НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

«На правах рукопису»
УДК 004.8

«До захисту допущено»
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
“___” _____ 2020р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 113 Прикладна математика _____
(код і назва)

на тему: Розпізнавання жестів на основі скелету руки та оцінка їх якості _____
_____
_____

Виконав (-ла): студент (-ка) 6___ курсу, групи ФІ-83мн_____
(шифр групи)

Дикий Владислав Сергійович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник в.о. зав. кафедри, д.ф.-м.н., чл.-кор. НАН України Савчук М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020_року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
Фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти: другий (магістерський) за освітньо–науковою програмою

Спеціальність: 113 «Прикладна математика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
«___»_____2020_ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Дикий Владислав Сергійович _____
(прізвище, ім'я, по батькові)

1. Тема дисертації: Розпізнавання жестів на основі скелету руки та оцінка їх якості,
науковий керівник дисертації зав. кафедри, д.ф.-м.н., чл.-кор. НАН України Савчук М.М. _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _____р. № _____

2. Термін подання студентом дисертації 11.05.2020

3. Об'єкт дослідження: системи комп'ютерного зору та розпізнавання об'єктів на зображені
4. Предмет дослідження (Вхідні дані – для магістерської дисертації за освітньо–професійною програмою): алгоритми розпізнавання об'єктів на зображені

5. Перелік завдань, які потрібно розробити: розробити та навчити моделі детектора кисті руки, моделі скелета, моделі класифікатора жестів; порахувати довірчі інтервали для оцінок навчання моделей.

6. Орієнтовний перелік ілюстративного матеріалу: презентація.

7. Орієнтовний перелік публікацій: частина результатів роботи представлена на XVIII Науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та

інформатики» (12-13 травня 2020 р., м. Київ)

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 01.09.2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Обробка літератури	01.09.2019-01.12.2019	
2	Аналіз метрик оцінювання розпізнавання та класифікації	01.12.2019 – 01.02.2020	
3	Розробка та навчання моделей детектора, скелета, класифікатора	01.02.2020-01.04.2020	
4	Обрахунок довірчих інтервалів для оцінок навчання моделей	01.04.2020-01.05.2020	

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Кваліфікаційна робота містить: 76 стор., 21 рисунок, 5 таблиць, 18 джерел.

По-перше, в даній роботі побудовано та натреновано моделі детектора руки та моделі скелета, з використанням First Person Hand Action датасету, які необхідні для подальшої класифікації жестів. Наведено всі необхідні перетворення над даними, які поступають на вхід моделям. Пораховано довірчі інтервали для математичного сподівання оцінок якості передбачення моделей за останні епохи навчання.

По-друге, побудовано та навчено модель класифікатора жестів. Даний класифікатор розпізнає динамічні жести, які є послідовністю скелетів з моделі скелету. Пораховано довірчі інтервали для математичного сподівання оцінок точності загального розпізнавання жесту та для кожного жесту окремо.

ДЕТЕКТОР РУКИ, КЛАСИФІКАЦІЯ ЖЕСТІВ, СКЕЛЕТ КИСТІ РУКИ, НЕЙРОНА МЕРЕЖА, СТАТИСТИЧНЕ ОЦІНЮВАННЯ, ДОВІРЧІ ІНТЕРВАЛИ, АЛГОРИТМИ РОЗПІЗНАВАННЯ

ABSTRACT

The qualifying paper contains: 76 pages, 21 figures, 5 tables, 18 sources.

First, in this paper hand detector and skeleton models are built and trained with the using of the First Person Hand Action dataset, what are necessary for further gesture classification. All necessary transformations over the data arriving on input to models are presented. The confidence intervals for expected value of the estimates of the quality of model prediction for the last epochs of training are calculated.

Second, a gesture classifier model is built and trained. This classifier is able to recognize the dynamic gestures that are a sequence of skeletons from a skeleton model. Cofidence intervals are calculated for the expected value of the accuracy as for the general gesture recognition as for the each gesture separately.

HAND DETECTOR, GESTURE CLASSIFICATION, HAND SKELETON, NEURAL NETWORK, STATISTICAL EVALUATION, CONFIDENCE INTERVALS, RECOGNITION ALGORITHMS

ЗМІСТ

ВСТУП

Актуальність дослідження полягає у тому, що сьогодні стають все більш популярні застосунки віртуальної та доповненої реальності та в цій роботі пропонуються моделі, які можуть покращити розпізнавання в цих середовищах.

Метою роботи – є побудувати моделі детектора руки, скелета кисті руки для покращення розпізнавання жестів у віртуальній та доповненій реальності.

Завдання роботи – навчити моделі детектора руки, скелета кисті руки, класифікатора жестів та оцінити роботу моделей побудувавши довірчі інтервали для математичного сподівання оцінок якості.

Об'єктом дослідження є системи комп'ютерного зору та розпізнавання об'єктів на зображені.

Предметом дослідження є алгоритми розпізнавання об'єктів на зображені.

При розв'язуванні поставлених завдань використовувались наступні методи дослідження: методи статистичного аналізу, лінійної алгебри та методи навчання та побудови нейронних мереж.

Наукова новизна отриманих результатів полягає у тому, що було запропоновано архітектуру моделі класифікатора жестів, яка є простою та дала гарні результати.

Практичне значення результатів показує, що отримана модель класифікатора має гарні оцінки точності розпізнавання при відносній простоті моделі ти швидкому її навчанні.

Апробація результатів та публікації. Частина даної роботи представлена на XVIII Науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики» (12-13 травня 2020 р., м. Київ).

1 ТЕХНОЛОГІЇ ТА ЗАГАЛЬНІ ВІДОМОСТІ

У даному розділі піде мова про інструменти оцінки якості передбачень нейронних мереж, які буде використовуватися у даній роботі. Буде описано які використовуються метрики для оцінки роботи моделі детектора, а саме метрика коректності, повноти, точності, симбіоз повноти та точності – F1 метрика, яка показує баланс між двома попередніми, та буде сказано кілька слів про коваріаційну матрицю. Також піде мова про міру Жаккара(IoU), яка необхідна при оцінки якості детектора при пошуку зони знаходження шуканого об'єкта. Та нарешті буде описано алгоритм обрахунку AP – метрики, яка буде головною при оцінці якості навченого детектора. Для оцінці якості моделі скелета буде використовуватися L_2 міра, яка також буде описана у даному розділі.

Також буде наведена інформація про архітектури, які будуть використовуватися для навчання моделі скелета, а саме MobileNetV1[2] та MobileNetV2[3]. Хоч модель скелету буде використовувати архітектуру MobileNetV2, все одно буде описано MobileNetV1, оскільки друга версія архітектури основана на першій, та використовує всі її особливості та переваги.

1.1 Метрики оцінки якості детекції або класифікації об'єктів

При розв'язанні задачі класифікації та детекції об'єктів часто виникає питання на скільки точна модель детектора чи класифікатора. Бо часто може здаватися, що модель працює бездоганно, але насправді в неї є деякі проблеми, також необхідно мати формальну та точну оцінку якості при презентації цієї моделі. Для оцінки таких моделей використовують різні метрики в залежності від постановки задачі. В даному розділі піде мова про метрики, які будуть використовуватися при

виконанні даної дипломної роботи. Отже, буде розглянуто такі метрики: Accuracy[4], Precision[5] та Recall[5], F1-score[5], а також Interseption over union[7], надалі - IOU.

1.1.1 Матриця класифікації якості

Матриця коваріації використовується для підсумку результатів при прогнозуванні класифікатора. Кількість правильних та неправильних прогнозів підсумовується та записується у свій клас в даній матриці. Матриця коваріації показує яким чином помиляється модель класифікатора, тобто матриця дає розуміння не лише значення помилки класифікації, а що важливіше, показує які типи помилок допускає модель.

Матриця має наступну структуру: по рядках знаходяться дійсні класи, по стовпцях знаходяться передбачені класи моделлю.

Нехай є два класи: позитивний та негативний. Наприклад, в задачі детекції руки позитивним класом може бути наявність руки на зображені, та негативний – відсутність.

Об'єкт передбачення може прийняти 4 стани:

– TP (правильний позитивний) – об'єкт передбачений як позитивний, коли дійсно є позитивним.

– TN (правильний негативний) – об'єкт передбачений як негативний, коли дійсно є негативним

– FP (неправильний позитивний) – об'єкт передбачений як позитивний, коли є негативним

– FN (неправильний негативний) – об'єкт передбачений як негативний, коли є позитивним

Матриця коваріації має наступний вигляд:

	Передбачене як позитивне	Передбачене як негативне
Дійсно позитивне	TP	TN
Дійсно негативне	FN	FP

1.1.2 Коректність класифікації

Дана метрика показує кількість правильно розпізнаних об'єктів у відношенні до загальної кількості передбачень. Обчислюється коректність за наступною формулою[4]:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN},$$

де TP, TN, FP, FN значення, що використовуються у матриці коваріації.

Дана метрика не є дуже гарною, бо вона не показує повної картини дійсності. Наприклад, якщо класи при валідації не збалансовані, то можна отримати гарний результат, тому, що прикладів на яких модель класифікатора помиляється буде суттєво менше, ніж на тих, на яких модель гарно класифікує, чи навпаки. Тому для оцінки якості моделі, необхідно використовувати більше різних метрик та проводити аналіз на основі їх значень.

1.1.3 Точність та повнота

Точність показує кількість правильних позитивних передбачень серед всіх передбачень, які були класифіковані як позитивні.

Точність обчислюється за наступною формулою[5]:

$$Precision = \frac{TP}{TP + FP},$$

де TP, FP - значення з матриці коваріації.

Високе значення точності свідчить: якщо модель класифікувала об'єкт як позитивний, то з високою ймовірністю об'єкт дійсно є позитивним.

Повнота показує кількість позитивних передбачень серед всіх дійсних позитивних передбачень. Обчислюється повнота за такою формулою[5]:

$$Recall = \frac{TP}{TP + FN},$$

де TP, FN - значення з матриці коваріації.

Високе значення повноти означає, що даний клас передбачається точно, бо маленьке значення FN.

1.1.4 F1-метрика

Дана метрика призначена для того, щоб показати баланс між метриками точність та повнота. F-метрика – це середнє гармонійне між точністю та повнотою. Обчислюється наступним чином[5]:

$$F1 - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall},$$

де $Presicion$ та $Recall$ поняття з попереднього пункту.

Для того, щоб дана метрика мала високий показник, необхідно, щоб точність і повнота мали високі значення. Бо якщо, хоч одне з цих буде низьке, то F1-метрика прийме також низьке значення.

1.1.5 Міра Жаккара(IOU)

Міра Жаккара[6] або міра подібності – це статистика, яка показує подібність та відмінність між екземплярами набору даних. Дана міра використовується для оцінки схожості двох множин і обчислюється як міра спільної частки, поділена на міру об'єднаної частини множин. Міра

Жаккара для множини A та B обчислюється як[7]:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

В задачах детекції об'єктів, в яких необхідно знайти зону розміщення об'єкта, міру Жаккара використовують для оцінки точності передбачення моделлю обмежувальної рамки. Також цю міру часто називають, як площа перетину(з англ. Interseption over Union, IoU).

Для того, щоб обчислити IoU необхідно мати:

1) оригінальні координати зони розміщення об'єкта

2) знайдені координати зони моделлю

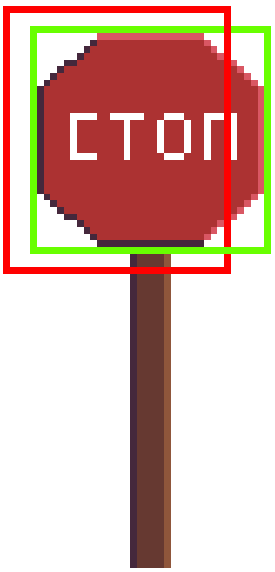

Рисунок 1.1 – Приклад вигляду обмежувальної рамки

На рисунку 1.1 зображено дорожній знак, який оточений двома рамками: зеленою(оригінальна) та червоною(знайденою моделлю). Обчислення IoU можна показати з допомогою наступного малюнка:

В чисельнику, зображеного рівняння на рисунку 1.2, знаходиться перетин між оригінальною рамкою та передбаченою моделлю. А в знаменнику знаходиться їх об'єднана площа. Отже, IoU – це звичайне відношення, між спільною площею двох рамок та їх об'єднаною площею.

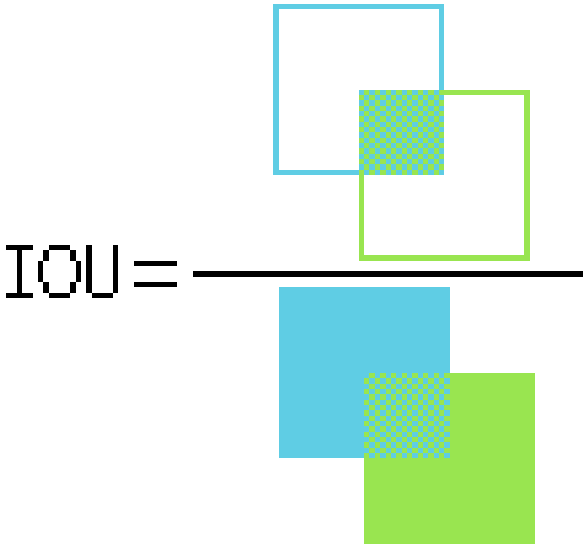

Рисунок 1.2 – Візуалізація обчислення IoU

Дана метрика необхідна для того, щоб покращити передбачувальні здібності детектора об'єкта. Бо детектор майже ніколи не зможе чітко знайти координати зони розміщення об'єкта, але в нього є всі шанси підібратися якомога ближче до оригінальної рамки завдяки цій метриці. На рисунку 1.3 зображено приклади можливої роботи детектора та оцінки IoU.

