Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп’ютерних систем
Кафедра комп’ютерної інженерії

«Розпізнавання долонь на зображеннях за допомогою згорткових нейронних мереж»

Кваліфікаційна робота бакалавра
студента напряму підготовки
123 «Комп'ютерна Інженерія»
Дімітрія ТКАЧЕНКА
_____(підпис)

Науковий керівник, к.т.н., доцент
Олександр САМОЩЕНКО.
_____(підпис)

До захисту допускаю
Протокол засідання кафедри від
доцент
“___” _____2022р. №_____

Завідувач кафедри
кандидат фіз.-мат. наук,

Юрій БОЙКО

КИЇВ - 2022

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра містить 55 сторінок, 29 рисунків, 3 таблиці, 2 додатки, використано 19 інформаційних джерел.

ЗМІСТ

ОСНОВНІ ПОНЯТТЯ ТА ТЕРМІНИ **5**
ВСТУП **6**
РОЗДІЛ 1. ВИЯВЛЕННЯ ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ **7**
1.1 Постановка задачі виявлення об'єктів на зображеннях 7
1.2 Методи виявлення об'єктів на зображеннях 9
1.2.1 Визначення порогів 9
1.2.2 Масштабонезалежне перетворення ознак (SIFT) 10
1.2.3 Методи машинного навчання 13
1.3 Нейронні мережі для виявлення та класифікації об'єктів 14
1.3.1 Загальне визначення нейронної мережі 14
1.3.2 Навчання нейронних мереж 16
1.3.3 Згорткові нейронні мережі 16
1.3.4 Faster R-CNN (Faster Region-based CNN) 19
1.3.5 YOLO (You Only Look Once) 21
1.3.6 SSD (Single-Shot Detector) 23
ВИСНОВОК ДО РОЗДІЛУ 1 ..... **25**
РОЗДІЛ 2. ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ЗАСТОСУНКУ **26**
2.1 Вибір мови програмування 26
2.2 Вибір бібліотек та фреймворків 27
2.3 Вибір набору даних 28
2.4 Вибір архітектури нейронної мережі 30
ВИСНОВОК ДО РОЗДІЛУ 2 ..... **32**
РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ВИЯВЛЕННЯ ДОЛОНЬ НА ЗОБРАЖЕННЯХ НА БАЗІ ЗМН ТИПУ SSD **33**
3.1 Реалізація моделі мережі 33
3.2 Підготовка набору даних 35
3.3 Навчання моделі 39
3.4 Оцінка натренованої моделі 42
3.5 Застосування готової моделі 44
ВИСНОВОК ДО РОЗДІЛУ 3 ..... **46**
ВИСНОВКИ **47**
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ **48**

Додаток А **51**
Додаток Б **53**

ОСНОВНІ ПОНЯТТЯ ТА ТЕРМІНИ

ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, МАШИННЕ НАВЧАННЯ, ГЛИБИННЕ НАВЧАННЯ, КОМП’ЮТЕРНИЙ ЗІР, CNN, SSD, YOLO, FR-CNN, PYTHON, TENSORFLOW, KERAS, TRANSFER LEARNING, OBJECT DETECTION

ВСТУП

Задача розпізнавання людської долоні на зображенні полягає в детектуванні наявності долоні на вхідному зображенні та її локалізації, наприклад для подальшого відстеження руху долоні у просторі. Подібні задачі, що вимагають виявлення та локалізації деякого об’єкту на зображенні, є одними з класичних задач комп’ютерного зору. Для її виконання пропонується застосування сучасних практик машинного навчання, а саме – застосування згорткових нейронних мереж як сучасного та точного методу для рішення поставленої проблеми в режимі реального часу. Мета цієї роботи – дослідити існуючі способи детектування об’єктів, що базуються на використанні нейронних мереж та розробити застосунок, що за допомогою згорткової нейронної мережі здатен виявляти долоні людини на зображеннях.

РОЗДІЛ 1. ВИЯВЛЕННЯ ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ

1.1 Постановка задачі виявлення об'єктів на зображеннях

При обробці зображень в галузі комп'ютерного зору можна виділити такі типові завдання:

Класифікація об'єктів (object classification) – визначення до якого з заздалегідь визначених класів належить певний об'єкт у вхідному зображенні. Наприклад відповідь на питання чи присутній один чи декілька котів на вхідному зображення (важливий лише факт наявності об'єкту на фото без його відносних координат).

Локалізація об'єктів (object localization) – визначення координат та фактичних розмірів (зазвичай визначають координати центру прямокутника та його довжину та ширину) обмежувального прямокутника (bounding box), що обрамляє собою об'єкт у вхідному зображенні. Розширюючи попередній приклад, тепер необхідно дати відповідь на питання чи присутній кіт на зображенні, а також де він знаходиться.

Виявлення об'єктів (object detection) – комбінація локалізації та класифікації, що полягає в пошуку об'єктів на зображенні, котрі належать до визначеної множини класів. Виявлення об'єктів зазвичай відбувається одночасно із їх класифікацією.

Сегментація об'єктів (object segmentation) – ідентична до виявлення об'єктів задача, але при локалізації замість пошук обмежувальних прямокутників, відбувається пошук фактичних контурів об'єкта.

Демонстрація шуканих результатів для кожної із задач наведена на рисунку 1.

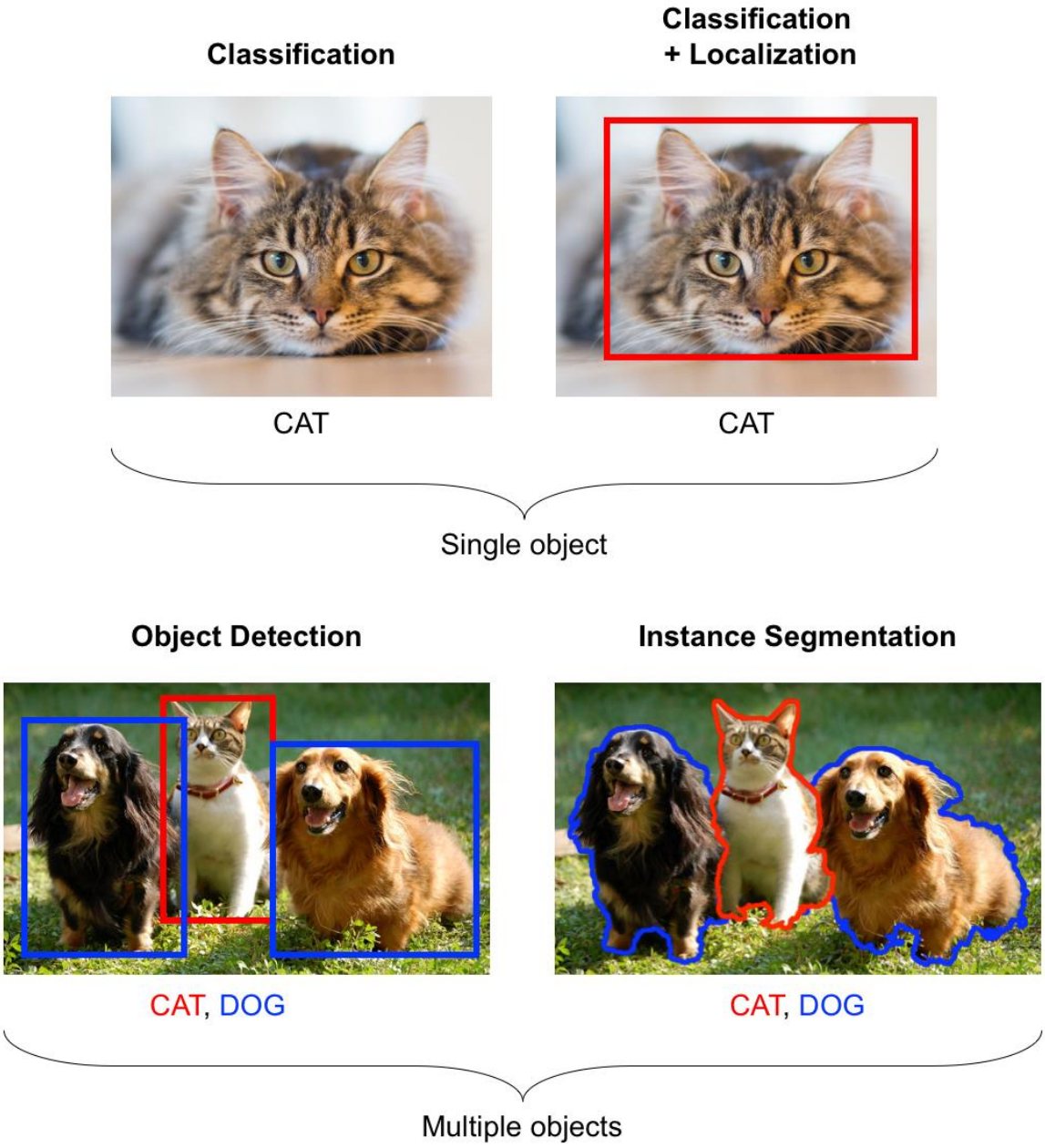

Рисунок 1. Демонстрація типових задач комп'ютерного зору

Задачу розпізнавання долонь можна класифікувати як задачу типу object detection або object segmentation з одним визначеним класом об’єктів – власне долоня будь-якої руки, або двома класами для випадків коли важливо мати можливість відрізнити долоні правої чи лівої руки.

1.2 Методи виявлення об’єктів на зображеннях

1.2.1 Визначення порогів

Найпростішим метод пошуку об’єктів на зображенні є метод визначення порогів (thresholding). В основі цього методу лежить перетворення вхідного зображення на вихідне бінарне по вибраному пороговому значенню певної ознаки зображення. Простим прикладом такого методу порогування за кольором, що відбувається наступним чином для зображень у форматі RGB: для кожного з трьох компонентів кольору пікселя вибирається поріг, чи частіше діапазон значень. Якщо значення відповідної компоненти пікселя вхідного зображення лежить у вибраному діапазоні, то відповідний піксель вихідного зображення стає білим, а якщо лежить за межами діапазону – чорним. На виході маємо три бінарні зображення, які за допомогою побітової операції І можна об’єднати в одне бінарне зображення, яке можна використати як маску для виділення об’єктів заданого кольору на зображенні (рисунок 2).

