ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Підвищення якості розпізнавання рукописних чисел за допомогою нейромережі»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артем МАТВІЙЧУК
(підпис)

Виконав: здобувач вищої освіти група ПДМ-64
Артем МАТВІЙЧУК

Керівник: Вікторія ЖЕБКА
д.т.н., професор

Рецензент: _____
науковий ступінь, вчене звання Ім’я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Матвійчуку Артему Миколайовичу

1. Тема кваліфікаційної роботи: Підвищення якості розпізнавання рукописних чисел за допомогою нейромережі

керівник кваліфікаційної роботи Вікторія ЖЕБКА д.т.н., професор,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, існуючі системи розпізнавання рукописних чисел, сучасні темоди та технології розробки інформаційних систем.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1.Огляд предметної галузі та актуальність .
2.Аналіз технологій розробки системи

3.Програмна реалізація системи.
4.Висновки.

5. Перелік графічного матеріалу: *презентація*

1. Критерії оцінювання якості розпізнавання.
2. Алгоритм роботи системи.
3. Математична модель попередньої обробки зображень.
4. Математична модель згорткової нейронної мережі.
5. Набір даних для навчання та тестування нейронної мережі.
6. Графік зміни кількості помилок від епохи.
7. Результати тестування розпізнавання.
8. Порівняння з існуючими системами.

6. Дата видачі завдання«19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Аналіз існуючих методів розпізнавання рукописних чисел	06.11-12.11.23	
3	Побудова алгоритму роботи системи та математичних моделей застосованих операцій	13.11-19.11.23	
4	Розробка алгоритму попередньої обробки зображень	20.11-26.11.23	
5	Розробка нейронної мережі	27.11-03.12.23	
6	Проведення навчання нейронної мережі та перевірка отриманих результатів	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Артем МАТВІЙЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Вікторія ЖЕБКА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 71 стор., 7 табл., 44 рис., 27 джерел.

Мета роботи – підвищення якості розпізнавання рукописних чисел.

Об'єкт дослідження – процес розпізнавання рукописних чисел.

Предмет дослідження – алгоритми штучного інтелекту, які використовуються для розпізнавання рукописних чисел.

Короткий зміст роботи: У роботі проведено дослідження процесу розпізнавання рукописних чисел нейронними мережами. Проаналізовано методи та алгоритми попередньої обробки зображень, враховуючи їх вплив на ефективність системи. Проаналізовано основні види нейронних мереж, що можуть бути використані для задачі класифікації зображень. розроблено реалізацію запропонованих методів, і отримані конкретні результати підтверджують ефективність запропонованих підходів.

КЛЮЧОВІ СЛОВА: РУКОПИСНІ ЦИФРИ, ПОПЕРЕДНЯ ОБРОБКА ЗОБРАЖЕНЬ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ЗВОРОТНЄ ПОШИРЕННЯ ПОМИЛКИ

ABSTRACT

Text part of the master's qualification work: 71 pages, 44 pictures, 7 tables, 27 sources.

The purpose of the work is improving the quality of recognition of handwritten numbers.

The object of research – the process of recognizing handwritten numbers.

The subject of research – artificial intelligence algorithms used to recognize handwritten numbers.

Summary of the work: The paper investigates the process of handwritten number recognition by neural networks. Image preprocessing methods and algorithms have been analyzed, considering their impact on system efficiency. The main types of neural networks that can be employed for image classification have been examined. The implementation of the proposed methods has been developed, and specific results obtained confirm the effectiveness of the proposed approaches.

KEYWORDS: HANDWRITTEN DIGITS, IMAGE PREPROCESSING, CONVOLUTIONAL NEURAL NETWORKS, BACKPROPAGATION

ЗМІСТ

ВСТУП

Зростаюча кількість даних, що генеруються в даний час в різноманітних сферах життя ставить виклик у розробці більш точних та швидких методів розпізнавання рукописних чисел. З приходом цифровізації у наше життя, все більші об’єми даних опираються на послідовності цифр, адже саме в такому форматі комп’ютери мають найбільшу продуктивність для роботи з даними. Щоправда, суспільство та держави, що будувалися століттями, не мають змоги так швидко адаптуватися до сьогоденних реалій, тому й досі більшість документів, заявок, посвідчень тощо мають паперовий вигляд.

Мета роботи – підвищення якості розпізнавання рукописних чисел.

Завдання магістерської роботи:

1. Проаналізувати існуючі методи розпізнавання рукописних чисел
2. Побудувати алгоритм роботи системи та математичні моделі застосованих операцій
3. Розробити алгоритм попередньої обробки зображень
4. Розробити нейронну мережу
5. Провести навчання нейронної мережі та перевірити отримані результати

Об'єкт дослідження – процес розпізнавання рукописних чисел

Предмет дослідження – алгоритми штучного інтелекту, які використовуються для розпізнавання рукописних чисел

Використаними методами дослідження є моделювання, порівняння, вимірювання та експеримент.

1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ ТА АКТУАЛЬНІСТЬ

1.1 Опис предметної галузі та її застосування

Розпізнавання рукописного тексту – це процес перетворення тексту, створеного рукою людини (на папері, в різноманітних формах для малювання на комп'ютері тощо) у формат, який може бути зрозумілий обчислювальними системами (комп'ютер, смартфон, форма веб-сторінки тощо). Після написання тексту, його необхідно перевести у формат зображення (існують пристрої, що дозволяють виконувати цей процес прямо з паперу, але, як результат, система на вхід отримує оптичні дані) для подальшого аналізу програмним забезпеченням. Цей процес включає в себе виявлення символів на вихідному зображенні та їх класифікацію.

Однак, це зовсім не тривіальна задача, адже в процесі розпізнавання комп'ютер стикається з великим переліком складнощів, що сильно впливають на точність та швидкість цього процесу. Основними викликами є [1]:

1. Різноманітність почерку – кожна людина має унікальний стиль написання. Це означає, що один і той же символ може виглядати по-різному у відповідних почерках. Сюди також входить різноманітність способів написання символів (наприклад, літера Т може бути написана як «т» та «m»).
2. Наявність шуму – рукописний текст може бути зіпсований шумом на зображенні, таким як розмазування, вицвітання чорнила, сліди ручки тощо.
3. Зміна розміру та масштабу – рукописні символи можуть бути написані в різних розмірах та масштабах, що вимагає адаптації системи.
4. Перекриття символів – у рукописному тексті символи можуть перекриватися один одним, що робить їх виділення та розпізнавання важким завданням.
5. Обмежена якість зображення – якщо зображення рукописного тексту має низьку якість або низьку роздільну здатність, це може призвести до втрати важливої інформації і зробити розпізнавання менш точним.

Зазначені складнощі відносяться як до всієї задачі розпізнавання рукописного тексту загалом, як і до розпізнавання чисел зокрема.

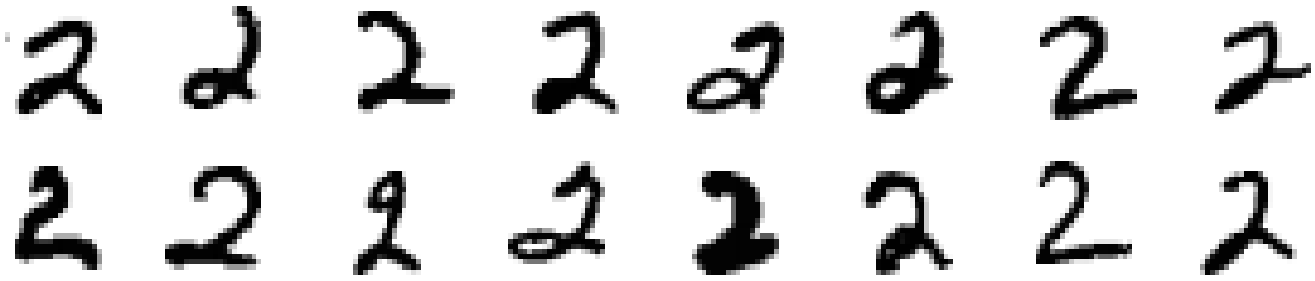


Рис. 1.1 Шістнадцять способів написання цифри «2»

Технологічний процес, призначений для автоматизованого розпізнавання рукописного тексту називають Intelligent Character Recognition (ICR). Ця назва походить від попередника – OCR (Optical Character Recognition), що використовується для сканування та розпізнавання друкованого тексту. Однією з провідних OCR систем є Tesseract OCR, розроблена компанією Hewlett-Packard. Не дивлячись на те, що це OCR система, сучасні версії системи мають змогу розпізнавати й рукописні числа. Так, у роботі «Ефективність попередньої обробки зображень у визначенні цифрових рукописів із застосуванням OCR Tesseract» [2] зазначається, що система має 30% точність на звичайних зображеннях, проте використовуючи попередню обробку зображень, автори змогли збільшити точність у кілька разів – до майже 80%.

У своїй роботі Томас Бреуель [3] зазначає, що перші системи ICR були створені в ранніх 1990-х роках. Для своєї роботи вони використовували існуючі на той момент методи обробки зображень, знаходження особливих ознак, притаманних для окремих символів, статистичні дані, структуру тексту та інші техніки для розпізнавання символів. Багато систем зрівнювали символи з наявною базою і видавали результат з найбільшим відсотком співпадінь.

Проте, ефективність таких систем не була достатньою, тому у 2010-х роках такі системи почали використовувати різні види нейронних мереж та їх комбінації

для підвищення швидкості та якості розпізнавання рукописного тексту. У своїй роботі Томас Бреуель підкреслює, що впровадження рекурентних нейронних мереж (зокрема Long Short-Term Memory, LSTM) для розпізнавання тексту покращила якість та точність систем ICR.

Рис. 1.2 Алгоритм роботи систем ICR

1.2 Актуальність дослідження в галузі розпізнавання рукописних чисел

Зростаюча кількість даних, що генеруються в даний час в різноманітних сферах життя ставить виклик у розробці більш точних та швидких методів розпізнавання рукописних чисел. З приходом цифровізації у наше життя, все більші об’єми даних опираються на послідовності цифр, адже саме в такому форматі комп’ютери мають найбільшу продуктивність для роботи з даними. Щоправда,

суспільство та держави, що будувалися століттями, не мають змоги так швидко адаптуватися до сьогоденних реалій, тому й досі більшість документів, заявок, посвідчень тощо мають паперовий вигляд.

До прикладу, сучасний формат паспорта громадянина України (ID-картка) у порівнянні застарілим паспортом у формі книжечки має такі особливості:

- було додано унікальний номер запису в Єдиному державному демографічному реєстрі, що складається з 13 цифр, представлених двома послідовностями з восьми та п'яти цифр розділених текстовим символом «-» [4];
- серію та номер документу формату «ЕЕ000000» замінено на номер документу, що складається з дев’яти цифр;
- повну назву державного органу, яким видано документ було замінено на його номер, що складається з чотирьох цифр;
- було додано реєстраційний номер облікової картки платника податків (РНОКПП);
- назви місяців на документі замінені на їх порядковий номер у році.

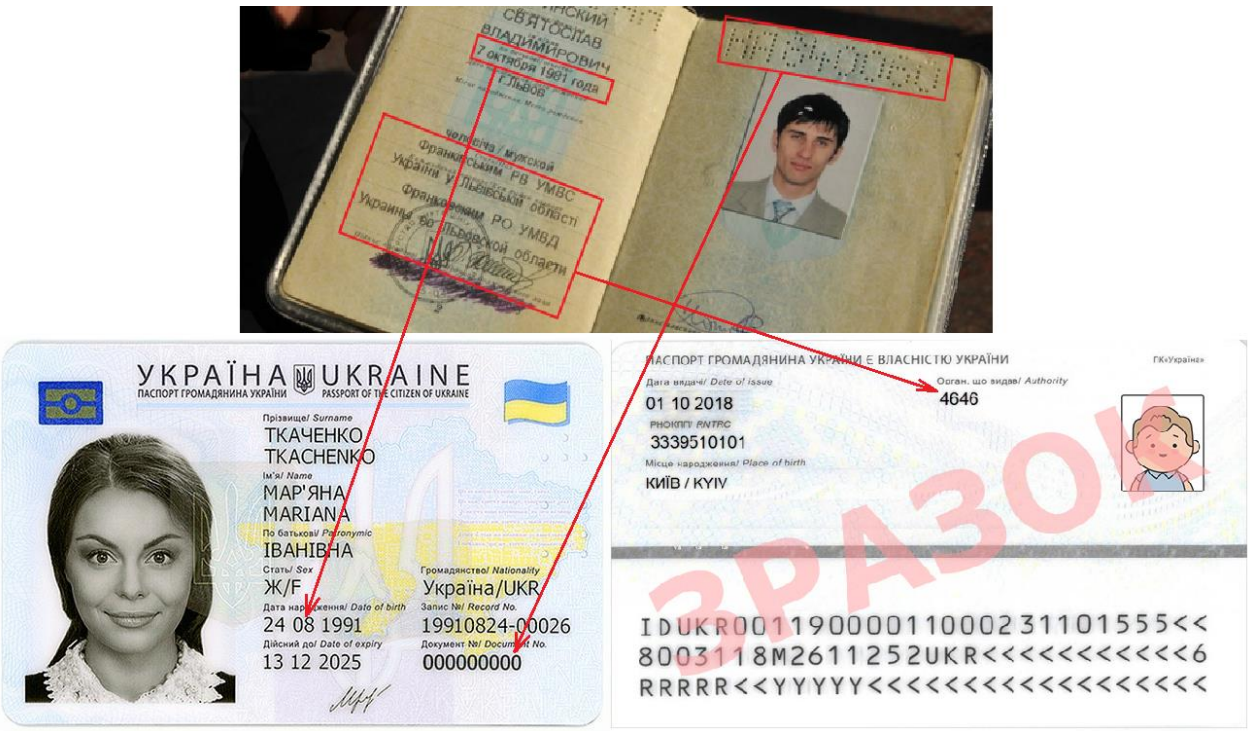


Рис. 1.3 Зміни у паспорті громадянина України

Проте, Україна не була першою державою, що запровадила документи, з біометричними даними та переважанням цифрової інформації. Тенденція на такі зміни прийшла із Заходу, де цифровізація почала відбуватися раніше, тому ці речі стосуються переважної частини людства.

Більшість цієї інформації є необхідною при заповненні паперових документів, бланків, заяв тощо. До того ж варто згадати про номер мобільного телефону, що за останні півтора десятки років став обов'язковим полем для заповнення в такого роду діяльності.

У разі необхідності перенесення заповнених документів в електронний формат, що, насправді, виконується у переважній більшості випадків, потрібно задіювати людський ресурс – людину, завдання якою вручну переписувати ці дані. Такий підхід, очевидно, не є високопродуктивним, адже кожна літера та цифра займає відносно велику кількість часу, у порівнянні з тим, як з цією задачею міг впоратися комп'ютер.

При збільшенні потоку даних може з'явитися потреба у додаткових працівниках, що виконуватимуть все ту ж роботу. Також, варто зазначити, що людина через втому, погане самопочуття чи звичайну неуважність може допустити помилку. До того ж, дослідження показують, що люди можуть залишатися уважними та продуктивними лише кілька годин на добу при роботі з комп'ютером [5]. Тож збільшення кількості залучених людей в обробку паперових документів збільшується і ймовірність виникнення помилок при їх заповненні.

Як можна зрозуміти, помилки в таких речах можуть коштувати дуже дорого, а крім того забирати важливий час. Це приносить збитки як підприємцям і державним структурам так і пересічним громадянам.

Окрім документообігу, люди також займаються ручним сортуванням у сфері поштових перевезень. До прикладу, у разі надсилання листів або посилок, в яких дані, такі як поштовий індекс, номер будинку, квартири тощо, вказуються від руки. Хоча такий тип комунікацій здається застарілим та вже виходить з масового вжитку, він все ще являється важливим по всьому світу.

Також, підвищення якості розпізнавання рукописних чисел може внести помітне покращення у галузь освіти та науки, а саме туди, де є математичні обрахунки. До прикладу, під час проведення аудиторних занять, або підготовці до них, часто є необхідним написання формул, розрахунків, математичних доведень тощо. У разі необхідності оцифрувати дані для створення звіту або презентації за темою занять, це може зайняти велику частину часу, адже наявні інструменти написання складних математичних обрахунків, через свою різноманітність та всеохопність, є нелегкими в освоєнні та навіть досвідченні користувачі можуть витратити більше часу, ніж така задача дійсно заслуговує.

Наразі існує два способи написання цифрових формул: текстовий та за допомогою конструкторів. Кожен з них має свої переваги та недоліки, але їх об’єднує те, що вони не значно зручні. Також існує графічний метод введення, до прикладу у програмі Microsoft Excel, але його точність та привильність залишає бажати кращого.

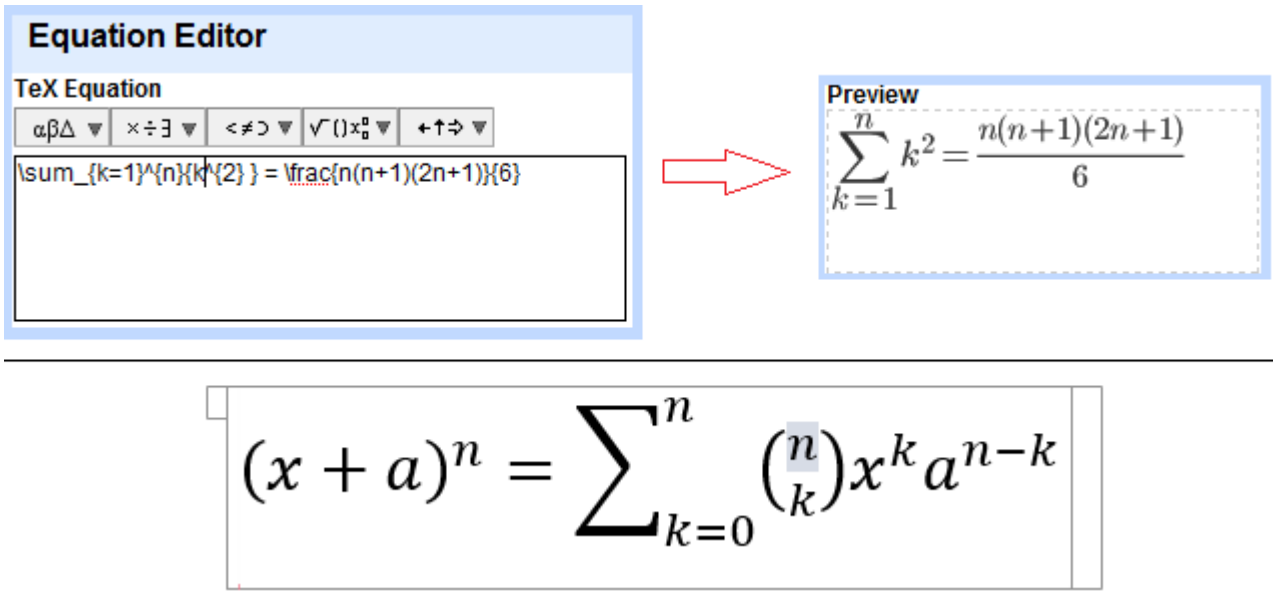


Рис. 1.4 Головні способи введення формул

Методи та технології, що будуть використанні для підвищення якості розпізнавання рукописних цифр можуть дати поштовх всій галузі загалом і

призвести до підвищення якості розпізнавання рукописних формул будь-якої складності, що допоможе скоротити час їх написання в цифровому вигляді.

Це може допомогти, до прикладу, у створенні онлайн завдань з математики, де у відповіді потрібно отримати вираз. Наразі для таких цілей можуть бути використані два види їх написання, зазначені вище, проте перший спосіб може для когось виявитися занадто складним, а другий може містити підказки, у разі, якщо використаний шаблон для написання відповіді. У разі, якщо відповідь може бути вписана через знаряддя для малювання, або завантаження фото рукописної відповіді, завдання потребує перевірки людиною, що може бути недопустимим у випадку великого об’єму даних, до прикладу в онлайн університетах чи курсах.

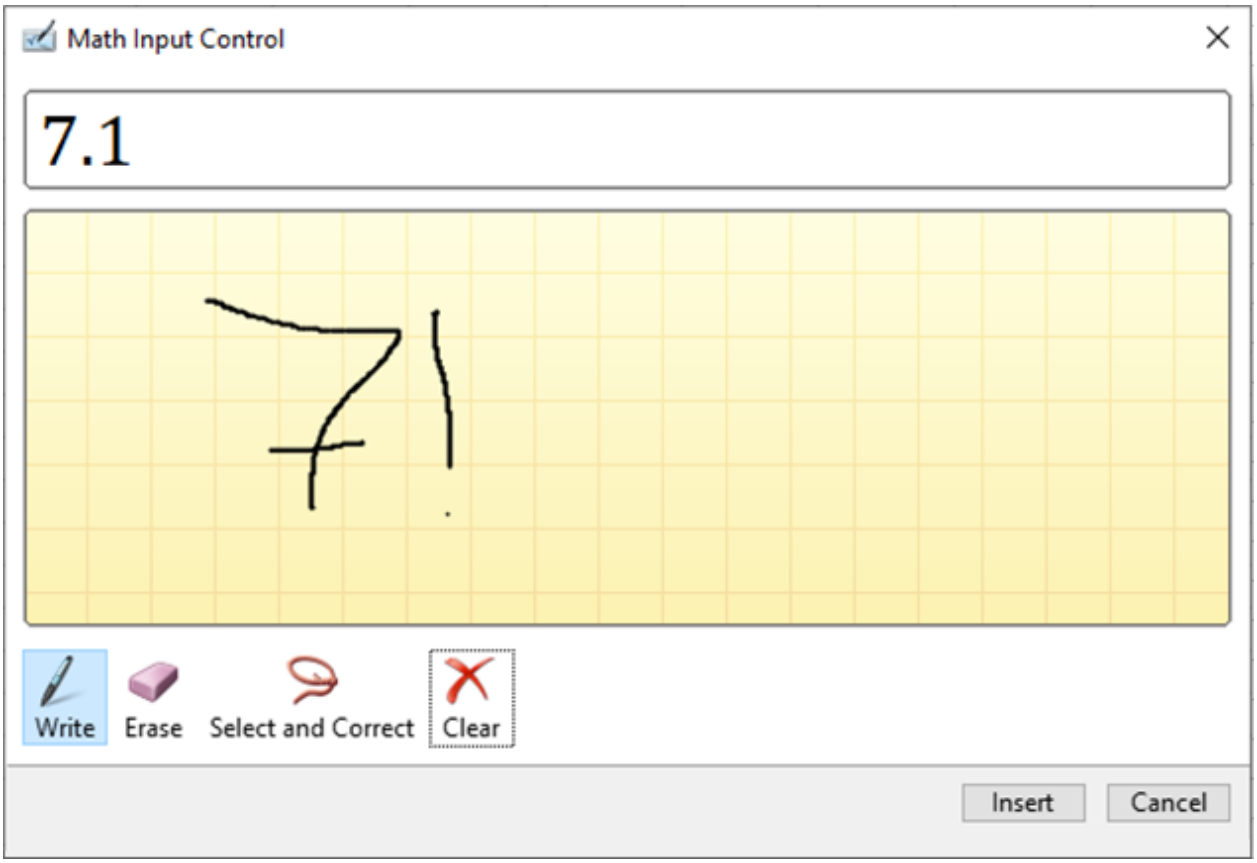


Рис. 1.5 Неточність розпізнавання формул в Excel

Також, покращення якості та ефективності розпізнавання рукописних чисел може збільшити швидкість та простоту створення електронних конспектів для

студентів, адже написання формул в таких конспектах сильно впливає на продуктивність їх створення.

