Міністерство освіти і науки України
Харківський національний автомобільно-дорожній університет

Механічний факультет

Кафедра комп'ютерних наук і інформаційних систем

ДИПЛОМНА РОБОТА
магістра

НЕЙРОМЕРЕЖЕВЕ МОДЕЛЮВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Завідувачка кафедри канд. техн. наук, доцентка Г.А. Плєхова

Нормоконтролер, к. т. н., доцентка Г. А. Плєхова

Керівник, д. т. н., професор А. М. Куцин

Студент гр. МК-61-23 О. С. Ріпний

Харків – 2024

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ АВТОМОБІЛЬНО-ДОРОЖНІЙ УНІВЕРСИТЕТ

Факультет Механічний
Кафедра Комп’ютерних наук та інформаційних систем
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань 12 Інформаційні технології
Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
Г. А. Плєхова
«__»_____2024 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Ріпний Олексій Сергійович

1. Тема роботи: Нейромережеве моделювання інформаційних систем.
Керівник: Куцин Андрій Миколайович, д. е. н., професор
Затверджені рішенням Вченої ради механічного факультету «10» жовтня2024 року, протокол № 136.
2. Термін здачі студентом закінченої роботи «9» грудня 2024 р.
3. Вихідні дані до роботи:
3.1. Мета: аналіз та дослідження моделей штучних нейронних мереж, побудова нейромережевої моделі визначення об’єктів, опис основних принципів розробки мобільного додатку для визначення дрібних деталей автомобіля та автомобільного обладнання.
3.2. Об’єкт дослідження – перспективні інструменти розробки та навчання нейромережевих моделей та середовища програмування на мові Python Keras для аналізу даних в інформаційних системах, для визначення типів та видів деталей автомобільного обладнання.
3.3. Предмет дослідження – автоматизація моделювання процесів визначення за допомогою нейронних мереж, обґрунтування принципів створення програмного додатку для визначення деталей автомобільних систем.
3.4 Завдання – Дослідження принципів нейромережевого моделювання визначення об’єктів, розробка навчання нейронних мереж за моделлю, визначення основних принципів розробки мобільного додатку для визначення деталей автомобіля за допомогою комп'ютерного зору.
3.5. Інструментальні засоби – Internet Google, Information Services: Data center ХНАДУ (файловий архів ХНАДУ: www.files.khadi.kharkov.ua); система комп'ютерної математики Matlab, середовище розробки на Keras Tenzor Flow, Google Colaboratory.
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити):
1. Аналіз існуючих рішень нейромережевого моделювання. Проблема та постановка задачі; 2. Теоретичне обґрунтування дослідження (визначення та отримання наукового результату); 3.Експериментальна (практична) частина (визначення та отримання практичного результату);4.Техніко-економічне обґрунтування та

рекомендації щодо впровадження наукового (2-й розділ) та практичного (3-й розділ) результатів. У додатках - демонстраційний матеріал (до 12 слайдів Power Point).
5. Перелік демонстраційного матеріалу (з точним зазначенням обов'язкових слайдів): постановка задачі; аналіз попередніх рішень; алгоритм виконання дослідження (проектування); інструментальні засоби; структурний аналіз предмету дослідження (проектування); результати дослідження (проектування) – опис функціональної, структурної схеми; наукові та практичні результати. Техніко – економічне обґрунтування результатів: рекомендації з впровадження, економічне обґрунтування результатів,висновки: усього 8-15 слайдів включно з титульним слайдом.

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів виконання дипломної роботи	Термін виконання	Примітка (обсяг у % тарезультат)
1	Аналіз існуючих рішень нейромережевого моделювання. Постановка задачі та визначення особливостей предмету дослідження	10.10.2024 р. – 15.10.2024 р.	20%Вступ та контент 1-го розділу ПЗ алгоритм виконання дослідження
2	Теоретичне обґрунтування дослідження	15.10.2024 р. – 24.10.2024 р.	40% 2 розділ ПЗ визначення наукового результату
3	Експериментальна частина розробки та імплементація наукового результату	25.10.2024 р. – 15.11.2024 р.	60% 3 розділ ПЗ практичний результат
4	Оформлення пояснювальної записки – ПЗ включно з демонстраційними матеріалами.	15.11.2024 р. – 30.11.2024 р.	80%
5	Створення електронного варіанту пояснювальної записки, презентації, підготовкадоповіді.	1.12.2024 р. – 9.12.2024 р.	100% файли: *.doc, *.pdf , *.ppt, підготовка до захисту (тренінг, отримання рецензії,переплести ПЗ, отримати допуск)

8. Дата видачі завдання « 10 » жовтня 2024 р.

Керівник професор кафедри КНІС ХНАДУ А. М. Куцин

9. Завдання отримано « 10 » жовтня 2024 р.

Студент О. С. Ріпний

РЕФЕРАТ

Дипломна робота: 93 с., 28 рис., 12 формул, 2 таблиці, 28 джерел.

НЕЙРОННІ МЕРЕЖІ, МОДЕЛЮВАННЯ, СИСТЕМИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ, ВИЗНАЧЕННЯ ОБ'ЄКТІВ, TENSORFLOW, PYTHON, МОБІЛЬНИЙ ПРОГРАМНИЙ ДОДАТОК.

Об'єкт дослідження – перспективні інструменти розробки та навчання нейромережевих моделей та середовища програмування на мові Python з бібліотеками TensorFlow і Keras для аналізу даних в інформаційних системах, для визначення типів та видів деталей автомобільного обладнання.

Предмет дослідження – автоматизація моделювання процесів визначення за допомогою нейронних мереж, обґрунтування принципів створення програмного додатку для визначення деталей автомобільних систем.

Метою дипломної роботи є аналіз та дослідження моделей штучних нейронних мереж, побудова нейромережевої моделі визначення об'єктів, опис основних принципів розробки мобільного додатку для визначення дрібних деталей автомобіля та автомобільного обладнання.

Основні завдання та характеристики бажаних результатів (наукові або практичні) – описати цілі, функції, види існуючих методів розробки моделей процесів за допомогою штучних нейронних мереж, побудувати нейромережеву модель визначення об'єктів, а саме деталей автомобільного обладнання, провести певне число моделювань для навчання роботи нейронної мережі та отримання результатів за моделюванням, наближених до дійсних результатів, визначити основні принципи створення мобільного додатку, що використовує навчану нейронну мережу, для визначення деталей автомобіля та інструментів. А також побудувати мережу для прогнозування значень часових рядів характеристик транспортного потоку за допомогою Deep Network Designer системи комп'ютерної математики Matlab.

При проведенні досліджень в цій роботі було проаналізовано досягнення у побудові моделей процесів за допомогою нейронних мереж, розроблена та

навчана нейронна мережа для визначення об’єктів, надані принципи розробки мобільних додатків з використання навченої нейронної мережі для визначення деталей автомобільного обладнання, окремим розділом проведено нейромережеве моделювання часових рядів значень характеристик транспортного потоку дорожньої мережі міста.

Під час розробки дипломної роботи автором опубліковано статтю у збірнику наукових праць Всеукраїнської науково-методичної конференції:

Ріпний О. С., Козачок Л. М. Основні принципи нейромережевого підходу до визначення об’єктів автомобільного обладнання. *«Інформаційні технології в освітньому процесі ЗВО»*: зб. наук. пр. за матеріалами Всеукраїнської науково-методичної конференції, м. Харків, 27 листопада 2024 р., Харків, 2024. С. 20-23.

ЗМІСТ

ВСТУП

Основними завданнями дослідження принципів та методів нейромережевого моделювання є моделювання процесів, що відбуваються в інформаційній системі на основі отриманої вхідної інформації. Даний метод моделювання використовується для аналізу закономірностей процесу або процесів, цілей прогнозування значень досліджуваних показників, виділення, визначення оь’єктів інформації, генерації нових даних. Основними завданнями даної роботи є дослідження принципів нейромережевого моделювання визначення об’єктів, розробка навчання нейронних мереж за моделлю, визначення основних принципів розробки мобільного додатку для визначення деталей автомобіля за допомогою комп'ютерного зору.

Основна мета - дослідження моделей штучних нейронних мереж, побудова нейромережевої моделі визначення об’єктів, опис основних принципів розробки мобільного додатку для визначення дрібних деталей автомобіля та автомобільного обладнання.

У дипломній роботі проводилася розробка основних принципів створення мобільного додатку для визначення та розпізнавання деталей автотранспортних засобів, дрібних механічних деталей та інструменту, в основі роботи використовується нейромережева модель, побудована на певних наборах даних, навчана найкращому розпізнаванню та виділенню потрібних об’єктів та перетворена для використання у мобільних пристроях. Також вирішується завдання прогнозування параметрів управління транспортними потоками, у рішенні побудовано нейромережеву модель транспортних процесів, за допомогою якої виконано прогнозування значень інтенсивності транспортного потоку у певні моменти часу.

Результати дипломної роботи мають подальший розвиток і можуть застосовуватись для побудови інших моделей та для дослідження інших характеристик процесів у транспортних системах.

Методи, які використовувались для поставленого завдання та для досягнення мети досліджень і отримання результату, включають в себе комп'ютерні технології систем комп'ютерної математики, методи програмування обчислювальних процесів, методи моделювання, методи побудови штучних нейронних мереж, методи їх навчання та отримання результатів роботи навчанної нейронної мережі.

Об'єкт дослідження – перспективні інструменти розробки та навчання нейромережевих моделей та середовища програмування на мові Python з бібліотеками TensorFlow і Keras для аналізу даних в інформаційних системах, для визначення типів та видів деталей автомобільного обладнання.

Предмет дослідження – автоматизація моделювання процесів визначення за допомогою нейронних мереж, обґрунтування принципів створення програмного додатку для визначення деталей автомобільних систем.

1 ОСНОВНІ ПОНЯТТЯ НЕЙРОМЕРЕЖЕВОГО МОДЕЛЮВАННЯ. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Одним із найпотужніших інструментів створення інтелектуальних систем є штучні нейронні мережі (ШНМ), що моделюють базові механізми обробки інформації, властиві людському мозку. Відомо, що мозок працює принципово іншим чином і найчастіше ефективніше, ніж будь-яка обчислювальна машина, створена людиною [3]. Саме цей факт протягом багатьох років спонукає вчених до роботи зі створення та дослідження штучних нейронних мереж.

Щоб усвідомити масштаби проблеми створення машини, яка працює так само досконало, як наш мозок, достатньо замислитись над деякими рутинними завданнями, які ми виконуємо щодня.

Мозок є надзвичайно складною системою обробки інформації. Він має здатність організовувати свої структурні компоненти, які називають нейронами, так, щоб вони могли виконувати конкретні завдання (розпізнавання образів, обробка сигналів органів чуття, моторні функції) у багато разів швидше, ніж найшвидше діючі сучасні комп'ютери. Повсякденні завдання не вимагають великої інтелектуальної напруги, але рішення кожної з них включає безліч точно розрахованих кроків [7]. Складність розв'язання таких завдань можна відчути, намагаючись програмувати комп'ютерну систему для розпізнавання об'єктів за їх зовнішнім виглядом або іншими ознаками, що приймає рішення, залежно від контексту тощо.

Розвиток нейронів пов'язаний з поняттям пластичності мозку – здатності налаштування нервової системи відповідно до навколишніх умов. Пластичність відіграє найважливішу роль у роботі нейронів як елементарні одиниці обробки інформації в людському мозку. Досвід накопичується з часом життя людини. Аналогічним чином в ШНМ відбувається налаштування штучних нейронів. У загальному випадку ШНМ є машиною, що моделює спосіб розв'язання мозком конкретної задачі. Ця мережа реалізується за допомогою електронних компонентів (нейропроцесорів) або моделюється програмою, яка виконується

на цифровому комп'ютері. Для того, щоб досягти високої продуктивності, в ШНМ використовують множину взаємозв'язків між елементарними клітинами для обчислень - нейронами. Серед безлічі визначень нейронних мереж найбільш точним є визначення ШНМ як адаптивної машини:

Штучна нейронна мережа - це розподілений паралельний процесор, що складається з типових елементів обробки інформації, що накопичують експериментальні знання, та надає їх для подальшої обробки [5].

Нейронна мережа має наступні властивості, що визначають її роботу:

- знання надходять у нейронну мережу з навколишнього середовища та використовуються мережею в процесі навчання;
- для накопичення знань використовуються міжнейронні зв'язки, які називаються синаптичними вагами.

Процедура, яка використовується для здійснення процесу навчання, називається алгоритмом навчання. Її функція полягає в модифікації синаптичних ваг ШНМ таким чином, щоб мережа набула необхідних властивостей. Модифікація ваг є традиційним способом навчання ШНМ також створюються нові синаптичні зв'язки [12].

Таким чином, ШНМ мають та використовують обчислювальну потужність завдяки двом основним своїм властивостям: паралельно-розподіленій структурі та здатності навчатися та узагальнювати отримані знання Під властивістю узагальнення розуміється здатність ШНМ генерувати правильні виходи для вхідних сигналів, які не були враховані у процесі навчання (тренування). Ці дві властивості роблять ШНМ системою обробки інформації, здатною вирішувати складні багатовимірні завдання, які на сьогоднішній день є важкорозв'язними.

Автономні ШНМ не можуть забезпечити готові рішення, їх слід інтегрувати у складні системи. Комплексне завдання можна розбити на ряд більш простих завдань, частина з яких може бути вирішена нейронними мережами. Області застосування ШНМ дуже різноманітні: розпізнавання та аналіз тексту та мовлення, семантичний пошук, експертні системи та системи

підтримки прийняття рішень, передбачення курсів акцій, системи безпеки, системи безпеки транспорті, нейромережеві програмні пакети для фінансових ринків, моніторинг та автоматична рубрикація новин, розпізнання теми текстових повідомлень інший приклад використання ШНМ.

1.1 Сфери застосування штучних нейронних мереж

Нині ШНМ є важливим розширенням поняття обчислення. Вони вже дозволили впоратися з низкою непростих проблем і обіцяють створення нових програм та пристроїв, здатних вирішувати завдання, які поки що під силу тільки людині.

Сучасні нейрокомп'ютери використовуються в основному у вигляді програмних продуктів і тому рідко використовують свій потенціал «паралелізму» [3]. Епоха справжніх паралельних нейрообчислювань розпочнеться з появою на ринку апаратних реалізацій спеціалізованих нейрочіпів та плат розширення, призначених для обробки мови, відео, статичних зображень та інших типів образної інформації.

Іншою сферою застосування ШНМ є їх використання у спеціалізованих програмних агентах-роботах, призначених для обробки інформації, а не для фізичної роботи. Інтелектуальні помічники повинні полегшувати користувачам спілкування з комп'ютером. Їхньою відмінністю буде прагнення якнайкраще зрозуміти, що від них вимагається, за рахунок спостереження та аналізу поведінки [18]. Намагаючись виявити в цій поведінці деякі закономірності, інтелектуальні агенти повинні своєчасно запропонувати свої послуги для виконання певних операцій, наприклад для фільтрації повідомлень новин, для резервного копіювання документів, над якими працює користувач, і т.п. Саме тому ШНМ, здатні узагальнювати дані та знаходити в них закономірності, є природним компонентом таких програмних агентів.

1.2 Біологічні нейрони. Нейрокомп’ютерна обробка інформації

Як було зазначено, штучні нейронні мережі моделюють базові механізми обробки інформації, властиві людському мозку. Розглянемо ті механізми та структуру біологічних нейронів. Нервову систему людини можна спрощено у вигляді триступінчастої структури. Центром цієї системи є мозок, що складається із мережі нейронів (рис. 1.1). Він отримує інформацію, аналізує її та видає відповідні рішення. Рецептори перетворюють сигнали з навколишнього середовища та внутрішніх органів на електричні імпульси, що сприймаються нейронною мережею (мозком). Рецептори забезпечують зв'язок нашого мозку із зовнішнім світом, реалізуючи надходження до нього зорової, слухової, смакової, нюхової інформації та інформації відчуттів. Ефектори перетворюють електричні імпульси, згенеровані мозком, у вихідні сигнали, що управляють м'язами, внутрішніми органами, стінками судин. Таким чином, мозок контролює роботу серця, дихання, кров'яний тиск, температуру, підтримує потрібний вміст кисню в крові і т.д. Проміжні нейрони обробляють інформацію, одержувану від сенсорних нейронів, і передають її ефекторним нейронам [14].

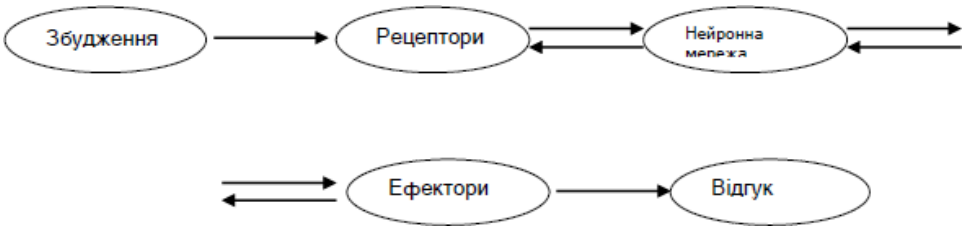


Рис. 1.1 Схема нервової системи

Слід зазначити, що мозок побудований з клітин двох типів: гліальних та нейронів. Зрозуміти роботу мозку можна при вивченні цих клітин і також нейронів, об'єднаних в єдину зв'язану мережу. Цей підхід і використовується при побудові штучних нейронних мереж [10].

Слід зазначити, що є інші думки. Деякі дослідники вважають, що головні процеси відбуваються не в нейронній мережі, а в самих клітинах, а саме в їхньому цитоскелетоні, в так званих мікротрубочках. Відповідно до цієї точки зору, і пам'ять, і навіть свідомість визначаються змінами білків у внутрішньоклітинних структурах та пов'язаними з ними квантовими ефектами.

У біологічному нейроні можна назвати такі структурні одиниці: тіло клітини; дендрити - множина коротких нервових волокон, що гілкуються, які збирають інформацію від інших нейронів; аксон єдине тонке довге нервове волокно, який забезпечує проведення імпульсу та передачу впливу на інші нейрони або м'язові волокна та на своєму закінченні також розгалужується та утворює контакти з дендритами інших нейронів; синапс – місце контакту нервових волокон, яке передає збудження від клітини до клітини.

Амплітуди сигналів, що передаються в нервовій системі, залишаються постійними і не залежать від важливості інформації, що передається, або від будь-яких інших причин. У той же час частота проходження імпульсів має пряме відношення до зовнішніх подій. Для кодування інформації в нейронах використовується частотний спосіб кодування. При цьому нервові імпульси поширюються групами.

Оскільки кількість нейронів та міжнейронних зв'язків величезна, помилка у спрацьовуванні окремого нейрона залишається непомітною у загальній масі взаємодіючих клітин. Нейронна мережа виявляє високу стійкість до перешкод, окремі збої несуттєво впливають на результати її функціонування. Така головна відмінність нейронних систем від звичайних електронних схем, створених людиною.

Поки що жодна сучасна технологія не дозволяє побудувати ІНС, близьку за масштабами до нейронної мережі мозку. Проте вивчення біологічних нейронних систем дозволяє сподіватися створення нового покоління електронних пристроїв з аналогічними характеристиками.

Інша важлива особливість нейронних систем – висока швидкість їх функціонування, незважаючи на відносно тривалий цикл спрацьовування

кожної окремої клітини, що вимірюється у мілісекундах. Вона досягається завдяки паралельній обробці інформації в мозку величезною кількістю нейронів. Такі операції, як розпізнавання образів і звуків або прийняття рішень, виконуються людським мозком за проміжки часу, що вимірюються сотнями мілісекунд. Це означає, що обчислення вимагають не більше 100 послідовних стадій. Іншими словами, для таких складних завдань мозок "запускає" паралельні програми, що містять близько 100 кроків. Ця властивість мозку відома як «правило ста кроків».

Розмірковуючи аналогічним чином, можна виявити, що кількість інформації, що надсилається за цей час від одного нейрона іншому, має бути дуже невеликою (кілька біт). Звідси випливає, що основна інформація не передається безпосередньо, а захоплюється та розподіляється у зв'язках між нейронами.

Якщо вдасться, взявши за зразок біологічну нейронну систему, створити пристрій з високим ступенем паралельності виконання окремих операцій, його швидкодія може бути істотно збільшена і наближена до рівня, що спостерігається в процесах обробки інформації біологічними системами.

Зупинившись на **нейрокомп’ютерній обробці інформації**, розглянемо пам’ять. Мозок використовує інший спосіб пошуку інформації: не за адресою, а за змістом, вірніше, за його досить представницькою частиною. Пам'ять, здатна відновлювати повну інформацію щодо її достатньої частини, називається змістовно-адресованою. При цьому мозок здатний отримувати інформацію і у випадку, якщо вихідні дані (ключ) не є її частиною, але пов'язані з нею стійким зв'язком. Така пам'ять називається в загальному випадку асоціативною.

Ще однією властивістю нашої пам'яті є її розподіленість. Це означає, що в мозку немає спеціалізованого нейрона, який відповідає за розпізнавання того чи іншого образу. Навпаки, у запам'ятовуванні деякої інформації бере участь безліч нейронів, тому руйнація деяких із них зазвичай не видаляє відповідний образ із пам'яті. Більше того, мозок має величезну компенсаторну здатність: ураження великих ділянок призводить до того, що відповідні функції беруть на

себе інші його частини. Така властивість системи називається робастністю (robust – міцний, здоровий). Навпаки, якщо зіпсувати кілька біт у комп'ютерній програмі, це призведе до катастрофічних нею наслідків. Отже, людська пам'ять відрізняється від комп'ютерної тим, що вона змістовно-адресована, асоціативна та робасна.

Розглядаючи процес мислення, потрібно сказати, що сучасні комп'ютери здатні вирішувати завдання високої обчислювальної складності: інтегрувати найскладніші системи диференціальних рівнянь, здійснювати логічний висновок, грати в шахи не гірше за людину. Однак усі ці завдання можуть бути вирішені шляхом формалізації – подання рішення у вигляді алгоритму – послідовності арифметичних та логічних операцій. Сучасний комп'ютер – це швидкодіючий арифмометр, здатний виконати будь-яку інструкцію, укладену в програмі. Структура комп'ютера є реалізацією універсальної машини Тьюринга, в якій структура повністю відокремлена та незалежна від даних, що обробляються.

Математичні обчислення та логічний висновок доступні і людському мозку. Реалізацію цих функцій зазвичай пов'язують з лівою його півкулею. Проте цим мислення не вичерпується. Робота лівої півкулі дозволяє нам говорити, будувати граматично правильні фрази, писати. Воно відповідальне за наше сприйняття часу та опрацьовує інформацію послідовно крок за кроком. Тим часом, математика, логіка та наука загалом є пізнішими досягненнями людського мислення, тобто розвиток обчислювальної техніки почався з імітації пізніших результатів еволюції людського мозку.

Пропущеним виявився цілий пласт його можливостей, реалізованих в правій півкулі. Багато досліджень свідчать на користь того, що права півкуля мозку відповідальна за наше сприйняття простору, за зміст слів, інтуїцію, образне мислення, вона обробляє інформацію паралельним способом. Однією з основних її особливостей є те, що вона працює не з абстрактними іменами об'єктів символами, а з образами конкретних об'єктів, інформацію про які мозок отримує із зовнішнього світу. Все бачене нами в житті зберігається у правій

півкулі, всі імена – у лівій. Права півкуля здатна вчитися впізнавати предмети за їх пред'явленням, а не за описом, у її способі функціонування дані та метод складають єдине ціле.

Здатність навчатися на прикладах пояснюється теорією нейронних мереж пластичністю синаптичних зв'язків – їх здатністю змінювати свою силу, налаштовуючись на вирішення певного завдання. Образне мислення, уява та інтуїція, які пов'язують із працею правої півкулі, дозволяють нам приймати рішення в тих випадках, коли ніякого рецепту не існує або він нам невідомий. Ці якості є більш давніми придбаннями мозку, ніж логічне мислення, і безпосередньо пов'язані з творчістю.

Нейрокомп'ютерний підхід до обробки інформації покликаний реалізувати можливості, закладені у правій півкулі мозку. Він повинен не замінити існуючі комп'ютерні методи, а лише заповнити можливості, для яких не вдається побудувати формальні алгоритмічні схеми. Подібно до того, як у людському мозку ліва та права півкулі працюють спільно, сучасні інформаційні системи повинні використовувати симбіоз традиційних алгоритмічних та нейрокомп'ютерних методів для повноцінної та продуктивної обробки інформації.

Особливості обробки інформації показано на наступній схемі:

Традиційна обробка інформації мації в ЕОМ (умовна ліва півкуля)	Нейрокомп'ютерна обробка інформації (умовна права півкуля)
Послідовна обробка	Паралельна обробка
Заданий алгоритм обробки	Алгоритм формується шляхом навчання на прикладах
Ієрархічна структура алгоритмів, розбиття складної задачі на прості	

Рис. 1.2 Нейрокомп’ютерна обробка інформації

1.3 Штучний нейрон та штучні нейронні мережі. Основні принципи побудови моделей

Штучний нейрон (або просто нейрон) є елементарним функціональним модулем, з яких будуються ШНМ. Він є модель біологічного нейрона тільки в сенсі здійснюваних ним перетворень, але не за способом функціонування.

Відомі логічні, безперервні та імпульсні моделі нейрона. Логічні моделі активно досліджувалися у 1960-70-х рр., але не набули подальшого розвитку. Імпульсні моделі більш близькі до фізичної природи процесів, що відбуваються в нервовій клітині, проте їхня теорія не так розвинена, як у безперервних, і вони все ще не знаходять широкого застосування.

Безперервна модель нейрона імітує в першому наближеннівластивості біологічного нейрона і включає набір синапсів або зв'язків, кожна з яких характеризується своєю вагою (*weight*).