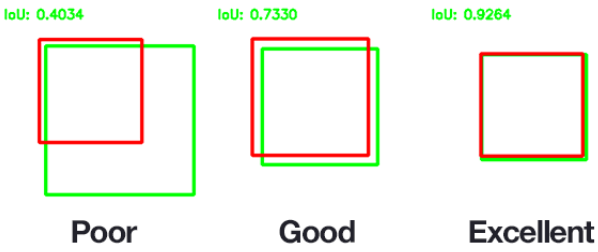

Рисунок 1.3 – Приклади результату роботи детектора та обчислені метрики IoU

1.1.6 L_2 метрика

L_2 метрика або Евклідова відстань – це пряма відстань між двома точками у просторі. В даній роботі ця метрика буде використовуватися для обрахування точності передбачень моделі скелету, оскільки вона буде гарно показувати середню помилку скелету у міліметрах.

Обраховується як квадратний корінь суми квадратів довжин:

$$l_2(a, b) = \sqrt{\sum_{i=0}^{m}(a_i - b_i)^2},$$

де a, b це точки у просторі, які мають розмірність m.

При навчанні та валідації дана метрика буде обрахована для кожного екземпляру датасету, тому кінцева метрика L_2 буде середнім між кожним l2 обрахованим раніше:

$$L_2 = \frac{\sum_{i=0}^{n} l_{2i}}{n},$$

де n – це кількість екземплярів тренувального або валідаційного датасету.

1.2 MobileNetV2

Згорткові нейроні мережі стають дуже популярні у задачах комп'ютерного зору. Та сьогодні основний тренд на згорткові нейроні мережі полягає у тому, що вони стають глибшими та складнішими, звісно це робиться з метою підвищення їх якості та точності передбачувальних здібностей. Тобто чим складнішими вони стають, тим краще їх результати роботи. Проте якість роботи вимірюється не тільки результатами обчислення моделей, а також суттєве значення має швидкодія нейронної мережі. Звісно, тільки в задачах, які потребують результати на льоту, оскільки є задачі, в яких все-таки необхідно досягти якомога кращого результату. Ускладнення нейронної мережі та покращення її результатів

приводить до збільшення часу обчислень.

В сучасних застосунках таких, як доповнена та віртуальна реальність, в робототехніці та інших, необхідно, щоб нейрона мережа могла ефективно проводити обчислення в умовах обмеженої обчислювальної потужності. Одним з прикладів може бути смартфон. Тому в даному підрозділі піде мова про нейронну мережу MobileNetV1, яка було у статті "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"[2] та MobileNetV2, яка описана в статті "MobileNetV2: Inverted Residuals and Linear Bottlenecks"[3]. У даній дипломній роботі буде використана мережа з архітектурою MobileNetV2, тому в даному розділі буде описана саме ця архітектура. Та також буде кілька слів про архітектуру MobileNet, бо MobileNetV2 основана на першій і перейняла її головні властивості.

1.2.1 Глибока сепарабельна згортка

Згорткові нейроні мережі використовують зазвичай звичайні згортки, які застосовуються до кожного каналу вхідного зображення. Згортка проходить через все зображення та рахує зважену суму відповідних пікселів з кожного каналу та записує це в новий піксель. Якщо зображення трьох канальне, то після застосування згортки вихідне зображення буде мати один канал. Тобто після застосування однієї згортки до зображення з будь-якою кількістю каналів, все одно вийде одно канальне зображення. Наспрвді, до одного зображення застосовується багато згорток, тому на виході вийде багатоканальне зображення, де кожен канал буде відноситися до певної згортки.

На рисунку 1.4 показано як працює згортка. В цьому випадку це 3D згорка, яка має розмір (3×3) та елементи з трьох каналів зображення відображаються в один піксель одного каналу.

В архітектурі MobileNet звичайна згортка використовується, але тільки один раз на першому шарі мережі. На всіх інших шарах

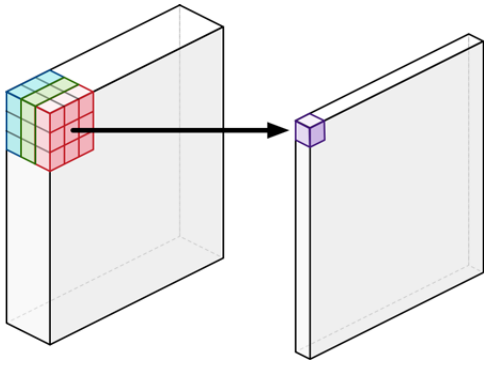

Рисунок 1.4 – Демонстрація роботи згортки на трьох канальному зображені

використовується глибока сепарабельна згортка про яку далі й піде мова. Дана згортка є комбінацією двох згорток, а саме глибокої згортки(depth convolution) та точкової згортки(pointwise convolution).

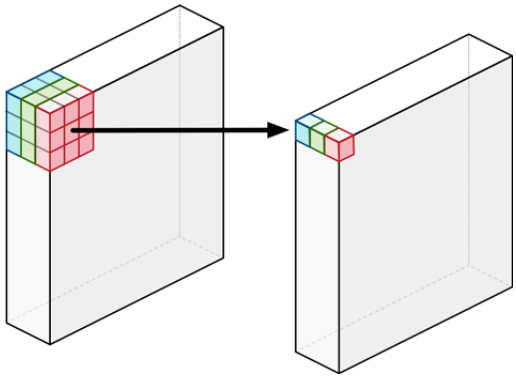

Рисунок 1.5 – Демонстрація роботи глибокої згортки на трьохканальному зображені

Глибока згортка працює як звичайна згортка, але на противагу їй, вона не комбінує вхідні канали в один канал, а залишає їх всі окремо. Тобто для кожного каналу застосовується звичайна 2D згортка окремо. Наприклад, якщо зображення має 3 канали, то на виході, після застосування глибокої згортки, також буде 3 канали. Сенс цієї згортки полягає у знаходженні особливостей, таких як детекція кутів наприклад, на кожному каналу окремо. На рисунку 1.5 показано схематично, як діє дана згортка.

Після глибокої згортки слідує точкова згортка. Точкова згортка насправді це звичайна згортка, але з розмірністю (1×1). Тобто дана згортка просто-таки складає відповідні пікселі в різних каналів. Це робиться з метою отримання нових особливостей(фітч) з різних каналів після глибокої згортки. На рисунку 1.6 показано роботу точкової згортки.

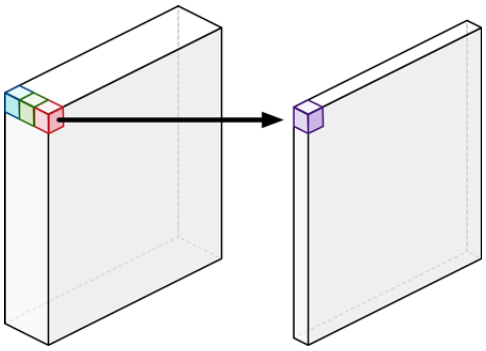

Рисунок 1.6 – Демонстрація роботи точкової згортки на трьох канальному зображені

Так чином комбінація глибокої та точкової згортки являється глибокою сепарабельною згорткою. Сенс використання глибокої сепарабельної згортки такий самий, як і використання звичайної згортки. Оскільки обидві згортки фільтрують зображення та генерують нові фітчі. Але для обчислення глибокої сепарабельної згортки необхідно проводити менше обчислень, ніж для звичайної згортки. Автори MobileNet показали, що обчислювальна складність глибокої сепарабельної згортки складає[2]:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F,$$

де D_K – розмір згортки, D_F – розмір зображення, M – кількість каналів зображення, N – кількість каналів в вихідному зображені.

Дана оцінка є сумою обчислювальної складності глибокої згортки та точкової згортки.

Обчислювальна складність звичайної згортки складає[2]:

$$D_K \cdot D_K \cdot N \cdot M \cdot D_F \cdot D_F$$

Таким чином зменшення обчислень складає[2]:

$$\frac{D_K \cdot D_K \cdot N \cdot M \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

1.2.2 Множник ширини α та множник роздільної здатності ρ архітектури MobileNet

Архітектура нейрнної мережі MobileNet вже є достатньо малою та швидкою, але іноді цього недостатньо в деяких специфічних випадках, які потребують ще більшої швидкості обчислень. Тому в архітектурі присутні гіперпараметри α та ρ. Ці гіперпараметри дозволяють налаштовувати мережу як потрібно.

Гіперпараметр α дозволяє зменшувати шари мережі, а точніше він зменшує кількість вхідних та вихідних каналів. Тобто кількість вхідних каналів буде становити αM та кількість вихідних каналів αN. Таким чином обчислювальна складність глибокої сепарабельної згортки буде складати[2]

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F,$$

де $\alpha \in (0, 1]$.

Зазвичай гіперпараметр α дорівнює 1, 0.75, 0.5 та 0.25. При значенні 1 це звичайна базова архітектура. Кількість обчислень зменшується при $\alpha < 1$.

Гіперпараметр ρ називається параметром роздільності та застосовується для регулювання роздільної здатності вхідного та вихідного зображення. Обчислювальна складність глибокої сепарабельної згортки з урахуванням гіперпараметра α буде складати[2]:

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F,$$

де $\rho \in (0, 1]$. При $\rho = 1$ буде звичайна архітектура MobileNet та зменшення

обчислень буде відбуватись при $\rho < 1$. Зазвичай, ρ обирають таким, щоб розмірність дорівнювала 224, 192, 160, 128.

Отже, гіперпараметри α та ρ є деяким компромісом між швидкістю та точністю обчислень.

1.2.3 Нововедення архітектури MobileNetV2: bottleneck

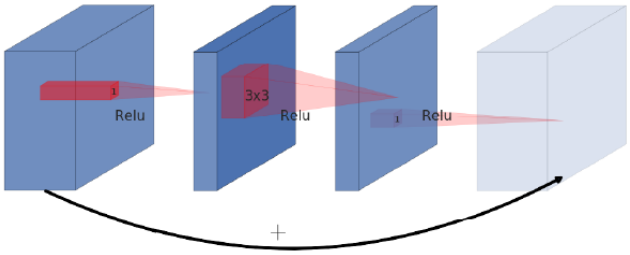

Рисунок 1.7 – Пропускний блок архітектури ResNet

Нова архітектура MobileNetV2 використовує так звані залишкові блоки. Робота цих блоків доволі проста, просто залишковий блок з'єднує початок і кінець згорткового блоку. Це робиться через пропускне з'єднання. Додавши початок і кінець цього блоку, нейрона мережа отримує доступ до попереднього стану, який не був змінений згорткою. Залишковий блок вперше був показаний у архітектурі нейронної мережі ResNet[8]. Але в MobileNetV2 цей блок трішки інакший, оскільки в ResNet пропускне з'єднання діє наступним чином: воно спочатку звужує зображення, а потім назад його розширює. Його роботу можна побачити на рисунку 1.7. а в MobileNetV2 навпаки, спочатку розширює, а потім звужує, що можна побачити на рисунку 1.8.

На рисунку 1.8 останній блок це згортка (1×1) з лінійною функцією активації, яка знижує кількість каналів. Автори MobileNetV2 мають гіпотезу, що інформація, отримана на попередніх кроках, може уміститися в підпростір меншої розмірності, без інформаційних втрат.

Отже, це і є блок який називається bottleneck. В даному блоці

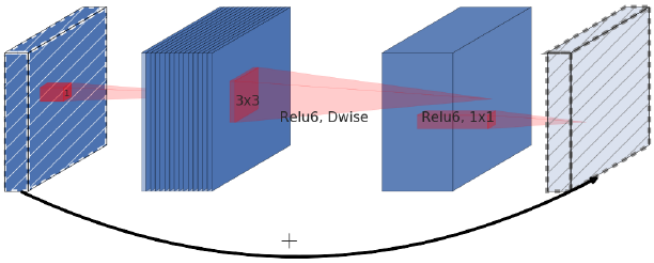

Рисунок 1.8 – Пропускний блок архітектури MobileNetV2

використовується активаційна функція $Relu6$, яка визначається наступним чином:

$$Relu6(x) = min(max(0, x), 6)$$

Повна архітектура MobileNetV2 зображена на рисунку 1.9, де t – коефіцієнт розширення, c – глибина каналів згортки, n – кількість повторень блока, s – параметр згортки, який задає з яким кроком згортці рухатися по зображені.

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	-	

Рисунок 1.9 – Архітектура MobileNetV2

Висновки до розділу 1

Отже, підведемо підсумки першого розділу. Даний розділ можна умовно розділити на дві частини: опис метрик та опис архітектури нейронної мережі MobileNetV2.

В першій частині мова йде про метрики, які дають змогу оцінити адекватність навчених моделей. Наведено такі метрики, як коректність, повнота, точність та F1–метрика. Дані метрики в роботі використовуються для оцінки правильності розпізнавання руки на зображені. Також описується роботу метрики IoU, яка необхідна для оцінки коректності знаходження координат зони розміщення шуканого об'єкта. І нарешті, було описано L_2 метрику, яка необхідна для оцінки знайденого скелета кисті руки.

В другій частині розділу мова йде про архітектуру нейронної мережі MobileNetV2, яка буде використана для розв'язання задачі пошуки суглобів кисті руки. Описується перша версія цієї архітектури MobileNet, а саме її особливості, як глибока сепарабельна згортка, та коефіцієнти α та ρ. Також описано нововведення вже архітектури MobileNetV2, а точніше описано як влаштований блок bottleneck та в чому його сенс.