1.3 Роль нейронних мереж та їх застосування в контексті покращення розпізнавання

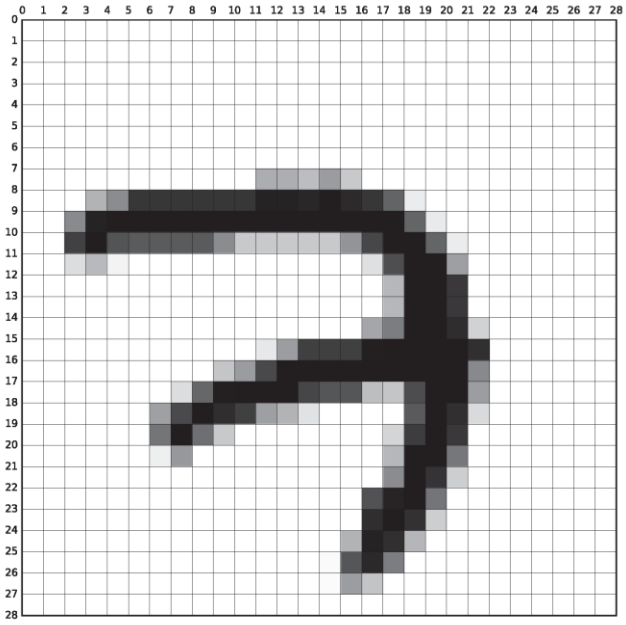
Точність сучасних систем розпізнавання рукописних чисел сильно варіюється в залежності від обраних технологій при розробці того чи іншого методу. ICR системи, що використовують алгоритми, статистичні дані та порівняння для визначення найбільшої кількості співпадінь мають точність, що не дозволяє їх практичне застосування [6]. Саме тому ця галузь перейшла до використання нейронних мереж. До того ж, окрім вищої точності, це дозволяє позбутися необхідності використання масивів даних при безпосередньому використанні, тому таким системам, зазвичай, не потрібне підключення до мережі Інтернет.

Попри всі спроби створення високоефективних засобів для розпізнавання рукописних цифр, наразі сучасні системи досягають лише 80-90% точності у своїй роботі [7]. Це, в свою чергу, означає, що при реальному використанні такі системи будуть робити помилку в кожному 5-10 символі, що є критично недопустимим у випадку оцифруванні, наприклад, рукописної інформації з документів.

Такі, хоча й високі, але не практичні показники пов’язані не лише зі складністю розпізнавання, через велику варіацію способів написання символів, але й через обмежений набір даних для навчання нейронних мереж та використання непідходящих видів нейронних мереж.

Так, у розробці системи, може бути використаний замалий набір даних, що зробить її неспроможною правильно розпізнавати числа, написані в іншій манері, навіть при чітких вхідних даних. Окрім великої вибірки, набір має включати якомога більше способів написання цифр, на різних поверхнях. Також для високої якості результату, вхідні картинки мусять містити різноманітні кольори, різне освітлення поверхні тощо.

Одним з найпопулярніших наборів даних, для створення моделей для розпізнавання рукописних цифр є MNIST (Mixed National Institute of Standards and Technology). Він містить 70 000 картинок з цифрами від 0 до 9, з яких 60 000 для навчання нейронних мереж та 10 000 для перевірки результатів навчання [8]. Для створення цього набору, оригінальні зображення цифр були стиснуті до розмірі 20*20 пікселів. Після цього було додано по декілька пікселів з кожної зі сторін для того, щоб цифра залишалася посередині зображення. Вихідні зображення мають розміри 28*28 пікселів.



(**a**) MNIST sample belonging to the digit ‘7’.

(**b**) 100 samples from the MNIST training set.

Рис. 1.6 Приклад цифри з набору MNIST

У статті авторства Елізабет Рані, Абхігна Редді та інших [9] досліджується застосування глибокої згорткової нейронної мережі (CNN) для розпізнавання рукописних чисел в банківській сфері. Основною метою дослідження є розробка та оптимізація архітектури CNN для точного та надійного визначення цифр, які можуть зустрічатися в банківських документах, зокрема в чеках та інших фінансових відомостях. Автори статті розглядають процес тренування глибокої згорткової нейронної мережі на наборі даних MNIST. Дослідження також включає порівняльний аналіз ефективності згорткової нейронної мережі на різних етапах

розпізнавання, зокрема етапу виділення ознак та етапу класифікації. Враховуються питання безпеки та швидкодії для забезпечення високої ефективності та надійності в застосуванні цієї технології у фінансовому секторі. Головною ціллю дослідження автори зазначають бажання зробити банківські операції, де використовуються написані від руки числа, більш простими та безпомилковими.

Ще однією роботою, що використовує набір MNIST є «Розпізнавання рукописного тексту MNIST за допомогою згорткових нейронних мереж (CNN)» [10]. Автори представили офлайн метод розпізнавання рукописних цифр, що базується на різних техніках машинного навчання. Запропонована у статті техніка є більш корисною для непов'язаних рукописних даних у порівнянні з пов'язаними даними. Основна мета цієї статті – забезпечити ефективні та надійні методи ідентифікації транскрибованих цифр. Тобто, створена система не надто добре працює з зображеннями, на яких знаходиться більше однієї цифри. Не дивлячись на такий недолік, дана робота робить важливий внесок у розуміння процесу розробки ефективної нейронної мережі для розпізнавання рукописних чисел.

Не зважаючи на переваги цього набору даних, він не може бути використаний для створення реальної системи розпізнавання рукописних чисел через декілька причин:

1. Всі вхідні зображення мають невеликий розмір, що обмежує їх потенційну варіативність.
2. Невеликі розміри картинок змушують стискати вхідні дані при реальному використанні, що може сильно їх спотворювати.
3. Всі картинки мають однотонний фон, що виключає ймовірність шумів на зображенні.

Набір даних MNIST можна назвати «тепличними умовами» для систем розпізнавання рукописних цифр, адже він має багато спрощень. Але попри це його часто використовують для перевірки нових видів нейронних мереж для таких задач. Останні праці дослідників показують, що з цим набором можна досягти майже 95% точності передбачень [11].

1.4 Огляд існуючих систем розпізнавання рукописних чисел

У мережі Інтернет можна знайти безліч безкоштовних та платних сервісів, що допомагають вилучати текст з зображень. Більшість з них орієнтуються саме на розпізнавання букв та слів, проте неважко знайти такі, що мають модулі для аналізу рукописних чисел. Для визначення точності сучасних систем для розпізнавання цифр, написаних від руки, було обрано чотири сервіси:

1. OCR Space (https://ocr.space/)
2. Pen to Print (https://www.pen-to-print.com/)
3. Image To Text (https://www.imagetotext.info/)
4. img2txt (https://img2txt.com/)

Для оцінки їх ефективності буде проведене тестування двома зображеннями, що містять цифри від одиниці до нуля (1, 2, 3, 4, 5, 6, 7, 8, 9, 0). Зображення матимуть різну складність: складність першого було оцінено в чотири бали з десяти, а друге – у сім.

Після виконання операцій буде створена таблиця з результатами тестування ресурсів. Результати тестування включатимуть в себе:

- кількість правильних символів – максимум десять;
- кількість секунд, що знадобилися системі для визначення результату (час на завантаження та відправку даних не враховується – лише тривалість обробки сервером);

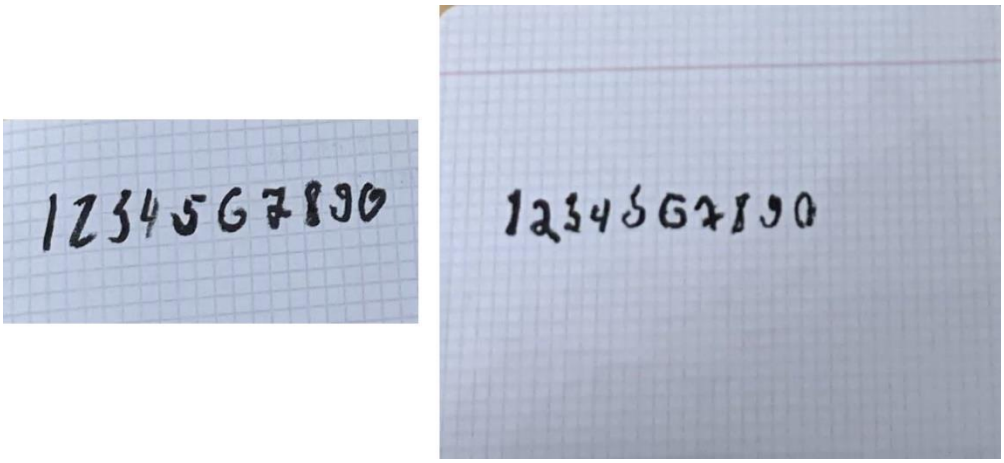


Рис. 1.7 Зображення, які будуть використанні для тестування систем

OCR Space – безкоштовний сервіс для перетворення сканованих документів або фотографій зі смартфону в редагований текст, також має платну версію. Має публічне API, що робить його можливим для використанні при розробці власних додатків, де є необхідною функція розпізнавання тексту. Підтримує десятки мов. Має три модуля – основний, покращений для розпізнавання цифр та спеціальних символів та експериментальний з підтримкою нових мов [12]. При тестуванні буде використано другий модуль.

При тестуванні сервісу на першому зображенні було отримано результат «0680201221». Настільки неточний результат може бути пояснений тим, що система допустила помилку в орієнтуванні зображення, адже цифри «068» на початку виглядають як перевернуті «890». Після коригування зображення, з другої спроби цей сервіс надіслав результат «1234567890», що є абсолютно вірною відповіддю. З урахуванням першої помилки, розпізнавання можна оцінити в 5 балів. Час, що знадобився для опрацювання – 1.82 секунди. При тестуванні на другому зображенні було отримано результат «1234567/30». Точність склала вісім символів з десяти, час – 1.02 секунди.

Рис. 1.8 Інтерфейс Pen To Print

Pen To Print – онлайн сервіс, що позиціонує себе як відмінним рішенням для переведення рукописного тексту в цифровий формат. Має обмежений доступ у безкоштовному вигляді – 10 зображень, проте має непогану точність. Щоправда, сервіс є чутливим до якості вхідного зображення.

При першому запуску сервіс зумів абсолютно правильно розпізнати всі цифри за 5.58 секунди. Проте, нажаль, допустився помилки при обробці другого зображення. Отриманий результат – «1234567830» за 5.61 секунди.

Image To Text – сервіс, що не сильно вирізняється від двох попередніх. З особливостей – наявність API, платного функціоналу та підтримка більшої кількості форматів зображення. При запуску тестування на першому зображенні було отримано відмінний результат «1234567890», проте запит виконувався доволі довго – 6.26 секунди. Точність обробки другого зображення була трішки нижчою – 8 символів («1234564830») за менший час у 5.62 секунди.

img2txt – безкоштовний онлайн сервіс, що пропонує послуги з розпізнавання тексту з зображень або PDF файлів, без можливості платної підписки. Існує вже майже десять років – з 2014 року. Має підтримку більше 35 мов, виведення результатів у чотирьох форматах. За час свого існування сервіс опрацював понад 43 мільйони запитів. Сервіс побудований на програмі Tesseract OCR, розробленою компанією Hewlett-Packard. Недоліком цього сервісу є його чутливість до якості вхідних даних [13].

Після завантаження першого зображення, було отримано такий результат: «113456#190», що становить 7 балів; час, витрачений на обробку – 0.784 секунди. Обробка другого зображення зайняла відчутно більше часу – 2.84 секунди, проте результат не містив ні однієї правильно визначеної цифри.

Результати, отримані при тестуванні існуючих моделей, що здатні розпізнавати рукописні числа наведено в таблиці 1.4.1.

Як видно за таблиці, найкращий результат був отриманий за допомогою сервісу Pen To Print, незважаючи на відносно велику кількість часу, що знадобилася йому для цього. Окремо варто виділити OCR Space та його модуль розпізнавання цифр, що не зважаючи на помилку в орієнтуванні першого зображення, зміг

правильно розпізнати всі символи на корегованому екземплярі, адже час, за який він впорався з завданням, у рази менший конкурентів з високим відсотком точності розпізнавання.

Таблиця 1.1

Результати тестування систем

Назва	**Результат тестування №1**	**Час, витрачений на тестування №1, с**	**Результат тестування №2**	**Час, витрачений на тестування №2, с**
OCR Space	5	1.82	8	1.02
Pen To Print	10	5.58	9	5.61
Image To Text	10	6.26	8	5.62
img2txt	7	0.784	0	2.84

Проте, таке тестування сучасних комерційних та безкоштовних сервісів, що навіть мають власні API для їх інтегрування у додатки сторонніх розробників, дає чітку картину того, що сьогоднішній рівень розпізнавання рукописних чисел не є достатнім для серйозного використання на повсякденній основі.

2 АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ СИСТЕМИ

2.1 Етапи роботи системи

Загалом, процес вилучення цифрової інформації з зображення можна розділити на такі етапи:

1. Попередня обробка зображення;
2. Вилучення об’єктів (цифр);
3. Підготовка до розпізнавання;
4. Розпізнавання об’єктів;
5. Генерація результату.

Попередня обробка зображення необхідна для кращого виділення областей інтересу на зображенні (рукописні цифри). Це дозволить наступним крокам бути більш продуктивними та швидкими. Окрім того, таким чином можна зменшити кількість шумів на вихідному зображенні.

Вилучення об’єктів – це процес, що виділятиме окремі області обробленого зображення, що містять рукописні цифри. Для цього буде застосовано Алгоритм обходу зображення для пошуку межових прямокутників з використанням обходу в ширину, що дозволяє виділяти об’єкти на зображенні за яскравістю пікселів.

Підготовка для розпізнавання є важливим етапом при підготовці до розпізнавання, адже після виділення об’єктів на зображенні всі вони мають свої унікальні розміри. При таких умовах розпізнавання неможливе, адже нейронні мережі проєктуються для зображень певного розміру.

Наступним етапом є розпізнавання цифр з зображень. Для цієї задачі буде створено згорткову нейронну мережу, що буде натренована на власному наборі даних, що буде створено за допомогою попередніх етапів. В такому разі тренувальний набір буде максимально близький до того, що мережа буде отримувати при своїй роботі.

Етап генерації результату – визначення рядків та відступів на зображенні для створення відповідного відношення між об’єктами в результуючому тексті.

2.2 Попередня обробка зображення

Як було зазначено вище, попередня обробка зображення необхідна для кращого виділення областей інтересу на зображенні. Вона складатиметься з кількох етапів:

1. Застосування фільтра підвищення різкості зображення;
2. Застосування розмивання Гауса;
3. Застосування порогового значення для яскравості пікселя.

Фільтр підвищення різкості — це лінійний фільтр, який використовується для підсилення контурів та підвищення різкості зображення. Він застосовується для підсилення високочастотних компонентів зображення, що відповідають за різкі переходи в інтенсивності пікселів.

Математично, операція фільтрації підвищення різкості може бути визначена за допомогою наступного ядра [14]:

$$H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Це ядро робить піксель у центрі більш вагомим, а сума всіх коефіцієнтів в рядку або стовпці дорівнює 0. Це призводить до збереження суми інтенсивностей пікселів та підсилення будь-яких різниць в інтенсивності.

Використання цього фільтру допоможе чіткіше виділити цифри на зображенні та прибрати велику кількість шуму.

Розмивання Гауса — це лінійний фільтр, який застосовується до зображення з метою розмиття, часткового видалення високочастотних шумів та згладжування різких переходів в інтенсивності пікселів. Цей фільтр базується на використанні функції Гауса для вагового коефіцієнту кожного пікселя у фільтрі. Математично, операція розмивання Гауса визначається конволюцією вхідного зображення I та ядра Гауссівського фільтра G [15].

Функція Гауса визначається виразом:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

де σ — стандартне відхилення Гауссівської функції, яке контролює ступінь розмиття.

Операція розмивання Гауса ефективно згладжує текстури та видаляє дрібні деталі, роблячи зображення більш м'яким. Застосування фільтра зі збільшеним значенням σ призводить до більшого розмиття. Розмивання Гауса широко використовується в обробці зображень, особливо перед застосуванням операцій виявлення країв, сегментації та інших обробок, де розмиття може поліпшити результати алгоритмів.

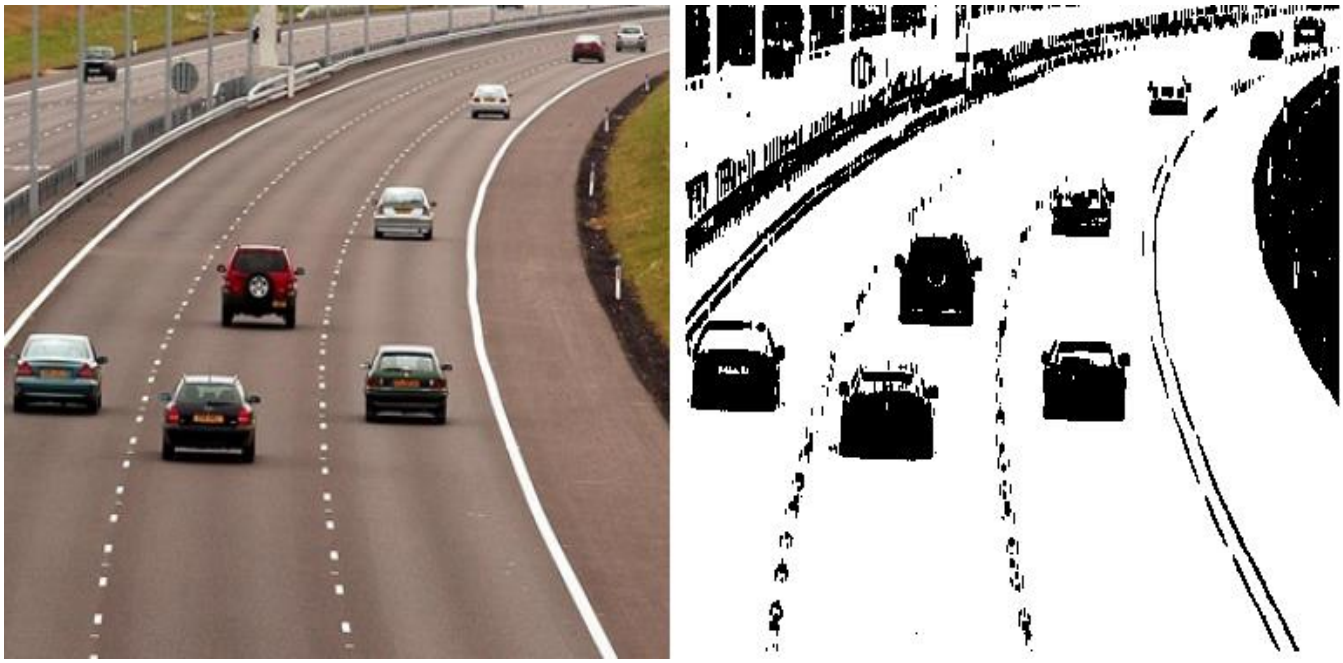


Рис. 2.9 Попередня обробка зображення

Після обробки вхідного зображення заданими фільтрами, необхідно зробити його чорно-білим. Для цього буде застосовано порогове значення яскравості пікселя. По суті, це означає, що якщо конкретний піксель має яскравість менше порогового значення – він стає чорним, у іншому випадку – білим. Такий алгоритм дозволяє перетворювати зображення таким чином, як показано на рисунку 2.2.1.

Як видно з прикладу, така обробка допомагає виділяти об’єкти інтересу на зображенні, а перетворення у чорно-білий формат полегшує та прискорює подальшу обробку.

2.3 Алгоритм обходу зображення для пошуку межових прямокутників з використанням обходу в ширину

Межовий прямокутник у контексті комп'ютерного бачення та обробки зображень є прямокутною областю, яка охоплює об'єкт чи область інтересу на зображенні. Цей прямокутник визначається за допомогою мінімальних та максимальних значень координат по горизонталі та вертикалі, які визначають положення верхнього лівого та нижнього правого кутів прямокутника відносно координатного простору зображення [16].

Межові прямокутники використовуються для локалізації та обгортання об'єктів або областей на зображенні. Основна мета — надати простий та ефективний засіб для розташування та визначення геометричних властивостей об'єктів для подальшого аналізу чи обробки.

