**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра комп'ютерних наук**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ ОБЛИЧ З ВИКОРИСТАННЯМ PYTHON»**

Виконав: студент 4 курсу, групи 6.1220-з
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
О.А. Накоренко
(ініціали та прізвище)
Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Д'яченко Н.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, к.ф.-м.н., Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2024

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

**ЗАТВЕРДЖУЮ**
Завідувач кафедри комп'ютерних наук,
д.т.н., доцент

_____ Шило Г.М.
(підпис)

" 22 " грудня 2023 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Накоренко Олегу Анатолійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка системи розпізнавання облич з використанням Python

керівник роботи Д'яченко Наталія Миколаївна, к.ф.-м. н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2181-с

2. Строк подання студентом роботи 15.05.2024

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка системи розпізнавання облич з використанням Python

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 22.12.2023

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 01.02.2024 | |
| | | | |
| 2. | Збір вихідних даних. | 07.03.2024 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 12.04.2024 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 20.04.2024 | |
| | | | |
| 5. | Розробка третього розділу. | 10.05.2024 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 15.05.2024 | |
| | кваліфікаційної роботи бакалавра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 30.05.2024 | |

Студент _____ О.А. Накоренко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.М. Д’яченко
(підпис) (ініціали та прізвище)

**Нормоконтроль пройдено**

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка системи розпізнавання облич з використанням Python»: 67 с., 19 рис., 10 джерел.

КОМП'ЮТЕРНИЙ ЗІР, МЕТОД ГРАДІЄНТНОГО СПУСКУ, РОЗПІЗНАВАННЯ ОБЛИЧ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, OPENCV, PYTHON, PYTORCH, YOLO V8.

Об'єкт дослідження – зображення, які розпізнаються в процесі обробки нейронними мережами.

Предмет дослідження – локалізація об'єктів на зображеннях.

Мета роботи: розробка програмного застосунку для розпізнавання облич на зображеннях.

Метод дослідження – практичний.

У кваліфікаційній роботі виконано такі завдання: розглянуто існуючі методи розпізнавання облич. Проведений вибір мови програмування та засобів розробки. Підготовлено базу даних із зображеннями для навчання нейромережі. Здійснено навчання нейромережі на базі даних із зображеннями. Реалізовано розпізнавання людського обличчя. Проведено тестування розробленої програми.

Розроблена програма може бути використана, наприклад, в системах контролю доступу на підприємствах або в правоохоронних органах для пошуку злочинців.

# SUMMARY

Bachelor’s Qualifying Theses «Development of a face recognition system using Python»: 67 pages, 19 figures, 10 references.

COMPUTER VISION, GRADIENT DESCENT METHOD, FACE RECOGNITION, CONVOLUTIONAL NEURAL NETWORK, OPENCV, PYTHON, PYTORCH, YOLO V8.

Object of the study – images that are recognized during processing by neural networks.

Subject of research – localization of objects in images.

Aim of the study: development of a software application for face recognition in images.

Method of research – practical.

The following tasks were performed in the qualification work: existing methods of face recognition were considered. The choice of programming language and development tools was made. A database of images for training the neural network was prepared. The neural network was trained on a database of images. Human face recognition was implemented. The developed application was tested.

The developed program can be used, for example, in access control systems at enterprises or in law enforcement agencies to search for criminals.

**ЗМІСТ**

Завдання на кваліфікаційну роботу ..... 2
Реферат ..... 4
Summary ..... 5
Вступ ..... 8
1 Аналіз предметної області ..... 9
1.1 Актуальність обраної теми ..... 9
1.2 Комп’ютерний зір ..... 9
1.3 Машинне навчання ..... 11
2 Огляд технологій нейронних мереж ..... 12
2.1 Поняття нейрона, нейронної мережі ..... 12
2.2 Функція активації ..... 14
2.3 Глибока нейронна мережа ..... 15
2.4 Структура згорткової нейронної мережі ..... 16
2.5 Навчання згорткових нейронних мереж ..... 19
2.6 Оцінка максимальної правдоподібності ..... 20
2.7 Функція вартості ..... 21
2.8 Метод зворотного розповсюдження помилки ..... 24
2.9 Метод градієнтного спуску ..... 25
2.10 Регуляризація нейронної мережі ..... 26
2.11 Метод Віоли-Джонса ..... 28
2.12 Афінні перетворення. ..... 30
3 Розробка системи розпізнавання облич ..... 32
3.1 Вибір навчальних даних ..... 32
3.2 Вибір програмного забезпечення ..... 33
3.3 Підготовка даних для навчання нейронної мережі ..... 34
3.4 Завантаження даних для навчання ..... 36
3.5 Помилки у задачі ідентифікації ..... 39

3.7 Архітектура нейронної мережі для отримання bottleneck ознак..... 40
3.7 Реалізація функції втрат на основі триплетів ..... 41
3.8 Процес навчання та тестування..... 42
Висновки ..... 46
Перелік посилань..... 47
Додаток А Тестування роботи программи ..... 50
Додаток Б Файл підготовки та обробки даних..... 51
Додаток В Програмна реалізація методів обробки зображень..... 56
Додаток Г Програмна реалізація архітектури нейронної мережі ..... 58
Додаток Д Програмна реалізація нейронної мережі для триплетів..... 62
Додаток Е Навчання нейронної мережі як класифікатора..... 64
Додаток Ж Донавчання нейронної мережі триплетами. ..... 65
Додаток И Підрахунок метрики для вимірювання якості ..... 67
Додаток К Тестування навченої мережі ..... 68

# ВСТУП

Технології комп’ютерного зору широко застосовують у різних сферах діяльності. Одним із завдань, яке можна вирішити за допомогою даної технології, є виявлення та ідентифікація людей на фото та відеоматеріалах. Виявлення характеристик, за допомогою яких з певною точністю, можна буде зробити висновок про те, чи присутнє на зображенні обличчя людини чи ні. Така потреба виникає у багатьох сферах діяльності: починаючи від соціальних мереж, де необхідно визначити всіх людей, зображених на фото, і закінчуючи виявленням небезпечних осіб із відеопотоку камер вуличного спостереження.

У цій кваліфікаційній роботі розглядається розробка програми для виявлення облич на зображеннях. У роботі розглянуто існуючі методи для розпізнавання облич, а також вирішується завдання виявлення суб’єкта на зображенні. У результаті отримана програма, робота якої розрахована на відносно невеликі обчислювальні потужності та на обмежену кількість навчальних даних. Точність роботи створеної програми близька до точності вже існуючих програм, які використовуються для розпізнавання облич.

У ході виконання кваліфікаційної роботи були виконані наступні завдання.

- розглянуто методи, що застосовуються для розпізнавання облич;
- розглянуто структури нейронних мереж;
- сформовано вибірку фотографій для навчання нейромережі;
- побудовано структуру згорткової нейронної мережі;
- навчено нейронну мережу;
- проведено тестування роботи програми.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Актуальність обраної теми

Розпізнавання облич є важливою сферою в області обробки зображень та комп’ютерного зору і є одним з найуспішніших додатків для аналізу та розуміння зображень.

У соціальній діяльності людини обличчя є основним об’єктом для визначення ідентичності та емоцій. Протягом життя люди бачать величезну кількість чужих облич, але навіть після довгих років розлуки здатні з першого погляду миттєво розпізнати знайомі обличчя. Такому візуальному сприйняттю не заважають навіть значні зміни обличчя через старіння або відволікаючі фактори, такі як зміни в зачісці або носіння окулярів. Відповідно розробники намагаються створювати такі автоматизовані системи, які можуть імітувати людське візуальне сприйняття і можуть бути використані для автоматизації процесу розпізнавання облич.

## 1.2 Комп’ютерний зір

Комп’ютерний зір – це область інформатики, яка навчає комп’ютер бачити об’єкти. Це спосіб, за допомогою якого комп’ютери збирають та інтерпретують візуальну інформацію з навколишнього середовища. Комп’ютер як людина намагається зрозуміти, що показано на зображенні.[1]

Принцип роботи комп’ютерного зору: зображення спочатку обробляється на низькому рівні, щоб поліпшити якість зображення, наприклад, видалити шум. Потім зображення обробляється на вищому рівні, щоб розпізнати малюнки або форми і таким чином визначити, що показано на зображенні.

На комп’ютері цифрове зображення можна подати у вигляді багатовимірного масиву. Наприклад, зображення в градаціях сірого можна представити як двовимірний масив, де кожен елемент визначає інтенсивність сірого кольору у конкретному пікселі, як показано на рис. 1.1. Якщо зображення кольорове, то масив описує значення кольору в каналі відповідно до колірної моделі у певному пікселі зображення, як показано на рис. 1.2.

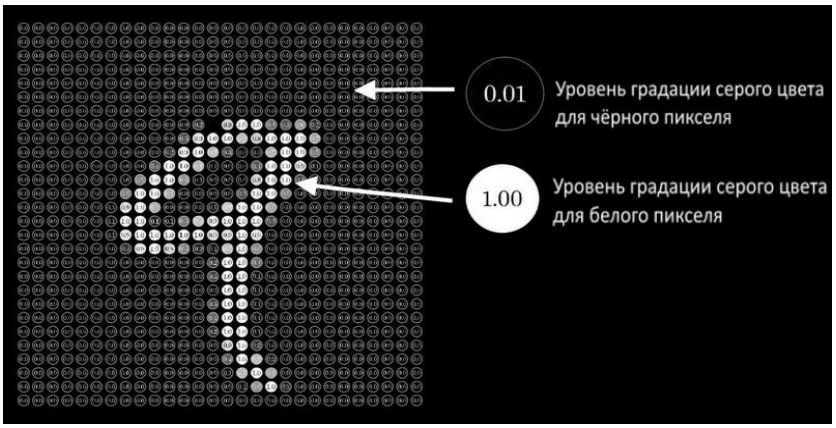



Рисунок 1.1 – Приклад опису зображення у градаціях сірого [2]

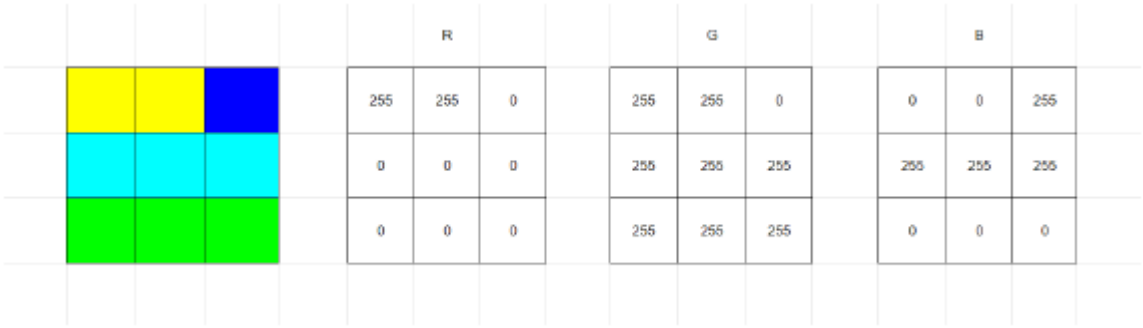



Рисунок 1.2 – Приклад опису кольорового зображення у форматі RGB [3]

Приклади використання комп’ютерного зору:

– виявлення об’єктів;
– розпізнавання об’єктів;
– класифікація зображення.

Методи та розробки, які були запропоновані в минулому столітті, поступово витісняються новими методами та розробками, що ґрунтуються на глибокому навчанні та нейромережах [1].

## 1.3 Машинне навчання

Машинне навчання – це розробка алгоритмів, що використовуються в комп'ютерних системах для виконання завдань без явних інструкцій. Комп'ютерні системи використовують алгоритми машинного навчання для обробки великих об'ємів статистичних даних. В результаті системи на основі заданого набору вхідних даних можуть точніше прогнозувати результати. [4]

Часто розробка програм потребує багато часу або взагалі неможлива. Наприклад, не вдасться вручну написати програму, яка дозволить комп'ютеру розпізнавати зображення різних людей. У свою чергу, людина може зробити це без проблем. Машинне навчання використовує такий підхід, який дозволяє комп'ютерам вчитися програмувати самих себе на основі отриманого досвіду.

Машинне навчання починається з підбору даних – чисел, фотографій чи тексту. Дані збираються та готуються для використання в якості навчальних даних, на яких буде навчатися модель машинного навчання.

Далі вибирається модель та надаються дані, щоб комп'ютерна модель могла знаходити закономірності чи робити прогнози. Згодом програміст може налаштовувати модель, наприклад, змінювати її параметри, щоб досягти точніших результатів.

Деякі дані з навчальної бази зберігаються і можуть бути використані як оціночні дані, щоб перевірити, наскільки точно працює модель. В результаті виходить модель, яку можна використовувати з різними наборами даних [4].

Принцип роботи машинного навчання показаний на рис. 1.3.

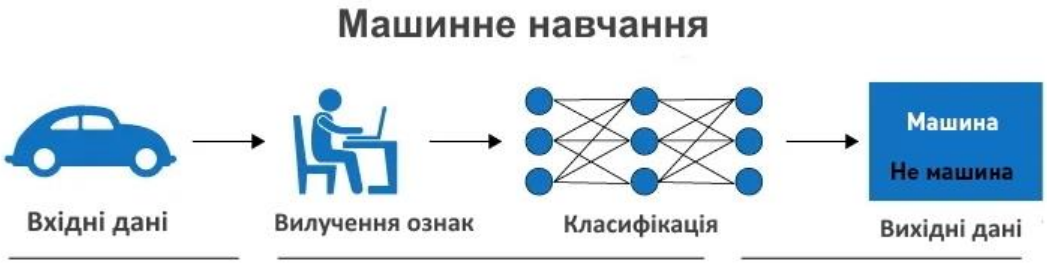



Рисунок 1.3 – Принцип роботи машинного навчання [5]

# 2 ОГЛЯД ТЕХНОЛОГІЙ НЕЙРОННИХ МЕРЕЖ

## 2.1 Поняття нейрона, нейронної мережі

Штучні нейронні мережі виникли внаслідок бажання людини відтворити роботу нейронів головного мозку. Їхній основний принцип – це вміти навчатися на помилках і не допускати їх надалі. Конструкція штучних нейронних мереж схожа на конструкцію їхніх біологічних прототипів – нейронів головного мозку. Модель штучного нейрона представлена на рис. 2.1.

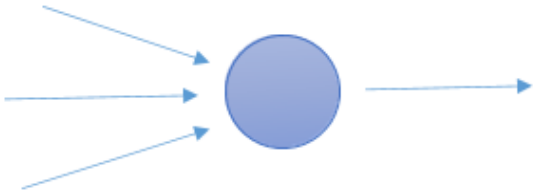


Рисунок 2.1 – Зображення штучного нейрона, що має кілька входів, ядро обробки інформації та один вихід

Штучний нейрон одержує вхідні сигнали через кілька вхідних каналів. Кожен вхідний сигнал проходить через з’єднання, що має певну вагу. Обчислюється виважена сума входів та її зміщення. В результаті одержуємо величину активації нейрона.