2 ЦИКЛ РОЗПІЗНАВАННЯ ЖЕСТІВ

В даному розділі піде мова про цикл розпізнавання жестів, який складається з 3 етапів: моделі детектора руки, моделі скелета та класифікатора жестів.

На першому етапі буде описано підготовку вхідних даних для навчання моделі детектора, описано архітектуру детектора, процес навчання, результати навчання та буде пораховані довірчі результати для цих оцінок.

На другому етапі буде описано підготовку даних для моделі скелета після роботи детектора і описано процес навчання скелета, і також писано конвертацію точок з 3D в 2D + D та навпаки. Також піде мова про датасет, який використовується для навчання моделей. Також буде наведено результати навчання моделі скелета та пораховані довірчі результати для цих оцінок.

На останньому етапі циклу буде описано роботу жестів та архітектуру класифікатора жестів, наведено результати точності розпізнавання жестів та пораховані для оцінок точності розпізнавання довірчі інтервали.

2.1 Підготовка тензора для детектора

Для того, щоб підготувати вхідний тензор для нейронної мережі детектора, необхідно зробити декілька перетворень на вхіднім кольоровим зображенням та зображенням глибини.

Для кольорового зображення це наступні перетворення:

1) зміна розміру

2) нормалізація

Для зображення глибини перетворень на одне більше:

1) зміна розміру

2) нормалізація

3) обмеження глибини

Ці перетворення наведені далі у підрозділах 2.1.1, 2.1.2, 2.1.3 та 2.1.4.

2.1.1 Зміна розміру зображень для детектора

Зміна розміру зображення необхідна, оскільки вхідне зображення може мати будь-який розмір, але нейрона мережа на вхід приймає тільки чітко визначену розмірність, в цьому випадку вхідна розмірність зображення для детектора складає (112×112). Зміна розміру кольорового зображення досягається з допомогою алгоритму передискретизації зображення. А для зміни розміру зображення глибини використовується алгоритм інтерполяції методом найближчого сусіда. Для зміни розміру зображення в даній роботі застосовується функція з бібліотеки OpenCV[9].

2.1.2 Нормалізація RGB-зображення

Будь-яке зображення складається з пікселів. Зазвичай у кольорового зображення піксель це кортеж з трьох значень, де кожне значення відповідає певному каналу: червоному(R), зеленому(G) та блакитному(B) каналу. Числа, що лежать по цім каналам, приймають значення від 0 до 255.

Нормалізація потрібна, щоб вкласти значення пікселів, які належать інтервалу $[0, 255]$, в інтервал $[0, 1]$.

Нормалізація буде відбуватися у два етапи:

1) ділення кожного значення пікселя на 255.0

2) віднімання середнього від пікселя та ділення його на дисперсію

Псевдокод алгоритму нормалізації наведено нижче. На вхід алгоритм приймає RGB-зображення, набір мат. очікувань та дисперсій для кожного

каналу зображення.

Algorithm 2.1 Нормалізація RGB-зображення

Require: Image, (rMean, gMean, bMean), (rStd, gStd, bStd)
  for $row \in Image.rows$ **do**
    for $col \in Image.cols$ **do**
      pixel $\leftarrow Image[row, col[$
      $r \leftarrow$ pixel.r $\setminus$ 255.0
      $g \leftarrow$ pixel.g $\setminus$ 255.0
      $b \leftarrow$ pixel.b $\setminus$ 255.0
      $r \leftarrow \frac{r - rMean}{rStd}$
      $g \leftarrow \frac{g - gMean}{gStd}$
      $b \leftarrow \frac{b - bMean}{bStd}$
      normalizedImage.insert(r)
      normalizedImage.insert(g)
      normalizedImage.insert(b)
    end for
  end for
  return normalizedImage

Нормалізацію проводять для того, щоб покращити процес навчання. По-перше, оскільки всі значення будуть лежати в інтервалі $[0, 1]$, то можна уникнути вибуху градієнта. Він зазвичай виникає, коли значення починають швидко збільшуватися. Отже, покращується стабільність навчання. По-друге, прискорюється процес навчання, оскільки параметр швидкості навчання буде підходити всім вхідним даним.

Параметри математичних сподівань та дисперсій обраховуються на датасеті, що обраний для навчання. Віднімання мат. очікування та ділення на дисперсію допомагає збільшити швидкість збігання алгоритму навчання.

2.1.3 Обмеження значень зображення глибини

Зображення глибини має один канал, в якому записано значення довжини від центру камери глибини до об'єкта зображеного на пікселі. Обмеження значень потрібно для того, щоб відфільтрувати зображення глибини, бо більшість пікселів знаходиться не в робочій зоні людської руки. Тому значення цих пікселів потрібно змінити на максимальне

допустиме значення, яке може бути опрацьовано. Це значення обрано як 1000 мм. Також деякі пікселі на зображені можуть мати значення 0, це може бути через неточність роботи камери, тому їм теж треба поставити значення 1000 мм.

Наведемо псевдокод алгоритму обмеження глибини. На вхід алгоритму подається зображення глибини, а на виході буде обрізане зображення глибини.

Algorithm 2.2 Обмеження зображення глибини

Require: DepthImage
  for $row \in DepthImage.rows$ **do**
    for $col \in DepthImage.cols$ **do**
      $depthValue \leftarrow$DepthImage$[row, col]$
      if $(depthValue == 0)$ || $(depthValue > 1000)$ **then**
        $depthValue \leftarrow 1000$
        DepthImage$[row, col] \leftarrow depthValue$
      end if
    end for
  end for
  return DepthImage

2.1.4 Нормалізація зображення глибини

Нормалізація зображення глибини проводиться з тієї ж самою причини, що й для кольорового зображення. Єдина відмінність тільки в тому, що зображення глибини одно канальне. Псевдокод алгоритму нормалізації даного зображення нижче. На вхід алгоритм приймає зображення глибини, мат. очікування, дисперсію та повертає нормалізоване зображення.

Algorithm 2.3 Обмеження зображення глибини

Require: DepthImage, depthMean, depthStd
  for $row \in DepthImage.rows$ **do**
    for $col \in DepthImage.cols$ **do**
      $depthValue \leftarrow$DepthImage$[row, col]$
      $depthValue \leftarrow \frac{depthValue - depthMean}{depthStd}$
      normalizedDepthImage.insert($depthValue$)
    end for
  end for
  return normalizedDepthImage

2.1.5 Підготовка тензора

Вхідний тензор для нейронної мережі детектора це чотирьох канальне зображення, де перші три канали це R, G, B канали, а останній це канал глибини. Тобто це зображення є об'єднанням RGB зображення та зображення глибини. Ці об'єднані зображення пройшли перетворення, які були описані в попередніх підрозділах.

Ці перетворення мають виконуватися в певному порядку. Для кольорового зображення порядок наступний:

1) зміна розміру

2) нормалізація

А для зображення глибини такий:

1) обрізання глибини

2) зміна розміру

3) нормалізація

Після цих перетворень зображення об'єднується і стають одним чотирьох канальним зображенням, що і є вхідним тензором моделі.

На рисунку 2.1 схематично зображено всі етапи обробки зображень для підготовки вхідного тензора моделі детектора.

2.2 Опис датасету

Для навчання моделі скелету був обраний датасет, який описаний у наступній статті [1]. Цей датасет складається з 105,459 пар RGB зображень та зображень глибини. До кожної пари зображень присутня розмітка скелету руки. Цей датасет складається з 45 різних повсякденних дій руками та в яких було також залучено 26 різних предметів. Ці дії згруповані у три різні категорії: дії на кухні, в офісі та соціальні взаємодії. До знімання датасету було залучено 6 людей. Всі люди виконували дії правою рукою та були проінформовані. як виконувати

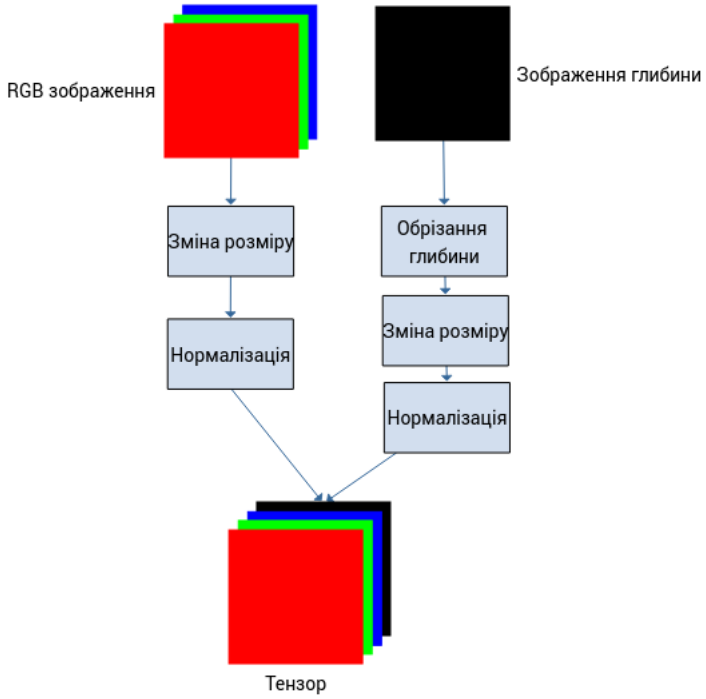

Рисунок 2.1 – Етапи підготовки вхідного тензора детектора

потрібні дії, проте кожен виконував зі своєю швидкістю та артикуляцією, для того, щоб датасет був більш-менш реальний.

Датасет збирався камерою Intel RealSense SR300[12], яка здатна збирати як і кольорові зображення, так і зображення глибини. Зображення були зібрані зі швидкістю 30 кадрів за секунду. Роздільна здатність кольорових зображень складає 1920×1080 та зображень глибини складає 640×480 пікселів. Для отримання якісної розмітки скелету руки, в даному датасеті, були використані шість магнітних сенсорів [13]. Один сенсор прикріплюється на зап'ястя та 5 на кінчики пальців. Кожен датчик задає положення та направлення, та має 6 ступенів свободи. 21 точка скелету руки відновлюється з допомогою інверсної кінематики з цих шести сенсорів.

Кожен сенсор має ширину в 2 мм, тому їх видно на кольоровому зображені, проте на зображені глибини вони непомітні.

У датасеті на кожному зображені присутня рука та розмітка, проте для даного дослідження необхідно натренувати детектор, який буде вміти

видавати ймовірність наявності руки на зображені. Тому було розширено датасет з парами зображень без рук, тобто просто фонові зображення. Зображення збиралися камерою Intel RealSense D435[14] у кількості 10000 пар зображень.

2.3 Підготовка тензора для моделі скелету

Для створення тензора необхідно мати два зображення(RGB зображення та зображення глибини), на яких присутня рука, а також інформацію про розміщення руки на зображені, яку можна отримати використовуючи модель детектора.

Підготовку тензора можна розділити на дві частини, які схожі на підготовку тензора для моделі детектора. Необхідно виконати певні перетворення для RGB - зображення та зображення глибини.

Перетворення над RGB - зображенням:

1) обрізання зображення
2) зміна розміру
3) нормалізація

Перетворення над кольоровим зображенням:

1) обрізання зображення
2) обрізання глибини
3) зміна розміру
4) нормалізація

Ці перетворення наведені далі у підрозділах 2.3.1, 2.3.2, 2.3.3 та 2.3.4.

2.3.1 Обрізання зображення глибини та RGB - зображення

Обрізання зображення необхідно для того, щоб видалити з зображення ті частини, на яких не має руки. Це робиться для того, щоб моделі скелету руки було легше шукати суглоби, бо фон зображення не

буде впливати на роботу. Для обрізання зображення необхідно мати координати, які виходять після роботи детектора руки.

Наведемо псевдокод алгоритму обрізання зображення:

Algorithm 2.4 Обрізання зображення

Require: Image, BoundingBox, imageWidth, imageHeight
$x = BoundingBox.x$
$y = BoundingBox.y$
$w = BoundingBox.w$
$h = BoundingBox.h$
$cropX = x \cdot imageWidth$
$cropWidth = w \cdot imageWidth$
$cropY = y \cdot imageHeight$
$cropHeight = h \cdot imageHeight$
$xMin = max(0, cropX)$
$xMax = min(imageWidth, cropX + cropWidth)$
$yMin = max(0, cropY)$
$yMax = min(imageHeight, cropY + cropHeight)$
$croppedImage = Image[cropY : cropY + cropHeight, cropX : cropX + cropWidth]$
return croppedImage

Алгоритм 4, що наведений вище, однаковий для, що для зображення глибини, що для RGB-зображення.

2.3.2 Зміна розміру зображення

Зміна розміру зображення проводиться таким же чином, як і для детектора. Але тільки вихідний розмір зображення буде 160×160, бо в цьому випадку потрібна саме така розмірність вхідного зображення для моделі скелету. Для зміни розміру кольорового зображення використовується алгоритм передискретизації зображення, в бібліотеці OpenCV для функції cv::resize, це параметр cv::INTER_AREA. Для зміни розміру зображення глибини використовується алгоритм інтерполяції методом найближчого сусіда, параметр функції cv::resize буде cv::INTER_NEAREST.

2.3.3 Обрізання зображення глибини, нормалізація зображень

Обрізання глибини виконується абсолютно таким же чином, як при підготовці тензора для моделі детектора. Значення пікселів, що понад 1000 мм прирівнюються до 1000 мм. Також пікселі, які мають значення 0 мм, теж прирівнюється до 1000 мм.

Нормалізація кольорового зображення та зображення глибини також виконується таким чином, як для моделі детектора. Тільки математичне сподівання та дисперсії будуть нові.

2.3.4 Створення тензору

Вхідний тензор для моделі скелету створюється з RGB - зображення та зображення глибини, після того, як вони пройшли перетоврення, що були описані вище.