Зокрема, сигнал *xi* на вході синапсу *i* множиться на вагу *wi*.

Позитивні значення ваг *wi* відповідають збуджуючим синапсам, негативні - гальмівним. Суматор обчислює зважену щодо відповідних синапсів суму вхідних сигналів нейрона.

Функція активації обмежує амплітуду вихідного сигналу. Ця функція також називається функцією стиснення.

Математичну модель формального нейрона можна представити рівнянням

$$p = \sum_{i=1}^{n} v_i \cdot x_i + v_o, \; y = f(p) \qquad (1.1)$$

де y - вихідний сигнал нейрона; $f(p)$ – функція активації нейрона; v_i - ваговий коефіцієнт синаптичного зв'язку i-го входу; x_i - i-й вхідний сигнал нейрона; v_0 - початковий стан (порушення) нейрона; i =1,2…n - номери входів нейрона; n-число входів.

Виразу (1.1) може бути поставлена у відповідність структурна схема формального нейрона, представлена на рис. 1.2.

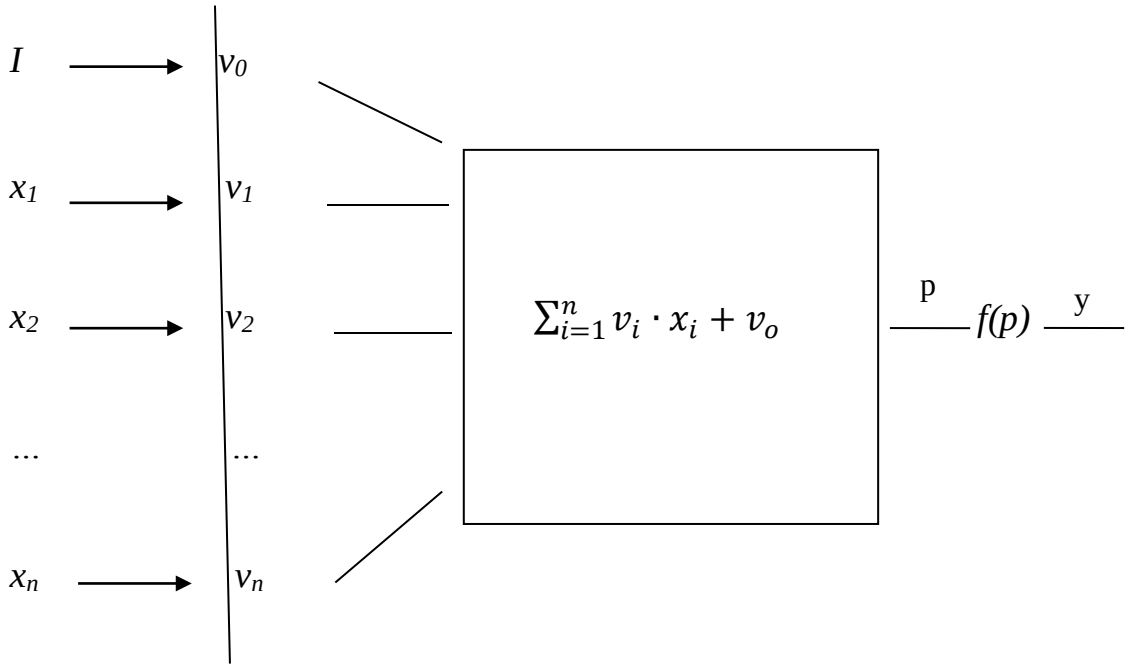


Рис. 1.3 Структурна схема формального нейрона

Робота штучного нейрона виконується наступним чином: перед початком роботи на блок суматора подають сигнал початкового стану (збудження) нейрона – v_0 ; на кожен *i*-й вхід нейрона надходять сигнали *xi* від інших нейронів або з пристрою введення вхідного образу; кожен *i*-й вхідний сигнал *xi* множиться на коефіцієнт синаптичного зв'язку v_i ; у блоці суматора зважені вхідні сигнали та початкове збудження v_0 алгебраїчно складаються.

Результат підсумовування (зважена сума) *p* подається на блок функціонального перетворення *f*(*p*). Нейрон, схему якого наведено на рис. 1.3 є класичним. Разом з тим, у деяких випадках використовуються й інші моделі формальних нейронів: нейрони з квадратичним суматором, нейрон з лічильником збігів та ін. Вибір моделі визначається насамперед характером завдання, що розв'язується.

Активаційна функція нейрона $f(p)$ визначає нелінійне перетворення, яке здійснюється нейроном. Існує безліч видів активаційних функцій, основні з них наведемо нижче.

Таблиця 1.1 Активаційні функції нейронів.

Табл. 1.1

Назва функції	Вигляд	Область значень
Лінійна	$f(x) = kx$	(-∞; +∞)
Напівлінійна	$f(x) = \begin{cases} kx, x \geq 0, \\ 0, x < 0. \end{cases}$	(0; +∞)
Логістична (сигмоїдна)	$f(x) = \frac{1}{1 + e^{-ax}}$	(0; 1)
Симетрична сигмоїдна	$f(x) = \frac{e^{ax} - e^{-ax}}{e^{ax} + e^{-ax}}$	(-1; 1)
Раціональна сигмоїдна	$f(x) = \frac{x}{a + \lvert x \rvert}$	(-1; 1)
Порогова	$f(x) = \begin{cases} 1, x \geq \partial, \\ 0, x < \partial. \end{cases}$	(0; 1)
Сигнатурна	$f(x) = \begin{cases} 1, x \geq \partial, \\ -1, x < \partial. \end{cases}$	(-1; 1)

Найбільш поширеними функціями активації є порогова, лінійна (у тому числі з насиченням) та сигмоїдні – логістична та симетрична (гіперболічний тангенс).

Порогова та сигнатурна (симетрична порогова) функції відносяться до класу дискретних. Основний недолік нейронів з пороговими функціями активації – відсутність достатньої гнучкості при навчанні та налаштуванні ШНМ на завдання, що вирішується. Якщо значення обчислюваного скалярного добутку p не досягає заданого порога, тоді вихідний сигнал не формується і нейрон “не спрацьовує”. Це означає, що втрачається інтенсивність вихідного сигналу даного нейрона i, таким чином, формується

невисоке значення рівня на зважених входах у наступному шарі нейронів. Тому порогові функції зазвичай використовуються для представлення нейромережами функцій логічного типу. Нейрони з пороговими функціями активації використовуються також у мережах, які навчаються методом конкуренції. Нейрон з найбільшим значенням скалярного добутку *p*, що обчислюється,

отримує право мати на виході значення *y*=1, що формується пороговою функцією.

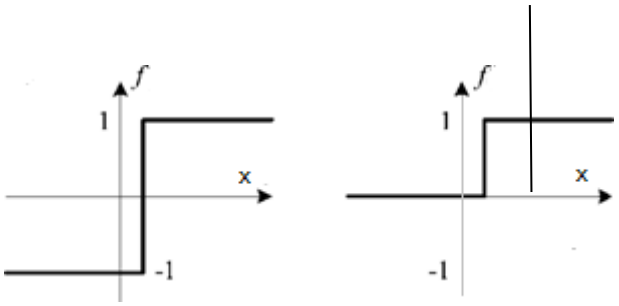
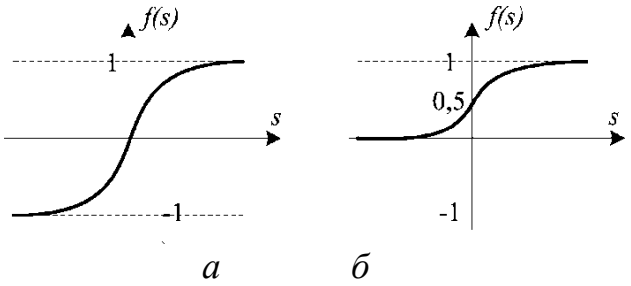


Рис. 1.4 Графіки сигнатурної та порогової функцій активації.

Сигмоїдні функції активації відносяться до безперервного класу (рис. 1.4). Вони задовольняють основним вимогам до безперервних активаційних функцій нейронів, що використовуються в навчальних ШНМ - безперервність, монотонне зростання і диференційність. Можна зазначити, що із зменшенням коефіцієнта нахилу *a* сигмоїди стають більш пологими, а при $a \to \infty$ перетворюються відповідно на порогову та сигнатурну функції.

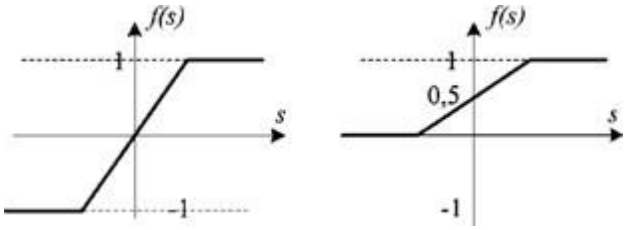


а *б*

Рис. 1.4 Графіки сигмоїдних функцій активації: *а* - симетрична, б - логістична

У числі їх переваг слід також згадати відносну простоту і безперервність похідних і властивість посилювати слабкі сигнали краще, ніж великі. Раціональна або «спрощена» сигмоїда становить інтерес через простоту її програмної реалізації. Нелінійність сигмоїдних функцій дозволяє їм виділяти у пошуковому просторі області складної форми, у тому числі неопуклі.

Лінійні функції активації також належать до класу безперервних (рис. 1.5). Лінійна ділянка такої функції дозволяє оперувати безперервними сигналами. Зони нечутливості визначаються фізичною реалізацією цих функцій.



а *б*

Рис. 1.5 Графіки лінійних функцій активації: *а* - лінійна,
б – напівлінійна.

Вибір функції активації визначається специфікою завдання, зручністю реалізації (на ЕОМ, у вигляді електричної схеми або іншим способом) та алгоритмом навчання: деякі алгоритми накладають обмеження на вигляд функції, які потрібно враховувати. Вдалий вибір функції активації може скоротити час навчання кілька разів.

Формальний нейрон фактично представляє собою процесор з дуже обмеженою спеціальною системою команд (нейромережевим базисом). За способом подання інформації формальні нейрони поділяються на аналогові та цифрові. І ті та інші виконують однакові обчислювальні дії та не вимагають зовнішнього управління. Велика кількість паралельно працюючих процесорів забезпечують високу швидкодію.

Розглянута проста модель штучного нейрона ігнорує багато властивостей свого біологічного двійника:

1. Обчислення виходу нейрона передбачаються миттєвими, не вносять затримки. Моделювати динамічні системи, що мають "внутрішній стан", за допомогою таких нейронів не можна.

2. У моделі відсутні нервові імпульси. Немає модуляції рівня сигналу щільністю імпульсів, як у нервовій системі. Не виявляються ефекти синхронізації, коли скупчення нейронівобробляють інформацію синхронно, під керуванням періодичних хвиль збудження-гальмування.

3. Немає чітких алгоритмів для вибору функції активації.

4. Немає механізмів, що регулюють роботу мережі в цілому (Приклад гормональне регулювання активності в біологічних нервовихмережах).

5. Надмірно формалізовані поняття: "поріг", "вагові коефіцієнти". У реальних нейронах немає числового порогу, він динамічно змінюється в залежності від активності нейрона і загального стану мережі.

6. Існує велика різноманітність біологічних синапсів. У різних частинах клітин вони виконують різні функції.

Гальмівні та збуджуючі синапси реалізуються в даній моделі у вигляді вагових коефіцієнтів протилежного знака, але різноманітність синапсів цим не обмежується. Дендро-дендритні, аксо-аксональні синапси не реалізуються в моделіформального нейрона.

Незважаючи на ці обмеження, мережі, побудовані з формальних нейронів, виявляють властивості, що сильнонагадують біологічну систему.

1.4 Схема абстрактного нейрокомп'ютера

Штучною нейронною мережею, або просто нейронною мережею, називається динамічна система, що складається з сукупності пов'язаних між собою формальних нейронів, здатна генерувати вихідну інформацію у

відповідь на вхідну дію. Зв'язок формальних нейронів здійснюється на кшталт вузлів направленого графа. Нейронна мережа є основною операційною частиною нейронних комп'ютерів (НК), що реалізуються апаратним чи програмним способом. Нейронна мережа реалізує алгоритм розв'язання поставленого завдання.

Структурну схему абстрактного ПК, побудованого з формальних нейронів, представлено на рисунку 1.6. Таку схему можна назвати узагальненою, оскільки вона пояснює принцип роботи будь-якого ПК незалежно від його конструктивного виконання. Ця схема нагадує класичну схему обчислювальної машини Дж. фон Неймана, представлену ним ще 1945 р. у звіті «Попередня доповідь про машину EDVAC». У ньому наводиться розподіл комп'ютера на алгоритмічно-логічний пристрій, управляючий пристрій і пам'ять, що стало згодом класичним. Тут же висловлено ідею програми, що зберігається в пам'яті. Однак ПК принципово відрізняється від цієї машини.

Рис. 1.6 Схема абстрактного ПК

Основним операційним блоком ПК, його процесором є ШНМ. Мережа є сукупність найпростіших модулів, званих формальними нейронами, з'єднаними каналами передачі. Кількісна характеристика кожного каналу визначається розв'язуваною задачею.

Нейронна мережа не робить обчислень, як це робить АЛП машини фон Неймана. Вона трансформує вхідний сигнал (вхідний образ) у вихідний відповідно до своєї топології та значення коефіцієнтів міжнейронного зв'язку. У запам'ятовуванні (ЗП) ПК зберігається не програма розв'язання задачі, як в ЕОМ, а програма зміни коефіцієнтів зв'язку між нейронами. Пристрої введення та виведення інформації в принципі виконують ті самі функції. Пристрій керування (ПУ) служить для синхронізації роботи всіх структурних блоків ПК під часвирішення конкретної задачі.

Для абстрактного ПК характерно наявність двох основних режимів роботи режиму навчання та режиму функціонування (вирішення задачі). Для того, щоб ПК вирішив необхідне завдання, його ШНМ має пройти через режим навчання, суть якого полягає в налаштуванні коефіцієнтів міжнейронних зв'язків на сукупність вхідних образів завдання. Налаштування коефіцієнтів здійснюється на прикладах, згрупованих у навчальні множини (вибірки).

При подачі на вхід ШНМ чергового еталонного образу вихідний сигнал відрізняється від бажаного. Блок навчання оцінює величину помилки та коригує коефіцієнти синаптичних зв'язків з метою її зменшення. При кожній подальшій подачі на вхід ШНМ цього еталонного вхідного образу помилка зменшується. Процес триває, поки помилка не досягне необхідного значення.

З математичної точки зору процес навчання є вирішенням задачі оптимізації, метою якої є мінімізація функції помилки на даному множині прикладів шляхом вибору коефіцієнтів синаптичних зв'язків.

У робочому режимі блок навчання, як правило, відключено і на вхід ШНМ подаються сигнали, що вимагають розпізнавання. Ці сигнали (вхідні образи), зазвичай, накладено шум. Навчена ШНМ фільтрує шум і відносить образ до потрібного класу.

Істотним недоліком ЕОМ є низька продуктивність,

обумовлена послідовним характером організації обчислювального процесу. З наявністю одного АЛП (процесора) пов'язаний інший недолік - низька ефективність використання пам'яті. Пам'ять ЕОМ можна уявити як довгу послідовність осередків. АЛП вибирає вміст однієї з них, дешифрує, виконує команду і при необхідності повертає результат у заздалегідь обумовлену комірку пам'яті. Потім звертається до чергової комірки для зчитування наступної команди, і процес повторюється до тих пір, поки не буде обрано останню команду програми, що виконується.

Неважко помітити, що переважна більшість осередків пам'яті не діє. Якщо ввести поняття коефіцієнта використання апаратури як відношення

$$\frac{\textit{число\_одночасно\_використовуваних\_елементів\_ЕОМ}}{\textit{загальне\_число\_елементів\_ЕОМ}},$$

то для ЕОМ цей коефіцієнт буде дуже низьким. ЕОМ і ПК розрізняються також за принципом взаємодії структури машини та задачі, що розв'язується. Вирішуючи завдання на ЕОМ, розробнику доводиться підлаштовувати алгоритм розв'язання задачі під жорстку структуру машини. При використанні ПК розробник, навпаки, підлаштовує структуру машини під завдання, що вирішується.

Висновки до розділу 1

Можна виділити три основні ознаки, покладені в основу класифікації ШНМ: архітектура міжнейронних зв'язків, тип навчання, клас розв'язуваних завдань. Архітектура нейронних мереж тісно пов'язана з алгоритмами навчання, що використовуються, і визначається постановкою завдання.

ШНМ може розглядатися як спрямований граф із зваженими зв'язками, в якому штучні нейрони є вузлами. За архітектурою зв'язків

ШНМ можуть бути згруповані у два класи: мережі прямого поширення, в яких графи не мають петель, та рекурентні мережі, або мережі зі зворотними зв'язками.

Нейронні мережі, які називаються персептронами (від лат. Perceptio - Сприйняття), представляють математичну модель процесу сприйняття образів. Ця модель реалізується у вигляді шарів нейронів: рецепторного шару та одного або декількох шарівнейронів, що перетворюють.

2 ОСНОВИ ТА ЗАГАЛЬНІ ПОНЯТТЯ ПРОЦЕСІВ ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ ВИЗНАЧЕННЯ, РОЗПІЗНАВАННЯ СКЛАДОВИХ ЧАСТИН АВТОТРАНСПОРТНИХ ЗАСОБІВ ТА ЧАСТИН ОБЛАДНАННЯ

2.1 Аналіз стану питання застосування згорткових нейронних мереж у мобільних пристроях

У сучасних реаліях, коли відбувається швидкий розвиток технологій, а автоматизація спрощує практично будь-який процес життя людини бурхливий розвиток теорій комп'ютерного зору призвів до появи технологій машинного зору. Машинний зір – це технології, які допомагають устаткуванню побачити процес виробництва чогось, проаналізувати дані та прийняти поінформоване рішення дуже швидко.

Згорткова нейронна мережа - це один з різновидів нейронних мереж прямого поширення, це коли змінні нейрони розбиті групи. Такі групи називають шарами. Коли ж така нейромережа використовується для застосування до даних, активація шарів (значення цих змінних) підраховується послідовно: спочатку значення активації першого шару, потім значення активації другого шару, і так до останнього шару.

Результати активації останнього шару є кінцевою точкою виходу результатів згорткової нейронної мережі [3].

2.1.1 Опис роботи загорткових нейромереж

Нейронна мережа являє собою математичну модель схожу за своїми якостями на мозок людини, а саме:

- можливістю обробляти інформацію, що надходить до нього з навколишнього середовища, яке згодом використовується для навчання;
- накопиченням знань шляхом налаштування синоптичних ваг на зв'язках між нейронами.

Приклад простої схеми нейронної мережі представлено на рисунку 2.1

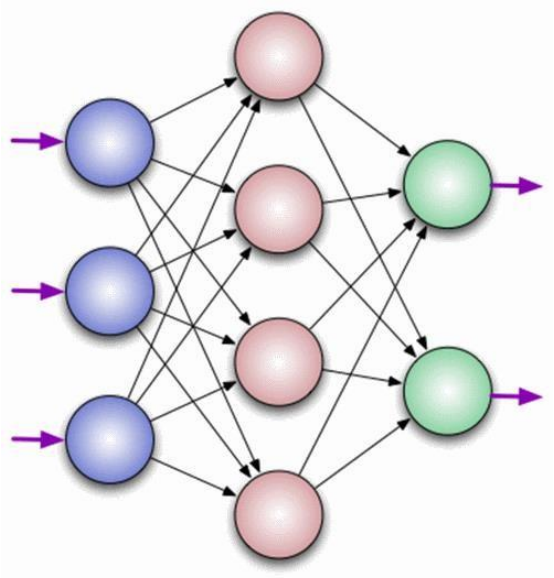


Рис.2.1 Приклад схеми простої нейронної мережі

Прийнято, що нейронні мережі складаються з трьох значних шарів:

- шар вхідних нейронів, позначений на схемі синіми нейронами;
- n-а кількість шарів прихованих нейронів, позначена на схемі шаром рожевих нейронів;

- вихідні нейрони, позначені на схемі зеленими нейронами.

Алгоритмом навчання називають процес, який використовується для навчання нейромережі. Робота даного алгоритму полягає у вибудовуванні синаптичних вагів нейронної мережі в певному порядку. Це необхідно для отримання необхідної структури взаємозв'язків нейронів.

Згорткові нейронні мережі ЗНМ (англ. convolutional neural network, CNN) є біологічно натхненими варіантами багатошарових персептронів (multilayer perceptron) [10]. З ранніх робіт Х'юбел і Візела з зорової кори тварини, було з'ясовано, що зорова кора містить складне розташування клітин. Дані клітини чутливі до невеликих підобластей поля зору, що називають рецептивним полем.

Субрегіони представлені сіткою, яка покриває поле зору. Дані осередки діють як локальні фільтри в просторі введення зорової інформації і добре підходять для використання сильної просторової локальної кореляції, яка є в природних зображеннях [12].

Якщо говорити більш детально, то згорткові нейронні мережі являють собою нейронну мережу, яка вміє визначати контури та кольорове представлення вхідного зображення. Відмінною рисою подібних мереж є процес згортки зображення. У ході якого ми “згортаємо” матрицю ознак вхідного зображення і отримуємо ще одну матрицю ознак, але вже в зменшеному варіанті [8].

Даний процес згортки проходить за участю ядра, що складається з матриці ваг, і вхідного зображення, яке представляє собою матрицю ознак, наприклад, яскравість пікселя в діапазоні від 0 до 255. Приклад процесу операції згортки представлений на рисунку 2.2. Ядро проходячи по заданій йому області прораховує вихідну матрицю шляхом добутку кожного параметра на вагу з подальшим підсумовуванням.

Кожен вихідний об'єкт буде виваженою сумою всіх об'єктів на вході. Згортка дозволяє виконати таку операцію лише з 9 параметрами, тому що кожен знак на виході виходить шляхом аналізу не кожного знака на

вході, а лише одного входу, розташованого «приблизно в одному місці» [15].

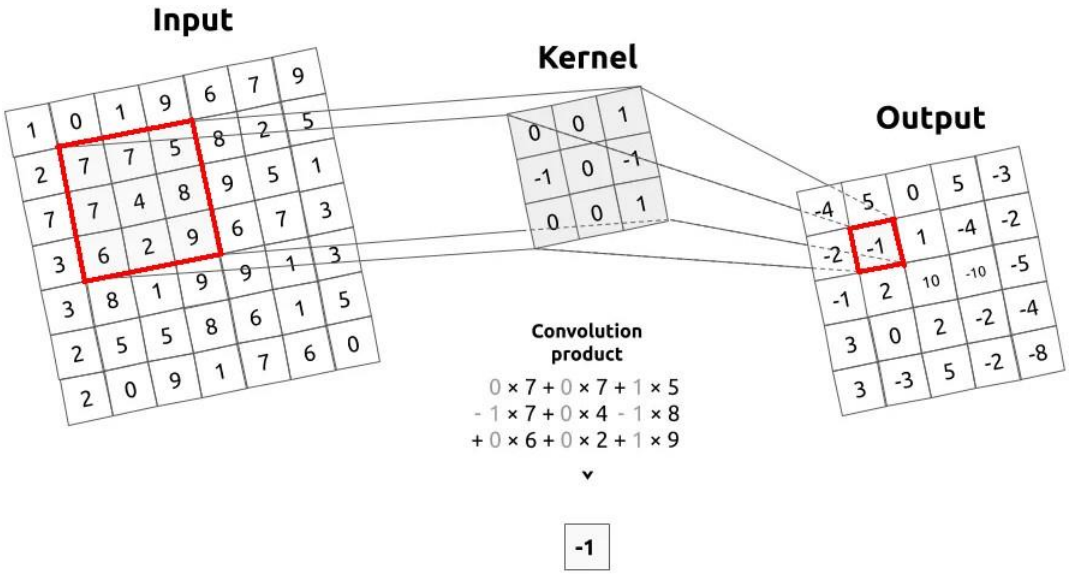


Рис. 2.2 Процес операції згортки

Наведене вище пояснення стосується лише одноканальних зображень. На практиці ми зазвичай маємо справу з багатоканальними зображеннями, а як приклад для подальшого пояснення візьмемо за основу триканальні, що складаються з червоного, зеленого і синього каналів, відомі як RGB зображення, так як вони є найбільш популярними.

Сам процес згортки будь-якого багатоканального зображення відрізняється не сильно і представлений на рисунку 2.3. Він полягає у тому, що для кожного каналу використовується своє окреме ядро зі своїми власними вагами нейронів [18]. При цьому кожне подібне ядро також може мати свою власну вагу, змінюючи значення цієї ваги можна регулювати, якому з каналів варто приділити більше уваги. Тобто чим вище вага конкретного ядра, припустимо червоного каналу, тим сильнішою буде реакція на відмінності в образах червоного. Після того, як пройде процес згортки кожного ядра, ми підсумовуємо отримані значення,

додаємо скалярне зміщення та отримуємо один загальний вихідний канал. Ця сукупність ядер всіх каналів та одного вихідного каналу називається фільтром. Процес згортки одного фільтра зображено на рисунку 2.3.

