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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.4

«До захисту допущено»

В.о. завідувача кафедри

_____ Едуард ЖАРІКОВ

«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: « Гібридний метод обробки зображень на конволюційних нейронних мережах»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІТ-з02мп
Федоряка Максим Григорович _____

Керівник:
доцент кафедри ІПІ, к.т.н.,
Мажара Ольга Олександрівна

_____

Рецензент:
Доцент кафедри АПЕПС, к.т.н.,
Шаповалова Світлана Ігорівна

_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Едуард ЖАРІКОВ

«___»_____2021р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Федоряці Максиму Григоровичу

1. Тема дисертації «Гібридний метод обробки зображень на конволюційних нейронних мережах», науковий керівник дисертації Мажара Ольга Олександрівна, к.т.н, затверджені наказом по університету від «25» жовтня 2021 р. № 3575-с

2. Термін подання студентом дисертації «6» грудня 2021 р.

3. Об’єкт дослідження – програмне забезпечення для обробки зображень

4. Предмет дослідження – алгоритм обробки зображень з гібридним підходом, що пристосований для використання на мобільному пристрої

5. Перелік завдань, які потрібно розробити – провести аналіз існуючих рішень у сфері покращення роздільної здатності зображення, розробити гібридний підхід до покращення роздільної здатності зображень, оптимізований для мобільних пристроїв, проаналізувати ефективність розробленого алгоритму у порівнянні з аналогами.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 5 плакатів

7. Орієнтовний перелік публікацій – одна публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Мелкумян К. Ю., к.т.н., доцент		

9.Дата видачі завдання «30» вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Отримання завдання	30.09.2020	
2	Аналіз існуючих рішень	15.10.2020	
3	Вибір інструментарію для навчання нейронних мереж	11.11.2020	
4	Вибір інструментарію для взаємодії з моделями на боці мобільного додатку	28.11.2020	
5	Проектування нейронної мережі	13.04.2021	
6	Розробка мобільного додатку	27.08.2021	
7	Виконання експериментальних досліджень	30.09.2021	
8	Оформлення пояснювальної записки	01.11.2021	
9	Подання дисертації на попередній захист	22.11.2021	
10	Подання дисертації на захист	06.12.2021	

Студент Максим ФЕДОРЯКА

Науковий керівник Ольга МАЖАРА

РЕФЕРАТ

Актуальність теми дослідження. Сучасний стан та швидкий розвиток інформаційних технологій, зокрема штучного інтелекту, робить повсякденне життя людини більш зручним та оптимізованим. Однією із задач, які вдалося оптимізувати на персональних комп'ютерах та високопродуктивних системах, є покращення якості зображень, зокрема збільшення роздільної здатності. Можливість збільшити роздільну здатність зображення дає розвиток новим класам програмного забезпечення, які у свою чергу покращують досвід користувачів.

Лише років тому обчислювальна потужність смартфонів зросла достатньо, щоб дозволити використання машинного навчання і на мобільних платформах. Разом з цим, розвиток під-процесорів для машинного навчання, якими комплектуються навіть бюджетні смартфони сьогодні, означає що зараз саме час інвестувати ресурси у розвиток штучного інтелекту і машинного навчання на мобільних платформах.

Однак, наразі не запропоновано архітектурних рішень, що дозволяють використати створені нейромережеві методи на мобільних пристроях.

Актуальною є задача розробки архітектур для вирішення задачі покращення роздільної здатності зображень на основі поєднання існуючих методів для використання на мобільних пристроях.

Покращення роздільної здатності фотографій у мобільному додатку є першим кроком до розвитку більш комплексних технологій, таких як майбутній аналог DLSS для мобільних платформ, покращення якості відео та інші.

Метою дослідження розробка архітектурного рішення для створення мобільного програмного застосунку з покращення роздільної здатності зображення з використанням гібридних підходів. Для досягнення мети необхідно виконати наступні завдання:

– провести аналіз особливостей та існуючих рішень для проблеми покращення роздільної здатності зображення для персональних комп'ютерів;

– провести аналіз фреймворків для роботи з машинним навчанням, що доступні на мобільних платформах;

– запропонувати та реалізувати архітектуру для покращення роздільної здатності зображень на мобільних пристроях;

– реалізувати програмне забезпечення, що використовує запропоновану архітектуру та демонструє її переваги та ефективність;

– виконати аналіз результатів роботи створеного програмного забезпечення.

Об'єктом дослідження є програмне забезпечення для обробки зображень.

Предметом дослідження є алгоритм обробки зображень з гібридним підходом, що пристосований для використання на мобільному пристрої.

Наукова новизна одержаних результатів дослідження полягає у тому, що запропоноване архітектурне рішення щодо створення системи покращення роздільної здатності зображень з використанням гібридного методу поєднання нейронних мереж та Unsharp Masking для використання на мобільних пристроях с операційною системою iOS. Реалізований метод є оптимізованим з точки зору швидкодії та використання ресурсів системою, при цьому вихідний результат має високу якість, особливо при збільшенні роздільної здатності у 2 або 3 рази. Практичне значення результатів полягає у тому, що розроблена архітектура може бути використана у великій кількості мобільних застосунків.

Публікації. Федоряка М. Г. Гібридний метод обробки зображень на конволюційних нейронних мережах / М. Г. Федоряка, К. Ю. Мелкумян. // Друковане видання «Адаптивні Системи Автоматичного Управління». Міжвідомчий науково-технічний збірник. – 2021. – №38. – С. 72–76.

МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ОБРОБКА ЗОБРАЖЕНЬ, МОБІЛЬНІ ПРИСТРОЇ, ПОКРАЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ, МАСШТАБУВАННЯ ЗОБРАЖЕНЬ

ABSTRACT

Relevance of the research topic. The current state and rapid development of information technology, including artificial intelligence, makes everyday life more convenient and optimized. One of the tasks that has been optimized on personal computers and high-performance systems is improvement of image quality, including increasing resolution. The ability to increase image resolution allows the development of new classes of software, which in turn improve the user experience.

Only a few years ago the computing power of smartphones has increased enough to allow the use of machine learning on mobile platforms. At the same time, the development of machine learning subprocessors, which even modern budget smartphones are equipped with today, means that now is the time to invest resources in the development of artificial intelligence and machine learning on mobile platforms.

However, currently no architectural solutions have been proposed that allow the use established neural network methods on mobile devices.

The task of developing architectures to solve the problem of improving image resolution based on a combination of existing methods for use on mobile devices is urgent.

Improving the resolution of photos in the mobile application is the first step towards the development of more complex technologies, such as the future analogue of DLSS for mobile platforms, improving video quality and others.

The purpose of the study is to develop an architectural solution for creating a mobile software application to improve image resolution using hybrid approaches.

Following tasks were outlined and implemented to achieve the aforementioned goal:

– analyze the features and existing solutions to the problem of improving the image resolution for personal computers;

– analyze frameworks for working with machine learning, which are available on mobile platforms;

– propose and implement an architecture to improve the resolution of images on mobile devices;

– implement software that uses the proposed architecture and demonstrates its benefits and effectiveness;

– perform analysis of the results of the created software.

The subject of the study is an image processing algorithm with a hybrid approach, which is adapted for use on a mobile device.

The scientific novelty of the results of the study is that architectural solution was proposed to create a system to improve image resolution using a hybrid method of combining neural networks and Unsharp Masking for use on mobile devices with iOS. The implemented method is optimized in terms of performance and resource use of the system, and the output result is of high quality, especially when increasing the resolution by 2 or 3 times. The practical significance of the results is that the developed architecture can be used in a large number of mobile applications.

Publications. Fedoriaka M.G. Hybrid method of image processing on convolutional neural networks / M. G. Fedoriaka, K. Yu. Melkumyan. // "Adaptive Automatic Control Systems". Interdisciplinary science and technology collection. - 2021. - No. 38. - P. 72–76.

MACHINE LEARNING, NEURAL NETWORKS, CONVOLUTIONAL NEURAL NETWORKS, IMAGE PROCESSING, MOBILE DEVICES, SUPER-RESOLUTION IMAGING, IMAGE SCALING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Масштабування зображень — зміна розміру зображення зі збереженням пропорцій. Під масштабуванням може матися на увазі як збільшення (англ. Scaling up), так і зменшення (англ. Scaling down) масштабу зображення. Це поняття широко використовується в комп'ютерній графіці, обробці відео та фото.

Super-resolution imaging — це клас підходів, що використовуються для покращення роздільної здатності зображень, широко застосовуваний у різних областях від загальної обробки зображень до комп’ютерної томографії.

Глибинне навчання (англ. Deep learning) — це сукупність методів машинного навчання (з вчителем, з частковим залучення вчителя, без вчителя та з підкріпленням), що ґрунтується на навчанні ознак з даних.

Глибока нейронна мережа (англ. Deep neural network) — це нейронна мережа з багатьма прихованими шарами.

Генеральні конкурентні мережі (англ. Generative adversarial networks, GANs) — це клас алгоритмів штучного інтелекту, який використовується у навчанні без вчителя, реалізована система двох штучних нейронних вимірів, які змагаються одна з одною в рамках гри з нульовою сумою.

Конволюційна нейронна мережа (англ. Convolutional neural network) — це клас нейронних мереж прямого поширення спеціального виду, що працює на основі фільтрів, які займаються розпізнаванням характеристик певних класів об’єктів на зображенні: прямих ліній, кривих певного типу тощо.

Метод зворотного поширення помилки (англ. Backpropagation) — це ітеративний алгоритм, що є модифікацією методу градієнтного спуску, та використовується з метою мінімізації помилки роботи багатошарової нейронної мережі.

Набір даних, датасет (англ. Dataset) — це оброблена та структурована певним чином інформація, необхідна для навчання нейронної мережі.

Помилка нейронної мережі (англ. Loss) — це значення, що визначається функцією помилки, відповіддю якої є дійсне число, що характеризує якість вирішення певної задачі нейронною мережею.

Розмір батча навчання нейронної мережі (англ. Batch size) — це гіпер-параметр алгоритму навчання нейронної мережі, що визначає кількість прикладів навчальної вибірки, які повинні бути подані на вхід перед оновленням її внутрішніх параметрів.

Штучна нейронна мережа (англ. Artificial neural network) — це обчислювальна система, побудована за принципом організації та функціонування біологічних нейронних мереж, що складається зі штучних нейронів та зв'язків між ними — синапсів. Вона навчається розв'язувати задачу з прикладів: анотованих зображень, оброблених текстів тощо.

Keras — це відкритий фреймворк для глибинного навчання, розроблений з акцентом на швидке експериментування, що є надбудовою над бібліотекою TensorFlow.

Python — це інтерпретована об'єктно-орієнтована мова програмування високого рівня із динамічною типізацією.

TensorFlow — це відкрита бібліотека для машинного навчання. Вона може бути використана для широкого кола задач, але особлива увага приділяється навчанню та використанню глибоких нейронних мереж.

Apple Neural Engine — частина процесорів серії A та M від компанії Apple, що відповідає за прискорення операції з нейронними мережами.

CoreML — це платформа машинного навчання, яка використовується у продуктах Apple (macOS, iOS, watchOS і tvOS) для швидкого прогнозування або висновку з легкою інтеграцією попередньо навчених моделей машинного навчання, що дозволяє виконувати прогнози в режимі реального часу. для зображень або відео на пристрої.

MLKit — це мобільний пакет SDK, який надає досвід машинного навчання Google у додатках Android та iOS у потужному, але простому у використанні пакеті.

Swift — це універсальна, багатопарадигмна, компільована мова програмування, розроблена компанією Apple Inc. та open-source спільнотою.

Нерізке маскування (англ. Unsharp masking) — технологічний прийом обробки фотографічних зображень, що дозволяє досягти підвищення суб'єктивної чіткості за рахунок посилення контрасту дрібних деталей при незміненому загальному контрасті.

Пікове співвідношення сигналу до шуму (англ. peak signal-to-noise ratio) позначається абревіатурою PSNR і є інженерним терміном, що означає співвідношення між максимумом можливого значення сигналу та потужністю шуму, що спотворює значення сигналу. Оскільки більшість сигналів мають широкий динамічний діапазон, PSNR зазвичай вимірюється логарифмічною шкалою в децибелах.

ВСТУП

Сучасний стан та швидкий розвиток інформаційних технологій, зокрема штучного інтелекту, робить повсякденне життя людини більш зручним та оптимізованим. Однією із задач, які вдалося оптимізувати на персональних комп'ютерах та високопродуктивних системах, є покращення якості зображень, зокрема збільшення роздільної здатності. Можливість збільшити роздільну здатність зображення дає розвиток новим класам програмного забезпечення, які у свою чергу покращують досвід користувачів.

Програмна система, що вирішує дану задачу, може бути впроваджена у багатьох видах програмного забезпечення. Найбільш очевидним застосуванням є покращення старих цифрових фотографій і відео, що було знято на низькоякісну камеру. Також, багато сучасних ігор використовують технологію DLSS (Deep Learning Super Sampling), що дозволяє проводити початковий рендер кадру у більш низькій роздільній здатності, а потім за допомогою нейронної мережі підвищувати роздільну здатність до тієї, що необхідна користувачу. Таким чином, знижується навантаження на графічний процесор, що у свою чергу дозволяє або зекономити на придбанні апаратного забезпечення, або грати у кращій якості чи більшій кількості кадрів на секунду.

Лише декілька років тому обчислювальна потужність смартфонів зросла достатньо, щоб дозволити використання машинного навчання і на мобільних платформах. Разом з цим, розвиток під-процесорів для машинного навчання, якими комплектуються навіть бюджетні смартфони сьогодні, означає що зараз саме час інвестувати ресурси у розвиток штучного інтелекту і машинного навчання на мобільних платформах.

Однак, наразі не запропоновано архітектурних рішень, що дозволяють використати створені нейромережеві методи на мобільних пристроях.

Актуальною є задача розробки архітектур для вирішення задачі покращення роздільної здатності зображень на основі поєднання існуючих методів для використання на мобільних пристроях.

Власники мобільних платформ (Apple та Google) створили швидкі та «легкі» інструменти та платформи для машинного навчання, які дозволяють використовувати ці технології навіть на бюджетних смартфонах.

Перші смартфони з можливостями штучного інтелекту з'явилися у 2017 році, це були Huawei Mate 10 та iPhone X. Але використання машинного навчання у вигляді взаємодії з вже готовою моделлю доступно для усіх пристроїв з iOS 11 та вище, тобто навіть для таких старих пристроїв як iPhone 5S, що вийшов у 2013 році.

Покращення роздільної здатності фотографій у мобільному додатку є першим кроком до розвитку більш комплексних технологій, таких як майбутній аналог DLSS для мобільних платформ, покращення якості відео та інші.

Таким чином, актуальним є створення системи для покращення роздільної здатності зображень, що буде працювати на мобільному пристрої з використанням під-процесорів для задач штучного інтелекту, які вбудовано в сучасні смартфони.

Мета і завдання дослідження полягають у розробці архітектурного рішення для створення мобільного програмного застосунку з покращення роздільної здатності зображення з використанням гібридних підходів, та створенні застосунку для апробації запропонованої архітектури.