Перетоврення мають йти в певному порядку, схожому, як для підготовки тензора для моделі детектора, але додається нове перетворення – вирізання руки з зображення.

Порядок перетворень для кольорового зображення наступний:

1) обрізання зображення
2) зміна розміру
3) нормалізація

Для зображення глибини порядок такий:

1) обрізання зображення
2) обрізання глибини
3) зміна розміру
4) нормалізація

2.4 Архітектура моделі детектора

Для моделі пошуку руки використовується згорткова мережа, яка складається з 5 шарів. Це доволі проста модель, але вона добре працює. Дана модель приймає на вхід тензор, який має розмірність [?, 112, 112, 4], де 112 означає ширину та висоту зображення, 4 кількість каналів, бо 3 три канали йдуть з кольорового зображення та 1 з зображення глибини, а знак '?' означає розмірність бачу(batch). Також модель має другий вхід, який називається $is\_training$. Цей параметр потрібен при нормалізації бачу. Тобто при тренуванні моделі буде йти нормалізація по батчу, а при запуску її без тренування нормалізація буде відбуватися по одному вхідному екземпляру. Параметр $is\_training$ це булевий параметр, який приймає значення $true$, якщо необхідна нормалізація бачу, та $false$, якщо ні.

Як було сказано вище, модель складається з п'яти шарів, де перші чотири шари абсолютно однакові. На цих шарах використовується по 32 згортки розміром 3×3, але тільки в першому шарі використовується 16 згорток. Також в цих згортках використовується падінг, який значення $SAME$ та параметр крок, який в перших 4 шарах має значення 2. В наступному абзаці буде пояснення, що таке падінг та крок в згортці.

Падінг у згортці, задає як поводитися з вікнами, які виходять за границю зображення і приймає два значення: $SAME$ та $VALID$.

При застосуванні згортки, яка має значення падінгу $SAME$, вихідний розмір зображення не зміниться. Тобто це означає, що вхідне зображення буде доповнене рамкою, яка заповнена нулями.

На рисунку 2.2 показано приклад роботи згортки з падінгом $SAME$. Тут видно, що вхідний масив було доповнено нулями для того, щоб вихідний мав таку ж розмірність, що і вхідний масив.

При застосуванні згортки, яка має значення $VALID$, приведе до того, що будуть застосовуватися згортки тільки на тих вікнах, які

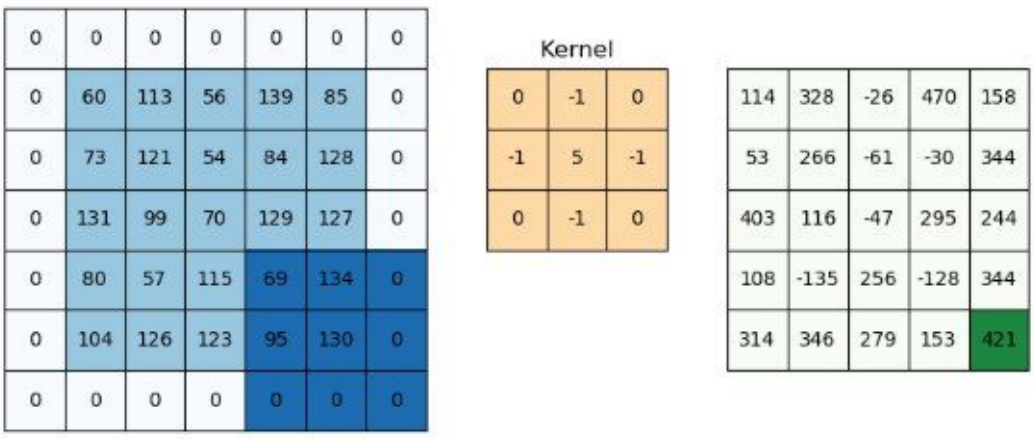

Рисунок 2.2 – Приклад згортки зі значенням падінгу $SAME$

повністю лежать у вхідному масиві. Це означає, що вихідний масив буде мати меншу розмірність, ніж вхідний.

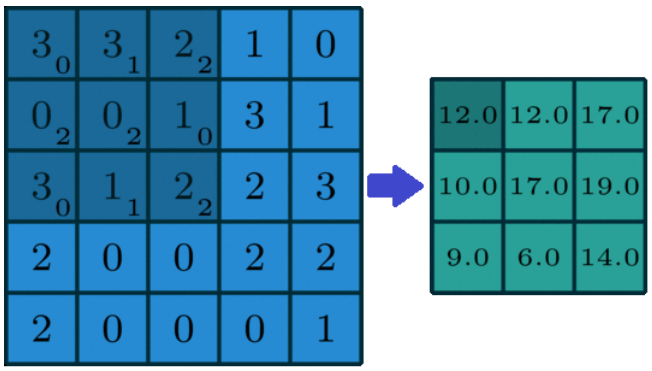

Рисунок 2.3 – Приклад згортки зі значенням падінгу $VALID$

На рисунку 2.3 показано приклад роботи згортки з падінгом $VALID$. Тут показано, що вихідний масив має менший розмір, ніж вхідний після застосування згортки з падінгом $VALID$.

Параметр крок визначає з яким кроком вікно буде рухатися по вхідному масиву та він впливає на вихідний розмір масиву.

Наведемо формули розрахунку вихідного розміру масиву при різних значеннях параметру падінгу та при ріхних значеннях параметру кроку вікна згортки.

Нехай падінг має значення p, де p – задає розмір рамки з нулями, та нехай s – розмір кроку вікна, $n \times n$ – розмір вхідного масиву, $f \times f$ – розмір

згортки, f – непарне число, тоді:

1) при параметрі падінгу $SAME$ розмір вихідного масиву можна розрахувати наступною формулою[15]:

$$\left\lfloor \frac{n+2p-f}{s} \right\rfloor + 1 \times \left\lfloor \frac{n+2p-f}{s} \right\rfloor + 1$$

2) при параметрі падінгу $VALID$ розмір вихідного масиву можна розрахувати наступною формулою[16]:

$$\left\lfloor \frac{n-f}{s} \right\rfloor + 1 \times \left\lfloor \frac{n-f}{s} \right\rfloor + 1$$

П'ятий шар моделі детектора має 32 згортки, але значення падінгу вже $VALID$ і крок згортки складає 1.

Таким чином архітектуру моделі детектора можна подати у вигляді таблиці 2.1, де L – це номер шару мережі, $conv$ – це тип згортки, s – крок вікна згортки, p – тип падінгу згортки, f – це розмір згортки, а N – це кількість згорток на шарі.

Таблиця 2.1 – Архітектура моделі детектора

L	conv	s	p	$f \times f$	N
1	$Conv2D$	2	$same$	3×3	16
2	$Conv2D$	2	$same$	3×3	32
3	$Conv2D$	2	$same$	3×3	32
4	$Conv2D$	2	$same$	3×3	32
5	$Conv2D$	1	$valid$	3×3	32

Модель детектора руки має два виходи: координати обмежувальної зони для двох рук і ймовірності наявності двох рук. Перший вихід складається з 6 чисел, де перші 3 відповідають координатам зони лівої руки, а наступні три відповідно – правій. Координати зони складаються з точки (x, y) та ширини w, за якими можливо відтворити квадратну зону розташування руки. Другий вихід складається з пари чисел, які

показують ймовірність присутності лівої та правої руки на зображені.

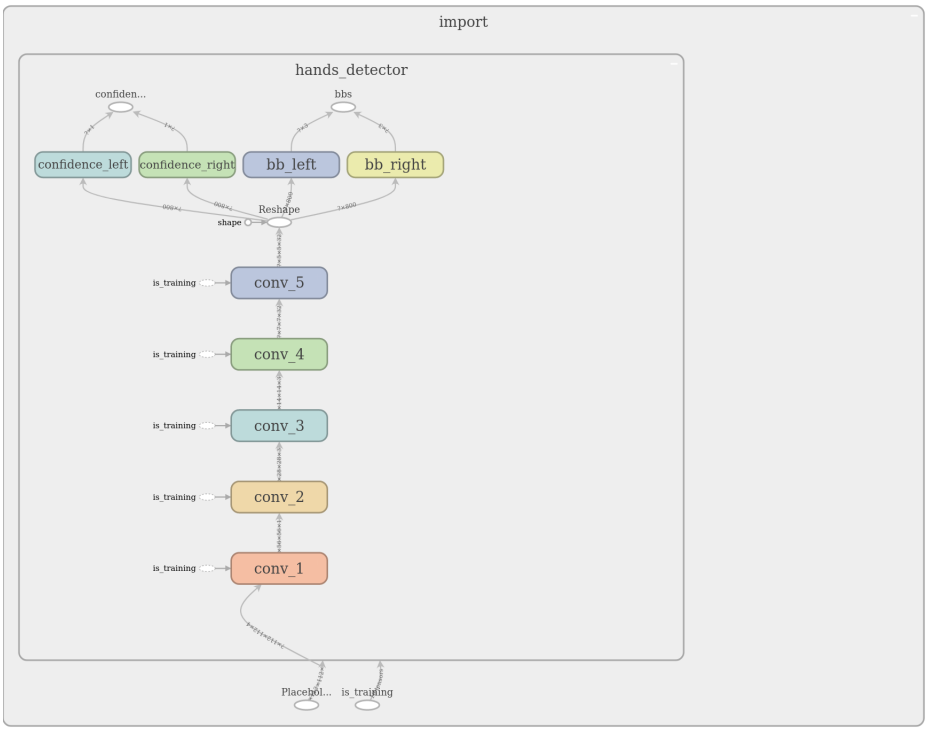

Рисунок 2.4 – Візуалізація графу моделі детектора

На рисунку 2.4 зображено візуалізацію графу моделі детектора.

2.5 Навчання моделі детектора

Навчання нейронної мережі складається з двох етапів: прямого поширення помилки та зворотного поширення помилки. Спочатку нейрона мережа ініціалізується випадковими вагами та виконується пряме поширення помилки. При прямому поширенню помилки, вхідні дані проходять через всю мережу. Кожен прихований шар приймає вхідні дані та обробляє їх, наприклад в моделі детектора це може бути операція $Conv2D$, та ці дані проходять через функцію активації та поступають в наступний прихований шар. Функція активації необхідна, щоб внести нелінійність в нейрону мережу, бо дані можуть мати складну, нелінійну природу. При прямому поширенню помилки вхідні дані проходять через

всю мережу та обраховується вихід, тому це і називається пряме поширення.

В даній моделі для внесення нелінійності використовується активаційна функція:

$$Relu6(x) = min(max(0, x), 6)$$

Після етапу прямого поширення помилки, обраховується значення функції помилки з отриманим виходом, назвемо його *predict* та істинним виходом, назвемо його *target*. В якості функції помилки для моделі детектора використовується метод найменших квадратів, в якому обчислюється сума квадратів відхилень передбачених значень від істинних:

$$Loss(target, predict) = \sum_{i=0}^{N}(target_i - predict_i)^2 = \sum_{i}^{N}(f(input_i) - target_i)^2,$$

де N – кількість екземплярів навчання, f – функція, яка і є нейронною мережею, *input* – вхідне значення нейронної мережі.

Суть навчання нейронної мережі полягає у мінімізації функції помилки, бо чим менше її значення, то тим ближче передбачуване значення до істинного значення. На етапі зворотного поширення помилки мінімізується функція помилки, яка була наведена вище, тим самим перераховуються ваги нейронної мережі. Для мінімізації функції помилки використовується адаптивний метод градієнтного спуску Adam[10]. Для даного методу параметр $\alpha = 0.001, \beta = 0.001$.

Навчання складається з епох, де на кожній епосі відбувається пряме поширення помилки та зворотне поширення помилки. Кількість епох залежить від того, як навчається нейрона мережа. На кожній епосі навчання рахується рівень помилки на тренувальному датасеті та якщо помилка перестає падати, то це означає, що модель навчилася. Крім тренувального датасету є ще валідаційний. На ньому також, які і на датасеті для тренування рахується помилка, але відмінність у тому, що

валідаційний датасет не бере участі в тренуванні нейронної мережі. Його основна задача показувати, як поводитися нейрона модель на абсолютно нових для неї даних. Також модель може перетренуватися і показувати гарні метрики на тренувальному датасеті, коли насправді це не так. Помилка на валідаційному датасеті покаже чи це так насправді.

2.5.1 Генерація bounding box для навчання

В датасеті, який використовується для навчання, присутня тільки 3d розмітка для скелета. Але для навчання моделі детектора, необхідно мати координати квадратної зони знаходження руки. Тому потрібно згенерувати ці координати. Для цього спочатку необхідно перевести 3d розмітку руки у 2d + d, тобто координати по X та Y перенести в площину зображення, а глибину так і залишити. Координати генеруються наступним чином: шукаються максимальна координата по X та Y, а також мінімальна координата також по X та Y. Мінімальна координата по X та Y буде новим значенням для зони, а ширина буде різницею між максимальною та мінімальною координатою по X, а висота звісно різниця між максимальною та мінімальною координатою по Y. Ширина сторони вибирається як максимальна між порахованою шириною та висотою. Але до порахованих значень додається ще падінг для того, щоб зона знаходження руки не була впритул до руки.

Наведемо псевдокод алгоритму генерації зони знаходження руки:

2.6 Результати навчання моделі детектора

У таблиці 2.2 наведено результати моделі детектора за останні 30 епох. В таблиці присутні значення наступних метрик: IoU, AP[11], точність та повнота для випадку коли рука дійсно на зображені, та точність та повнота, коли насправді рука відсутня на зображені.

