НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
_____
(підпис) (ініціали, прізвище)

“___” червня 2023 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою
«Системне програмування та спеціалізовані комп’ютерні системи»

спеціальності 123 комп'ютерна інженерія

на тему: «Модуль нейромережевого виділення обличчя у відеопотоці»

Виконав:
студент IV курсу, групи КВ-93

Мазенко Михайло Олегович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник проф. каф. СПіСКС, д.т.н. проф Терейковський І. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

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

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» червня 2023 р.

ЗАВДАННЯ
на дипломний проєкт студента
Мазенко Михайло Олегович
(прізвище, ім’я, по батькові)

1. Тема проєкту Модуль нейромережевого виділення обличчя у відеопотоці ,
керівник проєкту проф. каф. СПіСКС, д.т.н. проф Терейковський І.А ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « » травня 2023р. №___

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: див. ТЗ.

4. Зміст пояснювальної записки:

1) аналіз існуючих рішень та обґрунтування теми дипломного проєкту;
2) інструменти та методи розробки;
3) аналіз та розробка програмного продукту;

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо):

- Структурна схема класів програми.
- Алгоритм навчання нейромережі. Схема алгоритму.
- Алгоритм розпізнавання обличчя. Схема структурна.
- Схема взаємодії клієнта з додатком.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н.		

7. Дата видачі завдання 30 жовтня 2023р. _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів
1.	Вивчення літератури за тематикою проєкту	15.04.2023
2.	Розроблення та узгодження технічного завдання	28.04.2023
3.	Аналіз існуючих рішень	05.05.2023
4.	Підготовка матеріалів першого розділу дипломного проєкту	10.05.2023
5.	Підготовка матеріалів другого розділу дипломного проєкту	14.05.2023
6.	Підготовка матеріалів третього розділу дипломного проєкту	16.05.2023
7.	Підготовка графічної частини дипломного проєкту	18.05.2023
8.	Оформлення документації дипломного проєкту	20.05.2023
9.	Попередній огляд матеріалів диплому на кафедрі	26.05.2023

Студент _____ Мазенко М.О.
(підпис) (ініціали, прізвище)

Керівник проєкту _____ Терейковський І.А.
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проєкту.

АНОТАЦІЯ

Дипломний проєкт включає пояснювальну записку (68 сторінок, 4 додатки, 24 рисунки)

Мета розробки – розробка комп'ютерного модулю для забезпечення ефективного нейромережевого виділення обличчя у відеопотоці.

Нейронна мережа дозволяє: здійснювати навчання системи на даних зображень та розпізнавати обличчя в відеопотоці. Передбачена можливість користування системою через застосунок. В процесі розробки були використані технології мови C#.

В ході розробки:

- проведено аналіз методів побудови існуючих комп'ютерних нейромережевих систем виділення облич;
- сформульовані вимоги до нейромережевої системи виділення облич в відеопотоці;
- розроблено математичне забезпечення для нейромережевої системи виділення облич в відеопотоці;
- розроблена структура нейромережевої системи виділення облич в відеопотоці;
- проведено комп'ютерні експерименти, що довели ефективність розробленого модулю виділення;
- розроблено застосунок для управління і моніторингу роботи нейромережевої системи виділення облич в відеопотоці;

Впровадження цієї системи в своїх проектах дозволить використовувати виділення облич в своїх цілях.

Ключові слова:

модуль, згорткова нейронна мережа, нейромережева модель, відеопотік, виділення обличчя

ABSTRACT

The diploma project includes an explanatory note (68 pages, 4 appendices, 24 figures).

The aim of the development is to create a computer module for efficient neural network-based face detection in video streams.

The neural network allows: training the system on image data and recognizing faces in a video stream. It is possible to use the system through the application. C# language technologies were used in the development process.

During development:

- an analysis of methods for constructing existing computer neural network systems for face recognition was carried out;
- requirements for the neural network system for face detection in a video stream were formulated;
- mathematical support was developed for the neural network-based face detection system in video streams;
- the structure of the neural network system for face detection in a video stream was developed;
- computer experiments were conducted to prove the effectiveness of the developed face detection module;
- an application was developed for managing and monitoring the operation of the neural network system for face detection in a video stream;

The implementation of this system in projects allows for using face detection for various purposes.

Keywords:
module, convolutional neural network, neural network model, video stream, face detection

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.002 ТЗ	Нейромережева система	4		
			виділення облич в			
			Відеопотоці			
			Технічне завдання			
	А4	ІАЛЦ.045490.003 ТП	Нейромеежева система	2		
			виділення облич в			
			Відеопотоці			
			Відомість технічного			
			проєкту			
	А4	ІАЛЦ.045490.004 ПЗ	Нейромережева система	57		
			виділення облич в			
			Відеопотоці			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Взаємодія модулів системи	1		
			Схема структурна			
	А1	ІАЛЦ.045490.006 Д2	Діаграма класів в нотації	1		
			UML			
			Діаграма класів			

Змін.	Арк.	№ докум.	Підпис	Дата
Розробив		Мазенко М.О.		
Перевірив		Терейковський І.А.		
Консульт.				
Н. контроль		Клятченко Я.М.		
Зав. каф		Романкевич В.О.		

ІАЛЦ.045490.003 ТП

«Модуль нейромережевого виділення обличчя у відеопотоці»
Відомість технічного проекту

Літ.	Аркуш	Аркушів
	1	2

КПІ
ім. Ігоря Сікорського
ФПМ КВ-93

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А1	ІАЛЦ.045490.007 Д3	Схема взаємодії клієнта	1		
			з застосунком			
			Схема алгоритму			
	А1	ІАЛЦ.045490.008 Д4	Алгоритм обробки	1		
			зображення нейромережею			
			Схема алгоритму			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал.			

Змін.	Арк.	№ докум.	Підпис	Дата	ДП.045490.003 ТП	Арк. 2

ЗМІСТ

					ІАЛЦ.045490.002 ТЗ	
Зм.	Арк	№ докум	Підпис	Дата		
Розроб.		Мазенко М.О.			«Модуль нейромережевого виділення обличчя у відеопотоці» Технічне завдання	Літ. / Арк. 1 / Арк-ів 4
Перевір.		Терейковський І.А.				
						КПІ ім. Ігоря Сікорського, ФПМ КВ – 93
Н. контр.		Клятченко Я.М.				
Затв.		Романкевич В. О.				

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Модуль виділення обличчя у відеопотоці»

Галузь застосування: організація прийому та обробки вхідних відеоданих.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є створення нейромережевої системи для розпізнавання облич в відеопотоці.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з операційними системами: Windows, Linux.
- можливість роботи з відеофайлами;
- можливість роботи з файлами зображення;
- можливість додавати нові матеріали для навчання;
- наявність зручної системи для взаємодії з нейромережею;

5.2. Вимоги до апаратного забезпечення

- Процесор: 8-ядерний,Intel CORE i-7;
- Оперативна пам’ять: 8 Гб;
- Місце на жорсткому диску 5гб;

5.3. Вимоги до програмного та апаратного забезпечення користувача

- Операційна система Windows, Linux.
- .NET 7.0

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів
1.	Вивчення літератури за тематикою проєкту	15.04.2023
2.	Розроблення та узгодження технічного завдання	28.04.2023
3.	Аналіз існуючих рішень	05.05.2023
4.	Підготовка матеріалів першого розділу дипломного проєкту	10.05.2023
5.	Підготовка матеріалів другого розділу дипломного проєкту	14.05.2023
6.	Підготовка матеріалів третього розділу дипломного проєкту	16.05.2023
7.	Підготовка графічної частини дипломного проєкту	18.05.2023
8.	Оформлення документації дипломного проєкту	20.05.2023
9.	Попередній огляд матеріалів диплому на кафедрі	26.05.2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CNN – Convolutional neural network

RNN – Recurrent neural network

DNN – Deep neural network

MTCNN – Multi-task cascaded neural network

ШНМ – Штучна нейронна мережа

ReLU – Rectified linear unit

RGB – скорочено від англ. Red, Green, Blue – червоний, зелений, синій

MP – Max pooling

HF – Haar feature

TPU – Tensor Processing Unit

GPU – Graphics processing unit

DL – Deep learning

P-Net – Proposal network, перший модуль MTCNN

R-Net – Refine network, другий модуль MTCNN

O-Net – Output network, третій модуль MTCNN

BB – Bounding box, обмежуюча рамка

ML – Machine learning

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET

WPF – Windows Presentation Foundation .NET Framework

Microsoft – одна з найбільших та найвідоміших технологічних компаній у світі, має широкий спектр продуктів і послуг, пов'язаних з комп'ютерами, програмним забезпеченням та інтернет-технологіями

IoU – Intersection over Union

ВСТУП

За останні роки нейронні мережі здобули великий успіх в багатьох галузях, таких як комп'ютерне зорове сприйняття, машинне навчання, обробка мовлення, розпізнавання мови та інші. У кінці 2010-х років з'явилися глибокі нейронні мережі, які дозволили вирішувати складні завдання в багатьох галузях, де раніше було складно досягти гарної точності. Однією з головних переваг нейронних мереж є їх здатність до самонавчання, тобто здатність до адаптації до нових вхідних даних, що дозволяє їм стати універсальними інструментами для вирішення багатьох завдань. Крім того, нейронні мережі можуть працювати з неструктурованими даними, такими як звук, зображення або текст, що робить їх особливо корисними в багатьох застосуваннях. Останні роки були вирішальними для розвитку нейронних мереж, з'явилися нові архітектури, такі як глибокі згорткові мережі (CNN), рекурентні нейронні мережі (RNN) та глибокі нейронні мережі (DNN), які здатні досягати вражаючої точності в різних завданнях. Крім того, з'явилися нові методи тренування, такі як навчання з підсиленням та навчання без вчителя, які дозволяють нейронним мережам вчитися без потреби великої кількості розмічених даних. В сучасному світі відеоаналіз відіграє важливу роль в багатьох сферах, таких як безпека, медицина, розваги тощо. Аналіз відео потоків є важливим завданням в цих сферах і потребує високоточної обробки даних. Одним із ключових завдань є виділення та розпізнавання облич від зображення або відео. Модуль виділення обличчя є важливим кроком для більш складних систем, які використовуються в відеоаналізі.

Метою даної роботи є розробка комп’ютерного модулю для забезпечення ефективного нейромережевого виділення обличчя у відеопотоці. Цей модуль може бути використаний в більш складних системах для визначення особливостей облич, таких як аналіз емоцій, розпізнавання осіб, відстеження руху тощо.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

1.1. Постановка задачі виділення обличчя.

Проблема розпізнавання облич стає все більш актуальною в сучасному світі, де безпека та ідентифікація особистості відіграють ключову роль. Існує безліч моделей для її вирішення, від класичних, наприклад, базованими на ознаках обличчя, таких як система Віоли-Джонса для виявлення об'єктів, до найсучасніших методів глибокого навчання, таких як багатозадачна каскадна згортка нейронної мережі, також відома як MTCNN. Основне завдання модулю розпізнавання обличчя полягає в тому, щоб знайти одне або більше облич на фотографії, та локалізувати їх. Зазвичай знайдене лице поміщується в обмежувальну рамку, яка вказує на те, де закінчуються межі лиця. Отже, такий модуль повинен приймати на вхід зображення, а повертати – або нічого, якщо не було виявлено жодного лиця на фотографії, або список з даних, які вказують на розташування облич людей на фотографії. Завдання ускладнюється великою кількістю різних факторів, таких як: орієнтація фотографії, кут нахилу лиця, кут, під яким людина дивиться, аксесуари, колір волосся, волосся на обличчі, макіяж, вік рівень освітлення, тощо. Отже, найпростішим та найефективнішим способом вирішення цієї проблеми є нейронні мережі, адже вони можуть самі підбирати ознаки для ідентифікації шуканих об'єктів.

Найперші штучні нейронні мережі були створені за зразком біологічних нейронних мереж, що складають мозок тварин, і подалі більш нові варіанти ШНМ також беруть за основу біологічні процеси. Так, наприклад, згоркова нейронна мережа реалізує схему, за якою з'єднані нейрони зорової кори живих істот, повторюючи механізми взаємодії

стимулу та рецептивного поля. Сучасні нейронні мережі, зазвичай, організовано в шари – вхідний, вихідний та приховані. Вхідний шар повинен приймати інформацію, розподіляти її по нейронам, і передавати далі. Вихідний є останнім шаром нейронної мережі і містить вихідні значення, або прогнози моделі для певної задачі. Найважливішою частиною будь-якої нейронної мережі є, звісно, приховані шари, саме від їх кількості та конфігурації залежить те, як буде працювати нейронна мережа та який результат вона видасть. Приховані шари нейронної мережі отримують на вхід вектор ознак, який проходить через нейрони в цьому шарі, виконуючи лінійну комбінацію вхідних даних та зважуваних коефіцієнтів, які називаються вагами. Після цього відбувається активація шару, яка застосовує нелінійну функцію до результату лінійної комбінації. Це дозволяє нейронам шару виявляти складніші залежності між вхідними даними та вихідними значеннями, що дозволяє здійснювати більш складні завдання, ніж просте лінійне прогнозування.

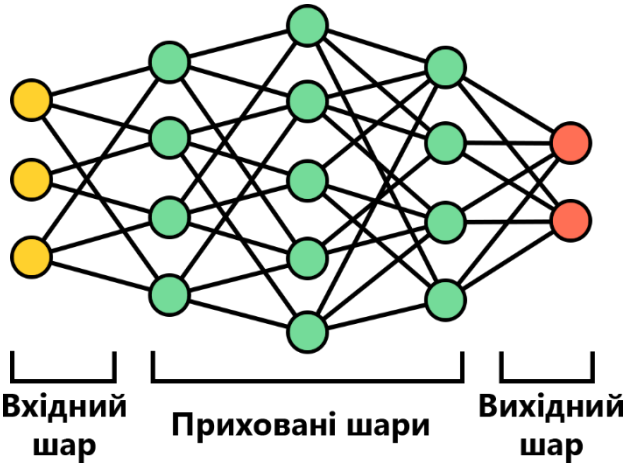


Рисунок 1.1 – Шари нейронної мережі

В такій системі, нейрони можуть приймати значення тільки від інших нейронів, які обов'язково мають знаходитись на попередньому шарі нейронної мережі, а передавати результат, в свою чергу, можуть лише тим нейронам, які знаходяться на наступному шарі, це дозволяє створити потік даних, які будуть рухатись лише в одному напрямку. Вихідні дані будь-якого нейрона будуть залежати лише від тих значень, які він отримає від

попереднього шару. Отримання значення на виході нейрона досягається застосуванням нелінійної функцію активації до значення сумації нейрона. Зазвичай в якості нелінійної функції використовують сигмоїду, гіперболічний тангенс, ReLU тощо. Отже, алгоритм отримання функції сумації виглядає так: значення на виході попередніх нейроннів, які з'єднані до поточного нейрона, перемножуються з відповідними вагами, сумуються, далі до них додається біас, який виступає у ролі константи, тим самим дозволяючи зміщювати функцію активації. Саме значення біасу та ваг дозволяють навчати нейронну мережу, адже змінивши їх ми можемо змінити важливість тих чи інших даних з попередніх нейронів, що вплине на отриманий результат.

