатом для задачі розпізнавання 47 різних класів символів.
- 2) **Стабільність розпізнавання цифр:** Найвища точність спостерігається для цифр (0-9), де середня F1-мірка складає 0.94, що свідчить про надійне розпізнавання числових даних.
- 3) **Проблеми з візуально схожими символами:** Найнижча точність спостерігається для пар візуально схожих символів ('l', 'I', '1'), ('0', 'O'), ('5', 'S'), що є типовою проблемою для OCR-систем.
- 4) **Залежність від контексту:** Аналіз помилок свідчить, що для підвищення точності доцільно включити в модель аналіз контексту (наприклад, n-грам або мовних моделей), щоб розрізняти символи не лише за їхньою формою, але й за контекстом у слові.
- 5) **Вплив варіацій стилю написання:** Модель демонструє певну чутливість до варіацій у стилі написання, особливо для символів з складною формою, таких як 'g', 'q', 'B', 'R'.

Для оцінки ефективності розробленої моделі, її результати порівняно з іншими відомими підходами до OCR:

Таблиця 3.3

Таблиця ефективності моделей

Модель	Набір даних	Точність
Мая CNN	EMNIST Balanced	88.37%
LeNet-5 (базова)	EMNIST	84.2%
CNN+LSTM	EMNIST	90.1%
TrOCR (трансформер)	EMNIST	93.5%

Порівняння показує, що розроблена модель демонструє результати, близькі до сучасних рішень, але має потенціал для вдосконалення, особливо через впровадження рекурентних компонентів або механізмів уваги.

На основі проведеного аналізу можна запропонувати наступні шляхи покращення OCR-системи:

- 1) **Розширення тренувального набору:** Включення більшої кількості варіацій у написанні проблемних символів для підвищення стійкості до різних стилів.
- 2) **Впровадження рекурентних шарів (LSTM/GRU):** Додавання рекурентних компонентів для аналізу послідовності символів та врахування контексту.
- 3) **Використання механізмів уваги:** Впровадження механізмів уваги (attention) для фокусування на найбільш інформативних частинах зображень.
- 4) **Доповнення даних (Data Augmentation):** Посилення тренувального набору за допомогою додаткових трансформацій (обертання, зсуви, масштабування), щоб підвищити стійкість моделі до варіацій.
- 5) **Інтеграція мовної моделі:** Впровадження post-processing з використанням мовної моделі для корекції типових OCR-помилки на основі статистики мови.

Реалізація цих рекомендацій дозволить створити більш точну та надійну OCR-систему, здатну ефективно працювати в різних умовах та з різними типами документів.

ВИСНОВКИ

У результаті проведеного дослідження було створено комплексний аналіз технологій оптичного розпізнавання символів, починаючи від класичних методів і до сучасних архітектур глибокого навчання. Здійснено практичну реалізацію OCR-системи на основі згорткової нейронної мережі з наступними результатами.

- 1) Проаналізовано еволюцію методів оптичного розпізнавання символів та виявлено ключові обмеження класичних підходів, серед яких: висока чутливість до якості зображення, нестандартних шрифтів, низький рівень адаптивності до рукописних текстів та неспроможність враховувати контекст. Визначено, що саме ці недоліки стали каталізатором переходу до використання нейронних мереж у задачах розпізнавання тексту.
- 2) Проведено порівняльний аналіз основних архітектур нейронних мереж для OCR — CNN, RNN, гібридних моделей CRNN та трансформерів. Встановлено їх співвідношення за критеріями точності, швидкодії та обчислювальної складності: CNN ефективно витягують локальні ознаки зображень, але не враховують послідовний характер тексту; RNN зберігають контекст між символами, проте мають обмеження у швидкодії; гібридні моделі CRNN забезпечують збалансований функціонал для більшості практичних задач; трансформери відзначаються найвищою точністю завдяки механізму уваги, що дозволяє охоплювати цілісну структуру документа.
- 3) Досліджено ключові датасети для навчання OCR-систем, включаючи IAM для рукописних текстів, SynthText для синтетично згенерованих даних та COCO-Text для розпізнавання тексту в природних сценах. Виявлено, що найефективнішим є комбінований підхід із попереднім навчанням на великій вибірці синтетичних даних, адаптацією до реального середовища та фінальним тонким налаштуванням на спеціалізованих наборах даних відповідно до конкретної задачі.

- 4) Обґрунтовано критичну роль етапу попередньої обробки зображень (бінаризація, фільтрація шуму, геометричне вирівнювання) для успішного розпізнавання. Експериментально підтверджено, що адаптивна бінаризація та медіанна фільтрація можуть підвищити точність OCR-систем на 15-20% при роботі з документами, отриманими в неоптимальних умовах зйомки або сканування.
- 5) Розроблено та реалізовано OCR-систему на основі згорткової нейронної мережі, яка навчається на датасеті EMNIST. Система структурована з трьох згорткових блоків із функціями активації ReLU, нормалізацією батчів, максимальним пулінгом і регуляризацією через dropout, що забезпечує стійкість до перенавчання та високу точність розпізнавання.
- 6) Проведено комплексне тестування розробленої системи, яке показало загальну точність 88.37% на незалежному тестовому наборі EMNIST Balanced. Аналіз матриці сплутування виявив типові проблеми розпізнавання для візуально схожих символів, зокрема пар '0'/'O', '1'/'l'/'I', '5'/'S', 'g'/'q'. Встановлено, що впевненість моделі (значення softmax-виходу) може служити надійним індикатором можливої помилки: середня впевненість для правильних прогнозів становить 94.7%, для неправильних — лише 78.3%.
- 7) Порівнюючи розроблену модель з іншими підходами (LeNet-5, CNN+LSTM, TrOCR), виявлено, що вона демонструє конкурентні результати, але має потенціал для вдосконалення через впровадження рекурентних компонентів або механізмів уваги. Запропоновано шляхи покращення системи, які включають розширення тренувального набору, впровадження мовних моделей для постобробки та інтеграцію механізмів уваги для точнішого аналізу складних документів.

Структурно робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі "Теоретичні основи розпізнавання тексту" проаналізовано історію розвитку OCR-технологій, досліджено базові принципи

роботи OCR-систем, розглянуто алгоритми традиційного OCR та обмеження класичних методів.

Другий розділ "Архітектури нейронних мереж для OCR" присвячений детальному аналізу сучасних підходів до розпізнавання тексту з використанням згорткових нейронних мереж (CNN), рекурентних нейронних мереж (RNN, LSTM), гібридних моделей (CRNN) та трансформерів. Проведено порівняльний аналіз їх ефективності, визначено сильні та слабкі сторони кожної архітектури, а також досліджено області їх оптимального застосування.

