МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проект

здобувача освіти денної форми навчання
РП.08.03.000.ДП

БОВИ
КАТЕРИНИ ЮРІЇВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: «**Розробка програмного забезпечення»**

Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж

Проектний матеріал складається з пояснювальної записки на 78 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Дипломник _____ (Бова К.Ю.)

Керівник _____ (Скорняков В.С.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «26» 06 2025 р. Протокол ЕК № 2

Оцінка ЕК 4/85

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"12" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Бові Катерині Юріївні*

(прізвище, ім'я, по батькові)

1. Тема проекту *Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж*

затверджена наказом по коледжу від "14" 11 202 р. № 246

2. Термін здачі закінченого проекту 16.06.2025

3. Вихідні данні до проекту *Використати згорткову нейронну мережу; Реалізувати пошук людей на фото з застосуванням технологій машинного навчання; Застосовувати відкриті бібліотеки TensorFlow, PyTorch, OpenCV; Створюваний застосунок має приймати початкове фото у форматі JPEG; Вихідний результат має бути реалізований у вигляді позначення розпізнаних обличь на фото; Реалізувати відсіювання невірно розпізнаних об'єктів та виведення відповідної інформації у застосунку*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналіз технологій побудови згорткових нейронних мереж; Вибір засобів розробки застосунку; Розробка структури застосунку і підключення необхідних програмних модулів; Розробка алгоритму пошуку; Розробка UML–діаграми класів; Реалізація застосунку обраною мовою програмування; Тестування розробленого застосунку на різних фото; Економічні розрахунки; Заходи ТБ

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Структура згорткової нейромережі; Порівняння існуючих програмних засобів для пошуку людей на зображеннях; Схема фреймворку; Архітектура бібліотек; Блок–схема алгоритму пошуку людей на фото; UML–діаграма класів; Схема зв'язку основних компонентів; Алгоритм відсіювання невірних результатів пошуку; Приклади роботи застосунку при пошуку людей на фото; Статистика роботи застосунку

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Скорняков В.С.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник *Скорняков В.С.* _____
(підпис)

Завдання прийняв до виконання *Бова К.Ю.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проектування*	13.05.2025	Виконано
2	*Аналіз технологій побудови згорткових нейронних мереж*	15.05.2025	Виконано
3	*Вибір засобів розробки застосунку*	17.05.2025	Виконано
4	*Розробка структури застосунку і підключення необхідних програмних модулів*	18.05.2025	Виконано
5	*Розробка алгоритму пошуку людей на фото*	24.05.2025	Виконано
6	*Розробка UML–діаграми класів*	26.05.2025	Виконано
7	*Реалізація застосунку мовою програмування*	29.05.2025	Виконано
8	*Тестування розробленого застосунку на різних фото*	02.05.2025	Виконано
9	*Аналіз отриманих результатів роботи застосунку*	04.05.2025	Виконано
10	*Економічні розрахунки та питання з охорони праці*	10.06.2025	Виконано
11	*Підготовка графічної частини проекту*	12.06.2025	Виконано
12	*Підготовка проекту до захисту та тестування ПЗ*	16.06.2025	Виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 03 000. 00 ДП	Дипломний проект		
А4			РП 08. 03 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 03 000. 00 ДП		
Розробив		Бова К.Ю.		18.06	Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж	Літ.	Аркуш 4 / Аркушів 78
Перевірив		Скорняков В.С.		18.06		Н Д П	
Н. контр.		Петрашова В.І.		18.06		ВСП "ОТФК ОНТУ" гр.4РП-08	
Затверд.		Кривченко Ю.В.		18.06			

ЗМІСТ

ВСТУП

Штучний інтелект (ШІ) – це розділ інформатики, що зосереджується на розробці систем, які здатні вирішувати завдання, що традиційно потребують людського інтелекту. Ці системи здатні навчатися на основі досвіду, обробляти дані, виявляти закономірності, робити висновки та приймати рішення. Одним з найперспективніших напрямів ШІ є комп'ютерний зір – здатність машин "бачити" та розуміти зображення.

Сучасні системи комп'ютерного зору використовують глибоке навчання, зокрема згорткові нейронні мережі (ЗНМ, англ. CNN). ЗНМ продемонстрували високу ефективність у задачах класифікації, сегментації та розпізнавання об'єктів на зображеннях. Розвиток ЗНМ став можливим завдяки збільшенню обчислювальної потужності, появі потужних бібліотек програмного забезпечення та наявності великих обсягів навчальних даних.

Розпізнавання облич на фотографіях є одним з ключових та найбільш практичних прикладів застосування згорткових нейронних мереж. Такі системи вже активно використовуються в сфері безпеки, для отримання доступу до пристроїв, організації фотоархівів, маркетингу та багатьох інших областях.

У цій дипломній роботі розглянуто розробку застосунку, що реалізує пошук людей на фотографіях за допомогою згорткової нейронної мережі. Реалізація виконана з використанням сучасних технологій, таких як Python, TensorFlow, OpenCV та PyQt5. Особливу увагу приділено правильному навчанню моделі, відфільтровуванню помилкових результатів та розробці зручного графічного інтерфейсу користувача.

Актуальність обраної теми визначається широким попитом на інтелектуальні системи аналізу зображень, які здатні автоматизувати рутинні процеси та підвищити ефективність роботи в багатьох сферах людської діяльності. Крім технічної реалізації, у роботі також розглянуто питання оцінки якості розпізнавання, адаптації моделі до нових даних та можливості розширення функціональності системи.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз технологій побудови згорткових нейронних мереж

Підґрунтя сучасних інтелектуальних систем – нейронні мережі. Їхній розвиток кидається в очі: прориви в алгоритмах, розширення обчислювальної спроможності, плюс оперативний доступ до значних обсягів інформації. Штучний інтелект вже широко застосовується, зокрема, в самостійних системах.

Ключові архітектури нейронних мереж охоплюють згорткові мережі (CNN), які ефективно обробляють зорові дані; рекурентні (RNN), що застосовуються для послідовностей (текст, часові ряди); також трансформери, які кардинально змінили обробку природної мови, завдяки механізмам уваги. Трансформери стали фундаментом для великих мовних моделей (BERT, GPT), що демонструють неймовірні результати в генерації тексту, перекладах та класифікації.

Для побудови нейронної мережі потрібне навчання. Глибоке навчання вимагає часу, адже навчальні дані повинні бути великими, щоб виявити ключові особливості та зв'язки. Навчальні приклади мають охоплювати різноманітні дані, щоб система могла розпізнавати різні зразки та вчитися взаємодіяти з ними. Якщо мережа навчається на одному прикладі, ваги, встановлені для нього, змінюються під час наступного навчання. Попередні приклади під час навчання просто забуваються. Отже, система повинна навчатися всьому одночасно, знаходячи найкращі ваги для всього набору прикладів.

Щоб уникнути перенавантаження під час тренувань, застосовується регуляризація. Вона включає Dropout, L1/L2–регуляризацію та нормалізацію шарів (Batch Normalization). Також існують фреймворки, які полегшують розробку нейромереж. TensorFlow, PyTorch, Keras та JAX – найвідоміші з них. Вони полегшують створення, навчання та впровадження моделей нейронних мереж.

Апаратне забезпечення задає основу роботи нейронних мереж. Графічні процесори (GPU) та спеціалізовані тензорні процесори (TPU) широко використовуються, прискорюючи обчислення для навчання складних моделей.

Розвиваються технології розподілених обчислень, що дозволяють обробляти

великі обсяги даних на серверних кластерах. Для зменшення потреб у ресурсах активно досліджують квантування (зниження точності числових значень), розрідження (видалення зайвих нейронів або зв'язків) та створення компактних архітектур (MobileNet, EfficientNet, TinyML), що оптимізовані для роботи на пристроях з обмеженими ресурсами (смартфони, вбудовані системи).

Автоматизація в розробці нейромереж розширює можливості, поєднуючи AutoML з пошуком найкращих архітектур, що відомо як NAS. Це дозволяє автоматично визначати найефективніші конфігурації моделей, адаптуючи їх до конкретних завдань. Ці стратегії демонструють високу продуктивність навіть з обмеженими даними (few–shot learning, transfer learning), зменшуючи потребу у великих обсягах розмічених даних. Актуальне впровадження енергоощадних підходів, що не тільки оптимізує споживання ресурсів, а й впливає на екологічний слід при навчанні великих моделей.

Сфера нейронних мереж перебуває у постійному розвитку: алгоритми покращуються, архітектури стають більш досконалими, а використання обчислювальних ресурсів – ефективнішим. Автоматизація процесу розробки та гнучкість до нових потреб роблять ці технології більш доступними. Ці досягнення відкривають шлях до створення передових рішень у різних галузях, від медицини до безпілотного транспорту, забезпечуючи високу точність, продуктивність та інтегруючи штучний інтелект в наше життя.

1.1.1 Принцип роботи згорткових нейронних мереж

Згорткові нейронні мережі (Convolutional Neural Networks, або скорочено CNN) – це один із найпотужніших інструментів сучасного штучного інтелекту, який знайшов широке застосування у сфері комп’ютерного зору. Вони належать до класу глибоких нейронних мереж і спеціально розроблені для обробки структурованих даних, насамперед – двовимірних зображень. Основна концепція, що лежить в основі CNN, полягає в імітації функціонування зорової кори головного мозку людини, яка здатна розпізнавати складні об’єкти, виділяючи характерні локальні особливості.

Архітектура CNN є модульною та включає кілька ключових типів шарів, які

взаємодіють між собою для побудови дедалі абстрактніших уявлень про зображення. Насамперед, згорткові шари (Convolutional Layers) – це серце згорткової мережі. У цих шарах використовуються спеціальні фільтри (або ядра згортки), які ковзають по вхідному зображенню та виконують математичну операцію згортки. Кожен фільтр виявляє конкретну ознаку, наприклад контури, кути, текстурні візерунки або інші характерні елементи. У процесі навчання мережа самостійно "вчиться" підбирати найінформативніші фільтри для розв’язання поставленого завдання. Наприклад, при розпізнаванні облич мережа може навчитися виявляти обриси деталей обличчя, тобто очей, носа чи губ.

Після згорткових операцій застосовуються пулінгові шари (Pooling Layers), метою яких є зменшення просторових розмірів карт ознак. Найпопулярнішим типом є максимальний пулінг (Max Pooling), який з кожної області вибирає максимальне значення. Це дозволяє не лише зменшити обчислювальні витрати, але й підвищити стійкість моделі до незначних змін у зображенні, таких як зміщення чи масштабування. Це означає те, що мережа зберігає найбільш значущу інформацію та стає менш чутливою до шумів на зображеннях.

У фінальних етапах мережі використовуються повнозв’язні (Fully Connected) шари, які виконують функцію класифікатора. На цьому рівні модель узагальнює знання, отримані з попередніх шарів, і приймає остаточне рішення. Щоб надати моделі здатність до моделювання складних залежностей між ознаками, використовуються функції активації, зокрема ReLU (Rectified Linear Unit), яка додає нелінійність та покращує здатність мережі до навчання. Завершується цей етап застосуванням функції softmax, яка перетворює вихід моделі на ймовірності для кожного з класів.

У прикладних задачах, таких як пошук людей на фотографіях, згорткові нейронні мережі демонструють вражаючу ефективність. Вони можуть автоматично виявляти обличчя на зображеннях, порівнювати їх з еталонними зразками у базі даних, класифікувати за статтю, віком або навіть емоційним станом. Завдяки багатошаровій структурі та здатності до самостійного виділення важливих ознак, CNN є надійною основою для реалізації інтелектуальних систем

розпізнавання образів.

1.1.2 Існуючі архітектури згорткових нейронних мереж

У розвитку комп'ютерного зору ключову роль відіграли кілька знакових архітектур згорткових нейронних мереж (CNN), кожна з яких зробила важливий внесок у вдосконалення методів розпізнавання зображень. Вони стали не просто етапами в історії глибинного навчання, а справжніми технологічними проривами, які сформували фундамент для подальших досягнень у галузі.

Однією з перших успішних моделей, яка проклала шлях сучасним CNN, була LeNet–5, розроблена у 1989 році дослідником Яном ЛеКуном. Вона створювалась для розпізнавання рукописних цифр на банківських чеках і включала у свою архітектуру (рис. 1.1) два згорткових шари, два шари субдискретизації (пулінгу) та кілька повнозв'язних шарів. Незважаючи на технічні обмеження того часу, LeNet–5 довела ефективність згорткових мереж для задач візуального аналізу та стала фундаментом для подальших інновацій.

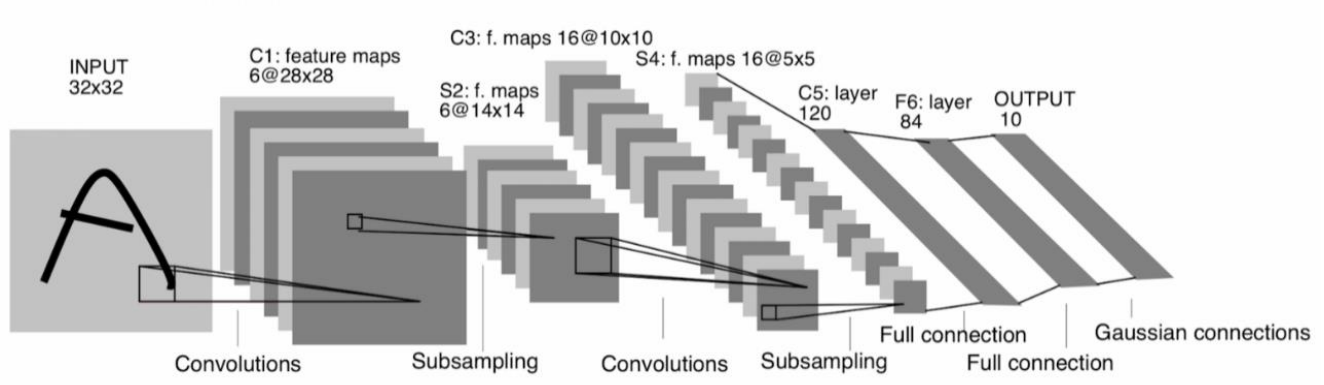


Рисунок 1.1. Архітектура LeNet

Але справжній прорив стався у 2012 році, коли модель AlexNet здобула переконливу перемогу на престижному конкурсі ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Ця модель мала вже значно глибшу архітектуру: п'ять згорткових шарів, три повнозв'язні та застосування інноваційних рішень, які на той час були революційними (рис. 1.2). Це впровадження нелінійної активації ReLU, механізму Dropout для боротьби з перенавчанням, а також використання графічних процесорів (GPU) для паралельних обчислень значно підвищили ефективність тренування глибоких мереж. AlexNet задала новий стандарт у галузі,

бо її глибоке навчання може значно перевершити традиційні підходи до аналізу зображень.

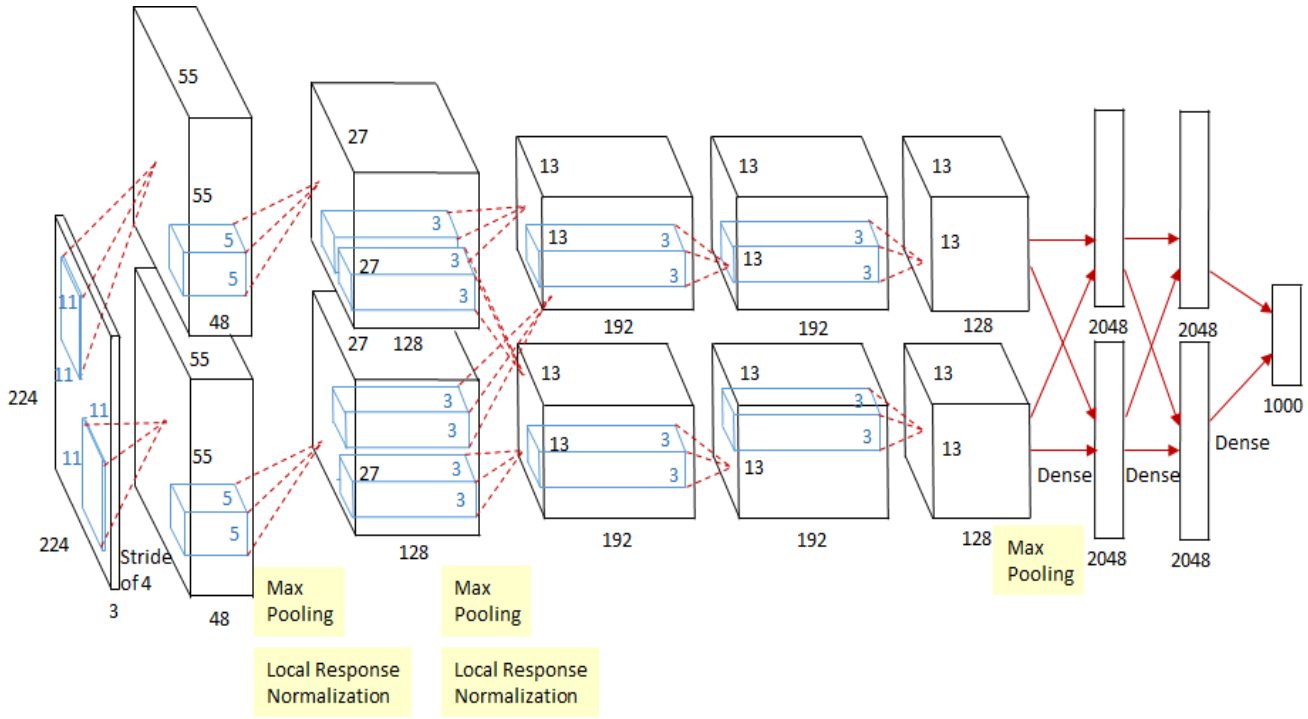


Рисунок 1.2. Архітектура AlexNet

Наступним важливим кроком стала модель Visual Geometry Group, тобто VGG, представлена у 2014 році. Від минулих моделей вона вирізнялася простою, але глибокою архітектурою (рис. 1.3), яка використовувала багато шарів (16 чи 19) з фільтрами невеликого розміру (3 на 3). Такий підхід дозволив досягти високої точності класифікації завдяки поступовому нарощуванню рівнів абстракції.

Ще одну важливу віху позначила поява ResNet (Residual Network) у 2015 році, яка вирішила критичну проблему деградації точності в дуже глибоких мережах. Основна інновація ResNet – це залишкові зв'язки (skip connections), що дозволяють інформації проходити через шари без змін. Завдяки цьому модель легко тренується навіть тоді, коли має понад 100 або навіть 1000 шарів. Це стало великим кроком уперед у побудові надглибоких нейронних мереж.У контексті задачі пошуку людей на зображеннях особливо ефективними є саме такі глибокі та стійкі архітектури, як ResNet, зокрема її модифікація ResNet–50, яка поєднує глибину з відносною обчислювальною ефективністю. Її здатність розпізнавати складні патерни візуальних даних робить її однією з провідних моделей для задач

виявлення та ідентифікації облич.

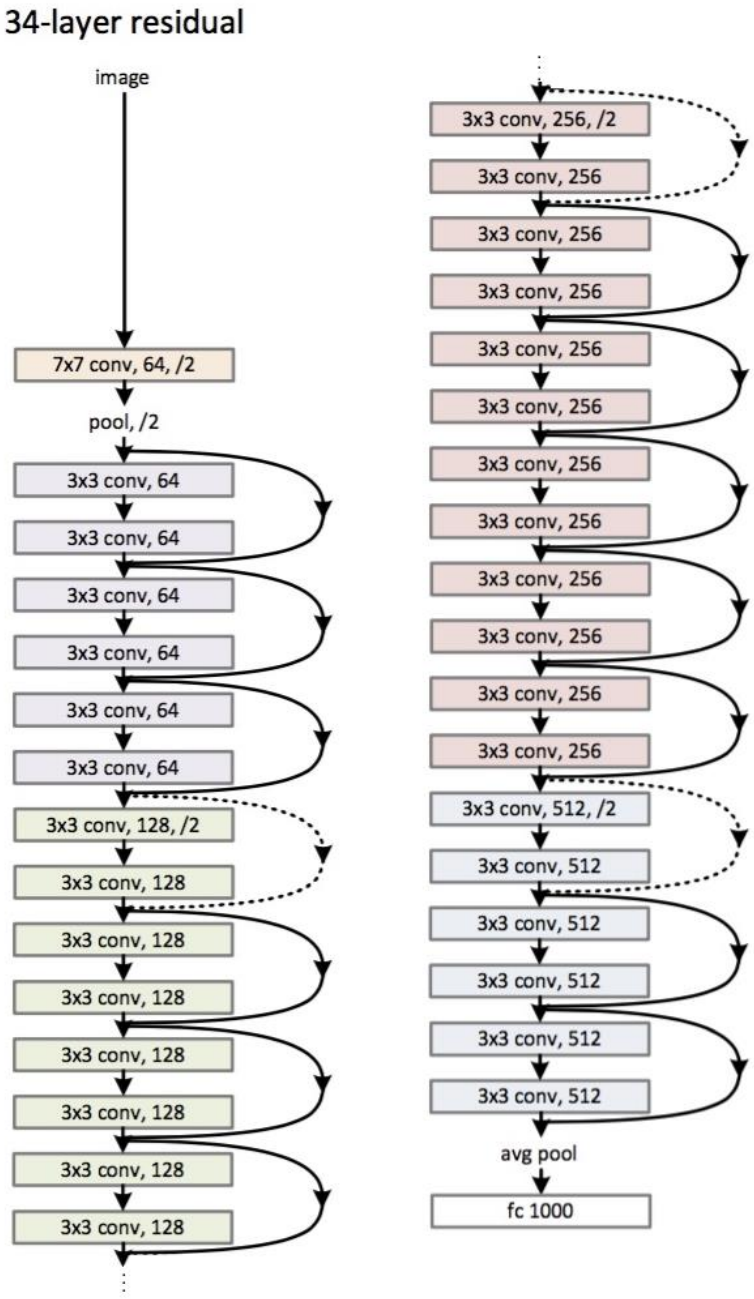


Рисунок 1.3. Архітектура ResNet

Також варто звернути увагу на інші сучасні архітектури, наприклад Inception, яка використовує багатоканальні фільтри різних розмірів в межах одного шару, або EfficientNet, що пропонує оптимізований баланс між точністю та швидкістю завдяки масштабуванню всіх параметрів моделі (глибини, ширини та роздільної здатності). Залежно від специфіки проєкту – обмежень по швидкодії, пам’яті чи точності –можна підібрати оптимальну архітектуру, яка забезпечить якісний результат у системі розпізнавання людей на фото.

