КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю122 Комп'ютерні науки

на тему:

ВИЗНАЧЕННЯ НАПРЯМКУ ОБЕРТУ ГОЛОВИ ЛЮДИНИ ЗА ЗОБРАЖЕННЯМ У 3D

Виконав студент 4 курсу
Сайтарли Олексій Юрійович

(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Федорус Олексій Мстиславович

(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики

«____» _____

2022 р., протокол № ___

Завідувач кафедри

Терещенко Василь Миколайович _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 41 сторінок, 14 ілюстрацій, 0 таблиці, 17 джерел посилань.

ВИЯВЛЕННЯ ОБЛИЧЧЯ, ДИСТИЛЯЦІЯ ЗНАНЬ, ЗНАХОДДЕННЯ ТРЬОХВИМІРНОЇ ПОЗИ, КОМП'ЮТЕРНИЙ ЗІР, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ.

Об'єктом роботи є спосіб знаходження обличчя та його параметрів по зображенню за допомогою алгоритмів машинного навчання. Предметом роботи є розробка програми, яка за допомогою нейронних мереж знаходить обличчя та їх трьох вимірний напрямок за зображенням.

Метою роботи є створення програмного засобу для визначення напрямку в якому обернене обличчя людини по зображенню.

Методи розроблення: штучні нейронні мережі, багатозадачні каскадні згорткові нейронні мережі, дистиляція знань. Мова програмування python, фреймворк PyTorch, середа розробки Google Colab.

Результати роботи: виконано загальний огляд способів знаходження обличь на зображеннях і визначення напрямку обличчя по фотографії. Запропонований спосіб комбінації окремих рішень в одне за допомогою методу дистиляції знань. Натренована нейронна мережа FPNet для визначення напрямку обличчя. Створений додаток, який знаходить обличчя та їх напрямки по зображенню.

Задачі по виявленню обличь разом з їх направленням по зображенню виникають в таких сферах як доповнена реальність, взаємодія користувача з розумними пристроями, забезпечення безпеки водіїв під час довгих поїздок тощо.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

МН – Машинне навчання

НМ – Нейрона мережа.

MTCNN – Multi-task Cascaded Neural Network

BB – Bounding box

NMS – Non Maximum Suppression

CNN – Convolutional neural network

FPNet – Face Pose Net

LR – Learning Rate

GPU – Graphics processing unit

API – Application Programming Interface

ВСТУП

Оцінка сучасного стану об’єкта розробки. Збільшення кількості «розумних» пристроїв у побуті людей призводить до необхідності пошуку нових способів взаємодії людини з машиною засобами штучного інтелекту, які будуть швидшими та ефективнішими за існуючі.

Сьогодні для вирішення задач комп’ютерного зору майже завжди використовуються методи машинного навчання. З винаходу AlexNet у 2012 році штучні нейронні мережі стали домінуючим підходом до вирішення задач комп’ютерного зору [1]. Однак, часто для тренування НМ потрібна велика кількості даних, розмічених людиною. Іншим недоліком НМ є необхідність у великій кількості обчислювальних потужностей. Переважно у цих двох напрямках і відбувається сьогоденний прогрес.

Часто трапляється що не вдається знайти датасет, в якому були б усі необхідні дані. Для рішення цієї проблеми існує багато різних підходів. За можливості необхідний датасет можна синтезувати штучно, наприклад використовуючи технології для створення комп’ютерних ігор з реалістичною графікою [2]. Недоліком цього підходу є те, що це доволі нетривіальна задача, яка потребує специфічних навичок. Іншим способом є так звана дистиляція даних. Цей підхід дозволяє поєднувати декілька НМ, які навчалися на різних датасетах, в одну.

Одним з способів створення більш ефективних рішень є використання декількох відносно невеликих НМ, які запускаються послідовно і вирішують кожна свою задачу, такий підхід називається каскадним.

Актуальність роботи та підстави для її виконання. В сучасному світі все більше пристроїв стає «розумними» в тому сенсі що імітують поведінку людини. Визначальну роль в цьому процесі відіграє ідентифікація людини. Таким чином задача по визначенню направлення обличчя трапляється при створенні інтерфейсів людина-комп’ютер.

Наприклад подібні системи використовуються для допомоги водіям не заснути за кермом під час довгого переїзду. Або нагадують стежити за дорогою під час використання автомобільного автопілоту.

Окремо треба зазначити використання подібних систем у рішеннях з доповненою реальністю, які зараз активно розробляються найбільшими технологічними компаніями світу. Знаючи в яку сторону дивляться люди навколо система доповненої реальності може реагувати та змінювати світ навколо відповідним чином. За допомогою цього можна створювати найрізноманітніші застосунки: від ігор до систем, які асистують лікарям під час операцій.

Тобто задача по знаходженню напрямку обличь є однією з фундаментальних у взаємодії людини з комп'ютером, тому є актуальною.

Мета й завдання роботи. Метою роботи є створення програмного застосунку для знаходження обличь та їх напрямків на зображенні. Для досягнення цієї мети поставлено такі завдання.

- Дослідити існуючі способи знаходження обличь на зображенні
- Дослідити існуючі способи знаходження напрямку обличь за зображенням
- На основі досліджених матеріалів запропонувати спосіб вирішення поставленої задачі.
- Підготувати необхідний датасет для тренування НМ.
- Підібрати оптимальні гіперпараметри та натренувати НМ.
- Створити програмний застосунок, що використовує натреновану модель.

Об'єкт, методи й засоби розроблення. Об'єктом цієї роботи є створення оптимальних рішень для знаходження обличь та їх параметрів по зображенню за допомогою нейронних мереж.

Для розробки цього рішення був використаний метод дистиляції знань для об'єднання можливостей двох різних підходів, а саме: багатозадачні

каскадні згорткові нейромережі для знаходження обличь на зображенні та 6DRepNet для визначення напрямку обличчя [3, 4]. Розробка відбувалась у хмарному сервісі Google Colab, який надає доступ до графічного процесора. Для розробки використовувалась мова програмування Python та фреймворк для глибокого навчання PyTorch [5, 6]. Для спостерігання за процесом тренування моделі використовувався інструмент Tensorboard [7]. Для тренування моделі використовувався датасет CelebA [8].

Можливі сфери застосування. Запропонована система знаходження обличь та їх напрямків може використовуватись у інтерфейсах людина-комп’ютер. Наприклад для попередження засинання водіїв під час довгих поїздок, для того щоб екран смартфону не вимикався поки на нього хтось дивиться тощо.

Взаємозв’язок з іншими роботами. У цій роботі запропоновано спосіб комбінування двох робіт, а саме: багатозадачні каскадні згорткові нейромережі та 6DRepNet за допомогою методу дистиляції знань [3, 4, 10].

РОЗДІЛ 1.
ВИРІШЕННЯ ЗАДАЧ КОМП'ЮТЕРНОГО ЗОРУ МЕТОДАМИ МАШИННОГО НАВЧАННЯ

1.1 Машинне навчання

Машинне навчання це вид штучного інтелекту, який дозволяє комп’ютерним програмам робити точні передбачення на основі вхідних даних. При цьому створення такої програми не потребує від програміста чіткого розуміння зв’язку між вхідними та вихідними даними. Отримавши велику кількість даних алгоритми МН самі знаходять зв’язки в них.

В сучасному світі алгоритми машинного навчання використовуються у багатьох звичних програмних рішеннях таких як пошукові сервіси, спам фільтри електронної пошти тощо. Найбільші технічні компанії світу такі як Facebook, Google, Uber, Qualcomm тощо активно розвивають цей напрямок і впроваджують його у свої сервіси.

Зазвичай алгоритми машинного навчання класифікують за способом, яким алгоритм навчається робити точніші передбачення. Можна виділити чотири основних типи алгоритмів машинного навчання:

1. Навчання з вчителем (Supervised learning)
2. Навчання без вчителя (Unsupervised learning)
3. Напівавтоматичне навчання (Semi-supervised learning)
4. Навчання з підкріпленням (Reinforcement learning)

Той чи інший тип МН може бути обраний в залежності від потреб та доступних даних. У цій роботі переважно використовується навчання з вчителем.

Навчання з вчителем. Алгоритми МН цього типу потребують розмічених наборів даних (датасетів), тобто таких де вхідні данні співставленні в вихідними людиною, або якимось іншим чином.

1.2 Комп’ютерний зір

Комп’ютерний зір це область комп’ютерних наук яка займається розробкою цифрових систем, які здатні аналізувати візуальні данні та генерувати якісь висновки на їх основі. Тобто, алгоритми комп’ютерного зору приймають на вхід ображення у вигляді матриці пікселів і генерують якесь судження на основі цього, наприклад «за зображенні яблуко».