Для досягнення мети необхідно виконати наступні завдання:

– провести аналіз особливостей та існуючих рішень для проблеми покращення роздільної здатності зображення для персональних комп'ютерів;

– провести аналіз фреймворків для роботи з машинним навчанням, що доступні на мобільних платформах;

– запропонувати та реалізувати архітектуру для покращення роздільної здатності зображень на мобільних пристроях;

– реалізувати програмне забезпечення, що використовує запропоновану архітектуру та демонструє її переваги та ефективність;

– виконати аналіз результатів роботи створеного програмного забезпечення.

Об'єктом дослідження є програмне забезпечення для обробки зображень.

Предметом дослідження є алгоритм обробки зображень з гібридним підходом, що пристосований для використання на мобільному пристрої.

Наукова новизна одержаних результатів дослідження полягає у тому, що запропоноване архітектурне рішення щодо створення системи покращення роздільної здатності зображень з використанням гібридного методу поєднання нейронних мереж та Unsharp Masking для використання на мобільних пристроях с операційною системою iOS. Реалізований метод є оптимізованим з точки зору швидкодії та використання ресурсів системою, при цьому вихідний результат має високу якість, особливо при збільшенні роздільної здатності у 2 або 3 рази. Практичне значення результатів полягає у тому, що розроблена архітектура може бути використана у великій кількості мобільних застосунків.

Матеріали роботи опубліковані в 38 номері (2021 рік) друкованого видання «Адаптивні Системи Автоматичного Управління». [1]

1 АНАЛІЗ ПРОБЛЕМИ ПОКРАЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕННЯ

1.1 Огляд проблеми покращення роздільної здатності зображення

Анна Манджарес у своїй дисертації на тему збільшення роздільної здатності дуже чітко окреслює проблему та основні частини, яким потрібно приділити увагу для вирішення задачі якісного масштабування зображень. [2]

Масштабування (збільшення або зменшення роздільної здатності) зображень — це окремий випадок більш загальної операції зображення, відомої як викривлення зображення. Викривлення зображення охоплює будь -який процес просторового перетворення зображення шляхом переміщення своїх пікселів відповідно до заданої функції відображення переміщення пікселів *F(m,n),* яка приймає координати пікселів *(m, n)* як вхідні дані та подає координати пікселів призначення *(u, v),* куди потрібно перемістити вхідний піксель (колір).

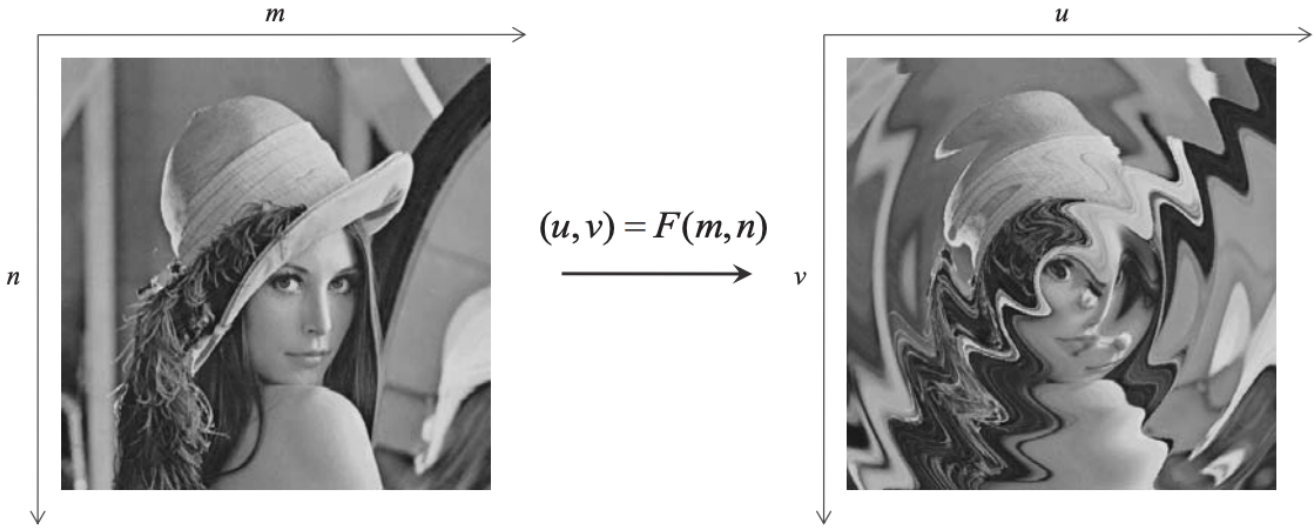

Рисунок 1.1 – Ілюстрація операції викривлення зображення

Найпоширеніші приклади викривлення зображення — це обертання, масштабування та обрізання серед багатьох інших. На Рис. 1.2 показані приклади цих типових операцій викривлення зображення.

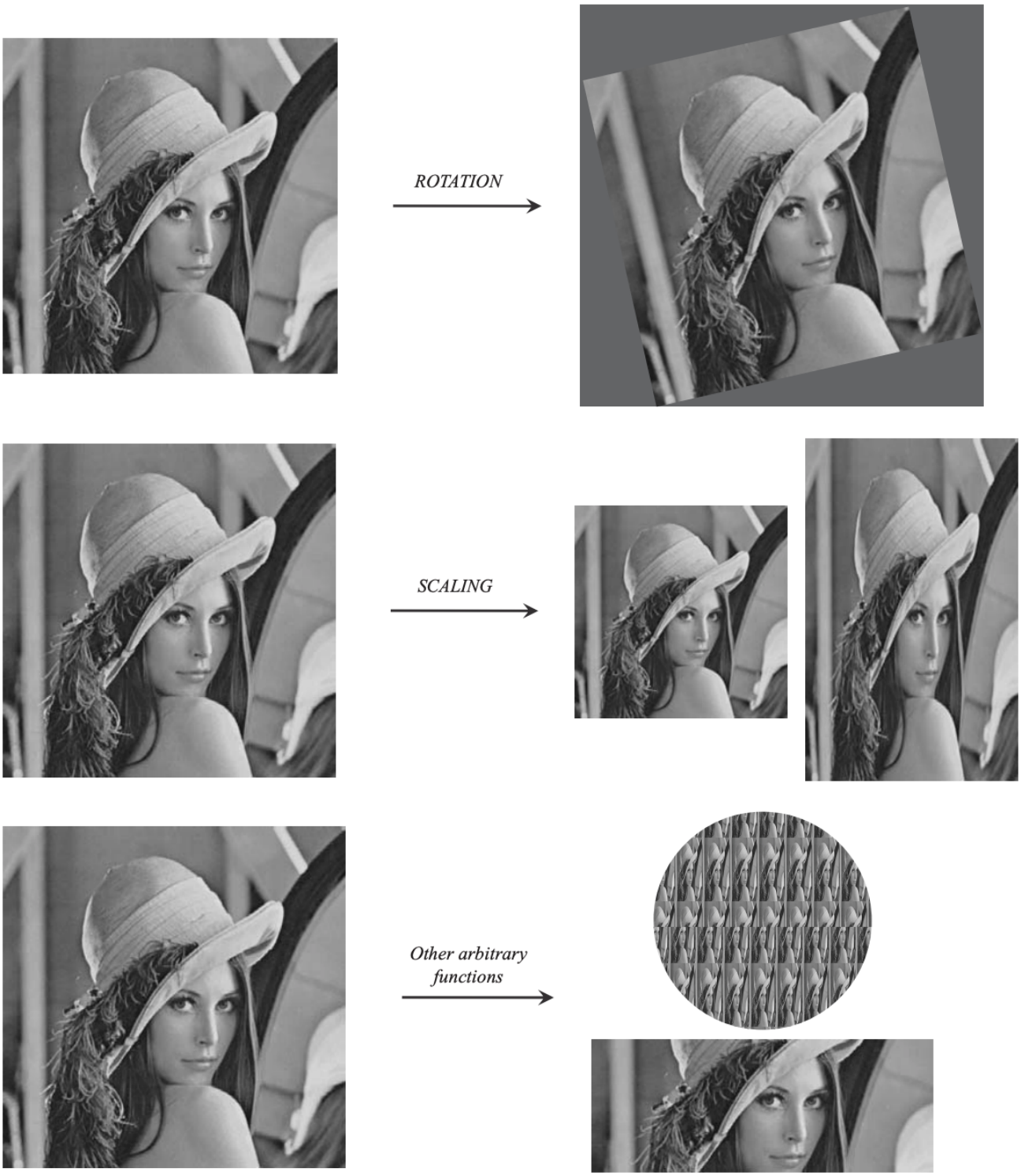

Рисунок 1.2 – Типові операції викривлення зображення

Будь -який тип викривлення зображення зазвичай передбачає дві основні та чітко диференційовані операції, а саме відображення пікселів та призначення кольору, які будуть детально пояснені далі.

1.1.1 Відображення пікселів

Як зазначається у публікації на тему методів викривлення зображення [3], операція відображення пікселів полягає в переміщенні пікселів. Асоціація визначається функцією переміщення пікселів F між місцями пікселів у вхідному зображенні (джерело) та місцями пікселів у вихідному зображенні (призначення), куди їх потрібно перемістити. Це не що інше, як просторове перетворення координат пікселя. Рух пікселів у вхідному зображенні визначає формування вихідного зображення.

Загалом, функція перетворення F може бути будь -якою функцією дійсних чисел, але на практиці її можна звести до випадку позитивних цілочисельних вхідних точок *(m,n)* за умови, що пікселі зображення вважаються завжди розподіленими у звичайній сітці з цілочисельними кроками, що починається від (0,0) у верхньому лівому куті зображення. Тому на практиці F стає функцією з натуральними вхідними та дійсними вихідними параметрами.

Існує два різні підходи до реалізації операції відображення пікселів, що відрізняються напрямком, у якому застосовується функція переміщення: пряме і зворотнє відображення пікселів.

У разі прямого відображення пікселів усі пікселі у вхідному зображенні скануються, і для кожного з них його місце призначення *(u,v),* куди піксель потрібно перемістити у вихідному зображенні, обчислюється шляхом застосування функції переміщення пікселів F у вихідне місце розташування *(m,n)*. Зрештою, всі пікселі з вхідного зображення відображаються на місцях у вихідному зображенні.

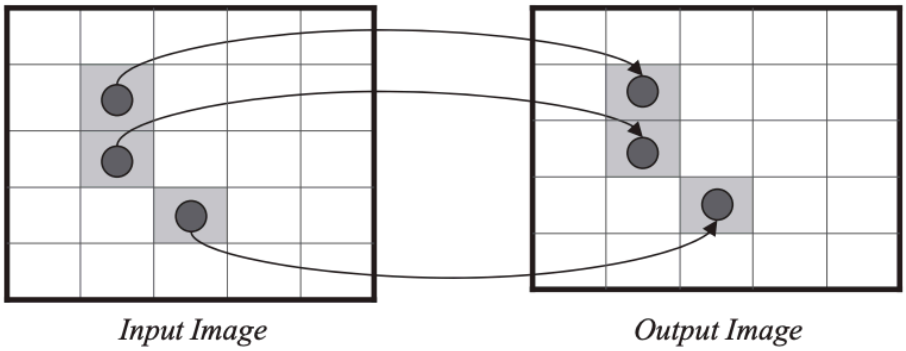

Рисунок 1.3 – Пряме відображення пікселів

З іншого боку, у разі зворотне відображення пікселів усі пікселі у вихідному зображенні відскановуються, і для кожного з них його місцезнаходження у вхідному зображені *(m,n),* обчислюється функцією зворотного переміщення пікселів F^{-1} у вихідне місце *(u,v),* тобто $(m,n) = F^{-1}(u, v)$.

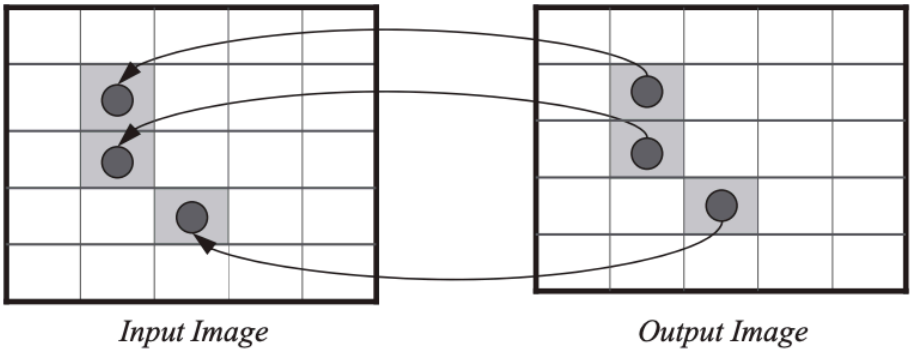

Рисунок 1.4 – Зворотне відображення пікселів

1.1.2 Призначення кольору

Друга операція, що бере участь у викривленні зображення, - це призначення кольору. Як зазначалося вище, для обох підходів до відображення пікселів результат функції відображення, прямої чи зворотної, належить дійсним числам. Але цифрові зображення містять лише пікселі в цілочисельних координатах, наданих їх базовою сіткою, так що ми змушені якимось чином перетворювати нецілі координати, задані процесом відображення пікселів, у цілі координати, дійсні для переміщення колірних значень пікселів. від вхідного зображення до вихідного зображення.

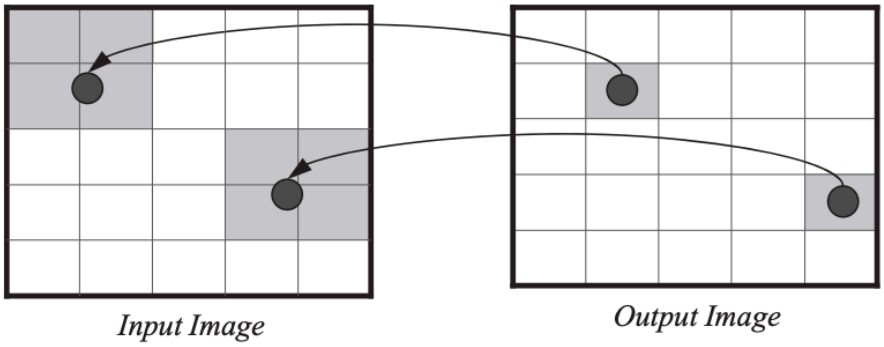

Рисунок 1.5 – Проблема призначення кольору

Цю проблему можна розглядати як проблему передискретизації зображення. Для випадку зворотного відображення пікселів ми зацікавлені в обчисленні оцінки значення кольору, яке було б захоплено кольоровим датчиком, розташованим у даній дійсній (не цілочисельній) точці у вхідній сітці зображення. Цей тип проблем зазвичай вирішується за допомогою методів рівномірної інтерполяції, які обчислюють оцінку кольору на основі значень навколишнього кольору у вхідній сітці зображення.