1.1.3 Порівняння ефективності різних архітектур

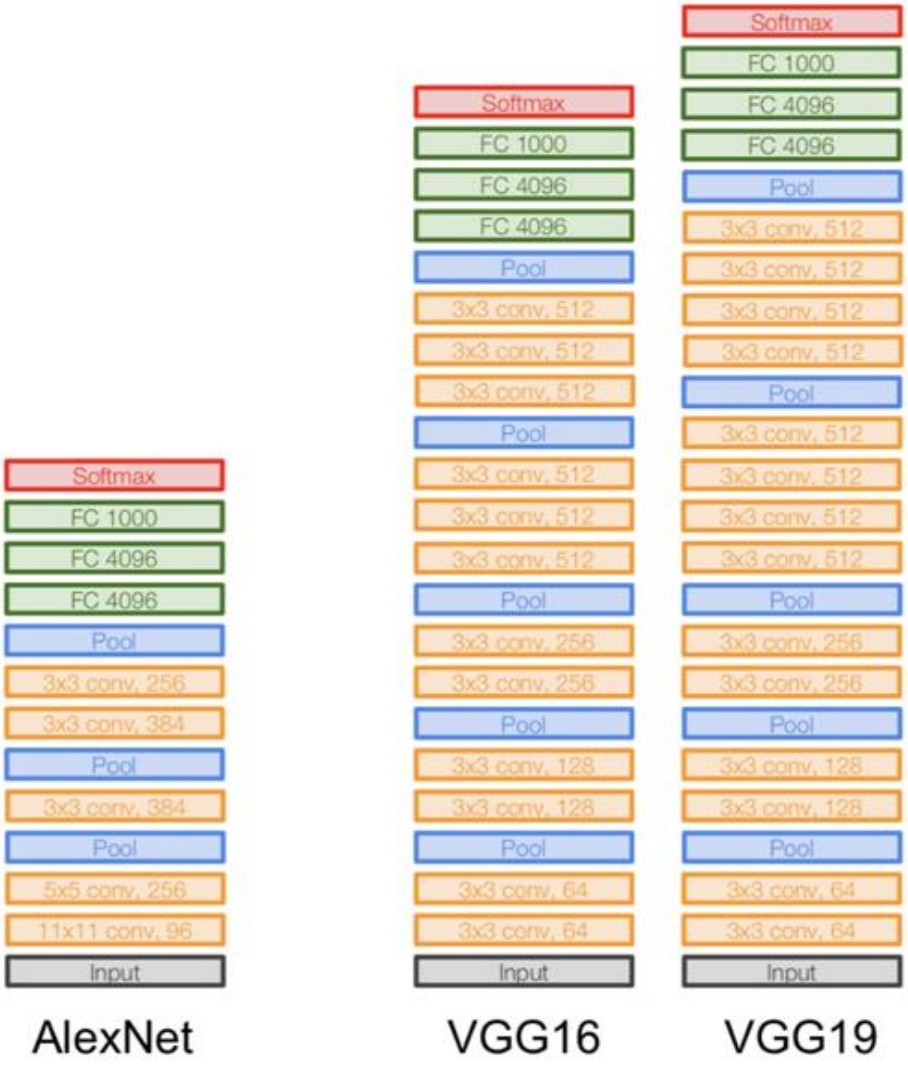


Рисунок 1.4. Порівняння архітектур AlexNet і VGG

Ефективність архітектур згорткових нейронних мереж (CNN) зумовлена кількома ключовими факторами: влучністю розпізнавання, швидкістю опрацювання даних, обчислювальною складністю та обсягом пам'яті, потрібним для збереження параметрів моделі. Ці властивості особливо важливі при виборі архітектури для конкретних застосувань, як–от, для виявлення людей на зображеннях.

Архітектура LeNet, одна з перших представників свого класу, вирізняється простотою та невеликою кількістю параметрів, що забезпечує її швидку роботу навіть на пристроях з обмеженими ресурсами. Проте вона вже не відповідає

потребам сучасних задач комп'ютерного зору, де потрібна висока точність та здатність розпізнавати складні об'єкти в умовах, максимально наближених до реальних.

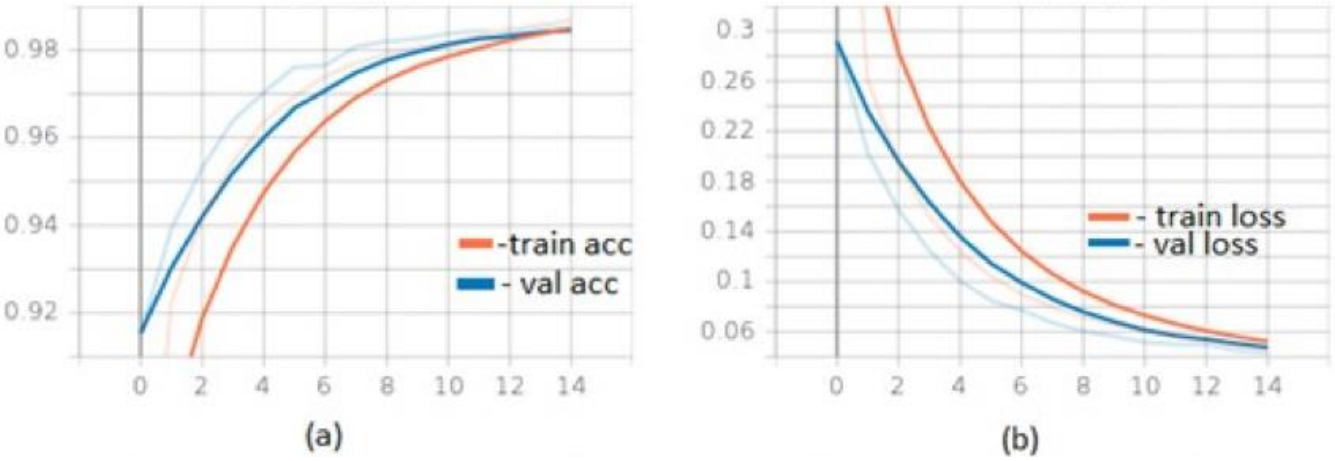


Рисунок 1.5. Залежність точності (a) і втрат (b) Lenet від кількості епох на тренувальних і валідаційних даних

Модель AlexNet суттєво підняла планку в точності розпізнавання, завдяки глибшій організації та застосуванню графічних процесорів значних обчислювальних потужностей. Це накладає обмеження на її використання у проєктах, де ключовими є швидкість обробки даних або обмежена апаратна база.

Архітектура VGG продемонструвала ще кращу точність, завдяки великій глибині та постійному використанню малих фільтрів розміром 3 на 3.

Розробка ResNet стала визначальною віхою у вирішенні проблеми погіршення точності, що виникала зі збільшенням глибини мережі. Завдяки введенню залишкових з'єднань, ResNet змогла зберегти високу якість класифікації навіть на великих глибинах.

Ще одним кроком у напрямі оптимізації постала архітектура EfficientNet, яка демонструє високу точність завдяки збалансованому масштабуванню трьох ключових параметрів мережі: глибини, ширини та розпізнавальної здатності. Це дає змогу здобувати ефективні результати з мінімальним використанням ресурсів.

У завданні пошуку людей на фотографіях критично важливими є висока точність знаходження облич, стійкість до змін умов зйомки (наприклад, освітлення, кут або якість зображення) та швидкість обробки. Архітектури ResNet–50 або EfficientNet–B0 визнаються одними з найліпших. Вони вміють

забезпечувати високу точність розпізнавання, не вимагаючи надмірних ресурсів, що робить їх ідеальними для використання у різних рішеннях. Це стосується, наприклад, мобільних додатків та десктопних програм, які розпізнають обличчя.

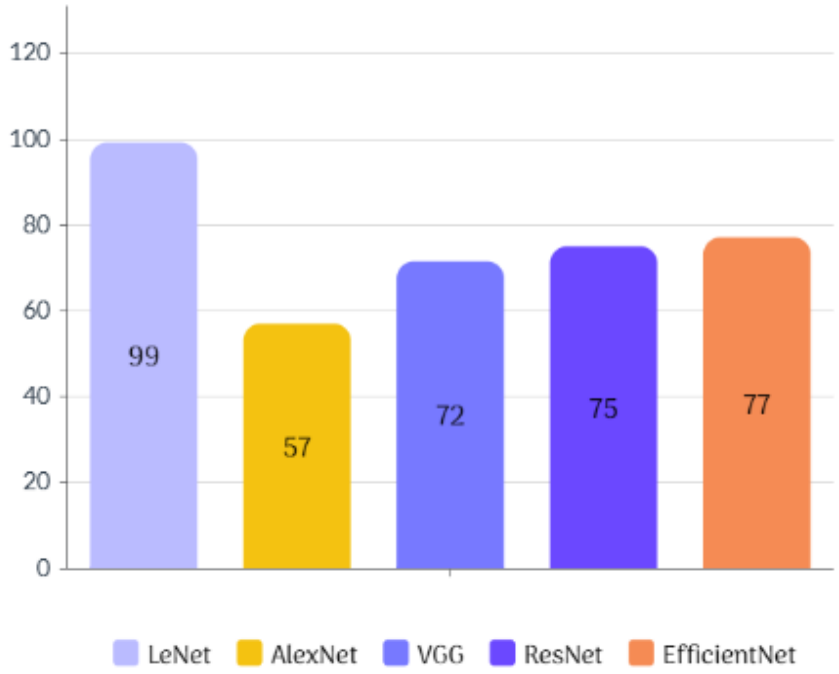


Рисунок 1.6. Діаграма порівняння архітектур з урахуванням приблизної точності в процентах

1.2 Вибір засобів розробки застосунку

Під час розробки програмного забезпечення для розпізнавання облич надзвичайно важливо обґрунтовано обрати інструменти, які дозволяють ефективно реалізувати як інтерфейс користувача, так і алгоритмічну частину системи.

Основною метою вибору засобів було досягнення оптимального балансу між функціональністю, продуктивністю та гнучкістю у реалізації складних обчислювальних процесів. У розробці я врахувала актуальні тенденції у сфері машинного навчання, обробки зображень та створення графічних інтерфейсів, що дало мені змогу сформувати інструменти, здатні забезпечити надійну роботу системи, простоту інтеграції та можливість подальшого масштабування.

Я обрала саме ці бібліотеки й середовище розробки, враховуючи їх популярність, доступні матеріали для навчання та те, як вони працюють один з

одним. Саме завдяки цьому я ефективно реалізувала всі необхідні функціональні компоненти, від завантаження зображень з їх обробкою, навчання та використання згорткової нейронної мережі для знаходження облич.

1.2.1 Обгрунтування вибору мови програмування

Я вирішила обрати мову програмування Python для свого застосунку, адже це дуже зручний інструмент, особливо коли йдеться про задачі машинного навчання. Його синтаксис простий і зрозумілий для написання і читання коду, а також ця мова має велику підтримку різноманітних бібліотек, що спрощують роботу з нейронними мережами, маючи на увазі такі інструменти, як TensorFlow, Keras, PyTorch та OpenCV.

Можливість швидко перевіряти нові ідеї є найкращою в Python. Розробнику можна без зайвих зусиль змінити архітектуру моделі, обрати інші налаштування чи протестувати інший підхід. Це дуже зручно, коли експериментуєш із різними варіантами мереж і намагаєшся знайти найкраще рішення для конкретного завдання, особливо для розпізнавання облич на фотографіях.

Серед альтернатив були C++ і Java, з кращою продуктивністю у деяких ситуаціях, особливо там, де важлива максимальна швидкість або обмежені ресурси, але вони не мають такої гнучкої системи налаштування, як і не мають різноманіття вже готових інструментів машинного навчання. Якщо робити висновок, розробка на цих мовах програмування потребує більше часу та сил, особливо на етапі експериментів та тестів.

У результаті я дійшла висновку, що Python це найкращий вибір для мого проєкту. Він дозволив мені сконцентрувати увагу на головних речах, таких як зробити модель точною та ефективною, а не витрачати час на технічні дрібниці. Це полегшило процес створення інтелектуальних систем обробки зображень, і допомогло досягти кращіх результатів.

1.2.2 Вибір бібліотек та фреймворків

Я обрала набір бібліотек і фреймворків для свого застосунку з розпізнавання облич на фотографіях, щоб вони забезпечили ефективну обробку зображень. Це

tkinter, Pillow (PIL), OpenCV (cv2), NumPy, threading, os, datetime, TensorFlow і scikit–learn (sklearn).

TensorFlow широко використовують фахівці з даних, розробники програмного забезпечення та викладачі. Це платформа з відкритим кодом для машинного навчання, заснована на графіках потоку даних. Вершини графа відповідають математичним операціям, а ребра представляють багатовимірні масиви даних (тензори), які циркулюють між ними. Ця гнучка архітектура дозволяє визначати алгоритми машинного навчання як графік взаємопов'язаних операцій. Вибір TensorFlow зумовлений його потужною системою, підтримкою попередньо навчених моделей і можливістю взаємодії з графічними процесорами (GPU) для прискорення обчислень, що є ключовим фактором для обробки зображень в реальному часі.

OpenCV (Open Source Computer Vision Library) – це бібліотека для комп'ютерного зору та машинного навчання, доступна з відкритим кодом. Основна мета OpenCV – надати базову інфраструктуру для комп'ютерного зору, а також прискорити впровадження технологій машинного сприйняття у комерційних продуктах. Бібліотека включає в себе понад 2500 оптимізованих алгоритмів, які охоплюють як класичні, так і сучасні методи комп'ютерного зору та машинного навчання. Ці алгоритми знаходять застосування у різноманітних задачах, таких як розпізнавання облич і об'єктів, аналіз дій на відео, відстеження руху камери та об'єктів. Також можливе створення 3D–моделей об'єктів, формування 3D–хмар точок зі стереокамер, зшивання зображень для створення панорам з високою роздільною здатністю. До того ж, OpenCV дозволяє здійснювати пошук схожих зображень у базах даних, корекцію ефекту "червоних очей" на фотографіях, трекінг руху очей, розпізнавання пейзажів та накладання маркерів для доповненої реальності. OpenCV стає вибором через свою універсальність.

Tkinter – це стандартна бібліотека Python GUI створення графічного інтерфейсу користувача, яка надає набір інструментів і віджетів для настільних програм із графічним інтерфейсом. Tkinter включено до більшості установок Python, що робить його легко доступним для розробників, які прагнуть розробляти

додатки з GUI, не вдаючись до додаткових інсталяцій чи бібліотек. Вона була обрана для розробки інтерактивного інтерфейсу застосунку завдяки своїй простоті, кросплатформності та інтеграції з Python.

Pillow – це унікальна Python бібліотека, що призначена для роботи із зображеннями. Хоча існують альтернативні Python–інструменти для обробки графіки, Pillow залишається ключовим інструментом для ознайомлення та подальшої роботи в цілому. В коді програми модулі Image та ImageTk з Pillow залучаються для підготовки графіки до відображення в мітці (Label) графічного інтерфейсу, зокрема для масштабування зображень до певних розмірів (наприклад, 600x400 пікселів) з підтримкою якості через алгоритм LANCZOS. Я зупинилася саме на Pillow через її підтримку різноманітних форматів зображень, включаючи JPEG, JPG і PNG, та легкість взаємодії з іншими бібліотеками.

NumPy – це основний пакет для наукових обчислень на Python. Це бібліотека Python, яка надає об’єкт багатовимірного масиву, різноманітні похідні об’єкти (такі як замасковані масиви та матриці), а також набір процедур для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції формою, сортування, вибір, введення/виведення, дискретне перетворення Фур’є, основну лінійну алгебру, базові статистичні операції, випадкове моделювання та багато іншого. Її інтеграція з TensorFlow і OpenCV дозволяє безперешкодно передавати дані між різними етапами обробки, що робить її незамінною для реалізації алгоритмів розпізнавання.

Scikit–learn – бібліотека, призначена для машинного навчання, написана мовою програмування Python і поширюється у вигляді безкоштовного програмного забезпечення. У її склад входять різні алгоритми, в тому числі призначені для завдань класифікації, регресійного та кластерного аналізу даних, включаючи метод опорних векторів, метод випадкового лісу, алгоритм посилення градієнта, метод k–середніх і DBSCAN. Хоча scikit–learn має ширший функціонал для машинного навчання, у цьому застосунку її використання обмежується базовими інструментами підготовки даних, що відповідає потребам задачі.

Threading використовується для виконання обчислювально складних

операцій, таких як аналіз зображень, у фоновому потоці, щоб уникнути блокування графічного інтерфейсу. У коді модуль threading дозволяє запускати функцію perform_analysis асинхронно, забезпечуючи плавну роботу інтерфейсу під час обробки зображень. Це підвищує зручність використання застосунку, дозволяючи користувачу взаємодіяти з інтерфейсом під час виконання аналізу фотографії.

OS і Datetime виконують допоміжні функції в застосунку. Модуль os використовується для роботи з файловою системою, зокрема для перевірки існування папок із даними (dataset/faces, dataset/non_faces) та ітерації по файлах із зображеннями. Модуль datetime додає мітки часу до логів, що полегшує відстеження історії аналізу. Ці модулі є частиною стандартної бібліотеки Python, що усуває необхідність додаткових залежностей і забезпечує надійність.

Тобто Tkinter і Pillow забезпечують зручний графічний інтерфейс, OpenCV і TensorFlow для комп'ютерного зору та машинного навчання, а NumPy і scikit–learn додані для оптимізації обробки даних. Допоміжні модулі threading, os і datetime доповнюють функціонал.

1.2.3 Середовище розробки

Для розробки застосунку з розпізнавання облич на фотографіях я обрала таке середовище розробки, яке забезпечує комфортне написання, налагодження, тестування та масштабування коду. Застосунок, побудований мовою Python, передбачає роботу з різноманітними компонентами – від графічного інтерфейсу до згорткової нейронної мережі та обробки зображень. Вибране середовище має також без проблем інтегруватися з ключовими бібліотеками. Саме тому для реалізації цього проекту було обрано Visual Studio Code – потужне, гнучке та кросплатформне середовище розробки, яке поєднує в собі легкість, високу функціональність та підтримку великої кількості розширень.

Visual Studio Code дозволяє ефективно працювати над проектами, які поєднують класичне програмування та машинне навчання. Його офіційне розширення Python від Microsoft забезпечує інтелектуальне автодоповнення, динамічну перевірку помилок, можливості рефакторингу та відлагодження, що є критичним для роботи з багатошаровими структурами, такими як згорткові

нейронні мережі, тобто CNN. Наприклад, налагодження таких етапів, як зчитування зображень за допомогою cv2.imread(), попередня обробка зображень із Pillow або навчання нейронної мережі через model.fit() у TensorFlow, відбувається швидко та зручно. Крім того, редактор дозволяє миттєво переглядати результат виконання, навіть у випадках роботи з потоками, асинхронними викликами чи взаємодії з файловою системою.

Завдяки підтримці віртуальних середовищ Python (venv) Visual Studio Code дає змогу ізолювати залежності для кожного проєкту, уникаючи конфліктів між версіями бібліотек. Наприклад, TensorFlow може мати специфічні вимоги до версії NumPy або protobuf, і запускати кілька проектів із різними конфігураціями без віртуальних середовищ було б складно. Через інтеграцію VS Code із pip та requirements.txt стає можливо швидко оновлювати, документувати та розгортати проєкт в ідентичному середовищі.

Крім основного функціоналу, Visual Studio Code надає широкий набір розширень. Наприклад:

- Jupyter – дозволяє запускати інтерактивні ноутбуки безпосередньо у VS Code. Це зручно для експериментів із фрагментами коду, візуалізацією результатів навчання моделей або перевірки роботи окремих функцій.
- TensorFlow Snippets – полегшує написання коду з TensorFlow.
- Python Environment Manager – допомагає швидко перемикатися між віртуальними середовищами.
- GitLens та Git Graph – покращують роботу з системою контролю версій, дозволяючи бачити історію змін, комітів, гілок, що є важливо при колективній роботі.

Visual Studio Code також забезпечує глибоку інтеграцію з Git, що дозволяє розробникам створювати коміти, виконувати злиття, переглядати зміни та пушити їх до віддаленого репозиторію. Для проекту, де реалізовані окремі класи, такі як FaceDetectionApp, FaceDetectionCNN, а також функції з обробки даних, це спрощує керування кодом, особливо при внесенні змін до архітектури мережі чи інтерфейсу.

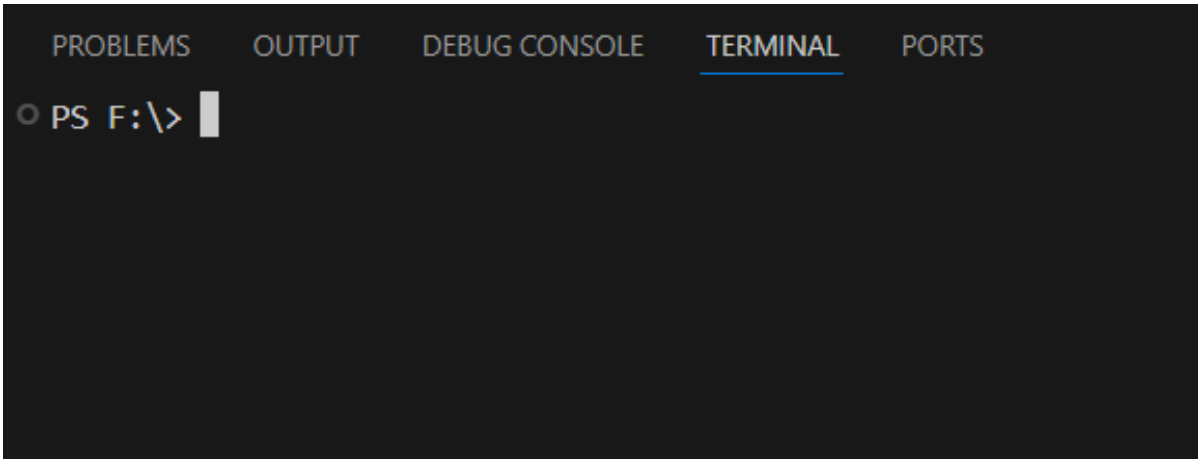


Рисунок 1.7. Інтегрований термінал VS Code

Інтегрований термінал у VS Code дозволив мені запускати скрипти напряму, без потреби відкривати окремі інтерпретатори або IDE. Ще одним вагомим плюсом є зручна навігація по структурі проекту.

Для обробки великих наборів даних, наприклад, папок dataset/faces та dataset/non_faces, розширення File Explorer допомагало мені швидко знаходити, відкривати та змінювати дані.

