НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

«На правах рукопису»
УДК 004.93:004.8

До захисту допущено
В.о. завідувача кафедри
_____ Іван ТЕРЕЩЕНКО
«__»_____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

зі спеціальності: 113 Прикладна математика

на тему: «**Методи попередньої обробки зображень історичних документів для підвищення якості розпізнавання**»

Виконала: студентка IV курсу, групи ФІ-21
Денисенко Анастасія Володимирівна _____
Керівник: доктор філософії, асистент
Железняков Дмитро Валентинович _____
Рецензент: доцент кафедри інформаційної безпеки
НН ФТІ КПІ ім. Ігоря Сікорського,
канд. техн. наук, Гальчинський Леонід Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ — 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ
ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти — перший (бакалаврський)
Спеціальність (освітня програма) — 113 Прикладна математика,
ОПП «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Іван ТЕРЕЩЕНКО
«__»_____ 2026 р.

ЗАВДАННЯ
на дипломну роботу

Студентка: Денисенко Анастасія Володимирівна

1. Тема роботи: «Методи попередньої обробки зображень історичних документів для підвищення якості розпізнавання»,
керівник: ассистент Железняков Д. В.,
затверджені наказом по університету № __ від «__» _____ 2026 р.

2. Термін подання студентом роботи: 7 червня 2026 р.

3. Вихідні дані до роботи: Оприлюднені зображення історичних документів з транскрипціями, публікації з застосуванням нейронних мереж для покращення якості таких зображень.

4. Зміст роботи: Виконано аналіз сучасних методів обробки історичних документів для підвищення якості розпізнавання. Проведено порівняння класичних методів бінаризації та методів з використанням нейронних мереж з

архітектурою U-Net. Отримані результати було проаналізовано з використанням метрик оцінки якості зображень та рівня помилок розпізнавання тексту на рівні слів та символів. Здійснено програмну реалізацію моделей та класичних методів на мові Python.

5. Перелік ілюстративного матеріалу: Презентація доповіді

6. Дата видачі завдання: 26 серпня 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Формування та уточнення плану роботи	26 серпня 2025	Виконано
2	Мета дослідження, постановка задачі	20 вересня 2025	Виконано
3	Огляд області дослідження та існуючих підходів	15 листопада 2025	Виконано
4	Опис архітектур нейронних мереж	12 січня 2026	Виконано
5	Навчання і тестування моделі	20 травня 2026	Виконано
6	Підведення підсумків, оформлення пояснювальної записки	30 травня 2026	Виконано

Студентка _____ Денисенко А.В.

Керівник роботи _____ Железняков Д.В.

РЕФЕРАТ

Дипломна робота містить 68 сторінок, 1 додаток, 10 зображення, 3 таблиці, і посилається на 26 джерел.

Дипломна робота присвячена дослідженню та порівняльному аналізу класичних та нейромережевих методів бінаризації зображень історичних документів для підвищення якості подальшого розпізнавання тексту (OCR). У роботі проаналізовано типи підходів за архітектурою алгоритмів, а також види деградацій історичних документів (плями, вицвітання, прояв тексту зі зворотної сторони). Окрема увага приділяється вибору функцій втрат при навчанні моделей, що дозволяє суттєво підвищити точність збереження деталей символів та особливостей шрифтів.

Основу дослідження становлять класичні адаптивні методи Оцу (Otsu) і Саувола (Sauvola), а також архітектури глибокого навчання: U-Net, Residual U-Net та Attention U-Net. Для навчання та оцінки нейромережевих моделей було застосовано різні функції втрат: MSE, комбіновану BCE+Dice loss та VGG16 perceptual loss. Реалізацію всіх методів виконано мовою Python із використанням бібліотек PyTorch, OpenCV, NumPy та Scikit-image. Для підвищення стійкості моделей до пошкоджень було реалізовано модуль для додавання синтетичних пошкоджень (гаусів шум, розмиття та імітація протікання чорнил).

Було проведено аналіз методів бінаризації при обробці історичних документів з різними синтетичними пошкодженнями. Проаналізовано залежності якості бінаризації від специфіки архітектури та обраної функції втрат за метриками якості зображення (PSNR, F-Measure, SSIM) та відсоток помилок символів (CER/WER) після етапу розпізнавання тексту.

Ключові слова: бінаризація зображень, історичні документи, розпізнавання тексту, OCR, U-Net, метод Саувола, метод Оцу, функції втрат, перцептивна втрата (perceptual loss).

ABSTRACT

The qualification work contains: 68 pages, 1 appendix, 10 images, 3 tables, and refers to 26 sources.

The thesis is devoted to the study and comparative analysis of classical and neural network methods for binarization of images of historical documents to improve the quality of subsequent text recognition (OCR). The article analyzes the types of approaches by algorithm architecture, as well as types of degradation of historical documents (spots, fading, bleed through). Special attention is paid to the choice of loss functions when training models, which allows to significantly increase the accuracy of preserving character details and font features.

The basis of the research is the classical adaptive methods of Otsu and Sauvola, as well as deep learning architectures: U-Net, Residual U-Net and Attention U-Net. Various loss functions were used to train and evaluate neural network models: MSE, combined BCE+Dice loss and VGG16 perceptual loss. All methods were implemented in Python using PyTorch, OpenCV, NumPy and Scikit-image libraries. To increase the robustness of models to damage, a module was implemented for adding synthetic damage (Gaussian noise, blurring and simulated ink leakage).

An analysis of binarization methods for processing historical documents with various synthetic damage was conducted. The dependence of binarization quality on the architecture specifics and the selected loss function was analyzed by image quality metrics (PSNR, F-Measure, SSIM) and the percentage of character errors (CER/WER) after the text recognition stage.

Keywords: image binarization, historical documents, text recognition, OCR, U-Net, Sauvola method, Otsu method, loss functions, perceptual loss.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

OCR – Optical Character Recognition (Оптичне Розпізнавання Символів)

MSE – Mean Squared Error (Середньоквадратична похибка)

SOTA – State Of Art (Найкращі, найефективніші рішення)

PSNR – Peak signal-to-noise ratio (Пікове співвідношення сигналу до шуму)

SSIM – Structural Similarity Index Measure (Структурна подібність)

GAN – Generative Adversarial Network (Генеративна змагальна мережа)

Loss function – Функція втрат

CER – Character Error Rate (Рівень помилок на символах)

WER – Word Error Rate (Рівень помилок на словах)

DIBCO – Document Image Binarization COmpetition (Змагання з бінаризації зображень документів)

HTR – Handwritten Text Recognition (Розпізнавання Рукописного Тексту)

BCE – Binary Cross-Entropy (Бінарна Кросс-Ентропія)

CNN – Convolutional Neural Network (Згорткова Нейронна Мережа)

FCNN – Fully Convolutional Neural Networks (Повністю згорткові нейронні мережі)

1 ВСТУП

Актуальність дослідження. Історичні документи є важливим джерелом інформації про події минулого, традиції та культуру. Для того, щоб зберегти їх (у випадках, наприклад, наявності плісняви), та зробити більш доступними для дослідників, варто цифровізувати їх. Ці документи часто мають значні фізичні ушкодження: плями, вицвітання чорнила, текстурні неоднорідності паперу та ефект протікання тексту зі зворотної сторінки (bleed-through). Наявність таких дефектів знижує точність роботи систем оптичного розпізнавання тексту (OCR). Тому важливим етапом обробки таких зображень є бінаризація — відділення корисного тексту від неоднорідного фону.

Традиційні локально-адаптивні методи бінаризації, такі як алгоритми Оцу (Otsu) та Сауволи (Sauvola), показують гарні результати на документах з рівномірним фоном, але вони часто показують незадовільні результати для документів з поганим освітленням, пошкодженим папером, розмиттям та ін.. Тому зараз активно розвиваються методи з застосуванням архітектур глибокого навчання, зокрема нейромережі сімейства U-Net (класичний U-Net, Residual U-Net та Attention U-Net). Як і в кожної нейронної мережі, їх ефективність залежить від конфігурації та обраної функції втрат (Loss function), в цій роботі розглядаються середньоквадратична помилка (MSE), бінарна крос-ентропія з коефіцієнтом Дайса (BCE+Dice loss), та перцептивна функція втрат на базі моделі VGG16 (Perceptual loss).

З огляду на це, виникає необхідність систематичного порівняльного аналізу класичних підходів та сучасних нейромережевих архітектур із різними функціями втрат. Це дозволить визначити оптимальні конфігурації для очищення зображень від пошкоджень та подальшого точного розпізнавання тексту.

Метою дослідження є порівняльний аналіз класичних алгоритмів та нейромережевих архітектур обробки зображень для бінаризації історичних документів з метою підвищення якості розпізнавання тексту.

Об’єктом дослідження є історичні документи, що містять ознаки фізичних чи цифрових пошкоджень.

Предметом дослідження є класичні методи локальної бінаризації (Оцу, Саувола), архітектури нейронних мереж (U-Net, Residual U-Net та Attention U-Net) та функції втрат (MSE, BCE+Dice, VGG16 perceptual loss) для покращення якості тексту.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Проаналізувати характерні дефекти зображень історичних документів, що ускладнюють процес автоматичного розпізнавання тексту.
2. Дослідити класичні методи бінаризації Оцу та Сауволи.
3. Обґрунтувати вибір нейромережевих архітектур (U-Net, Residual U-Net, Attention U-Net) та дослідити вплив різних функцій втрат на якість створення бінарних зображень.
4. Розробити програмне забезпечення для реалізації, навчання та тестування обраних моделей і алгоритмів.
5. Провести серію експериментів на наборах даних історичних документів, щоб оцінити результати за метриками візуальної відповідності (SSIM, PSNR) та якості бінаризації (F-measure).
6. Виконати етап оптичного розпізнавання тексту (OCR) на очищених зображеннях та оцінити точність за метриками помилок на рівні символів (CER) і слів (WER).
7. Сформулювати висновки щодо методів бінаризації.

При розв’язанні поставлених завдань використовувалися такі методи дослідження:

- методи цифрової обробки зображень та фільтрації (для реалізації класичних алгоритмів та створення синтетичних деградацій);
- методи глибокого навчання (для проектування та навчання моделей U-Net);
- методи оцінки якості зображень (F-measure, PSNR, SSIM) та метрики точності OCR;
- експериментальні методи порівняльного аналізу на тестових наборах історичних документів.

Практичне значення отриманих результатів полягає у створенні та тестуванні програмного комплексу, який дозволяє автоматизувати процес якісної бінаризації складних історичних документів. Результати дослідження дозволяють визначити, які конфігурації нейромереж та Loss-функцій забезпечують найменший рівень помилок розпізнавання (CER/WER). Отримані моделі можна застосувати для покращення якості систем OCR та їх спрощення, адже якщо застосовувати попередню обробку зображень, OCR системи не потрібно буде навчати на складних зображеннях та архітектура їх може бути більш простою.

Отримані рішення та рекомендації можуть бути використані у таких сферах:

1. Цифровізація та створення електронних архівів для бібліотек, музеїв та наукових інституцій.
2. Автоматизація потокового розпізнавання архівних документів зі складними дефектами паперу.

Розроблений програмний комплекс з використанням машинного навчання може слугувати основою для створення більш масштабних систем інтелектуального аналізу документів, наприклад, доповнений моделлю-трансформером, що буде використовувати контекст тексту для доповнення втрачених символів.

2 ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ

2.1 Методи попередньої обробки, бінаризації та реставрації зображень

Існує багато методів обробки зображень з метою підвищення якості оптичного розпізнавання символів (Optical Character Recognition, OCR). Вони поділяються на класичні та методи з застосуванням нейронних мереж. До класичних методів відносяться: бінаризація Оцу [1] чи Саувола [2], нормалізація, гістограмне вирівнювання та інші. До нейромережевих відносяться такі методи, як, наприклад, моделі, що очищають зображення від пошкоджень або трансформерні моделі, що використовують контекст для відновлення втрачених символів. В цьому розділі буде описано різні підходи, що застосовуються для покращення якості розпізнавання тексту.

2.1.1 Згорткові нейронні мережі та архітектура U-Net

Однією з найбільш розповсюджених архітектур нейронних мереж є Згорткова нейронна мережа (Convolutional Neural Network, CNN). Вони також широко застосовуються в роботі з зображеннями з текстом. U-Net є підвидом CNN, що відрізняється застосуванням пропускних зв’язків між контрактуючим та розширювальними шляхами. Цікавою варіацією U-Net є Повністю згорткові нейронні мережі (Fully Convolutional Neural Networks, FCNN), описані в роботі C. Tensmeyer і T. Martinez [3]. Вони чергують операції згортки та нелінійні операції для ефективної класифікації всіх пікселів вхідного зображення за один прямий прохід. Автори використовували функцію втрат, засновану на оптимізації метрики F-міри. Модель показала прекрасні результати на змаганнях DIBCO [4], що свідчить про ефективність такої архітектури для бінаризації зображень.