Рисунок 2. Приклад порогування

Зазвичай використовують зображення представлені в HSV замість RGB, оскільки таке представлення більше відповідає тому як людина сприймає колір, але алгоритм порогування при цьому залишається тим самим.

До більш просунутих реалізацій методів такого типу належать метод максимальної ентропії, метод Отсу (метод максимального відхилення) та кластеризація методом к-середніх. Перевагами таких методів є їх швидкість та ефективність обчислювання, а також відносна простота імплементації. Серед недоліків – низька точність при менш сприятливих умовах, таких як великий рівень шуму в зображеннях, нерівномірність освітлення, малий розмір шуканих об'єктів, а також у випадках коли внутрішньокласова дисперсія більша за міжкласову.

1.2.2 Масштабонезалежне перетворення ознак (SIFT)

SIFT – алгоритм виявляння об’єктів, який базується на виявленні та описі локальних ознак зображення за допомогою визначення особливих точок. Для визначення та опису таких точок в алгоритмах такого типу використовуються детектор та дескриптор. Детектор можна визначити як метод знаходження особливих точок на зображенні, а дескриптор – як ідентифікатор особливих точки, що дозволяє виділити її із загального набору особливих точок. Для особливих точок зазвичай виділяють такі вимоги: інваріантність щодо афінних перетворень, стійкість до шумів, глобальна унікальність та унікальність в певному локальному регіоні.

Для детектування особливих точок в SIFT використовується побудова піраміди гаусіанів та різниць гаусіанів. Гаусіаном називається зображення отримане після застосування до початкового зображення розмивання Гауса, а різницею гаусіанів – зображення отримане шляхом попіксельного віднімання одного гаусіана вихідного зображення від гаусіана з іншим радіусом розмиття.

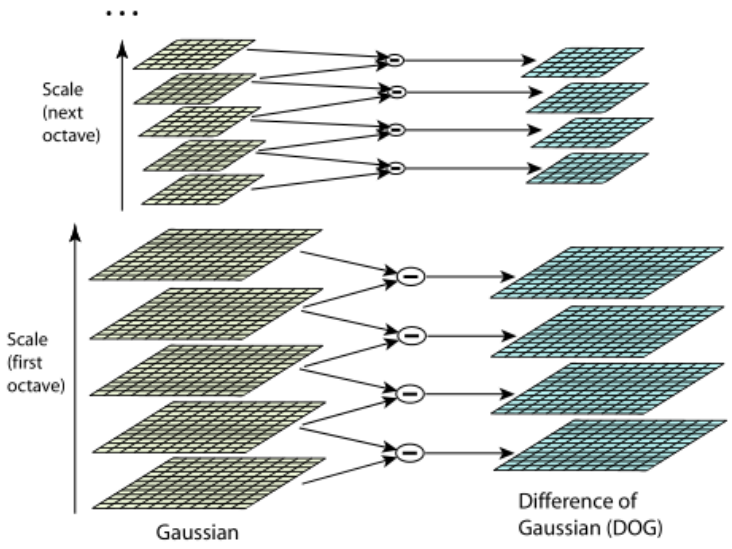

Рисунок 3. Зображення піраміди гаусіанів (зліва) та піраміди різниці гаусіанів (справа)

Точка вважається особливою, коли вона є локальним екстремумом різниці гаусіанів, тобто якщо значення різниці гаусіанів в цій більше за значення в інших точках визначеного регіону (рисунок 4). Для кожної особливої точки також визначається її напрямок базуючись на напрямках градієнтів сусідніх точок.

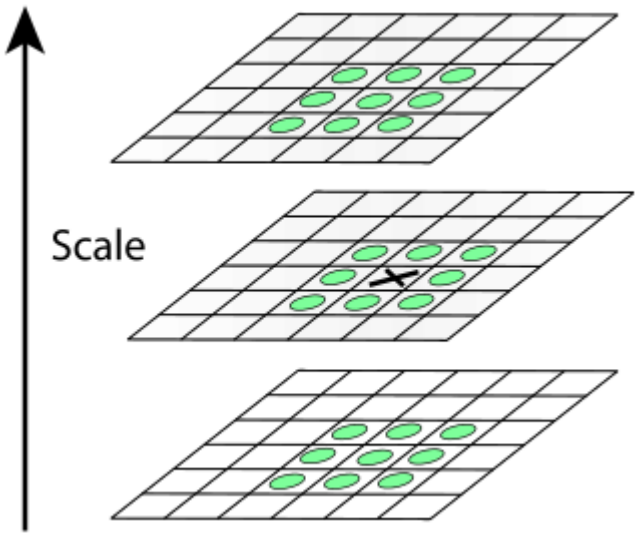

Рисунок 4. Схематичне зображення методу виявлення особливих точок

Дескриптором в SIFT є вектор. Для визначення дескриптору створюється набір гістограм напрямків на 4x4 сусідніх пікселях з 8 областями в кожній з них, отже вектор дескриптора містить у собі 128 елементів (4x4x8), що відповідають значенням гістограм з набору.

Для пошуку об’єктів на зображеннях спочатку знаходять ключові точки на еталонних зображеннях та визначають їх дескриптори. Потім на зображенні, на якому шукається об’єкт, необхідно також знайти ключові точки та дескрипотри, після чого порівняти їх з еталонним зображенням, і якщо достатня їх кількість співпадає, то можна зробити висновки про наявність об’єкта на зображенні.

Рисунок 5. Приклад роботи SIFT

Методи побудовані на детектуванні особливих точок зазвичай досить стійкі до поворотів та зміни масштабу об’єктів. Також такі методи мають змогу детектувати об’єкти, що частково загородженні. Недоліком є необхідність знаходження великої кількості особливих точок для ефективної роботу методу.

1.2.3 Методи машинного навчання

Машинне навчання – один із розділів науки про штучний інтелект, займається створенням алгоритмів, що надають комп'ютерам змогу «навчатися», тобто поступово покращувати продуктивність в поставленій задачі за рахунок обробки нових даних та попереднього досвіду, без строгого програмування. Трьома складовими машинного навчання є:

Дані – набір інформації на якому базується навчання, чим більше набір даних для навчання, тим більш точними є висновки;

Ознаки – ознаки об'єктів на які комп'ютеру треба звертати увагу при навчанні. Правильно обрані ознаки дозволяють спростити моделі, зменшити час їх навчання та знизити перенавчання;

Алгоритми – власне підхід до навчання моделей.

Можна виділити три основних категорії способів навчання:

Навчання з учителем – при такому навчання машині «вчителем» надаються приклади входів та виходів, а її метою при цьому є створення загального правила, що описує взаємозв'язок між входами та виходами.

Навчання без учителя – передбачає, що машині не надаються «правильні відповіді» (тобто приклади бажаних виходів) та стоїть мета виявити взаємозв'язки для наданих даних без втручання з боку «учителя».

Навчання з підкріпленням – при взаємодії з певним середовищем програмний агент виконує дії, за які може отримати винагороду або штраф. Метою є навчитися досягнути певної мети, при цьому максимізуючи сукупну нагороду.

Одним із видів машинного навчання є навчання штучних нейронних мереж (artificial neural networks). Обробка даних відбувається групами штучних нейронів із конективістським підходом до обчислень. Нейронні мережі дозволяють моделювати складні взаємозв'язки між входами та виходами для виявлення закономірностей в наборах даних. Сучасним розвитком нейронних мереж є глибинні ШНМ та використання глибинного навчання. Глибинні мережі містять велику кількість прихованих шарів та показують чудові результати при застосуванні в галузях комп'ютерного зору та обробки природної мови.

1.3 Нейронні мережі для виявлення та класифікації об'єктів

1.3.1 Загальне визначення нейронної мережі

Нейронна мережа це структура, що складається із нейронів, кожен з яких може отримувати певні дані на вхід та генерувати вихідні дані, які можуть бути надані іншим зв'язаним нейронам. Кожен із зв'язків між нейронами має визначену вагу, яку можна інтерпретувати як ступінь важливості даних, що надходять до нейрона по цьому зв'язку. Тобто структура нейронної мережі є зваженим, орієнтованим графом.

Дані на виході кожного нейрона визначаються на основі ваг вхідних зв'язків між ним та попередніми нейронами, а також функцією активації (рисунок 6).

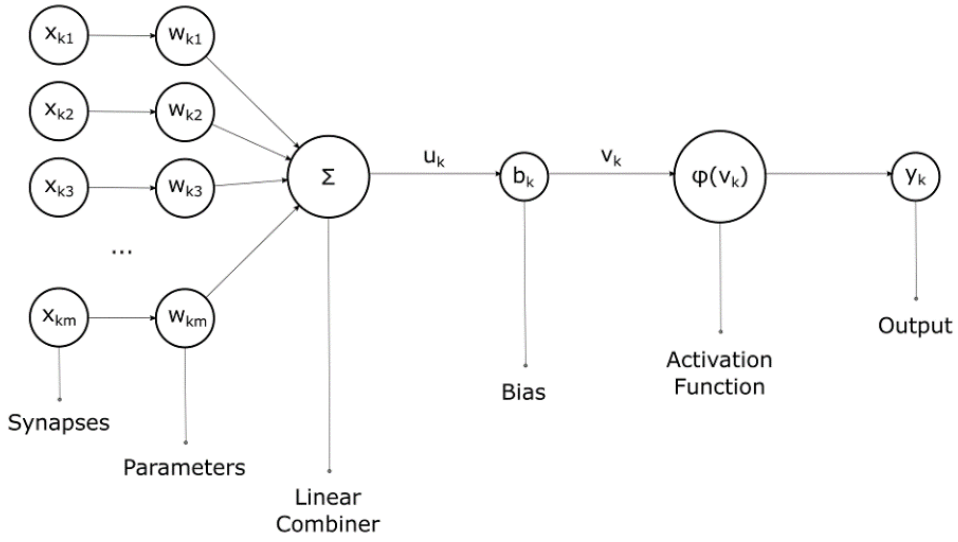

Рисунок 6. Модель нейрона

Зазвичай нейрони об’єднані в шари, при чому нейрони одного шару між собою не зв’язані та мають зв’язки лише з нейронами у шарах, що знаходяться безпосередньо перед або за ними. Типова мережа складається з трьох частин:

- вхідного шару, який отримує на вхід зовнішні дані (наприклад зображення)
- вихідного шару, який генерує кінцевий результат роботи мережі
- прихованих шарів

Дані проходять від першого (вхідного) до останнього (вихідного) шару, через приховані шари, якщо такі наявні.