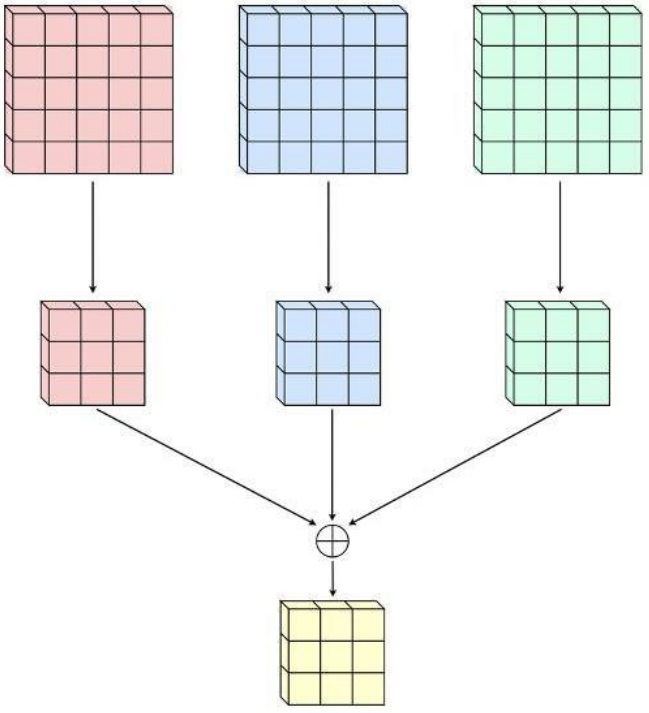


Рис. 2.3 Процес згортки одного фільтра

Результат для будь-якої кількості фільтрів ідентичний: кожен фільтр обробляє вхід з власним набором ядер та скалярним зміщенням відповідно до процесу, описаного вище, створюючи один вихідний канал. Потім вони об'єднуються, щоб отримати загальний вихідний сигнал з кількістю вихідних каналів, що дорівнює кількості фільтрів. В цьому випадку нелінійність зазвичай застосовується перед передачею введення на інший шар згортки, який потім повторює цей процес.

2.1.2 Огляд застосування алгоритмів згорткових нейронних мереж

До класичних завдань із застосуванням CNN можна віднести визначення границь, визначення об’єктів, сегментація, роспізнавання об'єктів. Визначення границь найнижчерівневе завдання серед перерахованих. Згорткові нейронні мережі, що виконують це завдання, застосовуються у множині різноманітних сфер застосування. Починаючи від дорожнього руху, де їх використовують для визначення автомобільних номерів із камер дорожнього спостереження, закінчуючи розпізнаванням усіх відомих captcha. Визначення вектора нормалі дозволяє реконструювати тривимірне зображення з двомірного [23]. Завдяки нейронним мережам, що виконують це завдання, отримують знімки нормалей і реконструюють об'ємну 3D модель будь чого, речі, об’єкта або поверхні на основі знімків. Для визначення об'єктів використовується нейронна мережа для локалізації об’єктів уваги, яка є загортковою мережею. Такі методи дозволяють ділити об'єкти на класи відповідно до їх структури, нічого не знаючи про ці об'єкти. Мабуть, одне з найголовніших завдань на сьогодні для нейронних мереж та комп'ютерного зору в цілому. Сфера застосування велика, тільки на сьогоднішній день дані згорткові нейронні мережі використовуються в автопілотованих транспортних засобах та при використанні віртуальної реальності. Як можна зрозуміти, були перераховані досить складні завдання, а це лише мала частина з тих, у яких може стати в нагоді сегментація.

Завдання розпізнавання (Detection) – найвищерівневе завдання з усіх перерахованих. Згорткові нейронні мережі, що виконують це завдання, знаходять та класифікують об'єкти на зображенні. За допомогою даних згорткових нейронних мереж можна визначати види та підвиди тварин на зображенні, розпізнавати людину по фотографії та таке інше.

2.1.3 Обмеження на умови роботи системи, технології

При визначенні основних принципів розробки систем визначення об’єктів потрібно зазначити, що такі системи створюються для використання фізичними особами та підприємствами для генерації, обробки та аналізу контенту.

Таким чином з’являється вимога до системи, що проектується, це можливість працювати на мобільних пристроях загального призначення, тобто, телефонах, планшетах та інших пристроях, оснащених відеокамерою та екраном з обчислювальними потужностями. Ця вимога, у зв'язку з технологічними особливостями мобільних пристроїв, накладає ряд обмежень на технології, що використовуються, а також архітектуру системи [14].

Навіть за умови використання сучасних техпроцесів при виробництві мобільних процесорів, отримати значні обчислювальні потужності на мобільному пристрої неможливо, тому що знижені тактові частоти обчислювальних ядер для запобігання перегріву [16]. Крім того, на відміну від комп'ютерів, у мобільних пристроях неможливо використовувати потужні графічні модулі, які дозволяють виконувати паралельні обчислення, тому доступними варіантами обробки графіки є компактні графічні співпроцесори з декілька обмеженими обчислювальними можливостями.

Третє обмеження мобільних платформ це час автономної роботи. Одночасно важлива як тривалість роботи без підключення до мережі, так і інтенсивність енергоспоживання. Якщо акумулятор розряджається занадто швидко, акумулятор нагрівається і існує ризик руйнування енергетичних елементів і навіть займання пристрою. Враховуючи цю особливість, наше програмне забезпечення не повинно бути вимогливим до ресурсів пристрою і повинно працювати стабільно.

2.2 Постановка задачі реалізації програмного забезпечення

Відштовхуючись від огляду класичних завдань, до виконання яких застосовуються згорткові нейронні мережі, можна дійти висновку, що застосування CNN не обмежується лише ними, і можна знайти ще спрсоби використання загорткових мереж у нестандартних сферах застосування.

Виходячи з вищеперерахованих аспектів, необхідно реалізувати програмне забезпечення, яке буде представляти мобільний додаток, який повинен буде взаємодіяти з моделлю машинного навчання, отримувати на вхід зображення, розпізнавати об'єкти REAL-time на зображенні, не бути вимогливою до ресурсів пристрою, працювати на платформі Android.

Таким чином, було описано поняття згорткової нейронної мережі та її відмінність від класичної нейронної мережі. Також були проаналізовані існуючі класичні завдання, що виконуються згортковими нейронними мережами. Для кожного завдання описано сферу застосування, для якої розробляються конкретні CNN. З отриманих даних було сформульовано та описано постановку задачі, виконання якої передбачає розробку основних принципів створення та роботи програмного забезпечення для виконання цієї задачі, в тому числі на мобільних пристроях, за допомогою нейромережевого моделювання.

2.3 Огляд програмних засобів для навчання нейронної мережі

1) TensorFlow – це бібліотека програмного забезпечення машинного навчання з відкритим вихідним кодом розроблена Google. Дана бібліотека дозволяє створювати та навчати нейронні мережі різної архітектури для виявлення та розпізнавання шаблонів та пошуку взаємозв'язків. TensorFlow

також включає TensorBoard, який є інструментом візуалізації на основі браузера для оцінки ефективності навчання та мережевих параметрів моделі. TensorFlow досягає своєї продуктивності шляхом розпаралелювання завдань між процесорами та графічними процесорами. Ядро кожної операції реалізовано на C++ з використанням бібліотек підвищення продуктивності [7].

Кожен розрахунок у TensorFlow представлений у вигляді потоку даних та його графу та у вигляді **графу обчислень - це моделі, які описують виконання розрахунків, план, порядок та закони цього виконання.** Важливо відзначити, що складання графу обчислень та виконання операцій у цій структурі – це два різні процеси. Граф складається зі змінних та операцій, він розраховує тензори - багатовимірні масиви, які можуть бути як числами, так і векторами.

Обчислення відповідно до графів виконується у сесіях, при цьому використовуються два типи сесій: регулярні та інтерактивні. Інтерактивний сеанс підходить для введення та виконання програмних команд у консолі, таким чином зберігається стан змінних та черг. Явне створення сеансів та графів забезпечує правильне звільнення ресурсів пам'яті [8].

У графі кожна вершина має 0 або більше входів та 0 або більше виходів, і є реалізацією операції. Тензори є ребра графа, це масиви довільного розміру (тип масиву вказується під час побудови графа).

У графі також можуть бути вершини певного призначення, залежности, що здійснюють управління, вони вказують, що вхідний вузол для певної залежності повинен закінчити виконання дій, відповідних до цього вузла, до того, як вузол отримувача контрольної залежності почне виконуватися. Кожна операція має назву і є абстрактним обчисленням (наприклад, підсумовування), також вони можуть мати атрибути. Ядро можна розглядати як специфічну реалізацію операції, яка може бути виконана на певному типі пристрою (центральний або графічний процесор). Змінна – особливий вид операції, що повертає покажчик на тензор, що постійно

змінюється: така змінна не зникає після одиничного використання графа. Вказівники на подібні тензори передаються численним операціям, які потім змінюють вказаний тензор.

У задачах машинного навчання параметри моделі зазвичай зберігають тензори в змінних, які оновлюються на кожному кроці навчання, при цьому унікальність Tensor Flow полягає у можливості проводити часткові підграфові обчислення. Ця особливість дозволяє зробити розбиття нейронної мережі, а отже, можна використовувати розподілене навчання [12]. TensorFlow має певні переваги:

Tensor board - візуалізація моделі та можливість досліджувати порядок обчислень у графі, можливість використовуватись як на мобільних, так і на найбільш потужних пристроях, у TensorFlow похідні задаються автоматично: цей процес називається автоматичним диференціюванням.

2) Scikit-learn – це бібліотека для машинного навчання для мови програмування Python Вона має різні алгоритми класифікації, регресії та кластеризації, випадкові ліси, підвищення градієнта, k-середнє, призначені для взаємодії з чисельними бібліотеками NumPy та SciPy. Розглянемо можливості цієї бібліотеки:

кластеризація (Clustering): для угруповання нерозмічених даних, наприклад, метод k-середніх (k-means);

перехресна перевірка (Cross Validation): для оцінки ефективності роботи моделі на незалежних даних;

набори даних (Datasets): для тестових наборів даних та для генерації наборів даних із певними властивостями для дослідження поведінкових властивостей моделі;

скорочення розмірності для зменшення кількості атрибутів для візуалізації та відбору ознак;

алгоритмічні композиції: для комбінування прогнозів кількох моделей;

вилучення ознак: визначення атрибутів у зображеннях та текстових даних;

відбір ознак: виявлення значимих атрибутів, на основі яких буде побудовано модель;

оптимізація параметрів алгоритму: отримання максимально ефективної віддачі від моделі;

множинне навчання: для нелінійного скорочення розмірності даних.

Величезний набір методів не обмежується узагальненими лінійними моделями, дискримінантним аналізом, наївним класифікатором Байєса, нейронними мережами, методом опорних векторів і деревами прийняття рішень тому застосовуються алгоритми навчання з вчителем. **TensorFlow оперує статичним обчислювальним графом. Тобто спочатку ми визначаємо граф, далі запускаємо обчислення і, якщо необхідно внести зміни до архітектури, знову навчаємо модель. Такий підхід обраний заради ефективності, але багато сучасних нейромережевих інструментів вміють враховувати уточнення в процесі навчання без істотної втрати швидкості навчання.**

3) PyTorch. На відміну від TensorFlow, бібліотека PyTorch працює з динамічно оновлюваним графом, тобто він дозволяє вносити зміни до архітектури в процесі [15].

У PyTorch можна використовувати стандартні налагоджувачі, наприклад, pdb або PyCharm. Процес навчання нейронної мережі простий і зрозумілий. У той же час PyTorch підтримує модель паралелізму даних та розподіленого навчання, а також містить безліч попередньо вивчених моделей. Але на відміну від TensorFlow, описане середовище глибокого навчання набагато менш гнучке у підтримці різних платформ. Також у PyTorch немає нативних інструментів для візуалізації даних. Тим не менш, існує сторонній аналог під назвою Tensorboard X.

4) Keras. Найбільш мінімалістичний підхід до використання TensorFlow дає високорівнева оболонка Keras, прототипування тут полегшено до краю. Створення масивних моделей глибокого навчання в

Keras зведено до однорядкових функцій. Але така стратегія робить Keras менш конфігурованим середовищем, ніж низькорівневі фреймворки.

5) Theano – це бібліотека Python для ефективної обробки математичних виразів за участю багатовимірних масивів (також відомих як тензори). Це загальний вибір для реалізації моделей нейронних мереж.

Деякі функції включають:

автоматичне диференціювання, яке потрібно лише реалізувати передову (прогнозуючу) частину моделі, і Theano автоматично з'ясує, як розрахувати градієнти у різних точках, що дозволить виконати градієнтний спуск для навчання моделі;

прозоре використання графічного процесора означає, що можна написати той самий код і запустити його або на CPU, або на GPU. Зокрема Theano з'ясує, які частини обчислень слід перенести на графічний процесор;

оптимізація швидкості та стабільності при цьому Theano внутрішньо реорганізує та оптимізує обчислення, щоб вони працювали швидше і були більш чисельно стійкі. Він також спробує скомпілювати деякі операції в код C, щоб прискорити обчислення.

Технічно Theano на справді не є бібліотекою машинного навчання, оскільки вона не надає готові моделі, які ви можете навчати на своєму наборі даних. Це математична бібліотека, яка надає інструменти для створення власних моделей машинного навчання.

У матеріалах конференцій зі штучного інтелекту та в конкурсах Kaggle дослідники нерідко віддають перевагу PyTorch. Пов'язано це з тим, що PyTorch набагато краще підходить для невеликих проектів та прототипування. Коли ж мова заходить про кросплатформні рішеннях, TensorFlow виглядає більш підходящим вибором.

2.4 Вибір платформи для розробки мобільного додатку

Наявність операційної системи є головною особливістю, яка відрізняє смартфон від звичайного мобільного телефону. У разі вибору конкретної моделі телефону або пристрою операційна система часто є визначальним фактором. Найбільш поширені операційні системи для смартфонів та мобільних платформ:

Android – портативна (мережева) операційна система для смартфонів, планшетних ПК, електронних книг, цифрових плеєрів, годинників, танетбуків на базі ядра Linux. Спочатку розроблена Android Inc., яку потім купив Google. Згодом Google ініціювала створення альянсу Open Handset Alliance (OHA), який зараз займається підтримкою та подальшим розвитком платформи. Android дозволяє створювати програмні додатки на основі Java, які керують пристроєм через розроблені Google бібліотеки. Android Native Development Kit дозволяє системі використовувати бібліотеки та компоненти додатків, написаних на інших мовах;

OCIOS – це мобільна операційна система, розроблена та виготовлена аамериканською компанією Apple. Вона була випущена у 2007 році; спочатку – для iPhone та Ipod Touch, а пізніше – для таких пристроїв, як Ipad та Apple TV. На відміну від Windows Phone і Google Android, доступна тільки для пристроїв Apple.

В даний час Android розвивається згідно геометричній прогресії: щорока кількість користувачів цієї операційної системи постійно зростає. Цей факт привертає увагу багатьох розробників створювати мобільні програми спеціально для Android. Можливо, на сьогоднішній день вона є найпопулярнішою та найцікавішою системою. Розробники дають користувачам унікальну можливість встановити набір вільного програмного забезпечення, можна створити програми для системи та продавати їх у спеціалізованому інтернет-магазині. Керуючись цим, було вирішено зупинити свій вибір на платформі Android.

2.5 Архітектура та опис принципу роботи нейронної мережі MobileNetV2

Скориставшись результатами наукових досліджень, наведеними у таблиці 2.1, у виборі моделі для навчання нейромережі, було прийнято рішення використовувати нейронну мережу MobileNetV2, оскільки вона має невелику вагу, що дозволяє зберігати навчену модель на мобільному пристрої. Навіть при невеликій вазі модель має дуже високу точність і час відгуку під час роботи на 4 ядрах мобільного пристрою [7].

При порівняльному аналізі нейронних мереж було здійснено наголос на пошук нейронної мережі, що підходить для використання на мобільному пристрої, а також підходить для роботи з бібліотекою TensorFlow. При порівнянні нейронних мереж, які використовуються для задач класифікації будемо мати наступні результати, які відображені у таблиці 2.1.

Таблиця 2.1 Порівняльний аналіз нейромереж для задач класифікації

Табл. 2.1

Нейронна мережа	Top 1 Accuracy	Params	MAdds	CPU
MobileNetV1	70.6	4.2 M	575 M	113 ms
ShuffleNet (x2)	73.7	5.4 M	524 M	-
NasNet-A	74.0	5.3 M	564 M	183 ms
MobileNetV2	74.7	6.9 M	585 M	143 ms

MobileNetV2 перевершує MobileNetV1 і ShuffleNet порівняно з розміром моделі та обчислювальними витратами. З множником ширини 1.4 MobileNetV2 перевершує ShuffleNet (×2) і NASNet, маючи меншу кількість часу на розпізнавання. У попередній версії нейронної мережі MobileNetV1 було представлено глибинну згортку, яка значно знижує розмір моделі

мережі, яка підходить для мобільних пристроїв або будь-яких пристроїв з низькою обчислювальною потужністю. У MobileNetV2 представлений новий покращений модуль з інвертованою залишковою структурою, причому усунуто нелінійності у шарах. Якщо казати про досягнення у задачах визначення об’єктів, а також у задачах семантичної сегментації, то **сучасні досягнення більш відображає ШНМ MobileNetV2, взятто її за основу для отримання ознак об’єктів.**

На рисунку 2.4 можна побачити структуру згорткових блоків MobileNetV2.

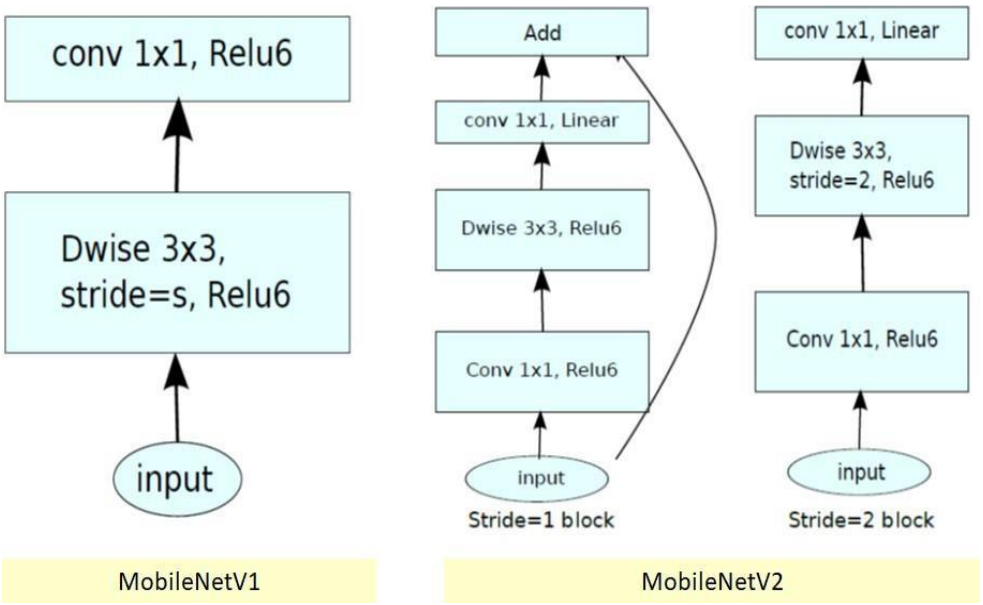


Рис. 2.4 Структура загорткових блоків MobileNetV2

MobileNet — це ефективна та переносна архітектура CNN, яка використовується в реальних додатках. MobileNets в основному використовує глибинно-роздільні згортки замість стандартних згортків, що використовуються в більш ранніх архітектурах, для побудови більш легких моделей. MobileNets представляє два нових глобальних гіперпараметри (множник ширини та множник роздільної здатності), які дозволяють розробникам моделей вибирати компроміс між затримкою або точністю та швидкістю та малим розміром залежно від їх вимог. Введено два параметри, щоб MobileNet можна було легко налаштувати: множник ширини α та множник роздільної здатності ρ .

Роздільна по глибині згортка - це згортка по глибині, за якою слідує згортка по точках таким чином:

- згортка по глибині – це просторова згортка з $D_k \times D_k$ каналами, Припустимо, що ми маємо 5 каналів, тоді у нас буде 5 просторових згорток $D_k \times D_k$ (a depthwise convolution);

- точкова згортка насправді є згорткою 1×1 для зміни розмірності (a pointwise convolution).

Нижче представлена архітектура MobileNet:

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Рис. 2.4 Архітектура мережі MobileNet.

У MobileNetV1 є шари двох типів

1. Перший рівень називається глибинним згортком, він виконує фільтрацію шляхом застосування одного згорткового фільтра на вхідний канал.

2. Другий рівень є згорткою 1×1, яка називається точковою згорткою, що відповідає за створення нових функцій за допомогою обчислення лінійних комбінацій вхідних каналів.