З іншого боку, для випадку прямого відображення пікселів необхідні більш складні методи нерівномірної інтерполяції для обчислення оцінок значень кольору для всіх цілочисельних розташувань у викривленому зображенні на основі наявних значень кольору в нерівномірній сітці, утвореній прямим відображення дійних розташувань. Тоді пряме відображення пікселів показує явний недолік у порівнянні зворотнім відображенням, через більшу складність, пов'язану з нерівномірною інтерполяцією.

Ця робота зосереджується на конкретному випадку викривленні зображення при збільшенні роздільної здатності зображення. З огляду на вхідне зображення з початковим розміром M x N, метою збільшення масштабу зображення є створення вихідного зображення з новим розміром U x V з $U > M$ і $V > N$, відповідно змінивши розмір вмісту вихідного зображення.

Якщо в цілому масштабування зображення може працювати з різними коефіцієнтами зміни розміру як для горизонтальних, так і для вертикальних розмірів,

у цій роботі для простоти розглядається лише випадок однорідного масштабування, коли обидва розміри змінюються за допомогою одного і того ж коефіцієнта K. Проте зауважте, що методи та результати, показані у цьому дослідженні, можна легко поширити на загальний випадок неоднорідного масштабування.

1.2 Аналіз рішень покращення роздільної здатності зображення

Темою даного дослідження є застосування алгоритмів покращення роздільної здатності зображення на мобільних пристроях, але, нажаль, наукових робіт про цей специфічний варіант використання знайдено не було, що підкреслює актуальність роботи. Уся наукова робота, що вдалося знайти, була проведена над алгоритмами, що застосовуються у персональних комп'ютерах, тому саме вони будуть розглянуті як приклади існуючих рішень.

Найбільш відомим та успішним комерційним мобільним додатком зі схожою функціональністю є Adobe Lightroom CC. Результати роботи функції апскейлінгу є задовільними, але далекими від результатів, що показують алгоритми, застосовувані на персональних комп'ютерах. Також, алгоритми цього додатку є пропрієтарними, тому неможливо дізнатися їх особливості та навіть категорію, до якої вони входять.

Оскільки у роботі пропонується гібридний підхід до обробки зображення, у цьому розділі будуть розглянуті як традиційні алгоритми, так і підходи з використанням нейронних мереж.

1.2.1 Інтерполяція методом найближчого сусіда

Найбільш широко використовуваними алгоритмами інтерполяції є найближча інтерполяція та білінійна інтерполяція. В обох з них достатньо розглянути чотири найближчих пікселя до потрібної нецілої позиції, щоб обчислити нове значення кольору.

У разі інтерполяції методом найближчого сусіда, значення кольору нецілочислового розташування пікселів *(m,n),* у вихідному зображенні, яке потрібно

призначити цілочисельному розташуванню пікселів uv, у збільшеному зображенні, безпосередньо оцінюється як значення кольору найближчого цілого пікселя у вихідному зображенні.

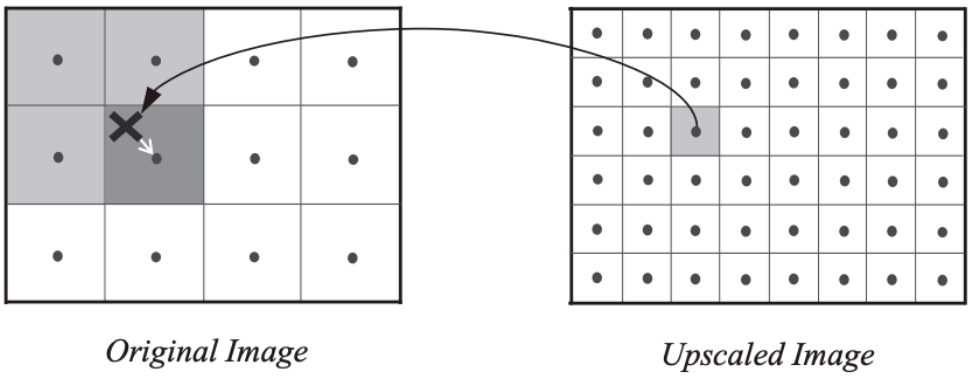

Рисунок 1.6 – Інтерполяція методом найближчого сусіда

Рисунок 1.6 показує приклад цього процесу, де чотири сусіди, які розглядалися спочатку, мають світло-сірий колір, а найближчий сусід - темно-сірий.

Інтерполяція методом найближчого сусіда - це дуже легкий і простий підхід до інтерполяції, дуже швидкий, але на нього сильно впливає згладжування частоти і створює небажані візуальні ефекти, такі як блокування та артефакти викривлення, які значно спотворюють лінії та краї зображення. Це добре видно на Рисунку 1.7.

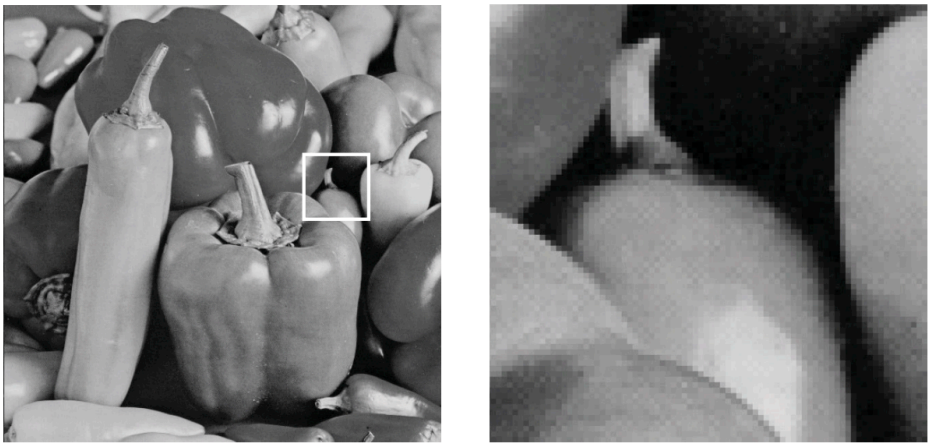

Рисунок 1.7 – Результат інтерполяції методом найближчого сусіда

1.2.2 Білінійна інтерполяція

У випадку білінійної інтерполяції, значення кольору нецілочислового розташування пікселів *(m,n),* у вихідному зображенні, яке потрібно призначити цілочисельному розташуванню пікселів *(u,v),* на збільшеному зображенні, оцінюється як лінійна комбінація чотирьох найближчих пікселів з цілочисельними координатами у вихідному зображенні з внесками (вагами), обернено пропорційними їх відстаням від *(m,n).* Іншими словами, інтерпольоване значення кольору є зваженим додаванням, залежно від відстані, значень кольору чотирьох найближчих сусідніх цілих пікселів.

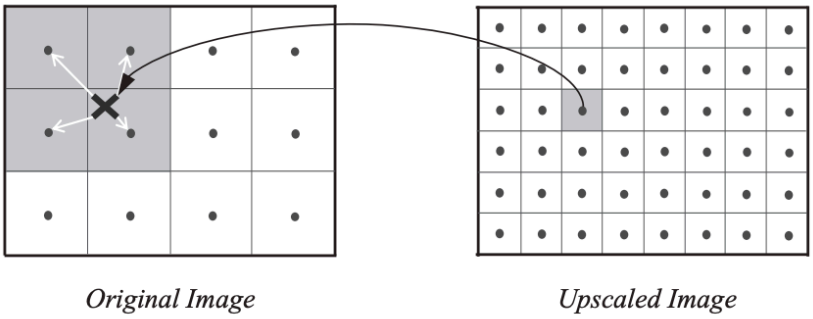

Рисунок 1.8 – Білінійна інтерполяція

Приклад збільшення роздільної здатності зображення за допомогою білінійної інтерполяції показаний на Рисунку 1.9, разом з методом найближчого сусіда для цілей порівняння.

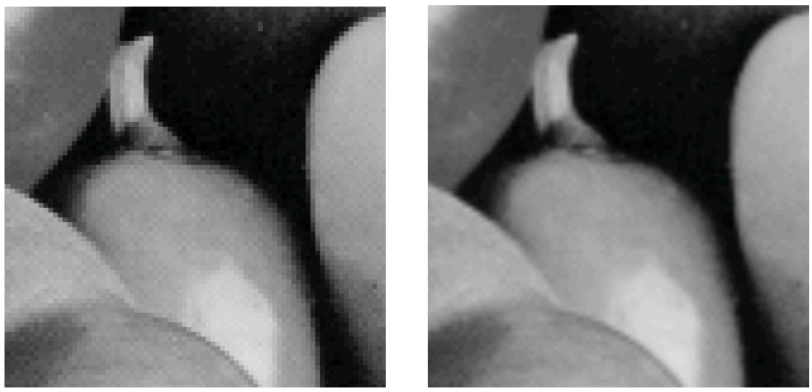

Рисунок 1.9 – Білінійна інтерполяція (справа), метод найближчого сусіда (зліва)

Хоча білінійна інтерполяція є більш обчислювально складною, ніж інтерполяція методом найближчого сусіда, вона дійсно усуває артефакти, отримані останнім підходом, уникаючи, до певної точки, псевдозміщення частоти. Але це робиться за рахунок розмиття деталей та країв, наявних у вихідному зображенні.

1.2.3 Нерізке маскування

Як сказано у огляді алгоритмів нерізкого маскування [3], покращення різкості зображення передбачає додавання до вихідного зображення сигналу, пропорційного фільтру високої частоти версії оригінального зображення. Критичним фактором тут є можливість операції фільтрації високих частот. У традиційному підході для реалізації високочастотного фільтра використовувалися лінійні фільтри. У разі пошкодження вихідного зображення шумом, лінійні підходи можуть дати нам несподівані та неналежні результати.

Рис. 10 ілюструє процедуру, яка називається нерізким маскуванням одновимірного сигналу. Як показано на Рис. 1.10, вихідне зображення спочатку фільтрується високочастотним фільтром, який видаляє високочастотні компоненти, а потім до вихідного зображення додається масштабована версія виходу фільтра високих частот, а отже, буде отримано чітку версію оригінального зображення.

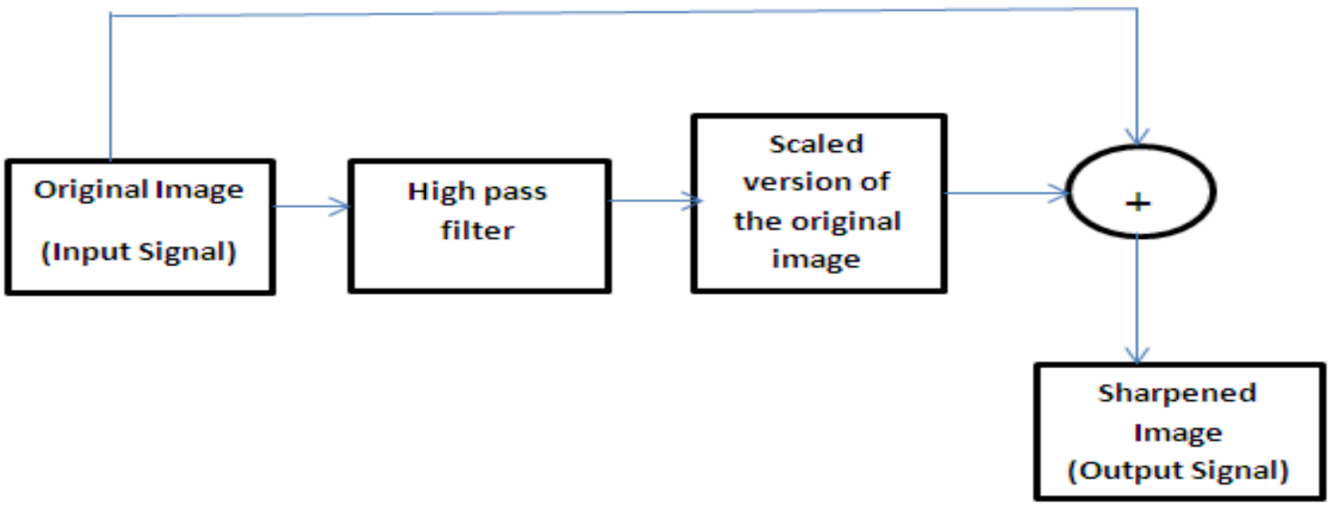

Рисунок 1.10 – Процедура нерізкого маскування

Найбільш розповсюдженим з цих алгоритмів є алгоритм підвищення різкості зображення з використанням звичайного нерізкого маскування (Image Sharpening Algorithm Using Unsharp Masking). Підвищення різкості зображення виконується за допомогою нерізкого маскування для підвищення контрастності. Основна концепція нерізкого маскування - спочатку розмити вихідне зображення, а потім відняти розмите зображення з вихідного зображення і в якості заключного етапу додати відмінності до вихідного зображення. Підхід лінійної нерізкої фільтрації використовується для поліпшення зашумленого зображення за допомогою фільтра високих частот. Маски нерізкості дуже підходять для підвищення різкості зображень. Але занадто велике підвищення різкості також може привести до того, що штучність зображення втратить свій природний вигляд. Цей метод має два основних недоліки, наприклад, контраст в темнішій області посилюється набагато глибше, ніж в найбільш світлій області. Наступна проблема полягає в тому, що метод також підсилює ефекти шуму і оцифровки. Через ці проблем зображення в більшості випадків втрачає свою оригінальність.

Інша модифікація цього алгоритму використовує буфер глибини. Цей підхід застосовується для поліпшення характеристик сприйняття зображень, що містять інформацію про глибину. Як і в разі UM (Unsharp Masking), різниця між вихідним вмістом буфера глибини і копією, відфільтрованої нижніми частотами, використовується для визначення інформації про просторово важливі області сцени. Залежно від цих даних поліпшуються контраст, колір та інші фактори зображення. Мета цього підходу - поліпшити сприйняття складних сцен за рахунок введення додаткових сигналів глибини. Це дозволяє використовувати даний алгоритм для різних сцен, від складних пейзажів до документів та масової обробки фото та відео.

Ще більше покращити якість вихідного результату дозволяє використання вейвлет-перетворення. Основна ідея підвищення різкості зображення полягає в тому, щоб додати до вхідного сигналу версію самого вхідного сигналу з фільтром верхніх частот. Вейвлет-коефіцієнти забезпечують високочастотні деталі зображення з декількома роздільними здатностями. Використовуючи цю концепцію, в [4] для підвищення різкості зображення використовувався підхід, заснований на вейвлетах.

Підвищення різкості зображення вводиться для поліпшення контрастності і яскравості зображення. В іншому методі використовується комбінація як дискретного вейвлет-перетворення (HAAR), так і техніки нерізкого маскування [5]. Вейвлет-коефіцієнти забезпечують високочастотні коефіцієнти зображення, наприклад інформацію про границі. Для підвищення різкості зображення тут використовується техніка нерізкого маскування. Цей алгоритм використовує кореляцію між різними вейвлет-коефіцієнтами, і високочастотні коефіцієнти вважаються краями зображення.