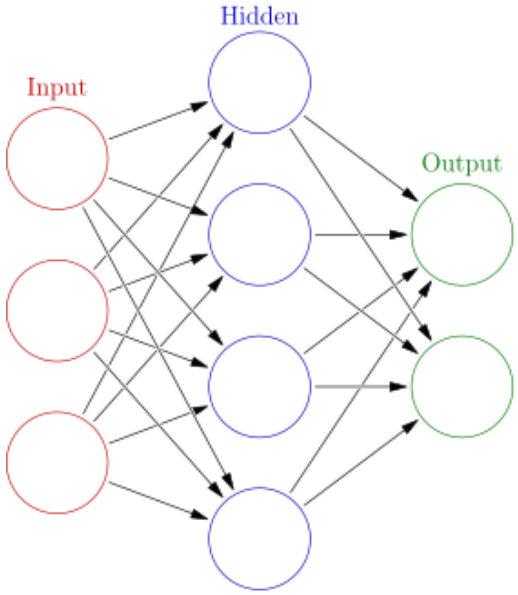

Рисунок 7. Шари нейронної мережі

1.3.2 Навчання нейронних мереж

Здатність до навчання є, можливо, найважливішою властивістю нейронних мереж. При навчанні мережі значення вагів зв'язків між нейронами змінюються з метою підвищення точності передбачень. Це відбувається шляхом мінімізації похибки між фактичним очікуваним результатом та результатом, що передбачила мережа за допомогою функції втрат (loss function). Навчання триває доти, доки при обробкою мережею нових вхідних даних зменшується значення функції витрат. Наприклад, поширеним методом навчання мереж є метод зворотного поширення помилки.

1.3.3 Згорткові нейронні мережі

Для аналізу зображень зазвичай використовується клас мереж відомий як згорткові нейронні мережі (ЗНМ, convolutional neural networks, CNN). Згортковими називають мережі, що серед прихованих шарів містять так звані

згорткові шари (convolutional layers), що виконують операції згортки. Такі шари дозволяють значно зменшити кількість параметрів які мережі треба навчити, що є дуже важливим при обробці зображень з великою роздільною здатністю.

При проході через шар згортки до вхідного зображення застосовується фільтр (ядро згортки), що дозволяє виділити із зображення певну ознаку (наприклад контур об’єкта чи навіть певну його частину або сам об’єкт у цілому). Фільтр являє собою певну матрицю ваг, на яку поелементно множиться фрагмент вхідного зображення. Фільтр проходить зображення в ширину та висоту з визначеним кроком, при цьому результати добутку записуються до карти ознак, які використовуються в якості вхідного зображення для наступних шарів. Самі фільтри формуються мережею під час її навчання.

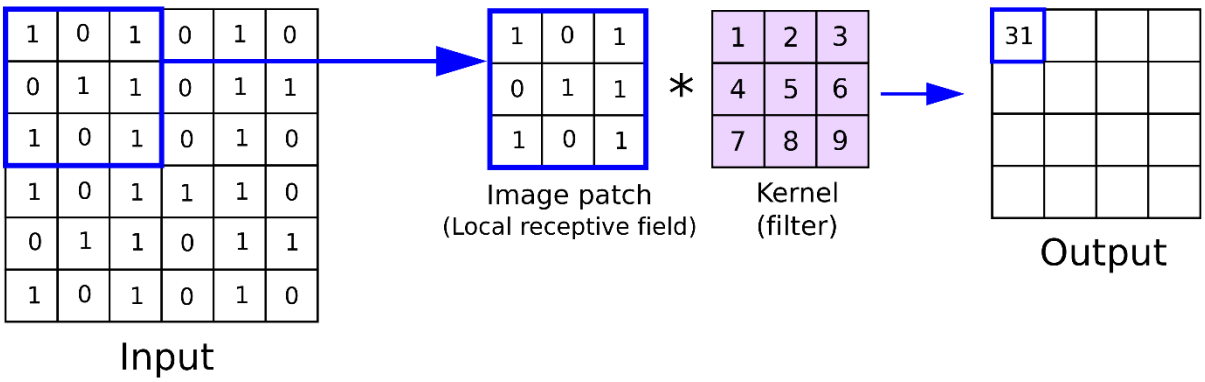

Рисунок 8. Приклад операції згортки

Окрім згорткових шарів, мережі також містять шари агрегації (pooling layers), котрі покликані нелінійно зменшити розмірність вхідної карти ознак але при цьому зберегти та додатково підсилити виділені попередніми шарами ознаки. Для зменшення використовуються різні методи, наприклад, пошук

максимальних чи середніх значень елементів (Max Pooling та Average Pooling відповідно).

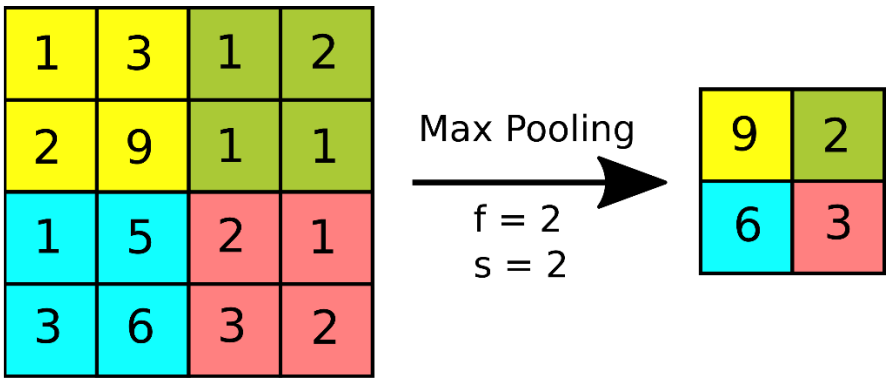

Рисунок 9. Приклад максимізаційного агрегування

Після комбінації згорткових шарів та шарів агрегації в згорткових мережах слідують повноз'єднані шари (fully connected layers). Нейрони в таких шарах з'єднані з усіма нейронами попереднього шару. Задачею повноз'єднаних шарів згорткової мережі є, найчастіше, класифікація об'єктів.

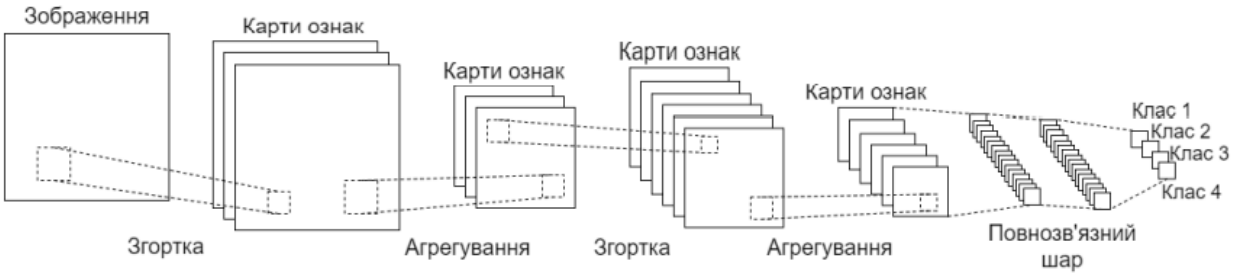

Рисунок 10. Загальна структура згорткової нейронної мережі

На основі ЗМН побудовано ряд популярних мережевих архітектур для розпізнавання та класифікації об'єктів. Серед таких архітектур найвідомішими є Faster R-CNN, YOLO та SSD. Проведемо порівняльний аналіз наведених

архітектур для того, щоб виявити яка з них найбільше підходить для поставленої задачі.

1.3.4 Faster R-CNN (Faster Region-based CNN)

Представимо процес виявлення об’єктів наступним чином:

З вхідного зображення ми виділяємо «вікна» певного розміру для кожної можливої позиції. Частину зображення, що видно через кожне окреме «вікно» далі подаємо до класифікатора, що розраховує імовірність знаходження об’єкту якось з попередньо визначених класів у виділеній частині зображення (або класифікації даної частини зображення як частини фону у разі відсутності жодних об’єктів відомих класів). Враховуючи необхідність застосування «вікон» багатьох розмірів та «вікон» з різним співвідношенням сторін, їх кількість є великою та зростає з підвищенням роздільної здатності зображень. Для класифікаторів, що використовують більш класичні методи комп’ютерного зору (використання гістограми напрямлених градієнтів), апаратні затрати для їх застосування до кожного з «вікон» є прийнятними. Якщо ж використовується класифікатор в основі якого лежить ЗНМ, то обчислення такої кількості «вікон» є непрактичним, або навіть не завжди можливим.