ReLU (Rectified Linear Unit) - це нелінійна функція активації, яка широко використовується в глибокому навчанні. Вона перетворює вхідне значення в значення від 0 до позитивної нескінченості, якщо вхідне значення менше або рівно нулю, то ReLU видає тільки 0, в протилежному випадку - вхідне значення.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-100, 100, 1000)
y = np.maximum(x, 0)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('ReLU(x)')
plt.title('ReLU')
plt.xlim(-5, 5)
plt.ylim(-0.5, 5)
plt.plot(x, y, linewidth=5)
plt.grid(True)
plt.show()
```

Математично ReLU визначається таким чином: ReLU(x) = max(0, x), де max - функція, що повертає максимальне значення з двох.

Графічно ReLU виглядає як лінійна функція з нульовим відсіком на осі абсцис у точці 0. Це означає, що функція має постійний нахил у всіх точках, крім точки 0, де відбувається відсічення. ReLU має кілька переваг у порівнянні з активацією сигмоїдної функції. По-перше, ReLU більш вимірно ефективна, оскільки вона є простою і швидкою операцією, яка не

вимагає обчислення експонент. По-друге, ReLU вирішує проблему затухання градієнта, так як вона не викликає затухання градієнта при зворотній формі помилки, як це відбувається у випадку активації сигмовидної функції. Однак ReLU має деякі недоліки. По-перше, при використанні ReLU, деякі нейрони можуть пропасти, тобто вони можуть отримати негативне значення і залишатися неактивними на всьому протязі навчання. По-друге, ReLU несимметрична відносно нуля, тому може виникнути проблема «розшарування» (кластеризації), коли нейрони можуть видавати тільки позитивні значення. Для вирішення цих проблем можуть бути використані інші функції активації.

Функція активації в нейронній мережі відіграє вирішальну роль у визначенні виходу вузла (вершини) шляхом обмеження амплітуди виходу. Ці функції також відомі як передатні функції. Функції активації допомагають нейронній мережі вивчати складні шаблони вхідних даних більш ефективніше, вимагаючи меншої кількості нейронів. Нейронні мережі, які використовують функції активації ReLU6 разом із пакетною нормалізацією партії перед кожним шаром, можна описати наступним чином.

Функція активації ReLU6 обмежує кількість одиниць до 6, що допомагає нейронній мережі швидко вивчати розріджені ознаки, а також запобігає безкінечному збільшенню градієнтів. Функція активації ReLU6 може бути математично визначена таким чином:

$$f(x) = \min(\max(0, x), 6) \tag{2.2}$$

ReLU6 використовується для надійності під час використання порівняно з низькою точністю обчислень на основі MobileNetV1. Слід зазначити, що Batch Normalization (BN) і ReLU застосовуються після кожної згортки.

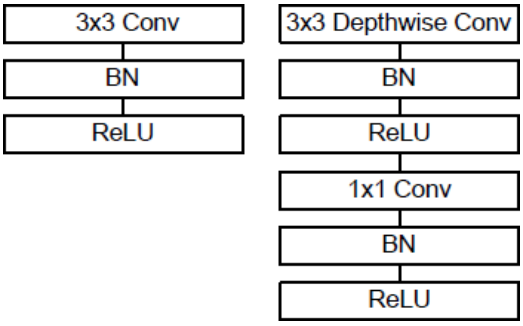


Рис. 2.5 Стандартна згортка (зліва), поглиблена згортка (справа) з BN і ReLU

Якщо порівняти стандартну згортку проти роздільної згортки по глибині для набору даних, то можна сказати, MobileNet втратив точність лише на 1%, але кількість параметрів та Multi-Adds значно зменшені.

В архітектуру MobileNetV1 було внесено кілька значних змін, що призвело до значного підвищення точності моделі. Основними змінами, внесеними до архітектури, було введення перевернутих залишкових блоків і лінійних вузьких місць і використання функції активації ReLU6.

У класифікаторі на основі MobileNet зазвичай в самому кінці знаходиться глобальний шар, що усереднює, шар об'єднання, за яким іде повністю зв'язаний шар класифікації або еквівалентна згортка 1×1, а також softmax. Насправді існує більше одного MobileNet. Він був розроблений як сімейство архітектур нейронних мереж. Існує кілька гіперпараметрів, які створюють різні компроміси архітектури. Найважливішим із цих гіперпараметрів є множник глибини, також відомий як «множник ширини». Він змінює кількість каналів у кожному шарі. Використання множника глибини 0,5 зменшить удвічі кількість каналів, що використовуються в кожному шарі, що скоротить кількість обчислень у 4 рази, а кількість параметрів, що вивчаються, — у 3 рази. Таким чином, він набагато швидше за повну модель, але також менш точний.

Завдяки інноваціям глибинно-роздільних згорток MobileNet доводиться виконувати приблизно в 9 разів менше роботи, ніж у порівняній нейронній мережі з тією ж точністю.

MobileNetV2, як і раніше, використовує глибинно-роздільні згортки, але її основний будівельний блок тепер виглядає так:

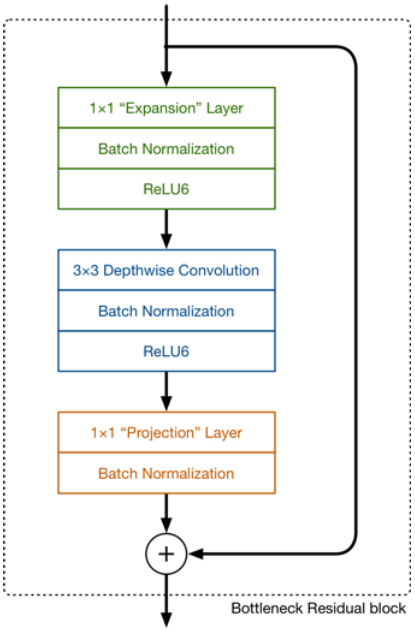


Рис. 2.7 Основна частина архітектури мережі MobileNetV2

Цього разу в блоці три згорткові шари. Останні два нам вже відомі: глибинна згортка, яка фільтрує вхідні дані, за якою слідує точковий згортковий шар 1×1. Однак тепер цей шар 1×1 виконує інше завдання.

У V1 точкова згортка або зберігала кількість каналів попереднім значенням, або подвоювала їх. У V2 вона робить навпаки: зменшує кількість каналів. Ось чому цей шар тепер називається проекційним шаром — він проектує дані з великою кількістю вимірів (каналів) у тензор із значно меншою кількістю вимірів [14].

Наприклад, шар глибинної згортки може працювати з тензором зі 144 каналами, який шар проекції потім скоротить до 24 каналів. Цей тип шару також називається шаром вузького місця, оскільки він зменшує обсяг

даних, що проходять через мережу. (Ось звідки і отримав свою назву «залишковий блок вузького місця»: вихід кожного блоку є вузьким місцем.)

Перший шар - новий у блоці. Це також згортка 1×1. Її мета – розширити кількість каналів у даних, перш ніж вони підуть у згортку по глибині. Отже, цей шар розширення має більше вихідних каналів, ніж вхідних — він значною мірою робить протилежне шару проекції. Точне значення того, наскільки розширюються дані визначається коефіцієнтом розширення. Це один із гіперпараметрів для експериментів із різними компромісами архітектури. Коефіцієнт розширення за замовчанням дорівнює 6.

Наприклад, якщо є тензор з 24 каналами, що входить до блоку, шар розширення спочатку перетворює його на новий тензор з 24 * 6 = 144 каналами. Потім згортка по глибині застосовує свої фільтри до цього тензора 144 каналів. І, нарешті, шар проекції проектує 144 відфільтровані канали назад на менше число, скажімо, 24 знову.

Розширення та проекція.

Таким чином, вхід і вихід блоку є тензорами низької розмірності, тоді як етап фільтрації, що відбувається всередині блоку, виконується на тензорі високої розмірності [14].

Друга нова річ у будівельному блоці MobileNetV2 – це залишкове з'єднання. Воно працює так само, як у ResNet, і існує для того, щоб допомогти з потоком градієнтів через мережу. (Залишкове з'єднання використовується тільки тоді, коли кількість каналів, що входять до блоку, дорівнює кількості каналів, що виходять з нього, що не завжди так, оскільки через кожні кілька блоків кількість вихідних каналів збільшується.)

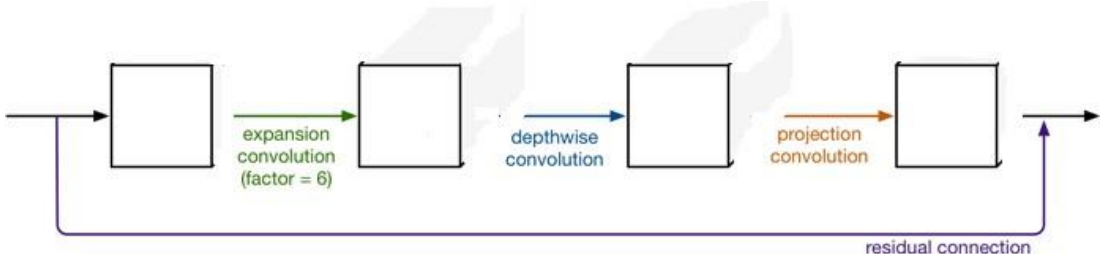


Рис. 2.8 Структура загорткового блоку мережі

Як завжди, кожен шар має пакетну нормалізацію, а функція активації – ReLU6. Проте вихід проекційного шару немає застосованої щодо нього функції активації. Оскільки цей шар виробляє низькорозмірні дані, то використання нелінійності після цього шару фактично знищує корисну інформацію. Попередньо навчені моделі використовують тільки пакетну нормалізацію після глибинного шару згортки, згортки 1×1 замість цього використовують зміщення [14].

Повна архітектура MobileNetV2, таким чином, складається з 17 таких будівельних блоків у ряд. Потім слідує звичайна згортка 1×1, глобальний середній шар об'єднання і шар класифікації. (Маленька деталь: перший блок трохи відрізняється, він використовує звичайну згортку 3×3 з 32 каналами замість шару розширення.)

Ідея V1 полягала в заміні великих за обчислювальними ресурсами згорток на менші, навіть якщо це означало використання більшої кількості шарів. Головні зміни в архітектурі V2 - це залишкові з'єднання та шари розширення/проекції. Якщо подивитися на дані, що проходять через мережу, можна помітити, що кількість каналів між блоками залишається невеликою.

Тензорні виміри між блоками

Як завжди для таких моделей, кількість каналів збільшується з часом і просторові вимірювання скорочуються вдвічі. Але в цілому тензори залишаються відносно невеликими завдяки шарам вузького місця, які складають з'єднання між блоками. У порівнянні з цим V1 дозволяє своїм

тензорам стати набагато більше (до 7×7×1024). Використання тензорів низької розмірності є ключем до скорочення кількості обчислень. Зрештою, що менше тензор, то менше множень повинні виконувати згорткові шари.

Але використання лише низькорозмірних тензорів не дуже добре працює. Застосування згорткового шару для фільтрації низькорозмірного тензора не дозволить отримати багато інформації. Тому для фільтрації даних ми в ідеалі хочемо працювати з великими тензорами. Блокова конструкція MobileNet V2 дає нам найкраще з обох сторін.

Потрібно вважати низькорозмірні дані, які передаються між блоками, як стислу версію реальних даних. Щоб застосувати фільтри до цих даних, потрібно спочатку їх розпакувати. Ось що відбувається всередині кожного блоку:

input tensor → EXPANSION LAYER → uncomress the data → DEPTHWISE LAYER → filter the data → PROJECTION LAYER → compress the data → output tensor

Декомпресія та компресія всередині залишкового блоку вузького місця. Шар розширення діє як декомпресор (подібно до unzip), який спочатку відновлює дані до їх повної форми, потім глибинний шар виконує всю необхідну фільтрацію на даному етапі мережі, і, нарешті, проекційний шар стискає дані, щоб знову зробити їх маленькими. Звичайно, трюк, який змушує все це працювати, полягає в тому, що розширення та проекції виконуються з використанням згорткових шарів з параметрами, що навчаються, і тому модель здатна навчитися найкращим чином стискати та дестискати дані на кожному етапі мережі.

Нижче наведено архітектуру MobileNetV2:

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Рис. 2.9 Архітектура MobileNetV2 ('c' - кількість вихідних каналів (або кількість фільтрів), 't' - коефіцієнт розширення, 'n' — кількість повторень блоку, 's' - це крок).

Таким чином, MobileNetV2 має два типи блоків. Один - залишковий блок з кроком, інший - блок з кроком 2 для зменшення.

Є 3 шари для обох типів блоків:

1. У новій версії перший рівень має згортку 1×1 з ReLU6.
2. Другий шар – це глибока згортка.
3. Третій шар - це ще одна згортка 1×1, але без будь-якої нелінійності.

MobileNetV2 перевершує MobileNetV1 і ShuffleNet з порівнянним розміром моделі та обчислювальними витратами.

Звичайна згортка є фільтром $D_k * D_k * C_{in}$, де D_k - розмір ядра згортки, C_{in} - кількість каналів на вході.

Блок MobileNet складається з трьох шарів:

1. Спочатку йде pointwise convolution з великою кількістю каналів, званий expansion layer. На вході цей шар приймає тензор розмірності

$$D_f * D_f * C_{in} \quad (2.3)$$

D_f - висота та шиина шару.

На виході цей шар видає наступний тензор

$$D_f * D_f * (t * C_{in}) \quad (2.4)$$

де t – новий гіперпараметр, названий рівнем розширення (expansion factor). Цей шар створює відображення вхідного тензору у просторі великої розмірності.

2. Потім йде depthwise convolution, глибинна згортка з ReLU6 активацією. Цей шар разом з попереднім, по суті, утворює вже знайомий нам будівельний блок MobileNetV1. На вході цей шар приймає тензор розмірності

$$D_f * D_f * (t * C_{in}) \quad (2.5)$$

На виході цей шар видає наступний тензор

$$(D_f/s) * (D_f/s) * (t * C_{in}) \quad (2.6)$$

де s - крок згортки (stride), адже, depthwise convolution не змінює число каналів.

3. Наприкінці йде 1x1 - згортка з лінійною функцією активації, знижувальна кількість каналів. Різноманіття високої розмірності, отримане після попередніх кроків, можна «вкласти» в підпростір меншої розмірності без втрати корисної інформації, що, власне, і робиться на цьому кроці. На вході такий шар приймає тензор розмірності

$$(D_f/s) * (D_f/s) * (t * C_{in}) \quad (2.7)$$

На виході цей шар видає наступний тензор

$$(D_f/s) * (D_f/s) * C_{out} \quad (2.8)$$

де C_{out} – кількість каналів на виході блоку.

Структуру шарів MobileNevV2 ви можете бачити на рисунку 2.10 [18].

Саме третій шар у цьому блоці, званий bottleneck layer, і визначає основну відмінність другого покоління MobileNet від першого.

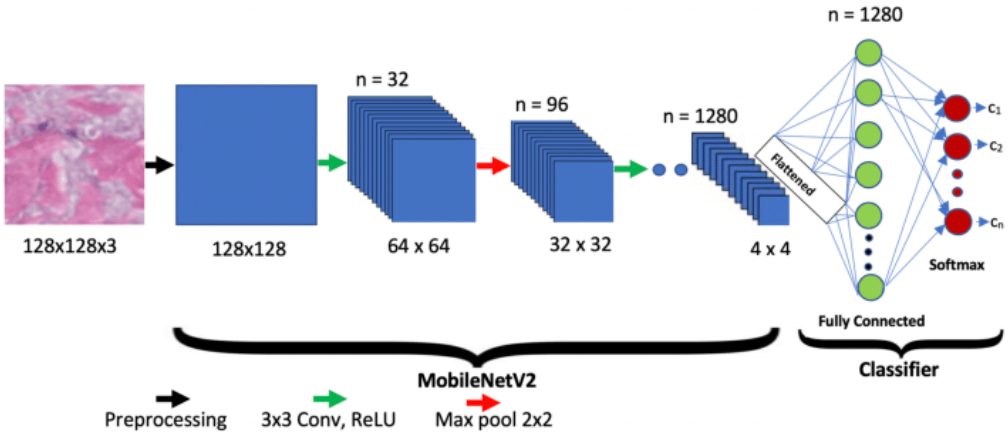


Рис. 2.10 Структура шарів MobileNetV2

Таким чином, у нашому додатку використовуватиметься модель нейромережі MobileNetV2.

Провівши огляд програмних засобів для навчання нейронної мережі, який було розглянуто вище, будемо обирати середовище TensorFlow для побудови та навчання нейронної мережі, Python і PyCharm у якості середовища розробки та мови програмування.

Висновки до розділу 2

В цьому розділі було зроблено постановку задачі реалізації програмного забезпечення для визначення, розпізнавання деталей складових частин автотранспортних засобів та частин обладнання у мобільних додатках для застосування на мобільних пристроях. У пунктах розділу описано роботу загорткових мереж за певними алгоритмами, описано архітектуру та принцип роботи нейронної мережі MobileNetV2, яку взято за основу, тобто нейронну мережу для навчання моделі задачі визначення та розпізнавання об’єктів. Проведено огляд програмних засобів для навчання нейронної мережі та вибору платформи для розробки мобільного додатку.

Також після аналізу та вибору програмного стеку для навчання нейронної мережі та розробки мобільного додатку визначено, що мобільний додаток буде розроблятися на платформі Android з використанням програмного забезпечення Android Studio, так як це найбільш популярне та відповідне програмне забезпечення для розробки мобільного додатка.

Отже, було розглянуто засоби розробки та навчання моделей машинного навчання. Вибір був зупинений на платформі TensorFlow, оскільки це найбільш підходящий інструмент для кроссплатформенної розробки. Було вивчено архітектуру нейронної мережі MobileNet, а також обрано програмний стек для реалізації поставленого завдання.

3 ОСНОВНІ ПРИНЦИПИ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ РОЗПІЗНАВАННЯ ДЕТАЛЕЙ АВТОМОБІЛЬНОГО ОБЛАДНАННЯ

3.1 Навчання нейронної мережі

Для навчання нейромережі розпізнаванню образів було вирішено скористатися сервісом Google Colaboratory та відкритою програмною бібліотекою для машинного навчання TensorFlow. Інтерактивне середовище блокноту Colab дозволяє писати та виконувати код Python у браузері. Google Colaboratory підтримує популярні бібліотеки TensorFlow і PyTorch, полегшує їх установку та впровадження. TensorFlow спрощує процес створення моделей нейронних мереж та їх навчання, при цьому основний інтерфейс програмного забезпечення API для роботи з бібліотекою реалізован для Python.

Метою виконання роботи є навчання нейромережі розпізнавати дрібні технічні деталі, автомобільні деталі, інструмент. Для початку збирається набір даних для навчання, для досліджень можна зупинитись на близько 200 фото деталей, ці зображення будуть представляти собою дані, що не пройшли підготовку, для тренування нейронної мережі. Також можна скористатися зібраними наборами даних, які зберігаються на сайті Kaggle. Kaggle - це найбільша у світі спільнота наукових співробітників із потужними інструментами та ресурсами для дослідження даних. При цьому API TensorFlow будемо використовувати для підготовки данних для навчання.

Для комфортної роботи нам знадобиться програмне забезпечення Anaconda. Напишемо скрипт для підготовки робочого оточення (рисунок 3.1):

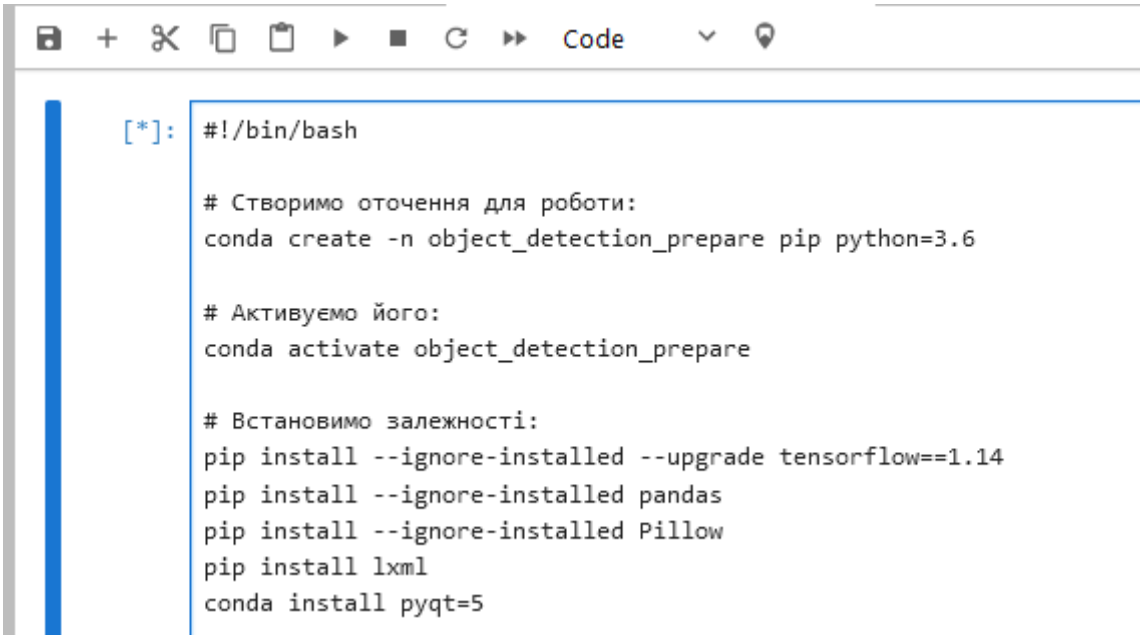


Рис. 3.1 Створення робочого оточення

При підготовці робочого оточення запущено програмне забезпечення Anaconda, активовано його та встановлено для нього певні залежності.

Тепер підготуємо зображення для навчання. Насамперед нам потрібно привести їх до однакового розміру. У Object API є інструмент для цього, ним і скористаємося. Скрипт для роботи з інструментом подано на рисунку 3.2.

Рис. 3.2 Зміна розміру зображень

Ми написали скрипт, який підготує зображення до обробки, змінить їх розмір та приведе розмір до значень 800 х 600. Після цього можна приступити до розмітки даних, анотування зображень, які підготовлені до обробки.

Анотація зображень є одним із найважливіших завдань у комп'ютерному зорі. За допомогою численних додатків та машинного навчання, комп'ютерний зір прагне дати програмам здатність бачити та інтерпретувати світ. Додатки на базі штучного інтелекту, такі як доповнена реальність, автоматичне розпізнавання мови та нейронний машинний переклад, можуть змінити життя людей та підприємства по всьому світу. Однак жодна з цих дивовижних технологій комп'ютерного зору була б неможлива без анотації зображення.

Анотація до зображень - це завдання людини, що полягає в тому, щоб позначити зображення ярликами. Залежно від проекту кількість міток на кожному зображенні може змінюватись. У деяких проектах потрібна лише одна мітка (класифікація зображень).

Для створення анотованих зображень нам буде потрібно: зображення, коментування зображень, платформа для анотування зображень. Більшість проектів анотацій зображень починаються зі створення та підготовки анотаторів для виконання завдань анотування. Більшість анотаторів не мають ступенів у машинному навчанні, тим не менш, ці коментатори повинні бути ретельно навчені специфікаціям та рекомендаціям кожного проекту, оскільки кожна компанія матиме свої вимоги. Після того, як анотатори навчаються, як анотувати дані, вони приступають до роботи, коментуючи сотні або тисячі зображень на платформі, призначеній для анотування зображень. Ця платформа є програмним забезпеченням, яке повинно мати всі необхідні інструменти для виконання певного типу анотації. Якщо ми розглянемо основні типи анотації зображень, то можна виділити наступні:

1. Обмежувальні рамки. Можна проводити анотацію у двовимірних прямокутниках, які будуть обмежувати об’єкт на зображенні, при цьому анотатори повинні малювати прямокутник навколо об'єкта, який вони хочуть анотувати всередині зображення. В загальних випадках може

бути кілька цільових об'єктів, в таких випадках після малювання рамки, анотатору потім доведеться вибирати зі списку міток.

2. Класифікація зображень.

У той час як рамки, що обмежують, мають справу з анотуванням декількох об'єктів на зображенні, класифікація зображень - це процес зв'язування всього зображення тільки з однією міткою. Простий приклад класифікації зображень - маркування видів, анотаторам будуть надані зображення і їх попросять класифікувати кожне зображення залежно від виду. Подання цих анотованих даних зображення у модель комп'ютерного зору навчить модель візуальним характеристикам, унікальним кожному виду. Теоретично модель зможе класифікувати нові анотовані зображення у відповідні категорії видів.

Спосіб анотування зображень визначає, як моделі працюватимуть після навчання. Погана інструкція часто відображається в навчанні і призводить до неточних прогнозів моделі. Анотовані дані особливо необхідні, якщо ми вирішуємо унікальну проблему та використовуємо ШІ в новій галузі. Для загальних завдань, таких як класифікація зображень та сегментація, часто доступні попередньо навчені моделі, які можна адаптувати до конкретних випадків, використовуючи мінімальні дані.

Навчання моделі з нуля зазвичай вимагає великої кількості анотованих даних, розділених на навчальний, валідаційний та тестовий набори, які створити важко та потребує багато часу.

Нам знадобиться інструмент для анотування зображень і досить якісні тренувальні дані. Вибір правильного інструмента анотування вимагає глибокого розуміння типу даних, які будуть розмічуватися, а також завдання. Особливу увагу слід приділити: модальності даних, типу необхідної інструкції, формату, в якому зберігатимуться інструкції. Для анотування часто використовуються різноманітні інструменти. Найвідоміші серед них - CVAT, Lebelme, LabelImg, Annotateme, кожен із них має свою специфіку.

LabelImg — інструмент для розмітки зображень з відкритим вихідним кодом, що має готові зібрані файли для Windows, завдяки чому його встановлення виконується надзвичайно швидко. Цей програмний додаток має безкоштовне завантаження та встановлення та виконує основні наступні функції: підтримує лише обмежуючі прямокутники (також є версія у форматі RotatedRect та оптимізована версія для однокласової розмітки), але щось складніше відсутнє, використовується формат PascalVoc XML, а файли анотацій зберігаються окремо для кожного зображення у вихідній папці. Утиліта LabelImg знаходиться у пакетному менеджері мови Python. На рисунку 3.3 показано приклад роботи з даною утилітою.

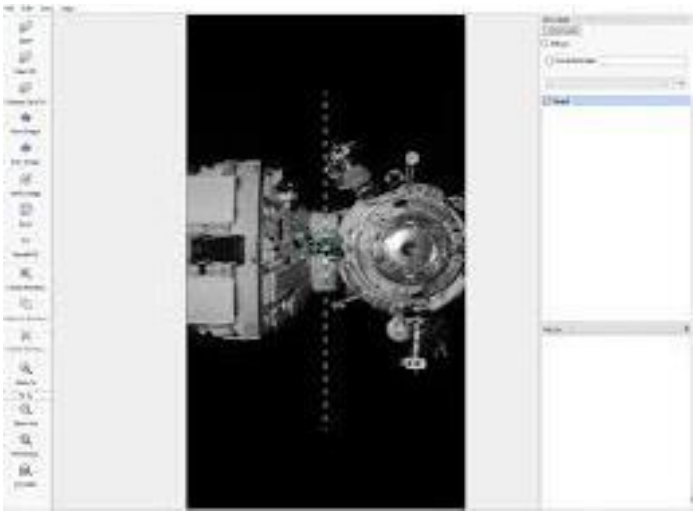


Рис. 3.3 Утиліта LabelImg

За допомогою даної утиліти виділяємо об'єкти для розпізнавання та вказуємо їх клас. У конкретному прикладі це комплектуючі, деталі автомобіля, дрібні механічні деталі та інструмент, далі потрібно зберегти метадані (файли *.xml) у тій самій папці.

Наступним кроком потрібно відсортувати зображення на тренувальні та тестові у наступному співвідношенні: 80/20, а потім перенести у папки train і test, які будуть використовуватися при навчанні.

Тепер потрібно створити папку annotations. У ній ми зберігатимемо файли з метаданими необхідними для навчання. Першим з них є файл з

мітками - label_map.pbtxt. У ньому ми вкажемо відношення класу об'єкта та цілого чисельного значення. Після цього для подальшої роботи необхідно конвертувати метадані у формат TFRecord. Для конвертації скористаємося наступними скриптами. На рисунку 3.4 зображено реалізацію методу конвертації Xml файлів у Csv.

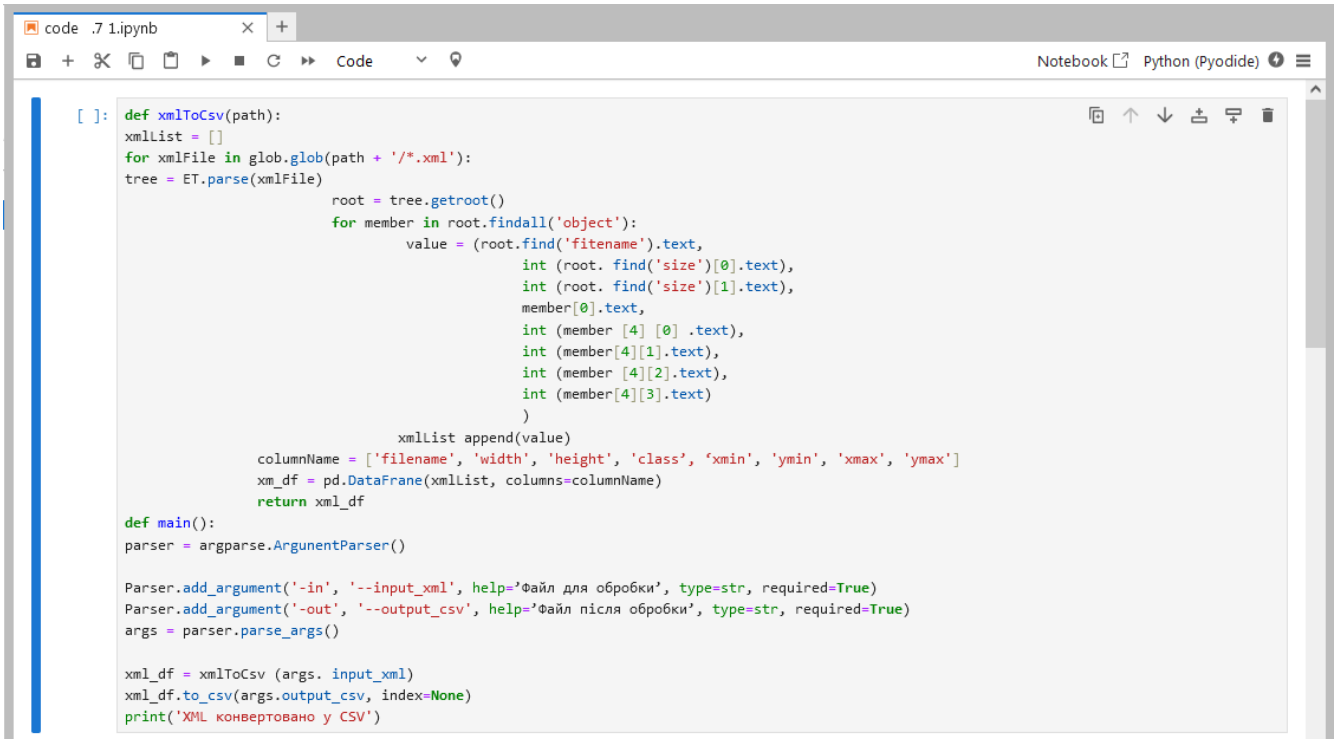


Рис. 3.4 Конвертація у формат CSV

Запустимо скрипт для підготовки наших даних. Скрипт для підготовки даних можна побачити рисунку 3.5:

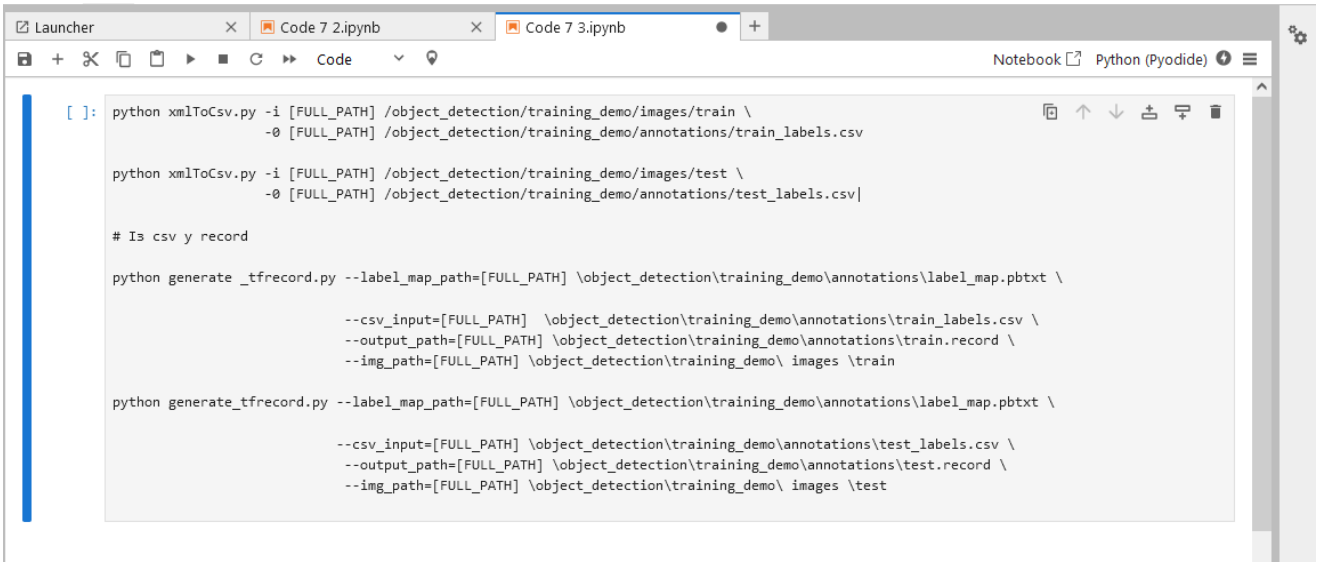


Рис. 3.5 Підготовка даних

На цьому закінчено підготовку даних, тепер налаштуємо модель, яку навчатимемо. Для цього трохи змінимо конфігурацію файлу моделі під наші потреби. Фрагмент конфігурації моделі представлений на рисунку 3.6

Рис. 3.6 Налаштування навчання

Після того, як підготовку даних закінчено, потрібно переходити до навчання нейронної мережі, запускаємо процес навчання, скрипт для запуску навчання представлений на рисунку 3.7.

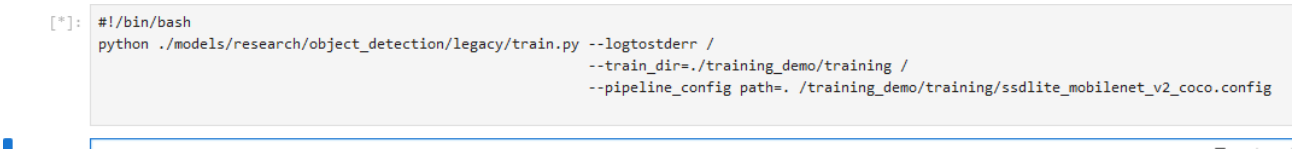


Рис. 3.7 Початок навчання нейронної мережі

Після навчання потрібно конвертувати отримані результати у frozen graph, який потім можна буде використовувати у мобільному додатку. Заморозка моделі означає створення окремого файлу, що містить інформацію про змінні графу і контрольні точки, збереження цих гіперпараметрів як констант у структурі графу. Це виключає додаткову інформацію, збережену у файлах контрольних точок, таку як градієнти в кожній точці, які включені, щоб модель можна було перезавантажити і продовжити навчання з того місця, на якому ви зупинилися. Оскільки це не потрібно для обслуговування моделі виключно для виведення, вони відкидаються при заморожуванні. Заморожена модель – це файл типу файлу Google .pb.

Заморожений граф у контексті TensorFlow відноситься до моделі, яка була повністю навчена і потім збережена у вигляді одного файлу, що містить як архітектуру моделі, так і навчені ваги. Цей заморожений граф потім може бути розгорнутий для виведення на різних платформах без необхідності вихідного визначення моделі або доступу до навчальних даних. Використання замороженого графа є важливим у виробничих середовищах, де основна увага приділяється прогнозуванню, а не навчанню моделі.

Однією з основних переваг використання замороженого графа є можливість оптимізувати модель для виведення. Під час навчання TensorFlow виконує ряд операцій, які не потрібні для виведення, наприклад, обчислення градієнта для зворотного поширення. Заморожування графа дозволяє видалити ці непотрібні операції, що призводить до більш

ефективної моделі, яка може прогнозувати швидше і з меншими обчислювальними ресурсами.

Крім того, заморожування графа також спрощує процес розгортання. Оскільки заморожений граф містить як архітектуру моделі, так і ваги в одному файлі, його набагато простіше розповсюджувати та використовувати на різних пристроях чи платформах. Це особливо важливо для розгортання серед обмежених ресурсів, таких як мобільні пристрої або периферійні пристрої, де обмежені пам'ять і обчислювальна потужність. Іншою ключовою перевагою використання замороженого графу є те, що він забезпечує узгодженість моделі. Після того, як модель навчена та заморожена, вона завжди видаватиме однаковий результат при однакових вхідних даних. Така відтворюваність має важливе значення для додатків, де узгодженість має вирішальне значення, наприклад, у охороні здоров'я чи фінансах.

Щоб заморозити граф TensorFlow, потрібно почати з навчання моделі з використанням TensorFlow API, що і було зроблено. Після завершення навчання та задоволення продуктивністю моделі вона зберігається як заморожений граф за допомогою функції `tf.train.write_graph()`. Ця функція бере граф обчислень моделі разом із навченими вагами та зберігає їх в одному файлі у форматі Protocol Buffers (файл `.pb`). Після заморожування графу можливо завантажити його назад у TensorFlow для виведення за допомогою класу `tf.GraphDef`. Це дозволяє подавати вхідні дані до моделі та отримувати прогнози без необхідності перенавчання моделі або доступу до вихідних навчальних даних. Використання замороженого графу TensorFlow має важливе значення для оптимізації моделей для виведення, спрощення розгортання, забезпечення узгодженості моделі та забезпечення відтворюваності на різних платформах та середовищах. Конвертація навченої моделі у заморожений граф надана на рисунку 3.8