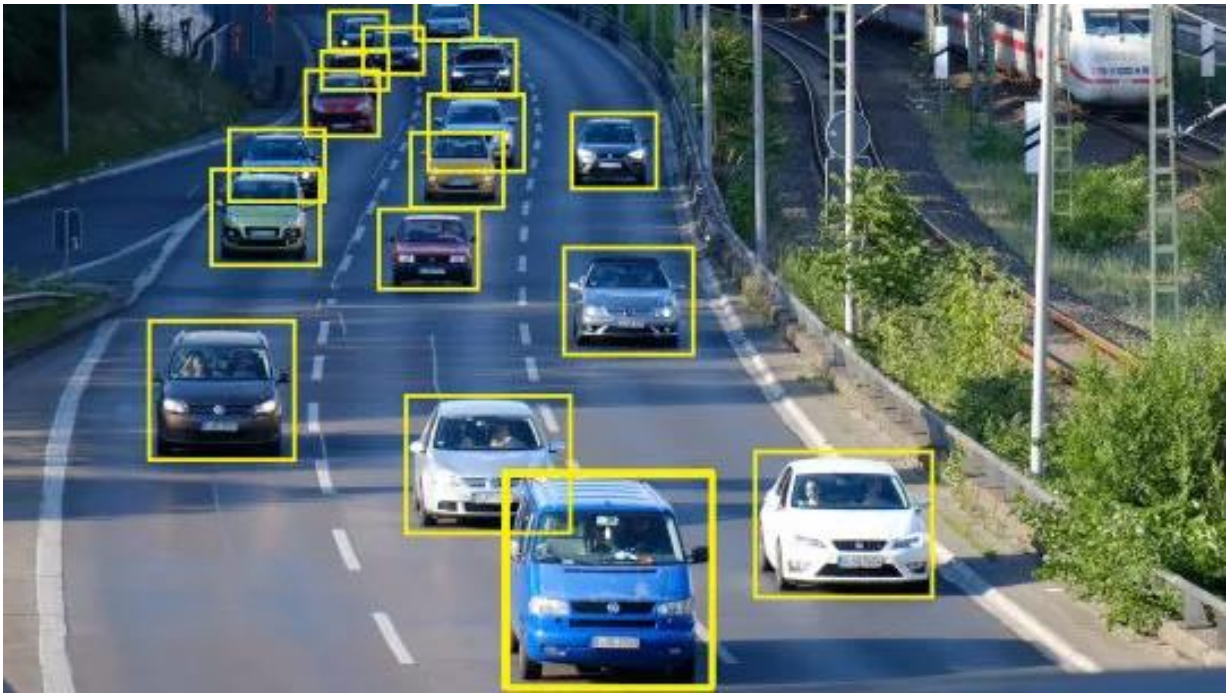


Рис. 2.10 Межові прямокутники навколо автомобілів

Межовий прямокутник описується координатами двох точок: лівого верхнього кута з координатами (x_{min}, y_{min}) та правого нижнього кута з координатами (x_{max}, y_{max}). Довжина сторін прямокутника визначається як x_{max} - x_{min}, а ширина — як y_{max} - y_{min}.

Використання межових прямокутників широко поширене у задачах виявлення об'єктів, відстеження об'єктів, сегментації зображень та інших областях комп'ютерного зору та обробки зображень. Застосування межових прямокутників дозволяє ефективно визначати положення та розміри об'єктів, що є важливим етапом для подальшого аналізу та розпізнавання зображень.

Алгоритм обходу зображення для пошуку межових прямокутників з використанням обходу в ширину є методом комп'ютерного зору, призначеним для виділення межових прямокутників на зображенні. Цей алгоритм використовує стратегію обходу в ширину для виявлення контурів та виділення областей, які оточують об'єкти на зображенні [17]. Алгоритм складається з таких етапів:

1. Ініціалізація:
    - Встановлення вихідного пікселя для обходу.
    - Позначення його як відвіданий.
2. Обхід в ширину:
    - Здійснення обходу в ширину, розпочинаючи з вихідного пікселя.
    - Відзначення відвіданих пікселів та виявлення сусідніх пікселів, які ще не були відвідані.
3. Виявлення межових прямокутників:
    - За кожного кроку обходу зберігається інформація про координати верхнього лівого та нижнього правого кутів кожного межового прямокутника.
    - Заключення кожного прямокутника у мінімальний межовий прямокутник.
4. Аналіз результатів:

- Оцінка та аналіз результатів алгоритму для визначення розташування та розмірів межових прямокутників.

Для коректної роботи алгоритму, необхідно налаштувати мінімальний розмір для межових прямокутників та обробляти можливі розриви цифр, що були допущені при їх написанні (особливості почерку людини та/або знаряддя написання). Після отримання списку прямокутників можна приступати до вилучення пікселів для кожного з них та переходити до нормалізації отриманих зображень для нейронної мережі.

2.4 Підготовка набору даних

Як було зазначено вище, підготовка набору даних для навчання нейронної мережі – дуже важливий етап, адже саме від нього залежить наскільки різноманітні дані зможе розпізнавати нейронна мережа. Оскільки було прийнято рішення не використовувати набір MNIST, було прийнято рішення створювати його самостійно. Задля різноманітності даних було залучено близько 20 сторонніх осіб, що записували цифри від 0 до 9 від руки на різних поверхнях, освітленні та різними знаряддями для написання. Загалом було отримано близько двох тисяч унікальних цифр (по 200 цифр), що не схожі між собою.

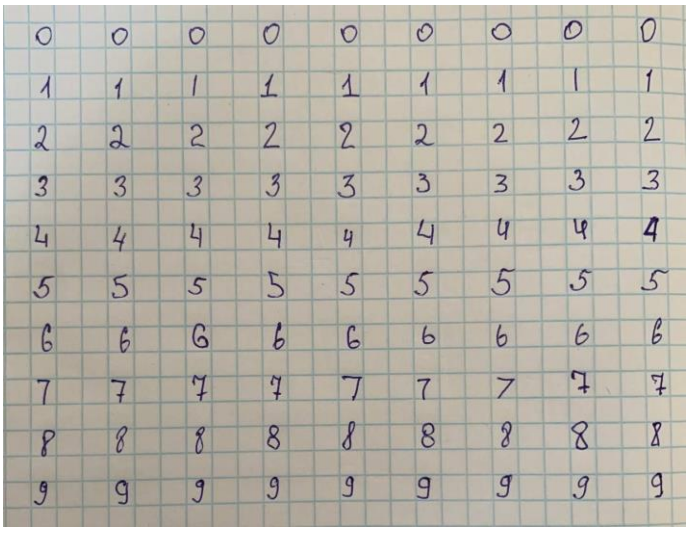


Рис. 2.11 Приклад даних, на який буде створено набір даних для навчання нейронної мережі

Після створення алгоритмів обробки зображень та пошуку межових прямокутників, ці дані будуть перетворені в готовий набір даних, на яких і буде тренувати нейронна мережа. В цьому і полягає головна перевага того, щоб не використовувати набір MNIST – нейронна мережа навчатиметься на реальних даних, що можуть потрапити до неї при використанні. Так, навіть при наявності шуму на зображенні вона матиме досвід таких випадків та зможе, ігноруючи шум, виділити головні ознаки, що ідентифікують ту чи іншу цифру.

2.5 Згорткові нейронні мережі

Нейронні мережі, є класом алгоритмів машинного навчання, що імітують структуру та функції біологічних нейронних мереж у мозку живих організмів. Ці моделі дозволяють автоматизоване вивчення та узагальнення складних залежностей у вхідних даних. Основними будівельними блоками нейронної мережі є штучні нейрони, що взаємодіють між собою через ваги та активаційні функції.

Основною характеристикою нейронної мережі є її архітектура, яка визначається кількістю шарів та кількістю нейронів у кожному з них. Першим шаром є вхідний, останнім — вихідний, а між ними можуть бути приховані шари. Кожен нейрон взаємодіє з нейронами попереднього та наступного шарів через ваги, які піддаються навчанню з метою оптимізації функції втрат.

Основним завданням нейронної мережі є мінімізація функції втрат, яка визначає різницю між передбачуваними та фактичними виходами. Цей процес досягається за допомогою алгоритмів оптимізації, таких як стохастичний градієнтний спуск, які підлаштовують ваги мережі для зменшення помилок прогнозування.

Сигнал передається через мережу від вхідного до вихідного шару, проходячи через кожен нейрон. Для введення нелінійності та здатності моделі вивчати складні залежності, застосовують функції активації. Популярні варіанти включають сигмоїдальні, гіперболічні та ReLU (Rectified Linear Unit) функції активації.

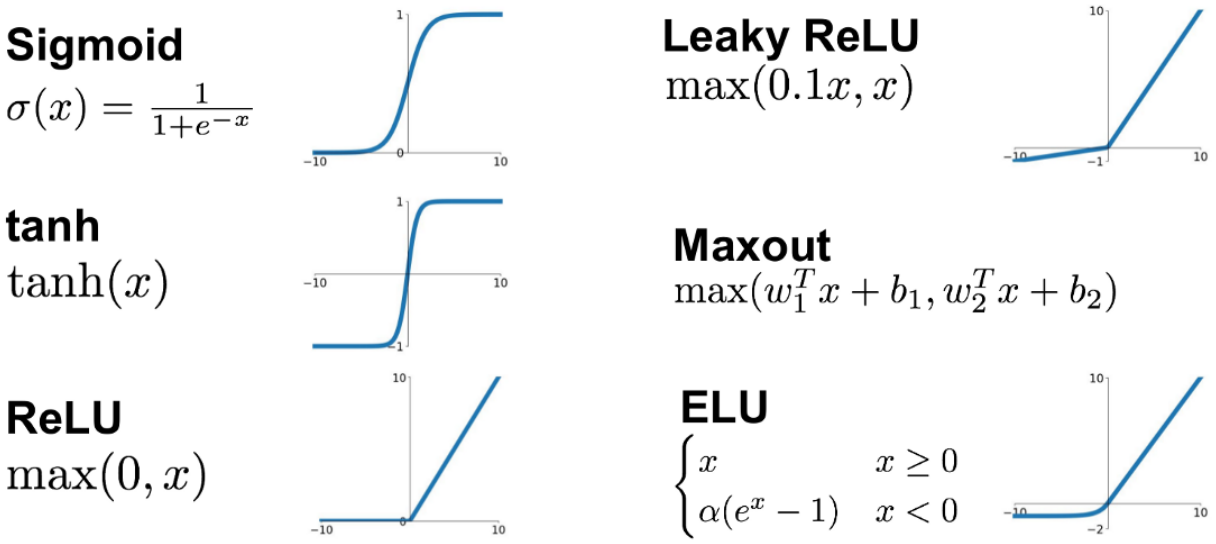


Рис. 2.12 Популярні функції активації та їх графіки

Після аналізу видів нейронних мереж, що можуть бути використані для реалізації заданої задачі постав вибір між двома видами нейронних мереж, а саме:

- повністю зв’язана (щільною) нейронна мережа (Fully Connected Neural Network або Dense Neural Network)
- згорткова нейронна мережею (Convolution Neural Network, CNN).

Щільна нейронна мережа складається щонайменше з трьох шарів, в кожному з яких всі нейрони пов’язані з усіма нейронами попереднього та наступного рівня. Звідси і походить їх назва. Щільні нейронні мережі використовуються в різноманітних завданнях, включаючи класифікацію, регресію та розпізнавання об'єктів. Їхній широкий спектр застосувань пояснюється їхньою здатністю моделювати складні залежності в даних та ефективністю навчання на великих обсягах даних.

Проте, провівши випробування виявилося, що щільні нейронні мережі не дуже добре підходять для розпізнавання об’єктів з зображень. Хоча вони можуть застосовуватися в таких цілях, вони не є найефективнішим засобом для таких задач з декількох причин:

1. зі збільшенням вхідного зображення їх розмірність швидко зростає, що призводить до зменшення продуктивності мережі та збільшення необхідних

обчислювальних ресурсів (для зображення 28*28 пікселів кількість вхідних нейронів становить 784, для зображення 64*64 пікселя – 4096);

2. вони ігнорують просторові властивості зображень, такі як просторові локальність та інваріантність до трансформацій (зміна положення або кута нахилу об’єкта на зображенні сильно впливає на кінцевий результат);
3. щільні нейронні мережі можуть втрачати просторову інформацію зображення через агрегацію ваг (кожен нейрон отримує інформацію одразу з усіх пікселів, що негативно впливає на точність мережі).

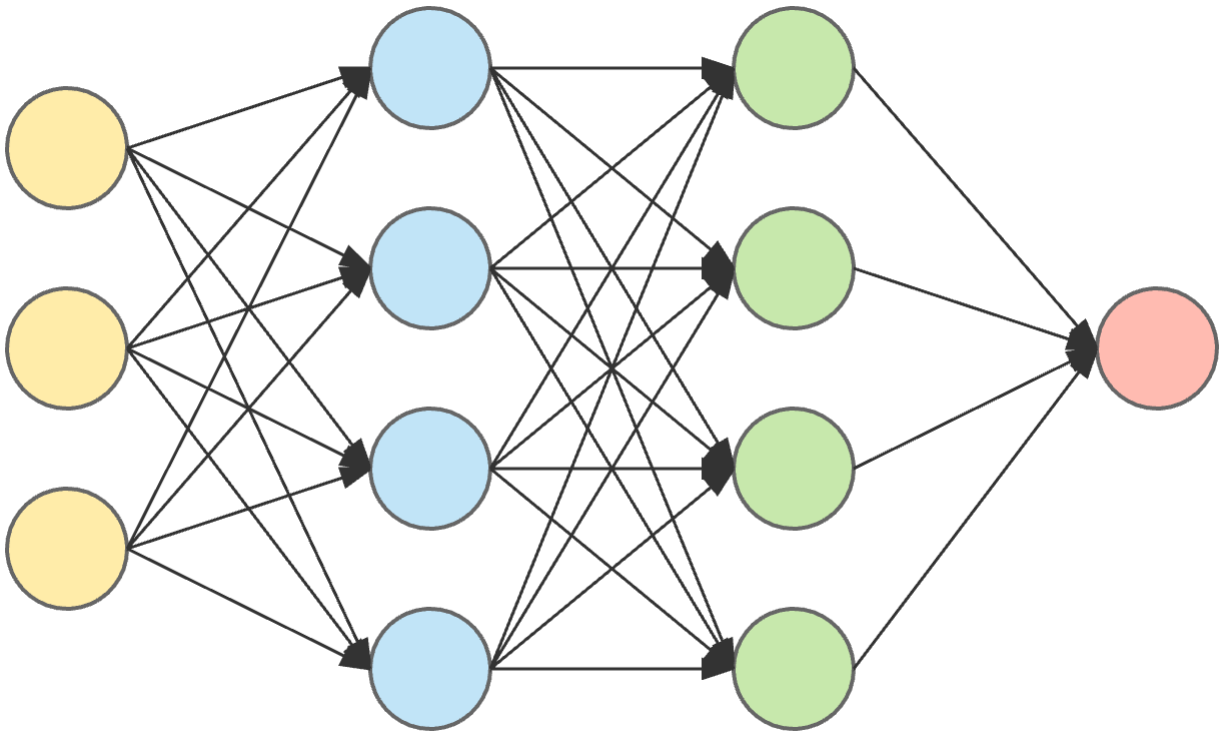


Рис. 2.13 Приклад структури щільної нейронної мережі

Тому, було обрано застосовувати згорткову нейронну мережу. Згорткова нейронна мережа - це клас нейронних мереж, який виник для обробки та класифікації структурованих даних, таких як зображення. Вона використовує спеціальні шари, відомі як згорткові шари, для ефективного виявлення локальних особливостей та шаблонів у вхідних даних [18].

Зазвичай, згорткові нейронні мережі мають три види шарів:

1. Згорткові (convolutional), що виконують операції згортки вхідного зображення.
2. Об’єднуючі (pooling), що виконують операцію об’єднання сусідніх пікселів, вони можуть виконувати операції максимального, середнього або мінімального об’єднання;
3. Повністю зв’язані (dense) шари, що генерують результат на основі даних з попередніх шарів.

Зображення у таку нейронну мережу подається як набір матриць, що містять значення яскравості кожного пікселя у трьох кольорових каналах (червоний, зелений та блакитний, RGB) у вигляді чисел з рухомою комою.

Для проведення операції згортки, такі нейронні мережі використовують фільтри (ядра), що являють собою матриці значень з рухомою комою вигляду:

$$\begin{bmatrix} -1 & 0.5 & 4.2 \\ -2.1 & 0.3 & 3.1 \\ -4.2 & -0.1 & 0.8 \end{bmatrix}$$

Ця операція дозволяє виділити такі ознаки, як границі, текстури та структури. Кількість та розмірність фільтрів задається при створенні нейронної мережі. Згортка виконується кожним каналом для кожного фільтру, що в результаті дає набір матриць, кількість яких дорівнює кількості фільтрів, розмірністю

$$\mathrm{N} - \mathrm{M} + 1$$

де *N* – довжина сторони вхідної матриці;

M – довжина сторони фільтру згорткового шару.

Операція згортки визначається наступним чином для двовимірної матриці (зображення) *I* та ядра *K* [19]:

$$S(i,j) = (I * K)(i,j) = \Sigma_m \Sigma_n I(m,n) K(i-m, j-n)$$

де *S(i, j)* - значення пікселя результуючої матриці;

I(m, n) - значення пікселя вхідного зображення;

K(i−m, j−n) - значення ядра.

Також, після операції згортки застосовується операція додавання зсуву (bias), що є важливим компонентом для забезпечення гнучкості та адаптивності мережі до

різних видів даних. Після виконання всіх вищезазначених дій, застосовується функція активації нейронів, що для згорткового шару зазвичай є ReLU.

Приклад результату, отриманого після виконання операції згортки зазначено на рисунку 2.5.4.

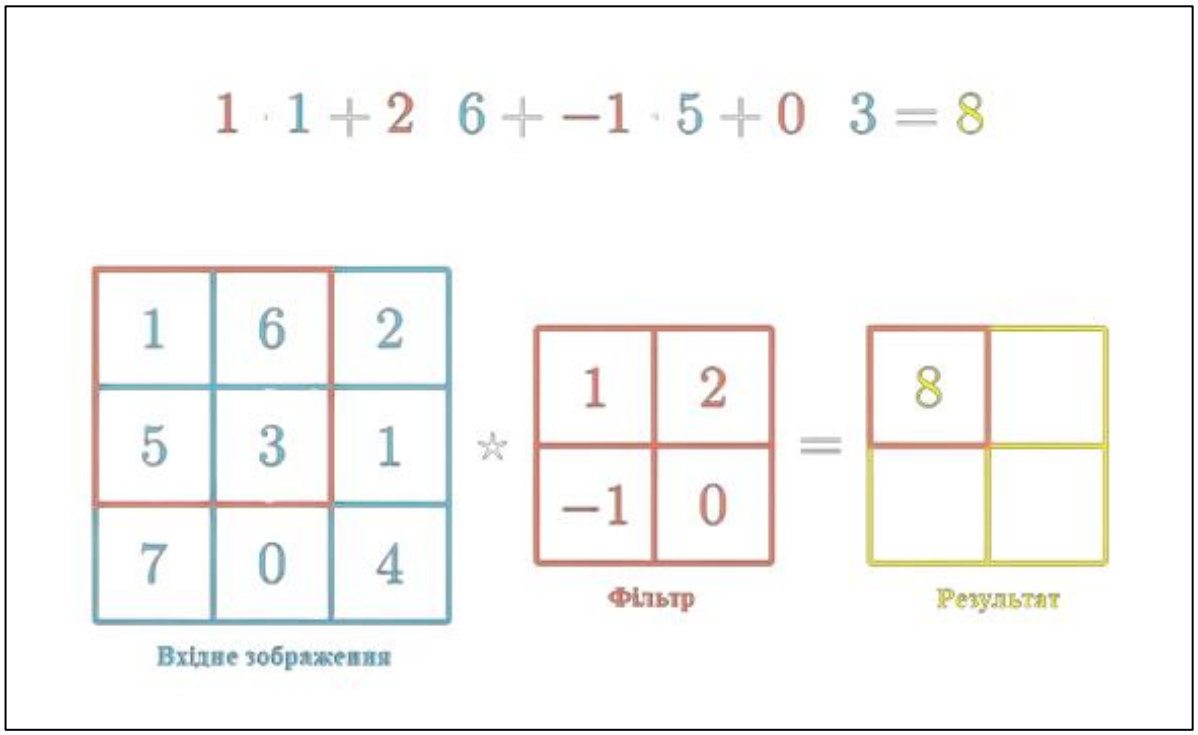


Рис. 2.14 Операція згортки

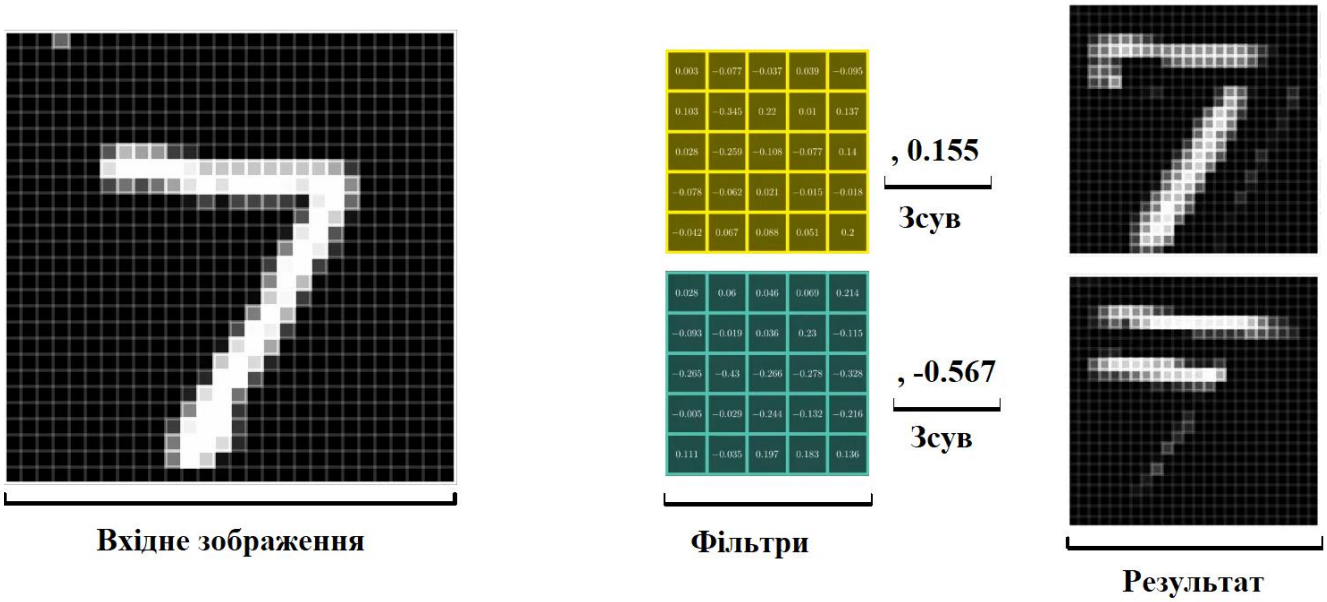


Рис. 2.15 Результат виконання згортки вхідного зображення

Після виконання згортки виконується операція об’єднання значень для кожної матриці (карт ознак) з попереднього шару. Вона використовуються для зменшення просторового розміру карт ознак, зберігаючи при цьому важливі інформаційні характеристики, що в свою чергу сприяє поліпшенню обчислювальної ефективності мережі. Найпоширенішими операціями об’єднання є максимальне та середнє, що використовують неперекриваючі області для взяття пікселів з регіонів зображення. Для створення нейронної мережі для розпізнавання рукописних чисел з чорно-білими зображеннями краще підходить операція максимального об’єднання, що визначається наступним чином:

$$P(i,j) = max_{m,n}\big(I(i * s + m, j * s + n)\big)$$

де *P(i, j)* – значення пікселя у результуючій матриці після максимального об’єднання;

*I(i*s+m, j*s+n)* – значення пікселя вхідної матриці;

s – крок об’єднання (stride);

m та *n* – розмірність вікна об’єднання.