Порівнюючи VS Code з альтернативними середовищами, можна зазначити таке:

– PyCharm має більш розширену функціональність для аналізу Python–коду, але його повна версія є платною, а сама IDE споживає більше ресурсів.
– Jupyter Notebook чудово підходить для навчання моделей та аналізу даних, але значно обмежений у можливостях управління проектами, особливо тими, що мають складну файлову структуру або включають GUI.
– Spyder – ще один кандидат, орієнтований на наукові розрахунки, проте має менше підтримки для таких бібліотек, як PyQt або tkinter, а також не надає такого широкого вибору розширень, як VS Code.
– VS Code є безкоштовним, регулярно оновлюється, має велику спільноту, що робить його не лише практичним, а й перспективним вибором для довгострокових проєктів. Він однаково добре підходить як для

дослідницького програмування (експерименти з CNN), так і для побудови готових застосунків із GUI.

1.3. Розробка структури застосунку

1.3.1. Архітектура застосунку

Додаток складається з двох основних Python–файлів – face_detection_app.py и face_detection_cnn.py.

Файл face_detection_app.py відповідає за графічний інтерфейс користувача (GUI), логіку взаємодії з користувачем, керування потоками виконання та координацію роботи з модулем машинного навчання. Він містить два класи: FaceDetectionApp (основний додаток) і TrainingDialog (діалогове вікно для навчання моделі).

Основний клас тут – FaceDetectionApp, який є ядром і основним модулем програми, який відповідає за за створення GUI, обробку подій користувача та виклик методів із модуля FaceDetectionCNN. Основні компоненти і методи:

1. Ініціалізація (__init__):

1.1. Атрибути:

– self.root: Кореневий об’єкт Tkinter (tk.Tk), основне вікно програми.
– self.detector: Екземпляр класу FaceDetectionCNN для роботи з нейронною мережею.
– self.current_image: Поточне зображення (масив NumPy у форматі RGB), яке аналізується.
– self.detected_faces: Список знайдених облич (словники з координатами, розмірами та впевненістю).
– self.image_path: Шлях до поточного зображення.
– self.status_var: Змінна Tkinter для відображення статусу в нижній панелі.

1.2. Дії:

Встановлює заголовок вікна ("Система розпізнавання облич") і розмір (1200x800). Створює GUI за допомогою методу create_widgets. Викликає auto_load_model для автоматичного завантаження моделі з

models/best_face_model.h5, якщо вона існує.

```
    def create_widgets(self):
        menubar = tk.Menu(self.root)
        self.root.config(menu=menubar)
        file_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Файл", menu=file_menu)
        file_menu.add_command(label="Відкрити зображення", command=self.open_image)
        file_menu.add_command(label="Зберегти результат", command=self.save_result)
        file_menu.add_separator()
        file_menu.add_command(label="Вихід", command=self.root.quit)
        model_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Модель", menu=model_menu)
        model_menu.add_command(label="Навчити модель", command=self.train_model_dialog)
        model_menu.add_command(label="Завантажити модель", command=self.load_model)
        toolbar = ttk.Frame(self.root)
        toolbar.pack(side=tk.TOP, fill=tk.X, padx=5, pady=5)
        ttk.Button(toolbar, text="Відкрити", command=self.open_image).pack(side=tk.LEFT, padx=5)
        ttk.Button(toolbar, text="Аналізувати", command=self.analyze_image).pack(side=tk.LEFT, padx=5)
        ttk.Button(toolbar, text="Очистити", command=self.clear_results).pack(side=tk.LEFT, padx=5)
        settings_frame = ttk.LabelFrame(toolbar, text="Налаштування", padding=5)
        settings_frame.pack(side=tk.RIGHT, padx=5)
        ttk.Label(settings_frame, text="Поріг впевненості:").pack(side=tk.LEFT)
        self.confidence_var = tk.DoubleVar(value=0.7)
        confidence_scale = ttk.Scale(settings_frame, from_=0.1, to=0.9,
```

Рисунок 1.8. Елемент коду графічного інтерфейсу

2. Метод create_widgets:

1.1. Функціонал: Створює всі елементи графічного інтерфейсу.

1.2. Компоненти GUI:

– Меню (menubar):

Меню "Файл": "Відкрити зображення", "Зберегти результат", "Вихід".

Меню "Модель": "Навчити модель", "Завантажити модель".

– Панель інструментів (toolbar):

Кнопки: "Відкрити", "Аналізувати", "Очистити".

Налаштування: Слайдер для порогу впевненості (0.1–0.9, за замовчуванням 0.7) із міткою, що відображає поточне значення.

– Основна область (main_frame):

Використовує ttk.PanedWindow для розділення на ліву та праву панелі.

– Ліва панель (left_frame):

Містить image_frame із міткою (image_label) для відображення зображення (початково показує текст "Оберіть зображення для аналізу").

– Права панель (right_frame):

Інформаційне поле (info_frame): Текстове поле (info_text) із прокруткою для логів (наприклад, "Завантажено зображення", "Аналіз завершено").

Таблиця облич (faces_frame): ttk.Treeview (faces_tree) із колонками "№", "Впевненість", "Координати", "Розмір" для відображення знайдених облич.

Кнопки керування (buttons_frame): Кнопка "Видалити помилкові" для видалення вибраних облич із таблиці.

– Статусна панель (status_bar):

Мітка внизу вікна, що показує поточний стан (наприклад, "Готовий", "Аналіз зображення...").

3. Метод auto_load_model:

```
90      def auto_load_model(self):
91          model_path = os.path.join("best_face_model.h5")
92          if os.path.exists(model_path):
93              try:
94                  self.detector.load_model(model_path)
95                  self.add_info(f"Автоматично завантажено модель: {os.path.basename(model_path)}")
96                  self.status_var.set("Модель автоматично завантажена")
97              except Exception as e:
98                  self.add_info(f"Помилка автоматичного завантаження моделі: {str(e)}")
99                  self.status_var.set("Помилка автоматичного завантаження моделі")
100         else:
101             self.add_info("Модель не знайдено в папці. Оберіть модель вручну.")
102             self.status_var.set("Модель не знайдено")
103
104     def update_confidence_label(self, value):
105         self.confidence_label.config(text=f"{float(value):.2f}")
106
107     def open_image(self):
108         file_path = filedialog.askopenfilename(
109             title="Оберіть зображення",
110             filetypes=[("JPEG файли", "*.jpg *.jpeg"), ("PNG файли", "*.png"),
111                        ("Всі зображення", "*.jpg *.jpeg *.png")]
```

Рисунок 1.9. Елемент коду методів auto_load_model і open_image

Перевіряє наявність файлу models/best_face_model.h5, якщо файл існує, завантажує модель через self.detector.load_model. Логує результат у info_text і оновлює status_var.

4. Метод open_image:

Відкриває діалогове вікно для вибору зображення (формати: JPG, PNG). Завантажує зображення через load_and_display_image, зберігає його в self.current_image і шлях у self.image_path. Логує подію та оновлює статус.

5. Метод load_and_display_image:

Зчитує зображення за допомогою OpenCV (cv2.imread). Конвертує з BGR у RGB (cv2.cvtColor). Змінює розмір для відображення (максимум 600x400 пікселів) за допомогою Pillow (Image.thumbnail). Конвертує у формат Tkinter (ImageTk.PhotoImage) і відображає в image_label.

6. Метод analyze_image:

Перевіряє, чи завантажено зображення та модель. Запускає аналіз у окремому потоці (threading.Thread) через метод perform_analysis. Оновлює статус і лог.

7. Метод perform_analysis:

```
122     def load_and_display_image(self, file_path):
123         image = cv2.imread(file_path)
124         image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
125         self.current_image = image_rgb
126         pil_image = Image.fromarray(image_rgb)
127         display_size = (600, 400)
128         pil_image.thumbnail(display_size, Image.Resampling.LANCZOS)
129         self.photo = ImageTk.PhotoImage(pil_image)
130         self.image_label.configure(image=self.photo)
131 
132     def analyze_image(self):
133         if self.current_image is None:
134             messagebox.showwarning("Попередження", "Спочатку оберіть зображення")
135             return
136         if self.detector.model is None:
137             messagebox.showerror("Помилка", "Модель не завантажена. Спочатку навчіть або завантажте модель.")
138             return
139         self.status_var.set("Аналіз зображення...")
140         self.add_info("Початок аналізу зображення...")
141         threading.Thread(target=self.perform_analysis, daemon=True).start()
142 
143     def perform_analysis(self):
144         try:
145             confidence_threshold = self.confidence_var.get()
146             image, faces = self.detector.detect_faces_in_image(self.image_path, confidence_threshold)
147             self.detected_faces = faces
148             self.root.after(0, self.update_results, faces)
149         except Exception as e:
150             self.root.after(0, self.show_error, f"Помилка аналізу: {str(e)}")
```

Рисунок 1.10. Код методів load_and_display_image, analyze_image і perform_analysis

Отримує поріг впевненості зі слайдера (self.confidence_var) і викликає self.detector.detect_faces_in_image для аналізу зображення. Зберігає знайдені обличчя в self.detected_faces і викликає update_results через root.after для безпечного оновлення GUI.

8. Метод update_results:

Очищає таблицю faces_tree, заповнює таблицю даними про знайдені обличчя

(номер, впевненість, координати, розмір). Викликає display_image_with_detections для відображення зображення з рамками навколо облич. Логує кількість знайдених облич і оновлює статус.

9. Метод display_image_with_detections:

Малює прямокутники навколо облич на копії зображення (self.current_image), колір рамки залежить від впевненості: зелений (>0.8), жовтий (0.6–0.8), червоний (<0.6). Додає текстові мітки з номером обличчя та впевненістю, змінює розмір і відображає результат у image_label.

```
166     def display_image_with_detections(self, faces):
167         if self.current_image is None:
168             return
169         image_with_boxes = self.current_image.copy()
170         for i, face in enumerate(faces):
171             x, y, w, h = face['x'], face['y'], face['width'], face['height']
172             confidence = face['confidence']
173             if confidence > 0.8:
174                 color = (0, 255, 0)
175             elif confidence > 0.6:
176                 color = (255, 255, 0)
177             else:
178                 color = (255, 0, 0)
179             cv2.rectangle(image_with_boxes, (x, y), (x + w, y + h), color, 2)
180             label = f"#{i+1} ({confidence:.2f})"
181             cv2.putText(image_with_boxes, label, (x, y - 10),
182                         cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)
183         pil_image = Image.fromarray(image_with_boxes)
184         display_size = (600, 400)
185         pil_image.thumbnail(display_size, Image.Resampling.LANCZOS)
186         self.photo = ImageTk.PhotoImage(pil_image)
187         self.image_label.configure(image=self.photo)
```

Рисунок 1.11. Код методу display_image_with_detections

10. Метод remove_false_positives:

Видаляє вибрані обличчя з таблиці faces_tree і списку self.detected_faces, оновлює відображення через update_results, логує видалені обличчя.

11. Метод save_result:

Відкриває діалогове вікно для збереження зображення з рамками, малює рамки та мітки на копії зображення (аналогічно display_image_with_detections). Зберігає зображення у форматі JPG або PNG через OpenCV (cv2.imwrite), логує подію.

12. Метод clear_results:

Очищає таблицю faces_tree і список self.detected_faces. Відображає оригінальне зображення без рамок, логує подію.

13. Метод train_model_dialog:

Створює екземпляр класу TrainingDialog для налаштування та запуску навчання моделі.

14. Метод add_info:

Додає повідомлення з часовою міткою до info_text, автоматично прокручує до кінця.

15. Метод show_error:

Показує спливаюче вікно з помилкою та оновлює статус.

TrainingDialog – це клас, який містить діалогове вікно для навчання моделі. Він створює модальне діалогове вікно для налаштування та запуску навчання моделі. Основні компоненти:

1. Ініціалізація (__init__):

Створює вікно tk.Toplevel із розміром 500x400, робить вікно модальним (transient, grab_set) і не дозволяє змінювати розмір. Викликає create_widgets для створення елементів інтерфейсу.

2. Метод create_widgets

2.1. Компоненти:

Заголовок: "Навчання моделі розпізнавання облич".

2.2. Текстове поле (tk.Text):

Пояснення щодо підготовки даних (папки dataset/faces і dataset/non_faces, 500+ зображень, формати JPG/PNG, розмір 64x64+).

2.3. Налаштування:

Кількість епох (10–200, за замовчуванням 50), розмір пакету (8–128, за замовчуванням 32), швидкість навчання (за замовчуванням 0.001), кнопки: "Почати навчання", "Скасувати", панель прогресу: Мітка (progress_label) та індикатор прогресу (ttk.Progressbar).

```
287     def create_widgets(self):
288         title_label = ttk.Label(self.dialog, text="Навчання моделі розпізнавання обличь",
289                                 font=("Arial", 12, "bold"))
290         title_label.pack(pady=10)
291         instructions = tk.Text(self.dialog, height=8, wrap=tk.WORD)
292         instructions.pack(fill=tk.X, padx=10, pady=5)
293         instructions.insert(tk.END,
294             "Для навчання моделі вам потрібно:\n\n"
295             "1. Створити папку 'dataset/faces' та додати туди 500+ зображень з обличчями\n"
296             "2. Створити папку 'dataset/non_faces' та додати туди 500+ зображень без обличь\n"
297             "3. Зображення мають бути у форматі JPG/PNG, розміром 64x64 або більше\n"
298             "4. Навчання може тривати кілька годин залежно від кількості даних\n\n"
299             "Рекомендації:\n"
300             "- Використовуйте різноманітні зображення (різні освітлення, кути, люди)\n"
301             "- Для негативних прикладів використовуйте пейзажі, предмети, тварин\n"
302             "- Більше даних = краща точність моделі"
303         )
```

Рисунок 1.12. Елемент коду методу create_widgets

3. Метод start_training

Перевіряє наявність папок dataset/faces і dataset/non_faces, рахує кількість зображень у кожній папці, якщо зображень менше 50, показує попередження з можливістю продовжити. Запускає навчання в окремому потоці через train_model.

```
333     def start_training(self):
334         if not os.path.exists('dataset/faces') or not os.path.exists('dataset/non_faces'):
335             messagebox.showerror("Помилка",
336                 "Папки з даними не знайдено. Створіть папки 'dataset/faces' та 'dataset/non_faces'")
337             return
338         faces_count = len([f for f in os.listdir('dataset/faces')
339                            if f.lower().endswith(('.jpg', '.jpeg', '.png'))])
340         non_faces_count = len([f for f in os.listdir('dataset/non_faces')
341                                if f.lower().endswith(('.jpg', '.jpeg', '.png'))])
342         if faces_count < 50 or non_faces_count < 50:
343             result = messagebox.askyesno("Попередження",
344                 f"Знайдено мало даних:\n"
345                 f"Обличчя: {faces_count}\n"
346                 f"Не обличчя: {non_faces_count}\n\n"
347                 f"Рекомендується мати мінімум 500 зображень кожного типу.\n"
348                 f"Продовжити навчання?")
349             if not result:
350                 return
351         self.progress_bar.start()
352         self.progress_var.set("Завантаження даних...")
353         training_thread = threading.Thread(target=self.train_model, daemon=True)
354         training_thread.start()
```

Рисунок 1.13. Код методу start_training

4. Метод train_model

Завантажує дані через self.app.detector.create_training_data, створює модель із заданою швидкістю навчання. Навчає модель. Завантажує збережену модель (models/best_face_model.h5). Викликає training_completed або training_error залежно від результату.

```
356     def train_model(self):
357         try:
358             X, y = self.app.detector.create_training_data('dataset/faces', 'dataset/non_faces')
359             if len(X) == 0:
360                 raise ValueError("Не вдалося завантажити дані для навчання")
361             self.app.detector.create_model(learning_rate=self.lr_var.get())
362             history = self.app.detector.train_model(
363                 X, y,
364                 epochs=self.epochs_var.get(),
365                 batch_size=self.batch_var.get()
366             )
367             self.app.detector.load_model('best_face_model.h5')
368             self.dialog.after(0, self.training_completed)
369         except Exception as e:
370             self.dialog.after(0, self.training_error, str(e))
```

Рисунок 1.14. Код методу train_model

5. Метод training_completed

Зупиняє індикатор прогресу, показує повідомлення про успіх і знищує діалогове вікно.

6. Метод training_error

Зупиняє індикатор прогресу, показує повідомлення про помилку.

Файл face_detection_cnn.py містить логіку створення, навчання та використання згорткової нейронної мережі для розпізнавання облич. Основний клас – FaceDetectionCNN, який інкапсулює всі операції, пов'язані з моделлю. Його основні компоненти:

1. Ініціалізація (__init__)

1.1. Атрибути:

self.model: Об'єкт моделі Keras (початково None).

self.input_shape: Розмір вхідного зображення (64x64x3, тобто 64x64 пікселі, 3 канали RGB).

2. Метод create_model

2.1. Архітектура моделі:

– Згортковий шар: 32 фільтри, ядро 3x3, активація ReLU.
– Макс–пулінг: 2x2.
– Згортковий шар: 64 фільтри, ядро 3x3, ReLU.
– Макс–пулінг: 2x2.
– Згортковий шар: 128 фільтрів, ядро 3x3, ReLU.

– Макс–пулінг: 2x2.
– Flatten: Перетворення у вектор.
– Повнозв’язний шар: 128 нейронів, ReLU.
– Dropout: 50% для запобігання перенавчанню.
– Вихідний шар: 1 нейрон, активація sigmoid (ймовірність наявності обличчя).

2.2. Компіляція:

– Оптимізатор: Adam із заданою швидкістю навчання.
– Функція втрат: Binary cross–entropy.
– Метрика: Accuracy.

```
16      def create_model(self, learning_rate=0.001):
17          model = models.Sequential([
18              layers.Conv2D(32, (3, 3), activation='relu', input_shape=self.input_shape),
19              layers.BatchNormalization(),
20              layers.MaxPooling2D((2, 2)),
21              layers.Dropout(0.25),
22              layers.Conv2D(64, (3, 3), activation='relu'),
23              layers.BatchNormalization(),
24              layers.MaxPooling2D((2, 2)),
25              layers.Dropout(0.25),
26              layers.Conv2D(128, (3, 3), activation='relu'),
27              layers.BatchNormalization(),
28              layers.MaxPooling2D((2, 2)),
29              layers.Dropout(0.25),
30              layers.Conv2D(256, (3, 3), activation='relu'),
31              layers.BatchNormalization(),
32              layers.MaxPooling2D((2, 2)),
33              layers.Dropout(0.5),
34              layers.Flatten(),
35              layers.Dense(512, activation='relu'),
36              layers.BatchNormalization(),
37              layers.Dropout(0.5),
38              layers.Dense(256, activation='relu'),
39              layers.BatchNormalization(),
40              layers.Dropout(0.5),
41              layers.Dense(1, activation='sigmoid')
42          ])
```

Рисунок 1.15. Код шарів архітектури моделі

3. Метод create_training_data

Завантажує зображення з папок faces_dir і non_faces_dir, змінює розмір до 64x64 пікселів, нормалізує пікселі (ділення на 255). Повертає масиви NumPy data і labels. Позначки: 1 для облич, 0 для не–облич.

```
63      def create_training_data(self, positive_dir, negative_dir):
64          X, y = [], []
65          if os.path.exists(positive_dir):
66              for filename in os.listdir(positive_dir):
67                  if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
68                      img_path = os.path.join(positive_dir, filename)
69                      img = self.preprocess_image_for_training(img_path)
70                      if img is not None:
71                          X.append(img)
72                          y.append(1)
73          if os.path.exists(negative_dir):
74              for filename in os.listdir(negative_dir):
75                  if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
76                      img_path = os.path.join(negative_dir, filename)
77                      img = self.preprocess_image_for_training(img_path)
78                      if img is not None:
79                          X.append(img)
80                          y.append(0)
81          X = np.array(X)
82          y = np.array(y)
83          return X, y
```

Рисунок 1.16. Код методу create_training_data

4. Метод train_model

Розділяє дані на тренувальні та валідаційні (80/20) за допомогою train_test_split. Налаштовує ModelCheckpoint для збереження найкращої моделі best_face_model.h5 за валідаційною втратою (val_loss). Навчає модель із заданими епохами та розміром пакету, повертає історію навчання (history).

5. Метод load_model

Завантажує модель із файлу models/best_face_model.h5 (або іншого заданого шляху) за допомогою keras.models.load_model.

6. Метод save_model

Зберігає модель у файл models/best_face_model.h5 (або інший заданий шлях).

7. Метод detect_faces_in_image

Зчитує зображення через OpenCV. Переміщує вікно розміром 64x64 пікселів із кроком (наприклад, 32 пікселі). Для кожного вікна нормалізує дані та передбачає ймовірність наявності обличчя за допомогою моделі. Якщо ймовірність перевищує поріг (confidence_threshold), додає координати, розмір і впевненість до списку faces. Повертає зображення у форматі RGB і список облич.

```python
    def detect_faces_in_image(self, image_path, confidence_threshold=0.7):
        image = cv2.imread(image_path)
        if image is None:
            raise ValueError(f"Не вдалося завантажити зображення: {image_path}")
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        haar_faces = self.face_cascade.detectMultiScale(gray, 1.1, 4)
        detected_faces = []
        if len(haar_faces) > 0:
            for (x, y, w, h) in haar_faces:
                margin = 20
                x1 = max(0, x - margin)
                y1 = max(0, y - margin)
                x2 = min(image_rgb.shape[1], x + w + margin)
                y2 = min(image_rgb.shape[0], y + h + margin)
                face_region = image_rgb[y1:y2, x1:x2]
                try:
                    face_resized = cv2.resize(face_region, (64, 64))
                except Exception:
                    continue
                face_normalized = face_resized.astype('float32') / 255.0
                face_batch = np.expand_dims(face_normalized, axis=0)
                confidence = float(self.model.predict(face_batch, verbose=0)[0][0])
                if confidence > confidence_threshold:
                    detected_faces.append({
                        'x': x1,
                        'y': y1,
                        'width': x2 - x1,
                        'height': y2 - y1,
                        'confidence': confidence
                    })
```

Рисунок 1.17. Код методу detect_faces_in_image

Для кращого розуміння архітектури розробленої системи розпізнавання облич було створено структурну схему (рис. 1.18), яка відображає основні компоненти системи та їх взаємозв'язки.

Взаємодія між компонентами системи організована наступним чином:

– Головний додаток отримує команди користувача через GUI інтерфейс;

– При запиті на аналіз зображення, воно передається до детектора облич;

– Детектор використовує навчену модель для виявлення облич;

– Результати обробляються системою NMS для видалення дублікатів;

– Оброблені результати відображаються в інтерфейсі користувача.

Така архітектура забезпечує модульність, легку розширюваність системи, чітке розділення відповідальності між компонентами, зручність у тестуванні та модифікації окремих модулів, а також ефективну обробку даних та взаємодію.