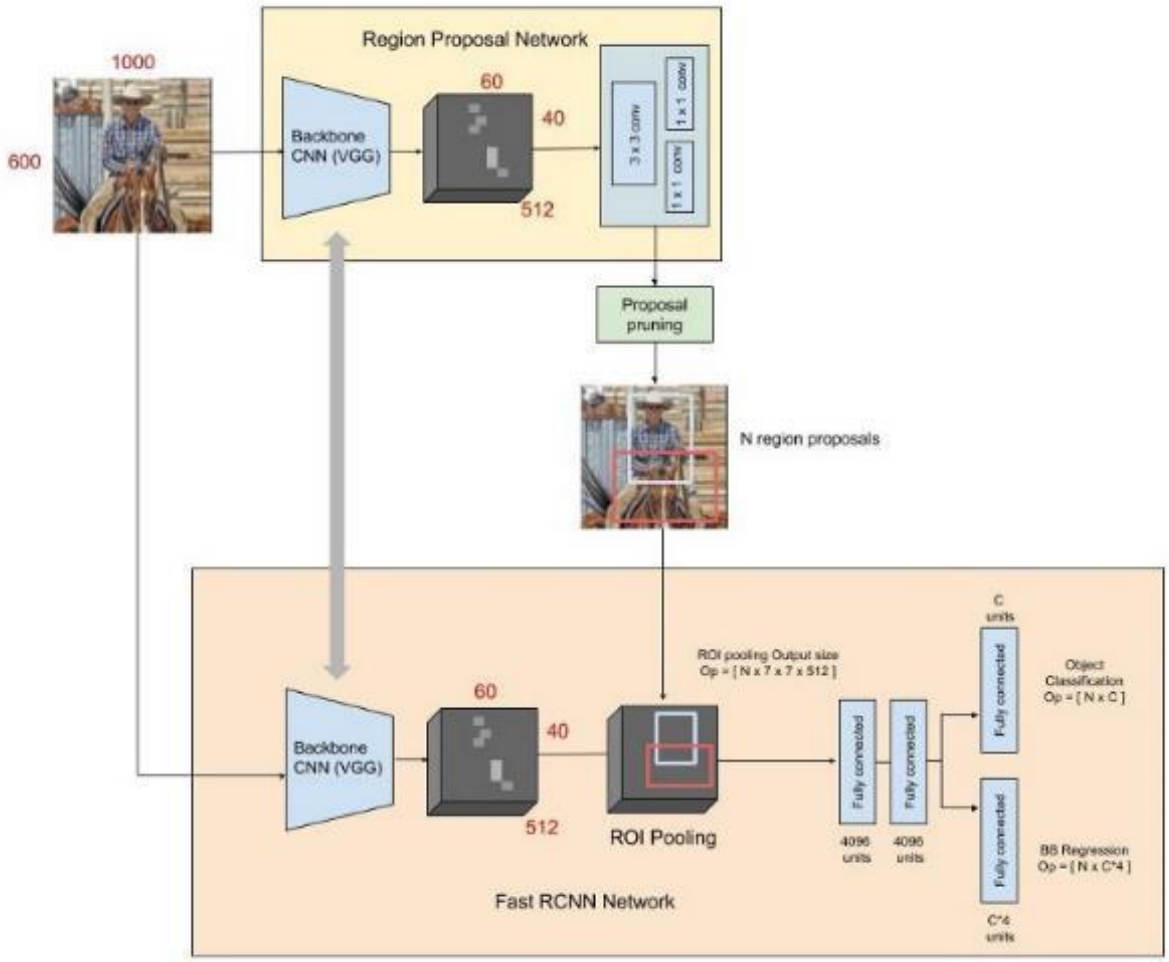

Рисунок 11. Приклад структури FR-CNN

Рішенням цієї проблеми й виступає FR-CNN.

Такі мережі складаються з двох частин (приклад такої мережі наведений на рисунку 11):

- Region Proposal Network (RPN)
- Detector

Замість обробки кожного можливого «вікна» RPN пропонує лише певну кількість «найцікавіших» регіонів зображення, до кожного з яких потім застосовується класифікатор (Detector)

Обидва компоненти такої мережі по суті є мережами самі по собі, та вимагають спеціального підходу до їх тренування, а також, що очевидно, більших часових витрат.

В результаті отримуємо мережу з високою точністю розпізнавання та локалізації об’єктів, але досить невеликою швидкодією (порівняно з архітектурами SSD та YOLO в обох аспектах)

1.3.5 YOLO (You Only Look Once)

Принцип роботи такої мережі кардинально відрізняється від принципу роботи FR-CNN, перш за все через те, що дана мережа є одно-етапною а передбачення обмежувальних прямокутників (bounding boxes) та власне детектування об’єктів в них виконується однією мережею та в один прохід.

Кожне зображення ділиться сіткою певної розмірності $M \times M$, після чого кожна сітка передбачає N-у кількість обмежувальних прямокутників для кожного з яких також надається оцінка впевненості (confidence score) щодо точності обмежувального прямокутника, та власне наявності в ньому об’єкту (незалежно від його класу). Також зразу надається класифікаційна оцінка (classification score) відносно кожного передбаченого класу, що означає ймовірність знаходження об’єкту того чи іншого класу в прямокутнику. Всього мережею передбачається $M \times M \times N$ обмежувальних прямокутників, кожен з двома відповідними оцінками confidence та classification. Далі ці прямокутники фільтруються пізніми шарами мережі, за величинами оцінок. В результаті отримуємо найбільш імовірні прямокутники для об’єктів всередині поданого зображення та їх класи.

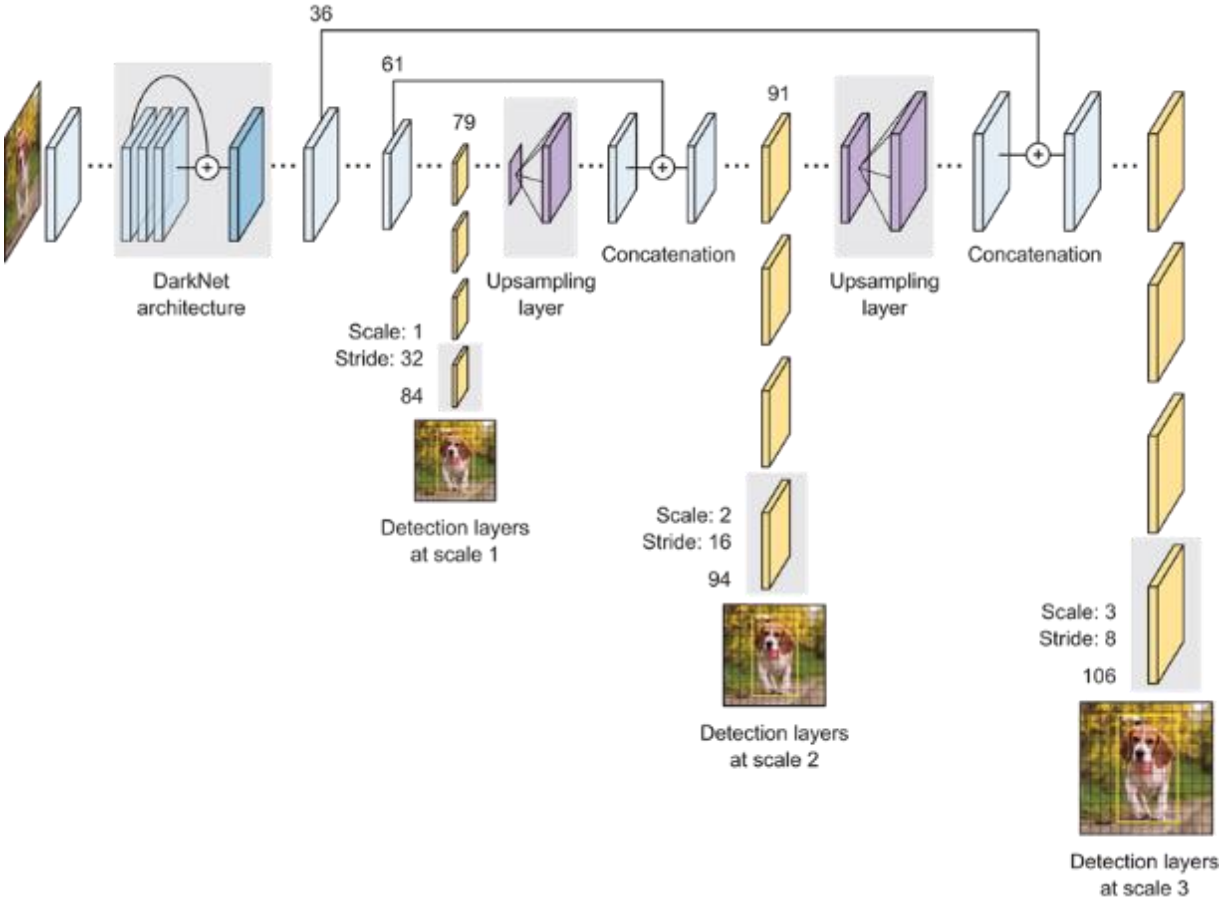

Рисунок 12. Схематичне зображення архітектури YOLO

Отже через те, що мережа виконує всю роботу за один прохід, а також через те, що мережа розглядає вхідне зображення повністю, швидкодія мережі значно підвищується порівняно з попередньо розглянутою FR-CNN. Також через відносно простішу архітектуру час тренування також зменшується. Тож головною превагою YOLO є її висока швидкодія, що є достатньою для виконання детекції навіть у режимі реального часу. Недоліком ж є порівняно знову з FR-CNN знижена точність.

Схематичне представлення архітектури даної мережі наведене на рисунку 12.

1.3.6 SSD (Single-Shot Detector)

SSD мережі в цілому схожі на YOLO мережі, бо також є одно-етапними, а принципові відмінності полягають в архітектурі шарів цих мереж (приклад такого порівняння наведено на рисунку 13)

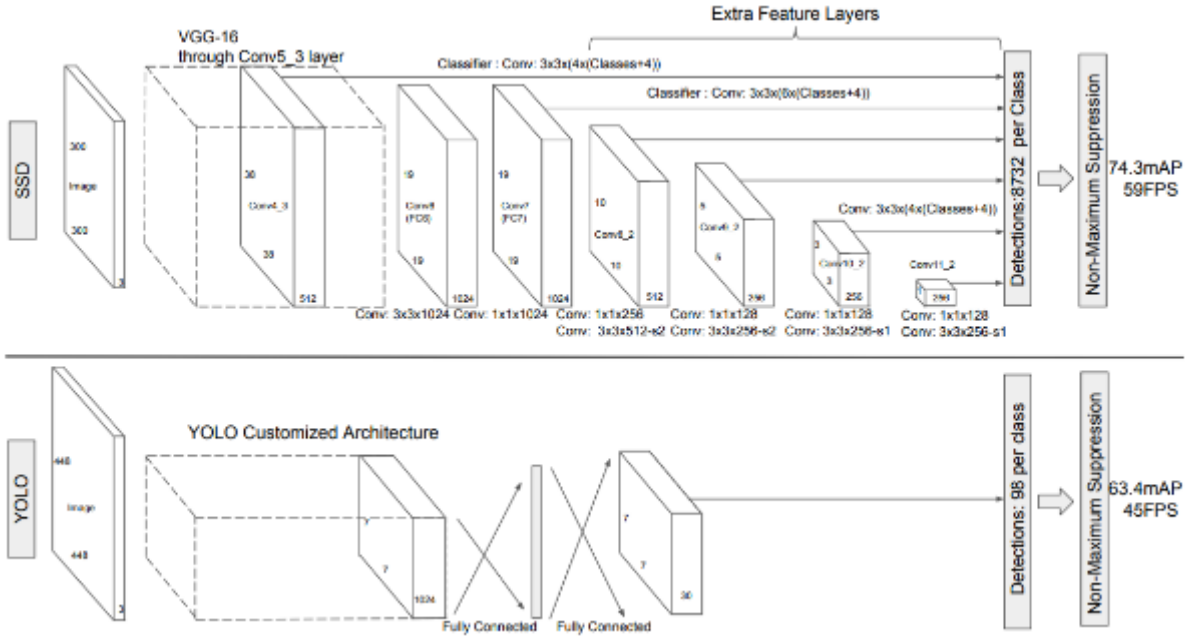

Рисунок 13. Схематичне порівняння архітектур SSD та YOLO

Одна з переваг SSD – модульна архітектура. Мережа SSD складається з backbone мережі, що використовується в якості feature extractor. Backbone мережа зазвичай являє собою раніше натреновану мережу для класифікації зображень (наприклад VGG чи ResNet), з видаленими з неї останніми шарами класифікаторами. До backbone мережі потім під’єднуються декілька шарів, що й власне відповідають за передбачення обмежувальних прямокутників та класів об’єктів на зображеннях, на основі feature maps виданих backbone

мережею. В іншому принцип роботи мережі схожий на принцип роботи раніше розглянутої YOLO.