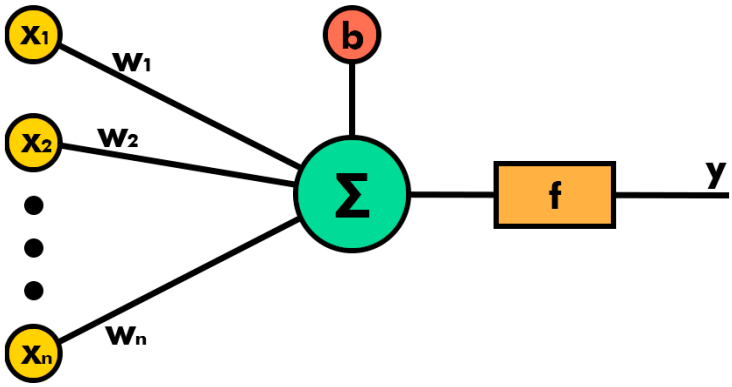


Рисунок 1.2 – Простий штучний нейрон

На самому початку роботи ШНМ створюються з'єднання між нейронами на різних шарах. Типи з'єднання для різних архітектур можуть відрізнятися, наприклад, для зв'язування звичайних шарів часто застосовують метод повного з'єднання шару (Fully connected layer): кожен нейрон попереднього шару повинен бути з'єднаний з кожним нейроном наступного шару, для архітектури CNN застосовують з'єднання по області тощо. Після створення з'єднань, встановлюються усі значення ваг та біосів за допомогою генератора випадкових чисел, для того, щоб уникнути уткнення нейронів у локальних мінімумах функції втрат, яка використовується для оцінки помилки прогнозування мережі. Враховуючи

те, що всі вагові коефіцієнти будуть починатися з одного й того ж значення, то під час навчання мережі, в найгіршому випадку, всі нейрони можуть спрямовуватися у одному напрямку і не знаходити оптимальних рішень для виконання завдання. Після цих двух етапів, нейронна мережа вже готова до експлуатації, і на її вхід вже можна надати дані, які будуть корегуватися та змінюватися за допомогою прихованих шарів, зважаючи на архітектуру ШНМ та значення ваг, які з'єднують нейрони.

Згорткова нейронна мережа (Convolutional neural network) є найдоречнішою для роботи з зображенням, адже саме ця архітектура була розроблена на основі біологічної зорової кори. Для початку потрібно визначитись, як слід перетворювати фотографію в дані, які будуть придатними для роботи в нейронній мережі. Зображення складається з пікселів, кожен піксель має свою координату на зображені, має свій колір та відтінок. Колір пікселя подається за допомогою RGB, де R – це червоний, G – зелений, B – синій, кожен з цих каналів може мати значення від 0 до 255, яке вказує на інтенсивність відповідного основного кольору. Комбінація цих трьох кольорів у кожному пікселі дозволяє створювати широку гаму кольорів, що дозволяє відтворювати різноманітні зображення та відео на екранах електронних пристроїв. Отже, потрібно знайти значення, які будуть характеризувати кожен окремий піксель, їх можна отримати, наприклад за допомогою зваженого метода перетворення RGB в градацію сірого. Червоний, зелений та синій кольори зважуються відповідно до довжини їх хвиль, сума зважених значень каналів і буде тим самим шуканим значенням. Отримавши одне число для кожного пікселю, процес передачі даних на вхід нейронної мережі стане простішим, але все ж таки перед цим ще бажано нормалізувати ці дані. Нормалізація є дуже важливою, тим більше якщо мова йде про CNN, адже вона зменшує вплив масштабу даних, забезпечує швидкість збіжності, забезпечує стабільність, оптимізує швидкодію. Дані в нормалізованому вигляді вже готові до

передачі на вхідний шар нейронної мережі, через те, що він не виконує ніяких обрахунків, його структура залишається дуже простою, і в навчанні приймати участі також не буде. Єдине, що слід зауважити щодо цього шару – це те, що кількість нейронів потрібна дорівнювати кількості пікселів зображення, адже кожен нейрон буде відповідати за один піксель.

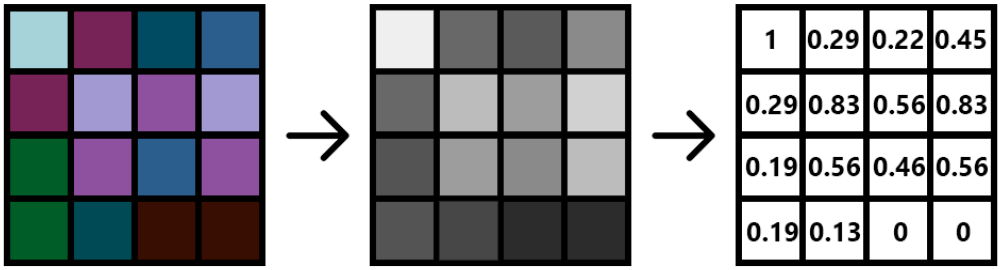


Рисунок 1.3 – Підготовка вхідних даних

Приховані шари в CNN є найважливішою частиною моделі, так само як і в інших ШНМ. Кожен прихований шар складається з набору фільтрів, які здійснюють операцію згортки над вхідним зображенням. Операція згортки полягає в тому, що кожен фільтр проходить по матриці і виконує певні математичні дії, наприклад множення кожного елемента вхідної матриці на відповідний елемент фільтру, після чого результати додаються та заносяться до масиву результуючих даних. Операція згортки показує, як матриця реагує на певні ознаки, наприклад, ребра, кути або текстуру. Після проходження через шар з фільтрами збільшується кількість каналів, які представляють різні характеристики зображення. Наприклад, на початку мережі можуть використовуватися фільтри, які реагують на різні кольори, а на наступному шарі - фільтри, які реагують на ребра або кути. Вихідна матриця називається картою ознак (Feature map), та може мати відступ з нульовими значеннями по боках, якщо це передбачено моделлю, а розмір відрізнятися залежачи від кроку та розмірності фільтра. Важливим етапом в обробці даних в CNN є пулінг (pooling). Враховуючи те, що навіть зображення, на першу думку, малого розміру, наприклад, 100x100 має в собі 10000 пікселів, які повинні оброблювати така сама кількість нейронів,

а кількість карт ознак на кожному наступному шарі, як правило, буде збільшуватись або залишатися сталою, потрібно якось зменшити зображення і запобігти втрачанню важливих даних. В дію вступає пулінг, який має потрібні корисні функції. По-перше, він дозволяє зменшити розмір зображення, що сприяє зниженню обчислювальних витрат і зменшенню кількості параметрів моделі. По-друге, він допомагає уникнути перенавчання (overfitting), що може виникнути при занадто великій кількості параметрів. Операція пулінгу полягає в тому, що для кожної зони вхідної карти ознак виконується певна математична операція, яка зменшує його розмір. Зазвичай використовуються дві основні операції пулінгу: максимальний пулінг та середній пулінг. Максимальний пулінг (MP) знаходить найбільше значення для кожної зони карти ознак, яке передається до наступного шару. Саме цей тип дозволяє виділити найбільш значущі ознаки зображення, що не є характерним для середнього пулінгу. Після виконання операції пулінгу, розмір зображення зменшується, а кількість параметрів в нейронній мережі стає менше. Це дозволяє прискорити навчання мережі та запобігти перенавчанню. Також операція пулінгу може допомогти збільшити інваріантність до масштабу та перенесення зображення, що дозволяє збільшити точність розпізнавання. Після згорткової частини зазвичай додається декілька повнозв'язних шарів, які вже мають повну інформацію і здатні визначити результат роботи нейронної мережі, який представлений вихідним шаром.

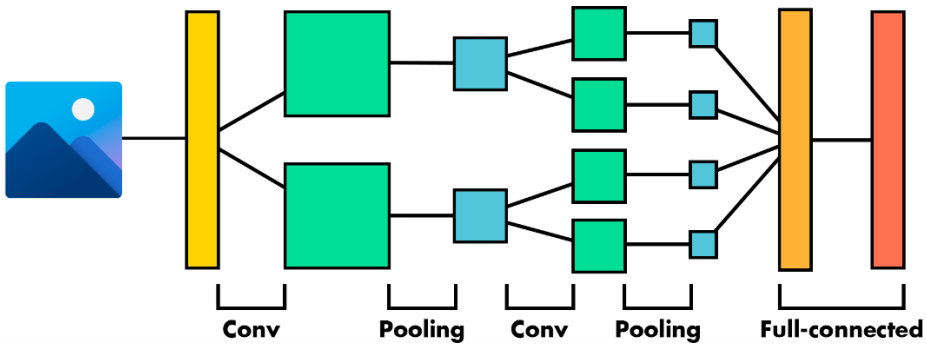


Рисунок 1.4 – структура простої згорткової нейронної мережі

Згорткова нейронна мережа є ефективним та потужним інструментом для обробки зображень, що робить її дуже популярною в будь-яких задачах комп'ютерного зору, наприклад виділення обличчя. Сучасні моделі згорткових нейронних мереж, такі як MTCNN дозволяють досягти рекордної точності та швидкості обробки порівняно з іншими моделями нейронних мереж.

1.2. Аналіз існуючих засобів

Існують різні засоби розпізнавання облич, які використовуються в різних галузях і мають свої переваги та недоліки. Першим типом таких засобів є алгоритми, які дозволяють відстежувати обличчя без застосування нейромереж. Основна ідея таких алгоритмів полягає в тому, що обличчя має певні типові риси, які можна використовувати для детекції. Найвідомішим з таких алгоритмів є алгоритм Віоли-Джонса, який використовує каскадний класифікатор, що складається з декількох ступенів. Кожен ступінь містить декілька класифікаторів, які визначають, чи містить підзображення (порція зображення) обличчя. Якщо підзображення пройшло перший ступінь класифікації, воно подається на другий ступінь, де знову проводиться класифікація і так далі, до тих пір, поки не буде визначено, що підзображення містить обличчя або ж буде відкинуто. Кожен класифікатор в алгоритмі Віоли-Джонса базується на так званому гаусіанському інтегралі. Гаусіанський інтеграл це інтеграл функції ваги (важливості) пікселів у певній області зображення. Інтегральне зображення створюється шляхом послідовного додавання значень пікселів зліва направо та зверху вниз. Це дозволяє дуже швидко обчислювати суму значень пікселів в будь-якій прямокутній області зображення.

Більшість засобів, звісно, використовують ШНМ для відстеження обличчя, наприклад OpenCV. OpenCV (Open Source Computer Vision

Library) - це бібліотека відкритого коду, призначена для обробки зображень та комп'ютерного зору. Ця бібліотека дозволяє розпізнавати обличчя, розпізнавати рух, визначати кольори та знаходити об'єкти на зображеннях та відео. OpenCV підтримує більшість популярних мов програмування, таких як C++, Python, Java та інші. OpenCV має багато функцій для роботи з розпізнаванням облич. Одна з них - це відстеження обличчя (Face Tracking). Відстеження обличчя дозволяє визначити положення та орієнтацію обличчя в кадрі відео, а також розпізнати особливі риси, такі як очі, ніс та рот. OpenCV використовує алгоритми, такі як Viola-Jones, для визначення обличчя на зображенні та його відстеження в режимі реального часу. Іншою функцією OpenCV є розпізнавання емоцій на обличчі (Facial Expression Recognition). Для цього використовуються глибокі нейронні мережі, які були навчені на великих наборах даних емоцій, щоб розпізнавати емоції на обличчі. Таким чином, OpenCV може розпізнавати емоції, такі як радість, сум, здивування тощо, на обличчі в режимі реального часу. OpenCV також має функцію розпізнавання осіб (Face Recognition). Для розпізнавання облич використовуються глибокі нейронні мережі та методи машинного навчання, такі як PCA (Principal Component Analysis) та LDA (Linear Discriminant Analysis).

DeepFace - це бібліотека для розпізнавання облич, розроблена компанією середовища програмування Python. Ця бібліотека забезпечує можливість розпізнавати обличчя в реальному часі, використовуючи нейронні мережі. DeepFace використовує дві моделі: VGG-Face і FaceNet. VGG-Face була розроблена у 2015 році, і її використовують для екстракції ознак з обличчя. Ознаки обличчя передаються в Facenet, яка використовує нейронну мережу для розпізнавання облич. Facenet використовує вхідні дані в форматі RGB з розміром 96x96 пікселів. На відміну від VGG-Face, FaceNet може працювати з довільними розмірами зображень, що дає можливість використовувати її з більш різноманітними джерелами

зображень. FaceNet розроблена для вирішення завдання розпізнавання облич. Вона була розроблена у 2015 році фахівцями з машинного навчання Google Research та може досягати дуже високої точності при розпізнаванні облич. Основна ідея FaceNet полягає в тому, що вона навчається відбивати властивості обличчя у вектори у такий спосіб, що обличчя, які належать до однієї людини, матимуть схожі вектори, а обличчя різних людей будуть мати відмінні вектори. Після цього можна порівнювати вектори, щоб визначити, чи належать два обличчя одній людині. Архітектура FaceNet базується на концепції віддаленості та універсальності вбудовування. Вона складається з трьох основних блоків: блоку згортки, блоку витягування ознак та блоку нормалізації. Після цього вектори ознак, отримані від блоку витягування ознак, пропускаються через блок нормалізації L2, що забезпечує універсальність вбудовування. Основною перевагою FaceNet є його висока точність та здатність до роботи з різними видами зображень обличчя. Вона досягає більш ніж 99% точності при розпізнаванні лиць у відкритих джерелах даних, таких як датасет Labeled Faces in the Wild. DeepFace дозволяє розпізнавати обличчя на фото та відео. Крім того, вона має можливість виконувати багато інших завдань, таких як оцінка віку та статі, визначення емоцій та ідентифікація сліпих зон на обличчі. Окрім того, DeepFace може бути використана в багатьох сферах, таких як безпека, дослідження та розваги.

Існують також хмарні сервіси розпізнавання облич, наприклад – Azure Face API. Цей сервіс надає програмістам і дослідникам доступ до широкого спектру функцій розпізнавання облич та аналізу відображень облич. Azure Face API надає такі функції, як розпізнавання облич, розпізнавання емоцій, визначення віку та статі, визначення настрою та багато іншого. Щоб використовувати Azure Face API, потрібно зареєструватися в Azure та створити ресурс для Azure Face API. Після цього можна створювати запити до API, використовуючи HTTP запити або

SDK для різних мов програмування, таких як Python, Java, C# та інші. Однією з особливостей Azure Face API є підтримка глибоких нейронних мереж, таких як ResNet, що дозволяє досягати високої точності розпізнавання облич. Крім того, Azure Face API має підтримку багатоформатних зображень, що дозволяє обробляти зображення у різних форматах, таких як JPEG, PNG, BMP, GIF та інші. Azure Face API також має можливість створювати персоналізовані моделі розпізнавання облич, що дозволяє досягати більшої точності розпізнавання облич для конкретних завдань. Наприклад, якщо потрібно розпізнавати обличча лише певної групи людей (наприклад, співробітників компанії), то можна створити персоналізовану модель, що буде оптимізована саме для цієї групи людей.

1.3. Обґрунтування теми дипломного проєкту

Одним з головних недоліків є вразливість до атак із зміною зображень. Це означає, що нейронні мережі можуть бути легко ошукані, коли їм надається фотографія зі штучно створеними змінами, такими як наложення маски, змінення освітлення, вирізання або зміна орієнтації. Ця проблема виникає через те, що нейронні мережі зазвичай засновані на підбірці певних особливостей, які залежать від конкретного зображення. Якщо ці особливості змінюються або видаляються, то мережа може зійти з ладу. нший недолік полягає у складності тренування і підтримки нейронних мереж. Для тренування ефективної мережі для розпізнавання облич необхідно мати велику кількість даних, що є складним завданням, особливо якщо дані повинні бути представленими різними людьми, різними орієнтаціями та поглядами. Крім того, тренування такої мережі може зайняти багато часу і потребує значних ресурсів обчислювальної техніки. Також слід враховувати, що більшість нейронних мереж розпізнавання облич засновані на задачах бінарної класифікації, де система

розпізнає лише дві категорії: обличчя і не обличчя. Це може призвести до помилок в розпізнаванні людей, які мають особливості, такі як шрами, бороди або окуляри. Іншим недоліком більшості нейронних мереж розпізнавання обличчя є необхідність великої кількості даних для тренування, що може бути проблематичним для бізнесу або в областях, де недостатньо даних для тренування.

