МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КУНЦОВ МАКСИМ СЕРГІЙОВИЧ

Допускається до захисту:
в. о. завідувача кафедри
прикладної математики та
кібербезпеки,
д – р філософії з математики,
_____ А.В.Луценко
«__»_____20__ р.

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІДЕНТИФІКАЦІЇ АВТОМОБІЛЯ МЕТОДАМИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

Спеціальність 125 Кібербезпека

Кваліфікаційна (бакалаврська) робота

Керівник:
канд. техн. наук, доцент,
доцент кафедри прикладної
математики та кібербезпеки,
Загоруйко Л. В.

Оцінка:___/___/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК:_____
(підпис)

Вінниця - 2024

АНОТАЦІЯ

Кунцов М. С. Розробка програмного забезпечення для ідентифікації автомобіля методами згорткових нейронних мереж. Спеціальність 125 «Кібербезпека». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (бакалаврській) роботі досліджується модель згорткової нейронної мережі та метод навчання зворотної помилки, а також написання програми за допомогою мови програмування Python.

Ключові слова: згорткові нейронні мережі, розпізнавання образів, машинне навчання, ідентифікація автомобіля.

42 с., 18 рис., 1 дод., 24 джерел.

ABSTRACT

Kuntsov M. S. Development of Software for Vehicle Identification Using Convolutional Neural Networks. Specialty 125 "Cybersecurity". Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

The qualification (bachelor's) thesis investigates the model of convolutional neural networks and the backpropagation learning method, as well as the development of a program using the Python programming language.

Keywords: convolutional neural networks, image recognition, machine learning, vehicle identification.

42 pp, 18 figures, 1 appendix, 24 sources.

ЗМІСТ

ВСТУП

Автомобільна промисловість постійно розвивається, зростаючи важливість безпеки та ефективності управління транспортними засобами. У цьому контексті автоматизована система ідентифікації автомобілів стає ключовою для впровадження різноманітних технологій та послуг, що сприяють забезпеченню безпеки, відстеженню та управлінню автопарком.

Методи штучного інтелекту, зокрема згорткові нейронні мережі, надають унікальні можливості для розробки програмного забезпечення, яке може автоматично розпізнавати автомобілі на зображеннях. Використання таких методів дозволяє забезпечити швидке та точне визначення типу, марки або інших характеристик автомобілів за їх візуальними даними.

Мета даної роботи полягає у розробці програмного забезпечення для ідентифікації автомобілів з використанням згорткових нейронних мереж.

Відповідне програмне забезпечення може знайти застосування у різних сферах, від автомобільної промисловості до систем безпеки та моніторингу транспорту, сприяючи підвищенню ефективності та безпеки автомобільного руху. Розробка та вдосконалення таких інноваційних рішень стане вагомим кроком у напрямку розвитку сучасної автомобільної технології та забезпеченням зручності та безпеки для кінцевих користувачів.

Актуальність: ідентифікації автомобілів є важливим завданням у багатьох сферах, включаючи безпеку дорожнього руху, відслідковування транспортних засобів, контроль за обмеженнями на території міста, тощо. Завдяки зростанню обчислювальної потужності та розвитку глибокого навчання, застосування згорткових нейронних мереж для розпізнавання автомобілів стає все більш ефективним і точним.

Мета роботи: розробка програмного забезпечення, яке використовує згорткові нейронні мережі для ідентифікації автомобілів на зображеннях.

Об’єкт дослідження є процес ідентифікації автомобілів на зображеннях.

Предмет дослідження є розробка програмного забезпечення, яке використовує згорткові нейронні мережі для ідентифікації автомобілів.

Методи дослідження. У дослідженні використовуються методи глибокого навчання, зокрема згорткові нейронні мережі.

Практичне значення полягає в розробці програмне забезпечення може бути застосоване в різних галузях, включаючи системи безпеки, моніторинг транспортного потоку, автоматизоване відслідковування автомобілів для охоронних агентств тощо. Воно може допомогти виявляти автомобілі на зображеннях, що робить його важливим інструментом у сучасному світі технологій.

РОЗДІЛ 1. ОСНОВНІ ПОНЯТТЯ ПРО ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ЇХ АРХІТЕКТУРА І СТРУКТУРА ТА МЕТОДИ НАВЧАННЯ

1.1. Згорткова нейронна мережа

Згорткова нейронна мережа (Convolutional Neural Networks, CNNs) являє собою інструментом для обробки зображень і використовуються в багатьох сферах, таку як розпізнавання образів та класифікацію зображень. Основна їх сутність полягає в тому, що вони можуть швидко вивчати властивості зображень, роблячи їх ідентифікацію та класифікацію більш точною та ефективною.

Згорткова нейронна мережа використовує дві основні складові: фільтри, які визначають ознаки, та карти ознак, які показують місця знаходження цих ознак на зображенні. Фільтри, такі як вертикальні або горизонтальні, застосовуються до вихідного зображення за допомогою операції згортки, щоб виявити певні властивості. Результати згортки, або карти ознак, показують розташування цих ознак.

Мета згортки - зменшити розмірність карт ознак до рівня, щоб їх можна було використовувати в мережі прямого поширення. Згортковий шар працює на основі локальних рецептивних полів, де кожен нейрон пов'язаний лише з обмеженою областю вхідної матриці, що відтворює деякі особливості людського зору.

Недоліки згорткових нейронних мереж включають високу складність архітектури, повнозв’язаність та фіксовану площу вікна згорткового шару. Для підвищення ефективності роботи таких мереж необхідно знаходити оптимальні значення параметрів, таких як кількість карт ознак, щільність зв'язків між ними та розмір вікна.

1.2. Архітектура згорткової нейронної мережі

У перцептроні, який являє собою повнозв'язну нейронну мережу, кожен нейрон має з'єднання з усіма нейронами попереднього шару, при цьому кожен зв'язок має свій власний ваговий коефіцієнт. У згортковій нейронній мережі використовується матриця ваг, яка обмежена та має невеликий розмір, яку так би мовити рухають по всьому оброблюваному шару (починаючи з вхідного зображення), що після кожного зсуву формує сигнал активації для нейрона наступного шару з такою самою позицією. Тобто різні нейрони вихідного шару використовують загальні ваги - матрицю ваг, також відому як ядро згортки, щоб виявити різні ознаки, наприклад, нахилена лінія під певним кутом. Під час проходження кожним ядром формується своя карта ознак, що робить нейронну мережу багатомірною (багато незалежних карт ознак на одному шарі). При переміщенні ядра згортки вони зазвичай зсуваються не на весь розмір, а лише на невелику відстань, наприклад, на один або два пікселі, щоб уникнути "перестрибування" шуканих ознак.

Операція субдіскретизації, також відома як операція пулінгу, зменшує розмірність карт ознак у згорткових нейронних мережах. Вона вважає, що важливіше виявлення наявності ознаки, ніж точне визначення її координат, тому вибирає максимальне або середнє значення з декількох нейронів карт ознак, щоб створити зменшену карту ознак. Це сприяє прискоренню обчислень та робить мережу менш чутливою до масштабу вхідного зображення.

Послідовне застосування декількох шарів згортки та субдіскретизації дозволяє побудувати згорткову нейронну мережу. Це дозволяє мережі розпізнавати складні ієрархії ознак. Зазвичай на виході мережі додають декілька повнозв'язаних шарів (перцептрон), яким на вхід переходять кінцеві карти ознак.

Якщо на першому шарі ядро згортки застосовується лише до одного вихідного зображення, то на наступних шарах це ядро застосовується паралельно до всіх карт ознак цього шару, а результат згортки сумується, утворюючи карту ознак для наступного шару.

Модель структури згорткової мережі яка зазначена на рис.1.1, вона складається з трьох шарів: згорткові шари, субдіскретізірующі (subsampling, підвибірка) шари і шари "звичайної" нейронної мережі.

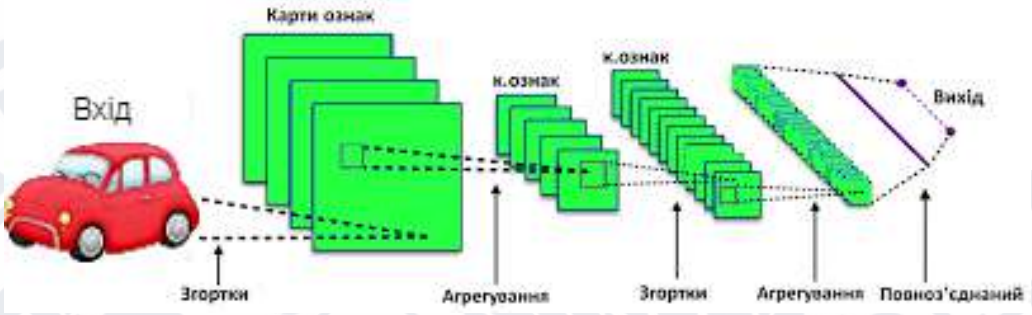

Рисунок 1.1 - Структура згорткової мережі.