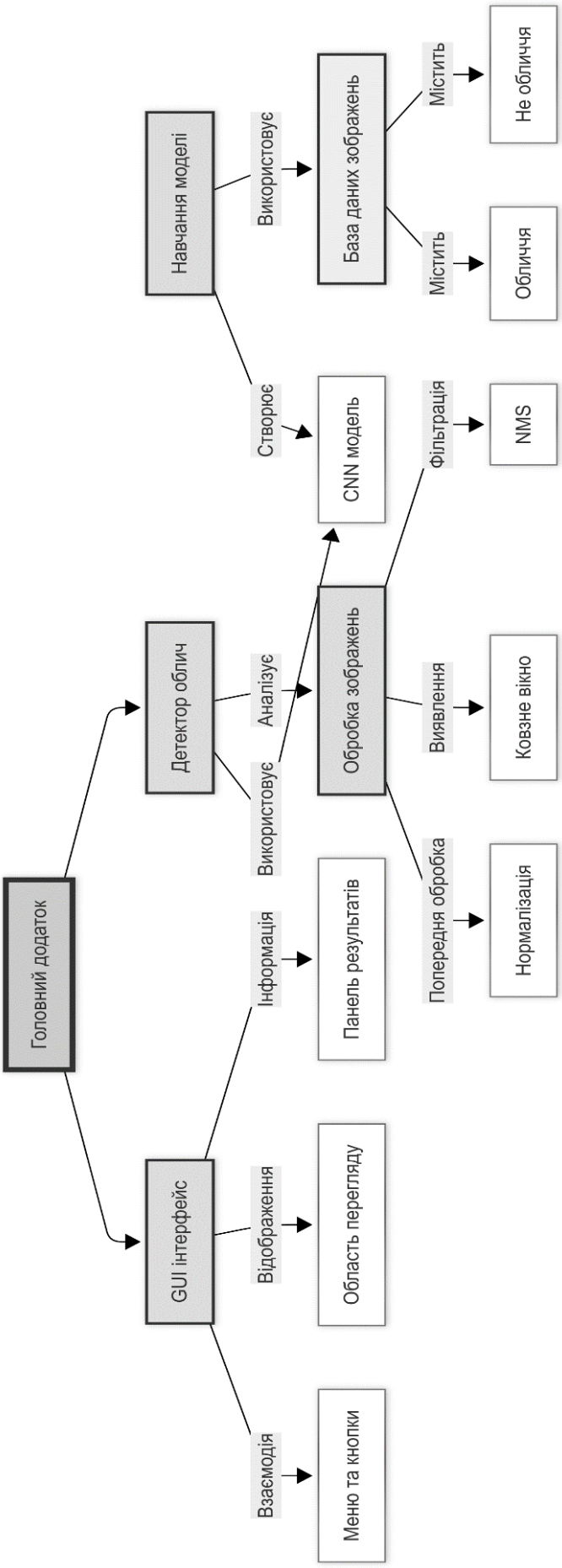


Рисунок 1.18. Структурна схема архітектури застосунку

1.3.2 Основні модулі системи

Структура застосунку складається з декількох логічних модулів, які реалізують окремі аспекти функціональності. Це:

1. Модуль графічного інтерфейсу (FaceDetectionApp)

Цей модуль відповідає за створення, відображення та управління елементами інтерфейсу. Він охоплює створення меню, кнопок, панелей, а також таблиць для виведення результатів. Наприклад, реалізовано таблицю для виведення знайдених облич так:

```
columns = ("№", "Впевненість", "Координати", "Розмір")
self.faces_tree = ttk.Treeview(faces_frame, columns=columns, show="headings", height=10)
for col in columns:
    self.faces_tree.heading(col, text=col)
    self.faces_tree.column(col, width=80)
faces_scrollbar = ttk.Scrollbar(faces_frame, orient=tk.VERTICAL, command=self.faces_tree.yview)
self.faces_tree.configure(yscrollcommand=faces_scrollbar.set)
self.faces_tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
faces_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
```

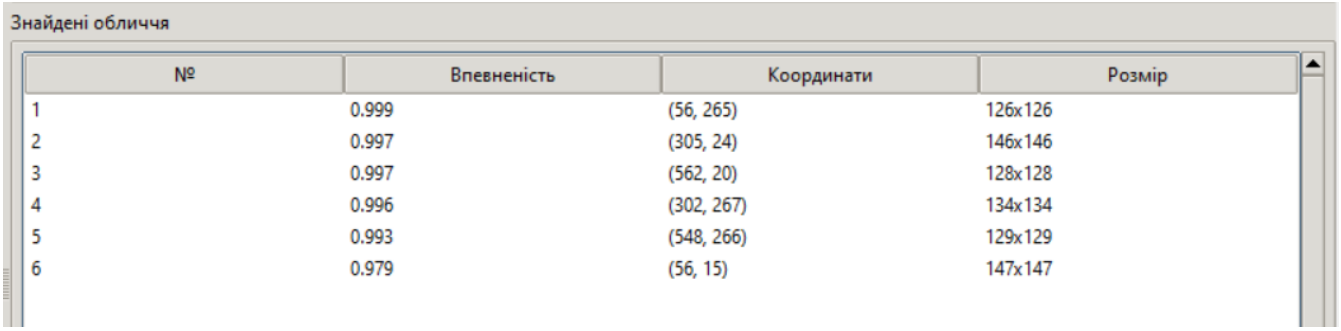


Рисунок 1.19. Таблиця для виведення знайдених облич

2. Модуль обробки зображень

Цей модуль відповідає за завантаження та попередню підготовку зображень до аналізу, також реалізує візуалізацію результатів, зокрема накладання рамок на виявлені обличчя.

```
def load_and_display_image(self, file_path):
    image = cv2.imread(file_path)
```

```
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        self.current_image = image_rgb
        pil_image = Image.fromarray(image_rgb)
        display_size = (600, 400)
        pil_image.thumbnail(display_size, Image.Resampling.LANCZOS)
        self.photo = ImageTk.PhotoImage(pil_image)
    self.image_label.configure(image=self.photo)
```

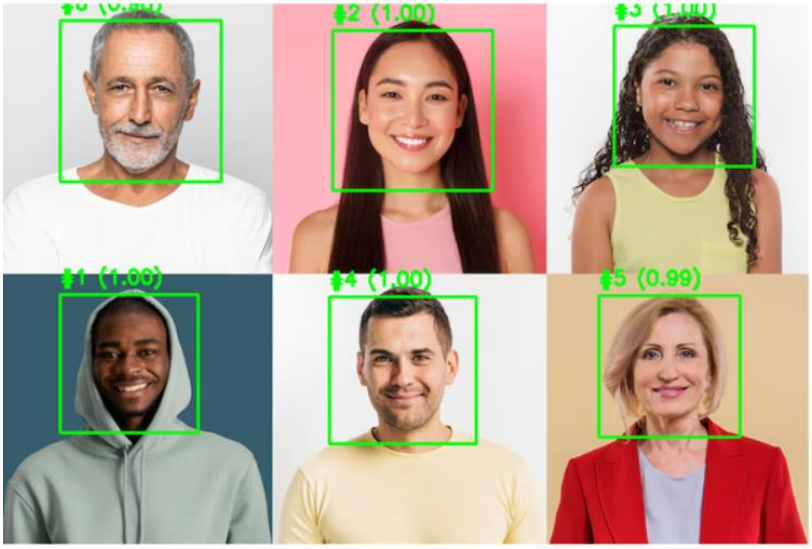


Рисунок 1.20. Накладання рамок на виявлені обличчя

3. Модуль розпізнавання облич

Основна задача цього модуля це знаходження обличчя на зображенні, використовуючи попередньо навчений CNN:

```
def detect_faces_in_image(self, image_path, confidence_threshold=0.7):
    ...
    return image_rgb, final_faces
```

4. Модуль навчання нейронної мережі

Цей модуль відповідає за створення архітектури згорткової нейронної мережі та її навчання на підготовленому датасеті. Він реалізований методами create_model і train_model:

```
def create_model(self, learning_rate=0.001):
    ...
    return model
```

1.3.3 Взаємодія між модулями

Уся логіка взаємодії між модулями зберігається всередині класу

FaceDetectionApp, який керує іншими компонентами за допомогою чітко визначених методів. Наприклад, після натискання кнопки «Аналізувати» застосунок запускає аналіз зображення в окремому потоці, не закриваючи інтерфейс програми:

def analyze_image(self):

...

threading.Thread(target=self.perform_analysis, daemon=True).start()

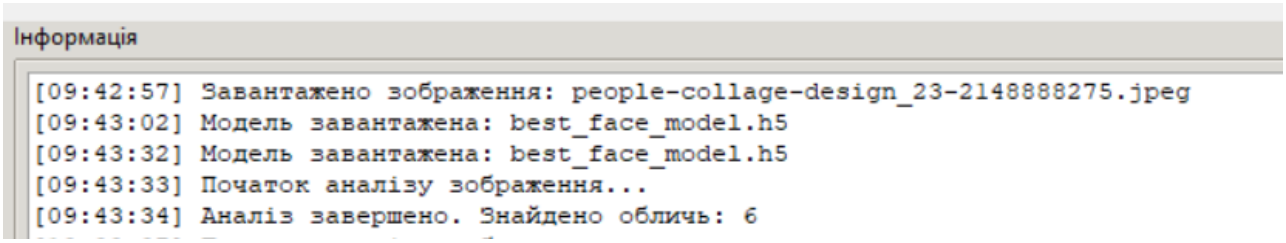


Рисунок 1.21. Поле виведення результатів

Метод perform_analysis викликає detect_faces_in_image із модуля FaceDetectionCNN, а результати передаються в update_results, який оновлює таблицю та зображення:

def update_results(self, faces)

Завдяки застосуванню багатопотоковості через модуль threading, застосунок може виконувати складні обчислення у фоновому режимі, не знижуючи чутливість інтерфейсу. Це критично важливо для забезпечення комфортної взаємодії користувача із системою.

1.4 Підключення необхідних програмних модулів

1.4.1 Імпорт і налаштування бібліотек

Всі необхідні бібліотеки були встановлені у ізольованому віртуальному середовищі, яке було створено у кореневій папці за допомогою команди у терміналі:

python –m venv face_detection_env

Після чого воно було активовано за допомогою компанди:

.\face_detection_env\Scripts\Activate

І вже після активації віртуального середовища за допомогою pip встановлюються необхідні бібліотеки:

pip install tensorflow
pip install opencv–python
pip install numpy
pip install scikit–learn
pip install pillow

Бібліотека tkinter є частиною стандартної бібліотеки Python і не потребує окремого встановлення. Модулі os, threading та datetime також є вбудованими у Python. Налаштування бібліотек здебільшого використовує стандартні параметри, за винятком TensorFlow, де модель конфігурується під час компіляції.

Система розпізнавання облич використовує низку бібліотек Python, кожна з яких виконує певну функцію у процесі розробки моделі, обробки зображень та створення інтерфейсу користувача:

1. TensorFlow

Використовується це як основа для побудови CNN для класифікації облич. Модуль tensorflow.keras забезпечує високорівневий API для створення архітектури моделі, включаючи шари згортки, нормалізації, пулінгу та щільні шари, а також для налаштування оптимізатора Adam та функції втрат binary_crossentropy.

2. OpenCV

Використовується для обробки зображень: завантаження (cv2.imread), конвертація кольорових просторів, зміна розміру (cv2.resize) та застосування класифікатора Haar Cascade для початкового виявлення облич.

3. NumPy

Використовується для роботи з масивами даних зображень та міток під час підготовки даних для навчання.

4. Scikit–learn

Надає функцію train_test_split для поділу даних на тренувальну та валідаційну частини.

5. os

Забезпечує роботу з файловою системою для зчитування зображень із директорій.

6. Tkinter

Створює графічний інтерфейс із меню, кнопками та таблицями результатів.

7. Pillow

Конвертує зображення з масивів NumPy для відображення у Tkinter та масштабує їх.

8. Threading

Дозволяє асинхронне виконання аналізу зображень, щоб уникнути зависання інтерфейсу.

9. Datetime

Datetime використовується для часових міток у логах.

1.4.2 Завантаження та обробка зображень

Завантаження та обробка зображень є ключовими етапами, що забезпечують підготовку даних для CNN, реалізовано у класі Face Detection CNN. Ці кроки є критично важливими, оскільки вони забезпечують коректну підготовку даних для навчання та аналізу моделі CNN, сумісність із вимогами моделі щодо розміру та формату вхідних даних, стабільність навчання завдяки нормалізації, якісне відображення результатів у графічному інтерфейсі для взаємодії з користувачем.

Зображення завантажуються за допомогою cv2.imread:

```
image = cv2.imread(image_path)
if image is None:
    return None
```

Це необхідно для створення набору даних для навчання моделі (позитивні та негативні приклади) та для аналізу зображень у реальному часі. Перевірка if image is None дозволяє пропускати пошкоджені або нечитабельні файли, що забезпечує стабільність обробки великих наборів даних.

Конвертуються з кольорової моделі BGR у RGB:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

OpenCV за замовчуванням завантажує зображення у форматі BGR, а CNN–модель та відображення у графічному інтерфейсі потребують формат RGB, що є стандартним для більшості моделей машинного навчання.

Масштабуються до розмірів 64 на 64 пікселів:

image = cv2.resize(image, target_size)

Зображення масштабуються до фіксованого розміру 64x64 пікселів, що відповідає вхідному розміру моделі (input_shape=(64, 64, 3)). Це необхідно, оскільки CNN вимагає однакового розміру всіх вхідних зображень для коректної обробки через шари згортки та пулінгу.

Нормалізуються до діапазону [0,1]:

image = image.astype('float32') / 255.0

Це покращує стабільність і швидкість навчання моделі, оскільки нормалізовані дані зменшують величину градієнтів під час оптимізації, що сприяє кращій збіжності моделі.

Метод create_training_data обробляє зображення з папок:

```
for filename in os.listdir(positive_dir):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
        img_path = os.path.join(positive_dir, filename)
        img = self.preprocess_image_for_training(img_path)
        if img is not None:
            X.append(img)
            y.append(1)
```

Метод create_training_data створює набір даних для навчання тим чином, що оброблює зображення з директорій dataset/faces (позитивні приклади, мітка 1) та dataset/non_faces (негативні приклади, мітка 0). Це необхідно для створення тренувального набору даних, який містить як зображення з обличчями, так і без них, щоб модель могла навчитися розрізняти ці класи. Окрім цього перевіряє лише допустимі формати файлів для якості даних.

Зображення конвертуються у формат Pillow та масштабуються до 600x400 пікселів:

```
pil_image = Image.fromarray(image_rgb)
pil_image.thumbnail(display_size, Image.Resampling.LANCZOS)
```

У класі FaceDetectionApp зображення конвертуються з масивів NumPy у формат Pillow для відображення у Tkinter. Зміна розміру до 600x400 пікселів за допомогою thumbnail із методом LANCZOS забезпечує якісне масштабування для відображення у графічному інтерфейсі, зберігаючи чіткість зображення. Це

дозволяє користувачу переглядати зображення з рамками навколо виявлених облич у зручному форматі.

1.4.3 Завантаження моделі згорткової нейронної мережі

Завантаження та керування згортковою нейронною мережею (CNN) є ключовим етапом системи розпізнавання облич, оскільки дозволяє використовувати попередньо навчену модель для аналізу зображень, зберігати результати навчання та інтегрувати модель у графічний інтерфейс користувача. Ці процеси реалізовано у класі FaceDetectionCNN, а також частково у класі FaceDetectionApp.

Створення моделі:

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=self.input_shape),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),
    # ... (інші шари)
    layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer=optimizers.Adam(learning_rate=learning_rate),
              loss='binary_crossentropy',
              metrics=['accuracy', 'Precision', 'Recall'])
```

Метод create_model створює архітектуру CNN, яка складається з кількох шарів згортки, нормалізації, пулінгу, dropout та щільних шарів. Це необхідно для визначення структури моделі, яка здатна класифікувати зображення як такі, що містять обличчя (мітка 1) або не містять (мітка 0). Компіляція моделі з оптимізатором Adam, функцією втрат binary_crossentropy та метриками (accuracy, precision, recall) забезпечує можливість навчання моделі для бінарної класифікації та оцінки її якості.

Завантаження моделі:

```
if os.path.exists(filepath):
    self.model = tf.keras.models.load_model(filepath)
else:
```

```
self.model = None
```

Метод load_model завантажує попередньо навчену модель із файлу формату .h5 (наприклад, face_detection_model.h5). Це дозволяє використовувати вже навчену модель без необхідності повторного навчання, що економить час та обчислювальні ресурси, особливо якщо навчання проводилося на великому наборі даних. Перевірка існування файлу (os.path.exists) запобігає помилкам, якщо файл відсутній або пошкоджений, дозволяючи системі коректно обробляти такі випадки та повідомляти користувача про необхідність навчання нової моделі.

Збереження моделі:

```
if self.model:
    self.model.save(filepath)
```

Метод save_model зберігає навчену модель у файл .h5, що дозволяє повторно використовувати її в майбутніх сеансах роботи програми. Це критично важливо для збереження результатів навчання, особливо враховуючи, що процес навчання CNN може бути тривалим і ресурсомістким. Збереження моделі забезпечує можливість її завантаження для подальшого аналізу зображень або донавчання, якщо це необхідно.

Інтеграція з графічним інтерфейсом:

```
file_path = filedialog.askopenfilename(
    title="Оберіть файл моделі",
    filetypes=[("H5 файли", "*.h5"), ("Всі файли", "*.*")]
)
if file_path:
    self.detector.load_model(file_path)
```

У класі FaceDetectionApp метод load_model дозволяє користувачу через графічний інтерфейс вибрати файл моделі за допомогою діалогового вікна. Це забезпечує зручність взаємодії, дозволяючи користувачу легко завантажувати попередньо навчені моделі без необхідності змінювати код чи вводити шляхи до файлів вручну. Інтеграція з інтерфейсом робить систему доступною для користувачів без глибоких технічних знань, що є важливим для практичного застосування.

Обробка помилок:

```
try:
    self.detector.load_model(file_path)
    self.add_info(f"Модель завантажена: {os.path.basename(file_path)}")
    self.status_var.set("Модель завантажена")
except Exception as e:
    messagebox.showerror("Помилка", f"Не вдалося завантажити модель:
{str(e)}")
```

Обробка помилок під час завантаження моделі забезпечує надійність системи. Якщо файл моделі пошкоджений, не існує або має некоректний формат, система відображає повідомлення про помилку через графічний інтерфейс (messagebox.showerror). Це дозволяє користувачу зрозуміти проблему та вжити відповідних заходів, наприклад, обрати інший файл або провести навчання моделі. Такий підхід підвищує стабільність і зручність використання програми.

Ці дії у сукупності забезпечують ефективне керування моделлю CNN, дозволяючи створити нову модель для навчання, якщо попередньо навченої немає, завантажити існуючу модель для економії часу, зберігти результати навчання для повторного використання, інтегрувати модель із графічним інтерфейсом для зручності користувача, обробити помилки для забезпечення надійності системи. Кожен крок відіграє важливу роль у забезпеченні функціональності системи розпізнавання облич.

1.5 Розробка алгоритму пошуку людей на фото

Розробка полягала в створенні згорткової нейронної мережі, яка буде класифікувати невеликі фрагменти зображення за ознакою наявності на них обличчя без використання готових моделей з контролем за навчанням і логікою обробки зображень.

```
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
```

```
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Все починається з роботи вхідного шару нейронної мережі, який приймає зображення розміром 64 на 64 пікселі у відтінках сірого. Для цього застосовуються два згорткові шари з фільтрами 3 на 3, що дозволяє виявляти ознаки, які можуть бути характерними для обличчя, тобто контури очей, носа, або рота. Шари згортки чередуються з шарами субдискретизації, тобто максимального пулінгу, бо вони зменшують розмірність і забезпечують розуміння незначних зміщень на зображенні. Після згортки та випрямлення дані передаються на повнозв'язні шари, які остаточно приймають рішення стосовно того, які фрагменти належать до облич, и чи є такі фрагменти взагалі.

Перед тим, як зображення поступає в мережу, воно розділяється на невеликі фрагменти, тобто вікна, які аналізуються окремо. Для цього є процедура ковзаючого вікна sliding window розміром 64 на 64 пікселі, відповідно до вхідного шару мережі – програма рухається по зображенню з кроком в 16 пікселів, аналізуючи та вирізаючи усі можливі об'єкти, які потім перетворюються на відтінки сірого та масштабуються:

```
def sliding_window(image, step=16, window_size=(64, 64)):
    for y in range(0, image.shape[0] – window_size[1], step):
        for x in range(0, image.shape[1] – window_size[0], step):
            yield (x, y, image[y:y + window_size[1], x:x + window_size[0]])
```

Кожне отримане вікно передається до попередньо натренованої згорткової нейронної мережі. Перед передачею фрагмент нормалізується та змінює форму відповідно до вимог TensorFlow (додається вимір batch та каналів):

```
def prepare_window(window):
    resized = cv2.resize(window, (64, 64))
    gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
    normalized = gray / 255.0
    return normalized.reshape(1, 64, 64, 1)
```

Після застосування моделі до кожного фрагменту ми отримуємо ймовірність того, що саме у окремих вікнах присутнє обличчя. Якщо ймовірність перевищує

визначений поріг, то координати цього фрагмента зберігаються – саме у цьому застосунку за допомогою повзунка можна налаштувати мінімальний поріг для визначення:

```
for (x, y, window) in sliding_window(image):
    input_data = prepare_window(window)
    prediction = model.predict(input_data)[0][0]
    if prediction > 0.9:
        detections.append((x, y, 64, 64, prediction))
```

Оскільки алгоритм може виявити кілька фрагментів, які перекривають одне й те ж обличчя, реалізовано алгоритм Non–Maximum Suppression (NMS) для фільтрації повторів і вибору найбільш ймовірних прямокутників:

```
def non_max_suppression(boxes, overlapThresh=0.3):
    if len(boxes) == 0:
        return []
    boxes = np.array(boxes)
    x1 = boxes[:, 0]
    y1 = boxes[:, 1]
    x2 = boxes[:, 0] + boxes[:, 2]
    y2 = boxes[:, 1] + boxes[:, 3]
    scores = boxes[:, 4]
    idxs = np.argsort(scores)[::–1]
    selected = []
    while len(idxs) > 0:
        i = idxs[0]
        selected.append(boxes[i])
        suppress = [0]
        for j in range(1, len(idxs)):
            xx1 = max(x1[i], x1[idxs[j]])
            yy1 = max(y1[i], y1[idxs[j]])
            xx2 = min(x2[i], x2[idxs[j]])
            yy2 = min(y2[i], y2[idxs[j]])
            w = max(0, xx2 – xx1)
            h = max(0, yy2 – yy1)
            overlap = (w * h) / ((x2[i] – x1[i]) * (y2[i] – y1[i]))
            if overlap > overlapThresh:
```

```
            suppress.append(j)
        idxs = np.delete(idxs, suppress)
    return selected
```

Після очищення списку прямокутників виводиться фінальний результат на зображенні. Для цього за допомогою OpenCV поверх вихідного зображення накладаються кольорові рамки:

```
for (x, y, w, h, _) in non_max_suppression(detections):
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.imshow("Result", image)
cv2.waitKey(0)
```

Тобто повний цикл обробки зображення містить поділ на вікна, обробку кожного фрагмента, передачу готових фрагментів до моделі, фільтр результатів і вивід фінального зображення.