Отже, сворення власної архітектури ШНМ розпізнавання обличчя може бути вигідним в тих випадках, коли необхідна більша точність та швидкість розпізнавання, а також коли доступна обмежена кількість даних для тренування. Також, створення власної архітектури дає можливість зберегти конфіденційність даних, оскільки дані можуть бути збережені та оброблені на локальному комп'ютері або сервері без необхідності використання хмарних сервісів. Власний софт може бути створений, враховуючи конкретні вимоги і особливості задачі. Наприклад, якщо вам потрібна розпізнавання облич в обмеженому середовищі, такому як підводна зйомка, можна створити архітектуру, яка буде адаптована до поганих умов зйомки. Власна архітектура може бути оптимізована для конкретних задач, що дозволяє досягти більшої ефективності порівняно з існуючими рішеннями. Використання сторонніх бібліотек та рішень може бути обмеженим правовими або етичними обмеженнями. Створення власного софту дає можливість контролювати всі етапи розробки, включаючи джерело даних та алгоритми обробки. Власні архітектури можуть бути розширені з урахуванням нових потреб або технологічних рішень. Наприклад, можна додати можливість розпізнавання емоцій обличчя або врахувати нові алгоритми обробки зображень. Використання сторонніх рішень може створювати додаткові ризики для безпеки та приватності даних. Створення власних архітектур нейронних мереж дозволяє контролювати захист даних та вибирати технології шифрування.

Головним завдання модулю є правильно виділяти обличчя в відеопотоці. Для цього слід обрати вірну структуру нейромережі, яка дозволить ефективно впоратися з цією задачею. Обраний метод побудови нейромережевої системи має бути актуальним та ефективним для використання при роботі з зображенням та великою кількістю даних. Основними вимогами до створеної мережі буде швидка робота та точність результатів, адже система повинна працювати в реальному часі з відеопотоком. Математичне забезпечення для нейромережевої системи включає в себе розробку математичних алгоритмів та моделей, які дозволяють ефективно обробляти вхідні дані, навчати та використовувати нейромережеву систему для досягнення певної мети. Це може включати в себе вибір оптимальної архітектури нейромережі, налаштування параметрів навчання, оптимізацію вагів та інші математичні методи, що допомагають забезпечити ефективну роботу нейромережі. Математичне забезпечення є важливим елементом розробки нейромережевих систем, оскільки воно дозволяє забезпечити їх ефективність та точність в завданні виділення обличчя. Для обґрунтування ефективності розробленого модулю нейромережі мають бути проведені наступні експерименти: порівняння точності розпізнавання облич у відеопотоці за допомогою розробленого модулю та інших аналогічних систем, що використовуються на ринку, оцінка швидкодії розробленого модулю в порівнянні з іншими системами, аналіз стійкості розробленого модулю до різних типів спотворень.

2. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЮ НЕЙРОМЕРЕЖЕВОГО ВИДІЛЕННЯ ОБЛИЧЧЯ У ВІДЕОПОТОЦІ

2.1. Базова нейромережева модель

В якості базової нейромережевої моделі використано, оскільки такий тип нейромережевої моделі є найбільш опробуваним у задачі обробки відеопотоку. Класична ЗНМ, як і більшість інших нейронних мереж скадається з шарів нейронів. Кількість шарів може варіюватися в залежності від складності нейромережі та її призначення, а самі шари в свою чергу діляться на типи, які виконують певні функції. Кожен шар може мати свою функцію активації, яка буде відмінною від інших шарів. Функція активації в нейромережі відповідає за додавання нелінійності до вихідних значень нейронів. Це важливо, оскільки без нелінійності нейромережа може обмежитися лінійними трансформаціями вхідних даних, що є обмеженою моделлю. Різні функції активації мають різний вплив на роботу нейромережі. Наприклад, сигмоїдна функція активації поводиться як порогова функція, яка переводить вхідні дані в діапазон від 0 до 1. Це призводить до того, що нейрони з сигмоїдною функцією активації можуть вивчати нелінійні залежності вхідних даних. Однак, сигмоїдна функція має проблему зі зниклим градієнтом, що може ускладнити процес навчання. Інші функції активації, такі як ReLU (Rectified Linear Unit), Leaky ReLU, ELU (Exponential Linear Unit) та інші, мають інші переваги та недоліки. Наприклад, ReLU має швидкий обчислювальний час, проте може виникнути проблема з "мертвими" нейронами, які неактивні для певного діапазону вхідних даних. Тому, вибір функції активації залежить від конкретної задачі та архітектури нейромережі, і є однією з важливих технік оптимізації нейромережі. Від

конфігурації шарів та порядку їх розташування залежить те, як буде працювати нейронна мережа, які результати вона видасть.

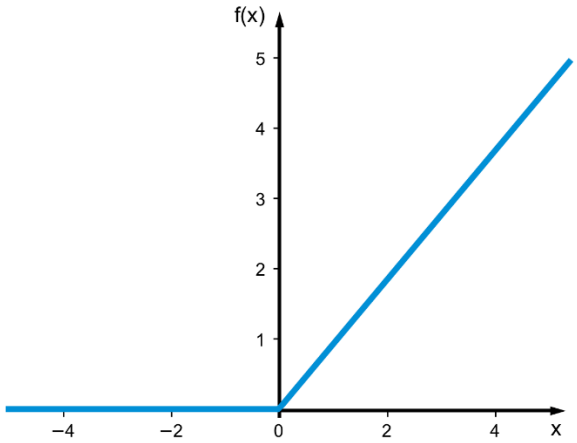


Рисунок 2.1 – нелінійна функція активації ReLU

Шар згортки є одним з основних типів шарів у згортковій нейронній мережі, який використовується для виявлення локальних функцій у вхідних даних. Згортковий шар приймає на вхід тривимірні дані, такі як зображення або відео, та застосовує до них набір фільтрів (також відомих як ядра або фільтри згортки). Кожен фільтр - це невеликий набір чисел, які використовуються для обчислення відповідних піксельних значень в вихідному зображенні. Кожен фільтр вирішує певну задачу, таку як визначення контурів, текстур, рисок тощо. Після обчислення вихідних значень для всіх фільтрів, шар згортки може застосувати до них додаткову операцію, пуллінг. Кожен шар згортки містить певну кількість фільтрів (або ядро), які застосовуються до вхідного зображення з різними параметрами. При застосуванні фільтрів згортки, шар пересуває їх по вхідних даних з певним кроком (так званим кроком згортки) та обчислює добуток між значеннями вхідних даних та відповідними елементами фільтра. Ці добутки згортаються разом (відтак назва "згортковий шар") та дають відповідні вихідні значення. В результаті згортки зображення повинно зменшитись, але якщо воно занадто мале, то відповідне вихідне зображення буде ще меншого розміру. Це може призвести до втрати

важливої інформації та зниження точності нейромережі. Щоб запобігти цьому, можна застосовувати заповнення нулями до вхідного зображення. Це означає, що до країв зображення додається деяка кількість пікселів, заповнених нулями. Це дозволяє збільшити розмір вихідного зображення і запобігти втраті інформації. Існує два типи заповнення нулями: 'valid' та 'same'. При використанні 'valid' заповнення нулями не використовується, тому вихідне зображення буде меншого розміру, ніж вхідне. При використанні 'same' заповнення нулями додається до країв зображення, щоб збільшити розмір вихідного зображення до розміру вхідного зображення. Застосування zero-padding допомагає зберегти важливу інформацію про контекст зображення та збільшити точність нейромережі. Однак використання заповнення нулями також збільшує обчислювальні витрати при обробці зображень, тому потрібно бути уважним при виборі розміру заповнення. Тип заповнення, кількість шарів згортки та їх параметри (кількість фільтрів, розмір фільтрів, крок зміщення тощо) визначаються на етапі проектування моделі згорткової нейронної мережі і зазвичай залежать від конкретної задачі, яку необхідно вирішити. В цілому, згортковий шар допомагає нейромережі виявляти локальні функції в зображенні та утворювати більш складні конструкції вищих рівнів, засновані на цих локальних функціях. Це робить згортковий шар потужним інструментом для розв'язання завдань з обробки зображень, розпізнавання облич та багатьох інших.

Шар пулінгу – це інший тип шару, який використовують в згорткових нейронних мережах. Цей шар використовується для зменшення розміру зображення та робить мережу більш стійкою до зміщень вхідних даних. Шар пулінгу ділить вхідні дані на підобласті і виконує певні обчислювальний операції для кожної з них. Зазвичай використовується операція максимального пулінгу, коли для кожної підобласті вибирається максимальне значення, яке зберігається в вихідних даних. Інші операції

пулінгу такі як середнє значення, L2-норму, і т.д. Шар пулінгу використовується для кількох цілей. По-перше, він зменшує розмір зображення, тому що максимальне значення в кожній підобласті перетворюються в одне число, це дозволяє зменшити кількість параметрів моделі та сприяє запобіганню перенавчання. Крім того, шар пулінгу може забезпечити інваріантність до зсуву. Якщо вхідні дані були зсунуті на деякий відстань, то максимальне значення для підобласті все ще буде тим самим. Шар пулінгу зазвичай застосовуються після згорткового шару, щоб зменшити розмір вихідних даних. Він також може бути застосований кілька разів для подальшого зменшення розміру. Функція активації може виконуватись після шару пулінгу. У цьому випадку, вхідні дані передаються до шару пулінгу, після чого до отриманої карти ознак застосовується функція активації. Після цього карту ознак можна передати на наступний шар згортки. Проте, у більшості згорткових нейронних мереж функція активації виконується після кожного шару згортки, а не після пулінгу. Це пов'язано з тим, що згорткові шари мають більшу кількість параметрів та, відповідно, здатні здійснювати більш глибоку та точну обробку даних, ніж шар пулінгу. Хоча шар пулінгу допомагає зменшити розмір вихідних даних та зробити мережу більш стійкою до зміщень, деякі дослідження показують, що він може втратити частину інформації, тому його надмірне використання може погіршити роботу нейронної мережі.

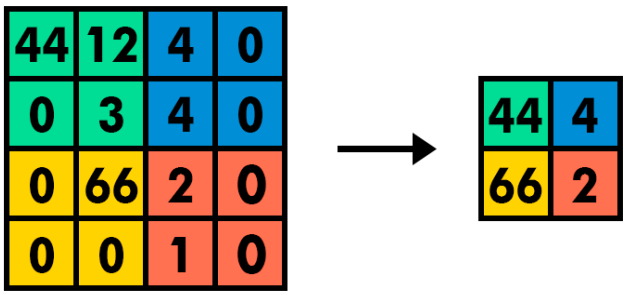


Рисунок 2.2 – операція максимального пулінгу

Повнозв'язний – ще один тип шару, який використовується в згортковій нейронній мережі, виконує ту ж саму функцію, що й у звичайній мережі — зведення вхідних даних до вектору ознак, який піддається подальшій обробці. Основна різниця полягає в тому, що в згортковій мережі перед повнозв'язним шаром зазвичай використовуються один або кілька шарів згортки та пулінгу, що дозволяє отримати більш складні та абстрактні ознаки від вхідних даних. Повнозв'язний шар отримує на вхід вектор ознак, що містить інформацію про різні аспекти вхідних даних, та виконує над ним операції лінійної алгебри: множення матриць та додавання векторів. На виході отримується вектор, що містить інформацію про класифікацію або регресію вхідних даних. У згортковій мережі повнозв'язний шар може складатися з кількох нейронів, кожен з яких виконує обчислення на основі підмножини вхідних ознак. Наприклад, якщо ми маємо зображення обличчя, то перші шари згортки можуть виявляти прості риси (наприклад, кути очей, ніс), наступні шари можуть виявляти складніші ознаки (наприклад, форму губ), а повнозв'язний шар може виконувати класифікацію зображення на основі цих ознак (наприклад, розпізнавання, чи на зображенні зображене обличчя чоловіка чи жінки). Функції активації, які використовують на повнозв’язних шарах нейромережі, в основному збігаються з тими, які використовуються на згорткових шарах. Однак, є кілька додаткових функцій, які можуть бути використані на повнозв’язних шарах. Одна з таких функцій - це лінійна функція активації, яка повертає значення, що є просто лінійною комбінацією входів, без якої-небудь не лінійної перетворення. Ця функція рідко використовується на згорткових шарах, але може бути корисною на повнозв’язних шарах, де інші функції активації можуть бути занадто складними для моделювання. Інша популярна функція активації для повнозв’язних шарів - це сигмоїдна функція, яка має багато мінусів, тому зазвичай замість неї використовується гіперболічний тангенс. До переваг

використання цієї функції активації можна віднести те, що вона може генерувати великі вихідні значення при великих вхідних значеннях, що може бути корисним у деяких випадках.

Вихідний шар в згортковій нейронній мережі - це останній шар, що оброблює вихідні дані з попередніх шарів, і генерує вихідний результат мережі. Зазвичай вихідний шар містить повнозв'язний шар (fully connected layer), який підключений до передостаннього шару мережі. Повнозв'язний шар вихідного шару має таку ж структуру, як і повнозв'язний шар з останнього шару у класичній нейронній мережі. Кількість нейронів у цьому шарі зазвичай відповідає кількості класів, на які поділено дані для класифікації або кількості вихідних змінних у випадку регресії. Функції активації, що використовуються на вихідному шарі, залежать від задачі, яку розв'язує мережа. Наприклад, у задачах бінарної класифікації можна використовувати сигмоїдну функцію активації, а у задачах багатокласової класифікації – функцію softmax. У випадку регресії можна використовувати лінійну функцію активації. Після застосування функції активації на вихідному шарі, отримується вектор вихідних значень, який можна інтерпретувати як ймовірності належності вхідних даних до кожного з класів у випадку класифікації, або як прогнозовані значення у випадку регресії.

Сьогоденні згорткові нейронні мережі є надзвичайно точними та швидкими. Наприклад, ResNet (Residual Network), що була запропонована у 2015 році, має дуже глибоку архітектуру (до 152 шарів) та досягає вражаючих результатів на багатьох завданнях комп'ютерного зору. Для прикладу, на датасеті ImageNet, ResNet забезпечує точність понад 95% для завдання класифікації зображень, що є значним покращенням порівняно з попередніми архітектурами. Також, варто зазначити, що з'явилися спеціалізовані згорткові нейронні мережі, призначені для конкретних завдань, наприклад, мережі для детекції облич та їх атрибутів (наприклад,

OpenFace), мережі для сегментації зображень (наприклад, U-Net) та мережі для генерації зображень (наприклад, StyleGAN). Щодо швидкості, згорткові нейронні мережі зазвичай вимагають значних обчислювальних ресурсів для навчання та застосування, але існують різні методи для покращення швидкості, такі як розпаралелення, оптимізація обчислювальної графи та використання апаратного прискорення, наприклад, за допомогою графічних процесорів (GPU) або тензорних процесорів (TPU). Деякі згорткові мережі, такі як MobileNet та SqueezeNet, були розроблені спеціально для прискорення обчислень та використання на мобільних пристроях.

Датасет є ключовим елементом в роботі згорткових нейронних мереж. Це набір даних, який використовується для навчання моделі. Важливість датасету полягає в тому, що він має відображати реальні умови, в яких буде використовуватися модель. Наприклад, якщо ми бажаємо створити згорткову нейронну мережу для класифікації зображень, то датасет має містити достатньо різноманітних зображень з різними класами об'єктів. Якщо датасет буде надто обмеженим або не репрезентативним для реальних умов, то модель не буде давати гарних результатів на нових даних, які не входили до датасету. Для забезпечення якості датасету використовуються різні стратегії, такі як збільшення обсягу даних шляхом аугментації, балансування класів, перевірка на перекладання та інші. Датасети для згорткових нейронних мереж можуть мати величезний обсяг - від декількох тисяч до мільйонів зображень. Підбір правильних даних може бути складним завданням, оскільки він включає в себе не тільки вибір самого датасету, але і визначення які дані слід включати в нього. Наприклад, у випадку з класифікацією зображень, важливо визначити, які категорії треба включати, скільки зображень необхідно включити в кожну категорію, які фактори мають вплив на зображення і як їх враховувати при виборі зображень для датасету. Окрім

того, важливо збирати дані, які відповідають реальному світу, щоб мережа була придатною для застосування в реальних ситуаціях. Наприклад, якщо мережа будується для розпізнавання обличчя, важливо включати зображення з різних куточків і під різними кутами, з різними відтінками шкіри та різними світловими умовами. Крім того, важливо підбирати дані, які мають достатню кількість зразків в кожному класі. Якщо деякі класи мають занадто малу кількість зразків, це може призвести до перенавчання (overfitting) або піднавчання (underfitting) мережі.