Перші два типи шарів, а саме згорткові та субдіскретизації (convolutional, subsampling), чергуються між собою, утворюючи вихідний вектор ознак для подальшого використання у багатошаровому перцептроні. Мережу можна тренувати за допомогою методів градієнтного спуску. Згорткова мережа отримала свою назву через операцію згортки, яка часто застосовується у обробці зображень. Вона описана цією формулою:

$$(d \times u)[a,b] = \sum_{j,z} f[a-j, \quad b-z] \times g[j,z]$$

d – вихідна матриця; u – ядро згортки.

1.3. Основи згорткових нейронних мереж

Основною ідею згорткових нейронних мереж є узгодження в застосуванні згорткових шарів (C-шарів), шарів субдіскретизації (S-шарів) і повнозв'язних шарів (F-шарів) на виході. Ця структура включає три ключові парадигми:

- Локальне сприйняття: Нейрони не отримують на вхід всю інформацію, а лише частину зображення. Це допомагає зберігати структуру зображення від шару до шару.

- Спільні ваги: Використання невеликого набору ваг для багатьох зв'язків. Наприклад, кожен нейрон наступного шару обробляє лише маленький фрагмент зображення одним і тим же набором ваг. Ці набори, часто називаються ядрами, застосовуються до всього зображення.

- Субдіскретізація: Ця операція зменшує розмірність зображення за допомогою математичних операцій, які виконуються на фрагментах зображення з використанням фільтрів. Результатом є карта ознак, де кожен піксель відображає ступінь схожості фрагмента зображення з фільтром.

Таким чином, згорткові нейронні мережі забезпечують виявлення ознак зображення шляхом згортки фільтрів через них, що дозволяє створювати карту ознак, що зберігає важливі характеристики зображення.

Кожен елемент зображення перемножується з відповідними елементами невеликої матриці ваг, після чого результати додаються. Ця сума представляє собою піксель на вихідному зображенні, яка має назву карта ознак. Важливо відзначити, що в ідеальному випадку усі фрагменти зображення паралельно проходять через однакові ядра, а не послідовно. Кількість ядер (наборів ваг) встановлюється розробником і залежить від кількості ознак, які потрібно виділити. Іншою особливістю згорткового шару є те, що він дещо зменшує розмірність зображення через крайові ефекти.

Суть субдискретизації та S-шарів полягає в зменшенні просторової розмірності зображення, тобто у значному зменшенні розмірів вхідного зображення у певну кількість разів. Зазвичай це зменшення відбувається вдвічі, але може бути змінено нерівномірно, наприклад, у 2 рази по вертикалі і 3 рази по горизонталі. Субдіскретизація необхідна для того, щоб забезпечити стійкість до масштабу. Чергування шарів дозволяє створювати складні ієрархії ознак, оскільки карти ознак з одного шару поєднуються для створення карти ознак наступного шару. Зазвичай після кількох шарів карта ознак перетворюється в вектор або навіть скаляр, але їх може бути сотні. Ці картки ознак подаються на один-два шари повно-зв'язаної мережі, де вихідний шар

може мати різні функції активації, такі як тангенціальна або радіальна базисна функція.

1.4. Структура згорткової нейронної мережі

У згортковій нейронній мережі, результати проміжних шарів формують матрицю або набір матриць, що відображають зображення. Наприклад, можна використовувати три шари зображення (RGB червоний, зелений, синій канали). Основними типами шарів у згортковій нейронній мережі є згорткові, пулінгові та повнозв'язані шари. Згорткові шари використовують фільтри для виділення локальних ознак зображення, таких як краї та текстури. Пулінгові шари зменшують розмірність вхідних даних, зберігаючи при цьому важливі характеристики, що робить обробку більш ефективною та зменшує ризик перенавчання. Повнозв'язані шари з'єднують всі нейрони з попереднього шару з кожним нейроном наступного, що дозволяє інтегрувати виділені ознаки для кінцевої класифікації або іншої задачі. Ці шари разом утворюють багатошарову структуру, яка поступово витягує все більш абстрактні ознаки з вихідного зображення, дозволяючи мережі ефективно вирішувати складні завдання.

1.4.1. Згортковий Шар

Згортковий шар, який є ключовим елементом нейронної мережі, складається з набору фільтрів або ядер, які мають невеликі, але глибокі рецептивні поля. Під час проходження вхідних даних через шар кожен фільтр проводить згортку по ширині і висоті вхідного об'єму, обчислюючи скалярний добуток між фільтром і входом, що формує двовимірну карту активації. У процесі навчання мережа визначає, які фільтри активуються при сприйнятті певних ознак у конкретних просторових позиціях на вході. Створення

активаційних карт для всіх фільтрів по глибині формує повний вихідний об'єм згорткового шару. Таким чином, кожен елемент вихідного об'єму може розглядатися як вихід нейрона, який реагує на певну область вхідних даних, і спільно використовує параметри з іншими нейронами на тій же активаційній карті.

Згортковий шар реалізує концепцію локальних рецептивних полів, де кожен вихідний нейрон взаємодіє лише з обмеженою частиною вхідної матриці. Це дає змогу моделювати певні аспекти людського зору. У спрощеній формі цей шар можна виразити математичною формулою:

$a^i = f(a^{i-1*}y^1 + c^1)$

Де a^i- результат шару i;

$f()$- функція активації;

c - це коефіцієнт зсуву, а операція згортки входу a з ядром і позначена символом *.

Це призводить до зменшення розміру вихідних матриць через вплив крайових ефектів, який розраховується за допомогою відповідної формули:

$$a_n^i = f(\sum_e a_n^{i-1*} y_n^i + p_n^i)$$

Де a_n^i – карта ознак n;

f() – функція активації;

p_j- коефіцієнт зсуву для карти ознак n;

y_j – ядро згортки номер n;

a^{i-1} – карти ознак попереднього шару.

1.4.2. Пулінговий шар

Функція рівня об’єднання полягає в поступовому зменшенні просторового розміру представлення, щоб зменшити кількість параметрів і

обчислень у мережі, а отже, також контролювати переналаштування. Навчання не відбувається на рівнях об’єднання[25].

Одиниці об’єднання отримують за допомогою таких функцій, як максимальне об’єднання, об’єднання середніх значень і навіть об’єднання норм L2. На рівні об’єднання результатом прямого поширення є $N \times N$ блок об'єднання зводиться до єдиного значення - значення «переможної одиниці». Зворотне розповсюдження шару об’єднання потім обчислює помилку, яку отримує ця єдина «переможна одиниця»[4].

Щоб відстежувати «блок-переможець», його індекс реєструється під час прямого проходу та використовується для градієнтної маршрутизації під час зворотного поширення. Градієнтна маршрутизація виконується такими способами[4]:

- Макспулінг – помилка просто присвоюється тому, звідки вона походить – «переможна одиниця», оскільки інші одиниці в блоках об’єднання попереднього рівня не внесли в неї вкладу, отже, всі інші призначені значення нульові[4];
- Середній пулінг - помилка множиться на $\frac{1}{N \times N}$ і призначається всьому блоку об’єднання (усі одиниці отримують однакове значення)[4].

1.4.3. Повнозв’язний шар

Шари даного типу виконують редукцію розмірності (рис 1.2). Це можна здійснити різними способами, але найчастіше використовується метод максимального підбору (max-pooling) – карта ознак розділяється на сегменти, з яких вибираються найбільші значення.

Dropout шар – це метод протидії перенавчанню у нейронних мережах, які зазвичай навчаються стохастичним градієнтним спуском, випадково виключаючи деякі об'єкти з набору даних. Під час навчання за такою розрідженою мережею відбувається градієнтний зворотний хід лише для

частини ваг, після чого всі викинуті нейрони повертаються. Таким чином, на кожному кроці стохастичного градієнтного спуску змінюється одна з можливих 2N архітектур мережі, де архітектура визначається як структура зв'язків між нейронами, а N - сумарна кількість нейронів.

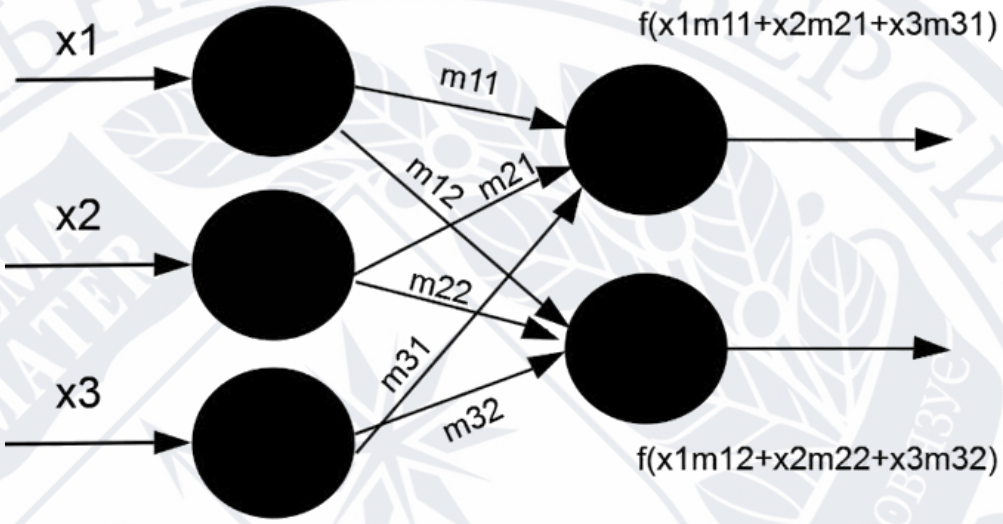

Рисунок 1.2 – Повнозв’язний шар

1.5. Висновки до розділу 1

Здійснений аналіз згорткових нейронних мереж та їх архітектури дозволив викласти основні підходи та методи, які застосовуються для ідентифікації об'єктів на зображеннях. Було розглянуто ключові складові згорткових нейроних мереж, зокрема згортковий, пулінговий та повнозв'язний шари, що забезпечують автоматичне вивчення та класифікацію ознак. Це дозволило зрозуміти високу ефективність згорткових нейроних мереж у розпізнаванні образів та ідентифікації об'єктів, що робить їх незамінними інструментами у задачах комп'ютерного зору.