Основні види задач, які виникають наступні:

1. **Знаходження об’єктів.** Система аналізує зображення та знаходить на ньому об’єкти наперед визначеного класу. Наприклад може знайти окремо усі столи та стільці на зображенні.
2. **Ідентифікація об’єкту.** Система аналізує зображення і знаходить конкретний об’єкт на зображенні. Наприклад може знайти чиєсь конкретне лице на спільній фотографії.
3. **Спостерігання за об’єктом.** Система знаходить на відео якийсь об’єкт та стежить за його переміщенням.

Також трапляються задачі, коли необхідно не тільки знайти об’єкт, а й якусь його характеристику. Наприклад, в цій роботі запропонований метод знаходження всіх обличь (це задача **Знаходження об’єктів)** та їх напрямку.

Створюючи алгоритми комп’ютерного зору науковці часто намагаються зрозуміти і відтворити процеси, які відбуваються у мозку живих істот. Але створення системи, аналогічної живому мозку це дуже складна задача, до того ж далеко не факт, що живий мозок працює оптимальним чином. Тому часто користуються спрощеною моделлю, яка припускає, що спочатку мозок живої істоти знаходить якісь патерни у зображенні, наприклад прямі лінії, кружечки тощо. Потім на основі цих патернів збираються патерни вищого порядку, наприклад дві лінії, що перетинаються. Цей процес поступового ускладнення відбувається доти доки нарешті не стане зрозумілим, що перед нами якийсь конкретний об’єкт.

1.3 Згорткові нейронні мережі

На практиці виявилось, що задачі комп'ютерного зору можна дуже ефективно вирішувати методами машинного навчання. Початком революції в цій сфері вважається поява нейронної мережі AlexNet у 2012 [1]. Це було вперше коли штучна НМ показала більш точні результати у ImageNet Challenge ніж популярні тоді «класичні» методи комп'ютерного зору. З того часу з'явилось багато нових рішень, які мали в основі той самий механізм що й AlexNet, а саме – згортки.

Згорткова нейронна мережа або CNN(Convolutional Neural Network) це алгоритм машинного навчання, а точніше його розділу «глибоке навчання». CNN складаються з декількох шарів. Перший шар приймає на вхід зображення, і знаходить у ньому різні патерни. Для кожного патерну вираховується значущість (вага). Сукупність значень, які повертає шар називається карта особливостей (feature map). Feature map, порахована першим шаром передається на наступний шар, якій робить все те саме. Останній шар обраховує фінальну карту особливостей, яка фактично являє тобою представлення картинки, яке комп'ютер здатний інтерпретувати.

Успіхи CNN часто пов'язують з тим, що принцип їх роботи дещо схожий на те що відбувається у зоровій корі мозку. Однак більш реалістична причина полягає у тому, що CNN добре вміють знаходити закономірності між близькими у просторі елементами зображення. Тобто аналізуючи дрібні частинки зображення лівої сторони зображення, права сторона не береться до уваги. І вже тільки на останніх шарах, мережа комбінує інформацію, зібрану з усього зображення, і синтезує результат. Це є інтуїтивно правильний підхід по обробки зображень, тому CNN добре справляються з задачами саме комп'ютерного зору.

Для того щоб мережа працювала, необхідно щоб вона знаходила «правильні» патерни і присвоювала їм правильні ваги. Для цього її «тренують» за допомогою алгоритмів машинного навчання.

1.4 Функції втрат

Основними методами машинного навчання, які знаходять параметри нейронної мережі, є методи засновані на градієнтному спуску.

Ці методи називаються алгоритми оптимізації. Вони змінюють параметри НМ таким чином, щоб зменшувалось значення так званої функції втрат.

Для задач тренування з вчителем функція втрат приймає два аргументи: значення яке повернула мережа та значення які вона мала повернути. На основі аргументів визначається те наскільки мережа помилилась. Чим більше мережа помилилась, тим більше значення помилки.

У цій роботі використано два види функції втрат, а саме:

1. Евклідова норма
2. Кросентропійна функція втрат

Евклідова норма часто використовується як функція втрат оскільки це природній спосіб визначення того, наскільки два вектори далеко один від іншого.

Кросентропійна функція втрат застосовується в задачах класифікації. Вона визначає наскільки відрізняються розподіли двох випадкових змінних.

1.5 Алгоритми оптимізації

Алгоритми оптимізації змінюють ваги таким чином, щоб значення функції втрат зменшувалось. Майже всі вони використовують метод градієнтного спуску, який в свою чергу потребує можливість обраховувати градієнт функції втрат, тобто значення часткових похідних по усім параметрам мережі. Градієнт рахується за допомогою методу зворотного поширення помилки.

Різні алгоритми оптимізації по суті є різними модифікаціями методу градієнтного спуску. У цій роботі використано метод оптимізації, який наздається Adam.

1.6 Методи регуляризації

Методи регуляризації дозволяють зробити помилку, пов’язану з перетренованістю (overfitting), меншою. Overfitting проявляється тим, що мережа робить дуже точні передбачення на тренувальній вибірці, але демонструє значно гірші результати на варіаційній вибірці.

Основні методи регуляризації такі:

Штрафування параметрів за нормою. При цьому підході до фінальної функції втрат додається якась норма Ω для параметрів мережі. Тобто при тренуванні параметри мережі не будуть сильно зростати у значенні. Емпірично доведено, що це сприяє зменшенню overfitting.

Згладжування міток. Цей метод можна використати для покращення результатів при вирішенні задач класифікації. Його суть у тому, що результат роботи функції softmax порівнюється не з точними мітками, значення яких або 0 або 1, а зі згладженими, значення яких α та 1 – α відповідно.

Виключення (Dropout). Цей метод полягає у випадковій заміні на нуль частини значень деякого проміжного тензора. Це робить мережу більш стійкою до overfitting.

У цій роботі використовувався метод виключення.

1.7 Аугментація даних

Аугментація даних це методика штучного збільшення кількості даних. Вона полягає у тому, що перед тим як навчати мережу на елементі з датасету він трохи змінюється.

Це створює ефект, схожий на регуляризацію, адже на кожній новій епосі мережа бачить «нові» дані тому не може їх запам’ятати.

Для картинок можна застосувати такі перетворення як:

1. Різноманітні оберти та відображення
2. Зміна контрастності
3. Зміна чіткості зображення

1.8 Огляд архітектури VGG

У цьому підрозділі розглянуто архітектуру мережі, яка називається VGG. Це CNN з простою архітектурою, яка тим не менш дає гарні результати.

На рисунку 1.1 наведено забраження з роботи авторів VGG. На ньому у колонках зазначені різні версії мережі. Зазвичай використовують варіанти D та E, які називають VGG-16 та VGG-19 відповідно.

Запис convK-N означає операцію згортки з ядром розміром KxK, та вихідною розмірністю N. Також в мережі наявна операція maxpool яка зміншує просторову розмірність вхідного тензору. Останні декілька шарів є повністю зв'язаними.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Рисунок 1.1 – Опис архітектури сімейства VGG [10]

1.9 Способи представлення даних у нейронних мережах

Оскільки комп’ютер здатен працювати тільки з числовими значеннями, перед тим як подавати картинки на вхід нейронної мережі їх необхідно певним чином підготувати.

У більшості фреймворках для глибинного навчання дані зберігаються у тензорах. Тензор це багато розмірний масив, більш наглядно це можна побачити рис. 1.2.

Тензори можуть мати різні типи даних, для тренування НМ зазвичай використовують 4-байтні числа з плаваючою комою.

Найпростіший спосіб перетворити картинку на числовий тензор – це просто записати в нього значення трьох канального коліру. Але тоді на вхід до мережі будуть поступати дані в діапазоні від 0 до 255. Взагалі, нейронні мережі здатні пристосуватися до будь-якого формату вхідних даних, але є серйозні причини так не робити.

По-перше, необхідність працювати з великими значеннями може призвести до зростань значень параметрів мережі, а це в свою чергу може призвести до проблем з overfitting.

По-друге, чим ближчі числа з плаваючою комою до нуля, тим з більшою точністю відбуваються операції над ними. Так відбувається через формат в якому вони зберігаються згідно формати IEEE [11]. На практиці проблеми з точністю виникають не часто, але в дуже глибоких мережах вони можуть накопичуватись і призводити до неочікуваних результатів.

По-третє, може так статися, що в тренувальній вибірці розподіл кольорів відрізняється від тих даних, на яких планується робота нейронної мережі. Наприклад, там може бути більше зелених картинок, а працювати НМ буде переважно на червоних картинках. У такому випадка існує дуже велика ймовірність того, що точність передбачень знизиться.

Через наведені причини перед тим як подати картинка на вхід НМ їх зазвичай нормалізують, тобто приводять до нормального розподілу з

математичним сподіванням, яке дорівнює нулю і одиничним стандартним відхиленням це відбувається за формулою 1.1.

$$I = \frac{I_0 - \mu(I_0)}{\sigma} \tag{1.1}$$

Де I_0 – це початкова картинка, σ – це стандартне відхилення кольорів, $\mu(I_0)$ – це математичне сподівання кольору картинки.

Значення дисперсії і математичного сподівання, використані для нормалізації автори досліджень зазвичай пишуть у своїх роботах.

