**МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ**
**ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ УПРАВЛІННЯ, ПСИХОЛОГІЇ ТА БЕЗПЕКИ**

**Кафедра інформаційних технологій**

# РОЗРОБКА ІНТЕГРОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ЕФЕКТИВНОГО УПРАВЛІННЯ ПРАВООХОРОННОЮ ДІЯЛЬНІСТЮ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ

**Кваліфікаційна робота**
здобувача вищої освіти
4 курсу денної форми навчання
**Олександри ЯВОРСЬКОЇ**

**Науковий керівник:**
доцент, кандидат технічних наук
**Олег ЗАЧЕК**

**Рецензент:**

_____
вчене звання, науковий ступінь

_____
(Ім’я ПРІЗВИЩЕ рецензента)

***Кваліфікаційна робота допущена до захисту***
«___» _____ 2025 р., протокол № ____

Завідувач кафедри інформаційних технологій
_____Ігор ФЕДЧАК_
(підпис)

Львів
2025

# СПИСОК УМОВНИХ СКОРОЧЕНЬ

| | |
|---|---|
| ШІ | штучний інтелект |
| CNN | згорткова нейронна мережа |
| VGG16 | 16-шарова CNN візуальної геометричної групи |
| CV | комп’ютерний зір |
| E3FRM | ансамблевий механізм розпізнавання облич |
| F1 | середнє гармонійне між точністю (precision) та повнотою (recall), що використовується для оцінки якості класифікаційної моделі |
| AlexNet | глибока нейронна мережа для класифікації зображень |
| RRNN | рекурентна регресійна нейронна мережа |
| ReL | Rectified Linear Unit (випрямлена лінійна одиниця) |
| CPU | центральний процесор |
| GPU | графічний процесор |
| MPI | інтерфейс передавання повідомлень |
| CUDA | уніфікована обчислювальна архітектура CUDA |
| ПК | персональний комп’ютер |
| ReLU | випрямлена лінійна функція |
| RGB | червоний, зелений, синій |
| POSIX | портативний інтерфейс операційних систем |
| GPGPU | загальнопризначені обчислення на графічних процесорах |
| MPI | інтерфейс передавання повідомлень |
| SPMD | єдина програма, множина даних |

# АНОТАЦІЯ

**Яворська О., Зачек О.** (керівник). Розробка інтегрованої інформаційної системи для ефективного управління правоохоронною діяльністю за допомогою технологій штучного інтелекту. Бакалаврська кваліфікаційна робота. – Львівський державний університет внутрішніх справ, Львів, 2025.

У цій роботі запропоновано новий підхід до розробки інтегрованої інформаційної системи для підвищення ефективності правоохоронної діяльності шляхом використання методів штучного інтелекту та технологій паралельних обчислень. Основна увага зосереджена на забезпеченні обробки даних у реальному часі без втрати точності.

Розглянуто та оптимізовано методи паралельної обробки, зокрема OpenMP, multiprocessing, MPI (mpi4py) та CUDA, що дозволило суттєво прискорити обчислювальні процеси. Для перевірки ефективності підходу проведено чисельний аналіз нейронної мережі VGG16 у задачі автоматизованого розпізнавання облич із використанням загальнодоступного набору даних, наближеного до реальних умов правоохоронної діяльності.

Результати дослідження підтверджують, що інтеграція технологій паралельних обчислень і штучного інтелекту дозволяє значно підвищити продуктивність та ефективність автоматизованих систем, орієнтованих на розв’язання завдань правоохоронних органів.

**Ключові слова:** інтегрована інформаційна система, штучний інтелект, паралельні обчислення, OpenMP, multiprocessing, MPI, CUDA, автоматизоване розпізнавання облич, правоохоронна діяльність.

# ABSTRACT

**Yavorska S.,** Zachek O. (supervisor). Development of an integrated information system for effective law enforcement management using artificial intelligence technologies. Bachelor's thesis. – Lviv State University of Internal Affairs, Lviv, 2025.

This paper proposes a novel approach to developing an integrated information system to enhance law enforcement efficiency by leveraging artificial intelligence methods and parallel computing technologies. The primary focus is on ensuring real-time data processing without compromising accuracy.

Various parallel processing techniques, including OpenMP, multiprocessing, MPI (mpi4py), and CUDA, have been examined and optimized to accelerate computational processes. To evaluate the effectiveness of the approach, a numerical analysis of the VGG16 neural network was conducted for automated face recognition using a publicly available dataset that closely resembles real-world law enforcement conditions.

The results confirm that integrating parallel computing technologies with artificial intelligence significantly improves the performance and efficiency of automated systems designed for law enforcement applications.

**Keywords:** integrated information system, artificial intelligence, parallel computing, OpenMP, multiprocessing, MPI, CUDA, automated facial recognition, law enforcement.

# ЗМІСТ

# ВСТУП

Сучасний світ неможливо уявити без ефективних інформаційних систем, що забезпечують комплексне управління правоохоронною діяльністю та гарантують безпеку громадян [1]. Впровадження технологій ШІ у такі системи дозволяє значно підвищити рівень автоматизації, ефективність обробки інформації та швидкість прийняття рішень. Одним із ключових аспектів інтегрованих інформаційних систем у сфері правоохоронної діяльності є контроль доступу до об’єктів та інформаційних ресурсів, що може бути реалізований на основі біометричних методів, зокрема розпізнавання облич.

Актуальність дослідження обумовлена необхідністю розробки інтегрованих інформаційних систем, які не лише забезпечують контроль доступу, але й виконують ширший спектр завдань, включаючи: ідентифікацію осіб у режимі реального часу; моніторинг відеопотоків з камер спостереження; виявлення підозрілих осіб та запобігання загрозам.

Новизна дослідження полягає у впровадженні методів ШІ, зокрема глибокого навчання, для створення ефективної системи розпізнавання облич, що інтегрується у комплексну інформаційну систему для правоохоронних органів. Використання сучасних архітектур нейронних мереж, таких як VGG16, дозволяє досягти високої точності розпізнавання та забезпечити надійність роботи системи в умовах реального часу.

**Аналіз останніх досліджень і публікацій.** Дослідженню проблем ефективної реалізації паралельних обчислень для прискорення навчання нейронних мереж присвячено чимало публікацій українських та закордонних вчених. Прикладами можуть слугувати роботи: Lixiang Li, Xiaohui Mu, Siying Li, Peng Haipeng [2], Muhammad Zamir, Nouman Ali, Amad Naseem, Areeb Ahmed Frasteen, Bushra Zafar, Mu As, Mahmoud Othman, El-Awady Attia [3], Feng Liu,

Minchul Kim, Anil Jain, Xiaoming Liu [4], Fan Liu, Delong Chen, Fei Wang, Zewen Li, Feng Xu [5], A Peña, Aythami Morales, I Serna, Julian Fierrez, Àgata Lapedriza [6], Панасюк [7] та інших. Їх праці заклали основу для подальших досліджень у сфері прискореного навчання нейронних мереж із використанням сучасних технологій паралельного обчислення.

Разом з цим слід зазначити, що алгоритми підготовки даних для навчання нейронних мереж постійно вдосконалюються, оскільки якість вхідних даних суттєво впливає на результати класифікації. У зв'язку з цим підхід, що враховує такі фактори, як рівень шуму, умови освітлення, положення об'єкта та наявність масок, і надалі залишається актуальним.

У зв'язку зі сказаним можемо сформулювати мету та завдання дослідження.

**Метою дослідження** є проектування та реалізація високоефективної інтегрованої інформаційної системи для правоохоронних органів, що використовує передові алгоритми ШІ для розпізнавання облич. **Завдання дослідження** включають: Аналіз існуючих методів розпізнавання облич та технологій штучного інтелекту; Визначення вимог до інформаційної системи, включаючи критерії точності, швидкості роботи, безпеки та адаптивності до змінних умов; Проектування архітектури інтегрованої системи з урахуванням модулів біометричної ідентифікації, відеоспостереження та аналізу поведінкових характеристик осіб; Реалізація системи, тестування її ефективності у реальних умовах та порівняння різних моделей ШІ; Аналіз отриманих результатів та розробка рекомендацій щодо вдосконалення системи.

**Об'єкт дослідження** – Інтегровані інформаційні системи для правоохоронної діяльності.

**Предмет дослідження** – Методи застосування технологій штучного інтелекту для підвищення ефективності управління правоохоронною

діяльністю, зокрема в контексті розпізнавання облич, контролю доступу та аналізу відеопотоків.

**Методи досліджень.** Для досягнення поставлених цілей будуть використовуватися: аналітичні методи – аналіз наукової літератури, статей, існуючих методів розпізнавання облич; експериментальні методи – перевірка та оцінка імплементованих та оптимізованих методів на реальних зображеннях; порівняльні методи – аналіз ефективності різних алгоритмів розпізнавання облич. Оптимізація за допомогою паралельних обчислень Оскільки аналіз відеопотоків передбачає великі обсяги даних, система повинна підтримувати паралельні обчислення. Використання CPU та GPU для прискорення алгоритмів дозволяє: збільшити швидкість обробки інформації; зменшити навантаження на обчислювальні ресурси; оптимізувати використання державного бюджету. Беручи до уваги ці переваги, можна зробити висновок, що застосування паралельних обчислень у системах розпізнавання облич сприятиме зменшенню витрат на обчислювальні ресурси, збільшенню швидкості обробки інформації та підвищенню ефективності правоохоронних органів.

Розроблена система контролю доступу, інтегрована у загальну інформаційну систему правоохоронних органів, дозволить автоматизувати процеси ідентифікації осіб, запобігати несанкціонованому доступу та значно підвищити рівень безпеки об'єктів і громадських місць. Крім того, впровадження методів оптимізації дозволить значно покращити швидкодію алгоритмів та зменшити витрати на їх використання.

**Структура роботи.** Кваліфікаційна робота складається із вступу, чотирьох розділів, висновків, списку використаних джерел, додатку. Обсяг основного тексту роботи складає 58 сторінок, 23 рисунки, 16 таблиць, 1 додаток і 18 бібліографічних джерел. Загальний обсяг роботи – 67 сторінок.

# РОЗДІЛ 1
# СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

## 1.1. Аналіз літературних джерел

Станом на початок 2025 року проблема ефективного управління правоохоронною діяльністю за допомогою технологій ШІ набуває все більшої актуальності. Розробка інтегрованих інформаційних систем дозволяє підвищити ефективність оперативно-розшукових заходів, аналізу великих обсягів даних та прогнозування криміногенних ситуацій. У науковій спільноті обговорюються такі ключові питання:

1. Автоматизація процесів прийняття рішень. Використання алгоритмів машинного навчання та глибокого навчання для підвищення ефективності аналізу оперативної інформації.
2. Оптимізація розподілу ресурсів. Використання методів штучного інтелекту для прогнозування рівня злочинності та оптимального розподілу поліцейських підрозділів.
3. Обробка великих обсягів даних. Інтеграція різних джерел інформації, таких як відеоспостереження, аудіоаналіз та текстові дані, для комплексного аналізу ситуацій.
4. Кібербезпека та захист даних. Забезпечення безпечного зберігання та обробки чутливої інформації, що використовується в правоохоронній діяльності.

CV – це напрям штучного інтелекту, що дозволяє автоматизованим системам виконувати виявлення, відстеження та класифікацію об'єктів. Використання методів машинного навчання в останні роки значно розширило можливості комп'ютерного зору, зокрема у сфері правоохоронної діяльності.

Для проведення систематичного аналізу літератури було використано стандартизовану методологію PRISMA [8], що дозволяє забезпечити

об'єктивність та наукову обґрунтованість дослідження. Відбір наукових публікацій здійснювався за допомогою наукометричної бази Scopus, що забезпечує доступ до релевантних та авторитетних джерел.

У роботі [9] описуються методи використання CV для ідентифікації підозрюваних у натовпі за допомогою технологій розпізнавання облич. автори пропонують механізм E3FRM, який ґрунтується на трьохпрохідному рішенні. Модель, описана в статті, демонструє покращення метрик F1, точності та повернення в порівнянні з існуючими методами. Це підтверджує ефективність пропонованого підходу. У статті не обговорюється питання, пов'язані з розпізнаванням обличчя в реальних умовах, таких як зміна освітлення, зміна виразу обличчя тощо. А це є важливим аспектом для даної роботи.

Дослідження [10] розглядає розробку системи розпізнавання обличчя для автоматичного блокування дверей, що є зручним, ефективним і досить точним рішенням для ідентифікації власників будинку. Для розпізнавання обличчя застосовувався метод CNN на основі архітектури AlexNet, який інтегровано в систему блокування дверей. Для навчання системи було зібрано 1048 зображень обличчя власника, що забезпечило точність моделі на рівні 97,5%. Це є високим результатом у порівнянні з іншими подібними дослідженнями. Проте слід зазначити, що автори не надали детальної інформації щодо процесу збору зображень, що ускладнює розуміння досягнутої точності. Крім того, дослідження обмежене специфічним застосуванням для блокування дверей, що знижує його релевантність для розробки загальних систем розпізнавання обличчя. Відсутність порівняння з іншими методами розпізнавання також ускладнює оцінку ефективності обраного підходу.

Стаття [11] зосереджена на порівнянні різних методологій розпізнавання обличчя за допомогою CNN з використанням різних баз даних. Автори акцентують увагу на проблемі one-shot learning, яка є важливою на сьогоднішній день. Мета статті — стати корисним ресурсом для дослідників у

галузі розпізнавання облич за одним зображенням. Це дослідження може бути корисним для аналізу різних баз даних і методів для розпізнавання обличчя. Однак для застосування нейронної мережі VGG16, яку планується використати в даній роботі, стаття може не надати необхідної інформації, оскільки вона фокусується на CNN загалом.

У статті [12] пропонується новий підхід до розпізнавання обличчя за допомогою RRNN, яка об’єднує завдання розпізнавання облич з різних ракурсів на статичних зображеннях та відео. Автори використовують потенційні залежності між зображеннями для регуляризації моделі навчання. Цей підхід може бути корисним для покращення точності моделей, однак RRNN є складною архітектурою, яка може вимагати значних обчислювальних ресурсів і часу на навчання, що робить її менш практичною для цього дослідження. Відсутність деталей про попередню обробку даних і тренування моделі також є недоліком.

У контексті правоохоронних органів авторами роботи [13] запропоновано систему для пошуку обличчя на зображеннях з використанням алгоритмів розпізнавання облич, що є важливим кроком для вдосконалення систем безпеки та моніторингу. Огляд попередніх досліджень показав актуальність та необхідність розвитку таких технологій, однак були виявлені й обмеження, що впливають на їх ефективність у реальних умовах. Система пошуку обличчя може бути використана не лише для ідентифікації осіб, а й для додаткових завдань, таких як підрахунок людей у певних зонах або розпізнавання осіб, що дивляться на цифрові дисплеї для запобігання небажаних подій. У цьому випадку пошук обличчя є основою для подальших застосувань, таких як відеоспостереження, відстеження осіб у реальному часі, або розпізнавання осіб у зонах з високим рівнем безпеки. Для підвищення ефективності запропоновано застосувати глибинні CNN, що дають високі результати в задачах комп'ютерного зору. В рамках розробки системи використано TensorFlow для навчання нейронних мереж, набір даних LFW для

тренування моделі, а також бібліотеку LabelImg для попередньої розмітки даних.

На основі аналізу літературних джерел було визначено, що інтеграція різних методів штучного інтелекту в інформаційні системи для правоохоронної діяльності є перспективним напрямом досліджень завдяки таким характеристикам:

- Покращена точність аналізу, адже використання сучасних алгоритмів дозволяє більш ефективно ідентифікувати загрози та підозрілу поведінку.
- Гнучкість адаптації, тобто інформаційні системи можуть бути налаштовані під конкретні задачі правоохоронних органів.
- Можливість інтеграції різних технологій. Останнє передбачає поєднання відеоспостереження, аналізу аудіо та текстових даних забезпечує комплексний підхід до кримінального аналізу.
- Баланс між швидкодією та точністю. Оптимізовані алгоритми дозволяють працювати в режимі реального часу без значного навантаження на обчислювальні ресурси.