2.1.2 **Генеративно-змагальні мережі (GAN)**

Мережі GAN широко застосовуються в комп’ютерному зорі. Наприклад, Tamrin et al. [5] у своїй роботі показали, що такий підхід є конкурентоспроможним навіть в порівнянні до найкращих на даний момент методів (SOTA). Особливістю цієї роботи є двоетапність тренування: навчаються дві мережі, одна з яких вчиться генерувати пошкоджене зображення, а інша – очищувати згенероване зображення. Це вирішує розповсюджену проблему в сфері роботи з архівними документами: нестачу даних для якісного навчання моделей.

Іншим прикладом застосування GAN є робота Bhunia et al. [6]. В цій статті описана мережа, яка переносить текстуру деградованого зображення на чисте бінарне зображення. Створюється кілька версій одного й того ж текстового контенту з різними пошкодженнями, що дозволяє навчити модель усувати різні типи деградації та подолати проблему нестачі даних. Ці згенеровані зображення пропускаються через мережу бінаризації. Така модель може навчатися на непарних даних.

2.1.3 **Багатоспектральний аналіз та глибоке навчання**

Особливо складною є робота з палімпсестами - документами, де початковий текст було стерто і переписано на інший. Новітній метод з застосуванням спектрального аналізу, описаний Starynska [7], дозволяє побачити перезаписаний текст під дією певної частини електромагнітного спектрального освітлення. Глибокі нейронні мережі (засновані на архітектурі GAN) в поєднанні з просторовою статистикою палімпсестного тексту дають можливість отримати історичні дані, зчитування яких раніше вважалося неможливим.

2.1.4 **Моделі з багатозадачним навчанням**

Багатозадачне навчання (Multi-task learning, MTL) – тип машинного навчання, за якого модель навчається виконувати кілька споріднених задач одночасно. У роботі He S., Schomaker L. [8] описано T-подібну мережу, адаптовану для задачі покращення зображень історичних документів. T-подібна мережа отримала таку назву через свою «T-подібну» структуру, де початковий спільний енкодер розгалужується на два окремі декодери для одночасної обробки двох завдань. Таким чином, запропонована модель не тільки бінаризує зображення, але й покращує його якість. Нейронна мережа для покращення вивчає деградації зображень документів, а ядра згорткової нейронної мережі адаптується до завдання бінаризації в процесі навчання. Покращене зображення після цього подається в мережу для точного налаштування (fine-tuning), що дозволяє розробляти ланцюгову каскадну мережу (CT-Net).

2.1.5 **Дифузійні моделі**

Дифузійні моделі – це ланцюги Маркова, навчені за допомогою варіаційного висновку. Вони вивчають структуру даних, моделюючи розсіювання точок в латентному просторі, в контексті роботи з зображеннями ці моделі навчаються зашумлювати зображення, що були розмиті гаусівським шумом шляхом навчання зворотному процесу дифузії. Прикладом такої моделі, застосованої до задачі роботи з історичними документами є модель дифузії з усуненням шуму (Diffusion Denoising Restoration Model, DDRM), запропонована в роботі Yoshizu, Y. et al. [9] , що зосереджена на роботі з японськими історичними рукописами. Вона використовує двоетапний процес генерації масок, що поєднує автоматичне вирівнювання кольорів (Automatic Color Equalization, ACE) та кластеризацію суміші гаусівських розподілів (Gaussian Mixture Model, GMM) в кількох колірних просторах. Наукова новизна цього підходу полягає в використанні бінаризованих

масок тексту як напрямних сигналів для DDRM за допомогою маскування шуму. Це обмежує процес дифузії суворо контурами символів та ефективно пригнічує фоновий шум, тобто модель не тільки підвищує якість тексту, а й відновлює текстуру фону, тоді як більшість моделей відновлюють лише текст. Автори провели порівняння представленого методу з традиційною дифузійною моделлю на кількох японських манускриптах, і експериментально підтвердили його перевагу.

2.2 Моделі оптичного розпізнавання тексту (OCR / HTR)

Для того, щоб розробити алгоритм обробки зображень, що буде підвищувати якість розпізнавання символів, варто розуміти, як саме працюють ці системи. Деякі системи працюють на цілих сторінках, інші потребують розділення даних на рядки, деякі системи чутливі до мови, або погано працюють на певних шрифтах. Застосування системи розпізнавання, що не пристосована до потреб конкретного документу, може значно погіршити отримані результати.

2.2.1 **Гібридні рекурентно-згорткові мережі (CRNN)**

Стаття Yousef і Bishop [10] є корисною, оскільки вона містить посилання на код моделі, що дозволяє отримати більш глибоке розуміння роботи OCR моделей. Як було згадано вище, деякі системи вимагають, щоб зображення було розділено на рядки. OrigamiNet було створено для розпізнавання тексту з цілої сторінки, без сегментації, що спрощує процес розпізнавання для користувача. Це досягається за допомогою розгортання вхідного багаторядкове зображення в однорядкове, при чому всі рядки вихідного зображення зшиваються в один довгий рядок. Фактично, сегментація та розпізнавання виконуються за одне пряме проходження мережі.
Автори підкреслюють, що деякі методи, що орієнтовані на роботу з сегментованим текстом, вимагають проходження 500 ітерацій на одне зображення сторінки, тоді як представлений метод вимагає тільки однієї, що дозволяє знизити споживання обчислювальних ресурсів та час обробки документу.

2.2.2 **Трансформери у комп'ютерному зорі (Vision Transformers)**

Vision transformer (ViT) [11] – архітектура моделі, що помічає зв’язки між різними частинами зображення, використовуючи механізм уваги, що дозволяє моделі вивчати просторові ознаки тексту. У випадку OCR модель ViT отримує на вхід зображення з текстом та обробляє його за допомогою шарів енкодера Transformer. Енодер складається з багатоголового механізму уваги, який дозволяє моделі звертати увагу на різні частини зображення, одночасно отримуючи контекстну інформацію з тексту. Виходом енкодера є послідовність векторів ознак, які містять семантичну інформацію тексту. Вектори ознак потім подаються в мережу декодерів, яка зіставляє їх з відповідними символами або словами. Перевагою використання моделі ViT для завдань OCR є те, що вона може фіксувати довгострокові залежності та контекстну інформацію тексту, що є позитивно впливає на якість розпізнавання.

Цікава модифікація ViT була представлена у роботі Li Y. et al. [12]: замість використання вбудовування патчів для генерації вхідних токенів, використовується ResNet модель для вилучення проміжних візуальних представлень ознак як вхідних токенів. Також використовується мінімізація з урахуванням різкості (Sharpness-Aware Minimization, SAM) [13] як оптимізатора, що змушує модель збігатися до пологих мінімумів, а випадкова заміна проміжних токенів на навчальні допомагає уникнути перенавчання та досягти стабільного покращення роботи моделі. Створена модель отримала назву HTR-VT, і її було протестовано на наборі даних Ludovico Antonio Muratori (LAM) – манускрипту італійською мовою [14]. В результаті експериментів модель HTR-VT показала значно кращі результати, ніж стандартний ViT. Автори дослідження наголошують, що внесені до існуючої архітектури зміни були мінімальними та не значно ускладнюють тренування моделі, що підтверджує доцільність застосування їх підходу.

2.3 Пост-корекція помилок розпізнавання (Post-OCR Correction)

Іноді навіть після застосування методів попередньої обробки не вдається відновити текст повністю, наприклад, через розриви паперу, велику кількість плям і інших пошкоджень. Тоді варто використовувати методи пост-корекції помилок розпізнавання, що дозволяє відновити текст за допомогою контексту, мовних моделей та інших методів.

2.3.1 **Огляд методів, наборів даних та метрик**

В роботі Jatowt A. et al. [15] було виконано опис існуючих методів пост-обробки, датастетів та бенчмарків. Автори розділяють існуючі підходи на ручні, напівавтоматичні підходи для ізольованих слів (Isolated-word) та напівавтоматичні контекстно-залежні підходи (Context-dependent). Ручні підходи спираються на платформи, що дозволяють людям вручну оцифровувати текст. Підходи для ізольованих слів аналізують окреме слово без урахування його контексту. Вони здатні виявляти та виправляти переважно "не-слова" (non-word errors) — послідовності символів, яких немає у словнику. Контекстно-залежні підходи враховують сусідні слова, що дозволяє виправляти помилки, коли розпізнане слово існує, але вжите в неправильному контексті (real-word errors).
Автори також описують стандартні методи оцінки якості роботи методів: Precision, Recall, F-score CER та WER. Згадуються численні манускрипти різними мовами (англійською, німецькою, французькою тощо) з різних епох, наприклад, GT4HistOCR [16] та Text+Berg [17].

2.3.2 **Моделі архітектури Sequence-to-Sequence (Seq2Seq)**

Seq2seq – підхід в машинному навчанні, що застосовується в обробці природної мови (Natural Language Processing, NLP). Мережа складається з енкодера та декодера, що приймають на вхід на подають на вихід послідовності символів, звідси назва (послідовність до послідовності – Sequence-to-Sequence). В роботі Asselborn et al. [18] розглядається застосування такого методу до пост-обробки результатів застосування OCR, при чому підхід TOOL (Treating OCR Output as a Language - обробка виводу OCR як мови) розуміє корекцію OCR як завдання машинного перекладу. Це означає, що модель сприймає текст з помилками, що виникли при OCR, як окрему мову, і намагається «перекласти» її на мову без помилок. Перевагою такого підходу є незалежність від мови.

2.3.3 **Контекстуальні мовні моделі та відновлення змісту**

Як було зазначено вище, для виправлення помилок OCR можна застосовувати контекст, щоб передбачити, як мав виглядати втрачений текст. Assael et al. [19] у своїй роботі описують модель PYTHIA, що також заснована на архітектурі Sequence-to-Sequence, але також застосовує елементи архітектури LSTM (Long short-term memory, Довга короткочасна пам'ять), та механізм уваги. Застосування запропонованого методу дозволило знизити рівень помилок на майже 20%, що підтверджує ефективність таких підходів.

Таблиця 2.1. Порівняння методів обробки документів з текстом

Етап обробки	Метод / Архітектура	Особливості та принцип роботи	Переваги	Проблеми, які вирішує / Обмеження
1. Попередня обробка, бінаризація та реставрація	Класичні методи (Оцу, Саувола, нормалізація, гістограмне вирівнювання)	Математичні алгоритми розділення тексту і фону, вирівнювання яскравості без нейромереж.	Прості у реалізації, але поступаються нейромережам на складних документах.	Базове підвищення якості для систем розпізнавання (OCR).
	U-Net та FCNN (Повністю згорткові мережі)	Використовують пропускні зв'язки; чергують згортки та нелінійні операції. Оптимізують метрику F-міри за один прямий прохід.	Хороші результати на змаганнях DIBCO; ефективна класифікація кожного пікселя.	Ефективна бінаризація зображень з текстом.
	Генеративно-змагальні мережі (GAN)	Підхід 1: Двоетапне тренування (одна мережа генерує пошкодження, інша — очищує). Підхід 2: Перенос текстури деградованого зображення на чисте бінарне.	Працює на рівні SOTA; можливість навчання на непарних даних.	Вирішує проблему нестачі даних для архівних та історичних документів.
	Багатоспектральний аналіз + Глибоке навчання (GAN)	Поєднання зйомки під різними хвилями електромагнітного спектра та просторової статистики тексту.	Дозволяє зчитувати історичні дані, які раніше вважалися повністю втраченими.	Робота з палімпсестами (де початковий текст було змито/записано зверху).
	Багатозадачне навчання (MTL / CT-Net)	Т-подібна мережа зі спільним енкодером та двома декодерами. Виконує дві задачі одночасно.	Одночасно покращує якість (прибирає деградації) та бінаризує зображення. Каскадне	Комплексна підготовка складних історичних документів.