Архітектура SSD дозволяє цим мережам бути більш точними за YOLO (але все ще менш точними за FR-CNN), при чому зберігаючи достатню порівняну з YOLO швидкодію.

ВИСНОВОК ДО РОЗДІЛУ 1

В даному розділі описана проблематика задачі розпізнавання об’єктів на зображеннях та розглянуті методи для вирішення задачі детектування об’єктів на зображеннях.

Розглянуті приклади класичних алгоритмічних методів, а також приклади сучасних методів, що базуються на застосуванні штучних нейронних мереж. Для кожного методу описані його особливості, основні переваги та недоліки.

РОЗДІЛ 2. ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ЗАСТОСУНКУ

2.1 Вибір мови програмування

Перед початком розробки застосунку необхідно визначити ряд ключових пунктів. Перш за все, потрібно обрати мову програмування для написання застосунку. Серед мов, що найчастіше використовуються в галузі машинного та глибинного навчання можна виділити такі: C++, Java, Python, JavaScript. З точки зори швидкості та ефективності розробки, найкращим вибором з наведених мов є Python або JavaScript, бо обидві ці мови є надзвичайно гнучкими, а синтаксис зручним та простим для розуміння. Такі переваги дозволяють швидко створювати прототипи застосунків, а також зменшити час на їх подальші ітерації.

Обрана була мова програмування Python з декількох причин: це одна з найпопулярніших мов програмування в світі в цілому, та найбільш широко застосована мова у галузі глибинних нейронних мереж. Висока розповсюдженість означає не менш високу підтримку зі сторони наукової спільноти, тому для Python існує значна кількість бібліотек та фреймворків, що полегшують як розробку в цілому, так і розробку штучного інтелекту.

Також Python може бути більш швидким за JavaScript, оскільки остання переважно застосовується для розробки Web-застосунків, і тому є інтерпретованою мовою програмування, що додатково вносить накладні витрати при роботі застосунків й тому зменшує їх загальну продуктивність. Хоча Python в загальному вигляді також є інтерпретованою мовою, використання бібліотек таких як, наприклад, Cython, дозволяє компілювати Python напряму в C/C++ або в ряд інших статично компільованих мов, що в деяких випадках значно підвищує швидкодію.

2.2 Вибір бібліотек та фреймворків

Після обрання мови програмування, наступним кроком необхідно вибрати фреймворк для проектування та тренування обраної мережевої архітектури в екосистемі обраної мови програмування.

Серед фреймворків для глибинного навчання, що мають інтерфейс для мови Python найпопулярнішими є TensorFlow, PyTorch та Caffe тому саме їх і розглянемо.

TensorFlow – відкрита бібліотека для машинного навчання розроблена компанією Google та випущена у 2015 році. Бібліотека написана переважно мовою C++ але надає API для використання в середовищі мови Python а також багатьох інших. TensorFlow підтримує паралельне виконання обчислень на декількох CPU або GPU для прискорення навчання (також підтримує архітектуру CUDA та роботу з тензорними ядрами на останніх поколіннях GPU Nvidia).

В суміжності з Keras, TensorFlow надає зручний високорівневий програмний інтерфейс для проектування нейронних мереж, надаючи користувачеві готові реалізації різних складових мереж. Також окрім Keras для TensorFlow доступні й інші бібліотеки, що полегшують різні аспекти взаємодії з фреймворком, таких як, наприклад навчання мереж. Крім бібліотек, спільнотою також створено велику кількість готових моделей різних архітектур нейронних мереж для використання при розробці власного застосунку.

PyTorch – ще один популярний фреймворк для машинного та глибинного навчання розроблений одним із підрозділів Facebook. В цілому пропонує аналогічні до TensorFlow можливості для високорівневого проектування нейронних мереж та прискорення навчання на CPU та GPU, в

тому числі з використанням архітектури CUDA. Окрім простоти розробки, серед додаткових переваг можна відокремити більш глибоку інтеграцію з Python, та більшу кількість доступних моделей створених науковою спільнотою, а серед недоліків – відсутність власних засобів візуалізації таких як TensorBoard.

Caffe – фреймворк глибинного навчання призначений переважно для вирішення задач в галузі комп'ютерного зору. Дозволяє швидко проектувати та тренувати нейронні мережі для обробки зображень (а саме згорткових), але погано пристосований для створення мереж інших типів. Серед іншх недоліків є погана розширюваність, необхідність використовувати мови C++ та CUDA у випадках коли необхідне прискорення навчання за допомогою GPU; а також низька підтримка спільнотою та погана робота з великими мережами, такими як ResNet.

В результаті, обраний фреймворк TensorFlow, оскільки він в поєднанні з Keras представляє зручний, високорівневий програмний інтерфейс для побудови та тренування глибинних нейронних мереж, а також для застосунку натренованих моделей в різноманітних прикладних задачах.

2.3 Вибір набору даних

Наступним кроком є вибір вже готового чи створення власноруч придатного набору даних на якому відбуватиметься тренування нейронної мережі. Основними необхідними параметрами такого набору даних є перш за все наявність анотацій щодо положення долонь на зображеннях у вигляді контуру, маски або обмежувального прямокутника та висока якість самих зображень. Також важлива наявність достатньої кількості зображень в наборі та їх різноманітність: до набору мають входити зображення долонь в різному навколишньому освітленні та з різноманітними фонами. Окрім цього важливо

мати різні ракурси зображень та велику кількість позицій рук та долонь. Бажаною, але необов’язковою є наявність класифікації долонь на долоні правої чи лівої руки.

Під час пошуку серед відкритих наборів даних, знайдений такий, що відповідає усім поставленим вимогам – EgoHands Dataset. Цей набір базується на 48 відео, знятих за допомогою Google Glass, на яких зображено, від першої особи, комплексну взаємодію між двома людьми, наприклад гру в шахи. Всього в наборі налічується позначених 4800 кадрів (по 100 вибраних кадрів із кожного з 48 відео) на яких зображено 15053 долоні. Анотації представлені у файлі Matlab. Основна правда (ground truth) подається у вигляді піксельної маски для кожної долоні в кадрі, також надається інформація про те, чи це долоня лівої руки або правої та кому з людей в кадрі ця долоня належить – спостерігачу чи комусь іншому. Розмірність кожного кадру становить 720x1280px, а самі вони збережені у форматі Jpeg. На рисунку 14 для прикладу наведені декілька зображень з набору даних на яких піксельними масками різних кольорів (в залежності від класу долоні) позначені долоні.

Рисунок 14. Приклад зображень з набору даних

Головними перевагами цього набору даних є велика кількість локацій на яких відбувалась зйомка (при цьому наявні локації як всередині приміщень, так і на вулиці, що означає хорошу різноманітність в освітленні та фоні зображень), а також велика кількість позицій рук під час взаємодії. Ці ключові переваги роблять його чудовим вибором для тренування нейронної мережі для виявлення долонь на зображеннях.

2.4 Вибір архітектури нейронної мережі

Для вибору підходящої архітектури необхідно порівняти кількісні характеристики швидкості та точності стандартних реалізацій раніше наведених ЗНМ при використанні ряду популярних наборів даних для тренування та подальшої оцінки мереж.

Для набору даних PASCAL VOC 2012 результати наступні:

Мережа	mAP (%)
Faster R-CNN	75.9
SSD	72.4
YOLO	52.7

Для набору MS COCO:

Мережа	mAP (%)
Faster R-CNN	36.2
SSD	31.2
YOLO	21.6

Параметр mAP (mean Average Precision) означає середню показану точність мережі серед об’єктів різних класів та розмірів.

Заміри швидкодії:

Мережа	FPS
Faster R-CNN	7
SSD	59
YOLO	155

Параметр FPS (frames per second) означає кількість оброблених мережею зображень за одну секунду часу.

Задача виявлення долонь на зображеннях набуває найбільшої цінності в контексті оброблення зображень режимі реального часу, наприклад під час сеансу відео зв’язку. З огляду на це мережі типу FR-CNN не можуть бути застосовані для вирішення цієї задачі. Між SSD та YOLO більш оптимальним вибором є мережі типу SSD через їх хороше співвідношення швидкості та точності, тому саме SSD обрана як основа для подальшої розробки програмного застосунку.

ВИСНОВОК ДО РОЗДІЛУ 2

В другому розділі розглянутий ряд популярних мов програмування, що застосовуються для розробки нейронних мереж. Серед цих мов обрано Python.

В середовищі мови порівняні декілька фреймворків для роботи з нейронними мережами, з яких для подальшої реалізації застосунку обраний фреймворк TensorFlow.

Обрана оптимальна архітектура згорткової нейронної мережі – SSD, для вирішенні поставленої задачі: детектування долонь на зображеннях в режимі реального часу. Для реалізації мережі обраний підходящий набір даних.

РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ВИЯВЛЕННЯ ДОЛОНЬ НА ЗОБРАЖЕННЯХ НА БАЗІ ЗМН ТИПУ SSD

3.1 Реалізація моделі мережі

Ключовим кроком при створенні застосунків, що використовують штучний інтелект є розробка моделі нейронної мережі. В одному з попередніх розділів була обрана архітектура SSD. Для підвищення точності детектування моделі та пришвидшення її розробки та тренування, використана готова, натренована модель для розпізнавання об’єктів, а до неї застосований прийом під назвою перенесення навчання (transfer learning). Перенесення навчання дозволяє використати готову модель, що натренована на великому, загальному наборі даних (наприклад ImageNet) та перенести набуті нею знання на менший набір даних, для вирішення специфічної задачі. Такий підхід є особливо корисним для задач комп’ютерного зору, оскільки знання моделі про загальні риси об’єктів, такі як їх контури наприклад, можна використати незалежно від множини класів, що використовується.

В якості базової моделі використана модифікована SSD із backbone мережею ResNet. Початково модель створена за допомогою TensorFlow Object Detection API натренована за допомогою набору даних COCO. Для подальшою роботи з цією моделлю також використовується бібліотека TensorFlow Object Detection API, що надає інтерфейс для швидкого створення нейронних мереж для детектування об’єктів, а також надає інструменти для їх тренування.

Обрана мережа містить шари класифікатори (classification head) та шари предиктори для обмежувальних прямокутників (box prediction head). Оскільки обраний набір даних містить лише один клас замість 90 класів, які може розпізнавати модель, для моделі створений новий набір шарів класифікаторів та натренований з нуля на новому наборі даних, тобто набуті знання про класифікацію замінюються новими, більш специфічними. Шари box prediction,

натомість, дотреновані (fine-tuned) з використанням нових даних. Fine-tuning – один із способів виконання перенесення навчання, під час якого декілька верхніх шарів натренованої моделі розблоковуються та дотреновуються з дуже малою швидкістю навчання (learning rate).

Отже для початку необхідно завантажити збережену модель та ініціалізувати ваги її шарів.

```
model_config = config_util.get_configs_from_pipeline_file(pipeline_config)
model_config = model_config['model']
model_config.ssd.num_classes = 1
model_config.ssd.freeze_batchnorm = True
base_pretrained_model = model_builder.build(model_config=model_config, is_training=True)
```

Рисунок 15. Завантаження моделі

Метод build() дозволяє з наданої конфігурації (використана конфігурація наведена в Додатку Б) створити модель необхідної архітектури. Готова модель створена з TensorFlow Object Detection API поставляється у вигляді її конфігурації та збережених значень вагів. Варто зазначити, що шари класифікатори тепер мають іншу розмірність відносно початкової моделі, оскільки визначається лише один клас. Наступним кроком є власне відновлення вагів моделі.

```
box_predictor = tf.compat.v2.train.Checkpoint(
    _base_tower_layers_for_heads=base_pretrained_model._box_predictor._base_tower_layers_for_heads,
    _box_prediction_head=base_pretrained_model._box_predictor._box_prediction_head,
)
restored_model = tf.compat.v2.train.Checkpoint(
    _feature_extractor=base_pretrained_model._feature_extractor,
    _box_predictor=box_predictor)
ckpt = tf.compat.v2.train.Checkpoint(model=restored_model)
ckpt.restore(checkpoint_path).expect_partial()
```

Рисунок 16. Ініціалізація вагів

Натреновані ваги моделі TensorFlow зберігаються у форматі Checkpoint. Як зазначено раніше, відновлюються усі шари окрім шарів класифікаторів. Клас Checkpoint надає інтерфейс для збереження до локального файлу та подальшого завантаження параметрів моделі. Для відновлення збережених параметрів використовується метод restore(). Після виконання цього методу необхідні ваги моделі проініціалізовані, а сама модель готова до тренування. Звіт щодо шарів моделі наведений в Додатку А.

3.2 Підготовка набору даних

Як зазначено раніше, анотації до обраного набору даних подаються в форматі Matlab, тому наступним кроком є підготовка даних: перетворення анотацій до формату зрозумілого TensorFlow та додатковий поділ набору на навчальну, валідаційну та тестову вибірки. Також доцільно виконати аугментацію даних для розширення навчальної вибірки. Аугментація дозволяє штучно збільшити кількість зображень за допомогою їх модифікацій, наприклад випадкових поворотів, змін контрастності чи яскравості, або обрізання зображень.

Спочатку необхідно привести формат ground-truth до формату, який приймає на вхід модель, тобто координати bounding box замість контурів. Кожне зображення перейменовано з урахуванням номеру кадру та локації, щоб забезпечити унікальність назв. Координати bounding box отримані шляхом пошуку мінімальних та максимальних значень по осям, серед координат точок, що становлять контур долоні.

```
for point in pointlist:
    if len(point) == 2:
        x = int(point[0])
        y = int(point[1])

        if findex == 0:
            min_x = x
            min_y = y
        findex += 1
        max_x = x if (x > max_x) else max_x
        min_x = x if (x < min_x) else min_x
        max_y = y if (y > max_y) else max_y
        min_y = y if (y < min_y) else min_y
```

Рисунок 17. Визначення координат крайніх точок обмежувального прямокутника

Для роботи з Matlab файлами з Python використана бібліотека SciPy та її метод loadmat() для відкриття файлу та представлення даних, що лежать в ньому, у вигляді словника. Визначенні координати разом з назвою файлу, розмірністю та класом зображення (в цьому випадку він лише один - hand) записані до csv таблиці. Такий формат дозволяє зручний доступ до анотацій зображень у подальшому.

```
def save_csv(csv_path, csv_content):
    with open(csv_path, 'w') as csvfile:
        wr = csv.writer(csvfile)
        for i in range(len(csv_content)):
            wr.writerow(csv_content[i])
```

Рисунок 18. Збереження до csv таблиці

filename	width	height	class	xmin	ymin	xmax	ymax
CARDS_COURTYARD_B_T_frame_0011.jpg	1280	720	hand	647	453	824	551
CARDS_COURTYARD_B_T_frame_0011.jpg	1280	720	hand	515	431	622	543

Рисунок 19. Приклад збережених анотацій

Також додатково для перевірки правильності визначених параметрів обмежувальних прямокутників, вони були візуалізовані поверх деяких зображень.

Рисунок 20. Приклад bounding box

Далі необхідно завантажити зображення, об’єднати їх з анотаціями та перетворити на вхідні тензори для навчання моделі. Також необхідно розділити набір даних на навчальну та тестову вибірки. Для цього використовується метод split_set_test_eval_train(), що випадковим чином виділяє зображення для різних вибірок в заданому співвідношенні (в даному випадку обрані 85% навчальних зразків та 15% тестових зразків). Випадковість вибору зразків дозволяє забезпечити наявність зображень з кожної із локацій в навчальній та тестовій вибірках. Також в розділених вибірках немає однакових зображень.

Як зазначено раніше до навчальної вибірки застосована аугментація, таким чином створені три копії навчальної вибірки: перша копія залишається без змін; до другої копії застосована аугментація у вигляді випадкового горизонтального віддзеркалення; до третьої копії застосоване випадкове обрізання зображення. Горизонтальне віддзеркалення дозволяє симулювати ситуації, в яких об’єкт сфотографований з різних ракурсів. Обрізання зображень симулює ситуації коли об’єкт не повністю знаходиться в кадрі, тобто коли видно лише його частину, або коли обрізається фон на якому знаходиться об’єкт, або його частина, таким чином змінюючи контекст. Приклад імплементації функції віддзеркалення зображення показано на рисунку 21. Використані методи із модуля tf.image. Разом з віддзеркаленням самого зображення, також важливо віддзеркалити координати bonding box для того, щоб вони й надалі відображали правильну інформацію. Приклад зображень після аугментації наведено на рисунку 22.

```
def flip_horizontal(image, boxes):
    image_flipped = tf.image.flip_left_right(image)

    ymin, xmin, ymax, xmax = tf.split(value=boxes, num_or_size_splits=4, axis=-1)
    flipped_xmin = tf.subtract(1.0, xmax)
    flipped_xmax = tf.subtract(1.0, xmin)
    flipped_boxes = tf.concat([ymin, flipped_xmin, ymax, flipped_xmax], axis=-1)

    return image_flipped, flipped_boxes
```

Рисунок 21. Метод для віддзеркалення зображення

Рисунок 22. Результат віддзеркалення

Обрізання зображень виконано за допомогою методу random_crop_image() із модулю Object Detection API, який повертає модифіковані зображення та bounding box.

Останнім кроком перед подачею даних моделі є зміна розмірності зображень та нормалізація їх кольорових компонент. Модель має визначену вхідну розмірність зображень, що становить 640x640. Значення кожної з трьох кольорових компонент (формат зображень RGB) для кожного пікселя зображення нормалізовано з діапазону [0, 255] до діапазону [0, 1]. Дані операції виконуються методом preprocess().

```
image, shapes = base_pretrained_model.preprocess(tf.zeros([1, 640, 640, 3]))
```

Рисунок 23. Приклад попередньої обробки даних для моделі

Отримані копії з різними аугментаціями об’єднані в одну тренувальну вибірку з якої вилучені однакові зображення (однакові зображення могли виникнути, оскільки застосування аугментацій здійснювалось випадковим чином, отже могли винукнути ситуації коли аугментації призводили до створення точної копії вхідного зображення)

3.3 Навчання моделі

Перед початком навчання необхідно визначити ряд його основних параметрів: функцію витрат, швидкість навчання, оптимізатор, кількість епох навчання та розмір партії.

Функція витрат (loss function) визначає розмір похибки між передбаченням моделі та ground truth, тобто оцінює наскільки точні

передбачення моделі. Задача навчання полягає в оптимізації параметрів моделі для мінімізації значення функції витрат. В архітектурі SSD присутні два типи функції витрат: localization loss та classification loss, що характеризують наскільки добре модель передбачає bounding box та клас об'єкту відповідно. В якості localization loss використовується Smooth L1-loss:

$$L = \begin{cases} |x|, & \text{якщо } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2, & \text{якщо } |x| \leq \alpha \end{cases},$$

де α – гіперпараметр, значення якого зазвичай становить 1, а x – значення різниці передбаченого результату та очікуваного. Classification loss представляє собою Softmax-loss.