Отже, аналіз літератури підтверджує актуальність використання штучного інтелекту для управління правоохоронною діяльністю, а обґрунтований вибір інтегрованої інформаційної системи дозволяє значно підвищити ефективність її роботи.

## 1.2. Аналіз існуючих аналогів

Аналізуючи існуючі аналоги, варто відзначити, що в Україні в найближчому майбутньому планується впровадження системи, яка передбачає масове встановлення відеокамер для ідентифікації осіб (див. рис. 1.1). Законопроєкт (№ 11031), що стосується створення єдиного відеомоніторингового центру для підтримки публічної безпеки, був представлений у лютому 2024 року і обговорений міністром внутрішніх справ Ігорем Клименком. Основною метою системи є забезпечення правопорядку та оперативне реагування на правопорушення. Система передбачає інтеграцію з

Єдиним державним демографічним реєстром, біометричною верифікацією та іншими реєстрами для ідентифікації осіб і визначення їхнього місцеперебування.


*Рис. 1.1. Система тотального розпізнавання облич*

Впровадження такої системи може істотно змінити соціальну ситуацію в Україні, зокрема спростити процеси пошуку злочинців. Однак, цей проект також викликає занепокоєння через можливість тотального контролю за пересуванням громадян, а також через ризики витоку даних, оскільки цифрові технології завжди мають уразливі місця. Зокрема, вказано на можливість втручання в особисті права громадян, адже система може передбачати не лише ідентифікацію, а й обмін даними між різними органами, зокрема поліцією та військовими структурами. Враховуючи ці питання, необхідно знайти баланс між підвищенням рівня безпеки та захистом прав громадян.

Face Recognition Library — це потужна опенсорс-бібліотека, заснована на C++ бібліотеці dlib, яка дозволяє реалізувати різноманітні задачі з розпізнавання облич. Вона вирізняється високою точністю, зручністю у використанні та можливістю розгортання на сервері без необхідності в дорогому GPU, а також має досить прості вимоги до апаратного забезпечення.

Основні можливості бібліотеки: Пошук облич на фото (рис. 1.2). Бібліотека дозволяє знаходити обличчя на зображеннях, підтримуючи кут повороту голови більше 30 градусів, що є важливою перевагою порівняно з іншими інструментами, такими як OpenCV.

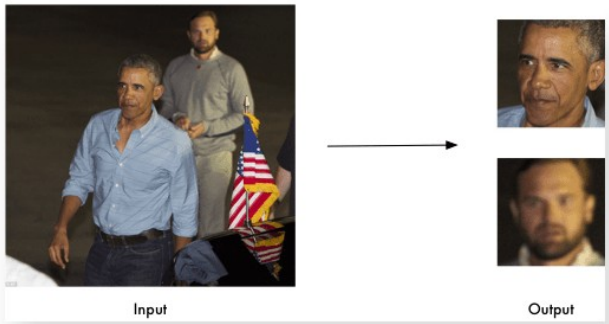


*Рис. 1.2. Пошук облич на фото на основі Face Recognition Library*

Пошук рис обличчя на фотографіях представлено на рис. 1.3. Розпізнавання ключових рис обличчя, таких як очі, ніс, рот і підборіддя, що є важливими для подальшої ідентифікації.

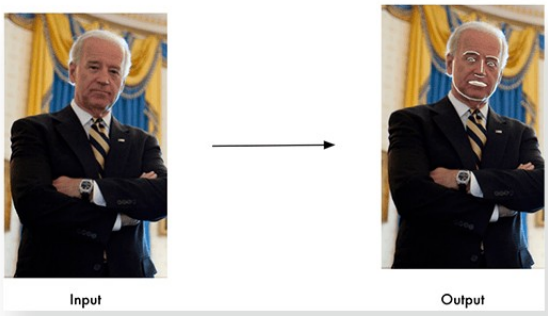


*Рис. 1.3. Пошук рис обличчя на фотографіях на основі Face Recognition Library*

На рис. 1.4 можемо бачити як працюють карти координат специфічних лицьових точок. Бібліотека надає детальні карти з 68 точок на обличчі, що дозволяє з великою точністю порівнювати та ідентифікувати обличчя за допомогою координат цих точок.

*Рис. 1.4. Карти координат специфічних лицьових точок*

Ідентифікація осіб на фотографіях показана на рис. 1.5. Завдяки збереженню бази "своїх" осіб (наприклад, співробітників або клієнтів), бібліотека здатна ідентифікувати осіб на фото та визначати, хто зображений на кожному знімку, порівнюючи координати лицьових точок.

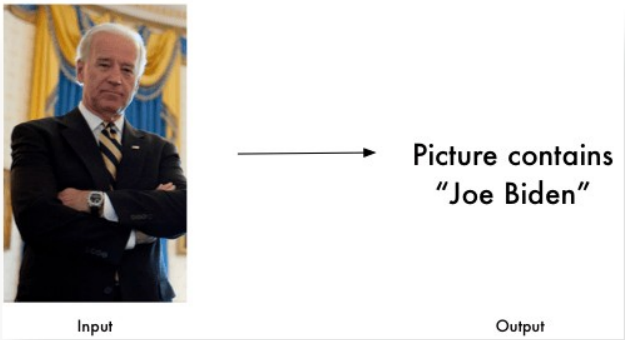


*Рис. 1.5. Ідентифікація осіб*

Ще одною перевагою даної бібліотеки є застосування в real-time рішеннях, що представлено на рис. 1.6. Face Recognition Library підтримує інтеграцію з потоковим відео і може бути використана в реальному часі. Вона легко інтегрується з іншими бібліотеками Python для більш складних рішень.

*Рис. 1.6. Застосування в real-time рішеннях*

Аналіз існуючих систем управління правоохоронною діяльністю виявив низку недоліків, зокрема недостатню інтеграцію з різними базами даних, що обмежує ефективність прийняття рішень у реальному часі. Багато з цих систем мають обмежену здатність до автоматичного аналізу великих обсягів даних, що ускладнює своєчасне виявлення правопорушень та управління ресурсами. Крім того, відсутність інтелектуальних алгоритмів для прогнозування та оптимізації дій правоохоронців на основі аналізу історичних та поточних даних знижує ефективність оперативного реагування. Технології штучного інтелекту, які можуть значно покращити автоматизацію процесів, зокрема за допомогою систем розпізнавання осіб та аналізу поведінки, все ще недостатньо інтегровані в існуючі системи, що обмежує їх потенціал у забезпеченні безпеки.

## 1.3. Постановка задачі

Нехай маємо набір зображень $X = \{x_1, x_2, x_3, \ldots, x_n\}$, де $n$ – загальна кількість зображень. Кожне $x_i$ у наборі представляє обличчя людини. Також у нас є набір $D = \{d_1, d_2, d_3, \ldots, d_m\}$, де $m$ – кількість людей у наборі, про яких є доступні дані. Кожне $d_i$ містить інформацію про особу, включно з її ім’ям та фотографією.

$$f: X \rightarrow D \tag{1}$$

$f$ – функція розпізнавання облич, яка приймає на вхід зображення обличчя і повертає вектор ймовірностей належності до різних класів (наприклад, особи, яку слід ідентифікувати). Вона є основним компонентом процесу розпізнавання облич. $S$ – множина підзадач розпізнавання облич, яка представляє собою різні аспекти та завдання, пов'язані з розпізнаванням облич, такі як класифікація облич, виявлення ключових точок на обличчі (очі, ніс, рот). Кожна підзадача вимагає власної конфігурації функції $f$. $\theta$ – параметри мережі VGG16.

Серед параметрів мережі VGG16 $\theta$ є наступні:

- Розмір вхідного зображення.
- Кількість класів, яка в нашому випадку буде задана розміром множини $D$, оскільки кожна людина описується як окремий клас.
- Гіперпараметри навчання, які включають швидкість навчання $learning\_rate$, кількість епох $epochs$, розмір пакета $batch\_size$ та функцію втрат $\boldsymbol{L(f(X), D)}$.
- Архітектурні параметри, які визначають архітектуру самої мережі VGG16, такі як кількість блоків $blocks$, кількість шарів у кожному блоці $layers\_per\_block$, розмір фільтрів $kernel\_size$ та кількість фільтрів на кожному шарі $lkernels\_per\_layer$.
- Функція активації, у нашому випадку $ReLU$.
- Регуляризація, а саме $L2\_regularization$, яка додає штраф до загальної втрати моделі на основі квадратів ваг параметрів. Це змушує модель уникати надмірного "вагового" навчання і забезпечує більшу стабільність у загальному вигляді.
- Dropout – це техніка, яка випадково вимикає деякі нейрони в мережі під час навчання. Це допомагає уникнути перенавчання та робить мережу більш стійкою до різних варіантів даних, оскільки нейрони повинні

навчатись без залежності від конкретних сусідніх нейронів. Ми будемо вимикати кожен п’ятий нейрон.

Функція $f$ використовується для розпізнавання облич у зображеннях, які належать до розподілу даних $D$, розв'язання різних підзадач з $S$, таких як класифікація та визначення ключових точок, і параметри мережі VGG16 $\theta$ допомагають оптимізувати цю функцію для кращої ефективності та генералізації. На виході модель повертає $D$ класів, кожен з яких представлений вектором ймовірностей $\hat{y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_n]$, де $\hat{y}_i$ – ймовірність належності зображення до класу $i$, та вектор істинних міток $y = [y_1, y_2, y_3, \dots, y_n]$, де $y_i = 1$, якщо зображення належить до класу $i$, та $y_i = 0$ для всіх інших класів. Формула $L(f(X), D)$ для цієї задачі має наступний вигляд:

$$L(\hat{y}, y) = \sum_{i=0}^{D} y_i * log(\hat{y}_i) \quad (2)$$

*Отже, задача полягає у мінімізації функції втрат* $\boldsymbol{L(f(X), D)}$ *за допомогою навчання мережі* $\boldsymbol{f}$ *з використанням даних* $\boldsymbol{D}$*. Тобто*

$$\sum_{i=0}^{D} y_i * log(\hat{y}_i) \rightarrow min. \quad (3)$$

Для реалізації алгоритму ми повинні окреслити план дій, у який входять наступні кроки:

1. Попередня обробка даних:
- Нормалізація розміру та освітлення зображення
- Виявлення обличчя на зображеннях
- Виявлення основних рис обличчя, таких як рот, ніс та очі
2. Векторизація:
- Перетворення виявлених рис обличчя у вектори характеристик
3. Паралелізація алгоритму:
- Розбиття мережі VGG16 на підшари та виконання кожного підшару на окремому процесорі (розпаралелювання за підзадачами).

# РОЗДІЛ 2
# МАТЕРІАЛИ ТА МЕТОДИ ДЛЯ РОЗРОБКИ ІНТЕГРОВАНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ ОБЛИЧ

## 2.1. Процес розпізнавання облич та аугментація даних

Процес розпізнавання обличчя – це технологія, яка дозволяє ідентифікувати та розпізнавати особу за її обличчям. Він заснований на аналізі геометричних та текстурних особливостей обличчя людини.

Основні кроки процесу розпізнавання обличчя включають наступне:

- Захоплення зображення, тобто спочатку використовується камера або інший пристрій для захоплення зображення обличчя. Це може бути фотографія або відеозапис.
- Виявлення обличчя, а саме отримане зображення обробляється для виявлення обличчя. Цей крок включає в себе використання алгоритмів комп'ютерного зору для пошуку ключових ознак, таких як очі, ніс, рот і форма обличчя.
- Нормалізація обличчя – після виявлення обличчя зображення нормалізується, щоб забезпечити однаковий масштаб і орієнтацію для подальшого аналізу. Це може включати зміну розміру, повороту або приведення до стандартної форми.
- Виділення особливостей обличчя, тобто в цьому кроці виконується аналіз обличчя для виділення особливостей, які можуть бути використані для розпізнавання. Це можуть бути такі риси, як контури обличчя, положення очей, рота, носу, брів тощо.
- Витягнення ознак обличчя відбувається після виділення особливостей обличчя зображення обробляється для витягнення числових або векторних даних, які представляють ці особливості. Це можуть бути такі

характеристики, як форма обличчя, текстура шкіри, розташування ключових точок і т.д.

- Зберігання та порівняння ознак та подібності зображень.

Для того щоб отримати ефективну модель, яка з високою точністю буде розпізнавати лице, потрібно мати великий набір даних. Важливо, щоб модель тренувалась на різних варіаціях обличчя людини. Для цього потрібно провести аугментацію даних.

Аугментація даних – це процес штучного збільшення обсягу навчального набору даних шляхом створення нових зображень на основі наявних [14]. В контексті розпізнавання облич, аугментація даних є важливим інструментом для покращення ефективності моделей нейромереж.

Основна мета аугментації даних – розширити різноманітність даних, навчальний набір яких може бути обмеженим. Це допомагає уникнути перенавчання (overfitting) моделі [15], коли вона стає надто специфічною для конкретних зображень навчального набору, і втрачає здатність узагальнювати на нові, раніше не бачені зображення.

Деякі типи аугментації даних, які можуть бути використані для розпізнавання облич, включають:

- Горизонтальне та вертикальне відображення, тобто зображення дзеркально відображаються горизонтально або вертикально.
- Повороти та нахили, а саме: зображення обертаються на певний кут або нахиляються, що дозволяє моделі бути більш робастною до різних позицій облич.
- Зміщення та масштабування передбачає, що зображення зсуваються горизонтально або вертикально, або збільшуються/зменшуються у масштабі. Це допомагає моделі розпізнавати облича з різних положень та розмірів.
- Додавання шуму – до зображення додається випадковий шум, що забезпечує більшу стійкість до шуму в реальних зображеннях.

- Зміна яскравості та контрастності, тобто рівні яскравості та контрастності зображення змінюються для створення різних варіацій.

Ці прийоми аугментації даних допомагають моделі навчитися розпізнавати облича в різних умовах освітлення, змінених позиціях, масштабах та інших перешкод. Вони доповнюють наявні дані, роблять модель більш стійкою до варіацій у вхідних зображеннях та поліпшують її загальну здатність до узагальнення.

Аугментація даних також допомагає уникнути перевантаження деяких класів даних, зокрема облич, які можуть бути представлені в навчальному наборі нерівномірно. Шляхом генерації нових варіацій зображень облич, які відповідають менш представленим класам, можна покращити здатність моделі розпізнавати ці класи.

Крім того, аугментація допомагає зменшити необхідну кількість реальних зображень, які потрібно зібрати для навчання моделі. Замість того, щоб вручну збільшувати обсяг навчального набору даних шляхом збору нових зображень, можна застосувати аугментацію даних, щоб створити штучні варіації і отримати додаткові навчальні приклади з вже існуючого обмеженого набору даних.

В цілому, аугментація даних є важливою стратегією для покращення роботи моделей розпізнавання облич. Вона допомагає зробити модель більш робастною, здатною до узагальнення і ефективною у різних умовах, покращуючи якість розпізнавання та знижуючи ризик перенавчання.

У якості моделі глибокого навчання будемо використовувати VGG16 [16]. Вона була розроблена групою вчених з Університету Оксфорд, Великобританія, у 2014 році. Модель VGG16 є однією з перших моделей, що використовують глибокі CNN для розв'язання завдань класифікації зображень. Модель є досить потужною і ефективною для розпізнавання облич. Вона має глибоку архітектуру з багатьма згортковими шарами, що дозволяє їй виявляти різні характеристики облич та розпізнавати їх з високою точністю.

VGG16 складається з 16 шарів з вагами [17], як представлено на рис. 2.1. Ці шари включають 13 згорткових шарів (Convolutional layers), які слідують один за одним, і 3 повністю з'єднаних шарів (Fully connected layers) в кінці мережі.

