МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

ПЕЛЕХ ГЄРМАН ВАЛЕРІЙОВИЧ
ДОСЛІДЖЕННЯ АЛГОРИТМІВ КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ РОЗПІЗНАВАННЯ ТЕКСТУ В УМОВАХ РІЗНОЇ ОСВІТЛЕНОСТІ ТА ЯКОСТІ ЗОБРАЖЕННЯ

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Робота на здобуття освітнього ступеня «магістр»

Науковий керівник:
БУЛАТЕЦЬКИЙ ВІТАЛІЙ ВІКТОРОВИЧ,
кандидат фізико-математичних наук, доцент

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 20___ р.
Завідувач кафедри
(_____)_____
(підпис) ПІБ

ЛУЦЬК – 2024

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1. АЛГОРИТМИ КОМП'ЮТЕРНОГО ЗОРУ
ТА ЗАСОБИ РОЗПІЗНАВАННЯ ТЕКСТУ 6

1.1. Поняття та основні принципи алгоритмів комп'ютерного зору 6

1.1.1. Визначення та складові комп'ютерного зору 6

1.1.2. Алгоритми машинного навчання 7

1.1.3. Згорткові нейронні мережі 8

1.2. Машинне навчання нейронних мереж 9

1.2.1. Метод зворотного поширення помилки 9

1.2.2. Підготовка даних 14

1.2.3. Ініціалізація значень 15

1.2.4. Контроль процесу навчання нейронної мережі 16

1.3. Алгоритми розпізнавання тексту 18

1.3.1. Традиційна технологія оптичного розпізнавання тексту 18

1.3.2. Сучасні методи оптичного розпізнавання тексту 19

1.3.3. Набори даних для тренування мереж для розпізнавання тексту 22

1.4. Огляд існуючих засобів розпізнавання тексту 23

1.4.1. Архітектура Tesseract 2.0 23

1.4.2. EAST: An Efficient and Accurate Scene Text Detector 25

1.4.3. CRAFT: Character Region Awareness for Text Detection 27

1.4.4. FOTS: Fast Oriented Text Spotting with a Unified Network 29

РОЗДІЛ 2. РОЗРОБКА АЛГОРИТМУ РОЗПІЗНАВАННЯ ТЕКСТУ
В УМОВАХ РІЗНОЇ ОСВІТЛЕНОСТІ ТА ЯКОСТІ ЗОБРАЖЕННЯ 31

2.1. Постановка задачі 31

2.2. Методологія дослідження 32

2.3. Теоретичні аспекти дослідження 33
2.4. Обгрунтування вибору інструментальних засобів 34
2.5. Етапи програмної реалізації 36
2.5.1. Модель пошуку тексту на зображенні на основі CRAFT 36
2.5.2. Взаємодія з наборами даних 45
2.5.3. Модель-класифікатор для розпізнавання зображень символів 49
2.6. Організація тестування та налагодження програмного засобу 50
2.7. Аналіз отриманих результатів дослідження, рекомендації щодо використання та впровадження 52
ВИСНОВКИ 54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55
Додаток А 57
Додаток Б 59

ВСТУП

Актуальність теми зумовлена широким використанням алгоритмів комп’ютерного зору як наслідку значних досягнень у сфері нейронних мереж та машинного навчання.

Мета роботи полягає у дослідженні засобів комп’ютерного зору, а саме алгоритмів пошуку та розпізнавання тексту, а також способів використання машинного навчання та нейронних мереж для виконання цих завдань.

Завдання дослідження:

- проаналізувати тенденції розвитку та останні досягнення у сфері комп’ютерного зору;
- дослідити алгоритми машинного навчання нейронних мереж;
- дослідити особливості процесу тренування нейронних мереж-класифікаторів;
- проаналізувати вплив використання нейронних мереж на точність результатів передбачень засобів розпізнавання тексту;
- розглянути існуюче програмне забезпечення для розпізнавання тексту та порівняти його із методами, що використовувались раніше;
- розглянути типову архітектуру засобу для розпізнавання тексту з використанням нейронних мереж;
- розробити та навчити нейронну мережу для розпізнавання тексту на зображенні.

Об’єкт дослідження – растрові зображення різної освітленості і якості та методи видобування із них окремих текстових елементів.

Предмет дослідження – процес розробки та реалізації алгоритму розпізнавання тексту на зображенні з використанням нейронних мереж.

Практичне застосування результатів: результати дослідження можуть бути використані у навчальних цілях для розробки та вдосконалення власних засобів розпізнавання тексту, а також навчання нейронних мереж.

Апробація результатів роботи: тези дослідження “Використання нейронних мереж у засобах розпізнавання тексту на зображенні” були представлені на 2-й Міжнародній науково-практичній конференції “Scientific Research: Modern Innovations and Future Perspectives” 25-27 листопада 2024 року, м.Монреаль, Канада.

РОЗДІЛ 1.
АЛГОРИТМИ КОМП'ЮТЕРНОГО ЗОРУ ТА ЗАСОБИ РОЗПІЗНАВАННЯ ТЕКСТУ

1.1. Поняття та основні принципи алгоритмів комп'ютерного зору

1.1.1. Визначення та складові комп'ютерного зору

Комп'ютерний зір – це галузь штучного інтелекту, яка використовує машинне навчання та нейронні мережі для навчанні комп'ютерних систем видобуванню інформації з цифрових зображень, відео та інших візуальних вхідних даних [1].

Розробка способів, за допомогою яких машини можуть бачити і розуміти візуальні дані, триває понад 60 років. Експерименти у 1959 році по співвідношенню сигналів у мозку із видимим зображенням виявили, що він спочатку реагує на жорсткі краї або лінії, що з наукової точки зору означає, що обробка зображень починається з простих загальних форм.

Технологія оптичного розпізнавання символів (OCR) була представлена ще у 1974 році та могла розпізнавати текст, надрукований будь-яким шрифтом. Проте вона використовувала алгоритми, повністю створені людиною власноруч, що означало необхідність їх модифікації при найменшій зміні вхідних даних.

У 2001 році фокус досліджень перемістився на розпізнавання об'єктів, з'явилася перша програма для розпізнавання обличчя в реальному часі. Стандартизація тегів та анотацій зображень призвела до появи у 2010 році набору даних ImageNet. Він містить мільйони позначених зображень із тисячами категорій об'єктів і забезпечує основу для тренування сучасних згорткових нейронних мереж і моделей глибокого навчання. У 2012 році команда з університету Торонто представила модель під назвою AlexNet, що стало початком використання згорткових мереж для розпізнавання об'єктів на зображенні та зменшило рівень помилок до кількох відсотків.

1.1.2. Алгоритми машинного навчання

Перша складова комп'ютерного зору, машинне навчання – це галузь інформатики, яка зосереджена на використанні даних і алгоритмів для створення та поступового підвищення точності (навчання) штучного інтелекту, імітуючи спосіб навчання людей [2].

Алгоритм машинного навчання можна поділити на три основні кроки.

Прийняття рішення. Алгоритми машинного навчання використовуються для прогнозування чи класифікації. На основі деяких вхідних даних алгоритм машинного навчання видає оцінку їх відповідності певному шаблону.

Вимірювання похибки. Функція помилку оцінює відповідність прогнозу моделі очікуваному результату.

Оптимізація моделі. При наявності значущої похибки вагові коефіцієнти моделі коригуються, щоб зменшити розбіжність між відомою істиною і оцінкою моделі. Алгоритм оптимізації повторює цей ітеративний процес «оцінки та оптимізації», доки не буде досягнуто порогу точності.

Існує три види машинного навчання: контрольоване машинне навчання, неконтрольоване машинне навчання, машинне навчання з підкріпленням.

Контрольоване машинне навчання навчає моделі за допомогою класифікованих наборів даних. Алгоритм навчається відрізняти дані, позначені у наборі різними класами. Кероване машинне навчання є найпоширенішим типом, який використовується сьогодні.

Неконтрольоване машинне навчання це коли програма шукає шаблони у некласифікованих даних. Машинне навчання без нагляду може знаходити шаблони або тенденції, непомітні для людини.

Машинне навчання з підкріпленням навчає машини методом проб і помилок виконувати найкращі дії, встановлюючи систему винагороди. Навчання з підкріпленням може навчити модель грати у гру, автономні транспортні засоби – керувати, повідомляючи машині правильність прийнятого рішення, що допомагає їй з часом навчитися, які дії їй слід виконати [3].

1.1.3. Згорткові нейронні мережі

Другою складовою комп'ютерного зору є нейронні мережі, а саме один із класів глибоких штучних нейронних мереж – згорткові нейронні мережі [4].

Нейронна мережа – це структура, що складається з взаємопов'язаних обчислювальних вузлів або «нейронів», розташованих шарами. Ці вузли виконують математичні операції над вхідними даними, вивчаючи деякі базові закономірності в даних та виробляючи вихідні дані на основі цих закономірностей. Звичайна нейронна мережа має вхідний шар, що отримує на вхід початкові дані, приховані шари, де обробляються вхідні дані, і вихідний шар, що повертає результат обчислення.

Вихідні дані кожного нейрону проходять через функцію активації – це математична функція, застосована до виходу нейрона [5]. Це додає нелінійність у модель, дозволяючи мережі вивчати та представляти складні закономірності в даних. Без цієї функції нелінійності нейронна мережа поводилася б як модель лінійної регресії незалежно від того, скільки у ній шарів. Функція активації вирішує, чи потрібно активувати нейрон, шляхом обчислення зваженої суми вхідних даних і додавання зсуву. Це допомагає моделі приймати складні рішення та прогнози, додаючи нелінійність до виходу кожного нейрона. Типовими функціями активації є: сигмоїд, гіперболічний тангенс, ReLU (Rectified Linear Unit).

Сигмоїд характеризується S-подібною формою. Математично це визначається як A = 1/(1+e-x). Формула забезпечує плавні і неперервні значення, які є важливими для методів оптимізації на основі градієнта. Діапазон результатів від 0 до 1 корисний для бінарної класифікації.

Гіперболічний тангенс – зміщена версія сигмоїду, зазвичай використовується в прихованих шарах через значення, центровані навколо 0.

ReLU (Rectified Linear Unit), визначена формулою A(x)=max(0,x), є найпоширенішою для прихованих шарів згорткових мереж. ReLU є менш затратною в обчислювальному плані, оскільки передбачає простіші математичні операції.

У згортковій нейронній мережі приховані шари включають один або кілька шарів дискретної згортки. Зазвичай це шар, який виконує скалярний добуток ядра згортки на вхідну матрицю шару. Коли ядро згортки ковзає вздовж вхідної матриці шару, операція згортки генерує карту ознак, що через функцію активації передається до наступного шару. Далі слідують інші шари об'єднання та нормалізаційні шари.

У порівнянні з іншими алгоритмами класифікування зображень згорткові мережі потребують найменше попередньої обробки. Мережа сама навчається фільтрів, які в традиційних алгоритмах конструювали вручну, що є її значною перевагою.

1.2. Машинне навчання нейронних мереж

1.2.1. Метод зворотного поширення помилки

Зворотне розповсюдження помилок є найпоширенішим способом навчання нейронної мережі. Це схема контрольованого машинного навчання, тобто мережа тренується на позначених навчальних даних.

Нейронна мережа складається з нейронів різних шарів; вхідний шар, проміжний прихований шар(и) і вихідний шар. З'єднання між вузлами сусідніх шарів мають пов'язані з ними «ваги». Мета навчання — призначити правильні ваги цим зв'язкам. У контрольованому навчанні для деяких заданих вхідних даних ми знаємо бажаний/очікуваний результат [6]. За заданих вхідних даних мережі ці ваги визначають, яким є вихідний вектор.

Спочатку всі ваги з'єднань призначаються випадковим чином. Для кожного набору навчальних даних активується нейронна мережа та спостерігається її результат. Цей результат порівнюється з бажаним, який ми вже знаємо, і похибка передається назад на попередній рівень. Ваги коригуються відповідно до цієї похибки, і цей процес повторюється, доки вихідна похибка не стане нижчою за заздалегідь визначене порогове значення.

Після завершення вищезазначеного алгоритму ми маємо навчену нейронну мережу, яка вважається готовою до роботи з новими даними.

Проілюструємо процес навчання методом зворотного поширення помилки на прикладі простої нейронної мережі (Рис. 1.1) [7].

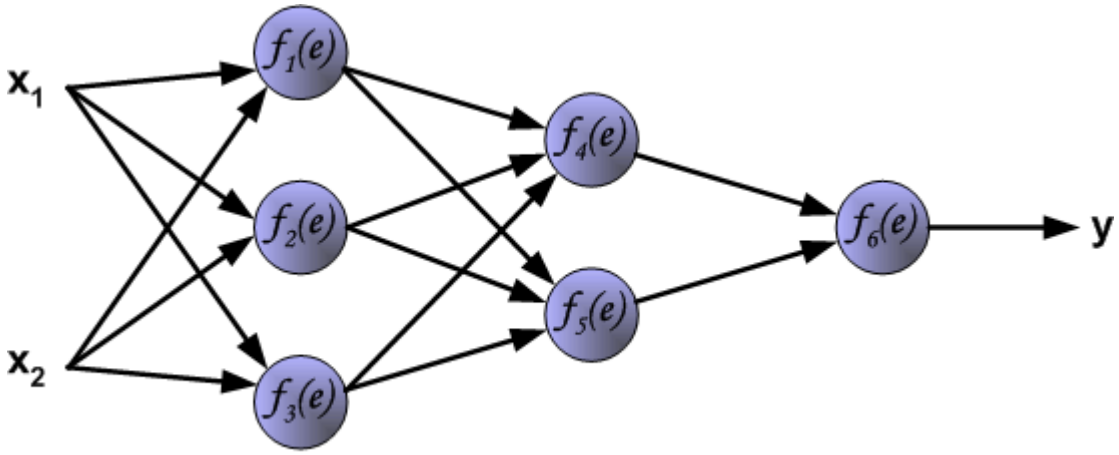


Рисунок 1.1 – Структура простої нейронної мережі

Кожен нейрон представляє зважену суму попередніх, проведену через функцію активації (Рис. 1.2).