Algorithm 2.5 Генерація bounding box

Require: points, padding, width, height
$xList = points[:, 0]$
$yList = points[:, 1]$
$xMin = min(xList)$
$xMax = max(xList)$
$yMin = min(yList)$
$yMax = max(yList)$
$x = xMin$
$y = yMin$
$w = xMax$
$h = yMax$
$x = x - w \cdot padding$
$y = y - h \cdot padding$
$w = w \cdot (1 + 2 \cdot padding)$
$h = h \cdot (1 + 2 \cdot padding)$
$w = max(w, h)$
$x_2 = min(x + w, width)$
$y_2 = min(y + w, height)$
$x = max(x, 0)$
$y = max(y, 0)$
$w = x_2 - x$
$w = min(w, y_2 - y)$
return $[x, y, w]$

На рисунку 2.5 показано приклад роботи детектора на RGB зображені та на рисунку 2.6 приклад роботи на зображені глибини.

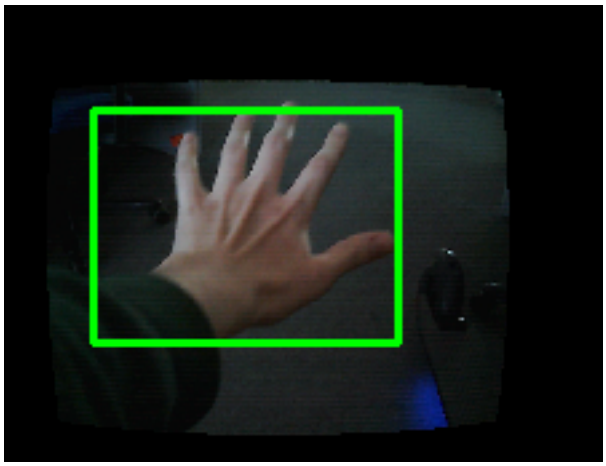

Рисунок 2.5 – Приклад роботи детектора на RGB зображені

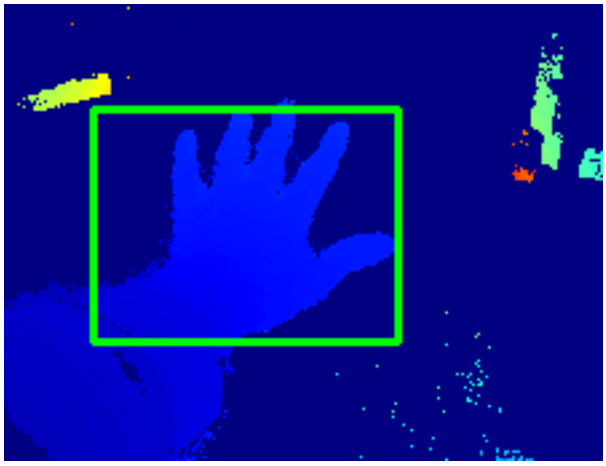

Рисунок 2.6 – Приклад роботи на зображені глибини

2.7 Довірчі інтервали оцінок навчання

В даному підрозділі буде оцінено довірчий інтервал для математичного оцінювання AP та IOU метрик. Вибірка для цих метрик складається з оцінок AP та IOU метрик на останніх 30 епохах навчання, бо оцінки вже більш-менш схожі.

Порахуємо довірчий інтервал математичного очікування для AP метрик за останні 30 епох навчання. Нехай AP оцінки на 30 останніх епохах мають нормальний розподіл та незалежні. Виберемо поріг належності математичного очікування певному інтервалу з ймовірністю $P = 0.95$, тоді рівень значущості $\alpha = 0.05$.

Порахуємо вибіркове середнє для вибірки:

$$\overline{X} = 0.88273$$

та вибіркову дисперсію:

$$S^2 = 0.0000098162$$

Таким чином середнє відхилення:

$$S = 0.0031330865$$

Квантиль розподілу Стюдента

$$t_{1-\frac{\alpha}{2}} = t_{0.975} = 2.0423$$

, при $n = 30$.

Порахуємо відхилення математичного очікування:

$$\varepsilon = t_{1-\frac{\alpha}{2}}(n)\frac{S}{\sqrt{n}} = \frac{2.0423 \cdot 0.0031330865}{\sqrt{30}} = 0.0011682379$$

Таким чином довірчий інтервал для математичного очікування для AP метрики:

$$M \in (\overline{X} - \varepsilon; \overline{X} + \varepsilon) = (0.8815620954290008; 0.8838985712376656)$$

Порахуємо тепер довірчий інтервал математичного очікування для IOU метрики. Нехай, як і в попередньому випадку, що IOU оцінки за останні 30 епох мають нормальний розподіл та незалежні. Знайдемо довірчий інтервал в який входить математичне очікування з ймовірністю $P = 0.95$, звідси рівень значущості $\alpha = 0.05$.

Вибіркове середнє для вибірки з IOU метрик:

$$\overline{X} = 0.750118333$$

та вибіркова дисперсія:

$$S^2 = 0.0000290792$$

, звідси середнє відхилення дорівнює

$$S = 0.0053925114$$

Квантиль розподілу Стюдента

$$t_{1-\frac{\alpha}{2}} = t_{0.975} = 2.0423$$

, при $n = 30$.

Порахуємо відхилення математичного очікування:

$$\varepsilon = t_{1-\frac{\alpha}{2}}(n)\frac{S}{\sqrt{n}} = \frac{2.0423 \cdot 0.0053925114}{\sqrt{30}} = 0.0020107125$$

Отже, довірчий інтервал для математичного очікування вибірки з IOU метрик:

$$M \in (\overline{X} - \varepsilon; \overline{X} + \varepsilon) = (0.748107620790787; 0.7521290458758795)$$

Довірчі інтервали для оцінок AP та IoU є невеликими, що свідчить про те, що оцінки збігаються до одного значення за останні 30 епох навчання в обох випадках. Ці оцінки свідчать про непогані результати застосування для практичного використання в застосунках віртуальної та доповненої реальності.

2.8 Конвертація точок скелету

Для навчання моделі використовується скелет, який зконвертований з 3D координат в 2D + D. Це означає, що координати по X та Y мають бути переведені з координат реального світу в координати зображення. Тобто координати по X та Y будуть вимірюватися не в міліметрах, а в пікселях зображення.

Для проєктування 3D точок на зображення необхідно мати матрицю параметрів калібрування камери(далі матриця камери) та матрицю дисторсії, матрицю повороту та матрицю паралельного перенесення. Матриця камери має параметри фокальної відстані по X та Y координаті та координати головної точки v_0, u_0, яка знаходиться в центрі камери.

Калібровочна матриця має наступний вигляд[17]:

$$\mathrm{K} = \begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Для проєктування 3D точок на площину зображення використовуються наступні формули[17]:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + T$$

$$x' = \frac{x}{z}$$

$$y' = \frac{y}{z}$$

$$u = f_x \cdot x' + u_0$$

$$v = f_y \cdot y' + v_0,$$

де (X, Y, Z) – це координати 3D точки, f_x, f_y, u_0, v_0 – параметри камери, що є в калібровачній матриці, R – матриця повороту, T – матриця паралельного перенесення.

Для проєктування 3D точок в роботі використовувалися функція cv2.projectPoints з бібліотеки OpenCV. Значення матриць було взято з репозиторію[18] датасету First Person Hand Action, який використовується для навчання моделей. Матриця дисторсій заповнена нулями.

Для конвертації 2D + D точки назад в 3D використовується функція cv2.undistortPoints з бібліотеки OpenCV. Конвертація виглядає наступним чином[17]:

$$(x', y') = undistortPoints(X, Y, K, D)$$

$$x = x' \cdot z$$

$$y = y' \cdot z,$$

де (x, y, z) – це 3D точка, (X, Y) – це 2D точка, K – матриця камери, D – матриця дисторсії, яка заповнена нулями.

2.9 Навчання моделі скелета

Для навчання нейронної мережі скелета використовується архітектура MobileNetV2, яка описана в першій частині даної роботи. Єдина змінна в даній архітектурі тільки на останньому шарі мережі. Останній шар даної моделі це повнозв'язний шар. Даний шар потрібен тому, що ця мережа повинна передбачувати 3D точки скелета, а не розв'язувати задачу класифікації. На вхід модель приймає два тензори.

Перший тензор має розмірність [?, 160, 160, 4], де ? – це розмірність батчу при навчанні, та являє собою зображення глибини та кольорове зображення на якому знаходиться рука обрізана після детектора. Тобто дане зображення це контент, який знаходиться в середині зони, яку знайшов детектор руки. Це потрібно для того, щоб модель скелета не отримувала зайву інформацію на зображені, а тільки кисть руки.

Другий тензор показує, який самий скелет повинен бути знайдений на зображені, лівий чи правий. Розмірність другого тензору складає [1, 1].

Для навчання моделі скелету використовувався, як і при навчанні моделі детектора, адаптивний метод градієнтного спуску Adam[10] з параметром навчання $\alpha = 0.001$ та $\beta = 0.001$.

2.10 Результати навчання моделі скелета

Для оцінки навчання моделі скелету використовується L_2 метрика, яка показує на скільки міліметрів точність передбаченого скелету відрізняється від істинного скелета. Для оцінки скелету використовується валідаційний датасет, де для кожного екземпляру датасету рахується L_2

та потім обраховується середнє L_2 для всього валідаційного датасету.

У таблиці 2.3 представлено результати моделі скелету, а саме L_2 метрику за останні 40 епох навчання. На рисунку 2.7 можна побачити результат передбачення скелету на кольоровому зображені, а на рисунку 2.8 на відповідному йому зображенню глибини.

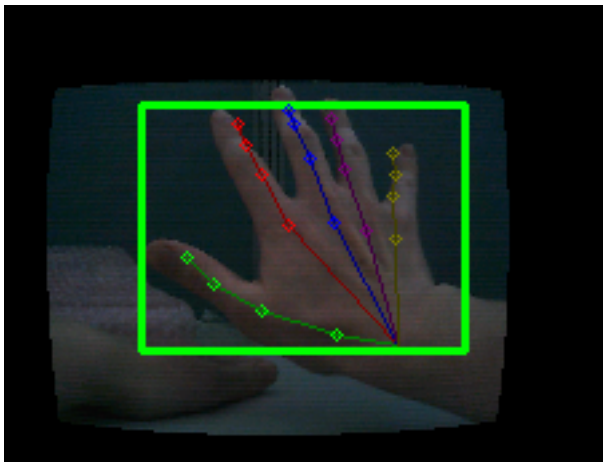

Рисунок 2.7 – Передбачений скелет на RGB зображені

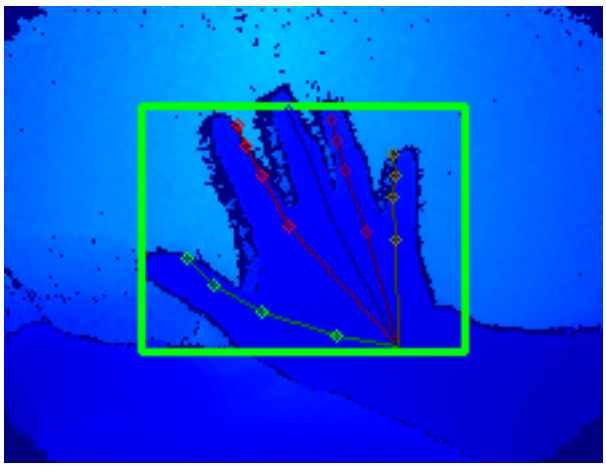

Рисунок 2.8 – Передбачений скелет на зображені глибини

2.11 Обрахунок довірчого інтервалу для L_2 метрики моделі скелету

В даному підрозділі буде пораховано довірчий інтервал для L_2 оцінок з 40 останніх епох навчання моделі скелету. Вважаємо, що оцінки незалежні та мають нормальний розподіл.

Знайдемо довірчий інтервал в який входить математичного оцінювання з ймовірністю $P = 0.95$, тоді рівень значущості дорівнює $\alpha = 0.05$. Вибіркове середнє дорівнює:

$$\overline{X} = 19.1013305$$

Та вибіркова дисперсія дорівнює:

$$S^2 = 24.7059137611382$$

, та звідси середнє відхилення:

$$S = 4.970504376935826$$

Квантиль при $n = 40$ та $\alpha = 0.005$ дорівнює:

$$t_{0.975} = 2.0211$$

Отже, можна тепер порахувати відхилення:

$$\varepsilon = \frac{t_{0.975} \cdot S}{\sqrt{n}} = 1.5883941063685867$$

Таким чином довірчий інтервал для математичного оцінювання наступний:

$$M_{L_2} \in (17.512936393631414, 20.689724606368586)$$

З даного довірчого інтервалу видно, що математичне сподівання для

оцінок L_2 лежить у доволі великому інтервалі, який складає приблизно 3 мм. За останні 40 епох навчання, найгірший результат був 28.5 мм, а найкращий – 14.43 мм. Тобто L_2 оцінка не збігається до одного значення. Але ця оцінка є більш-менш задовільною для практичних задач віртуальної та доповненої реальності, але не для більш серйозних задач. Цю оцінку можливо вийде покращити продовживши навчання далі.

2.12 Опис роботи жестів

Основна мета даної роботи натренувати класифікатор жестів. Для цього необхідно мати модель детектора та модель скелета руки. Модель детектора необхідна, щоб видиліти зону розташування руки на зображені та зменшити зображення для моделі скелета, щоб моделі скелета було легше видиліти основну інформацію про руку. Суть моделі скелета полягає у тому, щоб знайти 21 суглоб кисті руки за яким можна відновити скелет.