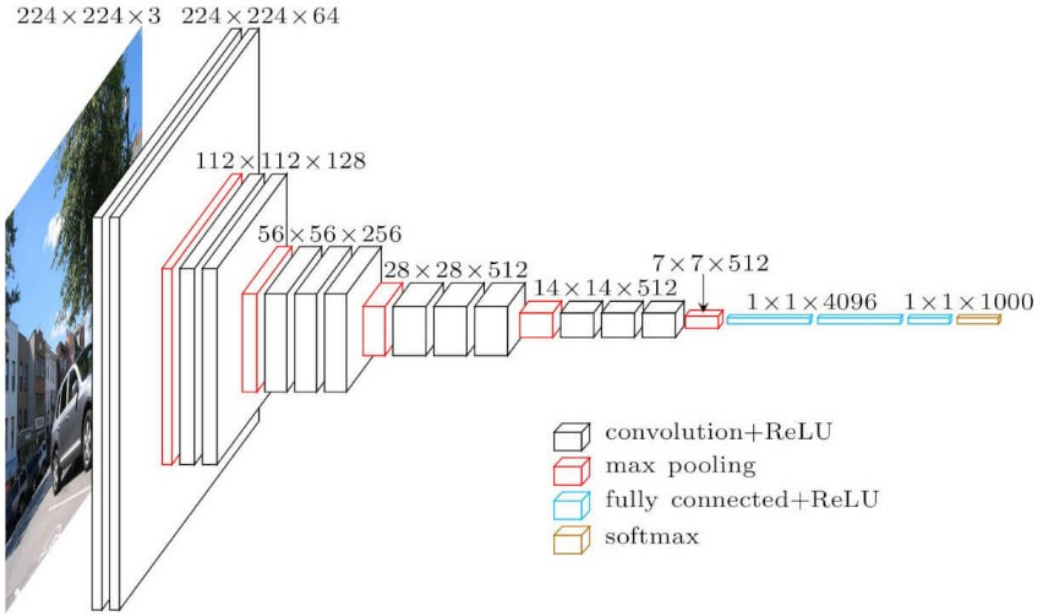


*Рис. 2.1. Архітектура VGG16*

Розглянемо архітектуру VGG16:

- Вхідний шар: вхідним елементом для VGG16 є зображення RGB розміром 224x224 пікселя.
- Згорткові шари: VGG16 починається з серії згорткових шарів. Загалом існує 13 згорткових шарів, і кожен шар має фільтр 3x3 (ядро) з кроком (stride) 1 і заповненням (padding) 1. Ці шари відповідають за вилучення функцій із вхідного зображення. Важливо відмітити, що використання багатьох шарів з невеликими фільтрами згортки є ефективнішим ніж використання меншої кількості шарів з великими фільтрами згортки.
- Max Pooling: після кожного набору згорткових шарів VGG16 використовує шари max pooling, щоб зменшити просторові розміри карт ознак, зберігаючи найважливішу інформацію. Max pooling мають фільтр 2x2 із кроком (stride) 2.

• Регуляризація: Для запобігання перенавчання в моделі VGG16 використовується метод Dropout перед повністю з'єднаними шарами.

• Повноз’язні шари: після шарів згортки та об’єднання VGG16 закінчується трьома Повноз’язними шарами. Кожен повноз’язний шар містить 4096 нейронів, за якими слідує функція активації ReLU. Останній повністю шар створює остаточні прогнози.

• Softmax шар: вихідні дані останнього повнозв’язного шару подаються на рівень softmax, який нормалізує результати в розподіл ймовірностей за класами. Клас з найвищою ймовірністю вважається прогнозованим класом.

Тренування VGG16 вимагає значних обчислювальних ресурсів, і ця модель може бути досить повільною при розгортанні на обмежених ресурсах. Але через її високу точність і зручність використання, вона продовжує бути популярною для задач класифікації зображень. Оскільки VGG16 має велику кількість параметрів, вона може бути схильна до перенавчання, особливо при тренуванні на невеликих наборах даних. Для вирішення цієї проблеми, модель часто використовується в комбінації з техніками аугментації даних.

## 2.2. Запропоноване розпаралелення моделі VGG16 за підзадачами

Модель VGG16 складається з 16 шарів, які мають налаштовані ваги (trainable layers). Ці шари включають 13 згорткових шарів (convolutional layers), 3 повнозв'язаних шари (fully connected layers) та 5 шарів підвибірки (max pooling layers).

Розглянемо схему їх розпаралелювання за підзадачами:

1. Вхідні дані

- Зображення з набору даних подаються на перший блок мережі

2. Поділ на блоки:

- Блок 1: Шари 1-3 (2 згорткових + 1 max pooling)
- Блок 2: Шари 4-6 (2 згорткових + 1 max pooling)
- Блок 3: Шари 7-10 (3 згорткових + 1 max pooling)

- Блок 4: Шари 11-14 (3 згорткових + 1 max pooling)
- Блок 5: Шари 15-18 (3 згорткових + 1 max pooling)
- Блок 6: Шари 19-21 (3 повнозв'язаних шари)

3. Процеси:

- Кожен блок обробляється окремим процесом.
- Процеси обмінюються даними через черги.

4. Черги для передачі даних:

- Черга 1: Передача вхідних даних до процесу 1 (Блок 1)
- Черга 2: Передача даних від процесу 1 до процесу 2 (Блок 2)
- Черга 3: Передача даних від процесу 2 до процесу 3 (Блок 3)
- Черга 4: Передача даних від процесу 3 до процесу 4 (Блок 4)
- Черга 5: Передача даних від процесу 4 до процесу 5 (Блок 5)
- Черга 6: Передача даних від процесу 5 до процесу 6 (Блок 6)
- Черга 7: Передача даних від процесу 6 до остаточних результатів

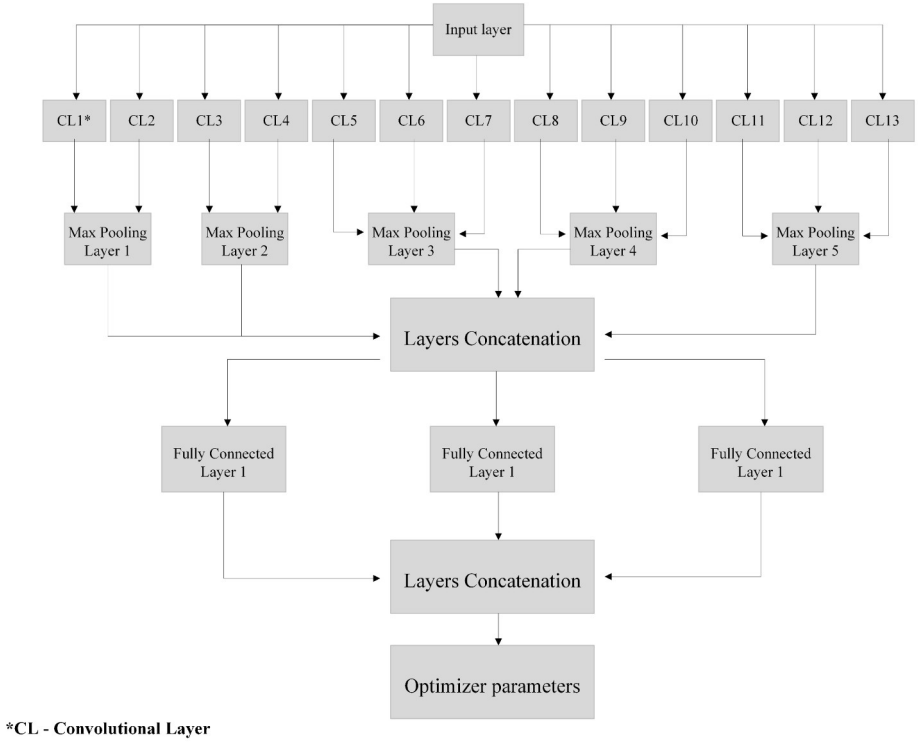


*Рис. 2.2. Схема розпаралелення за підзадачами моделі VGG16*

## 2.3. Використання технології паралельних обчислень Multiprocessing

Multiprocessing – це Python-пакет, який підтримує створення і керування процесами. Він працює як на POSIX-системах (Linux, Mac), так і на Windows.

Розглянемо основні елементи пакету:

1. *Process*

Процеси в пакеті multiprocessing створюються за допомогою класу Process. Для цього створюється об'єкт цього класу, в який передається цільова функція, що має бути виконана процесом, та необхідні аргументи. Після цього викликається метод start(), який ініціює виконання процесу.

2. *Queue*

Queue – клас, що дозволяє процесам обмінюватися даними. Це структура даних "Черга", яка гарантує, що кілька процесів можуть безконфліктно одночасно доступатися до неї.

3. *Pipe*

Pipe – засіб для двостороннього обміну інформацією між двома процесами. Вона може використовуватися для передачі об'єктів між процесами.

4. *Lock*

Lock – об'єкт, що гарантує, що лише один процес у будь-який момент часу може виконувати певну частину коду. Це може бути корисно для управління доступом до ресурсів, таких як функції введення/виведення даних, щоб кілька процесів не мали одночасного доступу до одного й того ж ресурсу.

Завдяки абстракціям Python, пакет multiprocessing забезпечує простоту в паралелізації виконання коду.

Процес розбиття моделі на етапи може виглядати наступним чином:

- Обробка зображення за допомогою згорткових шарів.
- Обчислення активаційних функцій для різних шарів.
- Оновлення ваг і обчислення градієнтів.
- Обробка результатів згорткових шарів за допомогою пов'язаних шарів.

- Класифікація зображення за допомогою вихідного шару.

Загальний опис запропонованого паралельного алгоритму з використанням multiprocessing:

- Спочатку модель розбивається на блоки, кожен з яких містить кілька шарів мережі (залежно від кількості створених процесів, кількість шарів у блоці може коливатися від 1 до 16).
- Для кожного блоку створюється окремий процес.
- Кожен процес отримує вхідні дані, обробляє їх через свій блок шарів і передає результат наступному процесу.
- Потрібно синхронізувати процеси та організувати потік даних за допомогою черги (Queue).
- Останній процес обробляє свої дані та передає фінальний результат головному процесу.

Цей підхід може бути ефективним для реалізації інтегрованої інформаційної системи для управління правоохоронною діяльністю, де штучний інтелект використовує паралельну обробку для швидкої та ефективної обробки великого обсягу даних.

## 2.4. Використання технології паралельних обчислень CUDA

CUDA (англ. Compute Unified Device Architecture) – технологія GPGPU (General-Purpose Computing on Graphics Processing Units), що дозволяє програмістам реалізовувати алгоритми для виконання на графічних процесорах відеокарт GeForce восьмого покоління та старших, використовуючи спрощену мову програмування, таку як C або Python. Технологія CUDA була розроблена компанією Nvidia і дозволяє впроваджувати спеціальні функції безпосередньо в код на C або Python.

CUDA містить два основних API:

– CUDA Runtime API (високорівневий)

– CUDA Driver API (низькорівневий)

Хоча ці API не можуть використовуватися одночасно в одній програмі, високорівневий API працює на основі низькорівневого і транслює виклики в інструкції, які обробляються низькорівневим Driver API. Проте навіть високорівневий API вимагає знання основ про пристрій і роботу відеокарт NVIDIA.

Модель програмування в CUDA передбачає групування потоків. Потоки об'єднуються в блоки потоків (thread blocks), які можуть бути одновимірними або двовимірними. Ці блоки взаємодіють між собою через спільну пам'ять і точки синхронізації. Програма (ядро, kernel) виконується над сіткою блоків потоків (grid), де одночасно виконується лише одна сітка. Кожен блок може мати одновимірну, двовимірну або тривимірну форму і може складатися до 512 потоків. Кожен потік виконує одну й ту саму функцію, але для різних даних, і GPU спеціально оптимізовано для такої роботи.

Загальний опис паралельного алгоритму з використанням Nvidia CUDA:

Основна операція в CNN – згортка. Для двовимірної згортки можна записати наступну формулу:

$$Y(i,j) = \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} X(i+m, j+n) \cdot W(m,n), \quad (4)$$

де

- $Y(i,j)$ – вихідний тензор,
- $X(i+m, j+n)$ – вхідний тензор,
- $W(m,n)$ – ядро згортки,
- $K$ – розмір ядра

Кожен елемент вихідного тензора $Y(i,j)$ обчислюється окремим потоком на GPU. Для цього кожен потік виконує згорткову операцію для відповідного елемента. Всі потоки організовані в блоки, і всі блоки організовані в сітки.

Наприклад, якщо ми маємо вхідний тензор розміром $N \times N$ і ядро розміром $K \times K$, кожен блок може обробляти окремий підмасив розміром $(N - K + 1) \times (N - K + 1)$.

Для активаційної функції ReLU:

$$ReLU(x) = max(0, x). \quad (4)$$

Кожен потік застосовує функцію активації до окремого елемента матриці.

Для max-pooling з вікном розміру $2 \times 2$:

$$MaxPool(x, y) = max\{X(x, y), X(x + 1, y), X(x, y + 1), X(x + 1, y + 1)\}. \quad (5)$$

Паралелізація max-pooling передбачає, що кожен потік обчислює окремий елемент вихідної матриці після підвибірки.

Паралельна реалізація згортки на CUDA:

1. Завантаження вхідних даних і ядра згортки в пам'ять GPU.
2. Розбиття даних на блоки. Наприклад, якщо ми маємо вхідний тензор розміром $N \times N$ і ядро розміром $K \times K$, кожен блок може обробляти окремий підмасив розміром $(N - K + 1) \times (N - K + 1)$.
3. Запуск паралельного ядра CUDA, де кожен потік відповідає за обчислення певного набору елементів вихідного тензора.
4. Копіювання результатів з пам'яті GPU назад у пам'ять CPU.

## 2.5 Використання технології паралельних обчислень MPI

Під паралельною програмою в рамках MPI розуміють набір незалежних процесів, кожен з яких виконує свою власну програму (необов'язково одну і ту ж). Процеси MPI програми взаємодіють один з одним за допомогою виклику комунікаційних процедур. Як правило, кожен процес виконується у своєму власному адресному просторі, однак допускається і режим поділу пам'яті.

Для ідентифікації наборів процесів вводиться поняття групи, вона об'єднує всі або якусь частину процесів. Кожна група утворює область зв'язку, з якою пов'язують спеціальний об'єкт комунікатор. Процеси всередині групи індексуються цілим числом в діапазоні від 0 до **groupsize** - 1. Всі комунікаційні операції з деяким комунікатором будуть виконуватися тільки всередині групи – MPI_COMM_WORLD. MPI – це бібліотека функцій, забезпечує взаємодію паралельних процесів за допомогою механізму передачі повідомлень. Це досить об'ємна і складна бібліотека, що складається приблизно з 130 функцій.

Розглянемо основні функції MPI:

1. MPI_Init – будь яка прикладна MPI-програма (додаток) повинна починатися з виклику функції ініціалізації MPI-функції MPI_Init. У результаті виконання цієї функції створюється група процесів, в яку поміщаються всі процеси додатки, створюється область зв'язку, яка об'єднує всі процеси-додатки.

2. MPI_Finalize – За допомогою цієї функції закриваються всі MPI-процеси і ліквідуються всі галузі зв'язку

3. MPI_Comm_size – Дана функція повертає кількість процесів у галузі зв'язку комунікатора.

4. MPI_Comm_rank – Ця функція повертає номер процесу, що викликав цю функцію. Номери процесів лежать в діапазоні від 0 до size-1 (значення size може бути визначено за допомогою попередньої функції).

5. MPI_Send – Для відправлення повідомлень іншим процесам, може відправляти як і одному конкретному процесу – за його номером, так і всім процесам.

