МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

15 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-наукової програми «Інформатика»

на тему: «Інформаційна технологія розпізнавання інвентаризаційних та логістичних штрих-кодів»

здобувача групи ІН.м-22 Псарьова Андрія Михайловича

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

Андрій ПСАРЬОВ

_____

(підпис)

Керівник,

старший викладач кафедри Артем КОРОБОВ

_____

(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»

здобувача групи ІН.м-22 Псарьова Андрія Михайловича

1. Тема роботи: «Інформаційна технологія розпізнавання інвентаризаційних та логістичних штрих-кодів»

затверджую наказом по СумДУ від «06» грудня 2023 р. № 1412-VI

2. Термін здачі здобувачем кваліфікаційної роботи до 18 травня 2023 року

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.

2) Огляд технологій, що використовуються для виявлення штрих-кодів на зображеннях.

3) Розробка інтелектуальної системи з виявлення штрих-кодів.

4) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.11.2023-13.11.2023	
2	*Огляд технологій, що використовуються для виявлення штрих-кодів на зображеннях*	14.11.2023-21.11.2023	
3	*Розробка інтелектуальної системи з виявлення штрих-кодів.*	22.11.2023-06.12.2023	
4	*Аналіз отриманих результатів*	07.12.2023-08.12.2023	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	08.12.2023-17.12.2023	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 74 стор., 24 рис., 1 додаток, 23 джерела.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною у зв'язку з постійним розвитком та вдосконаленням систем виявлення та розпізнавання об'єктів на зображеннях.

Об’єкт дослідження — процес розпізнавання штрих-кодів на зображеннях.

Мета роботи — дослідження, розробка та аналіз методів та алгоритмів для розпізнавання штрих-кодів на зображеннях з метою покращення точності та ефективності процесу їх ідентифікації.

Методи дослідження — аналіз сучасних методів розпізнавання штрих-кодів, порівняння та оптимізація алгоритмів.

Результати — розроблено програмне забезпечення, що ефективно впізнає та аналізує штрих-коди на фотографіях. Проведено тестування розробленого програмного засобу на реальних фотографіях з різними типами штрих-кодів, що підтвердило його високу ефективність та точність в розпізнаванні різноманітних штрих-кодів.

ІНФОРМАЦІЙНА СИСТЕМА, РОЗПІЗНАВАННЯ ШТРИХ-КОДІВ, KERAS, PYTHON, TENSORFLOW.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі розвиток технологій штучного інтелекту та машинного навчання, зокрема нейронних мереж, відкриває нові перспективи в різних сферах. Штрих-коди стали важливим елементом ідентифікації та автоматизації процесів у логістиці, роздрібній торгівлі та інших галузях. Розробка систем розпізнавання штрих-кодів на фотографіях за допомогою нейронних мереж є актуальною задачею, оскільки це може значно спростити та прискорити багато бізнес-процесів. Дослідження у цій області має великий практичний потенціал для впровадження нових технологій та покращення ефективності виробництва та обігу товарів.

Об’єкт дослідження. Об’єктом дослідження є процес розпізнавання штрих-кодів на фотографіях за допомогою нейронних мереж.

Предмет дослідження. Предметом дослідження є методи та алгоритми розпізнавання штрих-кодів на зображеннях з використанням нейронних мереж та їх реалізація у вигляді програмного продукту.

Гіпотеза. Використання нейронних мереж для розпізнавання штрих-кодів на фотографіях дозволить розробити систему, яка буде швидко та ефективно ідентифікувати штрих-коди різних форматів на зображеннях.

Наукова новизна. Дослідження спрямоване на вдосконалення процесу розпізнавання штрих-кодів на фотографіях за допомогою застосування нейронних мереж, враховуючи різноманітність типів штрих-кодів та високі вимоги до точності та швидкості ідентифікації.

Структура. Робота складається зі вступу, постановки задачі дослідження, аналізу методик, вибору методики та інструментів для вирішення описаної проблеми, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

В сучасному світі нейронні мережі визнані ключовим інструментом у галузі штучного інтелекту та машинного навчання. Їхній успіх полягає в тому, що вони

імітують роботу людського мозку, створюючи мережу взаємопов'язаних штучних нейронів. Ця інноваційна технологія забезпечує потужний інструмент для розв'язання різноманітних завдань, починаючи від класифікації та прогнозування до розпізнавання образів та автоматизації процесів.

Нейронні мережі можуть оптимально використовуватися в ситуаціях, де традиційні методи досягнення точності стають обмеженими. Функції активації, шари та ваги нейронів дозволяють цим мережам адаптуватися до складних завдань та виконувати їх ефективно.

У цьому контексті нейронні мережі визначають новий рівень гнучкості та універсальності в сферах економіки, медицини, технологій виробництва та багатьох інших. Роль нейронних мереж стає особливо важливою в контексті вирішення завдань, де необхідно працювати з великим обсягом даних та здійснювати аналіз складних зв'язків між ними.

Нейронні мережі відкривають нові можливості у сферах, де вимоги до аналізу та обробки інформації є особливо високими, роблячи їх ключовим інструментом для подальшого розвитку та впровадження інноваційних технологій.

У сучасному світі технології розпізнавання штрих-кодів на фотографіях набули великого значення, охоплюючи різні сфери від логістики до роздрібної торгівлі. Штрих-коди стали потужним інструментом ідентифікації, що дозволяє швидко та точно отримувати інформацію про товари, пристрої та різноманітні об'єкти.

Розробка систем розпізнавання штрих-кодів на фотографіях відіграє важливу роль у сучасній технологічній парадигмі. Використання штучного інтелекту та аналізу зображень для автоматизованого розпізнавання штрих-кодів може значно спростити бізнес-процеси, покращити інвентаризацію та відстеження товарів, а також забезпечити швидке та точне отримання інформації для кінцевих користувачів.

Мета цього дослідження полягає у створенні програмної системи, здатної автоматично розпізнавати штрих-коди на фотографіях. Для досягнення цієї мети передбачено аналіз різних методів розпізнавання, проектування та реалізації моделей нейронних мереж та впровадження методів комп'ютерного зору для точного і швидкого розпізнавання штрих-кодів на зображеннях.

Штучні нейронні мережі вже виявили свою велику потужність у розв'язанні подібних завдань. Використання їхньої гнучкості та ефективності в обробці зображень сприяє створенню систем, які можуть швидко та ефективно ідентифікувати штрих-коди на фотографіях, що робить їх цінним інструментом для різних галузей, від логістики до магазинів та складів.

Одним із ключових завдань цього дослідження є вдосконалення процесу розпізнавання штрих-кодів з використанням технологій нейронних мереж та аналізу зображень, щоб забезпечити точність та швидкість ідентифікації штрих-кодів на різноманітних фотографіях.

Робота складається з чотирьох розділів.

У першому розділі розглядаються основні теоретичні концепції нейронних мереж. Описуються основні принципи їхньої роботи, властивості та структура.

У другому розділі проаналізовані методи застосування нейронних мереж для розв'язання завдання виявлення штрих-кодів на зображеннях. Описані різні підходи та алгоритми, використані для виявлення штрих-кодів.

Третій розділ містить опис процесу розробки та реалізації нейронної моделі для виявлення штрих-кодів. Висвітлені кроки первинної обробки даних, вибір архітектури мережі, методи тренування та тестування моделі. Також наведено аналіз отриманих результатів та їх порівняння з іншими підходами.

У четвертому розділі проведено аналіз створеного програмного продукту, зокрема, оцінка його функціональності та точності. Враховуються переваги та недоліки розробленої системи в контексті виявлення штрих-кодів.

1. НЕЙРОННІ МЕРЕЖІ ТА ЇХ ОСОБЛИВОСТІ

1.1. Огляд основних понять нейронних мереж

1.1.1. Штучний нейтрон

Нейронні мережі — це складні системи обчислень, що створені на основі біологічної будовои людського мозку. Вони складаються зі штучних нейронів, які виконують розрахунки на основі вхідних даних, відтворюючи у спрощеній формі процеси, що відбуваються в біологічних системах [1].

Біологічні нейрони мають складну структуру, що дозволяє їм приймати, обробляти та передавати сигнали. Вони використовують дендрити для прийому сигналів, тіло нейрона для їх обробки та аксони для передачі вихідного сигналу до синапсу наступного нейрона. Ця властивість біологічних нейронів, з'єднуватись один з одним, дає можливість обробляти величезну кількість інформації паралельно. Будову біологічного нейрона зображено на рисунку 1.1.

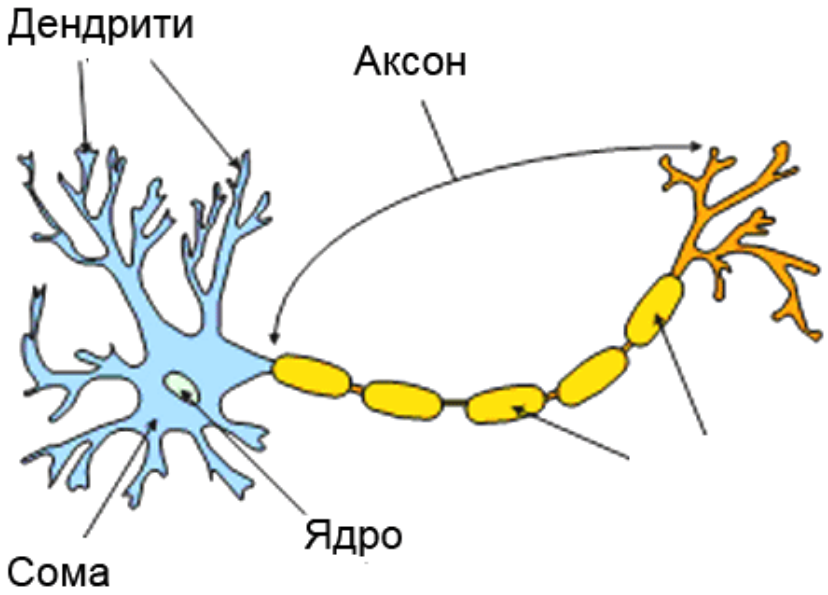


Рисунок 1.1. – Біологічний нейрон

Аналогічно біологічним, штучні нейрони отримують сигнали з інших нейронів та оброблюють їх, використовуючи ваги на вхідних зв'язках. Ці ваги визначають важливість кожного зв'язку, впливаючи на обчислення в штучному нейроні. Штучний нейрон проводить обробку сигналів за допомогою суматора та функції активації, які визначають кінцевий вихід сигналу [2].

Штучні нейрони організовані в шари, де кожен шар може мати свою функцію обробки сигналу. Інформація передається через ці шари від вхідних до вихідних, де кожен нейрон зв'язаний з нейронами наступного шару. Навіть із простою структурою, такі мережі можуть вирішувати складні завдання, виконуючи операції над великим обсягом даних.

Схожість штучного та біологічного нейронів полягає у структурі та основних процесах:

1. Вхід та ваги: Подібно до дендритів біологічного нейрона, штучний нейрон отримує вхідні сигнали через свої входи. Кожен вхід має свою вагу, яка визначає важливість цього сигналу для нейрона.
2. Суматор: Штучний нейрон використовує суматор, аналогічний тілу біологічного нейрона, для обчислення взваженої суми вхідних сигналів, помножених на їх ваги.
3. Функція активації: Після обчислення суми вхідних сигналів, штучний нейрон застосовує функцію активації, яка відповідає за активацію або пригнічення нейрона на основі вхідних даних.
4. Вихід: Остаточний результат передається як вихідний сигнал нейрона і може слугувати входом для інших нейронів у мережі.

Структура штучного нейрона зображена на рисунку 1.2.

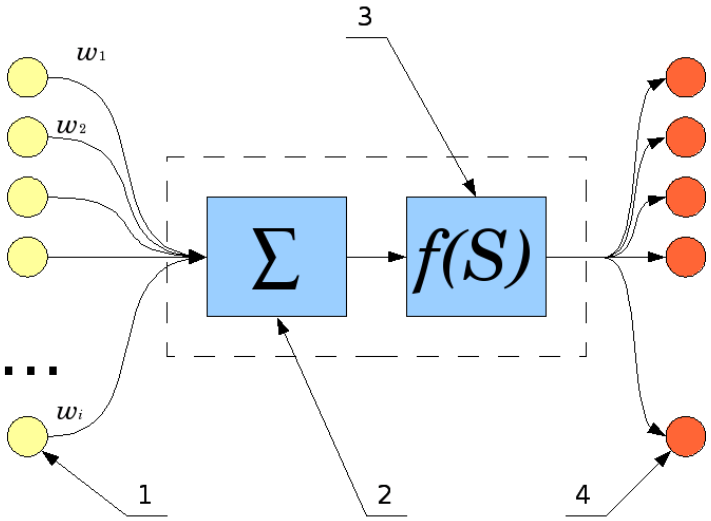


Рисунок 1.2. – Структура штучного нейтрона

Штучний нейрон оперує з ваговими коефіцієнтами (w_i), які визначають силу зв’язків між входами та нейроном. Кожен w_i представляє вагу вхідного сигналу (x_i), яка вказує на його важливість. Суматор (S) обчислює зважену суму вхідних сигналів, множачи кожний вхід на його відповідну вагу та додаючи їх. Формула, що використовується:

$$S = \sum_i w_i x_i$$

Після цього обробка сигналу відбувається через функцію активації (f(S)), яка нелінійно перетворює отриману суму за допомогою формули:

$$Y = f(S)$$

Після цих операцій, результат передається на вихід нейтрона.