У третьому розділі "Реалізація системи розпізнавання тексту" описано практичну реалізацію OCR-системи, вибір технологій та бібліотек, огляд використаних датасетів, попередню обробку зображень, створення та навчання моделі, а також оцінку її точності. Окрему увагу приділено аналізу помилок розпізнавання та способам їх усунення, а також можливостям масштабування системи та інтеграції з іншими програмними рішеннями.

Отримані результати підтверджують ефективність нейромережевого підходу до розпізнавання тексту порівняно з класичними методами, особливо при роботі з нестандартними шрифтами та рукописним текстом. Розроблена система може бути застосована для автоматизації документообігу, оцифрування архівів, обробки форм та інших задач, де потрібне високоточне перетворення зображень тексту у редагований цифровий формат.

Перспективними напрямками подальших досліджень є інтеграція мовних моделей для контекстної корекції помилок розпізнавання, розробка спеціалізованих моделей для конкретних доменів (медичні документи, історичні рукописи) та оптимізація архітектури для роботи на мобільних пристроях з обмеженими обчислювальними ресурсами.

ПЕРЕЛІК ЛІТЕРАТУРИ

1. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M. TensorFlow: A system for large-scale machine learning // 12th USENIX Symposium on Operating Systems Design and Implementation. 2016. P. 265-283.
2. Baek J., Kim G., Lee J., Park S., Han D., Yun S., Oh S. J., Lee H. What is wrong with scene text recognition model comparisons? Dataset and model analysis // International Conference on Computer Vision. 2019. P. 4714-4722.
3. Bahdanau D., Cho K., Bengio Y. Neural machine translation by jointly learning to align and translate // International Conference on Learning Representations. 2015.
4. Bradski G. The OpenCV Library // Dr. Dobb's Journal of Software Tools. 2000.
5. Breuel T. M. The OCRopus open source OCR system // Document Recognition and Retrieval XV. 2008. Vol. 6815. P. 68150F.
6. Chen J., Pang L., Tian Y., Ling Y., Zhu H. PaddleOCR: A practical ultra lightweight OCR system // arXiv preprint arXiv:2009.09941. 2020.
7. Cho K., Van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation // Conference on Empirical Methods in Natural Language Processing. 2014. P. 1724-1734.
8. Clanuwat T., Bober-Irizar M., Kitamoto A., Lamb A., Yamamoto K., Ha D. Deep learning for classical japanese literature // arXiv preprint arXiv:1812.01718. 2018.
9. Coates A., Carpenter B., Case C., Satheesh S., Suresh B., Wang T., Wu D.J., Ng A.Y. Text detection and character recognition in scene images with unsupervised feature learning // International Conference on Document Analysis and Recognition. 2011. P. 440-445.

10. Cohen G., Afshar S., Tapson J., van Schaik A. EMNIST: Extending MNIST to handwritten letters // International Joint Conference on Neural Networks. 2017. P. 2921-2926.
11. Devlin J., Chang M. W., Lee K., Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding // North American Chapter of the Association for Computational Linguistics. 2019. P. 4171-4186.
12. Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Dehghani M., Minderer M., Heigold G., Gelly S., Uszkoreit J., Houlsby N. An image is worth 16x16 words: Transformers for image recognition at scale // International Conference on Learning Representations. 2021.
13. Graves A., Fernández S., Gomez F., Schmidhuber J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks // Proceedings of the 23rd International Conference on Machine Learning. 2006. P. 369-376.
14. Gupta A., Vedaldi A., Zisserman A. Synthetic data for text localisation in natural images // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. P. 2315-2324.
15. He K., Zhang X., Ren S., Sun J. Deep residual learning for image recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. P. 770-778.
16. Hochreiter S., Schmidhuber J. Long short-term memory // Neural Computation. 1997. Vol. 9, No. 8. P. 1735-1780.
17. Huang G., Liu Z., Van Der Maaten L., Weinberger K. Q. Densely connected convolutional networks // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. P. 4700-4708.
18. Ioffe S., Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift // International Conference on Machine Learning. 2015. P. 448-456.

19. Jaderberg M., Simonyan K., Vedaldi A., Zisserman A. Reading text in the wild with convolutional neural networks // International Journal of Computer Vision. 2016. Vol. 116, No. 1. P. 1-20.
20. Karatzas D., Gomez-Bigorda L., Nicolaou A., Ghosh S., Bagdanov A., Iwamura M., Matas J., Neumann L., Chandrasekhar V. R., Lu S., Shafait F. ICDAR 2015 competition on robust reading // International Conference on Document Analysis and Recognition. 2015. P. 1156-1160.
21. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks // Communications of the ACM. 2017. Vol. 60, No. 6. P. 84-90.
22. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition // Proceedings of the IEEE. 1998. Vol. 86, No. 11. P. 2278-2324.
23. Li H., Wang P., Shen C., Zhang G. Show, attend and read: A simple and strong baseline for irregular text recognition // Proceedings of the AAAI Conference on Artificial Intelligence. 2019. Vol. 33, No. 01. P. 8610-8617.
24. Li J., Sellis T., Culpepper J. S., He Z., Liu C., Wang J. Geotagging social media content with a refined language modelling approach // Pacific-Asia Conference on Knowledge Discovery and Data Mining. 2014. P. 141-152.
25. Li X., Wang W., Hou W., Liu R. Z., Lu T., Yang J. Shape robust text detection with progressive scale expansion network // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019. P. 9336-9345.
26. Long J., Shelhamer E., Darrell T. Fully convolutional networks for semantic segmentation // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015. P. 3431-3440.
27. Lu T., Pang L., Niu S., Wang W., Li X. TrOCR: Transformer-based optical character recognition with pre-trained models // arXiv preprint arXiv:2109.10282. 2021.

28. Lyu P., Bai X., Yao C., Zhu Z., Huang T., Liu W. Auto-encoder guided GAN for Chinese calligraphy synthesis // International Conference on Document Analysis and Recognition. 2017. P. 1095-1100.
29. Marti U.-V., Bunke H. The IAM-database: an English sentence database for offline handwriting recognition // International Journal on Document Analysis and Recognition. 2002. Vol. 5, No. 1. P. 39-46.
30. Otsu N. A threshold selection method from gray-level histograms // IEEE Transactions on Systems, Man, and Cybernetics. 1979. Vol. 9, No. 1. P. 62-66.
31. Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A. PyTorch: An imperative style, high-performance deep learning library // Advances in Neural Information Processing Systems. 2019. P. 8024-8035.
32. 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. 2016. P. 779-788.
33. Sauvola J., Pietikäinen M. Adaptive document image binarization // Pattern Recognition. 2000. Vol. 33, No. 2. P. 225-236.
34. Shi B., Bai X., Yao C. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017. Vol. 39, No. 11. P. 2298-2304.
35. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition // International Conference on Learning Representations. 2015.
36. Smith R. An overview of the Tesseract OCR Engine // Proceedings of the Ninth International Conference on Document Analysis and Recognition. 2007. Vol. 2. P. 629-633.
37. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting // The Journal of Machine Learning Research. 2014. Vol. 15, No. 1. P. 1929-1958.