Параметри *s*, *m* та *n* задаються при створенні нейронної мережі. Операції об’єднуючого шару не потребують функцій активації, зсуву та навчання.

Операції згортки та об’єднання застосовуються послідовно в довільній кількості, тому іноді можна зустріти їх поєднання як один шар нейронної мережі (CP шар).

Після виконання операцій згортки та об’єднання набір карт ознак переходить для генерації результату на шар повністю зв’язаних нейронів, що має розмірність

$$N * M * K$$

де *N* – кількість карт ознак;

M та *K* – їх розмірності.

Оскільки нейрони цього шару не приймають дані у вигляді масивів, перед передачею даних до набору застосовується операція розгортки в одновимірний масив (вектор). Ця операція часто використовується у контексті обробки даних у машинному навчанні або обчисленнях, де потрібно перетворити багатовимірні структури даних у вектор для подальших операцій.

Оскільки щільний шар нейронної мережі є вихідним, кількість вихідних нейронів визначається кількістю можливих відповідей – 10. Визначена цифра позначатиметься індексом найбільшого елементу масиву. Для кожного вихідного нейрону значення вагової суми визначається за формулою [20]:

$$z_i = \sum_j w_{ij}^{(L)} \cdot a_j^{(L-1)} + b_i^{(L)},$$

де L вказує на останній (вихідний) шар мережі;

$w_{ij}^{(L)}$- вага між j-м нейроном у попередньому шарі та i-м нейроном у вихідному шарі;

$a_j^{(L-1)}$ - вихід j-го нейрона у попередньому шарі;

$b_i^{(L)}$- зсув для і-го нейрона у вихідному шарі.

Найкращою функцією активації для задач класифікації є softmax, або ж нормована експоненційна функція [21], що в даному контексті має формулу:

$$y_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

де y_i – вихід i-го нейрона після застосування функції;

e – число Ейлера;

z_i – вагова сума для i-го нейрона;

k – кількість класів, для яких визначаються ймовірності в рамках функції softmax.

Також, в рамках створення нейронної мережі можуть знадобитися функції нормалізації значень нейронів та їх ваг. Вона полягає в приведенні значень нейронів в діапазон від -1 до 1 або від 0 до 1. Це необхідно не лише для зменшення складності обрахунків, але й для уникнення «розгону» (англ. exploding), коли ваги або значення нейронів в процесі навчання стають дуже великими та не можуть помістится у виділену комп’ютером пам’ять. Це може призвести до нестабільності мережі та її недієздатності.

Перед використанням нейронної мережі її необхідно буде навчити на наявному наборі даних. Існують декілька видів навчання нейронних мереж:

1. Навчання з учителем (Supervised Learning):

- У цьому виді навчання для кожного вхідного прикладу надається пара "вхід-вихід" (input-output), і мережа навчається знаходити відповідність між входом і вихідним значенням.
- Головна мета - мінімізація різниці між передбаченими вихідними значеннями і справжніми вихідними значеннями.
- Приклади: класифікація, регресія.

2. Навчання без учителя (Unsupervised Learning):
   - У цьому виді навчання модель отримує тільки вхідні дані і намагається виявити приховані структури чи шаблони в цих даних.
   - Головна мета - групування, розкриття прихованих залежностей або побудова представлення даних.
   - Приклади: кластеризація, автокодування.
3. Навчання з частковим навчанням (Semi-supervised Learning):
   - Комбінує елементи навчання з учителем та навчання без учителя.
   - Має невелику кількість прикладів з відомими вихідними значеннями і багато прикладів без них.
   - Мета - використовувати велику кількість невідомих даних для поліпшення навчання.
   - Приклади: класифікація з обмеженим набором міток.
4. Навчання з підкріпленням (Reinforcement Learning):
   - Модель (агент) взаємодіє з оточенням і отримує винагороду чи покарання в залежності від своїх дій.
   - Мета - вивчення оптимальної стратегії для максимізації накопиченої винагороди в часі.
   - Використовується в задачах прийняття рішень та управління.

Оскільки розпізнавання рукописних чисел є задачею класифікації, найкращим видом навчання буде навчання з учителем. Так, при передачі зображення при навчанні мережі, необхідно буде передавати очікуваний результат від нейронної мережі.

Для реалізації такого навчання необхідно реалізувати метод зворотного поширення помилки (backpropagation) для мінімізації функції втрат шляхом оновлення ваг та зсувів у нейронній мережі.

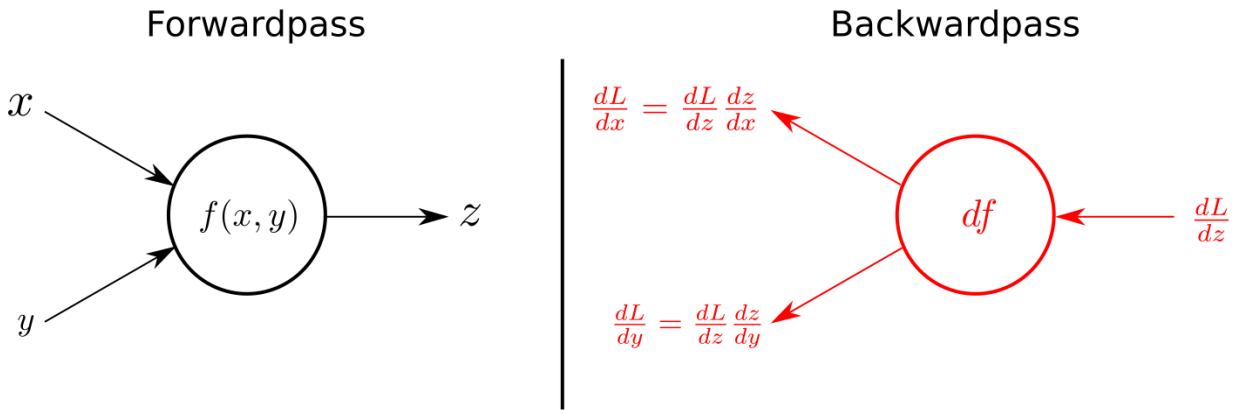


Рис. 2.16 Прямий прохід та зворотнє поширення помилки

Процес backpropagation відбувається у кілька етапів [22]:

1. Прямий прохід (Forward Pass):
   - Вхідні дані подаються в мережу, і вона виконує прямий прохід, обчислюючи вихідні значення для кожного шару від входу до виходу.
2. Обчислення функції втрат:
   - Порівнюються передбачені вихідні значення з справжніми, і обчислюється функція втрат, яка відображає різницю між ними.
3. Зворотнє поширення помилки (Backward Pass):
   - Розпочинаючи з останнього шару, обчислюється градієнт функції втрат відносно ваг та зсувів.
   - Градієнт передається в зворотного напрямку через мережу, використовуючи правило ланцюга (chain rule).
4. Оновлення параметрів:
   - Оновлення ваг та зсувів відбувається в напрямку, протилежному градієнту, з використанням методу градієнтного спуску.

- Коефіцієнт швидкості навчання визначає, наскільки швидко оновлюються параметри.

Цей ітеративний процес повторюється протягом кількох епох або до досягнення задовільної збіжності. Метод зворотного поширення помилки дозволяє мережі автоматично коригувати свої параметри, адаптуючись до особливостей вхідних даних і покращуючи точність передбачень.

2.6 Загальний вигляд додатку

Кінцевий додаток матиме клієнт-серверну архітектуру, а для взаємодії буде використовуватися протокол передачі даних HTTP (HyperText Transfer Protocol), що є звичним засобом для таких систем. Взаємодія між клієнтом та сервером буде реалізована за допомогою архітектурного стилю REST (Representational State Transfer). Він надає простий та ефективний спосіб комунікації між різними компонентами системи шляхом використання стандартних HTTP-методів, таких як GET, POST, PUT та DELETE, для виконання операцій з ресурсами на сервері. За допомогою такої архітектури кожен ресурс ідентифікується унікальним URI (Uniform Resource Identifier), а сприяння стандартам HTTP дозволяє легко взаємодіяти з цими ресурсами. Запити від клієнта та відповіді від сервера обмінюватимуться у форматі, JSON (JavaScript Object Notation). Цей підхід дозволяє досягти прозорості, масштабованості та легкості розробки та розгортання системи, забезпечуючи при цьому ефективну інтеграцію між різними компонентами.

Серверна частина додатку матиме монолітну структуру, що означає , що всі компоненти та функції, пов'язані із сервером, будуть об'єднані в один компактний блок програмного коду. Такий підхід дозволяє спростити розробку, випробування та розгортання додатку, а також забезпечити ефективну комунікацію між різними частинами системи.

Вона включатиме в себе такі структурні компоненти, як нейронну мережу, алгоритми навчання нейронної мережі, набір даних для навчання нейронної

мережі, алгоритми обробки зображень, стан навченої нейронної мережі, сервіси, що виконуватимуть логіку додатку, та HTTP WebAPI сервер (сервер, який служить для обробки HTTP-запитів та надає програмний інтерфейс для взаємодії з іншими компонентами чи додатками через веб).

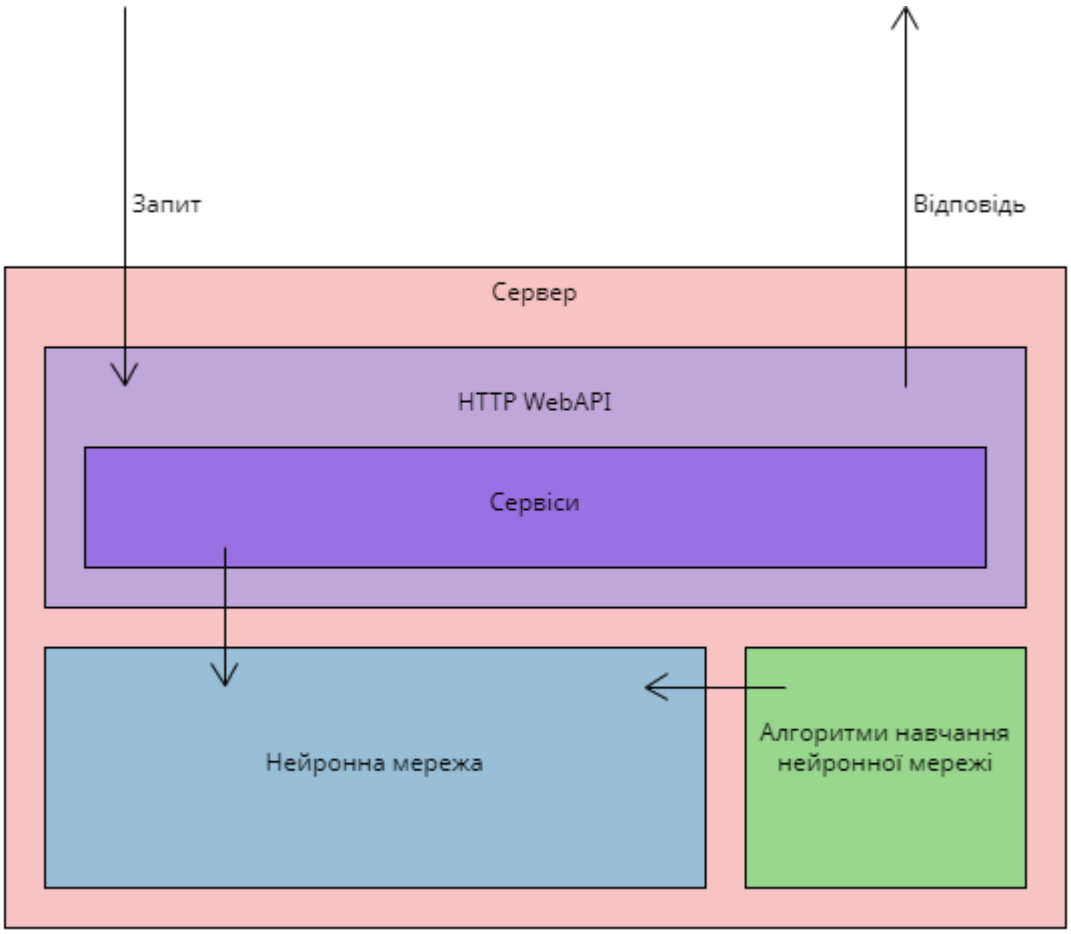


Рис. 2.17 Загальна структура серверної частини кінцевого додатку

Фіксація стану натренованої нейронної мережі відбуватиметься за допомогою процесу серіалізації (перетворення ключових властивостей мережі у текст, в даному випадку у формат JSON), а її відновлення – за допомогою зворотного процесу – десеріалізації. Для збереження цього стану використовуватиметься текстовий файл розширення .json. У даному контексті використання бази даних є недоцільним, адже окрім того, що є лише один запис для збереження, бази даних неефективно працюють з великими рядками тексту.

Мовою програмування для серверної частини кінцевого додатку було обрано C#. Цей вибір був обґрунтований кількома важливими причинами з точки зору ефективності, екосистеми та високого рівня підтримки. Це стратегічне рішення, спрямоване на забезпечення ефективної та продуктивної розробки. До переваг цієї мови програмування можна віднести:

1. Високу продуктивність та швидкість виконання;
2. Наявність широкого вибору бібліотек та інструментів розробки, велика спільнота розробників;
3. Можливість ефективного використання парадигм об'єктно-орієнтованого та функціонального програмування;

Ці фактори роблять розробку нейронних мереж та їх інтеграцію з серверною частиною додатку більш гнучкою та доступною.

Що стосується клієнтської частини додатку, вона буде створена з використанням фреймворку Angular, що розробляється підрозділом компанії Google. Мовою програмування для додатків цього фреймворку є TypeScript, що в свою чергу була представлена компанією Microsoft восени 2012 року, та є розширеною версією JavaScript. Цікавим фактом є те, що розробником обох мов програмування, що будуть використані в цій роботі (C# та TypeScript) є одна людина – Андерс Гейлсберг, видатний данський програміст, що став відомим завдяки своєму вкладу в ці та деякі інші мови програмування.

До науково обґрунтованих переваг TypeScript над JavaScript можна включити те, що він:

1. Надає можливість використовувати статичну типізацію, що дозволяє визначати типи змінних, параметрів та інших елементів коду на етапі розробки;
2. Пропонує більшу підтримку об'єктно-орієнтованого програмування (ООП) за рахунок введення інтерфейсів, абстракцій та інших концепцій, що полегшує розробку та підтримку складних систем;

3. Підтримує модульну структуру коду, що дозволяє зберігати код у вигляді логічних модулів. Це полегшує організацію та управління великими проєктами;
4. Включає сучасні функції та можливості, які визначені в стандарті ECMAScript, що дозволяє використовувати сучасний синтаксис та функціональність JavaScript.

Клієнтська частина додатку включатиме в себе форми завантаження зображень з рукописним текстом та виведення результатів їх аналізу. Після завантаження зображення, воно буде надіслане для аналізу до серверу, що поверне результат розпізнавання.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Підготовка до розробки

Перед початком розробки необхідно інсталювати всі потрібні компоненти на пристрій. Для серверної частини, це середовище .NET версії 8.0, що згідно з офіційним сайтом компанії Microsoft, є найактуальнішою стабільною версією .NET [23]. Для цього необхідно завантажити відповідний файл з сайту https://dotnet.microsoft.com/en-us/download/dotnet та встановити вміст.

Для клієнтської частини додатку необхідно становити два компоненти: Node.js версії 18.16.0 та Angular CLI версії 17.0.3. Node.js – це безкоштовне міжплатформенне середовище виконання JavaScript додатків з відкритим вихідним кодом. Завантажити його необхідно з офіційного сайту – https://nodejs.org/en.

Angular CLI - це інструмент інтерфейсу командного рядка, який ви використовуєте для ініціалізації, розробки та підтримки програм Angular безпосередньо з командної оболонки [24]. Для його встановлення необхідно виконати команду «npm install -g @angular/cli» пакетного менеджеру npm, що встановлюється за замовчуванням разом з Node.js у командному рядку Windows або будь якому іншому застосунку, що надає такі можливості. Позначка -g означає, що пакет встановлюється глобально на пристрій, що робить його доступним до використання в будь-якому проєкті.

```
PS C:\> npm install -g @angular/cli

added 18 packages, removed 35 packages, and changed 210 packages in 10s

45 packages are looking for funding
  run `npm fund` for details
```

Рис. 3.18 Встановлення Angular CLI

Після цього можна перейти до вибору на налаштувань IDE (Integrated Development Environment) - інтегрованих середовищ розробки для серверної та

клієнтської частин додатку. IDE – комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм.

Для серверної частини було обрано Visual Studio, оскільки це середовище має всі необхідні інструменти, що можуть знадобитися при розробці будь яких додатків і при цьому має зручний інтерфейс користувача. Для клієнтської частини було обрано Visual Studio Code, що також має приємний інтерфейс з можливістю встановлення розширень для покращення роботи з окремими видами програм та більше спеціалізується на роботі з JavaScript кодом. Обидві програми мають одного розробника – компанія Microsoft.

Також, у Visual Studio Code буде встановлено розширення Prettier, що допомагає форматувати код натисканням кількох клавіш і тримати його більш читабельним, та Angular Language Service, що допомагатиме не допускати помилок при написанні коду в процесі розробки клієнтської частини додатку.

3.2 Створення клієнтської частини додатку

Для початку розробки, створимо папку DigitRecognition, що міститиме в собі обидві частини додатку. Після цього необхідно відкрити Visual Studio Code та перейти до створеної папки. Натиснувши команду «Ctrl+Shift+`» внизу екрану з’явиться вікно терміналу, за допомогою якого буде створено та запущено клієнтську частину додатку.

Ввівши в терміналі команду «ng new digit-recognition-front» необхідно обрати тип файлів стилів, що буде використано при розробці. Для даного випадку було обрано використовувати SCSS. Після успішного створення проєкту, необхідно заново відкрити IDE, але папку digit-recognition-front.

У новоствореному додатку необхідно очистити вміст файлу app.component.html, адже він містить лише посилання на офіційні сторінки розробників фреймворку. Тут пізніше буде додане посилання на компонент, що

міститиме всі необхідні інструменти для завантаження файлу на сервер для розпізнавання рукописних чисел.

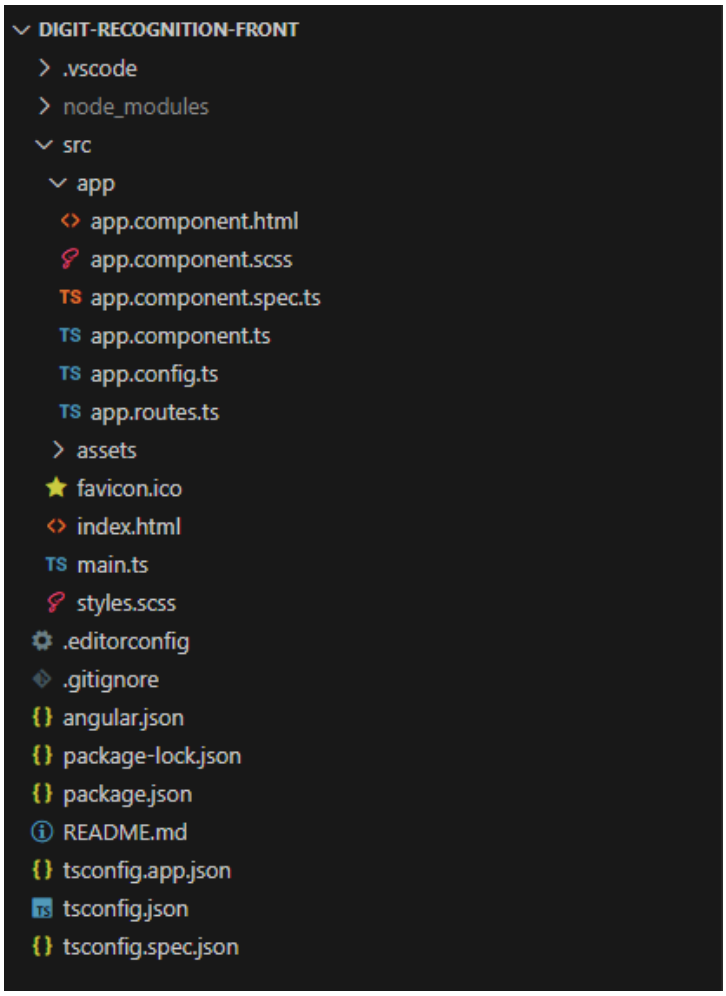


Рис. 3.19 Автоматично згенерована структура додатку Angular

Тепер необхідно встановити ще один пакет, що полегшить розробку приємного інтерфейсу та додасть нові можливості додатку – @angular/material. Це набір UI (User Interface) компонентів та інструментів від розробників Angular.