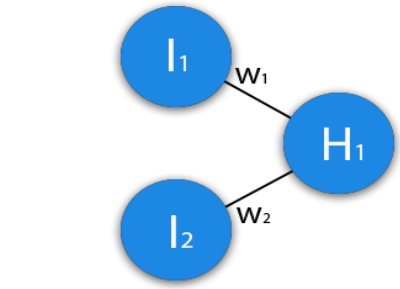
Залежно від розташування в нейронних мережах, нейрони можуть бути трьох видів: вхідні, приховані, вихідні. Нейрони розташовуються шарами. Шар нейронів – група нейронів, яка знаходиться на одному рівні в нейронній мережі та приймає дані від попереднього шару нейронів. [6]

Основні параметри нейрона – це його вхідні та вихідні дані. У вхідного нейрона вхідні дані рівні вихідним даним, оскільки вхідний нейрон не виконує жодних обчислень. Його завдання – передати сигнал від входу далі в нейронну мережу. Для двох видів нейронів (приховані і вихідні нейрони) вхідні дані

дорівнюють сумі входів у цей нейрон, а вихідні дані рівні значення функції активації, яка застосовується до активації даного нейрона.

Синапси – це зв'язок між нейронами. Синапси мають лише один параметр – вага, і використовуються для зміни інформації, яка передається від одного нейрона до іншого. Саме за рахунок зміни ваг, нейронна мережа може вчитися і певним чином аналізувати вхідні дані [6].

Продемонструємо роботу окремо взятого нейрона на рис. 2.2.




$$1) H_{1\,input} = (I_1 * w_1) + (I_2 * w_2)$$

$$2) H_{1\,output} = f_{activation}(H_{1\,input})$$

Рисунок 2.2 – Приклад розрахунку входу та виходу одного нейрона [7]

Нейронна мережа – кілька нейронів, об'єднаних в єдину архітектуру, що містить вхідні нейрони, проміжні (приховані) нейрони і вихідні нейрони.

Приклади нейронних мереж зображенні на рис.2.3.

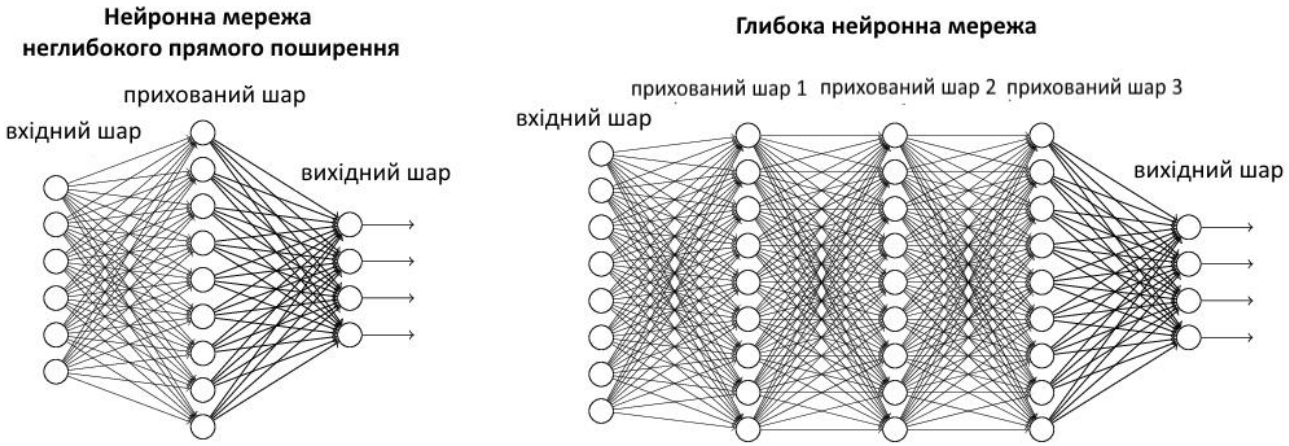



Рисунок 2.3 – Приклад неглибокої та глибокої нейронної мережі [8]

## 2.2 Функція активації

Функція активації виконує обробку вихідних даних після підрахунку суми добутків входів на їх синапси та додавання зміщення (активація нейрона). Вигляд функції активації визначається конкретним завданням. [9]

Зміщення – деяке число, яке робить внесок у значення активації, навіть якщо всі інші доданки дорівнюють 0. Вихідний сигнал нейрону – це сигнал активації перетворений за допомогою функції активації. Нейрон – це обчислювальна одиниця, яка здійснює обчислення над вхідними даними і передає оброблену інформацію далі.

Зміщення також є нейроном. Але його конструкція відрізняється від звичайного нейрона, тому він не був врахований у зазначеній вище класифікації. Головною особливістю переміщення є відсутність входу. Через відсутність вхідних даних, вихідне значення дорівнює певній константі, а значення з’єднання з усіма нейронами поточного шару дорівнює одиниці.

Для різних типів нейронів використовують різні функції активації: функція одиничного стрибка, функція сигмоїдальна, гіперболічний тангенс, ReLU (Rectified linear units) [9].

Схема роботи функції активації зображена на рис. 2.4.

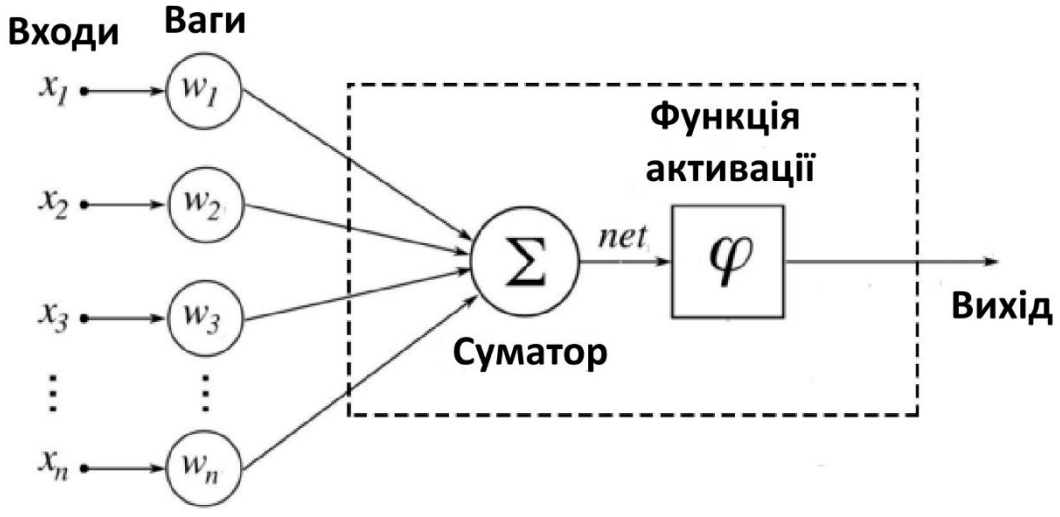



Рисунок 2.4 – Схема роботи функції активації [10]

## 2.3 Глибока нейронна мережа

Глибоке навчання – це набір алгоритмів машинного навчання для моделювання високорівневих абстракцій, які використовують архітектури, що включають численні нелінійні перетворення.

Глибока мережа прямого поширення, так званий багатошаровий персептрон, є типовим прикладом моделі глибокого навчання. Багатошаровий персептрон використовується для апроксимації деякої функції $f^*$. Він визначає деяке відображення $y = f(x, \theta)$ і під час навчання знаходить таке значення $\theta$, яке дає найкращу апроксимацію. З урахуванням введених визначень, можна зробити такий висновок: неперервну функцію кількох змінних можна апроксимувати з будь-якою точністю $\varepsilon > 0$, за допомогою двошарового персептрону, з достатньою кількістю нейронів у прихованому шарі та їхньою нелінійною активаційною функцією. [11]

Якщо уявити вхідні дані як набір змінних, від яких залежить функція, значення якої є міткою вхідних даних, з допомогою нейронних мереж можна побудувати якісне наближення цієї функції. Виникає питання: яку нейронну мережу необхідно побудувати, щоб вона змогла відобразити усі необхідні залежності із хорошою точністю.

При вирішенні завдань, для яких вхідними даними є зображення, використовуються нейронні згорткові мережі. На даний момент згорткова нейронна мережа та її модифікації вважаються кращими за точністю та швидкістю знаходження залежностей на зображеннях. Вже 6 років згорткові нейронні мережі займають перші місця на відомому міжнародному конкурсі з розпізнавання образів ImageNet, що є найкращим доказом застосування їх до поставленого завдання. [11]

Давайте визначимо, чому при роботі із зображеннями дана архітектура має перевагу щодо персептрона.

## 2.4 Структура згорткової нейронної мережі

Основною причиною хорошої функціональності згорткових нейронних мереж стала концепція загальної ваги. Незважаючи на їхній великий розмір, ці мережі мають невелику кількість параметрів, порівняно з їх попередником – багатошаровим персептроном.

Згорткова нейронна мережа будується з наступних шарів: згорткові шари (convolutional) та субдискретизуючі шари (subsampling, підвибірка). На виході мережі зазвичай використовують персептрон. Варто зауважити, що в сучасних архітектурах згорткових мереж відмовилися від використання пов’язних шарів. Ці шари мають неприємну властивість – наявність величезної кількості зв’язків між нейронами, що збільшує обчислювальну складність і відповідно навантаження на комп’ютерні системи. Згорткові та субдискретизуючі шари чергуючись між собою, формують вхідний вектор ознак для багатошарового персептрона або іншого вихідного шару мережі. [12]

Схема згорткової нейронної мережі показана на рис. 2.5

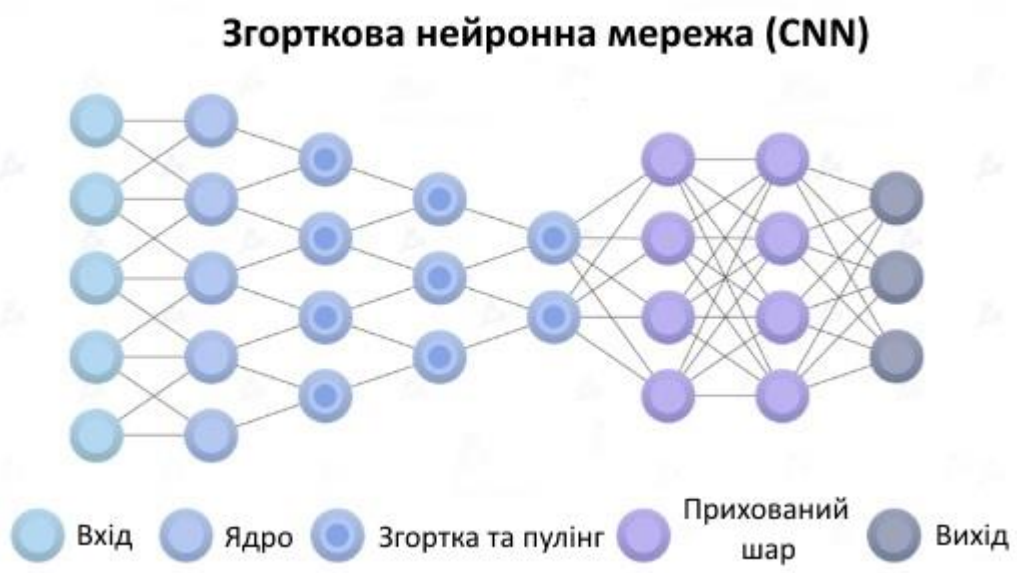



Рисунок 2.5 – Згорткова нейронна мережа [13]

Давайте розберемося що таке згортка. Для цього спочатку розглянемо поняття ядра згортки. Ядро згортки або фільтр є деяким тензором фіксованого розміру, який ковзає по вхідному тензору і виділяє певні ознаки вихідного об’єкта. На розмір фільтра накладаються такі вимоги: досить великий, щоб була можливість виділити будь-яку ознаку, і досить малий, щоб уникнути сильного зростання кількості зв’язків між нейронами. На практиці застосовуються фільтри розміром від 1x1 до 7x7 залежно від обраної архітектури нейронної мережі. Ядро згортки – це тензор з деякими числовими значеннями, за допомогою яких нейронна мережа виділяє ознаки об’єктів, які подаються на вхід. Процес згортки кольорового зображення нейронною мережею продемонстровано на рис. 9.

Виходом операції згортки є тензор або так звана «карта ознак». При необхідності до вхідного тензора застосовується процедура додавання нульових рядків і стовпців, щоб при здійсненні зміщення вікна згортки, не вийти за його межі або при певних вимогах розміру вихідної картки ознак.

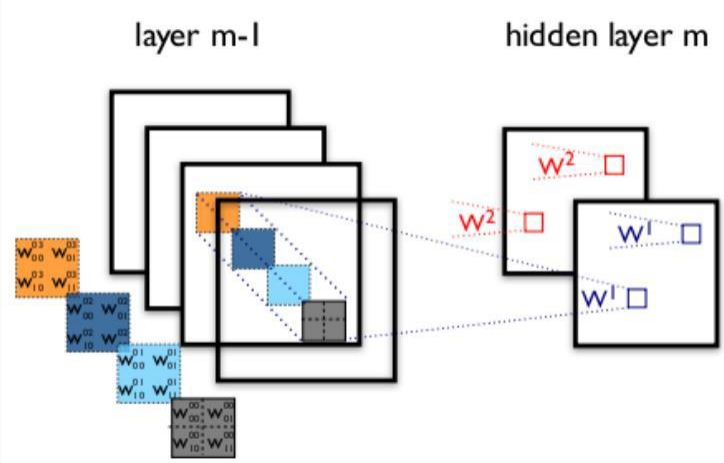



Рисунок 2.6 – Приклад згортки 4-мірного тензора [14]

Чим більше операцій згортки виконувалося перед черговим шаром згортки, тим складніші патерни здатні виділяти карти ознак. Це зумовлено тим, що наступні фільтри працюють із числовими значеннями не вихідного зображення, а вже виділеними ознаками на попередніх етапах.

Важливу роль у виявленні ознак і коректній обробці грає субдискредитуючий шар. З його допомогою вдається визначити інваріантність розташування ознаки. Також вдається реагувати на наявність ознаки, навіть якщо вона знаходиться трохи не в тому місці, де ми очікуємо її побачити або вона зображена під іншим кутом. Другим плюсом цього шару є зменшення розмірності тензора, що подається на нього. [15]

Аналогічно операції згортки тут застосовується фільтр чи вікно, яке проходить по вхідному тензору. Значення, що потрапили в область вікна, замінюються його статистикою. Це пулінговий шар. На практиці застосовується два види такого шару – Max Pooling та Average Pooling. Перший вид вибирає максимальний елемент з області, який потрапив у поле його фільтра, а другий вид бере середнє значення. Демонстрація їхньої роботи показана на рис. 2.7.

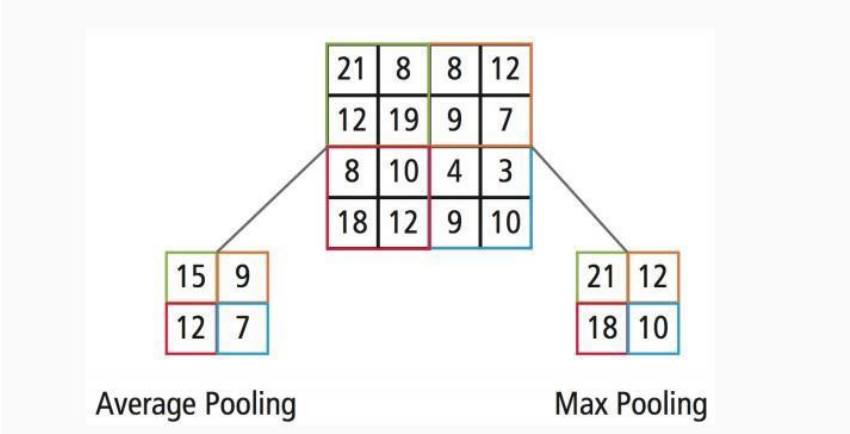



Рисунок 2.7 – Приклад роботи Average та Max пулінгів [16]

Після кількох проходів згортки зображення та ущільнення за допомогою пулінгу система перебудовується від конкретної сітки пікселів з високою роздільною здатністю до більш абстрактних карт ознак. Як правило, на кожному наступному шарі збільшується кількість каналів та зменшується розмірність зображення у кожному каналі.