РОЗДІЛ 2. НАВЧАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1. Навчання згорткової нейронної мережі

На самому початку, коли нейронна мережа тільки створена, вона ще не навчена і не налаштована. Загалом, навчання включає послідовне подання зображення на вхід мережі з набору для навчання, після чого порівнюється отримана відповідь з очікуваним результатом. У нашому випадку, якщо зображення представляє автомобіль, очікуваний вихід буде 1, якщо фон - мінус 1. Різниця між цими значеннями є помилкою (дельтою помилки). Потім цю помилку потрібно поширити на всі зв'язані нейрони мережі.

Отже, процес навчання мережі включає в себе завдання мінімізації функції помилки за допомогою корекції ваг синаптичних зв'язків між нейронами. Функція помилки визначається як різниця між отриманою відповіддю і бажаною. Наприклад, якщо на вході було подано зображення автомобіля і вихід мережі був 0.56, а бажаний результат - 1 (так як це зображення автомобіля), то помилка мережі буде різницею, а саме 0.44. Потім ваги нейронів виходного шару коригуються відповідно до цієї помилки. Для нейронів виходного шару відомі їх реальні та бажані значення виходів, тому коригування ваг для них є досить простим завданням. Однак для нейронів попередніх шарів цей процес не такий очевидний.

2.2. Алгоритм зворотного розповсюдження помилки

Серед алгоритмів навчання з учителем найбільш ефективним є метод зворотного поширення помилки. Цей метод для багатошарових нейронних мереж називається узагальненим дельта-правилом і був запропонований у 1986 році Румельхартом, Макклеландом і Вільямсом. Його введення відзначило відродження інтересу до нейронних мереж. Цей алгоритм став

першим практично застосовним для навчання багатошарових нейронних мереж[4].

Для оновлення ваг вихідного шару відомий метод, але для прихованих шарів протягом тривалого часу не було встановленої методики. Ваги прихованих нейронів мають змінюватися пропорційно до помилок тих нейронів, з якими вони пов'язані. Це означає, що зворотне поширення помилок через мережу дозволяє налаштовувати ваги зв'язків між усіма шарами належним чином, що призводить до зменшення функції помилки і навчання мережі[4].

Основні формули для методу зворотного поширення помилки виражені за такими символами:

E_k – представляє собою значення функції помилки для образу k;

$t_{k\,x}$ - визначає бажаний вихід нейрона x для образу k;

$y_{k\,x}$ - означає активований вихід нейрона x для образу k;

$s_{k\,j}$ - зважена сума виходів пов'язаних нейронів попереднього шару на вагу зв'язку, інакше ще позначається як неактивований стан x для k;

$w_{k\,x}$ – всі зв’язки між і да x нейронами

Розмір величини помилки розраховується за формулою середньоквадратична помилка:

$$E_k = \frac{1}{2}\sum_x (t_{p\,x} - y_{p\,x})^2$$

Де E_k – величина функції помилки для образу k;

$t_{k\,x}$ - бажаний вихід нейрона x для образу k;

$y_{k\,x}$ – активований вихід нейрона x для образу k.

Неактивований стан кожного нейрона x для образу k записується як виваженої суми за формулою;

$$S_{k\,x} = \sum_{i} w_{i\,x}\, y_{k\,i}$$

Де $S_{k\,x}$ - зважена сума виходів пов'язаних нейронів попереднього шару на вагу зв'язку, інакше ще позначається як неактивований стан нейрона x для образу k;

w_{ix} – вага зв’язку між i та x нейронами;

y_{pi} - активований стан нейрона x попереднього шару для образу k.

Вихід кожного нейрона x визначається значенням активаційної функції f_x, яка переводить нейрон у стан активації. Функція активації може бути будь-якою безперервно диференційованою монотонною функцією. Стан активації нейрона обчислюється за формулою:

$$y_{kx} = f_x(s_{kx})$$

Де y_{kx} – активований стан нейрона x для образу k;

f_x – функція активації;

s_{ki} – неактивований стан нейрона x для образу k.

В якості методу для зменшення помилки використовується градієнтний спуск, який полягає в пошуку мінімального значення (або найбільшого) функції шляхом прохождения вздовж вектора градієнта. Для знаходження мінімуму потрібно рухатися у напрямку градієнта.

Градієнт функції втрат це вектор, що складається з часткових похідних, обчислених за відповідною формулою:

$$\nabla E(W) = \left[\frac{dE}{dw_1}, \ldots, \frac{dE}{dw_n}\right]$$

$\nabla E(W)$ - градієнт функції втрати від матриці ваг;

$\frac{dE}{dw}$ - приватна похідна функції помилки за вагою нейрона;

n - загальна кількість ваг.

Похідна функції помилки щодо певного образу може бути обчислена за допомогою правила ланцюга, згідно з наступною формулою:

$$\frac{\partial E}{\partial w_{ix}} = \frac{\partial E}{\partial y_{ix}} * \frac{\partial y_x}{\partial s_x} * \frac{\partial s_x}{\partial w_{ix}}$$

Де $\frac{\partial E}{\partial w_{ix}}$ – значенняя від похідної функції помилки за вагою w_{ix}, між i і x нейронами;

$\frac{\partial E}{\partial y_{ix}}$ – помилка нейрона x;

$\frac{\partial y_x}{\partial s_x}$ – значення від похідної функції активації за її аргументом для нейрона x;

$\frac{\partial s_x}{\partial w_{ix}}$ – вихід I нейрона попереднього шару (стосовно нейрону x).

Звичайно, помилку нейрона $\frac{\partial E}{\partial y_{ix}}$ часто позначають символом ∂ (дельта). У вихідному шарі помилку можна явно визначити, обчисливши похідну від формули для величини помилки, що дорівнює t мінус y, тобто різниця між бажаним і фактичним виходом. Для розрахунку помилки для прихованих шарів був розроблений алгоритм зворотного поширення помилки. Його суть полягає у послідовному обчисленні помилок прихованих шарів з використанням значень помилки вихідного шару. Помилка розповсюджується по мережі у зворотному напрямку, від виходу до входу.

Для прихованого шару, помилку ∂ обчислюють за такою формулою:

$$\partial_i = \frac{\partial y_i}{\partial s_i} * \sum_x \partial_x * w_{ix}$$

Де $\frac{\partial y_i}{\partial s_i}$ - значення похідної функції активації за її аргументом для нейрона x;

∂_i – помилка нейрона i прихованого шару;

∂_x - помилка нейрона *x* наступного шару;

w_{ix} - вага зв'язку між нейроном *i* поточного (прихованого) шару та нейроном *x* вихідного або теж прихованого шару.

Процес поширення помилки в алгоритмі включає наступні етапи:

- проходження сигналу вперед по мережі, та обчислення стану нейронів;
- Визначення значення помилки ∂ для вихідного шару.
- зворотне поширення: послідовно від кінця до початку, для всіх прихованих шарів обчислення помилки ∂ за відповідною формулою;
- оновлення ваг мережі на визначені раніше значення помилки ∂.

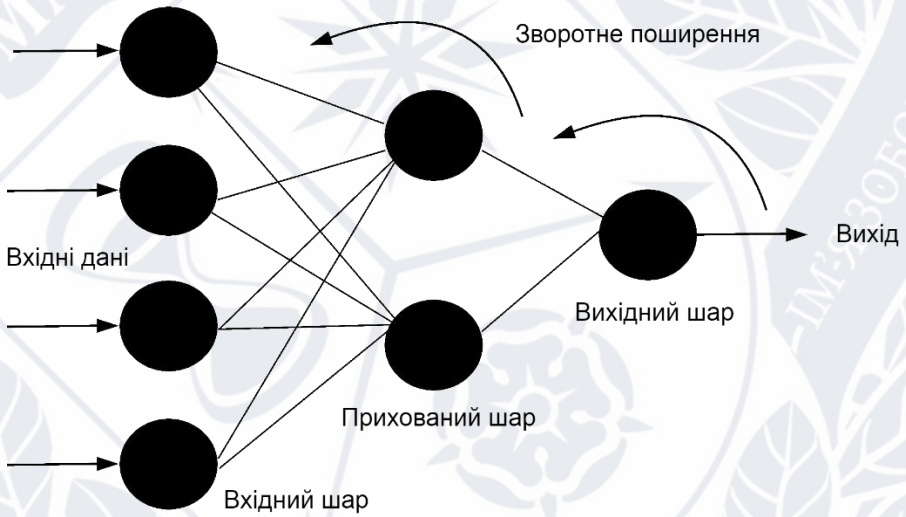

Рисунок 2.2 - Алгоритм зворотного поширення помилки у багатошаровому персептроні.