Перейдемо до реалізації завантаження файлів на сервер. Для цього у папці src/app необхідно створити папку home та додати до неї три файли:

1. home.component.ts – файл, що містить TypeScript код, в якому знаходитиметься вся логіка компоненту;
2. home.component.scss – файл, що містить стилі розмітки, який допоможе оформити зовнішній вигляд компоненту;

3. home.component.html – файл, що містить розмітку веб сторінки.

У файлі розширення .ts оголосимо компонент HomeComponent з селектором app-home. За допомогою цього селектору, даний компонент можна буде використати в будь-якій частині додатку. Приємним нововведенням, що з’явилося в Angular версії 14 є автономні компоненти, що не потребують класу NgModule, що допомагає зменшити кількість коду, складність програми та пришвидшити її роботу.

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss'],
  standalone: true,
  imports: [
    NgIf,
    MatFormFieldModule,
    MatInputModule,
    FormsModule,
    MatButtonModule,
  ],
})
export class HomeComponent { }
```

Рис. 3.20 Оголошення автономного компоненту HomeComponent

Посилання у полі imports надалі будуть використані при написанні вмісту файлу HTML.

Також створимо тип даних, що відображатиме актуальний стан програми. У TypeScript це можна реалізувати як набір унікальних рядків (Draw, Upload, Analize) та пусте значення, як початковий стан. Стан Draw необхідний для реалізації панелі

для малювання, що може використовуватися на початкових етапах для тестування продукту без завантаження реальних фото.

У файлі home.component.html необхідно створити відповідну розмітку та підлаштовувати вміст сторінки за допомогою структурної директиви ngIf за такою логікою:

1. Для початку доступні дві кнопки: «Малювати» та «Завантажити файл»;
2. При натисканні кнопки «Малювати» змінюється стан додатку, з’являється панель для малювання та зникає кнопка «Завантажити файл», але з’являються кнопки «Аналізувати», що відправляє вміст панелі для аналізу на сервер, «Очистити панель», що скидає весь вміст панелі та «Скинути», що повертає додаток у початковий стан;
3. При натисканні кнопки «Завантажити файл» відкривається вікно вибору файлу зображення, а після його завантаження одразу викликається метод, що відправляє його для аналізу на сервер;
4. Під час аналізу зображення на екрані з’являється напис, що інформує користувача про активний процес;
5. Після виконання аналізу, його результати виводяться на сторінку та стан додатку змінюється на початковий.

Після реалізації заданого коду необхідно створити відповідні методи в home.component.ts, а саме методи clearCanvas, reset, analizeImage, uploadImage та інші. Для реалізації малювання на панелі у стандартному методі ngAfterViewInit необхідно виконати відповідні підписки на події елементу Canvas.

Для того, щоб відправляти зображення для аналізу на сервер, необхідно створити сервіс, що матиме об’єкт типу HttpClient, що входить до стандартного пакету Angular. Для цього у папці src/app створимо файл AppService, позначимо його декоратором @Injectable, що дозволить отримувати екземпляр цього кладу в будь якій частині додатку та задамо як параметр для нього { providedIn: 'root' }.

У цьому сервісі необхідно створити метод analizeImage, що приймає аргумент типу Blob (формат, що використовується у JavaScript для зберігання файлової інформації) та повертає об’єкт типу Observable<AnalizeResult> (потік, що

після отримання даних з серверу поверне об’єкт AnalizeResult). Тип AnalizeResult необхідно створити у папці src/app/metadata і він повинен містити для поля: «value» типу string, що являтиме собою текст з розпізнаними числами та «image», що є тектовою інтерпретацією послідовності бітів файлу, що був завантажений для аналізу з позначеними об’єктами.

Після цього також варто створити компонент спливаючого вікна, що буде викликатися через AppService у разі, якщо виникатиме якась помилка у процесі використання програми. Для цього створимо новий компонент у src/app/dialog з назвою app-dialog. При отримані помилки від серверу буде викликатися метод showMessage класу AppService, що за допомогою MatDialog (один з класів Angular Material) відкриватиме діалогове вікно з описом помилки.

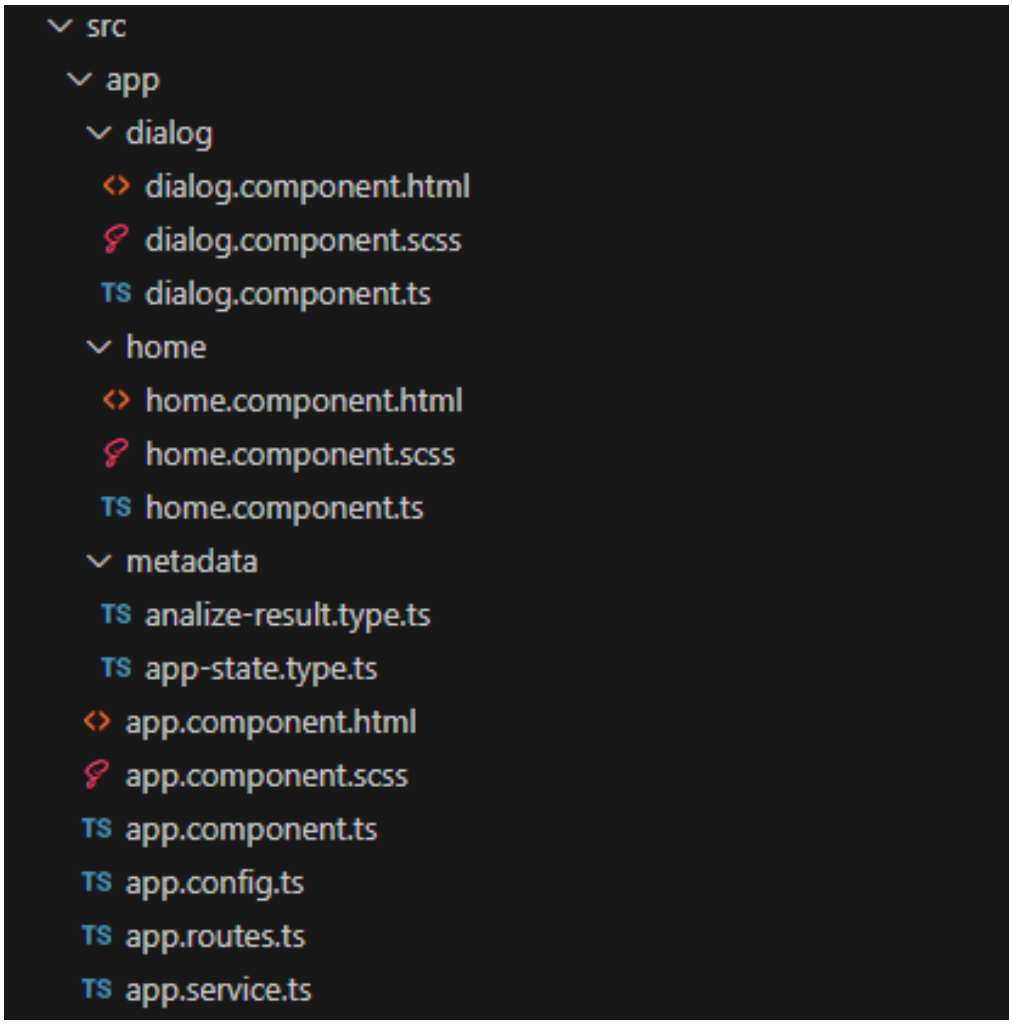


Рис. 3.21 Кінцева структура папки src

3.3 Реалізація алгоритму попередньої обробки зображень

Оскільки попередня обробка зображень та їх аналіз відбуватиметься на серверній частині кінцевого додатку, необхідно створити новий проєкт. Для цього потрібно за допомогою Visual Studio створити новий проєкт типу ASP.NET Core Web API, вказавши як місцезнаходження папку DigitRecognition. Назва проєтку – API, назва рішення – DigitRecognitionBackend. При виборі платформи обираємо .NET 8.0.

Прибравши тестовий код, згенерований при створенні проєкту, з файлу Program.cs (sampleTodos), можна переходити до налаштування серверу. В першу чергу необхідно додати папку папку Services до проєкту API, що міститиме всі файли з логікою додатку. У цій папці створюємо ще дві – Interfaces та Implementations. У папці Interfaces міститимуться інтерфейси всіх сервісів додатку, а в папці Implementations їх реалізації. Таке розділення необхідне для покращення якості дизайну шляхом зменшення з’язності коду. Так, у разі створення нові, покращенні варіації алгоритмів та за допомогою процесу Dependency Injection підставити нову реалізацію в додаток не видаляючи стару. Також це дозволяє легше тестувати додатки.

Першим файлом у Interfaces необхідно створити IDigitRecognitionService.cs – інтерфейс сервісу, що оброблятиме запити від клієнта. Він повинен мати один метод AnalizeImage, що приймає аргумент типу IFormFile, що втілює собою файл, відправлений за допомогою HTTP запиту на сервер. Метод повертатиме об’єкт типу AnalizeResponseDto, що містить дві властивості: «Value» типу string та «Image» типу byte[] (масив бітів). Цей тип необхідно помістити в папку API/Common/DTO. DTO (Data Transfer Object) – один із шаблонів проєктування, який використовують для передачі даних між підсистемами програми.

Після створення інтерфейсу необхідно перейти до створення його реалізації. У папці Services/Implementations необхідно створити файл DigitRecognitionService.cs, в якому оголосити клас, що реалізує інтерфейс IDigitRecognitionService.

На цьому етапі варто зареєструвати сервіс у файлі Program.cs, щоб у разі використання інтерфейсу в програмі, модулі програми отримували посилання на об’єкт сервісу. Цей процес показаний на рисунку 3.3.1. Є три види реєстрації сервісів:

1. Transient – створює новий об’єкт при кожному випадку його використання в програмі;
2. Scoped – створює новий об’єкт кожного запиту, але має однакові посилання на об’єкти одного типу всередині запиту;
3. Singleton – створює один об’єкт при першому використанні та не змінює його до закриття програми.

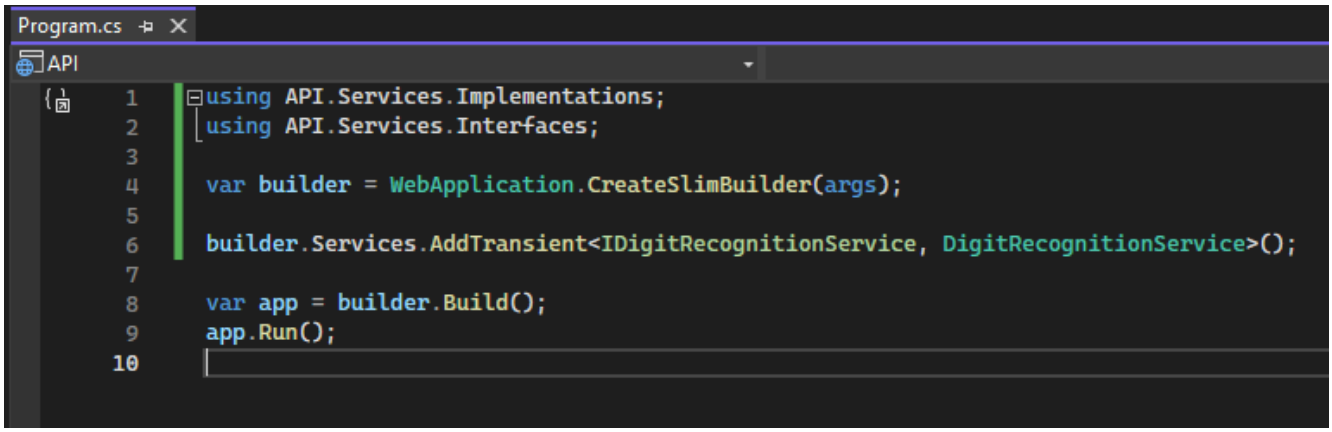


Рис. 3.22 Реєстрація IDigitRecognitionService

Після цього можна перейти до створення контролеру для перехоплення HTTP запитів від клієнта. Для цього у проєкт API необхідно додати папку Controllers та помістити туди файл DigitRecognitionController.cs. В ньому буде знаходитись клас, що наслідуватиметься від стандартного класу ControllerBase, матиме атрибути ApiController та Route (зі значенням “ [controller] ”), що дозволить обробляти HTTP запити. За допомогою Dependecy Injection додамо посилання на IDigitRecognitionService та реалізуємо метод AnalizeImage, що приймає аргумент типу ValueDto<IFormFile> від клієнту. ValueDto – ще один тип, що знаходиться в папці Common/DTO. Також було створено кілька допоміжних методів, що

допомагають перехоплювати помилки і відправляти зрозумілу відповідь користувачеві.

Після створення контролеру необхідно його зареєструвати у файлі Program.cs та дозволити CORS (Cross-Origin Resource Sharing), щоб сервіс став доступним для використання клієнтською частиною.

Надалі у файлі properties/launchSettings.json необхідно встановити поле applicationUrl, що визначає на якій адресі буде запущено сервер, в даному випадку – http://localhost:7258. Ці дані тепер можна перенести до клієнтської частини, щоб запити надсилалися на правильну адресу.

```
[ApiController]
[Route("[controller]")]
1 reference
public class DigitRecognitionController : ControllerBase
{
    private readonly IDigitRecognitionService _digitRecognitionService;

    0 references
    public DigitRecognitionController(IDigitRecognitionService digitRecognitionService)
    {
        _digitRecognitionService = digitRecognitionService;
    }

    [HttpPost("Analize")]
    0 references
    public IActionResult AnalizeImage([FromForm] ValueDto<IFormFile> dto)
    {
        var response = InterceptErrors(() => _digitRecognitionService.AnalizeImage(dto.Value));

        return ProcessResponse(response);
    }
```

Рис. 3.23 Уривок файлу DigitRecognitionController.cs

Тепер необхідно встановити пакет System.Drawing.Common за допомогою пакетного менеджеру NuGet, підтримка якого є частиною Visual Studio. Цей пакет необхідний для доступу для роботи з зображеннями, адже він містить клас Bitmap, за допомогою якого можна керувати ними з коду.

У папці Services/Interfaces створимо файл IImageProcessor.cs, що міститиме один метод GetNumberCards, що надалі повертатиме колекцію колекцій картинок з зображеннями, де колекція уособлюватиме собою рядок на зображені. Також,

варто одразу додати параметр, який буде використано в майбутньому – resizedSize, що являє собою довжину сторони картинки з цифрою. Після цього можна створити файл ImageProcessor.cs в папці Services/Implementations та зареєструвати ці типи у файлі Program.cs як це було з DigitRecognitionService.cs. На цьому етапі попереднє налаштування проєкту завершене і можна переходити до реалізації алгоритму попередньої обробки зображень.

Для цього переходимо в файл ImageProcessor.cs та починаємо реалізацію методу. Оскільки, ще не реалізована повний аналіз зображень, можна повернути з методу пусту колекцію. Це дозволить запустити програму та перевірити розроблений алгоритм, не реалізовуючи його повністю.

Виділимо попередню обробку зображення в окремий приватний метод з назвою PreprocessInputImage. Як було зазначено в розділі 2.2, попередня обробка зображень складатиметься з трьох етапів, тому необхідно створити метод для кожного з них.

```
public class ImageProcessor : IImageProcessor
{
    3 references
    public IEnumerable<IEnumerable<Bitmap>> GetNumberCards(Bitmap image, int resizedSize)
    {
        var processedImage = PreprocessInputImage(image);

        return Enumerable.Empty<IEnumerable<Bitmap>>();
    }

    1 reference
    private Bitmap PreprocessInputImage(Bitmap image)
    {
        Bitmap proccessedImage = ApplySharpenFilter(image);

        proccessedImage = ApplyGaussianBlur(proccessedImage, 1.9);

        proccessedImage = ApplyBrightnessThreshhold(proccessedImage);

        return proccessedImage;
    }
}
```

Рис. 3.24 Актуальний стан класу ImageProcessor

Перейдемо до реалізації фільтру підвищення різкості зображення. Для цього необхідно визначити матрицю ядра. В процесі підготовки та тестування, було знайдене оптимальне ядро (матриця), що дорівнює:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 11 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Оскільки операція полягає в проходженні всього зображення обраним ядром, для її реалізації знадобиться використання циклів, по ширині та висоті зображення, але з урахуванням межі фільтру. Тому границі значень в циклі обходу зображення будуть зміщені на одиницю від своїх крайніх значень.

Для кожного пікселя зображення (окрім тих, що не входять в область дії циклу), необхідно виконати обхід по ядру та записати нові значення кольорових каналів, що визначаються як сума добутків значень каналів, помножених на позицію пікселя відносно середини ядра (пікселя що обробляється). Після знаходження суми відбувається нормалізація значень, адже значення каналу кольору – це значення від 0 до 255. Результати таких перетворень записуються в новий об’єкт Bitmap під назвою sharpenedImage.

```
for (int x = 1; x < image.Width - 1; x++)
{
    for (int y = 1; y < image.Height - 1; y++)
    {
        float red = 0, green = 0, blue = 0;

        for (int i = 0; i < filterSize; i++)
        {
            for (int j = 0; j < filterSize; j++)
            {
                Color pixel = image.GetPixel(x - 1 + i, y - 1 + j);
                red += pixel.R * coreMatrix[i, j];
                green += pixel.G * coreMatrix[i, j];
                blue += pixel.B * coreMatrix[i, j];
            }
        }

        red = Math.Min(Math.Max(red, 0), 255);
        green = Math.Min(Math.Max(green, 0), 255);
        blue = Math.Min(Math.Max(blue, 0), 255);

        sharpenedImage.SetPixel(x, y, Color.FromArgb((int)red, (int)green, (int)blue));
    }
}
```

Рис. 3.25 Цикл обходу зображення та створення нового зображення з підвищеною різкістю

Наступним методом процесу попередньої обробки зображення є розмивання Гауса. Цей метод потребує коефіцієнт сигма, оптимальним значенням якого в процесі тестування було визначено 1.9. Для його застосування необхідно визначити ядро розмиття, що визначається за формулою у розділі 2.2, та нормалізувати його (привести всі значення в діапазон від 0 до 1). Розмір ядра визначається як сума подвоєного найменшого цілого числа, що більше або рівне сигма, плюс один.

Після визначення ядра розмиття, як і у випадку з операцією підвищення різкості необхідно визначити границі циклу. Відступи у даному випадку дорівнюють найбільшому цілому числу, що менше або рівне половині розміру ядра, назвемо цей параметр border. Для пришвидшення виконання розмиття Гауса, було прийнято рішення використовувати байтові операції при обробці зображення. Цей метод вимагає застосування спеціального синтаксису – блоку unsafe, що в мові програмування C# дозволить напряму оперувати адресами в пам’яті та байтами в них [25].

Значення кольорових каналів у новому зображенні визначається як сума добутків кольорових каналів сусідніх пікселів (визначених ядром розмиття), помножених на значення ядра в цій точці.

```
unsafe
{
    for (int y = border; y < image.Height - border; y++)
    {
        byte* srcRow = (byte*)srcData.Scan0 + (y - border) * srcData.Stride;
        byte* destRow = (byte*)destData.Scan0 + y * destData.Stride;

        for (int x = border; x < image.Width - border; x++)
        {
            double sumR = 0, sumG = 0, sumB = 0;

            for (int i = 0; i < size; i++)
            {
                byte* srcPixel = srcRow + (x - border) * 4 + i * srcData.Stride;
                double weight = kernel[i, 0];

                sumR += srcPixel[2] * weight;
                sumG += srcPixel[1] * weight;
                sumB += srcPixel[0] * weight;
            }

            destRow[x * 4] = (byte)sumB;
            destRow[x * 4 + 1] = (byte)sumG;
            destRow[x * 4 + 2] = (byte)sumR;
            destRow[x * 4 + 3] = 255;
        }
    }
}
```

Рис. 3.26 Виконання розмивання Гауса в блоці unsafe

Застосування порогу яскравості є найлегшою операцією при попередній обробці зображення , адже полягає в переписі значень пікселів. Для цього необхідно за допомогою двох циклів – по ширині та висоті, обійти все зображення і, у випадку якщо яскравість пікселя більше 90%, задати його колір білим, а в іншому випадку – чорним. Отримати значення яскравості можна за допомогою методу GetBrightness структури Color, що застосовується в класі Bitmap як значення пікселя.

На цьому етапі розробка алгоритмів попередньої обробки зображень завершена, але тимчасово необхідно додати збереження оброблених зображень, щоб перевірити правильність роботи написаного коду. Для цього необхідно використати метод Save на готовому зображенні та вказати шлях, куди буде збережено новий файл. Тепер необхідно додати виклик методу GetNumberCards у методі AnalizeImage класу DigitRecognitionService. Але перед цим необхідно виконати перевірку аргументу типу IFormFile.

Перше, що варто перевірити, – це чи об’єкт не дорівнює null – пустій області в пам’яті пристрою та чи файл не є пустим – містить 0 байтів. У випадку, якщо об’єкт не проходить перевірку – необхідно викинути помилку ValidationException, що потрібно створити у папці Common/Exceptions з кодом 400, що вказує на неправильні дані передані користувачем [26], та відповідним повідомленням.

Друга перевірка – чи справді користувач передав зображення. Для цього використовується перевірка розширення файлу, допустимі розширення – .png, .jpg, .jpeg. У разі не проходження перевірки викидається аналогічна помилка, що й у першому разі, але з іншим повідомленням.