			налаштування (fine-tuning).	
	Дифузійні моделі (DDRM)	Ланцюги Маркова. Використовують бінаризовані маски тексту як напрямні сигнали для усунення шуму вздовж контурів символів.	Відновлює не лише чіткість тексту, а й якісно реставрує текстуру фону (на відміну від інших моделей).	Ефективне пригнічення складного фонового шуму в історичних японських рукописах.
2. Оптичне розпізнавання (OCR / HTR)	Гібридні рекурентно-згорткові мережі (OrigamiNet)	Розгортає багаторядкову сторінку в один довгий рядок.Сегментація та розпізнавання відбуваються за один прохід.	Потребує лише 1 ітерацію замість ~500. Заощаджує обчислювальні ресурси та час.	Розпізнавання цілої сторінки без попереднього поділу на рядки.
	Трансформери у комп'ютерному зорі (ViT / HTR-VT)	Механізм уваги (Attention) аналізує зв'язки між частинами зображення. Модифікація HTR-VT додає ResNet-енкодер та оптимізатор SAM.	Фіксує довгострокові залежності та контекст. HTR-VT на манускриптах LAM показала результати, значно кращі за стандартний ViT.	Стабільне навчання без перенавчання; краще розуміння просторових ознак тексту.
3. Пост-корекція помилок розпізнавання	Sequence-to-Sequence (Seq2Seq / TOOL)	Підхід "Treating OCR Output as a Language": сприймає текст із помилками OCR як одну мову, а чистий текст — як іншу.	Абсолютна незалежність від мови тексту.	Виправлення залишків помилок (через розриви паперу, плями) шляхом "машинного перекладу".
	Контекстуальні мовні моделі (PYTHIA)	Архітектура Seq2Seq, підсилена елементами LSTM та механізмом уваги.	Знижує рівень помилок розпізнавання майже на 20%.	Відновлення втраченого змісту та виправлення символів на основі контексту.

Висновки до розділу

Підводячи підсумки, можна сказати, що нейромережеві методи попередньої обробки перевершують традиційні. Найчастіше застосовуються згорткові нейронні мережі, генеративно-змагальні мережі та трансформерні мережі. Хоча ці моделі дозволяють знизити відсоток помилок при розпізнаванні, вони все ще мають різноманітні недоліки (складність навчання, низька якість отриманих результатів, необхідність великих обчислювальних ресурсів), тому варто проводити подальші дослідження для знаходження відносно простих та одночасно ефективних методів.

3 ПОСТАНОВКА ЗАДАЧІ. ДАНІ І МЕТОДИ

3.1 Формалізація задачі підвищення якості зображень документів

Задача підвищення якості зображень документів полягає в тому, щоб застосувати певні методи усунення наявних деградацій, в залежності від типу пошкодження, наприклад: медіанні фільтри, гістограмне вирівнювання – доцільні при поганому освітленні документу, нейронні мережі для усунення плям, плісняви, бінаризація для документів з пожовтінням сторінок, контекстуальні моделі для сторінок з відірваними частинами і так далі.

В цій роботі будуть застосовуватися нейронні моделі типу U-Net, що працюють в форматі image-to-image, тобто приймають на вхід чорно-біле зображення та навчаються бінаризовувати його таким чином, щоб усунути одразу кілька деградацій: розмиття, плями чорнил та шум. Ці деградації створюються синтетично за формулою:

$$x = D(y) + \varepsilon$$

де y – еталонне зображення, x – результуюче зображення з пошкодженнями, D – оператор пошкоджень, ε – випадковий шум.

Оператор пошкоджень реалізовано наступним чином:

$$D(y) = B*K + I_{nonlin}$$

де B – матриця розмиття, K – ядро згортки, I_{nonlin} – нелінійні спотворення, наприклад просочування чорнила.

Таким чином, задача відновлення формулюється як пошук оператора:

$$\widehat{y} = R(x)$$

який забезпечує мінімальну відмінність між відновленим зображенням $\widehat{y}$

та еталонним зображенням y. В цьому випадку оператор R апроксимується параметризованою функцією f_θ, реалізованою у вигляді глибокої нейронної мережі.

Таким чином, задача зводиться до розв'язання оптимізаційної проблеми:

$$\theta^* = arg\ min_\theta E_{(x,y)\sim P}\left[L(f_\theta(x), y)\right]$$

L – функція втрат, P – спільний розподіл пар деградованих та еталонних зображень.

3.2 Типи даних та формат подання зображень

Вхідні дані складаються з RGB (Red, Green and Blue – червоний, зелений та синій) зображень з текстом. Для того, щоб стабілізувати навчання та забезпечити коректну роботу функцій активації, зображення переводиться в форму, що можна формально представити наступним чином:

$$I = \{I_{ij}\}, \quad I_{ij} \in [0,1], \quad i = 1, \ldots, H, \quad j = 1, \ldots, W$$

Тобто виконується нормалізація (H – висота зображення, W – ширина).

Також зображення переводиться у відтінки сірого, що дозволяє зменшувати обчислювальну складність (оскільки обробляється один канал замість трьох), та змушує модель зосередитись на структурі тексту, збереженні штрихів і так далі. Це переведення виконується методом зваженого усереднення (Weighted Grapyscale) [17], що є стандартом, рекомендованим Міжнародною спілкою електрозв'язку (International Telecommunication Union, ITU) [18]:

$$I_{gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

де R,G,B – червоний, зелений та синій канали зображення відповідно.

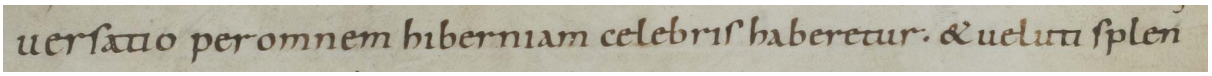
3.3 Джерела даних та обсяг вибірки

Оскільки основною ціллю цієї роботи є реалізація моделі, що буде підвищувати якість роботи OCR системи, очевидно, що дані мають містити транскрипцію зображень у формі тексту. Крім цього, варто пам’ятати, що U-Net моделі в даному випадку працюють в форматі image-to-image (тобто вхідні та вихідні дані це зображення, при чому вихідні зображення бінаризовані).

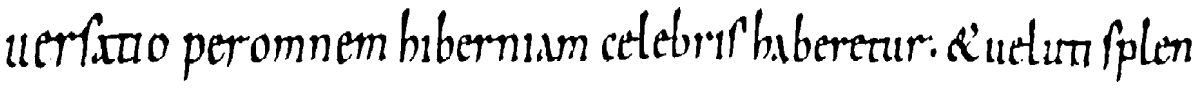
З цього можна зробити висновок, що для того, щоб навчити моделі та провести аналіз впливу їх застосування на якість розпізнавання потрібні набори даних, що містять оригінальні зображення тексту, еталонні бінаризовані та транскрипцію. Таких наборів в публічному доступі досить мало, тому було використано такі два датасети:

1. *Saint Gall* [20]*:* складається з сканів латинського рукопису IX століття. Він містить 1411 зображень рядків, нормалізовані та бінаризовані зображення та транскрипцію.
2. *Old Books* [21]: це набір англомовних друкованих видань кінця XIX століття. Він має складну типографічну верстку та складні деградації (наприклад, пожовтіння сторінок, плями). Він містить 323 сторінки.

Ці дані виглядають таким чином:



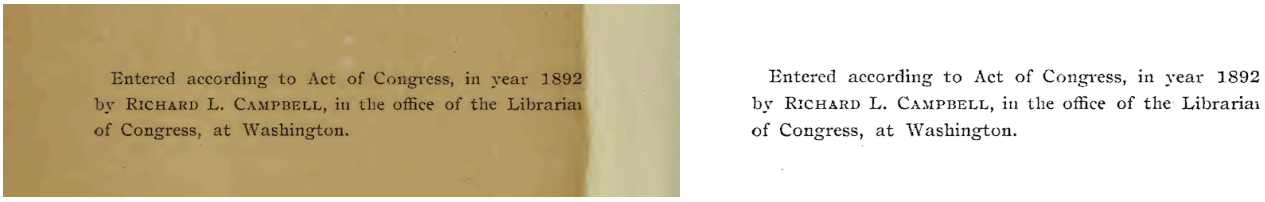
(а) вхідні дані



(б) еталонне зображення

рис. 3.1: Saint Gall

versatio per omnem hiberniam celebris habereturpt et veluti splen – текст на зображенні.



(а) вхідні дані **(б)** еталонне зображення

рис. 3.2: Old Books

Entered according to Act of Congress, in year 1892, by Richard L. Campbell, in the office of the Librarian of Congress, at Washington. – текст на зображенні.

3.4 Загальна структура методу розв'язання

В ході цієї роботи було реалізовано методи передобробки зображень, накладання синтетичних деградацій, реалізація класичних методів, реалізація моделей U-Net (з різними функціями втрат), та проведено аналіз роботи цих моделей за допомогою кількох різних метрик та систем OCR.

3.4.1 Попередня обробка та формування вхідних даних

Було реалізовано програмне забезпечення для конвертації вхідні зображення у відтінки сірого (Grayscale), та тензорної нормалізації (тобто переведення інтенсивності сигналів зображення в шкалу від 0 до 1). Це виконується за допомогою методу з модулю opencv IMREAD_GRAYSCALE та методів роботи з масивами Numpy.

3.4.2 **Синтетичне моделювання деградацій**

Як вже було згадано, в роботі реалізовано модуль для накладання синтетичних деградацій SyntheticDegradation. Він додає розмиття, шуми та імітацію протікання чорнил (Ink Bleed).

- Адитивний гаусів шум описується формулою:

$$x=y+n, \quad n\sim N(0,\sigma^2)$$

Тут y – ідеальне зображення документа без деградацій, x – деградоване зображення, n — значення випадкового гаусового шуму.

- Розмиття гаусівським ядром:

$$x=y*g, \qquad g(u,v)=\frac{1}{2\pi\sigma^2}exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)$$

Де $*$ – оператор згортки зображення з ядром розмиття g (гаусівської функції розмиття), u та v — просторові координати (зсуви по горизонталі та вертикалі) всередині матриці ядра розмиття, σ — стандартне відхилення гаусового розподілу.

- Нелінійне затемнення:

$$x(i,j) = y(i,j)\left(1 - \alpha \cdot \frac{\sqrt{(i-c_i)^2+(i-c_j)^2}}{R}\right)$$

де α – коефіцієнт затемнення, (c_i,c_j) – центр зображення, R – радіус застосування.

Результуючі зображення виглядають наступним чином:

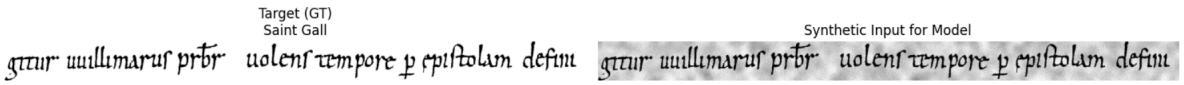


рис. 3.2: зліва – еталонні зображення з Saint Gall та Old Books відповідно, справа – зображення з синтетичними пошкодженнями

3.4.3 **Процедура відновлення та вихідний контроль**

Як було зазначено вище, для відновлення якості зображення будуть використовуватися нейронні мережі архітектури U-Net, що будуть виконувати апроксимацію оберненого оператора R(x). Постобробка зводиться до операцій відсікання значень (clipping), що виконується засобами Numpy.

3.5 Архітектура нейронної мережі U-Net

U-Net – це згорткова нейронна мережа, що була запропонована в 2015 році у роботі Olaf Ronneberger et al. [23]. Вона широко застосовується не лише в сегментації біомедичних зображень, але й в роботі з історичними документами, завдяки її здатності зберігати тонкі деталі штрихів.

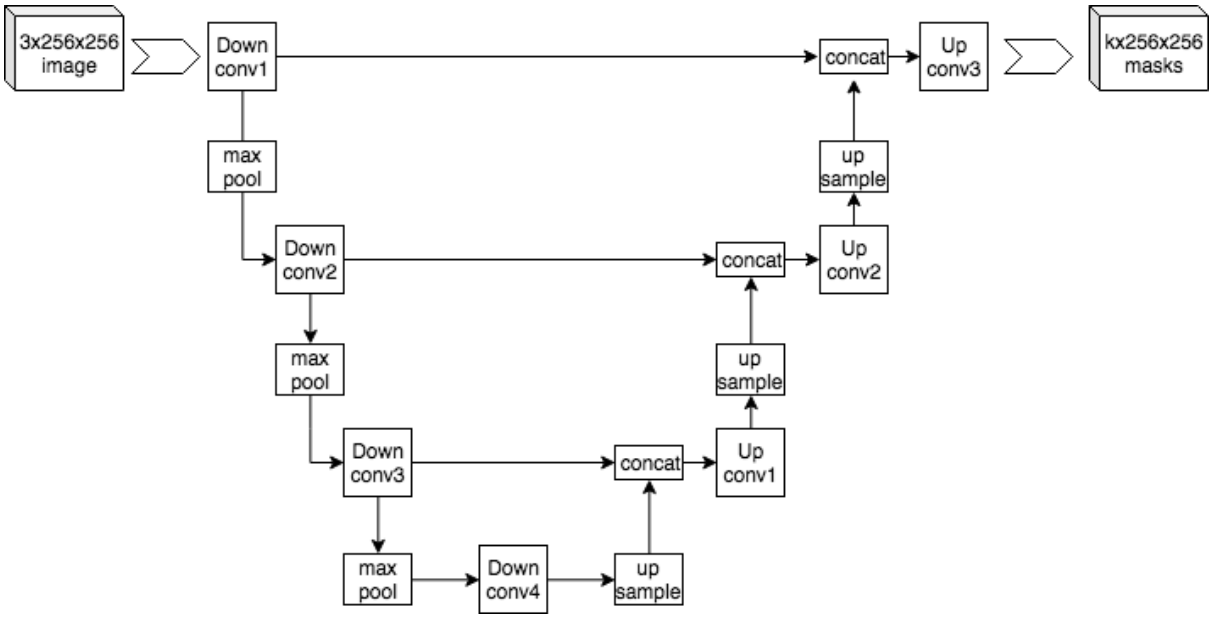


рис. 3.3 – архітектура U-Net для сегментації зображення [24]

Модель U-Net складається з контрактуючого шляху (encoder), розширювального шляху (decoder) та пропускних зв’язків (skip connections) між ними.