Під час тренування згорткової нейронної мережі, спочатку ініціалізуються випадкові ваги для кожного шару. Далі, для кожного зображення з навчального набору, здійснюється пряме поширення (forward pass), під час якого зображення проходить через кожен шар мережі, а вихідний результат порівнюється з відомими правильними мітками (ground truth). Після прямого поширення виконується зворотнє поширення (backpropagation), під час якого похибки визначаються і передаються назад через мережу. Кожен шар рахує відповідну похибку та коригує свої ваги, щоб зменшити цю похибку. Цей процес повторюється для кожного зображення в тренувальному наборі протягом певної кількості епох, після чого мережа може бути використана для передбачення міток для нових зображень. Процес навчання на повнозв'язних шарах згорткової нейронної мережі проходить так само, як і в класичних нейронних мережах, в свою чергу на згортковомих шарах роль ваг виконують фільтри, тому навчання стає більш складним. Одна з основних причин полягає в тому, що фільтри не використовуються для обчислення відповіді для всіх можливих вхідних значень, а лише для окремих областей вхідного зображення. Окрім того, фільтри здійснюють локальні перетворення зображень, що робить їх набагато менш стійкими до зміщень та інших геометричних перетворень зображення. Іншою причиною є кількість параметрів, які треба навчити в кожному фільтрі. Якщо у звичайному шарі нейронної мережі кожен

нейрон має вагу для кожного вхідного сигналу, то фільтр має значно меншу кількість параметрів. Наприклад, якщо фільтр має розмірність 3x3 та працює з кольоровими зображеннями, то кількість його параметрів буде 27 (3x3x3), що може бути досить значним навіть для маленької згорткової мережі.

2.2. Особливості виділення обличчя у відеопотоці

Виділення обличчя у відеопотоці є важливою задачею в багатьох застосуваннях, наприклад, в системах відеоспостереження, рекламі та розвагах. Однак, ця задача досить складна, оскільки відеопотік зазвичай містить багато різних об'єктів, які можуть маскувати обличчя людини. Тому для виділення обличчя у відеопотоці необхідно використовувати спеціальні алгоритми та методи. Одним з найпоширеніших підходів до виділення обличчя є використання згорткових нейронних мереж. Згорткові нейронні мережі можуть виявляти різноманітні об'єкти на зображеннях, зокрема обличчя, за допомогою спеціальних шарів згорток та функцій активації. Однак, для ефективного виділення обличчя у відеопотоці необхідно вирішити такі проблеми, як висока швидкість обробки та точність детектування. Одним із підходів до вирішення цих проблем є використання каскадних класифікаторів, які складаються з декількох послідовно підключених класифікаторів. Кожен класифікатор має свої ваги та порогове значення для прийняття рішення про наявність обличчя на зображенні. Перший класифікатор може бути досить простим та швидким, наприклад, використовувати метод гаусівської моделі шкіри. Далі можуть використовуватися більш складні класифікатори, які виявляють більш детальні ознаки обличчя.

Найвідомішим алгоритмом, який базується на використанні каскадних класифікаторів є алгоритм Віола-Джонса (Viola-Jones). Першим

етапом роботи алгоритма є навчання класифікаторів. На цьому етапі використовуються позитивні (зображення з обличчями) та негативні (зображення без облич) приклади, які використовуються для тренування класифікаторів на основі методу Адабуста (AdaBoost). На виході отримуються набори признаків, що вибираються на основі інформації про зміну яскравості пікселів. Наступний етап – створення каскаду класифікаторів. Кожен класифікатор працює з різними наборами HF, які складаються з прямокутних областей зображення, що містять різні комбінації світлоти або тіні. Наприклад, прямокутний шаблон може мати вагу 1 для пікселів, що знаходяться в лівій верхній частині шаблону, -1 для пікселів, що знаходяться в правій верхній частині, 1 для пікселів, що знаходяться в лівій нижній частині і -1 для пікселів, що знаходяться в правій нижній частині. Цей шаблон може виявляти різницю у світлоті між лівою та правою частинами обличчя. Інші шаблони можуть мати вагу 1 для пікселів, що знаходяться в верхній частині області і -1 для пікселів, що знаходяться в нижній частині. Цей шаблон може виявляти різницю у світлоті між верхньою та нижньою частинами обличчя. Інші приклади шаблонів можуть включати діагональні зміщення, прямокутні зміщення та інші комбінації ваг. Гаарові ознаки можуть бути використані для швидкого оброблення великої кількості зображень. Для виявлення обличчя образ спочатку пропускається через перший класифікатор, який швидко знімає більшість необличчевих областей, позбавляючи їх віддаленої перевірки. Якщо зображення проходить перший класифікатор, воно потрапляє до наступного, більш складного класифікатора, який видає результат із використанням більш детальних признаків. Якщо на цьому етапі образ не виявляється як обличчя, він відкидається і не проходить до наступних класифікаторів. Останній етап – виявлення обличчя на зображенні. Вся область зображення аналізується набором класифікаторів в каскаді. Якщо на кожному етапі класифікатор позначає зображення як обличчя, то образ

відправляється на наступний етап. Якщо на будь-якому етапі зображення не виявляється як обличчя, то воно відкидається. Алгоритм Віола-Джонса може бути ефективним для визначення облич у відеопотоці, оскільки він може обробляти кадри з високою швидкістю. Однак, він може мати проблеми з точністю, особливо якщо обличчя знаходиться в несподіваних місцях.

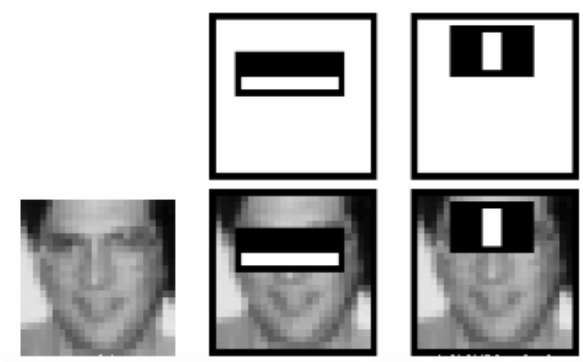


Рисунок 2.3 – Накладання гаарової ознаки на обличчя

Результати виділення обличчя можуть бути різними залежно від задачі та від способу виділення обличчя. Обмежуюча рамка – це прямокутник, який обмежує об'єкт на зображенні, в даному випадку, обличчя. Цей прямокутник складається з чотирьох точок, які вказують на координати верхнього лівого та нижнього правого кутів обмежувального прямокутника. Обмежувальна рамка зазвичай використовується для відокремлення обличчя від решти зображення та для подальшої обробки обличчя на зображенні. Відокремлення обличчя може бути важливим для таких задач, як розпізнавання облич, визначення емоцій або визначення різних параметрів обличчя.. Ключові точки обличчя – це особливі точки на обличчі людини, які використовуються для аналізу та обробки зображень обличчя. Ці точки зазвичай включають очі, ніс, рот, щоки та брови. Ключові точки обличчя можуть бути використані для різних завдань, таких як розпізнавання облич, визначення емоцій, вікового діапазону, статі та інших характеристик обличчя.

2.3. Нейромережева модель для виділення обличчя у відеопотоці

Незважаючи на те, що каскадні класифікатори справляються зі своєю задачею, але за деяких умов, вони видають хибні результати. На заміну класифікаторам приходять нейромережеві моделі, які є більш точними, швидкими та гнучкими. Нейромережеві моделі для виділення обличчя зображення зазвичай базуються на DL і використовуються для автоматичної обробки зображень, включаючи визначення розташування та контуру обличчя. Одна з найпоширеніших нейромережевих моделей для виділення обличчя – це згорткова нейронна мережа, що складається зі спеціальних шарів, що забезпечують розпізнавання зображень. Ці шари навчаються на великих наборах зображень, що містять обличчя, і здатні визначати ключові особливості обличчя, такі як очі, ніс та рот. Інші нейромережеві моделі, такі як Multi-task Cascaded Convolutional Networks (MTCNN) та Single Shot MultiBox Detector (SSD), використовуються для виявлення обличчя та розташування ключових точок обличчя, таких як очі, ніс та рот. Ці моделі також навчаються на великих наборах даних та можуть бути використані для виявлення обличчя у різних умовах освітлення, поз та масштабів зображення.

Використання нейромереж для виділення обличчя в відеопотоці має багато переваг. Висока точність: нейромережеві моделі для виділення обличчя в відеопотоці мають високу точність у порівнянні з іншими методами, такими як методи на основі кольору чи текстури зображення. Це дозволяє точніше виділяти обличчя та його ключові точки навіть в умовах низького контрасту чи затінення. Швидкість обробки: Завдяки використанню спеціалізованих архітектур нейромереж та глибокого навчання, моделі для виділення обличчя відносно швидко обробляють відеопотік у реальному часі. Автоматична адаптація до змін: нейромережеві моделі для виділення обличчя зображень мають здатність автоматично адаптуватись до змін в зображенні, таких як зміна освітлення,

кольору шкіри, а також до зміни положення обличчя у відеопотоці. Можливість виявлення багатьох облич: нейромережеві моделі для виділення обличчя відносно легко можуть виявляти багато облич на одному зображенні, що дозволяє використовувати їх для розпізнавання облич у великих групах людей, таких як на фестивалях, конференціях та інших подібних заходах.

Хоча використання нейромереж для виділення обличчя в відеопотоці має багато переваг, воно також має деякі недоліки. Обмежена швидкість обробки: використання нейромереж для виділення обличчя в режимі реального часу може бути складним через обмежену швидкість обробки. Навіть найкращі нейромережеві моделі можуть вимагати багато обчислювальних ресурсів та часу для обробки відеопотоку високої якості. Залежність від якості вхідних даних: нейромережеві моделі для виділення обличчя можуть давати невірні результати, якщо вхідні дані низької якості, наприклад, якщо обличчя частково приховане волоссям або на відео є багато шуму. Потребує попередньої підготовки даних: для навчання нейромережевих моделей необхідно мати достатню кількість позначених даних. Це може бути проблемою, якщо немає достатньої кількості позначених відео для навчання моделі. Ризик помилок: нейромережеві моделі можуть допускати помилки, особливо якщо дані з якими вони працюють не відповідають вимогам. Це може привести до невірного виділення обличчя на відео або навіть до втрати даних. Ризик безпеки: використання нейромережевих моделей може стати проблемою з точки зору безпеки, оскільки такі моделі можуть бути піддані атакам з метою зламу або перехоплення даних. Це особливо важливо при використанні нейромережевих моделей в областях, які стосуються безпеки.

Однією з найкращих моделей для виділення обличчя є MTCNN, яка базується одночасно на принципах звичайної згорткової нейронної мережі і має каскадну архітектуру. Кожен блок каскаду виконує свою задачу -

визначення області, яка містить обличчя, локалізація ключових точок на обличчі та визначення орієнтації обличчя. Кожен блок отримує на вхід зображення, яке попередньо піддається обробці та розміщується на відповідних шарах мережі. Однією з переваг MTCNN є те, що вона здатна працювати з зображеннями та відео, які мають різні умови освітлення, орієнтації та зміщення облич. Крім того, MTCNN може ефективно визначати обличчя у великих обсягах даних, що робить її корисною для застосування в різних сферах, таких як медицина, безпека та розваги. Ще одна перевага MTCNN полягає в тому, що вона може виявляти обличчя з різними формами та розмірами, що дозволяє використовувати її в різних умовах та завданнях. Отже, MTCNN - це потужна модель для виділення облич, яка здатна працювати з великими обсягами даних та різними умовами зйомки, що робить її ідеальною для максимально швидкого та точного аналізу відеопотока.

2.4. Висновки до другого розділу

Було розглянуто різні аспекти виділення обличчя на зображеннях та відеопотоках. Починаючи з традиційних методів, таких як каскадні класифікатори, і переходячи до більш сучасних підходів, зокрема нейромережевих моделей, стає зрозумілим, що виділення обличчя є важливою задачею в обробці зображень та має широкі застосування в різних галузях. Традиційні методи, такі як каскадні класифікатори, довгий час є ефективними засобами для виділення обличчя. Вони базуються на простих признаках та алгоритмах, що дозволяють швидко знаходити обличчя на зображенні. Однак такі методи мають свої обмеження. Вони можуть мати проблеми з точністю, особливо в умовах зміни освітлення, орієнтації або зміщення обличчя. Крім того, вони можуть давати хибні результати на зображеннях низької якості або в складних сценах. Також треба підготувати ручні ознаки для кожного класифікатора, що може бути

часо- та працезатратним завданням. У сучасному машинному навчанні та глибинному навчанні набуває популярності застосування нейромережевих моделей для виділення обличчя, зокрема MTCNN (Модифікована Каскадна Нейромережева Мережа). MTCNN є потужним інструментом, оскільки вона може виявляти обличчя на зображеннях та відео з високою точністю навіть у важких умовах. Вона використовує каскадну архітектуру, де кожен блок каскаду виконує свою задачу - визначення області з обличчям, локалізація ключових точок та визначення орієнтації обличчя. MTCNN здатна працювати з великими обсягами даних та різними умовами зйомки, що робить її корисною для різних сфер, включаючи медицину, безпеку та розваги. Тому саме MTCNN буде базою для створеної нейромережі.

Датасет грає важливу роль у навчанні мережі для виділення обличчя на фото. Датасет представляє собою набір зображень, які використовуються для тренування моделі. Якість та репрезентативність датасету безпосередньо впливають на результати навчання та продуктивність моделі. Зазвичай для успішного тренування MTCNN рекомендується мати великий датасет, що містить тисячі або навіть мільйони зображень обличчя з різних джерел та умов зйомки. Оскільки MTCNN вимагає навчання на трьох послідовних блоках (визначення області з обличчям, локалізація ключових точок та визначення орієнтації обличчя), він потребує достатнього обсягу даних для кожного з цих блоків. Орієнтовно, для навчання MTCNN може бути потрібно мінімум кілька тисяч зображень обличчя. Чим більше тренувальних файлів буде використано, тим краще модель зможе усвідомлювати різні варіації та особливості обличчя. Варто пам'ятати, що окрім кількості тренувальних даних, інші фактори, такі як якість та репрезентативність датасету, архітектура мережі, параметри навчання та розмір мережі, також впливають на ефективність навчання та результати моделі.

3. МОДЕЛЮВАННЯ СИСТЕМИ ВИДІЛЕННЯ ОБЛИЧЧЯ В ВІДЕОПОТОЦІ

3.1. Проектування системи виділення обличчя в відеопотоці

Після проведеного аналізу було створено моделі даних для системи виділення обличчя в відеопотоці. Програмне забезпечення системи складається з двох пакетів, як показано на рисунку 3.1. Ці пакети можуть включати компоненти програмного забезпечення, які взаємодіють між собою для забезпечення функціональності системи виділення обличчя. Кожен пакет може містити набір класів, модулів або інших компонентів, що спеціалізуються на певній функціональній групі або завданні в рамках системи. Ця архітектурна структура допомагає організувати та керувати компонентами системи виділення обличчя, забезпечуючи її ефективну роботу та розширюваність. Моделі були розроблені з використанням мови моделювання UML.

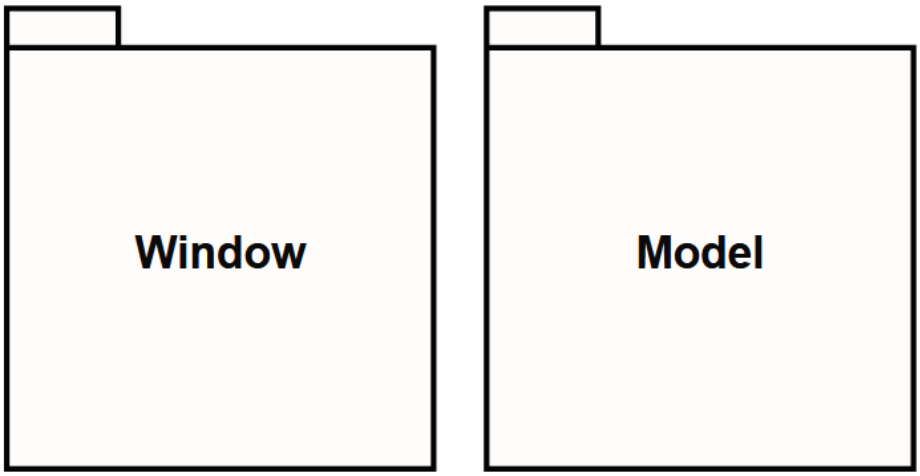


Рисунок 3.1 – UML діаграма пакетів програмного забезпечення

У пакеті Window знаходяться всі класи, які відповідають за виведення інформації на екран, використання камери, візуалізацію даних та відстеження обличчя, функціонал кнопок, завантаження даних та збереження даних. Цей пакет містить функціонал для створення та

управління вікнами програми, включаючи їх розміщення, розмір, заголовок та інші властивості.