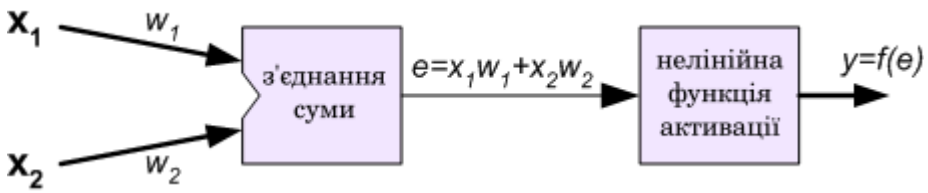


Рисунок 1.2 – Структура нейрона

Перший цикл обчислень проводиться із випадковими значеннями. По черзі обчислюються вихідні значення кожного нейрона у кожному шарі, починаючи від вхідного (Рис. 1.3). Функції активації нейронів у різних шарах можуть відрізнятись.

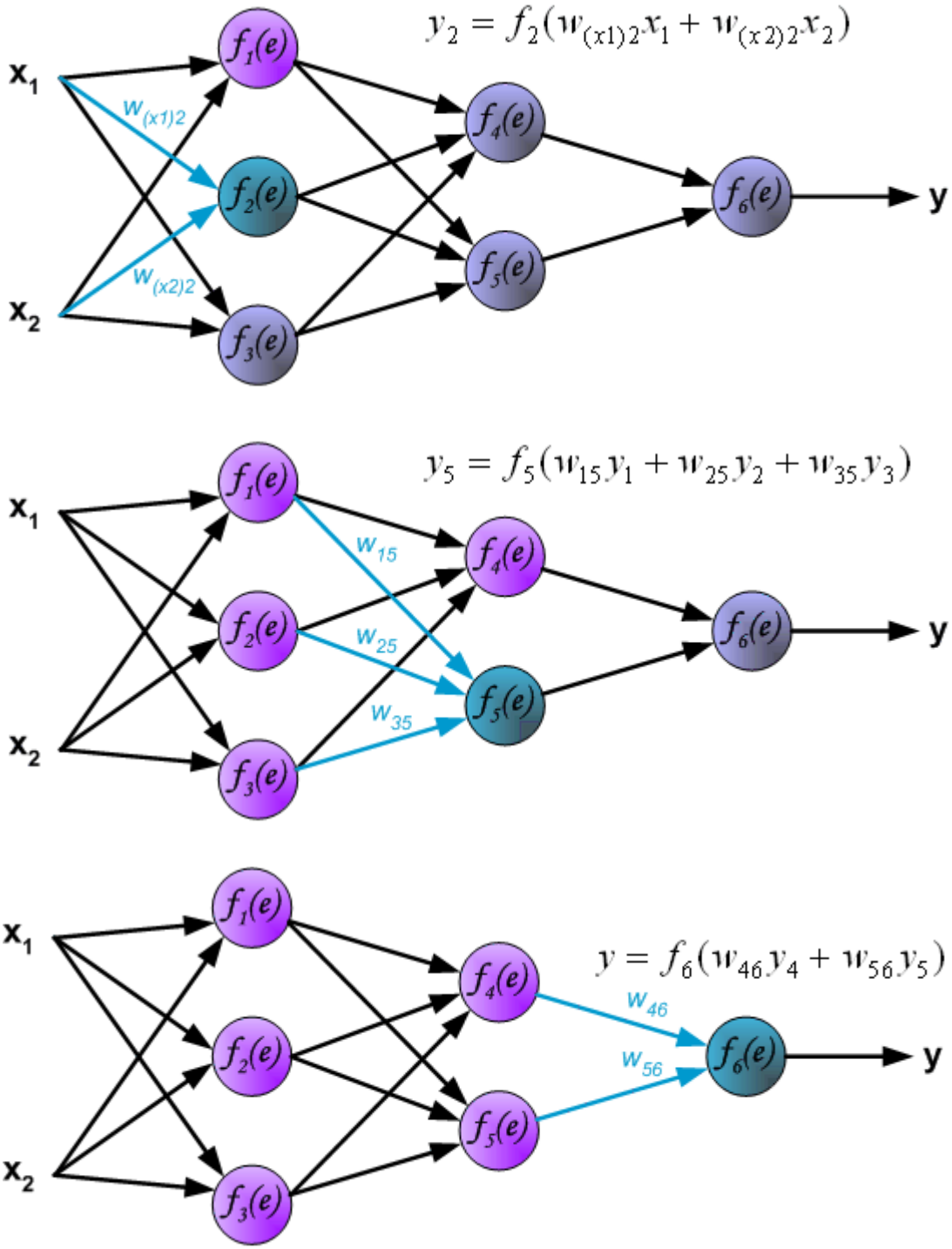


Рисунок 1.3 – Кроки обчислень значень кожного нейрона мережі

Результат на виході нейронної мережі y, згідно з принципом роботи контрольованого машинного навчання, порівнюється із бажаним значенням z через знаходження помилки δ (похибки). Знайдена похибка поширюється у зворотному напрямку пропорційно до ваги кожного зв'язку (Рис. 1.4).

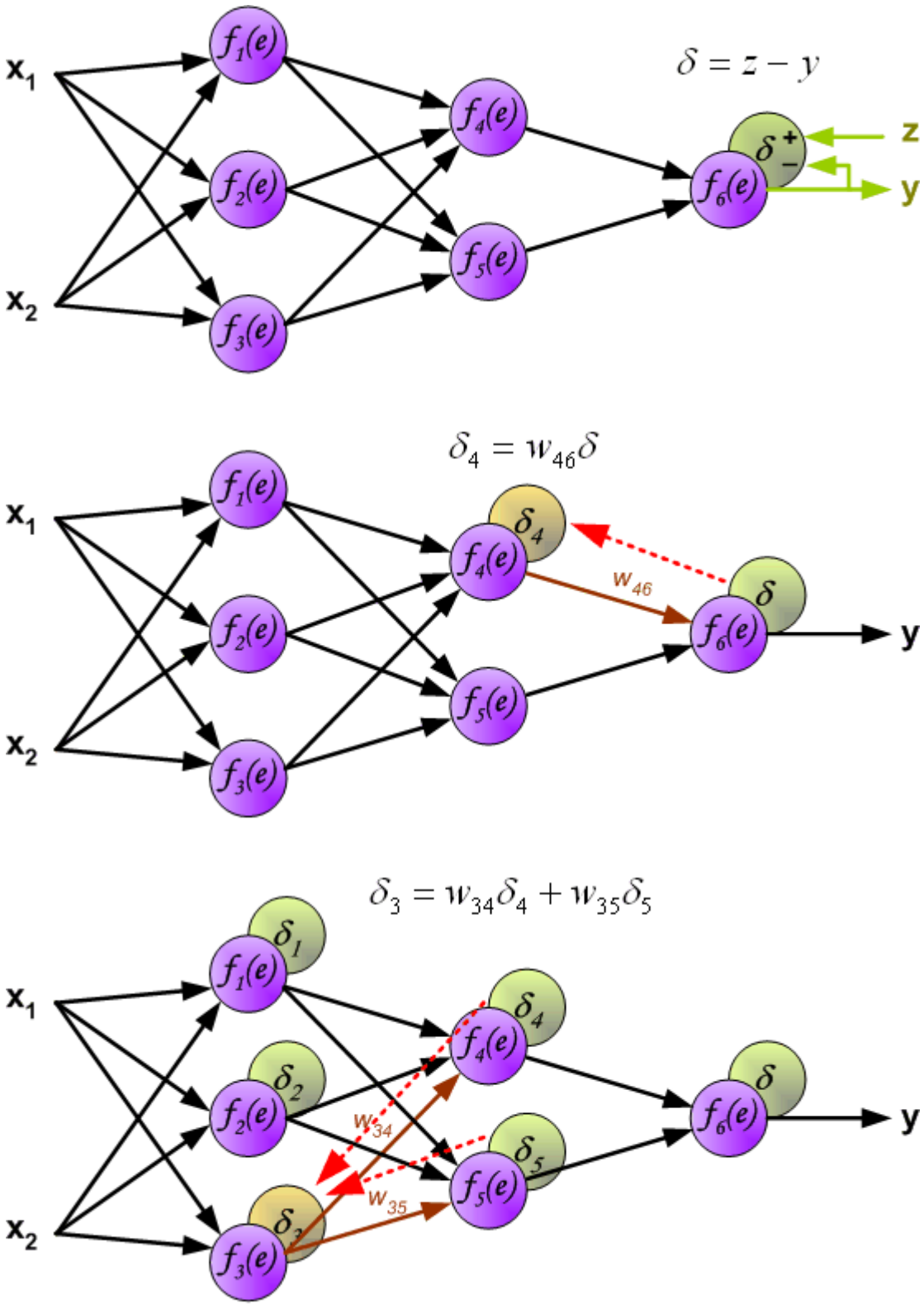


Рисунок 1.4 – Обчислення значення похибки для кожного нейрона

Після цього ваги усіх зв’язків коригуються відповідно до знайдених похибок. Вплив кожної ваги на похибку обчислюється за допомогою часткової похідної по значенню ваги *e* функції активації f_n відповідного нейрона (Рис. 1.5).

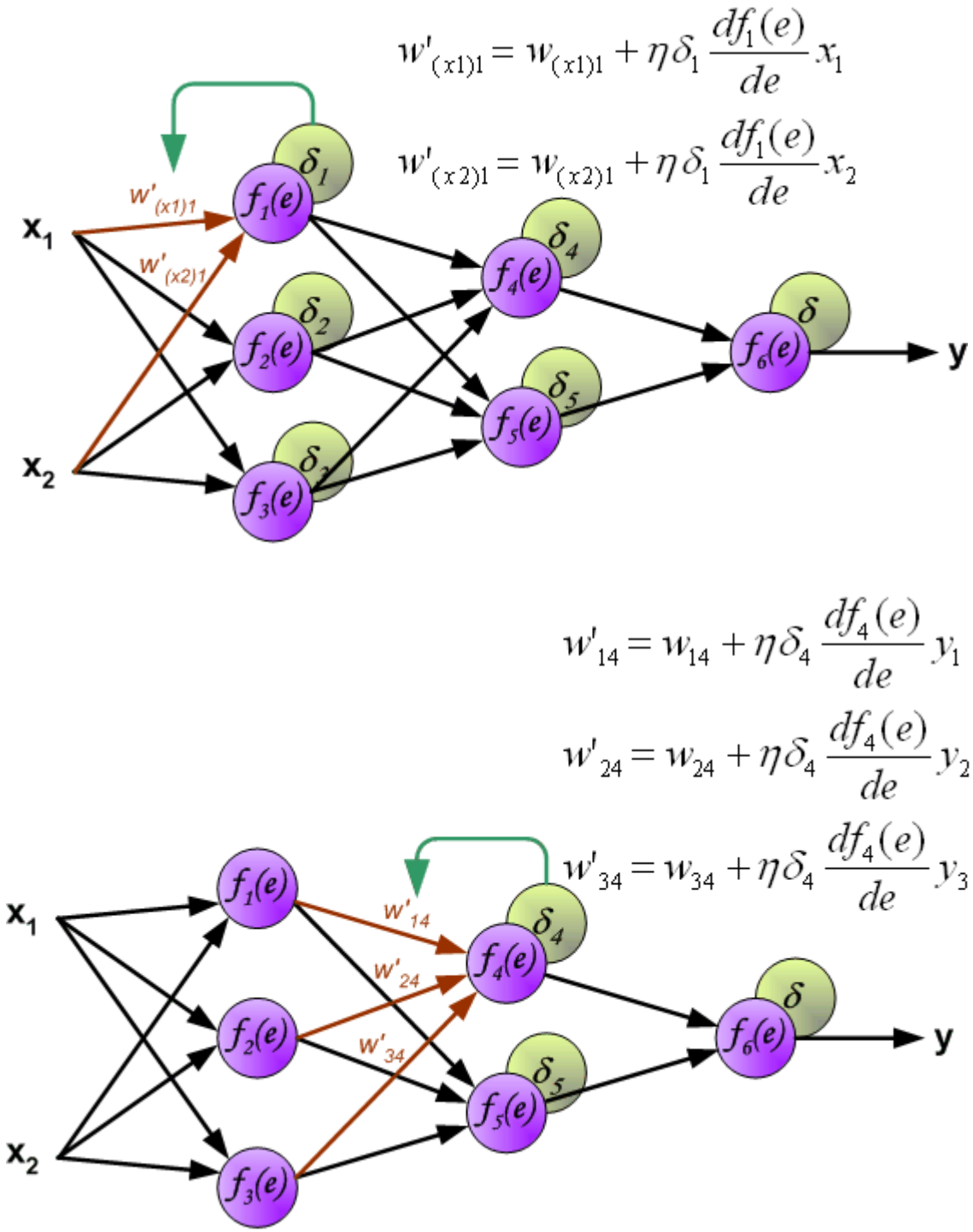


Рисунок 1.5 – Обчислення значення градієнту (зміни) для кожної ваги кожного нейрону

Параметр η є так званим гіпер-параметром, що коригує швидкість навчання мережі, та значення якого визначається заздалегідь. Результатом кожної ітерації є зменшення похибки вихідних даних мережі.

1.2.2. Підготовка даних

Навчання нейронної мережі для завдань комп'ютерного зору потребує великої кількості впорядкованих позначених даних (датасетів). Ці дані в свою чергу потребують попередньої обробки для зменшення шумів та надлишкових значень, що лише перешкоджають навчанню та перевантажують нейронну мережу (Рис. 1.6).

Віднімання медіани є найпоширенішою формою попередньої обробки. Віднімання медіани окремо для кожної ознаки даних в геометричному відображенні центрує хмару даних навколо початку координат вздовж кожного виміру.

Нормалізація означає зміну даних таким чином, щоб вони мали приблизно однаковий масштаб по кожній ознаці. Існує два поширених способи досягнення цієї нормалізації:

- розділити значення кожного виміру (ознаки) на його середнє квадратичне відхилення після того, як воно було відцентроване відносно нуля (шляхом віднімання медіани);
- масштабування кожної ознаки таким чином, щоб її значення належали проміжку [-1; 1].

У випадку обробки зображень відносні масштаби пікселів уже приблизно однакові (у діапазоні від 0 до 255), тому немає суворої необхідності виконувати цей додатковий етап попередньої обробки.

Важливо зауважити, що будь-яку статистику попередньої обробки (наприклад, медіану) потрібно обчислювати лише на даних навчання, а потім застосовувати до даних перевірки або тестування.

1.2.3. Ініціалізація значень

Для початку процесу навчання мережі необхідно встановити значення ваги кожного зв'язку.