У нейронних мережах нейрони зазвичай групуються у шари, де кожен шар виконує свою функцію обробки сигналів. Вхідний шар приймає вхідні дані, вихідний - надає вихідні значення, а між ними можуть бути приховані шари, які допомагають вирішувати складніші завдання, роблять обробку та витягують важливі ознаки з даних. Зв'язки між нейронами допомагають інформації поширюватись від вхідних сигналів до вихідних через обробку на кожному шарі мережі.

1.1.2. Функція активації

Функція активації в нейронних мережах відповідає за вироблення відповіді нейрона на вхідні дані. Коли дані проходять через нейронну мережу, їх обробляють різними математичними операціями, і функція активації визначає, чи має нейрон відгукнутися на ці дані, і, якщо так, то якою саме зміною свого внутрішнього стану.

Основна роль функцій активації полягає в тому, щоб додати нелінійність та неоднорідність у відповіді нейронної мережі. Вони дозволяють моделі вчитися складним залежностям між вхідними та вихідними даними [3].

Розглянемо основні типи функцій активації і їх вплив на виявлення штрих-кодів:

1. **ReLU (Rectified Linear Activation)**

Особливості:

- ReLU активує нейрон, якщо вхідне значення позитивне, інакше встановлює його на нуль: $f(x)=\max(0, x)$.
- Він має швидку обчислювальну швидкість, оскільки використовує просту операцію максимуму.

Переваги:

- Уникнення проблеми виціплення градієнту, що зменшує ймовірність "мертвих" нейронів у мережі.
- Швидке обчислення, що робить його популярним у багатьох моделях.

Недоліки:

- Може бути проблематичним для великих значень вхідних даних, оскільки нейрон перестає вивчати при великих додатних значеннях [4].

2. **Sigmoid**

Особливості:

- Функція перетворює значення у діапазон від 0 до 1: $f(x) = \frac{1}{1+e^{-x}}$.
- Зазвичай використовується для бінарної класифікації, де потрібна ймовірність.

Переваги:

- Добре підходить для вирішення проблеми виціплення градієнту у мережі з невеликою кількістю шарів.

Недоліки:

- Має проблеми з ненасиченістю градієнту, що може ускладнювати навчання глибоких мереж.

3. **Tanh**

Особливості:

- Аналогічно до Sigmoid, але перетворює значення у діапазон від -1 до 1: $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- Зручний для використання у мережах, де потрібне обмеження значень від -1 до 1.

Переваги:

- Допомагає уникнути проблем зі зниканням градієнту у порівнянні з Sigmoid.

Недоліки:

- Так само, як і у випадку Sigmoid, може бути проблема з ненасиченістю градієнту.

4. Leaky ReLU

Особливості:

- Модифікація ReLU, яка дозволяє витікати частково негативним значенням: $f(x)=max(ax, x)$, де a - невелике число, зазвичай дуже маленьке.

Переваги:

- Уникнення проблеми "мертвих" нейронів, які можуть стати проблемою у ReLU.

Недоліки:

- Може бути дещо менш ефективним у порівнянні з ReLU у деяких випадках.

У контексті виявлення штрих-кодів на зображеннях, використання ReLU або його модифікацій може бути корисним, оскільки вони дозволяють уникнути проблем з виціпленням градієнту та активувати нейрони згідно певним умовам вхідних даних. Sigmoid та Tanh можуть бути менш ефективними у глибоких мережах через проблеми з ненасиченістю градієнту [5].

Вплив функцій активації на роботу мережі:

1. Швидкість збіжності навчання: Різні функції активації можуть впливати на швидкість навчання мережі. Наприклад, ReLU швидше збігається через свою простоту, у порівнянні з Sigmoid або Tanh, які мають проблеми з ненасиченістю градієнту.
2. Уникнення проблеми зі зникуанням градієнту: Функції активації типу ReLU допомагають уникнути проблеми зникання градієнту у глибоких мережах. Це може сприяти кращій стабільності та швидкості навчання.

3. Узагальнення мережі: Деякі функції активації можуть сприяти кращому узагальненню мережі. Наприклад, Leaky ReLU може допомогти уникнути "мертвих" нейронів, що можуть знизити здатність мережі узагальнювати.

1.3. Оптимальна конфігурація нейронної мережі для виявлення штрих-кодів

При створенні нейронних мереж для виявлення об’єктів, вибір оптимальної архітектури відіграє ключову роль у досягненні високої точності та ефективності. В розділі, що слідує, розглядається важливий аспект: архітектура мережі. Зокрема, розглянемо кількість шарів та їх типи, а також оптимальний розмір та кількість фільтрів у згорткових шарах. Експерименти з різними параметрами дозволять досягти кращого розпізнавання візуальних штрих-кодів та підвищити ефективність роботи мережі.

Підбір оптимальної архітектури нейронної мережі виявляється вирішальним кроком у створенні системи розпізнавання. Для досягнення цієї мети, розглядаються наступні аспекти:

Кількість шарів та їх типи:

У процесі розробки мережі важливо розглянути різні типи шарів, які можуть бути використані для виявлення об’єктів. Існує безліч варіантів кількості шарів та їх типів у нейронних мережах. Ось деякі з найбільш популярних типів шарів:

Згорткові (Convolutional) шари:

- Згорткові шари (Convolutional layers): Вони використовують фільтри для здійснення згортки по вхідному зображенню, відокремлюючи важливі візуальні особливості. Це може бути корисним для розпізнавання штрих-кодів за їх візуальними атрибутами [6].

- Пулінгові шари (Pooling layers): Вони зменшують просторовий обсяг даних шляхом об'єднання (пулінгу) інформації з попередніх шарів. Це допомагає зменшити кількість параметрів та сприяє стабільності у моделі [7].

Повнозв'язані (Fully Connected) шари:

- Повнозв'язані шари (Fully Connected layers): Це шари, у яких кожен нейрон підключений до кожного нейрона попереднього шару. Вони часто використовуються в кінцевих частинах мережі для обробки виходів попередніх шарів [8].

Шари нормалізації та регуляризації:

- Шари пакетної нормалізації (Batch Normalization layers): Вони нормалізують вхідні дані для кожного батчу, що допомагає уникнути перенавчання та покращує стабільність моделі.
- Шари випадкового вимикання (Dropout layers): Вони випадково "вимикають" частину нейронів під час навчання для запобігання перенавчанню.

Оптимальний вибір шарів та їх кількості для виявлення об’єктів може вимагати досліджень та експериментів для досягнення кращих результатів.

Розмір та кількість фільтрів у згорткових шарах:

Ефективність згорткових шарів визначається розміром та кількістю застосованих фільтрів. Експерименти з різними параметрами фільтрів дозволяють знаходити оптимальний набір для кращого виявлення візуальних штрих-кодів. Отже, важливо дослідити різні комбінації розмірів та кількостей каналів у згорткових шарах для досягнення найкращих результатів у виявленні об’єктів на зображеннях.

Згорткові шари використовують фільтри для виділення важливих особливостей у зображеннях. Розмір фільтра визначає область, яка аналізується, а кількість фільтрів контролює скільки різних особливостей може виявити мережа. Експерименти з різними комбінаціями розмірів та кількості фільтрів у згорткових

шарах дозволяють виявити оптимальний набір параметрів для здійснення кращого виявлення візуальних штрих-кодів на зображеннях.

Наприклад, використання менших розмірів фільтрів може дозволити мережі виявляти більш дрібні деталі, тоді як більші розміри можуть виявити загальні ознаки. Крім того, збільшення кількості фільтрів може розширити спектр особливостей, які мережа може розпізнати.

Правильний вибір розміру та кількості фільтрів у згорткових шарах впливає на точність та ефективність виявлення об’єктів . Оптимальна архітектура мережі для розпізнавання об’єктів на зображеннях вимагає систематичного експериментування та аналізу різних комбінацій параметрів, щоб досягти найкращих результатів.

1.4. Навчання мережі для розпізнавання

1.4.1. Обробка даних

Ефективне навчання нейронної мережі для розпізнавання об'єктів вимагає попередньої обробки та підготовки набору даних. Зазвичай зображення об'єктів можуть мати відмінні розміри та формати. Для забезпечення однаковості та стабільності в аналізі, вони попередньо стандартизуються. Розмір та формат зображень можуть бути адаптовані до єдиного стандарту для спрощення подальшої обробки та аналізу.

Додатково, мітки чи анотації, пов'язані з об'єктами на зображеннях, грають важливу роль у тренуванні моделі. Вони надають інформацію про місцезнаходження та характеристики об'єктів. Отже, ці дані потребують

попередньої структурування та перевірки на коректність для успішного використання під час тренування моделі.

Попередня обробка зображень:

1. **Ресайз (Зміна розміру):**

Зображення можуть мати різні розміри, але модель може вимагати однакових розмірів для ефективного навчання. Зазвичай зображення змінюють до стандартного розміру.

Чому потрібен ресайз:

- Стандартизація розмірів: Нейронні мережі часто потребують вхідні дані однакового розміру. Різні зображення можуть мати різні розміри (висота, ширина), але для навчання моделі важливо, щоб всі дані мали однаковий розмір.
- Оптимізація обчислень: Різні розміри зображень можуть вплинути на швидкість обробки даних під час навчання. Ресайз може спростити обчислення та зробити їх більш ефективними.

Методи ресайзу:

- Зміна пропорцій: Зображення можна зменшити або збільшити, зберігаючи пропорції. Проте, при цьому може виникнути необхідність додатково обрізати або доповнювати зображення, щоб воно мало стандартний розмір.
- Обрізка та доповнення: Деякі методи ресайзу можуть вимагати обрізки частини зображення або його доповнення пікселями, щоб досягти бажаного розміру.

2. **Нормалізація:**

Перед подачею на вхід моделі, значення пікселів можуть бути нормалізовані до певного діапазону, наприклад, [0, 1] або [-1, 1], для полегшення обчислень. Нормалізація - це процес приведення даних до певного стандарту або діапазону, який сприяє покращенню навчання моделі. В контексті обробки зображень для

нейронних мереж нормалізація часто означає перетворення значень пікселів таким чином, щоб вони відповідали певним стандартам або діапазонам.

Основна мета нормалізації - це зробити дані більш стабільними та придатними для навчання, уникнути занадто великих чи малих значень, які можуть ускладнити процес навчання мережі. Також це може допомогти уніфікувати інформацію для різних шарів мережі.

Нормалізація зображень може бути виконана різними способами:

- Масштабування до [0, 1] або [-1, 1]: Цей метод полягає у діленні значень пікселів на максимальне значення (як правило, 255 для кожного кольору у RGB) для отримання значень від 0 до 1 або центрування значень до від -1 до 1.
- Стандартизація (Z-score normalization): Цей метод передбачає віднімання середнього значення від кожного пікселя та поділ на стандартне відхилення всієї вибірки. Це робить середнє значення пікселів близьким до нуля, а стандартне відхилення - близьким до одиниці.

Нормалізація допомагає уникнути зміщення масштабів між функціями активації та вузлами мережі, що може призвести до швидкого навчання та кращої загальної здатності моделі до узагальнення на нові дані.

Аугментація даних:

Для покращення роботи моделі можна використовувати аугментацію, що полягає в створенні додаткових варіантів зображень, зберігаючи при цьому їхні основні характеристики. Це допомагає моделі навчатися на більш різноманітних даних і підвищує її узагальненість.

1.4.2. Розділення даних

Розділення набору даних на тренувальний, валідаційний та тестувальний - це важливий крок у підготовці даних для навчання та оцінки нейронних мереж. Кожен з цих піднаборів виконує свою унікальну функцію в процесі розвитку та оцінки моделі машинного навчання. Розділення набору даних має кілька ключових цілей. Перш за все, це дозволяє моделі вчитися на певних даних, валідувати свою ефективність на інших та остаточно оцінювати свою точність на третій, резервований для тестування, набір даних. Розділення набору даних допомагає уникнути перенавчання моделі та сприяє здатності моделі генералізувати знання на нові, раніше не бачені дані. Крім того, цей процес дозволяє ефективніше оптимізувати параметри моделі та вибирати кращі варіанти, що підвищує її точність та надійність. Збалансованість класів у кожному з піднаборів даних, а також можливість тестування на різних наборах, дозволяють отримати більш достовірну та репрезентативну оцінку ефективності моделі. Таким чином, правильне розділення набору даних визначає не лише якість навчання моделі, а й її здатність до точного прогнозування на нових даних.

Тренувальний набір

Тренувальний набір є основною частиною для навчання моделі. Ці дані використовуються для налаштування ваг моделі та її оновлення під час процесу навчання. Чим більший та різноманітний тренувальний набір, тим краще модель може навчатися розпізнавати широкий спектр об'єктів.

Валідаційний набір

Валідаційний набір використовується для налаштування гіперпараметрів та оцінки ефективності моделі під час навчання. Це допомагає уникнути перенавчання та перевірити, наскільки добре модель генералізує знання на нових даних, які не були використані під час навчання.

Тестувальний набір

Тестувальний набір використовується для кінцевої оцінки роботи моделі. Цей набір даних не використовується в процесі навчання або налаштування

параметрів. Використання тестового набору дозволяє оцінити реальну ефективність моделі на нових, раніше не бачених даних.