6. MPI_Recv – Для прийняття повідомлень від інших процесів, так само як і MPI_Send, може приймати повідомлення як від одного конкретного процесу, так і від всіх.

Принцип паралелізації з використанням MPI (mpi4py):

- Модель розбивається на кілька блоків, кожен з яких обробляється окремим процесом.
- Кожен блок складається з кількох послідовних шарів моделі.
- Вхідні дані передаються від одного процесу до іншого після обробки кожним блоком шарів.
- Для передачі даних між процесами використовуються функції MPI, такі як *Send* і *Recv*.
- Останній процес збирає результати і передає їх для подальшої обробки або виведення.

Передача даних між процесами виглядає наступним чином:

Процес $P_i$ передає дані $Y_i$ процесу $P_{i+1}$: $MPI.Send(Y_i, dest = P_{i+1})$.

Процес $P_{i+1}$ отримує $Y_i$ від процесу $P_i$ : $Y_{i+1} = MPI.Recv(source = P_i)$.

## 2.6 Використання технології паралельних обчислень OpenMP

OpenMP – це стандарт, який включає директиви компілятора, бібліотеки та системні змінні для вказівки паралелізму в системах із спільною пам'яттю. OpenMP реалізує модель SPMD (Single Program Multiple Data), в рамках якої всі паралельні потоки виконують той самий код.

Основні директиви OpenMP:

1. parallel – При вході в паралельну область породжуються нові omp_num_threads-1 потоки, кожен потік отримує свій унікальний номер, причому потік, що породжує, отримує номер 0 і стає основним потоком групи. Інші потоки отримують номер від 1 до omp_num_threads-1. Кількість потоків, що виконують дану паралельну область, залишається незмінною до моменту виходу з області. При виході з паралельної області виробляється неявна синхронізація і знищуються всі потоки, окрім того, що породив.
2. single – директива, що визначає структурний блок програми, який буде виконуватися лише один раз.

3. master – схожа на директиву single, але визначений блок програми виконується лише потоком з номером 0 з усіх паралельних потоків. Вона не включає неявну синхронізацію.

4. barrier – директива синхронізації типу barrier, що змушує потоки очікувати завершення роботи всіх інших паралельних потоків перед досягненням точки barrier.

Функція omp_set_num_threads дозволяє змінювати кількість потоків у програмі, передаючи ціле число – бажану кількість потоків.

Оскільки стандартна реалізація CPython не підтримує OpenMP безпосередньо, паралелізація обчислень у CNN за підзадачами з використанням OpenMP у Python зазвичай здійснюється через бібліотеки, що підтримують багатопоточність і багатопроцесорність. У нашій роботі ми зупинилися на бібліотеці Cython. Принцип розпаралелювання з OpenMP у Cython подібний до попередніх, але ця бібліотека дозволяє автоматично розподіляти та виконувати цикли паралельно.

Кроки для реалізації паралелізації з OpenMP:

1. Підготовка Cython-файлу для OpenMP

- Створення файлу з розширенням .pyx для обчислень у Cython.
- Включення заголовка OpenMP і використання директив OpenMP у циклах.

2. Компіляція Cython коду

- Створення файлу setup.py для компіляції Cython-коду з підтримкою OpenMP.

3. Використання паралельної функції у Python

- Після компіляції Cython-функції її можна використовувати в Python-коді для паралельного обчислення згорткових шарів.

Цей підхід дозволяє ефективно реалізувати паралельну обробку даних в CNN, підвищуючи швидкість обчислень через використання можливостей OpenMP в рамках Python.

# РОЗДІЛ 3
# АПРОБАЦІЯ ІНТЕГРОВАНОЇ АВТОМАТИЗОВАНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ ОБЛИЧ НА ОСНОВІ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

## 3.1 Аналіз чисельних експериментів

Чисельні експерименти є важливою частиною дослідження, адже вони дозволяють перевірити ефективність оптимізацій та пришвидшень алгоритмів на даних з реального світу. Вони дозволяють оцінити ефективність застосування методів паралельних обчислень, виміряти наскільки швидше працює система автоматичного розпізнавання облич порівняно з не розпаралеленим варіантом, а також, виявити можливі недоліки та проблеми паралельних обчислень.

Експерименти також дозволяють нам перевірити різні сценарії роботи, перевірка яких в умовах реального використання викликала б труднощі. Наприклад: розпізнавання облич під різними кутами, з темним або занадто яскравим освітленням, за різних погодних умов.

Вибраний нами датасет містить більше 18000 фотографій близько 4000 осіб. Після тренування моделі протягом 10 епох нам вдалося досягти точності тренування, що становить ~98.5%, а точність перевірки сягала ~97%. Нижче знаходиться візуалізація результатів (див. рис. 3.1).

Експерименти будуть проводитися на процессорах:

1. AMD Ryzen 7 7735U @ 2.70GHz, 8 cores 16 threads
2. Intel® Core™ i5-1135G7 @ 2.40GHz 4 cores, 8 threads

Код реалізації алгоритму знаходиться в додатку.

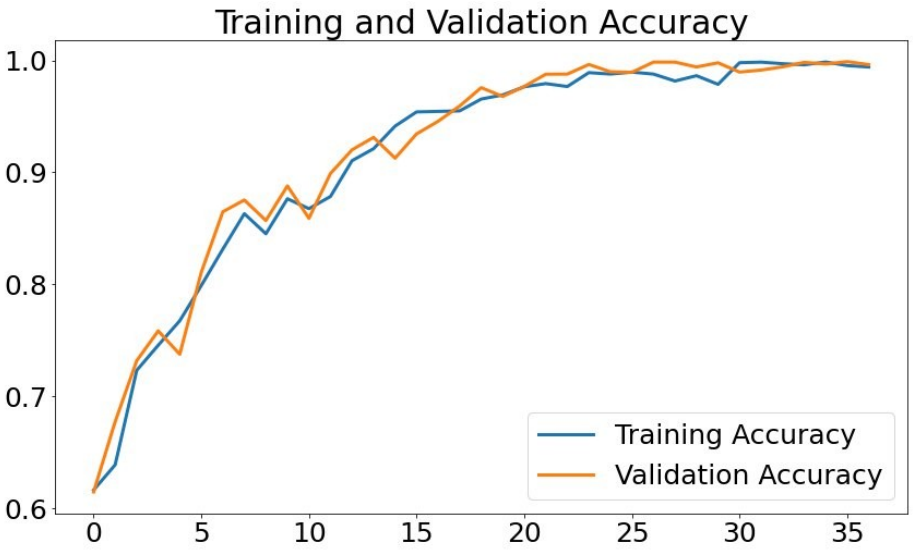


*Рис. 3.1. Візуалізація результатів тренування та тестування моделі*

## 3.2 Оцінка швидкодії тренування моделі без розпаралелення

Ми будемо розпаралелювати епохи тренування моделі відповідно часова оцінка складності алгоритму складає $O(N)$. Проведемо тренування з різною кількістю даних для перевірки оцінки складності та результати представимо в табл. 3.2.

*Табл. 3.1. Час послідовного тренування моделі на різних процесорах з різною кількістю даних, сек*

| Number of data | Ryzen 7 7735U | Core i5-1135G7 |
|---|---|---|
| 1000 | 1715 | 3702 |
| 2000 | 2543 | 5329 |
| 4000 | 4942 | 10702 |


*Рис. 3.2. Відношення часу послідовного тренування до кількості даних на різних процесорах*

Як бачимо по рис. 3.1, часова складність алгоритму є приблизно лінійною.

## 3.3 Оцінка швидкодії тренування моделі з розпаралеленням за допомогою NVIDIA CUDA

Для цього експерименту ми будемо використовувати відеокарту GPU NVIDIA Tesla T4 від Google Collab. CUDA працює виконанням однакових завдань на різних ядрах графічного процесора, відповідно використовується розпаралелення по даних. Результати представлено в табл. 3.2, і візуалізовано на рис.3.3.

*Табл. 3.2. Час тренування на GPU NVIDIA Tesla T4 з різною кількістю даних*, сек.

| Number of data | GPU NVIDIA Tesla T4 |
|---|---|
| 1000 | 52 |
| 2000 | 97 |
| 4000 | 160 |

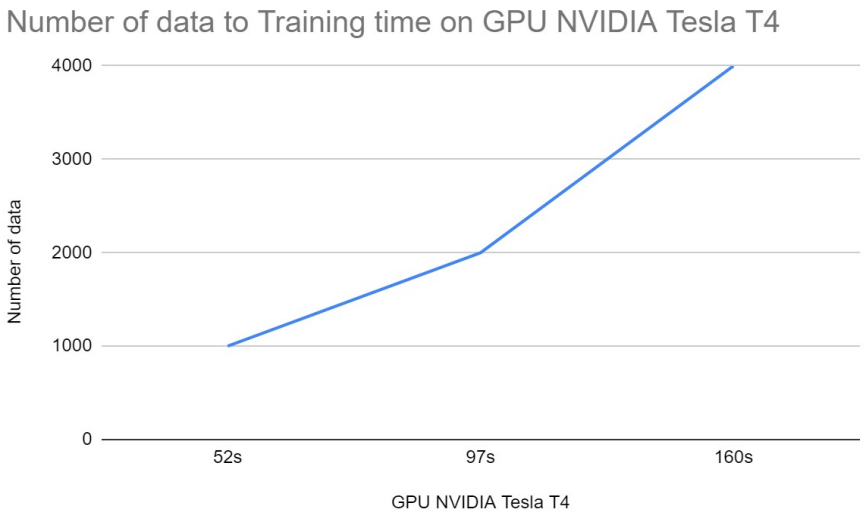


*Рис. 3.3. Відношення часу тренування до кількості даних на GPU NVIDIA Tesla T4*

## 3.4 Оцінка швидкодії тренування моделі з розпаралеленням за допомогою OpenMP

З використанням методів паралельних обчислень ми зменшуємо часову складність алгоритму до $O(N/P)$, де $P$ – кількість потоків. Результати подано в табл. 3.3 та візуалізовано на рис. 3.4.

Tensorflow – фреймворк який ми використовуємо для тренування нашої CNN моделі, має вбудовану підтримку OpenMP, для використання багатоядерного режиму потрібно зробити деякі налаштування змінних середовища:

```
import os
import tensorflow as ts

num_threads = 8

os.environ["OMP_NUM_THREADS"] = str(num_threads)
os.environ["TF_NUM_INTRAOP_THREADS"] = str(num_threads)
os.environ["TF_NUM_INTEROP_THREADS"] = str(num_threads)

ts.config.threading.set_inter_op_parallelism_threads(num_threads
)
ts.config.threading.set_intra_op_parallelism_threads(num_threads
)
```

```
ts.config.set_soft_device_placement(True)
```

*Табл. 3.3. Час паралельного тренування моделі за допомогою OpenMP на різних процесорах з різною кількістю даних, сек*

| number of threads | number of data | | | | | |
|---|---|---|---|---|---|---|
| | 1000 | | 2000 | | 4000 | |
| | Ryzen 7 7735U | Core i5-1135G7 | Ryzen 7 7735U | Core i5-1135G7 | Ryzen 7 7735U | Core i5-1135G7 |
| 2 | 1212 | 2460 | 1804 | 3903 | 2741 | 6015 |
| 4 | 568 | 1251 | 895 | 1951 | 1369 | 2971 |
| 8 | 399 | 851 | 631 | 1351 | 961 | 2101 |

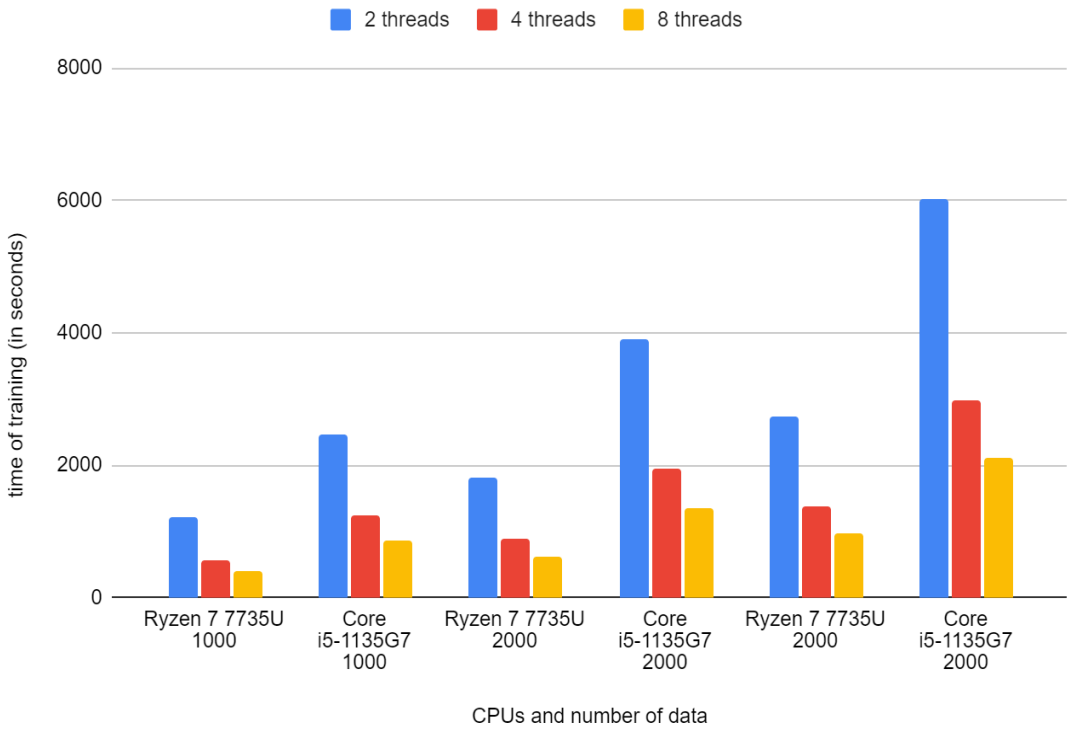


*Рис. 3.4. Відношення часу тренування з OpenMP до кількості даних на різних процесорах*

## 3.5 Оцінка швидкодії тренування моделі з розпаралеленням за допомогою multiprocessing

Так само, як і з OpenMP, з multiprocessing, ми очікуємо часову складність алгоритму $O(N/P)$, де $P$ – кількість процесів. Результати подано в табл. 3.4 та візуалізовано на рис. 3.5.