1.6 Розробка UML–діаграми класів

Для кращого розуміння архітектури розробленої системи розпізнавання облич було створено UML–діаграму класів, яка відображає основні компоненти системи та їх взаємозв'язки.

Діаграма демонструє три ключові класи системи:

1. FaceDetectionApp – головний клас системи, що відповідає за графічний інтерфейс та взаємодію з користувачем. Клас містить методи для:

– завантаження та відображення зображень

– керування процесом розпізнавання облич

– візуалізації результатів

2. FaceDetectionCNN – клас, що реалізує логіку роботи з нейронною мережею. Основні функції включають:

– створення та навчання моделі CNN

– попередню обробку зображень

– виявлення облич на зображеннях

3. TrainingDialog – допоміжний клас для організації процесу навчання моделі, який забезпечує:

– налаштування параметрів навчання

– візуалізацію процесу навчання

FaceDetectionApp використовує об'єкт класу FaceDetectionCNN для виконання основних операцій розпізнавання, потім FaceDetectionApp створює об'єкт TrainingDialog при необхідності навчання моделі, TrainingDialog має зворотній зв'язок з FaceDetectionApp для оновлення стану системи.

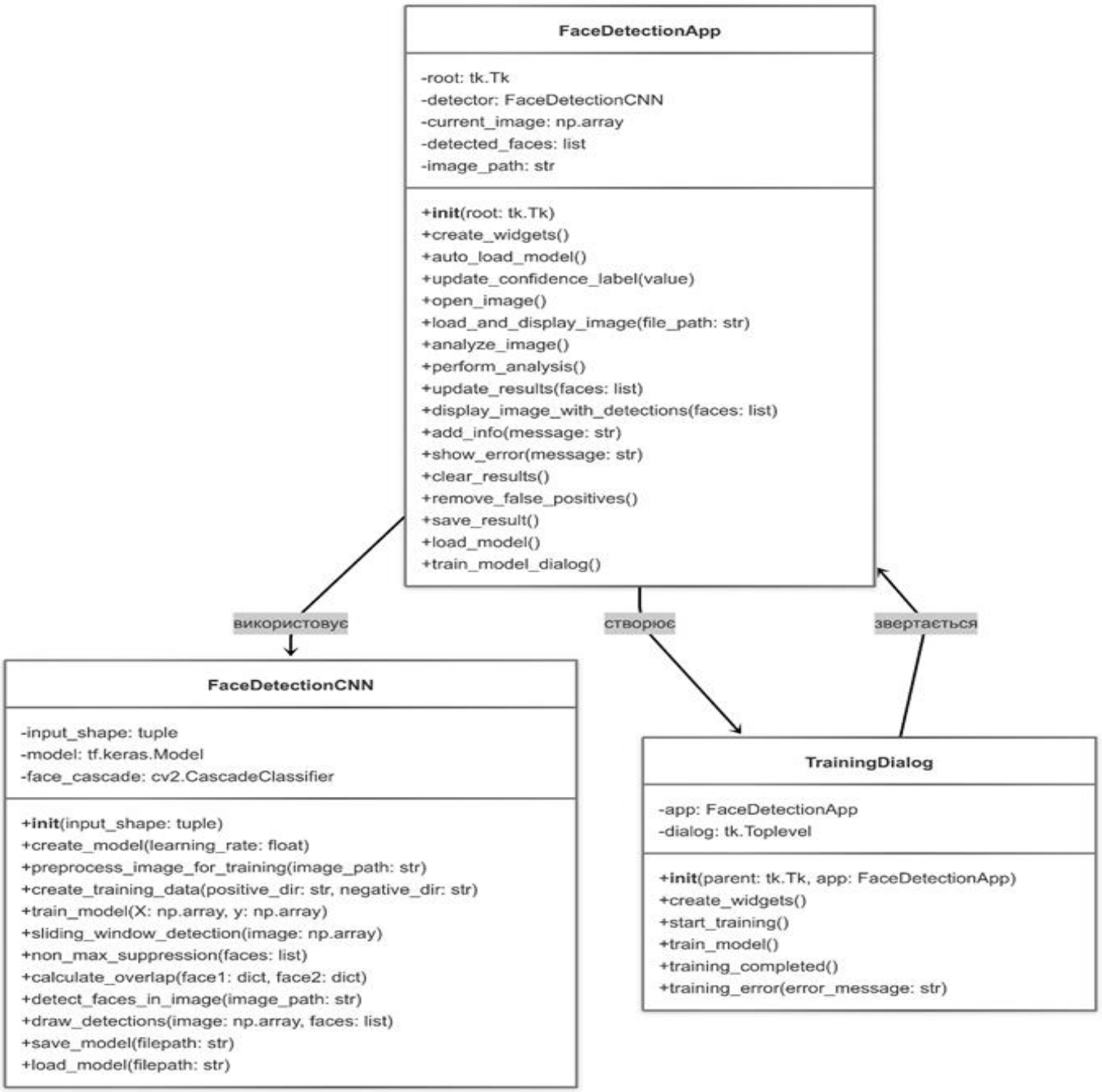


Рисунок 1.21. UML-діаграма класів застосунку

1.7 Реалізація застосунку

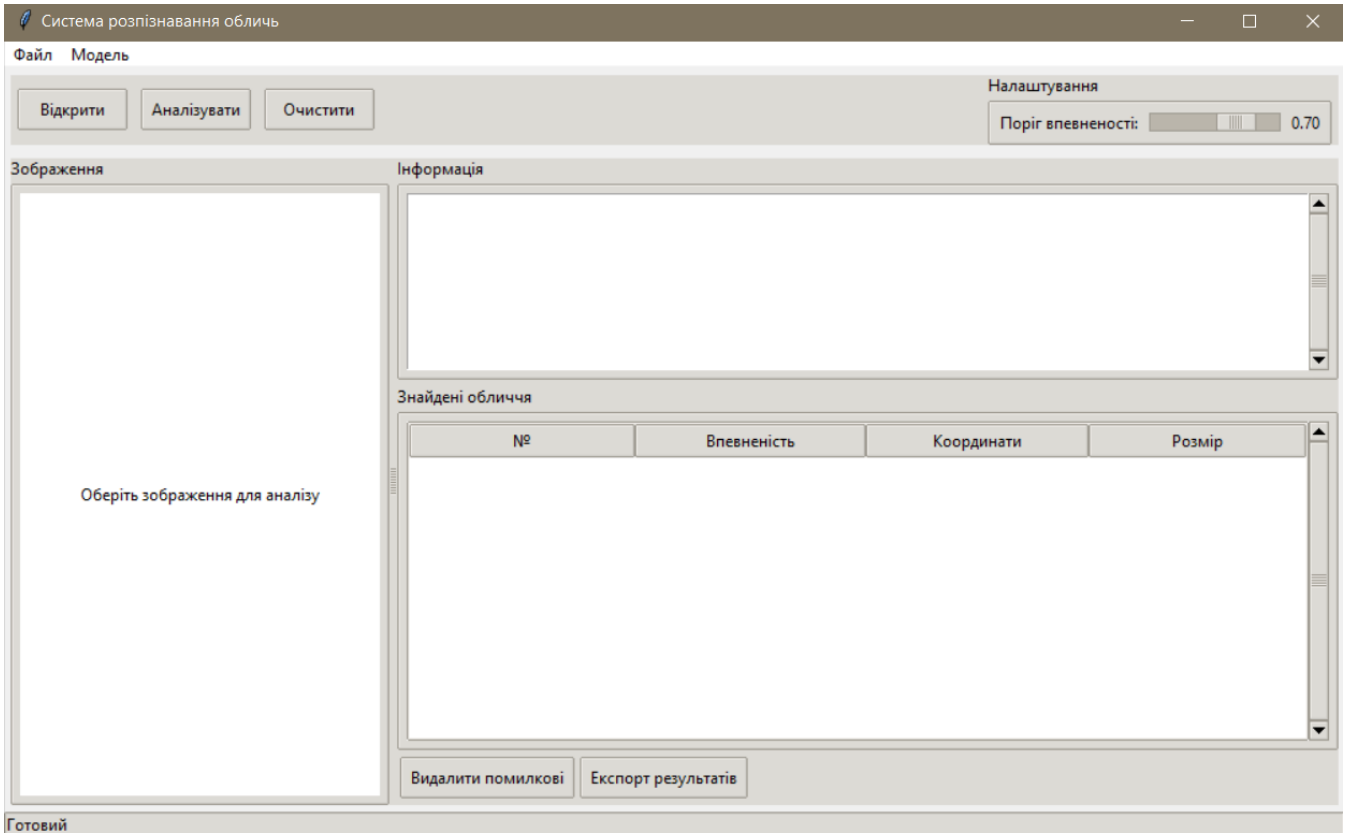


Рисунок 1.22. Інтерфейс застосунку

Застосунок для розпізнавання облич запускається як desktop–додаток із графічним інтерфейсом, створеним за допомогою Tkinter. Після запуску він дозволяє користувачу завантажувати зображення, аналізувати їх на наявність облич, переглядати результати у вигляді рамок навколо виявлених облич та інформації в таблиці, а також зберігати результати аналізу. Окрім цього, користувач може запустити вже наявну модель нейронної мережі, або навчити свою з необхідними налаштуваннями.

Після виконання скрипта (main() у face_detection_app.py), створюється головне вікно Tkinter із розмірами 1200 на 800 пікселів та заголовком "Система розпізнавання облич" (рис. 1.22).

Графічний інтерфейс складається з меню, панелі інструментів, області для відображення зображень, інформаційного текстового поля та таблиці результатів. Статусний рядок у нижній частині вікна відображає поточний стан програми (наприклад, "Готовий").

Після запуску користувач взаємодіє із застосунком через елементи графічного інтерфейсу:

1. Меню «Файл»:

Меню "Файл" дозволяє відкрити зображення, зберегти результати аналізу або вийти з програми.

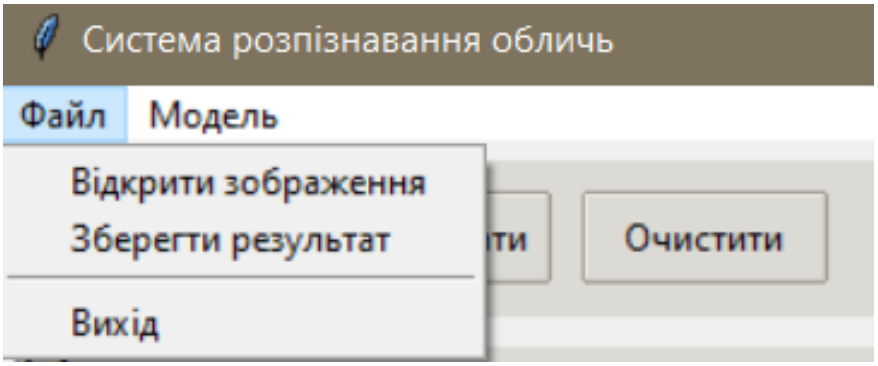


Рисунок 1.23. Меню «Файл»

2. Меню «Модель»:

Застосунок ініціалізує об’єкт класу FaceDetectionCNN для обробки зображень і намагається завантажити попередньо навчену модель із файлу face_detection_model.h5. Якщо модель не знайдена, вона залишається неініціалізованою, і користувач може або завантажити іншу модель, або навчити нову.

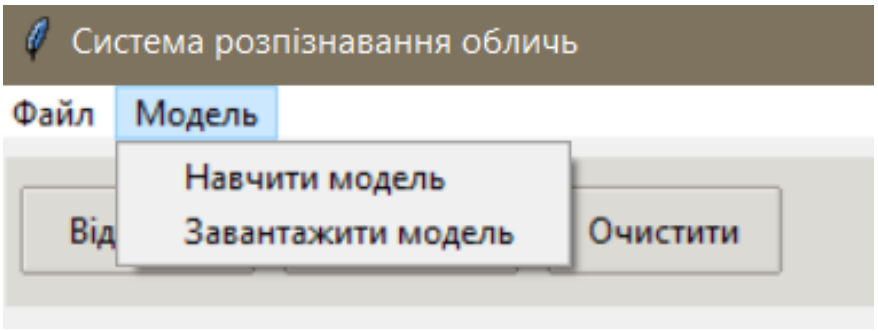


Рисунок 1.24. Меню «Модель»

2. Панель інструментів:

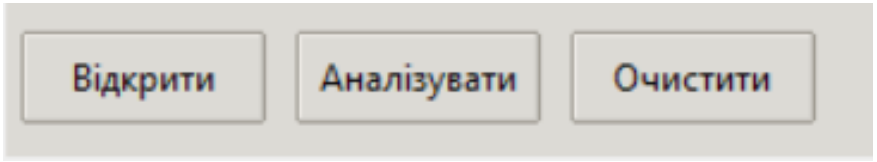


Рисунок 1.25. Кнопки панелі інструментів

Кнопки: "Відкрити" (для завантаження зображення у застосунок),

"Аналізувати" (запуск виявлення облич), "Очистити" (скидання результатів виявлених облич).

4. Повзунок для налаштування порогу впевненості (confidence_threshold, за замовчуванням 0.7), який визначає мінімальну ймовірність для визнання області як обличчя.

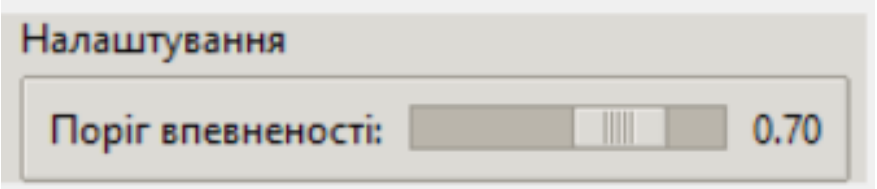


Рисунок 1.26. Повзунок налаштування впевненості

5. Область зображення:

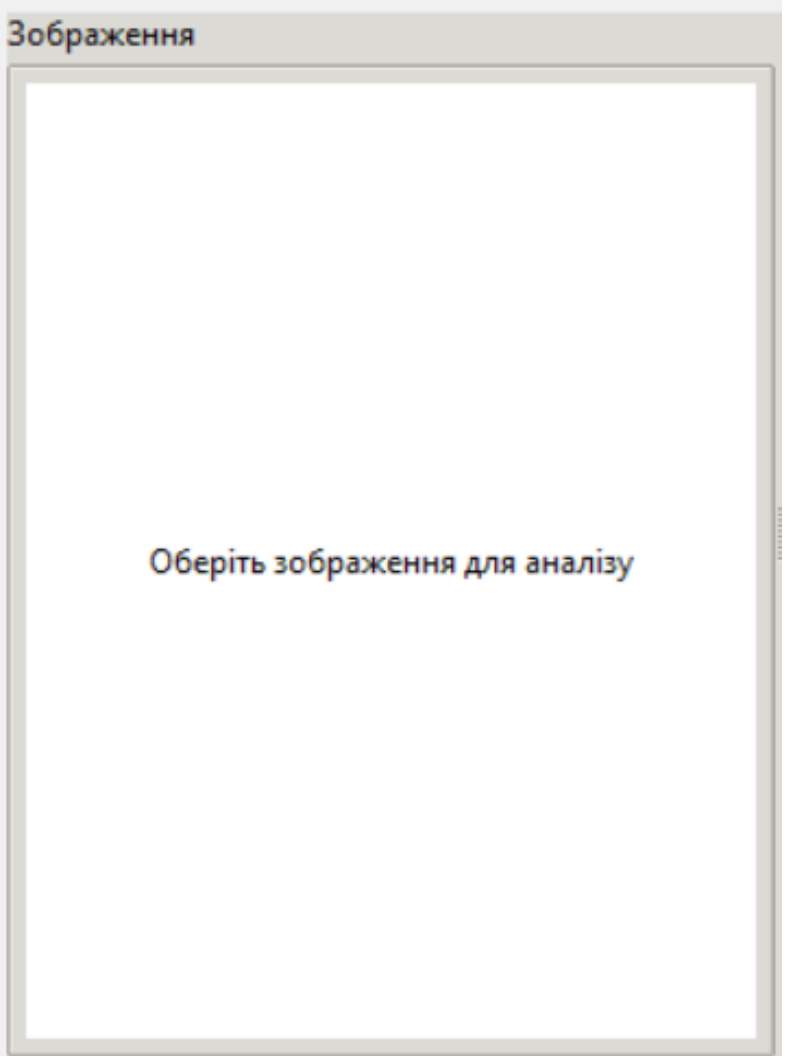


Рисунок 1.27. Пусте вікно аналізу зображення

Відображає завантажене зображення з рамками навколо виявлених облич після аналізу.

6. Інформаційне поле:

Логує події (наприклад, "Завантажено зображення", "Аналіз завершено") із часовими мітками.

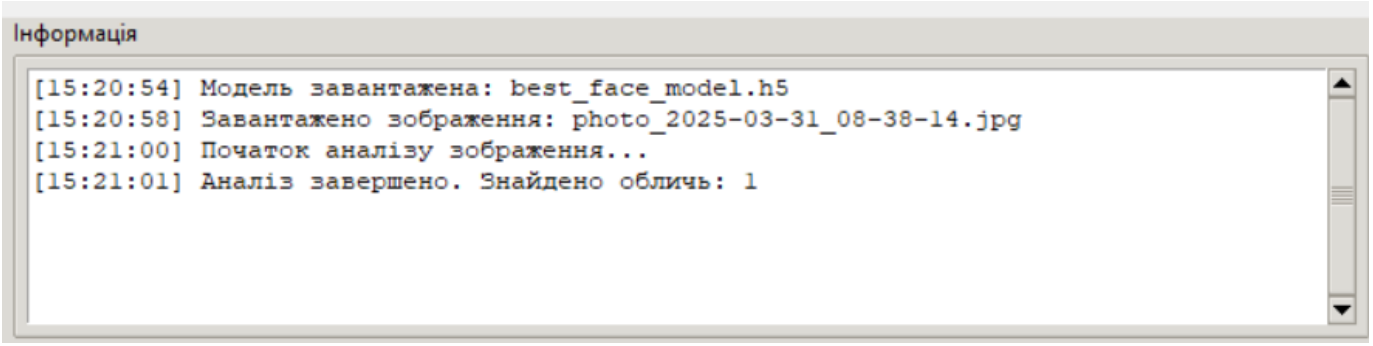


Рисунок 1.28. Інформаційне поле з подіями

7. Таблиця результатів:

Показує список виявлених облич із номером, впевненістю, координатами та розмірами.

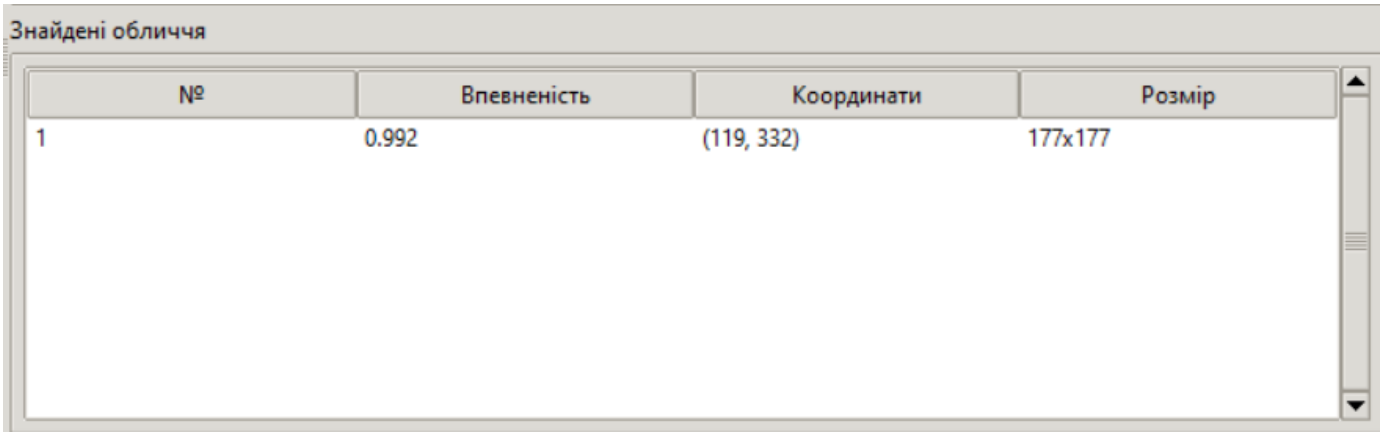


Рисунок 1.29. Таблиця результатів

Користувач може взаємодіяти з результатами через додаткові функції:

1. Видалення помилкових виявлень:

Користувач вибирає обличчя в таблиці та натискає "Видалити помилкові", вибрані записи видаляються з таблиці, і зображення оновлюється, прибираючи рамку видалених облич із зображення.

2. Збереження зображення:

Дозволяє зберегти оброблене зображення з накладеними рамками навколо розпізнаних облич, а також із зазначенням рівня впевненості моделі для кожного з них у форматі JPG або PNG.

1.8 Тестування застосунку на різних фото

1.8.1 Приклади тестів та результати

Застосунок тестувався на різних зображеннях формату JPEG, які містили в собі фотографії людей різних вікових груп, національностей та статей, а також у різних ракурсах. Першим використаним для тесту зображенням була фотографія однієї дівчини (рис. 1.30), яка була трохи повернена убік у полі серед купи квітів, і серед усіх об’єктів застосунок знайшов її обличчя з впевненістю у 0,949, тобто 94,9%.