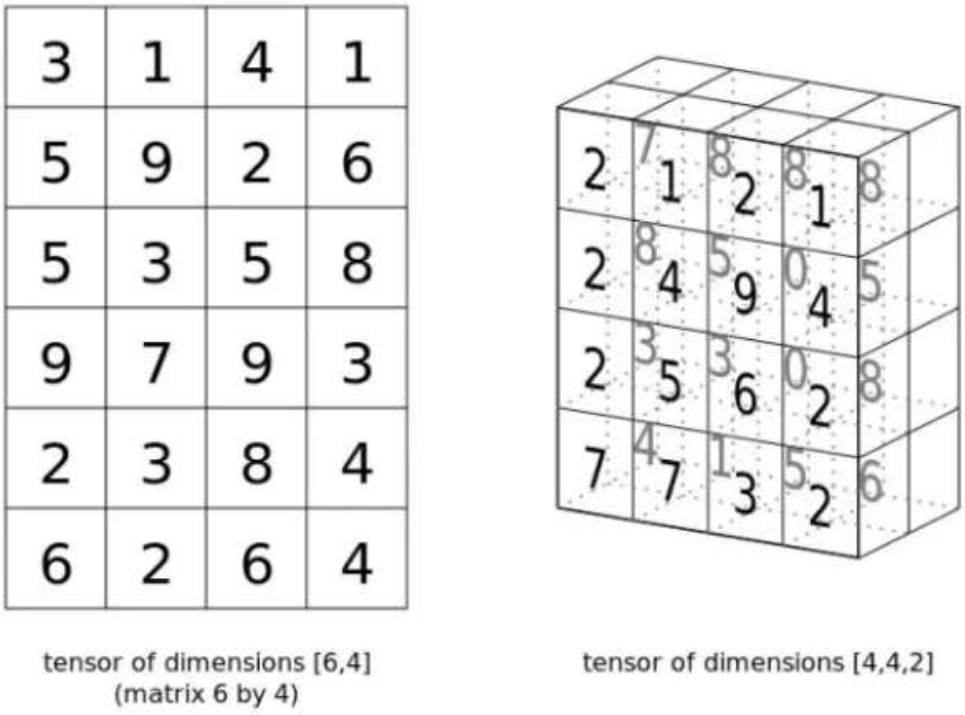

Рисунок 1.2 – Приклади двох та трьох розрядних тензорів

РОЗДІЛ 2.
ТЕХНОЛОГІЇ ТА ЗАСОБИ ВИРІШЕННЯ ЗАДАЧІ

2.1 Використання Python для машинного навчання

Python – це популярна мова програмування для досліджень в області алгоритмів штучного інтелекту [5]. Ця мова надає зручні математичні інструменти, що дозволяють легко і швидко програмувати складні алгоритми. Також для Python написано чимало бібліотек, які дозволяють вирішувати задачі з математичної статистики, аналізу даних тощо.

Ще одна перевага Python це його простий синтаксис, який дозволяє спеціалістам з області математики та іншим науковцям, які безпосередньо не пов’язані з програмуванням, швидко навчитися користуватися цією мовою.

2.2 Використання PyTorch для глибинного навчання

PyTorch це фреймворк для глибинного навчання, який розробляє Meta.

PyTorch дозволяє робити математичні перетворення і диференціювання цих перетворень [6]. Це досягається за допомогою обчислювального графу, який створюється непомітно для користувача.

До того ж ця бібліотека дозволяє виконувати операції на графічному процесорі за допомогою простого та інтуїтивного API. Що дозволяє пришвидшити такі операції як згортки та множення матриць в десятки а іноді і в сотні раз.

Найважливішою перевагою PyTorch над іншими фреймворками для глибокого навчання, такими як Tensorflow, є його зручність. Він зроблений таким чином щоб не відволікати увагу розробника на дрібні деталі, пов’язані зі специфікою виконання програми да GPU. Звісно це призводить до дещо повільнішої роботи (але далеко не у всіх випадках), але дозволяє робити експерименти швидше. А значить за один і той самий час можна спробувати більше підходів до задачі і отримати кращий результат.

2.3 Використання середовища розробки Google Colab

Google Colab — це хмарний сервіс, який постачається з попередньо встановленими фреймворками машинного навчання, такими як Tensorflow та PyTorch [7, 8]. Є можливість користуватися цим сервісом безкоштовно, але за додаткову плату можна отримати додаткові можливості, що значно покращать ефективність розробки.

Google Colab використовує Jupyter Notebook як середовище розробки, тому відразу можна побачити результат виконання коду та його окремих фрагментів.

За допомогою Colab зручно будувати нейронні мережі, адже він містить всі необхідні бібліотеки. А також безліч інформації про різні типи нейронних мереж та попередньо навчені моделі, що дає змогу добре розібратись в темі побудови нейронних мереж.

Colab надає потужні процесори для хмарних обчислень. Він має інтуїтивно зрозумілий інтерфейс, який дозволяє не перевантажувати комп'ютер розробника і робити всі обчислення швидко.

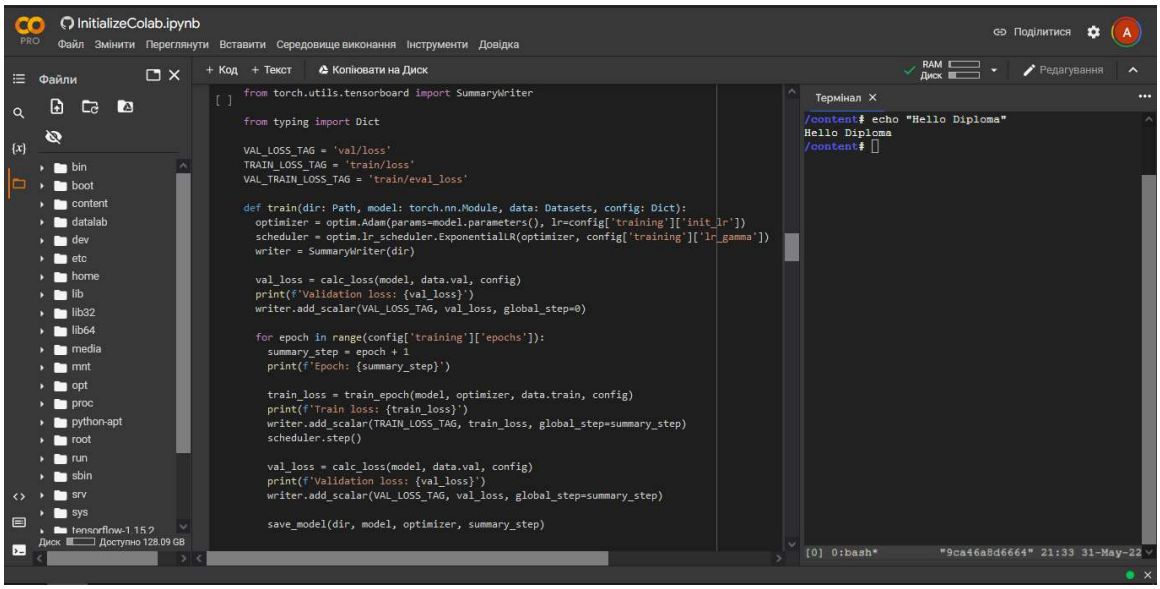

Рисунок 2.1 — Зовнішній вигляд Google Colab

2.4 Використання Tensorboard для моніторингу тренування

Тренування НМ це довгий ітеративний процес, тому важливо мати інструмент, який би дозволив стежити за ходом тренування та основних його характеристик.

Tensorboard це програмний застосунок з відкритим кодом, який дозволяє зручним чином відображати зміни з часом будь-яких значень, які потрібні розробнику. Зазвичай це використовують для побудови графіків зміни значення фцнкції втрат та метрик моделі.