```
#!/bin/bash

python /content/models/research/object_detection/export_inference_graph.py --input_type image_tensor \
                        --pipeline_config path /content/training_demo/training/ssdlite_mobilenet_v2_coco.config \
                        --trained_checkpoint_prefix /content/training_demo/training/model.ckpt-07070 \
                        --output_directory /content/training_demo/training/output_inference_graph_v1.pb
```

Рис. 3.8 Створення замороженого графу отриманої моделі

Для використання отриманих результатів у мобільному додатку потрібно конвертувати навчену модель у формат tflite, що необхідний для запуску моделі на мобільному пристрої. Спочатку конвертуємо отриманий результат навчання у frozen graph, який підтримує конвертацію у tflite. Потім вже конвертуємо в tflite. Ця конвертація представлена на рисунку 3.9.

```
#!/bin/bash

python /content/models/research/object_detection/export_tflite_ssd_graph.py --pipeline_config_path /content/training_demo/training/ssdlite_mobilenet_v2
--trained_checkpoint_prefix /content/training_demo/training/model.ckpt-07070 \
--output_directory /content/training_demo/training/output_inference_graph_tf_lite.pb
```

Рис. 3.9 Створення tflite моделі

Так як перший рядок скрипту не помістився на скрині, тому напишемо його: python /content/models/research/object_detection/export_tflite_ssd_graph.py

--pipeline_config_path /content/training_demo/training/ssdlite_mobilenet_v2_coco.config \

Щоб перетворити заморожений граф на модель TensorFlow Lite, необхідно виконати ряд кроків. TensorFlow Lite – це платформа, що дозволяє розгортати моделі машинного навчання на мобільних та вбудованих пристроях з акцентом на ефективність та виведення з малою затримкою. Перетворивши заморожений граф, який є серіалізованим графом TensorFlow, у модель TensorFlow Lite, ми зможемо скористатися

оптимізованим середовищем виконання та зменшеним обсягом пам'яті, що забезпечується TensorFlow Lite. Пояснення цього процесу:

- розуміння замороженого графа: заморожений граф — це граф TensorFlow, у якому змінні перетворюються на константи. Зазвичай він зберігається у двійковому форматі protobuf (pb). Перед перетворенням замороженого графа на модель TensorFlow Lite важливо мати чітке уявлення про структуру графа, включаючи вхідні та вихідні вузли;
- потрібно переконатися, що у системі розробки встановлено TensorFlow та TensorFlow Lite, їх можна встановити за допомогою pip та менеджера пакетів Python;
- далі завантажуємо заморожений граф. У Python можна використовувати TensorFlow API для завантаження замороженого графу, при цьому "frozen_graph.pb" є шлях до файлу замороженого графу;
- перетворення графа TensorFlow у формат TensorFlow Lite: TensorFlow надає можливості API Python для перетворення графа TensorFlow у формат TensorFlow Lite;
- зберігаємо модель TensorFlow Lite. Зрештою, можна зберегти модель TensorFlow Lite у файл.

Виконавши ці кроки, ми успішно перетворили заморожений граф на модель TensorFlow Lite. Отриману модель TensorFlow Lite можна використовувати для логічних висновків на мобільних та вбудованих пристроях, що дозволяє ефективно розгортати моделі машинного навчання.

Коли створюється мобільний додаток, тоді у ньому присутня папка Assets, в неї потрібно внести файл моделі training_demo/training/model_q.tflite, змінивши його ім’я на detect.tflite, і файл з мітками - labelmap.txt.

Потрібно сказати, що TensorFlow при навчанні нейронної мережі та при створенні моделі певного процесу збирає дані summary, де містяться

числові значення, зображення, текстові дані, гістограми розподілу значень та таке інше. Запис результату роботи моделі теж відбувається у файл, тому, навчивши мережу та зібравши всі потрібні результати, можна подивитись їх у браузері, запустивши інструмент TenzorBoard, запуск здійснюється з консолі:

tensorboard –logdir=’Log_Dir’.

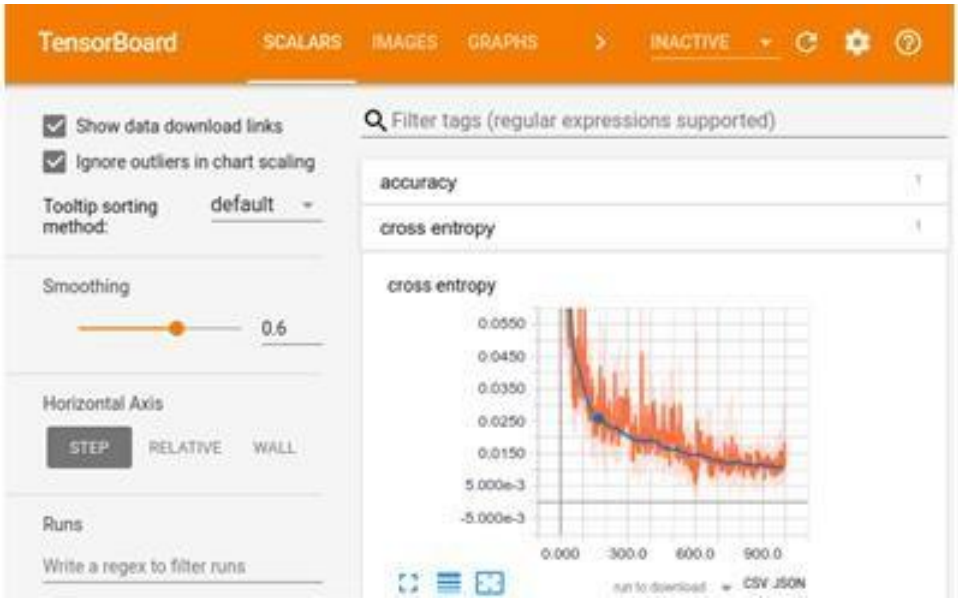


Рис. 3.10 Показ результатів навчання, параметрів моделі за допомогою інструмента візуалізації TensorBoard

У процесі тестування моделі було виявлено, що точність моделі машинного навчання становить близько 94 %, що відповідає поставленому завданню. Наступним кроком буде розробка програми використання моделі машинного навчання.

3.2 Основні принципи розробки та проектування мобільного додатку для розпізнавання деталей

Для розробки програми обирається платформа Android Studio та мова програмування Java. Розробники мобільних програм зазвичай взаємодіють з

типізованими об'єктами, такими як растрові зображення, або примітивами, такими як цілі числа. Інтерпретатор TensorFlow Lite, який запускає модель машинного навчання на пристрої, використовує тензори у формі ByteBuffer, які можуть бути важкими для налагодження. Бібліотека підтримки Android TensorFlow Lite розроблена для полегшення обробки введення та виведення моделей TensorFlow Lite та спрощення використання інтерпретатора TensorFlow Lite.

Перед переходом до розробки програми імпортуємо залежності збирача Gradle та інших налаштувань. Після того, як ми скопіювали файл моделі .tflite, нам потрібно вказати шлях у каталозі ресурсів до нашої моделі. Також вкажемо, що файл не повинен бути стиснутий, і додамо бібліотеку TensorFlow Lite у файл build.gradle модуля (рисунок 3.11).

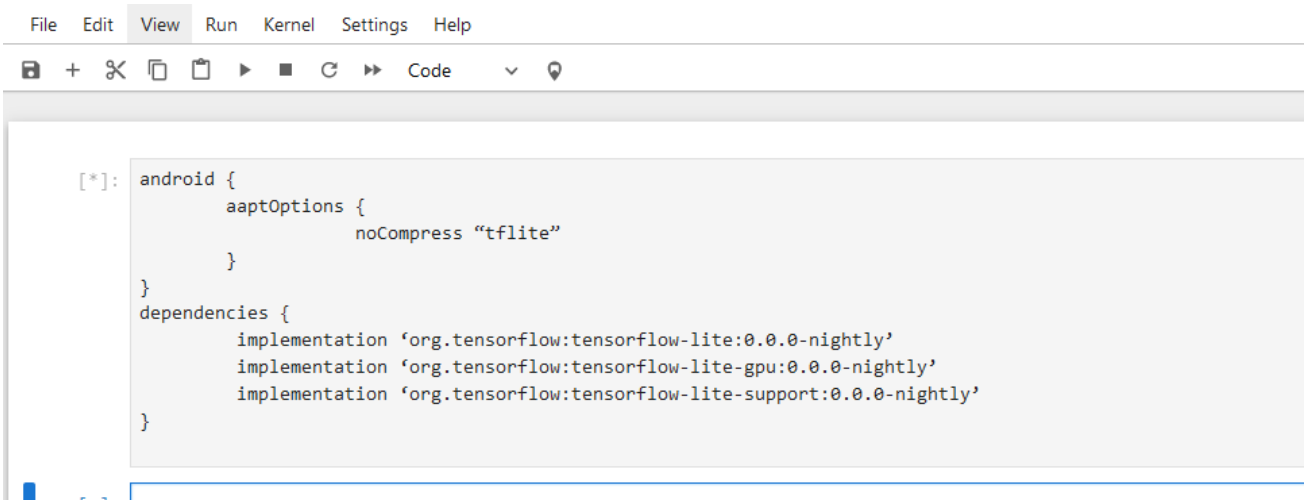


Рис. 3.11 Налаштування та імпорт залежностей

Спочатку нам потрібно отримати дані з камери. Наша мобільна програма повинна отримувати дані з камери, використовуючи функції, визначені у файлі CameraActivity.java (рис. 3.15). Цей файл залежить від AndroidManifest.xml для встановлення орієнтації камери, файл реалізовується у класі.

CameraActivity також містить код для захоплення налаштувань користувача з інтерфейсу користувача і надання їх іншим класам за допомогою зручних методів.

```
*]: model = Model.valueOf(modelSpinner.getSelectedItem().toString().toUpperCase());
    device = Device.value0f(deviceSpinner.getSelectedItem().toString());
    numThreads = Integer.parseInt(threadsTextView.getText().toString().trim());
```

Рис. 3.12 Клас CameraActivity

Основна частина логіки буде реалізована у класі Classifier. Файл Classifier.java містить більшу частину обробки введення з камери та виконання логічного виводу.

Існують два підкласи файлу, у ClassifierFloatMobileNet.java та ClassifierQuantizedMobileNet.java, щоб продемонструвати використання квантованих моделей. Після появи бібліотеки підтримки Android TensorFlow Lite ці підкласи переважно містять налаштування, а не логіку обробки. Клас Classifier реалізує статичний метод create, який використовується для створення екземпляра відповідного підкласу на основі наданого типу моделі (квантованої).

Для виконання висновку нам потрібно завантажити файл моделі та створити екземпляр інтерпретатора. Це відбувається у конструкторі класу Classifier (рисунок 3.13) разом із завантаженням списку міток класу. Інформація про тип пристрою та кількість потоків використовується для налаштування інтерпретатора за допомогою екземпляра Interpreter.Options, що передається до його конструктора.

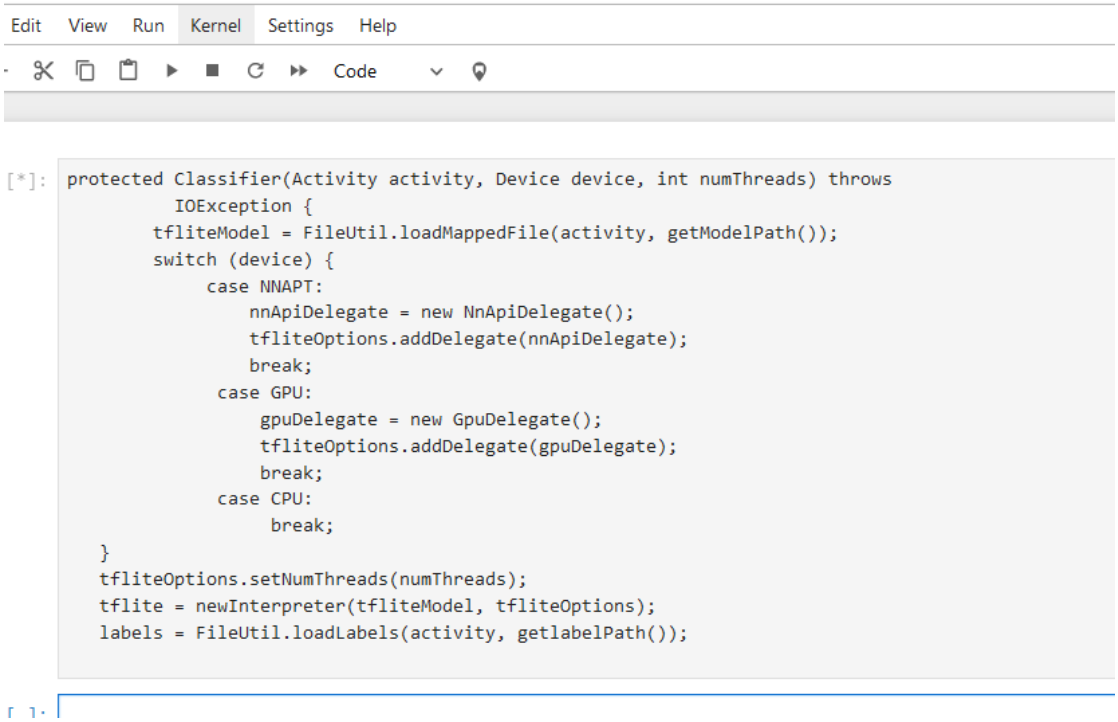


Рис. 3.13 Реалізація класу Classifier

Метод FileUtil.loadMappedFile виконує попереднє завантаження та зіставлення пам'яті файлу моделі, щоб прискорити завантаження, повертаючи MappedByteBuffer, що містить модель.

MappedByteBuffer передається до конструктора Interpreter разом з об'єктом Interpreter.Options. Цей об'єкт можна використовувати для налаштування інтерпретатора, наприклад, шляхом встановлення кількості потоків (.setNumThreads (1)) або включення NNAPI (.addDelegate (nnApiDelegate)).

На наступному кроці нам потрібно здійснити попередню обробку растрового зображення. Після того, як у конструкторі Classifier ми отримуємо растрове зображення вхідної камери, перетворимо його на формат TensorImage для ефективної обробки та попередньо обробляємо його. Запишемо кроки обробки у методі private loadImage (рисунок 3.14).

```
private TensorImage loadimage (final Bitmap bitmap, int
    sensorOrientation) {
      image.load(bitmap);

      int cropSize = Math.min(bitmap.getWidth(), bitmap.
          getHeight());
'     int numRoration = sensorOrientation / 90;
      ImageProcessor imageProcessor =
          new ImageProcessor.Builder()
              -add(new ResizeWithCropOrPadOp(cropSize, cropSize
                  ))
              -add(new ResizeOp(imageSizeX, imageSizeY,
                  ResizeMethod.BILINEAR))
              -add(new Rot90Op(numRoration))
              -add(getPreprocessNormalizeOp())
              -build();
          return imageProcessor.process(inputImageBuffer);
}
```

Рис. 3.14 Реалізація методу loadImage

Попередня обробка переважно однакова для квантованих і плаваючих моделей з одним винятком: нормалізація. Для ClassifierQuantizedMobileNet нормалізація не потрібна. Отже, параметри позначення плаваючої моделі визначаються наступним чином:

```
private static final float IMAGE_MEAN = 127.5f;
private static final float IMAGE_STD = 127.5f;
```

Рис. 3.15 Константи для плаваючої моделі

Тепер нам потрібно реалізувати виведення результатів нашої моделі, для цього ініціюємо метод виведення TensorBuffer для виведення моделі (рисунок 3.16):

```
private final TensorBuffer outputProbabilityBuffer;

int probabilityTensorIndex = 0;
int[] probabilityShape=
    tflite.getOutputTensor(probabilityTensorIndex).shape(); // {1,
        1001}
DataType probabilityDataType =
    tflite.getOutputTensor(probabilityTensorIndex).dataType();

outputProbabilityBuffer =
    TensorBuffer.createFixedSize(probabilityShape, probabilityDataType
        );

probabilityProcessor=
     new TensorProcessor.Builder().add(getPostprocessNormalizeOp
         ()).build();
```

Рис. 3.16 Здійснення виведення моделі

Висновок виконується з використанням наступного класу класифікатора. Замість безпосереднього виклику використовується метод Image. Він приймає растрове зображення та орієнтацію сенсора, виконує висновок та повертає відсортований список List of Recognition, кожен з яких відповідає мітці. Метод поверне кількість результатів, обмежених значенням MAX_RESULTS, яка за замовчуванням дорівнює 3. Розпізнавання – це простий клас, який містить інформацію про конкретний результат розпізнавання, включаючи його заголовок та достовірність. Використовуючи зазначений метод нормалізації після обробки, достовірність перетворюється на значення від 0 до 1 для даного класу, представленого зображенням. Запис даного методу та параметри його використання показані на рисунку 3.17.

```
Map<String, Float> labeledProbability =
        new TensorLabel(labels,
                probabilityProcessor.process (outputProbabilityBuffer
                    ))
                .getMapWithFloatValue();

private static List<Recognition> getTopKProbability(
            Map<String, Float> labelProb) {
      PriorityQueue<Recognition> pq =
             new PriorityQueue<>(
                    MAX_RESULTS,
                    new Comparator<Recognition>() {
                        @Override
                         public int compare(Recognition lhs,
                             Recognition rhs) {
                               return Float.compare(rhs.getConfidence
                                   (), lhs.getConfidence());
                         }
                        });

       for (Map.Entry<String, Float> entry : labelProb.entrySet()) {
          pq.add(new Recognition("" + entry.getKey(), entry.getKey(),
                                 entry.getValue(), null));
           }

   final Arraylist<Recognition> recognitions = new ArrayList<>();
   int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
   for (int i = 0; i < recognitionsSize; ++i) {
   recognitions.add(pq.poll());
   }
   return recognitions;
}
```

Рис. 3.17 Клас, який відповідає за розпізнавання

Після роботи класифікатора відображаються результати його роботи, причому результати виведення відображаються функцією processImage() у файлі ClassifierActivity.java. Метод ClassifierActivity реалізується як підклас CameraActivity, що містить реалізацію методів, які візуалізують зображення з камери, виконують класифікацію та відображають результати. Метод processImage() виконує швидку класифікацію у фоновому потоці, відображаючи інформацію в потоці інтерфейсу користувача, щоб уникнути

блокування і створення затримки. Представимо запис цього підкласу разом з методами на рисунку 3.18.