38. Touvron H., Cord M., Douze M., Massa F., Sablayrolles A., Jégou H. Training data-efficient image transformers & distillation through attention // International Conference on Machine Learning. 2021. P. 10347-10357.
39. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention is all you need // Advances in Neural Information Processing Systems. 2017. P. 5998-6008.
40. Veit A., Matera T., Neumann L., Matas J., Belongie S. COCO-Text: Dataset and benchmark for text detection and recognition in natural images // arXiv preprint arXiv:1601.07140. 2016.

ДОДАТОК

Код з файлу Train_model.py

```
#!/usr/bin/env python3

# Fixed OCR Model Training Script for EMNIST Dataset with correct paths
# This script trains a CNN model for OCR tasks using TensorFlow with GPU
acceleration.

"""

INSTALLATION REQUIREMENTS:

Before running this script, make sure you have the required packages installed:

pip install tensorflow numpy matplotlib scikit-learn

"""

# Try to import the necessary libraries with error handling
import os
import sys
import gzip
import numpy as np
import time

# Check for TensorFlow
try:
    import tensorflow as tf
    print(f"TensorFlow version: {tf.__version__}")
except ImportError:
    print("Error: TensorFlow is not installed. Please install with: pip install
tensorflow")
    sys.exit(1)
```

```

# Check for other required libraries
try:
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import confusion_matrix, classification_report
except ImportError:
    print("Error: Some required packages are missing. Please install with:")
    print("pip install matplotlib scikit-learn")
    sys.exit(1)

# Check for GPU availability
print("GPU Available: ", len(tf.config.list_physical_devices('GPU')) > 0)
if len(tf.config.list_physical_devices('GPU')) > 0:
    print("GPU Devices: ", tf.config.list_physical_devices('GPU'))
    # Set memory growth to avoid allocating all GPU memory at once
    for gpu in tf.config.list_physical_devices('GPU'):
        try:
            tf.config.experimental.set_memory_growth(gpu, True)
        except RuntimeError as e:
            print(f"Warning: {e}")

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Configuration parameters
class Config:
    # Dataset parameters

```

```
dataset_type = "balanced" # Choose from: balanced, byclass, bymerge,
digits, letters, mnist

# Get current script directory to set data_dir correctly
script_dir = os.path.dirname(os.path.abspath(__file__))
data_dir = script_dir # The data files are in the same directory as the script

batch_size = 128
img_height = 28
img_width = 28
validation_split = 0.2

# Model parameters
num_filters = [32, 64, 128]
kernel_size = (3, 3)
pool_size = (2, 2)
dropout_rate = 0.25
dense_units = 256

# Training parameters
learning_rate = 0.001
num_epochs = 20
early_stopping_patience = 5

# Paths
model_save_path = os.path.join(script_dir, 'ocr_model.h5')
results_dir = os.path.join(script_dir, 'results')

# Create results directory if it doesn't exist
os.makedirs(results_dir, exist_ok=True)
```

```

config = Config()

# Print the paths to verify
print(f'Script directory: {config.script_dir}')
print(f'Data directory: {config.data_dir}')
print(f'Model save path: {config.model_save_path}')
print(f'Results directory: {config.results_dir}')

def load_gz_idx_file(file_path):
    """
    Load IDX file format used by EMNIST dataset from a gzipped file.

    Args:
        file_path: Path to the .gz file containing IDX data

    Returns:
        data: Numpy array with the data
    """
    try:
        with gzip.open(file_path, 'rb') as f:
            # First 16 bytes contain header information
            # First 4 bytes: magic number
            # Second 4 bytes: number of items
            # Rest: dimensions
            data = f.read()

            # Parse header
            magic_number = int.from_bytes(data[0:4], byteorder='big')

```

```

# Check file type from magic number
if magic_number == 2051: # Image file
    num_items = int.from_bytes(data[4:8], byteorder='big')
    num_rows = int.from_bytes(data[8:12], byteorder='big')
    num_cols = int.from_bytes(data[12:16], byteorder='big')

    # Image data starts at byte 16
    image_data = data[16:]
    images = np.frombuffer(image_data, dtype=np.uint8)
    images = images.reshape(num_items, num_rows, num_cols)
    return images

elif magic_number == 2049: # Label file
    num_items = int.from_bytes(data[4:8], byteorder='big')

    # Label data starts at byte 8
    label_data = data[8:]
    labels = np.frombuffer(label_data, dtype=np.uint8)
    return labels

else:
    raise ValueError(f"Unknown magic number: {magic_number}")
except Exception as e:
    print(f"Error loading file {file_path}: {e}")
    raise

def create_default_mapping(dataset_type):
    """
    Create a default mapping of class indices to characters based on the dataset
    type.

```

Args:

dataset_type: Type of the dataset (balanced, byclass, bymerge, digits, letters, mnist)

Returns:

class_mapping: Dictionary mapping class indices to characters

"""

class_mapping = {}

if 'digits' in dataset_type:

For digits: 0-9

for i in range(10):

class_mapping[i] = str(i)

elif 'balanced' in dataset_type or 'byclass' in dataset_type or 'bymerge' in dataset_type:

For datasets with digits and letters

Digits (0-9)

for i in range(10):

class_mapping[i] = str(i)

Uppercase letters (A-Z)

for i in range(26):

class_mapping[i + 10] = chr(65 + i)

Lowercase letters (a-z)

for i in range(26):

class_mapping[i + 36] = chr(97 + i)

elif 'letters' in dataset_type:

For letters only: A-Z (uppercase)

for i in range(26):

class_mapping[i] = chr(65 + i)

For letters only: a-z (lowercase)

```

        for i in range(26):
            class_mapping[i + 26] = chr(97 + i)
    else:
        # Default for MNIST: 0-9
        for i in range(10):
            class_mapping[i] = str(i)

    return class_mapping

def find_emnist_files(data_dir, dataset_type):
    """
    Find EMNIST files in the data directory with enhanced error handling.

    Args:
        data_dir: Directory to search
        dataset_type: Type of EMNIST dataset

    Returns:
        dict: Dictionary with paths to train and test files
    """
    print(f'Searching for EMNIST {dataset_type} files in {data_dir}...')

    # List all files in the directory
    try:
        all_files = os.listdir(data_dir)
        print(f'Found {len(all_files)} files in directory')
    except Exception as e:
        print(f'Error listing directory {data_dir}: {e}')
        return None