Розглядалися сценарії передачі помилки через шари перцептрона, як вихідний, так і прихований. Однак у згортковій нейромережі є ще два види передачі помилки: підвибірковий та згортковий.

2.3. Розрахунок помилки на підвибірковому шарі

Розрахунок помилок на підвибірковому шарі може мати кілька випадків. Перший випадок виникає, коли підвибірковий шар розташований перед

повнозв'язним. У цьому випадку структура шару і зв'язки між нейронами аналогічні повнозв'язному шару, тому обчислення помилок проводиться аналогічно як для прихованого шару. Наступний випадок виникає, коли підвибірковий шар перед згортковим. У цьому випадку розрахунок виконується за допомогою зворотної згортки[5]. Для розуміння цього процесу, треба спочатку зрозуміти звичайну згортку та уявити, що кожне ковзне вікно по карті ознак можна розглядати як прихований шар зі зв'язками між нейронами. Однак головна відмінність полягає в тому, що ці зв'язки спільні, тобто один зв'язок із певним значенням ваги може використовуватися в кількох парах нейронів, а не лише в одній[4]. Інтерпретація операції згортки у звичному багатошаровому контексті відображена на рис. 2.3.

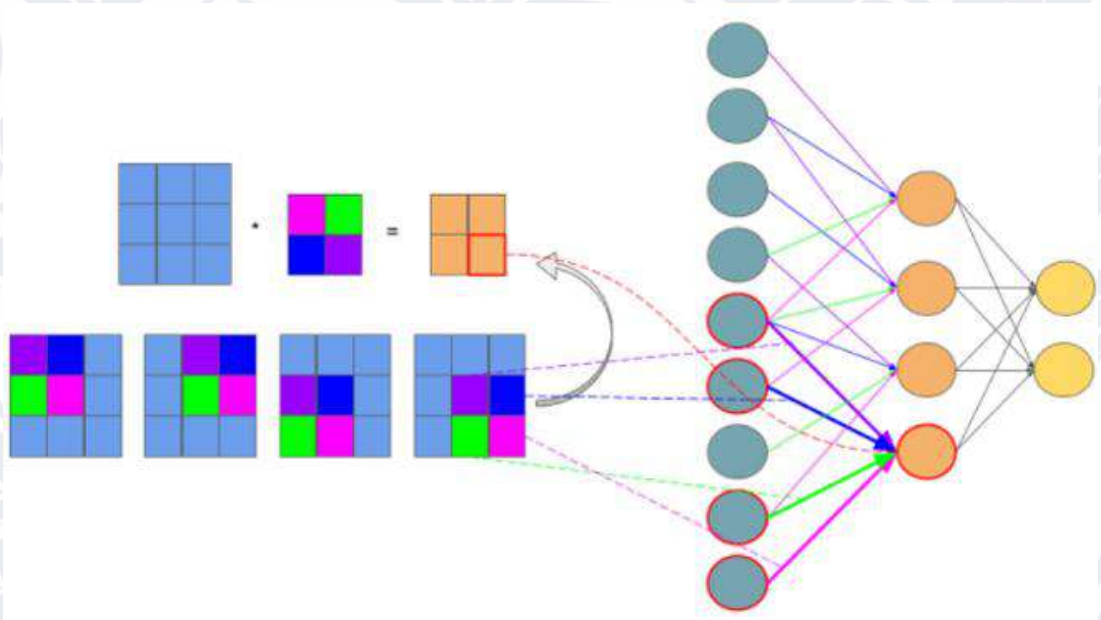

Рисунок 2.3 - Операція згортки у багатошаровому вигляді[5]

Коли операція згортки відображена у багатошаровому контексті, можна легше усвідомити, що обчислення дельт відбувається аналогічно до того, як це відбувається в прихованому шарі повнозв'язкової мережі[5]. Таким чином, виходячи з вже обчислених значень дельт для згорткового шару, можна розрахувати дельти підвибіркового шару відповідно до рис. 2.3.

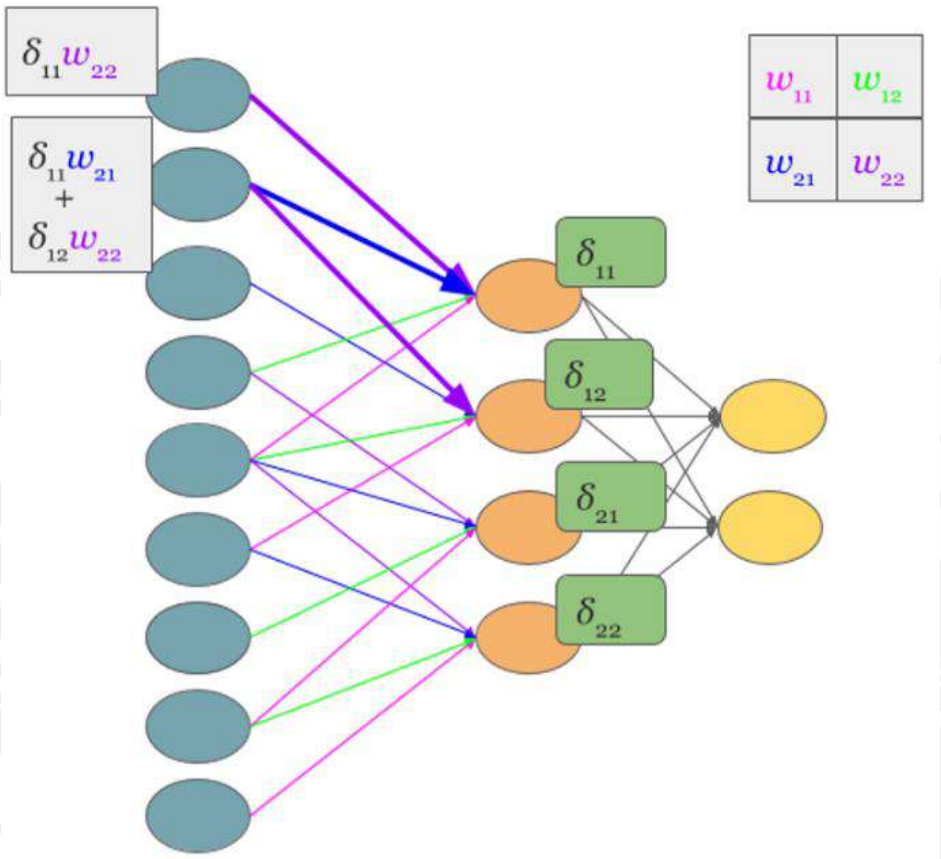

Рисунок 2.3 - Обчислення ∂ підвиборчого шару за рахунок ∂ згорткового шару та ядра[5]

Зворотне згортання - це варіант обчислення дельт, який використовується у процесі зворотного поширення помилки, проте він застосовується з урахуванням особливостей згорткових шарів. Цей процес включає поворот ядра на 180 градусів та сканування згорткової карти дельт з урахуванням змінених крайових ефектів[4]. Тобто, для виконання зворотного згортання потрібно взяти ядро згорткової карти (що знаходиться після підвибіркового шару), розгорнути його на 180 градусів і застосувати звичайну операцію згортки до зазначених раніше дельт згорткової карти. Проте при цьому необхідно також враховувати, для того щоб вікно сканування виходило за межі карти. Результатом цієї операції ілюстрований на рис. 2.4, а цикл проходження зворотного згортання можна побачити на рис. 2.5.