Остаточне значення кожної ваги в навченій мережі невідоме, але за умови належної нормалізації даних розумно припустити, що приблизно половина ваг

буде додатною, а половина – від'ємною. Встановлення всіх початкових ваг рівними нулю як найкраще припущення є помилкою, тому що якщо кожен нейрон у мережі має однакове вихідне значення, то всі вони також матимуть однакові градієнти під час зворотного поширення та проходитимуть абсолютно однакові оновлення параметрів. Іншими словами, джерело асиметрії між нейронами відсутнє, якщо їх ваги ініціалізовані як однакові.

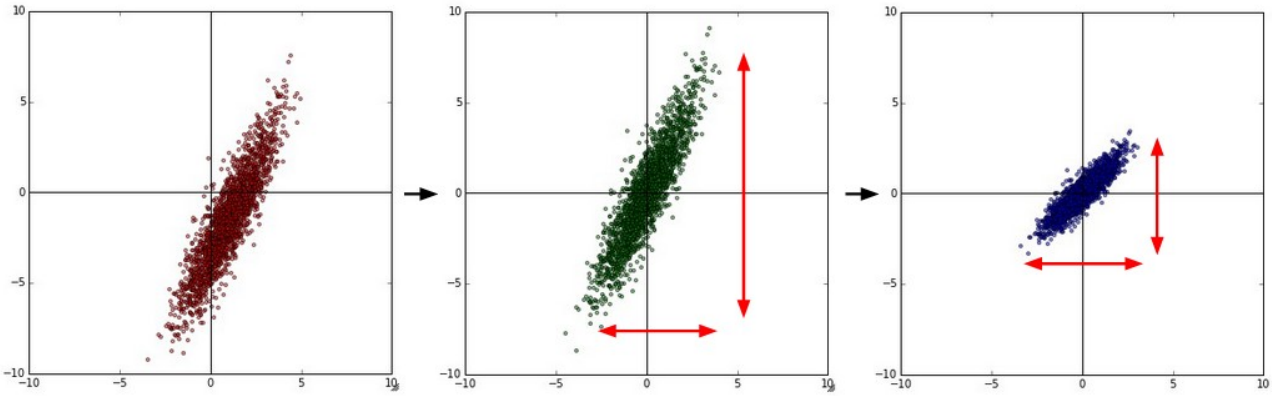


Рисунок 1.6 – Кроки нормалізації даних: віднімання медіани та нормалізація

Отже бажано, щоб ваги були дуже близькими до нуля, але не дорівнювали нулю. Всі нейрони повинні бути випадковими та унікальними на початку, тоді вони обчислюватимуть різні оновлення ваги зв'язків та інтегруватимуться як різні частини повної мережі. Менші числа не завжди працюватимуть краще. Шар нейронної мережі, який має дуже малі ваги, під час зворотного поширення повертатиме дуже малі градієнти, що може стати проблемою для глибоких мереж.

Окрім вхідних даних з попереднього шару, кожен нейрон має додаткове значення зміщення (біас). Його значення для зручності зазвичай дорівнює 1, а вплив залежить лише від відповідної ваги. Завдяки цьому зміщенню, що не залежить від вхідних даних, навіть при 0 на вході мережа не буде завжди даватиме нульову оцінку незалежно від ваг.

Зміщення можливо ініціалізувати нулем, оскільки порушення асиметрії забезпечується малими випадковими числами у вагових коефіцієнтах. Для нелінійності ReLU можна використовувати невелике постійне значення, наприклад 0,01, для всіх зміщень, оскільки це гарантує, що всі одиниці ReLU запускаються на початку, а отже, отримують і поширюють певний градієнт. Однак це не гарантує ніяких покращень, і більш поширеним варіантом є просто використання ініціалізації зміщення 0.

1.2.4. Контроль процесу навчання нейронної мережі

Основною величиною, яку слід відстежувати під час навчання мережі, є значення функції похибки. Значення функції бажано вимірювати не лише на навчальних даних, але і на окремому тестовому наборі даних. Порівняння двох величин допомагає запобігти вивченню мережею ознак, специфічних лише для тренувального набору даних, внаслідок чого вона працюватиме гірше з іншими вхідними даними. Існує кілька способів запобігання надмірного підлаштування мережі під конкретний набір даних.

Регуляризація L2 (другого рівня) – найпоширеніша форма регуляризації. Вона додає до похибки квадратичну величину всіх параметрів мережі. Тобто за кожну вагу w у мережі до похибки додамо доданок $\lambda w^2/2$, де λ є силою регуляризації. Зазвичай множник ½ додають, щоб градієнт цього доданка відносно параметра w був просто λw замість $2\lambda w$.

Регуляризація L2 має інтуїтивно зрозумілу інтерпретацію збільшеного штрафу за великі одиничні ваги та надає перевагу меншим розподіленим значенням. Це заохочує мережу використовувати всі свої вхідні дані рівномірно, без надання значної переваги окремим фрагментам. Під час оновлення параметрів використання регуляризації L2 означає, що кожна вага зменшується лінійно на λw.

Інша форма регуляризації полягає в застосуванні абсолютної верхньої межі величини вектора ваги для кожного нейрона. На практиці це відповідає виконанню оновлення параметра як зазвичай, а потім застосуванню обмеження

величин вектора ваги. Однією з властивостей даного методу є те, що мережа не може «вибухнути», навіть якщо встановлена надто велика швидкість навчання, оскільки оновлення завжди обмежені.

Відсівання (Dropout) є надзвичайно ефективною та простою технікою регуляризації, яка доповнює інші методи (Рис. 1.7). Під час навчання відсівання реалізується шляхом утримання нейрона активним лише з деякою ймовірністю p (гіперпараметр) або встановлення значення його виходу рівним нулю.

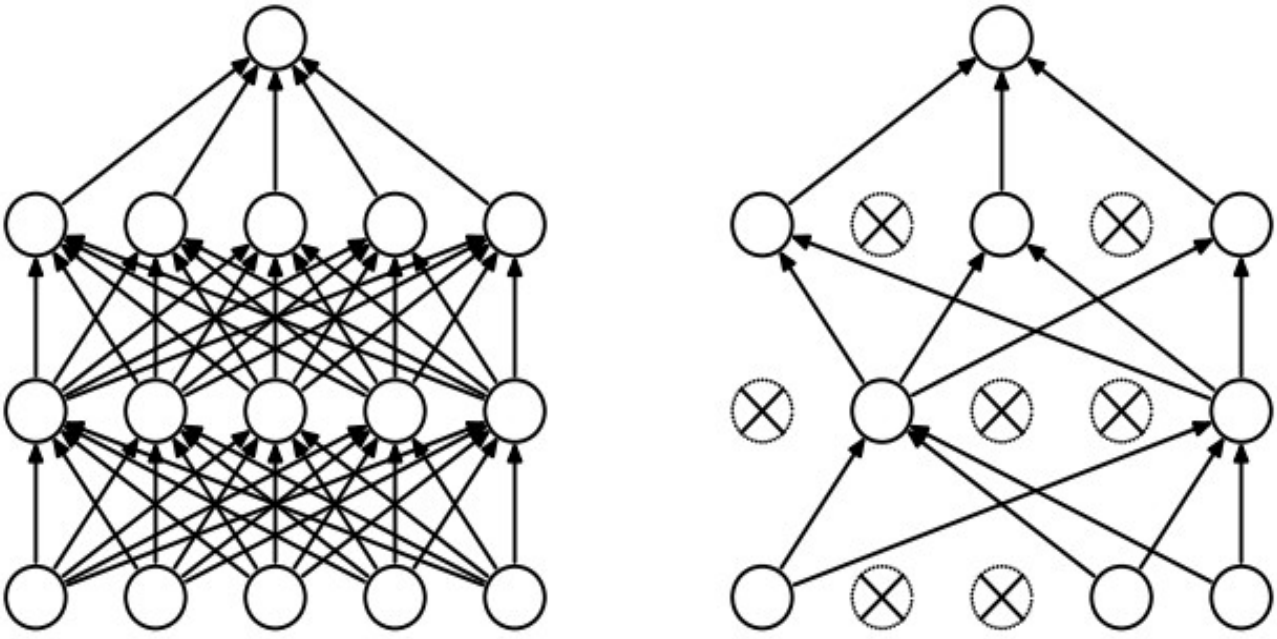


Рисунок 1.7 – Техніка відсівання

Небажаною властивістю представленої вище схеми є те, що ми повинні масштабувати значення у p разів під час тестування для зберігання значень. Оскільки продуктивність під час тестування є надзвичайно важливою, краще використовувати інвертоване відсівання, яке виконує масштабування під час тренування (домножуючи на $1/p$), залишаючи незмінними значення під час тестування.

1.3. Алгоритми розпізнавання тексту

1.3.1. Традиційна технологія оптичного розпізнавання тексту

Технологія оптичного розпізнавання символів (OCR), яка також називається розпізнаванням тексту, перетворює видобуті дані з документів, PDF-файлів і зображень у редагований текст. Компанії використовують технологію оптичного розпізнавання символів щоб скоротити, якщо не виключити, ручне введення даних у своїх робочих процесах.

Однак традиційне програмне забезпечення OCR обмежене технологіями свого часу. Програмне забезпечення було розроблено для видобування тексту з чорно-білих друкованих документів, що відрізняється від сучасних реалій.

Найчастіше програмне забезпечення OCR витягує дані із тексту на зображеннях. Однак зростання кількості сканованих документів із різними форматами, шрифтами, стилями та кольорами призвело до кількох обмежень OCR.

Точність. Як правило, удосконалене програмне забезпечення для оптичного розпізнавання символів має точність 99%, за умови, що вхідним матеріалом є високоякісне чорно-біле зображення з великими шрифтами. Однак рівень точності часто знижується, коли програмне забезпечення для обробки документів має справу з рукописним вмістом, складними макетами або перекошеними текстами. Програмне забезпечення оптичного розпізнавання символів також генерує неправильні показання з дрібних текстів і зображень низької якості. Ці неточності впливають на загальну якість і цілісність отриманих даних.

Підтримка мов та шрифтів. Платформа оптичного розпізнавання символів використовує алгоритми розпізнавання шаблонів, щоб зіставити відскановані тексти та символи з тими, які є в її базі даних. Звичайно, система генерує неточні показання, коли зустрічає шрифти або мови, які відрізняються від попередньо зафіксованих параметрів. Оскільки алгоритм не є адаптивним, він може не визначити або неправильно інтерпретувати унікальні мовні символи.

Отже, організаціям важко використовувати технологію OCR для ефективної багатомовної та різноманітної обробки документів.

Помилки форматування. Прості програми оптичного розпізнавання символів не зберігають вихідне форматування документа. Без допоміжного програмного забезпечення сторонніх розробників технологія оптичного розпізнавання тексту важко обробляє розриви рядків, стилі шрифтів, відступи, таблиці та графіки. Як наслідок, створений документ містить неправильно вирівняний текст, помилкові інтервали, неправильні розриви рядків і незрозумілі таблиці. Ці помилки форматування усуваються працівниками вручну, знижуючи продуктивність і додаючи непотрібного навантаження на робочу силу.

Залежність від якості зображення. Продуктивність програмного забезпечення OCR залежить від якості вихідних зображень або документів. Зображення з низькою роздільною здатністю, блідий текст або погані умови освітлення можуть створювати шум, який перешкоджає точному розпізнаванню символів. Розмиті або спотворені зображення можуть спричинити неправильне тлумачення символів програмним забезпеченням, що призведе до помилок транскрипції та вимагає втручання вручну для виправлення розбіжностей.

1.3.2. Сучасні методи оптичного розпізнавання тексту

Глибоке навчання нейронних мереж призвело до значного підвищення точності та продуктивності оптичного розпізнавання текстів [8]. Існує кілька підходів до розпізнавання на основі глибокого навчання.

Згорткові нейронні мережі (CNN) часто використовуються для розпізнавання тексту на основі зображень. Вхідне зображення обробляється згортковими шарами, які виділяють об’єкти та вивчають представлення тексту. Вихідні дані CNN потім передаються до рекурентної нейронної мережі (RNN) для подальшої обробки та розпізнавання тексту.

Рекурентні нейронні мережі (RNN) широко використовуються для розпізнавання тексту на основі послідовності, наприклад розпізнавання рукописного тексту та мовлення. RNN використовують цикли зворотного зв’язку

для обробки послідовних даних, що дозволяє їм отримувати довгострокові залежності та контекстну інформацію.

Мережі кодера-декодера використовуються для наскрізного розпізнавання тексту. Вхідне зображення спочатку кодується у вектор ознак, а потім декодується у послідовність символів або слів. Ці мережі можна навчати наскрізно, підвищуючи ефективність і точність.

Глибокі згорткові нейронні мережі (DCNN) і RNN є двома популярними типами нейронних мереж, які використовуються в глибокому навчанні. І DCNN, і RNN показують вражаючу продуктивність у різних завданнях, включаючи розпізнавання зображень і тексту. Однак вони відрізняються своєю архітектурою і функціональністю. Наведемо деякі важливі відмінності між DCNN і RNN.

Архітектура. DCNN зазвичай розроблені для завдань на основі зображень і мають архітектуру зі зворотним зв’язком, яка включає шари згортки та кластеризації для видобування ознак із вхідного зображення. RNN у свою чергу розроблені для завдань на основі послідовності та мають рекурентну архітектуру з циклом зворотного зв’язку для обробки даних послідовності.

Пам’ять. На відміну від DCNN, RNN мають компонент пам’яті, який дозволяє їм зберігати інформацію про попередні вхідні дані та послідовно обробляти дані. Це робить RNN краще придатними для завдань, пов’язаних з обробкою послідовностей даних, таких як розпізнавання мови або моделювання мови.

Вхідний розмір. DCNN розроблені для обробки вхідних кадрів фіксованого розміру, тоді як RNN можуть обробляти вхідні послідовності змінної довжини. Це робить RNN більш гнучкими та краще підходить для завдань, що включають послідовності змінної довжини, наприклад, обробку природної мови або розпізнавання мовлення.

Згорткові рекурентні нейронні мережі (CRNN) – архітектура нейронної мережі, яка поєднує в собі переваги CNN і RNN.