Encoder – це серія згорткових блоків, що зменшують розмірність вхідного зображення, що зберігає основні просторові ознаки.

Decoder – це серія транспонованих блоків, що відновлюють розмірність зображення.

Skip connections – прямі з’єднання між цими шарами, що передають інформацію без стискання, що дозволяє зберігати деталі.

Згортковий шар реалізовано наступним чином:

$$z_k^{(l)} = \sigma\left(\sum_m w_{km}^{(l)} * z_m^{(l-1)} + b_k^{(l)}\right)$$

де σ – функція активації (наприклад, ReLU), $w_{km}^{(l)}$ – ваги фільтрів, $b_k^{(l)}$ – зсуви. Для створення програмного коду використовувалися функції модулю PyTorch.

Пропускний зв’язок в свою чергу виглядає як конкатенація:

$$z_{dec}^{(l)} = concat(z_{enc}^{(l)}, z_{up}^{(l)})$$

Де $z_{dec}^{(l)}$ – вихідний тензор на l-му рівні розширювального шляху (decoder), l – індекс поточного рівня (шару) архітектури мережі, $z_{enc}^{(l)}$ – тензор ознак, що приходить напряму з l-го рівня контрактуючого шляху (encoder), $z_{up}^{(l)}$ – тензор ознак з попереднього шару декодера, який пройшов операцію транспонованої згортки.

Модель, створена в цій роботі, складається з 6 функціональних шарів. Енкодер виконує стиснення ознак за допомогою двох згорткових шарів (Conv2d) та операцій Max-Pooling, а декодер відновлює роздільну здатність через шари транспонованої згортки (ConvTranspose2d). Оптимізація параметрів здійснюється шляхом мінімізації однієї з трьох функцій втрат: комбінована (BCE + Dice loss), Perceptual (VGG Perceptual Loss) та MSE Loss.

3.6 Модифікації архітектури: Residual U-Net та Attention U-Net

Традиційна архітектура U-Net має багато варіацій, що можуть бути гірше або краще пристосовані до розв’язання конкретних задач. У цій роботі, крім звичайної (baseline) U-Net, було розглянуто ще 2 архітектури моделей, та вплив їх застосування на якість відновленого зображення і розпізнавання символів.

3.6.1 **Residual U-Net**

Ця модель використовує залишкових блоків (Residual Units) замість стандартних згорткових шарів. Такий блок виглядає наступним чином:

$$z^{(l)} = \sigma(F(z^{(l-1)}, w^{(l)}) + z^{(l-1)})$$

де F – послідовність згорткових операцій, $w^{(l)}$ – набір вагів та параметрів згорткових фільтрів, σ(…) – нелінійна функція активації.

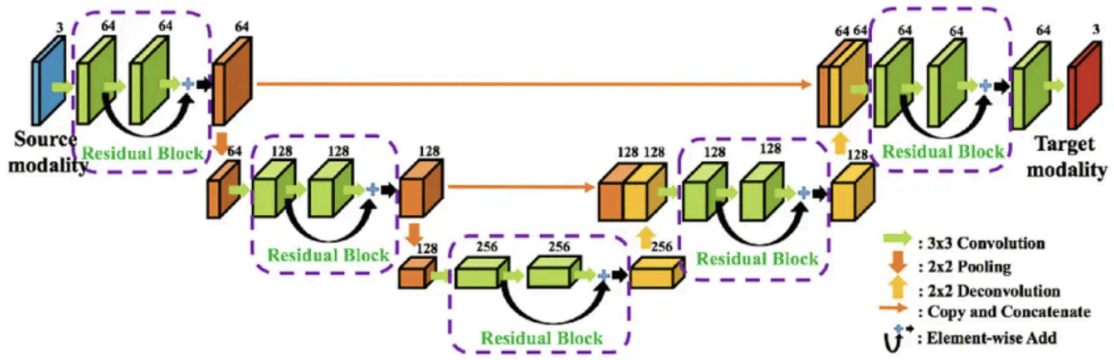


рис. 3.4: архітектура Residual U-Net [25]

Через додавання вхідного сигналу $z^{(l-1)}$ до результату згортки мережа вивчає залишковий сигнал. Це допомагає вирішити проблему затухання градієнтів при збільшенні глибини мережі та сприяє кращому збереженню фону та яскравості документу.

3.6.2 **Attention U-Net**

Ця архітектура додає до стандартної U-Net механізми блоків уваги (Attention Gates) на пропускних зв'язках. Замість простої конкатенації всіх ознак з енкодеру та декодеру, блок уваги фільтрує інформацію, виділяючи найбільш важливі області:

$$\hat{z}_{enc} = z_{enc} \cdot \alpha$$

де α — коефіцієнт уваги, що обчислюється на основі взаємодії поточного шару декодера та відповідного шару енкодера.

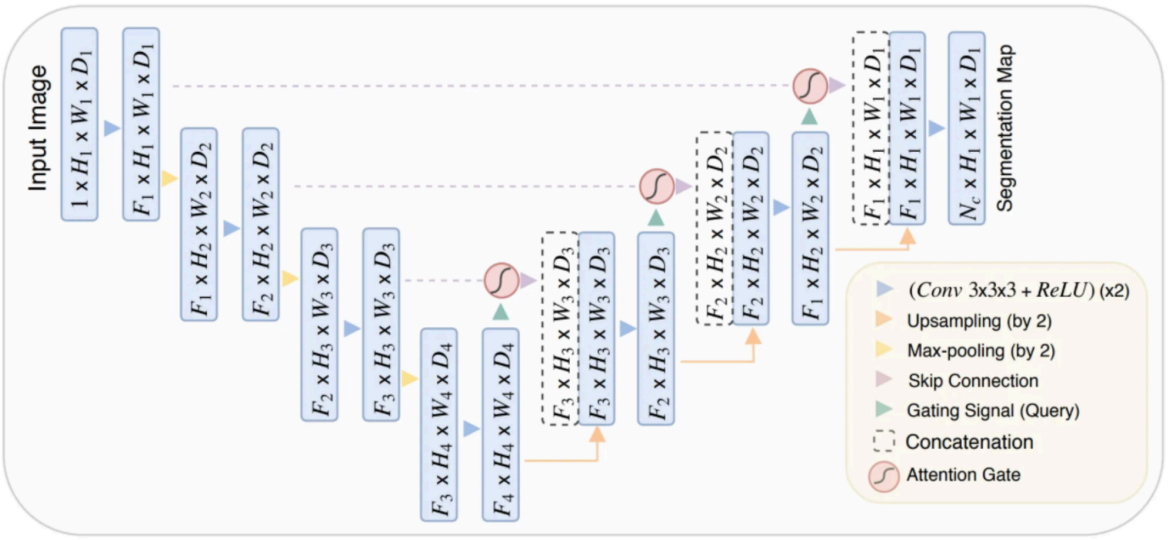


рис. 3.5: архітектура Attention U-Net [26]

Це дозволяє моделі ігнорувати шуми, плями та текстуру паперу, зосереджуючись на контурах символів та збереженні штрихів тексту.

3.7 Математична модель навчання та оптимізації

Навчання нейронної мережі f_θ полягає у мінімізації функції втрат на наборі парних зображень (x_i, y_i). Цей процес формалізується як задача мінімізації цільової функції:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (L(f_\theta(x_i), y_i)$$

θ – вектор параметрів нейронної мережі, що включає в себе всі її ваги (w) та зміщення (b, biases), N – кількість пар зображень у датасеті, $f_\theta(x_i)$ – передбачення нейронної мережі для і-го вхідного зразка, L(…) — функція втрат (loss function).

Для пошуку оптимальних параметрів θ застосовано алгоритм Adam (Adaptive Moment Estimation). Цей метод поєднує переваги адаптивних швидкостей навчання (AdaGrad) та інерційного підходу (Momentum). Оновлення ваг виконується з урахуванням оцінок першого (m_t) та другого (v_t) моментів градієнтів:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}}+\epsilon}\hat{m}_t$$

де η – швидкість навчання (learning rate), яка в цій роботі встановлена на рівні 0.001, $\widehat{m_t}$ – незміщена оцінка першого моменту градієнта, $\widehat{v_t}$ – незміщена оцінка другого моменту градієнта.

3.8 Функції втрат та критерії оптимізації

Для дослідження вибрано три різні функції втрат, щоб порівняти їх вплив на результат та зробити висновки про те, які функції сприяють найкращому результату при розпізнаванні символів.

3.8.1 **Середньоквадратична помилка (MSE)**

Середньоквадратична помилка (Mean Square Error) базується на мінімізації евклідової відстані між інтенсивністю пікселів передбаченого зображення $\widehat{y}$ та еталонного y:

$$L_{MSE} = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}(\hat{y}_{ij} - y_{ij})^2$$

Де H і W - це висота та ширина зображення відповідно, $\hat{y}_{ij}$ – інтенсивність передбаченого пікселю, y_{ij}– інтенсивність цього пікселю на ідеальному зображенні.

Ця функція забезпечує високе значення метрики PSNR, проте часто призводить до надмірного розмиття (blurring) дрібних деталей та штрихів символів, оскільки вона однаково штрафує за помилки в інтенсивності незалежно від структури об'єкта.

3.8.2 **Комбінована функція BCE + Dice Loss**

Як можна зрозуміти з назви, ця функція – це поєднання бінарної крос-ентропії та коефіцієнта подібності Дайса:

$$L_{Total} = \alpha L_{BCE} + (1 - \alpha)L_{Dice}$$

Binary Cross-Entropy (BCE) вимірює попіксельну ймовірність приналежності до класу (текст/фон):

$$L_{BCE} = -\frac{1}{N}\sum[y\,log(\hat{y}) + (1 - y)log(1 - \hat{y})$$

Dice Loss фокусується на перекритті областей, що допомагає краще відновлювати тонкі елементи літер, які займають малу частку загальної площі зображення:

$$L_{Dice} = 1 - \frac{2|\hat{y}\cap y|}{|\hat{y}|+|y|}$$

Де $\left|\widehat{y} \cap y\right|$ – кількість пікселів, у яких передбачення моделі повністю збіглося з оригіналом (площа їхнього спільного перекриття), а $\left|\widehat{y}\right| + |y|$ – сума площ (загальна кількість пікселів) передбаченої маски та маски-еталона.

3.8.3 **Перцептуальна функція втрат (Perceptual Loss)**

Перцептуальна функція втрат – оцінює різницю між зображеннями не попіксельно (як MSE), а на рівні високорангових ознак, видимих людському оку, наприклад, загальний зміст, текстуру, контури та геометрію символів. В програмній реалізації використовується попередньо навчена мережа VGG16. Передбачення моделі та еталонне зображення пропускаються крізь шари VGG. На виході отримуються матриці ознак, і вже між цими картами рахується середня квадратична помилка (MSE).

У цій роботі перцептуальний лос використовується як регуляризатор із ваговим коефіцієнтом 0.1 на додачу до попіксельного MSE:
Загальний лос = Попіксельний MSE+0.1×Перцептуальний лос
Це навчає модель одночасно очищати фон (завдяки MSE) та зберігати чіткі, нерозмиті краї шрифтів (завдяки VGG).

3.9 Метрики оцінювання якості

Для оцінювання якості зображень було використано три різні класичні метрики.

Пікове відношення сигналу до шуму (PSNR):

$$PSNR = 10log_{10}\left(\frac{MAX^2}{MSE}\right)$$

де MAX – максимальне значення пікселя (зазвичай 1), а MSE – Mean Square Error.