Одна з модифікацій алгоритму інкорпорує техніку, що поєднує збільшення різкості та зменшення шуму. Існують різні причини використання нечіткої логіки в обробці зображення, які полягають у наступному: по -перше, існують різні умови, які зображення має проходити, що включає такі проблеми, як проекція зображень у різних розмірах , оцифрування зображень. По-друге, двозначність, пов'язана з межами та неоднорідними областями, та небагато описів, таких як ребра, посилення контрасту, є дуже поширеними факторами нечіткості. По -третє, обробка зображення з недоліками в даних також є проблемою, і лише за умови достатнього розуміння ситуації проблему можна вирішити. Підхід, пояснений у [6], зосереджений на посиленні контрасту. Він базується на системі множинного виведення, яка приймає нечіткі моделі, щоб уникнути збільшення шуму під час різкості деталей зображення. Вважається, що це забезпечує кращу продуктивність з огляду на інші реалізовані підходи. У цьому підході немає складного налаштування параметрів нечіткого набору. Також загальна нелінійна поведінка системи розширення дуже легко контролюється лише одним параметром. Нечіткі мережі можуть ефективно моделювати конфліктні завдання, такі як видалення шуму та виділення країв. Поєднати нечіткі мережі в одній архітектурі обробки також дуже легко. У цьому підході система вдосконалення включає три мережі, що працюють на різних підмножинах вхідних даних. Однак не потрібна складна настройка нечітких параметрів, оскільки загальна нелінійна поведінка дуже легко контролюється лише одним параметром. Результати комп’ютерного моделювання показали, що

запропонований метод перевершує найсучасніші методи у покращенні шумових даних зображення.

Раціональна техніка нерізкого маскування. Лінійний підхід маскування без різкості, який використовується для покращення зображення, тут модифіковано з невеликими змінами. Контрольний термін виражається як раціональна функція локальних вхідних даних у цьому підході, що є основним параметром, що використовується для покращення зображення. Підсилення шуму або чутливість до шуму, як згадувалося раніше, як недолік нерізкого маскування, яке виникає під час різкості зображення, можна усунути в результаті цього параметра. Також ефект перекриття на гострих краях можна зменшити в кінці покращення зображення.

Таблиця 1.1 – Порівняння алгоритмів нерізкого маскування

Алгоритм	**Підхід**	**Результати**	**Переваги**
Підвищення різкості зображення з використанням звичайного нерізкого маскування	Лінійне нерізке маскування	Дуже хороша дисперсія фону, досягнута за рахунок меншого ефекту різкості	Можливість врахування реакції зорової системи людини Можливість покращити зображення навіть за наявності нелінійного підходу з шумом
Адаптивне нерізке маскування	Фільтр контролює внесок траєкторії різкості таким чином, що посилення контрасту відбувається в області високої деталізації	Добре покращення різкості в умовах низького контрасту	Покращення деталей середнього контрасту. Не може бути використано для онлайн-застосунків

Продовження таблиці 1.1

Алгоритм	**Підхід**	**Результати**	**Переваги**
Вейвлет-трансформації	Інформація про краї об’єктів, отримана з вейвлет коефіцієнтів	7% покращення параметрів в оригіналі і 30% покращення після застосування підвищення різкості	Покращення контрасту та яскравості
Раціональне нерізке маскування	Використовується контрольний термін, представлений як раціональна функція локальних вхідних даних	Забезпечує хорошу нечутливу до шуму методику покращення різкості без штучних артефактів перевищення яскравості	Уникається посилення шуму, а ефекти перекриття на гострих краях обмежені
Техніка підвищення різкості та зменшення шуму	Основна увага приділяється системі множинного виведення, яка приймає нечіткі моделі, щоб уникнути збільшення шуму під час покращення різкості	Загальна нелінійна поведінка контролюється одним параметром нечіткого набору	Краща продуктивність порівняно з іншими методами поліпшення, що застосовуються у зображеннях, що зазнають гауссового шуму, і без складного налаштування параметрів нечіткого набору. Використовується в мобільних додатках

Таким чином, раціональне нерізке маскування є оптимальним з цих алгоритмів для використання у мобільних додатках через простоту обчислення та один з найкращих результатів.

1.2.4 Super-Resolution з використанням GAN

GAN забезпечують потужну основу для створення правдоподібних природних зображень з високою якістю сприйняття. Процедура GAN заохочує реконструкції рухатися до областей простору пошуку з високою ймовірністю містити фотореалістичні зображення, а отже, ближче до різноманітності природного зображення. У роботі [8] описано першу дуже глибоку архітектуру ResNet, використовуючи концепцію GAN для формування функції втрат для фотореалістичного зображення високої роздільної здатності.

Основні досягнення цієї роботи:

– встановили новий рівень техніки для зображення SR з високими коефіцієнтами масштабування (4×), виміряними за допомогою PSNR та структурної подібності (SSIM), з 16 блоками глибокої ResNet (SRResNet), оптимізованою для середньої квадратичної похибки;

– пропонується SRGAN-мережа на базі GAN, оптимізовану для нових втрат сприйняття;

– підтверджено розширеним тестом середньої оцінки думок (MOS) на зображеннях із трьох загальнодоступних наборів даних, що SRGAN є новим сучасним рівнем техніки з великим відривом для оцінки фотореалістичних зображень SR з високими коефіцієнтами збільшення (4×).

Вчені заявляють, що неправильно поставлений характер невизначеної проблеми SR особливо яскраво виражений для високих коефіцієнтів масштабування, для яких деталі текстури в реконструйованих зображеннях SR зазвичай відсутні. Метою оптимізації керованих алгоритмів SR зазвичай є мінімізація середньоквадратичної помилки (MSE) між відновленим зображенням високої роздільної здатності та референсним зображенням.

Також у роботі згадується, що глибокі мережеві архітектури можуть бути важкими для навчання у випадку згорткових нейронних мереж, але мають потенціал істотно підвищити точність мережі, оскільки вони дозволяють моделювати відображення дуже високої складності. Для ефективного навчання цих глибших мережевих архітектур пакетна нормалізація часто використовується для протидії внутрішньому коваріантному зсуву. Саме тому вони вибрали GAN за основу, її набагато легше навчати у випадку більшої глибини мережі.

Архітектура рішення складається з мережі-генератора та дискримінатора, що відповідає за відрізнення оригінальних фотографій високої роздільної здатності від згенерованого виходу мережі-генератора, що дозволяє покращити передачу текстури при подальшому навчанні мережі-генератора.

Повна архітектура мереж продемонстрована на Рисунку 1.11.

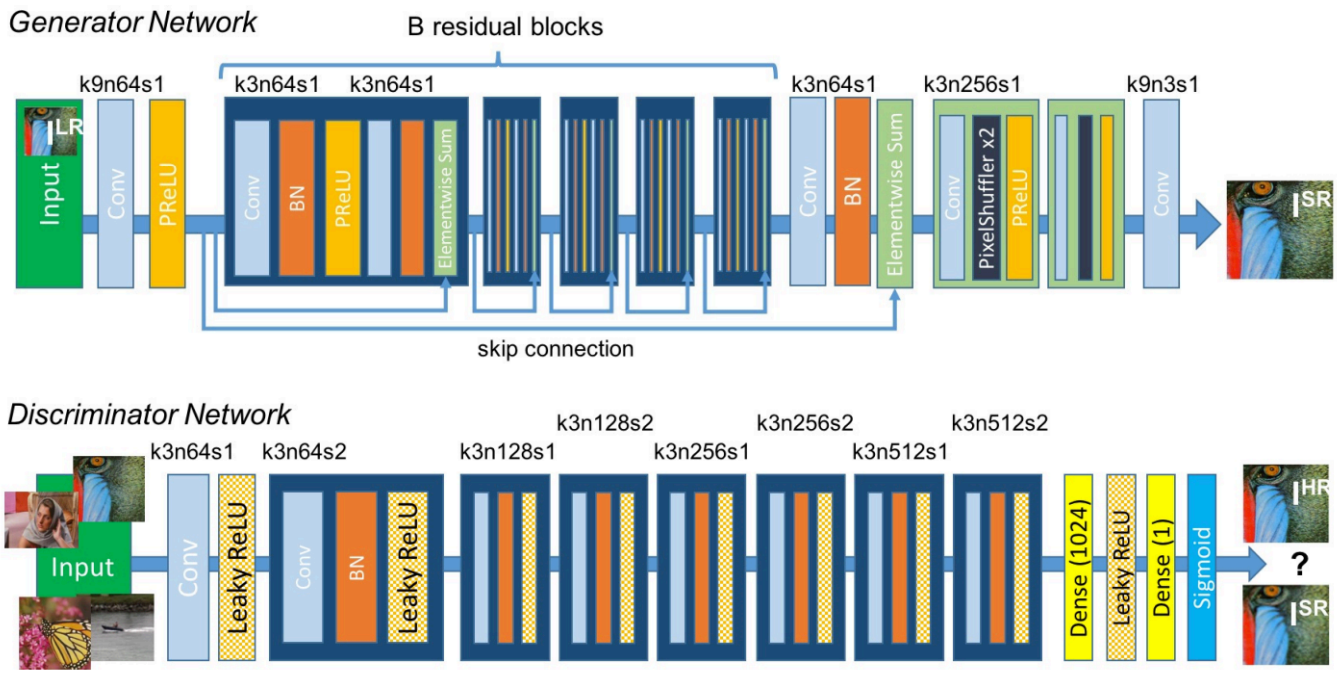

Рисунок 1.11 – Архітектура SRGAN

У центрі уваги цієї роботи була якість сприйняття SR зображень, а не ефективність обчислень. Представлена модель, не оптимізована для обробки відео у режимі реального часу. Однак попередні експерименти щодо мережевої архітектури

свідчать про те, що більш дрібні мережі мають потенціал забезпечити дуже ефективні альтернативи при невеликому зниженні якісної продуктивності.

На Рисунку 1.12 продемонстровано результати роботи SRGAN (у різних варіаціях) порівняно з іншими архітектурами нейронних мереж для SR.

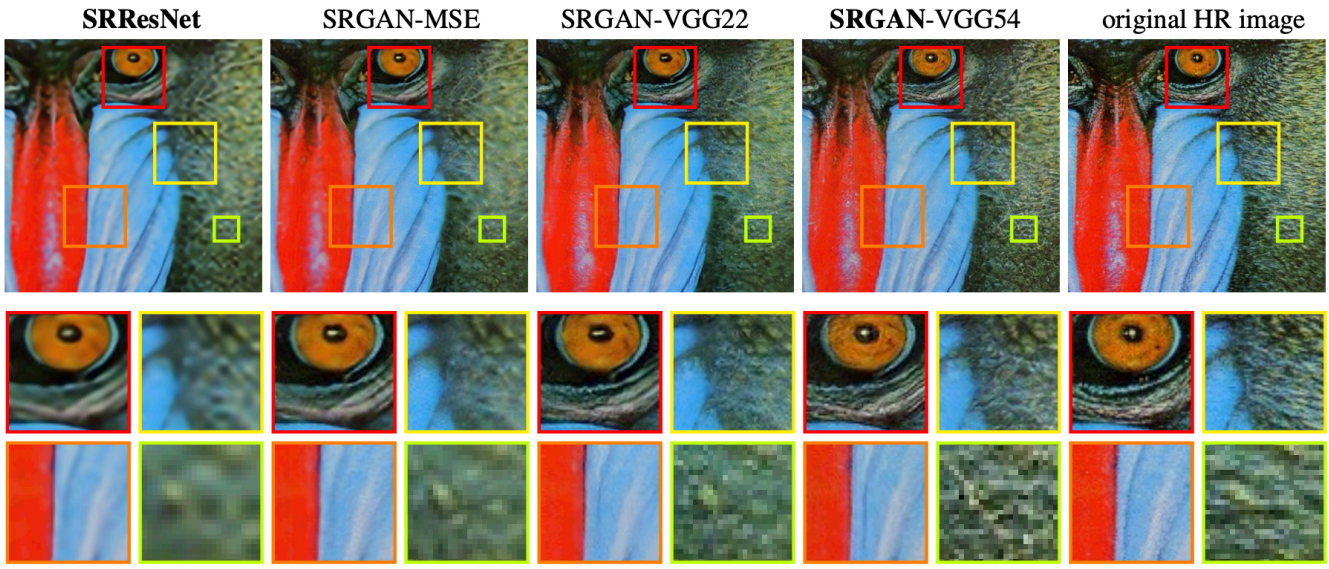

Рисунок 1.12 – Результати роботи SRGAN

Переваги:

– надзвичайна якість вихідних зображень;

– відносна простота тренування щодо результатів.

Недоліки:

– складна архітектура з двома мережами (генератор та дискримінатор);

– немає оптимізації швидкодії, що робить її не оптимальною у цьому аспекті;

– недостатня гнучкість як і усіх GAN, що застосовуються для цієї задачі.

1.2.5 Мережа визначення значущості другого порядку (Second-order Attention Network, SAN)

Автори [9] зазначають, що останнім часом глибокі згорткові нейронні мережі (CNN) широко досліджуються в контексті покращення роздільної здатності єдиного зображення (SISR) і досягають чудової продуктивності. Однак більшість існуючих методів SISR на базі CNN переважно зосереджені на ширшому або глибшому архітектурному дизайні, нехтуючи вивченням кореляцій особливостей проміжних шарів, що перешкоджає репрезентативним можливостям CNN. Щоб вирішити цю проблему, у цій статті вони пропонують мережу визначення значущості другого порядку (SAN) для більш потужного вираження функцій та навчання кореляції функцій. Зокрема, розроблений новий модуль визначення значущості другого порядку (SOCA) для адаптивного масштабування функцій каналу за допомогою статистики функцій другого порядку для більш дискримінаційних уявлень. Крім того, вони представляють нелокально розширену структуру залишкової групи (NLRG), яка не тільки включає нелокальні операції для збору просторової контекстуальної інформації, але також містить повторювані групи залишкової уваги з локальних джерел (LSRAG) для вивчення дедалі більш абстрактної інформації уявлення про особливості. Експериментальні результати демонструють перевагу цієї мережі SAN над найсучаснішими методами SISR з точки зору кількісних показників та візуальної якості.

Нелокально розширена залишкова група (NLRG) складається з кількох нелокальних модулів на рівні регіону (RL-NL) та однієї структури залишкової групи з джерелом спільного доступу (SSRG). RL-NL використовує рясні структурні ознаки в особливостях LR. SSRG складається з G груп залишкової уваги з локальним джерелом (LSRAG) із з'єднаннями пропуску джерел спільного доступу (SSC). Кожен LSRAG додатково містить M спрощених залишкових блоків з пропуском локального джерела, за яким слід модуль уваги каналу другого порядку (SOCA) для використання взаємозалежностей функцій.

Детально архітектуру мережі можна побачити на Рисунку 1.13.