```

```

# Pattern to look for
train_img_pattern = f"emnist-{{dataset_type}}-train-images-idx3-ubyte.gz"
train_lbl_pattern = f"emnist-{{dataset_type}}-train-labels-idx1-ubyte.gz"
test_img_pattern = f"emnist-{{dataset_type}}-test-images-idx3-ubyte.gz"
test_lbl_pattern = f"emnist-{{dataset_type}}-test-labels-idx1-ubyte.gz"

# Find matching files
train_img_path = None
train_lbl_path = None
test_img_path = None
test_lbl_path = None

for file in all_files:
    if train_img_pattern in file:
        train_img_path = os.path.join(data_dir, file)
    elif train_lbl_pattern in file:
        train_lbl_path = os.path.join(data_dir, file)
    elif test_img_pattern in file:
        test_img_path = os.path.join(data_dir, file)
    elif test_lbl_pattern in file:
        test_lbl_path = os.path.join(data_dir, file)

# Check if we found all required files
files_found = {
    'train_images': train_img_path,
    'train_labels': train_lbl_path,
    'test_images': test_img_path,
    'test_labels': test_lbl_path
}

```

```

print("EMNIST files found:")
for key, path in files_found.items():
    status = "✓ Found" if path else "✗ Not found"
    print(f" {key}: {status}")

    if path:
        print(f"   Path: {path}")

# If any file is missing, try alternative dataset types
if None in files_found.values():
    print("\nSome files were not found. Available dataset types in directory:")
    dataset_types = set()
    for file in all_files:
        if file.startswith('emnist-') and file.endswith('.gz'):
            parts = file.split('-')
            if len(parts) > 1:
                dataset_types.add(parts[1])

    if dataset_types:
        print(f"Found these dataset types: {' '.join(dataset_types)}")
        print("Try changing the dataset_type in the Config class to one of these
values.")
    else:
        print("No EMNIST dataset files found in directory.")

return files_found

def load_and_prepare_data():
    """
    Load and prepare the EMNIST dataset for training and validation.

```

Returns:

x_train: Training images

y_train: Training labels (one-hot encoded)

x_val: Validation images

y_val: Validation labels (one-hot encoded)

x_test: Test images

y_test: Test labels (one-hot encoded)

num_classes: Number of unique classes (characters)

class_mapping: Dictionary mapping class indices to characters

"""

```
print("Loading EMNIST dataset...")
```

```
# Find EMNIST files
```

```
emnist_files = find_emnist_files(config.data_dir, config.dataset_type)
```

```
if not emnist_files or None in emnist_files.values():
```

```
    print("Error: Could not find all required EMNIST files.")
```

```
    sys.exit(1)
```

```
# Load the data
```

```
try:
```

```
    print(f"Loading training images from {emnist_files['train_images']}")
```

```
    x_train = load_gz_idx_file(emnist_files['train_images'])
```

```
    print(f"Loading training labels from {emnist_files['train_labels']}")
```

```
    y_train = load_gz_idx_file(emnist_files['train_labels'])
```

```
    print(f"Loading test images from {emnist_files['test_images']}")
```

```
    x_test = load_gz_idx_file(emnist_files['test_images'])
```

```

    print(f>Loading test labels from {emnist_files['test_labels']}")
    y_test = load_gz_idx_file(emnist_files['test_labels'])
except Exception as e:
    print(f>Error loading dataset: {e}")
    sys.exit(1)

print(f>Training set shape: {x_train.shape}")
print(f>Test set shape: {x_test.shape}")

# EMNIST images are transposed, so we need to rotate them
# This is a known issue with the EMNIST dataset
try:
    x_train = np.rot90(np.fliplr(x_train.reshape(-1, config.img_height,
config.img_width)), k=3, axes=(1, 2))
    x_test = np.rot90(np.fliplr(x_test.reshape(-1, config.img_height,
config.img_width)), k=3, axes=(1, 2))
except Exception as e:
    print(f>Error preprocessing images: {e}")
    sys.exit(1)

# Normalize pixel values to [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape the data to include channel dimension (grayscale = 1 channel)
x_train = x_train.reshape(-1, config.img_height, config.img_width, 1)
x_test = x_test.reshape(-1, config.img_height, config.img_width, 1)

# Split the training data into training and validation sets

```

```

try:
    x_train, x_val, y_train, y_val = train_test_split(
        x_train, y_train, test_size=config.validation_split, random_state=42
    )
except Exception as e:
    print(f"Error splitting dataset: {e}")
    sys.exit(1)

# Create class mapping
class_mapping = create_default_mapping(config.dataset_type)

# Determine the number of classes
num_classes = len(np.unique(np.concatenate((y_train, y_test))))
print(f"Number of classes: {num_classes}")

print(f"Training set size: {x_train.shape[0]}")
print(f"Validation set size: {x_val.shape[0]}")
print(f"Test set size: {x_test.shape[0]}")

# One-hot encode the labels
try:
    y_train_hot = tf.keras.utils.to_categorical(y_train, num_classes)
    y_val_hot = tf.keras.utils.to_categorical(y_val, num_classes)
    y_test_hot = tf.keras.utils.to_categorical(y_test, num_classes)
except Exception as e:
    print(f"Error one-hot encoding labels: {e}")
    print("Trying alternative one-hot encoding method...")

# Alternative one-hot encoding using numpy
def to_categorical(y, num_classes):

```

```

        return np.eye(num_classes)[y]

    y_train_hot = to_categorical(y_train, num_classes)
    y_val_hot = to_categorical(y_val, num_classes)
    y_test_hot = to_categorical(y_test, num_classes)

    return x_train, y_train_hot, x_val, y_val_hot, x_test, y_test_hot,
    num_classes, class_mapping

def create_model(num_classes):
    """
    Create a CNN model for OCR using TensorFlow's Sequential API.

    Args:
        num_classes: Number of classes for the final output layer

    Returns:
        model: Compiled TensorFlow model
    """
    print("Creating CNN model...")

    try:
        model = tf.keras.Sequential([
            # First convolutional block
            tf.keras.layers.Conv2D(
                config.num_filters[0],
                config.kernel_size,
                activation='relu',
                padding='same',
                input_shape=(config.img_height, config.img_width, 1)
            )
        ])