CRNN зазвичай використовуються для обробки та класифікації послідовних даних, таких як мова, текст і зображення. Їхня здатність обробляти

послідовні дані змінної довжини та фіксувати довготривалі залежності робить їх особливо ефективними в завданнях, які вимагають розуміння та моделювання контекстної та часової інформації. CRNN — це потужні інструменти для моделювання та обробки послідовних даних, які демонструють максимальну продуктивність у різноманітних завданнях.

CRNN працюють наступним чином:

- вхідні дані (вхідними даними для CRNN є послідовність даних, наприклад зображень або зразків аудіо);

- згорткові шари (вхідна послідовність передається згортковим шарами, які витягують із вхідних даних ознаки, а ці шари подібні до тих, що використовуються в CNN, і особливо ефективні для вхідних даних на основі зображень);

- рекурентні шари (вихідні дані згорткового шару потім подаються одному або кільком повторюваним шарам, які особливо ефективні для обробки послідовних даних; повторювані шари зберігають прихований стан, який фіксує інформацію про попередні записи в послідовності);

- зв’язки між згортковим і рекурентним шарами (вихідні дані згорткового шару зазвичай фільтруються перед подачею на рекурентний шар, що зменшує обчислювальну складність мережі, зберігаючи основні вхідні характеристики);

- вихід (вихід останнього ітераційного шару проходить через повністю з’єднаний кінцевий шар, який повертає прогноз для вхідної послідовності, і це передбачення може являти собою послідовність символів, слів або інших вихідних даних, що стосуються завдання).

Хоча CRNN мають багато переваг, їх використання також створює певні проблеми. Наведемо деякі з основних проблем CRNN:

Висока обчислювальна складність. CRNN потребують більшої кількості обчислень, особливо порівняно з більш простими моделями, такими як CNN. Це може ускладнити навчання та розгортання на малопотужних пристроях, таких як смартфони чи вбудовані системи.

Складний архітектурний дизайн. CRNN вимагає ретельного проектування згорткових і повторюваних шарів та їх комбінацій. Вибір зразкової архітектури може бути тривалим і важким процесом.

Складність навчання. Навчання CRNN може бути складним завданням, особливо під час роботи з великими наборами даних. Ці моделі можуть страждати від таких проблем, як надмірне підлаштовування (overfitting), коли вони навчаються надто специфічних ознак навчальних даних і погано узагальнюють нові дані.

Обмежена інтерпретація. Як і інші моделі глибокого навчання, CRNN може бути важко інтерпретувати та зрозуміти, чому модель робить конкретні прогнози.

1.3.3. Набори даних для тренування мереж для розпізнавання тексту

Оскільки необроблені дані та неструктуровані дані не можна безпосередньо вводити в моделі машинного навчання, використовується попередня обробка даних, щоб зробити їх придатними для використання [9]. Зазвичай перший крок до запуску проекту машинного навчання – це переконатися, що дані, які використовуються для навчання моделі, правильно відформатовані та відфільтровані. Продуктивність моделі буде настільки ж високою, як і дані, за допомогою яких вона буде навчена. Навіть найскладніші алгоритми можуть давати погані результати або бути неточними та упередженими, якщо їх навчати на неправильно оброблених даних.

Існує значна кількість наборів даних, доступних для вільного користування, що можуть послужити відправною точкою у проектуванні нейронної мережі для перевірки її працездатності [10].

Набір даних MNIST (Modified National Institute of Standards and Technology) є одним із найбільш широко використовуваних для тестів у дослідженнях OCR. Він складається із чорно-білих зображень розміром 28x28 із рукописними цифрами (від 0 до 9) та їхніми відповідними мітками. Хоча цей набір в основному

використовується для розпізнавання цифр, MNIST служить чудовою відправною точкою для початківців OCR завдяки своїй простоті та доступності.

Набір даних SVT (Street View Text) розроблений для розпізнавання тексту у складних умовах, імітуючи сценарії реального світу, де текст фіксується в природному середовищі, як-от вуличні вивіски чи вітрини магазинів. Набір даних містить зображення тексту сцени разом із відповідними анотаціями, забезпечуючи складний і практичний ресурс для навчання OCR.

Набір даних UNLV-ISRI-ALPR зосереджений на автоматичному розпізнаванні номерних знаків. Він містить зображення номерних знаків із примітками, що дозволяє OCR-моделям точно розпізнавати буквено-цифрові символи.

1.4. Огляд існуючих засобів розпізнавання тексту

1.4.1. Архітектура Tesseract 2.0

Tesseract – це механізм OCR з відкритим вихідним кодом, розроблений в HP (Hewlett Packard) між 1984 і 1994 роками. Версія програми 2.0 добре представляє досягнення у сфері розпізнавання тексту до початку використання нейронних мереж. Його вхідними даними є двійкове зображення з можливістю додатково вказати області тексту, на виході отримуємо текст із зображення.

Першим кроком є аналіз з’єднаних контурів, у яких зберігаються фігури компонентів. На той час це було обчислювально дорогим проектним рішенням, але воно мало значну перевагу: шляхом перевірки вкладеності і кількості дочірніх контурів легко виявити інвертований текст. На цьому етапі контури збираються у скупчення, скупчення організовуються в текстові рядки, а рядки та області аналізуються на наявність фіксованого кроку або пропорційності тексту. Рядки тексту розбиваються на слова залежно від міжсимвольного інтервалу. Текст із фіксованим кроком одразу нарізається на клітинки символів. Пропорційний текст розбивається на слова за допомогою пробілів.

Саме розпізнавання відбувається як двопрохідний процес: під час першого проходу робиться спроба розпізнати кожне слово по черзі. Кожне задовільне слово передається в адаптивний класифікатор як навчальні дані. Адаптивний класифікатор отримує можливість точніше розпізнавати текст нижче сторінки. Оскільки адаптивний класифікатор може дізнатись щось корисне занадто пізно, виконується другий прохід по сторінці, під час якого слова, які не були розпізнані достатньо добре, розпізнаються знову. Остання фаза розв'язує нечіткі пробіли та перевіряє альтернативні гіпотези для висоти, щоб знайти текст малими літерами.

Якщо результат зі слова є незадовільним, Tesseract намагається покращити результат, нарізаючи скупчення з найгіршою ймовірністю з класифікатора символів. Відсічки виконуються в пріоритетному порядку. Будь-яке відсікання, яке не покращує достовірність результату, скасовується, але не відкидається повністю, щоб у разі потреби асоціатор міг повторно використати цей розріз пізніше. Коли потенційні розрізи вичерпано, якщо слово все ще недостатньо ймовірно класифіковано, воно передається асоціатору. Асоціатор виконує пошук можливих комбінацій об'єднання максимально розрізаних скупчень у символи-кандидати.

Під час навчання сегменти багатокутної апроксимації використовуються для ознак, але під час розпізнавання ознаки невеликої фіксованої довжини (у нормалізованих одиницях) витягуються з контуру та порівнюються з об'єктами прототипів із даних навчання. Таким чином, ознаки з невідомого є тривимірними (координати x, y, кут нахилу), зазвичай із 50-100 характеристиками в символі, а ознаки відомого прототипу є 4-вимірними (x, y, кут, довжина), зазвичай із 10-20 характеристиками на прототип.

Класифікація відбувається як двоетапний процес. На першому кроці відсіювач класів створює короткий список класів, яким може відповідати невідомий символ. Потім кожна ознака невідомого шукає прототипи обраних класів, яким вона може відповідати, і лише тоді обчислюється фактична подібність між ними.

Оскільки класифікатор здатний легко розпізнавати пошкоджені символи, він не навчався на пошкоджених символах, що суттєво відрізнялось від інших опублікованих класифікаторів того часу. Оскільки статичний класифікатор має добре узагальнювати будь-який тип шрифту, його здатність відрізняти символи від сторонніх об’єктів послаблюється. Більш чутливий до шрифту адаптивний класифікатор, який навчається на виході статичного класифікатора зазвичай використовується для отримання більшої точності в кожному окремому документі, де кількість шрифтів обмежена [11].

1.4.2. EAST: An Efficient and Accurate Scene Text Detector

EAST – алгоритм виявлення тексту на основі глибокого навчання, який використовує єдину нейронну мережу для виявлення тексту в природному середовищі та повертає результат у вигляді чотирикутної форми тексту[12]. Він використовує алгоритм Non-Max Suppression (NMS) разом із згортковою мережею. Згорткова мережа використовується для виявлення слова на зображенні, а NMS використовується для об’єднання всіх виявлених текстів/полів в один великий текст або поле. Нейронна мережа навчена безпосередньо передбачати текст і геометрію тексту в природній сцені. Цей підхід виводить щільне попіксельне передбачення текстів [13].

Спочатку зображення надсилається до повністю згорткової мережі (FCN), у якій генеруються піксельні текстові і геометричні карти. Виділення тексту на картах доступне двома способами: RBOX (координати області та кут повороту) і QUAD (координати чотирьох кутніх точок області). Ці область передаються далі до алгоритму NMS що об’єднує області та обирає ті, що найповніше охоплюють текст.

Нейронну мережу EAST можна поділити на три частини: видобування ознак, поєднання ознак, та шар виводу (Рис. 1.8).

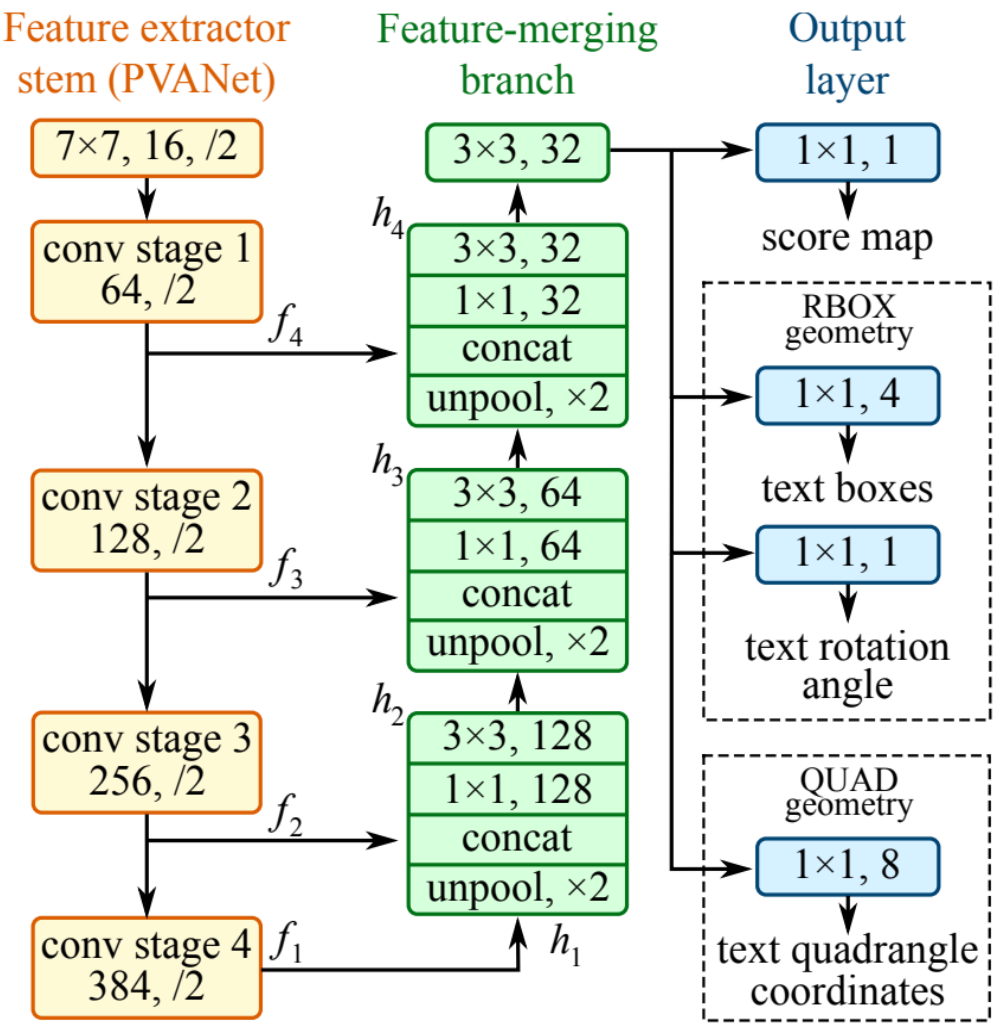


Рисунок 1.8 – Структура нейронної мережі EAST

Для видобування ознак за основу взято мережу PVANet – спрощену архітектуру видобування ознак для виявлення об’єктів [14]. PVANet використовує так звані inception-модулі, що включають поєднання кількох паралельних операцій згортки з різними параметрами (Рис. 1.9) [15]. Процес видобування ознак разом з їх пошаровим об’єднанням утворює U-подібну мережеву архітектуру для виявлення ознак різного розміру на зображенні.

EAST доступний для використання у останніх версіях бібліотеки для розробки систем комп’ютерного зору OpenCV.

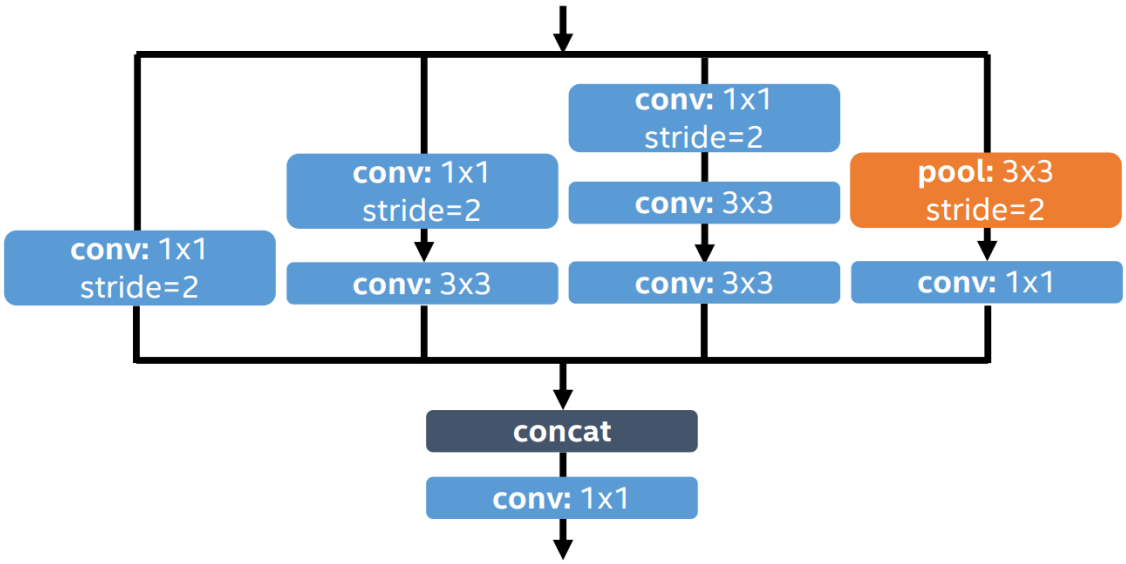


Рисунок 1.9 – Структура inception-модуля

1.4.3. CRAFT: Character Region Awareness for Text Detection

Більшість методів розпізнавання тексту обирають одиницею слово, проте визначення меж слова не завжди можна провести однозначно, оскільки слова можна розділяти за різними критеріями, такими як значення, пробіли або колір. Словесний сегмент у контексті комп'ютерного зору не має чіткого семантичного значення. Ця двозначність у позначенні слів у наборах даних розмиває значення істинності навчальних наборів даних.