Згорткові мережі – це щось середнє між біологічним пристроєм механізму сприйняття людиною зображення та багатошаровим персептроном.

На сьогоднішній день згорткові мережі показують найкращі результати у розпізнаванні зображень. У середньому точність розпізнавання таких мереж перевищує стандартні штучні нейронні мережі на 10-15%. Згорткові нейронні мережі – це ключова технологія у глибоких нейронних мережах [15].

## 2.5 Навчання згорткових нейронних мереж

Ми визначили, що використання нейронних мереж, це найкращий варіант вирішення поставленого завдання: розпізнавання зображень та виявлення осіб. Далі необхідно пояснити, яким чином нейронна мережа здійснюватиме апроксимацію шуканої функції та вчитиметься видавати нам бажаний результат. На самому початку, при описі штучної нейронної мережі, було визначено вимогу: здатність вчитися на помилках. Також була позначена функція $f(x, \theta)$, за допомогою якої здійснюватиметься апроксимація до невідомої функції $f$, яка описує існуючі залежності. Таким чином, завдання зводиться до коригування параметра $\theta$ на підставі помилок, які нейронна мережа допускає під час навчання.

Навчанням нейронної мережі називають процес обробки нейронною мережею різних позначених об’єктів і коригування внутрішніх параметрів мережі таким чином, щоб значення, що видається їй, збігалося з міткою об’єкта. Розглянемо докладніше, як відбувається процес навчання нейронної мережі. Програмна реалізація представлена у додатку А.

Навчання нейронної мережі зводиться до мінімізації функції помилки. Цей процес виконується за рахунок налаштувань вагових коефіцієнтів зв’язків між нейронами, зокрема і нейронами зміщення. Під функцією помилки слід розуміти різницю між отриманою відповіддю та бажаною відповіддю. Нехай на вхід нейронної мережі подаються 2 зображення, і вона повертає число в діапазоні від 0 до 1, де 0 – на зображеннях різні люди, 1 – однакові, а проміжні значення – це певна міра схожості. Наприклад, на вхід були подані два зображення однієї й тієї

ж людини. Припустимо, що вихід нейронної мережі дорівнював 0.6, а повинен дорівнювати 1. Після цього ваги вихідного шару нейронів коригуються відповідно до помилки, оскільки для нейронів вихідного шару мережі відомі їх фактичні та бажані значення. Програмна реалізація представлена у додатку Б.

Налаштування ваги зв’язків для таких нейронів є відносно простим. Але для попередніх шарів це завдання коригування ваг стає менш тривіальним.

Варто зауважити, що раніше не існувало ефективного алгоритму поширення помилки по прихованих шарах мережі, оскільки раніше був відсутній інтерес до глибоких нейронних мереж.

Для подальшого та більш докладного опису механізму навчання нейронної мережі потрібно запровадити низку понять та їх математичних описів. Багато алгоритмів глибокого навчання можна навчати за допомогою варіації наступних прийомів: функції вартості та процедури оптимізації моделі [15].

## 2.6 Оцінка максимальної правдоподібності

Перш ніж перейти до функції вартості, спершу визначимо поняття максимальної правдоподібності. Нам хотілося б не вгадувати функцію $f^*$, яка могла б виявитися хорошою оцінкою справжньої функції $f$, а мати якийсь загальний принцип, що дозволяє виводити функцію $f^*$, яка точно є досить хорошою оцінкою. Більш формально завдання оцінки максимальної правдоподібності полягає в тому, щоб розглядати її як мінімізацію розбіжності Кульбака-Лейблера між емпіричним розподілом, який визначається навчальним набором, і модельним розподілом. Отже, при навчанні моделі, що мінімізує розбіжність Кульбака-Лейблера, ми повинні мінімізувати тільки ту величину, яка є другою частиною різниці, оскільки вона залежить від параметрів нейронної мережі.

Мінімізація розбіжності Кульбака-Лейблера є мінімізацією перехресної ентропії між розподілами. Перехресна ентропія – функція вартості, що містить

логарифм різниці між істинним та емпіричним розподілом, що породжує нейронну мережу. Таким чином, бачимо, що максимальна правдоподібність – це спроба поєднати модельний розподіл з емпіричним розподілом.

Нам необхідний точний збіг істинного розподілу з розподілом, що породжує, але безпосереднього доступу до нього у нас немає.

## 2.7 Функція вартості

Функція вартості (цільова функція, функція втрат) складається як мінімум з одного члена, орієнтуючись на який у процесі навчання проводиться статистичне оцінювання. Мінімізація цієї функції дає оцінку максимальної правдоподібності. Вона також може включати додаткові члени, наприклад, для регуляризації.

Для поставленої задачі розпізнавання облич досить складно підібрати функцію вартості, оскільки це завдання не схоже ні на завдання класифікації, ні на задачу регресії, які є класичними в машинному навчанні. Для задачі ідентифікації людини за її двомірним зображенням використовуються нестандартні функції вартості: Range Loss, Triplet Loss, Center Loss та ін. Ідея навчання нейронної мережі за допомогою цих функцій полягає в тому, щоб проектувати зображення облич в деякий простір ознак таким чином, щоб різниці між ознаками одних і тих самих людей були малі, а між ознаками різних людей досить великими. [17]

У цій роботі використовуватиметься функція втрат під назвою TPE. Для навчання нейромережі необхідно підбирати по три зображення наступним чином: два зображення однієї людини та одне зображення іншої людини. У ході навчання нейромережі вагові коефіцієнти зв’язків нейронів повинні бути налаштовані таким чином, щоб ознаки тих самих людей наближалися один до одного, а ознаки третьої людини віддалялися, як показано на рис. 2.8.

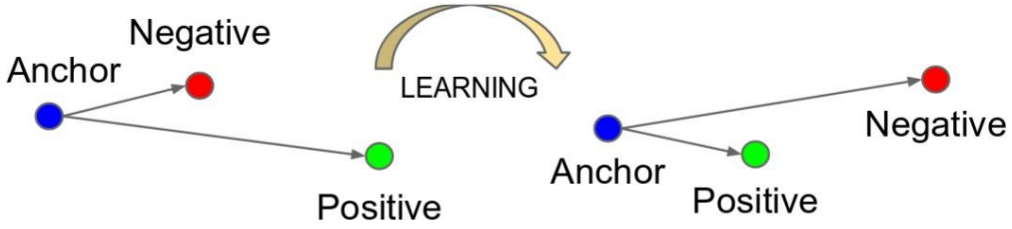



Рисунок 2.8 – Демонстрація процесу навчання нейронної мережі:
Positive, Anchor – це зображення однієї людини, Negative – іншої людини [18]

Ми не вимагаємо, щоб між об’єктами в триплетах одного класу та різних класів була мінімально допустима відстань. А також не вимагаємо, щоб була мінімальна ймовірність наявності триплетів, у яких відстань між класами різних об’єктів менша, ніж у однакових об’єктів. Як показали дослідження, такий підхід дає кращі результати за менший час, ніж більш старі підходи.

Далі визначимо підхід до навчання нейромережі та формальний вид функції втрат. Для початку позначимо триплет як:

$$t = (v_i, v_j, v_k)$$

де $v_i$ – розгорнуте зображення (anchor), $vj$ – зображення яке належить тій самій людині (positive), $vk$ – зображення іншої людини (negative).

Розглянемо функцію

$$S_W: R^N \times R^N \rightarrow R$$

яка має параметри матриці $W \in R^{n \times N}$, за допомогою якої вимірюється схожість між двома векторами $v_i, v_j \in R^N$.

В ідеалі для всіх триплетів t, які існують у наборі тренувань, ми хотіли б домогтися відповідності до наступного обмеження:

$$S_W(v_i, v_j) > S_W(v_i, v_k)$$

у нашому випадку

$$v_i \text{ та } v_j$$

є векторами характеристик, що генеруються нейронною мережею та нормованими на одиницю довжини.

Оптимізаційний вираз можна інтерпретувати як максимізацію ймовірності нерівності або мінімізацію негативної логарифмічної правдоподібності над множиною триплетів T.

Щоб навчити нейронну мережу виділяти ознаки із зображення обличчя людини, необхідно лише навчити її вирішувати завдання класифікації на закритій множині. Якщо нейронна мережа навчиться досить добре вирішувати задачу класифікації в закритій множині, то на передостанньому шарі буде зосереджена вся важлива інформація про зображення обличчя, навіть якщо суб'єкта не було в навчальній множині. Техніка вилучення числових значень ознак з останніх шарів навченої мережі зветься bottleneck.

На вихідному шарі в задачі класифікації стандартна нейронна мережа містить таку кількість нейронів, яка дорівнює кількості класів, а також містить функцію активації виду Softmax. Ця функція непросто повідомляє про те, якому класу належить об'єкт, а дає ймовірнісну оцінку приналежності, тобто як сильно нейронна мережа впевнена у своєму рішенні. Функція Softmax перетворює вектор Z розмірності K на вектор $\sigma$ тієї ж розмірності, де кожна координата представлена як числа в діапазоні від [0;1].

Як функція втрат у цьому випадку виступає перехресна ентропія в чистому вигляді, яка була описана в підрозділі 2.6.

## 2.8 Метод зворотного розповсюдження помилки

У процесі навчання згорткових мереж для коригування вагових коефіцієнтів використовується алгоритм зворотного поширення помилки (Backpropagation). Цей метод навчання називається узагальненим дельта-правилом і є основним для навчання багатошарових нейронних мереж. У цьому методі ваги прихованих нейронів повинні змінюватися прямо пропорційно до помилки тих нейронів, з якими дані нейрони мають зв’язки. Саме тому зворотне поширення цих помилок через нейронну мережу дозволяє коректно налаштовувати ваги зв’язків між усіма шарами нейромережі. Завдяки цьому розмір функції помилки зменшується і нейромережа навчається.

Далі розглянемо теоретичний опис методу зворотного розповсюдження помилки. А потім покажемо, як розраховуватиметься помилка для функції активації Softmax та функції втрат Triplet Loss.

Для опису роботи методу зворотного поширення помилки спочатку необхідно показати, як поширюється помилка через різні структурні елементи нейронної мережі. Список елементів, через які розповсюджується помилка: вихідний шар, повнозв’язковий шар, згортковий шар, пулінг шар.

Коригування вагових коефіцієнтів починається з вихідного шару мережі. Спочатку для кожного структурного елемента нейронної мережі обчислюються градієнти, які вплинули на прийняття рішення. Градієнти показують напрямок якнайшвидшого зростання функції за кожною з її параметрів, а його зворотні значення – спадання. Оскільки точність нейронної мережі та зміна вагових коефіцієнтів пов’язане безпосередньо з функцією вартості, то зменшення її значення якраз і дозволяє мінімізувати помилку, яку допускає недонавчена нейронна мережа.

Тепер розглянемо поширення помилки через шар Softmax. Як ми визначили раніше, мірою, за допомогою якої обчислюється помилка, є перехресна ентропія, яка використовується для знаходження оцінки максимальної правдоподібності.

Щоб розповсюдити помилку всередину нейромережі, необхідно застосувати правило диференціювання складних функцій (рис. 2.9). Помилка нейронної мережі залежить від значень, що видаються функцією Softmax, які залежать від значень нейронів вихідного шару нейронної мережі.

Перейдемо до подальшого розповсюдження помилки в глиб нейромережі. У звичайних згорткових мережах перед вихідними нейронами з функцією активації знаходився двошаровий персептрон. Розглянемо поширення помилки через цей шар [9].

Сигмоїдна або тангенціальна функції активації є нелінійними, але призводять до проблем із загасанням або збільшенням градієнтів. Для вирішення цієї проблеми можна використовувати набагато більш простий варіант функції – випрямлену лінійну функцію активації (ReLU).

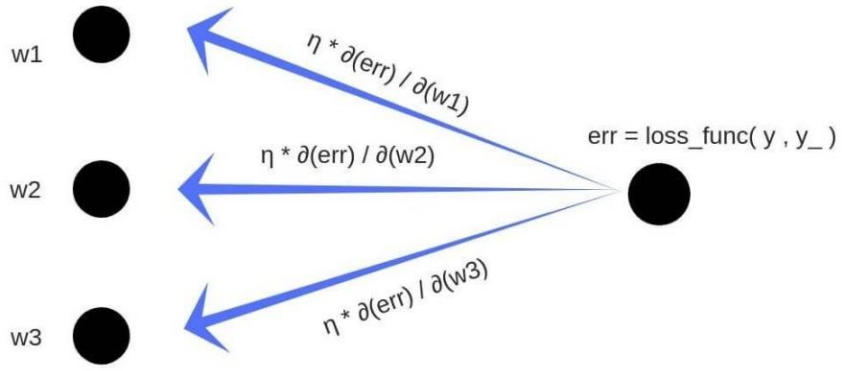



Рисунок 2.9 – Метод зворотного розповсюдження помилки [19]

## 2.9 Метод градієнтного спуску

Градієнтний спуск – це процес знаходження локальних екстремумів функції за допомогою руху вздовж градієнта. Градієнт – це вектор, який визначає кут нахилу гіперплощини і вказує на його напрямок щодо будь-якої з точок, що належать цій гіперплощині.

Градієнт обчислюється як похідна функції у точці, щодо якої ведеться дослідження. Рухаючись у напрямку цього градієнта, ми будемо плавно

скочуватись до екстремуму функції. Початковою точкою прийнято вважати помилку, отриману за деякої початкової ініціалізації вагових коефіцієнтів нейронної мережі. У міру навчання ми коригуємо всі навчальні структурні елементи нейронної мережі, за допомогою градієнтів, які ми вирахували при зворотному розповсюдженні помилки.

Цей метод не дає гарантії збіжності, тобто точного потрапляння до локального мінімуму. Однак найчастіше він досить швидко знаходить дуже мале значення функції вартості, щоб вважатися корисним. Але нас цікавить узагальнення алгоритму градієнтного спуску. Оскільки кількість параметрів нейронної мережі дуже велика, то оптимізувати всі параметри з погляду часу і обчислювальних потужностей дуже важко. [20]

Функція вартості, застосовувана в алгоритмах машинного навчання, часто представляється як суми всіх прикладів деякої функції втрат, визначеної для одного приклада.

Головною ідеєю стохастичного градієнтного спуску є заміна градієнта на його оцінку за невеликою вибіркою. Точніше, на кожному кроці алгоритму ми можемо взяти міні-пакет (minibatch) – невелику рівномірну вибірку з навчального набору B = $\{x(1), \ldots, x(m')\}$. Таким чином ми можемо апроксимувати багатомільярдний навчальний набір, роблячи обчислення лише на сотні прикладів.

## 2.10 Регуляризація нейронної мережі

Нейронні мережі є методами машинного навчання. Відповідно нейронні мережі мають проблеми перенавчання та недонавчання. Алгоритм машинного навчання працює добре в тому випадку, коли складність моделі відповідає справжній складності завдання та обсягу навчальних даних. Модель із недостатньою складністю не здатна вирішувати складні завдання. А модель із надмірною складністю призводить до небажаних результатів, оскільки починає

пояснювати залежності, яких насправді немає. Приклади недонавченої та перенавченої мережі показані на рис. 2.10.