```

```
),  
tf.keras.layers.BatchNormalization(),  
tf.keras.layers.Conv2D(  
    config.num_filters[0],  
    config.kernel_size,  
    activation='relu',  
    padding='same'  
),  
tf.keras.layers.BatchNormalization(),  
tf.keras.layers.MaxPooling2D(config.pool_size),  
tf.keras.layers.Dropout(config.dropout_rate),  
  
# Second convolutional block  
tf.keras.layers.Conv2D(  
    config.num_filters[1],  
    config.kernel_size,  
    activation='relu',  
    padding='same'  
),  
tf.keras.layers.BatchNormalization(),  
tf.keras.layers.Conv2D(  
    config.num_filters[1],  
    config.kernel_size,  
    activation='relu',  
    padding='same'  
),  
tf.keras.layers.BatchNormalization(),  
tf.keras.layers.MaxPooling2D(config.pool_size),  
tf.keras.layers.Dropout(config.dropout_rate),
```

```

# Third convolutional block
tf.keras.layers.Conv2D(
    config.num_filters[2],
    config.kernel_size,
    activation='relu',
    padding='same'
),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Conv2D(
    config.num_filters[2],
    config.kernel_size,
    activation='relu',
    padding='same'
),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.MaxPooling2D(config.pool_size),
tf.keras.layers.Dropout(config.dropout_rate),

# Flatten and fully connected layers
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(config.dense_units, activation='relu'),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(num_classes, activation='softmax')
])

# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=config.learning_rate)
model.compile(
    optimizer=optimizer,

```

```

        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    # Print model summary
    model.summary()

    return model

except Exception as e:
    print(f'Error creating model: {e}')
    sys.exit(1)

def train_model(model, x_train, y_train, x_val, y_val):
    """
    Train the OCR model with error handling.

    Args:
        model: Compiled TensorFlow model
        x_train: Training images
        y_train: Training labels
        x_val: Validation images
        y_val: Validation labels

    Returns:
        history: Training history
    """
    print("Training model...")

    try:

```

```

# Define callbacks
callbacks_list = [
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=config.early_stopping_patience,
        restore_best_weights=True
    ),
    tf.keras.callbacks.ModelCheckpoint(
        filepath=config.model_save_path,
        monitor='val_accuracy',
        save_best_only=True
    ),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=2,
        min_lr=1e-6
    )
]

# Train the model
history = model.fit(
    x_train, y_train,
    batch_size=config.batch_size,
    epochs=config.num_epochs,
    validation_data=(x_val, y_val),
    callbacks=callbacks_list,
    verbose=1
)

```

```
print("Model training complete!")
return history
```

```
except Exception as e:
    print(f"Error during model training: {e}")
    sys.exit(1)
```

```
def evaluate_model(model, x_test, y_test_hot, class_mapping):
    """
```

Evaluate the trained model on the test set with comprehensive error handling.

Args:

```
    model: Trained TensorFlow model
    x_test: Test images
    y_test_hot: Test labels (one-hot encoded)
    class_mapping: Dictionary mapping class indices to characters
```

Returns:

```
    None
```

```
    """
```

```
print("Evaluating model on test set...")
```

```
try:
```

```
    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(x_test, y_test_hot, verbose=1)
    print(f"Test accuracy: {test_accuracy:.4f}")
    print(f"Test loss: {test_loss:.4f}")
```

```
    # Get predictions
```

```

y_pred_prob = model.predict(x_test)
y_pred = np.argmax(y_pred_prob, axis=1)
y_true = np.argmax(y_test_hot, axis=1)

# Classification report
print("Classification Report:")
# Create target_names for classification report
# Ensure we only include classes that exist in the test set
unique_classes = sorted(np.unique(np.concatenate((y_true, y_pred))))
target_names = [class_mapping.get(i, f'Class_{i}') for i in
unique_classes]

print(classification_report(y_true, y_pred, target_names=target_names))

# Confusion matrix
# Use only a subset of classes if there are too many
max_classes_for_cm = 30 # Maximum number of classes to include in
confusion matrix
if len(unique_classes) > max_classes_for_cm:
    # Use only most frequent classes
    class_counts = np.bincount(y_true)
    top_classes = np.argsort(class_counts)[-max_classes_for_cm:]
    mask_true = np.isin(y_true, top_classes)
    mask_pred = np.isin(y_pred, top_classes)
    mask = mask_true & mask_pred

    cm_y_true = y_true[mask]
    cm_y_pred = y_pred[mask]
    cm_classes = [class_mapping.get(i, f'Class_{i}') for i in top_classes]

```

```

cm = confusion_matrix(cm_y_true, cm_y_pred, labels=top_classes)

plt.figure(figsize=(20, 20))
disp = plt.matshow(cm, cmap=plt.cm.Blues)
plt.colorbar(disp)

# Set class labels
plt.xticks(np.arange(len(cm_classes)), cm_classes, rotation=90)
plt.yticks(np.arange(len(cm_classes)), cm_classes)
plt.title('Confusion Matrix (Top Classes)')
else:
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(20, 20))
    disp = plt.matshow(cm, cmap=plt.cm.Blues)
    plt.colorbar(disp)

    # Set class labels
    plt.xticks(np.arange(len(target_names)), target_names, rotation=90)
    plt.yticks(np.arange(len(target_names)), target_names)
    plt.title('Confusion Matrix')

plt.savefig(os.path.join(config.results_dir, 'confusion_matrix.png'),
bbox_inches='tight')
plt.close()

# Visualize some predictions
visualize_predictions(model, x_test, y_true, class_mapping)

except Exception as e:
    print(f'Error during model evaluation: {e}')

```

```

import traceback
traceback.print_exc()

def plot_training_history(history):
    """
    Plot training and validation accuracy and loss.

    Args:
        history: Training history from model.fit()

    Returns:
        None
    """
    print("Plotting training history...")

    try:
        # Plot training & validation accuracy
        plt.figure(figsize=(12, 5))

        # Plot accuracy
        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('Model Accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend(['Train', 'Validation'], loc='lower right')
        plt.grid(True)

        # Plot loss

```

```

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.grid(True)

```

```

plt.tight_layout()
plt.savefig(os.path.join(config.results_dir, 'training_history.png'))
plt.close()

```

```
except Exception as e:
```

```
    print(f'Error plotting training history: {e}')
```

```
def visualize_predictions(model, x_test, y_true, class_mapping):
```

```
    """
```

Visualize some predictions from the model.