Все це необхідно для того, щоб можна було побудувати модель класифікатора жестів. Мета даної роботи класифікувати динамічні жести, тобто жести які знаходяться у певній дії. Динамічний жест складається з 25 послідовних скелетів. Вигода цього полягає у тому, що динамічний жест класифікується не на 25 зображеннях, які мають велику розмірність, а на 25 3D скелетах, тому звідси вхідний тензор буде мати розмірність $[25, 21, 3]$. Через те, що класифікатор жестів приймає на вхід 3D скелет, то модель класифікатора буде дуже простою, через те, що с основна інформація про руку вже отримана попередніми моделями детектора та скелета.

В даній роботі будується класифікатор жестів, який має вміти розпізнавати наступні жести: «swipe», «home» та «other».

Жест «swipe» – це рух розкритої долоні зліва направо чи навпаки. В доповненій реальності його можна застосовувати, як рух закриття об'єктів, що знаходяться на екрані. На рисунку 2.9 зображено жест «swipe» на 3

зображеннях з послідовності 25 кадрів.

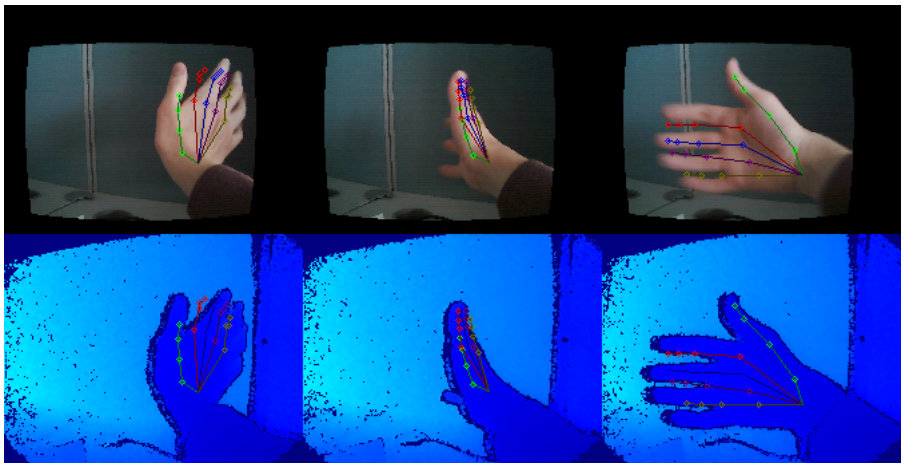

Рисунок 2.9 – Приклад жесту «swipe» на RGB зображені та зображені глибини

Жест «home» – це рух кисті руки з положення закритого кулака, який переходить в відкриту долоню. В доповненій реальності його можна було б використовувати, як жест, який викликає об'єкт. Тобто цей жест є протилежним до жесту «swipe». На рисунку 2.10 зображено жест «home» на 3 зображеннях з послідовності 25 жестів.

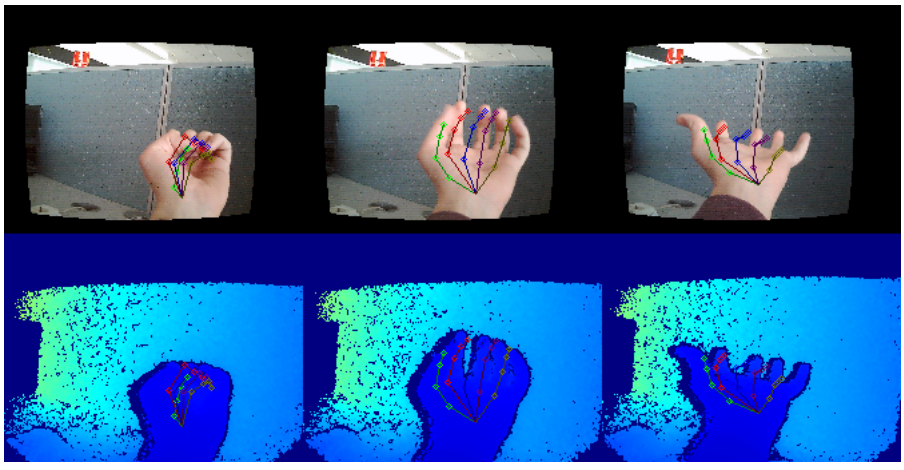

Рисунок 2.10 – Приклад жесту «home» на RGB зображені та зображені глибини

2.13 Архітектура моделі класифікатора жестів та навчання

Для побудови моделі класифікатора використовується згорткова мережа, як і в моделі детектора руки та моделі скелета. Дана мережа дуже маленька та складається всього з трьох згорток.

Дана модель на вхід приймає тензор, який складається просто з 25 скелетів у 3D, тобто вхідний тензор має розмірність $[batch\_size, 25, 21, 3]$, де $batch\_size$ – це розмір батчу. На перших трьох шарах використовується 2D згортка, але з різними параметрами. На першому шарі згортка має наступні параметри: кількість фільтрів дорівнює 64, розмір згортки $(1, 5)$, значення падінгу згортки $VALID$, значення кроку 1 та активаційна функція $Relu6$. Другий шар має такі ж параметри, як і перший, але тільки кількість фільтрів 32. На третьому шарі кількість згорток вже 16 та активаційна функція вже не використовується. Далі використовується шар сплющення(з англ. flatten), який сплющує згортки в вектор стовпчик та після цього застосовується повнозв'язний шар на виході якого буде 4 елементи. На рисунку 2.11 зображено дію двох останніх шарів, на першому показано як сплющуються згортки, а на другому показано повнозв'язний шар.

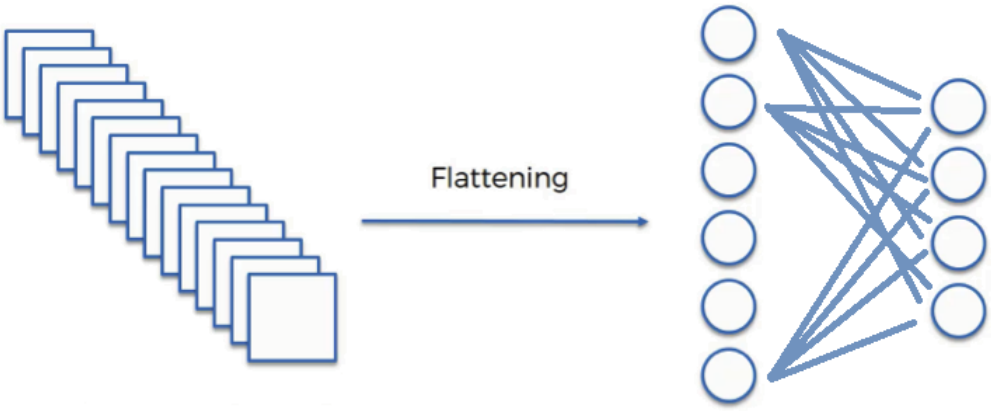

Рисунок 2.11 – Останній шар моделі класифікатора

Вихід моделі класифікатора складається з вектора в якому записані

числа, які показують з якою ймовірністю був розпізнаний той чи інший жест. В таблиці 2.4 показана архітектура моделі класифікатора з усіма параметрами, де s – значення кроку, p – тип падінгу, kernel size – розмір згортки, L – номер шару, N – кількість згорток на шарі.

На рисунку 2.12 зображено візуалізацію моделі класифікатора жестів фреймворком tensorboard.

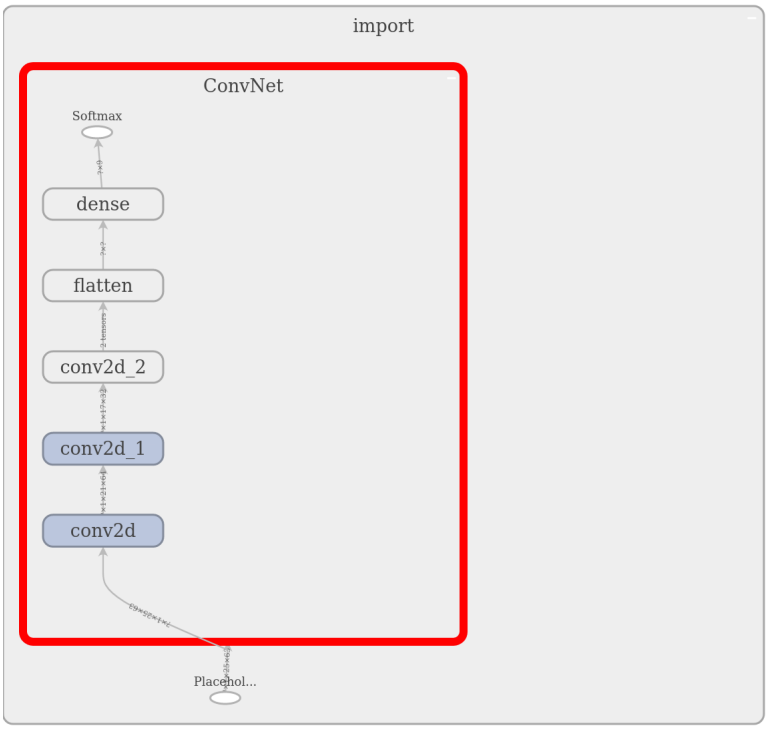

Рисунок 2.12 – Візуалізація моделі класифікатора жестів

При навчанні моделі використовується адаптивний метод градієнтного спуску Adam, такий самий, як і при навчанні моделі детектора та моделі скелета. Параметри Adam для моделі класифікатора $\alpha = 0.001$ та $\beta = 0.001$.

При навчанні моделі класифікатора жестів використовувався метод кросс-валідації, бо датасет з жестами не дуже великий. За один цикл роботи кросс-валідації датасет розбивається на дві частини: на тренувальний датасет та валідаційний датасет. Датасет розбивався випадковим чином на тренувальний піддатасет та валідаційних

піддатасет на кожному циклі. У тренувальному датасеті знаходиться 80% екземплярів, а валідаційному відповідно 20% екземплярів з основного датасету.

В датасеті представлено 571 екземпляр жесту «swipe», 540 екземплярів жесту «home» та 558 екземплярів жесту «other».

2.14 Результати класифікації жестів та обрахунок довірчих інтервалів для цих оцінок

В даному підрозділі наведено результати класифікатора жестів за 30 останніх епох навчання. Пораховано 4 різні оцінки, де «Main Acc.» в таблиці 2.5 – це загальна точність розпізнавання жестів, оцінка «Home» означає точність розпізнавання жесту «home», «Swipe» та «Other» – відповідно точніть розпізнавання жестів «swipe» та «other».

Порахуємо довірчий інтервал для оцінок основної точності розпізнавання жестів(Main Acc.), точності розпізнавання жесту «home», точності розпізнавання жесту «swipe» та точності розпізнавання жесту «other». Будемо шукати довірчий інтервал в який входить математичне оцінювання з ймовірністю $P = 0.95$ та рівнем значущості $\alpha = 0.05$.

Порахуємо вибіркове математичне очікування для оцінок:

$$\overline{X}_{MainAcc.} = 0.962699$$

– вибіркове мат. оцінювання точності розпізнавання жестів

$$\overline{X}_{Home} = 0.971753$$

– вибіркове мат. оцінювання точності розпізнавання жесту «home»

$$\overline{X}_{Swipe} = 0.925469$$

– вибіркове мат. оцінювання точності розпізнавання жесту «swipe»

$$\overline{X}_{Other} = 0.966723$$

– вибіркове мат. оцінювання точності розпізнавання жесту «other»

Знайдемо тепер вибіркову дисперсію та вибіркове середнє для оцінок:

1) вибіркова дисперсія та середнє для оцінок точності розпізнавання жестів:

$$\sigma^2_{MainAcc.} = 0.000109$$

$$\sigma_{MainAcc.} = 0.010482$$

2) вибіркова дисперсія та середнє для оцінки точності розпізнавання жесту «home»:

$$\sigma^2_{Home} = 0.0002$$

$$\sigma_{Home} = 0.014282$$

3) вибіркова дисперсія та середнє для оцінки точності розпізнавання жесту «swipe»:

$$\sigma^2_{Swipe} = 0.00044$$

$$\sigma_{Swipe} = 0.02098$$

4) вибіркова дисперсія та середнє для оцінки точності розпізнавання жесту «other»:

$$\sigma^2_{Other} = 0.00022$$

$$\sigma_{Other} = 0.01499$$

Тепер можна порахувати відхилення для математичного оцінювання. Квантиль розподілу Стюдента

$$t_{1-\frac{\alpha}{2}} = t_{0.975} = 2.0423$$

, при $n = 30$.

1) Для основної оцінки точності розпізнавання жестів:

$$\varepsilon_{MainAcc.} = 0.003908$$

2) Для оцінки точності розпізнавання жесту «home»:

$$\varepsilon_{Home} = 0.005325$$

3) Для оцінки точності розпізнавання жесту «swipe»:

$$\varepsilon_{Swipe} = 0.007826$$

4) Для оцінки точності розпізнавання жесту «other»:

$$\varepsilon_{Other} = 0.0055908$$

Тепер нарешті можна отримати довірчі інтервали для математичних сподівань для всіх оцінок розпізнавання:

1) довірчий інтервал для мат. сподівання оцінки точності розпізнавання жестів

$$M_{MainAcc.} \in (0.9587, 0.966)$$

2) довірчий інтервал для мат. сподівання оцінки точності розпізнавання жесту «home»:

$$M_{Home} \in (0.966, 0.977)$$

3) довірчий інтервал для мат. сподівання оцінки точності розпізнавання жесту «swipe»:

$$M_{Swipe} \in (0.9176, 0.9332)$$

4) довірчий інтервал для мат. сподівання оцінки точності

розпізнавання жесту «other»:

$$M_{Other} \in (0.9611, 0.9723)$$

Довірчі інтервали для оцінок точності розпізнавання для всіх жестів вийшли невеликими, що свідчить про те, що оцінки сходять до одного значення за останні епохи та модель навчилася. До того ж, за цими оцінками слідує, що модель класифікатора має гарну точність розпізнавання жестів. Найгірше розпізнається жест «swipe», бо можливо в датасеті є схожі жести класу «other» та моделі важко їх відрізнити один від одного.