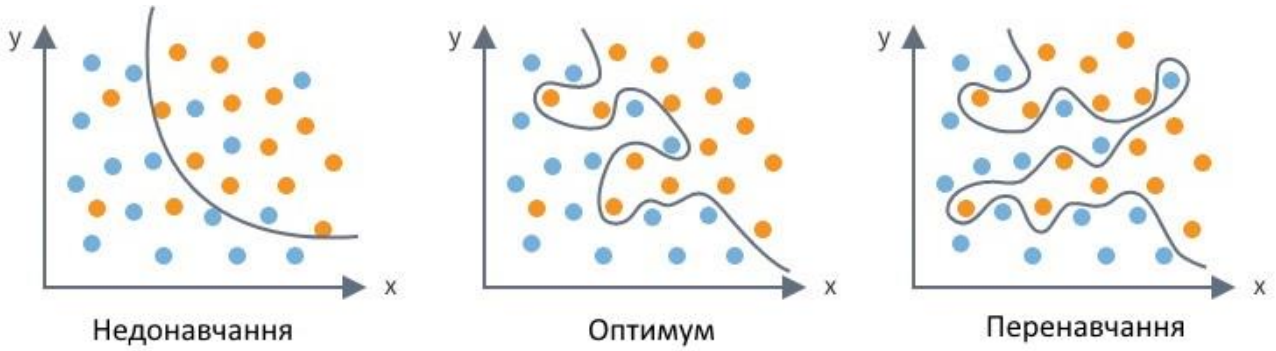



Рисунок 2.10 – Ілюстрація недонавченої, оптимальної та перенавченої моделі машинного навчання [21]

Перенавчання характеризується великою точністю моделі на навчальній множині і водночас частими помилками на тестовій вибірці, які вона не бачила під час навчання. Недонавчання характеризується однаково поганими результатами як тестової, і на навчальної вибірці [7].

У цій роботі використовується досить хороша архітектура нейронної мережі та ретельно підібраний навчальний та тестовий набір даних. Завдання регуляризації зводиться до уникнення перенавчання та забезпечення коректної роботи алгоритму навчання. Принцип вибору ємності моделі описано на рис. 2.11.

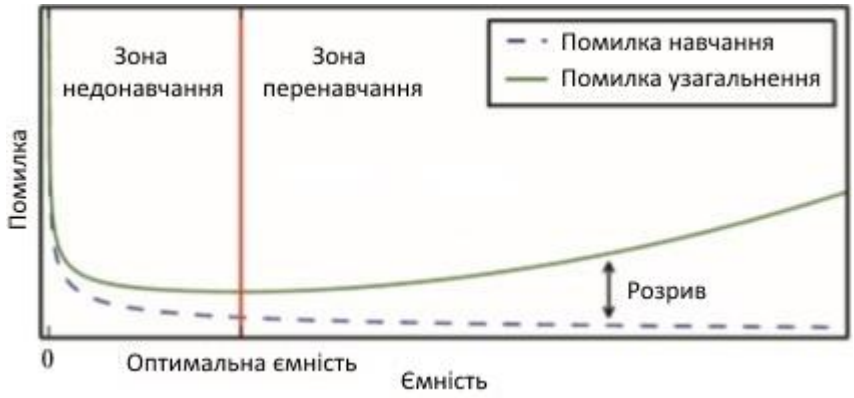



Рисунок 2.11 – Ілюстрація принципу вибору ємності моделі машинного навчання [22]

Для уникнення перенавчання нейронної мережі, хоч як це дивно не звучало, необхідно штучно заважати їй навчатися. Для цього застосовується штучне спотворення навчальних зображень, а також прийом, який називається Dropout. Він випадково «відключає» деякі зв’язки між нейронами. Його робота заснована на принципі, що недетермінований ансамбль простіших нейронних мереж у середньому працює краще, ніж одна складна мережа, що детермінує. Аугментація навчальних даних – прийом штучного спотворення навчальних даних. Докладніше опишемо його в практичній частині роботи.

Крім проблем, пов’язаних з машинним навчанням, ще потрібно вирішити проблеми, що виникають у найглибших нейронних мережах. Проблеми виникають у методі навчання нейронної мережі з використанням градієнтів. Існують дві проблеми: надто великі та надто малі градієнти.

## 2.11 Метод Віоли-Джонса

Метод Віоли-Джонса дозволяє знаходити особи на оброблюваному зображенні. Він був запропонований на початку 2000-х років Паулем Віолою та Майклом Джонсом. Алгоритм здатний знаходити на зображенні об’єкти різних класів, хоча основною метою його розробки було саме знаходження облич. Він відноситься до методів машинного навчання, а значить перед застосуванням потребує тренування. Сам процес навчання проходить довго, однак знаходить обличчя навчений детектор дуже швидко. До його плюсів можна віднести низький відсоток хибних спрацьовувань та кут, під яким із зображення безпомилково розпізнаються обличчя.

Даний метод базується на трьох основних принципах та прийомах:

- принцип ковзаючого вікна;
- інтегральне представлення зображення;
- ознаки Хаара.

Розглянемо принцип ковзного вікна та інтегральне представлення зображення.

Формально завдання виявлення ознак, характерних для обличчя виглядає так: нехай ми маємо фотографію, на якій зображені різні об’єкти, в тому числі й ті, які нас цікавлять (тобто обличчя). Вона є матрицею с параметрами w*h для чорно-білого зображення, і тензор w*h*3 для кольорового зображення. Кожен елемент матриці або тензора є значенням від 0 до 255, що означає інтенсивність чорного кольору для чорно-білої картинки, або інтенсивність одного з кольорів палітри "RGB" для кольорового зображення. Алгоритм у ході своєї роботи визначає риси особи, обличчя цілком і відзначає їх.

Для роботи із зображеннями у методі Віоли-Джонса необхідно перетворити дані на їх інтегральні уявлення. З їхньою допомогою можна швидко розраховувати сумарну яскравість довільного прямокутника на даному зображенні. Інтегральне уявлення зображення – це матриця, яка збігається за розмірами вихідним зображенням. У кожному елементі її зберігається сума інтенсивностей всіх пікселів, що знаходяться ліворуч і вище за цей елемент.

Кожен елемент матриці L (x, y) є сумою пікселів у прямокутнику від (0,0) до (x, y), тобто значення кожного пікселя (x,y) дорівнює сумі значень всіх пікселів ліворуч і вище за цей піксель (x,y). Розрахунок матриці L (x, y) можна здійснити за формулою:

$$L(x, y) = I(x, y) - L(x - 1, y - 1) + L(x, y - 1) + L(x - 1, y)$$

Завдяки формулі можна вирахувати суму значень пікселів у прямокутнику будь-якого розміру.

Ознакою прийнято вважати відображення f: X → $D_f$, де $D_f$ – безліч допустимих значень. Якщо задані ознаки f1, … , fn то вектор x = ($f_1$(x), … , $f_n$(x)) називають ознаковим описом об’єкта x ϵ X. Припустимо, що ознакові об’єкти дорівнюють самими об’єктами. При цьому множину X = $D_{f1} * … * D_{fn}$ називають ознаковим простором. Ознаки поділяються на наступні типи залежно від

множини $D_f$ : бінарна ознака, номінальна ознака, порядкова ознака, кількісна ознака. В оригінальному методі Віоли-Джонса використовуються прямокутні ознаки. Вони називаються примітивами Хаара, та зображені на рис. 2.12.

Значення ознаки дорівнює різниці суми пікселів у білому та чорному прямокутниках. Вони обчислюються за допомогою поняття інтегрального зображення, описаного вище.

У результаті, використовуючи деякі технічні та вищеописані прийоми, у методі Віоли-Джонса ознаки групуються за стадіями. Перші стадії містять невелику кількість ознак, у наступних стадіях їх кількість збільшується. Область, яка пройшла всі стадії, це і є обличчя.

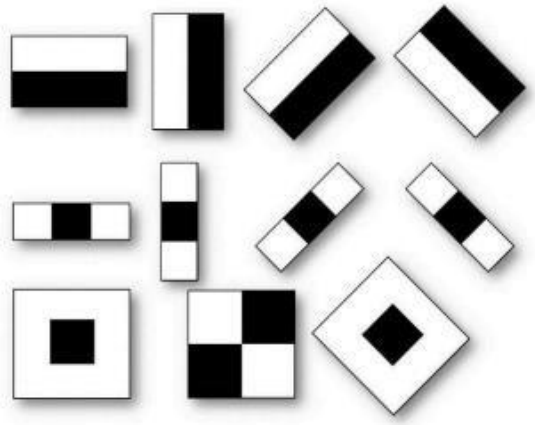


Рисунок 2.12 – Карти ознак Хаара [23]

Даний метод є провідним у пошуку об’єктів на зображенні в реальному часі, його основний недолік полягає в тому, що результат роботи значною мірою залежить від навчальної вибірки. Це пояснюється тим, що в якості вхідних даних виступає насичене зображення, яке чутливе до деякої освітленості.

## 2.12 Афінні перетворення

При завантаженні зображення його формат може бути різним залежно від багатьох факторів: різні розміри фотографій і самих осіб, їх положення і т.д.

Тому виникає необхідність у їх нормалізації та приведенні до однієї системи відліку. Для цього ми деформуємо всі зображення облич до єдиного розміру так, щоб ключові точки обличчя мали однакові координати. Давайте назвемо цю систему відліку кінцевою системою координат, а координати вихідних зображень – початковою системою координат.

Основна ідея перетворення полягає в тому, що визначаються 68 специфічних точок (міток), що є на кожній особі, – частина підборіддя, що виступає, зовнішній край кожного ока, внутрішній край кожної брови і т.п. Потім відбувається налаштування алгоритму навчання машини на пошук цих 68 специфічних точок на кожному обличчі, як показано на рис.2.13.

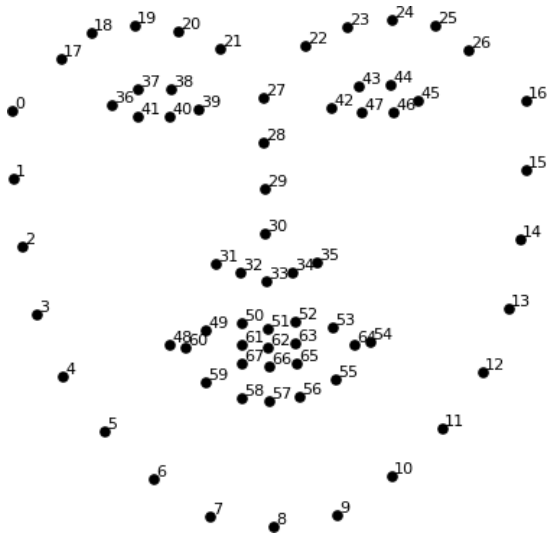


Рисунок 2.13 – Ілюстрація розташування 68 антропометричних точок обличчя людини [24]

Антропометричні точки дозволяють визначити місце розташування куточків очей та підборіддя на вихідних зображеннях. Це дозволяє обчислити перетворення подібності (обертання, перенесення, масштабування) і перевести точки з початкової координатної системи в кінцеву.

# 3 РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ ОБЛИЧ

## 3.1 Вибір навчальних даних

Для навчання нейронної мережі нам необхідно підібрати правильний набір даних (датасет) із зображеннями людей. В інтернеті є безліч датасетів для навчання алгоритмів розпізнавання осіб. Усі вони мають різні параметри доступності, різні умови фотографування, різні розміри зображень та інші особливості. Залежно від поставленого завдання виникають певні вимоги до датасету.

Для виконання практичної частини цієї кваліфікаційної роботи нам необхідно підібрати такий датасет, який містить зображення людей із реального життя. Переглянувши кілька різних датасетів, я зупинив свій вибір на датасеті Labelled Faces in the Wild (LFW), який містить 13 000 зображень, на яких показано обличчя 5700 осіб, з яких 1700 осіб мають по два і більше зображення. Зображення людей мають великі відмінності у позі, віці, освітленості, етнічній приналежності та професії (наприклад, актори, спортсмени, політики).

Для того, щоб підготувати дані до потрібного виду та розділити їх на тестову та навчальну множини був використаний наступний алгоритм:

- створити структуру для зберігання одного суб’єкта як картера з параметрами (суб’єкт, ім’я, посилання);
- створити необхідні директорії для тестової та навчальної множини;
- перемістити в папки файли, поділяючи на тестові та навчальні, де кожна директорія відповідає лише одному суб’єкту;
- винести в окрему директорію тестові дані.

Навчання нейромережі на повному наборі даних (датасеті) відбувається дуже довго і потрібні високі обчислювальні потужності. Щоб прискорити процес навчання, набір даних (датасет) був випадковим чином скорочено до 240 суб’єктів, по 4 зображення загалом кожного. Тестові дані становлять 4% від

усього набору. Важливо пам’ятати, що алгоритм повинен побачити тестові дані тільки на моменті тестування, тобто суб’єкти, які зображені на тестових даних не повинні зустрічатися в навчальному наборі даних. Таким чином, тестові дані – це 4% фотографій, які належать невеликій кількості суб’єктів і відсутні в навчальному наборі. Обробка даних розроблена за допомогою мови Python та бібліотеки для роботи з масивами NumPy. Програмна реалізація представлена у додатку В.

## 3.2 Вибір програмного забезпечення

Виконувати навчання нейронних мереж на звичайних процесорах дуже довго. Для прискорення процесу навчання я вирішив використати графічні процесори, використання яких дозволяють прискорити обчислення у 20-30 разів. Тому при виборі програмного забезпечення необхідно врахувати такі вимоги:

- навчання нейронної мережі на графічних процесорах, які дозволяють виконувати обчислювальні операції у десятки разів швидше, ніж на звичайних процесорах;
- зберігання результатів обчислення у вигляді обчислювального графа, що дозволяє повторно використовувати значення, пораховані на попередніх етапах;
- дозволяють запускати навчену модель на графічних процесорах.

Для аналізу було взято два фреймворки: Tensorflow і Theano. Після вивчення документації та відгуків розробників я вибрав фреймворк Tensorflow. Цей фреймворк розроблено та підтримується компанією Google, а також регулярно оптимізується та оновлюється. Завдяки використанню фреймворку Tensorflow, можливо ефективно реалізувати модель машинного навчання будь-якої складності. Однак особливості роботи з цим фреймворком підштовхнули мене до пошуку більш комфортного для програмування розширення, який також сумісний з Tensorflow і Python.

Мова розробки Python була обрана з метою мінімізувати витрати на вивчення синтаксису та особливостей мови для програмування нейронних мереж, та їх навчання. Python є дуже зручним та підходящим засобом для поставлених цілей. А наявність безлічі вже налаштованих бібліотек для роботи із зображеннями та машинного навчання остаточно визначило його як основний засіб розробки.

Найбільш зручним розширенням Python для машинного навчання, на мою думку, є бібліотека Keras. Завдяки цій бібліотеці дуже зручно проводити ініціалізацію та навчання нейронної мережі. Бібліотека Keras працює під управлінням фреймворку Tensorflow, тобто всі операції на програмному рівні переводяться в синтаксис Tensorflow, чим полегшує процес розробки людині, яка до цього не мала справу з цим фреймворком. [10]

### 3.3 Підготовка даних для навчання нейронної мережі

Для навчання нейронної мережі необхідно правильно підготувати навчальні дані (зображення). Спочатку на зображеннях потрібно обрізати все зайве і залишити лише обличчя людей. Це потрібно зробити для того, щоб нейронна мережа шукала закономірності лише на обличчях, а не на сторонніх об’єктах у зображенні. Програмна реалізація представлена у додатку Г.