Збалансованість класів та рандомізація

Збалансованість класів у кожному з піднаборів - це важливий аспект при розділенні даних. Це допомагає уникнути перекосу моделі у відповідь на конкретні типи об'єктів та забезпечити, що модель навчається розпізнавати всі класи об'єктів з однаковою увагою.

Рандомізація під час розділення набору даних допомагає уникнути впливу конкретного порядку даних на процес навчання моделі. Це сприяє покращенню узагальненості моделі та забезпеченню того, що вона робить прогнози на основі справжніх закономірностей, а не лише на основі порядку даних.

1.5 Метод зворотнього розповсюдження помилки

Зворотнє розповсюдження помилки - це ітеративний процес, що використовується для навчання штучних нейронних мереж. Його мета - мінімізувати помилку між фактичними виходами мережі та бажаними значеннями. Цей алгоритм особливо ефективний для багатошарових перцептронів з диференційованими функціями активації.

Процес розпочинається ініціалізацією ваг нейронної мережі на початкових значеннях, часто випадкових. На кожній ітерації, яку називають епохою, вхідні дані подаються через мережу, отримуються відповіді та обчислюється загальна помилка моделі [9].

На початку процесу, ваги мережі встановлюють на певному, зазвичай невеликому та випадковому, значенні

$$\{\Delta w_{ij}\}_{i,j} = 0, \tag{1.1}$$

Подаємо вхідний сигнал x_i та отримуємо вихід уі для кожного шару мережі.

Обчислення похибки

$$E = \frac{1}{2}\sum_{i=1}^{n}(y_i - t_i)^2 \quad (1.2)$$

де t_i - це бажані виходні значення.

Після подачі вхідного сигналу та отримання виходу та значення похибки проводиться корегування ваг.

Обчислення помилки для виходного шару (дельта для виходу):

$$\delta_i = y_i(1 - y_i)(t_i - y_i) \quad (1.3)$$

Тут δ_i - це помилка для кожного виходу.

Обчислення помилки для прихованих шарів:

$$\delta_i = y_i(1 - y_i) \sum_j \in_{i,j} \delta_j \omega_{i,j} \quad (1.4)$$

Для шарів прихованого рівня δ_i - це помилка вузла, $\in_{i,j}$ - посилання на попередній шар, δ_j - помилка виходу попереднього шару, $\omega_{i,j}$ - вага між вузлами.

Оновлення ваг:

$$\omega_{i,j} = \omega\omega_{i,j} + \Delta\omega_{i,j} \quad (1.5)$$

$$\Delta\omega_{i,j} = -\eta\delta_i x_i \quad (1.6)$$

Тут $\Delta\omega_{i,j}$ - це зміна ваги, η - швидкість навчання, x_i - вихідний сигнал вузла і.

Повторення процесу:

Ця послідовність дій виконується протягом кожної епохи до досягнення бажаної точності, або доки помилка майже не змінюється, або до досягнення максимальної кількості епох [10].

1.6 Проблема перенавчання

В процесі використання тренувальних даних для навчання з учителем, ми стикаємося з проблемою, коли модель стає надто пристосованою до тренувальних даних і втрачає здатність до точного прогнозування реальних даних. Це називається перенавчанням мережі [11].

Це часто виникає у складних та глибоких моделях з багатьма параметрами та шарами. Для виявлення цієї проблеми часто застосовують підхід, де вихідні дані розбиваються на тренувальний та тестовий набори. Співвідношення цього розділу залежить від конкретної задачі та вхідних даних. Тестові дані використовуються для оцінки ефективності моделі після її навчання на тренувальних даних.

Для вирішення цієї проблеми або запобігання їй використовують методи, такі як перехресне затвердження (cross-validation), ранню зупинку навчання моделі до моменту перенавчання (наприклад, коли точність на тестовому наборі даних не покращується), регуляризацію мережі та порівняння з результатами інших моделей. Також застосовуються методи, які змінюють основні параметри моделі або її архітектуру, наприклад, використання випадкового відкидання нейронів у шарах.

1.7. Архітектури нейронних мереж

Нейронні мережі прямого поширення

Нейронні мережі прямого поширення (Feedforward Neural Networks) є одними з найбільш поширених та базових моделей у галузі глибокого навчання. Ці мережі мають просту структуру та використовуються для вирішення різноманітних завдань, починаючи від класифікації до регресії та вирішення складних задач обробки природної мови та зображень [12].

У нейронних мережах прямого поширення інформація переміщується лише в одному напрямку, без циклів або зворотніх зв'язків. Найпростіша модель перцептрона без прихованих шарів зображена на рисунку:

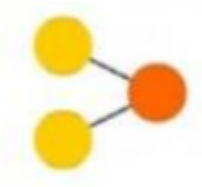


Рисунок 1.3. – Перцептрон без прихованих шарів

Частіше модель складається з трьох типів шарів: вхідного, прихованих та вихідного. Вхідний шар приймає дані, приховані шари здійснюють обчислення, а вихідний шар повертає результати.

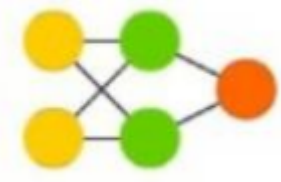


Рисунок 1.4. – Перцептрон з прихованим шаром

Мережі прямого поширення складаються з великої кількості нейронів, організованих у шари. Кожен нейрон у попередньому шарі пов'язаний з кожним нейроном у наступному шарі, передаючи сигнали через ваги, які змінюються під час навчання для досягнення оптимального результату [13].

Коли маємо більше одного прихованого шару в нейронній мережі, така структура отримує назву "глибокої нейронної мережі" (DNN) або "Deep Neural Network" [14]. Глибокі нейронні мережі володіють властивістю більшої глибини обробки інформації порівняно з традиційними нейронними мережами з одним чи невеликою кількістю прихованих шарів.

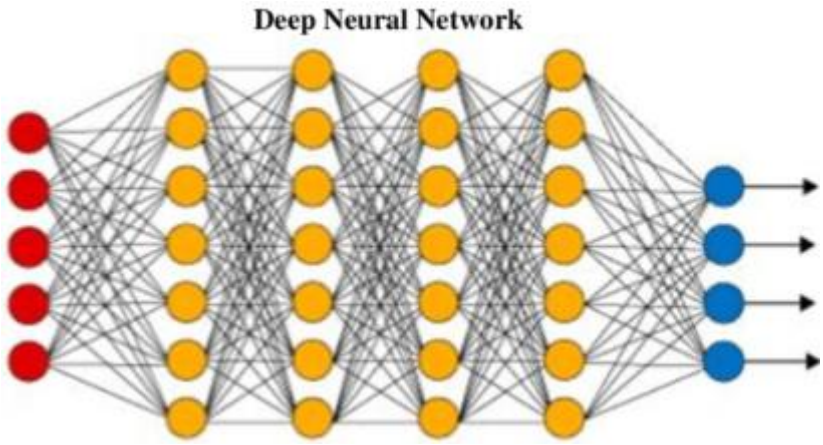


Рисунок 1.5. – Глибока нейронна мережа

Глибокі мережі дозволяють враховувати складні та абстрактні залежності в даних через послідовне представлення прихованих шарів. Кожен прихований шар може відбувати власні абстракції на основі вхідних даних, побудовуючи більш складні та інформативні представлення. Це дозволяє глибоким мережам автоматично вивчати властивості даних та відображати їх у власних шарах, що часто сприяє вдосконаленню точності прогнозування та робить їх більш адаптивними до різних складних завдань.

Одношарові та багатошарові перцептрони є двома основними формами штучних нейронних мереж, кожна з яких має свої власні переваги та обмеження. Одношарові перцептрони мають просту структуру, яка дозволяє їм легко вивчати та реалізовувати. Вони можуть швидко навчатися у випадках, коли дані лінійно роздільні, і мають обмежену кількість параметрів. Однак такі перцептрони не можуть вирішити складні не лінійно роздільні задачі та не можуть ефективно працювати зі завданнями типу XOR.

З іншого боку, багатошарові перцептрони, що мають більше одного прихованого шару, називаються глибокими нейронними мережами. Ці мережі можуть розв'язувати складні не лінійні проблеми та моделювати складні залежності у вхідних даних, що робить їх ефективними для різноманітних завдань. Однак багатошарові перцептрони схильні до перенавчання, особливо у разі недостатньої кількості даних для ефективного навчання всіх параметрів.

Також вони можуть потребувати багато часу та обчислювальних ресурсів для навчання через більшу складність [15].

Згорткові нейронні мережі

Згорткові нейронні мережі (CNN або Convolutional Neural Networks) є одним з ключових типів багатошарових нейронних мереж прямого поширення, які відрізняються своєю здатністю ефективно працювати з великими обсягами вхідних даних без значної попередньої обробки. Основна їх особливість полягає в використанні згорткових шарів, які складаються з фільтрів. Операція згортки дозволяє виконати операцію обчислення скалярного добутку між фільтром та вхідними даними, створюючи карту збудження фільтру [16].

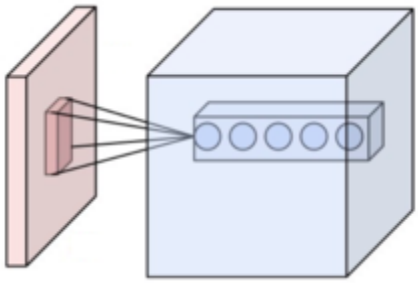


Рисунок 1.6. – Згортковий шар

Ця мережа включає не лише згорткові шари, але й шари субдискретизації, що дозволяють зменшити розмірність вхідного сигналу, використовуючи операції, такі як усереднення або вибір максимуму. Крім того, CNN включають звичайні повнозв’язні шари, які використовуються для фінальної класифікації вихідних даних.

Головною перевагою згорткових нейронних мереж є їхній здатний обробляти великі обсяги даних із зменшеною кількістю параметрів, що допомагає полегшити процес навчання та уникнути проблем перенавчання. Цей тип мережі зазвичай застосовується для завдань класифікації та генерації зображень, де ефективне розпізнавання та аналіз великих обсягів даних є ключовим.

Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) представляють собою архітектуру нейронних мереж, які відрізняються від звичайних нейронних мереж прямого поширення наявністю циклічних зв'язків між нейронами. Це означає, що вихід одного нейрона може слугувати вхідним сигналом для нього самого в наступному моменті часу, утворюючи послідовність даних залежності в часі [17].

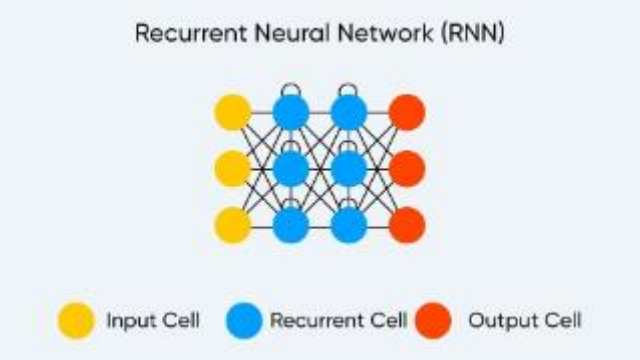


Рисунок 1.7. – Рекурентна нейронна мережа

Однією з головних переваг рекурентних мереж є їх здатність моделювати плин часу і робити передбачення в залежності від попередньої інформації. Це робить їх ефективними для обробки послідовних даних, таких як часові ряди, текстові послідовності або дані зі структурою контексту. Додатково, RNN володіють Тьюрінг-повнотою, що означає їхню здатність виконувати будь-які обчислення.

Однак у рекурентних мереж є проблема зникновення або вицвітання градієнту (vanishing or exploding gradient problem) під час тренування, особливо в разі великої кількості часових кроків. Ця проблема може призвести до складнощів у збереженні пам'яті та долі навчання мережі. Для її вирішення були розроблені покращені варіанти RNN, такі як мережі довгої короткочасної пам'яті (LSTM) [18].

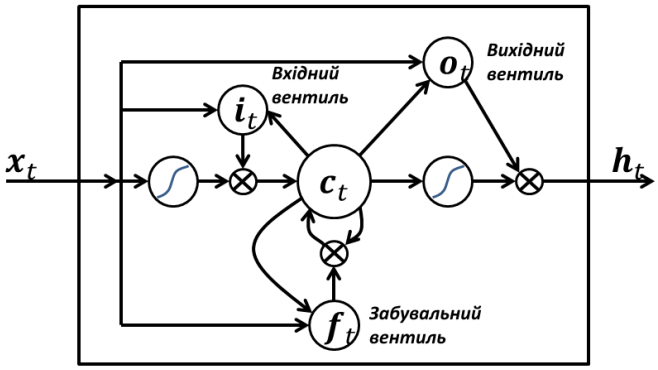


Рисунок 1.8. – Схема мережі довгої короткострокової пам’яті

Мережі довгої короткочасної пам'яті (LSTM) є модифікацією RNN, призначеною для ефективного збереження та використання пам'яті протягом тривалого часу. Вони мають спеціальні структури внутрішніх блоків, які дозволяють мережі вирішувати проблему зникання градієнту та здійснювати більш довгострокові залежності.

Отже, LSTM представляють собою потужний інструмент для обробки послідовних даних та збереження контексту в нейронних мережах, а їхній успіх полягає у здатності ефективно працювати з довгими часовими залежностями в данних.