*Табл. 3.4. Час паралельного тренування моделі за допомогою multiprocessing на різних процесорах з різною кі*лькістю даних

| number of threads | number of data | | | | | |
|---|---|---|---|---|---|---|
| | 1000 | | 2000 | | 4000 | |
| | Ryzen 7 7735U | Core i5-1135G7 | Ryzen 7 7735U | Core i5-1135G7 | Ryzen 7 7735U | Core i5-1135G7 |
| 2 | 1377s | 3219s | 2109s | 4412s | 3521s | 8121s |
| 4 | 710s | 3031s | 1127s | 2381s | 2621s | 5133s |
| 8 | 601s | 1353s | 991s | 1994s | 1849s | 4241s |

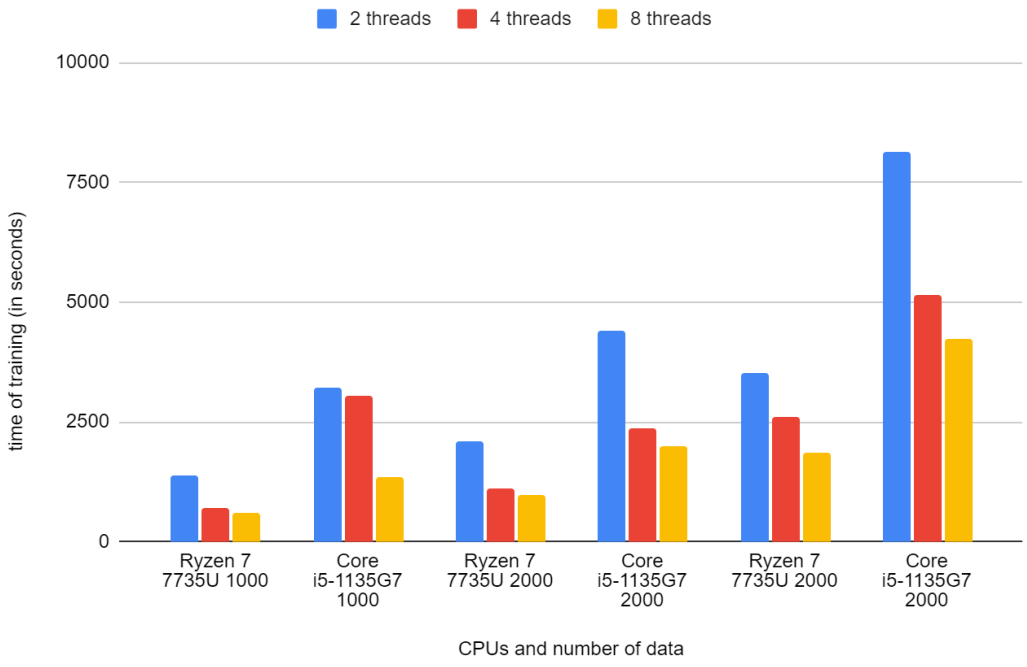


*Рис. 3.5. Відношення часу тренування з multiprocessing до кількості даних на різних процесорах*

## 3.6 Оцінка швидкодії тренування моделі з розпаралеленням за допомогою MPI

Ми все ще очікуємо часову складність алгоритму $O(N/P)$. Але Tensorflow не має вбудованої підтримки MPI, тому її прийдеться реалізувати за допомогою бібліотеки mpi4py. Результати подано в табл. 3.5 та візуалізовано на рис. 3.6.

*Табл. 3.5. Час паралельного тренування моделі за допомогою MPI на різних процесорах з різною кількістю даних, сек.*

| number of threads | number of data | | | | | |
|---|---|---|---|---|---|---|
| | 1000 | | 2000 | | 4000 | |
| | Ryzen 7 7735U | Core i5-1135G7 | Ryzen 7 7735U | Core i5-1135G7 | Ryzen 7 7735U | Core i5-1135G7 |
| 2 | 1358 | 2551 | 1917 | 4001 | 2813 | 6207 |
| 4 | 721 | 1338 | 995 | 2044 | 1445 | 3031 |
| 8 | 519 | 944 | 712 | 1493 | 1003 | 2233 |

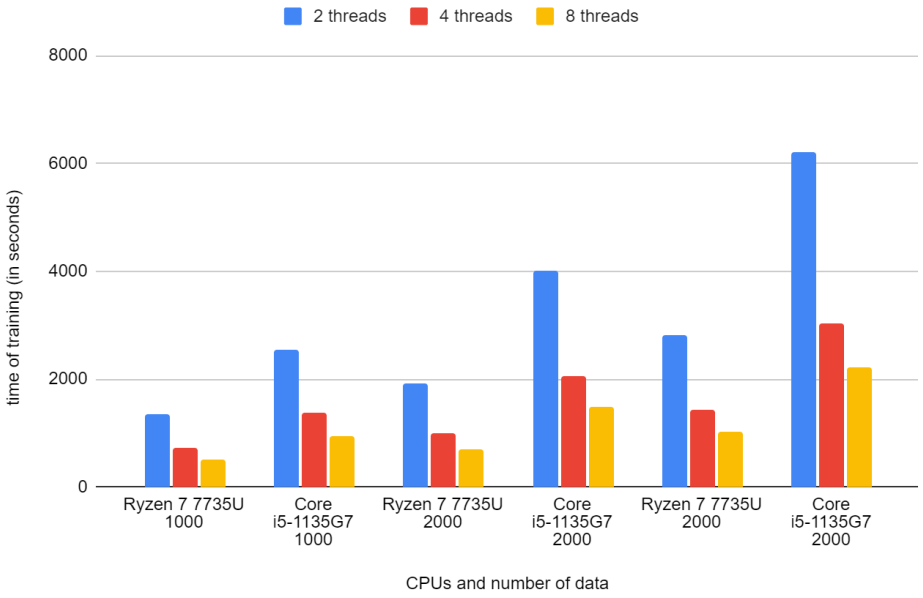


*Рис. 3.6. Відношення часу тренування з MPI до кількості даних на різних процесорах*

# РОЗДІЛ 4

# ОБГОВОРЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

## 4.1. Аналіз отриманих показників ефективності запропонованого паралельного підходу

Для оцінки використання методів паралельних обчислень використовують коефіцієнти пришвидшення та ефективності.

Паралельне пришвидшення ($S$) визначається як відношення часу тренування моделі послідовним шляхом (використовуючи одне ядро) до часу тренування моделі паралельним шляхом (використовуючи $k$ ядер, де $k$ – більше рівне двох). Пришвидшення вказує в скільки разів час паралельного тренування швидше за час послідовного.

$$S = \frac{T_s}{T_p}, \text{ де } T_s - \text{послідовний час}, T_p - \text{паралельний час}.$$

Паралельна ефективність ($E$) визначається як відношення пришвидшення до кількості ядер k. Ефективність вказує наскільки ефективно використовується кожне ядро, тут 1 є максимальним значенням.

$$E = \frac{S}{k}, \text{ де } S - \text{паралельне пришвидшення}, k - \text{кількість ядер}.$$

Для обговорення наших результатів порахуємо ці коефіцієнти для всіх способів паралелізації (див. таблиці 4.1-4.14 та рис. 4.1-4.14).

***Табл. 4.1. Показники пришвидшення Nvidia CUDA***

| Number of data | Пришвидшення для Ryzen 7 7735U | Пришвидшення для Core i5-1135G7 |
|---|---|---|
| 1000 | 32,981 | 72,192 |
| 2000 | 26,216 | 54,938 |
| 4000 | 30,888 | 66,897 |

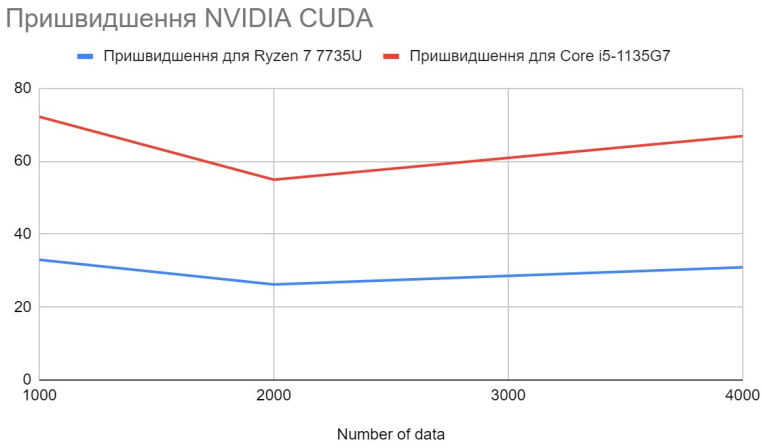


*Рис. 4.1. Візуалізація показників пришвидшення Nvidia CUDA*

***Табл. 4.2. Показники пришвидшення OpenMP для Ryzen 7 7735U для різної кількості даних***

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 1,415 | 1,409 | 1,779 |
| 4 | 3,019 | 2,731 | 3,610 |
| 8 | 4,298 | 4,030 | 5,143 |

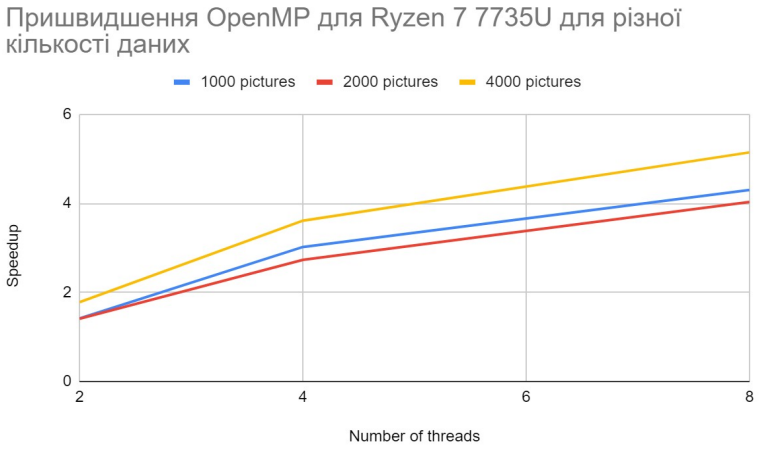


*Рис. 4.2. Візуалізація показників пришвидшення OpenMP для Ryzen 7 7735U для різної кількості даних*

*Табл 4.3.* Показники пришвидшення OpenMP для Core i5-1135G7 для різної кількості даних

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 1,505 | 1,365 | 1,803 |
| 4 | 2,959 | 2,731 | 3,602 |
| 8 | 4,350 | 3,944 | 5,093 |

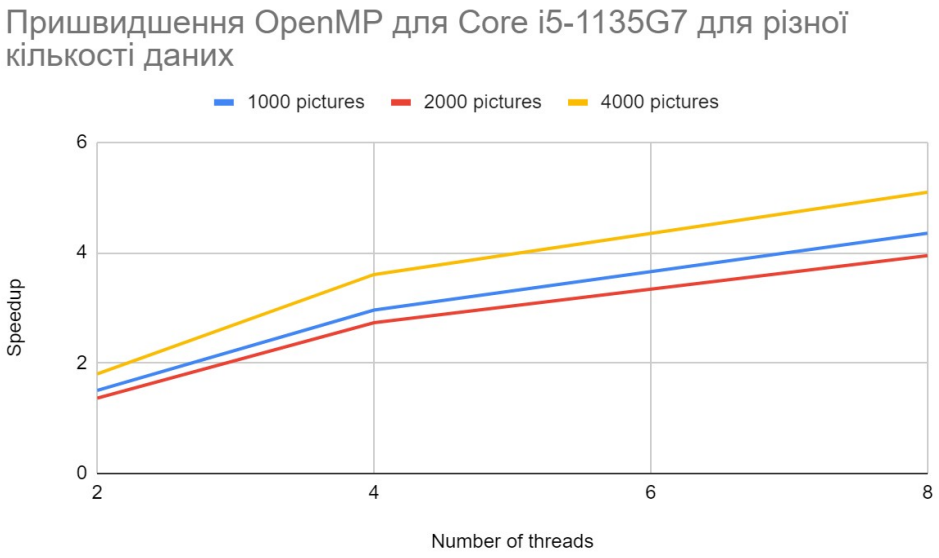


*Рис. 4.3. Показники пришвидшення OpenMP для Core i5-1135G7 для різної кількості даних*

*Табл 4.4. Показники ефективності OpenMP для Ryzen 7 7735U для різної кількості даних*

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 0,708 | 0,705 | 0,902 |
| 4 | 0,755 | 0,710 | 0,903 |
| 8 | 0,537 | 0,504 | 0,643 |

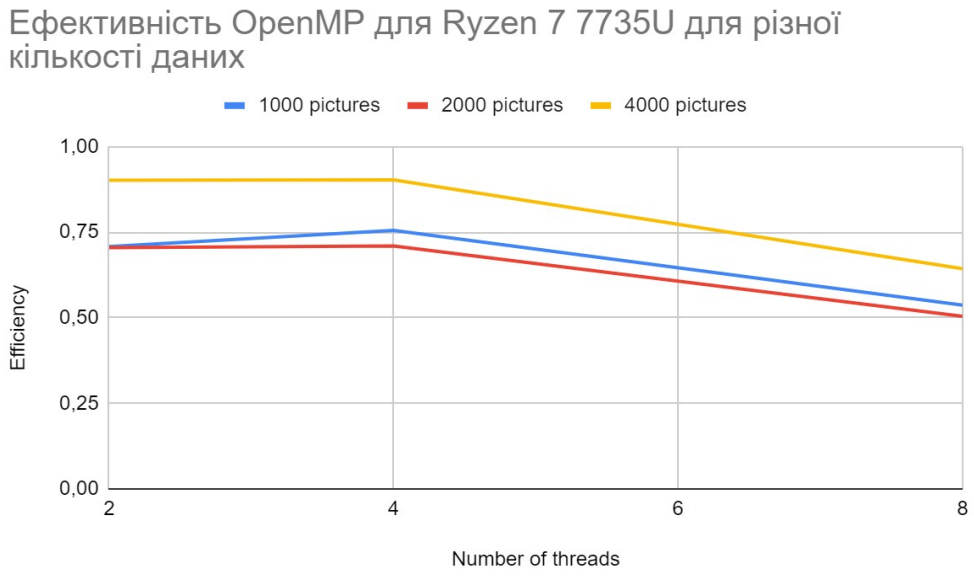


*Рис. 4.4. Показники ефективності OpenMP для Ryzen 7 7735U для різної кількості даних*

*Табл 4.5. Показники ефективності OpenMP для Core i5-1135G7 для різної кількості даних*

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 0,753 | 0,683 | 0,890 |
| 4 | 0,740 | 0,683 | 0,901 |
| 8 | 0,544 | 0,493 | 0,637 |

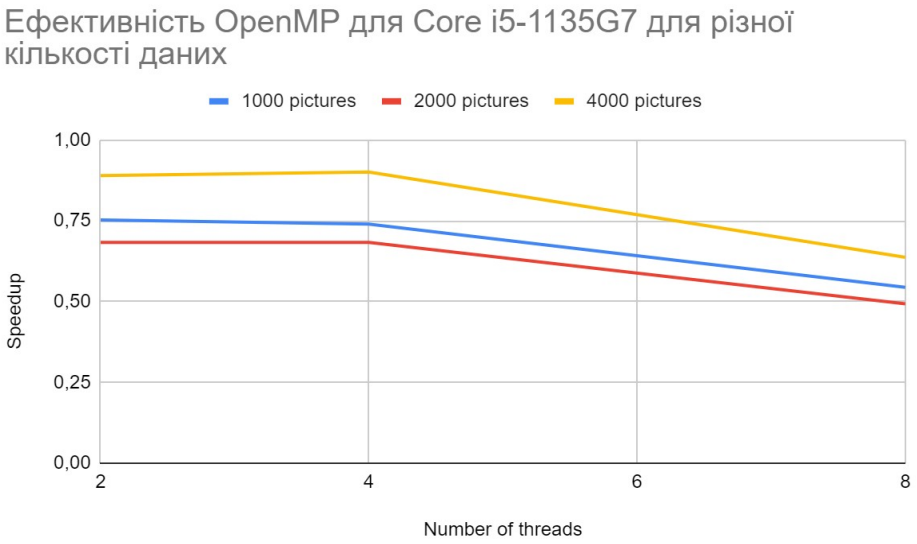


*Табл. 4.5. Показники ефективності OpenMP для Core i5-1135G7 для різної кількості даних*

***Табл. 4.6. Показники пришвидшення multiprocessing для Ryzen 7 7735U для різної кількості даних***

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 1,245 | 1,206 | 1,404 |
| 4 | 2,415 | 2,256 | 1,886 |
| 8 | 2,853 | 2,566 | 2,669 |