У теоретичній частині були описані два методи виявлення осіб на фотографії: метод Віоли-Джонса і метод градієнтного спуску. Протестуємо кожний з них на мові Python. Метод Віоли-Джонса та метод градієнтного спуску ми будемо використовувати з уже навченими технологіями детекторів облич dlib та Opencv. У процесі тестування було проведено випробування на наборі зображень, які містять як групові, і портретні фото людей. Було відібрано близько 100 зображень. Кількість облич, знайдених обома алгоритмами, виявилася практично ідентичною.

Було отримано досить несподівані результати. При розпізнаванні портретних фото, обидва алгоритми показали себе на однаково високому рівні. При розпізнаванні групових фото, метод Віоли-Джонса видав чимало хибних результатів. Виходячи з отриманих результатів тестування, я вирішив використати метод спрямованих градієнтів. Далі потрібно було вибрати одну із двох реалізацій детектора осіб: від Opencv чи dlib. З суб'єктивних переваг, я вибрав детектор осіб від компанії dlib. Варто відзначити, що детектор від Opencv також працює добре. Програмна реалізація представлена у додатку Д. Результати на випадкових зображеннях із тестового набору продемонстровані на рис. 3.1 – 3.3

Рисунок 3.1 – Порівняння результатів роботи методу спрямованих градієнтів (ліворуч) та методу Віоли-Джонса (праворуч)

На фотографії із зображеними на ній 50 людьми алгоритми визначили по 48 осіб. Однак алгоритм Віоли-Джонса дав 2 помилкові спрацьовування.

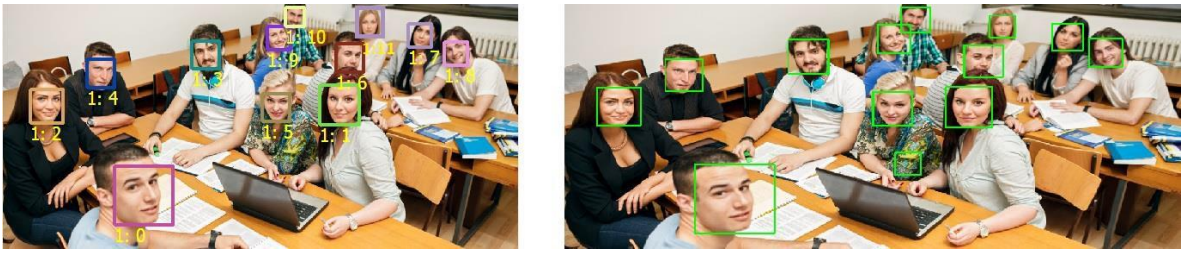


Рисунок 3.2 – Порівняння результатів роботи методу спрямованих градієнтів (ліворуч) та методу Віоли-Джонса (праворуч)

На фотографії із зображеними на ній 12 людьми алгоритми визначили по 12 осіб. Однак алгоритм Віоли-Джонса дав 1 хибне спрацювання. Воно обумовлено складною текстурою одягу суб’єкта.

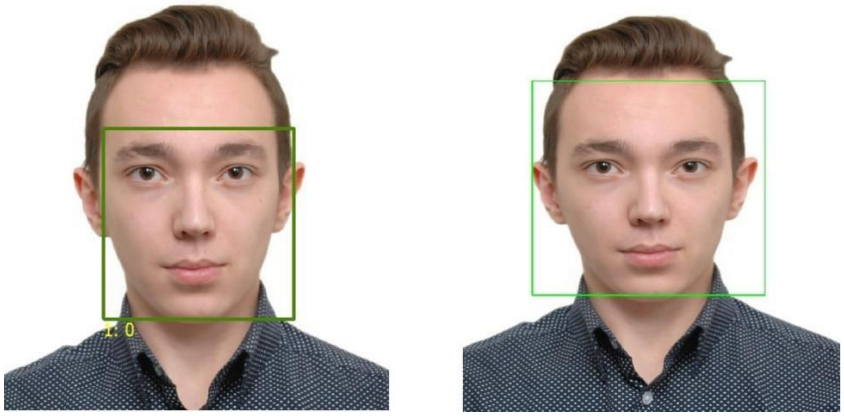


Рисунок 3.3 – Порівняння результатів роботи методу спрямованих градієнтів (ліворуч) та методу Віоли-Джонса (праворуч)

### 3.4 Завантаження даних для навчання

Ініціалізувати та навчати нейронну мережу я буду за допомогою бібліотеки Keras і мови Python. Ознайомившись з використанням даної бібліотекою та вивчивши її основні функції, з’ясувалося, що навчальні дані мають бути представлені як NumPy масиви. Виходячи з цих вимог, на всіх фотографіях з навчального та тестового наборів даних, необхідно виділити обличчя, провести їхню обробку та зберегти у вигляді NumPy масиву. Мітки фотографій, а саме приналежність одному з суб’єктів, зберігатимуться у вигляді іншого масиву, як і описано в документації до бібліотеки Keras. [25]

Нейронна мережа має фіксований вхід, тому всі зображення, що подаються до неї, повинні мати однаковий розмір. Інтуїтивно зрозуміло, що цю умову важко виконати навіть на навчальному наборі. Проблему ускладнює той факт, що навчання буде проводитись на обличчях, знайдених методом спрямованих градієнтів. Навіть за однакового розміру фотографій є дуже маленька

ймовірність, що всі обличчя виявляться однакового розміру. Для вирішення цієї проблеми потрібно використовувати операцію масштабування зображення. Існують різні способи здійснення цієї операції, наприклад:

- якщо зображення має менший розмір, ніж потрібно, тоді додаємо рамки до потрібного розміру. Згорткова нейронна мережа сама в процесі навчання навчиться виділяти зображення з версії з доповненою рамкою;
- розтягування/стискання зображення із збереженням пропорцій сторін та відкидання тієї частини зображення, що виходить за рамки вхідних розмірів нейронної мережі тощо.

Вимоги до зміни розмірів зображення полягає в тому, щоб ключові антропометричні точки на зображеннях завжди знаходилися в тому самому місці. [25]

Алгоритм, яким проходить процес обробки зображень:

– виділити обличчя на фотографії та знайти антропометричні точки;
– побудувати афінне перетворення, яке переводять обличчя у потрібний формат;
– застосувати афінне перетворення до зображення, обрізати чи розтягнути його.

Для пошуку ключових точок на зображенні використовувалася навчена модель dlib. Ця модель реалізує алгоритм виявлення антропометричних точок обличчя, описаний у теоретичній частині.

Для афінного перетворення використовуються 39, 42 та 57 антропометрична точка обличчя (рис. 3.4). Простіше кажучи, для спотворення обличчя зі збереженням його вихідних пропорцій необхідно знати лише розташування країв очей та нижньої частини губи.

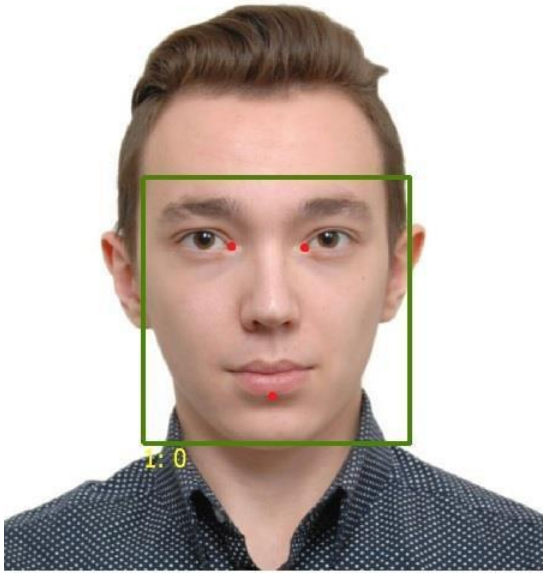



Рисунок 3.4 – Ключові точки (39, 42 і 57) на зображенні обличчя людини. Вони позначені червоними крапками

Застосування афінних перетворень реалізовано за допомогою методу wrap та бібліотеки scipy-image.

Використовується функція compute_accuracy для визначення точності передбачення моделі на основі передбачень зображень у відповідних міток класів з y_test.

Ще потрібно нормалізувати вхідні дані. Це необхідно для підвищення швидкості та ймовірності збіжності алгоритму оптимізації – методу градієнтного спуску. Це пов’язано зі значеннями градієнтів, а саме з проблемою їх надто великих чи малих значень. З кожного пікселя вхідного зображення необхідно відняти середній піксель і по всьому набору даних.

У результаті навчальні дані являтимуть собою numpy-масиви з попередньо обробленими та виділеними особами людей з навчального набору. На цьому етапі виникає проблема щодо розміру цих масивів. Необхідно розділити масиви приблизно по 4000 зображень, інакше вони просто не помістяться в оперативну пам’ять комп’ютера.

## 3.5 Помилки у задачі ідентифікації

Стандартне поняття помилки мережі як ставлення правильно класифікованих об’єктів до всього обсягу об’єктів у разі розробки сервісу ідентифікації не підходить. Це можна пояснити тим, що для нас важливе значення помилки першого роду та другого. Простими словами, не можна ставити в один ряд помилку щодо хибної ідентифікації людини, внаслідок якої людина «обходить» систему авторизації, насправді не маючи доступу. І помилку з хибного недопуску людини, яка є в базі користувачів. У математичній статистиці такі помилки мають назву першого та другого роду. Помилка першого роду – це можливість прийняти гіпотезу в припущенні, що вона неправильна. Помилка другого роду – можливість відкинути гіпотезу, припускаючи, що вона вірна.

У біометрії є поняття – FAR (False Acceptance Rate) та FRR (False Rejection Rate). Завдання ідентифікації людини за антропометричними ознаками його обличчя відноситься до сфери біометрії. FAR – це можливість помилкового збігу антропометричних характеристик двох людей. FRR – ймовірність відмови доступу людині, яка має допуск. Чим менше значення FRR за однакових значень FAR, тим точніше працює система.

Обидва показники залежать від деякого параметра, який визначає певний рівень «схожості» d, починаючи від якого прийнято вважати, що це дві однакові або різні людини. Цей показник визначається залежно від сфери застосування сервісу розпізнавання людини. Допустимо якщо сервіс застосовується на прохідній великого стратегічного підприємства, то важливіше підібрати якомога мінімальній FAR при допустимому FRR. А якщо це автоматична система позначки ваших друзів на фотографії в соціальних мережах, можна пожертвувати FAR, для збільшення FRR.

Існує показник, що дозволяє узагальнити та замінити FFR та FAR, і він називається EER (Equal Error Rate). ERR є значенням, у якому ці показники рівні, тобто він показує який відсоток помилок першого та другого роду допускає

сервіс. Графічна інтерпретація вищезазначених показників точності моделі ідентифікації зображена на рис. 3.5.

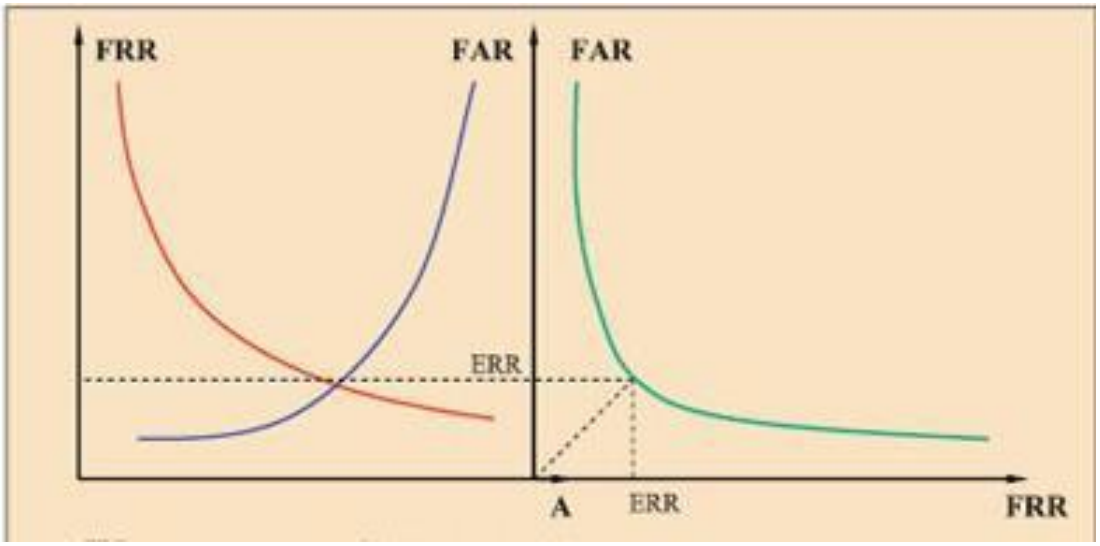



Рисунок 3.5 – Графіки FRR, FAR та ERR

На графіці ліворуч (рис. 3.5) на осі ординат відкладено ймовірність помилки. На осі абсцис зазначено значення параметра, від якого залежить функція FRR, FAR. На графіці праворуч на осі ординат відкладено можливість помилки, на осі абсцис відкладено значення функції FRR.

## 3.7 Архітектура нейронної мережі для отримання bottleneck ознак

При виборі архітектури були сформовані вимоги як достатня ємність і в той же час невелика кількість параметрів, що навчаються. Одними із найсучасніших архітектур є Residuals Networks. Основна їх особливість полягає в тому, що поглиблення мережі відбувається в основному завширшки, а не в глибину. Мережі такої архітектури працюють як ансамбль простіших мереж, що дає їм перевагу по відношенню до звичайних нейронних мереж. Ще однією особливістю є відмова від повнозв’язкових шарів на виході мережі. Їм на заміну прийшов Average Pooling. Завдяки такому прийому значно скорочується

кількість параметрів навчання та з’являється інваріантність до розташування об’єкта на вхідному зображенні.

На сьогоднішній день існує компроміс між глибиною та паралельністю в згорткових нейронних мережах, який за рахунок паралельного використання фільтрів різних розмірів збільшує сприйнятливе поле нейронної мережі, має достатню глибину, і в якій відсутні повно зв’язні шари, що значно збільшують обчислювальну складність мережі.

Ще однією особливістю є паралельне використання пулінгу та згорткових шарів, яке дозволяє в середньому зберігати інваріантність та чіткість патернів, на які нейронна мережа реагує на кожному шарі. На виході мережі використовується функція Softmax, що дає оцінку ймовірності приналежності вхідного об’єкта до одного з класів. Програмна реалізація представлена у додатку Е

## 3.7 Реалізація функції втрат на основі триплетів

Програмна реалізація навчання нейронної мережі полягала в наступному:

– отримати числові характеристики антропометричних ознак людини;
– додати повнозв’язковий шар із лінійною активацією;
– запрограмувати метод розрахунку виходу нейронної мережі під час навчання;
– запрограмувати цільову функцію.

Для реалізації першого пункту знадобилося запрограмувати метод, який міг би витягувати вектор числових значень їх прихованих шарів нейронної мережі. Для цього використовується Keras Backend, за допомогою якого можна «висмикнути» шматок нейронної мережі та проводити з ним подальші операції. У нашому випадку необхідним «шматком» є нейронна мережа з розділу 2.6, починаючи з входу до передостаннього шару.

Далі додається повнозв’язний шар з кількістю нейронів, що дорівнює

кількості ознак. Питання щодо кількості нейронів у доданому повнозв'язному шарі не дуже однозначне. Якщо кількість ознак є надлишковою, можна застосувати метод основних компонентів і перейти до меншої розмірності простору з досить малою втратою інформації. Він вплине лише на здатність верифікувати залежності, а оскільки до цього без детального аналізу не було ясно, на які патерни реагує нейронна мережа, то це не принесе незручності. Надмірністю ознак можна вважати залежності між один одним або недостатню статистичну значимість.