Висновки до розділу 2

В даному розділі розглядається перетворення над зображеннями, які потрібні для створення вхідного тензора моделі детектора. Описується та наводиться псевдокод для нормалізації зображень, для обрізання зображення глибини. Та описується яким чином змінюється розмір зображень. Також розроблено архітектуру моделі детектора руки та описано перебіг навчання детектора. Також описано датасет, який використовувався для навчання моделі детектора та моделі скелета. Наведено результати роботи моделі детектора, а саме наведено IOU та AP метрики за останні 30 епох навчання, та пораховано довірчі інтервали для математичних сподівань цих оцінок. Довірчі інтервали для оцінок вийшли невеликими, що свідчить, що оцінки збіглися та не стрибають між епохами навчання. Також в даному розділі розглядається перетворення над вхідними даними для моделі скелета і описується процес навчання моделі скелета. Отримано результати L_2 метрик за останні 40 епох навчання та пораховано довірчий інтервал для математичного сподівання L_2 метрики моделі скелета. Довірчий інтервал вийшов достатньо великим, що говорить про те, що модель недовчилася. Це можна виправити подальшим її навчанням чи покращенням датасету.

Та нарешті описується робота жестів та самі жести, які використовувалися. Розроблено та навчена архітектура моделі класифікатора жестів. Отримано результати точності розпізнавання жестів, а саме загальну точність по всім жестам, по жесту «home», «swipe» та «other» за останні 30 епох навчання. Для математичних сподівань цих оцінок пораховано довірчі інтервали, які вийшли не великими, що свідчить гарне розпізнавання на практиці. За цими результатами можна свідчити, що модель класифікатора може мати гарне практичне застосування у застосунках віртуальної та доповненої реальності.

Таблиця 2.2 – Результати моделі детектора

Епоха	AP	IOU	Точність+	Точність-	Повнота+	Повнота-
1971	0.88016	0.74864	0.979	0.932	0.936	0.978
1972	0.88371	0.75041	0.980	0.946	0.950	0.979
1973	0.8781	0.73905	0.972	0.941	0.945	0.970
1974	0.88199	0.742	0.982	0.942	0.946	0.981
1975	0.88513	0.74954	0.978	0.944	0.948	0.976
1976	0.8822	0.74385	0.985	0.938	0.941	0.984
1977	0.89491	0.75069	0.980	0.942	0.946	0.979
1978	0.88081	0.74501	0.971	0.930	0.934	0.969
1979	0.88497	0.75098	0.981	0.943	0.947	0.980
1980	0.88264	0.74947	0.979	0.948	0.952	0.977
1981	0.88262	0.75015	0.977	0.951	0.954	0.975
1982	0.87724	0.74529	0.975	0.930	0.934	0.974
1983	0.88335	0.7569	0.979	0.939	0.943	0.977
1984	0.8787	0.74056	0.973	0.926	0.930	0.971
1985	0.88091	0.75489	0.983	0.940	0.943	0.982
1986	0.88261	0.74463	0.982	0.942	0.946	0.981
1987	0.88118	0.75717	0.972	0.942	0.946	0.970
1988	0.8827	0.74944	0.977	0.944	0.948	0.976
1989	0.88172	0.74706	0.978	0.939	0.943	0.977
1990	0.88588	0.75861	0.980	0.941	0.944	0.979
1991	0.88217	0.74732	0.978	0.929	0.932	0.976
1992	0.88156	0.75359	0.981	0.934	0.938	0.980
1993	0.88047	0.75072	0.977	0.939	0.943	0.975
1994	0.88385	0.75289	0.979	0.943	0.947	0.978
1995	0.88418	0.75342	0.979	0.940	0.944	0.978
1996	0.88534	0.75899	0.984	0.933	0.937	0.983
1997	0.88283	0.74884	0.979	0.949	0.952	0.977
1998	0.88088	0.74846	0.987	0.934	0.937	0.987
1999	0.88367	0.7553	0.981	0.942	0.946	0.979
2000	0.88543	0.75968	0.981	0.935	0.939	0.979

Таблиця 2.3 – Результати моделі скелета

Епоха	L_2	Епоха	L_2
1732	16.02268	1733	16.57841
1734	25.04232	1735	41.73027
1736	15.9145	1737	15.95985
1738	16.68848	1739	20.18343
1740	15.43058	1741	22.00369
1742	28.55571	1743	14.43221
1744	21.18224	1745	15.5863
1746	26.43394	1747	15.65418
1748	21.97506	1749	16.34971
1750	15.40928	1751	23.53944
1752	18.24147	1753	17.9104
1754	15.33235	1755	19.86008
1756	17.71897	1757	15.35627
1758	20.55525	1759	21.40121
1760	17.07145	1761	17.64623
1762	17.17638	1763	19.13297
1764	22.45797	1765	17.80642
1766	14.78387	1767	18.67771
1768	16.39497	1769	19.65824
1770	15.36923	1771	16.83951

Таблиця 2.4 – Архітектура моделі класифікатора жестів

L	Шар	s	p	kernel size	N
1	$Conv2D$	1	$VALID$	1×5	64
2	$Conv2D$	1	$VALID$	1×5	32
3	$Conv2D$	1	$VALID$	1×5	16
4	Flatten	–	–	–	–
5	Dense	–	–	–	–

Таблиця 2.5 – Результати класифікації жестів

Епоха	Main Acc.	Home	Swipe	Other
171	0.972	0.9649	0.9245	0.9797
172	0.974	0.9825	0.9057	0.9822
173	0.972	0.9825	0.9245	0.9772
174	0.968	0.9643	0.9038	0.9773
175	0.972	0.9649	0.9057	0.9822
176	0.954	0.9825	0.9434	0.9518
177	0.972	0.9821	0.9245	0.9772
178	0.958	0.9821	0.9231	0.9596
179	0.964	0.9643	0.9245	0.9696
180	0.966	0.9649	0.9231	0.9722
181	0.966	0.9649	0.8868	0.9772
182	0.956	0.9825	0.9057	0.9594
183	0.960	0.9825	0.9231	0.9620
184	0.968	0.9825	0.9623	0.9670
185	0.966	0.9643	0.9245	0.9722
186	0.962	0.9821	0.9434	0.9620
187	0.950	0.9825	0.9245	0.9492
188	0.970	0.9821	0.8679	0.9823
189	0.942	0.9821	0.9434	0.9367
190	0.968	0.9649	0.9423	0.9722
191	0.964	0.9821	0.9245	0.9671
192	0.956	0.9649	0.9434	0.9569
193	0.960	0.9643	0.9057	0.9671
194	0.976	0.9825	0.9245	0.9822
195	0.978	0.9643	0.9245	0.9873
196	0.931	0.9649	0.9623	0.9213
197	0.964	0.9825	0.9434	0.9645
198	0.950	0.9298	0.9615	0.9519
199	0.952	0.9821	0.9245	0.9519
200	0.970	0.9298	0.9231	0.9823

ВИСНОВКИ

В даній роботі проаналізовано метрики, які застосовуються при оцінці якості алгоритмів розпізнавання та классифікації. Також розглянуто архітектуру нейроної мережі MobileNetV2, яка використовується в роботі при навчані моделі скелета.

Розроблено та реалізовано архітектуру моделі дектора кисті руки, яка видає ймовірнсть наявності руки на зображені та пкоазує зону знаходження руки. Ця модель навчена на датасеті First Person Hand Action та отримано результати за останні епохи навчання. Пораховано довірчі інтервали для математичного сподівання отриманих результатів, які вийшли невеликими, що говорить про те, що ці результати є правдоподібними та модель має достатню якість розпізнавання для застосування в застосунках.

Навчено модель скелету та отримано результати L_2 метрики за останні епохи навчання. Ці результати є досить різноманітними та порахований довірчий інтервал для математичного сподівання цих оцінок вийшов достатньо великий, що свідчить, що модель недовчилася. Покращити її результати можна продовживши навчання чи покращенням датасету, наприклад збільшенням. Проте, для подальшої задачі класифікації жестів даної якості скелета вистачає.

Розроблено та навчено модель класифікатора жестів. Отримано результати точності розпізнавання всіх жестів в цілому та для кожного жесту окремо. Пораховано довірчі інтервали для математичного сподівання для отриманих оцінок точності та ці інтервали вийшли невеликими, що каже, що дані оцінки правдоподібні. Та дана модель є приємлема для класифікації жестів в застосунках віртуальної та доповненої реальності.

ПЕРЕЛІК ПОСИЛАНЬ

1. First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations. Guillermo Garcia-Hernando, Shanxin Yuan, Seungryul Baek, Tae-Kyun Kim [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1704.02463.

2. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1704.04861.

3. MobileNetV2: Inverted Residuals and Linear Bottlenecks. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1801.04381.

4. Metrics to Evaluate your Machine Learning Algorithm [Електронний ресурс]. — Режим доступу: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

5. Accuracy, Precision, Recall or F1? [Електронний ресурс]. — Режим доступу: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

6. What is the Jaccard Index? [Електронний ресурс]. — Режим доступу: https://deepai.org/machine-learning-glossary-and-terms/jaccard-index.

7. Intersection over Union (IoU) for object detection [Електронний ресурс]. — Режим доступу: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

8. Deep Residual Learning for Image Recognition. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1512.03385.

9. OpenCV [Електронний ресурс]. — Режим доступу: https://opencv.

org/

10. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. Diederik P. Kingma, Jimmy Lei Ba [Електронний ресурс]. — Режим доступу: https://arxiv.org/pdf/1412.6980.pdf.

11. mAP (mean Average Precision) for Object Detection. Jonathan Hui [Електронний ресурс]. — Режим доступу: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

12. Intel Realsense SR300 [Електронний ресурс]. — Режим доступу: https://software.intel.com/en-us/articles/introducing-the-intel-realsense-camera-sr300

13. NDItrakSTAR [Електронний ресурс]. — Режим доступу: –https://www.ndigital.com/about/ascension-technology-corporation/

14. Intel Realsense D435 [Електронний ресурс]. — Режим доступу: https://www.intelrealsense.com/depth-camera-d435/

15. MachineLearning Same convolution [Електронний ресурс]. — Режим доступу: https://machinelearning.wtf/terms/same-convolution/

16. MachineLearning Valid convolution [Електронний ресурс]. — Режим доступу: https://machinelearning.wtf/terms/valid-convolution/

17. Camera Calibration and 3D Reconstructions [Електронний ресурс]. — Режим доступу: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

18. First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations [Електронний ресурс]. — Режим доступу: https://github.com/guiggh/hand_pose_action

ДОДАТОК А ТЕКСТИ ПРОГРАМ

А.1 Код обрахунку довірчих інтервалів

```
import math as m

def conf_interval(data):
    """
    Estimate confidence interval for input data
    :param data: array of data
    """
    mean = sum(data) / len(data)
    print("mean = {}".format(mean))
    std = sum([(datum - mean) ** 2 for datum in data]) / \
          (len(data) - 1)
    print("std = {}".format(std))
    std2 = m.sqrt(std)
    print("std2 = {}".format(std2))
    if len(data) == 30:
        t = 2.0423
    elif len(data) == 40:
        t = 2.0211
    else:
        raise RuntimeError
    eps = t * std2 / m.sqrt(len(data))
    print("eps = {}".format(eps))
    print("Conf interal = ({}, {})".format(mean - eps,
                                           mean + eps))

if __name__ == '__main__':
```

```
APs = [0.88016, 0.88371, 0.8781, 0.88199, 0.88513,
       0.8822, 0.89491, 0.88081, 0.88497, 0.88264,
       0.88262, 0.87724, 0.88335, 0.8787, 0.88091,
       0.88261, 0.88118, 0.8827, 0.88172, 0.88588,
       0.88217, 0.88156, 0.88047, 0.88385, 0.88418,
       0.88534, 0.88283, 0.88088, 0.88367, 0.88543]

IOU = [0.74864, 0.75041, 0.73905, 0.742, 0.74954,
       0.74385, 0.75069, 0.74501, 0.75098, 0.74947,
       0.75015, 0.74529, 0.7569, 0.74056, 0.75489,
       0.74463, 0.75717, 0.74944, 0.74706, 0.75861,
       0.74732, 0.75359, 0.75072, 0.75289, 0.75342,
       0.75899, 0.74884, 0.74846, 0.7553, 0.75968]

L2 = [16.02268, 16.57841, 25.04232,
      41.73027, 15.9145, 15.94985, 16.68848, 20.18343,
      15.43058, 22.00369, 28.55571, 14.43221, 21.18224,
      15.5863, 26.43394, 15.65418, 21.97506, 16.34971,
      15.40928, 23.53944, 18.24147, 17.9104, 15.33235,
      19.86008, 17.71897, 15.35627, 20.55525, 21.40121,
      17.07145, 17.64623, 17.17638, 19.13297, 22.45797,
      17.80642, 14.78387, 18.67771, 16.39497, 19.65823,
      15.36923, 16.83951]

main_accuracy = [0.972, 0.974, 0.972, 0.968, 0.972,
                 0.954, 0.972, 0.958, 0.964, 0.966,
                 0.966, 0.956, 0.960, 0.968, 0.966,
                 0.962, 0.950, 0.970, 0.942, 0.968,
                 0.964, 0.956, 0.960, 0.976, 0.978,
                 0.931, 0.964, 0.950, 0.952, 0.970]
```

```
home = [0.9649, 0.9825, 0.9825, 0.9643, 0.9649,
        0.9825, 0.9821, 0.9821, 0.9643, 0.9649,
        0.9649, 0.9825, 0.9825, 0.9825, 0.9643,
        0.9821, 0.9825, 0.9821, 0.9821, 0.9649,
        0.9821, 0.9649, 0.9643, 0.9825, 0.9643,
        0.9649, 0.9825, 0.9298, 0.9821, 0.9298]

swipe = [0.9245, 0.9057, 0.9245, 0.9038, 0.9057,
         0.9434, 0.9245, 0.9231, 0.9245, 0.9231,
         0.8868, 0.9057, 0.9231, 0.9623, 0.9245,
         0.9434, 0.9245, 0.8679, 0.9434, 0.9423,
         0.9245, 0.9434, 0.9057, 0.9245, 0.9245,
         0.9623, 0.9434, 0.9615, 0.9245, 0.9231]

other = [0.9797, 0.9822, 0.9772, 0.9773, 0.9822,
         0.9518, 0.9772, 0.9596, 0.9696, 0.9722,
         0.9772, 0.9594, 0.9620, 0.9670, 0.9722,
         0.9620, 0.9492, 0.9823, 0.9367, 0.9722,
         0.9671, 0.9569, 0.9671, 0.9822, 0.9873,
         0.9213, 0.9645, 0.9519, 0.9519, 0.9823]

print("Main Accuracy")
conf_interval(main_accuracy)
print("Home")
conf_interval(home)
print("Swipe")
conf_interval(swipe)
print("Other")
conf_interval(other)
```