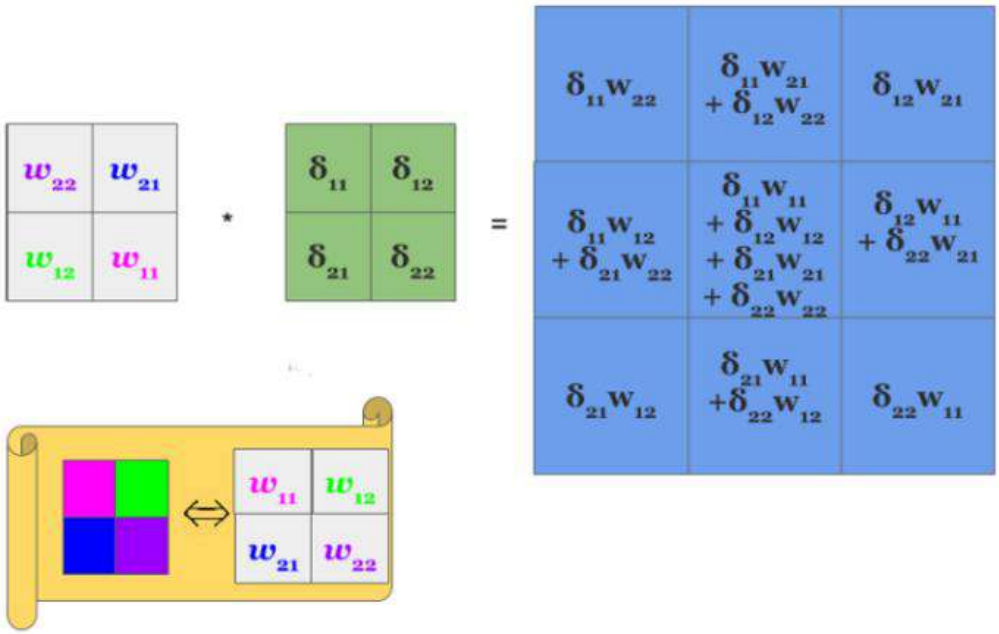

Рисунок 2.4 - Результат операції зворотного згортки[5]

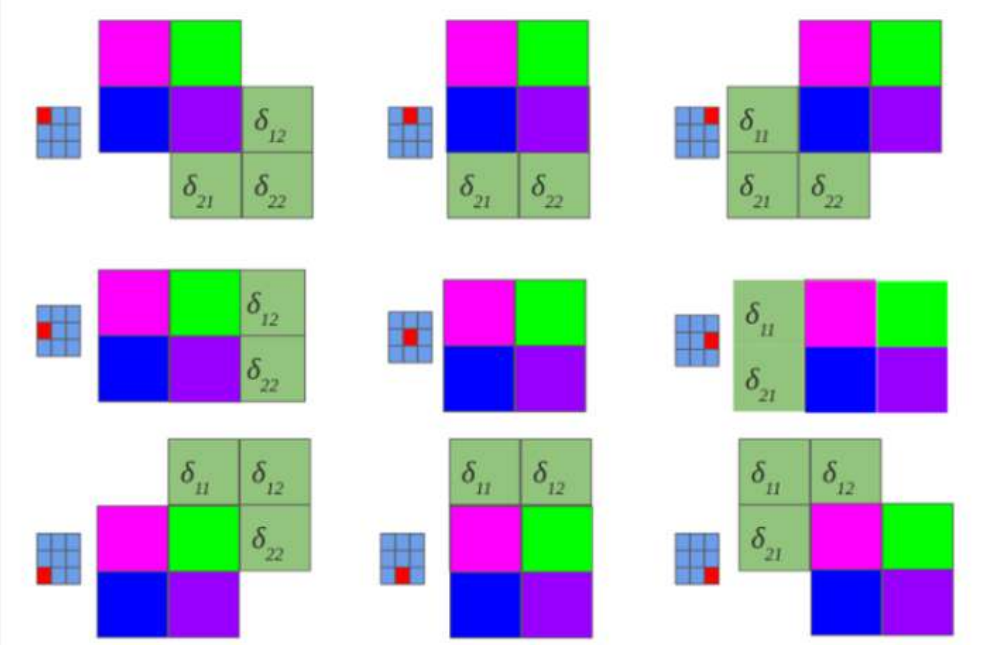

Рисунок 2.5 - Розгорнуте ядро на 180 градусів яке сканує карту згортки[5]

Після згорткового шару наступний є підвибірковим, що означає, що обчислення дельти поточного шару (згорткового) виконується відповідно. Дельта помилки не розраховується, а просто копіюється. Під час прямого поширення сигналу нейрони підвибіркового шару формуються шляхом сканування згорткового шару за допомогою не перекриваючогося вікна, під

час якого обираються нейрони з максимальним значенням. У зворотному поширенні помилки, дельта повертається до раніше обраного максимального нейрону, а інші отримують нульову дельту помилки.

2.4. Висновки до розділу 2

Здійснений аналіз методів навчання згорткових нейронних мереж яке продемонструвало ефективність алгоритмів, зокрема алгоритму зворотного розповсюдження помилки. Було розглянуто процес навчання мережі, включаючи подання зображень на вхід та порівняння отриманих результатів з очікуваними, що дозволяє мінімізувати функцію помилки шляхом коригування ваг синаптичних зв'язків. Дослідження алгоритму зворотного розповсюдження помилки показало, що він є ключовим для налаштування ваг між усіма шарами мережі, забезпечуючи зменшення функції помилки та покращення точності мережі. Зокрема, було розглянуто розрахунок помилки на підвибірковому та згортковому шарах. Це дозволило зрозуміти, як нейрони кожного шару вносять свій вклад у загальну помилку мережі і як коригуються їх ваги для досягнення оптимальних результатів. Таким чином, проведений аналіз підтвердив, що методи навчання, особливо алгоритм зворотного розповсюдження помилки, є важливими для забезпечення ефективної роботи згорткових нейронних мереж у завданнях розпізнавання та класифікації зображень.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Ознайомлення з інструментами розробки

Розглянувши різні варіанти для розробки програмного забезпечення для ідентифікації автомобілів, я обрав найбільш підходящих інструментарій. Почавши з огляду на існуючі бібліотеки та фреймворки для розробки програм на мові Python, я звернув увагу на ImageAI. Після детального аналізу можливостей цієї бібліотеки, я переконався, що вона відповідає вимогам проекту і має необхідний функціонал для реалізації задачі ідентифікації автомобілів.

Далі, для навчання та розпізнавання об'єктів на зображеннях, вирішив використовувати модель існуючої згорткової нейронної мережі RetinaNet. Її висока точність та швидкодія роботи були вирішальними факторами у виборі цієї моделі.

Для створення інтерфейсу користувача і взаємодії з програмою вирішив скористатися бібліотекою Tkinter, яка є інструментом для розробки GUI-додатків в мові Python. Її простота у використанні та гнучкість надали можливість швидко створити зручний інтерфейс для користувача.

3.1.1. Мова Python

Python - це мова програмування, яка відома своєю простотою вивчення та ефективністю в розв'язанні різноманітних завдань. Вона має чистий та читабельний синтаксис[16]. Python часто використовується для нейронних мереж з-за кількох причин. По-перше, він має велику кількість бібліотек та фреймворків для машинного навчання та глибинного навчання, наприклад TensorFlow яку я використав у свої роботі, вони спрощують розробку нейронних мереж.

TensorFlow - це програмне забезпечення для машинного навчання та глибинного навчання, розроблене компанією Google[19]. Він надає різноманітні інструменти та бібліотеки для створення та тренування нейронних мереж, включаючи конструктори графів обчислень, набори алгоритмів оптимізації, можливості розподіленого обчислення та інше[19]. TensorFlow є одним з найпопулярніших фреймворків для роботи з нейронними мережами завдяки своїй ефективності, гнучкості та розширюваності.

Друга причина - це широкий спектр функціональних можливостей мови, що дозволяє легко працювати з даними, візуалізувати результати та розробляти складні алгоритми.

3.1.2. Бібліотека ImageAi

ImageAI - це проста, але дуже потужна бібліотека Python з відкритим вихідним кодом, яка дає розробникам програмного забезпечення можливість розробляти програми та програмні утиліти з автономними можливостями глибокого навчання та комп'ютерного зору[10]. Ця бібліотека містить в собі кілька розширених функцій, пов'язаних з обробкою зображень, які включають розпізнавання об'єктів, виявлення об'єктів у відео, аналіз відео і камер, а також розпізнавання об'єктів за допомогою готових моделей[10]. Бібліотека ImageAI використовує фреймворк PyTorch для виконання операцій комп'ютерного зору[10]. PyTorch є програмним інструментом, який дозволяє працювати з нейронними мережами та глибоким навчанням[10]. Одна з його переваг - це можливість оптимізувати обчислення для роботи як на центральних, так і на графічних процесорах, що дозволяє прискорити обробку даних і покращити продуктивність.

3.1.3. Згорткова нейронна мережа RetinaNet

RetinaNet - це модель глибокого навчання для об'єктного виявлення в зображеннях[6]. Вона використовується для автоматичного розпізнавання та локалізації об'єктів на зображеннях і відео. RetinaNet здатний виявляти об'єкти різних розмірів та класів, забезпечуючи точні результати.

Архітектура згорткової нейронної мережі RetinaNet складається з 4 основних частин (рис. 3.1), кожна з яких має своє призначення:

1. Backbone - основна (базова) мережа, що служить для вилучення ознак з зображення, що надходить на вхід. Ця частина мережі є варіативною і до її основи можуть входити класифікаційні нейромережі, такі як ResNet. ResNet - це скорочення від "Residual Network" (Мережа зі збереженням залишків) [6]. Це тип глибоких нейронних мереж, який використовується для різних завдань машинного навчання, зокрема для класифікації зображень. Однією з основних особливостей ResNet є використання "залишкових блоків", які дозволяють мережі легше навчатися та уникнути проблеми витухання градієнта у глибоких мережах[6].