Процес навчання нейромережі проходить за допомогою трьох зображень, з яких витягуються низькорозмірні ознаки bottleneck і множаться на матрицю вагових коефіцієнтів пункту 2 даного списку. Матриця характеризує зв'язок між нейронами різних верств. Необхідне для виконання нерівність виглядає так:

$$a * p - a * n > 0 \rightarrow a * (p - n) > 0$$

де $a, p$ − вектора числових значень характеристик одного суб'єкта,

$n$ − вектор числових значень характеристик іншого суб'єкта

### 3.8 Процес навчання та тестування

Процес навчання поділяється на дві частини. Перша частина – це навчання класифікатора на закритій множині з кількістю суб'єктів, що дорівнює 240. Для цього ініціалізується нейронна мережа з обраною та запрограмованою архітектурою. Генеруються міні-пакети з 16 зображень, за допомогою яких навчається нейронна мережа.

Перед формуванням міні-пакетів використовується аугментація даних. Зображення випадково повертаються на кут до 20 градусів. Розтягуються або стискаються по горизонталі чи вертикалі до 20% від вихідних пропорцій. Збільшуються із збереженням пропорцій до 10% від вихідного розміру.

Від навчальних даних відокремлюється 6% суб’єктів, як у випадку з тестовим набором для проміжної перевірки точності моделі і контролю збіжності. Така перевірка відбувається раз на епоху. Епоха – це один перегляд усіх прикладів навчальної вибірки, з одночасною корекцією ваги мережі (на цих прикладах, залежно від правильності їх вирішення мережею). Для досягнення гарної точності проводиться 512 епох. Вагові коефіцієнти, за допомогою яких було отримано найкращий результат, зберігаються.

Таким чином підготовлено основу для продовження навчання нейронної мережі за допомогою триплетів. Проведено початкове рознесення об’єктів у просторі ознак. Однак воно все ще не дозволяє з достатньою упевненістю говорити про те, які ознаки належать одному суб’єкту, а які іншому. Відстань між суб’єктами у просторі ознак у кращому разі недостатньо великі, а в гіршому випадку суб’єкти перетинаються. Для того, щоб це виправити, буде виконуватися донавчання нейромережі триплетами. Програмна реалізація представлена в додатку Ж. Процес навчання навіть урізаним дата-сетом займає близько доби.

Програмна реалізація донавчання триплетам є досить нетривіальною задачею. Якщо у разі навчання класифікатора використовувалися стандартні шари, генератори даних, функції активацій, то для триплетів необхідно запрограмувати свою функцію втрат і підбір самих триплетів.

У теоретичній частині був опущений один момент, який стосується вибору триплетів. Якщо ж зображення однієї й тієї людини не так критично вибирати випадковим чином, то для вибору негативного прикладу такий принцип відбору не підходить. Негативний приклад повинен підбиратися як найближчий по всій навчальній множині. Але його пошук допускається за деякою досить великою, випадково сформованою множиною негативних прикладів з усієї множини.

Для генерації триплетів використовувався такий алгоритм:

– перетворити масив міток всіх суб’єктів навчальної множини в numpy-масив;
– отримати список унікальних тег суб’єктів;

- створити множини, які зберігатимуть індекси для зображень однієї й тієї ж людини;
- здійснити перебір унікальних значень міток суб’єктів пункту 2;
- отримати індекси елементів з вихідної множини об’єктів, які мають відповідну поточному ітерабельному значенню мітку, тобто належать одному суб’єкту;
- отримати всі поєднання з безлічі індексів пункту 6 по два. По черзі заповнити множини з пунктів 3 та 4 відповідними значеннями;
- за допомогою навченого класифікатора отримати числові значення характеристик об’єктів навчальної множини;
- підрахувати кількість парасполучень;
- випадковим чином згенерувати підмножини паросполучення.

Далі опишемо процес донавчання. Для донавчання не потрібно перенавчати глибоку структуру, оскільки зі своїм завданням вона вже справляється. Для донавчання створюється звичайний двошаровий персептрон з лінійною активацією нейронів, який навчається за допомогою векторів числових характеристик, які витягують із передостаннього шару класифікатора. Даний персептрон має три входи та три виходи. Для виходів обчислюється негативна логарифмічна правдоподібність. Програмна реалізація представлена у додатку И.

Навчання проходить міні-пакетами по 4 об’єкти, які містять по 512 помічених триплетів. Кількість ітерацій навчання дорівнює 5000. Час, витрачений на навчання, склав близько 2 години. В середньому по 1-2 секунди на обробку міні-набору, що складається з 512 зображень. Паралельно обробляється по 4 таких міні-набори. У середньому на обробку одного триплету потрібно 0,02 секунди. В процесі навчання ваги моделі змінюються для зменшення втрат та покращення продуктивності на тестовій вибірці. Шкала прогресу навчання представлена на рис. 3.6.

Тестування навченої нейронної мережі є важливим етапом створення програми.

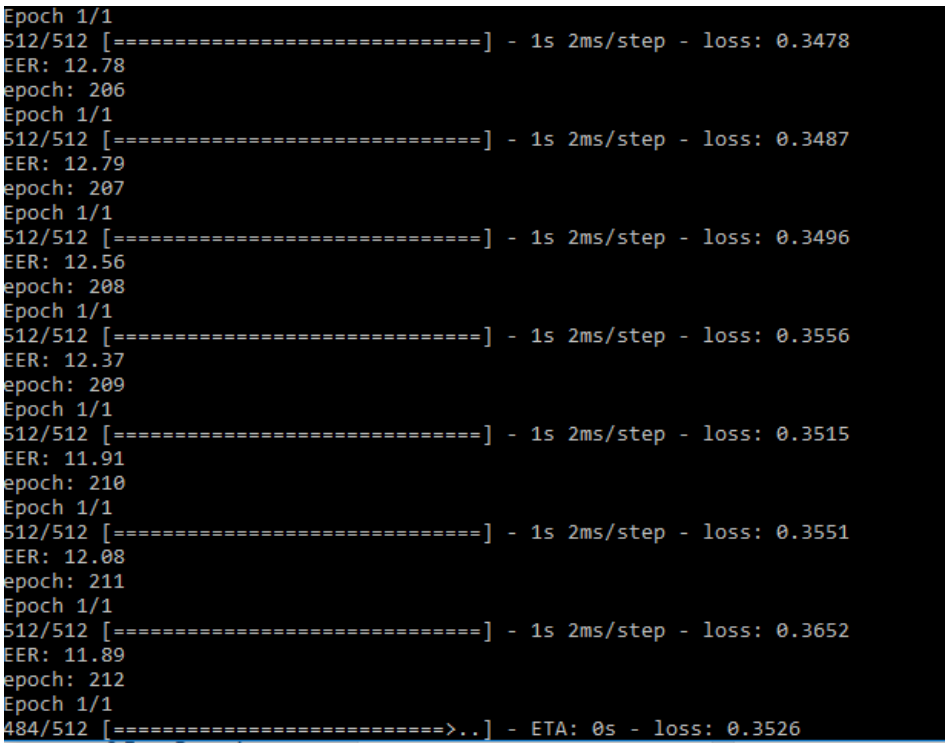



Рисунок 3.6 – Процес навчання нейронної мережі триплетами

Точність підсумкової навченої нейронної мережі на цих дата-сетах склала 93%, тобто. ERR = 6.98%. Побачити це можна на рис.3.7. Програмна реалізація у додатку К.

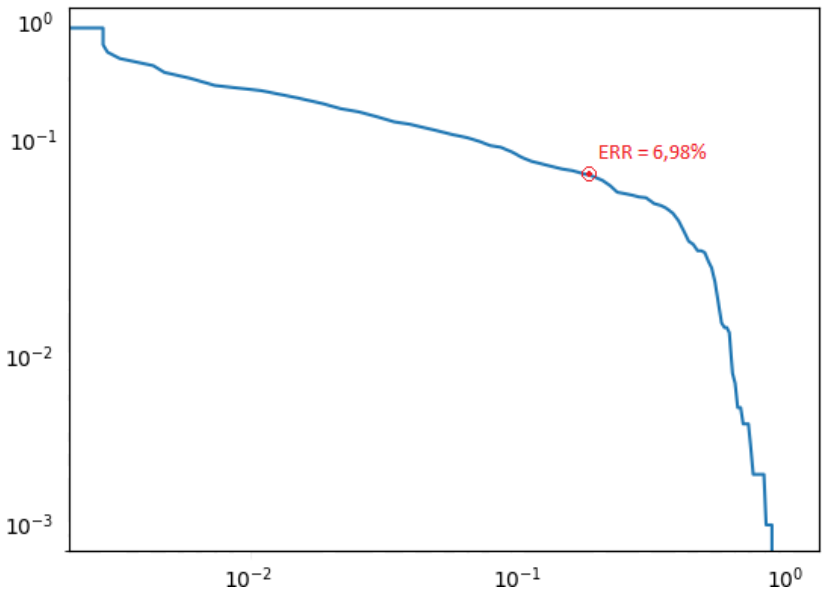



Рисунок 3.7 – Залежність FAR від FRR у логарифмічному масштабі

# ВИСНОВКИ

У ході виконання кваліфікаційної роботи було виконано такі завдання:

– вивчення основних методів та структури нейронних мереж, що застосовуються для розпізнавання облич на зображеннях;
– формування вибірки, підготовка даних для навчання нейронної мережі;
– побудова структури згорткових нейронних мереж;
– навчання згорткової нейронної мережі;
– тестування розробленої програми.

У висновку кваліфікаційної роботи з розпізнавання осіб за допомогою згорткових нейронних мереж можна зробити висновок, що використання нейронних мереж до виконання поставленої завдання є ефективним підходом.

Розроблена модель для розпізнавання облич за допомогою нейронних мереж показала високу точність та ефективність у роботі.

Розроблена програма може бути використана в системах безпеки, наприклад, системах контролю доступу на великих підприємствах або в правоохоронних органах для пошуку злочинців.

Функціональність програми можна покращити, додавши функцію розпізнавання облич на відеопотоці. Це завдання може бути розглянуто та вирішено у дипломній роботі магістра.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Що таке комп’ютерний зір. Машинне навчання. URL : https://forklog.com/cryptorium/ai/chto-takoe-kompyuternoe-zrenie-mashinnoe-obuchenie (дата звернення: 02.04.2024).

2. Neural Networks and ‘Machines That Dream. URL : https://kushankpatel7.medium.com/google-neural-networks-and-machines-that-dream-9183a4d5225 (дата звернення: 06.04.2024).

3. Customize the Data and Produce the Drastic Groundwater Vulnerability Layer. URL : https://www.e-education.psu.edu/geog487/node/308 (дата звернення: 07.04.2024).

4. Що таке машинне навчання? Як працює машинне навчання. URL : https://aws.amazon.com/ru/what-is/machine-learning/ (дата звернення: 06.04.2024).

5. Artificial Intelligence. A Journey into the Future. URL : https://medium.com/@info_52759/artificial-intelligence-a-journey-into-the-future-0048e62d10cd (дата звернення: 08.04.2024).

6. Бодянский Е. В. Штучні нейронні мережі: архітектури, навчання, застосування. Харків : Телетех, 2004. 369 с.

7. Нейронні мережі: види, принцип роботи та сфери застосування. URL : https://corvet46.ru/izuchaem-neironnye-seti-za-chetyre-shaga-neironnye-seti-vidy-princip-raboty/ (дата звернення: 10.04.2024).

8. Cost–Benefit Prediction of Asset Management Actions on Water Distribution Networks. URL : https://www.researchgate.net/figure/The-architecture-of-the-proposed-Deep-Learning-model_fig2_365008907 (дата звернення: 11.04.2024).

9. Функції активації нейромережі: сигмоїда, лінійна, ступінчаста, ReLu, Tahn. URL : https://neurohive.io/ru/osnovy-data-science/activation-functions/ (дата звернення: 12.04.2024).

10. Реализация слоев в нейронных сетях. URL : https://www.slideshare.net/grigorysapunov/deep-learning-75853648 (дата звернення: 13.04.2024).

11. Що таке глибоке навчання? Принципи роботи глибокого навчання. URL : https://www.oracle.com/cis/artificial-intelligence/machine-learning/what-is-deep-learning/ (дата звернення: 16.04.2024).

12. Згорткові нейромережі: що це і для чого вони потрібні. URL : https://forklog.com/cryptorium/ai/svertochnye-nejroseti-chto-eto-i-dlya-chego-oni-nuzhny (дата звернення: 18.04.2024).

13. Deep Learning, искусственный интеллект и машинное обучение. URL : https://javarush.com/groups/posts/309-deep-learning-iskusstvennihy-intellekt-i-mashinnoe-obuchenie-dlja (дата звернення: 18.04.2024).

14. Classification of Lombok Songket and Sasambo Batik Motifs Using. URL : https://www.researchgate.net/figure/Convolution-process-on CNN_fig2_379444069 (дата звернення: 19.04.2024).

15. Руденко О.Г. Штучні нейронні мережі: Навчальний посібник. Київ : Компанія СМІТ, 2006. 404 с.

16. Вступ в Deep Learning. Convolutional Neural Network. URL : https://www.slideshare.net/grigorysapunov/deep-learning (дата звернення: 20.04.2024).

17. Функція вартості: формула функції вартості та способи її отримання. URL : https://fastercapital.com/ru/content/%D0% %BD1%86%D0%B8%D1%8F (дата звернення: 21.04.2024).

18. Pedestrian reidentification based on multiscale convolution. URL : https://link.springer.com/article/10.1007/s11760-021-02125-8-02125-8 (дата звернення: 22.04.2024).

19. Метод зворотного поширення помилки: математика, приклади, код. URL : https://neurohive.io/ru/osnovy-data-science/obratnoe-rasprostranenie/ (дата звернення: 23.04.2024).

20. Градієнтний спуск: все, що треба знати. Що таке градієнтний спуск. URL : https://neurohive.io/ru/osnovy-data-science/gradient-descent/ (дата звернення: 24.04.2024).

21. Machine Learning: In regularization, why do we always seek to minimize the norm of the weights. URL : https://www.quora.com/Machine-Learning-In-regularization-why-do-we-always-seek-to-minimize-the-norm-of-the-weights (дата звернення: 25.04.2024).

22. Нейронні мережі. Принципи роботи глибокого навчання. URL : https://se.moevm.info/lib/exe/fetch.php/courses:artificial_neural_netwok (дата звернення: 26.04.2024).

23. Популярні методи виявлення та розпізнавання обличчя. URL : https://macroscop.com/o-kompanii/blog/populyarnye-metody-obnaruzheniya-i-raspoznavaniya-lits (дата звернення: 26.04.2024).

24. Emotion Analysis using Facial Expressions in Video. URL : https://www.researchgate.net/figure/Facial-Landmarks-10_fig1_352088928 (дата звернення: 27.04.2024).

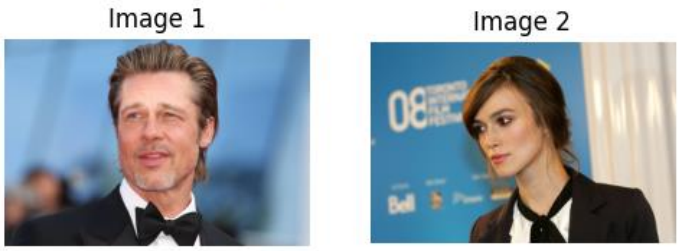
25. Паскарюк Д. О. Розпізнавання образів за допомогою нейронних мереж. Харків : ХНУРЕ, 2020. 77 с.