```
@Override
protected void processImage() {
      rgbFrameBitmap.setPixels(getRgbBytes(), 0, previewWidth, 0, 0, previewhidth,
          previewHeight) ;
      final int imageSizeX - classifier. getImageSizeX();
      final int imageSizeY - classifier. getImageSizeY();

      runInBackground(
          new Runnable() {
             @Override
             public void run() {
                if (classifier != null) {
                  final Long startTime = SystemCLock.uptimeMillis();
                  final List<Classifier.Recognition> results =
                        classifier. recognizeImage(rgbFrameBitmap, sensorOrientation);
                  lastProcessingTimeMs = SystemClock.uptimeMillis() - startTime;
                  LOGGER.v("Detect: %s", results);

                  runOnUiThread(
                     new Runnable() {
                       @Override
                       public void run() {
                          showResultsInBottomSheet (results)
                          showFrameInfo(previewWidth + "x" + previewHeight);
                          showCropInfo(imageSizeX + "x" + imageSizeY);
                          showCameraResolution(imageSizeX + "x" + imageSizeY);
                          showRotationInfo(String. valueOF(sensorOrientation));
                          showInference(lastProcessingTimeMs + "ms");
                       }
                       })
             }
             readyForNextImage();
          }
      });
}
```

Рис. 3.18 Робота класифікатора разом з методами для виведення результатів

Інша важлива роль ClassifierActivity полягає у визначенні користувацьких уподобань та створенні екземпляра відповідним чином налаштованого підкласу Classifier. Це відбувається, коли починається подача відео (через onPreviewSizeChosen()) і коли параметри змінюються в

інтерфейсі користувача. Далі ми представимо створення класифікатора (рисунок 3.19):

```
private void recreateClassifier(Model model, Device device, int numThreads) {
 if (classifier != null) {
  LOGGER.d("Closing classifier.");
  classifier. close();
  classifier = null;
 }
  if (device == Device.GPU && model==Model.QUANTIZED) {
   LOGGER.d("Not creating classifier: GPU doesn't support quantized models.");
   runOnUiThread(
       () -> {
         Toast.makeText(this, "GPU does not yet supported quantized models.",
             Toast. LENGTH_LONG)
             .show();
       });
    return;
  }
  try {
   LOGGER.d(
       "Creating classifier (model=%s, device=%s, numThreads=%d)", model,
        device, numThreads);
   classifier = Classifier.create(this, model, device, numThreads);
  } catch (IOException e) {
   LOGGER.e(e, "Failed to create classifier.");
  }
}
```

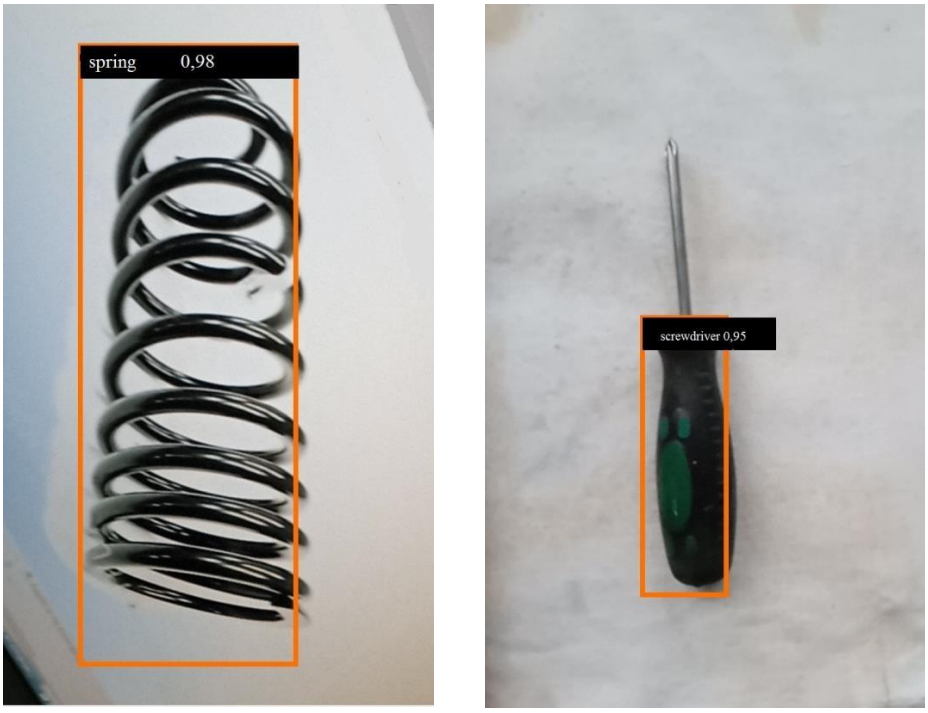
Рис. 3.19 Визначення користувацьких уподобань при створенні екземпляра класу

Після виконання всіх кроків та реалізації всіх класів ми отримали програмний код для розробки мобільного додатку, який може запускатися на мобільних пристроях під керуванням операційної системи Android та розпізнавати деталі, наприклад автомобіля та інструмент. Тепер проведемо тестування та визначимо, наскільки якісно було проведено навчання моделі із застосуванням штучних нейронних мереж та розробка програми для використання при створенні мобільного додатку.

3.3 Робота мобільного додатку визначення деталей

Після навчання моделі та реалізації основних принципів побудови програмних мобільних додатків ми будемо мати виконання процесу виділення леталей автомобільного обладнання та інструменту при роботі з ними.

Скріншот виконання розпізнавання деталей мобільним додатком, розробленим за основними принципами побудови та з використанням навчених загорткових нейронних мереж, буде виглядати наступним чином (рисунок 3.20):



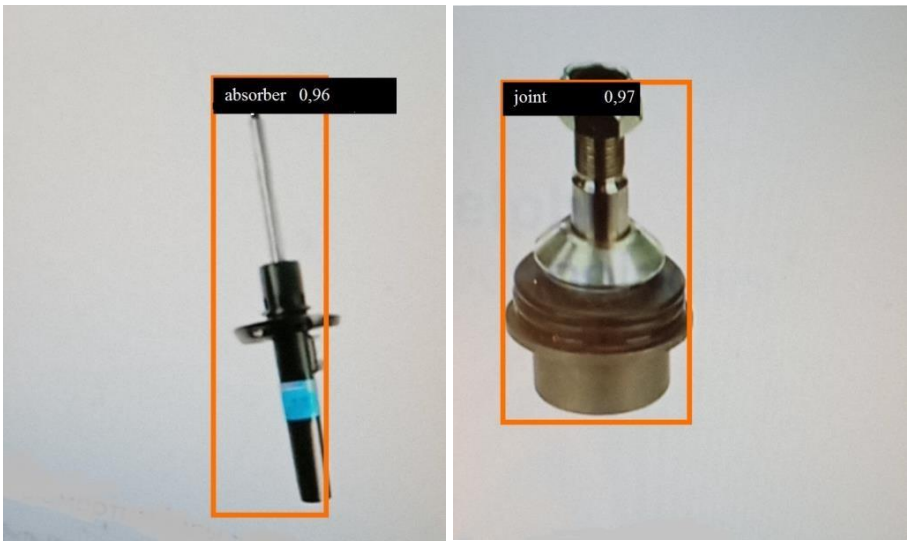


Рис. 3.20 Скріншот тестування програми

Як можна бачити на даному скріншоті, програма змогла з високою точністю визначити, що знаходиться на зображенні. Таким чином, можна зробити висновок, що програма була реалізована успішно і задовольняє вимогам. Ця програма показує як з використанням нейронних мереж можна створювати програми, які будуть реалізовані та працювати у мобільних додатках.

Висновки до розділу 3

Мета роботи визначила основний напрямок досліджень у розділі 3 - навчання моделі розпізнаванню деталей обладнання, а також опис основних принципів розробки програми для комфортного використання навченої моделі.

Для цього були досліджені моделі машинного навчання, які задовольняли б нашим вимогам, а точніше: мати невелику вагу. У процесі досліджень було виявлено, що на даний момент найбільш ефективно підходить нейронна мережа MobileNetV2, оскільки розпізнавання відбувається навіть з 5 FPS, а вага складає всього пару мегабайт, що дозволяє зберігати навчену модель у мобільному пристрої.

Навчання проводилося з використанням мови Python та платформи TensorFlow. Для тренування було підготовлено невеликий набір даних (dataset), якій містить зображення деталей. Після навчання отриману модель потрібно було конвертувати у формат tflite для використання моделі в мобільному додатку. Було визначено основні принципи розробки програми на мові Java. Завдяки добре задокументованому API від TensorFlow, програма вийшла простою в реалізації, а також без спеціальних технічних параметрів, необхідних для запуску. Вона складається з кількох класів, які відповідають за зчитування зображення з камери, передачу зображення у класифікатор і виведення класифікатором результату. Таким чином, надано основні принципи проектування програмного додатку для мобільних пристроїв.

4 ПРОГНОЗУВАННЯ РЯДІВ ЗНАЧЕНЬ ХАРАКТЕРИСТИК ТРАНСПОРТНОГО ПОТОКУ ЗА ДОПОМОГОЮ DEEP NETWORK DESIGNER MATLAB

Дорожній трафік можна розглядати як потік автомашин, автотранспортних засобів, що приймають участь у дорожньому русі та заїджають у транспортну мережу, що розглядається, і рухаються ділянками транспортної мережі. Потоком подій називається послідовність подій, які відбуваються одна за одною у випадкові моменти часу. Потік подій можна зобразити у вигляді послідовності випадкових точок $t_1, t_2, \dots, t_n, \dots$ на віссі *Ot*, де n – кількість подій у потоці, а точки відображають моменти часу настання подій, в наших дослідженнях це моменти часу заїзду автомобілів на ділянки доріг, що входять у транспортну мережу, що розглядається. Випадкові інтервали часу між цими точками визначаються за формулами: $T_1 = t_2 - t_1, T_2 = t_3 - t_2, \dots, T_{n-1} = t_n - t_{n-1}, \dots$.

Нехай випадкова величина $X(t,\ \Delta t)$ - число подій, що з’явилися на проміжку $(t, t + \Delta t)$, у якості подій розглядається заїзд автомобіля у

транспортну мережу, а для чисельного відображення фіксується момент часу цього заїзду. Якщо обчислювати границю відношення математичного очікування цієї випадкової величини до довжини інтервалу часу, що розглядається, при прямуванні цієї довжини до нуля і якщо ця границя існує, то вона називається інтенсивністю ординарного потоку подій у момент t, в наших дослідженнях це інтенсивність потоку автомобілей у транспортній мережі:

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{M(X(t,\ \Delta t))}{\Delta t} \tag{4.1}$$

Таким чином, інтенсивністю потоку називається середнє число подій, які відбуваються за одиницю часу. Будемо досліджувати часовий ряд значень інтенсивності дорожнього руху у певні моменту часу.

В цьому розділі показано, як створити просту мережу з довгою короткостроковою пам'яттю (LSTM) для прогнозування даних часових рядів за допомогою Deep Network Designer.

Довга короткострокова пам'ять (Long short-term memory; LSTM) – особливий різновид архітектури рекурентних нейронних мереж, здатна до навчання довгострокових залежностей. Вони були представлені Зеппом Хохрайтером та Юргеном Шмідхубером (Jürgen Schmidhuber) у 1997 році, а потім удосконалені та популярно викладені в роботах багатьох інших дослідників [22]. Вони чудово вирішують цілу низку різноманітних завдань і нині широко використовуються.

Мережа LSTM - це рекурентна нейронна мережа (RNN), яка обробляє вхідні дані, проходячи цикл за часовими кроками та оновлюючи стан RNN. Стан RNN містить інформацію, що запам'яталася за всі попередні інтервали часу. Це мережі, що містять зворотні зв'язки та дозволяють зберігати інформацію. Можна використовувати нейронну мережу LSTM для прогнозування наступних значень часового ряду або послідовності значень,

використовуючи значення на попередніх часових інтервалах або у попередні моменти часу як вхідні дані. Саме так ми будемо прогнозувати значення моментів часу появи автотранспортних засобів у досліджуваній транспортній мережі, що свідчить про інтенсивність транспортного потоку дорожнього руху.

Щоб створити мережу LSTM для прогнозування часових рядів, будемо використовувати пакет, вбудований у систему комп'ютерної математики Matlab – toolbox Deep Network Designer.

Завантаження даних з нобору Waveform Data. Щоб отримати доступ до цих даних, відкриємо приклад як живий скрипт. Набір даних Waveform містить синтетично згенеровані сигнали різної довжини із трьома каналами. У цьому прикладі нейронна мережа LSTM навчається прогнозуванню майбутніх значень з урахуванням значень попередніх часових інтервалів або моментів часу. Також переглянемо розміри перших кількох послідовностей.

Синтетично згенеровані значення, які представлені у наборі даних візьмемо за значення інтенсивності потоку автотранспортних засобів на мережі доровнього руху у певні моменти часу і будемо використовувати для побудови нейронної мережі у Matlab для прогнозування значень інтенсивності у наступні моменти часу, які слідують за тими, що розглядаються. На практиці можна вимірювати значення інтенсивності транспортного потоку, а потім робити прогнозування наступних значень за допомогою нейронної мережі.

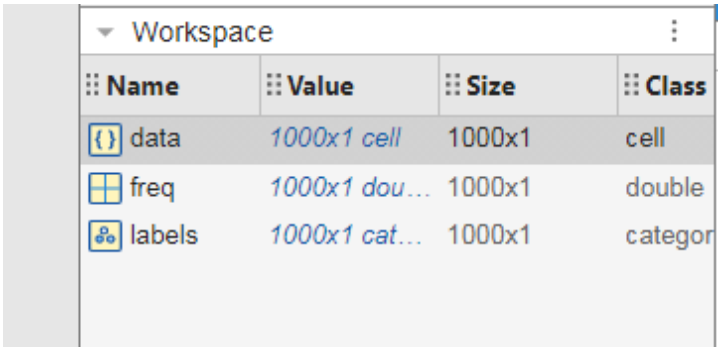
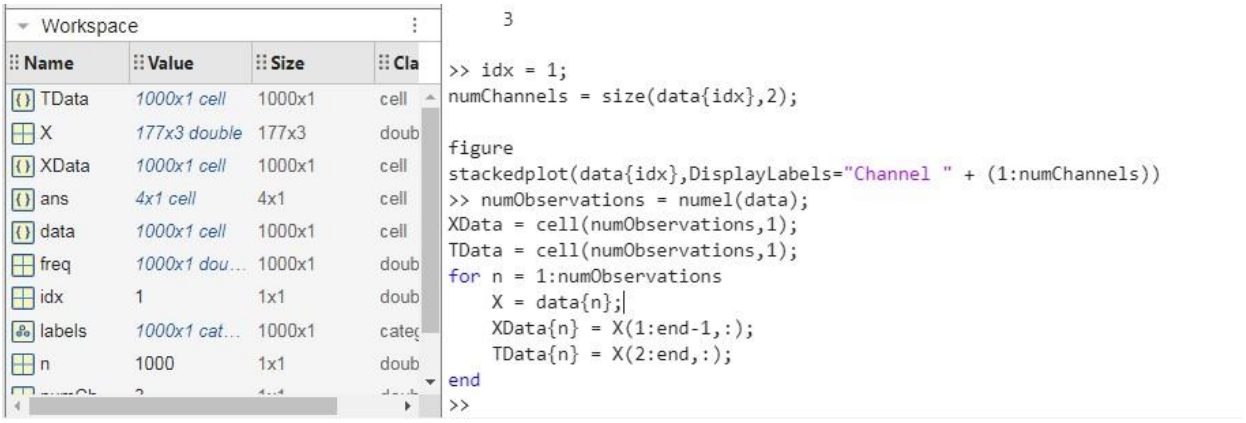


Рис. 4.1 Завантаження набору даних

Потрібно подивитися кількість каналів. Для навчання нейронної мережі LSTM кожна послідовність повинна мати однакову кількість каналів. Візуалізуємо деякі послідовності.



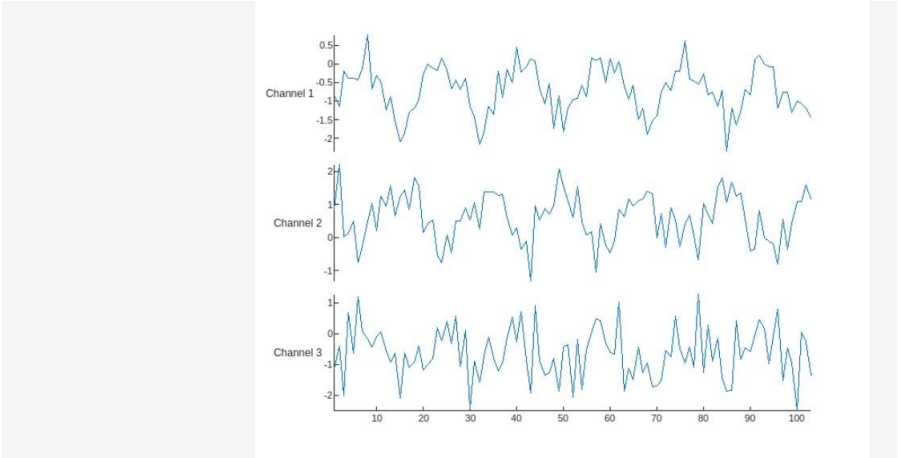


Рис. 4.2 Візуалізація даних

Підготуємо дані для навчання. Щоб спрогнозувати значення майбутніх часових кроків послідовності, поставимо цілі як навчальні послідовності зі значеннями, зрушеними на один часовий крок, при цьому не включаємо кінцевий часовий крок у навчальні послідовності. Іншими словами, на кожному часовому етапі вхідної послідовності нейронна мережа LSTM навчається прогнозувати значення наступного кроку часу. Підготовка даних показана на попередньому слайді справа.

Розділемо дані на навчальні, перевірочні та тестові набори при цьому використовуємо 80% даних для навчання, 10% для перевірки та 10% для тестування. Щоб розділити дані, використовуємо trainingpartitions функцію.

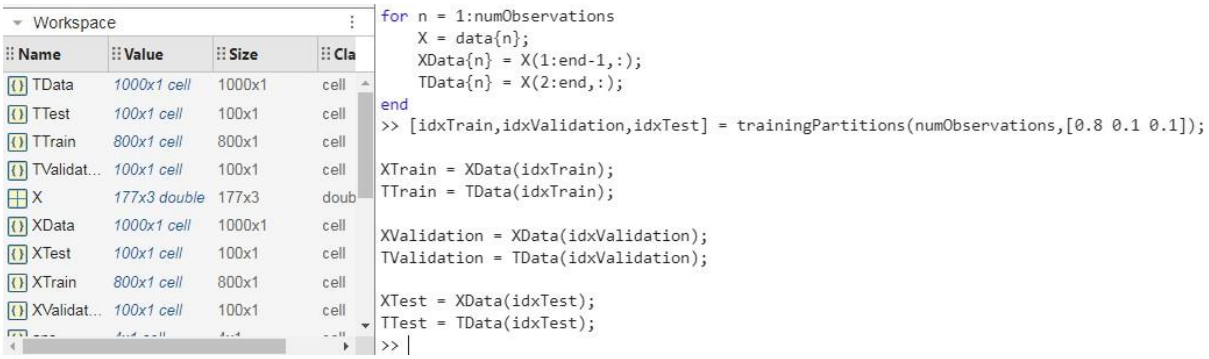


Рис. 4.3 Розділення даних

Для кращої відповідності та запобігання розбіжності у навчанні нормалізуємо предиктори та цілі так, щоб канали мали нульове середнє значення та одиничну дисперсію. Коли робимо прогноз також потрібно нормалізувати дані перевірки та тестування, використовуючи ту саму статистику, що й дані навчання.

Далі розраховуємо середнє значення та стандартне відхилення для кожного каналу для послідовностей. Щоб легко обчислити середнє та стандартне відхилення для навчальних даних, створимо числові масиви, що містять об'єднані послідовності, за допомогою функції cell2mat.

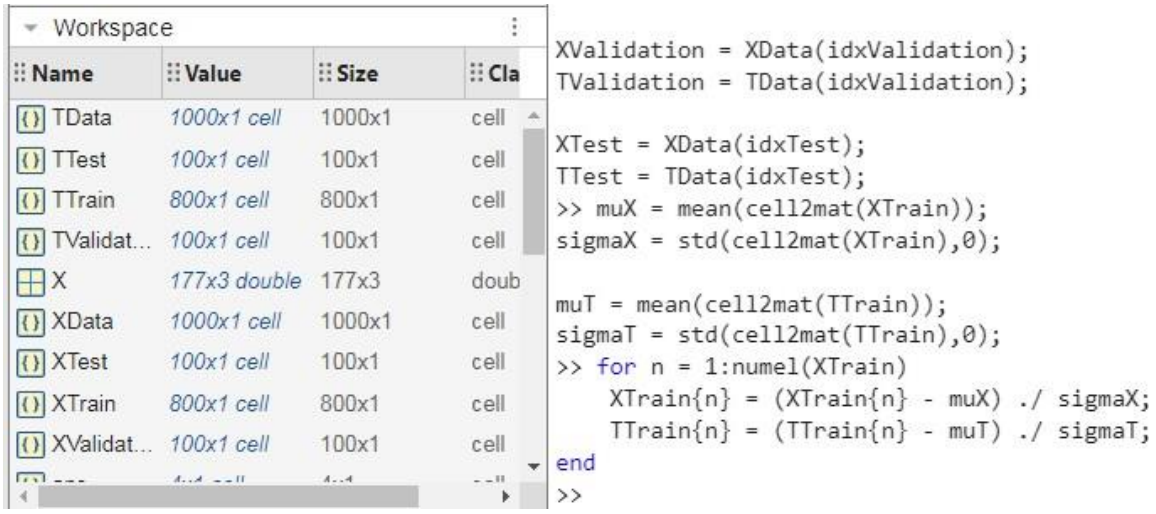


Рис. 4.4 Розрахунок середніх значень та стандартних квадратичних відхилень

Нормалізуємо дані навчання, використовуючи обчислені значення середнього та стандартного відхилення.

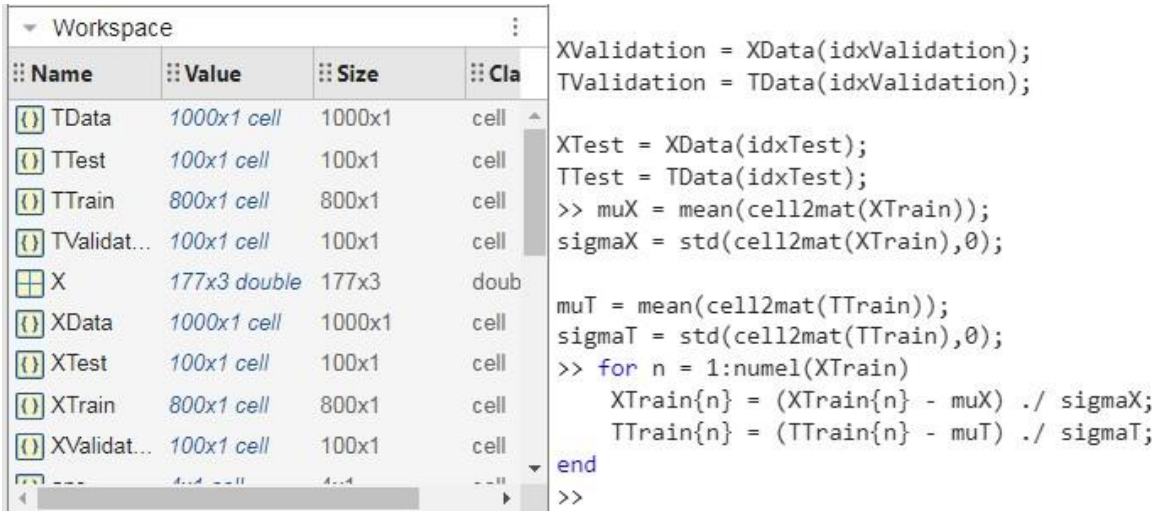


Рис. 4.5 Нормалізація даних для навчання, перевірки та тестування

Також нормалізуємо дані перевірки та тестування, використовуючи статистику, розраховану на основі даних навчання.

Далі вже будемо визначати мережеву архітектуру.Щоб побудувати мережу, відкриємо програму Deep Network Designer.

Щоб створити мережу послідовностей, у розділі «Мережі послідовностей» зупинемось на «Послідовність-до-послідовності» та натиснемо «Відкрити».

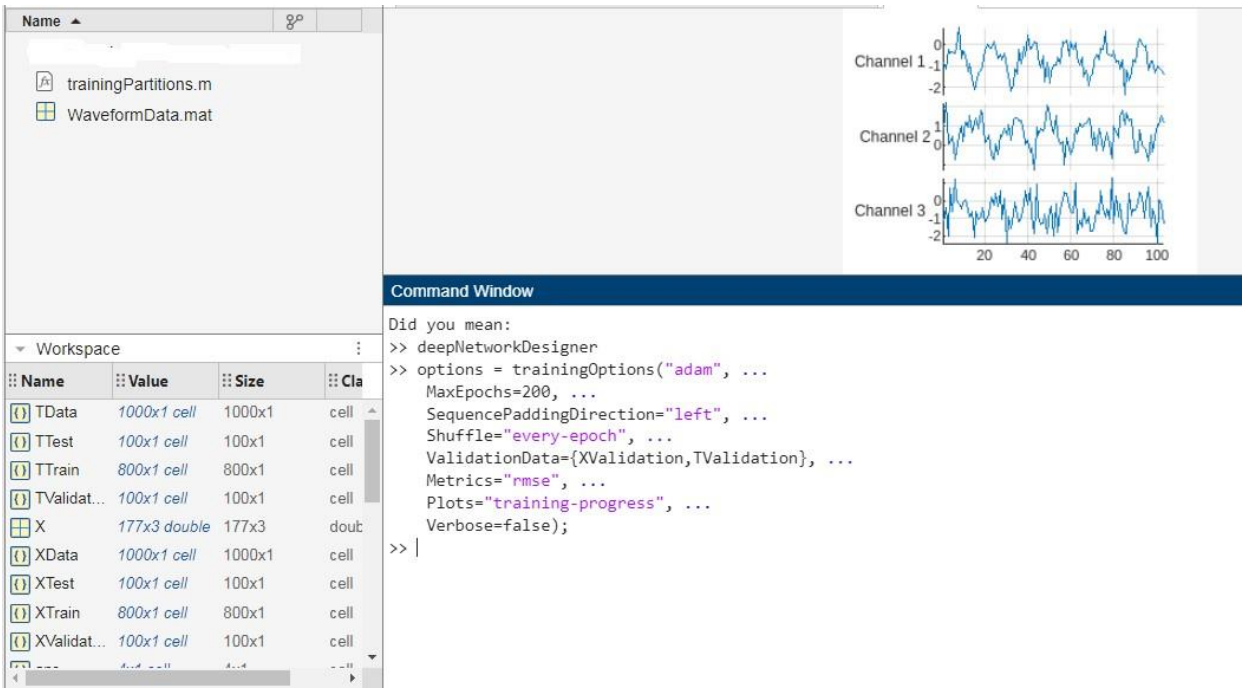
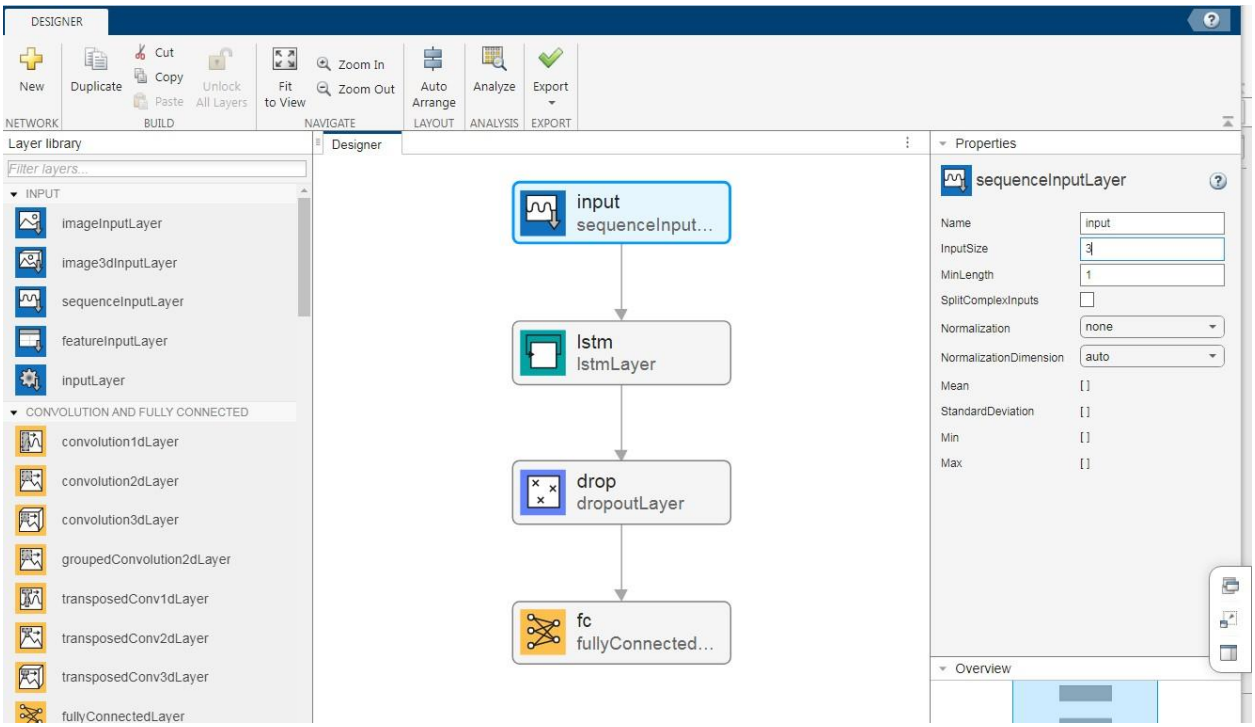


Рис. 4.6 Відкриття Deep Network Desiner

При цьому відкривається заздалегідь створена мережа, що підходить для класифікації завдань послідовностей. Мережа містить такі шари: sequenceInputLayer, lstmLayer, dropoutLayer, fullyConnectedLayer, softmaxLayer.

Ви можете перетворити класифікаційну мережу на мережу, придатну для прогнозування часових рядів, відредагувавши останні гілки. Спочатку видаліть шар softmax. Потім налаштуйте властивості шарів так, щоб вони підходили до набору даних Waveform. Оскільки мета полягає в тому, щоб спрогнозувати майбутні точки даних у часовому ряду, розмір вихідних даних повинен бути таким самим, як розмір вхідних даних. У цьому прикладі вхідні дані мають три вхідні канали, тому вихід мережі також повинен мати три вихідні канали. Виберіть вхідний шар послідовності input і встановіть для параметра InputSize значення 3. Виберіть повністю підключений шар fc і встановіть для OutputSize значення 3.



Шар LSTM має 128 прихованих модулів. Кількість прихованих модулів визначає, скільки інформації буде засвоєно шаром. Використання більшої кількості прихованих одиниць може дати точніші результати, але з більшою ймовірністю призведе до перенавчання даних навчання. Шар виключення допомагає уникнути перенавчання, випадково встановлюючи вхідні дані шару на нуль і ефективно змінюючи мережеву архітектуру між ітераціями навчання. Вища ймовірність відсіву може покращити узагальнення моделі за рахунок втрати інформації та уповільнення процесу навчання.

Щоб перевірити, чи готова мережа до навчання, натисніть «Аналіз» . Аналізатор мережі глибокого навчання не повідомляє про помилки чи попередження, тому мережа готова до навчання. Щоб експортувати мережу, натисніть «Експорт» . Програма зберігає мережу змінної net_1.

Далі будемо задавати параметри навчання.

Вкажемо варіанти навчання. Вибір одного з варіантів потребує емпіричного аналізу. Щоб вивчити різні конфігурації варіантів навчання шляхом проведення експериментів, можна використовувати програму Experiment Manager .

• Навчання за допомогою оптимізації Адама.

• Навчання за 200 епох. Для більших наборів даних, можливо, не потрібно тренуватися на такій великій кількості епох для доброго пристосування

• У кожному міні-пакеті натискаємо ліву кнопку миші на послідовності, щоб вони мали однакову довжину. Заповнення ліворуч не дозволяє RNN прогнозувати значення заповнення на кінцях послідовностей.

• Перетягуємо дані кожної епохи.

• Слідкуємо за перенавчанням, використовуючи дані перевірки.

• Контролюємо середньоквадратичну помилку.

• Відобразимо хід навчання на графіку.

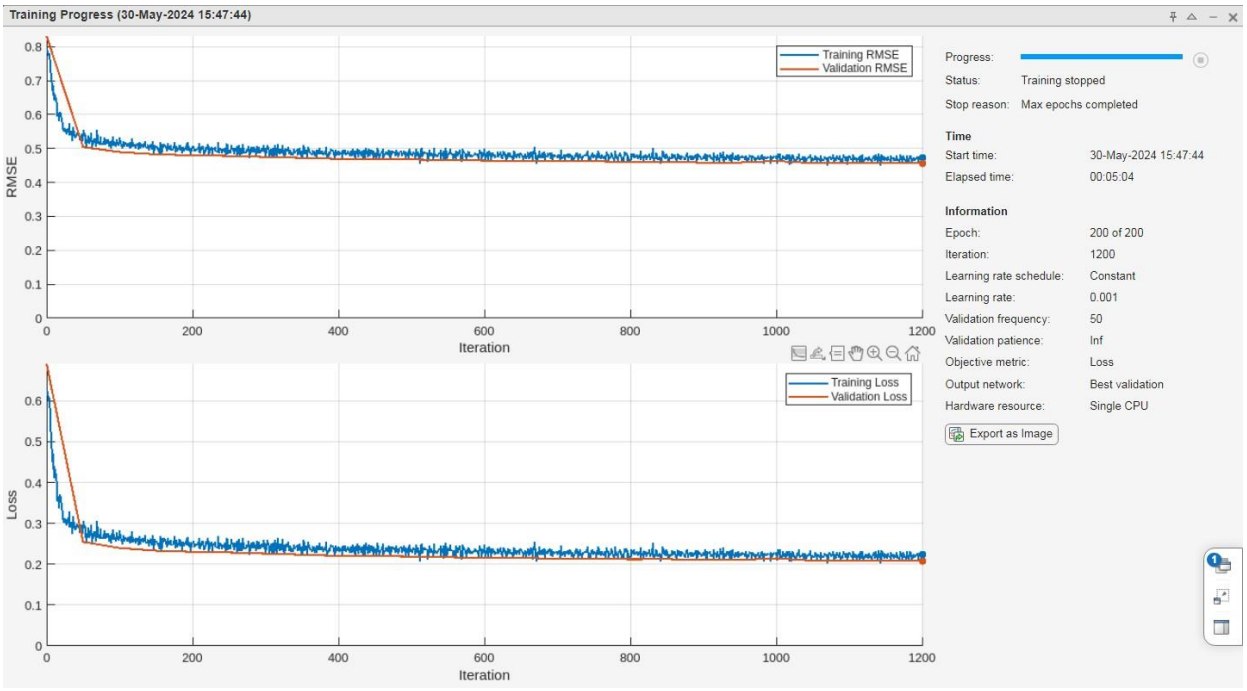
• Вимкнемо докладний висновок.