2. Feature Pyramid Net (FPN) – згорткова нейронна мережа, побудована у вигляді піраміди, що служить для поєднання переваг карт ознак нижніх і верхніх рівнів мережі, перші мають високу роздільну здатність, але низьку семантичну, узагальнюючу здатність; другі - навпаки[6];

3. Classification Subnet – підмережа, що витягує з FPN інформацію про класи об'єктів, вирішуючи завдання класифікації[6];

4. Regression Subnet – підмережа, що отримує з FPN інформацію про координати об'єктів на зображенні, вирішуючи завдання регресії[6].

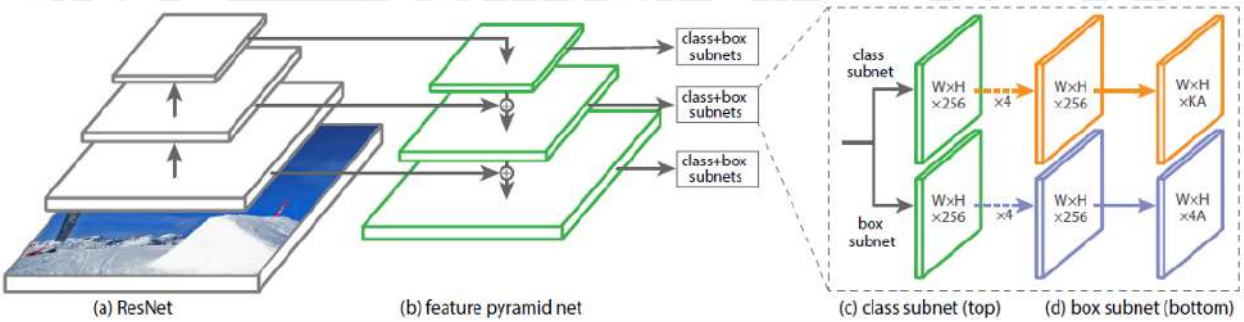

Рисунок 3.1 - Архітектура RetinaNet з backbone-мережею ResNet[6].

3.1.3. Бібліотека Tkinter

Tkinter- це стандартна бібліотека мови програмування Python, яка використовується для створення графічного інтерфейсу користувача (GUI) [22]. Вона надає набір інструментів і функцій для створення вікон, кнопок, полів введення, списків, меню та інших візуальних елементів для взаємодії з користувачем. Tkinter базується на бібліотеці Tk, яка була розроблена для мови програмування Tcl, але Tkinter надає доступ до функцій Tk через Python, що робить його доступним[22]. Tkinter є простою в освоєнні і потужною бібліотекою, яка дозволяє створювати широкий спектр GUI додатків для різних потреб[22].

3.2. Встановлення, написання та тестування програмного коду

Для початку розробки програмного забезпечення для ідентифікації автомобілів методами згорткових нейронних мереж, я встановив Python 3.12 на своєму комп'ютері. Далі, я обрав середу розробки PyCharm для зручного написання, налагодження та тестування мого коду. PyCharm - це інтегроване середовище розробки (IDE) для Python, розроблене компанією JetBrains[14]. Воно має багато корисних функцій, таких як підтримка автодоповнення, вбудована система керування версіями, інструменти для відлагодження коду та багато іншого[14].

Після встановлення PyCharm, я додав та оновив до свого проєкту необхідні бібліотеки. ImageAI, TensorFlow та Tkinter. Я встановив та оновив ці бібліотеки за допомогою менеджера пакетів pip, який дозволяє легко керувати залежностями Python для різних проєктів.

3.2.1. Написання коду

Спершу я імпортував всі необхідні бібліотеки (рис. 3.2).

```
import os
import tkinter as tk
from tkinter import filedialog, Label
import tkinter.messagebox as messagebox

import numpy as np
import cv2
from PIL import Image, ImageTk
from imageai.Detection import ObjectDetection, VideoObjectDetection
```

Рисунок 3.2 – Імпорт бібліотек

Далі я розпочав налаштування моделей для виявлення об'єктів на зображеннях і відео (рис. 3.3). Обравши модель RetinaNet та завантаживши попередньо навчену модель, я також визначив кастомні об'єкти, які хочу виявити за допомогою цієї моделі.

```
EXECUTION_PATH = os.getcwd()


detector = ObjectDetection()
video_detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
video_detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(EXECUTION_PATH, "_internal/retinanet_resnet50_fpn_coco-eeacb38b.pth"))
video_detector.setModelPath(os.path.join(EXECUTION_PATH, "_internal/retinanet_resnet50_fpn_coco-eeacb38b.pth"))
detector.loadModel()
video_detector.loadModel()
custom_objects = detector.CustomObjects(car=True)
video_custom_objects = video_detector.CustomObjects(car=True)
```

Рисунок 3.3 – Налаштування моделей для виявлення об'єктів на зображеннях і відео.

Потім я написав різні функцій для роботи з файлами. Я створив функції для відкриття файлів, обробки зображень і відео (рис. 3.4), а також для збереження оброблених зображень (рис. 3.5). Це допоможе мені легко працювати з вхідними даними та зберігати результати моєї роботи.

Воно поділяється в п’ять етапів:

1. Функція, яка відповідає за відкриття діалогового вікна для вибору файлу. Після того як користувач вибрав файл, він завантажується та

відображається у вікні програми. Функція також зберігає зображення у глобальну змінну `selected_image`, яка може використовуватися для подальшої обробки. Це допомагає вибрати вхідні дані для подальшого аналізу чи обробки в моїй програмі.

2. Функція, яка обробляє вибране користувачем зображення. Після перевірки, чи було вибране зображення, воно передається в метод `detectObjectsFromImage` об'єкта `detector`, який використовується для виявлення об'єктів на зображенні. Після обробки отриманих даних, нове зображення з позначеннями об'єктів зберігається у змінну `processed_image`. Це зображення потім відображається у вікні програми. Таким чином, ця функція відповідає за обробку вхідного зображення та показ результів аналізу.

3. Функція, яка дозволяє користувачеві завантажити оброблене зображення. Спочатку перевіряється, чи було оброблене зображення (`processed_image`). Якщо так, то виводиться діалогове вікно для вибору місця та імені файлу, де буде збережене зображення. Після цього, якщо користувач вибрав шлях та ім'я файлу, оброблене зображення зберігається за вказаним шляхом та виводиться повідомлення про успішне збереження.

4. Функція, яка відкриває діалогове вікно для вибору відеофайлу. Після вибору файлу користувачем та отримання шляху до вибраного відеофайлу, шлях до цього відеофайлу зберігається у глобальній змінній `video_path`. Ця функція дозволяє користувачеві вибрати відеофайл для подальшої обробки.

5. Функція, яка обробляє відеофайл за допомогою моделі виявлення об'єктів `video_detector`. Спочатку перевіряється, чи було вибрано відеофайл (чи `video_path` не є `None`). Якщо відеофайл вибраний, то користувачу відображається діалогове вікно для вибору місця збереження вихідного відеофайлу. Після отримання шляху для збереження результату обробки, викликається метод `detectObjectsFromVideo` з параметрами, включаючи вхідний шлях до відеофайлу, вихідний шлях для результату, власне модель для

виявлення об'єктів, параметр для ведення прогресу (log_progress) та параметр для відображення відсоткової ймовірності (display_percentage_probability).

Разом ці функції виглядають наступним чином:

```
def open_file_dialog():
    file_path = filedialog.askopenfilename()
    if file_path:
        image = Image.open(file_path)
        photo = ImageTk.PhotoImage(image)
        label.config(image=photo)
        label.image = photo
        global selected_image
        selected_image = np.array(image)


1 usage   Maks
def process_image():
    if selected_image is not None:
        image, detections = detector.detectObjectsFromImage(
          custom_objects=custom_objects,
          input_image=selected_image,
          minimum_percentage_probability=55,
          display_percentage_probability=False,
          output_type='array',
        )
        image = Image.fromarray(image)
        global processed_image
        processed_image = image
        photo = ImageTk.PhotoImage(image)
        label.config(image=photo)
        label.image = photo
```

Рисунок 3.4 – Функції для відкриття файлів, обробки зображень і відео.

```python
def download_image():
    if processed_image is not None:
        file_path = filedialog.asksaveasfilename(
            defaultextension=".jpg", filetypes=[("JPEG files", "*.jpg")]
        )
        if file_path:
            processed_image.save(file_path)
            print(f"Зображення успішно збережено як {file_path}")


1 usage  Maks
def open_video_dialog():
    file_path = filedialog.askopenfilename()
    if file_path:
        global video_path
        video_path = file_path


1 usage  Maks
def process_video():
    if video_path is not None:
        output_path = filedialog.asksaveasfilename()
        video_detector.detectObjectsFromVideo(
            input_file_path=video_path,
            output_file_path=output_path,
            custom_objects=video_custom_objects,
            log_progress=True,
            display_percentage_probability=False
        )
        messagebox.showinfo("Повідомлення", "Відео оброблено успішно!")
```

Рисунок 3.5 – Функції збереження оброблених зображень.