Single Shot MultiBox Detector

SSD - це архітектура нейронних мереж, яка використовується для виявлення об'єктів на зображеннях. Вона спеціалізується на виконанні двох ключових завдань: локалізації (визначення місцезнаходження об'єктів) та класифікації (визначення категорії об'єктів).

Основна ідея SSD полягає у використанні набору згорткових шарів, які аналізують зображення на різних масштабах та рівнях деталізації. Ці шари генерують пропозиції (англ. proposals) щодо місцезнаходження об'єктів різних розмірів і форм, використовуючи так звані anchor boxes та конволюційні фільтри.

Після отримання пропозицій SSD використовує комбінацію згорткових та пулінгових шарів для визначення класу об'єктів у кожній пропозиції та коригування їх місцезнаходження. Це дозволяє одночасно виявляти об'єкти різних типів та розмірів на зображенні.

Однією з ключових переваг SSD є його здатність ефективно впоратися з реальними часами обробки зображень, завдяки оптимізації процесу виявлення об'єктів. Використання згорткових шарів на різних рівнях абстракції дозволяє моделі SSD впевнено розпізнавати об'єкти на зображеннях з різними контекстами та складностями.

Inception (GoogLeNet)

одна з перших глибоких згорткових нейронних мереж, розроблена компанією Google. Її основна ідея полягає в тому, щоб ефективно використовувати інформацію на різних рівнях деталізації зображення.

Однією з ключових особливостей Inception є використання модулів "інцепцій". Ці модулі представляють собою велику кількість різних згорткових фільтрів (зазвичай 1x1, 3x3, 5x5) та пулінгових операцій, які обробляють одну й ту ж вхідну інформацію паралельно. Таке паралельне використання фільтрів різного розміру дозволяє мережі одночасно виявляти різні візуальні особливості на різних рівнях абстракції.

Іншою важливою характеристикою є використання 1x1 згорткових шарів перед використанням більших фільтрів. Це дозволяє зменшити кількість параметрів та обчислювальний обсяг, оптимізуючи процес роботи мережі.

Ще однією цікавою особливістю Inception є використання "блоків згорткових еквівалентів", які дозволяють замінювати складні згорткові шари більш простими структурами, зберігаючи при цьому продуктивність та точність.

Inception стала досить популярною архітектурою завдяки своїй здатності ефективно використовувати обчислювальні ресурси та виявляти різні візуальні особливості на різних рівнях.

1.8. Постановка задачі

Основні задачі роботи:

- провести огляд основних архітектур та особливості будов штучних нейронних мереж;
- розглянути підходи до проблеми розпізнавання штрих-кодів на зображеннях;
- обрати набір даних для навчання та тестування моделі;
- розробити ПЗ у вигляді нейронної мережі для розпізнавання штрих-кодів на зображеннях;
- провести прогнозування та проаналізувати отримані результати.

1.9. Висновки до розділу 1

В результаті аналізу та дослідження основних аспектів нейронних мереж в даній роботі було проведено глибоке ознайомлення з основними поняттями та принципами їх функціонування. Відповідно до проведеного огляду, було розглянуто основні аспекти створення та налаштування нейронних мереж, включаючи конфігурацію, методи навчання, та проблематику перенавчання.

Один з ключових аспектів вивчення нейронних мереж становить розділ, присвячений їх архітектурі. Виявлено, що різноманітність архітектур, таких як прямі, згорткові та рекурентні нейронні мережі, впливають на їхню здатність вирішувати різноманітні завдання.

Окрім того, були розглянуті питання навчання нейронних мереж, зокрема методи зворотного розповсюдження помилки та проблема перенавчання, яка виникає при використанні складних моделей.

Отже, результати цієї роботи створили тверду базу для подальших досліджень у галузі нейронних мереж, а також можуть слугувати важливим вихідним пунктом для розробки та застосування нейронних мереж у виявленні штрих-кодів та подібних задачах розпізнавання об'єктів.

2. ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ У ВИЯВЛЕННІ ШТРИХ-КОДІВ

2.1. Загальна проблематика виявлення штрих-кодів на фотографіях

Сфера розпізнавання штрих-кодів має важливе значення для бізнесу та споживачів. Залежно від конкретних типів штрих-кодів і їх застосування, складність завдання може варіюватися. Враховуючи різні формати, стандарти та особливості штрих-кодів, процес їх розпізнавання може потребувати різних підходів та алгоритмів.

Розпізнавання штрих-кодів може бути викликане різними факторами, такими як розмір, якість зображення, типи штрих-кодів та їх структура. Для точного та швидкого розпізнавання потрібні ефективні алгоритми обробки зображень, які враховують різноманітні умови зйомки та можливі помилки.

Суб’єктами процесу розпізнавання штрих-кодів можуть бути різні сектори промисловості, логістики, торгівлі тощо. Метою розпізнавання може бути отримання інформації про продукт (ціну, характеристики, походження), автоматизація процесів логістики та відстеження постачання товарів, що є критичним для оптимізації бізнес-процесів.

Аналіз залежності різних факторів, таких як різні формати штрих-кодів, умови зйомки, типи помилок, може виявитися важливим для розробки надійних та ефективних алгоритмів розпізнавання. Такий аналіз дозволяє покращити точність та швидкість роботи систем розпізнавання штрих-кодів, що має значення для їх широкого застосування в різних галузях та сферах.

2.1.1. Огляд сучасних методів виявлення штрих-кодів

1. Класичні методи розпізнавання:

Класичні методи розпізнавання штрих-кодів використовують ряд технік та алгоритмів обробки зображень для виділення штрих-кодів на фотографіях чи зображеннях. Ці методи зазвичай базуються на простих операціях обробки зображень, які дозволяють визначати та розшифровувати інформацію, закодовану у штрих-коді.

Деякі з найпоширеніших класичних методів розпізнавання включають:

- Бінаризація зображення: Цей підхід полягає у перетворенні зображення у чорно-біле, де області, що представляють штрих-коди, стають чорними пікселями, а фон - білими. Це дозволяє легше виділяти контури штрих-кодів для подальшого аналізу [20].
- Фільтрація та обробка зображень: Застосування фільтрів для підвищення контрастності та виділення штрих-кодів, а також обробка для зменшення шуму чи виправлення спотворень.
- Порогові методи: Використання певного порогового значення для виділення штрих-коду, коли пікселі досягають певного яскравості чи контрастності.
- Детектори країв: Виявлення контурів об'єктів на зображенні, у тому числі і штрих-кодів, що може сприяти їх виділенню та подальшому розпізнаванню.

Ці методи можуть бути ефективними в певних умовах, коли якість зображення висока та штрих-коди чітко видимі. Однак вони можуть мати обмежену ефективність у випадках низької якості зображень, наявності спотворень, низькому контрасті або навіть у складних умовах освітлення.

В останні роки з появою глибинного навчання та нейронних мереж ці класичні методи отримують конкуренцію від більш складних та адаптивних підходів, які можуть ефективніше працювати навіть у складних умовах роботи з зображеннями штрих-кодів.

2. Машинне навчання та нейронні мережі:

- Машинне навчання: Машинне навчання включає в себе алгоритми, які можуть вдосконалювати свою продуктивність використанням великої кількості даних. Це метод, який дозволяє комп'ютерам вчитися на основі даних та вдосконалювати свою продуктивність без явного програмування. У контексті розпізнавання штрих-кодів, моделі машинного навчання можуть бути навчені розпізнавати особливості та шаблони штрих-кодів, використовуючи великі набори зображень з різними типами штрих-кодів та умовами зйомки. Це дозволяє їм ефективно впізнавати штрих-коди навіть у складних умовах, де традиційні методи можуть бути менш ефективними [21].
- Нейронні мережі: Глибинне навчання, або нейронні мережі, представляють собою підтип машинного навчання, який моделює роботу людського мозку. У випадку розпізнавання штрих-кодів, нейронні мережі використовуються для автоматичного вивчення характеристик штрих-кодів на основі великої кількості тренувальних зображень. Вони пристосовуються до різних форматів та умов зйомки, що робить їх ефективними у виявленні штрих-кодів навіть у складних умовах, таких як змінне освітлення, спотворення або різні типи штрих-кодів.

2.1.2. Визначення складнощів виявлення штрих-кодів на зображеннях

Визначення складнощі виявлення штрих-кодів на зображеннях включає аналіз різних аспектів, що впливають на успішність процесу розпізнавання цих кодів. Особливо в контексті комп'ютерного зору та обробки зображень існують деякі фактори, що роблять виявлення штрих-кодів завданням складним та вимагають специфічних рішень.

Перш за все, складність полягає у різноманітності форматів та типів штрих-кодів. Існує безліч різних стандартів штрих-кодів: EAN-13, UPC-A, QR-коди, Data

Matrix та інші. Кожен з них має власні особливості, що включають в себе різницю у розмірі, кількості інформації, способів кодування та умов роботи. Розпізнавання різних форматів вимагає спеціальної обробки та адаптації алгоритмів для кожного типу коду.

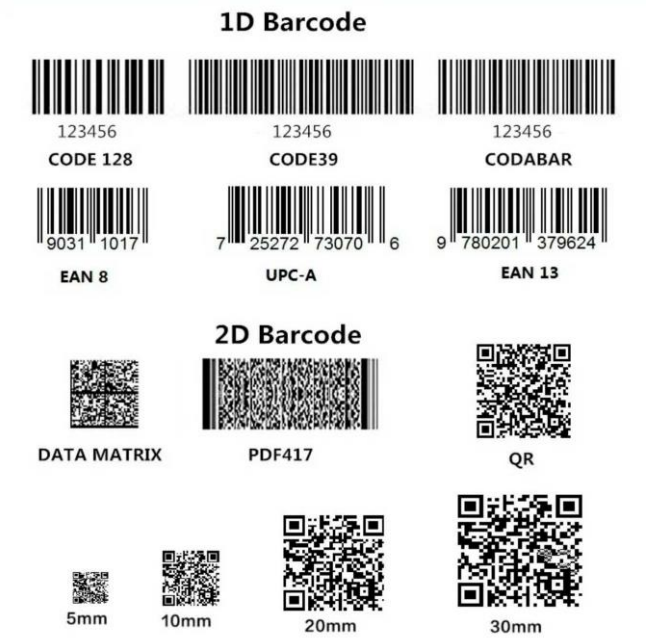


Рисунок 2.1. – види штрих-кодів

Другий важливий аспект - це умови зйомки. Штрих-коди можуть знаходитися на різних поверхнях, мати різний розмір, бути пошкодженими або спотвореними. Виклики виникають у разі поганого освітлення, тіней, різниці у контрастності, а також при зміні перспективи чи розміру штрих-коду. Це робить процес виявлення та розпізнавання складним завданням, оскільки алгоритми повинні бути гнучкими та адаптивними до різних умов зйомки.

Крім того, швидкість виявлення штрих-кодів також є важливим фактором. У випадку реального часу або швидкого сканування, алгоритми повинні бути ефективними та швидкими, щоб у реальному часі виявляти та обробляти інформацію з штрих-кодів.

Остаточно, складність виявлення штрих-кодів на зображеннях постає через різноманітність форматів, умови зйомки та потребу у високій швидкості реакції.

Подолання цих труднощів вимагає розробки алгоритмів та методів, які можуть ефективно працювати в різних умовах та з різними типами штрих-кодів.

2.2. Переваги використання нейронних мереж для виявлення штрих-кодів

Потенціал нейронних мереж у сфері розпізнавання штрих-кодів величезний, оскільки вони забезпечують високу точність та адаптивність до різних умов зйомки та типів штрих-кодів. У цьому розділі досліджуються переваги та можливості використання нейронних мереж для точного та ефективного виявлення штрих-кодів на зображеннях. Використання нейронних мереж для виявлення штрих-кодів має ряд вагомих переваг, які роблять цей підхід вельми привабливим у порівнянні з іншими методами:

- Адаптивність до різних умов:

Нейронні мережі проявляють високу адаптивність до різноманітних умов зйомки. Вони можуть пристосовуватися до різних освітлених умов, кутів та розмірів штрих-кодів, навіть при їх пошкодженні або спотворенні. Ця гнучкість дозволяє нейронним мережам ефективно розпізнавати штрих-коди в різних сценаріях, що робить їх важливим інструментом у широкому спектрі застосувань.

- Узагальнення та навчання:

Нейронні мережі можуть узагальнювати знання, яке вони отримали від тренувальних даних, та використовувати його для розпізнавання нових штрих-кодів. Це означає, що вони можуть розпізнавати штрих-коди, які раніше не були представлені у навчальному наборі даних, підвищуючи свою ефективність та точність.

- Здатність роботи з великим обсягом даних:

Нейронні мережі можуть ефективно обробляти великі масиви даних. Це особливо важливо у випадках, коли потрібно обробляти великі потоки зображень з штрих-кодами, як у випадку сканування товарів на касах в супермаркетах або у системах ідентифікації та відстеження.

- Висока точність розпізнавання:

Глибинне навчання, що використовується у нейронних мережах, може дати вражаючі результати у точності розпізнавання. Ці мережі можуть виявляти штрих-коди з високою точністю, навіть у складних умовах, забезпечуючи надійність у роботі.

- Підтримка реального часу:

Деякі архітектури нейронних мереж можуть бути оптимізовані для роботи в реальному часі, що робить їх ідеальними для задач, де потрібно швидко виявляти та обробляти штрих-коди, наприклад, у сканерах або мобільних додатках.