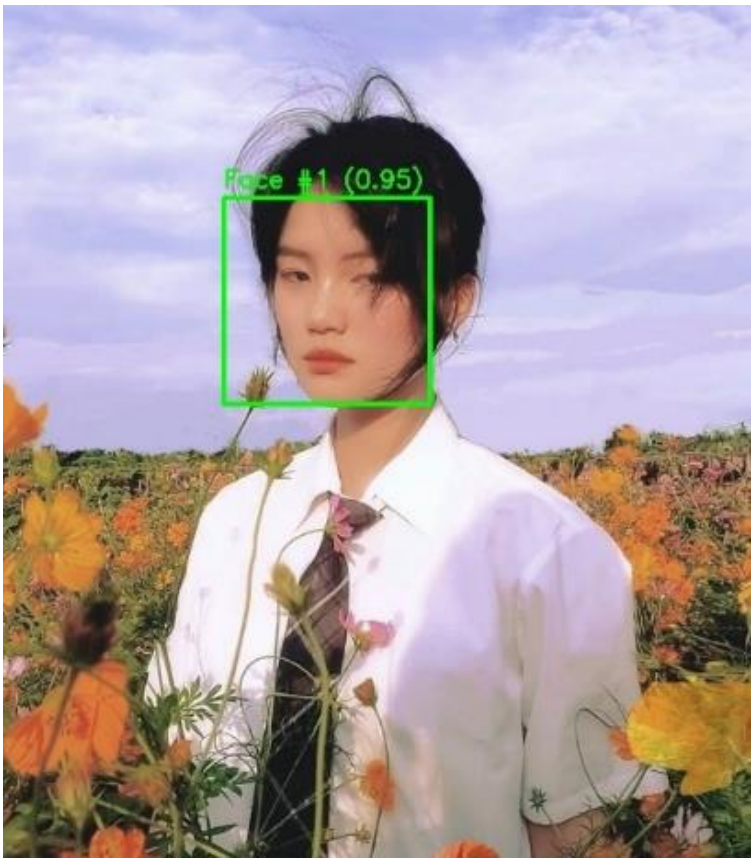


Рисунок 1.30. Перший тест: одна дівчина в полі квітів

Так само застосунок добре знаходить обличчя, якщо на фотографії знаходиться декілька людей різної зовнішності, особливо якщо вони стоять прямо, як на рисунку 1.31: майже усі обличчя на зображені були знайдені з високою впевненістю того, що на фрагменті є саме обличчя – впевненість була у диапазоні від 0,999 до 0,979, тобто від 97,9% до 99.9%.

На іншому зображенні (рис. 1.32), де знаходяться 9 людей, кожен на різній дистанції від камери, обличчя були знайдені вже не так чітко, але все ще з високим

порогом впевненості – найвищий поріг сягав впевненість 100%, коли найменшим показником було 72%, що є нормальним середнім показником для знаходження обличчя. Всі інші обличчя також були знайдені з високим порогом від 96,4% до 99,9%.

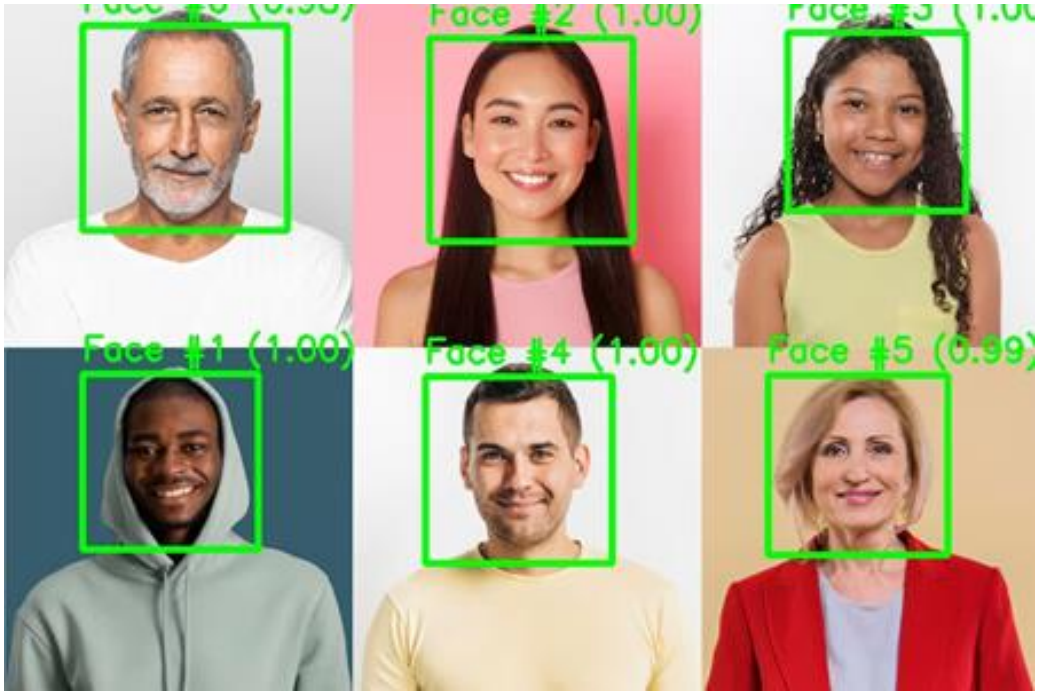


Рисунок 1.31. Другий тест: шестеро людей різних категорій

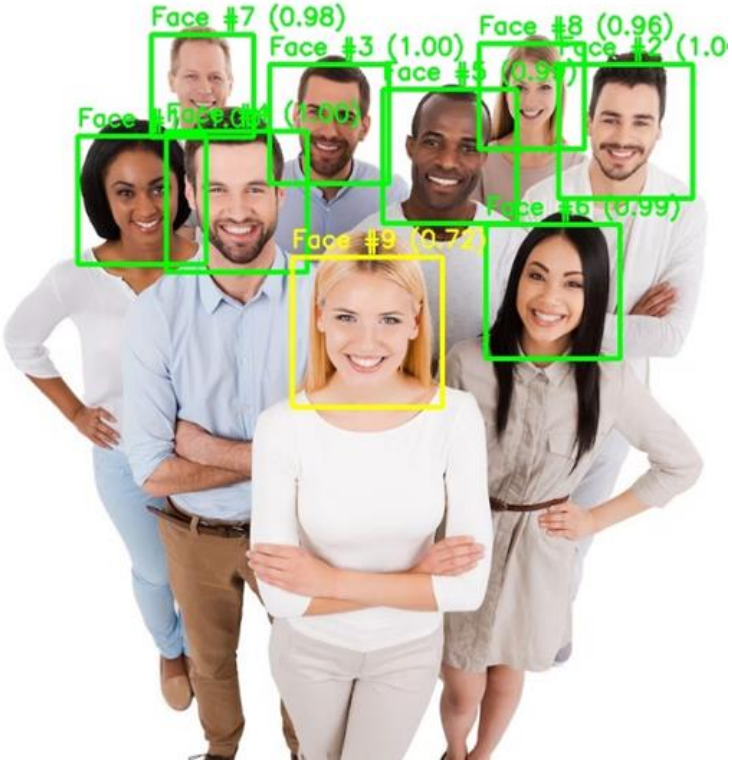


Рисунок 1.32. Третій тест: дев’ятеро людей різних категорій

Різна міміка, як на рисунку 1.33, також добре розпізнається як обличчя – кожне було знайдено з діапазоном впевненості від 98,5% до 100%.

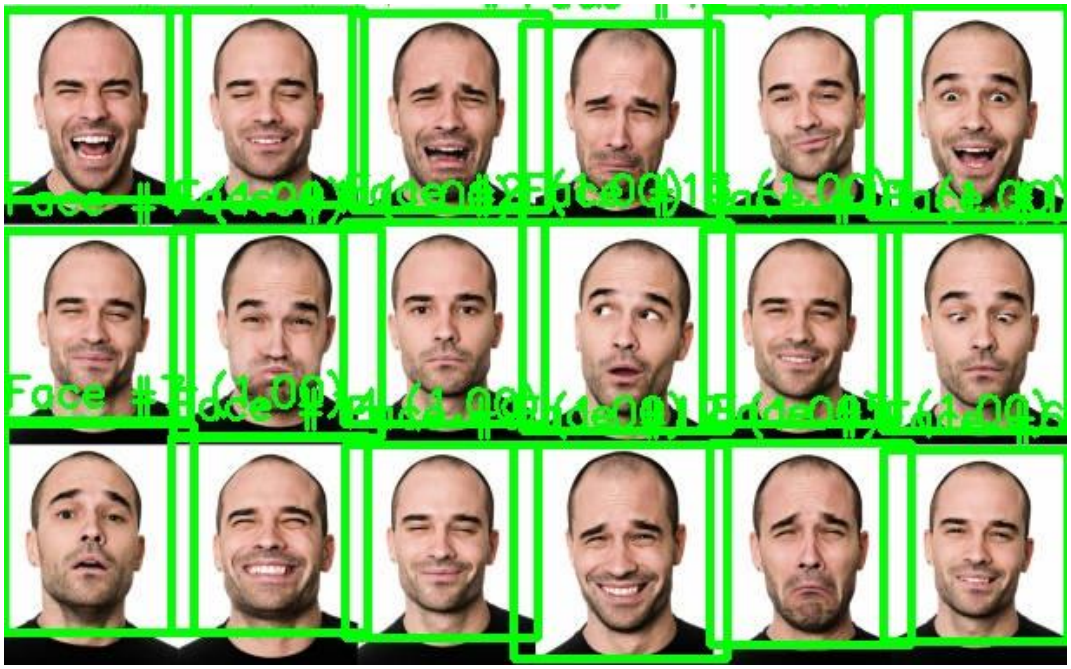


Рисунок 1.33. Четвертий тест: багато фотографій однієї людини з різною мімікою

Коли ж для тестування було взято зображення з тваринами чи іншими елементами, застосунок після аналізу не знаходив жодного обличчя з відповідним результатом у вікні «інформації» (рис. 1.34).

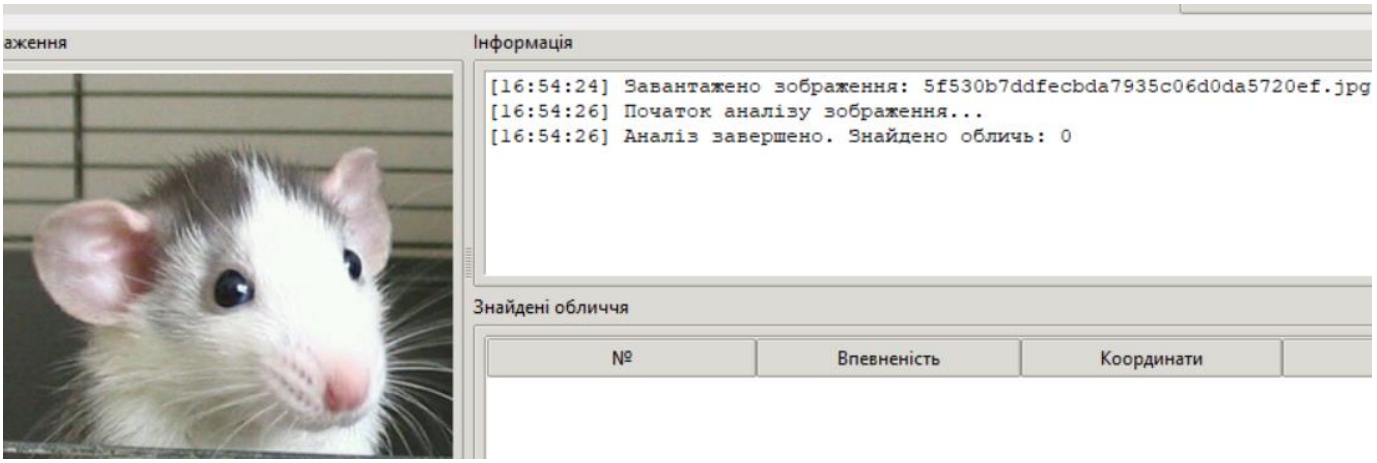


Рисунок 1.34. П’ятий тест: фотографія миші

1.8.2 Аналіз помилок

Застосунок не знаходив обличчя людей, які було складно розпізнати – тобто, якщо людина носила аксесуар на обличчі, який закривав його, або вона була повністю розвернута вбік, її обличчя не вважалося застосунком за «обличчя»

навіть з мінімальним порогом впевненості.

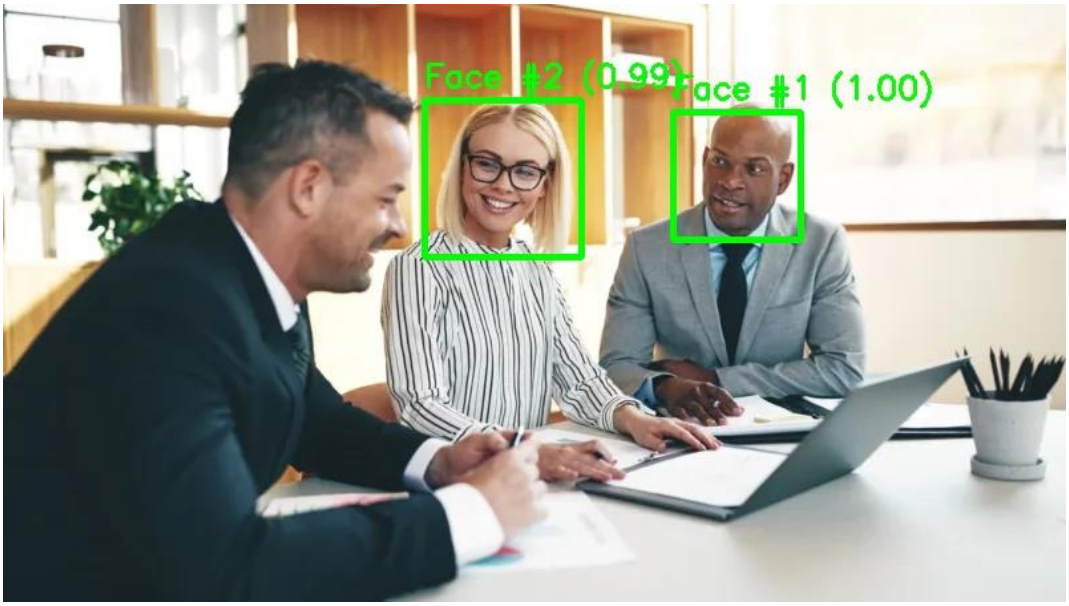


Рисунок 1.35. Шостий тест: дві людини, одна розгорнута

Це може бути пов’язано з тим, що серед бази даних відсутні приклади облич, які би були повернені так само, або з тим, що модель недостатньо навчалась протягом відзначеної кількості епох (у випадку даного тестування це 20 епох – саме універсальне значення, щоб модель встигла достатньо навчитись, але і не навчалась занадто довго) не навчилась знаходити подібні обличчя. Але це навпаки є плюсом, бо нейронна мережа має знаходити саме чіткі обличчя, і під час тестування вони були знайдені з великою впевненістю в 99,7% та 99,2%.

Окрім цього, тестування проводилось з тим, яка кількість епох краща. Початкова кількість епох в 50 опинилася занадто великою, через що застосунок виділяв ділянки без облич на них, а при меншій кількості епох (в даному випадку 10) модель не встигала сформувати достатньо узагальнених ознак, через що точність розпізнавання значно знижувалась, і при створенні моделі нейронної мережі прийшлося зупинитися на середньому значенні в 20 епох, яке виявилося оптимальним.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проекті розроблено програмний продукт (ПП) – застосунок реалізації алгоритмів пошуку людей на знімках з використанням нейромереж.

Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам'яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

Проведемо розрахунки визначення трудомісткості розробки даного програмного продукту.

2.2 Розрахунок ціни програмного продукту нормативним методом

2.2.1 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку.

Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога.

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт.

Таблиця 2.1. Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл. 2.2:

Таблиця. 2.2. Таблиця трудомісткості

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244

На підставі отриманого значення, по таблиці 2.2 визначається укрупнена норма часу на розробку аналога програмного забезпечення, яка коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8:

$$T^a p = 244 \times 0,8 = 195,2 \text{ (люд/годин).}$$

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

Розробка технічного завдання

$$T_{ТЗ} = T^a p \times L_1 \times K_H \qquad (2.1)$$

Розробка технічного проекту

$$T_{ТП} = T^a p \times L_2 \times K_H \qquad (2.2)$$

Розробка робочого проекту

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \qquad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і–го етапу розробки (табл. 2.3.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (табл. 2.4.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (табл. 2.5.).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Тому що розробка системи є ПО, що має аналоги програмних продуктів, то код ступеня новизни для мого ПО – В, а значення коефіцієнта $К_н$= ?. По таблиці 2.4, знаючи код ступеня новизни, тепер можна визначити значення питомих коефіцієнтів трудомісткості (по таблиці 2.3):

$L_1 = 0,12$

$L_2 = 0,11$

$L_3 = 0,61$

Таблиця 2.4. Значення поправочного коефіцієнта по ступені новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $К_т$
60 і вище	0,6
40–60	0,7

У розробленому програмному продукті використовується від 60 до 100 відсотків існуючих функцій, це значить, що $К_т$= 0,6. Тепер потрібно розрахувати трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$T_{тз} = T^a p * L_1 * К_н = 195,2*0,12*0,7 = 16,4 \text{ (люд/годин)} \quad (2.1)$$

Трудомісткість розробки технічного проекту

$$T_{тп} = T^a p * L_2 * К_н = 195,2*0,11*0,7 = 15,03 \text{ (люд/годин)} \quad (2.2)$$

Трудомісткість розробки робочого проекту

$$T_{рп} = T^a p * L_3 * К_н * К_т = 195,2*0,61*0,7*0,6 = 50,01 \text{ (люд/годин)} \quad (2.3)$$

Для подальших розрахунків необхідно визначити кількість папера, витраченого на кожен етап. $N_{тз}$= 2 (стр), $N_{тп}$= 28 (стр), $N_{рп}$= 22 (стр), $N_{пз}$= 66 (стр) – технічне завдання, розробка технічного проекту, розробка робочого проекту, пояснювальна записка відповідно. Розрахунок зведений у таблицю 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, години.		
	Розробка ПП	Контроль керівника	Нормоконтроль
1.ТЗ	$Т_{Ртз}$=16,4	$Т_{кк}$=0,7*$N_{тз}$= 1,4	$Т_{нк}$=0,15*$N_{тз}$= 0,3
2.Розробка ТП	$Т_{Ртп}$=15,03	$Т_{кк}$=0,7*$N_{тп}$=19,6	$Т_{нк}$=0,15*$N_{тп}$= 4,12
3.Розробка РП	$Т_{Ррп}$= 50,01	$Т_{кк}$=0,7*$N_{рп}$= 15,4	$Т_{нк}$=0,15*$N_{рп}$= 3,3
4.Розробка пояснювальної записки	$Тр_{пз}$=1,5*$N_{пз}$= 99	$Т_{кк}$=0,7*$N_{пз}$= 46,2	$Т_{нк}$=0,15*$N_{пз}$= 9,9
Усього, в т.ч.:	$Т_{пп}$=0,133495		
– на розробку	$\Sigma Т_р$ = 180,44		
– контроль керівника		$\Sigma Т_{кк}$ = 82,6	
– нормоконтроль			$\Sigma Т_{нк}$ = 17,62

На основі таблиці 2.6 розрахуємо тривалість розробки в роках:

$$Tпп = T/(8.0*0.73*360) = 0,133495 \text{ (р)}, \quad (2.4)$$

Де 8,0 – тривалість робочого дня, 0,73 – коефіцієнт перекладу в календарні дні;

2.2.2 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 6.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2024» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2025 року – 8000 гривень; мінімальну погодинну тарифну ставку – 46 грн.

Таблиця 2.7 Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість робіт, роб.години	Годинна тарифна ставка,грн.	Розрахунок, грн.
1.Розробка ПП	$\Sigma T_р$=180,44	56	10104,64
2.Контроль керівника	$\Sigma T_{кк}$=82,6	90	7434
3.Нормоконтроль	$\Sigma T_{кк}$=17,62	90	1585,8
Усього ($З_о$)			Σ Зо= 19124,44

Розрахунок основної заробітної плати виконуємо по формулі:

$$Зо = Тмj \times Згод, \qquad (2.5)$$

Де $Т_{мj}$ – трудомісткість j – того виду робіт, робоч. год, Згод – погодинна тарифна ставка, грн.

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 2.8

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	Тип, модель	Кількість, шт	Ціна одиниці, грн.	Вартість, грн.
Папір А4	Лист А4	18	4.0	72
Разом	–	–	–	$В_{м\,і}$=72
Транспортно – заготівельні Витрати 10%				$Втр\_з = 0{,}1 \times Вм1 =$ 7,2
Усього				$В_м=В_{м1}+В_{тр\_з=}$ 79,2

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	79,2	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	19124,44	$З_о$ (див. табл. 2.7)
3. Додаткова заробітна плата	1912,44	$Зд = 0{,}1 \times Зо =$
4. Відрахування до єдиного фонду соціального внеску	4628,12	$Вє.с.в. = 0{,}22 \times (Зо + Зд) =$
5. Накладні витрати	11474,66	$Внак. = 0{,}6 \times Зо =$
6. Повна собівартість	37218,86	$С_{пов} = В_м + З_о + З_д + Вє.с.в. + Внак.$

Розмір прибутку, що включається в ціну, визначається по наступній формулі:

$$П=(С_{пов}*Р)/100 = 3721{,}89 \ (грн); \quad (2.6)$$

Де Р – плановий рівень рентабельності (10–15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о=С_{пов}+П= 40940{,}75 \ (грн); \quad (2.7)$$

Податок на додану вартість визначається по наступній формулі:

$$ПДВ = 0.2 * Ц_о = 8188{,}15 \ (грн); \quad (2.8)$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$$Ц_р=Ц_о+ПДВ= 48678{,}9 \ (грн); \quad (2.9)$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Охорона праці є однією з ключових складових забезпечення безпечних умов роботи, особливо в умовах сучасних інформаційних технологій, де основним робочим інструментом є комп'ютер. Робота над розробкою алгоритмів пошуку людей на фотознімках із використанням нейронних мереж передбачає тривале перебування за комп'ютером, що може супроводжуватися впливом фізичних і психофізіологічних факторів. Основна мета охорони праці – аналіз небезпечних і шкідливих факторів, які можуть виникати під час виконання такої роботи, та розробка заходів для їх мінімізації, щоб забезпечити безпеку, збереження здоров'я та працездатності працівника.

Охорона праці включає комплекс організаційних, технічних і санітарно–гігієнічних заходів, спрямованих на створення комфортних умов праці, зниження ризику професійних захворювань і травматизму. У контексті розробки програмного забезпечення особлива увага приділяється ергономіці робочого місця, нормам освітлення, мікроклімату, електробезпеці та протипожежним заходам. У цьому розділі розглядаються основні аспекти безпеки праці програміста, який працює над створенням та тестуванням алгоритмів нейронних мереж, а також рекомендації щодо організації робочого місця та забезпечення пожежної безпеки.

3.1 Аналіз небезпечних і шкідливих факторів на робочому місці

Робота програміста, який займається розробкою алгоритмів пошуку людей на фотознімках із використанням нейронних мереж, зазвичай виконується в офісному приміщенні за комп'ютером. Основними небезпечними та шкідливими факторами, які можуть впливати на працівника, є фізичні та психофізіологічні фактори виробничого середовища.