# ДОДАТОК А

## Тестування роботи программи

```
image_path1 = 'bred.jpg'
image_path2 = 'kira_1.jpg'
plot_images(image_path1, image_path2)
distance = extract_faces_and_compute_distance(model, image_path1, image_path2, distance_metric='euclidean')
print(f"Евклидово расстояние между изображениями лиц: {distance}")
```

```
Евклидово расстояние между изображениями лиц: 2.3561477661132812
```




```
[ ] image_path1 = 'kira.jpg'
    image_path2 = 'kira_1.jpg'
    plot_images(image_path1, image_path2)
    distance = extract_faces_and_compute_distance(model, image_path1, image_path2, distance_metric='euclidean')
    print(f"Евклидово расстояние между изображениями лиц: {distance}")
```

```
Евклидово расстояние между изображениями лиц: 1.2601863145828247
```

# ДОДАТОК Б

## Файл підготовки та обробки даних

```
import os
import os.path
import random
import math
import itertools
import shutil
import numpy as np
from collections import namedtuple
from skimage import io
from preprocessing import FaceDetector, FaceAligner, clip_to_range
from tqdm import tqdm
from itertools import repeat
FORMATS = {'.jpg', '.jpeg', '.png'}
DATA_DIR = './data/'
TRAIN_DIR = DATA_DIR + 'train/'
TEST_DIR = DATA_DIR + 'test/'
DEV_DIR = DATA_DIR + 'dev/'
PROTOCOLS_DIR = './data/'
BASE_DIR = 'C:/Users/dimah/OneDrive/Рабочий стол/'
VGG_DIR = BASE_DIR + 'train_temp/'
DEV_ RATIO = 0.04
TEST_RATIO = 0.06
DO_NOT_COPY = False
#Создание кортежа Entry с параметрами субъект, имя, патч
Entry = namedtuple('Entry', ['subject', 'name', 'path'])
#Считываем план базы данных
def grab_db_plain(path, divisor):
    res = []
    #Проходим переменной файл по папке с адресом path
    for file in os.listdir(path):
        #Создаем адресацию имя_папки/файл
        file_path = os.path.join(path, file)
        #Вытаскиваем формат итерабельного файла
        ext = os.path.splitext(file)[1]
        #Если итерабельный объект файл и его формат графический
        if os.path.isfile(file_path) and ext in FORMATS:
            #Определяем то, кому принадлежит файл и его имя
            subject, name = file.split(divisor)
            #Записываем в результат в рез
            res.append(Entry(path + subject, name, file_path))
    return res
```

```
  #Считываем папки базы данных
  def grab_db_folders(path):
    res = []

  #Проходимся по файлам дирнектории патч
  for dir in os.listdir(path):
    #Получаем название итерабельного файла
    dir_path = os.path.join(path, dir)
    #Если итерабельный файл папка
    if os.path.isdir(dir_path):
      #Проходимся по файлам папки
      for file in os.listdir(dir_path):
        #Получаем название итерабельного файла
        file_path = os.path.join(dir_path, file)
        #Получаем формат файла
        ext = os.path.splitext(file)[1]
        #Если итерабельный объект файл и графического формата
        if os.path.isfile(file_path) and ext in FORMATS:
          #Добавляем этот файл в результат
          res.append(Entry(path + dir, file, file_path))
  return res
#Парсим входные параметры в список
def get_entry_subjects(xs):
  return list(set(map(lambda e: e.subject, xs)))

#Получаем входные параметры
def get_subjects(entries):
  #Получаем список входных параметров
  subjects = get_entry_subjects(entries)
  #Получаем длину списка параметров
  n_subjects = len(subjects)
  #Получаем количесвто dev объектов
  n_dev_subjects = max(1, math.ceil(n_subjects * DEV_RATIO))
  #Перемешиваем входные параметры
  random.shuffle(subjects)
  #Возвращаем кортеж из двух одинаковых массивов, в которых содержится по 4% входных данных перемешанные
  return subjects[:n_dev_subjects], subjects[n_dev_subjects:]
#Копируем файлы в папку дест
def copy_files(files, dest_dir):
  if DO_NOT_COPY:
    return
  #Создаем папку, если ее нет
  if not os.path.exists(dest_dir):
    os.makedirs(dest_dir)
  h = 0
  for file in files:
    ext = os.path.splitext(file)[1]
    #Копируем файл в папку дест с номером h
    shutil.copy(file, os.path.join(dest_dir, str(h) + ext))
    h += 1
```

```
def main():
    for dir in {DATA_DIR, TRAIN_DIR, TEST_DIR, DEV_DIR, PROTOCOLS_DIR}:
        if not os.path.exists(dir):

            print('Creating {}'.format(dir))
            os.makedirs(dir)
    entries = [
        #('fei', grab_db_plain(FEI_DIR, '-')),
        #('caltech_faces', grab_db_folders(CAL_DIR)),
        ('vgg', grab_db_folders(VGG_DIR))
        # ('lfw2', grab_db_folders(LFW2_DIR))
    ]
    #Объект, который хранит все фотографии из обучающего набора
    all_entries = list(itertools.chain.from_iterable((map(lambda p: p[1], entries))))
    #Объект, который хранит количество субъектов из обучающего набора
    all_subjects = get_entry_subjects(all_entries)
    #Объект, который хранит количество фогорафий
    n_files_total = len(all_entries)
    #Объект, который хранит количество субъектов
    n_subjects_total = len(all_subjects)
    print('-' * 10)
    print('Taking for development set {:.2f}% of subjects'.format(DEV_RATIO * 100))
    #Создаем массив для субъектов тестового множетсва
    subjects_dev = []
    #Создаем массив для остальных субъектов
    subjects = []
    print('-' * 10)
    for name, e in entries:
        #Определяем количество объектов в
        n_e_files = len(e)
        #Подсчет количества изображений в каждом из обучающих множеств
        print('Total files in "{}" set: {}'.format(name, n_e_files))
        #В тестовое множество 4% субъектов, все остальное в обучающее
        subjects_dev_e, subjects_e = get_subjects(e)
        subjects_dev += subjects_dev_e
        subjects += subjects_e
    print('-' * 10)
    print('Total files: {}'.format(n_files_total))
    print('Total subjects: {}'.format(n_subjects_total))
    print('-' * 10)
    n_subjects_dev = len(subjects_dev)
    n_subjects = len(subjects)

    print('Number of subjects for development set: {}'.format(n_subjects_dev))
    print('Number of subjects for train/test set: {}'.format(n_subjects))

    dev_files = []
    protocol_data = []
    for subj in subjects_dev:
        subj_entries = list(map(lambda e: e.path, filter(lambda e: e.subject == subj, all_entries)))
        n_subjects_entries = len(subj_entries)
        dev_files.extend(subj_entries)
```

```
#Запоминаем сколько фотографий есть у каждого субъекта
    protocol_data.append(n_subjects_entries)

n_dev_files = sum(protocol_data)
protocol = np.zeros((n_dev_files, n_dev_files), dtype=np.bool)

k = 0
for i in protocol_data:
    protocol[k:k + i, k:k + i] = 1
    k += i
print('-' * 10)
print('Total dev files: {}'.format(n_dev_files))
copy_files(dev_files, DEV_DIR)
np.save(PROTOCOLS_DIR + 'dev_protocol', protocol)
n_test_files = 0
h = 0
for subj in subjects:
    subj_name = 'subject_' + str(h)
    h += 1
    subj_entries = list(map(lambda e: e.path, filter(lambda e: e.subject == subj, all_entries)))
    n_subj_entries = len(subj_entries)
    random.shuffle(subj_entries)
    for_test = 0
    if n_subj_entries > 1:
        for_test = max(1, math.ceil(n_subj_entries * TEST_RATIO))
    n_test_files += for_test
    entries_test, entries_train = subj_entries[:for_test], subj_entries[for_test:]
    subj_train_dir = os.path.join(TRAIN_DIR, subj_name)
    subj_test_dir = os.path.join(TEST_DIR, subj_name)
    copy_files(entries_train, subj_train_dir)
    copy_files(entries_test, subj_test_dir)
print('Test files: {}'.format(n_test_files))
print('Train files: {}'.format(n_files_total - k - n_test_files))
print('Done!')
fd = FaceDetector()
fa = FaceAligner('../model/shape_predictor_68_face_landmarks.dat', '../model/face_template.npy')
IMAGE_FORMATS = {'.jpg', '.jpeg', '.png'}
IMSIZE = 289
BORDER = 5
train, test, dev = list_data('./data')
###
print('Loading train files...')
train_x, train_y = load_data(train, imsize=IMSIZE, border=BORDER, not_found_policy='throw_away')
del train
mean = train_x.mean(axis=0)
stddev = train_x.std(axis=0)
np.save('../model/mean', mean)
np.save('../model/stddev', stddev)
train_x -= mean
train  x /= stddev
```

```
            np.save('./data/train_x', train_x)
            np.save('./data/train_y', train_y)
            ###
            del train_x
            ###
            print('Loading test files...')
            test_x, test_y = load_data(test, imsize=IMSIZE, border=BORDER, not_found_policy='throw_away',
available_subjects=train_y)
            del test, train_y
            test_x  -=  mean
            test_x /= stddev
            np.save('./data/test_x', test_x)
            np.save('./data/test_y', test_y)
            ###
            del test_x, test_y
            ###
            print('Loading dev files...')
            dev_x, _ = load_data(dev, imsize=IMSIZE, border=BORDER, not_found_policy='replace_black')
            del dev
            dev_x -= mean
            dev_x /= stddev
            np.save('./data/dev_x', dev_x)
            ###
        if name_== '_main_':
            main()
```

# ДОДАТОК В

## Програмна реалізація методів обробки зображень за допомогою dlib.

```
import dlib

import numpy as np
import skimage.transform as tr

from enum import Enum

class FaceDetectorException (Exception):
  pass

class FaceDetector:
  def init (self):
    self.detector = dlib.get_frontal_face_detector()

  def detect  faces(self,
            image, *,
            upscale_factor=1,
            greater_than=None,
            get_top=None):
    try:
      face_rects = list(self.detector(image, upscale_factor))
    except Exception as e:
      raise FaceDetectorException(e.args)

    if greater  than is not None:
      face_rects = list(filter(lambda r:
                r.height() > greater_than and r.width() > greater_than,
                face_rects))

    face_rects.sort(key=lambda r: r.width() * r.height(), reverse=True)

    if get_top is not None:
      face_rects = face_rects[:get_top]

    return face_rects

class FaceAlignMask(Enum):
  INNER_EYES_AND_BOTTOM_LIP = [39, 42, 57]
  OUTER  EYES  AND  NOSE = [36, 45, 33]

class FaceAligner:
```

```
def init (self,
         dlib_predictor_path,
         face_template_path):
   self.predictor = dlib.shape_predictor(dlib_predictor_path)
   self.face_template = np.load(face_template_path)
def get  landmarks(self,
              image,
              face_rect):
   points = self.predictor(image, face_rect)
   return np.array(list(map(lambda p: [p.x, p.y], points.parts())))

def align_face(self,
            image,
            face_rect, *,
            dim=96,
            border=0,
            mask=FaceAlignMask.INNER_EYES_AND_BOTTOM_LIP):
   mask = np.array(mask.value)

   landmarks = self.get_landmarks(image, face_rect)
   proper_landmarks = border + dim * self.face_template[mask]
   A = np.hstack([landmarks[mask], np.ones((3, 1))]).astype(np.float64)
   B = np.hstack([proper_landmarks, np.ones((3, 1))]).astype(np.float64)
   T = np.linalg.solve(A, B).T

   wrapped = tr.warp(image,
                tr.AffineTransform(T).inverse,
                output_shape=(dim + 2 * border, dim + 2 * border),
                order=3,
                mode='constant',
                cval=0,
                clip=True,
                preserve_range=True)

   return wrapped

   def align_faces(self,
                image,
                face_rects,
                *args,
                **kwargs):
      result = []

      for rect in face_rects:
         result.append(self.align_face(image, rect, *args, **kwargs))

      return result
def clip_to_range(img): return img / 255.0
```