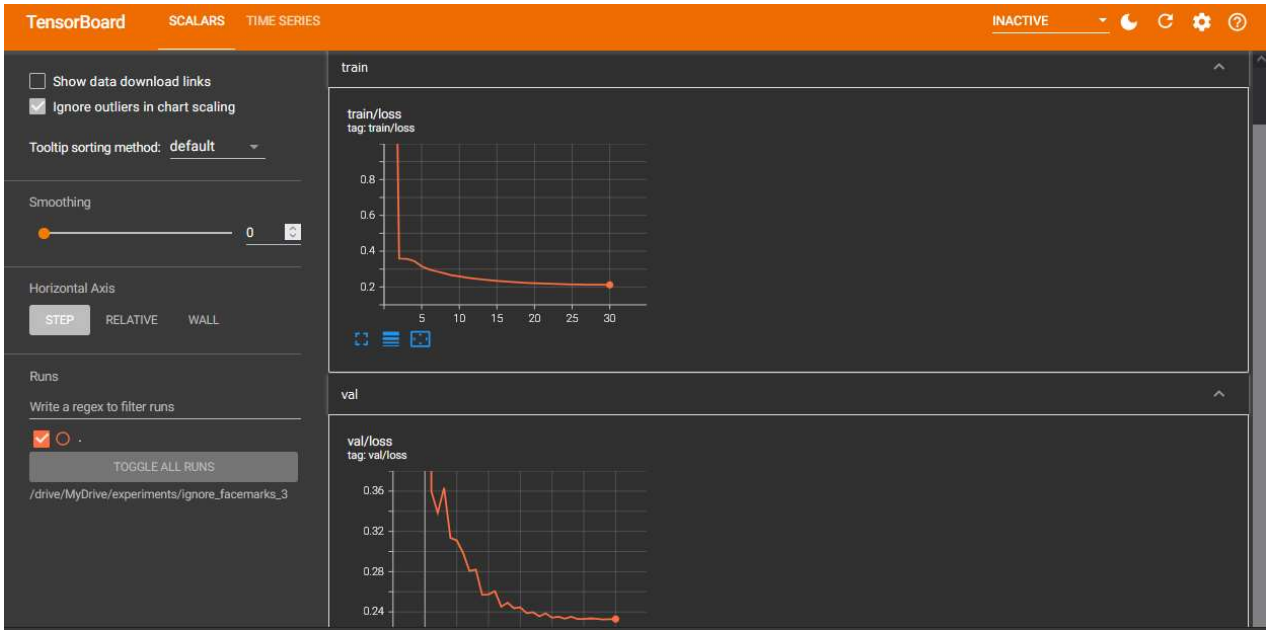

Рисунок 2.2 – Зовнішній вигляд Tensorboard

РОЗДІЛ 3
ОГЛЯД ВИКОРИСТАНИХ НАУКОВИХ РОБІТ

3.1 Багатозадачні каскадні згорткові нейронні мережі

Multi-task Cascaded Network (MTCNN) це фреймворк, який був запропонований для знаходження обличь та їх параметрів, зокрема лицевих якорів (face landmarks) у 2016 році [3]. Лицеві якорі це точки, розміщені на очах носі та губах лиця. Незважаючи на те, що в цьому методі застосовуються нейронні мережі, він швидко працює на центральному процесорі. Це можливо за рахунок того, що у цьому методі, на відміну від, наприклад, YOLO застосовується не одна велика нейронна мережа, а декілька невеликих – тобто каскад нейронних сіток, звідси і назва [12].

У цьому підрозділі детально розглянуто цей метод. Описані основні його складові, переваги та недоліки.

Метод, описаний авторами цієї статті вирішує задачу локалізації обличчя та якорів у три фази. Обличчя шукають у вигляді обмежувального прямокутника (Bounding Box). Спочатку знаходяться усі можливі кандидати BB, потім вони поступово відсіюються і уточняються. На останньому етапі окрім відкидання не обличь і уточнення обчислюються ще якорі.

Перша фаза. Для зображення рахується піраміда масштабів і кожен масштаб подається на згорткову нейронну мережу, яку автори назвали P-Net див. рис. 3.1 а. Ця мережа для кожного вікна, розміром у 12 x 12 пікселів повертає ймовірність того що у цьому вікні є обличчя та приблизне його положення в межах цього вікна, тобто грубий BB. Далі для всіх знайдених BB та відповідних їм імовірностям застосовується NMS.

Друга фаза. Частини зображень, які були знайдені а минулому етапі, вирізаються, змінюються у розмірі до 24 x 24 пікселів та подаються на вхід другої CNN, яку автори називають R-Net див. рис. 3.1 б, ця мережа повертає ймовірність того, що на зображенні є обличчя і повертає вектор здвигів, які

треба застосувати до BB, для того, щоб він краще відповідав реальному положенню обличчя. До отриманих BB знову застосовується NMS. Таким чином на другому етапі уточняються результати першого.

Третя фаза. Частини зображень, які були знайдені а минулому етапі, вирізаються, змінюються у розмірі до 48 x 48 пікселів та подаються на вхід третьої CNN, яку автори називають O-Net див. рис. 3.1 в, ця мережа повертає ймовірність того, що на зображенні обличчя і повертає вектор здвигів, які треба застосувати до BB, для того, щоб він краще відповідав реальному положенню обличчя та більш широку інформацію про обличчя, таку як якорі. До отриманих BB знову застосовується NMS.

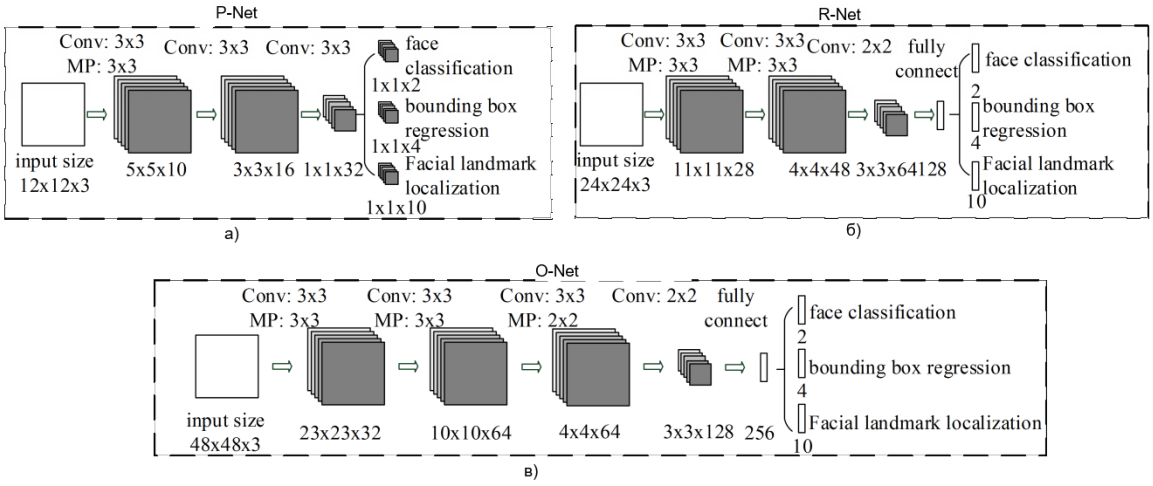

Рисунок 3.1 – Архітектура MTCNN [3]: а – архітектура P-Net; б – архітектура R-Net; в – архітектура O-Net

Оскільки мережі на кожному етапі повертають декілька значень, функція втрат для кожної з мереж формується як сума функцій втрат відповідної величини. А саме:

- Для класифікації обличчя (face classification) використовується кросентропійна функція втрат вигляду:

$$L_i^{det} = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i))) \quad (3.1)$$

Де p_i це ймовірність, яку повернула мережа, а y_i^{det} це індикатор того, що обличчя наявне в BB.

- Для визначення BB (bounding box regression) у якості функції втрат використовується евклідова відстань між передбаченням мережі та реальним значенням BB.

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2 \quad (3.2)$$

Де $\hat{y}_i^{box}$ значення, отримане від НМ, а y_i^{box} це дійсне значення BB.

- Для визначення якорів (Facial landmarks localization) у якості функції втрат використовується також евклідова відстань між передбаченням мережі та реальним значенням.

$$L_i^{landmarks} = \left\| \hat{y}_i^{landmarks} - y_i^{landmarks} \right\|_2^2 \quad (3.3)$$

Де $\hat{y}_i^{landmarks}$ значення, отримане від НМ, а $y_i^{landmarks}$ це дійсне положення якорів.

Загальна функція втрат, таким чином, виглядає так:

$$L_i = \quad L_i^{det} + \quad y_i^{det} * (L_i^{box} + L_i^{landmarks}) \quad (3.4)$$

зверніть увагу, що функції втрат L_i^{box} та $L_i^{landmarks}$ беруться до уваги тільки в тому випадку, якщо на вхід мережі було подане зображення саме обличчя. Це робиться для того, щоб у випадках, коли неможливо визначати параметри обличчя, мережа навчалась тільки задачі класифікації.

Також у своїй роботі автори порівнюють свій підхід з існуючими на той момент і доходять висновку, що їх рішення є найкращим тобто State of the art.

Таким чином MTCNN це швидкісний і точний метод виявлення обличь і їх властивостей по зображенню [3]. У своїй статті автори використовують цей метод для визначення якорів, але замість цього можна навчити мережу визначати будь-які інші параметри обличчя. Такі як, наприклад, напрямок обличчя. Основне обмеження для цього це наявність відповідних датасетів. Власне, це одна з задач моєї роботи.

3.2 Шестипросторове представлення пози голови

У цьому підрозділі ми роздивимось підхід до вирішення задачі визначення напрямку голови людини по зараженню, запропонований у статті 6DRepNet [4]. У цій статті автори пропонують метод визначення напрямку голови без попереднього визначення якорів. Тобто результат роботи НМ може бути безпосередньо перетворений в зручне представлення оберту, наприклад у кути Ейлера.