CRAFT – це згорткова нейронна мережа, яка на відміну від EAST, повертає на виході оцінки областей і оцінки спорідненості. Оцінка областей використовується для локалізації окремих символів на зображенні, а оцінка спорідненості використовується для групування кожного символу в один екземпляр. Щоб компенсувати відсутність анотацій на рівні символів, CRAFT використовує слабо-контрольоване машинне навчання, яке видобуває значення на рівні символів із існуючих реальних наборів даних на рівні слів. Використовуючи розпізнавання регіону на рівні символів, CRAFT легко знаходить тексти різних форм.

На вхід нейронна мережа отримує оригінальне зображення із трьома кольоровими каналами. На виході мережа повертає дві матриці з вдвічі меншою

роздільною здатністю: мапу областей із символами та мапу із областями між символами або карту їх спорідненості (Рис. 1.10).

Рисунок 1.10 – Мапи областей та спорідненості у мережі CRAFT

Архітектура нейронної мережі CRAFT досить подібна до архітектури EAST (Рис. 1.11). Перша видобуває дещо більшу кількість ознак, щоб забезпечити розпізнавання низькорівневих послідовностей на зображенні, що відповідають тексту великого розміру. Для цього використовується мережа VGG16-BN, замість аналогічної PVANet у EAST. Остання є більш пристосованою до розпізнавання у реальному часі.

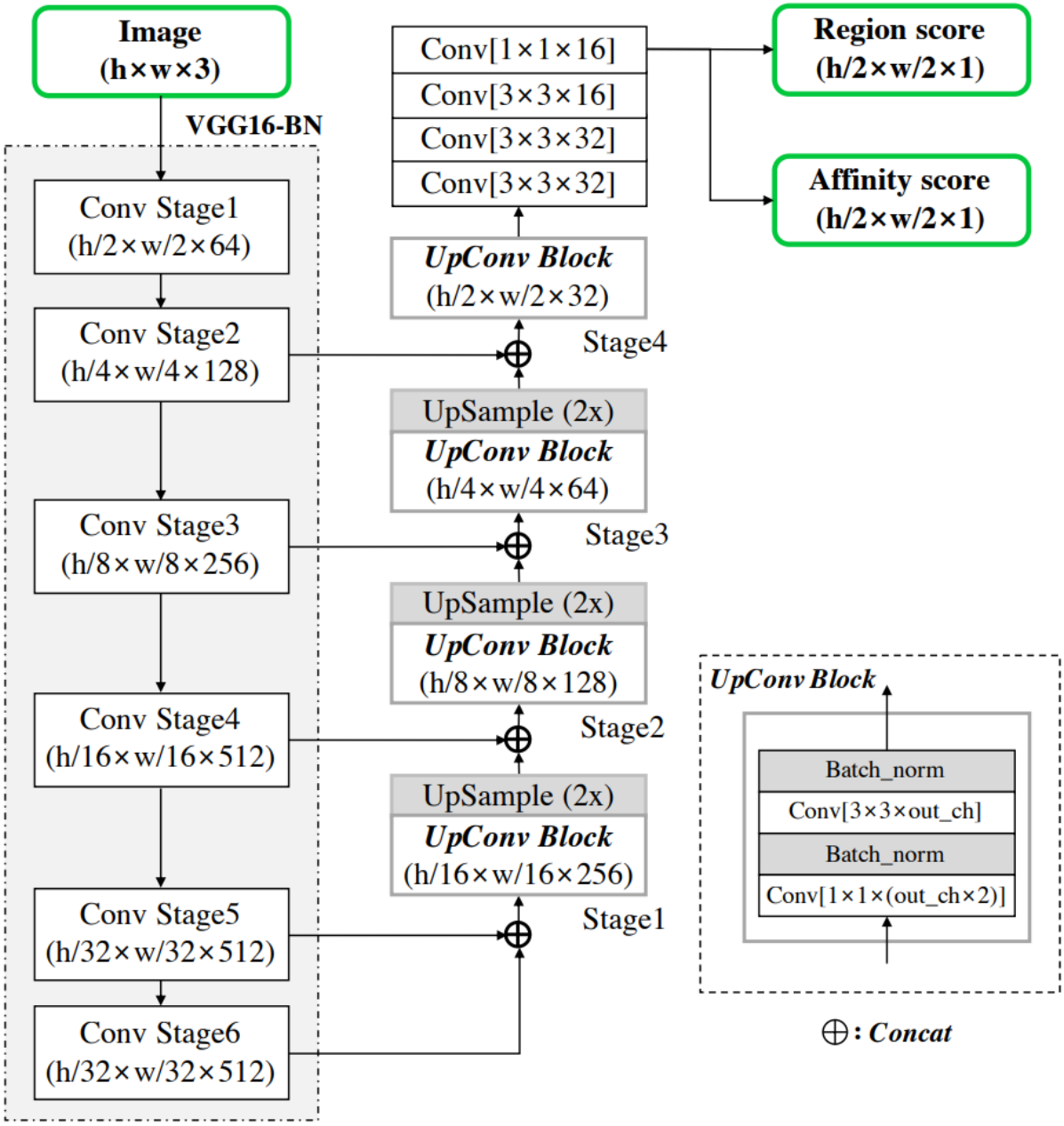


Рисунок 1.11 – Структура мережі CRAFT

1.4.4. FOTS: Fast Oriented Text Spotting with a Unified Network

Найпоширенішим способом зчитування тексту із зображення є окреме виявлення тексту і його розпізнавання, що обробляється як дві окремі задачі. Підходи на основі глибокого навчання домінують у обох завданнях. При виявленні тексту зазвичай використовується згорткова нейронна мережа для добування карт ознак із зображення, а потім декодери для декодування областей на цих картах. У розпізнаванні тексту окрема мережа проводить послідовне

передбачення для кожної з областей тексту, одна за одною, що призводить до великих витрат часу, особливо для зображень із великою кількість областей тексту. Інша проблема полягає в ігноруванні кореляції візуальних ознак, спільних для виявлення та розпізнавання. Мережа пошуку тексту не може користуватись ознаками, виявленими мережею розпізнавання, і навпаки.

FOTS пропонує одночасно виконувати пошук та розпізнавання тексту [16]. На відміну від попереднього двоетапного виявлення тексту, згорткова мережа у FOTS, яку можна навчати наскрізно, виявляє більше загальних ознак, що стають спільними для виявлення та розпізнавання тексту (Рис. 1.12). Оскільки виділення ознак зазвичай займає найбільше часу, загальний час розпізнавання тексту у порівнянні з іншими підходами зменшується майже вдвічі. З'єднання ознак пошуку та розпізнавання здійснює модуль ROIRotate.

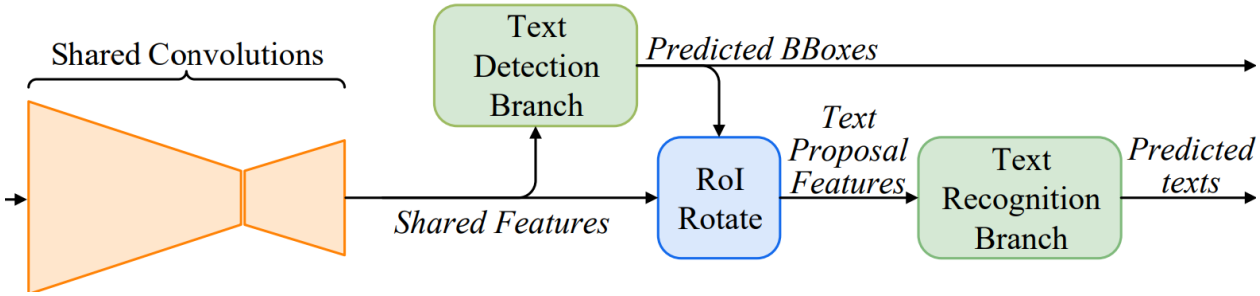


Рисунок 1.12 – Структура мережі FOTS

Цей модуль вирівнює відносно осей текстової області ознаки, видобуті під час пошуку тексту, та передає їх у мережу розпізнавання.

РОЗДІЛ 2.
РОЗРОБКА АЛГОРИТМУ РОЗПІЗНАВАННЯ ТЕКСТУ В УМОВАХ РІЗНОЇ ОСВІТЛЕНОСТІ ТА ЯКОСТІ ЗОБРАЖЕННЯ

2.1. Постановка задачі

Методи виявлення тексту на основі нейронних мереж, що з’явилися останнім часом показали гарні результати. Основною проблемою, яку вони вирішують, є виявлення та компонування тексту на випадкових зображеннях в умовах різної освітленості та якості зображення. Для вирішення аналогічної проблеми дана розробка повинна виконувати наступні задачі:

- працювати із основними форматами зображень;
- знаходити текст на вхідних зображеннях;
- розпізнавати знайдений текст посимвольно;
- працювати із стандартним латинським алфавітом та арабськими цифрами;
- повертати користувачу текст, знайдений на зображенні, у форматі, придатному до редагування.

Розробка повинна надавати можливість користувачу взаємодіяти з нейронними мережами, а саме:

- тренувати мережу;
- тестувати мережу на окремому наборі даних;
- використовувати мережу для отримання результатів розпізнавання.

Кожен варіант взаємодії включає у себе відповідний режим роботи розробки.

Для покращення процесу тренування розробка повинна містити інструмент для побудови впорядкованих наборів даних із позначених зображень.

Взаємодія з користувачем повинна відбуватись через консоль та вікна із зображенням, що обробляється. Консольний інтерфейс повинен бути зрозумілим

користувачеві та надавати вичерпну допомогу з використання субкоманд, доступних у розробці.

Навчальний характер розробки не передбачає жорстких критеріїв швидкодії програми та надає перевагу послідовності, чіткості та зрозумілості коду програми для інших користувачів, явність використання обраних алгоритмів та легкість модифікації розробки для власних потреб із тренування нейронних мереж.

2.2. Методологія дослідження

Хоча дослідження та програмування розробки проводилось за участі однієї людини і не потребувало моделей розробки, притаманних при роботі у команді, у якості моделі розробки було обрано методологію Agile.

Гнучка методологія Agile – це підхід до управління проектом, який передбачає розбиття проекту на фази та наголошує на безперервній співпраці та вдосконаленні. Команди дотримуються циклу планування, виконання та оцінювання. Методологія Agile зосереджена на регулярній доставці невеликих фрагментів роботи замість однієї великої ітерації. Ця методологія є ітеративною та відома тим, що розбиває проект на менші частини, пристосовуючись до мінливих вимог (Рис. 2.1).

Перевагою даного методу розробки програмного забезпечення є гнучкість і здатність реагувати на зміни. Команди можуть легко коригувати свої плани та пріоритети на основі нових вимог або відгуків під час проекту.

Недоліком ітераційної та адаптивної природи Agile є те, що вона іноді може призводити до невизначеності, особливо в проектах з нечіткими або швидко мінливими вимогами. Це може створити труднощі для точної оцінки термінів і витрат.

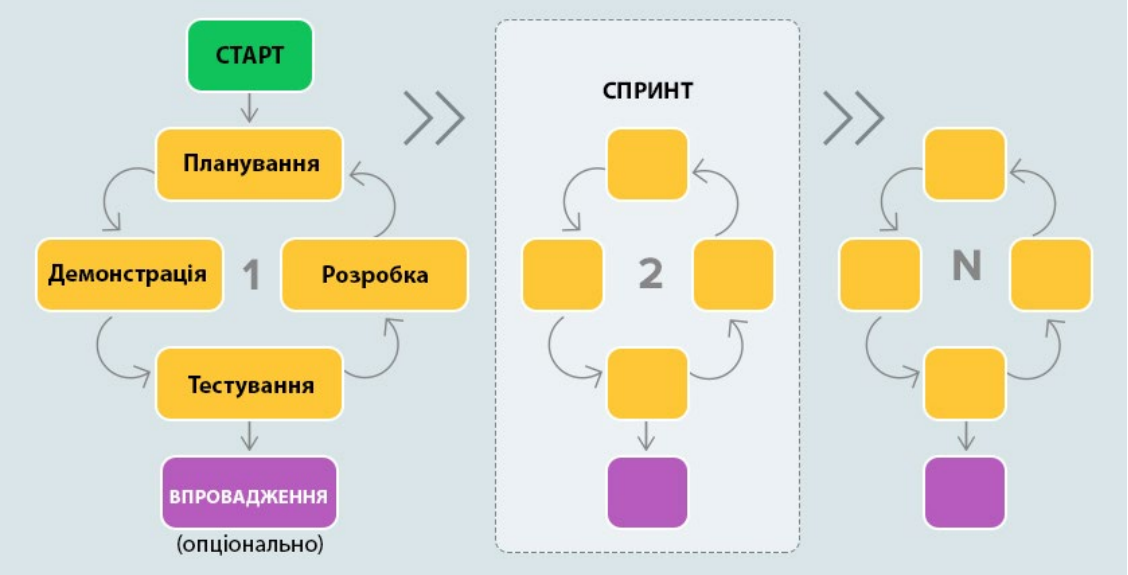


Рисунок 2.1 – Схема методології Agile

У рамках створення розробки із використанням нейронних мереж, кожне тренування моделі на новому наборі даних може відповідати окремому спринту Agile.