Індекс структурної подібності (SSIM) – на відміну від PSNR, що обчислює абсолютну похибку, ця метрика припускає, що є зв’язки в структурі пікселів, та оцінює зміну цієї структури:

$$SSIM(x,\ y)\ =\ \frac{(2\mu_x\mu_y+C_1)(2\sigma_{xy}+C_2)}{(\mu_x^{\ 2}+\mu_y^{\ 2}+C_1)(\sigma_x^{\ 2}+\sigma_y^{\ 2}+C_2)},$$

Де μ_x, μ_y – середні значення інтенсивності пікселів; $\sigma_x^{\ 2}$, $\sigma_y^{\ 2}$ – дисперсії інтенсивностей; σ_{xy} – коваріація між x та y; C_1, C_2 – константи для стабілізації розрахунку при близьких до нуля знаменниках.

Також використовується метрика на основі відстані між бінарними масками (F-measure):

$$Precision\ =\ \frac{TP}{TP+FP},\ Recall\ =\frac{TP}{TP+FN}$$

$$F\ =\ \frac{2\cdot Precision\cdot Recall}{Precision+Recall}$$

Де TP (True Positives – істинно позитивні) – кількість пікселів тексту, які модель визначила правильно; FP (False Positives – хибно позитивні) – кількість фонових пікселів, які модель помилково визначила як текст; FN (False Negatives – хибно негативні) – кількість пікселів тексту, які модель пропустила.

Для оцінки якості відновлення тексту застосовувалися метрики WER та CER. CER – рівень помилок на символах, WER (World Error Rate) – рівень помилок на словах. Вони використовують відстань Левенштейна між згенерованим текстом та еталоном. Цей алгоритм динамічного програмування дозволяє визначити мінімальну кількість операцій заміни (S), видалення (D) та вставки (I) елементів для перетворення одного рядка в інший.

$$CER = \frac{S_c + D_c + I_c}{N_c}$$

Де S_c – кількість замінених символів, D_c – кількість видалених символів, I_c – кількість вставлених символів, N_c – загальна кількість символів в еталонному тексті.

$$WER = \frac{S_w + D_w + I_w}{N_w}$$

Де S_w – кількість замінених слів, D_w – кількість видалених слів, I_w – кількість вставлених слів, N_w – загальна кількість слів в еталонному тексті.

Використання декількох метрик дозволяє оцінити не тільки чисельну, але й візуальну якість результатів.

3.10 Пояснення вибору методу

U-Net було обрано через її здатність зберігати просторову інформацію завдяки пропускним зв’язкам. Тренування моделей U-Net вимагає набагато менше ресурсів, ніж, наприклад, GAN моделі, і є більш стабільним ніж деяких інших архітектур, як дифузійних.

Висновки до розділу

У цьому розділі було розглянуто постановку задачі, типи вхідних даних та методи підвищення якості зображень документів. Задачу було формалізовано як обернена задача відновлення зображення, що буде розв’язана за допомогою моделі U-Net. Було описано кілька модифікацій архітектури U-Net та функцій втрат. Також розглянуто математичне формулювання метрик візуальної оцінки отриманих результатів, та метрик оцінки якості розпізнавання.

4 ПОРІВНЯЛЬНИЙ АНАЛІЗ ВИБРАНИХ МЕТОДІВ

4.1 **Технологічний стек та середовище реалізації**

Експериментальна частина роботи була реалізована мовою програмування Python 3.12. Для виконання обчислювально складних операцій використовувалася платформа Kaggle. Обчислення проводилися з використанням графічних прискорювачів GPU NVIDIA Tesla T4 x 2. Це дозволило прискорити процес навчання завдяки паралелізації обчислень, що важливо для обробки великих кількостей зображень високої роздільної здатності.

Програмний стек та бібліотеки:

1. **PyTorch**: бібліотека для глибокого навчання, що використовувався для проектування архітектури U-Net.
2. **OpenCV**: бібліотека для обробки зображень, конвертації в відтінки сірого, обрізання зображень та реалізації класичних алгоритмів обробки.
3. **NumPy**: бібліотека для роботи з багатовимірними масивами, що була потрібна для виконання математичних операцій над матрицями інтенсивностей пікселів.
4. **Scikit-image**: застосовувалася для розрахунку метрик якості відновлення, таких як PSNR та SSIM.
5. **Tesseract OCR**: система оптичного розпізнавання символів.

Всі ці інструменти дозволили ефективно підготувати дані, реалізувати архітектуру моделей та їх навчання і провести порівняння їх роботи.

4.2 **Програмна реалізація алгоритмів синтетичної деградації**

Було реалізовано клас SyntheticDegradation, який виконує моделювання дефектів на основі еталонних зображень.

Він має три основні модулі:

1. **Моделювання адитивного шуму**: реалізовано через генерацію випадкових значень за Гаусовим розподілом. Цей метод імітує електронні шуми матриці сканера. Параметр інтенсивності (severity) дозволяє змінювати рівень зашумлення, в цій роботі це – згенероване випадкове значення в межах [0.05, 0.12].
2. **Розмиття (Blur)**: застосовується для імітації розфокусування або природного стирання чорнила на папері. В реалізації застосовано Гауссову згортку з змінним середньоквадратичним відхиленням (σ). Це дозволяє розмивати чіткі контури символів, які модель має відновити. Значення σ рандомно генерується в межах [1.0, 2.5].
3. **Протікання чорнил (Ink Bleed)**: Алгоритм створює шумову маску, нормалізує її до діапазону [0,1], а потім віднімає її від інтенсивностей вихідного зображення. Це створює помітні плями та сторонні візуальні об'єкти.

Кожен тип деградації (крім базового шуму) накладається з імовірністю 50%. Такий підхід дозволяє моделі навчатися на різних видах деградацій, що знижує ризик перенавчання та наближає навчальні дані до стану справжніх історичних документів, які рідко мають лише один вид пошкодження.

4.3 Параметри навчання та конфігурація моделей

Використання різних оптимізаторів, кількості епох навчання, та конфігурацій має значний вплив на якість результатів, тому важливо правильно добрати ці параметри.

4.3.1 **Параметри оптимізації та навчання**

- Оптимізатор**:** Використано алгоритм Adam із початковою швидкістю навчання (learning rate) 10^{-4}. Вибір зумовлений його здатністю адаптивно налаштовувати крок навчання, що прискорює збіжність при роботі з різноманітними цільовими функціями.
- Тривалість навчання**:** Кожна модель проходила 10 епох навчання на повній вибірці даних. Тестування показало, що значення функцій втрат стабілізуються на цьому етапі, тому продовження навчання недоцільне.

4.3.2 **Реалізація функцій втрат**

Функція для навчання моделей train_model була реалізована з можливістю вибору однієї з трьох різних функцій втрат, які порівнюються у цій роботі:

1. MSE: Відстань між інтенсивністю еталону та передбачення.
2. BCE + Dice: Комбінація бінарної крос-ентропії (BCELoss) та коефіцієнта подібності Дайса (dice_loss).
3. Perceptual: з використанням моделі VGG.

4.3.3 **Результати процесу навчання**

За результатами тесту, Residual U-Net продемонструвала найшвидшу збіжність – вона завжди мала найменше значення функції втрат вже на початку навчання, що підтверджує корисність її залишкових зв'язків. Attention U-Net збігається трохи швидше, ніж базова архітектура завдяки зосередженню на областях тексту, а не фону.

Для всіх моделей найнижчі значення функції втрат були отриманні при тренуванні з функцією MSE, а найвищі – з перцептуальною. Використання комбінованої функції BCE+Dice давало стабільно добрі результати.

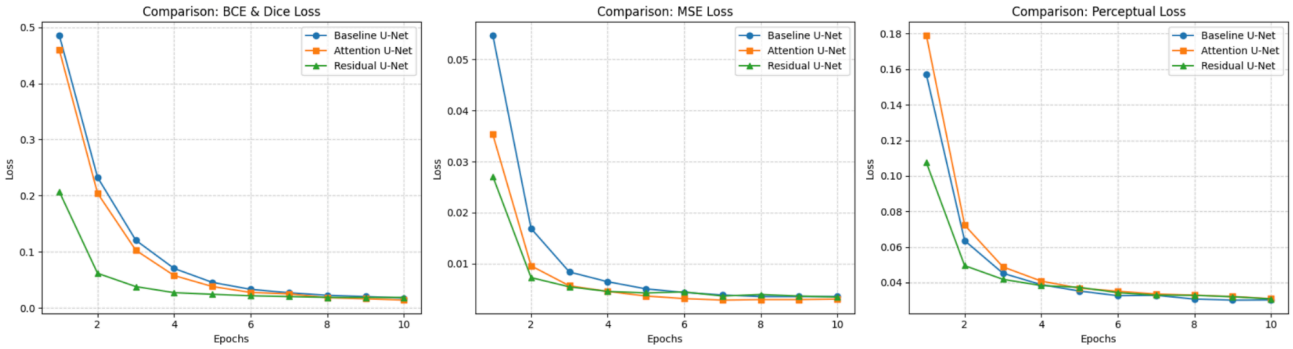


рис. 4.1: історія змін значення функції втрат протягом навчання

4.4 Експериментальне порівняння функцій втрат

За допомогою описаних раніше чисельних метрик проведемо порівняння впливу обраних функцій втрат на якість відновлення документів.

4.4.1 **Кількісний аналіз результатів**

Згідно з отриманими експериментальними даними (див. табл. 4.1),

спостерігається чітка залежність між типом функції втрат та фінальними метриками якості.

Таблиця 4.1. Порівняння ефективності моделей за різними функціями втрат

Метод	**F-Measure**	**PSNR (дБ)**	**SSIM**
Baseline BCE + Dice	0.999172	57.904992	0.996655
Baseline MSE	0.999208	37.803271	0.990209
Baseline Perceptual	0.999366	40.360337	0.996481
Attention BCE + Dice	0.998962	53.299922	0.996057
Attention MSE	0.999346	30.517017	0.979267
Attention Perceptual	0.999208	37.803271	0.990209
Otsu	0.922357	11.471157	0.659674
Sauvola	0.945433	10.885204	0.605146
Residual BCE + Dice	**0.999417**	**64.882229**	**0.997197**
Residual MSE	0.999007	46.123130	0.993869
Residual Perceptual	0.999301	46.979125	0.996960

Очевидно, що найкращі результати показала Residual модель з комбінованою функцією втрат BCE+Dice. Загалом, використання функції BCE+Dice позитивно вплинуло на результат, в порівнянні з іншими функціями втрат, моделі, які навчалися з нею, мають значно більші показники PSNR та SSIM.

Також ця порівняльна таблиця демонструє перевагу нейромережевих методів над традиційними: якщо різниця в значеннях F-Measure не є настільки помітною, то при порівнянні за метрикою PSNR виявляється, що традиційні методи програють в 4 рази, а за SSIM – приблизно в 1.5, отже, застосування моделей U-Net є набагато більш ефективним рішенням.

4.4.2 **Аналіз впливу функцій втрат**

BCE + Dice Loss: Ця функція продемонструвала найкращі результати у поєднанні з усіма архітектурами, особливо за показником PSNR. Це свідчить про те, що для даного типу деградацій (шум, розмиття, проступання чорнила) оцінка сегментації тексту є значно ефективнішою за пряму попіксельну регресію.

MSE Loss: Показала помірні результати. Можна помітити, що вона досить погано впливала на значення PSNR, але на інших метриках вона мала приблизно такі самі результати, як і інші функції.

Perceptual Loss: Її результати є дещо кращими за результати MSE, але все ще поступаються BCE + Dice.

4.4.3 **Вплив архітектури**

Модель Residual BCE + Dice продемонструвала найкращий баланс між збереженням структури та точністю виділення текстових областей, досягши найвищого значення F-Measure=0.999417 та структурної схожості SSIM=0.997197, а також низьке значення шуму, при значенні PSNR=64.882229. Всі інші моделі також дали гарні результати, але Residual U-Net є беззаперечним лідером.

4.5 Оцінка ефективності відновлення через OCR-тестування

Для визначення практичної цінності розроблених моделей проведено тестування точності розпізнавання тексту за допомогою системи Tesseract. Оцінка здійснювалася за метриками CER (відсоток помилок у символах) та WER (відсоток помилок у словах).

4.5.1 **Аналіз результатів розпізнавання**

Порівняння результатів на деградованих та відновлених зображеннях (див. табл. 4.2) виявило суттєве покращення читабельності документів після обробки нейромережами.