Пакет Window складається з таких класів:

1. WindowView
2. CameraCapture
3. TrackingBox
4. ImageContainer

Класс WindowView (рис 3.2) необхідний для взаємодії користувача з інтерфейсом програми та функціоналом Model.

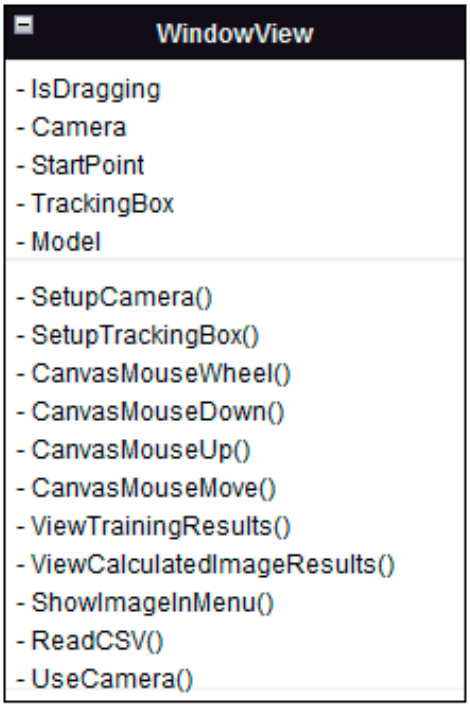


Рисунок 3.2 – UML діаграма класу WindowView

Розглянемо детальніше атрибути класу WindowView.

1. IsDragging – булева змінна, визначає чи зафіксоване активне зображення користувачем.
2. Camera – екземпляр класу для взаємодії з камерою.
3. StartPoint – визначає де на полотні знаходиться активне зображення до фіксування.
4. TrackingBox – екземпляр класу TrackingBox для візуалізації відстеження обличчя.

5. Model – екземпляр класу MainModel для взаємодії з штучною мережею та даними.

Розглянемо детальніше методи класу WindowView.

1. SetupCamera() – метод, який встановлює базові дані та підключає доступну камеру.
2. SetupTrackingBox() – метод, який встановлює базові дані для візуалізації відстеження обличчя.
3. CanvasMouseWheel() – метод, який викликається при прокруті колеса прокрутки миші, дозволяє змінювати масштаб активного зображення за допомогою миші.
4. CanvasMouseDown() – метод, який викликається при початку натискання правої кнопки миші, для того, щоб розпочати переміщення активного зображення.
5. CanvasMouseUp() – метод, який викликається при відтисканні правої кнопки миші, закінчує переміщення активного зображення, і робить його позицію статичною.
6. CanvasMouseMove() – метод, який викликається при натиснутій миші, рахує позицію активного зображення.
7. ViewTrainingResults() – метод, який викликає відповідний метод тренування нейронної мережі екземпляру FaceDetector, виводить на екран відповідну інформацію про процес і результати тренування нейронної мережі.
8. ViewCalculatedImageResults() – метод, який викликає відповідний метод визначення обличчя активного зображення екземпляру FaceDetector, виводить на екран відповідну інформацію про результати, які були отримані після обробки зображення нейромережевою моделю.

9. ShowImageInMenu() – метод, який додає зображення до меню, дозволяє додавати проміжні результати, які отримані з нейронної мережі.
10. ReadCSV() – метод, який дає можливість завантажити дані, які потім можуть бути використані для тренування або тестування з файлу формату csv.
11. UseCamera() – метод, який робить кадри отримані з камери активними зображеннями для обробки нейронною мережею.

Клас TrackingBox відповідає за візуалізацію отриманих результатів на екрані.

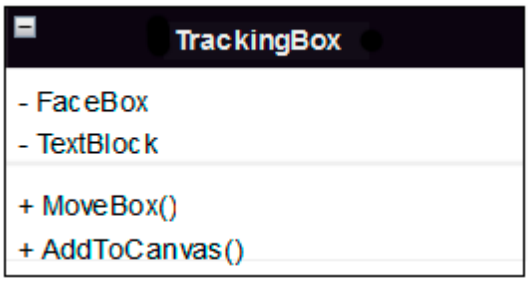


Рисунок 3.3 – UML діаграма класу TrackingBox

Розглянемо детальніше атрибути класу TrackingBox.

1. FaceBox – екземпляр типу Rectangle, містить в собі інформацію про рамку, яка повина окреслювати обличчя людини.
2. TextBlock – екземпляр типу TextBlock, показує на екрані ймовірність того, що в рамці виділенне обличчя.

Розглянемо детальніше методи класу TrackingBox.

1. MoveBox() – метод, який дозволяє змінити розташування рамки для відстеження обличчя та тексту, який відповідає значенню ймовірності того, що виділене саме обличчя.
2. AddToCanvas() – додає FaceBox та TextBlock до полотна та виводить на екран.

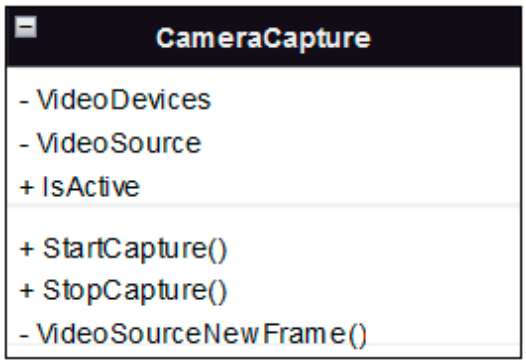


Рисунок 3.4 – UML діаграма класу CameraCapture

Розглянемо детальніше атрибути класу CameraCapture.

1. VideoDevices – екземпляр класу, який відповідає за роботу з вебкамерою.
2. VideoSource – екземпляр класу, який відповідає за роботу з відеопотоком.
3. IsActive – булева змінна, відповідає за те чи активна камера в даний момент.

Розглянемо детальніше методи класу CameraCapture.

1. StartCapture() – метод, який розпочинає запис інформації з камери.
2. StopCapture() – метод, який зупиняє роботу з камерою.
3. VideoSourceNewFrame() – метод, який викликається для обробки кожного кадру отриманого з камери.

Клас ImageContainer відповідає за візуалізацію зображень та даних для роботи з нейромережею.

Розглянемо детальніше атрибути класу ImageContainer.

1. Image – містить в собі зображення, яке буде показуватись в інтерфейсі.
2. ImageName - ім'я, яке буде показуватись в інтерфейсі.
3. Path – шлях до файлу зображення, якщо такий є.
4. DataCount – кількість даних для відображення в інтерфейсі.

5. NodeSet – екземпляр типу NodeSet, який відповідає за виведення лицевих маркерів на екран.
6. SizeCoefficient – коефіцієнт ширини зображення на полотні.
7. ContainsData – булева змінна, визначає чи має дане зображення дані.

Розглянемо детальніше методи класу ImageContainer.

1. AddImage() – додає зображення до списку.
2. InitButton() – створює кнопку для доступу до зображення та його даних в інтерфейсі.
3. AddData() – додає дані про зображення.
4. SetActive() – робить зображення активним.

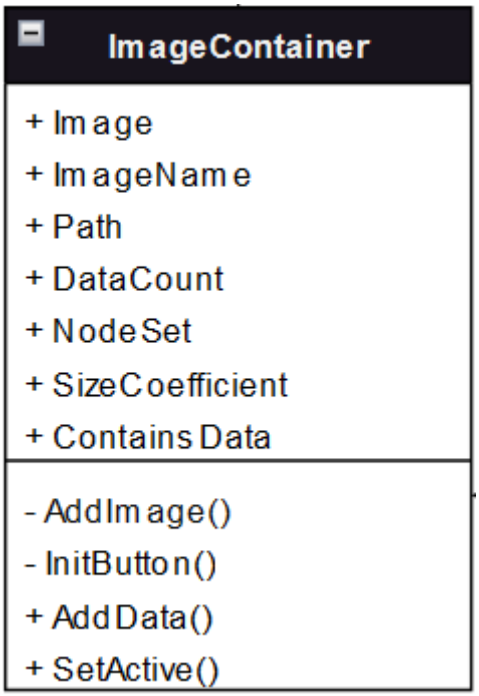


Рисунок 3.4 – UML діаграма класу ImageContainer

У пакеті Model знаходяться класи, які відповідають за взаємодію з нейромережою та обробкою даних, яка дозволяє подати зображення на обробку нейромережевою моделю. Класи в пакеті дозволяють керувати архітектурою нейромережі, включаючи кількість шарів, розмір шарів, функції активації та зв'язки між шарами.

Пакет Model складається з таких класів:

1. MainModel
2. FaceDetector
3. TrainingData
4. NetworkModel

Клас MainModel відповідає за завантаження та підготовку даних для обробки нейромережею.

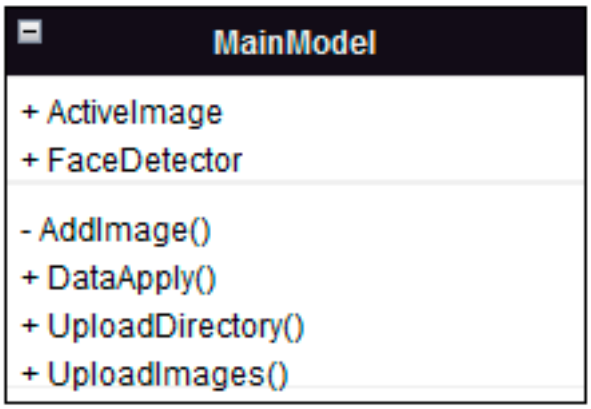


Рисунок 3.5 – UML діаграма класу MainModel

Розглянемо детальніше атрибути класу MainModel.

1. ActiveImage – екземпляр класу ImageContainer, в якому зберігаються дані про активне зображення, які потім використовуються для обробки нейронною мережею.
2. FaceDetector – екземпляр класу FaceDetector, в якому виконується робота з нейромережею.

Розглянемо детальніше методи класу MainModel.

1. AddImage() – метод, який додає нове зображення до масиву зображень, які можуть бути використані для тренування, якщо є дані, або для обробки нейромережею.
2. DataApply() – метод, який робить зображення активним.
3. UploadDirectory() – метод, який дозволяє завантажити всі зображення, які знаходяться в вибраній папці на комп'ютері.

4. UploadImages() – метод, який дозволяє завантажити вибране користувачем зображення, яке знаходиться на комп'ютері.

Клас FaceDetector є місцем взаємодії та керування нейронною мережею. Методи класу дозволяють або отримувати певні значення з нейромережі, або надавати дані на обробку.

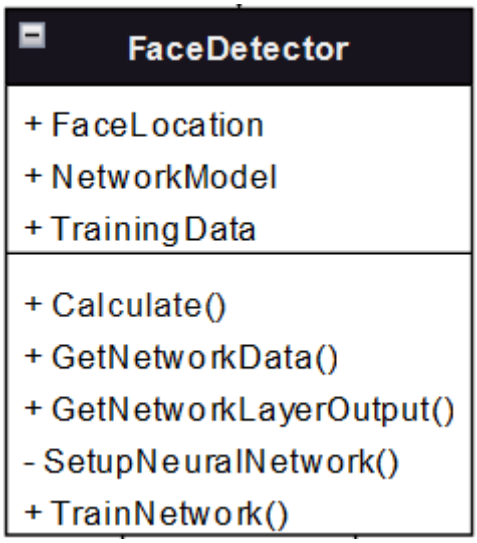


Рисунок 3.6 – UML діаграма класу FaceDetector

Розглянемо детальніше атрибути класу FaceDetector.

1. FaceLocation – містить значення розташування обличчя, яке було отримано в результаті останньої обробки.
2. NetworkModel – екземпляр класу NetworkModel, який є реалізацією нейронної мережі.
3. TrainingData – екземпляр класу TrainingData, відповідає за завантаження даних з бази даних та їх обробку.

Розглянемо детальніше методи класу FaceDetector.

1. Calculate() – приймає зображення, конвертує їх до потрібного формату, компресує та нормалізує, а потім подає його на обробку нейромережею, в результаті повертає результат обробки зображення нейромережею.
2. GetNetworkData() – повертає значення всіх вагів, для збереження поточного налаштування нейромережі.

3. GetNetworkLayerOutput() – повертає значення на виході і-го шару нейромережі.
4. SetupNeuralNetwork() – налаштовує архітектуру нейромережі, задає кількість шарів, їх структуру.
5. TrainNetwork() – приймає оброблений масив даних для тренування нейромережі, кількість епох навчання та інші параметри.

Клас TrainingData потрібен для взаємодії з базою даних та зберігання тренувальних даних.

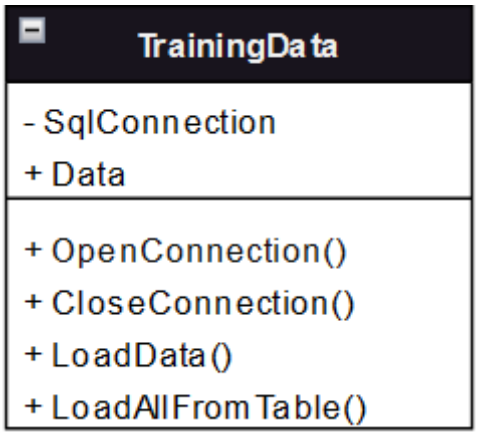


Рисунок 3.7 – UML діаграма класу TrainingData

Розглянемо детальніше атрибути класу TrainingData.

1. SqlConnection – екземпляр класу SqlConnection, потрібен для створення зв'язку з базою даних та взаємодією з нею.
2. Data – масив з структур, які містять дані для тренування нейромережі.

Розглянемо детальніше методи класу TrainingData.

1. OpenConnection – створює підключення з базою даних.
2. CloseConneciton – закриває підключення з базою даних.
3. LoadData – завантажує тренувальні дані в потрібному для обробки форматі.
4. LoadAllFromTable() – завантажує всі дані з таблиці.

Клас NetworkModel потрібен для створення нейронної мережі, її тренування, та використання.

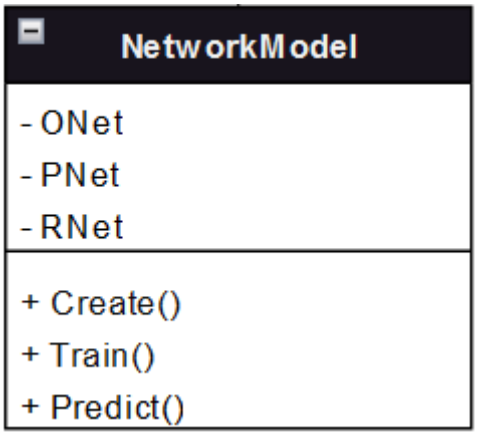


Рисунок 3.8 – UML діаграма класу NetworkModel

Розглянемо детальніше атрибути класу NetworkModel.

1. PNet – реалізація першого модулю MTCNN.
2. RNet – реалізація другого модулю MTCNN.
3. ONet – реалізація третього модулю MTCNN.

Розглянемо детальніше методи класу NetworkModel.

1. Create() – метод, який дозволяє створити нейромережу з заданою архітектурою.
2. Train() – метод, який відповідає за тренування нейромережі.
3. Predict() – метод, який відповідає за обробку зображень нейромережею.

3.2. Підготовка тренувальної вибірки перед поданням в нейромережу

Для тренування MTCNN потрібний датасет, який містить зображення облич та відповідні анотації. Ось опис ключових характеристик, які повинні бути присутні у датасеті:

1. Зображення облич: Датасет повинен містити широкий спектр зображень облич з різними властивостями, такими як розмір, освітлення,

поза, вираз обличчя та фон. Зображення можуть бути збережені у форматі JPEG або іншому підтримуваному форматі.