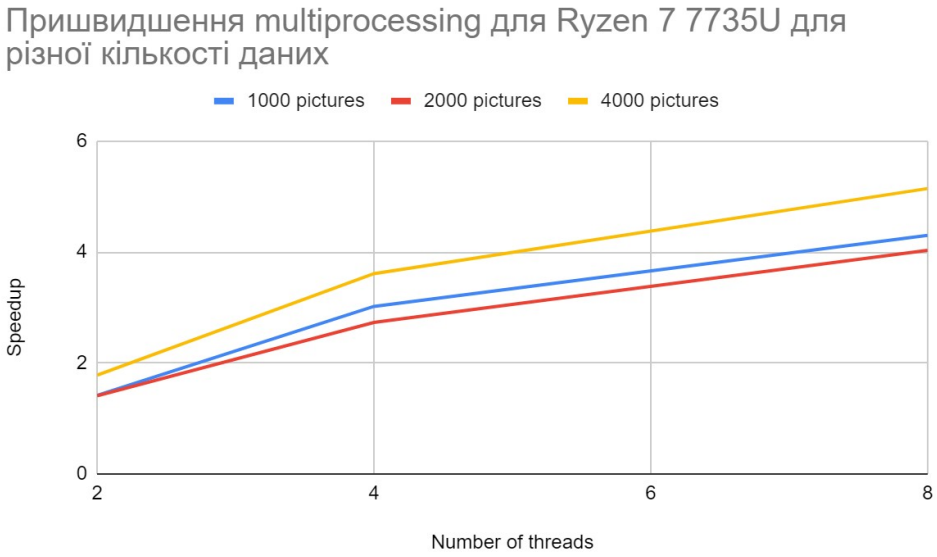


*Рис. 4.6. Показники пришвидшення multiprocessing для Ryzen 7 7735U для різної кількості даних*

*Табл. 4.7. Показники пришвидшення multiprocessing для Core i5-1*135G7 для різної кількості даних

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 1,151 | 1,208 | 1,318 |
| 4 | 1,221 | 2,238 | 2,085 |
| 8 | 2,736 | 2,673 | 2,523 |

*Табл. 4.8. Показники ефективності multiprocessing для Ryzen 7 7735*U для різної кількості даних

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 0,623 | 0,603 | 0,702 |
| 4 | 0,604 | 0,564 | 0,472 |
| 8 | 0,357 | 0,321 | 0,334 |

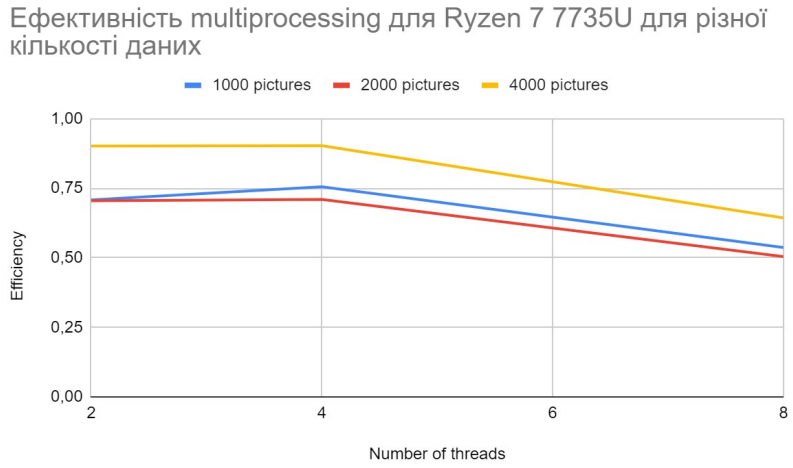


*Рис. 4.8. Показники ефективності multiprocessing для Ryzen 7 7735U для різної кількості даних*

*Табл 4.9. Показники ефективності multi*processing для Core i5-1135G7 для різної кількості даних

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 0,576 | 0,604 | 0,659 |
| 4 | 0,305 | 0,561 | 0,521 |
| 8 | 0,342 | 0,334 | 0,315 |

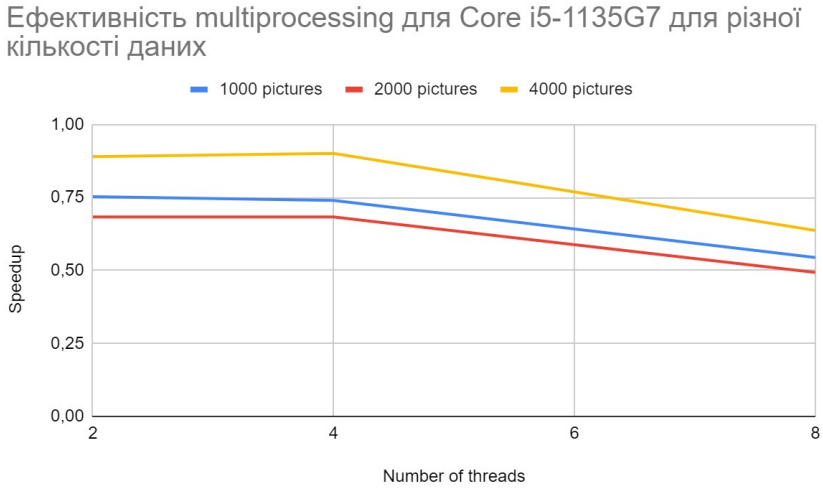


*Рис. 4.9. Показники ефективності multiprocessing для Core i5-1135G7 для різної кількості даних*

***Табл. 4.10. Показники пришвидшення MPI для Ryzen 7 7735U для різної кількості даних***

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 1,263 | 1,327 | 1,757 |
| 4 | 2,379 | 2,556 | 2,417 |
| 8 | 3,304 | 3,572 | 3,311 |

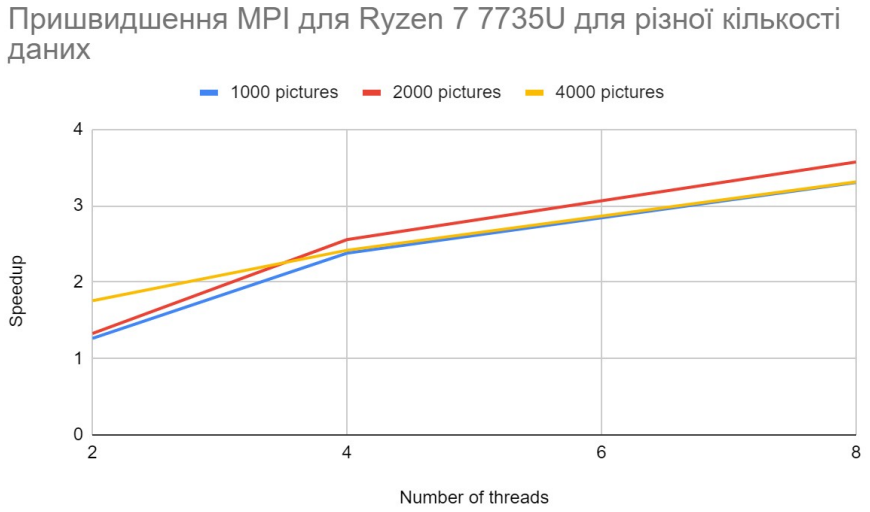


*Рис. 4.10. Показники пришвидшення MPI для Ryzen 7 7735U для різної кількості даних*

*Табл. 4.11.* Показники пришвидшення MPI для Core i5-1135G7 для різної кількості даних

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 1,451 | 1,331 | 1,724 |
| 4 | 2,767 | 2,607 | 3,531 |
| 8 | 3,922 | 3,567 | 4,793 |

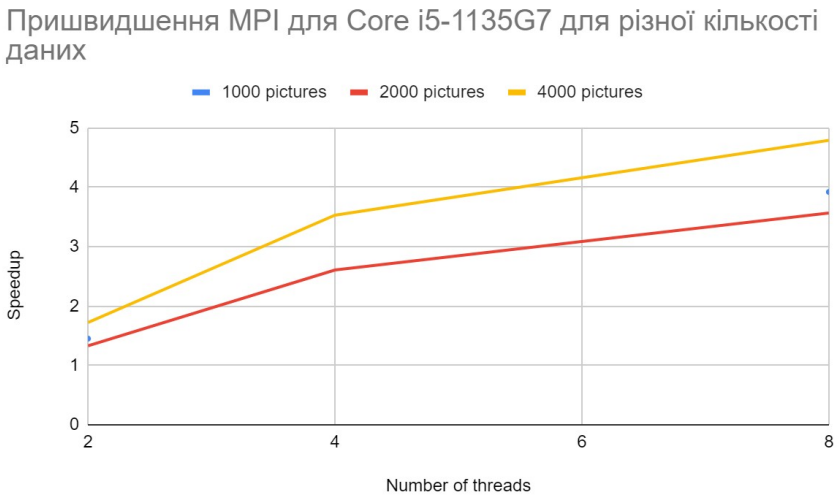


*Рис. 4.11. Показники пришвидшення MPI для Core i5-1135G7 для різної кількості даних*

*Табл. 4.12. Показники ефективності MPI для Ryzen 7 7735U для різної кількості даних*

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 0,632 | 0,663 | 0,879 |
| 4 | 0,595 | 0,639 | 0,604 |
| 8 | 0,413 | 0,447 | 0,414 |

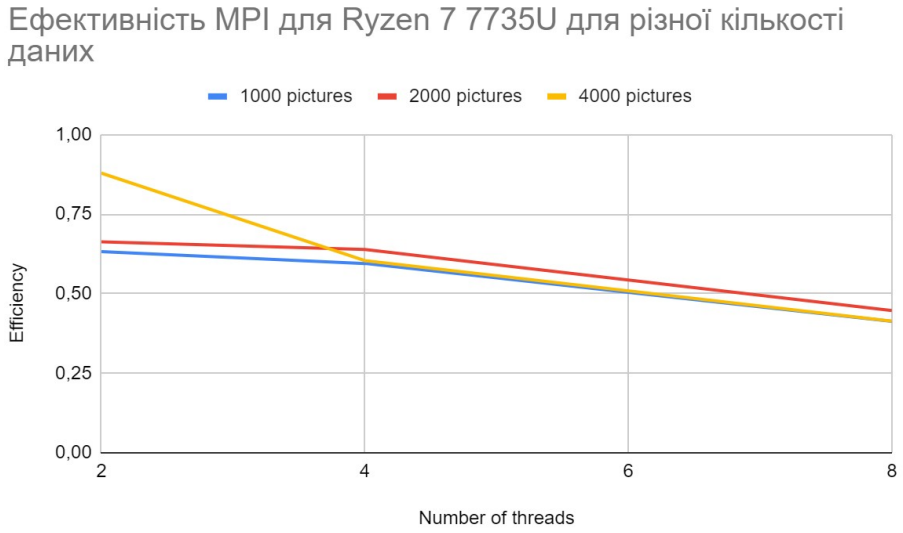


*Рис. 4.12. Показники ефективності MPI для Ryzen 7 7735U для різної кількості даних*

*Табл 4.13.* Показники ефективності MPI для Core i5-1135G7 для різної кількості даних

| Number of threads | 1000 pictures | 2000 pictures | 4000 pictures |
|---|---|---|---|
| 2 | 0,726 | 0,666 | 0,862 |
| 4 | 0,692 | 0,652 | 0,883 |
| 8 | 0,491 | 0,446 | 0,599 |

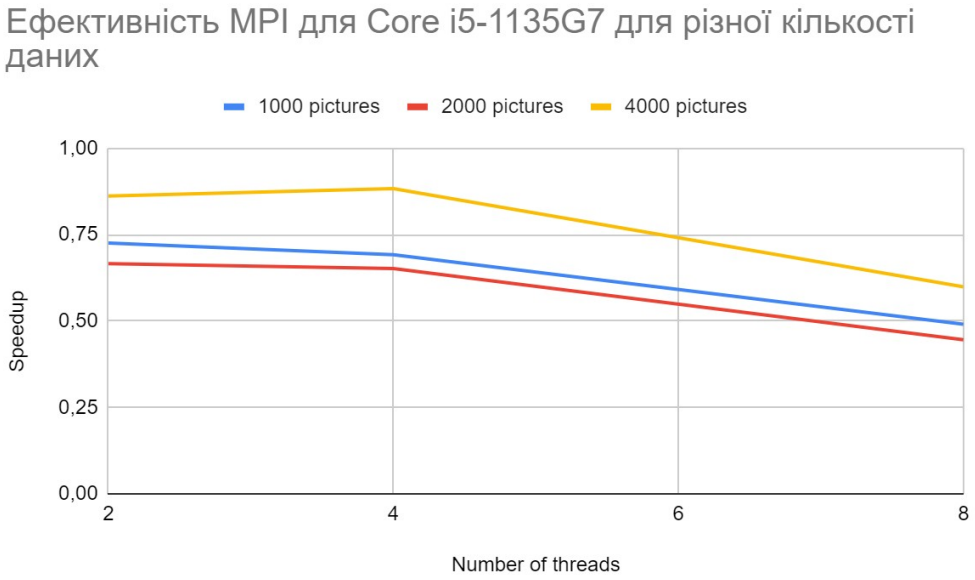


*Табл 4.14. Показники ефективності MPI для Core i5-1135G7 для різної кількості даних*

Проаналізувавши отримані результати, ми спостерігаємо, що пришвидшення росте при збільшенні кількості ядер, але при цьому падає ефективність кожного окремого ядра.

Також за допомогою отриманих коефіцієнтів можемо порівняти роботу різних методів паралельних обчислень, для цього побудуємо табл. 4.15 з найкращих замірів часу тренування CNN моделі кожного методу.

*Табл. 4.15. Порівняння найкращого часу тренування кожної технології, сек*

| Num of data | Sequential | Nvidia CUDA | OpenMP | multiprocessing | MPI |
|---|---|---|---|---|---|
| 1000 | 1715 | 52 | 399 | 601 | 519 |
| 2000 | 2543 | 97 | 631 | 991 | 712 |
| 4000 | 4942 | 160 | 961 | 1849 | 1003 |

Для більш зрозумілого вигляду, зобразимо цю таблицю на рис. 4.15.

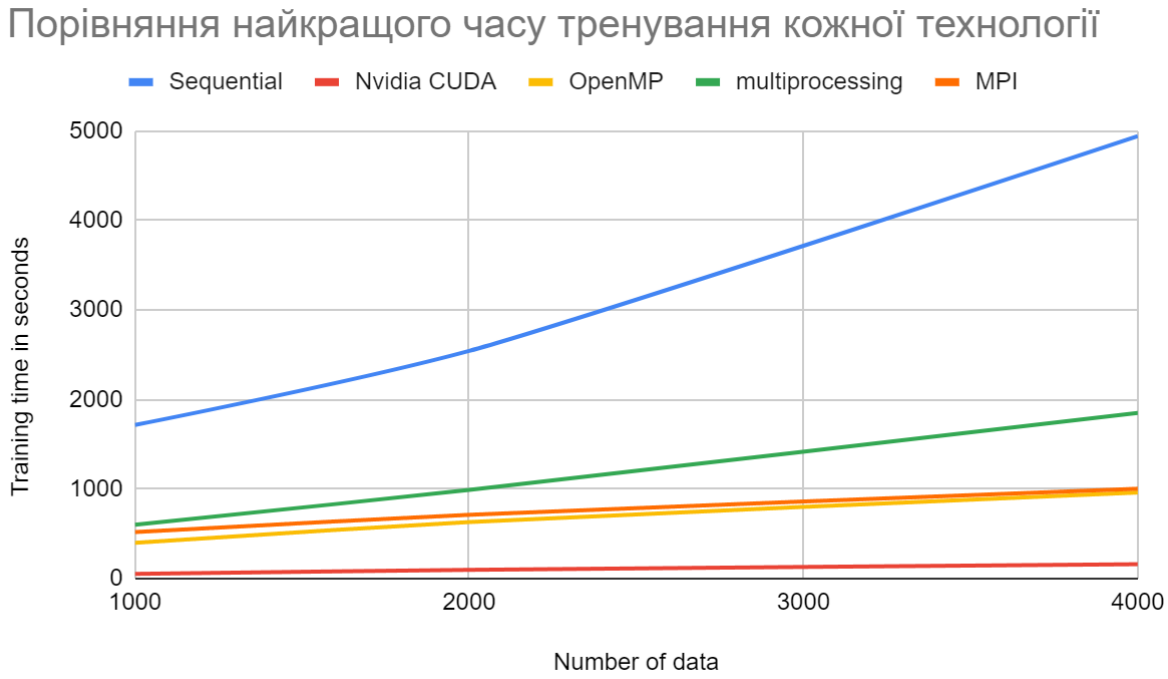


*Рис. 4.15. Порівняння найкращого часу тренування кожної технології*

З рис. 4.15 можемо зробити висновок, що найкращою технологією для розпаралелення є Nvidia CUDA, єдиний її мінус, це те, що для неї необхідна відеокарта Nvidia, також є можливість використання AMD відеокарт з застосуванням технології OpenCL.