Серед найпоширеніших шкідливих фізичних факторів є електромагнітне випромінювання. Комп'ютери, монітори та інші електронні пристрої генерують електромагнітні поля в діапазонах частот від 5 Гц до 400 кГц. Тривалий вплив

таких полів може викликати втому, головний біль і порушення концентрації. У офісних приміщеннях може бути недостатнє або надмірне освітлення, наприклад, неправильно організоване штучне освітлення (надто тьмяне або надмірно яскраве) яке може спричинити подальше перенапруження зору, що особливо актуально під час роботи з графічними даними, такими як фотознімки. Окрім цього, комп'ютери та сервери, які використовуються для тренування нейронних мереж, виділяють тепло, що може підвищувати температуру в приміщенні. Знижена вологість повітря (менше 40%) сприяє накопиченню статичної електрики, що може впливати на самопочуття працівника, спричинити дискомфорт або навіть незначні електростатичні розряди. Робота вентиляторів комп'ютерів і серверів може створювати підвищений рівень шуму, що негативно впливає на концентрацію та викликає втому.

До психофізіологічних факторів належать фізичні фактори, такі як перенапруження зору через тривале фокусування на екрані монітора під час аналізу фотознімків або налагодження алгоритмів, яке може викликати синдром комп'ютерного зору, що проявляється у сухості очей, почервонінні та зниженні гостроти зору, або зниження концентрації та підвищеної втомлюваності через одноманітну роботу над кодом або аналізом даних, а також нервово–психічні, які можуть проявлятися як стрес, розмовну напругу або емоційне вигорання через нагружений графік, необхідність вирішення складних алгоритмічних завдань або надмірний контакт з клієнтами.

3.2 Заходи з охорони праці

Для забезпечення безпечних умов праці необхідно впроваджувати заходи, які мінімізують вплив зазначених факторів. Основна увага приділяється організації робочого місця, підтриманню оптимального мікроклімату, правильному освітленню та дотриманню режиму праці й відпочинку.

Робоче місце програміста має відповідати ергономічним і санітарно–гігієнічним вимогам, визначеним у ДСН 3.3.6.042–99: екран має бути розташований на відстані 60–90 см від очей, верхній край екрана – на рівні очей

або нижче. Частота мерехтіння екрана повинна бути не менше 70 Гц (рекомендується 100 Гц). Стілець і стіл повинні мати можливість регулювання висоти, щоб забезпечити зручну позу. Сидіння має підтримувати природний вигин хребта, а стіл – мати матову поверхню для зменшення відблисків. Робоче місце має бути розташоване перпендикулярно до вікон, щоб уникнути відблисків на екрані від природного або штучного освітлення.

Освітлення в приміщенні має відповідати нормам, визначеним у ДБН В.2.5–28:2018. Для офісних приміщень рекомендується використання системи загального рівномірного освітлення з рівнем освітленості 300–500 ЛК, у разі роботи з документами або фотознімками допускається комбіноване освітлення з додатковими світильниками місцевого освітлення. Світильники повинні мати розсіювачі для запобігання відблискам.

Для забезпечення комфортних умов праці необхідно підтримувати оптимальні параметри мікроклімату: температуру повітря на рівні 22–25°C у холодний період року, і 23–26°C у теплий період, відносну вологість 40–60%, швидкість руху повітря не повинна перевищувати 0,1 м/с. У приміщеннях із великою кількістю комп’ютерів або серверів необхідно встановлювати системи кондиціонування та зволоження повітря. Якщо ці норми перевищені, робочий день скорочується на 10% відповідно до норм.

Щоб зменшити вплив психофізіологічних факторів на працівників, рекомендується робити перерви кожні 45–60 хвилин роботи за комп’ютером тривалістю 10–15 хвилин, виконувати вправи для очей (наприклад, переведення погляду на далекі об’єкти, моргання) для профілактики синдрому комп’ютерного зору і забезпечувати періодичне провітрювання приміщення для підтримання якості повітря.

3.3 Пожежна безпека

Забезпечення пожежної безпеки є важливою частиною створення безпечних умов праці, особливо в приміщеннях із великою кількістю електроприладів.

Аварійні виходи мають мати вільній доступ, проходи до виходів повинні

бути шириною не менше 1 м і не захаращуватися. Усі комп'ютери, сервери та інші пристрої повинні використовуватися відповідно до їх призначення, і у разі виявлення несправностей обладнання необхідно негайно відключити від електромережі. Папір, картон та інші горючі матеріали повинні зберігатися в спеціальних шафах на відстані не менше 1 м від електрощитів і 0,6 м від датчиків пожежної сигналізації.

У приміщенні мають бути розміщені первинні засоби пожежогасіння для боротьби з пожежами на початковій стадії, такі як: порошкові або вуглекислотні вогнегасники (не менше двох на поверх із масою заряду 5 кг і більше), пожежні щити, обладнані лопатами, відрами, піском і сокирами, які розміщуються з розрахунку один щит на 5000 м2, пожежні крани, підключені до системи водопостачання. Експлуатація приладів без призначення відповідального за організацію не допускається.

У приміщеннях, де розташовані комп'ютери, сервери або інше обладнання, паління категорично заборонено через ризик займання горючих матеріалів (наприклад, паперу, який може використовуватися для документації) та можливість пошкодження обладнання через попіл або дим. На підприємстві мають бути облаштовані відведені зони для паління, які будуть розташовані на відкритому повітрі або в ізольованих приміщеннях із потужною вентиляцією, знаходитись на безпечній відстані (не менше 10 м) від місць зберігання легкозаймистих матеріалів, електрощитів або входів до будівлі, бути обладнаними негорючими попільницями та контейнерами для недопалків, які регулярно очищаються. і позначеними відповідними знаками, наприклад, "Місце для паління".

Щоб запобігти розповсюдження пожежі, приміщення має бути обладнане автоматичною пожежною сигналізацією, наприклад, датчики диму та тепла. звукові сповіщувачі, аварійні кнопки для виклику пожежної охорони або приймально–контрольну панель для обробки сигналів. Система пожежної сигналізації повинна регулярно перевірятися ліцензованою організацією (не рідше одного разу на місяць).

Якщо ситуації, у якій виникла пожежа, не удалось уникнути, необхідно:

1. Негайно повідомити пожежну охорону за номером 101, вказавши адресу, місце виникнення пожежі, кількість поверхів, наявність людей і своє прізвище.

2. Організувати евакуацію людей і цінного обладнання.

3. Вимкнути електроживлення та вентиляцію, якщо це безпечно.

4. Розпочати гасіння пожежі первинними засобами до прибуття пожежної охорони.

5. Забезпечити зустріч підрозділів пожежної охорони та надати їм необхідну інформацію.

Також важливо проводити регулярні інструктажі з охорони праці та пожежної безпеки, особливо при впровадженні нових технологій або зміні умов праці. Працівники мають бути ознайомлені з алгоритмами дій у надзвичайних ситуаціях, вміти користуватися вогнегасниками та знати розташування аварійних виходів.

Упровадження систематичного підходу до безпеки праці дозволяє не лише знизити ризики для здоров’я, але й підвищити продуктивність працівників, що сприяє ефективній реалізації проєктів у сфері інформаційних технологій. У сфері IT, де якість програмного продукту безпосередньо залежить від уваги, зосередженості та самопочуття розробника, створення безпечного й комфортного робочого середовища є не лише обов’язком, а й важливим стратегічним ресурсом підприємства.

ВИСНОВКИ

У ході виконання даного дипломного проєкту було розроблено та впроваджено застосунок для розпізнавання облич на фотографіях із використанням згорткової нейронної мережі. Основні результати та досягнення проєкту можна узагальнити наступними пунктами.

Було створено програмний продукт, який виконує автоматичне виявлення та ідентифікацію облич на зображеннях. У якості ядра системи застосовано модель згорткової нейронної мережі, попередньо навченої на локальному наборі даних. Для реалізації системи використано сучасні засоби Python–програмування, зокрема TensorFlow для машинного навчання, OpenCV для обробки зображень та PyQt5 для побудови графічного інтерфейсу, що дозволяє завантажувати зображення, здійснювати аналіз і переглядати результати розпізнавання у візуальному вигляді.

Особливу увагу приділено точності розпізнавання: завдяки застосуванню алгоритмів попередньої обробки та фільтрації хибних спрацьовувань вдалося досягти високого рівня достовірності результатів. Модель проходила локальне навчання, а результати тестування підтвердили її ефективність у практичних умовах. У рамках проєкту проведено тестування системи на різноманітних прикладах зображень. Тестування підтвердило стабільність роботи застосунку, його здатність адаптуватися до різних типів зображень та справно функціонувати при змінному навантаженні.

У дипломному проєкті також виконано економічне обґрунтування розробки та розглянуто аспекти охорони праці. Реалізація проєкту демонструє практичне застосування технологій комп’ютерного зору та машинного навчання у вирішенні актуальних завдань автоматизації аналізу інформації.

Результати проєкту можуть бути використані в різних галузях, включаючи системи безпеки, облік відвідувачів, персоніфікацію доступу та інші сфери, де необхідне точне та швидке розпізнавання облич.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Гудфеллоу Я., Бенджіо І., Курвілль А. Глибоке навчання. – Київ: Основи, 2016. – 652 с.
2. Литвин В. В., Пелещак Р. М., Висоцька В. А. Глибинне навчання : навч. посіб. / за ред. В. В. Литвина. – Львів : Львівська політехніка, 2021. – 264 с.
3. Шеліскі Р. Р. Комп'ютерний зір: алгоритми та застосування. – 2-ге вид. – Київ: Техніка, 2018. – 1232 с.
4. Gonzalez R. C., Woods R. E. Digital Image Processing. Global Edition. – 4th ed. – Pearson Education, 2017. – 1022 p.
5. Rosebrock A. Deep Learning for Computer Vision with Python. – 1st ed. – PyImageSearch, 2017. – 323 p.
6. Тернопільський національний технічний університет ім. І. Пулюя. Типові архітектури нейронних мереж. – [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/5740125/page:4/ – Дата звернення: 08.05.2025.
7. Chollet F. Keras 3 API documentation. – [Електронний ресурс]. – Режим доступу: https://keras.io/api/ – Дата звернення: 11.05.2025.
8. NVIDIA Corporation. TensorFlow. – [Електронний ресурс]. – Режим доступу: https://www.nvidia.com/en-us/glossary/tensorflow/ – Дата звернення: 09.05.2025.
9. OpenCV. Open Source Computer Vision Library. – [Електронний ресурс]. – Режим доступу: https://opencv.org/ – Дата звернення: 02.05.2025.
10. Redmon J. YOLO: Real-Time Object Detection. – [Електронний ресурс]. – Режим доступу: https://pjreddie.com/darknet/yolo/ – Дата звернення: 01.05.2025.
11. TensorFlow. TensorFlow. – [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/ – Дата звернення: 30.04.2025.
12. Batchelder N. Tkinter – Python interface to Tcl/Tk / Python Software Foundation. – [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/tkinter.html – Дата звернення: 05.05.2025.

ДОДАТОК А

Елементи коду класів FaceDetectonCNN, TrainingModel і FaceDetectionApp на Python

Клас FaceDetectionCNN:

```
class FaceDetectionCNN:
    def __init__(self, input_shape=(64, 64, 3)):
        self.input_shape = input_shape
        self.model = None
        # Завантаження класифікатора Хаара для первинної детекції облич
        self.face_cascade = cv2.CascadeClassifier(
            cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
        )
    def create_model(self, learning_rate=0.001):
        # Побудова згорткової нейромережі
        model = models.Sequential([
            layers.Conv2D(32, (3, 3), activation='relu', input_shape=self.input_shape),
            layers.MaxPooling2D(2, 2),
            layers.Conv2D(64, (3, 3), activation='relu'),
            layers.MaxPooling2D(2, 2),
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(1, activation='sigmoid')
        ])
        # Компіляція моделі
        model.compile(optimizer=optimizers.Adam(learning_rate),
                      loss='binary_crossentropy',
                      metrics=['accuracy', 'Precision', 'Recall'])
        self.model = model
        return model

    def train_model(self, X, y, validation_split=0.2, epochs=50, batch_size=32):
        # Розбиття даних на навчальну та валідаційну вибірки
        X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=validation_split)
        # Колбеки для зупинки та збереження найкращої моделі
        callbacks = [
            tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
            tf.keras.callbacks.ModelCheckpoint('best_face_model.h5',
save_best_only=True)
        ]
        # Навчання моделі
```

```
        history = self.model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=epochs,
            batch_size=batch_size,
            callbacks=callbacks
        )
        return history
    def detect_faces_in_image(self, image_path, confidence_threshold=0.7):
        # Завантаження зображення
        image = cv2.imread(image_path)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Пошук облич за допомогою Хаара
        haar_faces = self.face_cascade.detectMultiScale(gray, 1.1, 4)
        # Класифікація облич мережею (відбір за порогом), повернення координат знайдених облич
```

Клас FaceDetectionApp:

```
    class FaceDetectionApp:
    def __init__(self, root):
        self.root = root
        self.detector = FaceDetectionCNN()
        self.create_widgets()  # Ініціалізація інтерфейсу
    def analyze_image(self):
        # Перевірка наявності моделі
        if self.detector.model is None:
            messagebox.showerror("Помилка", "Модель не завантажена.")
            return
        # Запуск аналізу в окремому потоці
        threading.Thread(target=self.perform_analysis, daemon=True).start()
    def perform_analysis(self):
        # Детекція облич у зображенні
        image, faces = self.detector.detect_faces_in_image(self.image_path)
        self.root.after(0, self.update_results, faces)
    def update_results(self, faces):
        # Виведення результатів у таблицю і на зображення
        pass
    def save_result(self):
        # Збереження зображення з рамками навколо знайдених облич
        Pass
```

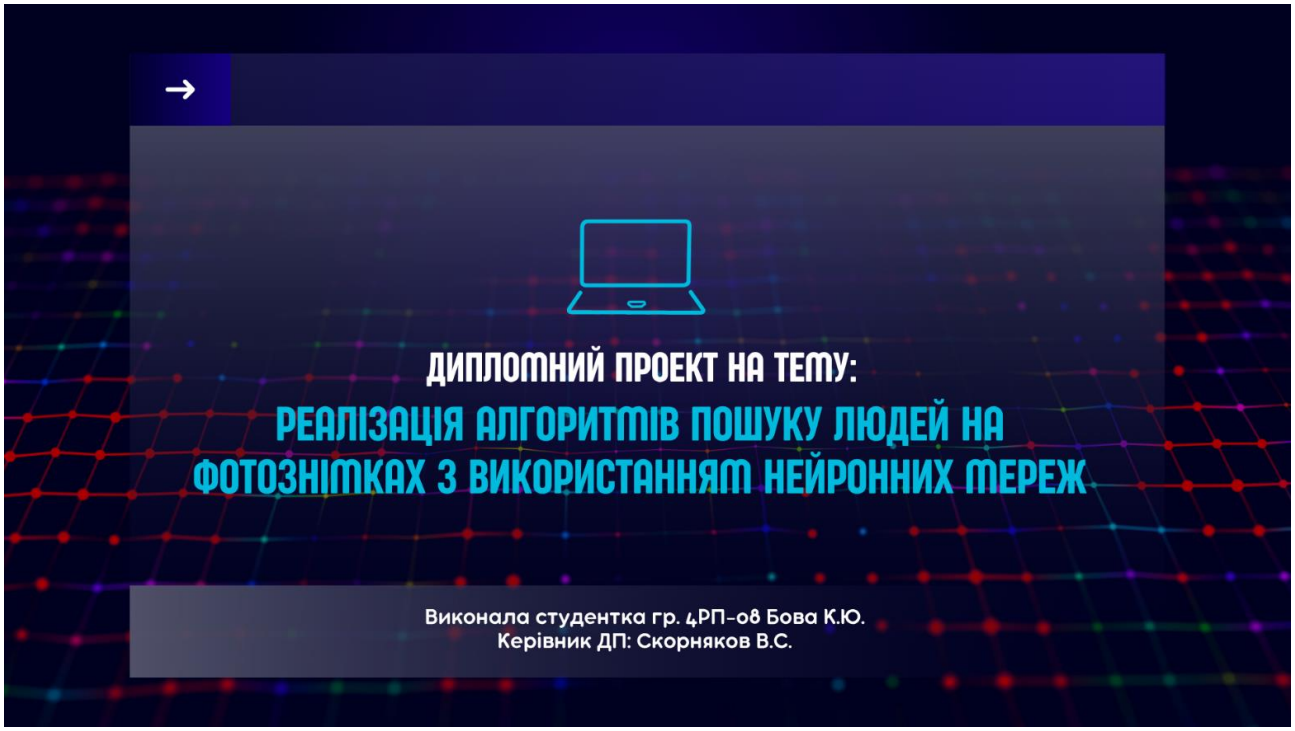
Клас TrainingDialog

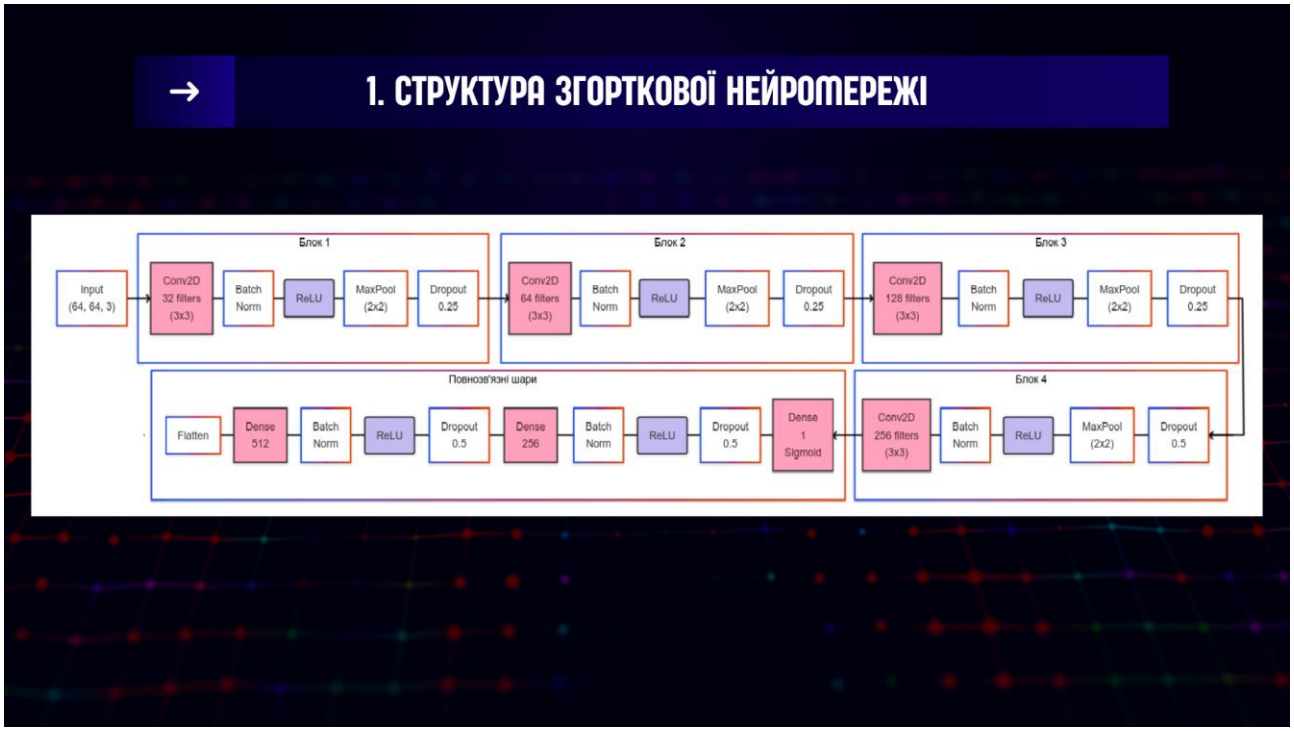
```
class TrainingDialog:
```

```
def __init__(self, parent, app):
    self.app = app
    self.dialog = tk.Toplevel(parent)
    self.create_widgets()  # Створення інтерфейсу
def start_training(self):
    # Перевірка наявності датасету
    if not os.path.exists('dataset/faces') or not os.path.exists('dataset/non_faces'):
        messagebox.showerror("Помилка", "Папки з даними не знайдено.")
        return
    # Запуск навчання в окремому потоці
    training_thread = threading.Thread(target=self.train_model, daemon=True)
    training_thread.start()
def train_model(self):
    # Завантаження даних, створення та навчання моделі
    X, y = self.app.detector.create_training_data('dataset/faces', 'dataset/non_faces')
    self.app.detector.create_model(learning_rate=0.001)
    self.app.detector.train_model(X, y, epochs=20, batch_size=32)
    # Завантаження найкращої збереженої моделі
    self.app.detector.load_model('best_face_model.h5')
```

ДОДАТОК Б

Слайди мультимедійної презентації





→ 2. ПОРІВНЯННЯ ІСНУЮЧИХ ПРОГРАМНИХ ЗАСОБІВ

1

OPENCV HAAR CASCADE

Точність (%): 65-75
Швидкість (мс/кадр): 15-25
Використання RAM (мб): 30-50
Підтримка GPU: ні

2

DLIB HOG

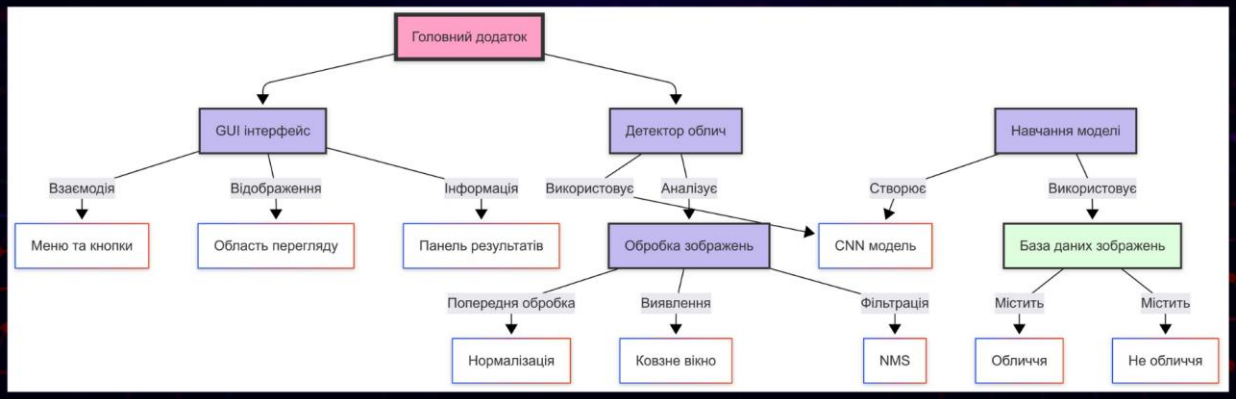
Точність (%): 80-85
Швидкість (мс/кадр): 40-60
Використання RAM (мб): 100-150
Підтримка GPU: ні