2.3. Теоретичні аспекти дослідження

В якості головних компонент розробки повинні бути використані дві нейронні мережі. Перша має відповідати наступним вимогам:

- приймає на вхід повнорозмірне зображення із трьома кольоровими каналами;

- повертає координати всіх знайдених символів у форматі QUAD (координати кутів чотирикутника, описаного навколо символа).

Використовуючи отримані координати, розробка повинна видобути із початкового зображення трансформовані фрагменти з зображеннями знайдених символів та передати їх на вхід другій нейронній мережі, що відповідає за розпізнавання.

Друга нейронна мережа повинна відповідати наступним вимогам:

- отримує на вхід фрагмент початкового зображення, що відповідає окремому символу;

- повертає на виході закодоване значення, що відповідає одному із символів набору даних, на якому було навчено мережу.

Окремий файл з конфігурацією роботи програми повинен містити усі основні налаштування для роботи з мережею, нівелюючи потребу пошуку змінюваних значень у коді розробки. Налаштування усіх режимів роботи повинні міститись у одному файлі.

У режимі тренування моделі розробка повинна показувати користувачеві перебіг процесу тренування та значення важливих параметрів. Збереження результатів тренування моделі повинно відбуватись у зазначені користувачем у файлі конфігурації проміжки часу.

У режимі тестування розробка повинна обчислити точність моделі, використовуючи набір даних, вказаний користувачем.

У режимі передбачення користувач повинен мати змогу переглядати виконувані дії над зображеннями та їх проміжні результати для полегшення налагодження моделей.

2.4. Обгрунтування вибору інструментальних засобів

Найпоширенішою мовою програмування розробок з використання машинного навчання та нейронних мереж є мова Python. Вона відрізняється легкістю вивчення та використання через простоту синтаксису та відсутність специфічних особливостей, що ускладнюють її використання у порівнянні з іншими мовами.

На мові Python написана велика кількість бібліотек, що полегшують роботу із нейронними мережами та алгоритмами обробки візуальних даних, у нашому випадку – зображень.

Ще однією з переваг використання мови програмування Python є віртуальні середовища, що дозволяють уникати конфліктів залежностей між встановленими бібліотеками та вести паралельну розробку проектів, що використовують різні версії тієї ж бібліотеки.

PyTorch – це бібліотека машинного навчання з відкритим кодом, створена на основі мови програмування Python. Вона пропонує динамічний і гнучкий підхід до побудови нейронних мереж і широко використовується в дослідницьких і виробничих середовищах.

OpenCV – це бібліотека функцій програмування, головним чином для комп'ютерного зору в реальному часі. Вона містить значну кількість інструментів для роботи із зображеннями.

У процесі розробки також використано Git – систему контролю версій, що дозволяє зберігати історію процесу розробки продукту, працювати із будь-якою його версією, взаємодіяти з кодом одночасно багатьом командам та керувати ходом розробки окремих фрагментів розробки.

У якості середовища розробки використовується редактор коду Visual Studio Code. Серед його переваг – можливість підключення різноманітних плагінів та розширень, розумне доповнення коду на основі контексту, додаткові інструменти налагодження, підсвічування помилок, аналіз в залежності від обраної мови програмування тощо. VSCode поєднує в собі простоту редактора коду з тим, що потрібно розробникам для основного циклу редагування-складання-налагодження (Рис. 2.2).

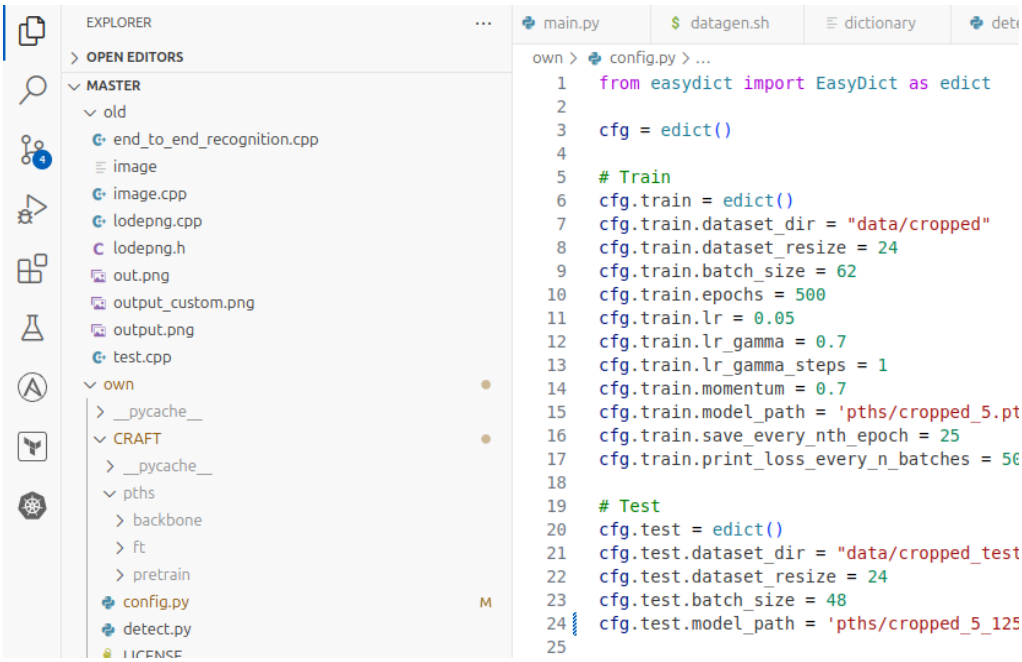


Рисунок 2.2 – Інтерфейс редактора коду VSCode

2.5. Етапи програмної реалізації

2.5.1. Модель пошуку тексту на зображенні на основі CRAFT

Розпізнавання тексту у розробці відбувається посимвольно, тому в якості основи для моделі пошуку найкраще підійде модель CRAFT.

Спочатку відтворимо структуру моделі, створивши відповідний клас та метод у ньому (Рис. 2.3).

```
149  class CRAFT(nn.Module):
150      def __init__(self, pretrained=True):
151          super(CRAFT, self).__init__()
152          self.extractor = extractor(pretrained)
153          self.merge   = merge()
154
155      def forward(self, x):
156          return self.merge(self.extractor(x))
```

Рисунок 2.3 – Клас моделі CRAFT

Модель складається із двох основних частин: видобування ознак (extractor) та їх об’єднання (merge). Через відсутність апаратної можливості тренування мережі CRAFT власноруч використаємо значення параметрів моделі, навченої на наборах даних ICDAR2013, ICDAR2017 та SynthText. Це забезпечує універсальність мережі та здатність працювати із зображеннями різної візуальної структури.

Видобування ознак здійснюється послідовними шарами згортки та нормалізації (Рис. 2.4). Велика кількість шарів згортки дозволяє мережі враховувати ознаки різного масштабу та складності.

Клас extractor за основу використовує модель VGG16_BN. Для її ініціалізації використовуються допоміжні функції. Функція make_layers допомагає згенерувати шари нейронної мережі з визначеного списку (Рис. 2.5).

Функція init_parameters налаштовує значення шарів видобування ознак зображення (Рис. 2.6). Клас VGG також використовує допоміжні функції для ініціалізації частини нейронної мережі.

Для додавання шарів об’єднання ознак використовується ще одна допоміжна функція upconv_block (Рис. 2.8).

```
53  class extractor(nn.Module):
54      def __init__(self, pretrained):
55          super(extractor, self).__init__()
56          vgg16_bn = VGG(make_layers(cfg))
57          if pretrained:
58              vgg16_bn.load_state_dict(torch.load('./CRAFT/pths/backbone/
                vgg16_bn-6c64b313.pth', weights_only=True))
59          self.features = vgg16_bn.features
60
61          self.conv_6 = nn.Sequential(
62                        nn.Conv2d(512, 512, 3, padding=1, bias=False),
63                        nn.BatchNorm2d(512),
64                        nn.ReLU(inplace=True),
65                        nn.Conv2d(512, 512, 3, padding=1, bias=False),
66                        nn.BatchNorm2d(512),
67                        nn.ReLU(inplace=True))
68          _init_parameters(self.conv_6.modules())
69
70      def forward(self, x):
71          out = []
72          for m in self.features:
73              x = m(x)
74              if isinstance(m, nn.MaxPool2d):
75                  out.append(x)
76          x = self.conv_6(x)
77          out.append(x)
78          return out[1:]
```

Рисунок 2.4 – Клас видобування ознак

```
6   cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512,
    512, 'M']
7
8
9   def make_layers(cfg):
10      layers = []
11      in_channels = 3
12      for v in cfg:
13          if v == 'M':
14              layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
15          else:
16              conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
17              layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
18              in_channels = v
19      return nn.Sequential(*layers)
20
```

Рисунок 2.5 – Функція генерації шарів підмережі VGG16_BN

```
21
22  def _init_parameters(net):
23      for m in net:
24          if isinstance(m, nn.Conv2d):
25              nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
26              if m.bias is not None:
27                  nn.init.constant_(m.bias, 0)
28          elif isinstance(m, nn.BatchNorm2d):
29              nn.init.constant_(m.weight, 1)
30              nn.init.constant_(m.bias, 0)
31          elif isinstance(m, nn.Linear):
32              nn.init.normal_(m.weight, 0, 0.01)
33              nn.init.constant_(m.bias, 0)
34
```

Рисунок 2.6 – Ініціалізація параметрів мережі

```
36  class VGG(nn.Module):
37      def __init__(self, features):
38          super(VGG, self).__init__()
39          self.features = features
40          self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
41          self.classifier = nn.Sequential(
42              nn.Linear(512 * 7 * 7, 4096),
43              nn.ReLU(True),
44              nn.Dropout(),
45              nn.Linear(4096, 4096),
46              nn.ReLU(True),
47              nn.Dropout(),
48              nn.Linear(4096, 1000),
49          )
50          _init_parameters(self.modules())
51
```

Рисунок 2.7 – Клас підмережі VGG16_BN

```
79
80
81  class upconv_block(nn.Module):
82      def __init__(self, in_channel, out_channel):
83          super(upconv_block, self).__init__()
84
85          self.conv1 = nn.Conv2d(in_channel, out_channel*2, 1, bias=False)
86          self.bn1 = nn.BatchNorm2d(out_channel*2)
87          self.relu1 = nn.ReLU(inplace=True)
88          self.conv2 = nn.Conv2d(out_channel*2, out_channel, 3, padding=1, bias=False)
89          self.bn2 = nn.BatchNorm2d(out_channel)
90          self.relu2 = nn.ReLU(inplace=True)
91
92      def forward(self, x):
93          return self.relu2(self.bn2(self.conv2(self.relu1(self.bn1(self.conv1(x))))))
94
95
```

Рисунок 2.8 – Функція для створення блоку шарів об’єднання ознак

```
class merge(nn.Module):
    def __init__(self):
        super(merge, self).__init__()

        self.upconv1 = upconv_block(1024, 256)
        self.upconv2 = upconv_block(512+256, 128)
        self.upconv3 = upconv_block(256+128, 64)
        self.upconv4 = upconv_block(128+64, 32)

        self.conv = nn.Sequential(
                    nn.Conv2d(32, 32, 3, padding=1, bias=False),
                    nn.BatchNorm2d(32),
                    nn.ReLU(inplace=True),
                    nn.Conv2d(32, 32, 3, padding=1, bias=False),
                    nn.BatchNorm2d(32),
                    nn.ReLU(inplace=True),
                    nn.Conv2d(32, 16, 3, padding=1, bias=False),
                    nn.BatchNorm2d(16),
                    nn.ReLU(inplace=True),
                    nn.Conv2d(16, 16, 1, bias=False),
                    nn.BatchNorm2d(16),
                    nn.ReLU(inplace=True))

        self.region_head = nn.Conv2d(16, 1, 1)
        self.affinity_head = nn.Conv2d(16, 1, 1)
        self.sigmoid1 = nn.Sigmoid()
        self.sigmoid2 = nn.Sigmoid()
        _init_parameters(self.modules())
```

Рисунок 2.9 – Клас модуля об’єднання ознак

```

    def forward(self, x):
        y = torch.cat((x[4], x[3]), 1)
        y = self.upconv1(y)
        y = F.interpolate(y, scale_factor=2, mode='bilinear', align_corners=True)

        y = torch.cat((y, x[2]), 1)
        y = self.upconv2(y)
        y = F.interpolate(y, scale_factor=2, mode='bilinear', align_corners=True)

        y = torch.cat((y, x[1]), 1)
        y = self.upconv3(y)
        y = F.interpolate(y, scale_factor=2, mode='bilinear', align_corners=True)

        y = torch.cat((y, x[0]), 1)
        y = self.upconv4(y)
        y = F.interpolate(y, scale_factor=2, mode='bilinear', align_corners=True)

        y = self.conv(y)
        region_score = self.sigmoid1(self.region_head(y))
        affinity_score = self.sigmoid2(self.affinity_head(y))

        return region_score, affinity_score

```

Рисунок 2.10 – Код обчислення значень нейронів модуля об’єднання ознак

Підмережа об’єднання ознак складається з великої кількості шарів згортки, створених через допоміжну функцію upconv_block.

З допомогою бібліотеки easydict реалізується файл конфігурації використання нейронної мережі для пошуку символів CRAFT. Файл містить параметри нормалізації вхідних зображень, використаних під час тренування мережі, шляхи запису вхідних та вихідних зображень, та обмеження для областей із символами та зв’язуючих областей. Коригування цех обмежень дозволяє працювати із зображеннями з різним відносним розташуванням символів та розділяти щільно розміщені області символів (Рис. 2.11).

```
from easydict import EasyDict as edict

cfg = edict()

cfg.train = edict()
cfg.train.mean = [0.5, 0.5, 0.5]
cfg.train.std = [0.25, 0.25, 0.25]

cfg.test = edict()
cfg.test.model_pth = './CRAFT/pths/pretrain/model_iter_50000.pth'
cfg.test.out_img = './res.bmp'
cfg.test.region_thresh = 0.29
cfg.test.affinity_thresh = 0.37
cfg.test.remove_thresh = 6 * 1e-6
cfg.test.long_side = 960

```

Рисунок 2.11 – Файл конфігурації моделі CRAFT

Для роботи з будь-яким розміром зображення воно спочатку масштабується до розмірів, вказаних у конфігурації, записуючи відношення старих розмірів до нових (Рис. 2.12).