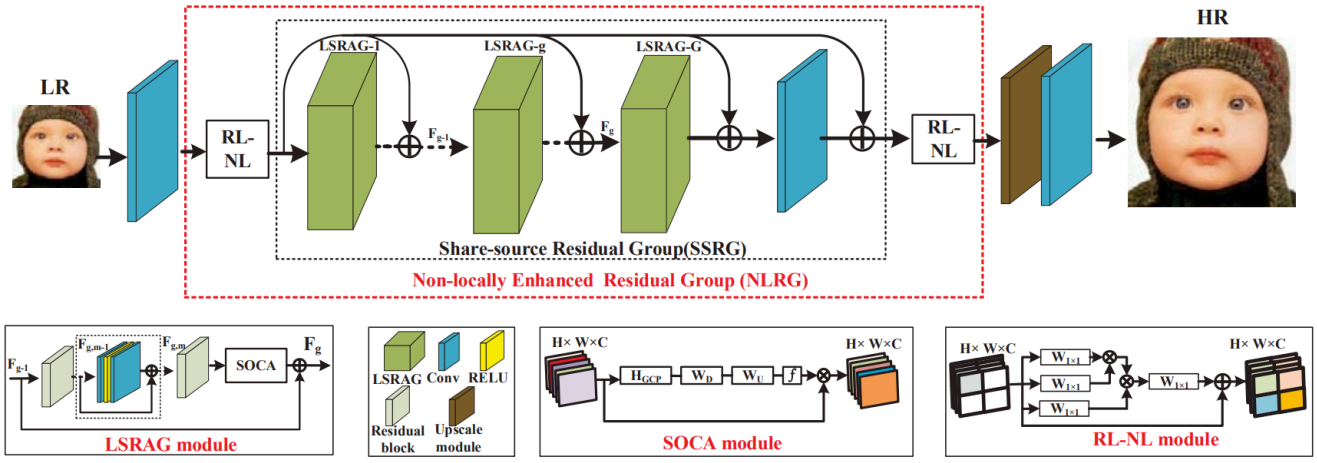

Figure 2. Framework of the proposed second-order attention network (SAN) and its sub-modules.

Рисунок 1.13 – Архітектура SAN та її модулів

Обчислення та порівняння параметрів порівнюють цей метод із 8 найсучаснішими методами SR: SPMSR, SRCNN, FSRCNN, VDSR, IRCNN, SRMD, RDN та RCAN. Усі результати для 3× покращення дають зрозуміти, що SAN досягає стабільно кращих показників, ніж інші методи.

Переваги:

– висока якість вихідних зображень;
– більш проста архітектура ніж і SRGAN;
– краща швидкодія ніж у SRGAN.

Недоліки:

– навчання заскладне для мобільних пристроїв;
– недостатня гнучкість як і усіх GAN, що застосовуються для цієї задачі.

1.2.6 Швидке та точне покращення роздільної здатності за допомогою CNN із пропуском підключення та мережею у мережі

У [11] автори зазначають, що останні методи глибокого навчання (особливо зі згортковими мережами) досягли високої продуктивності у проблемі 2 SISR від зображень з низькою роздільною здатністю (LR) до зображень з високою роздільною здатністю (HR). Вони вважають, що це тому, що глибоке навчання може поступово охоплювати як локальні, так і глобальні структури зображення одночасно, каскадуючи CNN та нелінійні шари. Однак, що стосується споживання електроенергії та обробки в режимі реального часу, глибокі згорткові мережі вимагають великих обчислень та тривалого часу обробки. У цій роботі вони пропонують полегшену мережу шляхом оптимізації структури мережі за допомогою останніх методів глибокого навчання, як показано на Рисунку 1.14. Наприклад, останні сучасні моделі SISR на основі глибокого навчання, які ми представимо в розділі 2 мають від 20 до 30 шарів CNN, тоді як нашій запропонованій моделі (DCSCN) потрібно лише 11 шарів, а загальні обчислення фільтрів CNN у 10-100 разів менші за інші.

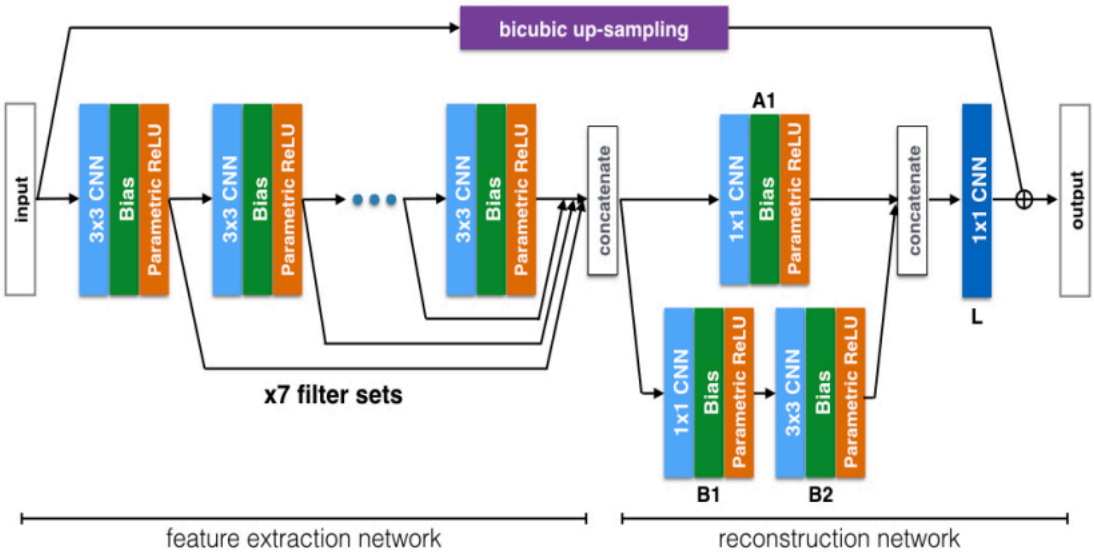

Рисунок 1.14 – Архітектура DCSCN

У разі збільшення вибірки даних зазвичай використовується транспонований згортковий шар (також відомий як шар деконволюції), запропонований Метью Д.

Зейлером [12]. Транспонований згортковий шар може вивчати ядра вибіркою, однак процес подібний до звичайного згорткового шару, а здатність до реконструкції обмежена. Щоб отримати кращі показники реконструкції, транспоновані згорткові шари потрібно укладати глибоко, що означає, що процес потребує важких обчислень. Тому автори статті пропонують паралелізовану структуру CNN, як мережа в мережі, яка зазвичай складається з однієї (або декількох) 1x1 CNN. Примітно, що шар CNN 1x1 не тільки зменшує розміри попереднього шару для більш швидких обчислень з меншими втратами інформації, але й додає більше нелінійності для збільшення потенційного представлення мережі. За допомогою такої структури можна значно зменшити кількість фільтрів CNN або транспонованих CNN. 1x1 CNN має в 9 разів менше обчислень, ніж 3x3 CNN, тому ця мережа реконструкції набагато легша, ніж інші методи глибокого навчання.

Оскільки завдання SISR зараз починають використовуватися у мережі (пристрої точки входу таких сервісів, як мобільні, планшетні та пристрої IoT), побудова невеликої, але все ще ефективної моделі є досить важливим завданням. Хоча ця модель була запропонована за допомогою численних процесів проб і помилок, повинен бути кращий спосіб налаштування структури моделі та гіпер параметрів. Необхідно встановлення методу розробки відповідної складності моделі для кожної проблеми.

Переваги:

- проста архітектура системи;
- низькі вимоги до обчислень.

Недоліки:

- гірші результати за більш складні системи;
- незакінченість архітектури.

1.3 Постановка задачі

Метою роботи є покращення моделі CNN для підвищення роздільної здатності зображень, а також розробка нового гібридного підходу, що комбінує CNN та пост-обробку.

Для досягнення мети необхідно виконати наступні завдання:

– аналіз особливостей та існуючих рішень для проблеми покращення роздільної здатності зображення для персональних комп’ютерів;

– аналіз моделей глибоких нейронних мереж для покращення роздільної здатності зображення;

– аналіз фреймворків для роботи з машинним навчанням, що доступні на мобільних платформах;

– реалізація підходу до обробки зображень;

– виконати аналіз результатів роботи створеного програмного забезпечення.

Створений продукт повинен відповідати наступним вимогам:

– простота у використанні;

– висока швидкодія;

– простота додаткового тренування на пристрої.

Висновки розділу

У першому розділі проведено аналіз існуючих рішень проблеми та предметної області, проаналізовані основні сучасні розробки, визначено переваги та недоліки цих рішень. В результаті проведеного аналізу сформульована постановка задачі, наведене призначення, цілі та задачі розробки.

2 РОЗРОБКА ГІБРИДНОГО ПІДХОДУ ДО ОБРОБКИ ЗОБРАЖЕНЬ НА МОБІЛЬНИХ ПРИСТРОЯХ

2.1 Огляд фреймворків машинного навчання, розроблених для мобільних пристроїв

2.1.1 Apple Core ML

Core ML від Apple дебютував у червні 2017 року як простий спосіб для розробників інтегрувати навчальні моделі машинного навчання у свої програми iOS, macOS та tvOS; навчені моделі завантажуються в середовище розробки Xcode від Apple і упаковуються в пакет додатків. Core ML 2 майже такий же, але більш ефективний. Apple каже, що нова версія фреймворку на 30 відсотків швидше, завдяки пакетному прогнозуванню, і що це може зменшити розмір моделей до 75 відсотків за допомогою квантування.

Тим не менш, він все ще не ідеальний. На відміну від ML Kit від Google, він не є кроссплатформенним (не підтримує Android), і хоча є можливість завантажувати моделі, такі функції, як версії, вимагають стороннього сервісу, такого як Watson Studio IBM. (Розробники, звичайно, можуть протестувати різні моделі машинного навчання за допомогою функції Apple TestFlight.)

Найновіша версія Core ML підтримує 16-розрядну плаваючу крапку і весь рівень квантування, включаючи до 1 біта, що може значно зменшити розмір моделей ШІ. Він може оновлювати моделі з хмарних служб, таких як Amazon Web Services (AWS) або Microsoft Azure під час роботи, і поставляється з конвертером, який працює з Caffe та Caffe2 Facebook, Keras, scikit-learn, XGBoost, LibSVM і Google TensorFlow Lite. (Розробники можуть створювати спеціальні конвертери для фреймворків, які не підтримуються.)

Apple рекламує свої переваги конфіденційності (додаткам не потрібно передавати дані через мережу), і повідомляє, що Core ML оптимізований для енергоефективності.

Новим також є Create ML. Це новий інструмент, прискорений графічним процесором, для навчання рідної моделі штучного інтелекту на комп'ютерах Mac, який підтримує зір та природну мову. А оскільки він кодується у Swift, розробники можуть використовувати інтерфейси програмування перетягуванням, наприклад Xcode Playgrounds, для навчання моделей.

Для розробників, які шукають універсальне рішення, яке не потребує навчання, є Apple Vision API та Natural Language Framework, що полегшує створення додатків із функцією розпізнавання облич на пристрої, скануванням штрих-коду, аналізом тексту, розпізнаванням щільності імен, та іншими функціями, не турбуючись про створення алгоритму.

2.1.2 Google ML Kit

На своїй конференції розробників I/O 2018 у травні Google представила ML Kit, кроссплатформенний набір інструментів машинного навчання для своєї платформи Firebase. ML Kit використовує API нейронної мережі на пристроях Android і iOS і призначений для стиснення та оптимізації моделей машинного навчання для мобільних пристроїв.

ML Kit використовує потужність технології машинного навчання Google Cloud Platform для «підвищеної» точності. Служба маркування зображень Google на пристрої, наприклад, містить близько 400 міток, тоді як її хмарна версія налічує понад 10 000.

Він також пропонує пару простих у використанні API для базових випадків використання: розпізнавання тексту, розпізнавання облич, сканування штрих-коду, маркування зображень та розпізнавання орієнтирів.

Спеціальні моделі, навчені з TensorFlow Lite, полегшеною системою машинного навчання Google для мобільних пристроїв, можна розгортати за допомогою ML Kit через консоль Firebase, яка обслуговує їх під час роботи програми. (Google каже, що вони також працюють над інструментом стиснення, який перетворює повноцінні моделі TensorFlow у моделі TensorFlow Lite.) Розробники

мають можливість від'єднати моделі машинного навчання від програм та обслуговувати їх під час виконання, видаляючи мегабайти від розмірів встановлення додатків та гарантуючи, що моделі завжди залишаться в актуальному стані.

Нарешті, ML Kit працює з такими функціями Firebase, як A/B тестування, що дозволяє користувачам динамічно тестувати різні моделі машинного навчання та Cloud Firestore, де зберігаються мітки зображень та інші дані.

2.1.3 Порівняння швидкодії фреймворків

У статті [16] дослідники порівняли швидкодію цих фреймворків на прикладі задачі класифікації зображень, що в цілому відображає порівняльну швидкодію технологій.

Core ML має кілька плюсів і мінусів, якщо порівнювати його з ML Kit. По - перше, він підтримує графічний процесор, що є великою перевагою. З іншого боку, набагато простіше оновити модель комплекту ML на льоту, ніж оновити модель Core ML. Крім того, Core ML є фреймворком Apple і тому працює тільки на iOS, а не на Android, тоді як ML Kit підтримує обидва.

Вони змінили приклад, щоб підтримувати перехід між комплектом ML та Core ML, що працює під однією моделлю MobileNet. Усі тести були проведені на iPhone 7. Вони також спробували запустити камеру зі швидкістю 30 або 60 кадрів в секунду, одночасно поставивши в чергу один або два кадри одночасно, і порівняли продуктивність обох моделей.

Дослідники намагалися обробляти лише один кадр у будь -який момент часу (і скидали кадри, коли вони зайняті) в обох моделях, і виявили, що Core ML трохи швидше, ніж ML Kit. Core ML займає в середньому 30 мс для обробки кожного кадру, тоді як ML Kit займає 32 мс, що показано на Рисунку 2.1. Ця невелика відмінність пояснюється тим, що Core ML використовує графічний процесор, якого ML Kit не використовує. Було очікувано, що різниця буде більшою, але ми відзначили, що Core ML іноді не працює так швидко, як інші рішення, які безпосередньо використовують Metal. З іншими моделями ця різниця також може бути більшою.

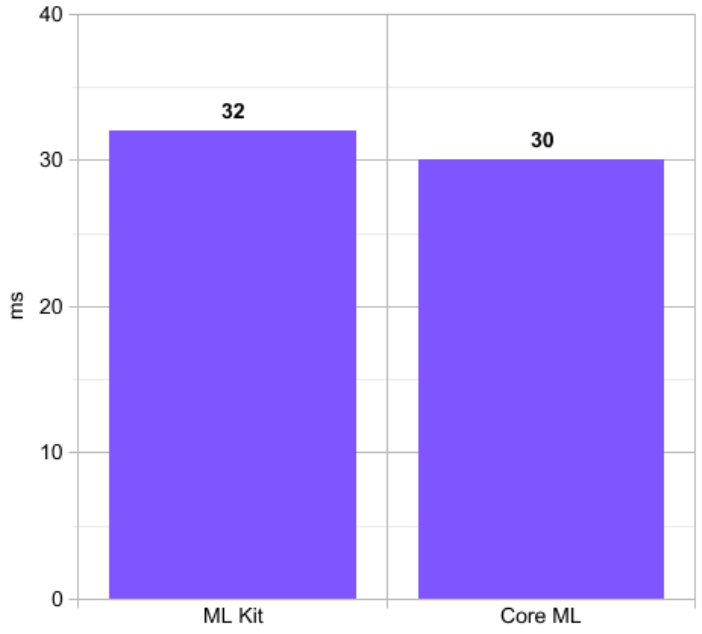

Рисунок 2.1 – Порівняння часу на обробку кадру, менше – краще

Якщо дозволяти запускати до двох кадрів одночасно, ми можемо побачити, як змінюється продуктивність цих моделей на кадр. У цьому тесті показано результати окремо для випадків, коли камера налаштована на 30 або 60 кадрів в секунду.