Архітектура запропонованої авторами нейронної мережі складається з двох елементів: основи у вигляді RepVGG та перетворенням, яке приймає на вхід два вектори і формує з них матрицю оберту, що відображає поворот обличчя [13]. Також автори запропонували функцію втрат для матриць оберту див. рис. 3.2.

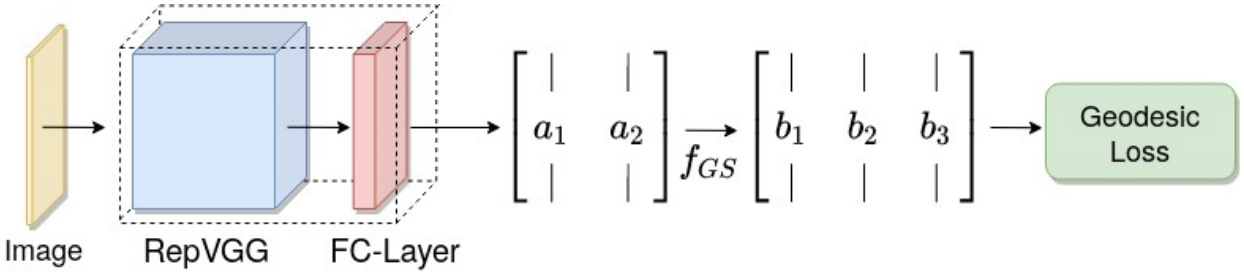

Рисунок 3.2 – Архітектура 6DRepNet [13]

Далі ми детально розберемося в кожному елементі цієї НМ а також визначмо її переваги та недоліки.

RepVGG. У статті «RepVGG: Making VGG-style ConvNets Great Again» автори пропонують нейронну мережу, яка під час тренування має архітектуру, схожу на ResNet див. рис. 3.3. а) та б). Але після тренування, за допомогою операцій, які не змінюють результат роботи мережі, архітектура НМ змінюється на подібну до VGG див. рис. 3.3 в).

Під час тренування додаткові зв’язки у архітектурі типу ResNet сприяють більш вільному поширенню градієнту вглиб НМ, сприяючі кращій точності. У той же час, архітектура типу VGG більш ефективна при обчисленні, бо не потребує зберігати тензори, що залишилися від минулих

операцій. Тобто RepVGG вбирає в себе кращі риси двох популярних архітектур та позбавлена деяких їх недоліків.

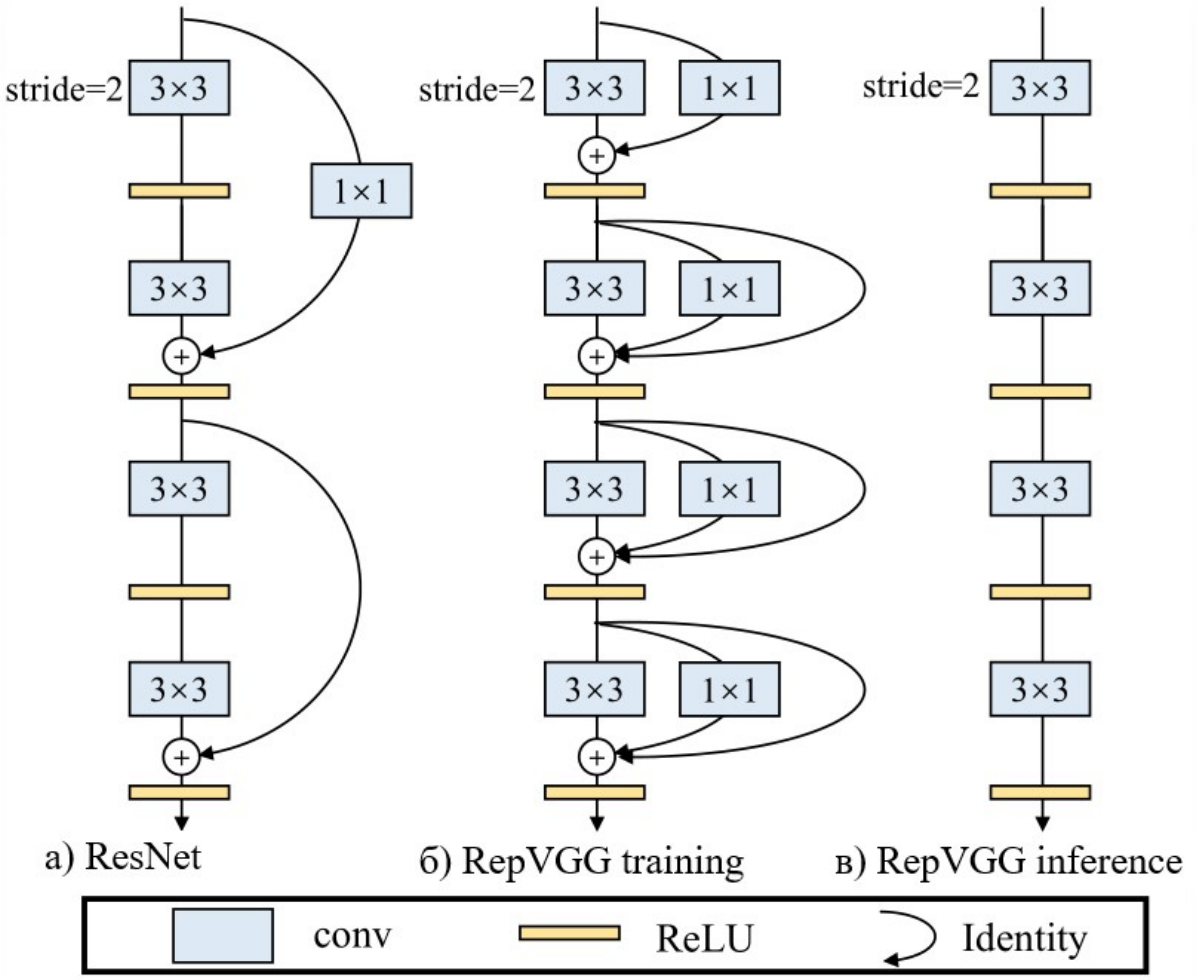

Рисунок 3.3 – Архітектура RepVGG [13]: а) – архітектура ResNet; б) – архітектура RepVGG під час тренування; в) – архітектура RepVGG під час виконання.

Визначення матриці повороту у 6DRepNet. Поворот можна представити декількома способами. Найбільш відомі це за допомогою кутів Ейлера та кватерніонів. Однак обидва ці способи страждають від проблем з неперервністю представлення оберту. Для їх уникнення автори пропонують наступний метод. НМ повертає два трьох вимірних вектори, по яким будується базис оберту. Записуючі отримані базисні вектори в стовпці матриці, отримуємо матрицю оберту, по якій потім обчислюється функція втрат.

Обчислення базису відбувається за наступними формулами:

$$\vec{b}_1 = \frac{\vec{a}_1}{|\vec{a}_1|},\ \vec{b}_3 = \frac{[\vec{b}_1,\vec{a}_2]}{|[\vec{b}_1,\vec{a}_2]|},\ \vec{b}_2 = [\vec{b}_3,\vec{b}_1] \tag{3.5}$$

Де $\vec{a}_1, \vec{a}_2$ це вектори, які є результатом роботи НМ, а $\vec{b}_1, \vec{b}_2, \vec{b}_3$ це шукані базисні вектори оберту.

Геодезична функція втрат. Для тренування мережі потрібна функція втрат, яка була б мінімальна при збігу двох обертів і монотонно зростала коли вони відрізняються. Для цього автори спочатку вираховують різницю двох обертів за допомогою множення однієї матриці на обернену іншу, а потів використовують відому тотожність, яка пов'язує слід матриці оберту з кутом цього оберту. В результаті отримуємо таку функцію втрат:

$$L_g = \cos^{-1}(\frac{tr(\widehat{R}\ R^T)-1}{2}) \tag{3.6}$$

Де $\widehat{R}$ це матриця оберту, отримана від НМ, а R це істинне значення.

У своїй роботі автори стверджують що 6DRepVGG досягає найкращих (State of the art) результатів у знаходженні направлення напрямку обличчя. Однак, ця мережа все ж таки є достатньо затратною в сенсі необхідних обчислювальних ресурсів. І це її основний мінус.

3.3 Дистиляція знань в нейронних мережах

За допомогою НМ вдається вирішувати найрізноманітніші задачі з доволі великою точністю. Проте часто отримані моделі виходять дуже громіздкими, через що для їх виконання потрібні значні обчислювальні потужності.

Дистиляція знань – це підхід, який дозволяє «стискати» НМ, тобто створювати моделі з меншою кількістю параметрів на основі натренованих великих моделей.

Суть підходу полягає у тому, що ми вчимо меншу модель (учня) в точності відтворювати поведінку великої моделі (вчителя). Тобто даними на

яких тренується модель учень у цьому випадку виступає результат роботи моделі вчителя.

Ефективність цього підходу була показана в статті від дослідників з Google [9]. У ній автори використали дистиляцію знань для тренування однієї моделі на основі великого ансамблю з сіток і отримали гарні результати.