3

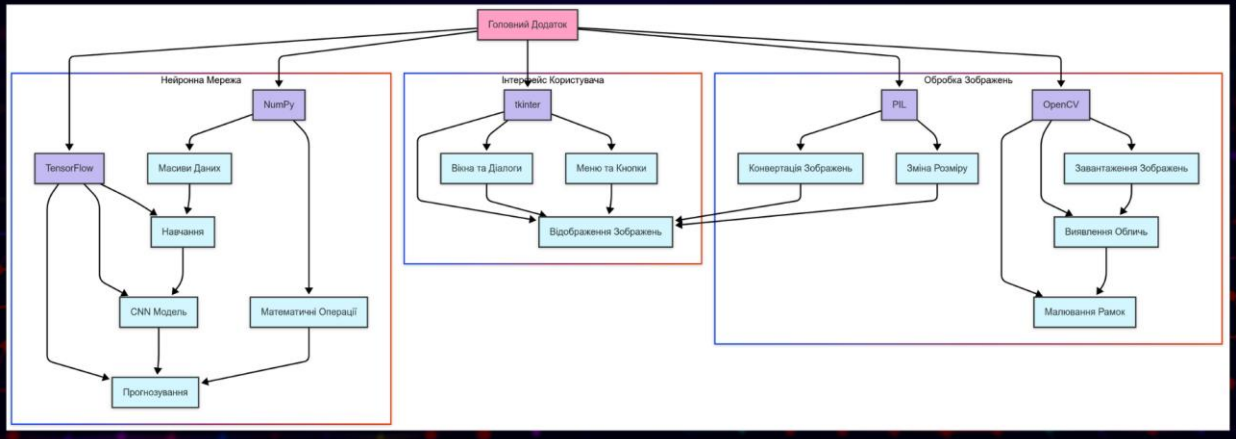
YOLOV5-FACE

Точність (%): 95-97
Швидкість (мс/кадр): 20-40
Використання RAM (мб): 250-400
Підтримка GPU: так

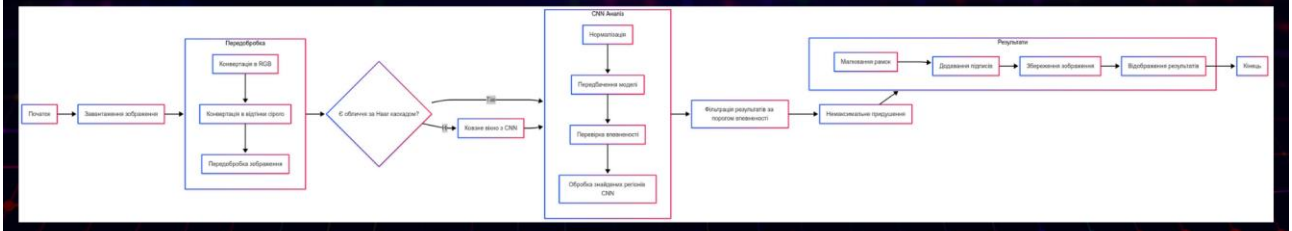
→ 3. СХЕМА ФРЕЙМВОРКУ



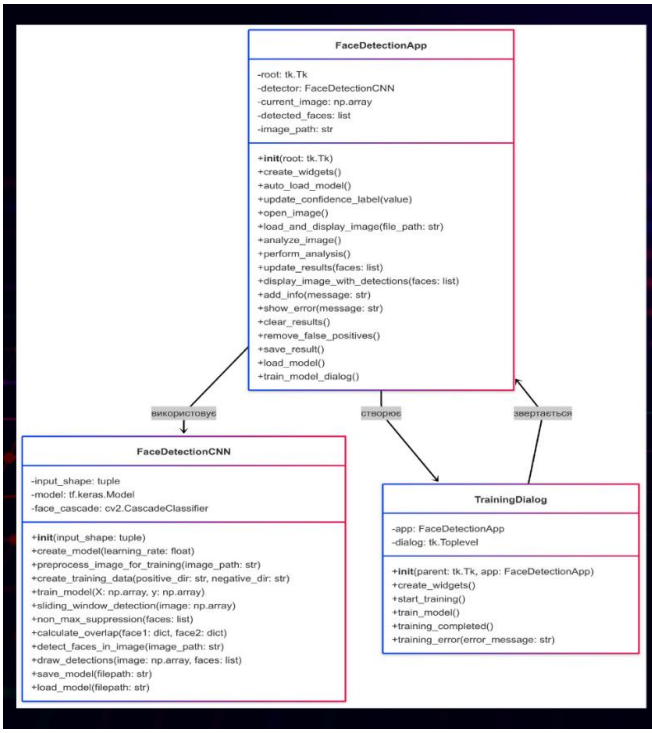
4. АРХІТЕКТУРА БІБЛІОТЕК



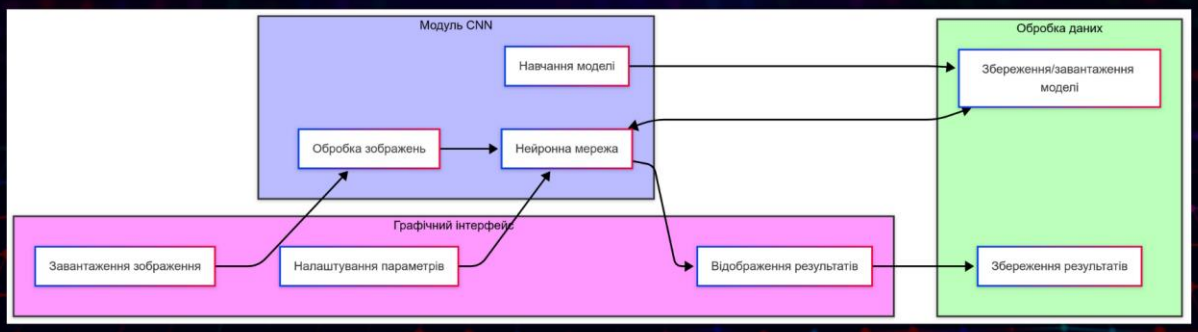
5. БЛОК СХЕМА АЛГОРИТМУ ПОШУКУ ЛЮДЕЙ



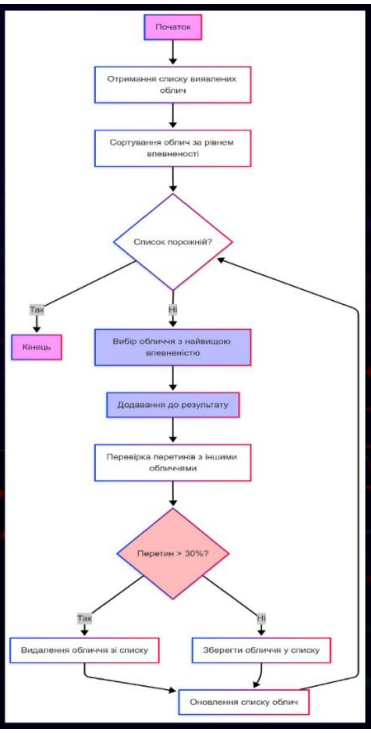
6. UML-ДІАГРАМА КЛАСІВ



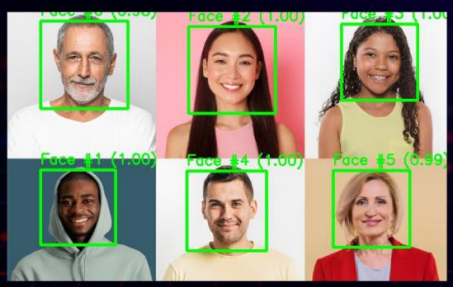
7. СХЕМА ЗВ'ЯЗКУ ОСНОВНИХ КОМПОНЕНТІВ



8. АЛГОРИТМ ВІДСІЮВАННЯ НЕВІРНИХ РЕЗУЛЬТАТІВ



9. ПРИКЛАДИ РОБОТИ ЗАСТОСУНКУ

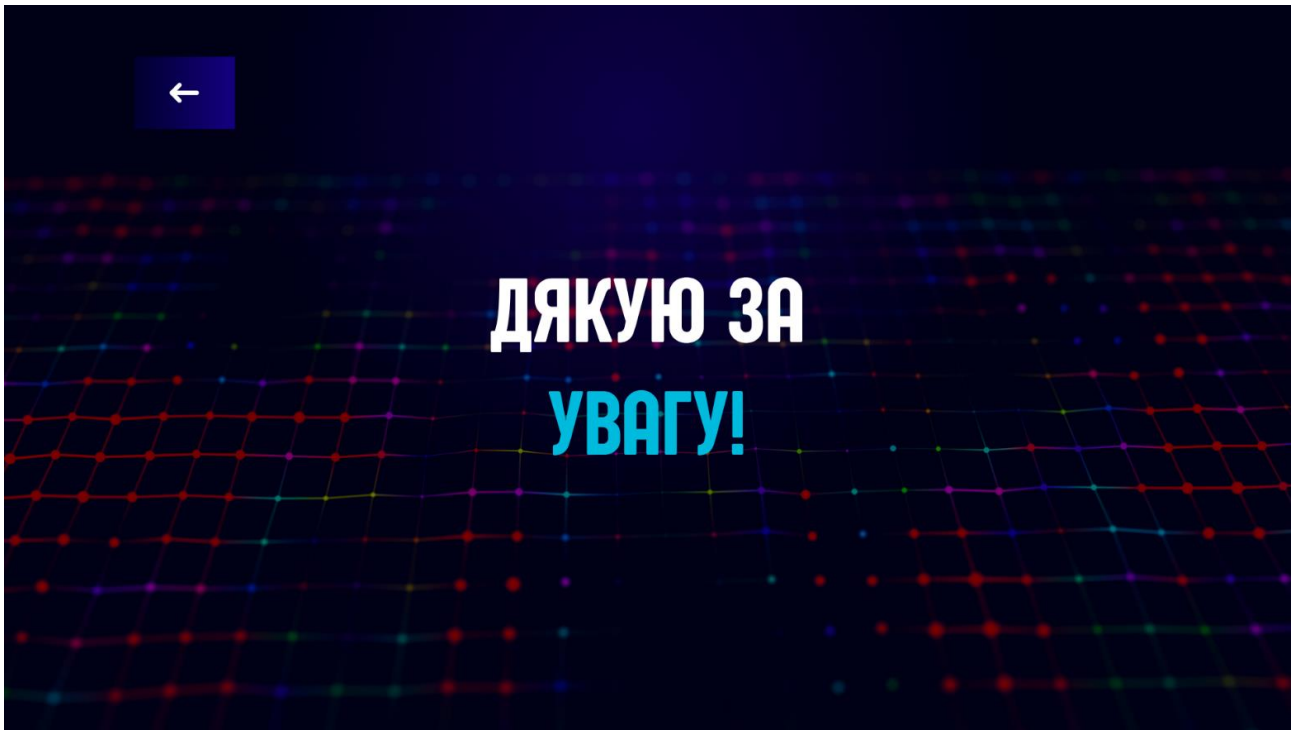


Знайдені обличчя

№	Впевненість	Координати	Розмір
1	0.999	(56, 265)	126x126
2	0.997	(305, 24)	146x146
3	0.997	(562, 20)	128x128
4	0.996	(302, 267)	134x134
5	0.993	(548, 266)	129x129
6	0.979	(56, 15)	147x147

№	Впевненість	Координати	Розмір
1	1.000	(85, 213)	129x129
2	0.999	(569, 141)	135x135
3	0.998	(278, 141)	120x120
4	0.998	(174, 208)	142x142
5	0.990	(392, 166)	135x135
6	0.988	(496, 302)	135x135
7	0.976	(159, 111)	103x103
8	0.964	(489, 120)	106x106
9	0.720	(301, 335)	151x151


10. АНАЛІЗ РОБОТИ ЗАСТОСУНКУ
СЕРЕДНЄ ЗНАЧЕННЯ ТЕСТУВАННЯ
З 20 облич було знайдено:
20
Впевненість (%)
97,35
Час (секунди)
1,5


ДЯКУЮ ЗА
УВАГУ!

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Бови Катерини Юріївни

(прізвище, ім'я та по батькові)

Спеціальність *121 " Інженерія програмного забезпечення "*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Скорняков В'ячеслав Сергійович*

(прізвище, ім'я та по батькові)

Тема дипломного проекту (роботи) *Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж*

Обсяг розрахунково-пояснювальної записки *78* сторінок

Обсяг графічної (презентаційної) частини *12* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню *Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення алгоритмів пошуку людей на фотознімках з використанням нейронних мереж*

б) характеристика виконання кожного розділу дипломного проекту (роботи) *Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У основному розділі наведено аналіз технологій побудови згорткових нейронних мереж, вибір засобів розробки застосунку, розробка структури застосунку, підключення необхідних програмних модулів, розробка алгоритму пошуку людей на фото, розробка UML-діаграми класів, реалізація застосунку, тестування застосунку на різних фото*

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) *Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату ідеї у роботі не виявлено*

г) перелік позитивних якостей дипломного проекту (роботи)

Комплексний підхід і широке охоплення тематики;
Добре структурована технічна частина;
Вибір сучасних інструментів і бібліотек;
Практична перевірка роботи системи

д) основні недоліки дипломного проекту (роботи)

Аналіз помилок подано досить коротко. В деяких підрозділах ПЗ згадується, що для оптимальної роботи мережі було обрано 20 епох. Проте, тестування вказує, що при 50 епохах модель починала виділяти хибні детекції, а при 10 епохах не дісталася достатньої узагальненості. Наявні деякі помилки у зображеннях блок-схем алгоритмів

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

« *20* » *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проєкт здобувача (здобувачки) освіти

відділення комп'ютерних систем

Бови Катерини Юріївни

(прізвище, ім`я та по батькові)

Спеціальність: *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма: *«Розробка програмного забезпечення»*

Тема дипломного проекту: *Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 78 сторінок. У пояснювальній записці виконано огляд та аналіз алгоритмів пошуку людей на фотознімках з використанням нейронних мереж, розроблено алгоритмічне та програмне забезпечення. Графічна частина складається з 12 слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки добра, але у блок-схемах алгоритмів і на діаграмах присутні деякі неточності*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувачка освіти Бова Катерина виконувала етапи дипломного проектування з порушенням термінів, але проявила ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувачка освіти виконувала самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувачка освіти Бова Катерина під час роботи над дипломним проектом вивчила достатньо багато літературних та інтернет-джерел за даною тематикою.*

Вважаю, що теоретична підготовка дипломниці достатня і вона готова до захисту проекту.

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувачка освіти Бова Катерина показала вміння організовано працювати над поставленим завданням, застосовувати знання у галузі програмування, розробляти, встановлювати та налаштовувати спеціалізоване програмне забезпечення, користуватись сучасними засобами розробки*

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові керівника дипломного проекту *Скорняков В'ячеслав Сергійович*

Місце роботи і посада керівника дипломного проекту *ВСП «ОТФК ОНТУ», викладач спецдисциплін комісії КТ та ПІ*

Підпис

«18» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Бова К.Ю.,
здобувач освіти гр. 4РП-08, та

Скорняков В.С.,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж» (автор роботи – Бова К.Ю., керівник роботи – Скорняков В.С.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Бова К.Ю. /

Керівник _____ / Скорняков В.С. /

«18» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4РП-08

Бови Катерини Юріївни

на тему *Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з несуттєвими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування Петрашова

(підпис) *18.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 18.06.2025 р. значення коефіцієнту подібності в роботі становить 19,97%, коефіцієнт цитування – 1,15%.*

(підпис) *18.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Бови К.Ю.* (П.І.Б.)

проведена « *18* » *червня* 20*25* р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

Звіт подібності

метадані

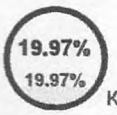
Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж

Автор
Бова Катерина Юріївна
Науковий керівник / Експерт
Скорняков В'ячеслав Сергійович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

КЦ

25	13306	105941
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	22
Інтервали	A→	0
Мікропробіли	¶	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	102

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	114 0.86 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	84 0.63 %
3	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	68 0.51 %
4	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	67 0.50 %
5	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	66 0.50 %

6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	49 0.37 %
7	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	48 0.36 %
8	https://ela.kpi.ua/bitstreams/60a6ddc3-5a06-452a-80ce-462dfe293856/download	47 0.35 %
9	https://ena.lpnu.ua:8443/server/api/core/bitstreams/ab12208a-0285-466a-a823-11ea6b2beeb7/content	45 0.34 %
10	Sokolov_diplom_bak_Kuklin_fkn_2024 6/4/2024 V. N. Karazin Kharkiv National University (KKNU) (ННІ комп'ютерних наук та штучного інтелекту - кафедра математичного моделювання та аналізу даних)	45 0.34 %

з домашньої бази даних (0.14 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	19 (2) 0.14 %

з програми обміну базами даних (1.01 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**Sokolov_diplom_bak_Kuklin_fkn_2024** **6/4/2024** **V. N. Karazin Kharkiv National University (KKNU) (ННІ комп'ютерних наук та штучного інтелекту - кафедра математичного моделювання та аналізу даних)**	**45 (1) 0.34 %**
2	122_Nahorniuk_Roman_Ruslanovych_it2024 12/6/2024 National University of Food Technologies (Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки)	35 (3) 0.26 %
3	bitstream_f8cdaf0e-6b12-4508-968f-b20a722fb2c2 12/7/2024 National Technical University "Kharkiv Polytechnic Institute" students papers (National Technical University "Kharkiv Polytechnic Institute" students papers)	12 (1) 0.09 %
4	Development of gesture recognition systems using Teachable Machine for user interfaces 11/27/2024 Kharkiv National University of Economics named after S.Kuznets (KNUE) (KNUE)	12 (2) 0.09 %
5	Побудова гібридних згорткових нейронних мереж 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	10 (1) 0.08 %
6	Методи глибокого навчання в згорткових нейронних мережах для класифікації та сегментації зображень 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	8 (1) 0.06 %
7	Системи розпізнавання алфавітно-цифрової інформації методами інтелектуального аналізу даних 10/28/2024 University of Customs and Finance course papers (University of Customs and Finance course papers)	7 (1) 0.05 %

8	Диплом_Гончарук_Іван [redacted] 2/17/2025 Interregional Academy of Personnel Management (Інститут комп'ютерно-інформаційних технологій та дизайну)	5 (1) 0.04 %

з Інтернету (18.82 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**620 (23) 4.66 %**
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/44c16132-5f53-48e2-b6c0-61e9a2f0fd75/content	576 (48) 4.33 %
3	**https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download**	**157 (6) 1.18 %**
4	**https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download**	**148 (3) 1.11 %**
5	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	115 (10) 0.86 %
6	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content**	**104 (7) 0.78 %**
7	**https://ela.kpi.ua/bitstreams/60a6ddc3-5a06-452a-80ce-462dfe293856/download**	**96 (5) 0.72 %**
8	**https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download**	**83 (2) 0.62 %**
9	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	65 (5) 0.49 %
10	**https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download**	**62 (3) 0.47 %**
11	https://elartu.tntu.edu.ua/bitstream/lib/44844/1/Mag_2024_SNnm_61_%D0%93%D0%B0%D0%B9%D0%B4%D1%83%D0%BA.pdf	60 (6) 0.45 %
12	https://www.ela.kpi.ua/bitstream/123456789/60511/1/Zyblyi_bakalavr.pdf	58 (6) 0.44 %
13	https://github.com/Nasserahm/UML-3	53 (4) 0.40 %
14	**https://ena.lpnu.ua:8443/server/api/core/bitstreams/ab12208a-0285-466a-a823-11ea6b2beeb7/content**	**52 (2) 0.39 %**
15	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	38 (5) 0.29 %
16	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	31 (3) 0.23 %
17	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	23 (1) 0.17 %
18	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	23 (2) 0.17 %
19	https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/	22 (3) 0.17 %
20	https://openarchive.nure.ua/bitstreams/d6adea9d-b281-4e0d-a432-aec52df9427c/download	19 (1) 0.14 %
21	https://cyberleninka.ru/article/n/podavlenie-belogo-shuma-na-izobrazheniyah-na-osnove-vinerovskoy-filtratsii-v-oblasti-diskretnogo-veyvlet-preobrazovaniya-s	17 (1) 0.13 %
22	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	15 (2) 0.11 %
23	https://dut.edu.ua/repozitorii/ipz/2020/%D0%93%D0%BE%D1%80%D0%B4%D1%96%D1%94%D0%BD%D0%BA%D0%BE%20%D0%9A%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D0%BD%D0%B0%20%D0%9E%D0%BB%D0%B5%D0%BA%D1%81%D0%B0%D0%BD%D0%B4%D1%80%D1%96%D0%B2%D0%BD%D0%B0.pdf	14 (1) 0.11 %
24	https://digikogu.taltech.ee/en/Download/14503575-8027-40fe-a8f6-c4c1cf776672	12 (1) 0.09 %

25	https://card-file.ontu.edu.ua/bitstreams/62baa43e-b968-4993-bb54-8cf8761a89b2/download	11 (1) 0.08 %
26	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	10 (1) 0.08 %
27	https://revolution.allbest.ru/programming/01236208_0.html	8 (1) 0.06 %
28	https://essuir.sumdu.edu.ua/bitstream/123456789/78111/1/bachelor_thesis_Mirosiedi.pdf	7 (1) 0.05 %
29	https://card-file.ontu.edu.ua/server/api/core/bitstreams/4bb7255e-46d4-4349-9726-9698476da02d/content	5 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»_
Освітньо-професійна програма: «Розробка програмного забезпечення» Група: 4РП- 08

Дипломний проект здобувача освіти денної форми навчання РП. 08.03.000.ДП

БОВИ
КАТЕРИНИ ЮРІЇВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: «Розробка програмного забезпечення»
Група: 4РП-08

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проекту на тему:

Проектний матеріал складається з пояснювальної записки на _____ сторінках та графічного (презентаційного) матеріалу на _____ аркушах (слайдах)

Дипломник _____ **(Бова К.Ю.)**
Керівник _____ **(Скорняков В.С.)**

Консультанти:
з економічного розділу _____ **(Канський М. Ю.)**
з розділу охорони праці та техніки безпеки _____ **(Чорновол Н.І.) з нормоконтролю** _____ **(Петрашова В.І.) старший консультант** _____ **(Кривченко Ю. В.)**

До захисту допущений
Голова циклової комісії _____ **(Кривченко Ю.В.)**
Завідувач відділення _____ **(Краснокутська К.Г.)**

Захист «____» _____ 2025 р. Протокол ЕК No_____ Оцінка ЕК _____ Секретар ЕК _____

Реалізація алгоритмів пошуку людей на фотознімках з використанням нейронних мереж
12
78

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