Загальною перевагою використання нейронних мереж для розпізнавання штрих-кодів є їхня гнучкість, точність та здатність пристосовуватися до різних умов, що робить їх потужним інструментом для рішення різноманітних завдань у сфері розпізнавання штрих-кодів.

2.2.1. Аналіз ефективності нейронних мереж у виявленні штрих-кодів

Аналіз ефективності нейронних мереж у виявленні штрих-кодів — це комплексна оцінка їхньої працездатності та точності в роботі з цими символами на зображеннях. Оцінка включає детальний аналіз кількох аспектів, що відображають потужність та обмеження мереж.

Точність розпізнавання - це ключовий показник ефективності. Вона є ключовою метрикою у вимірюванні ефективності систем розпізнавання, включаючи нейронні мережі для виявлення штрих-кодів. Вона визначає, наскільки вірно система ідентифікує цільові мітки (у нашому випадку — штрих-коди) серед усіх виявлених міток.

Метрики точності, такі як accuracy (точність) та F1-score, часто використовуються для оцінки.

- Accuracy — це відсоток правильно визначених штрих-кодів усіх штрих-кодів, виявлених моделлю. Вона може бути доброю метрикою, якщо класи (типи штрих-кодів) рівномірно розподілені у наборі даних, проте не є найкращою, якщо класи несбалансовані.
- F1-score — це гармонічне середнє між точністю (precision) та повнотою (recall). Precision визначає відсоток правильно визначених штрих-кодів серед усіх, що були визначені як штрих-коди моделлю. Recall вимірює відсоток правильно визначених штрих-кодів серед усіх реальних штрих-кодів у наборі даних [22].

Час обробки у контексті розпізнавання штрих-кодів — це час, який система потребує для аналізу та виявлення штрих-кодів на зображеннях. Важливою є ефективність роботи системи, особливо в сучасному світі, де швидкість виконання завдань часто є критичною.

Стійкість до помилок визначає, наскільки надійно система впізнає штрих-коди навіть у випадках, коли вони пошкоджені, змінені, або коли умови зйомки не є ідеальними. Наприклад, штрих-код може бути частково закритий, розмитий, або відсутність освітлення може ускладнити процес виявлення. Стійкість до таких ситуацій важлива для практичного застосування системи.

Оптимізація часу обробки та забезпечення високої стійкості до помилок — ключові аспекти, які розглядаються під час розробки системи розпізнавання

штрих-кодів. Для вирішення цих завдань можуть використовуватися різні методи оптимізації алгоритмів обробки зображень, а також покращення навчання моделей для більш точного розпізнавання штрих-кодів у різних умовах зйомки.

Стійкість до помилок може досягатися через застосування технологій покращеного фільтрування зображень, адаптивних методів розпізнавання, чи роботи з більш детальною інформацією про штрих-коди та їх особливості.

Оптимізація часу обробки може включати в себе використання більш швидких алгоритмів обробки зображень, оптимізованих обчислювальних процедур, або застосування методів побудови моделей, які ефективно працюють при обробці великого потоку даних.

Ці аспекти мають велике значення для реалізації ефективних та практично застосовних систем розпізнавання штрих-кодів у різних умовах використання.

2.2.2. Порівняння з іншими методами розпізнавання

Порівняння нейронних мереж із іншими методами розпізнавання штрих-кодів є ключовим аспектом визначення їхньої ефективності та застосовності. Традиційні методи, такі як класичні алгоритми обробки зображень часто базуються на екстракції особливостей, які потім використовуються для розпізнавання. Однак, вони можуть бути менш ефективними у складних умовах зйомки або при виявленні пошкоджених штрих-кодів. Методи машинного навчання на основі класичних алгоритмів можуть вимагати вручну створених правил для екстракції функцій, що ускладнює їх адаптацію до різних типів штрих-кодів. Навпаки, нейронні мережі можуть автоматично вивчати ці особливості,

адаптуватися до різних форматів штрих-кодів та показувати високу точність розпізнавання в умовах, де інші методи можуть справлятися менш ефективно.

Генетичні алгоритми, нечітка логіка та експертні системи використовують різні підходи до вирішення задач розпізнавання.

Генетичні алгоритми базуються на ідеї еволюції: вони використовують перетворення та відбір найкращих рішень шляхом спадковості. У порівнянні з нейронними мережами, генетичні алгоритми можуть бути менш ефективними при розпізнаванні штрих-кодів через необхідність оптимізації параметрів алгоритму та відбору найкращих "генетичних" послідовностей.

Нечітка логіка, натомість, оперує з нечіткими концепціями та правилами, дозволяючи моделювати нечіткі або приблизні зв'язки. Вона може бути ефективною у вирішенні проблем з нечіткими даними, але у розпізнаванні штрих-кодів може виявитися менш адаптованою до різних умов зйомки та штрих-кодів.

Експертні системи базуються на правилах, розроблених експертами у конкретній галузі. Вони можуть бути ефективними, але можуть мати обмежену здатність адаптуватися до нових сценаріїв або непередбачуваних ситуацій.

У порівнянні з цими методами, нейронні мережі володіють здатністю до самонавчання та адаптації до різноманітних умов, роблячи їх потужним інструментом для розпізнавання штрих-кодів у різних ситуаціях.

2.3. Альтернативні методи та їх порівняння з нейронними мережами в контексті виявлення штрих-кодів

2.3.1. Генетичний алгоритм

Генетичний алгоритм (Genetic Alghrithm), використовується для розв’язання завдань з моделювання та оптимізації. Він працює шляхом комбінування випадкових параметрів і відбору найкращих з них, використовуючи принцип природного відбору. Він є еволюційним алгоритмом оптимізації, і його роботу можна описати за допомогою п'яти основних етапів: ініціалізація популяції, відбір, схрещування, мутація та оцінювання. ГА базується на природних процесах еволюції та спадковості, що дозволяє йому ефективно шукати оптимальні рішення в складних просторах параметрів [23].

Генетичні алгоритми можуть застосовуватися для пошуку штрих-кодів на зображеннях. Вони можуть бути використані для оптимізації параметрів алгоритмів обробки зображень, які використовуються для виявлення штрих-кодів.

- Пошук оптимальних параметрів обробки зображень: Генетичні алгоритми можуть шукати оптимальні параметри, такі як розмір ядра фільтру, порогові значення для бінаризації, або параметри морфологічних операцій, які допомагають виділити штрих-коди на зображеннях.
- Вибір найкращих алгоритмів обробки: Генетичні алгоритми можуть бути використані для вибору найкращих алгоритмів обробки зображень для конкретної задачі розпізнавання штрих-кодів. Вони можуть проводити оптимізацію шляхом комбінації різних методів обробки зображень для досягнення оптимального результату.
- Автоматизований пошук шляхом еволюції: Генетичні алгоритми можуть ефективно виконувати автоматизований пошук оптимальних параметрів або алгоритмів обробки зображень. Вони використовують принципи еволюції (кроссовер, мутація, відбір) для покращення результатів пошуку штрих-кодів на зображеннях.

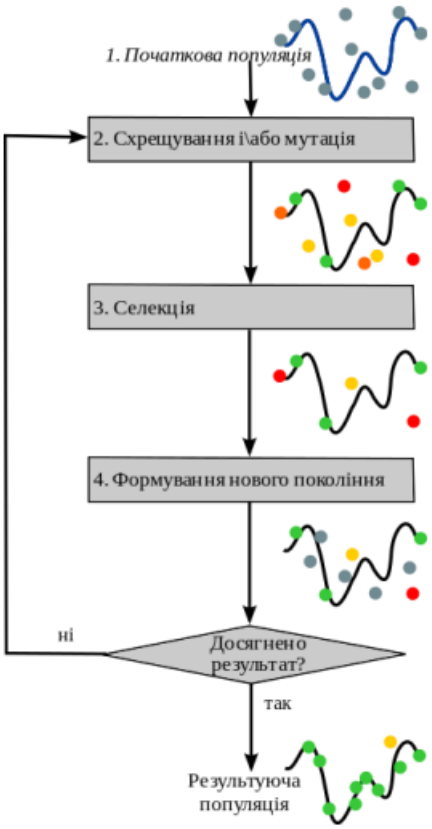


Рисунок 2.2. – Генетичний алгоритм

2.3.2. Мережі із нечіткою логікою

Нейро-нечіткі мережі (FN, Fuzzy Network) або мережі із нечіткою логікою – системи, що об’єднують у собі системи із нечіткою логікою та нейронні мережі.

- Нечітка логіка: Це модель, яка дозволяє враховувати нечіткість у висловлюваннях. Наприклад, вона дозволяє описувати поняття "висока температура" або "швидкість" з точки зору нечітких множин та принципів приналежності до певних категорій.
- Нейронні мережі: Це моделі, що імітують роботу мозку та здатність використовувати велику кількість даних для вирішення завдань. Вони володіють здатністю навчання на прикладах та адаптації до нової інформації.

Нейро-нечіткі мережі використовуються для розв'язання завдань, де даних не завжди достатньо або коли вони не є точними. В контексті виявлення штрих-кодів на фотографіях, цей підхід може бути корисним у роботі з зображеннями різної якості, освітленням, чіткістю штрих-кодів та іншими факторами, що можуть впливати на їхню якість.

Нейро-нечіткі мережі можуть враховувати різноманітні умови освітлення, різний контраст, шуми на зображенні та інші фактори, що можуть впливати на чіткість штрих-коду.

Нейро-нечіткі мережі можуть враховувати ці аспекти, працюючи з великим набором даних, щоб здійснювати аналіз та розпізнавання штрих-кодів. Їхні можливості у виявленні штрих-кодів полягають у здатності адаптуватися до різних умов та забезпеченні більш гнучкого та точного розпізнавання навіть у складних умовах.

2.4. Висновки до розділу 2

У даному розділі ми зосередили увагу на аналізі та порівнянні різних методів виявлення штрих-кодів на фотографіях. Огляд сучасних методів, їхнє порівняння та визначення складнощів у виявленні штрих-кодів на зображеннях розширили уявлення про різні можливості та особливості використання підходів у цій галузі.

Аналіз ефективності нейронних мереж у виявленні штрих-кодів підкреслив їхню потужність та високу точність у порівнянні з альтернативними методами, включаючи генетичні алгоритми та мережі із нечіткою логікою. Виявлення можливостей та обмежень різних типів штрих-кодів у контексті їх виявлення нейронними мережами підкреслило значущість врахування специфіки кодування під час розробки алгоритмів виявлення.

Отже, аналіз та порівняння методів виявлення штрих-кодів дозволило виявити переваги та обмеження нейронних мереж у цій сфері. Розглянуті аспекти створили базу для подальшого вдосконалення та розробки методів, спрямованих на вдосконалення процесу виявлення штрих-кодів на фотографіях з використанням нейронних мереж.

3. РЕАЛІЗАЦІЯ НЕЙРОННОЇ МОДЕЛІ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1. Вибір даних для навчання моделі

Як вихідні дані було обрано пакет зображень предметів зі штрих-кодами. До кожного зображення предмету з штрих-кодом в наборі даних надана анотація, яка включає інформацію про положення та параметри штрих-коду на зображенні. Ця анотація містить в собі:

- Положення штрих-коду: Координати місцезнаходження штрих-коду на зображенні - це прямокутна область, яка визначає положення штрих-коду на фото.
- Розміри та пропорції штрих-коду: Інформація про ширину, висоту та пропорції штрих-коду на зображенні, що допомагає визначити його форму та розмір.
- Клас штрих-коду

Для коректної роботи з даними, розберемося, що означають анотації до зображень. Анотації мають вигляд «0 0.465625 0.396622 0.334375 0.741892», де :

- 0: Це мітка або клас, пов'язаний з цією областю інтересу. У багатьох задачах машинного навчання це ціле число, що позначає клас об'єкту на зображенні.
- 0.465625: Це відносна координата по осі X центру області інтересу відносно ширини зображення. Вона знаходиться у діапазоні від 0 до 1 і вказує, наскільки праворуч від лівого краю зображення розташований центр цієї області.
- 0.396622: Це відносна координата по осі Y центру області інтересу відносно висоти зображення. Аналогічно до попередньої координати, вона також знаходиться у діапазоні від 0 до 1 і показує, наскільки знизу від верхнього краю зображення розташований центр цієї області.

- 0.334375: Це відносний розмір області інтересу по осі X відносно ширини зображення. Це значення також у діапазоні від 0 до 1 і вказує на ширину цієї області відносно ширини зображення.
- 0.741892: Це відносний розмір області інтересу по осі Y відносно висоти зображення. Аналогічно до попередньої координати, воно у діапазоні від 0 до 1 і показує висоту цієї області відносно висоти зображення.

Розглянемо кілька основних методів обробки анотацій у наведеному коді:

1. Читання анотаційних файлів: Ваш код читає анотаційні файли з вказаної папки та обробляє їх, щоб отримати інформацію про мітки та координати областей інтересу.
2. Перетворення в абсолютні координати у пікселях: Після читання анотаційних файлів відносні координати міток та областей інтересу перетворюються в абсолютні координати у пікселях на зображенні. Це робиться шляхом множення відносних координат на ширину та висоту відповідного зображення.
3. Розрахунок координат рамок: Для кожної області інтересу розраховуються координати лівого верхнього та правого нижнього кутів рамки, які відображають цю область на зображенні. Ці координати використовуються для малювання прямокутників або рамок навколо об'єктів на зображенні.