Args:

model: Trained TensorFlow model

x_test: Test images

y_true: True labels (indices, not one-hot encoded)

class_mapping: Dictionary mapping class indices to characters

Returns:

None

```
    """
```

```
    print("Visualizing predictions...")
```

```

try:
    # Get predictions for the test set
    y_pred_prob = model.predict(x_test)
    y_pred = np.argmax(y_pred_prob, axis=1)

    # Select 25 random samples from the test set
    indices = np.random.choice(range(len(x_test)), size=25, replace=False)

    # Create a figure
    plt.figure(figsize=(12, 12))

    for i, idx in enumerate(indices):
        # Get the image and predictions
        img = x_test[idx].squeeze()
        true_label = y_true[idx]
        pred_label = y_pred[idx]

        # Get the character representations
        true_char = class_mapping.get(true_label, f'Class_{true_label}')
        pred_char = class_mapping.get(pred_label, f'Class_{pred_label}')

        # Calculate prediction confidence
        confidence = y_pred_prob[idx][pred_label] * 100

        # Plot the image
        plt.subplot(5, 5, i + 1)
        plt.imshow(img, cmap='gray')

        # Set the title with prediction and ground truth
        color = 'green' if true_label == pred_label else 'red'

```

```

plt.title(f"True: {true_char}, Pred: {pred_char}\nConf:
{confidence:.1f}%", color=color)
plt.axis('off')

plt.tight_layout()
plt.savefig(os.path.join(config.results_dir, 'sample_predictions.png'))
plt.close()
except Exception as e:
    print(f"Error visualizing predictions: {e}")

def main():
    """
    Main function to run the training pipeline with comprehensive error
handling.
    """
    print("\n" + "="*50)
    print(" OCR Model Training with TensorFlow - EMNIST Dataset ")
    print("="*50 + "\n")

    start_time = time.time()

    try:
        # Step 1: Load and prepare data
        x_train, y_train, x_val, y_val, x_test, y_test, num_classes, class_mapping
= load_and_prepare_data()

        # Step 2: Create model
        model = create_model(num_classes)

        # Step 3: Train the model

```

```

history = train_model(model, x_train, y_train, x_val, y_val)

# Step 4: Plot training history
plot_training_history(history)

# Step 5: Evaluate the model
evaluate_model(model, x_test, y_test, class_mapping)

# Calculate and print total execution time
end_time = time.time()
execution_time = end_time - start_time
hours, remainder = divmod(execution_time, 3600)
minutes, seconds = divmod(remainder, 60)
print(f'Total execution time: {int(hours)}h {int(minutes)}m
{seconds:.2f}s")

print("\nTraining completed successfully!")
print(f'Model saved to: {config.model_save_path}')
print(f'Results saved to: {config.results_dir}')

except KeyboardInterrupt:
    print("\nTraining interrupted by user.")
except Exception as e:
    print(f'\nAn unexpected error occurred: {e}')
    import traceback
    traceback.print_exc()
    sys.exit(1)

if __name__ == "__main__":
    main()

```

Статистика під час тренування :

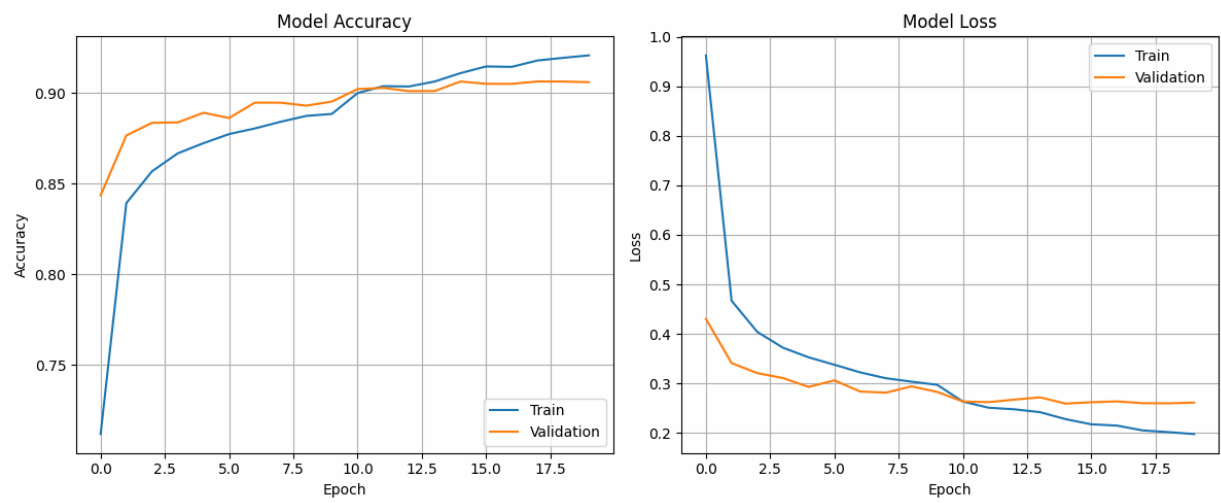


Рисунок Історії тренування моделі



Рисунок 3 демонстрацією точності моделі під час тренування

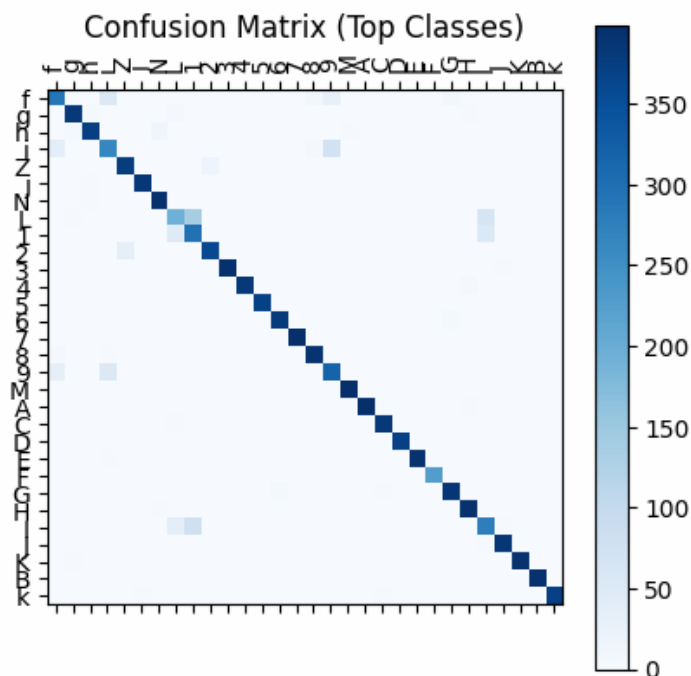


Рисунок матриці тренування

Далі код з файлу Test_ocr_model.py

```
#!/usr/bin/env python3
```

```
# Скрипт для тестування OCR моделі на окремих зображеннях
```

```
# або на тестовому наборі EMNIST
```

```
import os
```

```
import gzip
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from sklearn.metrics import classification_report
```

```
import random
```

```
# Отримання поточної директорії скрипту
```

```
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
```

```
# Шляхи до файлів (використовуємо абсолютні шляхи)
```

```

MODEL_PATH = os.path.join(SCRIPT_DIR, 'ocr_model.h5') # Шлях до
збереженої моделі

TEST_IMAGES_PATH = os.path.join(SCRIPT_DIR, 'emnist-balanced-test-
images-idx3-ubyte.gz') # Тестові зображення

TEST_LABELS_PATH = os.path.join(SCRIPT_DIR, 'emnist-balanced-test-
labels-idx1-ubyte.gz') # Тестові мітки

print(f"Шлях до моделі: {MODEL_PATH}")
print(f"Шлях до тестових зображень: {TEST_IMAGES_PATH}")
print(f"Шлях до тестових міток: {TEST_LABELS_PATH}")

# Перевірка існування файлів
if not os.path.exists(MODEL_PATH):
    print(f"ПОМИЛКА: Файл моделі не знайдено за шляхом
{MODEL_PATH}")

# Пошук моделі в поточній директорії
possible_models = [f for f in os.listdir(SCRIPT_DIR) if f.endswith('.h5')]
if possible_models:
    print(f"Знайдені можливі файли моделей: {possible_models}")
    print(f"Спробуйте використати один з цих файлів, оновивши
значення MODEL_PATH")
else:
    print("Моделі у форматі .h5 не знайдено в поточній директорії.")

if not os.path.exists(TEST_IMAGES_PATH):
    print(f"ПОМИЛКА: Файл тестових зображень не знайдено за шляхом
{TEST_IMAGES_PATH}")

if not os.path.exists(TEST_LABELS_PATH):