# ДОДАТОК Г

## Програмна реалізація архітектури нейронної мережі для класифікатора

```
from functools import partial
from keras.models import Model
from keras.layers import Activation
from keras.layers import BatchNormalization
from keras.layers import Concatenate
from keras.layers import Conv2D
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import GlobalAveragePooling2D
from keras.layers import Input
from keras.layers import Lambda
from keras.layers import MaxPooling2D
from keras.layers import add
from keras import backend as K
def scaling(x, scale):
    return x * scale
def conv2d_bn(x,filters,kernel_size,strides=1,padding='same',activation='relu',
 use_bias=False, name=None):
    x = Conv2D(filters,kernel_size,strides=strides,padding=padding,           use_bias=use_bias,name=name)(x)
    if not use_bias:
        bn_axis = 1 if K.image_data_format() == 'channels_first' else 3
        bn_name = _generate_layer_name('BatchNorm', prefix=name)
        x = BatchNormalization(axis=bn_axis, momentum=0.995, epsilon=0.001,
                               scale=False, name=bn_name)(x)
    if activation is not None:
        ac_name = _generate_layer_name('Activation', prefix=name)
        x = Activation(activation, name=ac_name)(x)
    return x

def _generate_layer_name(name, branch_idx=None, prefix=None):
    if prefix is None:
        return None
    if branch_idx is None:
        return '_'.join((prefix, name))
    return '_'.join((prefix, 'Branch', str(branch_idx), name))
```

```
def _inception_resnet_block(x, scale, block_type, block_idx, activation='relu'):
    channel_axis = 1 if K.image_data_format() == 'channels_first' else 3
    if block_idx is None:
        prefix = None
    else:
        prefix = '_'.join((block_type, str(block_idx)))
    name_fmt = partial(_generate_layer_name, prefix=prefix)


if block_type == 'Block35':
    branch_0 = conv2d_bn(x, 32, 1, name=name_fmt('Conv2d_1x1', 0))
    branch_1 = conv2d_bn(x, 32, 1, name=name_fmt('Conv2d_0a_1x1', 1))
    branch_1 = conv2d_bn(branch_1, 32, 3, name=name_fmt('Conv2d_0b_3x3', 1))
    branch_2 = conv2d_bn(x, 32, 1, name=name_fmt('Conv2d_0a_1x1', 2))
    branch_2 = conv2d_bn(branch_2, 32, 3, name=name_fmt('Conv2d_0b_3x3', 2))
    branch_2 = conv2d_bn(branch_2, 32, 3, name=name_fmt('Conv2d_0c_3x3', 2))
    branches = [branch_0, branch_1, branch_2]
elif block_type == 'Block17':
    branch_0 = conv2d_bn(x, 128, 1, name=name_fmt('Conv2d_1x1', 0))
    branch_1 = conv2d_bn(x, 128, 1, name=name_fmt('Conv2d_0a_1x1', 1))
    branch_1 = conv2d_bn(branch_1, 128, [1, 7], name=name_fmt('Conv2d_0b_1x7', 1))
    branch_1 = conv2d_bn(branch_1, 128, [7, 1], name=name_fmt('Conv2d_0c_7x1', 1))
    branches = [branch_0, branch_1]
elif block_type == 'Block8':
    branch_0 = conv2d_bn(x, 192, 1, name=name_fmt('Conv2d_1x1', 0))
    branch_1 = conv2d_bn(x, 192, 1, name=name_fmt('Conv2d_0a_1x1', 1))
    branch_1 = conv2d_bn(branch_1, 192, [1, 3], name=name_fmt('Conv2d_0b_1x3', 1))
    branch_1 = conv2d_bn(branch_1, 192, [3, 1], name=name_fmt('Conv2d_0c_3x1', 1))
    branches = [branch_0, branch_1]
else:
    raise ValueError('Unknown Inception-ResNet block type. '
                     'Expects "Block35", "Block17" or "Block8", '
                     'but got: ' + str(block_type))

mixed = Concatenate(axis=channel_axis, name=name_fmt('Concatenate'))(branches)
up = conv2d_bn(mixed,
               K.int_shape(x)[channel_axis],
               1,
               activation=None,
               use_bias=True,
               name=name_fmt('Conv2d_1x1'))
up = Lambda(scaling,
            output_shape=K.int_shape(up)[1:],
            arguments={'scale': scale})(up)
x = add([x, up])
if activation is not None:
    x = Activation(activation, name=name_fmt('Activation'))(x)
return x
```

```
def build_cnn(dim,classes,dropout_keep_prob=0.8, weights_path=None):
    inputs = Input(shape=(dim,dim,3))
    x = conv2d_bn(inputs, 32, 3, strides=2, padding='valid', name='Conv2d_1a_3x3')
    x = conv2d_bn(x, 32, 3, padding='valid', name='Conv2d_2a_3x3')
    x = conv2d  bn(x, 64, 3, name='Conv2d  2b  3x3')
    x = MaxPooling2D(3, strides=2, name='MaxPool_3a_3x3')(x)
    x = conv2d_bn(x, 80, 1, padding='valid', name='Conv2d_3b_1x1')
x = conv2d_bn(x, 192, 3, padding='valid', name='Conv2d_4a_3x3')
x = conv2d_bn(x, 256, 3, strides=2, padding='valid', name='Conv2d_4b_3x3')

# 5x Block35 (Inception-ResNet-A block):
for block_idx in range(1, 6):
   x = _inception_resnet_block(x,
                        scale=0.17,
                        block_type='Block35',
                        block_idx=block_idx)

# Mixed 6a (Reduction-A block):
channel_axis = 1 if K.image_data_format() == 'channels_first' else 3
name_fmt = partial(_generate_layer_name, prefix='Mixed_6a')
branch_0 = conv2d_bn(x,384,3,strides=2, padding='valid',
                name=name  fmt('Conv2d  1a  3x3', 0))
branch_1 = conv2d_bn(x, 192, 1, name=name_fmt('Conv2d_0a_1x1', 1))
branch_1 = conv2d_bn(branch_1, 192, 3, name=name_fmt('Conv2d_0b_3x3', 1))
branch_1 = conv2d_bn(branch_1,
                256,
                3,
                strides=2,
                padding='valid',
                name=name_fmt('Conv2d_1a_3x3', 1))
branch_pool = MaxPooling2D(3,
                    strides=2,
                    padding='valid',
                    name=name_fmt('MaxPool_1a_3x3', 2))(x)
branches = [branch_0, branch_1, branch_pool]
x = Concatenate(axis=channel_axis, name='Mixed_6a')(branches)

# 10x Block17 (Inception-ResNet-B block):
for block_idx in range(1, 11):
   x = _inception_resnet_block(x,
                        scale=0.1,
                        block_type='Block17',
                        block_idx=block_idx)

# Mixed 7a (Reduction-B block): 8 x 8 x 2080
name_fmt = partial(_generate_layer_name, prefix='Mixed_7a')
branch_0 = conv2d_bn(x, 256, 1, name=name_fmt('Conv2d_0a_1x1', 0))
branch_0 = conv2d_bn(branch_0,
```

```
            384,
            3,
            strides=2,
            padding='valid',
            name=name_fmt('Conv2d_1a_3x3', 0))
branch_1 = conv2d_bn(x, 256, 1, name=name_fmt('Conv2d_0a_1x1', 1))
branch_1 = conv2d_bn(branch_1,
            256,
              3,
              strides=2,
              padding='valid',
              name=name_fmt('Conv2d_1a_3x3', 1))
 branch_2 = conv2d_bn(x, 256, 1, name=name_fmt('Conv2d_0a_1x1', 2))
 branch_2 = conv2d_bn(branch_2, 256, 3, name=name_fmt('Conv2d_0b_3x3', 2))
 branch_2 = conv2d_bn(branch_2,
              256,
              3,
              strides=2,
              padding='valid',
              name=name_fmt('Conv2d_1a_3x3', 2))
 branch_pool = MaxPooling2D(3,
                  strides=2,
                  padding='valid',
                  name=name_fmt('MaxPool_1a_3x3', 3))(x)
 branches = [branch_0, branch_1, branch_2, branch_pool]
 x = Concatenate(axis=channel_axis, name='Mixed_7a')(branches)

 # 5x Block8 (Inception-ResNet-C block):
 for block_idx in range(1, 6):
    x = _inception_resnet_block(x,
                    scale=0.2,
                    block_type='Block8',
                    block_idx=block_idx)
x =  inception_resnet_block(x,
                  scale=1.,
                  activation=None,
                  block_type='Block8',
                  block_idx=6)

 # Classification block
 x = GlobalAveragePooling2D(name='AvgPool')(x)
 x = Dropout(1.0 - dropout_keep_prob, name='Dropout')(x)
 # Bottleneck
 x = Dense(classes, activation='softmax')(x)

 # Create model
 model = Model(inputs, x, name='inception_resnet_v1')

 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
 return model
```

# ДОДАТОК Д

## Програмна реалізація нейронної мережі для триплетів

```
from keras.layers import Dense, Lambda, Input, merge
from keras.models import Model, Sequential
from keras.optimizers import SGD

import keras.backend as K

import numpy as np


def triplet_loss(y_true, y_pred):
    return -K.mean(K.log(K.sigmoid(y_pred)))


def triplet_merge(inputs):
    a, p, n = inputs

    return K.sum(a * (p - n), axis=1)


def triplet_merge_shape(input_shapes):
    return (input_shapes[0][0], 1)


def build_tpe(n_in, n_out, W_pca=None):
    a = Input(shape=(n_in,))
    p = Input(shape=(n_in,))
    n = Input(shape=(n_in,))

    if W_pca is None:
        W_pca = np.zeros((n_in, n_out))

    base_model = Sequential()
    base_model.add(Dense(n_out, input_dim=n_in, bias=False, weights=[W_pca], activation='linear'))
    base_model.add(Lambda(lambda x: K.l2_normalize(x, axis=1)))

    a_emb = base_model(a)
    p_emb = base_model(p)
    n_emb = base_model(n)

    e = merge([a_emb, p_emb, n_emb], mode=triplet_merge, output_shape=triplet_merge_shape)

    model = Model(input=[a, p, n], output=e)
```

```
predict = Model(input=a, output=a_emb)

model.compile(loss=triplet_loss, optimizer='rmsprop')
return model, predict
```

# ДОДАТОК Е

## Навчання нейронної мережі як класифікатора на закритій множині

```
import os.path
import numpy as np
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import OneHotEncoder
from cnn import build_cnn
WEIGHTS_DIR = './utils/data/weights/'
NB_EPOCH = 512
BATCH_SIZE = 16
oh = OneHotEncoder()
train_x, train_y = np.load('./utils/data/train_x.npy'), np.load('./utils/data/train_y.npy')
test_x, test_y = np.load('./utils/data/test_x.npy'), np.load('./utils/data/test_y.npy')
n_subjects = len(set(train_y))
n_train = train_x.shape[0]
n_test = test_x.shape[0]
oh.fit(train_y.reshape(-1, 1))
train_y = oh.transform(train_y.reshape(-1, 1)).todense()
test_y = oh.transform(test_y.reshape(-1, 1)).todense()
print('n_train: {}'.format(n_train))
print('n_test: {}'.format(n_test))
print('n_subjects: {}'.format(n_subjects))
mc1 = ModelCheckpoint(WEIGHTS_DIR + 'weights.best.h5', monitor='val_acc', verbose=0, save_best_only=True,
mode='max')
if AUGMENTATION: datagen = ImageDataGenerator(rotation_range=20, width_shift_range=0.2,
height_shift_range=0.2,zoom_range=0.1, horizontal_flip=True)
model = build_cnn(299, n_subjects)
model.summary()
weights_to_load = WEIGHTS_DIR + 'weights.best.h5'
if os.path.exists(weights_to_load):
  model.load_weights(weights_to_load)
try:
  if AUGMENTATION:
    model.fit_generator(datagen.flow(train_x, train_y, batch_size=BATCH_SIZE)
steps_per_epoch=train_x.shape[0]/BATCH_SIZE,epochs=NB_EPOCH,validation_data=[test_x, test_y],callbacks=[mc1])
  else:
    model.fit(train_x, train_y,
        batch_size=BATCH_SIZE,
        epochs=NB_EPOCH,
        validation_data=[test_x, test_y],
        callbacks=[mc1])
finally:model.save_weights(WEIGHTS_DIR + 'weights.finally.h5')
```

# ДОДАТОК Ж

## Донавчання нейронної мережі триплетами.

```
import itertools
import numpy as np
from cnn import build_cnn
from tpe import build_tpe
from bottleneck1 import Bottleneck
from identification import get_scores, calc_metrics
from sklearn.decomposition import PCA
n_in = 1792
n  out = 1792
train_x, train_y = np.load('./utils/data/train_x.npy'), np.load('./utils/data/train_y.npy')
test_x, test_y = np.load('./utils/data/test_x.npy'), np.load('./utils/data/test_y.npy')
n_subjects = len(set(train_y))
cnn = build_cnn(299,n_subjects)
cnn.load_weights('./utils/data/weights/weights.best.h5')
bottleneck = Bottleneck(cnn, -1)
train_x = np.vstack([train_x, test_x])
train_y = np.hstack([train_y, test_y])
dev_x = np.load('./utils/data/dev_x.npy')
dev_protocol = np.load('./utils/data/dev_protocol.npy')
train_emb = bottleneck.predict(train_x, batch_size=128)
dev_emb = bottleneck.predict(dev_x, batch_size=128)
del train_x
Pca = PCA(n_out)
pca.fit(train_emb)
W  pca = pca.components
tpe, tpe_pred = build_tpe(n_in, n_out, W_pca.T)
#Преобразуем массив в бинарный
train_y = np.array(train_y)
#Получаем метки субъектов
subjects = list(set(train_y))
#Создаем массивы с индексами для элементов одного субъекта
anchors_inds = []
positives_inds =  []
#Создаем массив для меток
labels = []
for subj in subjects:
```

```
    #Получаем массив со значениями true субъектов с метокой subj
    mask = train_y == subj
    #Получаем индексы объектов явл. subj
    inds = np.where(mask)[0]
    #Получаем позитивные примеры
    for a, p in itertools.permutations(inds, 2):
        anchors_inds.append(a)
        positives_inds.append(p)
        labels.append(subj)

anchors = train_emb[anchors_inds]


positives = train_emb[positives_inds]
n_anchors = len(anchors_inds)
NB_EPOCH = 5000
COLD_START = 2000
BATCH_SIZE = 4
BIG_BATCH_SIZE = 512
inds = np.arange(n_anchors)
def get_batch(hard=False):
    batch_inds = np.random.choice(inds, size=BIG_BATCH_SIZE, replace=False)
    train_emb2 = tpe_pred.predict(train_emb, batch_size=1024)
    scores = train_emb2 @ train_emb2.T
    negative_inds = []
    for i in batch_inds:
        label = labels[i]
        mask = train_y == label
        if hard:
            negative_inds.append(np.ma.array(scores[label], mask=mask).argmax())
        else:
            negative_inds.append(np.random.choice(np.where(np.logical_not(mask))[0], size=1)[0])
    return anchors[batch_inds], positives[batch_inds], train_emb[negative_inds]
def test():
    dev_emb2 = tpe_pred.predict(dev_emb)
    tsc, isc = get_scores(dev_emb2, dev_protocol)
    eer, _, _, _ = calc_metrics(tsc, isc)
    return eer
z = np.zeros((BIG_BATCH_SIZE,))
mineer = float('inf')
for e in range(NB_EPOCH):
    print('epoch: {}'.format(e))
    a, p, n = get_batch(e > COLD_START)
    tpe.fit([a, p, n], z, batch_size=BATCH_SIZE, nb_epoch=1)
    eer = test()
    print('EER: {:.2f}'.format(eer * 100))
    if eer < mineer:
        mineer = eer
        tpe.save_weights('./utils/data/weights/weights.tpe.mineer.h5')
```

# ДОДАТОК И

## Підрахунок метрики для вимірювання якості прогнозу нейронної мережі

```
import numpy as np
def get_scores(data_y, protocol):
    data_y = data_y / np.linalg.norm(data_y, axis=1)[:, np.newaxis]
    scores = data_y @ data_y.T

    return scores[protocol], scores[np.logical_not(protocol)]
def calc_metrics(targets_scores, imposter_scores):
    min_score = np.minimum(np.min(targets_scores), np.min(imposter_scores))
    max_score = np.maximum(np.max(targets_scores), np.max(imposter_scores))
    n_tars = len(targets_scores)
    n_imps = len(imposter_scores)
    N = 100
    fars = np.zeros((N,))
    frrs = np.zeros((N,))
    dists = np.zeros((N,))
    min_gap = float('inf')
    eer = 0
    for i, dist in enumerate(np.linspace(min_score, max_score, N)):
        far = len(np.where(imposter_scores > dist)[0]) / n_imps
        frr = len(np.where(targets_scores < dist)[0]) / n_tars
        fars[i] = far
        frrs[i] = frr
        dists[i] = dist
        gap = np.abs(far - frr)
        if gap < min_gap:
            min_gap = gap
            eer = (far + frr) / 2
    return eer, fars, frrs, dists
```

# ДОДАТОК К

## Тестування навченої мережі

```
import numpy as np
from cnn import build_cnn
from tpe import build_tpe
from bottleneck1 import Bottleneck
from identification import get_scores, calc_metrics
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
n_in = 1792
n_out = 1792
cnn = build_cnn(299, 240)
cnn.load_weights('./utils/data/weights/weights.best.h5')
bottleneck = Bottleneck(cnn, -1)
train_x, train_y = np.load('./utils/data/train_x.npy'), np.load('./utils/data/train_y.npy')
dev_x = np.load('./utils/data/dev_x.npy')
dev_protocol = np.load('./utils/data/dev_protocol.npy')
train_emb = bottleneck.predict(train_x, batch_size=1)
dev_emb = bottleneck.predict(dev_x, batch_size=1)
del train_x
pca = PCA(n_out)
pca.fit(train_emb)
W_pca = pca.components_
print(W_pca.shape)
np.save('./utils/data/w_pca', W_pca)
W_pca = np.load('./utils/data/w_pca.npy')
tpe, tpe_pred = build_tpe(n_in, n_out, W_pca.T)
train_y = np.array(train_y)
subjects = list(set(train_y))
tpe.load_weights('./utils/data/weights/weights.tpe.mineer.h5')
dev_emb2 = tpe_pred.predict(dev_emb)
protocol = np.load('./utils/data/dev_protocol.npy')
tsc, isc = get_scores(dev_emb2, protocol)
eer, fars, frrs, dists = calc_metrics(tsc, isc)
print('EER: {}'.format(eer * 100))
plt.figure()
plt.hist(tsc, 20, color='g', normed=True, alpha=0.3)
plt.hist(isc, 20, color='r', normed=True, alpha=0.3)
plt.figure()
plt.loglog(fars, frrs)
plt.show()
for a, b, c in zip(fars, frrs, dists):
    print('a: {:.2f} | r: {:.2f} | d: {:.2f}'.format(a, b, c))
```