A.2 Реалізація мережі MobileNetV2

```
import numpy as np
import tensorflow as tf
from tensorflow.contrib import layers

class MobileNetV2:
    def __init__(self, width_mult=1.0, input_size=224,
                 num_joints=21, num_channels=4,
                 is_training_supported=True):
        self.is_training_supported = is_training_supported
        self.input_size = input_size
        self.normalizer = tf.layers.batch_normalization
        self._init_variables(num_joints)
        self.width_mult = width_mult

        with tf.variable_scope('MobileNetV2'):
            self._create_placeholders(num_channels)
            self._build_model()
            self._create_head()
            self._create_loss()

        self.skeleton_session, self.skeleton_graph, \
        self.interpreter = None, None, None

        self.output_nodes = "MobileNetV2/output_node"

    def _init_variables(self, num_joints):
        self.joints = num_joints
        self.joint_dim = 3
```

```
        self.number_outputs = self.joints * self.joint_dim

    def _create_head(self):
        filters = self.net.shape[-1] * 4
        self.net = layers.conv2d(
            self.net, filters, 1,
            normalizer_fn=self.normalizer,
            normalizer_params=self.bn_params)

        last_layer_side = int(self.net.shape[1])
        self.net = layers.avg_pool2d(self.net,
                                     last_layer_side)
        self.net = tf.squeeze(self.net, axis=(1, 2))
        self.net = tf.concat(
            [self.net, self.placeholder_hand_side], axis=1)
        self.net = layers.fully_connected(
            self.net, self.number_outputs,
            activation_fn=None)

        self.net = tf.reshape(self.net, [-1, self.joints,
                                         self.joint_dim])
        self.net = tf.nn.sigmoid(self.net,
                                 name="output_node")

    def _create_placeholders(self, num_channels):
        self.placeholder_input = tf.placeholder(
            dtype=tf.float32, name="input_node",
            shape=[None, self.input_size, self.input_size,
                   num_channels])
        self.placeholder_hand_side = tf.placeholder(
            dtype=tf.float32, shape=[None, 1],
```

```
            name="hand_side")
        self.placeholder_output = tf.placeholder(
            dtype=tf.float32,
            shape=[None, self.joints, self.joint_dim])
        if self.is_training_supported:
            self.bn_params_placeholder = tf.placeholder(
                tf.bool, name='is_training')
            self.bn_params = {
                'training': self.bn_params_placeholder}
        else:
            self.bn_params = {'training': False}

    def _create_loss(self):
        self.loss = tf.losses.mean_squared_error(
            self.placeholder_output, self.net)

    def set_pb_predictor(self, skeleton_path):
        self.skeleton_session, self.skeleton_graph = \
            load_pb_model(
                skeleton_path)

    def predict(self, tensor, sess=None, target=None):
        input_tensor, hand_side = tensor

        input_dict = {
            'MobileNetV2/input_node': input_tensor,
            'MobileNetV2/hand_side': hand_side}
        output_nodes = ['MobileNetV2/output_node']

        if self.interpreter:
            prediction = run_tflite_model(self.interpreter,
```

```
                                        input_dict,
                                        output_nodes)
            return prediction
        elif self.skeleton_session:
            if self.is_training_supported:
                input_dict.update(
                    {'MobileNetV2/is_training': False})
            pb_model = self.skeleton_session, \
                       self.skeleton_graph
            prediction = run_pb_model(pb_model, input_dict,
                                      output_nodes)
            return prediction
        else:
            if target is None: raise Exception(
                "Target must be not None.")
            if sess is None: raise Exception(
                "Session must be not None.")
            if self.is_training_supported:
                prediction, batch_loss = sess.run(
                    [self.net, self.loss],
                    feed_dict=self.get_feed_dict(
                        tensor, target, is_training=False))
            else:
                prediction, batch_loss = sess.run(
                    [self.net, self.loss],
                    feed_dict=self.get_feed_dict(tensor,
                                                 target))
            return prediction, batch_loss

    def _build_model(self):
        self.i = 0
```

```
with tf.variable_scope('init_conv'):
    output = layers.conv2d(
        self.placeholder_input,
        32, 3, 2,
        normalizer_fn=self.normalizer,
        normalizer_params=self.bn_params)

self.net = self._inverted_bottleneck(output, 1, 16,
                                      0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      24, 1)
self.net = self._inverted_bottleneck(self.net, 6,
                                      24, 0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      32, 1)
self.net = self._inverted_bottleneck(self.net, 6,
                                      32, 0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      32, 0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      64, 1)
self.net = self._inverted_bottleneck(self.net, 6,
                                      64, 0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      64, 0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      64, 0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      96, 0)
self.net = self._inverted_bottleneck(self.net, 6,
                                      96, 0)
```

```
        self.net = self._inverted_bottleneck(self.net, 6,
                                             96, 0)
        self.net = self._inverted_bottleneck(self.net, 6,
                                             160, 1)
        self.net = self._inverted_bottleneck(self.net, 6,
                                             160, 0)
        self.net = self._inverted_bottleneck(self.net, 6,
                                             160, 0)
        self.net = self._inverted_bottleneck(self.net, 6,
                                             320, 0)

    def _inverted_bottleneck(self, input, up_sample_rate,
                             channels, subsample):
        channels = int(channels * self.width_mult)
        with tf.variable_scope(
                'inverted_bottleneck{}_{}_{}'.format(
                    self.i, up_sample_rate, subsample)):
            self.i += 1
            stride = 2 if subsample else 1
            output = layers.conv2d(
                input, up_sample_rate *
                       input.get_shape().as_list()[
                           -1], 1,
                activation_fn=tf.nn.relu6,
                normalizer_fn=self.normalizer,
                normalizer_params=self.bn_params)
            output = layers.separable_conv2d(
                output, None,
                3, 1,
                stride=stride,
                activation_fn=tf.nn.relu6,
```

```
            normalizer_fn=self.normalizer,
            normalizer_params=self.bn_params)
        output = layers.conv2d(
            output, channels, 1,
            activation_fn=None,
            normalizer_fn=self.normalizer,
            normalizer_params=self.bn_params)
        if input.get_shape().as_list()[-1] == channels:
            output = tf.add(input, output)
        return output
```

A.3 Реалізація мережі детектора руки

```
import tensorflow as tf
from tensorflow.contrib import layers
from tensorflow.contrib.layers.python.layers import nn
from tensorflow.python.ops import variable_scope


class HandsDetector():
    def __init__(self, input_size=224, num_channels=4,
                 is_training_supported=True):
        self.is_training_supported = is_training_supported
        self.weight_confidence = 0.25
        self.input_size = input_size
        self.bb_outputs = 3
        self.inputs = tf.placeholder(tf.float32, shape=(
            None, input_size, input_size, num_channels))
        self.normalizer = tf.layers.batch_normalization
        if self.is_training_supported:
            self.bn_params_placeholder = tf.placeholder(
```

```
            tf.bool, name='is_training')
        self.bn_params = {
            'training': self.bn_params_placeholder}
    else:
        self.bn_params = {'training': False}

    with variable_scope.variable_scope('hands_detector',
                                       values=[
                                           self.inputs]):
        self.net = self._build_net()
        bbs, confidences = self.net
        bb_left, bb_right = bbs
        confidence_left, confidence_right = confidences

        self.concatenated = tf.concat(
            [bb_left, confidence_left, bb_right,
             confidence_right], axis=1,
            name="concated_output")

        self.output_bb = tf.stack([bb_left, bb_right],
                                  1, name="bbs")
        self.output_confidences = tf.stack(
            [confidence_left, confidence_right], 1,
            name="confidences")

        self.loss = self._loss(bbs, confidences)

    self.output_nodes_list = (
        'hands_detector/bbs',
        'hands_detector/confidences',
        'hands_detector/concated_output')
```

```
        self.output_nodes = ",".join(self.output_nodes_list)
        self.interpreter = None
        self.detector_graph = None

    def _build_net(self):
        with tf.variable_scope('conv_1'):
            net = layers.conv2d(
                self.inputs, 16, [3, 3], 2,
                padding='SAME',
                biases_initializer=None,
                normalizer_fn=self.normalizer,
                normalizer_params=self.bn_params,
                weights_initializer= \
                    tf.initializers.variance_scaling,
                activation_fn=nn.relu6)

        with tf.variable_scope('conv_2'):
            net = layers.conv2d(
                net, 32, [3, 3], 2,
                padding='SAME',
                biases_initializer=None,
                normalizer_fn=self.normalizer,
                normalizer_params=self.bn_params,
                weights_initializer= \
                    tf.initializers.variance_scaling,
                activation_fn=nn.relu6)

        with tf.variable_scope('conv_3'):
            net = layers.conv2d(
                net, 32, [3, 3], 2,
                padding='SAME',
```

```
            biases_initializer=None,
            normalizer_fn=self.normalizer,
            normalizer_params=self.bn_params,
            weights_initializer=tf.initializers.variance_s
            activation_fn=nn.relu6)

    with tf.variable_scope('conv_4'):
        net = layers.conv2d(
            net, 32, [3, 3], 2,
            padding='SAME',
            biases_initializer=None,
            normalizer_fn=self.normalizer,
            normalizer_params=self.bn_params,
            weights_initializer= \
                tf.initializers.variance_scaling,
            activation_fn=nn.relu6)

    with tf.variable_scope('conv_5'):
        net = layers.conv2d(
            net, 32, [3, 3], 1,
            padding='VALID',
            biases_initializer=None,
            normalizer_fn=self.normalizer,
            normalizer_params=self.bn_params,
            weights_initializer= \
                tf.initializers.variance_scaling,
            activation_fn=nn.relu6)

    net = tf.reshape(net, [-1, 5 * 5 * 32])

    with tf.variable_scope('confidence_left'):
```

```
        confidence_left = layers.fully_connected(
            net, 1,
            activation_fn=tf.nn.sigmoid,
            weights_initializer= \
                tf.initializers.variance_scaling)

    with tf.variable_scope('confidence_right'):
        confidence_right = layers.fully_connected(
            net,
            1,
            activation_fn=tf.nn.sigmoid,
            weights_initializer= \
                tf.initializers.variance_scaling)

    with tf.variable_scope('bb_left'):
        bb_left = layers.fully_connected(
            net,
            self.bb_outputs,
            activation_fn=tf.nn.sigmoid,
            weights_initializer= \
                tf.initializers.variance_scaling)

    with tf.variable_scope('bb_right'):
        bb_right = layers.fully_connected(
            net,
            self.bb_outputs,
            activation_fn=tf.nn.sigmoid,
            weights_initializer= \
                tf.initializers.variance_scaling)

    return (bb_left, bb_right), (
```

```
            confidence_left, confidence_right)
```

A.4 Реалізація класифікатора жестів

```
import tensorflow as tf
from tensorflow.contrib import layers

class GesturesNet:
    def __init__(self, sequence_size=30, n_classes=4):
        self.kernel_size = (1, 5)
        self.inputs = tf.placeholder(
            tf.float32,
            shape=(None, 1, sequence_size, 3 * 21))
        self.n_classes = n_classes
        self.target = tf.placeholder(tf.uint8, [None, 1])
        self.weights = tf.placeholder(
            tf.float32, [n_classes])
        self.net = self.conv_net()
        self.loss = self.calculate_loss()
        self.interpreter, self.detector_session, \
        self.detector_graph = None, None, None
        self.output_nodes = "ConvNet/Softmax"

    def conv_net(self):
        with tf.variable_scope('ConvNet'):
            conv1 = tf.layers.conv2d(
                self.inputs, filters=64,
                kernel_size=self.kernel_size,
                padding="VALID",
                activation=tf.nn.relu6)
            conv2 = tf.layers.conv2d(
```

```
            conv1, 32, self.kernel_size,
            padding="VALID", activation=tf.nn.relu6)
        conv3 = tf.layers.conv2d(
            conv2, 16, self.kernel_size,
            padding="VALID")

        net = tf.layers.flatten(conv3)
        self.fc1 = tf.layers.dense(net, self.n_classes)
        out = tf.nn.softmax(self.fc1)

    return out
```