Рисунок 3.1. – приклад даних з набору

Першим кроком попередньої обробки даних перед використанням є зменшення зображення до необхідних розмірів.

```
# Вивід проміжного результату (зменшене зображення)
plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
plt.title(f"Resized Image: {image_file}")
plt.axis('off')
plt.show()
```

Рисунок 3.2. – код виводу зменшених зображень

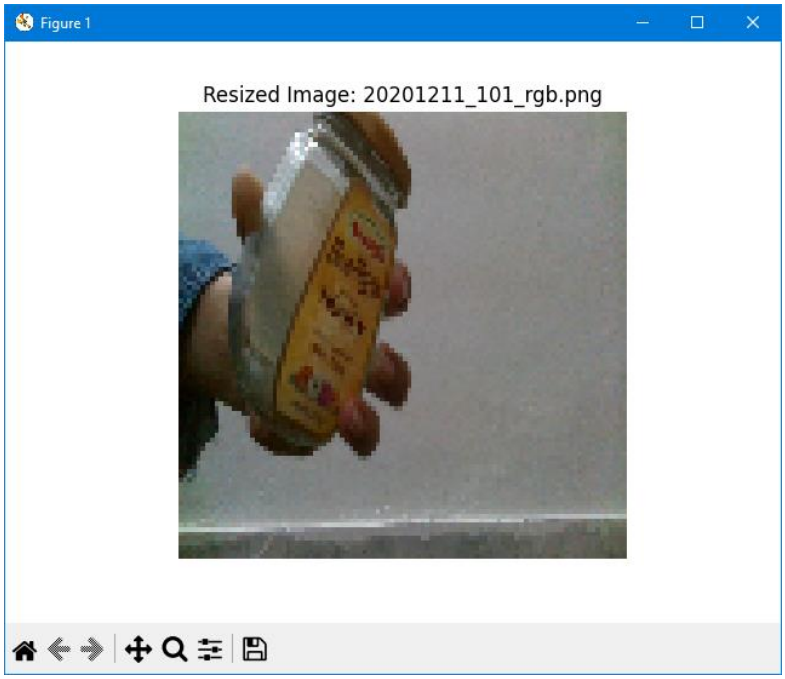


Рисунок 3.3. – зменшене зображення

Ще одним кроком для обробки зображень є аугментація. Аугментація – це техніка, яка використовується для створення нових варіантів даних шляхом застосування різноманітних перетворень до наявних зображень. Основна мета полягає в розширенні набору даних для покращення навчання моделі шляхом різноманітності даних.

Основні перетворення, що включаються у процес аугментації зображень:

- Повороти, зсуви та відображення: Виконання обертань на певний кут, зсуви по горизонталі або вертикалі, відображення зображення.
- Зміна розміру та обрізка: Зменшення або збільшення розмірів зображення, вирізання частини зображення.

- Зміна яскравості, контрастності, насиченості: Зміна освітлення, контрастності або насиченості зображення для покращення його розпізнаваності.
- Шуми, розмивання: Додавання шуму до зображення або застосування ефектів розмивання.
- Комбіновані перетворення: Комбінація кількох перетворень для створення більш складних варіацій даних.

```
# Отримання партії зображень після аугментації
augmented_batch = augmented_images.next()

# Виведення зображень після аугментації
fig, axes = plt.subplots(3, 3, figsize=(10, 10))
for i, ax in enumerate(axes.flat):
    ax.imshow(augmented_batch[i].astype('uint8'))
    ax.axis('off')
plt.tight_layout()
plt.show()
```

Рисунок 3.4. – код для виводу даних після аугментації

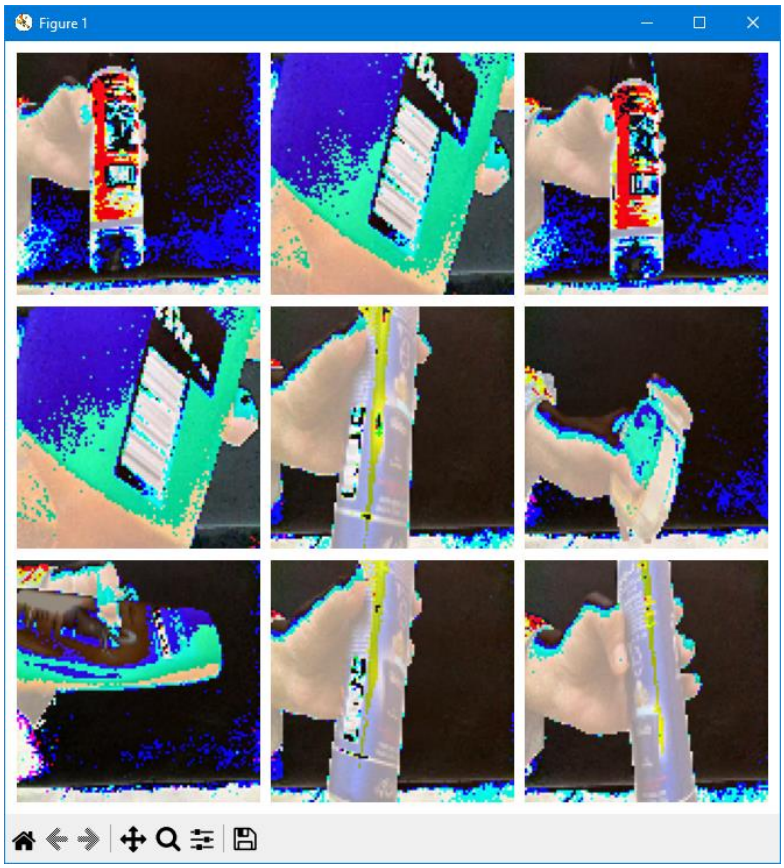


Рисунок 3.5. – результат аугментації зображень

3.2. Обрані програмні засоби для реалізації нейронної мережі

Обрана мова програмування для цього проекту - Python. Це мова програмування високого рівня, яка володіє простим синтаксисом та великою читабельністю коду. Її динамічна типізація полегшує розробку та знижує час написання програм. Python підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, функціональне та процедурне програмування, що дозволяє використовувати різні підходи при створенні програм.

Велика кількість вбудованих бібліотек та додаткових модулів робить Python відмінним вибором для машинного навчання, обробки даних та розробки нейронних мереж. Існує значна кількість спеціалізованих бібліотек, таких як TensorFlow, Keras, PyTorch, що спрощують розробку та реалізацію складних алгоритмів машинного навчання та нейронних мереж.

Однією з головних переваг Python є його кросплатформенність. Мова працює на різних операційних системах, таких як Windows, macOS, Linux, що робить її вибором номер один для розробки програм, що мають бути запущені на різних платформах.

В роботі була використана бібліотека Keras для розробки архітектури нейронної мережі. Keras - це відкрита бібліотека, спрямована на швидкі експерименти з мережами, і вона побудована на основі TensorFlow - ще однієї потужної бібліотеки для роботи з нейронними мережами в середовищі Python. Використання Keras спрощує розробку моделей, надаючи інтуїтивний інтерфейс та широкі можливості для експериментів з архітектурою мережі.

Keras був розроблений як інтерфейс для швидкої та простої реалізації нейронних мереж. Початково ця бібліотека була самостійним проєктом, але пізніше стала складовою частиною TensorFlow, головної бібліотеки для глибинного навчання від Google. Це об'єднання відбулося у 2017 році з метою спрощення та узгодження роботи з нейронними мережами, дозволяючи

використовувати зручний інтерфейс Keras у межах TensorFlow, що значно полегшило розробку моделей та їх подальше використання.

Для реалізації методів оцінки основних показників якості побудованої нейромережевої моделі та результатів прогнозування в дипломній роботі використані модулі з бібліотеки Keras, такі як model.compile для компіляції моделі з вибором метрики (metrics) і функції втрати (loss). Крім того, при оцінці моделі під час її навчання використовуються метрики, такі як точність (accuracy).

Також використовується метод EarlyStopping. EarlyStopping - це метод в Keras, який дозволяє припиняти процес навчання моделі, якщо певна метрика (наприклад, втрата на валідаційному наборі даних) перестає покращуватися або починає погіршуватися.

Це корисний підхід для уникнення перенавчання моделі. Коли метрика перестає покращуватися, досить можливо, що модель перестала вчитися загальним закономірностям і почала навчатися шуму в даних, що може призвести до погіршення її загальної продуктивності.

EarlyStopping реалізує цей принцип: під час навчання моделі він відстежує зміну обраної метрики (наприклад, втрати на валідаційному наборі). Якщо ця метрика перестає покращуватися впродовж певної кількості епох (параметр patience), навчання припиняється.

Це дозволяє заощадити час і ресурси, уникнути перенавчання та забезпечити більш раціональне навчання моделі.

За середовище розробки було обрано PyCharm - це інтегроване середовище розробки (IDE) для програмування на Python, яке надає широкий набір інструментів для створення програм. Воно є як потужним редактором коду, що оснащене різноманітними функціями, що полегшують розробку, відлагодження та тестування програм.

PyCharm дозволяє працювати з різноманітними мовами програмування, але основною його спеціалізацією є підтримка Python. Воно забезпечує інтерактивне редагування, автодоповнення коду, перевірку синтаксису, аналіз коду, рефакторинг, а також інші інструменти, спрямовані на полегшення розробки.

Зокрема, PyCharm має вбудовану підтримку віртуальних середовищ, що дозволяє ізолювати проекти та їх залежності, підтримує інтеграцію з різними системами керування версіями, а також надає можливості для налагодження коду та відстеження його виконання.

3.3. Попередня обробка даних

Перед подальшим використанням для розпізнавання інвентаризаційних та логістичних штрих-кодів, вихідні дані пройшли необхідну попередню обробку для забезпечення точності та ефективності роботи нейронної мережі.

Вихідні дані містять інформацію про штрих-коди та їх розташування на зображеннях. Завантажені зображення були зменшені до стандартного розміру, згідно з технічними вимогами для подальшої обробки.

Анотації штрих-кодів були перетворені у відповідний формат, включаючи координати та класифікацію штрих-кодів для оптимальної взаємодії з нейронною мережею. Кожна анотація включає в себе позначку наявності штрих-коду та відомості про його положення та розмір у форматі: наявність x_center y_center width height.

Нормалізація анотацій відбувається через вирази, які перетворюють відносні координати та розміри штрих-кодів у певному вигляді:

```
    absolute_x_center = x_center * image_width
absolute_y_center = y_center * image_height
```

```
absolute_box_width = box_width * image_width
absolute_box_height = box_height * image_height
```

Ці рядки перетворюють відносні координати та розміри, які представлені у відсотках відносно розмірів зображення, у абсолютні координати та розміри в масштабах відповідних розмірів зображення. Це допомагає адаптувати анотації до конкретних розмірів кожного зображення для подальшого використання у нейронній мережі.

Після завантаження та обробки даних у коді було проведено розділення на навчальний та тестовий набори за допомогою функції train_test_split з бібліотеки sklearn.model_selection. Цей крок використовувався для створення навчального набору даних train_images та train_annotations, а також тестового набору test_images та test_annotations. Розмір тестового набору був встановлений на 30% від загального набору даних. Це дозволило ефективно використовувати дані для навчання та оцінки моделі розпізнавання штрих-кодів.

3.4. Розробка архітектури нейронної мережі для виявлення штрих-кодів

Перший крок у розробці моделі для виявлення штрих-кодів полягав у завантаженні та обробці вихідних даних.

На початку роботи було отримано набір зображень та відповідні до них анотації. Ці анотації містять важливу інформацію про наявність штрих-кодів на зображеннях, їхнє місцезнаходження (координати) та розміри на зображенні.

Завантаження зображень виконується за допомогою бібліотеки OpenCV. Після цього, відбувається стандартизація зображення до бажаного розміру (target_width, target_height), щоб усі дані мали однаковий формат для подальшого використання в нейронній мережі. Це є важливим кроком, оскільки однаковий

розмір зображень допомагає моделі працювати ефективно та коректно виявляти штрих-коди на них.

Паралельно з цим відбувається обробка анотацій. Кожен рядок анотації містить важливі дані про штрих-коди на зображеннях. Інформація в анотаціях допомагає зв'язати зображення з відповідними штрих-кодами та їхніми позначеннями на зображенні.

Побудова моделі для виявлення штрих-кодів базувалася на використанні нейронної мережі. Для виявлення штрих-кодів було використано попередньо навчену модель InceptionV3, яка має досить високу точність у визначенні об'єктів на зображеннях. Першим кроком було завантаження InceptionV3 без останніх повнозв'язаних шарів, які відповідають за класифікацію зображення. Це дозволило використовувати глибинні функції, які вже навчені визнавати властивості об'єктів на зображеннях.

Перед InceptionV3 було додано кілька повнозв'язаних шарів для виконання завдання виявлення штрих-кодів. Ці шари мали активацію LeakyReLU для уникнення проблем з градієнтами та зменшенням втрат під час навчання. LeakyReLU - це варіація функції активації ReLU (Rectified Linear Unit), яка використовується у шарах нейронних мереж. У порівнянні зі звичайним ReLU, LeakyReLU має невеликий нахил для від'ємних значень, що дозволяє уникнути проблеми "мертвих нейронів", коли нейрони у шарі з ReLU можуть стати неактивними та не вивчати жодні залежності. Перевагою LeakyReLU є його здатність уникнути "мертвих нейронів", коли у звичайного ReLU нейрони з від'ємними вагами можуть не активуватися. Це дозволяє зберігати градієнти під час навчання нейронної мережі і покращує швидкість та стабільність процесу навчання.