2. Анотації BB: Для кожного зображення облича має бути анотація, яка визначає прямокутну область, що містить обличчя на зображенні. Анотація містить координати верхнього лівого кута та нижнього правого кута bounding box. Кількість анотацій повинна співпадати з кількістю облич на зображенні.

3. Анотації орієнтації: Додатково до bounding boxes, включаються анотації, що вказують на орієнтацію обличчя. Це напрямок обличчя у просторі, такий як yaw (горизонтальний нахил), pitch (вертикальний нахил) і roll (поворот). Ці анотації допомагають моделі отримати додаткову інформацію про положення обличчя.

4. Анотації ключових точок: Інша важлива характеристика датасету - це анотації ключових точок обличчя, таких як очі, ніс, рот і т.д. Ці анотації дозволяють моделі виконувати більш детальний аналіз обличчя, наприклад, для виконання розпізнавання емоцій або відстеження рухів.

5. Розмітка негативних прикладів: Поміж зображеннями облич повинні бути й негативні приклади, які не містять обличчя. Це допомагає моделі розрізняти області з обличчями від інших областей на зображенні.

6. Розподіл класів: Для кожного зображення облича або негативного прикладу необхідно мати відповідну мітку класу. Класифікаційні мітки можуть бути, наприклад, "обличчя" і "немає обличчя".

Отже, в якості тренувальної вибірки був вибран датасет з зображеннями людських облич, який повністю підходить до вище наведених вимог, має фотографії людей під кутом, з різним освітленям та різного полу, вибірка налічує 8 тисяч зображень. В якості негативних прикладів було вибрано датасети, зображення в якому не мають в собі людських облич, а для

більшої ефективності додатково було вибрано датасети, які мають в собі зображення тварин.

Отримана вибірка хоч і є обширною, але не є придатною для ефективного навчання нейронної мережі. Розширення датасету є важливим етапом підготовки даних для тренування моделі MTCNN. Його мета полягає в збільшенні кількості доступних прикладів і поліпшенні рознообразності даних. Це дозволяє моделі краще узагальнювати і підвищує її здатність до впізнавання облич. Для розширення використовується метод випадкової обрізки - зображення випадково обрізаються до різних розмірів та пропорцій, зберігаючи при цьому обличчя в межах області обрізки. Це дозволяє отримати різні масштаби та позиції облич на зображенні. Також до зображень використовується метод зміщення – зображення зміщуються горизонтально або вертикально на випадкову величину пікселів. Це допомагає моделі вчитися розпізнавати обличчя в різних положеннях на зображенні. Зміна масштабу - зображення збільшує або зменшує масштаб з випадковими коефіцієнтами масштабування. Це дозволяє моделі працювати з обличчями різних розмірів і відстаней до камери. Також використовується метод обертання – зображення обертаються на випадковий кут. Це допомагає моделі навчатися розпізнавати обличчя в різних орієнтаціях. Аугментація кольору - зображення піддається різним кольоровим перетворенням, таким як яскравість, контрастність, насиченість та тон. Це допомагає моделі бути більш стійкою до змін в освітленні та кольорах зображень. Останнім методом розширення тренувальної вибірки є додавання шуму – шум може бути доданий до зображень, наприклад, гауссівський шум або шум соль і перець. Це допомагає моделі бути більш стійкою до шуму на реальних зображеннях. Розширення датасету за допомогою цих методів допомагає моделі навчитися розпізнавати обличчя в різних ситуаціях і забезпечує більш ефективне тренування. Варто врахувати, що при застосуванні

розширення датасету необхідно дотримуватися здорового розуму та уникати збільшення шуму або втрати якості даних. Ці методи дозволяють кратно збільшити вибірку та значно покращити гнучкість системи.

В MTCNN використовується пірамідальний підхід для розпізнавання обличчя на зображенні. Піраміда зображень є ключовою концепцією MTCNN і дозволяє виявляти обличчя різних розмірів на зображенні з варіюючою деталізацією. Основна ідея полягає в тому, що зображення розбивається на кілька масштабних рівнів або окремих зображень різних розмірів. Кожен рівень має певний масштаб та розмір зображення. Зазвичай піраміда зображень має декілька розмірних лейблів, що відповідають різним масштабам.

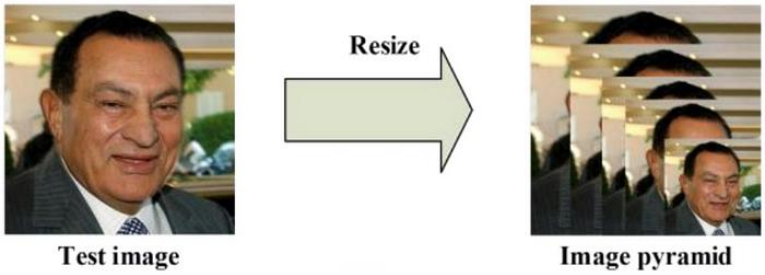


Рисунок 3.6 – Піраміда зображень для виділення облич різного розміру

3.3. Архітектура нейромережевої моделі виділення обличчя в відеопотоці

MTCNN складається з 3 модулей, окремих нейромереж. MTCNN забезпечує послідовну обробку зображень, дозволяючи виявити облича різних розмірів та забезпечуючи високу точність локалізації облич. Кожен модуль має свою власну мережу з згортковими шарами, шарами пулінгу та іншими шарами, які виконують необхідні операції для обробки зображень. Після пройдення через всі три модулі MTCNN виходи O-Net можуть бути використані для додаткової обробки, такої як відсіювання низькоімовірних областей та вибір найбільш точних прямокутників, що охоплюють облича.

P-Net (Proposal Network) є першим модулем в алгоритмі MTCNN (Multi-task Cascaded Convolutional Networks), призначеним для знаходження облич у зображеннях та генерації кандидатів на облича. Його структура складається з кількох ключових елементів. P-Net приймає зображення 12x12 трьох кольорових каналів як вхідний сигнал. Попередньо зображення буде піддане попередній обробці, нормалізації значень пікселів (3.1) та зменшенню розміру для покращення швидкості обробки.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3.1)$$

Далі, відбувається послідовний прохід через конволюційні шари. Кожен шар використовує набір фільтрів 3x3 для згортки зображення з метою вилучення різних ознак. Після згортки, для введення нелінійності використовується функція ReLU (3.2). Після першої згортки також відбувається операція MP розміру 2x2.

$$f(x) = max(0, x) \quad (3.2)$$

Одним з основних вихідних результатів P-Net є карта ознак, що відображає ймовірності присутності облич у різних регіонах зображення. Останній конволюційний шар з сигмоїдною функцією активації (3.3) використовується для отримання цієї карти.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3.3)$$

Крім того, P-Net також видає карту координат, що вказує на місцезнаходження облич у формі прямокутників. Ця карта містить відносні координати та розміри областей, які містять облича. Для відбору найбільш ймовірних кандидатів на наявність обличчя та прибирання накладених областей, застосовується алгоритм Non-Maximum Suppression (NMS). Він використовує порогове значення для відсіювання низькоімовірних областей та вибір найбільш ймовірних прямокутників, які містять обличчя.

Таким чином, на виході P-Net отримуємо відфільтровані кандидати на облича разом із їхніми ймовірностями присутності облич. Ці координати можуть бути подані наступному модулю MTCNN для подальшої обробки та точнішої локалізації облич в зображенні. P-Net генерує різноманітні кандидати на облича, що охоплюють різні розміри та масштаби, що робить алгоритм MTCNN гнучким та здатним виявляти облича незалежно від їхнього розміру чи масштабування.

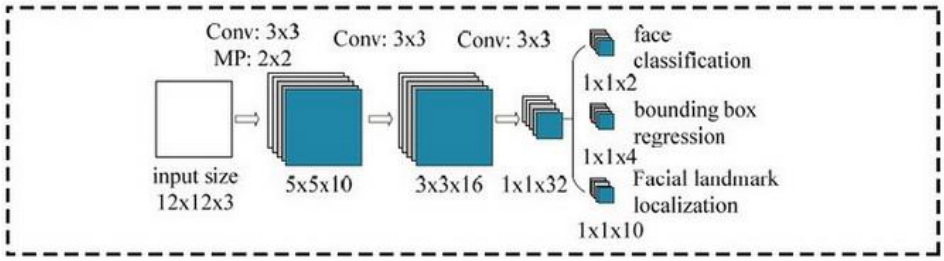


Рисунок 3.7 – Структура моделі P-Net

R-Net (Refine Network) є другим модулем в алгоритмі MTCNN. Його головна мета - покращити точність локалізації облич, отриману в попередньому модулі P-Net. R-Net дозволяє зменшити кількість ложних спрацьовувань та покращити якість виявлення облич. Має схожу структуру з попереднім модулем, але має декілька відмінних деталей. Приймає зображення 24x24x3, що дозволяє значно детальніше аналізувати надане зображення. Згортка на перших двух згорткових шарах відбувається за допомогою фільтрів 3x3 з кроком 1, після кожного з цих шарів відбуваєтья операція MP розміру 3x3. Останній згортковий шар використовує згортку 2x2. Після згорткових шарів вібувається операція вирівнення. R-Net покращує результати, отримані від P-Net, шляхом збільшення точності локалізації та відсіювання ложних спрацьовувань. Використання більш складних шарів та архітектури дозволяє отримати більш точні результати та покращити продуктивність алгоритму MTCNN в знаходжені облич.

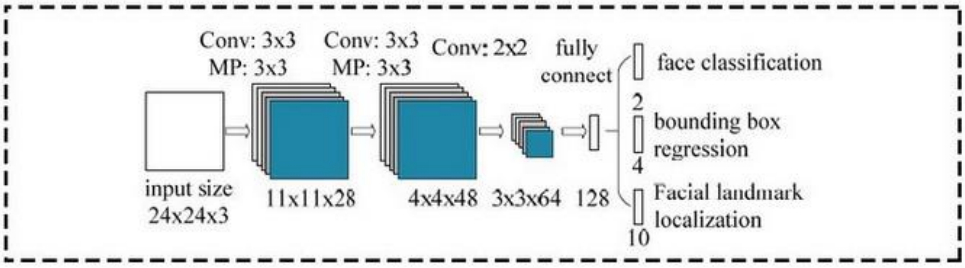


Рисунок 3.8 – Структура моделі R-Net

O-Net (Output Network) є третім та останнім модулем в алгоритмі MTCNN. O-Net відповідає за додаткову покращену локалізацію та класифікацію облич, що були виявлені в попередніх модулях P-Net і R-Net. Основна мета O-Net - забезпечити надійні результати виявлення облич з високою точністю та мінімальною кількістю ложних спрацьовувань. Приймає на вхід зображення 48x48x3, має схожу на модель R-Net структуру. Єдина відмінність, крім розмірності, є те, що після другуго шару згортки додається ще один додатковий шар згортки 3x3 з операцією максимального пулінгу 2x2 на виході. O-Net використовується для додаткової фільтрації та покращення результатів, отриманих від попередніх модулів. Він дозволяє отримати більш точні та надійні результати виявлення облич та може бути використаний для подальшої обробки, такої як вибір найбільш точних прямокутників, що охоплюють облича, та відсіювання ложних спрацьовувань.

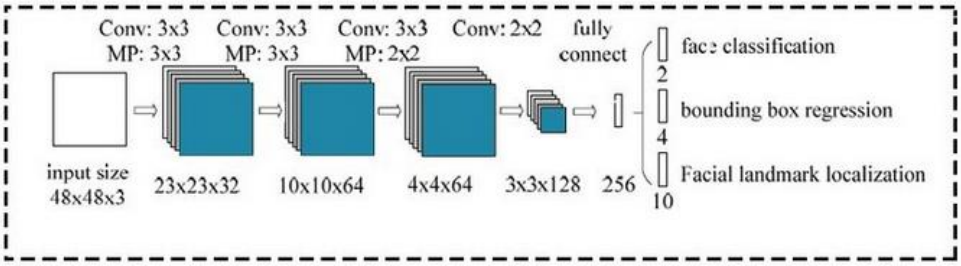


Рисунок 3.9 – Структура моделі O-Net

4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Опис інструментарію

Для розробки програмного забезпечення було вибрано мову програмування C#. C# – це мова програмування, розроблена компанією Microsoft, яка вперше була випущена в 2000 році. Вона базується на мові C++ і отримала широку популярність серед розробників завдяки своїм перевагам та потужним можливостям. Однією з ключових переваг C# є простота її вивчення. Синтаксис мови є лаконічним та зрозумілим, що робить процес вивчення менш складним для новачків у програмуванні. Це особливо важливо для розробників, які вже мають досвід у мовах програмування, таких як C, C++ або Java. Ще однією перевагою C# є її інтеграція з платформою .NET (фреймворком розробки програмного забезпечення, розробленим Microsoft). Це дає розробникам доступ до великого набору класів, бібліотек та інструментів, які спрощують процес створення різноманітних програмних продуктів. Крім того, .NET забезпечує кросплатформенну сумісність, що означає, що програми, написані на C#, можуть працювати на різних операційних системах, таких як Windows, macOS та Linux. C# також пропонує строгу типізацію, що означає, що змінні мають строго визначений тип даних, який дозволяє виявляти помилки на ранніх етапах розробки. Це допомагає забезпечити більшу надійність та стабільність програм. Іншою важливою характеристикою C# є підтримка об'єктно-орієнтованого програмування (ООП). Вона дозволяє розробникам створювати класи, об'єкти та використовувати успадкування, поліморфізм та інші концепції ООП для створення ефективного та модульного коду. C# також має велику кількість сторонніх бібліотек та фреймворків, що значно полегшують розробку програмного забезпечення. Наприклад, ASP.NET дозволяє розробляти веб-

додатки, Windows Forms - десктопні програми, а Xamarin - мобільні додатки для різних платформ. Крім того, C# має потужну систему керування пам'яттю, відому як "сборка сміття". Це означає, що розробнику не потрібно вручну виконувати операції з виділенням та звільненням пам'яті, оскільки .NET Framework самостійно виконує ці операції, що спрощує життя розробника та зменшує ймовірність помилок, пов'язаних з управлінням пам'яттю. Загалом, C# є потужною та простою у використанні мовою програмування, яка надає широкі можливості для розробки різноманітних програмних продуктів, починаючи від десктопних додатків та веб-сайтів до мобільних додаткі. Її поширена популярність, підтримка .NET Framework та велика спільнота розробників роблять C# гарним варіантом для використання.

C# є відмінним варіантом для роботи з нейромережами завдяки кільком факторам. По-перше, мова C# має потужну і виразну синтаксичну структуру, що дозволяє розробникам зручно виразити складні алгоритми навчання нейромереж. Чітка структура мови сприяє зрозумілості коду і полегшує розуміння складних концепцій, пов'язаних з нейромережами. По-друге, C# має розширені бібліотеки та фреймворки для машинного навчання. Наприклад, Keras.NET є потужним інструментом, розробленим компанією Microsoft, який надає розробникам легкий доступ до алгоритмів навчання машинного навчання, включаючи нейромережі. Цей фреймворк дозволяє швидко і зручно побудувати, навчити та використовувати нейромережі для вирішення різноманітних задач. Крім того, C# інтегрується з .NET Framework, що дозволяє розробникам легко використовувати інші бібліотеки та інструменти, які вже наявні в .NET екосистемі. Наприклад, TensorFlow.NET - це бібліотека, яка забезпечує зручний доступ до фреймворка TensorFlow для навчання нейромереж. Таким чином, розробники можуть використовувати всі переваги та функціонал TensorFlow, одночасно працюючи в середовищі C#. Отже,

завдяки своїй синтаксичній структурі, розширеним бібліотекам для машинного навчання та інтеграції з .NET Framework, C# стає гарним вибором для розробки та роботи з нейромережами. Ця мова програмування надає зручність, продуктивність та доступність, що робить її привабливим інструментом для дослідників та розробників у галузі нейронних мереж.