Хоча на практиці показано, що цей метод часто працює і дає гарні результати, з теоретичної точки зору неможливо сказати в яких випадках він застосовний, а в яких ні. Також неможливо сказати наскільки можна зменшити кількість параметрів мережі учня по відношенню до вчителя.

Таким чином основним недоліком цього методу є те, що для того щоб він добре спрацював треба ретельно підбирати гіперпараметри моделі учня, що в свою чергу означає проведення великої кількості експериментів, що може дути доволі довгим процесом. Однак в результаті може вийти модель набагато швидша за модель вчителя, що дуже часто є пріоритетною задачею.

РОЗДІЛ 4.
ТРЕНУВАННЯ НЕЙРОННОЇ МЕРЕЖІ ТА СТВОРЕННЯ ЗАСТОСУНКУ

4.1 Загальний підхід

Дослідивши існуючі способи знаходження обличь і їх напрямків на зображенні була висунута гіпотеза, що поєднавши MTCNN для знаходження обличь з 6DRepNet для знаходження їх напрямку, можна отримати гарний результат [3, 4].

Найпростішим способом поєднати ці два підходом був би наступний:

1. На вхід подається картинка.
2. MTCNN знаходить усі обличчя.
3. 6DRepNet знаходить напрямки знайдених обличь.

Однак при такому підході довелось би запускати 6DRepNet для кожного знайденого обличчя, а як зазначалося в розділі 3.2 це доволі повільна мережа, для швидкої роботи якої необхідно мати значні обчислювальні потужності. Оскільки однією з вимог до шуканого рішення є швидка робота, цей підхід нам не підходить.

Оскільки основна проблема в «важкості» 6DRepNet, логічним було б замінити її на більш ефективну мережу таку мережу можна було б натренувати за допомогою методу дистиляції знань. Такий метод можливий, але було б ефективніше якби вдалося зробити так, щоб напрямок голови обчислювався безпосередньо під час виконання MTCNN.

На третьому етапі MTCNN вирішує декілька задач: уточнення BB та визначення положення якорів див. рис. 3.1 в). Оскільки визначення якорів не є задачею цієї роботи, можна замість них шукати направлення обличчя. Взагалі кажучи, можна було б шукати одночасно і якорі і направлення, проте під час експериментів виявилось, що такий підхід має складності з тренуванням, тому було прийняте рішення відмовитися від якорів на користь визначення направлення обличчя.

Далі у цьому розділі піде мова про те як відбувалося тренування НМ яка замінить O-Net на третьому етапі MTCNN, назвемо її FacePoseNet (FPNet).

4.2 Вибір датасету

Перед тим як тренувати НМ потрібно обрати датасет що підходить. Для цього треба зрозуміти вимоги, які на нього мають накладатися, провести огляд існуючих датасетів та обрати один з них, або якусь їх комбінацію.

Нам потрібен датасет з обличчями та ВВ для них. Це, в принципі, всі дані, які нам необхідні для тренування. Оскільки ми використовуємо підхід дистиляції знать, направлення обличь у датасеті буде визначене мережею вчителем.

Але майже будь-яка НМ дає неточні передбачення на даних, які не траплялися в розподілі тренувальної вибірки. В нашому випадку це означає, що мережа зможе добре передбачати тільки ті напрямки обличь, які траплялись у датасеті. Також дуже важливо щоб обличчя у датасеті були сфотографовані при різноманітному освітленні і мали якомога різну зовнішність.

Ще одним дуже важливим критерієм при підборі датасета є його розмір. По-перше, це напряму пов’язано з його різноманітністю, бо в більшому датасеті може поміститися більше унікальних обличь в унікальних умовах освітлення, навколишньої обстановки тощо. По-друге, емпірично доведено, що чим більший датасет, тим менше виникають проблеми з перенасиченням мережі, що на практиці є дуже важливим [14].

Отже основними критеріями для датасета є:

1. В датасеті мають бути лиця з ВВ
2. В датасеті мають бути якомога різноманітні дані
3. Датасет має бути якомога більшим

4.3 Датасет CelebA

Для тренування НМ в цій роботі був використаний датасет CelebA [8].

Це великий датасет з розміченими обличчями знаменитостей. Розмір цього датасету близько двохсот тисяч картинок. Кожне обличчя має BB та якорі див. рис. 3.4.

В цьому датасеті велика кількість різноманітних картинок різних знаменитостей на різному фоні. Окрім того це один з найбільших доступних датасетів з розміченими лицями.

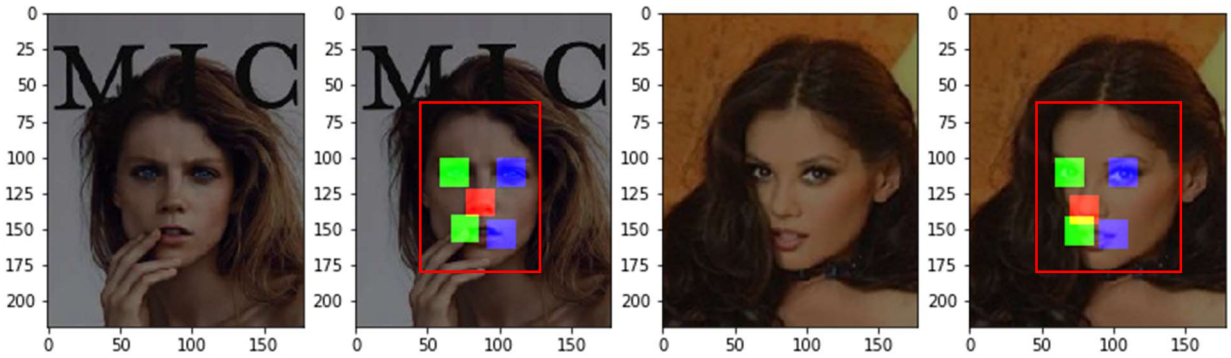

Рисунок 4.1 – Приклад даних с датасету CelebA

4.4 Генерація додаткових даних

Мережа FPNet має визначати BB та направлення обличчя, яке міститься в межах цього BB, але окрім цього вона також має визначати імовірність того, що це саме обличчя а не щось інше.

В результаті роботи другого етапу MTCNN ми маємо багато BB, які не містять обличчя див. рис. 4.2. Це стається в наслідок того, що мережі P-Net та R-Net див. 3.1. створенні таким чином щоб забезпечувати максимальну частку дійсно вірних передбачень (BB у нашому випадку), через що також зростає частка хибно позитивних передбачень. Це відома закономірність, яка відображається у ROC-кривих [15].

Саме для того, щоб відсіяти ці хибно позитивні результати FPNet має повертати імовірність того, що на вхід подається саме зображення лиця. Фактично це задача бінарної класифікації.

Для того щоб навчити мережу класифікувати (відрізняти) лиця від всього іншого потрібно мати як зображення з лицями так і зображення без лиць.

До того ж дуже важливим є те, щоб під час тренування мережі мають використовуватись такі зображення, які вона буде отримувати від R-Net під час виконання. Це важливо тому що нейронні мережі можуть гарно працювати тільки в тому випадку, коли їм на вхід подаються дані подібні до використаних для її тренування.

Позитивна підмножина (частина датасету, яка містить лиця) формується наступним чином:

1. Для всіх картинок з CelebA запускається перші два етапи MTSNN
2. З отриманих BB вибираються ті, які містять усі лицеві якорі
3. Кожного лиця обирається один BB з тих що йому відповідають
4. Всі знайдені BB зберігаються

Третій пункт потрібен через те, що дуже часто після другого етапу одному лицю відповідає декілька BB див. рис. 4.2

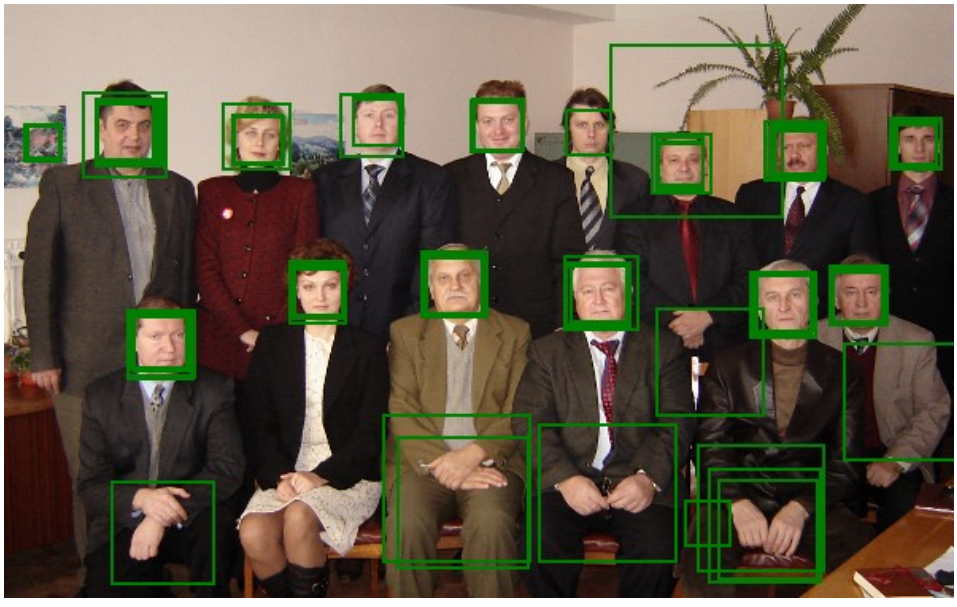

Рисунок 4.2 – Результат роботи другого етапу MTCNN

Спостерігаючи за тим, які BB пропонує R-Net, можна зробити висновок, що хибно позитивні пропозиції можна поділити на два типи: ті що лежать поза

межами обличь це добре помітно на рис. 4.2 і ті, що лежать всередині обличь, зазвичай навколо очей див. рис 4.3. Важливим є те, щоб у фінальний датасет потрапили хибно позитивні ВВ з обох цих класів.