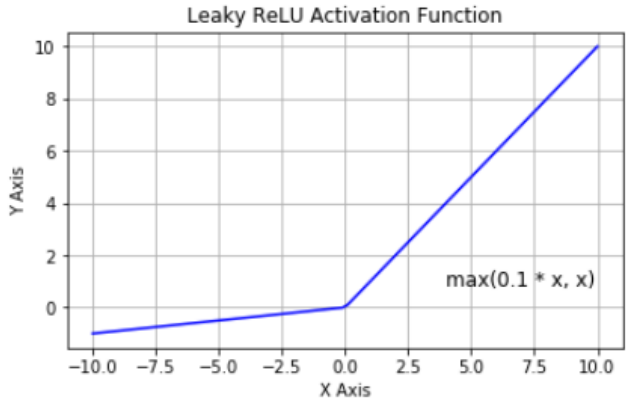


Рисунок 3.6. – графік Leaky ReLU

Типи шарів, що використовувалися при навчанні:

- Глобальний середній пулінг:

Після використання InceptionV3 додано шар глобального середнього пулінгу (GlobalAveragePooling2D). Цей шар обчислює середнє значення кожного фільтра з попереднього шару по всіх його позиціях, зменшуючи розмірність даних.

- Повнозв'язані шари:

Модель містить декілька повнозв'язаних шарів (Dense). Вони відповідають за згортання отриманих векторів у відповідь класифікації.

- Шари активації та регуляризації:

Для активації використовувалася LeakyReLU (LeakyReLU), яка дозволяє уникнути проблем "мертвих нейронів".

Щоб уникнути перенавчання, застосовувалися шари регуляризації: Dropout та BatchNormalization.

- Шар класифікації:

Останній повнозв'язаний шар має активацію softmax, що визначає ймовірність належності зображення до конкретного класу.

В процесі навчання нейронних мереж важливо мати оптимізатор, який ефективно керує швидкістю навчання та здатний адаптуватися до різних типів даних. У даній роботі для цього було використано оптимізатор Adam.

Adam - це оптимізатор, який поєднує в собі переваги двох типів адаптивних методів оптимізації: адаптивний градієнтний спуск і метод моментів. Він працює, використовуючи внутрішні пам'яті для зберігання попередніх градієнтів та квадратів градієнтів. Це дозволяє методу виправляти швидкість навчання для кожного параметра незалежно.

Однією з ключових переваг Adam полягає в його здатності пристосовуватися до швидкості навчання в залежності від різних типів даних або областей навчання. Він може ефективно працювати з великими обсягами даних та різними архітектурами нейронних мереж.

У цій роботі модель скомпільована з використанням функції втрат 'categorical_crossentropy' та метрики 'accuracy'.

Функція втрат 'categorical_crossentropy' широко використовується для задач класифікації з багатьма класами. Вона вимірює відмінність між фактичними та передбаченими розподілами класів. Метрика 'accuracy' визначає точність класифікації, вимірюючи відсоток правильно класифікованих зразків.

Після компіляції модель навчалася на згенерованому навчальному наборі даних, що містив зображення та анотації до них. Під час навчання моделі було створено графіки втрат та точності, які дозволяли візуалізувати якість навчання. Графіки втрат демонстрували зміни втрат під час процесу навчання, в той час як графіки точності відображали ефективність моделі на навчальних та тестових даних. Це допомагало зрозуміти, як модель навчається та чи є явище перенавчання чи недонавчання.

3.5. Аналіз результатів роботи навченої мережі

Загальна точність мережі склала 85%. Це означає, що вона правильно класифікувала більшу частину усіх прикладів у тестовому наборі. Проведемо огляд статистичних показників якості моделі, які були отримані за допомогою бібліотеки Scikit-learn.

```
Accuracy: 0.8511764705882353
Precision: 0.42473529411764705
Recall: 0.5197926421404682
F1 Score: 0.46798665261678962
```

Рисунок 3.7. – оцінка метрик

Precision для класів є високим, що означає, що модель досить точно визначає ці класи.

Recall для класу "0" є низьким що означає, що модель пропускає багато екземплярів цього класу. Проте,ми маємо високе значення Recall для класу "1" (74%), що відповідає за наявність штрих-коду на зображенні.

	precision	recall	f1-score	support
0	0.84	0.42	0.56	38
1	0.95	0.74	0.83	27

Рисунок 3.8. – аналіз метрики Recall

Де Precision відображає точність у визначенні позитивних прогнозів, recall описує частку правильно передбачених позитивних результатів серед усіх дійсних позитивних випадків, а f1-score представляє гармонічну середню між precision та recall. Support вказує на кількість екземплярів у кожному класі, на яку базується обчислення метрик.

Рисунок 3.9.-3.11. – правильно спрогнозовані дані

У даній роботі застосування bounding box пов'язане з обробкою анотаційних файлів, які містять інформацію про розміри та положення об'єктів на зображеннях.

- Зчитування анотаційних файлів:

Код перебирає файли в папці з анотаціями (annotation_folder), завантажує кожен файл із списку та читає вміст.

Інформація про об'єкти на зображенні знаходиться у цих анотаційних файлах. Вони містять координати та розміри (наприклад, координати центру та ширину/висоту у відносних значеннях до розмірів зображення).

- Перетворення в абсолютні координати:

Отримані в анотаційних файлах відносні координати переводяться у відповідні піксельні значення в абсолютній системі координат зображення.

- Розрахунок обмежувальних рамок (bounding box):

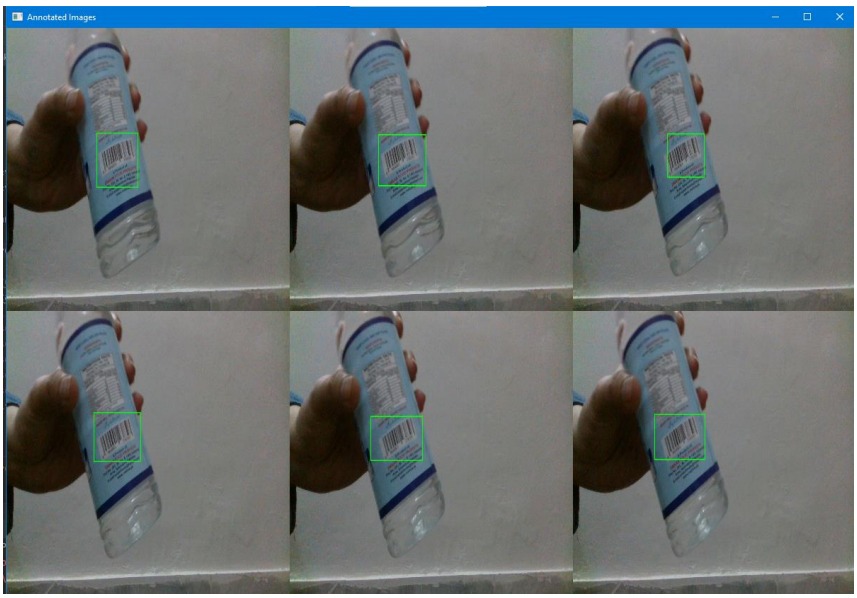
За допомогою отриманих піксельних координат об'єктів розраховуються координати та розміри прямокутних обмежувальних рамок навколо об'єктів. Ці обмежувальні рамки позначають положення та розмір об'єктів на зображенні.

- Малювання bounding box на зображенні:

Отримані координати обмежувальних рамок (x1, y1, x2, y2) використовуються для намальовання прямокутних рамок на відповідних зображеннях за допомогою cv2.rectangle. Це допомагає візуалізувати області, де знаходяться об'єкти.

- Збереження та відображення зображень з bounding box:

Зображення з намальованими bounding box показуються в окремому вікні за допомогою OpenCV (cv2.imshow). Це дозволяє переглянути, як обмежувальні рамки охоплюють об'єкти на зображеннях.



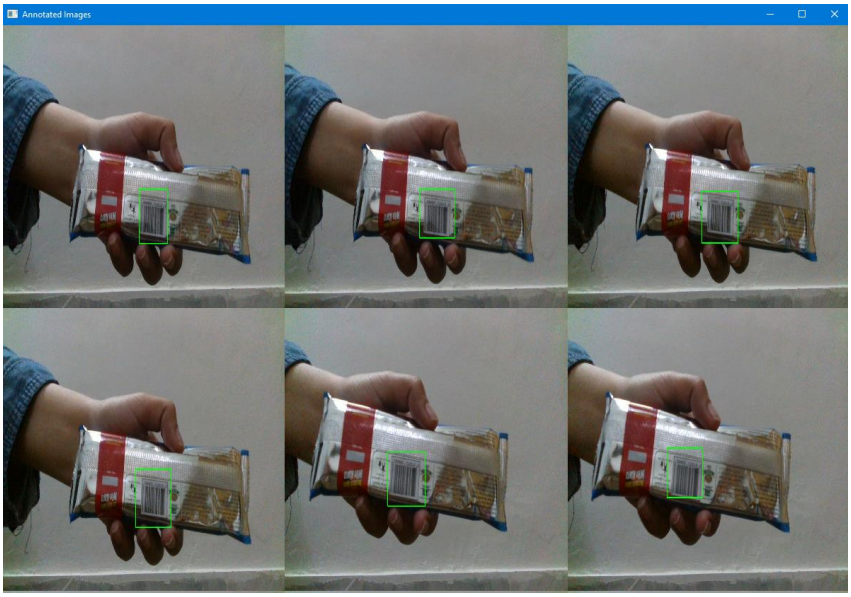


Рисунок 3.12.-3.14. – результати використання bounding box

ВИСНОВКИ

У ході дослідження, присвяченого процесу розпізнавання штрих-кодів на зображеннях, було проведено детальний аналіз та дослідження методів та алгоритмів, спрямованих на ефективне виявлення та декодування штрих-кодів у різноманітних умовах.

Перед створенням програмного забезпечення було проведенр глибокий аналіз проблеми розпізнавання штрих-кодів, визначено основні вимоги та функціональні можливості, які має вирішувати програма. Визначено найбільш підходящий метод та інструменти для реалізації програми.

На цьому етапі проводиться активна розробка програми, створення алгоритмів обробки зображень, інтеграція зовнішніх бібліотек та інструментів, тестування та виправлення помилок. Після введення базової функціональності до програми важливо провести аналіз результатів та виявлення слабких місць, які можуть бути вдосконалені чи оптимізовані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Neural Computing and Applications (2020). – Режим доступу до ресурсу: http://link.springer.com/10.1007/s00521-020-04780-3 /.
2. BOHR International Journal of Biocomputing and Nano Technology 2022, Vol. 1, No. 1, pp. 9–15. – Режим доступу до ресурсу: https://doi.org/10.54646/bijbnt.002 www.bohrpub.com
3. The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition. – Режим доступу до ресурсу: https://doi.org/10.3390/app10051897
4. Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., & Yan, S. (2016). Deep Learning with S-Shaped Rectified Linear Activation Units. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). – Режим доступу до ресурсу: https://doi.org/10.1609/aaai.v30i1.10287
5. IEEE Transactions on Neural Networks and Learning Systems (Volume: 34, Issue: 4, April 2023). – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/9526915
6. J. Cao et al., "DO-Conv: Depthwise Over-Parameterized Convolutional Layer," in IEEE Transactions on Image Processing, vol. 31, pp. 3726-3736, 2022. – Режим доступу до ресурсу: doi: 10.1109/TIP.2022.3175432.
7. Jie, H. J., & Wanda, P. (2020). Runpool: A dynamic pooling layer for convolution neural network. International Journal of Computational Intelligence Systems, 13(1), 66–76. – Режим доступу до ресурсу: doi: https://doi.org/10.2991/ijcis.d.200120.002
8. Basha, S. H. S., Dubey, S. R., Pulabaigari, V., & Mukherjee, S. (2020). Impact of fully connected layers on performance of convolutional neural networks for image classification. Neurocomputing, 378, 112–119. – Режим доступу до ресурсу: https://doi.org/10.1016/j.neucom.2019.10.008
9. Wright, L. G., Onodera, T., Stein, M. M., Wang, T., Schachter, D. T., Hu, Z., & McMahon, P. L. (2022). Deep physical neural networks trained with

backpropagation. Nature, 601(7894), 549–555. – Режим доступу до ресурсу: https://doi.org/10.1038/s41586-021-04223-6

10. Ojha, V., & Nicosia, G. (2022). Backpropagation Neural Tree. Neural Networks, 149, 66–83. – Режим доступу до ресурсу: https://doi.org/10.1016/j.neunet.2022.02.003

11. Ying, X. (2019). An Overview of Overfitting and its Solutions. In Journal of Physics: Conference Series (Vol. 1168). Institute of Physics Publishing. – Режим доступу до ресурсу: https://doi.org/10.1088/1742-6596/1168/2/022022

12. Gabella, M. (2021). Topology of Learning in Feedforward Neural Networks. IEEE Transactions on Neural Networks and Learning Systems, 32(8), 3588–3592. – Режим доступу до ресурсу: https://doi.org/10.1109/TNNLS.2020.3015790

13. Marijanović, D., Nyarko, E. K., & Filko, D. (2022). Wound Detection by Simple Feedforward Neural Network. Electronics (Switzerland), 11(3). – Режим доступу до ресурсу: https://doi.org/10.3390/electronics11030329

14. Montavon, G., Samek, W., & Müller, K. R. (2018, February 1). Methods for interpreting and understanding deep neural networks. Digital Signal Processing: A Review Journal. Elsevier Inc. – Режим доступу до ресурсу: https://doi.org/10.1016/j.dsp.2017.10.011

15. Salahuddin, Z., Woodruff, H. C., Chatterjee, A., & Lambin, P. (2022, January 1). Transparency of deep neural networks for medical image analysis: A review of interpretability methods. Computers in Biology and Medicine. Elsevier Ltd. – Режим доступу до ресурсу: https://doi.org/10.1016/j.compbiomed.2021.105111

16. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., … Chen, T. (2018). Recent advances in convolutional neural networks. Pattern Recognition, 77, 354–377. https://doi.org/10.1016/j.patcog.2017.10.013

17. Yu, J., de Antonio, A., & Villalba-Mora, E. (2022, February 1). Deep Learning (CNN, RNN) Applications for Smart Homes: A Systematic Review. Computers. MDPI. https://doi.org/10.3390/computers11020026

18. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

19. Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., … Hochreiter, S. (2021). HOPFIELD NETWORKS IS ALL YOU NEED. In ICLR 2021 - 9th International Conference on Learning Representations. International Conference on Learning Representations, ICLR.