Окрім Keras.NET, мова програмування C# підтримує багато бібліотек для роботи з нейромережами, такі як: Accord.NET, TensorFlow.NET, CNTK, Keras.NET, Caffe.NET. Кожна з цих бібліотек має свої переваги, але була вибрана саме Keras.NET через свою простоту, гнучкість та ефективність. Keras.NET має простий та зрозумілий інтерфейс програмування, що спрощує розробку та використання нейромереж. Він надає розробникам доступ до широкого спектру алгоритмів машинного навчання, включаючи нейронні мережі, і дозволяє легко налаштовувати їх параметри та архітектуру. Це дозволяє ефективно експериментувати з різними моделями нейромереж та швидко знаходити оптимальні рішення для задач. Keras.NET надає гнучкість та розширюваність. Він підтримує інтеграцію з іншими фреймворками машинного навчання, такими як TensorFlow і ONNX, що дозволяє використовувати нейронні мережі, створені в цих фреймворках, в середовищі Keras.NET. Також існує можливість використовувати власні моделі нейромереж, розроблені з використанням інших інструментів або мов програмування, і імпортувати їх в Keras.NET для подальшого використання. Keras.NET надає можливість роботи з нейромережами в різних сценаріях, включаючи класифікацію, регресію, кластеризацію та обробку текстових даних. Цей фреймворк також підтримує паралельну обробку, що дозволяє ефективно виконувати обчислення на багатоядерних системах або у розподілених обчислювальних середовищах. Крім того, Keras.NET має добре розвинену спільноту розробників та документацію, що полегшує навчання та використання фреймворку. Розробники можуть знайти велику кількість

прикладів, довідників та ресурсів, які допоможуть їм зрозуміти основи роботи з нейромережами та використовувати Keras.NET для своїх проектів. Отже, Keras.NET є найкращим варіантом для роботи з нейромережами завдяки своєму зручному інтерфейсу, гнучкості, розширюваності та підтримці різних сценаріїв машинного навчання. Використання цього фреймворку дозволить розробити та використовувати нейромережу з заданою задачею.

4.2. Опис інтерфейсу

Для створення інтерфейсу програми було використано WPF. Windows Presentation Foundation є технологією розробки програмного забезпечення, яка входить до складу фреймворку .NET, розробленого компанією Microsoft. Вона надає потужні інструменти для створення графічних інтерфейсів користувача (GUI) для Windows-платформи. WPF дозволяє створювати багатофункціональні, візуально привабливі інтерфейси, що поєднують графіку, мультимедіа, анімацію та взаємодію з користувачем. Він пропонує модель програмування на основі XAML (eXtensible Application Markup Language), яка дозволяє окремо визначати вигляд і поведінку елементів інтерфейсу. XAML є декларативною мовою розмітки, яка дозволяє легко визначати ієрархію елементів, їх властивості та зв'язки. Цей фрейморк є більш зручним для користування, ніж Windows Forms, а також підтримується Microsoft.

Створена програма має зручний інтерфейс для роботи з нейронною мережею. Інтерфейс програми логічно поділений на чотири частини: меню інструментів, меню вибору зображень, вікно виводу повідомлень та область візуалізації отриманих результатів. Візуалізація результатів відбувається таким чином, що вибране зображення або відеопотік з камери транслюється на центрально-праву частину екрану, а в свою чергу

результат роботи нейронної мережі, обмежуюча рамка, накладається на цю область, обмежуючи обличчя.

Рисунок 4.1 – Повний інтерфейс програми

В вікні виводу повідомлень користувач може побачити інформацію про результат своїх дій, програма записує кожен крок користувача, що сприяє аналізу отриманих з нейромережі результатів. Вікно виводу повідомлень допомогає визначити час роботи алгоритму навчання, ефективність навчання, точність розрахунку та іншу корисну інформацію.

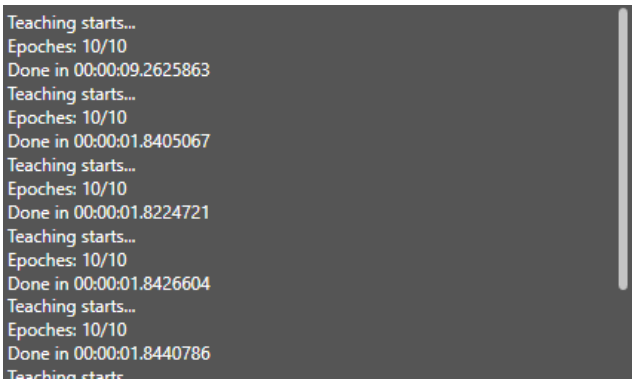


Рисунок 4.2 – Вікно виводу повідомлень

Меню вибору зображень дозволяє користувачу бачити завантажені зображення. В кожному об'єкті на панелі зберігається інформація щодо наявності даних для тренування, якщо об'єкт відмічено як тренувальний, то він буде використаний для процесу тренування нейронної мережі. Якщо

об'єкт відмічено як не тренувальний, він може бути використаний тільки для аналізу нейромережею.

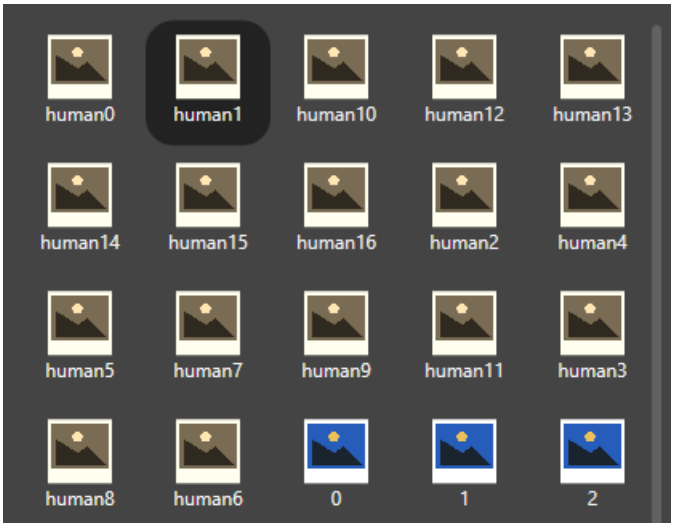


Рисунок 4.3 – Меню вибору зображень

Меню інструментів є найголовнішою частиною інтерфейсу, на ній розміщено основний функціонал, який допоможе взаємодіяти з нейромережею. Верхня частина панелі починається з координат курсора на зображені, що робить процес детального аналізу більш зручним та швидким. На верхній частині також знаходяться кнопки завантаження папки та завантаження зображення, вони дозволяють завантажувати всі зображення з папки або одне зображення з комп'ютера користувача. Кнопка даних відкриває окрему панель, на якій показуються координати контрольних точок або межі обмежуючих рамок. Кнопка камери ввмикає або відповідно вимикає камеру, якщо вона вже ключена, дозволяє нейромережі аналізувати зображення з камери, виділяти обличчя. Кнопка машинного навчання відкриває окрему панель взаємодії з нейромережею. Кнопка навчання дозволяє розпочати процес навчання, в якості даних для тренування будуть використані взі зображення з даними на панелі вибору зображень. Кнопка зберегти в файл, зберегає в файл формату .csv інформацію про налаштування нейронної мережі. Кнопка розрахунку дозволяє проаналізувати зображення на наявність обличчя, та виділити його, якщо воно присутнє. Кнопка завантаження даних дозволяє

завантажувати зображення з даними, які можуть бути використані або для аналізу, або для навчання. Також на панелі присутній вибір кількості епох навчання.

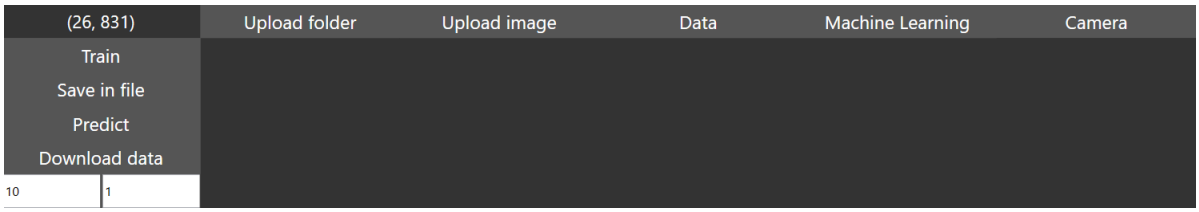


Рисунок 4.4 – Меню інструментів

4.3. Експериментальні дослідження

Для навчання нейромережевої моделі виділення обличчя в відеопотоці було використано датасет, який налічує 16000 зображень, серед яких половина зображень є зображеннями людей з різним освітленням, фоном, якістю, кутом нахилу голови. Друга половина містить в собі фотографії, на яких гарантовано немає людей. Весь датасет був перевірений для того щоб бути впевненим, що всі дані, які буде отримувати нейромережа були правильними. Результатом навчання стала працююча нейромережа, яка здата досить точно та досить швидко виділяти обличчя людей в відеопотоці.

Для верифікації отриманих результатів проведено ряд комп'ютерних експериментів. Для оцінки роботи системи виділення обличчя використано наступні параметри:

1. Точність результатів роботи системи.
2. Швидкість роботи системи.
3. Завадостійкість.

Точність нейронної мережі визначається шляхом оцінки її продуктивності на тестових даних, які вона раніше не бачила. Для оцінки точності виділення обличчя було використано декілька вибірок з тренувальними даними, які були заздалегідь підготовлені і не

використовувались у навчанні нейромережі. Кожна вибірка налічує 1000 зображень, на кожному з яких знаходиться мінімум одне людське обличчя. В результаті обробки було визначено, що точність роботи системи складає від 90% до 99% залежно від вибірки, найбільше проблем виникає при роботі з зображенями з поганою якістю та поганим освітленням. Кажучи про тренувальні дані, точність виділення обличчя досягає 99% вже після першого десятку епох тренування. Для розрахунку точності було використано метрику IoU, яка дозволяє максимально точно розрахувати точність в задачах виділення об'єктів. Швидкість роботи системи є періодом, за який система здатна обробити зображення та виділити обличчя. Головним критерієм при оцінці швидкодії створеної системи є те, що нейромережа повинна працювати саме в реальному часі. Створена нейромережева система оброблює одне зображення від 40 до 50 мілісекунд, що дозволяє їй аналізувати близько 20 кадрів за секунду на процесорі середньої потужності. На більш старіших та слабіших процесорах мережа також показує результат, який влаштовує основні критерії. Завадостійкість нейронної мережі відноситься до властивості моделі, що вона здатна показувати стійкі та надійні результати навіть в умовах наявності завад або змін у вхідних даних. Ймовірність того, що нейромережа видасть якійсь інший об'єкт за людське обличчя хоч і не є значною, але все-таки має місце, як і в будь-якій іншій моделі. Найбільш вразливою модель є до зображень тварин, які ззовні схожі на людину, наприклад, мавп, через велику кількість спільних ознак.

ВИСНОВКИ

В результаті виконання дипломного проєкту було створено модуль нейромережевого виділення обличчя у відеопотоці. Базовою архітектурою для нейромережі є MTCNN, яка в свою чергу складається з трьох модулей. Підхід до машинного навчання моделі базується методі навчання з учителем. Мережа потребує багато ретельно підготовлених та відфільтрованих даних для правильного навчання, а неправильний датасет може погіршити точність результатів.

Для взаємодії з нейромережою було створено десктопний додаток з зручним та зрозумілим інтерфейсом. Додаток пропонує можливість навчання та використання нейромережі з ефективною архітектурою для відстеження не тільки людських облич, а й інших будь-яких об'єктів, якщо буде наданий датасет з іншим об'єктом виділення.

Для вирішення поставлених завдань було використано мову програмування C#, фреймворк для створення десктопних додатків WPF, фреймворк для створення і роботи з нейромережами Keras.NET.

Створена модель продемонструвала ефективність, дозволяючи зі значною точністю, 96%, виділити обличчя людини. Швидкість роботи влаштовує головну задачу – здатність ефективно працювати в відеопотоці, демонструючи швидкість обробки – 20 кадрів за секунду.

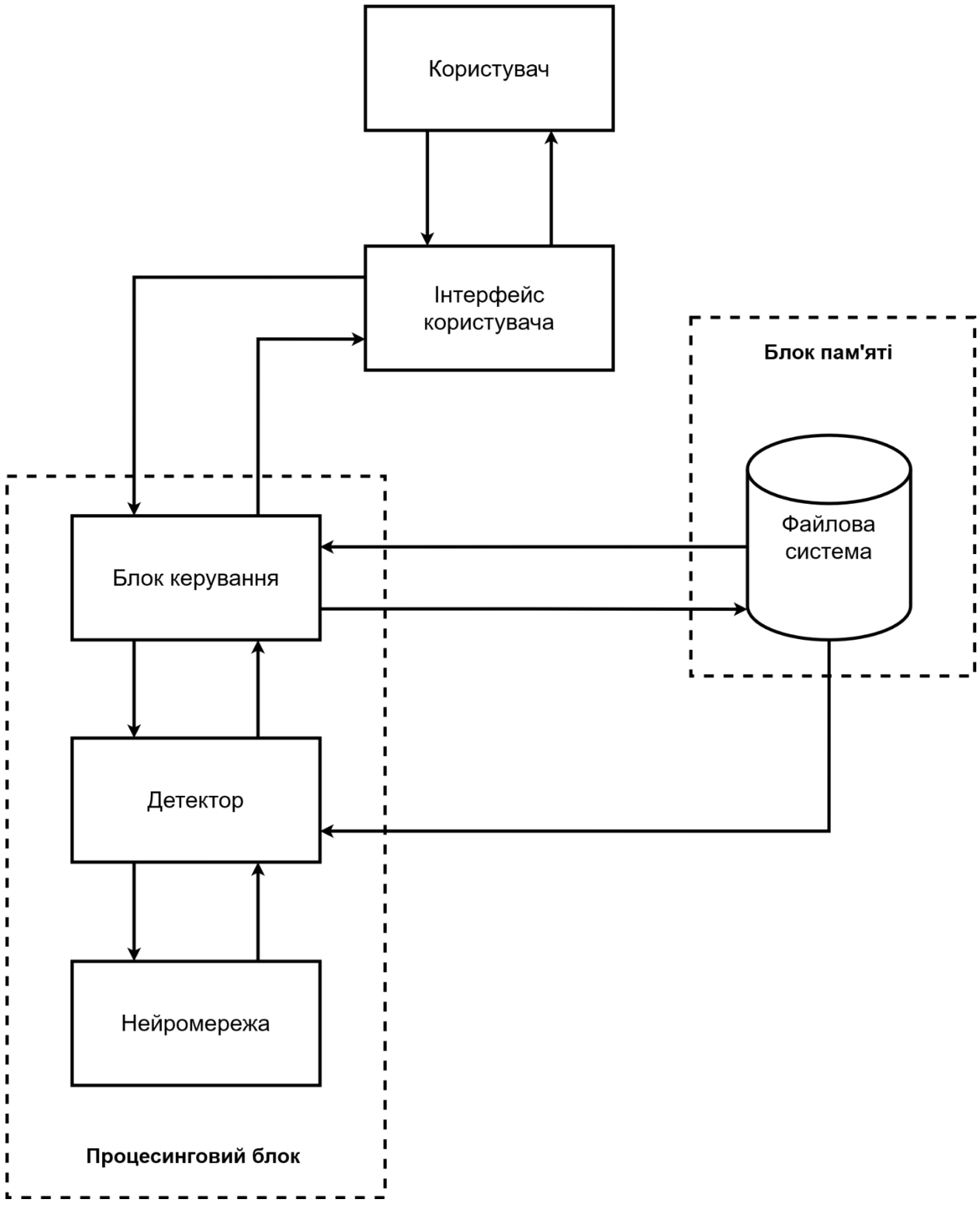
Створена архітектура може бути використана як частина іншого модулю, наприклад для розпізнання емоцій людей на основі їхнього виразу обличчя, для оцінки настрою клієнтів, вимірювання емоційної реакції на рекламу або для покращення взаємодії з користувачами в додатках та системах. Модуль може використовуватись для ідентифікації осіб на контрольованих територіях, наприклад, у в'їзних воротах, на пунктах доступу до об'єктів або в офісних приміщеннях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