Після написання функцій для роботи з файлами, я створив графічний інтерфейс користувача, щоб легше взаємодіяти з програмою (рис 3.6). Використовуючи Tkinter, я створив вікно програми, розмістив на ньому кнопки для взаємодії з користувачем, а також мітку для відображення зображень. Таким чином, я зробив мою програму більш зручною для використання.

Спочатку створив головне вікно програми за допомогою об'єкта **tk.Tk()**. Встановлюються мінімальні розміри та назва вікна за допомогою методів **minsize** та **title** відповідно.

Потім створив кнопки для виклику функцій з обробки зображень та відео: "Завантажити фото", "Завантажити відео", "Обробити фото", "Обробити відео" та "Зберегти фото". Кожній кнопці назначається текст, а також функція, яка буде викликатися при натисканні кнопки.

Також створив мітку (**label**), яка використовуватиметься для відображення вибраного зображення.

Потім визначаються глобальні змінні **selected_image**, **processed_image** та **video_path**, які використовуються для зберігання вибраного зображення, обробленого зображення та шляху до відеофайлу відповідно.

Викликається метод **mainloop()**, який запускає головний цикл обробки подій для вікна програми, що дозволяє користувачу взаємодіяти з інтерфейсом користувача.

```
root = tk.Tk()
root.minsize(200, 200)
root.title("Робота")

#кнопка для загрузки фото
button_load = tk.Button(root, text="Загрузити фото", command=open_file_dialog)
button_load.pack(pady=5)

#кнопка для загрузки відео
button_load = tk.Button(root, text="Загрузити відео", command=open_video_dialog)
button_load.pack(pady=5)

#кнопка для обработки фото
button_process = tk.Button(root, text="Обробка фото", command=process_image)
button_process.pack(pady=5)

#кнопка для обработки відео
button_process = tk.Button(root, text="Обробка відео", command=process_video)
button_process.pack(pady=5)

#кнопка для збереження фото
button_download = tk.Button(root, text="Зберегти фото", command=download_image)
button_download.pack(pady=5)

label = tk.Label(root)
label.pack()

selected_image = None
processed_image = None
video_path = None

root.mainloop()
```

Рисунок 3.6 - Створення графічного інтерфейсу користувача.

3.2.2. Тестування та налагодження програми для впевненості в її працездатності

Після написання коду для програми, я запустив його за допомогою середовища PyCharm, натиснувши кнопку "Run". Це ініціювало запуск програми, і відкрилося нове вікно із графічним інтерфейсом(рис. 3.7).

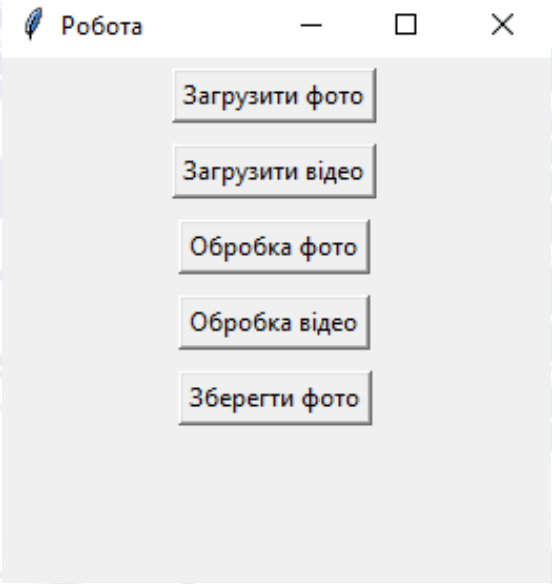

Рисунок 3.7 – Інтерфейс програми

У цьому вікні п’ять кнопок:

- "Завантажити фото": Натиснувши на цю кнопку, мені відкрилося вікно діалогу, де я міг вибрати зображення на моєму комп'ютері.
- "Завантажити відео": При натисканні на цю кнопку, також відкривалося вікно діалогу, де я міг вибрати відеофайл.
- "Обробити фото": Після того, як я завантажив зображення, я натиснув на цю кнопку, щоб запустити обробку фотографії. Програма розпізнала об'єкти на зображенні та відобразила їх на екрані.
- "Обробити відео": Після натискання обробки відео буде запропоновано вказати шлях куди оброблене відео буде зберігатися та вказати його назву за бажанням.

- "Зберегти фото": Після натискання буде запропоновано вказати шлях куди оброблене фото буде зберігатися та вказати його назву за бажанням.

Я відкрив фото в програмі та воно відобразилося нижче (рис. 3.8). Після чого натиснув кнопку Обробка фото та воно оновилося на обробленне фото з виділеними автомобілями (рис. 3.9) Після чого я зберіг його собі на робочій стіл (рис.3.10).

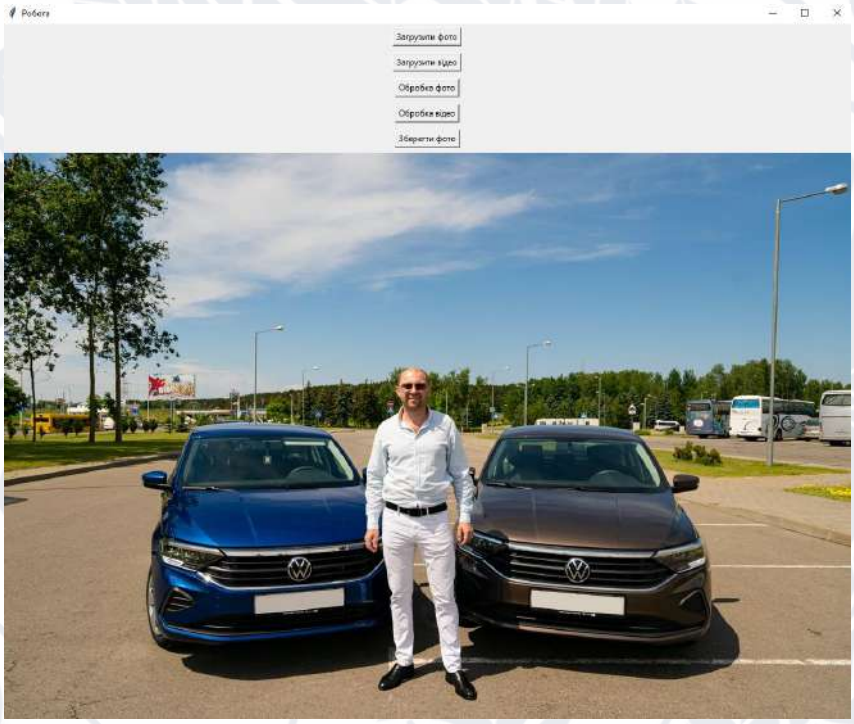

Рисунок 3.8 – Відкрив тестове фото

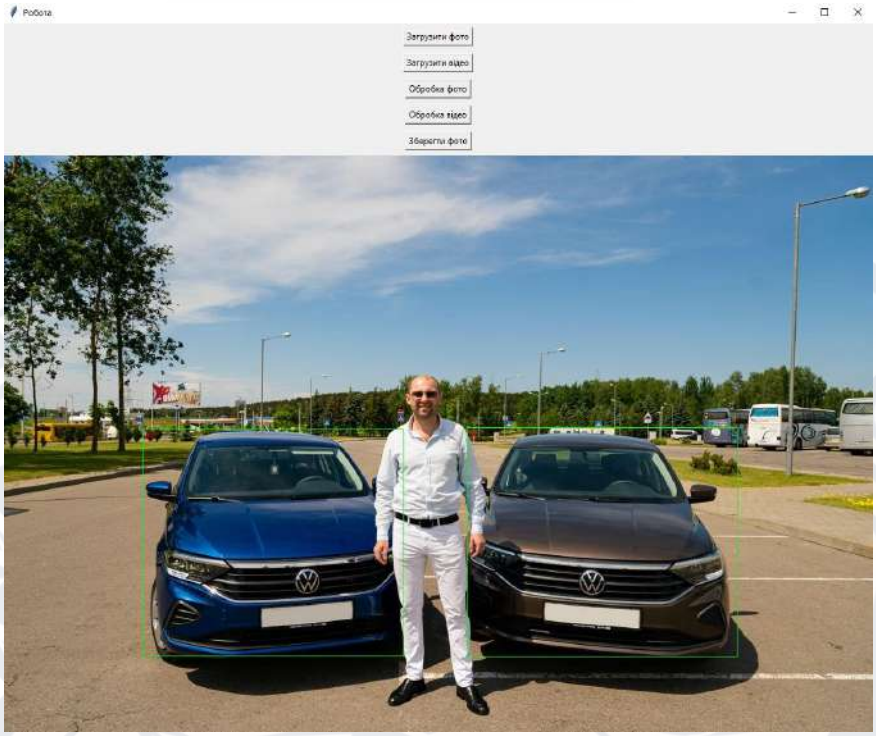

Рисунок 3.9 – Оброблене тестове фото

Рисунок 3.10 – Збережене фото на робочому столі

Після перевірки правильної роботи обробки фото, я почав перевіряти відео. Я натиснув на кнопку «Загрузити відео», після чого обрав тестове відео. Через велике навантаження на систему я обрав коротке відео протяжністю 1 секунда та частотою кадрів 25. Після чого натиснувши кнопку обробити відео мені запропонувало обрати шлях куди зберегти оброблене відео та вказати його назву. Я почекав деякий час і коли відео було оброблене, вискочило повідомлення «Відео оброблено успішно!» (рис 3.11).