```
13 references
public class CustomException : Exception
{
    2 references
    public virtual int ErrorCode { get; }

    0 references
    public CustomException() { }

    2 references
    public CustomException(string message)
        : base(message) { }

    2 references
    public CustomException(string message, CustomException innerException)
        : base(message, innerException) { }
}
```

```
5 references
public class ValidationException : CustomException
{
    2 references
    public override int ErrorCode => 400;

    0 references
    public ValidationException() { }

    2 references
    public ValidationException(string message)
        : base(message) { }

    0 references
    public ValidationException(string message, CustomException innerException)
        : base(message, innerException) { }
}
```

Рис. 3.27 Ієрархія помилок

Після перевірки зображення, з нього необхідно створити об’єкт Bitmap. Для цього використовується відкриття потоку для читання об’єкту IFormFile, а отриманий потік передається в конструктор класу Bitmap. Отриманий об’єкт можна передати у метод GetNumberCards та перевірити те, як він працює. Запустивши клієнтську частину додатку та наявну серверну, спробуємо передати зображення з рисунку 2.4.1 та перевірити як виглядатиме новий збережений файл.

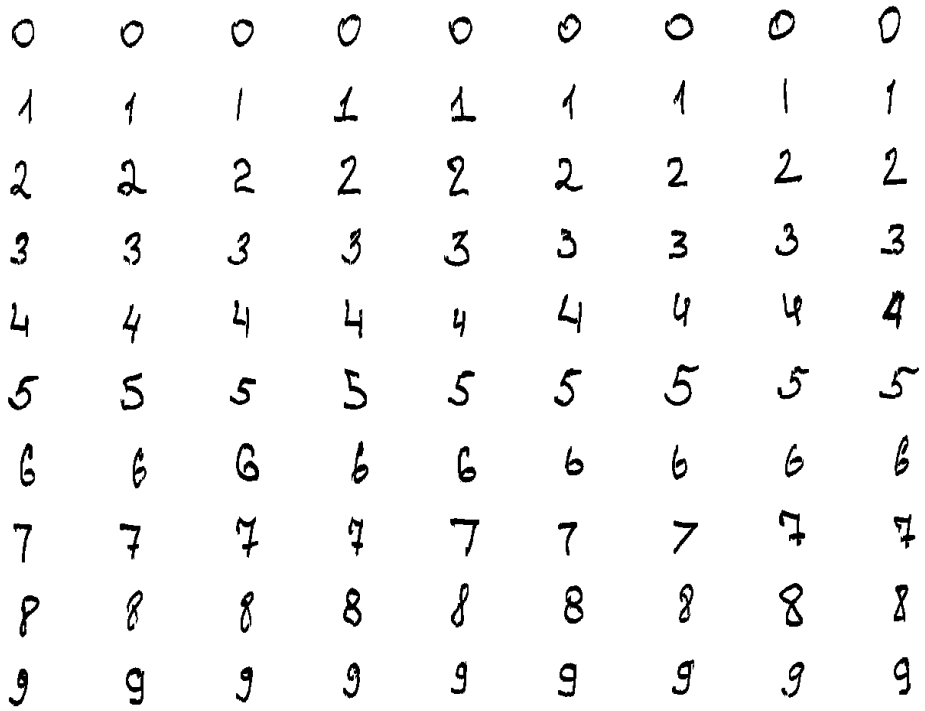


Рис. 3.28 Результат попередньої обробки зображення

3.4 Реалізація алгоритму пошуку межових прямокутників

Після попередньої обробки додамо виклик нового методу GetBoundingRectangles, що приймає оброблене зображення, знаходить всі межові прямокутники та сортує їх відповідно до їх положення.

Для повної реалізації алгоритму пошуку межових прямокутників знадобиться кілька методів:

1. FindBoundingBoxes – робить повний обхід по зображенню, записує відвідані пікселі у булеву матрицю та шукає чорний піксель. У разі знаходження

викликає метод FindBoundingBox та додає отриманий прямокутник в результуючу колекцію;

2. FindBoundingBox – виконує обхід об’єкту з чорних пікселів та записує його крайні точки, з яких в результаті будує результуючий прямокутник;
3. OrderRectancles – метод, що використовує рекурсію та сортує всі прямокутники по рядками та зліва направо.

Для реалізації обходу чорних об’єктів на зображенні в методі FindBoundingBox використовується структура даних під назвою черга, що дозволяє виконувати обхід навколо кожного знайденого пікселя почергово. Також важливо додати в реалізацію цього методу обробку можливих розривів цифр, що може бути особливістю почерку людини та/або особливістю знаряддя написання (ручка, олівець тощо). Для цього необхідно надати можливість алгоритму проходити на кілька білих пікселів та у разі знаходження чорних – продовжувати роботу.

Рис. 3.29 Можливі дефекти написання цифр

Також існує ймовірність створення межових прямокутників навколо шуму на зображенні. Хоча більшість непотрібної інформації видаляється під час попередньої обробки зображення, можуть знайтися, наприклад, крапки, залишені людиною випадково. Щоб не дозволити алгоритму створювати такі прямокутники необхідно задати його мінімальні розміри і не дозволяти обходити по білим пікселям, якщо його параметри менші та мінімум.

Для реалізації рекурсивного сортування отриманих прямокутників необхідно виконати такий порядок дій:

1. Перевірити колекцію на наявність елементів, у разі їх відсутності повернути пустий список;
2. Обрати перший прямокутник зверху – опорний елемент;
3. Знайти всі прямокутники, що знаходяться нижче опорного максимум на його половину;
4. Відсортувати всі отримані прямокутники (разом з опорним) по їх місцезнаходженню зліва;
5. Повернути отриману колекцію з об’єднанням результату виклику методу сортування без отриманих прямокутників.

Після виконання створеного алгоритму на зображенні з рисунку 3.3.7 було отримано 90 межових прямокутників, відсортованих на 10 рядків по 9 об’єктів в кожному, що повністю відповідає очікуваним показникам.

Для наочності роботи алгоритму, після виконання всіх операцій варто додати код, що накладає визначені межові прямокутники на вхідне зображення. Застосовані параметри:

- колір – червоний;
- товщина – два пікселя.

Отримане зображення буде повернуте користувачу.

3.5 Підготовка набору даних для навчання нейронної мережі

На даному етапі розробки інформаційної системи, майже все готове для створення набору даних для навчання нейронної мережі з наявних зображень рукописних цифр. Щоб перетворити межові прямокутники у зображення, необхідно для кожного з них створити свій об’єкт Bitmap та скопіювати у нього відповідні пікселі.

Для цього знадобиться два цикли, що проходитимуть всю колекцію прямокутників (по рядку та кожен прямокутник окремо). Метод створення колекції картинок з цифрами було названо ExtractDigits. Він приймає оброблене

зображення, колекцію прямокутників та параметр resizedSize, що буде необхідний для зміни розміру зображення в пікселях.

Першим етапом перетворення прямокутника в зображення є знаходження довшої сторони прямокутника, адже спотворення зображення через зміну відношення сторін є неприпустимим. Після знаходження довшої сторони необхідно обрахувати кількість нових пікселів для кожної сторони, що дорівнює нулю у разі, якщо обрана сторона є довшою, та половині різниці між коротшою да довшою в іншому випадку. Наступним етапом є створення нового об’єкту Bitmap з шириною і довжиною, що дорівнює довшій стороні прямокутника.

Після цього відбувається копіювання кольору пікселя з обробленого зображення в новостворене, у разі якщо він знаходиться в межах прямокутника, або задається білий колір пікселя, якщо координати відповідають новоствореній зоні.

```
using var map = new Bitmap(maxSide, maxSide);
for (int i = 0; i < map.Width; i++)
{
    for (int j = 0; j < map.Height; j++)
    {
        var color = Color.White;

        if (!(widthOffset > i
            || widthOffset + rectangle.Width < i
            || heightOffset > j
            || heightOffset + rectangle.Height < j))
        {
            color = processedImage.GetPixel(rectangle.Left - widthOffset + i, rectangle.Top - heightOffset + j);
        }

        map.SetPixel(i, j, color);
    }
}
```

Рис. 3.30 Створення зображення з межового прямокутника

Коли створення об’єкту Bitmap завершене, постає проблема того, що він має випадкову ширину та висоту, що відповідає параметрам на вхідному зображені. Тому, для використання таких зображень необхідно привести їх до однакового розміру, що задається параметром resizedSize. Оскільки розмірність цих зображень впливає на те, яку розмірність матиме нейронна мережа – це важливий вибір. Для даної роботи використано значення параметру 64.

Отже, перед використанням отриманих зображень необхідно обробити їх методом зміни розміру, назвемо цей метод ResizeBitmap, що прийматиме об’єкт Bitmap та розмір нового зображення, а повертатиме нове зображення. Сам процес відбувається за допомогою влаштованих в пакет System.Drawing.Common класів та методів, але потребує правильних параметрів для запобігання дефектів на вихідних зображеннях.

На даному етапі можна видалити код, що зберігає оброблене зображення та додати новий тимчасовий код, що зберігатиме зображення 64 на 64 пікселя для створення набору даних для тренування майбутньої нейронної мережі. Після цього потрібно запустити клієнтську та серверну частини додатку та почати завантажувати наявні зображення рукописних цифр.

Варто також додати їх нумерацію, щоб всі вони збереглися та не перезаписувалися одна на одну. Для цього в найменуванні зображення було використано тимчасове приватне поле, що збільшується на один після кожного збереження зображення.

```
1 reference
private static Bitmap ResizeBitmap(Bitmap originalBitmap, int newSide)
{
    var toReturn = new Bitmap(newSide, newSide);

    using (var graphics = Graphics.FromImage(toReturn))
    using (var attributes = new ImageAttributes())
    {
        toReturn.SetResolution(originalBitmap.HorizontalResolution, originalBitmap.VerticalResolution);

        attributes.SetWrapMode(WrapMode.TileFlipXY);

        graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;
        graphics.PixelOffsetMode = PixelOffsetMode.Half;
        graphics.CompositingMode = CompositingMode.SourceCopy;
        graphics.DrawImage(originalBitmap,
            Rectangle.FromLTRB(0, 0, newSide, newSide),
            0,
            0,
            originalBitmap.Width,
            originalBitmap.Height,
            GraphicsUnit.Pixel,
            attributes);
    }

    return toReturn;
}
```

Рис. 3.31 Метод зміни розміру зображення

Після обробки всіх наявних зображень та збереження карток (зображень) з рукописними цифрами можна видалити весь тимчасовий код та перейти до їх ручного сортування. Щоб створити набір даних для навчання нейронної мережі, в окремій папці створимо папки, підписавши їх цифрами від 0 до 9, та перенесемо туди відповідні картки. Також, необхідно виділити деяку частину зображень для тестування нейронної мережі.

В даному випадку картки були розподілені наступним чином:

- 1200 зображень (по 120 на кожну цифру) – для навчального набору;
- 200 (по 20 на кожну цифру) – для тестування.

Рис. 3.32 Частина набору даних для цифри 3

3.6 Розробка моделі нейронної мережі

Першим, що необхідно зробити перед початком розробки нейронної мережі, – це додати в існуюче рішення (solution) серверного додатку новий проєкт типу «Бібліотека класів» (Class Library), назвавши його NeuralNetwork. Після цього, створивши папку під назвою Infrastructure можна створювати класи нейронної мережі.

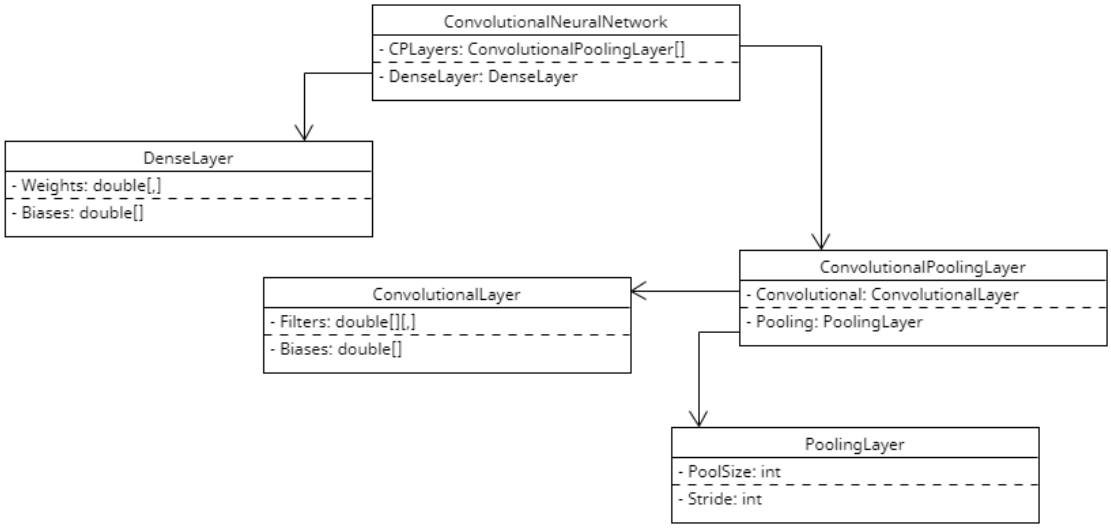


Рис. 3.33 Ієрархія класів нейронної мережі

Як видно з діаграми, зображеній на рисунку 3.6.1, у дана реалізація нейронної мережі міститиме три шари: два шари, що поєднують в собі шари згортки та об’єднання, і один щільний шар нейронів. Першим класом, що буде реалізовано є ConvolutionalLayer – шар згортки.

У папці Infrastructure необхідно створити файл з відповідною назвою. Одразу необхідно реалізувати конструктор, що прийматиме параметри:

1. inputSize цілочисельного типу – довжина сторони вхідного зображення;
2. filtersCount цілочисельного типу – кількість фільтрів (ядер) згорткового шару;
3. filtersShape типу кортеж двох цілочисельних значень – форма фільтру (оскільки для квадратного зображення будуть застосовуватися квадратні ядра, перша величина визначає висоту та ширину фільтрів, а друга – їх глибину).

За допомогою цих параметрів можна ініціалізувати всі необхідні для роботи шару поля класу, а саме кількість виходів шару, фільтри, зсуви і градієнти фільтрів та зсувів. Значення, що знаходяться у масивах ядер та зсувів задаються випадковим чином за допомогою методу NextDouble класу Random в межах від -0.5 до 0.5. Реальні значення будуть записані в ці поля під час навчання нейронної мережі.

```
1 reference
private double CalculateOutput(int filter, int xOffset, int yOffset, double[][,] input)
{
    double sum = 0;
    for (int i = 0; i < input.Length; i++)
    {
        for (int x = 0; x < _filtersShape.xy; x++)
        {
            for (int y = 0; y < _filtersShape.xy; y++)
            {
                for (int z = 0; z < _filtersShape.z; z++)
                {
                    sum += _filters[filter][x, y, z] * input[i][x + xOffset, y + yOffset];
                }
            }
        }
    }

    return Math.Max(0, sum + _biases[filter]);
}
```

Рис. 3.34 Метод, що визначає значення виходу

Після цього можна переходити до реалізації методу ForwardPropagation, що являє собою прямий прохід даних шаром нейронної мережі. Для полегшення читабельності коду, сама операція згортки для кожного вихідного значення винесена в окремий метод CalculateOutput, що також включає в себе функцію активації ReLU. Останньою операцією в методі ForwardPropagation є нормалізація отриманих значень в межах від 0 до 1 (від'ємні значення не враховуються, адже функція активації їх не допускає) методом NormalizeOutput.

Наступним класом, в якому буде реалізовано прямий прохід є PoolingLayer (об'єднуючий шар). Конструктор класу приймає два цілочисельні параметри – poolSize, що являє собою розмір вікна об'єднання, та stride, що визначає крок вікна після кожної операції.

Метод ForwardPropagation цього класу приймає такий самий тип, що повертає однойменний метод класу ConvolutionalLayer. Розмірності вихідних карт ознак визначаються на початку методу як різниця між розмірністю вхідних карт та розміру вікна, поділена на крок вікна плюс один. Кількість карт ознак при операції залишається незмінною, тому для кожного значення матриці з результуючого набору виконується метод MaxPooling, що являє собою виділення найбільшого значення з вікна об'єднання.

```
1 reference
private double MaxPooling(double[,] input, int startRow, int startCol, int poolSize)
{
    double max = double.MinValue;

    for (int i = 0; i < poolSize; i++)
    {
        for (int j = 0; j < poolSize; j++)
        {
            max = Math.Max(max, input[startRow + i, startCol + j]);
        }
    }

    return max;
}
```

Рис. 3.35 Операція максимального об’єднання

У тій же папці створюємо наступний файл з класом DenseLayer, що являє собою вихідний шар нейронної мережі, який складається з повністю зв’язаних нейронів. Конструктор цього класу приймає два цілочисельні параметри – кількість входів шару та кількість виходів. Кількість входів розраховуватиметься під час створення нейронної мережі з кількості виходів передостаннього шару, а кількість вихідних нейронів для даної задачі – 10. Не дивлячись на це, значення вихідних нейронів не буде задане строго, щоб дозволити подальші застосування розроблених моделей в майбутньому.

Параметри конструктора необхідно зберегти та за їх допомогою ініціалізувати обов’язкові поля класу, а саме ваги та зсуви. Зсуви являють собою одновимірний масив реальних чисел, кількість яких визначається кількістю виходів шару (по одному на кожен нейрон), а ваги – двовимірний масив реальних чисел, розмірності якого визначаються кількістю виходів та кількістю входів шару (кожен вихідний нейрон має власні ваги для кожного вхідного нейрону). Значення ваг та зсувів в конструкторі задаються випадковим чином за аналогією з класом ConvolutionalLayer.

У методі ForwardPropagation необхідно за допомогою двох вкладених циклів знайти для кожного вихідного нейрона суму відповідного зсуву та добутків значення кожного вхідного нейрона на відповідне значення ваг. Після знаходження значень всіх вихідних нейронів необхідно виконати функцію активації softmax для отриманих значень.

```
1 reference
private static double[] Activate(double[] outputs)
{
    for (int i = 0; i < outputs.Length; i++)
    {
        outputs[i] = Math.Exp(outputs[i]);
    }

    double sum = outputs.Sum();
    for (int i = 0; i < outputs.Length; i++)
    {
        outputs[i] = outputs[i] / sum;
    }

    return outputs;
}
```

Рис. 3.36 Функція активації softmax

Для зручності застосування шарів згортки та об’єднання було прийнято рішення створити клас, що поєднає їх функціонал (CP шар) та дозволить виконувати дії над ними інкапсульовано через відповідний інтерфейс взаємодії, що відповідає принципам ООП (об’єктно орієнтованого програмування). Новий клас матиме назву ConvolutionalPoolingLayer і прийматиме як параметри конструктора розмірність вхідних даних для шару згортки та об’єкт нового допоміжного класу конфігурації CPLayerConfiguration, що необхідно створити у новій папці Helpers.

```
4 references
public ConvolutionalPoolingLayer(int shape, CPLayerConfiguration conf)
{
    Convolutional = new ConvolutionalLayer(shape, conf.FiltersCount, conf.FiltersShape);
    Pooling = new PoolingLayer(conf.PoolSize, conf.Stride);

    FiltersCount = conf.FiltersCount;
    Side = (shape - conf.FiltersShape.xy + 1 - conf.PoolSize) / conf.Stride + 1;
}
```

Рис. 3.37 Конструктор класу ConvolutionalPoolingLayer

Цей клас міститиме чотири властивості, що необхідні для створення об’єктів класу ConvolutionalLayer та PoolingLayer. За допомогою цих даних, у конструкторі буде створено по екземпляру кожного з цих класів. Також, для майбутнього

використання, необхідно зберегти кількість фільтрів у шару згортки та обрахувати довжину сторін карт ознак після виконання операції об’єднання. Для збереження цих даних використано публічні властивості з приватним установником в класі ConvolutionalPoolingLayer.

Метод ForwardPropagation цього класу почергово викликає однойменний метод у класі ConvolutionalLayer та передає результат його виконання метод класу PoolingLayer, повертаючи отримане значення.

На даному етапі розробки реалізація прямого проходу готова для кожного шару, тому все готове для початку розробки класу нейронної мережі. У папці Infrastructure створимо файл ConvolutionalNeuralNetwork.cs, що міститиме однойменний клас. В цьому класі знаходитиметься дві властивості: масив ConvolutionalPoolingLayer та один DenseLayer.

Конструктор класу прийматиме довжину сторони вхідного зображення в пікселях та масив конфігурацій для створення класу ConvolutionalPoolingLayer та кількість вихідних нейронів мережі. За допомогою такого підходу можна налаштовувати розмірність вхідних даних, кількість та параметри шарів згортки та об’єднання і кількість вихідних нейронів для кожного екземпляру мережі. Для колекції конфігурацій також варто додати атрибути Required та MinLength з параметром 1, що запобіжить передачі null та пустої колекції в конструктор.