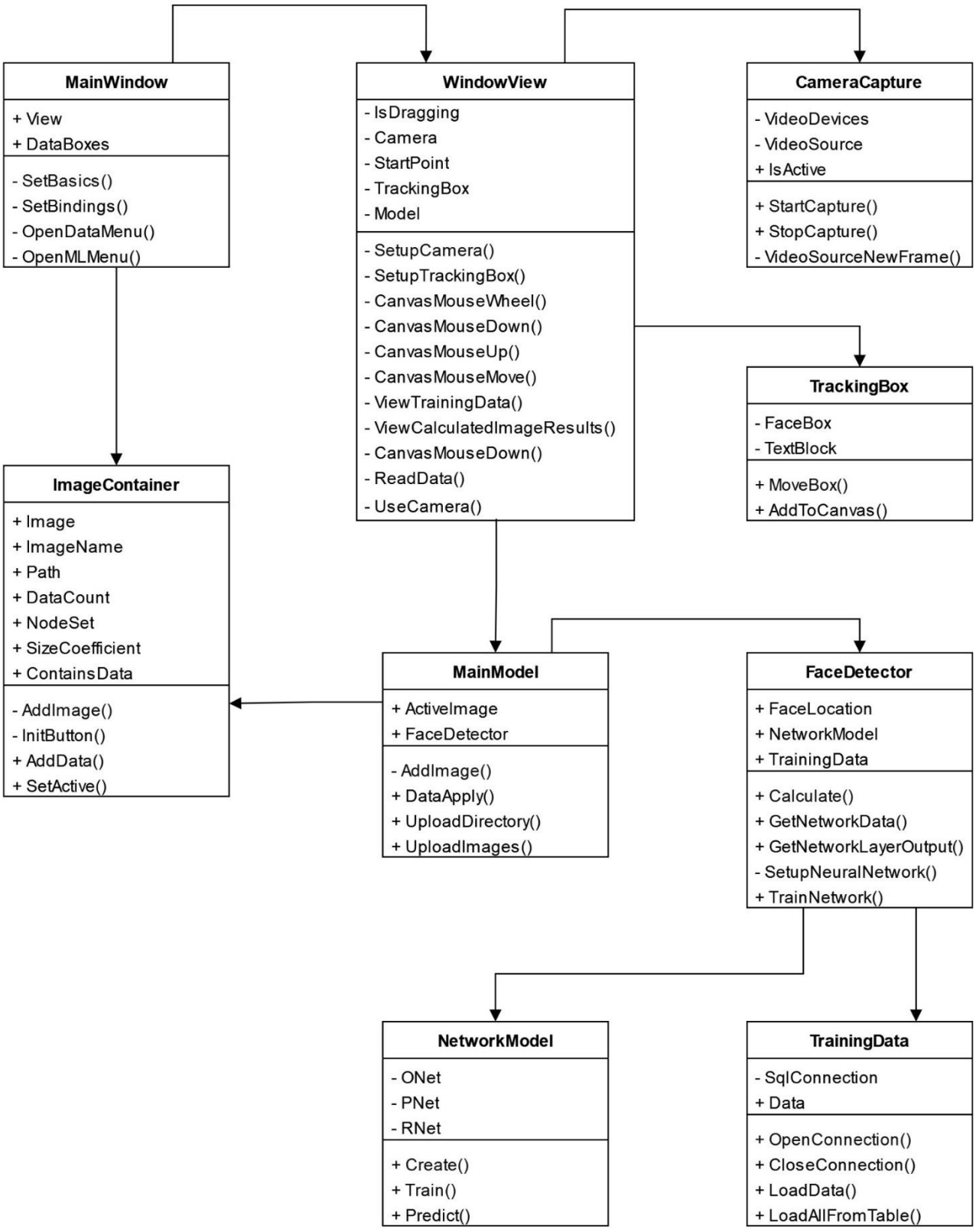
1. Chang K.K., Chang L.-L., Bowyer K. W. Face detection and recognition: A comparative study // Machine Vision and Applications. — 2003. — Vol. 14, Issue 1. — C. 1-18.
2. Turk M. A., Pentland A. P. Eigenfaces for recognition // Journal of Cognitive Neuroscience. — 1991. — Vol. 3, Issue 1. — C. 71-86.
3. Viola P., Jones M. J. Rapid object detection using a boosted cascade of simple features // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). — 2001. — Vol. 1. — C. 511-518.
4. Wang Z., Yan H., Zhou Z., Huang T. S. Advances in human-computer interaction: Human face detection and recognition // Proceedings of the 4th International Conference on Audio- and Video-Based Biometric Person Authentication (AVBPA). — 2003. — C. 57-64.
5. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks // Advances in Neural Information Processing Systems (NIPS). — 2012. — C. 1097-1105.
6. Blanz V., Vetter T. Face recognition based on fitting a 3D morphable model // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2003. — Vol. 25, Issue 9. — C. 1063-1074.
7. Mollahosseini A., Chan D., Mahoor M. H. Going deeper in facial expression recognition using deep neural networks // Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV). — 2016. — C. 1-10.
8. Jaiswal P., Singh V. Face recognition using convolutional neural networks: A survey // Proceedings of the 2nd International Conference on Signal Processing and Integrated Networks (SPIN). — 2015. — C. 390-394.

9. Кравчук В. В., Миняйло М. П., Чернікова І. М. Використання нейромережевих моделей у задачах обробки зображень // Науковий вісник Ужгородського університету. Серія "Прикладна математика і інформатика". — 2018. — №48. — С. 68-79.
10. Asit Kumar Datta Face Detection and Recognition/ Madhura Datta, Pradipta Kumar Banerjee // CRC Press. — 2015. — С. 83-133. [Електронний ресурс] — Режим доступу: https://www.google.com.ua/books/edition/Face_Detection_and_Recognition/oyfSCgAAQBAJ?hl=ru&gbpv=1
11. Keras: Deep learning API // [Електронний ресурс] — Режим доступу: https://keras.io/
12. Keras.Net: Api Documentation // [Електронний ресурс] — Режим доступу: https://scisharp.github.io/Keras.NET/api/

Додаток 1
Копії графічних матеріалів



					ІАЛЦ.045490.005 Д1		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розробив		Мазенко М.О			Взаємодія модулів системи. Схема структурна.	Літ.	Аркуш 1 / Аркушів 1
Перев.		Терейковський І.А.					
Н. контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-93	
Затвер.		Романкевич В.О.					



					ІАЛЦ.045490.006 Д2	
Зм.	Арк.	№ докум.	Підп.	Дата		
Розробив		Мазенко М.О			Діаграма в нотації UML. Діаграма класів.	Літ. / Аркуш / Аркушів: 1 / 1
Перев.		Терейковський І.А.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-93
Н. контр.		Клятченко Я.М.				
Затвер.		Романкевич В.О.				

					ІАЛЦ.045490.007 ДЗ	
Зм.	Арк.	№ докум.	Підп.	Дата		
Розробив		Мазенко М.О			Схема взаємодії клієнта з застосунком Схема алгоритму.	Літ. / Аркуш: 1 / Аркушів: 1
Перев.		Терейковський І.А.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-93
Н. контр.		Клятченко Я.М.				
Затвер.		Романкевич В.О.				

					ІАЛЦ.045490.008 Д4	
Зм.	Арк.	№ докум.	Підп.	Дата		
Розробив		Мазенко М.О			Алгоритм обробки зображення нейромережею. Схема алгоритму.	Літ. / Аркуш: 1 / Аркушів: 1
Перев.		Терейковський І.А.				
Н. контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-93
Затвер.		Романкевич В.О.				

Додаток 2
Лістинг програми

```
using System.Drawing;
using ProjectAI.Network;
using ProjectAI.View;

namespace ProjectAI.MainModel;

public class FaceDetector
{
    public const int MinSize = 12;
    public const int MaxSize = 96;
    public const float ValidationValue = 0.99999f;
    public const float ScaleFactor = 0.6f;
    public const int ScanStep = 5;
    public (TrackingData, TrackingData) FaceLocation { get; private set; }

    public TrainingData TrainingData;

    private NetworkModel _networkModel;

    public FaceDetector()
    {
        _networkModel = new NetworkModel();
        TrainingData = new TrainingData();
        SetupNeuralNetwork();
    }

    public void LoadData()
    {
        TrainingData.LoadData();
    }

    public void LoadModel()
    {
        _networkModel.LoadModel("PNet", _networkModel.PNet);
        _networkModel.LoadModel("RNet", _networkModel.RNet);
        _networkModel.LoadModel("ONet", _networkModel.ONet);

    }

    public void SaveData()
    {
        _networkModel.SaveModel("PNet", _networkModel.PNet);
        _networkModel.SaveModel("RNet", _networkModel.RNet);
        _networkModel.SaveModel("ONet", _networkModel.ONet);
    }

    private void SetupNeuralNetwork()
    {
    }

    public void TrainNetwork(int epoches, int subEpoches)
```

```
    {
        _networkModel.TrainPNet(TrainingData.Data, epoches);
        _networkModel.TrainRNet(TrainingData.Data, epoches);
        _networkModel.TrainONet(TrainingData.Data, epoches);

    }

    public (TrackingData, TrackingData)? CalculateImage(Data data)
    {
        var sizeP = InputData.PNetImageInputSize;
        var sizeR = InputData.RNetImageInputSize;
        var sizeO = InputData.ONetImageInputSize;
        var pyramid = CreateImagePyramid(
            ConvertForInputData(data.Image, data.Width, data.Height), ScaleFactor, MinSize);
        for (int i = 0; i < pyramid.Count; i++)
        {
            (var image, var coefficient) = pyramid[i];
            var height = image.GetLength(0);
            var width = image.GetLength(1);
            for (int y = 0; y < height - sizeP; y += ScanStep)
            {
                for (int x = 0; x < width - sizeP; x += ScanStep)
                {
                    var transformed = InputData.ConvertForPrediction(new List<float[,]>()
                    {
                        Crop(image, x, y, sizeP, sizeP)
                    }, sizeP);
                    var result = NetworkModel.Predict(transformed, _networkModel.PNet);
                    var position = FindMaxIter(result);
                    if (result[position] >= ValidationValue)
                    {
                        var keypoints = NetworkModel.Predict(transformed, _networkModel.RNet);
                        var coordinates = NetworkModel.Predict(transformed, _networkModel.ONet);
                        return TrackingData.TrackingDataFromNetwork(coordinates, sizeP, coefficient,
keypoints);
                    }
                }
            }
        }
        return null;
    }
}

using Keras;
using Keras.Models;
using Keras.Layers;
using Numpy;
using Python.Runtime;
using ProjectAI.View;

namespace ProjectAI.Network;
```

```
class NetworkModel
{
    private InputData[] _trainingData;
    public Sequential PNet { get; private set; }
    public Sequential RNet { get; private set; }
    public Sequential ONet { get; private set; }

    static NetworkModel()
    {
        string pythonDll = @"M:\Programs\Python\python37.dll";
        Environment.SetEnvironmentVariable("PYTHONNET_PYDLL", pythonDll);
        PythonEngine.Initialize();
        Keras.Keras.DisablePySysConsoleLog = true;
    }
    public NetworkModel()
    {
        _trainingData = new InputData[0];
        SetupNetwork();
    }

    private void SetupNetwork()
    {
        PNet = CreateModelPNet();
        RNet = CreateModelRNet();
        ONet = CreateModelONet();
    }

    private Sequential CreateModelPNet()
    {
        var model = new Sequential();

        model.Add(new Conv2D(
            filters: 10,
            kernel_size: new Tuple<int, int>(3, 3),
            activation: "relu",
            input_shape: new Shape(InputData.PNetImageInputSize, InputData.PNetImageInputSize,
1)));
        model.Add(new MaxPooling2D(pool_size: new Tuple<int, int>(2, 2)));
        model.Add(new Conv2D(filters: 16, kernel_size: new Tuple<int, int>(3, 3), activation:
"relu"));
        model.Add(new Conv2D(filters: 32, kernel_size: new Tuple<int, int>(3, 3), activation:
"relu"));
        model.Add(new Flatten());
        model.Add(new Dense(units: 1, activation: "sigmoid"));

        model.Compile(loss: "mean_squared_error",
            optimizer: "sgd", metrics: new string[] { "accuracy" });

        return model;
    }
```

```
    private Sequential CreateModelRNet()
    {
        var model = new Sequential();

        model.Add(new Conv2D(
            filters: 28,
            kernel_size: new Tuple<int, int>(3, 3),
            activation: "relu",
            input_shape: new Shape(InputData.RNetImageInputSize,
InputData.RNetImageInputSize, 1)));
        model.Add(new MaxPooling2D(pool_size: new Tuple<int, int>(3, 3)));
        model.Add(new Conv2D(filters: 48, kernel_size: new Tuple<int, int>(3, 3), activation:
"relu"));
        model.Add(new MaxPooling2D(pool_size: new Tuple<int, int>(3, 3)));
        model.Add(new Conv2D(filters: 64, kernel_size: new Tuple<int, int>(2, 2), activation:
"relu"));
        model.Add(new Flatten());
        model.Add(new Dense(units: 128, activation: "sigmoid"));
        model.Add(new Dense(units: InputData.KeypointsCount, activation: "sigmoid"));

        model.Compile(loss: "mean_squared_error",
            optimizer: "sgd", metrics: new string[] { "accuracy" });

        return model;
    }
    private Sequential CreateModelONet()
    {
        var model = new Sequential();

        model.Add(new Conv2D(
            filters: 32,
            kernel_size: new Tuple<int, int>(3, 3),
            activation: "relu",
            input_shape: new Shape(InputData.ONetImageInputSize,
InputData.ONetImageInputSize, 1)));
        model.Add(new MaxPooling2D(pool_size: new Tuple<int, int>(3, 3)));
        model.Add(new Conv2D(filters: 64, kernel_size: new Tuple<int, int>(3, 3), activation:
"relu"));
        model.Add(new MaxPooling2D(pool_size: new Tuple<int, int>(2, 2)));
        model.Add(new Conv2D(filters: 128, kernel_size: new Tuple<int, int>(2, 2), activation:
"relu"));
        model.Add(new Flatten());
        model.Add(new Dense(units: 256, activation: "sigmoid"));
        model.Add(new Dense(units: 4, activation: "sigmoid"));

        model.Compile(loss: "mean_squared_error",
            optimizer: "sgd", metrics: new string[] { "accuracy" });

        return model;
    }
```

```
public void TrainPNet(Data[] data, int epoches)
{
   (var x, var y, _, _) = ConvertData(data, InputData.PNetImageInputSize);

   PNet.Fit(np.array(x), np.array(y), batch_size: 2, epochs: epoches, verbose: 0);

   var score = PNet.Evaluate(np.array(x), np.array(y), verbose: 0);
   SaveModel("PNet", PNet);
}

public void TrainRNet(Data[] data, int epoches)
{
   (var x, _, var y, _) = ConvertData(data, InputData.RNetImageInputSize);

   RNet.Fit(np.array(x), np.array(y), batch_size: 2, epochs: epoches, verbose: 0);

   var score = RNet.Evaluate(np.array(x), np.array(y), verbose: 0);
   SaveModel("RNet", RNet);
}

public void TrainONet(Data[] data, int epoches)
{
   (var x, _, _, var y) = ConvertData(data, InputData.ONetImageInputSize);

   ONet.Fit(np.array(x), np.array(y), batch_size: 2, epochs: epoches, verbose: 0);

   var score = ONet.Evaluate(np.array(x), np.array(y), verbose: 0);
   SaveModel("ONet", ONet);
}

public void SaveModel(string name, Sequential network)
{
   string json = network.ToJson();
   System.IO.File.WriteAllText($"{name}.json", json);
   network.SaveWeight($"{name}.h5");
}

public void LoadModel(string name, Sequential network)
{
   network.LoadWeight($"{name}.h5");
}

private (float[,,,], float[,], float[,], float[,]) ConvertData(Data[] data, int size)
{
   _trainingData = data.Select(x => new InputData(
      ConvertForInputData(x.Image, size, size),
      x.IsFace,
      x.FacialKeypoints,
      new float[] {0, 0, InputData.ImageMaxTrainingSize, InputData.ImageMaxTrainingSize}
      )).ToArray();
```

```
        return InputData.ConvertForNetwork(_trainingData);
    }


    public static float[] Predict(float[,,] data, Sequential network)
    {
        var prediction = network.Predict(np.array(data), verbose: 0);
        return prediction.GetData<float>();
    }
}
```