```
options = TrainingOptions( "adam" , ...
   MaxEpochs=200, ...
   SequencePaddingDirection= "left" , ...
   Shuffle= "every-epoch" , ...
   ValidationData={XValidation,TValidation}, ...
   Metrics = "rmse" , ...
   Plots= "прогресс обучения" , ...
   Verbose=false);
```

Навчаємо нейронну мережу LSTM за допомогою функції trainnet.

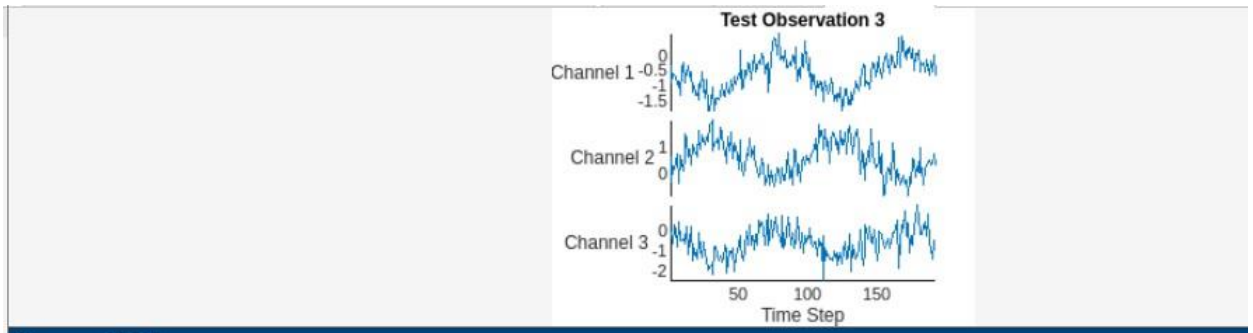
Для регресії використовуємо середньоквадратичну помилку втрат. За замовчуванням trainnet функція використовує графічний процесор, якщо він доступний. Інакше функція використовує ЦП. Щоб вказати середовище виконання, використовуйте ExecutionEnvironment - опцію навчання.

```
net = trainnet(XTrain,TTrain,net_1,"mse",options);
```



Прогноз майбутніх часових кроків

Враховуючи вхідний часовий ряд або послідовність, щоб спрогнозувати значення кількох майбутніх часових кроків, використовуємо функцію predict для прогнозування часових кроків по одному і оновлюємо стан RNN при кожному прогнозі. Для кожного прогнозу використовуємо попередній прогноз як вхідні дані для функції. Візуалізуємо одну із тестових послідовностей на графіку.




```
Command Window
>> mean(err,"all")

ans =

  single

    0.5215

>> idx = 3;
X = XTest{idx};
T = TTest{idx};

figure
stackedplot(X,DisplayLabels="Channel " + (1:numChannels))
xlabel("Time Step")
title("Test Observation " + idx)
>>
```

Існує два методи прогнозування: прогнозування з відкритим циклом та прогнозування із закритим циклом.

• Open loop forecasting – прогнозування наступного часового кроку у послідовності, використовуючи лише вхідні дані. Роблячи прогнози для наступних часових кроків, збираються справжні значення з джерела даних та використовуються як вхідні дані.

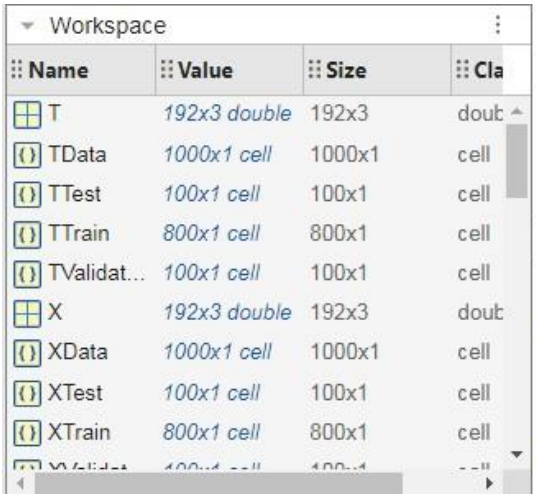
• Close loop forecasting – прогнозування наступних часових кроків в послідовності, використовуючи попередні прогнози як вхідні дані. У цьому випадку модель не потребує справжніх значень прогнозування.

Використовується прогнозування зі зворотним зв'язком, щоб спрогнозувати кілька наступних часових кроків або коли ви немає справжніх значень, які можна надати RNN перед виконанням наступного прогнозу.

Розімкнене прогнозування. Виконаємо прогнозування у відкритому циклі. Ініціалізуємо стан RNN, спочатку скинувши стан за допомогою resetState функції, а потім зробимо початковий прогноз, використовуючи

перші кілька часових кроків вхідних даних. Оновимо стан RNN за допомогою перших 75 часових кроків вхідних даних.

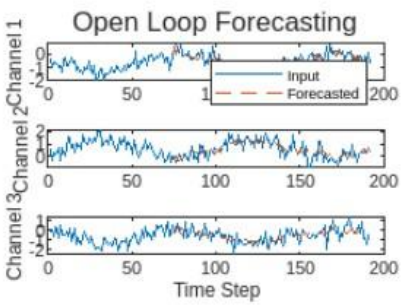
Щоб спрогнозувати подальші прогнози, виконаємо цикл за тимчасовими кроками і зробимо прогнози за допомогою цієї попередньої функції. Після кожного прогнозу оновлюємо стан RNN. Прогнозуємо значення для часових кроків тестового спостереження, що залишилися, перебираючи часові кроки вхідних даних і використовуючи їх як вхідні дані для RNN. Останній часовий крок початкового прогнозу є першим прогнозованим часовим кроком.




```
title("Test Observation " + idx)
>> net = resetState(net);
offset = 75;
[Z,state] = predict(net,X(1:offset,:));
net.State = state;
>> numTimeSteps = size(X,1);
numPredictionTimeSteps = numTimeSteps - offset;
Y = zeros(numPredictionTimeSteps,numChannels);
Y(1,:) = Z(end,:);

for t = 1:numPredictionTimeSteps-1
    Xt = X(offset+t,:);
    [Y(t+1,:),state] = predict(net,Xt);
    net.State = state;
end
>>
```

Порівняємо прогнози із вхідними значеннями.



Command Window

```
end
>> figure
t = tiledlayout(numChannels,1);
title(t,"Open Loop Forecasting")

for i = 1:numChannels
    nexttile
    plot(X(:,i))
    hold on
    plot(offset:numTimeSteps,[X(offset,i) Y(:,i)'],"--")
    ylabel("Channel " + i)
end

xlabel("Time Step")
nexttile(1)
legend(["Input" "Forecasted"])
>>
```

Прогнозування з закритим циклом дозволяє прогнозувати довільну кількість часових кроків, але може бути менш точним порівняно з прогнозуванням з відкритим циклом, оскільки RNN не має доступу до справжніх значень під час прогнозування.

Висновки до розділу 4

Наведено опис нейромережевого підходу для вирішення завдання управління транспортними потоками в мережі міських доріг. Управління транспортними потоками здійснюється завдяки вирішенню задачі оптимального управління на математичній моделі, побудованій на основі теорії керованих мереж. Штучна нейронна мережа забезпечує налаштування параметрів моделі при виникненні розбіжності між вихідними даними, отриманими на моделі, та вихідними даними на реальному об'єкті, ділянці мережі доріг. За допомогою нейронної мережі у

Matlab виконано прогнозування значень інтенсивності транспортного потоку у певні моменти часу.

ВИСНОВКИ

У дипломній роботі розглянуто актуальну проблему аналізу та моделювання даних в інформаційних системах за допомогою нейронних мереж, перспективні інструменти розробки та навчання нейромережевих моделей та середовища програмування на мові Python з бібліотеками TensorFlow і Keras. Проведена автоматизація моделювання процесів визначення за допомогою нейронних мереж, надано обґрунтування принципів створення програмного додатку для визначення деталей автомобільних систем.

Висвітлено загальні поняття, що використовуються у комп'ютерних системах, основні принципи роботи у комп'ютерних системах, методи застосуванні комп'ютерних технологій при дослідженні процесів динаміки величин, що характеризують транспортні системи.

В роботі детально обґрунтовано основні поняття нейромережевого моделювання, принципи побудови штучних нейронних мереж, їх архітектура, значення нейромережевого моделювання та сфери його застосування. Було зроблено постановку задачі визначення основних принципів розробки програмного забезпечення для визначення, розпізнавання деталей складових частин автотранспортних засобів та частин обладнання у мобільних додатках для застосування на мобільних пристроях. Після проведення огляду та опису роботи загорткових мереж за певними алгоритмами, описано архітектуру та принцип роботи нейронної мережі MobileNetV2, яку взято за основу, тобто нейронну мережу для навчання моделі задачі визначення та розпізнавання об'єктів. Проведено огляд програмних засобів для навчання нейронної мережі та вибору платформи для розробки мобільного додатку.

Також після аналізу та вибору програмного стеку для навчання нейронної мережі, а саме платформи TensorFlow та мови програмування Python, для

створення основних принципів розробки мобільного додатку для визначення деталей обладнання обрано середовище проектування з використанням програмного забезпечення Android Studio, так як це найбільш популярне та відповідне програмне забезпечення для розробки мобільного додатка.

Мета роботи визначила основний напрямок досліджень - навчання моделі розпізнаванню деталей обладнання, а також розробка програми для комфортного використання навченої моделі. Для цього були досліджені моделі машинного навчання, які задовольняли б нашим вимогам, а точніше: мати невелику вагу. Таким чином, надано основні принципи проектування програмного додатку для мобільних пристроїв.

Також, окремим розділом виконано прогнозування значень характеристик процесів, що відбуваються при функціонуванні транспортних систем, за допомогою Deep Network Designer системи комп’ютерної математики Matlab. Наведено опис нейромережевого підходу для вирішення завдання управління транспортними потоками в мережі міських доріг. За допомогою нейронної мережі у Matlab виконано прогнозування значень інтенсивності транспортного потоку у певні моменти часу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Darken, C. Learning rate schedules for faster stochastic gradient search / Darken C., Chang, J. Moody, J. // Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop, September 1–11, 1992.
2. Dauphin Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization / Dauphin, Y., Pascanu, R., Gulcehre C., Cho K., Ganguli S., Bengio Y. – URL: http://arxiv.org/abs/1406.2572 (дата звернення 18.11.2024).
3. Nielsen M. Neural Networks and Deep Learning. URL: http://neuralnetworksanddeeplearning.com/ (дата звернення 28.10.2024)
4. Contributors to Wikimedia projects. Artificial intelligence - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Artificial_intelligence (дата звернення: 30.11.2024).
5. Goodfellow I. Deep Learning / Goodfellow I., Bengio Y. and Courville A.; Cambridge MA: MIT Press [2017] – 780 p.
6. Turing A. M. Computing machinery and intelligence. *Harmondsworth* : Penguin, 1981.
7. Hackeling Gavin. Mastering Machine Learning with scikit-learn / Hackeling Gavin. - Packt Publishing Ltd. - 2014. – P. 1–32.
8. Densely Connected Convolutional Networks. URL: https://arxiv.org/pdf/1608.06993v3.pdf (дата звернення 22.11.2024).
9. Zhou Z.-H. Machine Learning. Singapore : Springer Singapore, 2021. URL: https://doi.org/10.1007/978-981-15-1967-3 (дата звернення: 29.11.2024).
10. Densely Connected Convolutional Networks. URL: https://arxiv.org/pdf/1608.06993v3.pdf (дата звернення: 21.11.2024).
11. The Benefits of AI in Website Development. Unicorn Platform AI Website Builder for Busy Founders. URL: https://unicornplatform.com/blog/the-benefits-of-ai-in-website-development-how-artificial-intelligence-is-revolutionizing-web-design/ (дата звернення: 15.11.2024)

12. Park S. B., Lee J. W., Kim S. K. Content based image classification using a neural network // Pattern Recognition Lett. 2004. V. 25. N 3. P. 287–300.
13. Fan A. L. The Traffic Prediction and Control Based on Rough Set Theory. Advanced Materials Research. 2013. Vol. 756-759. P. 632–635. URL: https://doi.org/10.4028/www.scientific.net/amr.756-759.632 (дата звернення: 2.12.2024).
14. Human-centered AI and robotics / S. Doncieux et al. *AI Perspectives*. 2022. Vol. 4, no. 1. URL: https://doi.org/10.1186/s42467-021-00014-x (дата звернення: 14.11.2024).
15. 30 Best AI Image Generators (Free & Paid) 2023. *Rigorous Themes*. URL: https://rigorousthemes.com/blog/best-ai-image-generators/ (дата звернення: 3.12.2024)
16. Springen Berg J. T. Striving for Simplicity: The All-Convolutional Net / Springenberg, J. T. Dosovitskiy, A.; Brox, T. & Riedmiller, M. URL: https://arxiv.org/abs/1412.6806
17. Sermanet P. Traffic Sign Recognition with Multi-Scale Convolutional Networks / Sermanet, P., LeCun, Y. // The 2011 International Joint Conference on Neural Networks, September 2011.
18. Feng, Pufei & Fu, Zhiyi & Hu, Linshu & Wu, Sensen & Wang, Yuanyuan & Zhang, Feng. (2023). 3D-EddyNet: A Novel Approach for Identifying Three-Dimensional Morphological Features of Mesoscale Eddies in the Ocean. Journal of Marine Science and Engineering.
19. Gonzalez A. C., Sossa J. H., Felipe E. M. Wavelet transforms and neural networks applied to image retrieval // Intern. conf. on pattern recognition. Hong Kong (China), 20–24 Aug. 2006. P. 909–912.
20. Express/Node introduction - Learn web development | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (дата звернення: 15.11.2024).
21. The Most Modern Mobile Touch Slider. Swiper. URL: https://swiperjs.com/ (дата звернення: 2.12.2024).

22. Schmidhuber, J. (2002) The Speed Prior: A New Simplicity Measure Yielding Near-Optimal Computable Predictions. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*. Lecture Notes in Artificial Intelligence, pages 216--228.
23. Охорона праці при роботі з комп'ютером / ПК. *Довідник спеціаліста з охорони праці*. URL: https://pro-op.com.ua/article/183-ohoron-prats-pri-robot-z-kompyuterom(дата звернення: 30.11.2024).
24. ДСТУ 3396.2-97. *ТЗІ - інформаційна безпека та захист інформації*. URL: https://tzi.com.ua/478.html (дата звернення: 5.12.2024).
25. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. *Офіційний вебпортал парламенту України*. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення: 8.11.2024).
26. Kingma D. P., Adam: A Method for Stochastic Optimization / Kingma D. P., Ba J.L. // International Conference on Learning Representations, May 7–9, 2015, San-Diego, USA.
27. Про затвердження Правил технічної експлуатації електроустановок споживачів : Наказ М-ва палива та енергетики України від 25.07.2006 р. № 258 : станом на 21 лют. 2017 р. URL: https://zakon.rada.gov.ua/laws/show/z1143-06#Text (дата звернення: 26.11.2024).
28. Ріпний О. С., Козачок Л. М. Основні принципи нейромережевого підходу до визначення об’єктів автомобільного обладнання. *«Інформаційні технології в освітньому процесі ЗВО»*: зб. наук. пр. за матеріалами Всеукраїнської науково-методичної конференції, м. Харків, 27 листопада 2024 р., Харків, 2024. С. 20-23.

ДОДАТОК А
ТЕЗИ НА КОНФЕРЕНЦІЮ

Основними завданнями дослідження принципів та методів нейромережевого моделювання є моделювання процесів, що відбуваються в інформаційній системі на основі отриманої вхідної інформації. Даний метод моделювання використовується для аналізу закономірностей процесу або процесів, цілей прогнозування значень досліджуваних показників, виділення, визначення оь’єктів інформації, генерації нових даних. Основними завданнями даної роботи є дослідження принципів нейромережевого моделювання визначення об’єктів, розробка навчання нейронних мереж за моделлю, визначення основних принципів розробки мобільного додатку для визначення деталей автомобіля за допомогою комп'ютерного зору.

Нейрокомп'ютерний підхід до обробки інформації покликаний реалізувати можливості, закладені у правій півкулі мозку. Він повинен не замінити існуючі комп'ютерні методи, а лише заповнити можливості, для яких не вдається побудувати формальні алгоритмічні схеми. Подібно до того, як у людському мозку ліва та права півкулі працюють спільно, сучасні інформаційні системи повинні використовувати симбіоз традиційних алгоритмічних та нейрокомп'ютерних методів для повноцінної та продуктивної обробки інформації.

Відомі логічні, безперервні та імпульсні моделі нейрона. Логічні моделі активно досліджувалися у 1960-70-х рр., але не набули подальшого розвитку. Імпульсні моделі більш близькі до фізичної природи процесів, що відбуваються в нервовій клітині, проте їхня теорія не так розвинена, як у безперервних, і вони все ще не знаходять широкого застосування.

Безперервна модель нейрона імітує в першому наближеннівластивості біологічного нейрона і включає набір синапсів або зв'язків, кожна з яких характеризується своєю вагою (*weight*).

Зокрема, сигнал *xi* на вході синапсу *i* множиться на вагу *wi*.

Позитивні значення ваг *wi* відповідають збуджуючим синапсам, негативні - гальмівним. Суматор обчислює зважену щодо відповідних синапсів суму вхідних сигналів нейрона.

Функція активації обмежує амплітуду вихідного сигналу. Ця функція також називається функцією стиснення.

Математичну модель формального нейрона можна представити рівнянням

$$p = \sum_{i=1}^{n} v_i \cdot x_i + v_o, \; y = f(p) \qquad (1)$$

де y - вихідний сигнал нейрона; $f(p)$ – функція активації нейрона; v_i - ваговий коефіцієнт синаптичного зв'язку i-го входу; x_i - i-й вхідний сигнал нейрона; v_0 - початковий стан (порушення) нейрона; $i = 1,2 \ldots n$ - номери входів нейрона; n-число входів.

Відштовхуючись від огляду класичних завдань, до виконання яких застосовуються згорткові нейронні мережі, можна дійти висновку, що застосування CNN не обмежується лише ними, і можна знайти ще спрсоби використання загорткових мереж у нестандартних сферах застосування.

Виходячи з вищеперерахованих аспектів, необхідно реалізувати програмне забезпечення, яке буде представляти мобільний додаток, який повинен буде взаємодіяти з моделлю машинного навчання, отримувати на вхід зображення, розпізнавати об'єкти REAL-time на зображенні, не бути вимогливою до ресурсів пристрою, працювати на платформі Android.

Таким чином, було описано поняття згорткової нейронної мережі та її відмінність від класичної нейронної мережі. Також були проаналізовані існуючі класичні завдання, що виконуються згортковими нейронними мережами. Для кожного завдання описано сферу застосування, для якої розробляються конкретні CNN. З отриманих даних було сформульовано та описано постановку задачі, виконання якої передбачає розробку основних принципів створення та роботи програмного забезпечення для виконання

ції задачі, в тому числі на мобільних пристроях, за допомогою нейромережевого моделювання.

Функція активації в нейронній мережі відіграє вирішальну роль у визначенні виходу вузла (вершини) шляхом обмеження амплітуди виходу. Ці функції також відомі як передатні функції. Функції активації допомагають нейронній мережі вивчати складні шаблони вхідних даних більш ефективніше, вимагаючи меншої кількості нейронів. Нейронні мережі, які використовують функції активації ReLU6 разом із пакетною нормалізацією партії перед кожним шаром, можна описати наступним чином.

Функція активації ReLU6 обмежує кількість одиниць до 6, що допомагає нейронній мережі швидко вивчати розріджені ознаки, а також запобігає безкінечному збільшенню градієнтів. Функція активації ReLU6 може бути математично визначена таким чином:

$$f(x) = \min(\max(0, x), 6) \quad (2)$$

ReLU6 використовується для надійності під час використання порівняно з низькою точністю обчислень на основі MobileNetV1. Слід зазначити, що Batch Normalization (BN) і ReLU застосовуються після кожної згортки.

У нашому додатку використовуватиметься модель нейромережі MobileNetV2. Провівши огляд програмних засобів для навчання нейронної мережі, який було розглянуто вище, будемо обирати середовище TensorFlow для побудови та навчання нейронної мережі, Python і PyCharm у якості середовища розробки та мови програмування.

ДОДАТОК Б

ЛІСТИНГ КОДУ