Цікавим питанням є те, в якому співвідношенні мають траплятися хибно позитивні ВВ цих двох класів. У цій роботі на кожні два ВВ, що лежать поза межами обличчя приходиться один, що лежить всередині.

Негативна підмножина (частина датасету, яка не містить лиця) Формується наступним чином:

1. Обирається необхідна кількість ВВ, які не містять обличь
2. З випадково обраних картинок датасету CelebA обираються випадковим чином ВВ які належать до одного з класів.
3. Всі знайдені ВВ зберігаються

Знайдені описаним вище способом позитивна і негативна підмножини об’єднуються в один датасет, який буде використаний для тренування.

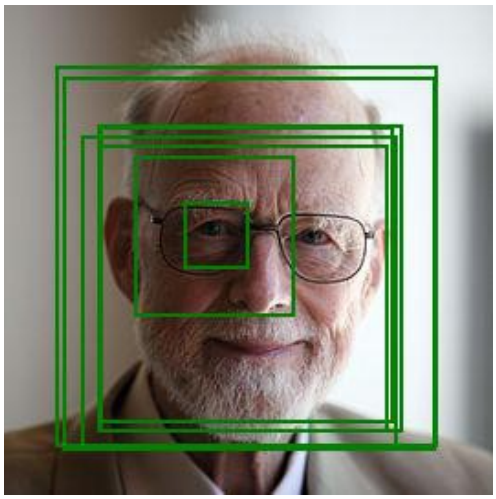

Рисунок 4.3 – Помилкові ВВ всередині обличчя після другого етапу MTCNN

4.5 Реалізація дистиляції знань

У минулому підрозділі описаний спосіб підготовки даних для того аби на них можливо було тренувати FPNet. Однак серед цих даних досі немає

повороту обличчя. У цьому підрозділі розповідається яким чином вони генеруються та використовуються у тренуванні.

Для зручності на всіх картинках з датасету CelebA був знайдений напрямок обличчя за допомогою 6DRepNet див. 3.2. Важливим є те, що при обчисленні зберігаються не тільки знайдені кути Ейлера, а й безпосередньо два тривимірних вектори, які повертає мережа.

Це робиться у відповідності до методу дистиляції знань. Легко побачити, що перетворення, за допомогою якого з векторів, повернутих мережею, отримують матрицю оберту, не можна обернути. Тобто при цьому перетворенні частина інформації втрачається. Це ніяк не впливає на тренування 6DRepNet, але тренуючи FPNet безпосередньо на цих двох векторах ми створюємо більше обмежень для мережі. Тобто для кожної картинки в нас начебто трохи більше даних. Це сприяє кращому навчанню.

Мережа 6DRepNet навчалась на картинках розміром 224 x 224 пікселів. Для того щоб запустити цю мережу на CelebA потрібно вирізати з них обличчя, масштабувати його до відповідного розміру і подати на вхід мережі.

Приклад обертів, визначених 6DRepNet можна побачити на рисунку 4.4.

Рисунок 4.4 – Оберти обличь з CelebA, визначені за допомогою 6DRepNet

4.6 Архітектура мережі

Архітектурно FPNet повторює O-Net з MTCNN за виключенням того, що на останньому шарі замість десяти значень для якорів, повертається шість

значень, які формують два трьох вимірні вектори, по яким потім визначається матриця оберту аналогічно до того, як це робиться у 6DRepNet див. 3.2.

Під час тренування використовувався метод регуляризації, який називається Dropout див. 1.6. Це дозволило зробити процес тренування дещо стабільнішим.

Функція втрат також є аналогічною до тої, яка використовується в MTCNN див. формулу 3.4. За тим виключенням, що функція втрат $L_i^{landmarks}$ замінюється на функцію втрат для векторів, що визначають оберт.

$$L_i^{rot} = \left\| \hat{y}_i^{rot} - y_i^{rot} \right\|_2^2 \tag{4.1}$$

4.7 Тренування FPNet

Тренування відбувалось за допомогою Minibatch Gradient Descend p оптимізатором Adam. Початковий коефіцієнт втрат (loss rate) дорівнює 0.01. Розмір батчу під час тренувань становить 64 картинки. Тренування відбувається протягом 30 епох. Начення функцій втрат для кожної епохи можна побачити на рис. 4.5.

Кожну нову епоху LR множиться на 0.83 тобто його значення на епосі t:

$$L_t = L_1 k^{t-1} \tag{4.2}$$

де L_t – LR на t-тій епосі, L_1— LR на першій епосі, k – коефіцієнт.

Основними гіперпараметрами цієї мережі є безпосередньо розміри ядер згорток, кількість шарів та параметри оптимізатора.

Під час тренування виявилось, що ця мережа схильна до потрапляння в локальні мінімуми це проявлялось тим, що при деяких значеннях LR та k мережа починала повертати однакові значення на будь-яких вхідних даних. При цьому ці значення відповідали матимачним сподіванням відповідних величин. Таким чином виявилось вкрай важливим підібрати вдалі значення для цих двох гіперпараметрів.

Простеживши за тим як поводить себе мережа під час тренування була сформульована гіпотеза, по на вхід подається багато «легких» прикладів.

Тобто таких, що мають приблизно однакові параметри BB та направлення. Оскільки загалом у вибірці багато таких прикладів, навчена мережа може бути більш схильна до помилок на «складних» прикладах. Для вирішення цієї проблеми з кожного мінібатчу обирається 60% прикладів, які є ані надскладними, тобто такими що мають великі значення функції втрат і можуть зробити тренування нестабільним, ані надлегкими, тобто такими що мають малі значення функції втрат і сприяють потраплянню в локальний мінімум. Такий підхід зазвичай називають пошуком складного прикладу (Online Hard sample mining) [16].

Змінюючи кількість та розмір прихованих шарів можна збільшити точність моделі. Проте виявилось, що для значного покращення точності необхідно сильно збільшити кількість параметрів мережі, що сказується на її швидкодії. Тому було вирішено залишити архітектуру аналогічною до O-Net.

Результат роботи натренованої мережі можна побачити на рис. 4.6

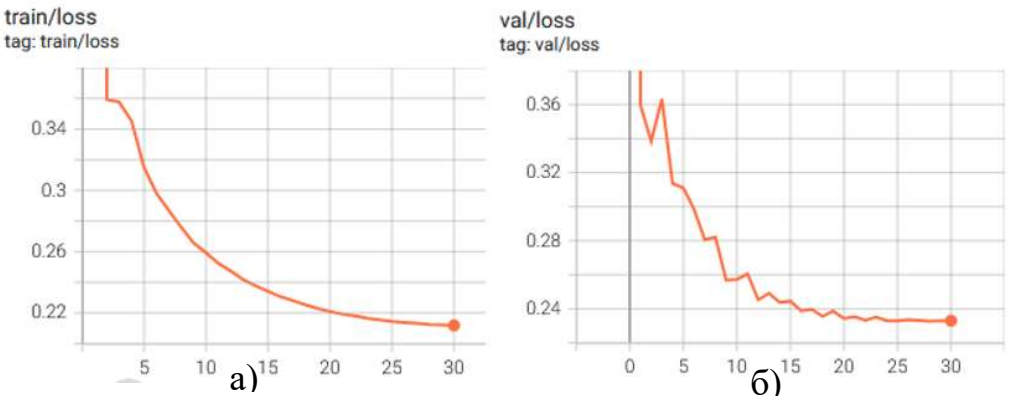

Рисунок 4.5 – Значення функції втрат: а) -- для тренувальної вибірки; б) – для валідаційної вибірки