Таблиця 4.2. Метрики точності розпізнавання OCR (CER та WER)

Метод	**CER**	**WER**
З синтетичними пошкодженнями без передобробки	0.505124	0.610440
Otsu	0.520242	0.676946
Sauvola	0.677614	0.818853
Baseline BCE+DICE	0.222963	0.439798
Baseline MSE	0.236892	0.478972
Baseline Perceptual	0.241488	0.447686
Attention BCE+DICE	0.214971	0.438513
Attention MSE	0.295319	0.501732

Attention Perceptual	0.270001	0.473117
Residual BCE+DICE	**0.195755**	**0.438375**
Residual MSE	0.276891	0.554464
Residual Perceptual	0.240440	0.441821

4.5.2 **Висновки тестування**

Неефективність класичних методів: Бінаризація за методом Оцу не покращила результати порівняно з деградованим зображенням (Tesseract CER 0.520242), що підтверджує складність обраних деградацій для стандартних алгоритмів. Так само, метод бінаризації Саувола тільки погіршив якість розпізнавання тексту.

Перевага Residual BCE+DICE моделі: Саме ця модель продемонструвала низькі відсотки помилок як на словах, так і на символах, що підтверджує ефективність залишкових блоків для зберігання дрібних деталей. Окрім неї, дуже гарні результати показала модель Attention U-Net з функцією BCE+Dice, очевидно, завдяки здатності фокусуватися на областях тексту.

Необхідність методів попередньої обробки: Застосування згорткових моделей знизило відсоток помилок більше, ніж в 2 рази, що доводить їх ефективність.

4.5.3 **Візуалізація передбачень моделей**

Для порівняння впливу роботи моделей на вигляд документів варто представити передбачення моделей у вигляді зображень. Це дозволяє оцінити

читабельність тексту для людського ока та побачити, як модель працює з фоном зображення, розмиттям, наскільки багато пошкоджень залишається після очищення.

рис. 4.2: порівняння передбачень моделей

На зображенні можна помітити, що моделі Attention гірше очищують фон, що пов’язано з тим, що вони зосереджуються на роботі з текстом. Також можна

зауважити, що застосування функції MSE стабільно дає незадовільні результати, що особливо помітно у випадку Residual моделі – текст дуже помітно розмитий.

Ще одна цікава закономірність – застосування перцептуальної функції втрат, схоже, досить часто призводить до часткового усунення символів (особливо це помітно у Attention моделі). Застосування функції BCE+Dice стабільно дає гарні результати для всіх моделей.

Висновки до розділу

На основі проведених експериментів встановлено, що найбільш ефективною конфігурацією для відновлення історичних документів з точки зору візуальної якості відновленого зображення є Residual U-Net у поєднанні з функцією втрат BCE + Dice. Дана комбінація забезпечує найвищі показники структурної подібності (SSIM 0.997197), зберігаючи гарне відношення сигналу до шуму.

З точки зору якості розпізнавання, використання реставрації U-Net моделями дозволило знизити похибку розпізнавання символів (CER) для Tesseract майже у **3 рази** (з 0.5 до 0.19). Найкращі за всіма метриками результати продемонструвала модель Residual з комбінованою функцією втрат, що робить її пріоритетним вибором для передобробки в системах автоматизованого оцифрування архівів.

5 ВИСНОВКИ

Було проведено аналіз існуючих методів попередньої обробки зображень, проаналізовано теоретичні підґрунтя класичних методів бінаризації Оці та Саувола, а також нейромережевих методів U-Net, Residual U-Net та Attention U-Net. Описано функції втрат: Середньоквадратична похибка, комбінована функція Бінарна кросс-ентропія + Коефіцієнт Дайса, та Перцептуальна функція з використнням попередньо навченої моделі VGG16. Описано обрані набори даних з їх особливостями, а також метрики оцінювання візуальної якості зображень, згенерованих мережею (PSNR, SSIM, F-measure), та метрики оцінки якості розпізнавання (CER, WER). Також описано модуль для накладання синтетичних деградацій.

Реалізовано програмний комплекс для класичних, нейромережевих методів та їх тренування, обрахунку метрик та накладання пошкоджень на зображення. Проведено серію експериментів, в ході яких виявлено, що найкращою моделлю за показниками PSNR (64.882), SSIM (0.997), F-measure (0.999), CER (0.19) та WER (0.43) є модель Residual U-Net з використанням функції втрат BCE+Dice. Перцептуальна функція втрат показала стабільно добрі результати, тоді як використання Середньоквадратичної похибки завжди призводило до незадовільних результатів.

Таким чином, було експериментально доведено важливість передобробки зображень пошкоджених історичних документів для зниження кількості помилок при розпізнаванні (відсоток помилок зменшився у 2.6 рази), перевагу нейромережевих методів у порівнянні до традиційних та перевагу спеціальних функцій втрат, що враховують структуру тексту та деталі, в порівнянні до стандартних функцій втрат.

Розроблений програмний комплекс можна використовувати в дослідницьких інститутах, бібліотеках та архівах з метою покращення якості оцифрованих документів, а також в системах OCR для їх спрощення.

В подальшому дослідженні можна додати функції втрат, що враховують помилки OCR, таким чином, модель навчатиметься видаляти пошкодження спеціально для підвищення якості розпізнавання, а не тільки для збереження деталей. Також можна застосувати модель Residual Attention U-Net, що поєднує переваги моделей Residual U-Net та Attention U-Net, хоча й є більш складною в реалізації.

Список використаних джерел

[1] N. Otsu, 1979. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62-66.

[2] J. Sauvola, M. Pietikäinen, 2000. Adaptive document image binarization. Pattern Recognition, vol. 33, pp. 225-236, ISSN 0031-3203.

[3] Tensmeyer, C. and Martinez, T., 2017. Document Image Binarization with Fully Convolutional Neural Networks. Proceedings of the International Conference on Document Analysis and Recognition (ICDAR), pp. 99-104.

[4] B. Gatos, K. Ntirogiannis, I. Pratikakis, 2009. *Icdar 2009 document image binarization contest (dibco 2009),* vol. 9. IEEE, pp. 1375– 1382.

[5] Tamrin, M.O., Ech-Cherif, M.E. and Cheriet, M., 2021. A Two-Stage Unsupervised Deep Learning Framework for Degradation Removal in Ancient Documents. *IEEE Access*, 9, pp. 12345-12356.

[6] Bhunia A., Sain A., Roy, P., 2018. Improving Document Binarization via Adversarial Noise-Texture Augmentation. 10.48550/arXiv.1810.11120.

[7] Starynska, A., 2023. *Text restoration in palimpsested historical manuscripts using deep learning methods*. Ph.D. dissertation. Rochester Institute of Technology.

[8] He S., Schomaker L., 2021. CT-Net: Cascade T-shape deep fusion networks for document binarization, *Pattern Recognition*, Volume 118.

[9] Yoshizu, Y., Kaneko, H., Ishibashi, 2025. Restoration of ancient Japanese manuscripts via the diffusion denoising restoration model and color space-based masking, *npj Herit. Sci.* 13, 634.

[10] Yousef, M. and Bishop, T.E., 2020. OrigamiNet: Weakly-Supervised Hand-Written Text Recognition. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 3210-3219.

[11] Afkari-Fahandari A., Shabaninia E., Asadi-Zeydabadi F., and Nezamabadi-Pour H., 2025. A Comprehensive Survey of Transformers in Text Recognition:

Techniques, Challenges, and Future Directions, *ACM Computing Surveys*, 58(5), pp. 1–42.

[12] Li Y., Chen D., Tang T., and Shen X., 2024. HTR-VT: Handwritten Text Recognition with Vision Transformer, arXiv preprint arXiv:2409.08573.

[13] Foret P., Kleiner A., Mobahi H., Neyshabur B., 2020. Sharpness-aware minimization for efficiently improving generalization, International Conference on Learning Representations (ICLR).

[14] Cascianelli S., Pippi V., Maarand M., Cornia M., Baraldi L., Kermorvant C., and Cucchiara R., 2022. The LAM Dataset: A Novel Benchmark for Line-Level Handwritten Text Recognition, Proceedings of the 26th International Conference on Pattern Recognition (ICPR), pp. 1506–1513.

[15] Nguyen T. T. H., Jatowt A., Coustaty M., and Doucet A., 2022. Survey of Post-OCR Processing Approaches, *ACM Computing Surveys*, Vol. 54, No. 6, Article 124, pp. 1–37.

[16] Uwe Springmann, Christian Reul, Stefanie Dipper, and Johannes Baiter. 2018. Ground Truth for training OCR engines on historical documents in german fraktur and early modern latin. J. Lang. Technol. Comput. Ling. 33, 1 (2018), 97– 114.

[17] Asselborn, T. et al., 2025. Treating OCR Output as a Language (TOOL) – Improving OCR Output with Seq2Seq Translation. *Computational Linguistics and Heritage*, 11(1), pp. 74-89.

[18] Assael, Y., Sommerschield, T. and Prag, J., 2019. Restoring ancient text using deep learning: a case study on Greek epigraphy. Nature, 567, pp. 356–359.

[19] Gonzalez, R., & Woods, R., 2018. Digital Image Processing (4th ed.). Pearson.

[20] ITU-R Recommendation Rec. 601. "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios". International Telecommunication Union.

[21] Manuscript images of the Codex Sangallensis 562, 2006. St. Gallen, Stiftsbibliothek.

[22] Barcha, P. (2017). *Old Books Dataset* [Data set]. GitHub. https://github.com/PedroBarcha/old-books-dataset

[23] Ronneberger O., Fischer P., Brox T., 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. arxiv:1505.04597

[24] File:Example architecture of U-Net for producing k 256-by-256 image masks for a 256-by-256 RGB image.png. (2025, October 27). *Wikimedia Commons*. Retrieved May 19, 2026, from https://commons.wikimedia.org/w/index.php?title=File:Example_architecture_of_U-Net_for_producing_k_256-by-256_image_masks_for_a_256-by-256_RGB_image.png&oldid=1105642406.

[25] Xiang, L., Li, Y., Lin, W., Wang, Q., Shen, D., 2018. Unpaired Deep Cross-Modality Synthesis with Fast Training. In: Stoyanov, D., *et al.* Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support. DLMIA ML-CDS 2018. Lecture Notes in Computer Science, vol 11045. Springer, Cham. https://doi.org/10.1007/978-3-030-00889-5_18

[26] Oktay O., Schlemper J., Le Folgoc L., Lee M., Heinrich M., Misawa K., Mori K., McDonagh S., Hammerla N., Kainz B., Glocker B., Rueckert D., 2018. Attention U-Net: Learning Where to Look for the Pancreas. arxiv:1804.03999