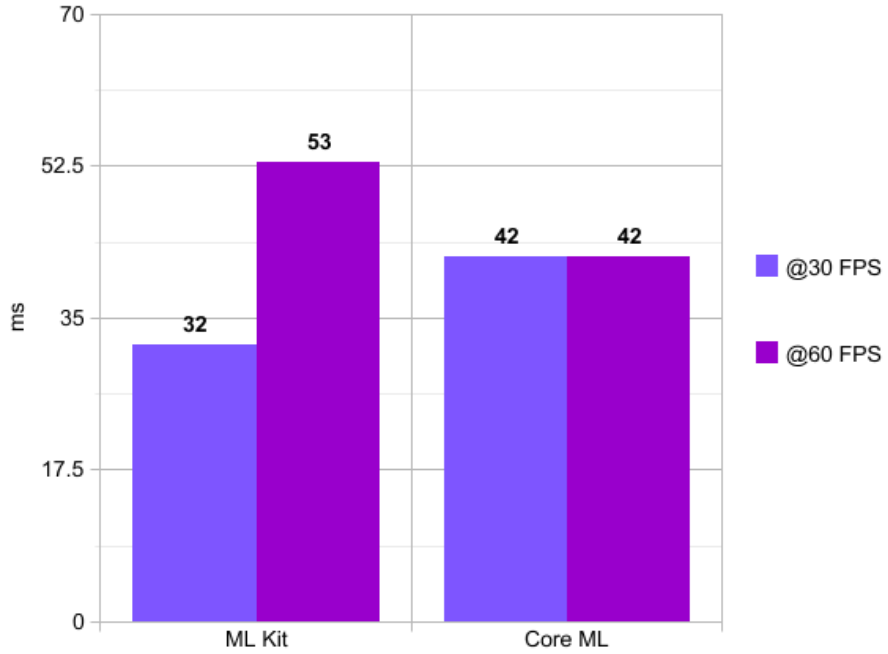

Рисунок 2.2 – Порівняння часу на обробку кадру, подвійна буферизація, менше – краще

Ця діаграма показує, скільки часу потрібно для обробки одного кадру. На цій діаграмі ми бачимо, що Core ML займає в середньому 42 мс, незалежно від того, працює камера зі швидкістю 30 або 60 кадрів в секунду, що слід очікувати, через велике навантаження моделі на графічний процесор. Однак під час роботи зі швидкістю 60 кадрів в секунду ми іноді отримуємо періоди, коли обробка кадру займає лише 33 мс. Однак також спостерігається збільшення використання процесора, що ми побачили у вікні налагодження Xcode, що пов'язано зі збільшенням кількості кадрів, які обробляються та змінюються розміри.

З іншого боку, ми бачимо, що з комплектом ML є більша різниця між камерою на 30 або 60 кадрів в секунду. Якщо ми працюємо зі швидкістю 60 кадрів в секунду, то обробка кожного кадру триває довше, оскільки відбувається більше перекриття між часом процесу кадрів, і тому, що все робиться в центральному процесорі. Однак слід зазначити, що обидва закінчуються однаковою кількістю оброблених кадрів в секунду.

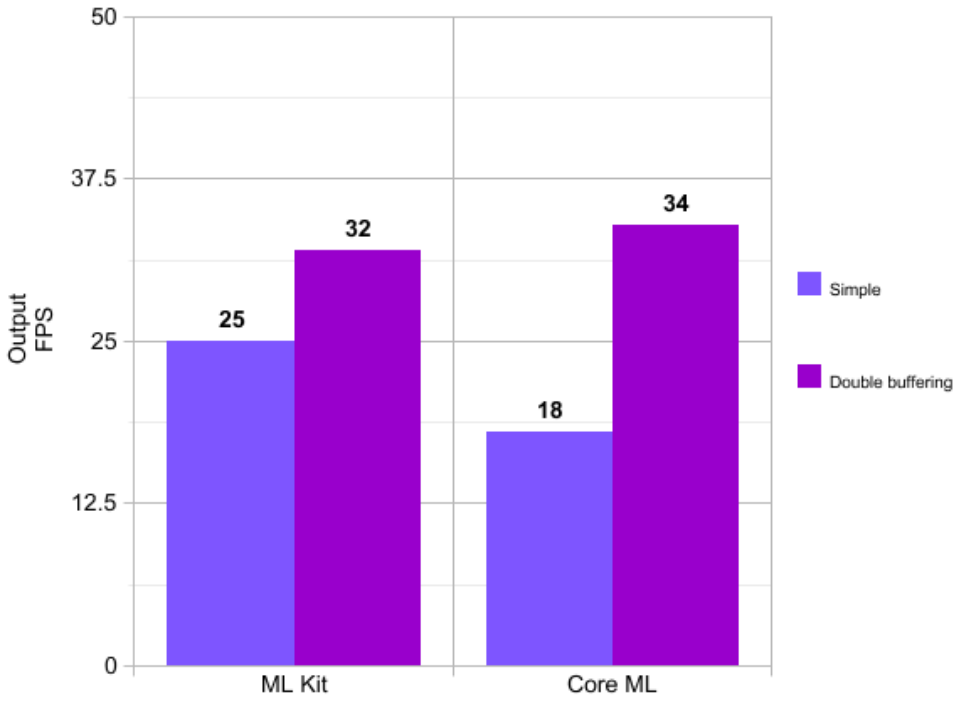

Рисунок 2.3 – Порівняння кадрів на секунду, більше – краще

На кінцевій діаграмі видно, що загалом Core ML показує трохи кращу швидкодію, при цьому будучи повністю нативним до платформ Apple.

2.2 Огляд фреймворків для створення та навчання нейронних мереж

Існує можливість тренування Core ML моделей як на комп’ютерах macOS, але це питання недостатньо задокументоване через низьку популярність та новизну технології. Також, існує можливість тренування на пристрої, яка буде розглянута далі для додаткого тренування, але для початкового тренування це не підходить через те, що користувач у додатку повинен одразу мати готову модель, а також тренування на мобільному пристрої є дуже довгим та енерговитратним процесом.

У цьому пункті буде розглянуто фреймворки машинного навчання для мови Python, оскільки вона є стандартом у індустрії штучного інтелекту та налічує багато фреймворків, детальну документацію та інше.

2.2.1 Фреймворк PyTorch

PyTorch був розроблений Facebook і вперше був опублікований у 2016 році. Він був створений, щоб запропонувати оптимізацію розробки, подібну до TensorFlow, одночасно спрощуючи написання моделей. [17]

Оскільки програмісти Python вважають його нативним у використанні, PyTorch швидко завоював користувачів, надихнувши команду TensorFlow прийняти багато найпопулярніших функцій PyTorch у TensorFlow 2.0.

PyTorch має репутацію більш широко використовуваного у дослідженнях, ніж у виробництві. Однак з моменту його випуску через рік після TensorFlow, використання PyTorch різко зросло серед професійних розробників.

У списку найпопулярніших "Інших фреймворків, бібліотек та інструментів" опитування розробників Stack Overflow 2020 повідомляється, що 10,4 відсотка професійних розробників вибирають TensorFlow, а 4,1 відсотка - PyTorch. У 2018 році відсоток становив 7,6 відсотка для TensorFlow і лише 1,6 відсотка для PyTorch. [19]

Що стосується досліджень, PyTorch є популярним вибором, і такі програми інформатики, як Стенфордська, зараз використовують його для курсів з глибокого навчання.

PyTorch заснований на Torch, фреймворку для швидких обчислень, написаному на C. Torch має обгортку Lua для побудови моделей.

PyTorch обертає ту саму основу на C в інтерфейс Python. Але це більше, ніж просто обгортка. Розробники створили його з нуля, щоб полегшити написання моделей для програмістів на Python. Основний, низькорівневий код C та C ++, оптимізований для запуску коду Python. Завдяки такій тісній інтеграції ви отримуєте такі переваги:

– краща пам'ять і оптимізація;
– більш розумні повідомлення про помилки;
– більш точний контроль структури моделі;
– більш прозора поведінка моделі;
– краща сумісність з NumPy.

Це означає, що ви можете писати високо кастомізовані компоненти нейронної мережі безпосередньо на Python, не використовуючи багато низькорівневих функцій.

PyTorch додає модуль C ++ для автодиференціації до бекенда Torch. Автодиференціація автоматично обчислює градієнт функцій, визначених у torch.nn під час зворотного поширення.

За замовчуванням PyTorch використовує обчислення в режимі eager execution. Ви можете запускати нейронну мережу під час її створення, рядок за рядком, що полегшує налагодження. Це також дозволяє створювати нейронні мережі з умовним виконанням. Це динамічне виконання є більш інтуїтивним для більшості програмістів Python.

Деякі основні API, розширення та корисні інструменти розширеної екосистеми PyTorch включають:

– API Fast.ai, що дозволяє надзвичайно легко швидко створювати моделі;
– TorchServe, модельний сервер з відкритим кодом, розроблений у співпраці між AWS та Facebook;
– TorchElastic для тренування глибоких нейронних мереж за допомогою Kubernetes;

– PyTorch Hub, активна спільнота для обміну та розширення найсучасніших моделей.

2.2.2 Фреймворк Tensorflow

TensorFlow був розроблений Google і випущений з відкритим кодом у 2015 році. Він виріс із домашнього програмного забезпечення машинного навчання Google, яке було перероблено й оптимізовано для використання у розробці. [17]

Назва «TensorFlow» описує, як ви організуєте та виконуєте операції з даними. Основною структурою даних як для TensorFlow, так і для PyTorch є тензор. Коли ви використовуєте TensorFlow, ви виконуєте операції з даними в цих тензорах, будуючи графік потоку даних, подібний до блок -схеми, яка запам'ятовує минулі події.

До TensorFlow 2.0 TensorFlow вимагав від вас вручну зробити абстрактне дерево синтаксису - графік - шляхом викликів tf.* API. Потім потрібно було вручну скомпілювати модель, передаючи набір вихідних тензорів і тензорів входу до виклику session.run().

Об’єкт Session — це клас для виконання операцій TensorFlow. Він містить середовище, в якому оцінюються об'єкти Tensor і виконуються об'єкти Operation, і може володіти такими ресурсами, як об'єкти tf.Variable. Найпоширеніший спосіб використання сеансу – це як менеджер контексту.

У TensorFlow 2.0 ви все ще можете створювати моделі таким чином, але легше використовувати eager execution режим виконання, з яким зазвичай працює Python. Eager execution негайно оцінює операції, тому ви можете написати свій код за допомогою потоку керування Python, а не потоку керування графіком.

Якщо ви не хочете або вам не потрібно створювати низькорівневі компоненти, рекомендований спосіб використання TensorFlow — Keras. Він має простіші API, згортає звичайні варіанти використання в готові компоненти для вас і забезпечує кращі повідомлення про помилки, ніж базовий TensorFlow.

TensorFlow має велику і добре налагоджену базу користувачів і безліч інструментів, які допомагають у розробці машинного навчання. Для мобільної

розробки він має API для JavaScript і Swift, а TensorFlow Lite дозволяє стискати й оптимізувати моделі для пристроїв Інтернету речей та мобільних пристроїв.

Ви можете швидко розпочати використання TensorFlow завдяки багатству даних, попередньо підготовленим моделям і записникам Google Colab, які надають як Google, так і сторонні розробники.

Багато популярних алгоритмів машинного навчання та наборів даних вбудовані в TensorFlow і готові до використання. На додаток до вбудованих наборів даних ви можете отримати доступ до наборів даних Google Research або скористатися пошуком наборів даних Google, щоб знайти ще більше.

Keras полегшує запуск моделей, тож ви можете випробовувати нові методики за менший час. Справді, Keras є найбільш використовуваною системою глибокого навчання серед п’ятірки найкращих команд Kaggle.

Одним з недоліків є те, що оновлення з TensorFlow 1.x до TensorFlow 2.0 змінило таку кількість функцій, що легко у них заплутатися. Оновлення коду є стомлюючим і схильним до помилок. Багато ресурсів, як-от навчальні посібники, можуть містити застарілі поради.

Tensorboard є чудовим інструментом, коли справа доходить до візуалізації. Цей інструмент поставляється з TensorFlow і дуже корисний для налагодження та порівняння різних навчальних запусків. Наприклад, вважайте, що ви навчили модель, потім налаштували деякі гіперпараметри і знову навчили її. Обидва запуски можуть відображатися на Tensorboard одночасно, щоб вказати можливі відмінності. Tensorboard може:

- показати графік моделі;
- побудувати графічні скалярні змінні;
- візуалізувати розподіли та гістограми;
- візуалізувати зображень;
- візуалізувати вбудовування;
- відтворити аудіо.

Tensorboard може відображати різні підсумки, які можна зібрати за допомогою модуля tf.summary.

Конкурентом Tensorboard з боку PyTorch є visdom. Він не настільки повноцінний, але дещо зручніший у використанні. Крім того, існує інтеграція з Tensorboard. Крім того, ви можете вільно використовувати стандартні інструменти для складання графіків - matplotlib та seaborn.

Деякі основні API, розширення та корисні інструменти розширеної екосистеми TensorFlow включають:

– TensorFlow Hub, бібліотека для багаторазових модулів машинного навчання;

– Model Garden-офіційна колекція моделей, які використовують високорівневі API TensorFlow.

Також, варто зазначити що перевагою Tensorflow останніх версій є підтримка навчання на AMD GPU у комп’ютерах Apple з процесорами Intel, а також підтримка Neural Engine у останніх десктопних процесорах лінійки Apple M1. Оскільки мобільні розробники для платформи iOS та кросплатформенних додатків працюють зазвичай саме на комп’ютерах від Apple, це прискорення дуже спростить роботу з машинним навчанням для них.

2.2.3 Конвертація між форматами моделей нейронної мережі за допомогою Core ML Tools

Пакет coremltools використовується для перетворення моделей машинного навчання із сторонніх бібліотек у формат Core ML. [23] Пакет для Python містить допоміжні інструменти для перетворення моделей з навчальних бібліотек, наприклад:

– TensorFlow 1.x;

– TensorFlow 2.x;

– PyTorch;

– TensorFlow's Keras APIs.

Ненейронні мережі:

– scikit-learn;

– XGBoost;

– LibSVM.