```

```
print(f"ПОМИЛКА: Файл тестових міток не знайдено за шляхом
{TEST_LABELS_PATH}")
```

```
# Функція для завантаження IDX файлів
def load_gz_idx_file(file_path):
    """
    Завантажує IDX файл з gzip архіву.
    """
    try:
        with gzip.open(file_path, 'rb') as f:
            data = f.read()

        # Парсимо заголовок
        magic_number = int.from_bytes(data[0:4], byteorder='big')

        if magic_number == 2051: # Файл зображень
            num_items = int.from_bytes(data[4:8], byteorder='big')
            num_rows = int.from_bytes(data[8:12], byteorder='big')
            num_cols = int.from_bytes(data[12:16], byteorder='big')

            # Дані зображень починаються з байту 16
            image_data = data[16:]
            images = np.frombuffer(image_data, dtype=np.uint8)
            images = images.reshape(num_items, num_rows, num_cols)
            return images

        elif magic_number == 2049: # Файл міток
            num_items = int.from_bytes(data[4:8], byteorder='big')

            # Дані міток починаються з байту 8
```

```

        label_data = data[8:]
        labels = np.frombuffer(label_data, dtype=np.uint8)
        return labels
    else:
        raise ValueError(f"Невідомий magic number: {magic_number}")
except Exception as e:
    print(f"Помилка завантаження файлу {file_path}: {e}")
    raise

# Створення відображення класів до символів
def create_default_mapping(dataset_type="balanced"):
    """
    Створює відображення індексів класів до символів.
    """
    class_mapping = {}

    if dataset_type == "balanced" or dataset_type == "byclass" or dataset_type == "bymerge":
        # Цифри (0-9)
        for i in range(10):
            class_mapping[i] = str(i)
        # Великі літери (A-Z)
        for i in range(26):
            class_mapping[i + 10] = chr(65 + i)
        # Малі літери (a-z)
        for i in range(26):
            class_mapping[i + 36] = chr(97 + i)
    elif dataset_type == "letters":
        # Лише літери: A-Z (великі)
        for i in range(26):

```

```

        class_mapping[i] = chr(65 + i)
    # Лише літери: a-z (малі)
    for i in range(26):
        class_mapping[i + 26] = chr(97 + i)
elif dataset_type == "digits":
    # Лише цифри: 0-9
    for i in range(10):
        class_mapping[i] = str(i)
else:
    # За замовчуванням для MNIST: 0-9
    for i in range(10):
        class_mapping[i] = str(i)

return class_mapping

# Завантаження і підготовка тестових даних
def load_test_data():
    """
    Завантажує тестові дані EMNIST.
    """
    print("Завантаження тестових даних...")

    # Завантажуємо тестові зображення та мітки
    x_test = load_gz_idx_file(TEST_IMAGES_PATH)
    y_test = load_gz_idx_file(TEST_LABELS_PATH)

    print(f"Форма тестового набору: {x_test.shape}")

    # Зображення EMNIST транспоновані, тому їх потрібно обертати
    x_test = np.rot90(np.fliplr(x_test.reshape(-1, 28, 28)), k=3, axes=(1, 2))

```

```

# Нормалізуємо значення пікселів до [0, 1]
x_test = x_test.astype('float32') / 255.0

# Додаємо вимір каналу (grayscale = 1 канал)
x_test = x_test.reshape(-1, 28, 28, 1)

return x_test, y_test

# Завантаження моделі
def load_model():
    """
    Завантажує збережену модель OCR.
    """
    print("Завантаження моделі...")
    try:
        model = tf.keras.models.load_model(MODEL_PATH)
        print("Модель успішно завантажена.")
        return model
    except Exception as e:
        print(f"Помилка завантаження моделі: {e}")
        return None

# Оцінка моделі на тестовому наборі
def evaluate_model(model, x_test, y_test, class_mapping):
    """
    Оцінює модель на тестовому наборі.
    """
    print("Оцінка моделі на тестовому наборі...")

```

```

# Визначаємо кількість класів з моделі
num_classes = model.output_shape[-1]
print(f"Кількість класів у моделі: {num_classes}")
print(f"Кількість класів у тестовому наборі: {len(np.unique(y_test))}")

# Перевіряємо, чи відповідають класи в y_test очікуваним класам
моделі
unique_classes = np.unique(y_test)
if max(unique_classes) >= num_classes:
    print("УВАГА: Тестовий набір містить класи, яких немає в моделі!")
    print(f"Максимальний індекс класу в тестовому наборі:
{max(unique_classes)}")
    print(f"Кількість класів у моделі: {num_classes}")
    print("Обмежуємо тестовий набір тільки до класів, які модель може
розпізнати...")

# Вибираємо тільки зразки з класами, які підтримує модель
valid_indices = y_test < num_classes
x_test = x_test[valid_indices]
y_test = y_test[valid_indices]
print(f"Новий розмір тестового набору: {len(x_test)} зразків")

# Оцінка моделі
test_loss, test_accuracy = model.evaluate(x_test,
tf.keras.utils.to_categorical(y_test, num_classes), verbose=1)
print(f"Точність на тестовому наборі: {test_accuracy:.4f}")
print(f"Втрата на тестовому наборі: {test_loss:.4f}")

# Отримуємо прогнози
y_pred = model.predict(x_test)