```
0 references
public ConvolutionalNeuralNetwork(
    int inputSide,
    [Required, MinLength(1)] CPLayerConfiguration[] cpLayersConfigurations,
    int outputSize)
{
    CPLayers = new ConvolutionalPoolingLayer[cpLayersConfigurations.Length];

    CPLayers[0] = new ConvolutionalPoolingLayer(inputSide, cpLayersConfigurations[0]);
    for (int i = 1; i < cpLayersConfigurations.Length; i++)
    {
        CPLayers[i] = new ConvolutionalPoolingLayer(CPLayers[i - 1].Side, cpLayersConfigurations[i]);
    }

    int lastLayersFiltersCount = CPLayers[^1].FiltersCount;
    int lastLayersSideSquare = (int)Math.Pow(CPLayers[^1].Side, 2);
    DenseLayer = new DenseLayer(lastLayersFiltersCount * lastLayersSideSquare, outputSize);
}
```

Рис. 3.38 Конструктор класу ConvolutionalNeuralNetwork

У конструкторі необхідно створити для кожного об’єкту конфігурацій відповідний екземпляр ConvolutionalPoolingLayer, де у якості довжини сторони вхідного зображення для першого шару виступає відповідний параметр конструктора ConvolutionalNeuralNetwork, а для кожного наступного – властивість Side попереднього шару. Кількість вхідних нейронів для шару щільних нейронів визначається як кількість фільтрів попереднього шару, помноженої на квадрат властивості Side останнього шару згортки та об’єднання.

Після реалізації конструктора необхідно реалізувати метод FeedForward, що відповідатиме за прямий прохід усією нейронною мережею. Цей метод приймає на вхід набір матриць для значень кожного кольорового каналу зображення та спочатку пропускає ці дані через всі шари згортки та об’єднання, а результат цих операцій передає в шар щільних нейронів для генерації остаточного результату.

Проте, оскільки результатом виконання методу ForwardPropagation класу ConvolutionalPoolingLayer є набір двовимірних матриць, а клас DenseLayer приймає на вхід методу прямого проходу одновимірний масив значень, дані необхідно привести до необхідного формату. Для цього потрібно перед передачею даних на шар щільний нейронів застосувати на них метод перетворення масиву під назвою FlattenArray. Цей метод копіює значення з набору матриць в новий одновимірний масив, що своєю довжиною дорівнює кількості матриць, помноженій на кількість об’єктів в кожній з них. Результат виконання прямого проходу даних через шар щільних нейронів є результатом методу FeedForward усієї нейронної мережі.

На цьому розробка інфраструктури для прямого проходу даних нейронною мережею завершена. Але оскільки для використання нейронної мережі потрібне її тренування, необхідно реалізувати алгоритми зворотного поширення помилки.

Першим класом, в якому буде реалізовано метод Backpropagation є DenseLayer. Цей метод приймає в себе два параметри – outputGradient, що являє собою одновимірний масив значень градієнт функції втрат (похідної від квадратичної функції втрат) між результатом, отриманим від прямого проходу мережею, та очікуваними від нейронної мережі значеннями, та learningRate – число від 0 до 1, що визначає швидкість навчання шару.

```
1 reference
public double[] Backpropagation(double[] outputGradients, double learningRate)
{
    double[] inputGradients = new double[_inputSize];

    for (int i = 0; i < _outputSize; i++)
    {
        for (int j = 0; j < _inputSize; j++)
        {
            inputGradients[j] += outputGradients[i] * _weights[i, j];
            _weights[i, j] -= learningRate * outputGradients[i] * _inputs[j];
        }

        _biases[i] -= learningRate * outputGradients[i];
    }

    return inputGradients;
}
```

Рис. 3.39 Метод Backpropagation класу DenseLayer

У цьому методі відбувається оновлення всіх ваг, віднімаючи від кожного значення добуток між швидкістю навчання, значення градієнта функції втрат того вихідного нейрона, до якого належить це значення ваг, та значенням вхідного нейрона, до якого належить це значення ваг. Паралельно до цього процесу відбувається накопичення значень помилки для кожного вхідного нейрону, що необхідно для подальшої реалізації зворотного поширення помилки. Також, для кожного нейрону оновлюється значення зсуву на різницю між наявним зсувом та добутком між швидкістю навчання та значенням градієнта функції втрат цього нейрона.

Також варто зазначити, що кожен з трьох видів шарів нейронної мережі зберігає в собі останні дані, що були передані для виконання прямого проходу. Це необхідно для реалізації методів зворотного поширення помилки (Backpropagation).

Перейдемо до створення методу Backpropagation для класу PoolingLayer. Суть цього методу заключається в передачі градієнтів з попереднього шару на пікселі, які взяли участь у виборі максимального значення під час операції об’єднання. Для цього за допомогою циклів відбувається копіювання всіх значень матриць градієнтів у новий набір, що має такі ж розмірності як і вхідні дані для

прямого проходження. Оскільки розмірність вхідних даних більша, ніж вихідних, використовується метод, що знаходить необхідні індекси у вхідних даних.

Метод зворотного поширення помилки в класі ConvolutionalLayer прийматиме два параметра: набір матриць градієнтів від методу Backpropagation класу PoolingLayer (під назвою delta) та швидкість навчання нейронної мережі. Першим кроком в реалізації цього методу є обчислення градієнтів для кожного ядра та зсуву. Ці значення для зсувів кожного фільтру визначаються як сума всіх градієнтів цього фільтру. Для визначення значень помилки фільтрів використовується сума всіх добутків між вхідними даними (для прямого проходження) та вхідним градієнтом, за аналогією з виконанням методу ForwardPropagation. Після цього відбувається визначення градієнту помилок вхідних даних (рис. 3.6.8), який стане результатом виконання методу та буде повернений для подальшого використання. Останнім кроком є оновлення значень фільтрів та зсувів на різницю між наявним значенням та добутком швидкості навчання на відповідне значення градієнту.

```
var inputGradients = new double[_filtersCount][,];
for (int f = 0; f < _filtersCount; f++)
{
    inputGradients[f] = new double[_inputSize, _inputSize];

    for (int i = 0; i < _outputSize; i++)
    {
        for (int j = 0; j < _outputSize; j++)
        {
            for (int x = 0; x < _filtersShape.xy; x++)
            {
                for (int y = 0; y < _filtersShape.xy; y++)
                {
                    for (int z = 0; z < _filtersShape.z; z++)
                    {
                        inputGradients[f][x + i, y + j] += delta[f][i, j] * _filters[f][x, y, z];
                    }
                }
            }
        }
    }
}
```

Рис. 3.40 Визначення градієнту помилок вхідних даних

Наступним кроком є реалізація методу Backpropagation для класу ConvolutionalPoolingLayer, що спочатку виконує однойменний метод для шару об’єднання, а результат передає у шар згортки і повертає отримане значення.

Для реалізації Backpropagation для класу нейронної мережі необхідно вирішити одну проблему. Оскільки метод Backpropagation для класу ConvolutionalPoolingLayer прийматиме дані такого типу, як повертає метод ForwardPropagation (колекцію двовимірних матриць), а шар щільних нейронів при виконанні зворотного поширення помилки повертає одновимірний масив накопичених помилок, необхідно перетворити дані протилежним чином тому, як це робить метод FlattenArray. Цей метод матиме назву UnflattenArray та прийматиме два параметри: одновимірний масив та необхідну кількість матриць. Розміри матриці визначаються як квадратний корінь з частки від ділення довжини вхідного масиву на кількість матриць.

Для наочності результатів навчання, метод зворотного поширення помилки в нейронній мережі повертатиме логічне значення, що визначатиме чи правильно мережа визначила очікуване значення.

Задля подальшого відновлення натренованої нейронної мережі необхідно створити класи, що зберігатимуть всю необхідну інформацію і в подальшому будуть серіалізовані та записані у файл. У новій папці Snapshots необхідно створити 5 класів з відповідними властивостями.

Таблиця 3.2

Класи знімків стану нейронної мережі

Клас	**Функція класу**	**Властивість**	**Функція властивості**
ConvolutionalLayerSnapshot	Знімок згорткового шару	InputSize	Розмір сторони вхідної матриці
		Filters	Значення фільтрів шару
		Biases	Значення зсувів шару
PoolingLayerSnapshot	Знімок шару об’єднання	PoolSize	Зозмір вікна об’єднання
		Stride	Крок об’єднання
ConvolutionalPoolingLayerSnapshot	Знімок СР шару	Convolutional	Знімок згорткового шару
		Pooling	Знімок шару об’єднання

Продовження таблиці 3.2

Класи знімків стану нейронної мережі

Клас	**Функція класу**	**Властивість**	**Функція властивості**
DenseLayerSnapshot	Знімок щільного шару нейронів	InputSize	Кількість вхідних нейронів
		OutputSize	Кількість вихідних нейронів
		Weights	Значення ваг шару
		Biases	Значення зсувів шару
NetworkSnapshot	Знімок нейронної мережі	CPLayers	Колекція знімків об’єднаних шарів згортки та об’єднання
		DenseLayer	Знімок щільного шару нейронів

Також для кожного класу шарів нейронної мережі та самої нейронної мережі потрібно реалізувати конструктори, що приймають об’єкт відповідного класу знімку, та методи, що повертають такі знімки.

3.7 Навчання нейронної мережі

Для реалізації навчання нейронної мережі необхідно створити новий проєкт NetworkTraining типу консольного додатку (Console Application). До нього необхідно додати пакети System.Drawing.Common для роботи з зображеннями та Newtonsoft.Json для серіалізації та десеріалізації знімку нейронної мережі у форматі JSON.

Першим етапом процесу навчання нейронної мережі є завантаження набору даних та очікуваним результатом для кожного зображення. Оскільки набір даних було розсортовано за очікуваними значеннями, для цього можна скористатися масивом всіх цифр та за допомогою вбудованих бібліотек .NET та завантажити у

масив об’єктів нового класу DatasetItem, що зберігатиме всю необхідну інформацію для кожного зображення. Шлях до папки з набором даних було поміщено у константу ImagesPath.

```
int[] numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
DatasetItem[] data = numbers.SelectMany(n =>
{
    double[] expected = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
    expected[n] = 1;

    var imageFiles = Directory.GetFiles($@"{ImagesPath}/{n}");

    return imageFiles.Select(e => new DatasetItem(e, BitmapToFloatArray(new Bitmap(e)), expected));
}).ToArray();