При завантаженні зображення у мережу воно перетворюється у спеціальний тип даних, з яким працює бібліотека Torch (Рис. 2.13).

Результати обчислень моделі необхідно перемістити назад з пам’яті обчислювального пристрою, де виконувались розрахунки (Рис. 2.14).

```
19
20
21  def resize_img(img, long_side):
22      w, h = img.size
23      if long_side is not None:
24          if w > h:
25              resize_w = long_side
26              ratio = long_side / w
27              resize_h = h * ratio
28          else:
29              resize_h = long_side
30              ratio = long_side / h
31              resize_w = w * ratio
32      else:
33          resize_h, resize_w = h, w
34
35      final_h = int(resize_h) if resize_h % 32 == 0 else (int(resize_h / 32) + 1) * 32
36      final_w = int(resize_w) if resize_w % 32 == 0 else (int(resize_w / 32) + 1) * 32
37      img = img.resize((final_w, final_h), Image.BILINEAR)
38      ratio_h = final_h / h
39      ratio_w = final_w / w
40      return img, ratio_h, ratio_w
41
42
```

Рисунок 2.12 – Масштабування вхідних даних мережі CRAFT

```
41
42
43  def load_pil(img):
44      t = transforms.Compose([transforms.ToTensor(),
        transforms.Normalize(cfg.train.mean, cfg.train.std)])
45      return t(img).unsqueeze(0)
46
47
```

Рисунок 2.13 – Функція переформатування вхідних даних

```
46
47
48  def get_score(img, model, device):
49      with torch.no_grad():
50          region, affinity = model(load_pil(img).to(device))
51      return list(map(lambda x: x[0][0].cpu().numpy(), [region, affinity]))
52
53
```

Рисунок 2.14 – Перенесення даних між обчислювальними пристроями

Результатом обчислень мережі CRAFT є теплова мапа з областями символів та зв’язуючими областями. У класичному варіанті використання мережі межі слів генеруються з суми значень цих мап. У даній модифікації для виділення окремих символів метод генерації меж віднімає від мапи символів

зв’язуючу мапу (операція множення або кон’юнкції з оберненою мапою, 57-й рядок коду) (Рис. 2.15). Це забезпечує відокремлення символів для їх легкого розпізнавання іншою нейронною мережею.

```
52
53
54  def restore_boxes(region, affinity, region_thresh,
    affinity_thresh, remove_thresh, ratio):
55      # return [[x1, y1, x2, y2, x3, y3, x4, y4], [], ...]
56      boxes = []
57      M = (region > region_thresh) * np.invert(affinity >
        affinity_thresh)
58      # cv2.imwrite('/tmp/M.png', np.float32(M)*255.0)
59      ret, markers = cv2.connectedComponents(np.uint8(M * 255))
60      for i in range(ret):
61          if i == 0:
62              continue
63          y,x=np.where(markers==i)
64          if len(y) < region.size * remove_thresh:
65              continue
66          cords = 2 * np.concatenate(
67              (x.reshape(-1,1)/ratio[1],y.reshape(-1,1)/ratio[0]),
68              axis=1)
69          a = np.array([
70              cords[:,0].min(), cords[:,1].min(), cords[:,0].max(),
                cords[:,1].min(), cords[:,0].max(), cords[:,1].max(),
                cords[:,0].min(), cords[:,1].max()])
71          boxes.append(a)
72      return boxes
73
74
```

Рисунок 2.15 – Функція визначення координат областей символів

Функція detect_single_image проводить повну процедуру пошуку символів на окремому зображенні та повертає їх координати (Рис. 2.16).

```
75  def detect_single_image(img, model, device, cfg):
76      img, ratio_h, ratio_w = resize_img(img, cfg.long_side)
77      region, affinity = get_score(img, model, device)
78      boxes = restore_boxes(region, affinity, cfg.region_thresh,
        cfg.affinity_thresh, cfg.remove_thresh, (ratio_h, ratio_w))
79      return boxes
80
```

Рисунок 2.16 – Кінцева функція використання мережі із зображенням

Функція plot_boxes наносить знайдені межі усіх символів на вхідне зображення (Рис. 2.17).

```
11
12  def plot_boxes(img, boxes):
13      if boxes is None:
14          return img
15      draw = ImageDraw.Draw(img)
16      for box in boxes:
17          draw.polygon([box[0], box[1], box[2], box[3], box[4],
            box[5], box[6], box[7]], outline=(0,255,0))
18      return img
19
```

Рисунок 2.17 – Нанесення меж областей на зображення

Основна функція роботи з нейронной мережею CRAFT завантажує значення її параметрів, зчитує зображення та конвертує його у потрібний формат, знаходить межі символів на ньому, наносить їх та зберігає за вказаним шляхом (Рис. 2.18).

```
98
99  def main(img_path, model_path=cfg.test.model_pth, out=cfg.
    test.out_img):
100     device = torch.device("cuda:0" if torch.cuda.is_available
        () else "cpu")
101     model = CRAFT().to(device)
102     model.load_state_dict(torch.load(model_path,
        map_location=device, weights_only=True))
103     model.eval()
104     img = Image.open(img_path).convert('RGB')
105     boxes = detect_single_image(img, model, device, cfg.test)
106     img = plot_boxes(img, boxes)
107     if out:
108         img.save(out)
109     return img, boxes
110
111
```

Рисунок 2.18 – Функція взаємодії із мережею CRAFT

Класи, функції, файл конфігурації та значення параметрів навчених моделей для роботи з мережею пошуку символів CRAFT впорядковані у окремій папці (Рис. 2.19).

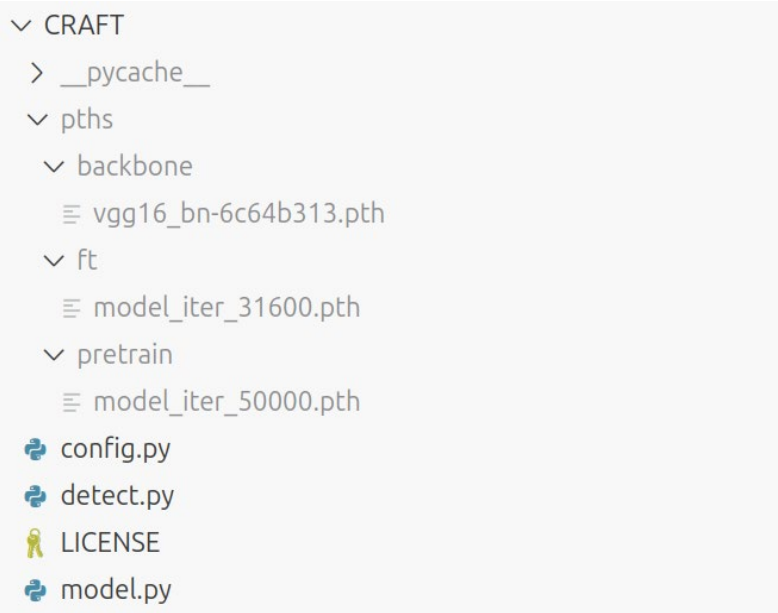


Рисунок 2.19 – Структура директорії моделі CRAFT

2.5.2. Взаємодія з наборами даних

Для відображення зображення, завантаженого у пам’ять програми, використовується функція imshow, що впорядковує кольорові канали перед виведенням (Рис. 2.10).

```
30
31  def imshow(img, label=None):
32      if label:
33          print(label)
34      img = np.transpose(img.numpy(), (1,2,0))
35      plt.imshow(img)
36      plt.show()
37
```

Рисунок 2.20 – Функція відображення результатів роботи мережі

Для генерації наборів даних для тренування мережі використовується сторонній засіб Text Recognition Data Generator [17] . Варіант з запуском генерації всередині Docker-контейнера використовується у скрипті для створення наборів даних (Рис. 2.21).

```
#!/bin/sh

# mkdir data/skewed
# docker run -v ./data/skewed/:/app/out/ -v ./data/dictionary:/app/dictionary
-t belval/trdg:latest trdg -t 4 -b 3 -tc '#000000,#FFFFFF' -i /app/dictionary
-k 30 -rk -bl 1 -rbl -fi -m 0 -wd 0 -c 62000

cd data/skewed
echo 'Creating classdirs...'
for let in $(ls | cut -d'_' -f1 | sort | uniq); do mkdir $let; done;
echo 'Moving to classdirs...'
for file in $(ls | grep .jpg); do mv $file "$(echo $file | cut -d'_' -f1)/";
done;
echo 'Creating test data...'
mkdir ../skewed_test
for folder in $(ls); do mkdir ../skewed_test/$folder && mv "${folder}/$(ls
$folder | head -n1)" ../skewed_test/$folder/; done;

```

Рисунок 2.21 – Скрипт генерації зображень символів для набору даних

Результатом роботи скрипта є папка набору даних та підпапки з назвами класів, що містять зображення символів відповідного класу. У даній розробці використано набори даних з 62-х класів: великі і маленькі букви латинського алфавіту та цифри від 0 до 9. Також окремо створюється директорія з меншою кількістю екземплярів класів, що не залучається до тренування та використовується у наборі даних для тестування точності мережі. З такою структурою класи, що розпізнаються мережею, можна отримати з переліку директорій у наборі даних (Рис. 2.22).

```

def get_classes(dataset_dir):
    return sorted(os.listdir(dataset_dir))

```

Рисунок 2.22 – Отримання переліку класів у згенерованому наборі даних

При завантаженні набору даних зображення у ньому змінюються випадковим чином для покращення здатності мережі до узагальнення, що полегшує створення репрезентативного набору даних (Рис. 2.23).

```
10
11  def load_dataset_torch(dataset_dir, resize=16, batch_size=32, show=False):
12      transform = transforms.Compose([
13          transforms.Resize((resize,resize)),
14          transforms.RandomPerspective(distortion_scale=0.3, p=0.5, fill=128.0),
15          # transforms.CenterCrop(9),
16          transforms.ToTensor()
17      ])
18      dataset = datasets.ImageFolder(dataset_dir, transform=transform)
19      dataloader = torch.utils.data.DataLoader(dataset,
20                                               batch_size=batch_size,
21                                               shuffle=True)
22
23      if show:
24          classes = get_classes(dataset_dir)
25          images, labels = next(iter(dataloader))
26          print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
27          imshow(utils.make_grid(images))
28
29      return dataloader
30
```

Рисунок 2.23 – Масштабування та випадкові модифікації набору даних при навчанні

Результатом проходження згенерованих зображень через зазначені трансформації є набір даних, що дозволяє мережі навчитись розпізнавати зображення у неідеальних умовах (Рис. 2.24).

Рисунок 2.24 – Фрагмент тренувального набору даних після застосування випадкових трансформацій

Для навчання мережі було використано набір даних із 6200 зображень розмірності 24x24x3, що передавались на вхід до мережі порціями по 24 зображення.

2.5.3. Модель-класифікатор для розпізнавання зображень символів

Модель для розпізнавання символів має класичну архітектуру класифікаторів. Вона складається із двох послідовних шарів згортки і об'єднання, та чотирьох повністю з'єднаних шарів (Рис. 2.25).

```
 9
10  class Net(nn.Module):
11      def __init__(self):
12          super().__init__()
13          self.conv1 = nn.Conv2d(3, 18, 5)
14          self.pool1 = nn.MaxPool2d(2, 2)
15          self.conv2 = nn.Conv2d(18, 48, 3)
16          self.pool2 = nn.MaxPool2d(2, 2)
17          self.fc1 = nn.Linear(48 * 4 * 4, 512)
18          self.fc2 = nn.Linear(512, 256)
19          self.fc3 = nn.Linear(256, 128)
20          self.fc4 = nn.Linear(128, 62)
21
22      def forward(self, x):
23          x = self.pool1(F.relu(self.conv1(x)))
24          x = self.pool2(F.relu(self.conv2(x)))
25          x = torch.flatten(x, 1) # flatten all dimensions except batch
26          x = F.relu(self.fc1(x))
27          x = F.relu(self.fc2(x))
28          x = F.relu(self.fc3(x))
29          x = self.fc4(x)
30          return x
31
```

Рисунок 2.25 – Структура мережі для розпізнавання символів

Для підвищення ефективності процесу навчання мережі використовуються різноманітні методи оптимізації (Рис. 2.26). Збереження імпульсу, тобто додавання до поточного градієнту зменшене попереднє значення, прискорює навчання мережі в цілому. Згасання значень ваг нейронів запобігає надмірному використанню певної частини мережі та забезпечує рівномірний вклад кожного її нейрону. Зниження швидкості навчання при досягненні плато у точності допомагає мережі навчитись враховувати дрібніші та менш помітні ознаки.

```
38      optimizer = optim.SGD(net.parameters(), lr=lr, momentum=momentum,
        weight_decay=0.001)
39      scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer,
        mode='min', cooldown=lr_cd, factor=lr_f, patience=lr_p)
```

Рисунок 2.26 – Застосування оптимізацій у процесі навчання мережі

Обрані оптимізаційні параметри: збереження інерції 0.25, множник швидкості навчання 0.1, що домножується на 0.5 після 2-х епох без підвищення точності. Графік значень похибки на тренувальному та тестовому наборах даних при навчанні мережі на згенерованому наборі даних відображає поточну точність похибку мережі кожної епохи (Рис. 2.27).

Рисунок 2.27 – Графік похибки мережі у процесі тренування

Результатом навчання є мережа класифікації символів із точністю 83.9% на тестовому наборі даних.

2.6. Організація тестування та налагодження програмного засобу

Кожна ітерація навчання нейронної мережі супроводжується її тестуванням на окремому наборі даних. Це не дозволяє мережі використовувати для класифікації символів ознаки, що присутні лише у тренувальному наборі даних.

Аналіз графіків точності мережі під час кожної епохи тренування допомагає правильно налаштувати параметри навчання та виявити достатню кількість епох, після якої зміна точності мережі не є суттєвою.

Для автоматичного створення графіків використовується AimStack – спеціалізований трекер метаданих процесів машинного навчання з відкритим вихідним кодом [18]. Для його використання достатньо передавати йому значення, що потрібно відслідкувати у ході тренування мережі (Рис. 2.28). Ці значення автоматично обробляються та стають доступні для візуалізації засобами AimStack.

```
76
77          aim_run.track(val_loss.item(), 'val_loss', epoch=epoch)
78          aim_run.track(running_loss/steps, 'loss', epoch=epoch)
79
```

Рисунок 2.28 – Передавання значень похибки трекеру AimStack

Головною проблемою у перших ітераціях тренування мережі була подібність певних символів при однаковому обтинанні зображення, зокрема символів “m” та “T” (Рис. 2.29). Генерація символу “m” більшої ширини з наступним горизонтальним стисненням допомогло навчити мережу краще їх розрізняти.