```

```

y_pred_classes = np.argmax(y_pred, axis=1)

# Виводимо звіт класифікації
print("\nЗвіт класифікації:")
target_names = [class_mapping.get(i, f"Клас_{i}") for i in
range(num_classes)]
print(classification_report(y_test, y_pred_classes,
target_names=target_names))

return y_pred_classes

# Візуалізація прогнозів
def visualize_predictions(x_test, y_test, y_pred, class_mapping,
num_samples=25):
    """
    Візуалізує прогнози моделі на тестових зображеннях.
    """
    print("Візуалізація прогнозів...")

    # Вибираємо випадкові зразки
    indices = random.sample(range(len(x_test)), num_samples)

    # Створюємо сітку для візуалізації
    fig, axes = plt.subplots(5, 5, figsize=(15, 15))
    axes = axes.flatten()

    for i, idx in enumerate(indices):
        img = x_test[idx].squeeze()
        true_label = y_test[idx]
        pred_label = y_pred[idx]

```

```

# Отримуємо символи
true_char = class_mapping.get(true_label, f"Клас_{true_label}")
pred_char = class_mapping.get(pred_label, f"Клас_{pred_label}")

# Зображуємо зображення
axes[i].imshow(img, cmap='gray')

# Встановлюємо заголовок з істинною міткою та прогнозом
color = 'green' if true_label == pred_label else 'red'
axes[i].set_title(f"Істина: {true_char}\nПрогноз: {pred_char}",
color=color)
axes[i].axis('off')

plt.tight_layout()
predictions_path = os.path.join(SCRIPT_DIR,
'predictions_visualization.png')
plt.savefig(predictions_path)
print(f"Візуалізацію збережено у {predictions_path}")
plt.show()

# Передбачення для окремого зображення
def predict_single_image(model, image_path, class_mapping):
    """
    Робить прогноз для окремого зображення.
    """
    try:
        # Завантажуємо зображення
        img = tf.keras.preprocessing.image.load_img(image_path,
color_mode='grayscale', target_size=(28, 28))

```

```
img_array = tf.keras.preprocessing.image.img_to_array(img)
```

```
# Попередня обробка зображення
```

```
img_array = img_array / 255.0
```

```
img_array = img_array.reshape(1, 28, 28, 1)
```

```
# Робимо прогноз
```

```
prediction = model.predict(img_array)
```

```
pred_class = np.argmax(prediction)
```

```
confidence = np.max(prediction) * 100
```

```
# Отримуємо символ
```

```
predicted_char = class_mapping.get(pred_class, f"Клас_{pred_class}")
```

```
print(f"Прогноз: {predicted_char} (Впевненість: {confidence:.2f}%)")
```

```
# Показуємо зображення з прогнозом
```

```
plt.figure(figsize=(6, 6))
```

```
plt.imshow(img_array.reshape(28, 28), cmap='gray')
```

```
plt.title(f"Прогноз: {predicted_char}\nВпевненість: {confidence:.2f}%)")
```

```
plt.axis('off')
```

```
plt.show()
```

```
return predicted_char, confidence
```

```
except Exception as e:
```

```
    print(f"Помилка при прогнозуванні зображення: {e}")
```

```
    return None, 0
```

```
# Тестування на вибірці зображень з тестового набору
```

```
def test_on_sample(model, x_test, y_test, class_mapping, num_samples=10):
```

```

"""
Тестує модель на вибірці з тестового набору.
"""

print(f"Тестування на {num_samples} випадкових зображеннях з
тестового набору...")

# Вибираємо випадкові зразки
indices = random.sample(range(len(x_test)), num_samples)

for i, idx in enumerate(indices):
    img = x_test[idx:idx+1] # Вибираємо одне зображення
    true_label = y_test[idx]
    true_char = class_mapping.get(true_label, f"Клас_{true_label}")

    # Робимо прогноз
    prediction = model.predict(img, verbose=0)
    pred_class = np.argmax(prediction)
    confidence = np.max(prediction) * 100
    pred_char = class_mapping.get(pred_class, f"Клас_{pred_class}")

    # Вивід результату
    result = "✓" if pred_class == true_label else "✗"
    print(f"Зразок {i+1}: Істина={true_char}, Прогноз={pred_char},
Впевненість={confidence:.2f}% {result}")

# Зображуємо зображення з прогнозом
plt.figure(figsize=(3, 3))
plt.imshow(img.reshape(28, 28), cmap='gray')

# Встановлюємо заголовок з істинною міткою та прогнозом

```

```

        color = 'green' if pred_class == true_label else 'red'
        plt.title(f'Істина: {true_char}\nПрогноз: {pred_char}', color=color)
        plt.axis('off')
        sample_path = os.path.join(SCRIPT_DIR,
f'sample_prediction_{i+1}.png')
        plt.savefig(sample_path)

    print(f'Тестування завершено. Зображення збережені у директорії
{SCRIPT_DIR}")

def main():
    # Завантажуємо відображення класів
    class_mapping = create_default_mapping("balanced")

    # Перевіряємо наявність файлів перед початком
    for path, desc in [(MODEL_PATH, "моделі"), (TEST_IMAGES_PATH,
"тестових зображень"),
        (TEST_LABELS_PATH, "тестових міток")]:
        if not os.path.exists(path):
            print(f"ПОМИЛКА: Файл {desc} не знайдено за шляхом: {path}")
            if desc == "моделі":
                print("Переконайтеся, що файл ocr_model.h5 існує у вашій
директорії.")
            return

    # Завантажуємо модель
    model = load_model()
    if model is None:
        print("Не вдалося завантажити модель. Вихід...")
        return

```

```

# Завантажуємо тестові дані
try:
    x_test, y_test = load_test_data()
except Exception as e:
    print(f"Помилка завантаження тестових даних: {e}")
    return

# Меню вибору дії
while True:
    print("\n--- Меню тестування OCR моделі ---")
    print("1. Оцінити модель на всьому тестовому наборі")
    print("2. Візуалізувати випадкові прогнози")
    print("3. Протестувати на вибірці випадкових зображень")
    print("4. Передбачити символ для користувацького зображення")
    print("0. Вихід")

    choice = input("Виберіть опцію: ")

    if choice == "1":
        y_pred = evaluate_model(model, x_test, y_test, class_mapping)
    elif choice == "2":
        if 'y_pred' not in locals():
            print("Прогнозування...")
            y_pred = model.predict(x_test)
            y_pred = np.argmax(y_pred, axis=1)
            visualize_predictions(x_test, y_test, y_pred, class_mapping)
    elif choice == "3":
        try:

```

```

        num_samples = int(input("Введіть кількість зразків для
тестування: "))
        test_on_sample(model, x_test, y_test, class_mapping, num_samples)
    except ValueError:
        print("Введіть ціле число!")
    elif choice == "4":
        image_path = input("Введіть шлях до зображення: ")
        if os.path.exists(image_path):
            predict_single_image(model, image_path, class_mapping)
        else:
            print(f"Файл не знайдено: {image_path}")
    elif choice == "0":
        print("Завершення програми...")
        break
    else:
        print("Невірний вибір. Спробуйте ще раз.")

if __name__ == "__main__":
    main()

```

Зображення роботи моделі а саме розпізнання рукописних символів