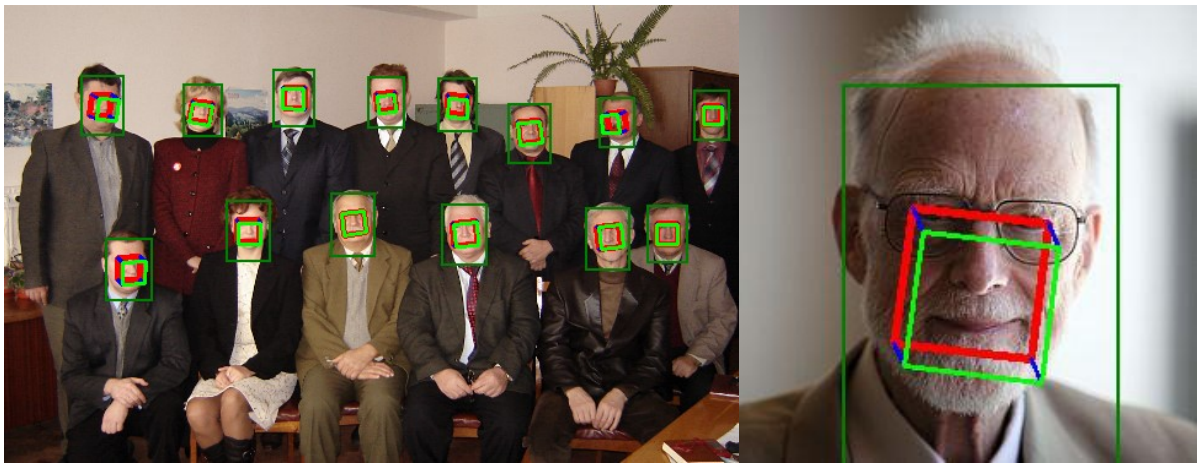

Рисунок 4.6 – Приклад роботи запропонованого методу

4.8 Створення застосунку

Оскільки розробка відбувалась в середовищі Google Colab див. 2.3, взаємодія з програмним кодом відбувалась через інтерфейс Jupyter Notebooks. Це зручно під час розробки, проте не дає можливість запускати пошук напрямків обличь користувачем напряму.

Тому було прийняте рішення створити простий консольний додаток, який дозволив би швидко і без зайвих проблем запускати рішення.

Оскільки вся кодова база цього проекту написана на мові Python, додаток також було створено як Python програму.

Для розбору аргументів був використаний модуль argparse, для роботи з зображеннями бібліотека PIL.

Програма може приймати на вхід як одну картинку і відповідний шлях за яким буде створено результат, так і шлях до папки з картинками, у такому випадку на усіх картинки, які містяться в цій папці будуть знайдені лиця разом з їх напрямком, результат буде записаний в зазначену в аргументах паку для результатів.

Для того щоб нею скористуватися потрібно завантажити репозиторій з кодом проекту. Встановити його за допомогою команди pip install. І викликати Запустити скрипт detect.py, передавши в нього необхідні аргументи. Список усіх доступних аргументів див. рис. 4.7.

```
usage: detect.py [-h] [--face_pose_net ] [--cuda] src dest

обов’язкові аргументи:
  src                   Шлях до окремої картинки або до папки з картинками
  dest                  Шлях до папки, в якій буде збережено результат

необов’язкові аргументи:
  -h, --help            Показує повідомлення з допомогою
  --face_pose_net       Шлях до збережених вагів FPNet
  --cuda                Чи треба використовувати GPU
```

Рисунок 4.7 – Доступні аргументи для додатку

ВИСНОВКИ

У цій роботі було створено систему, що дозволяє знаходити обличчя, та їх напрямок за зображенням. Для цього було проаналізовано багато наукових статей на цю тему. В результаті цього аналізу було запропоновано поєднати два рішення, а саме MTCNN та 6DRepNet. Також був запропонований та реалізований метод поєднання цих рішень.

Оскільки MTCNN дозволяє швидко і точно знаходити обличчя, було прийнято рішення розширити його функціонал таким чином, щоб він додатково знаходив напрямок обличчя. Це було зроблено шляхом заміни однієї з нейронних мереж методу MTCNN, а саме O-Net на мережу FPNet.

Для тренування FPNet був застосований метод дистиляції знань. При цьому мережею вчителем була 6DRepNet.

Були проаналізовані існуючі датасети з обличчями, визначені їх переваги та недоліки. В решті решт для тренування мережі було обрано датасет з великою кількістю різноманітних лиць CelebA.

Був підготовлений відповідний датасет для тренування, зроблений на основі датасету CelebA. Для тренування мережі потрібна була значна кількість не тільки зображень обличь, а й значна кількість зображень, на яких їх нема. Зображення без обличь були знайдені алгоритмічним чином з того ж датасету.

Для більш ефективного тренування мережі був реалізований метод пошуку складних прикладів (Online Hard sample mining) шо дозволило збільшити точність для крайніх випадків, а також зробило тренування більш стійким.

Для зручності використання розробленої системи був створений простий консольний застосунок.

Оцінку одержаних результатів. Створена модель добре знаходить обличчя та їх напрямки. Окрім того вона здатна швидко працювати навіть без використання графічного процесора. Це можливо за рахунок того, що у методі

використовується декілька невеликих нейронних мереж, кожна з яких працює швидко.

Запропонований метод відповідає сучасним концепціям комп'ютерного зору. І є результатом поєднання технологія, які активно використовуються у сьогоденні.

Можливі галузі використання результатів роботи. Сьогодні чітко прослідковується тенденція до цифровізації людства. В побуті стає все більше «розумних» пристроїв, оснащених камерами, які можуть використовуватись для взаємодії з людиною. Однак часто такі пристрої не мають значних обчислювальних потужностей, які б дозволили використовувати сучасні дуже вимогливі до ресурсів нейронні мережі. При цьому деякі обрахунки такі пристрої зобов'язані робити локально, тобто без використання хмарних сервісів. Такі вимоги продиктовані необхідністю збереження приватної інформації користувачів.

Також треба зазначити що задачі з розпізнавання обличь та їх параметрів можуть виникати в таких сферах як доповнена реальність, допомога в управлінні пересувними засобами (машинами, літаками та ін.) тощо.

Наукова значущість. У цій роботі показана ефективність використання методу дистиляції знать, застосовано до задачі знаходження напрямку обличь на зображенні. Доведена можливість створення швидкого рішення поставленої задачі, яке б змогло ефективно працювати навіть без використання GPU або інших прискорювачів.

Доцільність продовження досліджень або розробок за відповідною тематикою. Як і для будь-якої іншої НМ точність дуже сильно залежить від використаного датасету. Але відомо багато різних методів, які дозволяють отримувати кращі результати, використовуючи ті самі данні.

Тому в подальших дослідженнях цього методу потрібно сконцентруватися саме на них. Можна, наприклад, запропонувати спосіб аугментації даних для датасету, описаного у цій роботі. Також можна

зосередитись на підборі більш вдалих гіперпараметрів мережі, використавши один з методів автоматичного пошуку архітектури (Automatic neural architecture search) [17]. Також існує можливість оптимізації самого застосунку. Скоріш за все, після перенесення коду з Python PyTorch на фреймворк, який більше підходить для швидкого виконання, і оптимізації коду, можна очікувати на більшу швидкодію застосунку.

Таким чином виконано всі поставлені задачі.

ПОСИЛАННЯ

1. Krizhevsky A. ImageNet Classification with Deep Convolutional Neural Networks / A. Krizhevsky, I. Sutskever, G. E. Hinton. – 2012.
2. Synthetic Data Generation [Електронний ресурс] – Режим доступу до ресурсу: https://research.aimultiple.com/synthetic-data-generation/.
3. Kaipeng Z. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks / Z. Kaipeng, Z. Zhanpeng, L. Zhifeng. – 2016.
4. Hempe T. 6D Rotation Representation For Unconstrained Head Pose Estimation / T. Hempe, A. A. Abdelrahman, A. Al-Hamadi. – 2022.
5. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/.
6. PyTorch [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/.
7. Tensorboard [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/tensorboard.
8. CelebA Dataset [Електронний ресурс] – Режим доступу до ресурсу: https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.
9. Hinton G. Distilling the Knowledge in a Neural Network / G. Hinton, O. Vinyals, J. Dean. – 2015.
10. Simonyan K. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION / K. Simonyan, A. Zisserman. – 2014.
11. IEEE [Електронний ресурс] – Режим доступу до ресурсу: https://www.ieee.org/.
12. You Only Look Once: Unified, Real-Time Object Detection / J.Redmon, S. Divvala, R. Girshick, A. Farhadi. – 2015.
13. Xiaohan D. RepVGG: Making VGG-style ConvNets Great Again / D. Xiaohan, Z. Xiangyu. – 2021.

14. Althnian A. Impact of Dataset Size on Classification Performance: An Empirical Evaluation in the Medical Domain / A. Althnian, D. AlSaeed, H. Al-Baity. – 2021.
15. Classification: ROC Curve and AUC [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.
16. Shrivastava A. Training Region-based Object Detectors with Online Hard Example Mining / A. Shrivastava, A. Gupta, R. Girshick. – 2016.
17. What is neural architecture search? AutoML for deep learning [Електронний ресурс] – Режим доступу до ресурсу: https://www.infoworld.com/article/3648408/what-is-neural-architecture-search.html.