```
#!/bin/bash

# Створимо оточення для роботи:
conda create -n object_detection_prepare pip python=3.6

# Активуємо його:
conda activate object_detection_prepare

# Встановимо залежності:
pip install --ignore-installed --upgrade tensorflow==1.14
pip install --ignore-installed pandas
pip install --ignore-installed Pillow
pip install lxml
conda install pyqt=5


#!/bin/bash

python ./object_detection/preprocessing/image_resize.py -i /
        ./object_detection/images --imageWidth=800 --imageHeight-600


def xmlToCsv(path):
        xmlList = [ ]
        for xmlFile in glob.glob(path + '/*.xml'):
                tree = ET.parse(xmlFile)
                 root = tree.getroot()
                 for member in root.findall('object'):
                        value = (root.find('fitename').text,
                                int (root. find('size')[0].text),
                                int (root. find('size')[1].text),
                                member[0].text,
                                int (member [4] [0] .text),
                                int (member[4][1].text),
                                int (member [4][2].text),
                                int (member[4][3].text)
                                )
                        xmlList append(value)
            columnName = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
            xm_df = pd.DataFrane(xmlList, columns=columnName)
            return xml_df
def main():
parser = argparse.ArgunentParser()

Parser.add_argument('-in', '--input_xml', help='Файл для обробки', type=str, required=True)
Parser.add_argument('-out', '--output_csv', help='Файл після обробки', type=str, required=True)
args = parser.parse_args()
```

```
xml_df = xmlToCsv (args. input_xml)
xml_df.to_csv(args.output_csv, index=None)
print(‘XML конвертовано у CSV')


#!/bin/bash
python ./models/research/object_detection/legacy/train.py --logtostderr /
                                                          --train_dir=./training_demo/training /
                                                          --pipeline_config path=.
/training_demo/training/ssdlite_mobilenet_v2_coco.config


#!/bin/bash

python /content/models/research/object_detection/export_inference_graph.py --input_type image
tensor \
--pipeline_config path /content/training_demo/training/ssdlite_mobilenet_v2_coco.config \
--trained_checkpoint_prefix /content/training_demo/training/model.ckpt-07070 \

--output_directory /content/training_demo/training/output_inference_graph_v1.pb


android {
      aaptOptions {
            noCompress “tflite”
      }
}
dependencies {
       implementation ‘org.tensorflow:tensorflow-lite:0.0.0-nightly’
       implementation ‘org.tensorflow:tensorflow-lite-gpu:0.0.0-nightly’
       implementation ‘org.tensorflow:tensorflow-lite-support:0.0.0-nightly’
}


protected Classifier(Activity activity, Device device, int numThreads) throws
        IOException {
      tfliteModel = FileUtil.loadMappedFile(activity, getModelPath());
      switch (device) {
          case NNAPT:
             nnApiDelegate = new NnApiDelegate();
             tfliteOptions.addDelegate(nnApiDelegate);
             break;
           case GPU:
              gpuDelegate = new GpuDelegate();
              tfliteOptions.addDelegate(gpuDelegate);
              break;
           case CPU:
               break;
  }
  tfliteOptions.setNumThreads(numThreads);
  tflite = newInterpreter(tfliteModel, tfliteOptions);
  labels = FileUtil.loadLabels(activity, getlabelPath());
```

ДОДАТОК В

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ДИПЛОМНОЇ РОБОТИ

Міністерство освіти і науки України
Харківський національний автомобільно-дорожній університет
Механічний факультет
Кафедра комп'ютерних наук і інформаційних систем

ДИПЛОМНА РОБОТА
магістра

НЕЙРОМЕРЕЖЕВЕ МОДЕЛЮВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

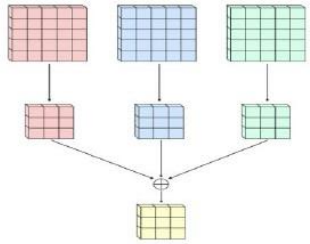
Завідувачка кафедри канд. техн. наук, доцентка Г.А. Плєхова
Нормоконтролер, к. т. н., доцентка Г. А. Плєхова
Керівник, д. т. н., професор А. М, Кущин

Студент гр. МК-61-23 О. С. Рішний

ПОСТАНОВКА ЗАДАЧІ.

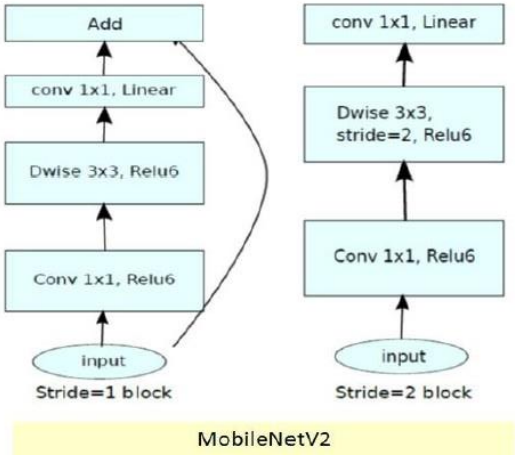
- **Об’єкт дослідження** – перспективні інструменти розробки та навчання нейромережевих моделей та середовища програмування на мові Python з бібліотеками TensorFlow і Keras для аналізу даних в інформаційних системах, для визначення типів та видів деталей автомобілів.
- **Предмет дослідження** – автоматизація моделювання процесів визначення за допомогою нейронних мереж, обґрунтування принципів створення програмного додатку для визначення деталей автомобільних систем.
- **Метою дипломної роботи** є аналіз та дослідження моделей штучних нейронних мереж, побудова нейромережевої моделі визначення об’єктів, опис основних принципів розробки мобільного додатку для визначення дрібних деталей автомобіля та автомобільного обладнання.
- **Основні виконані завдання та результати** – описати цілі, функції, види існуючих методів розробки моделей процесів за допомогою штучних нейронних мереж, побудувати нейромережеву модель визначення об’єктів, а саме деталей автомобільного обладнання, визначити основні принципи створення мобільного додатку, що використовує навчану нейронну мережу, для визначення деталей автомобіля та інструментів.

Застосування нейронних мереж для визначення, розпізнавання складових частин обладнання

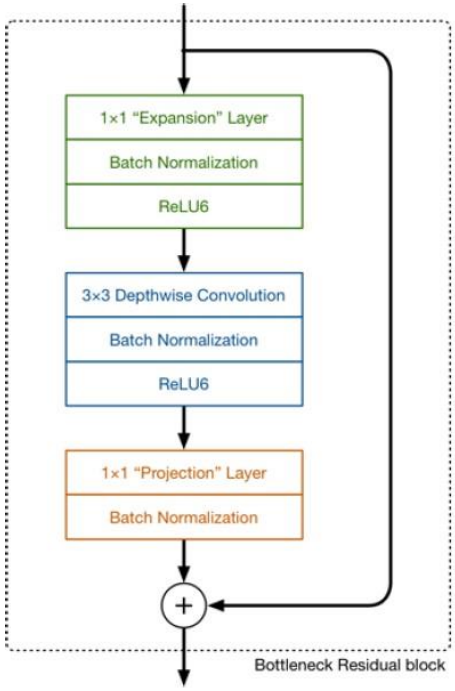


Згорткові нейронні мережі ЗНМ (англ. convolutional neural network, CNN)

Структура згорткових блоків MobileNetV2



В блоці три згорткові шари. Останні два: глибинна згортка, яка фільтрує вхідні дані, за якою слідує точковий згортковий шар 1×1.
У V2 точкова згортка навпаки: зменшує кількість каналів. Ось чому цей шар тепер називається проекційним шаром — він проектує дані у тензор.
Перший шар - новий у блоці. Це також згортка 1×1. Його мета – розширити кількість каналів у даних, перш ніж вони підуть у згортку по глибині. Це один із гіперпараметрів.



Основна частина архітектури мережі MobileNetV2

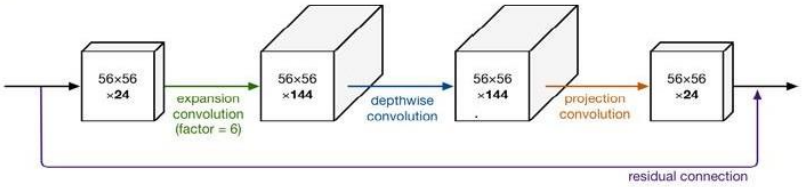
Функція активації в нейронній мережі відіграє вирішальну роль у визначенні виходу вузла (вершини) шляхом обмеження амплітуди виходу:

ReLU (Rectified Linear Unit) - це нелінійна функція активації. Вона перетворює вхідне значення в значення від 0 до позитивної нескінченості, якщо вхідне значення менше або рівно нулю, то ReLU видає тільки 0, в протилежному випадку - вхідне значення:

$$\mathbf{ReLU(x) = max(0, x)}$$

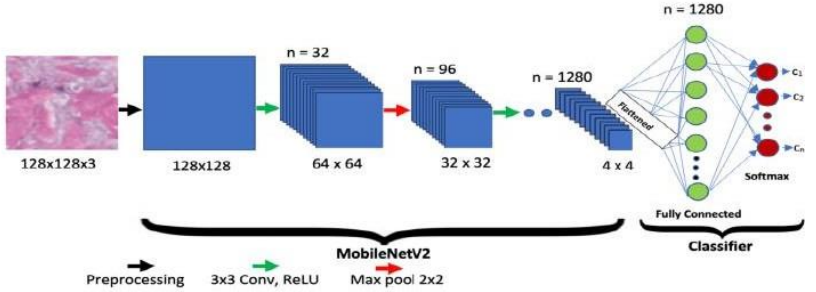
$$\mathbf{\mathit{ReLU6(x)} = min(max(0, \mathit{x}), 6)}$$

Структура згорткового блоку мережі



Структура шарів

MobileNetV2



Налаштування та виконання навчання нейронної мережі

```
[ ]: # Кількість класів
model.ssd.num_classes: 9

# Визначимо розмір пакету (кількість даних для навчання за одну ітерацію), кількість ітерацій та шлях до збереженої моделі:
train_config.batch_size: 18
train_config.num_steps: 20000
train_config. fine_tune_checkpoint: ”./training_demo/pre-trained-model/ssdlite_mobilenet_v2_coco/model..ckpt”

# Вкажемо кількість фото у тренувальному наборі

object_detection/training_demo/images/traineval_config.num_examples: 200

# Вкажемо шлях для набору даних для тренування

train_input_reader.label_map_path: “./training_demo/annotations/label_map.pbtxt”
train_input_reader.tf_record_input_reader.input_path:”. /training_demo/annotations/train.record”

# Вкажемо шлях до текстового набору даних

eval_input_reader.label_map_path: “./training_demo/annotations/label_map.pbtxt”
```

```
[*]: #!/bin/bash
python ./models/research/object_detection/legacy/train.py --logtostderr /
                                                           --train_dir=./training_demo/training /
                                                           --pipeline_config path=. /training_demo/training/ssdlite_mobilenet_v2_coco.config
```

Створення tflite моделі

```
[*]: #!/bin/bash

python /content/models/research/object_detection/export_tflite_ssd_graph.py --pipeline_config_path /content/training_demo/training/ssdlite_mobilenet_v2_
--trained_checkpoint_prefix /content/training_demo/training/model.ckpt-07070 \
--output_directory /content/training_demo/training/output_inference_graph_tf_lite.pb
```

Показ результатів навчання, параметрів моделі за допомогою інструмента візуалізації TensorBoard

Основні принципи розробки та проектування мобільного додатку для розпізнавання деталей

Отримання даних з камери клас CameraActivity

```
model = Model.valueOf(modelSpinner.getSelectedItem().toString().toUpperCase());
device = Device.value0f(deviceSpinner.getSelectedItem().toString());
numThreads = Integer.parseInt(threadsTextView.getText().toString().trim());
```

Реалізація класу Classifier

```
protected Classifier(Activity activity, Device device, int numThreads) throws
          IOException {
        tfliteModel = FileUtil.loadMappedFile(activity, getModelPath());
        switch (device) {
             case NNAPT:
                 nnApiDelegate = new NnApiDelegate();
                 tfliteOptions.addDelegate(nnApiDelegate);
                 break;
              case GPU:
                  gpuDelegate = new GpuDelegate();
                  tfliteOptions.addDelegate(gpuDelegate);
                  break;
              case CPU:
                   break;
   }
   tfliteOptions.setNumThreads(numThreads);
   tflite = newInterpreter(tfliteModel, tfliteOptions);
   labels = FileUtil.loadLabels(activity, getlabelPath());
```

Клас, який відповідає за розпізнавання

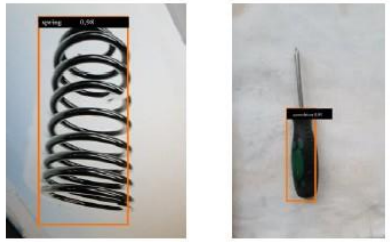
```
Map<String, Float> labeledProbability =
        new TensorLabel(labels,
                probabilityProcessor.process (outputProbabilityBuff
                    ))
                .getMapWithFloatValue();

private static List<Recognition> getTopKProbability(
            Map<String, Float> labelProb) {
    PriorityQueue<Recognition> pq =
            new PriorityQueue<>(
                    MAX_RESULTS,
                    new Comparator<Recognition>() {
                        @Override
                        public int compare(Recognition lhs,
                                Recognition rhs) {
                            return Float.compare(rhs.getConfidenc
                                (), lhs.getConfidence());
                        }
                    });

    for (Map.Entry<String, Float> entry : labelProb.entrySet()) {
        pq.add(new Recognition("" + entry.getKey(), entry.getKey(),
                entry.getValue(), null));
    }

final Arraylist<Recognition> recognitions = new ArrayList<>();
int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
for (int i = 0; i < recognitionsSize; ++i) {
recognitions.add(pq.poll());
}
return recognitions;
}
```

Після навчання моделі та реалізації основних принципів побудови програмних мобільних додатків ми будемо мати виконання процесу виділення леталей автомобільного обладнання та інструменту при роботі з ними.



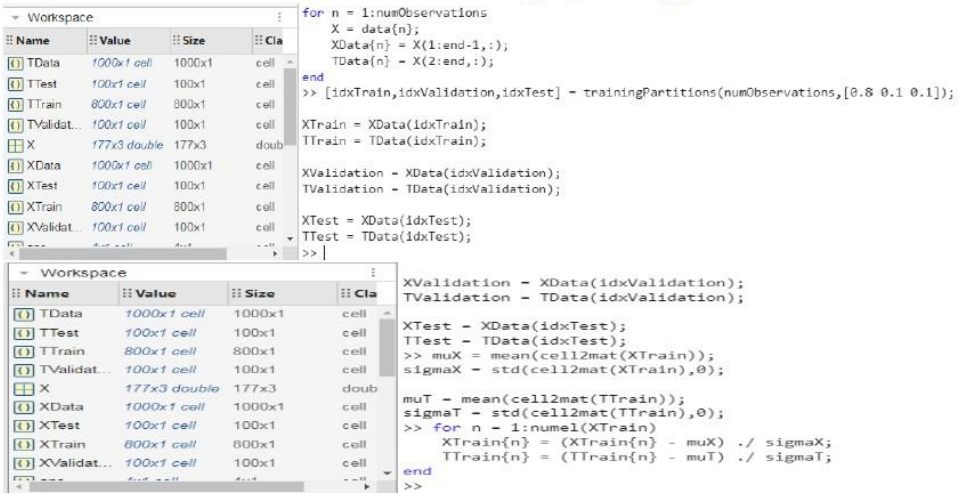
Штучні нейронні мережі для транспортного потоку дорожнього руху

Структура керування транспортними потоками за допомогою штучної нейронної мережі

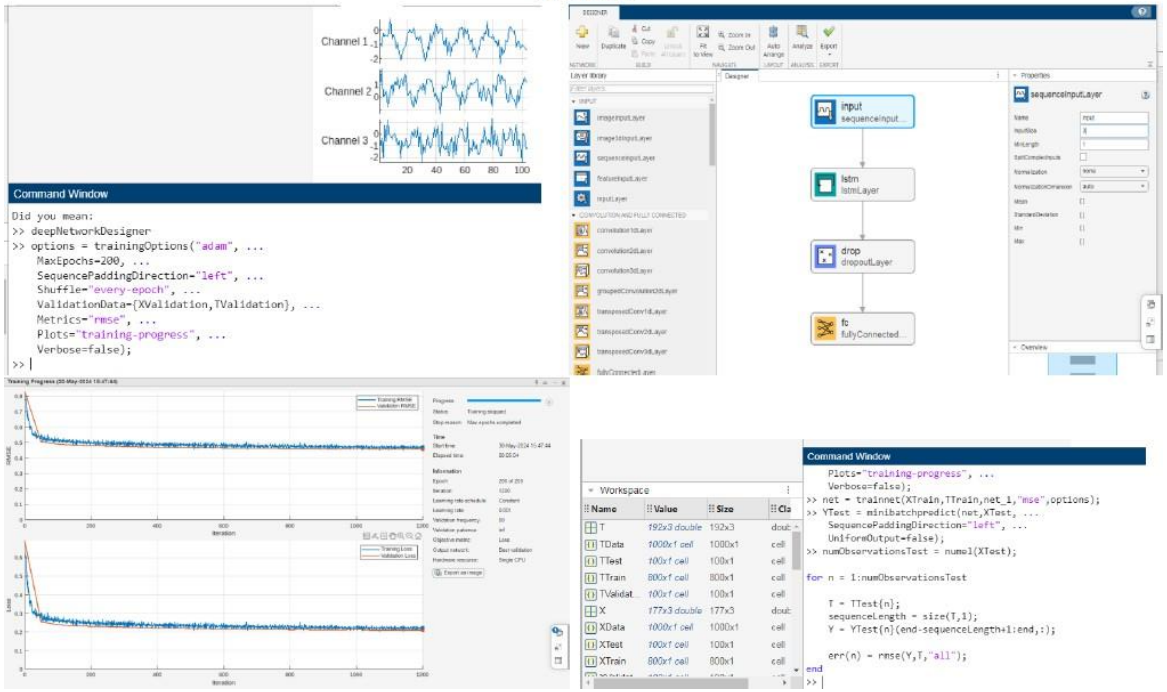
Побудова мережі прогнозування часових рядів за допомогою Deep Network Designer Matlab

Нехай випадкова величина $X(t, \Delta t)$ - число подій, що з'явилися на проміжку $(t, t + \Delta t)$. Інтенсивність потоку автомобілей у транспортній мережі:

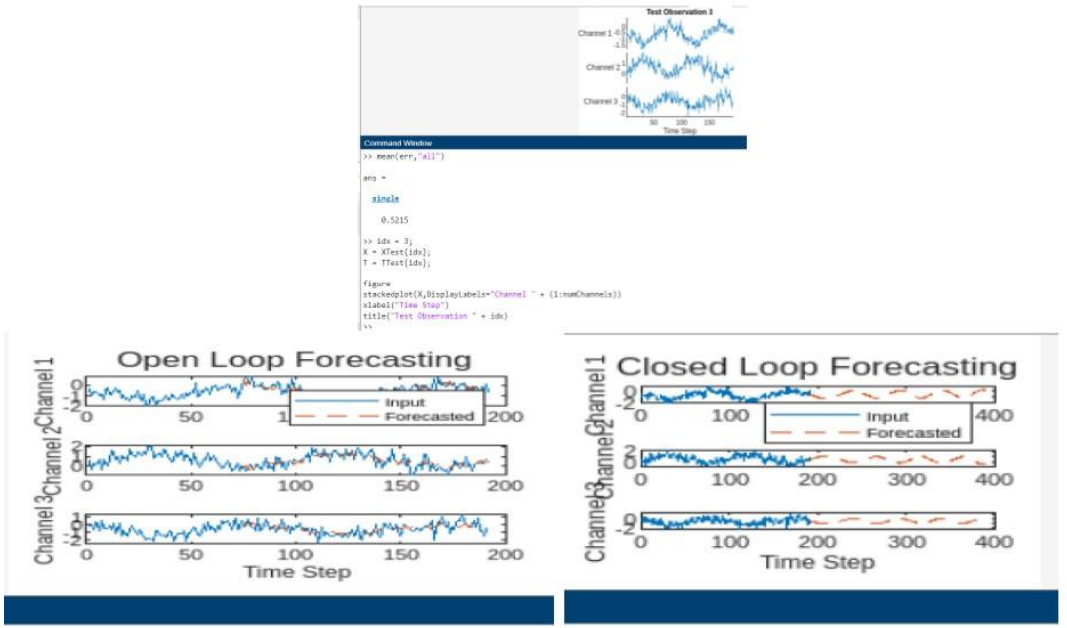
$$\lambda(t) = \lim_{\Delta t \to 0} \frac{M(X(t, \Delta t))}{\Delta t}$$



Побудова мережі прогнозування за допомогою Deep Network Designer Matlab



Побудова мережі прогнозування за допомогою Deep Network Designer Matlab



ВИСНОВКИ

В роботі детально обґрунтовано основні поняття нейромережевого моделювання, принципи побудови штучних нейронних мереж, їх архітектура, значення нейромережевого моделювання та сфери його застосування.

Мета роботи визначила основний напрямок досліджень - навчання моделі розпізнаванню деталей обладнання, а також розробка програми для комфортного використання навченої моделі. Для цього були досліджені моделі машинного навчання, які задовольняли б нашим вимогам, а точніше: мати невелику вагу. Таким чином, надано основні принципи проектування програмного додатку для мобільних пристроїв.

Також, окремим розділом виконано прогнозування значень характеристик процесів, що відбуваються при функціонуванні транспортних систем, за допомогою Deep Network Designer системи комп'ютерної математики Matlab. Наведено опис нейромережевого підходу для вирішення завдання управління транспортними потоками в мережі міських доріг. За допомогою нейронної мережі у Matlab виконано прогнозування значень інтенсивності транспортного потоку у певні моменти часу.