Додатки

```
import os
import glob
import cv2
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
import pytesseract
from jiwer import wer

# Set device
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {DEVICE}")
import numpy as np
import cv2
import random
from scipy.ndimage import gaussian_filter

class SyntheticDegradation:
    @staticmethod
    def add_gaussian_noise(image, severity=0.05):
        noise = np.random.normal(0, severity, image.shape).astype(np.float32)
        return np.clip(image + noise, 0, 1)

    @staticmethod
    def add_blur(image, sigma=1.5):
        return gaussian_filter(image, sigma=sigma)

    @staticmethod
    def add_ink_bleed(image, severity=0.3):
        """Simulates ink bleeding from the back of the page."""
        bleed = gaussian_filter(np.random.random(image.shape), sigma=10)
        bleed = (bleed - bleed.min()) / (bleed.max() - bleed.min())
        return np.clip(image - (bleed * severity), 0, 1)

    @staticmethod
    def apply_all(image):
        img = image.copy()

        if random.random() > 0.5:
            img = SyntheticDegradation.add_blur(img, sigma=random.uniform(1.0, 2.5))
```

```
        if random.random() > 0.5:
            img = SyntheticDegradation.add_ink_bleed(img, severity=random.uniform(0.4, 0.7))

            img = SyntheticDegradation.add_gaussian_noise(img, severity=random.uniform(0.05,
0.12))
        return img
import os
import glob
import cv2
import torch
import numpy as np
from torch.utils.data import Dataset, DataLoader

class CombinedDocumentDataset(Dataset):
      def __init__(self, image_paths, mask_paths, patch_size=256, samples_per_image=10,
is_synthetic=None):
        """
        image_paths: Paths to input images (noisy)
        mask_paths: Paths to Ground Truth images (clean/binarized)
          is_synthetic: Boolean list indicating if the image at that index needs synthetic
degradation
        """
        self.image_paths = image_paths
        self.mask_paths = mask_paths
        self.patch_size = patch_size
        self.samples_per_image = samples_per_image
            self.is_synthetic = is_synthetic if is_synthetic is not None else [False] *
len(image_paths)

    def __len__(self):
        return len(self.image_paths) * self.samples_per_image

    def __getitem__(self, idx):
        img_idx = idx // self.samples_per_image

        # Load the Ground Truth (Mask)
        mask = cv2.imread(self.mask_paths[img_idx], cv2.IMREAD_GRAYSCALE)

        # Load or generate the Input Image
        if self.is_synthetic[img_idx]:
            clean_base = mask.astype(np.float32) / 255.0
            img = (SyntheticDegradation.apply_all(clean_base) * 255).astype(np.uint8)
        else:
            img = cv2.imread(self.image_paths[img_idx], cv2.IMREAD_GRAYSCALE)

        # Handle size mismatch or small images
        h, w = img.shape
```

```
        if h <= self.patch_size or w <= self.patch_size:
            img = cv2.resize(img, (max(w, self.patch_size+1), max(h, self.patch_size+1)))
            mask = cv2.resize(mask, (max(w, self.patch_size+1), max(h, self.patch_size+1)))
            h, w = img.shape

        # Random Patch Extraction
        y = np.random.randint(0, h - self.patch_size)
        x = np.random.randint(0, w - self.patch_size)

        img_patch = img[y:y+self.patch_size, x:x+self.patch_size].astype(np.float32) / 255.0
                    mask_patch    =    (mask[y:y+self.patch_size,    x:x+self.patch_size]    >
127).astype(np.float32)

        return (torch.from_numpy(img_patch).unsqueeze(0),
                torch.from_numpy(mask_patch).unsqueeze(0))


def      prepare_all_data(original_base_path,      saint_gall_base_path,      batch_size=16,
patch_size=256):
    """
    Scans directories, pairs images with masks, and returns DataLoaders + Path Lists.
    """
    # --- 1. Load Original Dataset ---
    orig_images = sorted(glob.glob(os.path.join(original_base_path, "300dpi/tiff/*.tif*")))
                    orig_masks       =       sorted(glob.glob(os.path.join(original_base_path,
"binarized_300dpi/Otsu/*.tif*")))
    orig_texts = sorted(glob.glob(os.path.join(original_base_path, "groundtruth/*.txt")))
    orig_is_synth = [random.choice([True, False]) for _ in range(len(orig_images))]

    # --- 2. Load Saint Gall DB ---
    sg_inner_path = "saintgalldb-v1.0/data/line_images_normalized/*.png"
    sg_gt_paths = sorted(glob.glob(os.path.join(saint_gall_base_path, sg_inner_path)))
    if not sg_gt_paths:
                        sg_gt_paths    =    sorted(glob.glob(os.path.join(saint_gall_base_path,
"data/line_images_normalized/*.png")))

    sg_images = sg_gt_paths  # Use GT to generate noisy inputs synthetically
    sg_is_synth = [random.choice([True, False]) for _ in range(len(sg_gt_paths))]

    sg_texts = [""] * len(sg_gt_paths)

    # --- 3. Combine & Shuffle ---
    all_img_paths = orig_images + sg_images
    all_mask_paths = orig_masks + sg_gt_paths
    all_synth_flags = orig_is_synth + sg_is_synth
    all_txt_paths = orig_texts + sg_texts

    # Use a fixed seed for reproducibility
```

```
    indices = np.arange(len(all_img_paths))
    np.random.seed(42)
    np.random.shuffle(indices)

    all_img_paths = [all_img_paths[i] for i in indices]
    all_mask_paths = [all_mask_paths[i] for i in indices]
    all_synth_flags = [all_synth_flags[i] for i in indices]
    all_txt_paths = [all_txt_paths[i] for i in indices]

    # --- 4. Split (80/20) ---
    split = int(len(all_img_paths) * 0.8)

    train_imgs, test_imgs = all_img_paths[:split], all_img_paths[split:]
    train_masks, test_masks = all_mask_paths[:split], all_mask_paths[split:]
    train_synth, test_synth = all_synth_flags[:split], all_synth_flags[split:]
    train_texts, test_texts = all_txt_paths[:split], all_txt_paths[split:]

    # --- 5. Create Objects ---
        train_ds = CombinedDocumentDataset(train_imgs, train_masks, patch_size=patch_size,
is_synthetic=train_synth)
        test_ds = CombinedDocumentDataset(test_imgs, test_masks, patch_size=patch_size,
is_synthetic=test_synth)

    train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, num_workers=2)
    test_loader = DataLoader(test_ds, batch_size=batch_size, shuffle=False, num_workers=2)

          print(f"Total: {len(all_img_paths)} | Train: {len(train_imgs)} | Test:
{len(test_imgs)}")

    return train_loader, test_loader, test_imgs, test_masks, test_texts

train_loader, test_loader, test_imgs, test_masks, test_texts = prepare_all_data(
    'old-books-dataset-master',
    'saintgalldb-v1.0'
)
class ConvBlock(nn.Module):
    def __init__(self, in_c, out_c):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_c, out_c, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_c),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_c, out_c, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_c),
            nn.ReLU(inplace=True)
        )
    def forward(self, x): return self.conv(x)
```

```
class UNet(nn.Module):
    """Standard Baseline U-Net"""
    def __init__(self):
        super().__init__()
        self.e1 = ConvBlock(1, 32)
        self.e2 = ConvBlock(32, 64)
        self.e3 = ConvBlock(64, 128)
        self.pool = nn.MaxPool2d(2)

        self.up2 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
        self.d2 = ConvBlock(128, 64)
        self.up1 = nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2)
        self.d1 = ConvBlock(64, 32)

        self.out = nn.Conv2d(32, 1, kernel_size=1)

    def forward(self, x):
        s1 = self.e1(x)
        s2 = self.e2(self.pool(s1))
        b = self.e3(self.pool(s2))

        d2 = self.d2(torch.cat([self.up2(b), s2], dim=1))
        d1 = self.d1(torch.cat([self.up1(d2), s1], dim=1))
        return torch.sigmoid(self.out(d1))
class AttentionBlock(nn.Module):
    def __init__(self, F_g, F_l, F_int):
        super().__init__()
        self.W_g = nn.Sequential(nn.Conv2d(F_g, F_int, 1), nn.BatchNorm2d(F_int))
        self.W_x = nn.Sequential(nn.Conv2d(F_l, F_int, 1), nn.BatchNorm2d(F_int))
        self.psi = nn.Sequential(nn.Conv2d(F_int, 1, 1), nn.BatchNorm2d(1), nn.Sigmoid())

    def forward(self, g, x):
        g1 = self.W_g(g)
        x1 = self.W_x(x)
        psi = self.psi(torch.relu(g1 + x1))
        return x * psi

class AttentionUNet(UNet):
    """Extends Baseline with Attention Gates"""
    def __init__(self):
        super().__init__()
        self.att2 = AttentionBlock(F_g=64, F_l=64, F_int=32)
        self.att1 = AttentionBlock(F_g=32, F_l=32, F_int=16)

    def forward(self, x):
        s1 = self.e1(x)
```

```
        s2 = self.e2(self.pool(s1))
        b = self.e3(self.pool(s2))

        # Apply attention before concatenating skip connections
        up2_out = self.up2(b)
        x2_att = self.att2(g=up2_out, x=s2)
        d2 = self.d2(torch.cat([up2_out, x2_att], dim=1))

        up1_out = self.up1(d2)
        x1_att = self.att1(g=up1_out, x=s1)
        d1 = self.d1(torch.cat([up1_out, x1_att], dim=1))

        return torch.sigmoid(self.out(d1))
class ResBlock(nn.Module):
    """A Convolutional Block with a Residual Shortcut Connection"""
    def __init__(self, in_c, out_c):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_c, out_c, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_c),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_c, out_c, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_c)
        )
        self.shortcut = nn.Sequential()
        # If the number of channels changes, we need a 1x1 conv on the shortcut
        if in_c != out_c:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_c, out_c, kernel_size=1),
                nn.BatchNorm2d(out_c)
            )
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        # The difference between baseline and res: F(x) + x
        return self.relu(self.conv(x) + self.shortcut(x))

class ResidualUNet(nn.Module):
    """U-Net architecture using Residual Blocks instead of standard convolutions"""
    def __init__(self):
        super().__init__()
        # Use ResBlock instead of ConvBlock
        self.e1 = ResBlock(1, 32)
        self.e2 = ResBlock(32, 64)
        self.e3 = ResBlock(64, 128)
        self.pool = nn.MaxPool2d(2)
```

```
        self.up2 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
        self.d2 = ResBlock(128, 64)
        self.up1 = nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2)
        self.d1 = ResBlock(64, 32)

        self.out = nn.Conv2d(32, 1, kernel_size=1)

    def forward(self, x):
        s1 = self.e1(x)
        s2 = self.e2(self.pool(s1))
        b = self.e3(self.pool(s2))

        d2 = self.d2(torch.cat([self.up2(b), s2], dim=1))
        d1 = self.d1(torch.cat([self.up1(d2), s1], dim=1))
        return torch.sigmoid(self.out(d1))


import cv2
import numpy as np
from skimage.metrics import peak_signal_noise_ratio as psnr
from skimage.metrics import structural_similarity as ssim
from sklearn.metrics import f1_score

def apply_traditional_cleaning(img_uint8):
    """Standard filtering + normalization pipeline."""
    # 1. Normalization
    norm = cv2.normalize(img_uint8, None, 0, 255, cv2.NORM_MINMAX)
    # 2. Filtering (Denoising)
    denoised = cv2.fastNlMeansDenoising(norm, None, 10, 7, 21)
    # 3. Sharpness enhancement (Unsharp Mask)
    blurred = cv2.GaussianBlur(denoised, (0, 0), 3)
    sharpened = cv2.addWeighted(denoised, 1.5, blurred, -0.5, 0)
    return sharpened

def calculate_stats(gt_norm, pred_float):
    """Compares Ground Truth to Prediction with full metrics suite."""
    # Scale to uint8 for visual metrics
    pred_uint8 = (np.clip(pred_float, 0, 1) * 255).astype(np.uint8)
    gt_uint8 = (gt_norm * 255).astype(np.uint8)

    # Basic Metrics
    mse = np.mean((gt_norm - pred_float) ** 2)
    p_val = 100.0 if mse == 0 else psnr(gt_norm, pred_float, data_range=1)
    s_val = ssim(gt_norm, pred_float, data_range=1, win_size=3)

    # Binary F1 (Thresholding at 0.5)
    f1 = f1_score((gt_norm > 0.5).flatten(), (pred_float > 0.5).flatten(), zero_division=0)
```

```
    return {
        "F-Measure": f1, "PSNR": p_val, "SSIM": s_val,
    }
def run_patch_inference(model, image_uint8, patch_size=256):
    """
    Handles padding, patching, and reconstruction for full-page inference.
    """
    # Convert to float32 and normalize
    img_float = image_uint8.astype(np.float32) / 255.0
    h, w = img_float.shape

    # Calculate padding so the image is perfectly divisible by patch_size
    pad_h = (patch_size - h % patch_size) % patch_size
    pad_w = (patch_size - w % patch_size) % patch_size

        padded_img = np.pad(img_float, ((0, pad_h), (0, pad_w)), mode='constant',
constant_values=1.0)

    # Create an empty canvas for the reconstruction
    reconstructed = np.zeros_like(padded_img)

    model.eval()
    with torch.no_grad():
        for y in range(0, padded_img.shape[0], patch_size):
            for x in range(0, padded_img.shape[1], patch_size):
                # Extract patch
                patch = padded_img[y:y+patch_size, x:x+patch_size]

                # Convert to tensor: (Batch, Channel, H, W)
                patch_tensor = torch.from_numpy(patch).unsqueeze(0).unsqueeze(0).to(DEVICE)

                # Forward pass
                prediction = model(patch_tensor)

                # Squeeze back to 2D numpy and place in canvas
                                        reconstructed[y:y+patch_size, x:x+patch_size] =
prediction.cpu().squeeze().numpy()

    # Crop the canvas back to the original image dimensions
    final_output = reconstructed[:h, :w]

    return final_output
import torch.nn.functional as F
import torchvision.models as models

def dice_loss(pred, target):
    smooth = 1e-5
```

```
    intersection = (pred * target).sum()
    return 1 - ((2. * intersection + smooth) / (pred.sum() + target.sum() + smooth))

class VGGPerceptualLoss(nn.Module):
    def __init__(self):
        super().__init__()
        # Load pre-trained VGG16 features
        vgg = models.vgg16(weights=models.VGG16_Weights.DEFAULT).features
        self.blocks = nn.Sequential(*list(vgg.children())[:16]).eval()
        for param in self.blocks.parameters():
            param.requires_grad = False

    def forward(self, pred, target):
        # VGG expects 3-channel RGB images, so we repeat the 1-channel grayscale
        pred_3c = pred.repeat(1, 3, 1, 1)
        target_3c = target.repeat(1, 3, 1, 1)

        f_pred = self.blocks(pred_3c)
        f_target = self.blocks(target_3c)
        return F.mse_loss(f_pred, f_target)

def train_model(model, train_loader, epochs=10, loss_type="bce_dice",
save_name='model.pth'):
    optimizer = optim.Adam(model.parameters(), lr=1e-4)
    bce_criterion = nn.BCELoss()
    mse_criterion = nn.MSELoss()
     perceptual_criterion = VGGPerceptualLoss().to(DEVICE) if loss_type == "perceptual" else
None

    # List to store history
    history = []

    model.train()
    for epoch in range(epochs):
        epoch_loss = 0
                for images, masks in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs}
[{loss_type}]", leave=False):
            images, masks = images.to(DEVICE), masks.to(DEVICE)

            optimizer.zero_grad()
            preds = model(images)

            if loss_type == "bce_dice":
                loss = bce_criterion(preds, masks) + dice_loss(preds, masks)
            elif loss_type == "mse":
                loss = mse_criterion(preds, masks)
            elif loss_type == "perceptual":
```

```
                    loss = mse_criterion(preds, masks) + 0.1 * perceptual_criterion(preds,
masks)

            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()

        avg_loss = epoch_loss / len(train_loader)
        history.append(avg_loss)
        print(f"Epoch {epoch+1} | {loss_type.upper()} Loss: {avg_loss:.4f}")

    torch.save(model.state_dict(), save_name)
    return history # Return the list of losses
# ========= Baseline U-Net =========
print("Training Baseline U-Net with BCE & Dice loss...")
baseline_bce = UNet().to(DEVICE)
bas_bce   =   train_model(baseline_bce,   train_loader,   epochs=10,   loss_type="bce_dice",
save_name="baseline_bce.pth")

print("Training Baseline U-Net with MSE loss...")
baseline_mse = UNet().to(DEVICE)
bas_mse    =    train_model(baseline_mse,    train_loader,    epochs=10,    loss_type="mse",
save_name="baseline_mse.pth")

print("Training Baseline U-Net with perceptual loss...")
baseline_perceptual = UNet().to(DEVICE)
bas_per = train_model(baseline_perceptual, train_loader, epochs=10, loss_type="perceptual",
save_name="baseline_perceptual.pth")

# ========= Residual U-Net =========
print("\nTraining Residual U-Net with BCE & Dice loss...")
residual_bce = ResidualUNet().to(DEVICE)
res_bce   =   train_model(residual_bce,   train_loader,   epochs=10,   loss_type="bce_dice",
save_name="residual_bce.pth")

print("Training Residual U-Net with MSE loss...")
residual_mse = ResidualUNet().to(DEVICE)
res_mse    =    train_model(residual_mse,    train_loader,    epochs=10,    loss_type="mse",
save_name="residual_mse.pth")

print("Training Residual U-Net with perceptual loss...")
residual_perceptual = ResidualUNet().to(DEVICE)
res_mse = train_model(residual_perceptual, train_loader, epochs=10, loss_type="perceptual",
save_name="residual_perceptual.pth")

# ========= Attention U-Net =========
print("\nTraining Attention U-Net with BCE & Dice loss...")
```

```
attention_bce = AttentionUNet().to(DEVICE)
att_bce = train_model(attention_bce, train_loader, epochs=10, loss_type="bce_dice",
save_name="attention_bce.pth")

print("Training Attention U-Net with MSE loss...")
attention_mse = AttentionUNet().to(DEVICE)
att_mse = train_model(attention_mse, train_loader, epochs=10, loss_type="mse",
save_name="attention_mse.pth")

print("Training Attention U-Net with perceptual loss...")
attention_perceptual = UNet().to(DEVICE)
att_per = train_model(attention_perceptual, train_loader, epochs=10,
loss_type="perceptual", save_name="attention_perceptual.pth")


models_to_test = {
    "Baseline BCE": baseline_bce,
    "Baseline MSE": baseline_mse,
    "Baseline Perceptual": baseline_perc,
    "Attention BCE": attention_bce,
    "Attention MSE": attention_mse,
    "Attention Perceptual": attention_perc,
    "Residual BCE": residual_bce,
    "Residual MSE": residual_mse,
    "Residual Perceptual": residual_perc
}

import matplotlib.pyplot as plt
import random
import cv2
import torch
import numpy as np

valid_pairs = []
for img, txt in zip(test_imgs, test_texts):
    if img and txt and os.path.exists(img) and os.path.exists(txt):
        valid_pairs.append((img, txt))

clean_test_imgs = [p[0] for p in valid_pairs]
clean_test_texts = [p[1] for p in valid_pairs]

def show_denoising_results(test_img_paths, models_dict, patch_size=256):
    """
    Shows a comparison of Degraded Input vs. Model Predictions.
    """
    # 1. Setup degradation tool and pick a random image
    deg_tool = SyntheticDegradation()
```

```
    img_path = random.choice(test_img_paths)

    # 2. Load and degrade
    raw_u8 = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if raw_u8 is None:
        print(f"Error: Could not load {img_path}")
        return

    # Convert to float for degradation tool, then back to u8 for inference
    deg_f = deg_tool.apply_all(raw_u8.astype(np.float32) / 255.0)
    deg_u8 = (deg_f * 255).astype(np.uint8)

    # 3. Generate predictions
    display_list = [("Degraded Input", deg_u8)]

    for name, model in models_dict.items():
        model.eval()
        model.to(DEVICE)

        with torch.no_grad():
            # Perform patch-based reconstruction
            pred_f = run_patch_inference(model, deg_u8, patch_size)
            # Clip and convert back to 0-255 image format
            pred_u8 = (np.clip(pred_f, 0, 1) * 255).astype(np.uint8)
            display_list.append((name, pred_u8))

    # 4. Plotting the grid
    n_images = len(display_list)
    fig, axes = plt.subplots(1, n_images, figsize=(24, 8))

    for ax, (title, img) in zip(axes, display_list):
        ax.imshow(img, cmap='gray')
        ax.set_title(title, fontsize=12)
        ax.axis('off')

    plt.tight_layout()
    plt.show()

# --- RUN IT ---
show_denoising_results(clean_test_imgs, models_to_test)
import os
import cv2
import numpy as np
import pandas as pd
import torch
import pytesseract
from tqdm import tqdm
```

```
from jiwer import wer, cer
from skimage.filters import threshold_sauvola

# --- CONFIGURATION & SETUP ---
DEVICE = torch.device("mps" if torch.backends.mps.is_available() else ("cuda" if torch.cuda.is_available() else "cpu"))
LIMIT_IMAGES = 20
PATCH_SIZE = 256

print(f"Using device: {DEVICE}")

# Initialize the synthetic degradation tool
deg_tool = SyntheticDegradation()

# --- PATH CLEANUP ---
valid_triplets = []
for i in range(len(test_imgs)):
    img_p = test_imgs[i]
    mask_p = test_masks[i] if i < len(test_masks) else None
    txt_p = test_texts[i] if i < len(test_texts) else None

    if img_p and mask_p and txt_p and os.path.exists(img_p) and os.path.exists(mask_p) and os.path.exists(txt_p):
        valid_triplets.append((img_p, mask_p, txt_p))

test_subset = valid_triplets[:LIMIT_IMAGES]
print(f"Verified {len(test_subset)} complete data triplets for evaluation.")


def run_patch_inference_fast(model, img_u8, patch_size=256):
    h, w = img_u8.shape
    pad_h = (patch_size - h % patch_size) % patch_size
    pad_w = (patch_size - w % patch_size) % patch_size
    padded = np.pad(img_u8 / 255.0, ((0, pad_h), (0, pad_w)), mode='constant', constant_values=1.0)

    output = np.zeros_like(padded)
    model.eval()
    model.to(DEVICE)

    with torch.no_grad():
        for y in range(0, padded.shape[0], patch_size):
            for x in range(0, padded.shape[1], patch_size):
                patch = padded[y:y+patch_size, x:x+patch_size]
                patch_t = torch.from_numpy(patch).unsqueeze(0).unsqueeze(0).float().to(DEVICE)
                pred = model(patch_t)
```

```
                output[y:y+patch_size, x:x+patch_size] = pred.cpu().squeeze().numpy()

    return output[:h, :w]


# --- STRUCTURAL & OCR METRICS COLLECTOR ---
image_metrics_results = []
ocr_storage = {}

def update_ocr_metrics(storage, prep, ocr, pred, gt):
    if prep not in storage: storage[prep] = {}
    if ocr not in storage[prep]: storage[prep][ocr] = {'W': [], 'C': []}
    pred_clean = pred.strip() if (pred and pred.strip()) else " "
    storage[prep][ocr]['W'].append(wer(gt, pred_clean))
    storage[prep][ocr]['C'].append(cer(gt, pred_clean))


# --- MAIN EVALUATION LOOP ---
for idx in tqdm(range(len(test_subset)), desc="Evaluating Dataset"):
    img_path, mask_path, txt_path = test_subset[idx]
    img_name = os.path.basename(img_path)

    # 1. Load ground truth text and images
    with open(txt_path, 'r', encoding='utf-8') as f:
        gt_text = f.read().strip()

    gt_img = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
    gt_norm = gt_img.astype(np.float32) / 255.0

    raw_u8 = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if raw_u8 is None or gt_img is None: continue

    # Resize large files to standard width
    if raw_u8.shape[1] > 1000:
        ratio = 1000 / raw_u8.shape[1]
        raw_u8 = cv2.resize(raw_u8, (1000, int(raw_u8.shape[0] * ratio)))
                gt_norm = cv2.resize(gt_norm, (1000, int(gt_norm.shape[0] * ratio)),
interpolation=cv2.INTER_NEAREST)

    # 2. Apply synthetic degradation
    deg_f = deg_tool.apply_all(raw_u8.astype(np.float32) / 255.0)
    deg_u8 = (deg_f * 255).astype(np.uint8)

    # ------------------------------------------
    # METHOD A: Otsu Binarization
    # ------------------------------------------
    _, otsu_bin = cv2.threshold(deg_u8, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