За допомогою coremltools ви можете зробити наступне:

– перетворити навчені моделі у формат Core ML;

– читати, писати та оптимізувати моделі Core ML;

– перевірити перетворення/створення (на macOS), роблячи прогнози за допомогою Core ML;

– після перетворення ви можете інтегрувати моделі Core ML зі своїм додатком за допомогою Xcode.

Цей пакет розроблений самою корпорацією Apple, тому є найбільш надійним варіантом конвертації.

Пакет coremltools 5 (остання версія на момент написання роботи) пропонує кілька покращень продуктивності порівняно з попередніми версіями, включаючи такі нові функції:

– пакет моделі Core ML: новий формат контейнера моделі, який розділяє модель на компоненти і пропонує більш гнучке редагування метаданих і кращий контроль версій;

– програма ML: новий тип моделі, який представляє обчислення як програмні інструкції, пропонує більше контролю над точністю його проміжних тензорів і кращу продуктивність.

2.3 Розробка архітектури нейронної мережі

Згорткові (конволюційні) нейронні мережі - це алгоритми глибокого навчання, які зазвичай використовуються для розпізнавання зображень та обробки природної мови. Їхня архітектура натхнена організацією нейронів зорової кори людини, завдяки чому вони дуже добре вловлюють шаблони з вхідних зображень.

Причина, чому CNN віддають перевагу при обробці зображень, пов’язана з кількістю параметрів, які вони використовують. Якщо ми розглянемо вхідні дані з розмірами 1920x1080, у нас буде понад 2 мільйони пікселів із зазвичай трьома кольоровими каналами кожен, у більшості глибоких нейронних мереж це зажадає

величезної кількості обробки, а вихід не буде таким хорошим, оскільки дані буде зведено в одновимірний масив, що призведе до втрати інформації з вихідного зображення.

Згорткові мережі по суті зменшують вхідні зображення до меншої матриці, зберігаючи при цьому оригінальні характеристики, щоб гарантувати кращий кінцевий результат. Щоб досягти цього, вхідні дані надходять у згортковий шар та шар об'єднання, і, нарешті, до шару класифікації, також відомого як повністю підключений шар. На відміну від більшості інших нейронних мереж, всі нейрони в CNN мають однакову вагу і, як правило, не всі з'єднані між шарами. [20]

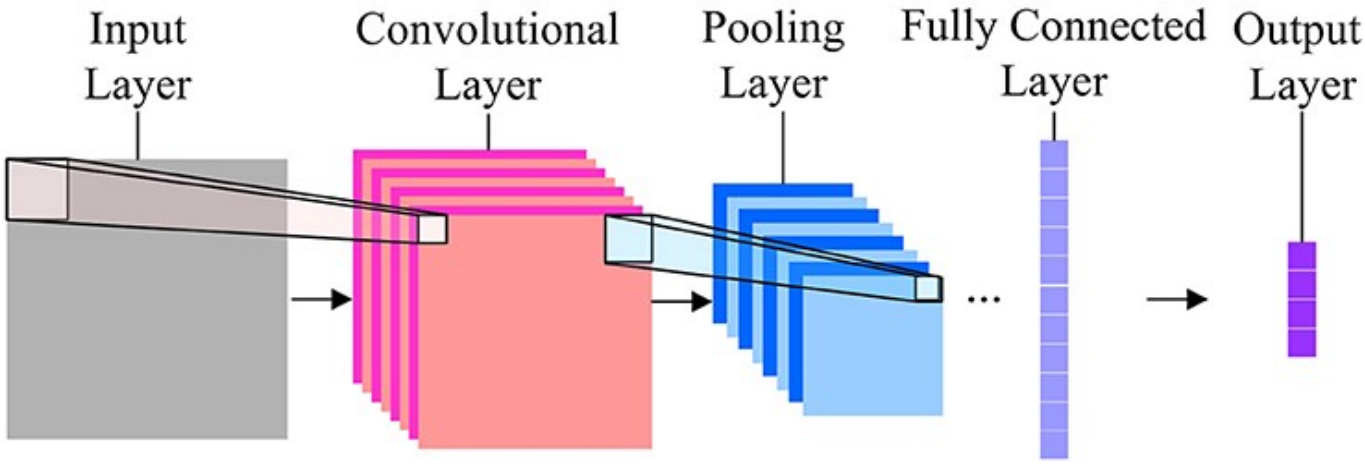

Рисунок 2.4 – Загальна архітектура згорткових нейронних мереж

Щоб зменшити розмір вхідних даних, до зображення застосовуються фільтри, які називаються ядрами, зазвичай вони мають розмір 3x3 або 5x5 і виділяють високорівневі функції, такі як краї, або застосовують перетворення, такі як розмиття. Ці операції можуть призвести до збільшення або збереження розмірів згорнутого елемента за допомогою Same Padding або зменшення за допомогою Valid Padding. [21] Ядро рухається по всьому зображенню за шаблоном, показаному на Рис 2.5:

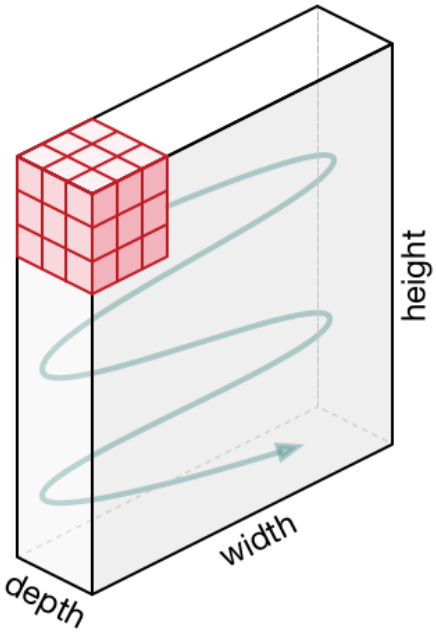

Рисунок 2.5 – Шаблон руху фільтрів вхідних даних

Згорткові нейронні мережі можуть використовувати набір фільтрів для виділення функцій із вхідного зображення, які дозволяють категоризувати.

Перші шари фіксують менші та простіші функції, наприклад виявлення країв. Якщо ви об’єднаєте кілька шарів і фільтрів, мережа може почати виявляти складніші ознаки.

Кожне ядро виявляє певну ознаку з вхідного зображення, як показано на Рис. 2.6. Карта ознак створюється нейронами, використовуючи те саме ядро, і вона змінюється під час навчання моделі з мінімізованою функцією втрат. Ці операції передаються до функції ReLU, де шари та карти об’єктів об’єднуються, а за допомогою зворотного поширення оновлюються значення в матрицях фільтрів.

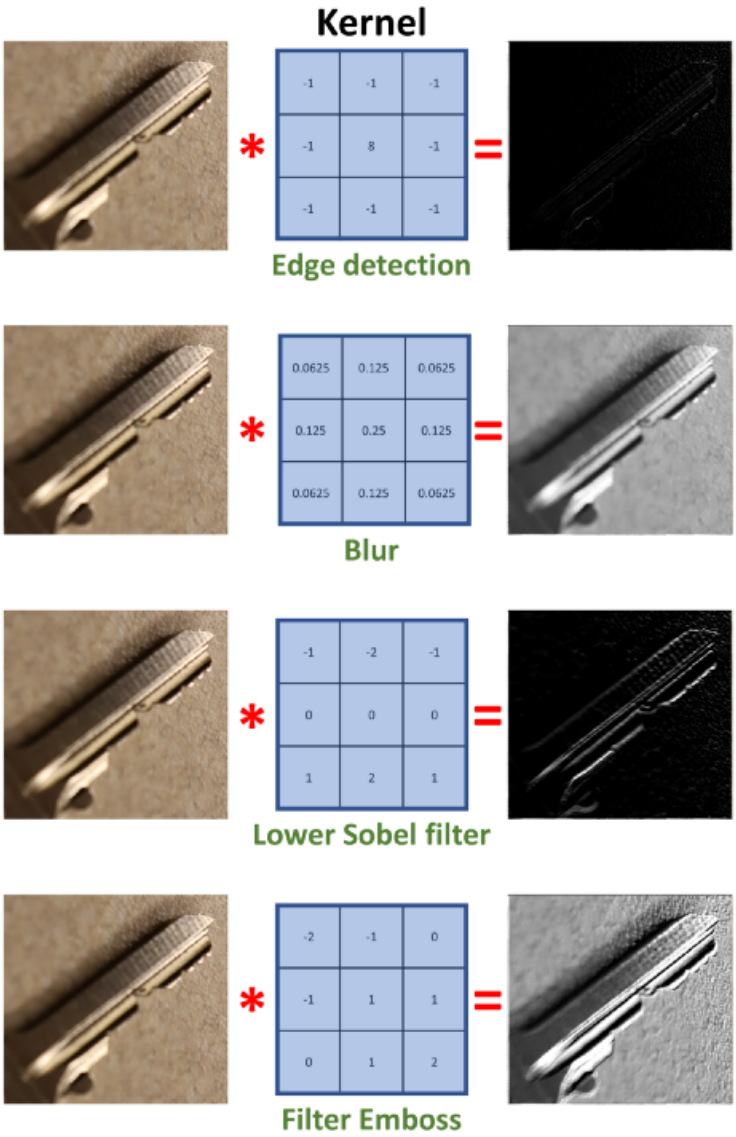

Рисунок 2.6 – Застосування ядер у згортковому шарі

Рівень об’єднання використовується для зменшення розмірів виводу згорткового шару, зменшуючи обробку, необхідну для інтерпретації даних, він також корисний для виділення позиційних інваріантних домінуючих ознак, підтримуючи процес навчання моделі. Цей шар отримує вихідне зображення з попереднього шару і повертає максимальне сукупне значення або максимальний пул, він також може мінімізувати або перетворити його в іншу функцію, відому як середнє об’єднання. Це зменшує ймовірність переобладнання шляхом з’ясування ймовірності найбільш впливових функцій.

Згортковий шар і шар об’єднання можна об’єднати в ланцюжок, що дозволяє знаходити об’єкти нижнього рівня в більш складних зображеннях. Однак збільшення

кількості шарів має недоліки, оскільки для обробки мережі знадобиться набагато більше обчислювальної потужності.

Нарешті, ми можемо вирівняти вихідні дані об’єднання та передати їх у класифікаційну нейронну мережу. Цей шар розрізняє та визначає ймовірність приналежності низькорівневої або домінантної ознаки до певного класу за допомогою функцій класифікації Softmax або Sigmoid. Як і більшість звичайних нейронних мереж, модель навчається ітераційним способом, використовуючи кілька епох і зворотне поширення.

Наша CNN складатиметься з 4 основних шарів та останнього шару перетворення. Ми будемо використовувати «tanh» як нашу функцію активації та MSE як нашу функцію втрати. Максимальна кількість епох буде встановлено 100, оскільки база даних досить мала, і ми виберемо ADAM як наш алгоритм оптимізації.

Шари CNN:

1) шар 64x5x5, що використовується для ідентифікації об'єктів;

2) шар 32x3x3, що використовується для більш тонкої ідентифікації ознак;

3) ще один шар 32x3x3, який використовується для більш точної ідентифікації ознак;

4) підпіксельний шар згортки, де кожен новий прогнозований піксель розміщується відповідно до розміру вихідного зображення;

5) шар, у якому отримана матриця перетворюється в матрицю двовимірного зображення.

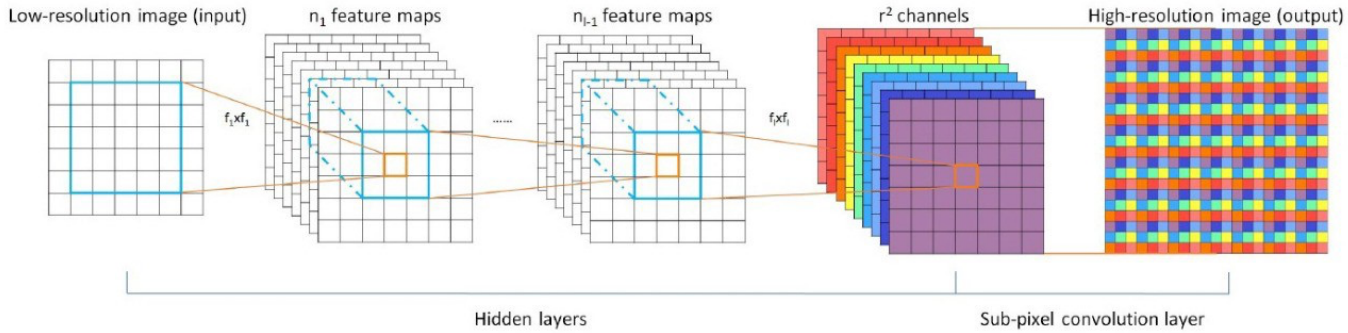

Рисунок 2.7 – Схема шарів нейронної мережі

2.4 Вибір датасету для тренування

Основними вимогами до датасету є велика кількість та різноманітність фотографій, що дасть гарну репрезентацію типових фото, які робляться на мобільний телефон.

Для виконання навчання моделі, потрібно або знайти датасет з готовим валідаційним набором, або розбити зображення на групи самостійно.

Berkeley Segmentation Data Set 500 (BSDS500) є стандартним датасетом для визначення контурів. [24] Цей набір даних призначений для оцінки виявлення природних країв, що включає не лише контури об’єкта, а й внутрішні межі об’єкта та межі фону. Цей же датасет дуже часто використовують і у задачах покращення роздільної здатності. Він містить 500 природних зображень із ретельно позначеними межами, зібраними від кількох користувачів. Набір даних розділений на три частини: 200 для навчання, 100 для перевірки та решта 200 для тестування.

Набір даних *Urban100* містить 100 зображень міських сцен. Він зазвичай використовується як тестовий набір для оцінки продуктивності моделей для покращення роздільної здатності. [25] Самі фотографії є досить різноманітними, як показано на Рис. 2.8, але для задачі цієї роботи він є замалим.

DIV2K – це датасет RGB-зображень з великою різноманітністю вмісту, який дуже часто використовується для хакатонів та інших змагань у сфері машинного навчання. [26]

Набір даних DIV2K поділяється на:

– дані для тренування: починаючи з 800 зображень високої чіткості з високою роздільною здатністю, отримано відповідні зображення з низькою роздільною здатністю та надаємо зображення як з високою, так і з низькою роздільною здатністю для 2, 3 і 4 коефіцієнтів зменшення;

– дані перевірки: 100 зображень високої чіткості з високою роздільною здатністю використовуються для створення відповідних зображень з низькою роздільною здатністю, низька роздільна здатність надається з початку завдання та призначена для того, щоб учасники отримували онлайн-відповідь від сервера

перевірки; зображення з високою роздільною здатністю будуть опубліковані, коли почнеться останній етап завдання;

– тестові дані: 100 різноманітних зображень використовуються для створення відповідних зображень з низькою роздільною здатністю; учасники отримають зображення з низькою роздільною здатністю, коли почнеться фінальна фаза оцінювання, а результати будуть оголошені після завершення змагання та визначення переможців.