data.Shuffle();
```

Рис. 3.41 Завантаження набору даних з носія

Метод BitmapToFloatArray перетворює об’єкт Bitmap в набір матриць, що містить значення яскравості кожного пікселя, що є реальним числом в межах від 0 до 1, тому ці дані не потрібно нормалізовувати. Оскільки зображення складаються з двох кольорів, набір містить одну матрицю. Також, для зменшення обрахунків нейронною мережею доцільно використовувати обернене значення яскравості (один мінус значення яскравості).

Метод Shuffle, що застосовується до набору даних, перемішує їх порядок, щоб не створювати довгої послідовності даних, що очікують однієї відповіді. Це необхідно щоб нейронна мережа не навчалася залежностям, які виникають від конкретного порядку даних, а вивчала важливі характеристики зображення. Такі дії допомагають стабільному та швидкому навчанню і робить модель здатною працювати з новими невідомими даними.

Наступним йде код вибору дій, а саме створення нової нейронної мережі чи відновлення вже існуючої моделі, за допомогою команд консолі. Для визначення оптимальної конфігурації шарів згортки та об’єднання було проведено велику кількість тестувань з різними налаштуваннями. Параметри, що проявили себе найкраще, було використано в кінцевій конфігурації нейронної мережі.

Таблиця 3.3

Конфігурація нейронної мережі

Об’єкт	**Назва параметру**	**Значення параметру**
Нейронна мережа	Кількість СР шарів	2
	Розмірність вхідної матриці	64*64
СР шар №1	Кількість фільтрів	3
	Розмірність фільтрів	3*3*1
	Розмір вікна об’єднання	2
	Крок об’єднання	2
СР шар №2	Кількість фільтрів	5
	Розмірність фільтрів	3*3*2
	Розмір вікна об’єднання	2
	Крок об’єднання	2
Шар щільних нейронів	Кількість вхідних нейронів	980
	Кількість вихідних нейронів	10

Навчання нейронної мережі полягає в багаторазовому виклику методу Backpropagation для кожного об’єкту з набору даних. Повний прохід всіх даних прийнято називати епохою. Для кожної епохи буде обраховуватися кількість помилок і виводитися в консоль разом з її номером, це дасть уявлення про перебіг навчання. Навчання буде розпочате з відносно великим значенням змінної його швидкості (learningRate), але в його процесі вона буде зменшуватися, що дозволить спочатку дати нейронній мережі можливість швидко вивчити основні особливості кожної з цифр, а в кінці дозволить точно налаштувати значення фільтрів, ваг та зсувів [27]. Після закінчення навчання, знімок мережі серіалізується в новий файл network-state.json.

Таблиця 3.4

Зміна швидкості навчання нейронної мережі

Епохи	0-30	31-70	71-100
Швидкість навчання	0.01	0.002	0.0005

Під час навчання точність нейронної мережі склала 99.75% (3 помилки) на навчальному наборі даних. Для набору з 200 тестових зображень, що не приймали участь в навчанні, точність склала 97.5% (5 помилок), що більше ніж у розглянутих сучасних систем. Досягнення 100% точності при навчанні нейронної мережі не завжди є бажаним результатом і може сигналізувати про деякі проблеми чи обмеження моделі.

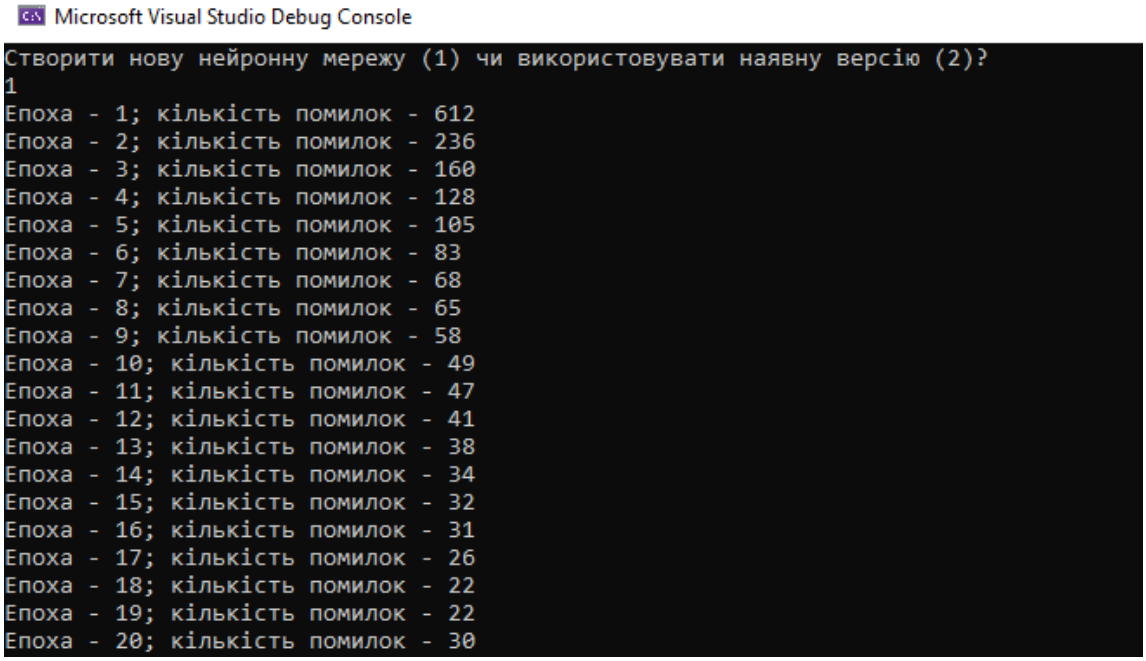


Рис. 3.42 Процес навчання нейронної мережі

Таблиця 3.5

Причини небажаності 100% точності нейронної мережі при навчанні

Причина	**Пояснення**
Перенавчання (Overfitting)	Модель "вивчає" не тільки загальні закономірності, але й шум чи випадковість в навчальних даних.
Невідомість	Деякі дані можуть бути неочікувано важкими для моделі через аномалії чи невивчені патерни.
Надмірна складність моделі	Занадто складні моделі можуть витратжувати занадто багато ресурсів та перенавчатися.
Несумісність з реальністю	Ідеальна точність на навчальному наборі не завжди відображає реальні умови застосування моделі.
Вартість обчислень	Досягнення 100% точності може вимагати значних обчислювальних ресурсів та часу.

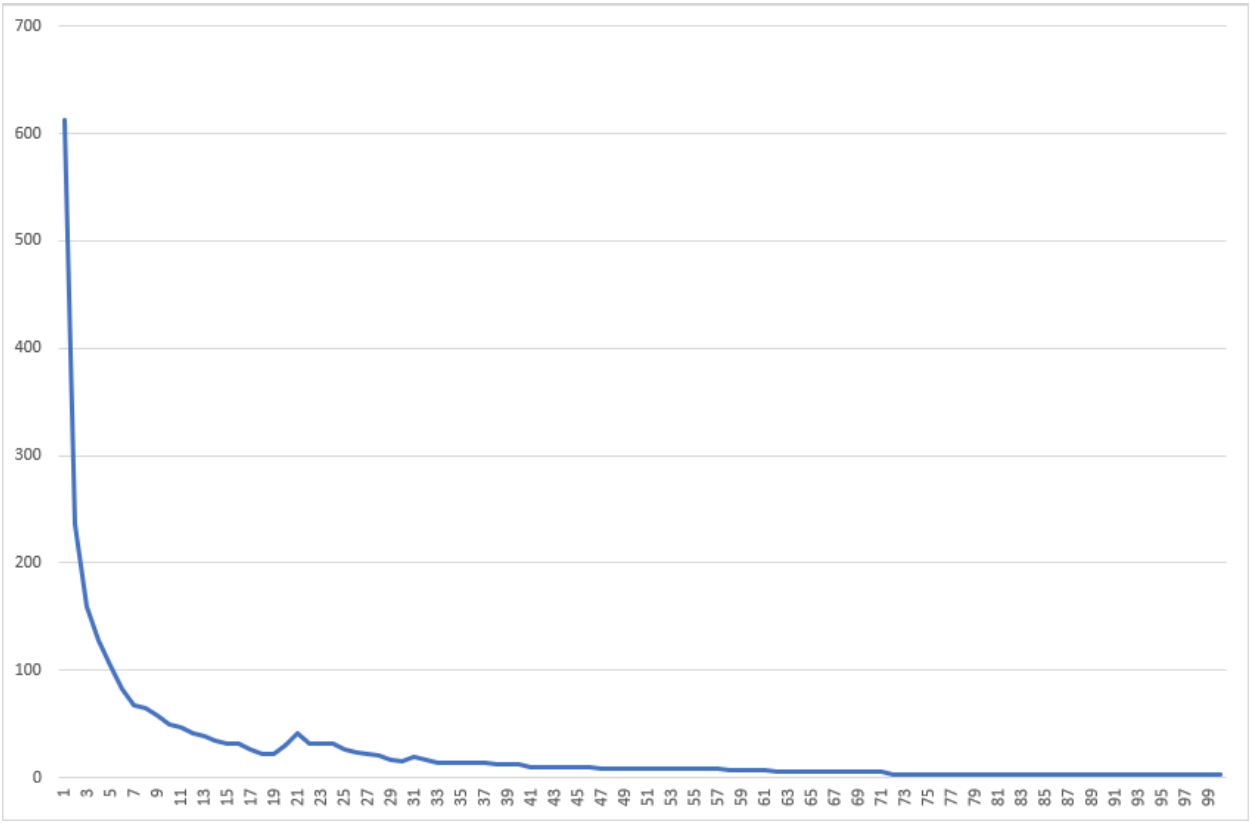


Рис. 3.43 Графік зміни кількості помилок за епоху під час навчання нейронної мережі

3.8 Перевірка результатів реалізації розпізнавання

Для використання створеної нейронної мережі в додатку, необхідно створити спеціальний сервіс NeuralNetworkHolder, що при створенні буде десеріалізувати її знімок та на його основі створювати екземпляр натренованої моделі. Такий сервіс буде зареєстровано як Singleton. Єдиний його метод прийматиме об’єкт Bitmap, аналізуватиме його вміст за допомогою нейронної мережі та повертатиме результат.

Для перевірки створеної системи будуть використані зображення з рисунку 1.4.1 (зображення №1 та №2) та кілька інших, що не приймали участь в створенні набору даних для навчання. Загалом було використано 6 фото, з загальною кількістю цифр - 166. Результати тестування системи наведені в таблиці 3.8.1.

Таблиця 3.6

Результати тестування системи наборі зображень

№ зображення	Роздільна здатність зображення, пікселів	Кількість цифр	Кількість правильно розпізнаних цифр	Час, витрачений на розпізнавання, с
1	556*272	10	10	1.46
2	674*463	10	10	2.23
3	1280*1136	10	9	7.5
4	954*545	21	20	3.83
5	1280*850	23	23	7.52
6	1280*655	92	87	7.40

Точність нейронної мережі в даному тестуванні склала 95.78%. З даної таблиці можна зробити висновок, що точність мережі не сильно залежить від роздільної здатності зображення, але вона сильно впливає на швидкість розпізнавання. Тому, ймовірну подальшу оптимізацію системи задля пришвидшення її роботи системи варто проводити з алгоритмами роботи з зображеннями.

Таблиця 3.7

Оновлена таблиця 1.4.1 з даними тестування створеної системи

Назва	Результат тестування №1	Час, витрачений на тестування №1, с	Результат тестування №2	Час, витрачений на тестування №2, с
OCR Space	5	1.82	8	1.02
Pen To Print	10	5.58	9	5.61
Image To Text	10	6.26	8	5.62

Продовження таблиці 3.7

Оновлена таблиця 1.4.1 з даними тестування створеної системи

Назва	**Результат тестування №1**	**Час, витрачений на тестування №1, с**	**Результат тестування №2**	**Час, витрачений на тестування №2, с**
img2txt	7	0.784	0	2.84
Створена система	10	1.46	10	2.23

Як видно з таблиці, реалізована система має вищу точність та продуктивність, ніж комерційні аналоги, оглянуті в розділі 1.4. Також, її особливістю є те, що вона є відкритою до оновлення, модифікацій та допрацювань завдяки вірно застосованим сучасним методам і підходам до проектування та розробки інформаційних систем.

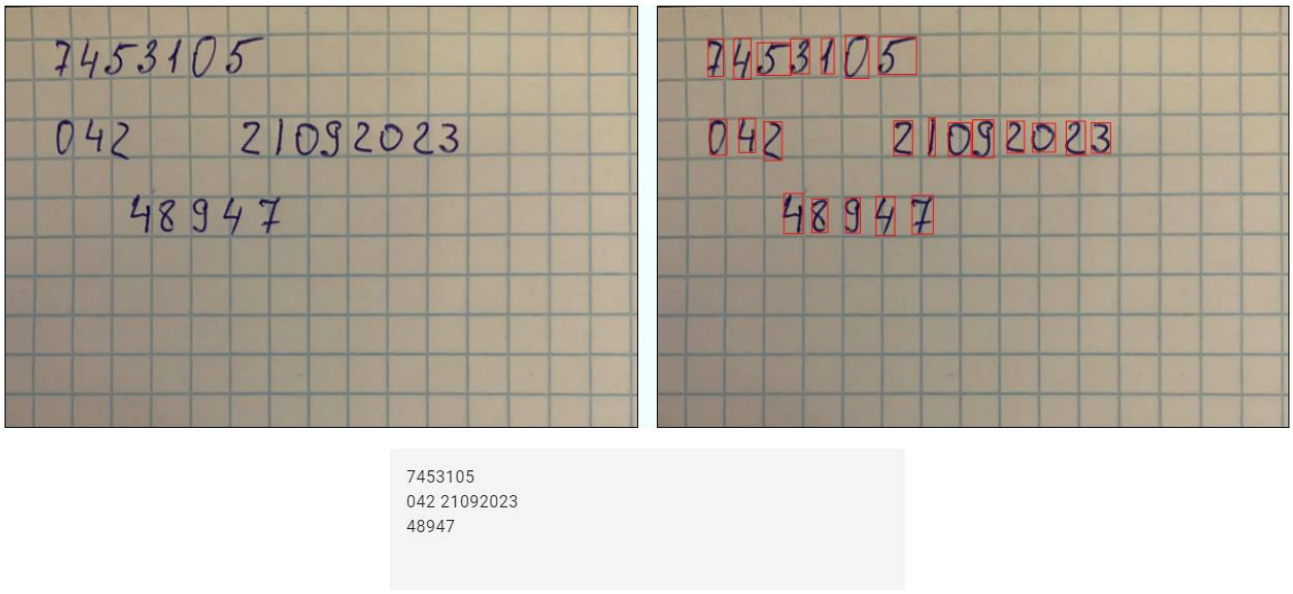


Рис. 3.44 Процес тестування створеної системи

ВИСНОВКИ

У даній роботі було проведено дослідження роботи нейронних мереж з метою підвищити точність систем розпізнавання рукописних чисел. Головними сферами застосування таких систем можуть стати документообіг, освіта та наука, банківська сфера та поштові служби. Проведено аналіз сучасних наукових робіт та виявлено, що наявні системи мають точність близько 90%. Тестування цих систем на складних даних показали ще меншу точність.

За результатами огляду наукових джерел було складено порядок операцій, за яким працюватиме майбутня система, а саме:

1. Попередня обробка зображення;
2. Вилучення об’єктів (цифр);
3. Підготовка до розпізнавання;
4. Розпізнавання об’єктів;
5. Генерація результату.

Попередня обробка зображень складається з трьох етапів: фільтр підвищення різкості зображення, розмивання Гауса та застосування порогового значення пікселя з отриманням чорно-білого зображення. Вилучення об’єктів відбувається за допомогою алгоритму пошуку межових прямокутників з використанням обходу в ширину.

Обраним видом нейронних мереж для даної задачі є згорткова нейронна мережа, адже вона добре пристосована для роботи з зображеннями. Обраним методом навчання є зворотнє поширення помилки з градієнтним спуском, бо він є найкращим рішенням для задач класифікації.

Реалізовано систему мовою програмування C# на платформі .NET версії 8.0, з використанням інтегрованого середовища розробки Visual Studio. Для наочності отримуваних результатів було розроблено клієнтську частину, використовуючи фреймворк Angular, з використанням Visual Studio Code.

Модель згорткової нейронної мережі складається з п’яти шарів, що були об’єднанні в три, а саме двох шарів згортки та об’єднання і одного повнозв’язного

шару. Навчання системи проводилося на власно створеному наборі даних, що складався з 1200 зображень (по 120 на кожну цифру).

Результати тестування моделі показали практичну точність у 95.78%, що є більшим показником, ніж в оглянутих системах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ghosh R. RNN based online handwritten word recognition in Devanagari and Bengali scripts using horizontal zoning [Електронний ресурс] / Rajib Ghosh, Chirumavila Vamshi, Prabhat Kumar // Pattern recognition. – 2019. – Т. 92. – С. 203–218. – Режим доступу: https://doi.org/10.1016/j.patcog.2019.03.030 (дата звернення: 06.11.2023)
2. Mursari L. The effectiveness of image preprocessing on digital handwritten scripts recognition with the implementation of OCR Tesseract / Lily Mursari, Antoni Wibowo // Computer engineering and applications. – 2021. – Т. 10, № 3. – С. 177–186.
3. High-Performance OCR for printed english and fraktur using LSTM networks [Електронний ресурс] / Thomas M. Breuel [та ін.] // 2013 12th international conference on document analysis and recognition (ICDAR), Washington, DC, USA, 25–28 серп. 2013 р. – [Б. м.], 2013. – Режим доступу: https://doi.org/10.1109/icdar.2013.140 (дата звернення: 06.11.2023)
4. Що таке УНЗР та де його взяти? [Електронний ресурс] // Державна міграційна служба. – Режим доступу: https://dmsu.gov.ua/news/region/6563.html (дата звернення: 11.11.2023)
5. MacKay J. Productivity in 2017: What we learned from analyzing 225 million hours - rescueTime [Електронний ресурс] / Jory MacKay // RescueTime Blog. – Режим доступу: https://blog.rescuetime.com/225-million-hours-productivity/ (дата звернення: 09.11.2023)
6. Al-Taee M. M. Handwritten Recognition: a survey [Електронний ресурс] / May Mowaffaq Al-Taee, Sonia Ben Hassen Neji, Mondher Frikha // 2020 IEEE 4th international conference on image processing, applications and systems (IPAS), Genova, Italy, 9–11 груд. 2020 р. – [Б. м.], 2020. – Режим доступу: https://doi.org/10.1109/ipas50080.2020.9334936 (дата звернення: 01.12.2023)

7. Handwritten digit recognition using machine learning algorithms [Електронний ресурс] / S. M. Shamim [та ін.] // Indonesian journal of science and technology. – 2018. – Т. 3, № 1. – С. 29. – Режим доступу: https://doi.org/10.17509/ijost.v3i1.10795 (дата звернення: 11.11.2023)
8. Baldominos A. A survey of handwritten character recognition with MNIST and EMNIST [Електронний ресурс] / Alejandro Baldominos, Yago Saez, Pedro Isasi // Applied sciences. – 2019. – Т. 9, № 15. – С. 3169. – Режим доступу: https://doi.org/10.3390/app9153169 (дата звернення: 11.11.2023).
9. MNIST handwritten digit recognition using machine learning [Електронний ресурс] / Elizabeth Rani G [та ін.] // 2022 2nd international conference on advance computing and innovative technologies in engineering (ICACITE), Greater Noida, India, 28–29 квіт. 2022 р. – [Б. м.], 2022. – Режим доступу: https://doi.org/10.1109/icacite53722.2022.9823806 (дата звернення: 03.12.2023)
10. Digit recognition of MNIST handwritten using convolutional neural networks (CNN) [Електронний ресурс] / Sakthimohan M [та ін.] // 2023 international conference on intelligent systems for communication, iot and security (iciscois), Coimbatore, India, 9–11 лют. 2023 р. – [Б. м.], 2023. – Режим доступу: https://doi.org/10.1109/iciscois56541.2023.10100602 (дата звернення: 03.12.2023)
11. Shashank R. Handwritten character recognition using deep convolutional neural networks [Електронний ресурс] / R. Shashank, A. Adarsh Rai, P. Srinivasa Pai // Advances in intelligent systems and computing. – Singapore, 2021. – С. 253–262. – Режим доступу: https://doi.org/10.1007/978-981-16-3342-3_21 (дата звернення: 11.11.2023)
12. Free OCR API V2023, online OCR, searchable PDF creator, on-premise OCR software [Електронний ресурс] // Free OCR API V2023, Online OCR, Searchable PDF Creator, On-Premise OCR Software. – Режим доступу: https://ocr.space/ (дата звернення: 13.11.2023)

13. Free Online OCR - Convert PDF or image to text, word, docx or odf [Електронний ресурс] // Free Online OCR - Convert PDF or image to text, word, docx or odf. – Режим доступу: https://img2txt.com/ (дата звернення: 13.11.2023)
14. Al-Ameen Z. Improving the sharpness of digital image using an amended unsharp mask filter [Електронний ресурс] / Zohair Al-Ameen, Alaa Muttar, Ghofran Al-Badrani // International journal of image, graphics and signal processing. – 2019. – Т. 11, № 3. – С. 1–9. – Режим доступу: https://doi.org/10.5815/ijigsp.2019.03.01 (дата звернення: 04.12.2023)
15. Estimating generalized gaussian blur kernels for out-of-focus image deblurring [Електронний ресурс] / Yu-Qi Liu [та ін.] // IEEE transactions on circuits and systems for video technology. – 2020. – С. 1. – Режим доступу: https://doi.org/10.1109/tcsvt.2020.2990623 (дата звернення: 04.12.2023)
16. Vo X.-T. Accurate bounding box prediction for single-shot object detection [Електронний ресурс] / Xuan-Thuy Vo, Kang-Hyun Jo // IEEE transactions on industrial informatics. – 2021. – С. 1. – Режим доступу: https://doi.org/10.1109/tii.2021.3138336 (дата звернення: 22.11.2023)
17. Матвійчук А. М. Алгоритм обходу зображення для пошуку межових прямокутників з використанням обходу в ширину. // IV науково-практична конференція «Проблеми комп'ютерної інженерії». – Київ: ДУТ, 2023. – с. 123-125
18. Simonyan K. Very Deep Convolutional Networks for Large-Scale Image Recognition / Karen Simonyan, Andrew Zisserman // ICLR 2015, Сан-Дієго, 7 лип. 2014 р. – [Б. м.].
19. Dumoulin V. A guide to convolution arithmetic for deep learning / Vincent Dumoulin, Francesco Visin. – [Б. м.] : Монреа. ун-т, 2018. – 31 с.
20. LeCun Y. Deep learning [Електронний ресурс] / Yann LeCun, Yoshua Bengio, Geoffrey Hinton // Nature. – 2015. – Т. 521, № 7553. – С. 436–444. – Режим доступу: https://doi.org/10.1038/nature14539 (дата звернення: 05.12.2023)

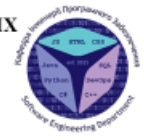
21. Bishop C. M. Pattern recognition and machine learning / Christopher M. Bishop. – [Б. м.] : Springer, 2016. – 758 с.
22. Zhang D. The application research of neural network and BP algorithm in stock price pattern classification and prediction [Електронний ресурс] / Dehua Zhang, Sha Lou // Future generation computer systems. – 2021. – Т. 115. – С. 872–879. – Режим доступу: https://doi.org/10.1016/j.future.2020.10.009 (дата звернення: 05.12.2023)
23. .NET downloads (linux, macos, and windows) [Електронний ресурс] // Microsoft. – Режим доступу: https://dotnet.microsoft.com/en-us/download/dotnet (дата звернення: 26.11.2023)
24. Angular [Електронний ресурс] // Angular. – Режим доступу: https://angular.io/cli (date of access: 26.11.2023).
25. Richter J. CLR via C# / Jeffrey Richter. – 4-те вид. – Redmond : Microsoft Press, 2012. – 863 с.
26. RFC 9110: HTTP semantics [Електронний ресурс] // RFC Editor. – Режим доступу: https://www.rfc-editor.org/rfc/rfc9110.html#name-400-bad-request (дата звернення: 27.11.2023)
27. Back-Propagation [Електронний ресурс] // Deep learning neural networks. – [Б. м.], 2016. – С. 23–32. – Режим доступу: https://doi.org/10.1142/9789813146464_0003 (дата звернення: 01.12.2023).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення



МАГІСТЕРСЬКА РОБОТА

«ПІДВИЩЕННЯ ЯКОСТІ РОЗПІЗНАВАННЯ РУКОПИСНИХ ЧИСЕЛ ЗА ДОПОМОГОЮ НЕЙРОМЕРЕЖІ»

Виконав: Студент групи ПДМ-64 Матвійчук Артем Миколайович
Керівник: д.т.н., проф., доцент кафедри ТЦР Жебка Вікторія Вікторівна

Київ - 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета дослідження: підвищення якості розпізнавання рукописних чисел

Об'єкт дослідження: процес розпізнавання рукописних чисел

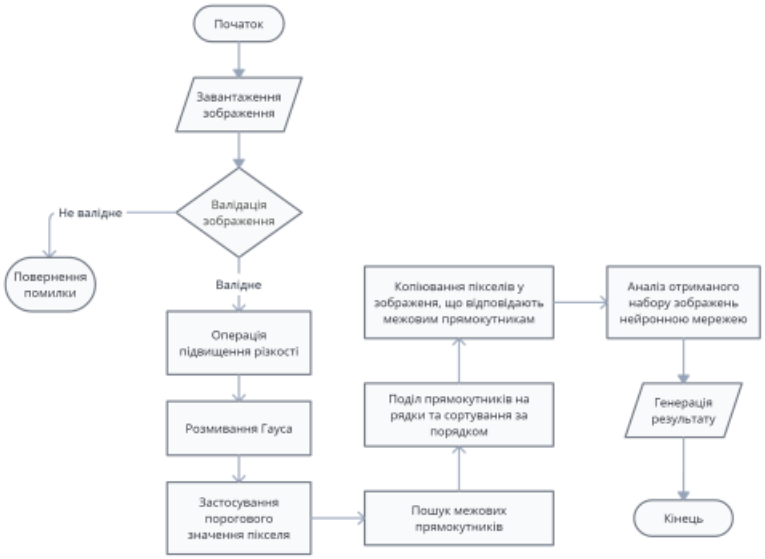
Предмет дослідження: алгоритми штучного інтелекту, які використовуються для розпізнавання рукописних чисел

КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ РОЗПІЗНАВАННЯ

Критерій	Опис
Витрачений час	Час, який потрібно системі для проведення розпізнавання
Точність	Відсоток правильно розпізнаних цифр
Пристосованість	Стійкість до зміни якості зображення

3

АЛГОРИТМ РОБОТИ СИСТЕМИ



4

МАТЕМАТИЧНА МОДЕЛЬ ПОПЕРЕДНЬОЇ ОБРОБКИ ЗОБРАЖЕНЬ

1. Операція підвищення різкості:

$$I'_{x,y} = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I_{x+i,y+j} * w_{i+k+1,j+k+1}, \quad (1)$$

де I – матриця значень пікселів вхідного зображення;
I' – матриця значень пікселів результуючого зображення;
w – ядро підвищення різкості (квадратна матриця непарного порядку n);
k – радіус ядра ($k=(n-1)/2$, де n – порядок матриці w);

2. Розмиття Гауса:

$$I''_{x,y} = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I_{x+i,y+j} \cdot H_{i+k+1,j+k+1}, \quad (2)$$

де I' - матриця значень пікселів зображення з попереднього кроку;
I'' - значення пікселя вихідного зображення в точці $(x+i, y+j)$;
H – ядро підвищення різкості (квадратна матриця непарного порядку n);
k – радіус ядра ($k=(n-1)/2$, де n – порядок матриці w);
Значення матриці H в координатах (i, j) визначаються формулою:

$$H_{i,j} = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{i^2+j^2}{2\sigma^2}}, \quad (3)$$

де x та y - відстані від центру ядра;
e – число Ейлера;
σ - параметр розмиття, що визначає ширину гаусівського розподілу.

5

МАТЕМАТИЧНА МОДЕЛЬ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

1. Операція згортки:

Шар згортки

$$S'_{i,j} = \sum_{m=1}^{M} \sum_{n=1}^{N} S_{i+m,j+n} \cdot K_{m,n}, \quad (4)$$

де S - матриця значень пікселів обробленого для аналізу зображення;
S' - вихідна матриця значень;
K – ядро згортки (матриця розміру M x N).

2. Функція активації (ReLU):

$$ReLU(x) = \max(0, x). \quad (5)$$

3. Операція об'єднання:

$$P(i,j) = max_{m,n}\left(S'_{i*s+m\ j*s+n}\right), \quad (6)$$

де P – матриця значень пікселів після максимального об'єднання;
S' – матриця з попереднього кроку;
s – крок об'єднання (stride);
m та n – розмірність вікна об'єднання.

Шар об'єднання

4. Значення вагової суми вихідного нейрону:

$$z_i = \sum_j w_{ij}^{(L)} \cdot a_j^{(L-1)} + b_i^{(L)}, \quad (7)$$

де L вказує на останній (вихідний) шар мережі;
$w_{ij}^{(L)}$ - вага між j-м нейроном у попередньому шарі та i-м нейроном у вихідному шарі;
$a_j^{(L-1)}$ - вихід j-го нейрона у попередньому шарі;
$b_i^{(L)}$ - зсув для i-го нейрона у вихідному шарі.

5. Функція активації (Softmax):

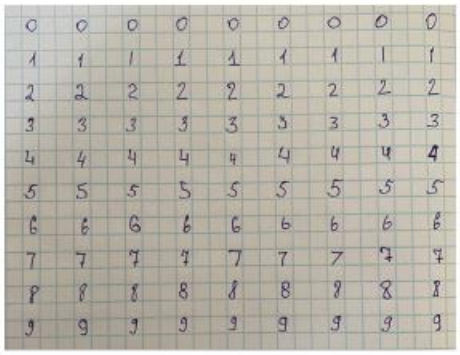
$$y_i = \frac{e^{z_i}}{\sum_p e^{z_p}}, \quad (8)$$

де y_i – вихід i-го нейрона після застосування функції;
e – число Ейлера;
z_i – вагова сума для i-го нейрона;
p – кількість класів, для яких визначаються ймовірності в рамках функції Softmax.

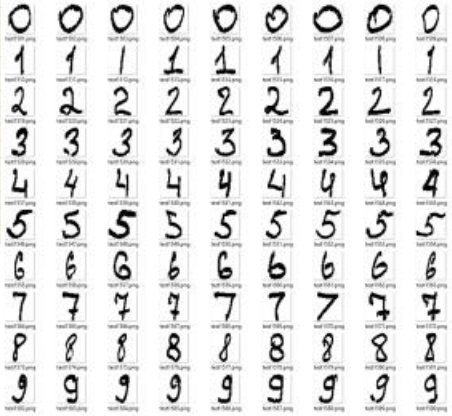
Вихідний повнозв'язний шар

6

НАБІР ДАНИХ ДЛЯ НАВЧАННЯ ТА ТЕСТУВАННЯ НЕЙРОННОЇ МЕРЕЖІ



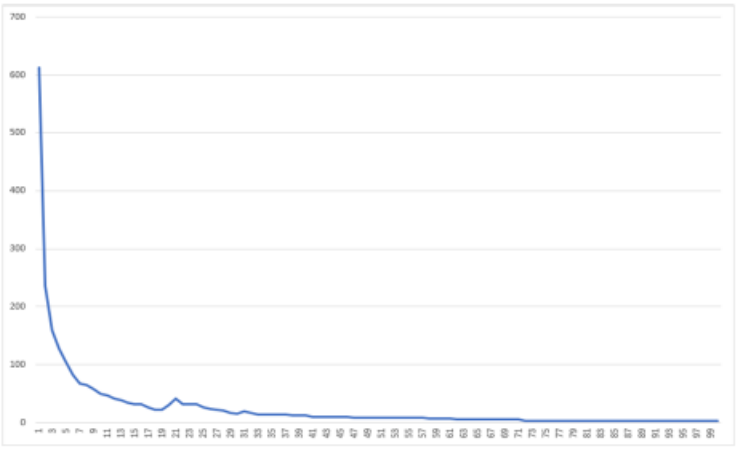
Вхідне зображення



Вилучений набір цифр

7

ГРАФІК ЗМІНИ КІЛЬКОСТІ ПОМИЛОК ВІД ЕПОХИ



Епохи	0-30	31-70	71-100
Швидкість навчання	0.01	0.002	0.0005

8

РЕЗУЛЬТАТИ ТЕСТУВАННЯ РОЗПІЗНАВАННЯ

№ зображення	Роздільна здатність зображення, пікселів	Кількість цифр	Кількість правильно розпізнаних цифр	Час, витрачений на розпізнавання, с
1	556*272	10	10	1.46
2	674*463	10	10	2.23
3	1280*1136	10	9	7.5
4	954*545	21	20	3.83
5	1280*850	23	23	7.52
6	1280*655	92	87	7.40

Точність - 95.78%

ПОРІВНЯННЯ З ІСНУЮЧИМИ СИСТЕМАМИ

Назва	Середня точність	Середній час на один символ, с
OCR Space	91,5%	0,14
Pen To Print	94,7%	0,45
Image To Text	92,5%	0,5
img2txt	74,8%	0,2
Створена система	95,78%	0,18

ВИСНОВКИ

1. Визначено критерії оцінювання продуктивності системи.
2. Описано алгоритм розпізнавання рукописних чисел.
3. Розроблено математичні моделі попередньої обробки зображень та згорткової нейронної мережі.
4. Реалізовано алгоритм пошуку межових прямокутників.
5. Розроблено та навчено модель нейронної мережі з практичним результатом у 95.78% точності розпізнавання

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Матвійчук А. М. Аналіз архітектур нейронних мереж для задачі розпізнавання рукописних цифр // Телекомунікаційні та інформаційні технології. №4, 2023. (прийнято до друку)

Тези доповідей:

1. Матвійчук А. М. Згорткові нейронні мережі: механіка та застосування. // І Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу україни і світу». – Київ: ДУТ, 2023. – (прийнято до друку)
2. Матвійчук А. М. Алгоритм обходу зображення для пошуку межових прямокутників з використанням обходу в ширину. // IV науково-практична конференція «Проблеми комп'ютерної інженерії». – Київ: ДУТ, 2023. – с. 123-125

ДЯКУЮ ЗА УВАГУ!