20. Xiong, W., Jia, X., Yang, D., Ai, M., Li, L., & Wang, S. (2021). DP-LinkNet: A convolutional network for historical document image binarization. KSII Transactions on Internet and Information Systems, 15(5), 1778–1797. https://doi.org/10.3837/tiis.2021.05.011

21. Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

22. Wikipedia contributors. (2016). F1 score. Wikipedia, The Free Encyclopedia, 70.

23. Whitley, D. (1994). A genetic algorithm tutorial. Statistics and Computing, 4(2), 65–85. https://doi.org/10.1007/BF00175354

ДОДАТОК

```python
import cv2
import os
from sklearn.model_selection import train_test_split
from keras.utils import Sequence
from keras.layers import Dropout, BatchNormalization, Dense, GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
import numpy as np
from keras.applications.resnet50 import preprocess_input
from keras.layers import Conv2D, MaxPooling2D, Flatten, LeakyReLU
import matplotlib.pyplot as plt
from keras.applications import InceptionV3

target_width = 100
target_height = 100

# Путь к папке с изображениями
image_folder = 'G:\\BarcodeRecognitionProject\\im3'

# Загрузка изображений и уменьшение их размеров
images = []
image_sizes = []
for image_file in os.listdir(image_folder):
    if image_file.endswith(('.jpg', '.jpeg', '.png')):
        image_path = os.path.join(image_folder, image_file)
        image = cv2.imread(image_path)
        resized_image = cv2.resize(image, (target_width, target_height))
        images.append(resized_image)
        height, width, _ = image.shape
        image_sizes.append((width, height))


# Путь к папке с файлами аннотаций
annotation_folder = 'G:\\BarcodeRecognitionProject\\an3'

annotations = []
for annotation_file in os.listdir(annotation_folder):
    if annotation_file.endswith('.txt'):
        annotation_path = os.path.join(annotation_folder, annotation_file)
        with open(annotation_path, 'r') as file:
```

```
            annotation_content = file.read()
            annotations.append(annotation_content)

annotated_data = []

image_idx = 0  # Індекс для зображень
annotated_images = []  # Зберігатиме зображення з анотаціями

for idx, image_file in enumerate(os.listdir(image_folder)):
    if image_file.endswith(('.jpg', '.jpeg', '.png')):
        image_path = os.path.join(image_folder, image_file)
        image = cv2.imread(image_path)
        height, width, _ = image.shape

        # Зчитування анотаційного файлу
        annotation_file = os.path.splitext(image_file)[0] + '.txt'
        annotation_path = os.path.join(annotation_folder, annotation_file)

        with open(annotation_path, 'r') as file:
            annotation_content = file.readline().split()
            relative_x_center = float(annotation_content[1])
            relative_y_center = float(annotation_content[2])
            relative_width = float(annotation_content[3])
            relative_height = float(annotation_content[4])
            class_label = float(annotation_content[0])

            # Перетворення в абсолютні координати у пікселях
            absolute_x_center = int(relative_x_center * width)
            absolute_y_center = int(relative_y_center * height)
            absolute_width = int(relative_width * width)
            absolute_height = int(relative_height * height)

            # Розрахунок лівого верхнього та правого нижнього кутів рамки
            x1 = max(0, absolute_x_center - absolute_width // 2)
            y1 = max(0, absolute_y_center - absolute_height // 2)
            x2 = min(width, absolute_x_center + absolute_width // 2)
            y2 = min(height, absolute_y_center + absolute_height // 2)

            target_label = [x1, y1, x2, y2, class_label]
            annotated_data.append(target_label)
```

```
            # Намалюємо рамку на зображенні
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)



        annotated_images.append(image)  # Зберігаємо зображення з анотаціями

    annotated_images_resized = []
    for img in annotated_images:
        img_resized = cv2.resize(img, (target_width, target_height))
        annotated_images_resized.append(img_resized)

    # Після зміни розміру зображень у `annotated_images`

    if (idx + 1) % 9 == 0 or (idx + 1) == len(os.listdir(image_folder)):
        canvas = np.zeros((target_height * 3, target_width * 3, 3),
                          dtype=np.uint8)  # Створюємо "холст" для відображення 3x3
        row = 0
        col = 0
        for i in range(len(annotated_images_resized)):
            canvas[row * target_height:(row + 1) * target_height, col * target_width:(col + 1) * target_width] = \
            annotated_images_resized[i]
            col += 1
            if col == 3:
                col = 0
                row += 1

        #cv2.imshow('Annotated Images', canvas)
        #cv2.waitKey(0)

        annotated_images = []  # Очищаємо список для наступних 9 зображень

#cv2.destroyAllWindows()  # Закриваємо всі вікна після завершення процесу

# Класс генератора данных
class DataGenerator(Sequence):
    def __init__(self, data, batch_size=32, shuffle=True, augmentation=None):
        self.data = data
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.augmentation = augmentation
```

```
        self.indexes = np.arange(len(self.data))
        if self.shuffle:
            np.random.shuffle(self.indexes)

    def __len__(self):
        return int(np.ceil(len(self.data) / self.batch_size))

    def __getitem__(self, index):
        indexes = self.indexes[index * self.batch_size:(index + 1) * self.batch_size]
        batch_data = [self.data[k] for k in indexes]

        images = []
        annotations = []

        for image, annotation in batch_data:
            processed_image = cv2.resize(image, (target_width, target_height))

            # Добавление аугментации
            if self.augmentation is not None:
                processed_image = self.augmentation.random_transform(processed_image)

            images.append(processed_image)
            annotations.append(annotation)

        return np.array(images), np.array(annotations)


train_images, test_images, train_annotations, test_annotations = train_test_split(
    images, annotated_data, test_size=0.3, random_state=42)

train_data = list(zip(train_images, train_annotations))
test_data = list(zip(test_images, test_annotations))

def apply_contrast(image, factor=1.0):
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=factor, tileGridSize=(8, 8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl, a, b))
```

```
    final_image = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    return cv2.cvtColor(final_image, cv2.COLOR_BGR2RGB)

def custom_augmentation(image):
    image = apply_contrast(image, factor=1.5)  # Застосування збільшення контрастності
    # Додаткові операції аугментації, які ви вже використовуєте
    return image

train_images_scaled = preprocess_input(np.array(train_images))
test_images_scaled = preprocess_input(np.array(test_images))

train_augmentation = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    preprocessing_function=preprocess_input  # Функція аугментації
)

test_augmentation = ImageDataGenerator(
    preprocessing_function=preprocess_input  # Тут застосовується масштабування
)


train_data, test_data = train_test_split(list(zip(images, annotated_data)), test_size=0.3, random_state=42)
train_images, train_annotations = zip(*train_data)
test_images, test_annotations = zip(*test_data)

images_per_page = 9
num_images = len(test_images)
num_pages = -(-num_images // images_per_page)  # Округлення угору

for page in range(num_pages):
    start_idx = page * images_per_page
    end_idx = min((page + 1) * images_per_page, num_images)

    num_rows = min(3, end_idx - start_idx)
    num_cols = min(3, end_idx - start_idx)
```

```
    fig, axs = plt.subplots(num_rows, num_cols, figsize=(10, 10))
    idx = 0

    for i in range(num_rows):
        for j in range(num_cols):
            if start_idx + idx < end_idx:
                current_image = test_images[start_idx + idx].copy()
                annotation = test_annotations[start_idx + idx]

                x1, y1, x2, y2, class_label = annotation

                # Draw rectangle on the image
                cv2.rectangle(current_image, (x1, y1), (x2, y2), (0, 255, 0), 2)

                axs[i, j].imshow(current_image)
                axs[i, j].axis('off')
                idx += 1

    #plt.tight_layout()
    #plt.show()


train_generator = DataGenerator(list(zip(train_images, train_annotations)), batch_size=32, shuffle=False, augmentation=train_augmentation)
test_generator = DataGenerator(list(zip(test_images, test_annotations)), batch_size=32, shuffle=False, augmentation=test_augmentation)

from keras.models import Model
from keras.layers import Reshape

# Використовуємо InceptionV3 без верхніх шарів
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(target_width, target_height, 3))

for layer in base_model.layers:
    layer.trainable = False


# Додаємо шар глобального середнього пулінгу
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Додаємо декілька повнозв'язаних шарів без регуляризації
from keras.regularizers import l2
```

```
x = Dense(1024, activation=LeakyReLU(alpha=0.1), kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)
x = Dropout(0.7)(x)

# Додатковий повнозв'язаний шар
x = Dense(512, activation=LeakyReLU(alpha=0.1), kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)
x = Dropout(0.6)(x)

# Зменшуємо розмірність
x = Dense(256, activation=LeakyReLU(alpha=0.1), kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)

# Ще один повнозв'язаний шар
x = Dense(128, activation=LeakyReLU(alpha=0.1), kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)
x = Dropout(0.4)(x)

# Ще один повнозв'язаний шар
x = Dense(64, activation=LeakyReLU(alpha=0.1), kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)

# Останній повнозв'язаний шар з softmax для класифікації
predictions = Dense(5, activation='softmax')(x)

# Збираємо модель
model = Model(inputs=base_model.input, outputs=predictions)
optimizer = Adam(learning_rate=0.0001)

# Компілюємо модель з використанням оптимізатора
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Отримання історії навчання моделі
history = model.fit(train_generator, epochs=20, validation_data=test_generator)

print(f"Формат першої мітки у навчальному наборі: {train_annotations[0]}")
print(f"Формат першої мітки у тестовому наборі: {test_annotations[0]}")
predictions = model.predict(test_generator)
```

```python
print(f"Формат передбачень моделі: {predictions[0]}")

# Отримання значень втрати та точності з історії навчання
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Побудова графіків втрати
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

# Побудова графіків точності
plt.subplot(1, 2, 2)
plt.plot(train_accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.tight_layout()
plt.show()

# Отримання значень втрати та точності з історії навчання
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

epochs = range(1, len(train_loss) + 1)

# Функція для оновлення графіків за кожну епоху
def update(epoch):
```

```python
# Функція для оновлення графіків за кожну епоху
def update(epoch):
    plt.clf()

    plt.subplot(1, 2, 1)
    plt.plot(epochs[:epoch], train_loss[:epoch], label='Training Loss')
    plt.plot(epochs[:epoch], val_loss[:epoch], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs[:epoch], train_accuracy[:epoch], label='Training Accuracy')
    plt.plot(epochs[:epoch], val_accuracy[:epoch], label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()


predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
actual_classes = []  # Фактичні класи для порівняння з прогнозами
for annotation in test_annotations:
    actual_classes.append(annotation[-1])  # Останній елемент в аннотації - клас
correctly_predicted_indexes = np.where(predicted_classes == actual_classes)[0]

# Отримання передбачень для тестового набору даних
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Порівняння передбачених класів з фактичними класами
correctly_predicted_indexes = np.where(predicted_classes == actual_classes)[0]

# Виведення правильно відгаданих зображень
#for idx in correctly_predicted_indexes:
    #plt.imshow(test_images[idx])
    #plt.title(f"Predicted: {predicted_classes[idx]}, Actual: {actual_classes[idx]}")
    #plt.axis('off')
```

```python
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(actual_classes, predicted_classes)
print(conf_matrix)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# Отримання передбачень для тестового набору даних
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Створення списку фактичних класів для порівняння з прогнозами
actual_classes = []
for annotation in test_annotations:
    actual_classes.append(annotation[-1])  # Останній елемент в аннотації - клас

# Порівняння передбачених класів з фактичними класами
accuracy = accuracy_score(actual_classes, predicted_classes)
precision = precision_score(actual_classes, predicted_classes, average='macro')
recall = recall_score(actual_classes, predicted_classes, average='macro')
f1 = f1_score(actual_classes, predicted_classes, average='macro')

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')

# Classification report
print(classification_report(actual_classes, predicted_classes))
```