```
    otsu_norm = otsu_bin.astype(np.float32) / 255.0

    image_metrics_results.append({
        "Model": "Baseline Otsu", "Image": img_name, **calculate_stats(gt_norm, otsu_norm)
    })
                update_ocr_metrics(ocr_storage,      "Baseline      Otsu",      "Tesseract",
pytesseract.image_to_string(otsu_bin), gt_text)

    # ----------------------------------------------
    # METHOD B: Sauvola Binarization
    # ----------------------------------------------
    thresh_sauvola = threshold_sauvola(deg_u8, window_size=25)
    sauvola_bin = (deg_u8 > thresh_sauvola).astype(np.uint8) * 255
    sauvola_norm = sauvola_bin.astype(np.float32) / 255.0

    image_metrics_results.append({
              "Model":  "Baseline  Sauvola",  "Image":  img_name,  **calculate_stats(gt_norm,
sauvola_norm)
    })
               update_ocr_metrics(ocr_storage,      "Baseline      Sauvola",      "Tesseract",
pytesseract.image_to_string(sauvola_bin), gt_text)

    # ----------------------------------------------
    # METHOD C: Control (Unprocessed Degraded Data)
    # ----------------------------------------------
                update_ocr_metrics(ocr_storage,      "Degraded      Raw",      "Tesseract",
pytesseract.image_to_string(deg_u8), gt_text)

    # ----------------------------------------------
    # METHOD D: Deep Learning Models (The 6 Variants)
    # ----------------------------------------------
    for model_name, model_obj in models_to_test.items():
        # Get AI restored image
        f_pred = run_patch_inference_fast(model_obj, deg_u8, PATCH_SIZE)
        ai_gray = (np.clip(f_pred, 0, 1) * 255).astype(np.uint8)

        # Save structural scores
        image_metrics_results.append({
            "Model": model_name, "Image": img_name, **calculate_stats(gt_norm, f_pred)
        })

        # Run Tesseract on AI output text
                            update_ocr_metrics(ocr_storage,    model_name,    "Tesseract",
pytesseract.image_to_string(ai_gray), gt_text)


# --- POST-PROCESSING & DISPLAY TABLE GENERATION ---
```

```
print("\n=== FINAL STRUCTURAL METRICS (F1, PSNR, SSIM) ===")
df_structural = pd.DataFrame(image_metrics_results)
df_structural_summary = df_structural.groupby("Model").mean(numeric_only=True)
display(df_structural_summary)

print("\n=== FINAL OCR ACCURACY METRICS (CER, WER) ===")
final_ocr_rows = {}
for prep, ocrs in ocr_storage.items():
    row = {}
    for ocr, metrics in ocrs.items():
        row[(ocr, 'CER')] = np.mean(metrics['C'])
        row[(ocr, 'WER')] = np.mean(metrics['W'])
    final_ocr_rows[prep] = row

df_ocr = pd.DataFrame.from_dict(final_ocr_rows, orient='index')
df_ocr.columns = pd.MultiIndex.from_tuples(df_ocr.columns)
display(df_ocr)
```