Рисунок 2.29 – Подібність символів при неправильному обтинанні

2.7. Аналіз отриманих результатів дослідження, рекомендації щодо використання та впровадження

Створена програмна розробка з навченими нейронними мережами знаходить та розпізнає символи із точністю 83,87%. Для визначення точності

використовувався тестовий набір даних, що був згенерований аналогічно до тренувального.

Натренована мережа розпізнавання найкраще працює із зображеннями, на яких символи добре відокремлені (Рис. 2.30). Цей недолік можна компенсувати масштабуванням нейронної мережі, проте ефективнішим рішенням є використання рекурентних мереж для розпізнавання цілого слова.

Рисунок 2.30 – Приклад допустимого вхідного зображення

Нейронна мережа для пошуку тексту на основі CRAFT повертає мапу областей із символами (Рис. 2.31). Точність знаходження тексту у специфічних умовах можна покращити, тренуючи модель на відповідному наборі даних.

Коригування порогових значень при поділі областей на окремі символи може покращити результати у випадках незвичайного відносного розміщення символів у слові.

Області з надто малою площею не передаються до другої нейронної мережі через недостатню роздільну здатність вхідних даних, що призводить до значного зменшення точності розпізнавання (Рис. 2.32).

Рисунок 2.31 – Мапи областей символів, згенеровані мережею CRAFT

Рисунок 2.32 – Межі областей символів із результатами розпізнавання

ВИСНОВКИ

У ході дослідження було проаналізовано тенденції розвитку сфери комп’ютерного зору та досліджено алгоритми машинного навчання нейронних мереж. Результати враховувались при плануванні архітектури програмної розробки.

Розгляд особливостей процесу тренування нейронних мереж-класифікаторів дозволив ефективно контролювати процес навчання власної мережі. Аналіз впливу використання репрезентативних наборів даних на точність результатів передбачень засобів розпізнавання тексту пришвидшив виявлення проблем із вхідними зображеннями.

Також у роботі розглянуто існуюче програмне забезпечення для розпізнавання тексту та присутнє порівняння його із методами, що використовувались раніше. Проаналізовано типову архітектуру засобу для розпізнавання тексту з використанням нейронних мереж. Результатом роботи є розроблена та навчена нейронна мережа для розпізнавання тексту на зображенні.

Створену розробку можна покращити екстенсивним шляхом, через масштабування даних та мереж, що потребуватиме більших обчислювальних потужностей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IBM. What is Computer Vision? | IBM. IBM - United States. URL: https://www.ibm.com/topics/computer-vision (дата звернення: 08.12.2024).
2. IBM. What Is Machine Learning (ML)? | IBM. IBM - United States. URL: https://www.ibm.com/topics/machine-learning (дата звернення: 08.12.2024).
3. Machine learning, explained | MIT Sloan. MIT Sloan. URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (дата звернення: 08.12.2024).
4. Contributors to Wikimedia projects. Convolutional neural network - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network (дата звернення: 08.12.2024).
5. GeeksforGeeks. Activation functions in Neural Networks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/activation-functions-neural-networks/ (дата звернення: 08.12.2024).
6. A Quick Introduction to Neural Networks. Ujjwal Karn’s blog. URL: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/ (дата звернення: 08.12.2024).
7. Backpropagation. Strona główna -- Serwis Akademii Górniczo-Hutniczej. URL: https://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html (дата звернення: 08.12.2024).
8. Gill N. S. Convolutional Recurrent Neural Network For Text Recognition. Infuse AI Into Enterprise Systems with your Data. URL: https://www.xenonstack.com/insights/crnn-for-text-recognition (дата звернення: 08.12.2024).
9. 5 Steps to OCR Training Data. AIMultiple: High Tech Use Cases & Tools to Grow Your Business. URL: https://research.aimultiple.com/ocr-training-data/ (дата звернення: 08.12.2024).

10. Solution G. T. A Comprehensive List of OCR Datasets for Machine Learning. Medium. URL: https://medium.com/@shalinigts16/a-comprehensive-list-of-ocr-datasets-for-machine-learning-d3c29ed9983a (дата звернення: 08.12.2024).
11. Smith R. An Overview of the Tesseract OCR Engine. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf (дата звернення: 08.12.2024).
12. EAST: An Efficient and Accurate Scene Text Detector. arXiv.org. URL: https://arxiv.org/abs/1704.03155 (дата звернення: 08.12.2024).
13. Pandey P. An Efficient and Accurate Scene Text Detector [EAST]. Medium. URL: https://poshan0126.medium.com/an-efficient-and-accurate-scene-text-detector-east-973df9dfdd55 (дата звернення: 08.12.2024).
14. PVANET: Deep but Lightweight Neural Networks for Real-time Object Detection. arXiv.org. URL: https://arxiv.org/abs/1608.08021 (дата звернення: 08.12.2024).
15. Isbarov J. Going deeper with convolutions: The Inception paper, explained. Medium. URL: https://medium.com/aiguys/going-deeper-with-convolutions-the-inception-paper-explained-841a0c661fd3 (дата звернення: 08.12.2024).
16. FOTS: Fast Oriented Text Spotting with a Unified Network. arXiv.org. URL: https://arxiv.org/abs/1801.01671 (дата звернення: 08.12.2024).
17. GitHub - Belval/TextRecognitionDataGenerator: A synthetic data generator for text recognition. GitHub. URL: https://github.com/Belval/TextRecognitionDataGenerator (дата звернення: 08.12.2024).
18. Home | AimStack. AimStack. URL: https://aimstack.io/ (дата звернення: 08.12.2024).

Додаток А

Технічне завдання

Вступ

Програмний продукт для розпізнавання текстової інформації на растровому зображенні з використанням засобів комп'ютерного зору.

Комп'ютерний зір — це галузь штучного інтелекту, яка використовує машинне навчання та нейронні мережі для навчанні комп'ютерних систем видобуванню інформації з цифрових зображень, відео та інших візуальних вхідних даних

Підстави для розробки

Кваліфікаційна робота на здобуття освітнього ступеня «магістр» студента спеціальності "122 Комп'ютерні науки" освітньо-професійної програми "Комп'ютерні науки та інформаційні технології" Пелеха Гєрмана Валерійовича "Дослідження алгоритмів комп'ютерного зору для розпізнавання тексту в умовах різної освітленості та якості зображення".

Призначення розробки

Функціональне призначення: тренування, використання та обробка результатів роботи нейронних мереж для пошуку та класифікації символів на растровому зображенні.

Експлуатаційне призначення: використання у навчальних цілях для демонстрації видобування тексту із зображення з допомогою нейронних мереж.

Вимоги до програмного продукту

Вимоги до функціональних характеристик: програма повинна розпізнавати 62 символа: великі та маленькі латинські літери, цифри 0-9.

Вимоги до складу і параметрів технічних засобів: розробка повинна працювати на будь-якій платформі, що здатна запускати Python 3.12.

Вимоги до інформаційної і програмної сумісності: програмний продукт повинен приймати на вхід усі поширені формати файлів растрових зображень, та повертати координати областей символів з відповідними їм класами.

Вимоги до програмної документації

Програмна документація повинна містити основні команди для взаємодії з розробкою та способи налаштування її роботи.

Стадії і етапи розробки

Стадія досліджень: дослідження засобів комп’ютерного зору, поширених алгоритмів, існуючих розробок, їх переваг та недоліків.

Стадія проектування: створення архітектури програми, планування взаємодії її компонентів та обсягу функціоналу.

Стадія розробки: написання коду програмного продукту, створення моделей використовуваних нейронних мереж.

Стадія тренування: генерація наборів даних, навчання створених нейронних мереж, їх тестування та коригування.

Порядок контролю і приймання

Програма повинна здійснювати розпізнавання тексту за період часу менше 10 секунд.

Точність розпізнавання розробки повинна перевищувати 80% на тестовому наборі даних.

Додаток Б

Інструкція користувачу

1. **Загальні відомості**

Програмний продукт для розпізнавання текстової інформації на растровому зображенні з використанням засобів комп’ютерного зору.

2. **Функціональне призначення**

Програмний продукт призначений для тренування, використання та обробки результатів роботи нейронних мереж для пошуку та розпізнавання символів на растровому зображенні. Розробка дозволяє тренувати мережу для розпізнавання символів на основі власного набору даних.

Основним сценарієм використання є використання у навчальних цілях для демонстрації видобування тексту із зображення з допомогою нейронних мереж.

3. **Умови застосування програми**

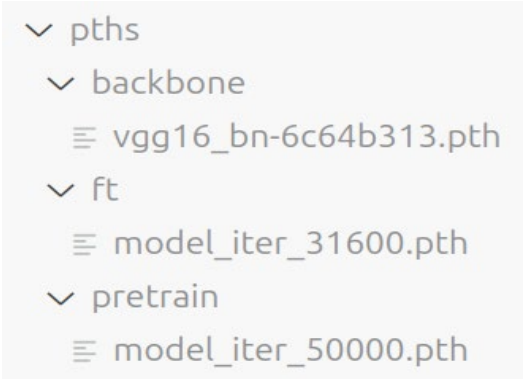
Програмний продукт доступний для використання на будь-якому пристрої, що підтримує інтерпретатор Python 3.12. Рекомендована кількість ядер (потоків) процесора - 8 і більше. Наявність графічного прискорювача необхідна лише при великому об’ємі власного набору даних для тренування.

4. **Повідомлення оператору**

Для інформування користувача про помилки у виконанні програми розробка спирається на вбудовані обробники помилок мови Python, що виводять повідомлення із розташуванням фрагменту коду, що є джерелом сигналу. При використанні даного програмного продукту за призначенням єдиною можливою помилкою є неправильний формат вхідних даних.

5. **Опис роботи програми**

Взаємодія з програмою здійснюється через консольний інтерфейс. Головний файл програми запускається за допомогою інтерпретатора Python. Перед використанням програми необхідно завантажити навчені моделі пошуку символів на зображенні CRAFT та помістити їх у відповідні директорії. Для цього у папці “CRAFT” потрібно створити підпапку “pths” з наступним вмістом.



Доступні три режими роботи програми, що обираються відповідним аргументом виконуваного головного скрипта.

Режим “train”: тренування та збереження нейронної мережі. У консоль виводиться повідомлення з архітектурою мережі, що навчається, та відображається перебіг процесу тренування з основними параметрами навчання. Для запуску цього режиму використовується команда “python main.py train”.

Режим “test”: тестування точності нейронної мережі. У консоль виводяться повідомлення з порівнянням результатів роботи мережі та істинних значень, і відсоткове співвідношення правильно класифікованих символів. У окремому вікні виводяться зображення вхідних даних, використаних для тестування. Для запуску цього режиму використовується команда “python main.py test”.

Режим “inference”: використання розробки за її прямим призначенням - для видобування тексту з вхідного зображення. Для запуску цього режиму використовується команда “python main.py inference”.

Конфігурація кожного режиму роботи налаштовується у окремому файлі “./config.py”. Окремі налаштування підмережі пошуку символів CRAFT знаходяться у файлі “./CRAFT/config.py”.

АНОТАЦІЯ

Пелех Г.В. Дослідження алгоритмів комп'ютерного зору для розпізнавання тексту в умовах різної освітленості та якості зображення. Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 122 Комп’ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2024 р.

Завданнями цієї кваліфікаційної роботи є дослідження алгоритмів машинного навчання нейронних мереж, розгляд існуючого програмного забезпечення для розпізнавання тексту, його типової архітектури при використанні у ньому нейронних мереж, та розробка і тренування нейронної мережі для розпізнавання тексту на зображенні.

Ця робота розглядає особливості процесу тренування нейронних мереж-класифікаторів, аналізує вплив використання нейронних мереж на точність результатів передбачень засобів розпізнавання тексту. Також у роботі розглянуто загальні принципи тренування нейронних мереж, їх різновиди, переваги та недоліки кожного типу мережі при виконанні завдань певного типу.

Ця робота включає опис методів, використовуваних при підготовці навчального набору даних, способи призначення початкових значень параметрів нейронної мережі та керування їх зміною протягом процесу її тренування.

Процес розробки програмного продукту включає у себе використання вдалих рішень, виявлених при аналізі існуючих програм для розпізнавання тексту. У роботі присутній огляд процесу розвитку технології розпізнавання тексту та впливу використання нейронних мереж на точність цих технологій.

В ході написання роботи було модифіковано існуючий метод пошуку тексту на зображенні та натреновано власну нейронну мережу для розпізнавання символів. Кінцевим призначенням розробки є її використання у складі іншого програмного продукту в якості модуля розпізнавання тексту на зображенні, або використання у навчальних та дослідницьких цілях.

Ключові слова: комп’ютерний зір, нейронна мережа, мережа-класифікатор, машинне навчання, розпізнавання тексту.

ABSTRACT

Pelekh H.V. Research on computer vision algorithms for text recognition under different lighting conditions and image quality. Manuscript

Qualification work for obtaining master's degree in the specialty 122 Computer Science. Lesya Ukrainka Volyn National University, Lutsk, 2024.

The objectives of this qualification work are to study machine learning algorithms for neural networks, overview existing software for text recognition, its typical architecture when using neural networks in it, and develop and train a neural network for text recognition in images.

This work goes through the features of the process of training neural network classifiers, analyzes the impact of using neural networks on the accuracy of the results of predictions of text recognition tools. The work also describes general principles of training neural networks, their varieties, advantages and disadvantages of each type of network when performing tasks of a certain type.

This work includes a description of the methods used in preparing the training data set, methods for assigning the initial values of the neural network parameters and controlling their change during the process of its training.

The process of developing a software product includes the use of successful solutions identified during the analysis of existing programs for text recognition. The work contains an overview of the process of developing text recognition technology and the impact of using neural networks on the accuracy of these technologies.

The process of writing the work included modifying the existing method of searching for text in an image and training a custom neural network for character recognition. The ultimate goal of the development is its use as part of another software product as a module for recognizing text in an image, or its use for educational and research purposes.

Keywords: computer vision, neural network, classifier network, machine learning, text recognition.