Оптимізатори (optimizer) – це алгоритми, що використовуються для зміни параметрів моделі під час навчання з метою мінімізації функції витрат.

Швидкість навчання (learning rate) визначає наскільки швидко налаштовуються параметри моделі під час її навчання. Чим вище значення, тим швидше зазвичай проходить навчання, але при цьому параметри моделі налаштовуються менш точно.

Значення кількості епох визначає кількість ітерації в яких модель виконує прямий та зворотній проходи через усю множину навчальних даних. Після кожної епохи точність передбачень моделі збільшується: занадто велика кількість епох може призвести до перенавчання (overfitting), а занадто мала – до недонавчання.

Розмір партії (batch size) визначає кількість навчальних зразків, що оброблюються моделлю за один крок. Розмір партії зазвичай менший за розмір всього набору даних, оскільки завантаження усього набору до оперативної пам'яті часто є проблематичним або неможливим через недостачу пам'яті.

Такі параметри як швидкість навчання, розмір партії та кількість епох часто підбираються експериментально, оскільки різні їх комбінації забезпечують різну кінцеву точність моделей.

Для тренування моделі створено метод fit(), що інкапсулює в собі створення циклу для тренування моделі з визначеними раніше розглянутими параметрами. На кожній ітерації циклу тренування завантажуються партії зразків визначеного розміру з навчальної вибірки та відбуваються прямий та зворотній проходи.

```
def fit(ground_truth_labels,
        model,
        optimizer,
        batch_size,
        epochs):
```

Рисунок 24. Сигнатура методу fit()

В якості оптимізатора використовується стохастичний градієнтний спуск (stochastic gradient descent, SGD), а саме його реалізація в TensorFlow.

```
optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)
```

Рисунок 25. Створення оптимізатора

Кожну ітерацію тренування можна описати наступним чином: спочатку дані з партії за допомогою методу preprocess() перетворюються на тензори необхідної розмірності. Попередньо оброблені дані далі подаються на вхід моделі, а модель в свою чергу передбачає результат. На основі результату моделі обчислюється значення функції витрат, після чого за допомогою оптимізатора визначається на скільки необхідно змінити значення ваг моделі. Приклад реалізації одної ітерації навчання наведено на рисунку 26.

```
with tf.GradientTape() as tape:
    preprocessed_images = tf.concat(
        [model.preprocess(image_tensor)[0]
         for image_tensor in image_tensors], axis=0)
    prediction_dict = model.predict(preprocessed_images, shapes)
    losses_dict = model.loss(prediction_dict, shapes)
    total_loss = losses_dict['Loss/localization_loss'] + losses_dict['Loss/classification_loss']
    gradients = tape.gradient(total_loss, vars_to_fine_tune)
    optimizer.apply_gradients(zip(gradients, vars_to_fine_tune))
return total_loss
```

Рисунок 26. Приклад навчальної ітерації

Після навчання отримуємо модель з налаштованими вагами яку можна зберегти для подальшого використання. Для тренування моделі використані такі параметри: кількість епох – 300, розмір партії – 64, швидкість навчання – 0.0001.

3.4 Оцінка натренованої моделі

Після навчання моделі необхідно оцінити його результати, та надати метрику точності моделі. Для оцінки моделі використана метрика mAP@IoU[.5:.5:.95]. Ця метрика означає значення середньої точності (AP – average precision) моделі при різних порогових значеннях IoU (intersection over union). Метрика mAP в залежності від IoU обрана, оскільки вона часто використовується для оцінки саме моделей, що створенні для розпізнавання об’єктів. Параметр IoU означає наскільки близькі передбачені моделлю обмежувальні прямокутники до ground truth. Значення IoU представляє собою співвідношення площі перетину фактичного та передбаченого прямокутника та площі їх об’єднання. Таким чином для ідеально передбаченого обмежувального прямокутника IoU = 1. Серед інших параметр, що входять до mAP, належать precision та recall.

Precision – це відношення між вірними передбаченнями моделі до сумарної кількості вірних та хибно позитивних передбачень. Коли значення precision високе, це означає, що модель рідко робить хибно позитивні передбачення. Recall – це відношення між кількістю дійсно вірних передбачень моделі та сумарною кількістю вірних та хибно негативних параметрів. Значення recall та precision обчислюються за наявності певного порогового значення, за допомогою якого можна класифікувати передбачення як вірне або хибне. У випадку AP, використовуються порогові значення IoU. Значення AP визначається за допомогою precision-recall curve, котра будується за значеннями recall та precision, отриманих з різними порогами IoU, та обчислюється окремо для кожного з класу об’єктів, отже значення mAP є середнім значенням AP для кожного із класів (тобто при наявності лише одного класу ці значення рівні).

Для обчислення значень IoU використовується метод calc_iou(), а для обчислень значень precision та recall метод з бібліотеки scikit.

```python
def calc_iou(pred_box, ground_truth_box):
    min_x1 = pred_box[1]
    min_x2 = ground_truth_box[1]
    min_y1 = pred_box[0]
    min_y2 = ground_truth_box[0]

    width1 = abs(pred_box[1] - pred_box[3])
    width2 = abs(ground_truth_box[1] - ground_truth_box[3])
    height1 = abs(pred_box[0] - pred_box[2])
    heigth2 = abs(ground_truth_box[0] - ground_truth_box[2])

    inter_box_top_left = [max(min_x2, min_x1), max(min_y2, min_y1)]
    inter_box_bottom_right = [min(min_x2 + width2, min_x1 + width1),
                              min(min_y2 + heigth2, min_y1 + height1)]

    inter_box_w = inter_box_bottom_right[0] - inter_box_top_left[0]
    inter_box_h = inter_box_bottom_right[1] - inter_box_top_left[1]

    intersection = inter_box_w * inter_box_h
    union = width2 * heigth2 + width1 * height1 - intersection

    iou = intersection / union
    return iou
```

Рисунок 27. Приклад імплементація обчислення IoU

Запис mAP@IoU[.5:.5:.95] означає, що для обчислення даної метрики використані значення precision та recall при порогових значеннях IoU в діапазоні [0.5; 0.95] із кроком 0.5. Для натренованої моделі значення mAP становить ~ 0.73. Оцінка точності моделі проводилась на раніше створеній з набору даних тестовій вибірці.

3.5 Застосування готової моделі

Для роботи з моделлю створений консольний застосунок, що дозволяє використовувати збережену натреновану модель для розпізнавання долонь. Параметри застосунку дозволяють розпізнавати долоні на збережених на

комп’ютері зображень чи відео, або використовувати наявну відеокамеру в якості джерела вхідних зображень. При роботі з відеопотоком створюється вікно в якому поверх зображення з відеокамери накладається обмежувальний прямокутник, що містить долоню.

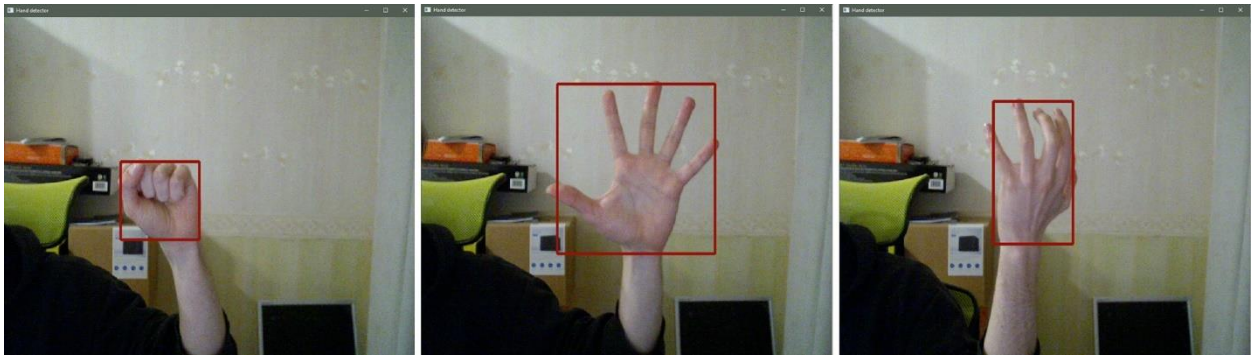

Рисунок 28. Приклад роботи програми з відео в якості джерела

При роботі з одиночними зображеннями, вихідні зображення з накладеними обмежувальними прямокутниками зберігаються не комп’ютер користувача.

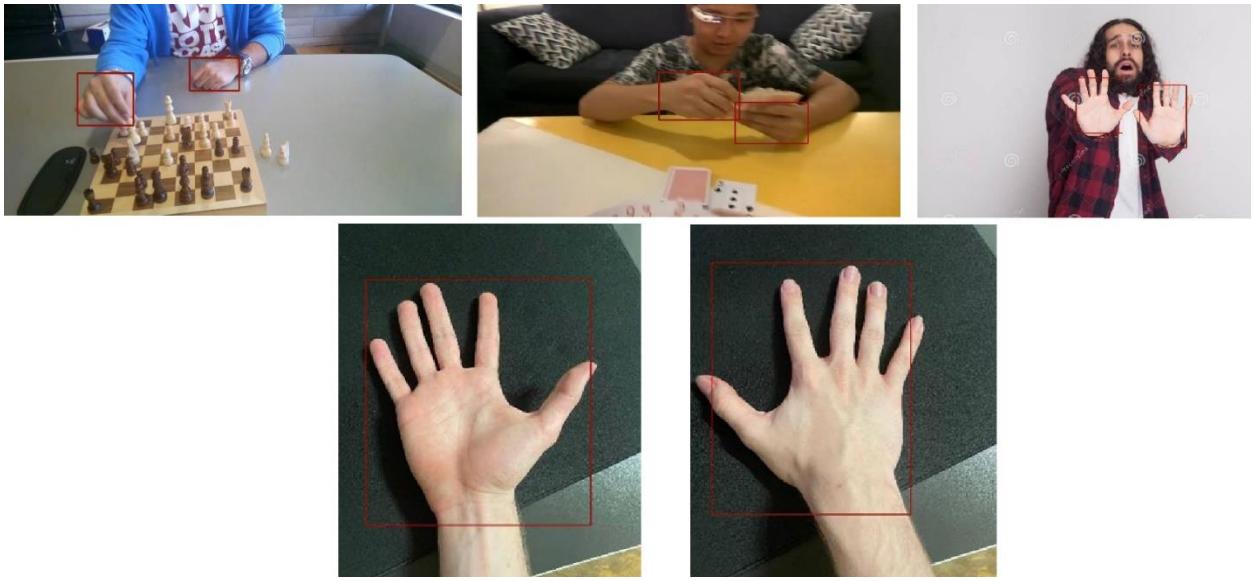

Рисунок 29. Приклад збережених оброблених зображень

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі з використанням перенесення навчання, донавчена модель нейронної мережі для розпізнавання долонь. Реалізовані програмні засоби для підготовки набору даних та його розділення на вибірки, для аугментації навчальних даних. Також створений функціонал для навчання моделі та її оцінки після навчання. Проведене навчання та оцінку моделі, після чого розроблено консольний застосунок для взаємодії з навченою моделлю.