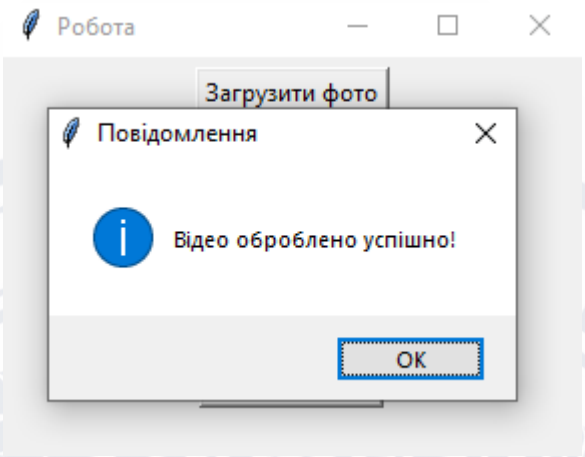

Рисунок 3.11 – Повідомлення про успішність операції.

3.2.3. Підготовка виконуваного файлу

Після завершення написання програми я перевірив її роботу та функціонал, щоб переконатися, що все працює коректно. Після успішних перевірок я скомпілював програму в виконуваний файл .exe за допомогою інструмента pipinstaller. Pipinstaller - це інструмент, який дозволяє створювати виконувані файли з Python-програм. Він дозволяє перетворювати Python-код у виконуваний файл, який можна запустити без встановлення інтерпретатора Python або будь-яких інших додаткових бібліотек.

3.4. Висновки до розділу 3

У цьому розділі здійснений аналіз інструментів розробки для успішної реалізації проекту. На етапі встановлення, написання та тестування програмного коду було створено робочу програму, яка успішно пройшла всі етапи тестування та налагодження, що забезпечило її працездатність і надійність.

ВИСНОВКИ

У ході даної роботи я досліджував можливості розробки програмного забезпечення для ідентифікації автомобілів за допомогою згорткових нейронних мереж. Провівши аналіз згорткових нейронних мереж та їхніх можливостей, а також ознайомившись з процесом навчання таких мереж, я реалізував практичну частину роботи. Використовуючи бібліотеки ImageAI, TensorFlow та Tkinter, я розробив програмне забезпечення з інтуїтивним інтерфейсом користувача, яке може виявляти автомобілі на зображеннях та відео. Результатом моєї роботи стало функціональне програмне забезпечення, яке може бути використане для різних завдань, пов'язаних з ідентифікацією автомобілів на зображеннях та відео. Ця робота дала мені можливість поглибити розуміння принципів роботи згорткових нейронних мереж та їхніх можливостей у сфері комп'ютерного зору, а також отримати практичні навички у розробці програмного забезпечення для подібних завдань.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Dumoulin, Vincent, and Francesco Visin. “A guide to convolution arithmetic for deep learning.” stat 1050 (2016): 23.
2. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.,Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation 1(4), 541–551 (1989)
3. Tan M., Le Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. 2019. URL: arxiv.org/abs/1905.11946
4. Backpropagation In Convolutional Neural Networks [Електронний ресурс]. 2016: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/
5. Convolutional Neural Networks backpropagation: from intuition to derivation [Електронний ресурс]. 2016: https://grzegorzgwardys.wordpress.com/2016/04/22/8/
6. Zeng N. RetinaNet Explained and Demystified [Електронний ресурс]. 2018 URL: blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified
7. Review: RetinaNet — Focal Loss (Object Detection) [Електронний ресурс]. 2019 URL: towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4
8. Tsung-Yi Lin Focal Loss for Dense Object Detection. 2017. URL: arxiv.org/abs/1708.02002
9. The intuition behind RetinaNet [Електронний ресурс]. 2018 URL: medium.com/@14prakash/the-intuition-behind-retinanet-eb636755607d
10. Official English Documentation for ImageAI [Електронний ресурс]. 2022 URL; https://imageai.readthedocs.io/en/latest/#
11. Deep Learnin by Ian Goodfellow, Yoshua Bengio, and Aaron Courville. [Електронний ресурс]. (https://www.deeplearningbook.org/)

12. Convolutional Neural Networks for Visual Recognition (CS231n) by Stanford University. [Електронний ресурс]. (https://cs231n.github.io/convolutional-networks/)

13. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way by Sumit Saha. [Електронний ресурс]. (https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215)

14. Official PyCharm Documentation: https://www.jetbrains.com/help/pycharm/quick-start-guide.html

15. Mastering PyCharm" by Quazi Nafiul Islam. (https://www.packtpub.com/product/mastering-pycharm/9781801077315)

16. Official Python Documentation: https://docs.python.org/3/

17. Automate the Boring Stuff with Python by Al Sweigart. (https://automatetheboringstuff.com/)

18. Python Crash Course" by Eric Matthes. (https://nostarch.com/pythoncrashcourse2e)

19. Official TensorFlow Documentation: https://www.tensorflow.org/

20. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron. (https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/)

21. Deep Learning with TensorFlow 2 and Keras by Antonio Gulli, Amita Kapoor, and Sujit Pal. (https://www.amazon.com/Deep-Learning-TensorFlow-Keras-Regression/dp/1838823417)

22. Official Tkinter Documentation (for Python 3.x): https://docs.python.org/3/library/tk.html

23. Python GUI Programming with Tkinter by Alan D. Moore. (https://www.amazon.com/Python-GUI-Programming-Tkinter-Moore/dp/1735868916)

24. Tkinter GUI Application Development Blueprints by Bhaskar Chaudhary. (https://www.packtpub.com/product/tkinter-gui-application-development-blueprints-second-edition/9781788837460)

25. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.,Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation 1(4), 541–551 (1989)

ДОДАТОК А

```
import os
import tkinter as tk
from tkinter import filedialog, Label
import tkinter.messagebox as messagebox
import numpy as np
import cv2
from PIL import Image, ImageTk
from imageai.Detection import ObjectDetection, VideoObjectDetection
EXECUTION_PATH = os.getcwd()
detector = ObjectDetection()
video_detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
video_detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(EXECUTION_PATH,
"_internal/retinanet_resnet50_fpn_coco-eeacb38b.pth"))
video_detector.setModelPath(os.path.join(EXECUTION_PATH,
"_internal/retinanet_resnet50_fpn_coco-eeacb38b.pth"))
detector.loadModel()
video_detector.loadModel()
custom_objects = detector.CustomObjects(car=True)
video_custom_objects = video_detector.CustomObjects(car=True)
def open_file_dialog():
    file_path = filedialog.askopenfilename()
    if file_path:
        image = Image.open(file_path)
        photo = ImageTk.PhotoImage(image)
        label.config(image=photo)
        label.image = photo
        global selected_image
        selected_image = np.array(image)
def process_image():
    if selected_image is not None:
        image, detections = detector.detectObjectsFromImage(
          custom_objects=custom_objects,
          input_image=selected_image,
          minimum_percentage_probability=55,
          display_percentage_probability=False,
          output_type='array',
        )
        image = Image.fromarray(image)
        global processed_image
        processed_image = image
        photo = ImageTk.PhotoImage(image)
        label.config(image=photo)
        label.image = photo
def download_image():
    if processed_image is not None:
```

```
        file_path = filedialog.asksaveasfilename(
            defaultextension=".jpg", filetypes=[("JPEG files", "*.jpg")]
        )
        if file_path:
            processed_image.save(file_path)
            print(f"Зображення успішно збережено як {file_path}")
def open_video_dialog():
    file_path = filedialog.askopenfilename()
    if file_path:
        global video_path
        video_path = file_path
def process_video():
    if video_path is not None:
        output_path = filedialog.asksaveasfilename()
        video_detector.detectObjectsFromVideo(
            input_file_path=video_path,
            output_file_path=output_path,
            custom_objects=video_custom_objects,
            log_progress=True,
            display_percentage_probability=False
        )
        messagebox.showinfo("Повідомлення", "Відео оброблено успішно!")
root = tk.Tk()
root.minsize(200, 200)
root.title("Робота")
#кнопка для загрузки фото
button_load = tk.Button(root, text="Загрузити фото", command=open_file_dialog)
button_load.pack(pady=5)
#кнопка для загрузки відео
button_load = tk.Button(root, text="Загрузити відео", command=open_video_dialog)
button_load.pack(pady=5)
#кнопка для обработки фото
button_process = tk.Button(root, text="Обробка фото", command=process_image)
button_process.pack(pady=5)
#кнопка для обработки відео
button_process = tk.Button(root, text="Обробка відео", command=process_video)
button_process.pack(pady=5)
#кнопка для збереження фото
button_download = tk.Button(root, text="Зберегти фото", command=download_image)
button_download.pack(pady=5)
label = tk.Label(root)
label.pack()
selected_image = None
processed_image = None
video_path = None
root.mainloop()
```