Щодо варіантів процесорного розпаралелення, лідерство поділяють технології OpenMP і MPI, коли ж пакет multiprocessing показав найгірші результати, що пов’язано з внутрішньою будовою пакету та мови програмування Python.

## 4.2. Візуалізація роботи автоматизованої системи

На рис. 4.16 представлено зображення з розглядуваного датасету, а результати розпізнавання облич представлено на рис. 4.17.


*Рис. 4.16. Зображення з датасету*

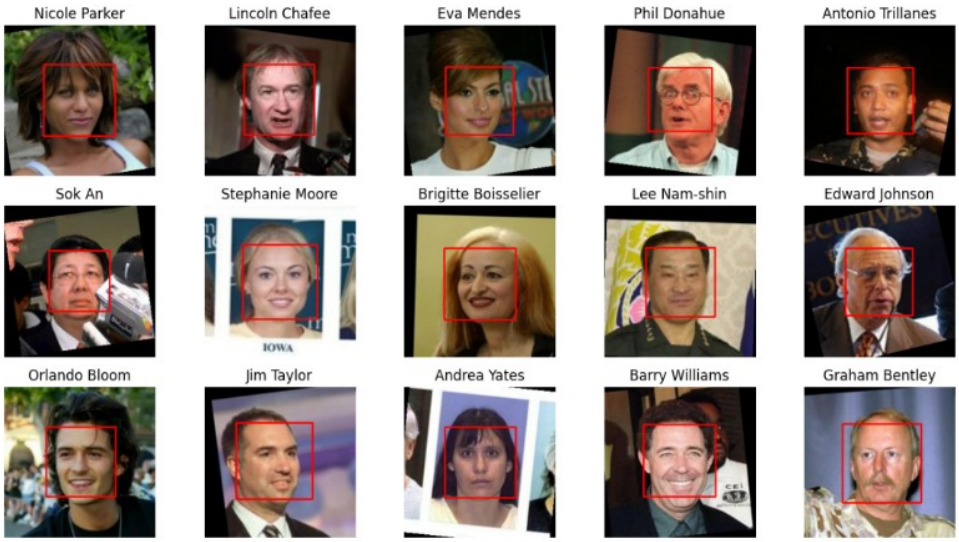


*Рис. 4.17. Ідентифікація людей*

Розглянемо на рис. 4.18 приклад оригінального зображення із датасету.

*Рис. 4.18. Оригінальне зображення*

Поглянемо на варіації штучно доданих зображень після аугментації (див. рис. 4.19).

*Рис. 4.19 Приклади зображень після аугментації*

Як бачимо після застосування аугментації даних кожен клас отримує нові, видозміненні фотографії. Кожне зображення мало множник аугментації 10, тому після штучного збільшення даних кожен клас буде містити по 100 фотографії та відповідно 1000 фотографій загалом для однієї людини.

На рис.4.20 зображено приклад розпізнавання обличчя.

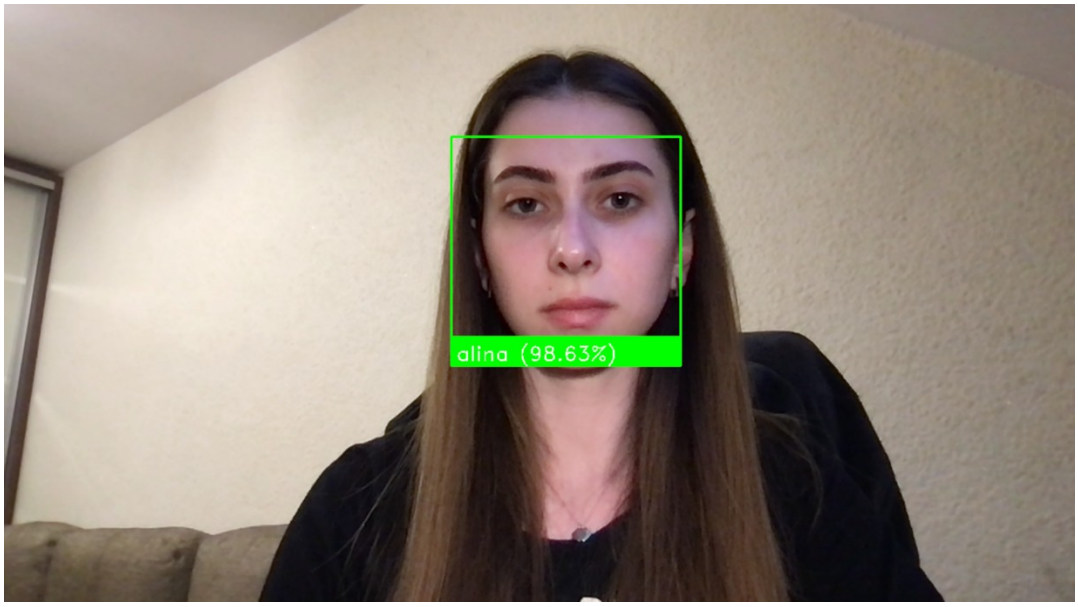


*Рис. 4.20. Приклад розпізнавання обличчя*

Отже, як бачимо, обличчя розпізнано, а також вказано особу. Дівчина була розпізнана правильно з точністю 98.63%. Рамка навколо обличчя зелена, що означає щоб мінімальний поріг точності пройдено.

Розглянемо варіант, коли обличчя розпізнано під кутом (див. рис.4.21.).

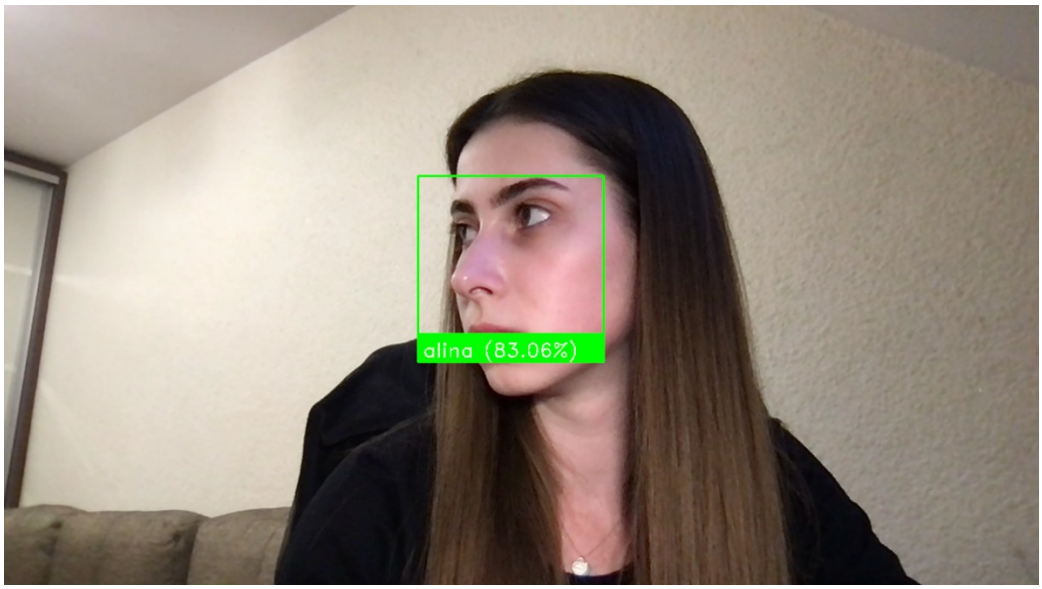


*Рис 4.21. Приклад розпізнавання обличчя під кутом*

Як видно, результати є досить хорошими. З точністю до 83.06% обличчя людини розпізнано вірно. Мінімальний поріг пройдено та рамка навколо обличчя зеленого кольору.

Розглянемо, коли декілька обличь знайдено на камері (див. рис. 4.22).

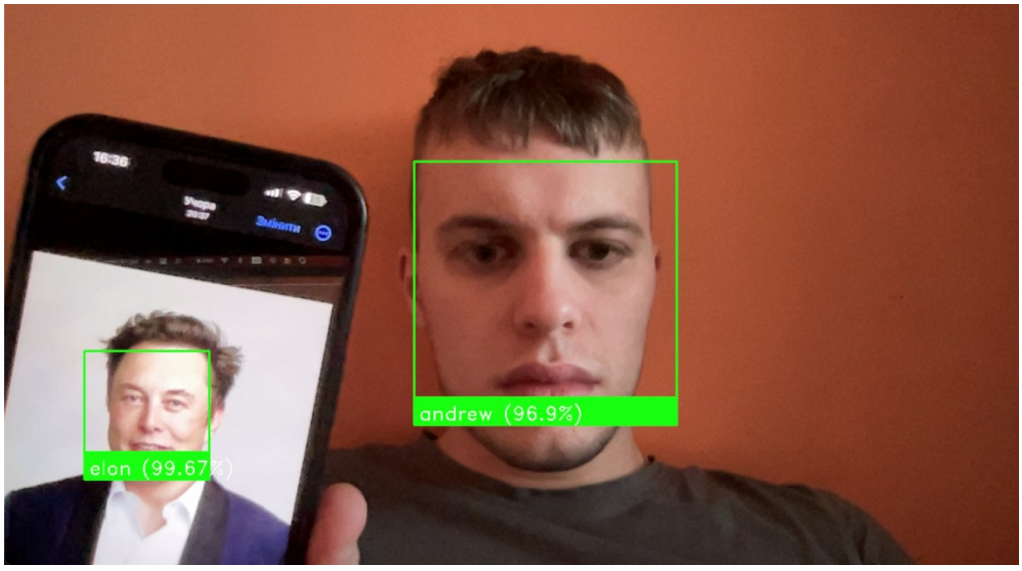


*Рис. 4.22. Приклад розпізнавання декількох обличь*

Система працює добре навіть, коли в кадрі знаходиться більше, ніж два обличчя. Розпізнано обличчя Ілона та Андрія, з точністю до 99.67% та 96.9%, відповідно. Тепер розглянемо ситуацію, коли модель знаходить невідоме обличчя (див. рис. 4.23).

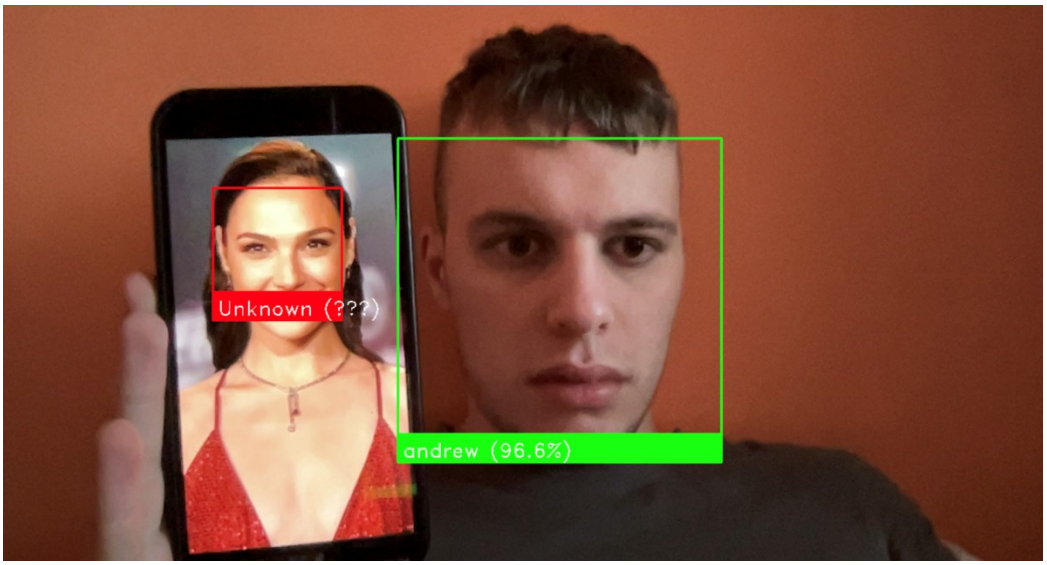


*Рис. 4.23. Приклад розпізнавання невідомого обличчя*

Коли у кадрі з'являється людина, про яку модель нічого не знає, результат – Unknown та обличчя обведене червоною рамкою. Це означає, що для цієї людини доступ заборонений. Проведемо порівняльну характеристику точності та часу роботи моделі для розпізнавання обличчя (табл. 4.1).

*Таблиця 4.16.* Порівняння ефективності моделі

| Зображення | Характеристика середовища | Точність розпізнавання | Час розпізнавання |
|---|---|---|---|
| Рис 4.20 | Одне обличчя, хороше освітлення, людина дивиться чітко в камеру | 98.63% | 0.0051485 |
| Рис 4.21 | Одне обличчя, хороше освітлення, людина дивиться вбік, | 83.06% | 0.0065942 |
| Рис 4.22 | Два обличчя, тьмяне освітлення, людина дивиться в камеру | 99.67% та 96.9% | 0.0069133 |
| Рис 4.23 | Два обличчя, тьмяне освітлення, людина дивиться в камеру | 96.6% та 0 | 0.0071965 |

Модель надійно та точно вміє передбачати людину на потоці кадрів з відео. Це видно з високої точності для різних характеристик середовища. Крім

того модель показує високу ефективність роботи. Вона швидко проводиться передбачення для різного набору обличь на зображення.

У якості покращення цією роботи, можна продовжувати працювати над моделлю, яка буде здатна з ще більшою точністю знаходити обличчя з різних ракурсів та проводити їх порівняння. Також ефективною буде модель, коли вона буде мати здатність розпізнавати обличчя при несприятливих умовах, наприклад, погане освітлення, велика контрастність, присутність шуму чи невдалий ракурс.

Щоб вирішити дану проблему можна застосувати алгоритми попередньої обробки. Наприклад, алгоритми фільтрації для видалення шуму чи алгоритми для покращення світла на фото. Це дозволить з ефективніше розпізнавати обличчя, таким чином, система зможе робити більш точні прогнози.

# ВИСНОВКИ

У кваліфікаційній роботі проведено чисельний аналіз автоматизованої системи розпізнавання облич у контексті правоохоронної діяльності, зокрема, з використанням методів штучного інтелекту – нейронної мережі VGG16. Для тестування було обрано набір даних LFW, який відзначається великою варіативністю, оскільки містить зображення з різними умовами освітлення, якістю та позами людей. Це забезпечує відповідність реальним сценаріям, з якими стикаються правоохоронні органи під час ідентифікації осіб. З метою оптимізації обробки даних та підвищення продуктивності системи розпізнавання було застосовано декілька підходів до розпаралелювання. Використано бібліотеки multiprocessing, Cython з підтримкою OpenMP, MPI (mpi4py) для розподілу задач між процесами, а також CUDA для прискорення обчислень на графічних процесорах. Кожен із цих методів було інтегровано для паралельної обробки шарів нейронної мережі, що дозволило значно скоротити час обчислень.

Результати експериментів засвідчили, що запропонований підхід забезпечує суттєве зменшення часу обробки даних та підвищення точності розпізнавання облич, що є критично важливим для ефективної діяльності правоохоронних органів. Зокрема, вдалося досягти покращення точності до 97%, що перевищує результати [18], де точність становила 85%. Аналіз ефективності технологій розпаралелювання показав, що використання Nvidia CUDA забезпечує прискорення обчислень приблизно у 65 разів, тоді як серед процесорних методів найкращі результати продемонстрував OpenMP, дозволяючи прискорити обробку приблизно у 5 разів. Таким чином, впровадження сучасних методів паралельних обчислень у системи розпізнавання облич значно підвищує їхню ефективність, що є ключовим аспектом для інтегрованих інформаційних систем, орієнтованих на потреби правоохоронної діяльності.

# СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ


1. Калюжний Д. Принципи використання інформаційних технологій правоохоронними органами. *Протидія дезінформації в умовах російської агресії проти України: виклики і перспективи: тези доп. учасників міжн. наук.-практ. конф.* (Анн-Арбор – Харків, 12-13 груд. 2023 р.) , 221-224. https://doi.org/10.32782/PPSS.2023.1.58
2. Li L., Mu X., Li S. and Peng H. A Review of Face Recognition Technology. *IEEE Access*, Vol. 8, 2020, pp. 139110-139120, 2020, doi: 10.1109/ACCESS.2020.3011028.
3. Zamir M., Ali N., Naseem A., Ahmed Frasteen A., Zafar B., Assam M., Othman M., Attia E-A. Face Detection & Recognition from Images & Videos Based on CNN & Raspberry Pi. *Computation.* 2022; 10(9):148. doi:10.3390/computation10090148.
4. Liu F., Kim M., Jain A., & Liu X. Controllable and guided face synthesis for unconstrained face recognition. *In European Conference on Computer Vision.* Cham: Springer Nature Switzerland. 2022, pp. 701-719. Cham: Springer Nature Switzerland.
5. Liu F., Chen D., Wang F., Li Z., & Xu F. Deep learning based single sample face recognition: a survey. *Artificial Intelligence Review*, 2023, 56(3), 2723-2748.
6. Peña A. et al. Facial expressions as a vulnerability in face recognition //*2021 IEEE International Conference on Image Processing (ICIP) – IEEE*, 2021. С. 2988-2992.
7. Панасюк В. І. Розумна система розпізнавання облич на базі штучного інтелекту : магістерська дис. : *121 Інженерія програмного забезпечення*. Київ, 2024. 130 с.

8. Page M. J., McKenzie J. E., Bossuyt P. M., Boutron I., Hoffmann T. C., Mulrow C. D. et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews *BMJ* 2021; 372 :n71 doi:10.1136/bmj.n71
9. Shah A., Ali B., Habib M., Frnda J., Ullah I., Anwar M.S. An ensemble face recognition mechanism based on three-way decisions. *Journal of King Saud University. Computer and Information Sciences*, Volume 35, Issue 4, 2023, pp. 196-208.
10. Nourman S. Irjanto and Nico Surantha. Home Security System with Face Recognition based on Convolutional Neural Network. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 11(11), 2020. http://dx.doi.org/10.14569/IJACSA.2020.0111152.
11. Arkan Mahmood Albayati. One-Shot Learning for Face Recognition Using Deep Learning: A Survey. *Int J Intell Syst Appl Eng*, vol. 12, no. 4, 2024. Pp. 2473.
12. Li Yang & Zheng Wenming & Cui Zhen & Zhang Tong. Face Recognition based on Recurrent Regression Neural Network. *Neurocomputing.* 297. 2018. 10.1016/j.neucom.2018.02.037.
13. Яловега О. В., Мельник Р. А. Система виявлення обличчя на зображенні з використанням глибинної згорткової нейронної мережі. *Scientific Bulletin of UNFU*, 32(2), 2022, 55-60. https://doi.org/10.36930/40320209.
14. Yang S., Xiao W., Zhang M., Guo S., Zhao J., & Shen F. Image data augmentation for deep learning: A survey. *arXiv preprint arXiv:2204.08610*, 2022.
15. What is Overfitting? – Overfitting in Machine Learning Explained – AWS/. [Online]. Available: https://aws.amazon.com/what-is/overfitting/. [Accessed: 14-Трав-2024].
16. Hussain M., Thaher T., Almourad M.B. et al. Optimizing VGG16 deep learning model with enhanced hunger games search for logo classification. *Sci Rep* 14, 31759, 2024. https://doi.org/10.1038/s41598-024-82022-5.

17. Everything you need to know about VGG16 | by Great Learning | Medium/. [Online]. Available: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918. [Accessed: 14-Трав-2024].
18. Kyal C., Poddar H., & Reza M. Human Emotion Recognition from Spontaneous Thermal Image Sequence Using GPU Accelerated Emotion Landmark Localization and Parallel Deep Emotion Net. *Computer Science, Engineering*, 2020, DOI:10.1007/978-981-15-5113-0_78.

# ДОДАТКИ

# Додаток

## Фрагмент програмної реалізації запропонованої у роботі інтегрованої автоматизованої системи на основі методів та технологій штучного інтелекту

```
import warnings
warnings.filterwarnings('ignore')
import os
import tensorflow as ts

num_threads = 8

os.environ["OMP_NUM_THREADS"] = str(num_threads)
os.environ["TF_NUM_INTRAOP_THREADS"] = str(num_threads)
os.environ["TF_NUM_INTEROP_THREADS"] = str(num_threads)

ts.config.threading.set_inter_op_parallelism_threads(num_threads)
ts.config.threading.set_intra_op_parallelism_threads(num_threads)
ts.config.set_soft_device_placement(True)
import os
extract_to_path = 'kaggle/input/lfw-dataset'
os.makedirs(extract_to_path, exist_ok=True)
extracted_files = os.listdir(extract_to_path)
extracted_files
import os
import pandas as pd
import shutil

# Define paths
base_dir = 'kaggle/input/lfw-dataset/lfw-deepfunneled/lfw-deepfunneled/'
metadata_dir = 'kaggle/input/lfw-dataset'
output_dir = "kaggle/working/"
train_dir = os.path.join(output_dir, 'train')
test_dir = os.path.join(output_dir, 'test')

# Ensure the output directories exist
os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

# Load metadata for splits
train_metadata = pd.read_csv(os.path.join(metadata_dir, 'peopleDevTrain.csv'))
test_metadata = pd.read_csv(os.path.join(metadata_dir, 'peopleDevTest.csv'))

# Function to normalize person names
def normalize_name(name):
    return '_'.join(name.replace('_', ' ').split()).strip()

# Function to copy images into training and testing folders
def prepare_dataset(df, img_folder, dest_folder):
    for index, row in df.iterrows():
        person_name = normalize_name(row['name'])
        images_count = int(row['images'])
        person_dir = os.path.join(dest_folder, person_name)  # Folder for the person
        os.makedirs(person_dir, exist_ok=True)
```

```
        # Format the image filename
        for i in range(1, images_count + 1):
            filename = f"{person_name}_{i:04d}.jpg"
            source_path = os.path.join(img_folder, person_name, filename)
            dest_path = os.path.join(person_dir, filename)

            # Check if the source file exists and copy
            if os.path.exists(source_path):
                shutil.copy(source_path, dest_path)
            else:
                print(f"Warning: File not found - {source_path}")

# Copy training images
prepare_dataset(train_metadata, base_dir, train_dir)

# Copy testing images
prepare_dataset(test_metadata, base_dir, test_dir)

# Function to count files in directories
def count_files(directory):
    total_files = 0
    for root, dirs, files in os.walk(directory):
        total_files += len(files)
    return total_files

# Print counts to verify
print(f"Total training images: {count_files(train_dir)}")
print(f"Total testing images: {count_files(test_dir)}")
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define image dimensions and path to directories
img_width, img_height = 224, 224  # Size required by VGG16 model
train_dir = 'kaggle/working/train/'
test_dir = 'kaggle/working/test/'

# Initialize VGG16 base model, pre-trained on ImageNet data
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width,
img_height, 3))

# Freeze the layers of the base model to prevent them from being updated during
training
for layer in base_model.layers:
    layer.trainable = False

# Add new layers on top of VGG16
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)  # Dropout for regularization

# Assuming the number of classes is fetched from the directory structure
num_classes = len(os.listdir(train_dir))  # This should match the number of folders
in train_dir
```

```
output_layer = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output_layer)
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Print model summary to check the final model architecture
model.summary()
# Setup data generators with augmentation settings for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)  # Only rescaling for validation
data

# Create data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=20,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_width, img_height),
    batch_size=20,
    class_mode='categorical'
)
import os

train_dir = 'kaggle/working/train/'
test_dir = 'kaggle/working/test/'

# List all classes from both directories
train_classes = os.listdir(train_dir)
test_classes = os.listdir(test_dir)

# Combine lists and remove duplicates
all_classes = sorted(set(train_classes + test_classes))

print(f"Total classes: {len(all_classes)}")
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_width, img_height = 224, 224  # Size required by VGG16 model

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
```

```
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Use the complete list of classes for both generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=20,
    class_mode='categorical',
    classes=all_classes  # specify all classes explicitly
)

validation_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_width, img_height),
    batch_size=20,
    class_mode='categorical',
    classes=all_classes  # specify all classes explicitly
)
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Load the base VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width,
img_height, 3))

# Freeze all layers in the base model
for layer in base_model.layers:
    layer.trainable = False

# Create new top layers
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
output_layer = Dense(len(all_classes), activation='softmax')(x)  # Use the total
number of unique classes

model = Model(inputs=base_model.input, outputs=output_layer)
model.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy',
metrics=['accuracy'])

# Print the model summary to confirm the setup
model.summary()

# Set training parameters
epochs = 1
batch_size = 20
steps_per_epoch = train_generator.samples // batch_size
validation_steps = validation_generator.samples // batch_size

# Start training the model
history = model.fit(
    train_generator,
```

```
    steps_per_epoch=steps_per_epoch,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_steps
)

# Save the trained model
model.save('kaggle/working/vgg16_face_recognition_model.h5')

# Optionally, print the history of training
print(history.history)

import warnings
warnings.filterwarnings('ignore')
import os
extract_to_path = 'kaggle/input/lfw-dataset'
os.makedirs(extract_to_path, exist_ok=True)
extracted_files = os.listdir(extract_to_path)
extracted_files
import os
import pandas as pd
import shutil

# Define paths
base_dir = 'kaggle/input/lfw-dataset/lfw-deepfunneled/lfw-deepfunneled/'
metadata_dir = 'kaggle/input/lfw-dataset'
output_dir = "kaggle/working/"
train_dir = os.path.join(output_dir, 'train')
test_dir = os.path.join(output_dir, 'test')

# Ensure the output directories exist
os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

# Load metadata for splits
train_metadata = pd.read_csv(os.path.join(metadata_dir, 'peopleDevTrain.csv'))
test_metadata = pd.read_csv(os.path.join(metadata_dir, 'peopleDevTest.csv'))

# Function to normalize person names
def normalize_name(name):
    return '_'.join(name.replace('_', ' ').split()).strip()

# Function to copy images into training and testing folders
def prepare_dataset(df, img_folder, dest_folder):
    for index, row in df.iterrows():
        person_name = normalize_name(row['name'])
        images_count = int(row['images'])
        person_dir = os.path.join(dest_folder, person_name)  # Folder for the person
        os.makedirs(person_dir, exist_ok=True)

        # Format the image filename
        for i in range(1, images_count + 1):
            filename = f"{person_name}_{i:04d}.jpg"
            source_path = os.path.join(img_folder, person_name, filename)
            dest_path = os.path.join(person_dir, filename)

            # Check if the source file exists and copy
            if os.path.exists(source_path):
                shutil.copy(source_path, dest_path)
```

```
            else:
                print(f"Warning: File not found - {source_path}")

# Copy training images
prepare_dataset(train_metadata, base_dir, train_dir)

# Copy testing images
prepare_dataset(test_metadata, base_dir, test_dir)

# Function to count files in directories
def count_files(directory):
    total_files = 0
    for root, dirs, files in os.walk(directory):
        total_files += len(files)
    return total_files

# Print counts to verify
print(f"Total training images: {count_files(train_dir)}")
print(f"Total testing images: {count_files(test_dir)}")
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
with tf.device('/device:GPU:0'):
  import tensorflow as tf
  from tensorflow.keras.applications import VGG16
  from tensorflow.keras.layers import Flatten, Dense, Dropout
  from tensorflow.keras.models import Model
  from tensorflow.keras.optimizers import Adam
  from tensorflow.keras.preprocessing.image import ImageDataGenerator

  # Define image dimensions and path to directories
  img_width, img_height = 224, 224  # Size required by VGG16 model
  train_dir = 'kaggle/working/train/'
  test_dir = 'kaggle/working/test/'

  # Initialize VGG16 base model, pre-trained on ImageNet data
  base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width,
img_height, 3))

  # Freeze the layers of the base model to prevent them from being updated during
training
  for layer in base_model.layers:
      layer.trainable = False

  # Add new layers on top of VGG16
  x = Flatten()(base_model.output)
  x = Dense(512, activation='relu')(x)
  x = Dropout(0.5)(x)  # Dropout for regularization

  # Assuming the number of classes is fetched from the directory structure
  num_classes = len(os.listdir(train_dir))  # This should match the number of folders
in train_dir

  output_layer = Dense(num_classes, activation='softmax')(x)
  model = Model(inputs=base_model.input, outputs=output_layer)
  model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])

  # Setup data generators with augmentation settings for training
  train_datagen = ImageDataGenerator(
```

```
      rescale=1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest'
  )
  test_datagen = ImageDataGenerator(rescale=1./255)  # Only rescaling for validation
data
  # Create data generators
  train_generator = train_datagen.flow_from_directory(
      train_dir,
      target_size=(img_width, img_height),
      batch_size=20,
      class_mode='categorical'
  )

  validation_generator = test_datagen.flow_from_directory(
      test_dir,
      target_size=(img_width, img_height),
      batch_size=20,
      class_mode='categorical'
  )

  import os

  train_dir = 'kaggle/working/train/'
  test_dir = 'kaggle/working/test/'

  # List all classes from both directories
  train_classes = os.listdir(train_dir)
  test_classes = os.listdir(test_dir)

  # Combine lists and remove duplicates
  all_classes = sorted(set(train_classes + test_classes))

  print(f"Total classes: {len(all_classes)}")

  from tensorflow.keras.preprocessing.image import ImageDataGenerator

  img_width, img_height = 224, 224  # Size required by VGG16 model

  train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest'
  )

  test_datagen = ImageDataGenerator(rescale=1./255)

  # Use the complete list of classes for both generators
  train_generator = train_datagen.flow_from_directory(
```

```
      train_dir,
      target_size=(img_width, img_height),
      batch_size=20,
      class_mode='categorical',
      classes=all_classes  # specify all classes explicitly
  )

  validation_generator = test_datagen.flow_from_directory(
      test_dir,
      target_size=(img_width, img_height),
      batch_size=20,
      class_mode='categorical',
      classes=all_classes  # specify all classes explicitly
  )

  from tensorflow.keras.applications import VGG16
  from tensorflow.keras.layers import Flatten, Dense, Dropout
  from tensorflow.keras.models import Model
  from tensorflow.keras.optimizers import Adam

  # Load the base VGG16 model
  base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width,
img_height, 3))

  # Freeze all layers in the base model
  for layer in base_model.layers:
      layer.trainable = False

  # Create new top layers
  x = Flatten()(base_model.output)
  x = Dense(512, activation='relu')(x)
  x = Dropout(0.5)(x)
  output_layer = Dense(len(all_classes), activation='softmax')(x)  # Use the total
number of unique classes

  model = Model(inputs=base_model.input, outputs=output_layer)
  model.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy',
metrics=['accuracy'])

  epochs = 1
  batch_size = 20
  steps_per_epoch = train_generator.samples // batch_size
  validation_steps = validation_generator.samples // batch_size

  # Start training the model
  history = model.fit(
      train_generator,
      steps_per_epoch=steps_per_epoch,
      epochs=epochs,
      validation_data=validation_generator,
      validation_steps=validation_steps
  )

  # Save the trained model
  # model.save('kaggle/working/vgg16_face_recognition_model.h5')

  # Optionally, print the history of training
  print(history.history)
```