ВИСНОВКИ

У даній роботі, у перших її розділах розглянута проблематика розпізнавання об’єктів на зображеннях та описані методи із комп’ютерного зору для вирішення таких задач. Під час порівняння методів виявлено, що використання згорткових нейронних мереж є найоптимальнішим способом вирішення поставленої задачі – розпізнавання долоней людини.

В подальших розділах досліджені способи програмної реалізації нейронних мереж, вибрано сучасну мову програмування та фреймворк для роботи з нейронними мережами – Python та TensorFlow. Також обрана архітектура майбутньої мережі, а саме – SSD. Сформульовані вимоги до набору даних, на якому відбувалося навчання моделі, і за цими вимогами обрано відповідний набір даних.

В останньому розділі, на основі обраних програмних засобів готову модель для розпізнавання об’єктів донавчена з використанням прийому під назвою перенесення навчання. Для навчання моделі та роботи з набором даних створено необхідний функціонал, також створено функціонал для оцінки результатів навчання моделі. Для роботи з навченою моделлю створений консольний застосунок, наведені приклади його застосування.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. D. G. Lowe, "Object recognition from local scale-invariant features," Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999, C. 1150-1157.
2. Fixed and Adaptive Thresholding [Електронний ресурс] – Режим доступу до ресурсу: https://sites.wustl.edu/clouddetection/cloud-detection/fixed-and-adaptive-thresholding/.
3. Introduction to Neurons in Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/artificial-neural-networks/introduction-to-neurons-in-neural-networks-71828d040a65.
4. Understanding Convolutions and Pooling in Neural Networks: a simple explanation [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/understanding-convolutions-and-pooling-in-neural-networks-a-simple-explanation-885a2d78f211.
5. Convolutional Neural Networks (CNNs) [Електронний ресурс] – Режим доступу до ресурсу: https://anhreynolds.com/blogs/cnn.html.
6. Machine Learning 101 | Supervised, Unsupervised, Reinforcement & Beyond [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/machine-learning-101-supervised-unsupervised-reinforcement-beyond-f18e722069bc.
7. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1311.2524.
8. Ross Girshick. Fast R-CNN [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1504.08083.
9. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1506.01497.

10. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1506.02640.
11. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1512.02325.
12. Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3) [Електронний ресурс] – Режим доступу до ресурсу: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359.
13. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár. Focal Loss for Dense Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1708.02002.
14. Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab.
15. Comparison of AI Frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.pathmind.com/comparison-frameworks-dl4j-tensorflow-pytorch.
16. Bambach, S. Lee, D. J. Crandall and C. Yu, "Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, C. 1949-1957.
17. Transfer Learning - Machine Learning's Next Frontier [Електронний ресурс] – Режим доступу до ресурсу: https://ruder.io/transfer-learning/.

18. An overview of gradient descent optimization algorithms [Електронний ресурс] – Режим доступу до ресурсу: https://ruder.io/optimizing-gradient-descent/.
19. Evaluating Object Detection Models Using Mean Average Precision (mAP) [Електронний ресурс] – Режим доступу до ресурсу: https://blog.paperspace.com/mean-average-precision/.

Додаток А

```
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
SSD300                                                  --                        --
├─ModuleList: 1-1                                       --                        --
├─ModuleList: 1-2                                       --                        --
├─ModuleList: 1-3                                       --                        --
├─ResNet: 1-4                                           [3, 1024, 38, 38]         --
│    └─Sequential: 2-1                                  [3, 1024, 38, 38]         --
│    │    └─Conv2d: 3-1                                 [3, 64, 150, 150]         9,408
│    │    └─BatchNorm2d: 3-2                            [3, 64, 150, 150]         128
│    │    └─ReLU: 3-3                                   [3, 64, 150, 150]         --
│    │    └─MaxPool2d: 3-4                              [3, 64, 75, 75]           --
│    │    └─Sequential: 3-5                             [3, 256, 75, 75]          215,808
│    │    └─Sequential: 3-6                             [3, 512, 38, 38]          1,219,584
│    │    └─Sequential: 3-7                             [3, 1024, 38, 38]         7,098,368
├─ModuleList: 1-1                                       --                        --
│    └─Sequential: 2-2                                  [3, 512, 19, 19]          --
│    │    └─Conv2d: 3-8                                 [3, 256, 38, 38]          262,144
│    │    └─BatchNorm2d: 3-9                            [3, 256, 38, 38]          512
│    │    └─ReLU: 3-10                                  [3, 256, 38, 38]          --
│    │    └─Conv2d: 3-11                                [3, 512, 19, 19]          1,179,648
│    │    └─BatchNorm2d: 3-12                           [3, 512, 19, 19]          1,024
│    │    └─ReLU: 3-13                                  [3, 512, 19, 19]          --
│    └─Sequential: 2-3                                  [3, 512, 10, 10]          --
│    │    └─Conv2d: 3-14                                [3, 256, 19, 19]          131,072
│    │    └─BatchNorm2d: 3-15                           [3, 256, 19, 19]          512
│    │    └─ReLU: 3-16                                  [3, 256, 19, 19]          --
│    │    └─Conv2d: 3-17                                [3, 512, 10, 10]          1,179,648
│    │    └─BatchNorm2d: 3-18                           [3, 512, 10, 10]          1,024
│    │    └─ReLU: 3-19                                  [3, 512, 10, 10]          --
│    └─Sequential: 2-4                                  [3, 256, 5, 5]            --
│    │    └─Conv2d: 3-20                                [3, 128, 10, 10]          65,536
│    │    └─BatchNorm2d: 3-21                           [3, 128, 10, 10]          256
│    │    └─ReLU: 3-22                                  [3, 128, 10, 10]          --
│    │    └─Conv2d: 3-23                                [3, 256, 5, 5]            294,912
│    │    └─BatchNorm2d: 3-24                           [3, 256, 5, 5]            512
│    │    └─ReLU: 3-25                                  [3, 256, 5, 5]            --
│    └─Sequential: 2-5                                  [3, 256, 3, 3]            --
│    │    └─Conv2d: 3-26                                [3, 128, 5, 5]            32,768
│    │    └─BatchNorm2d: 3-27                           [3, 128, 5, 5]            256
│    │    └─ReLU: 3-28                                  [3, 128, 5, 5]            --
│    │    └─Conv2d: 3-29                                [3, 256, 3, 3]            294,912
│    │    └─BatchNorm2d: 3-30                           [3, 256, 3, 3]            512
│    │    └─ReLU: 3-31                                  [3, 256, 3, 3]            --
│    └─Sequential: 2-6                                  [3, 256, 1, 1]            --
│    │    └─Conv2d: 3-32                                [3, 128, 3, 3]            32,768
│    │    └─BatchNorm2d: 3-33                           [3, 128, 3, 3]            256
│    │    └─ReLU: 3-34                                  [3, 128, 3, 3]            --
│    │    └─Conv2d: 3-35                                [3, 256, 1, 1]            294,912
│    │    └─BatchNorm2d: 3-36                           [3, 256, 1, 1]            512
│    │    └─ReLU: 3-37                                  [3, 256, 1, 1]            --
```

```
├─ModuleList: 1-2                                --                        --
│    └─Conv2d: 2-7                               [3, 16, 38, 38]           147,472
├─ModuleList: 1-3                                --                        --
│    └─Conv2d: 2-8                               [3, 324, 38, 38]          2,986,308
├─ModuleList: 1-2                                --                        --
│    └─Conv2d: 2-9                               [3, 24, 19, 19]           110,616
├─ModuleList: 1-3                                --                        --
│    └─Conv2d: 2-10                              [3, 486, 19, 19]          2,239,974
├─ModuleList: 1-2                                --                        --
│    └─Conv2d: 2-11                              [3, 24, 10, 10]           110,616
├─ModuleList: 1-3                                --                        --
│    └─Conv2d: 2-12                              [3, 486, 10, 10]          2,239,974
├─ModuleList: 1-2                                --                        --
│    └─Conv2d: 2-13                              [3, 24, 5, 5]             55,320
├─ModuleList: 1-3                                --                        --
│    └─Conv2d: 2-14                              [3, 486, 5, 5]            1,120,230
├─ModuleList: 1-2                                --                        --
│    └─Conv2d: 2-15                              [3, 16, 3, 3]             36,880
├─ModuleList: 1-3                                --                        --
│    └─Conv2d: 2-16                              [3, 324, 3, 3]            746,820
├─ModuleList: 1-2                                --                        --
│    └─Conv2d: 2-17                              [3, 16, 1, 1]             36,880
├─ModuleList: 1-3                                --                        --
│    └─Conv2d: 2-18                              [3, 324, 1, 1]            746,820
==========================================================================================
Total params: 22,894,902
Trainable params: 22,894,902
Non-trainable params: 0
```

Додаток Б

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_resnet50_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 0.00039999998989515007
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.029999999329447746
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    box_predictor {
```

```
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 0.00039999998989515007
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 256
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    scales_per_octave: 2
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.99999993922529e-09
    iou_threshold: 0.6000000238418579
    max_detections_per_class: 100
    max_total_detections: 100
    use_static_shapes: false
  }
  score_converter: SIGMOID
}
normalize_loss_by_num_matches: true
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_loss {
    weighted_sigmoid_focal {
      gamma: 2.0
      alpha: 0.25
    }
  }
  classification_weight: 1.0
  localization_weight: 1.0
}
encode_background_as_zeros: true
```

```
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
```