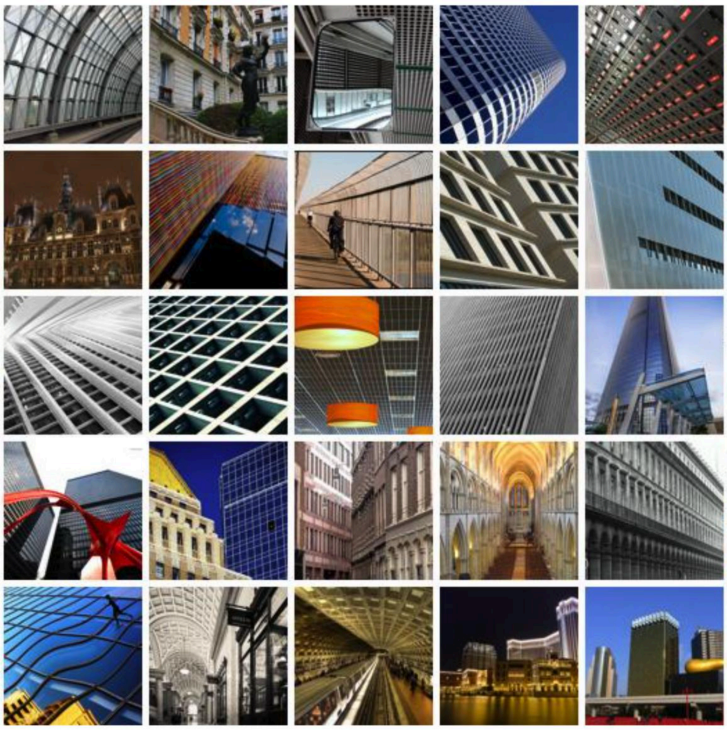

Рисунок 2.8 – Фотографії з датасету Urban100

Intel Image Classification Dataset – це набір зображень природних сцен у всьому світі. [27] Цей датасет містить близько 25 тисяч зображень розміром 150x150, розподілених за 6 категоріями. Ці дані оригінально були опубліковані компанією Intel для проведення конкурсу щодо класифікації зображень. Кількість зображень у цьому датасеті більш ніж чудова, але початкова роздільна здатність занадто мала, адже датасет пристосований для інших задач.

Flickr30k Dataset – це набір зображень розміром у 30 тисяч штук, що складається з фото, розміщених у соціальній мережі / фото хостингу Flickr. [28] Цей

датасет є ідеальним для цього дослідження, оскільки він уособлює різноманітну вибірку фотографій, які люди роблять на смартфони, має багато різних сценаріїв та об’єктів на фото. Також, зображення для навчання мають досить високу роздільну здатність, але не занадто високу, враховуючи що тренування та ітерації розробки повинні проходити швидко на звичайному комп’ютері. Єдиним вагомим недоліком цього набору зображень є відсутність валідаційного набору, який доведеться створити самостійно.

Висновки розділу

В даному розділі було розглянуто та проаналізовано фреймворки для машинного навчання, що використовуються на мобільних пристроях, наведено опис і характеристики ML Kit та Core ML, їх основні переваги та недоліки. Для розробки було зроблено вибір у користь Core ML через кращу швидкодію та нативність до платформ Apple.

Також, було розглянуто фреймворки для процесу опису моделі та тренування, наведено опис і характеристики TensorFlow та PyTorch, їх основні переваги та недоліки. Для тренування системи було вибрано TensorFlow через більшу поширеність фреймворку та систему TensorBoard, що використовується для візуалізації процесу навчання мережі.

Наостанок, було вибрано датасет для тренування – Flickr30k через різноманітність та велику кількість фотографій у ньому, що репрезентують типові фотографії, що робляться на смартфон.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Навчання нейронної мережі для покращення роздільної здатності зображень

3.1.1 Налаштування середовища для навчання

Першим етапом, необхідним для навчання моделі для покращення роздільної здатності зображень, є налаштування робочого середовища. Навчання було проведено на двох комп'ютерах с наступними характеристиками:

1. Apple MacBook Pro 13" 2020:
   - процесор Apple M1 (8 ядер процесора, 8 ядер графічного процесора, 16 ядер нейронного двигуна);
   - 16 ГБ оперативної пам'яті;
   - операційна система macOS Big Sur 11.6.
2. Apple MacBook Pro 16" 2019:
   - процесор Intel Core i7 9750H (6 ядер);
   - графічний процесор AMD Radeon 5300M (4 ГБ відеопам'яті);
   - 16 ГБ оперативної пам'яті;
   - операційна система macOS Big Sur 11.6.

Для створення та навчання моделі потрібно встановити фреймворки TensorFlow та Keras та їх залежності: бібліотеки PIL та matplotlib, а також набір інструментів для конвертації моделей у формат CoreML – coremltools. Для коректної роботи з процесорами родини Apple M1 потрібно було використовувати версію TensorFlow 2.4 alpha3. Також, тільки ця версія фреймворку підтримує графічні процесори у системах на базі Intel для прискорення навчання при використанні на операційній системі macOS. Нажаль, остання на момент виконання дослідження остання стабільна версія coremltools 4.1 не підтримує TensorFlow 2.4, але бета-релізи версії 5 працюють нормально.

3.1.2 Підготовка зображень з датасету Flickr30k

Для початку нам потрібно буде знайти та імпортувати набір даних зображень для навчання нашої моделі. Оскільки датасет Flickr30k не надходить з набором для валідації, нам потрібно виділити його самостійно. Після імпорту зображень нам доведеться розділити набори даних для навчання та перевірки, а також визначити, який розмір вихідного зображення та коефіцієнт збільшення буде використано. Процес імпорту та розділу набору даних показано на Лістингу 3.1.

Лістинг 3.1 – Імпорт та розділ наборів даних

```
url = "https://storage.googleapis.com/kaggle-data-
sets/31296/39911/bundle/archive.zip"
model_name = "image_upscaling_model"
dataset_name = "flickr_images/images"
download_file_path = tf.keras.utils.get_file(origin=url,
fname="flickr30k_images.zip", extract=True)
parent_path = Path(download_file_path).parent.absolute()
small_data_path = os.path.join(parent_path,
"flickr30k_images/flickr30k_images")

target_size = 300
upscale_factor = 3
input_size = target_size // upscale_factor

train_data_raw =
tf.keras.preprocessing.image_dataset_from_directory(small_da
ta_path, label_mode=None, image_size=(target_size,
target_size),validation_split=0.2, subset="training",
seed=42)
```

```
validate_data_raw =
tf.keras.preprocessing.image_dataset_from_directory(small_da
ta_path, label_mode=None, image_size=(target_size,
target_size),validation_split=0.2, subset="validation",
seed=42)
```

Після імпорту набору даних зображення ми можемо масштабувати колірні канали пікселів у діапазоні від 0 до 1 замість 0 до 255 і змінити формат з RGB на YUV. Це полегшує процес навчання, а кінцевий результат буде краще сприйматися людьми, оскільки під час навчання модель буде приймати значення яскравості кожного зображення. Після цього ми розділимо набори даних навчання та перевірки та зменшимо розмір зображень для навчання, що показано на Лістингу 3.2.

Лістинг 3.2 – Обробка зображень для тренування

```
train_data = train_data_raw.map(scale)
validate_data = validate_data_raw.map(scale)

def scale(img):
    return (img)/255.0

def process_input(img, target_size):
    return tf.image.resize(img, [target_size, target_size],
method="area")

def process_target(img):
    img = tf.image.rgb_to_yuv(img)
    return tf.split(img, 3, axis=3)[0]

train_data_yuv = train_data.map(process_target)
```

```
train_data_scaled = train_data_yuv.map(lambda img:
(process_input(img, input_size, upscale_factor), img))

train_ds = train_data_scaled.prefetch(buffer_size=32)

validate_data_yuv = validate_data.map(process_target)
validate_data_scaled = validate_data_yuv.map(lambda img:
(process_input(img, input_size, upscale_factor), img))

valid_ds = validate_data_scaled.prefetch(buffer_size=32)
```

На Рисунку 3.1 показано приклад зображення для навчання та валідації після попередньої обробки. Зауважте, що для цього дослідження ми використовуємо коефіцієнт збільшення 3.

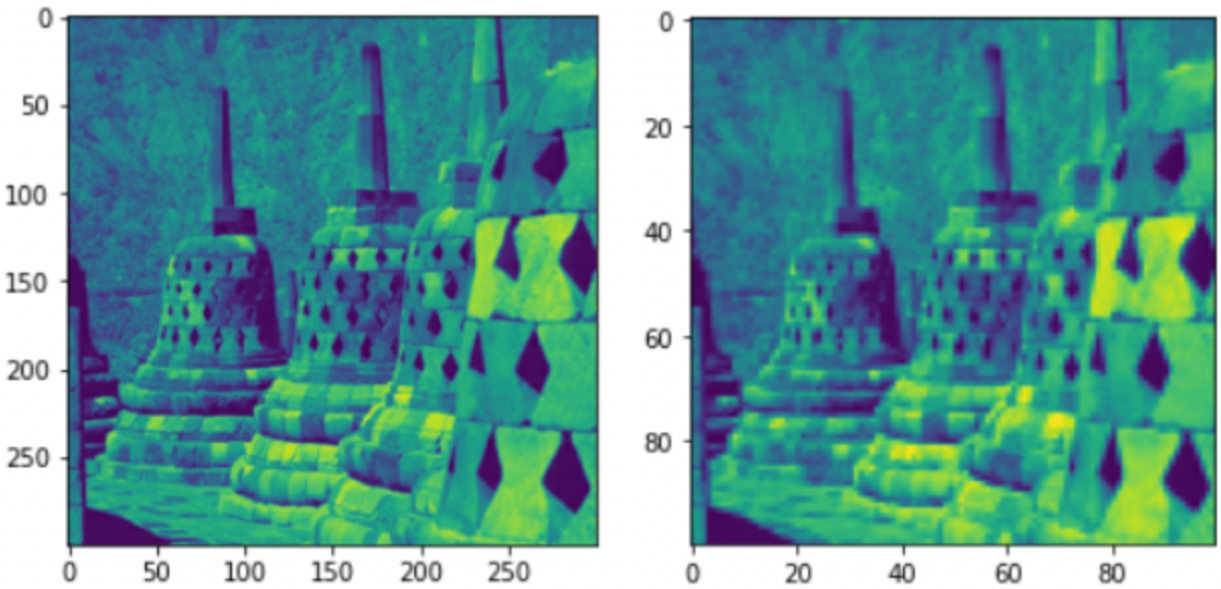

Рисунок 3.1 – Зображення після попередньої обробки

3.1.3 Опис моделі та процес тренування

Тепер, коли наші зображення готові до навчання, ми можемо описати нашу модель за допомогою Keras і почати процес навчання.

Наша CNN складатиметься з 4 основних шарів та останнього шару перетворення. Ми будемо використовувати «tanh» як нашу функцію активації та MSE як нашу функцію втрати. Максимальна кількість епох буде встановлено 100, оскільки база даних досить мала, і ми виберемо ADAM як наш алгоритм оптимізації.

Шари CNN:

1) шар 64x5x5, що використовується для ідентифікації об'єктів;

2) шар 32x3x3, що використовується для більш тонкої ідентифікації ознак;

3) ще один шар 32x3x3, який використовується для більш точної ідентифікації ознак;

4) підпіксельний шар згортки, де кожен новий прогнозований піксель розміщується відповідно до розміру вихідного зображення;

5) шар, у якому отримана матриця перетворюється в матрицю двовимірного зображення.

Код опису моделі надано у Лістингу 3.3.

Лістинг 3.3 – Опис моделі для покращення роздільної здатності зображень

```
loss_function = keras.losses.MeanSquaredError()
optimizer = keras.optimizers.Adam(learning_rate=0.001)
epochs = 100

conv_args = {"activation": "tanh",
             "kernel_initializer": "Orthogonal",
             "padding": "same"}
inputs = keras.Input(shape=(input_size, input_size, 1))
x = layers.Conv2D(64, 5, **conv_args)(inputs)
x = layers.Conv2D(32, 3, **conv_args)(x)
```

```
x = layers.Conv2D(32, 3, **conv_args)(x)
x = layers.Conv2D(1 * (upscale_factor ** 2), 3,
**conv_args)(x)
outputs = tf.nn.depth_to_space(x, upscale_factor)
```

Опис шарів відбувається за допомогою `layers.Conv2D` у випадку згорткових шарів, як у нашому випадку. Кожному шару передається функція активації та функція втрати. Після цього ми додаємо можливість моделі зупинити тренування якщо досянуто бажаний поріг втрат, а також запускаємо моніторинг втрат для подальшого аналізу у TensorBoard.

Також для подальшого аналізу варто під’єднати TensorBoard callback, що дозволить цьому інструменту скласти повну інформаційну сторінку з деталями тренування, дані з якої знадобляться для оцінки ефективності моделі. Код тренування та налаштування TensorBoard показано на Лістингу 3.4.

Лістинг 3.4 – Налаштування TensorBoard та тренування моделі

```
early_stopping_callback =
keras.callbacks.EarlyStopping(monitor="loss", patience=10)

log_dir = "logs/fit/" +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

callbacks = [early_stopping_callback, tensorboard_callback]

model = keras.Model(inputs, outputs)
model.compile(optimizer=optimizer, loss=loss_function,
metrics=['accuracy'])
```

```
model.fit(train_ds, epochs=epochs, validation_data=valid_ds,
verbose=2, callbacks=callbacks)
keras.models.save_model(model, model_name)
```

Перш ніж перейти до конвертації моделі у формат для мобільного пристрою та інтеграції у мобільний додаток, варто перевірити результати на персональному комп'ютері за допомогою коду на Python.

Після навчання нашої моделі нам доведеться застосувати її до тестових зображень і перетворити вихідні зображення назад у колірний простір RGB, перш ніж аналізувати результати. У нашому коді ми визначили функцію під назвою upscale_image (Лістинг 3.5), яка буде робити саме це за нас.

Лістинг 3.5 – Функція покращення роздільної здатності зображення на Python

```
def upscale_image(model, img):
    ycbcr = img.convert("YCbCr")
    y, cb, cr = ycbcr.split()
    y = img_to_array(y)
    y = y.astype("float32") / 255.0

    input = np.expand_dims(y, axis=0)
    out = model.predict(input)

    out_img_y = out[0]
    out_img_y *= 255.0

    out_img_y = out_img_y.clip(0, 255)
    out_img_y = out_img_y.reshape((np.shape(out_img_y)[0],
np.shape(out_img_y)[1]))
```

```
    out_img_y = PIL.Image.fromarray(np.uint8(out_img_y),
mode="L")
    out_img_cb = cb.resize(out_img_y.size,
PIL.Image.BICUBIC)
    out_img_cr = cr.resize(out_img_y.size,
PIL.Image.BICUBIC)
    out_img = PIL.Image.merge("YCbCr", (out_img_y,
out_img_cb, out_img_cr)).convert("RGB")
    return out_img
```

Порівняння оригінального зображення, зображення з низькою роздільною здатністю та результатом роботи нейронної мережі продемонстровано на Рисунку 3.2.

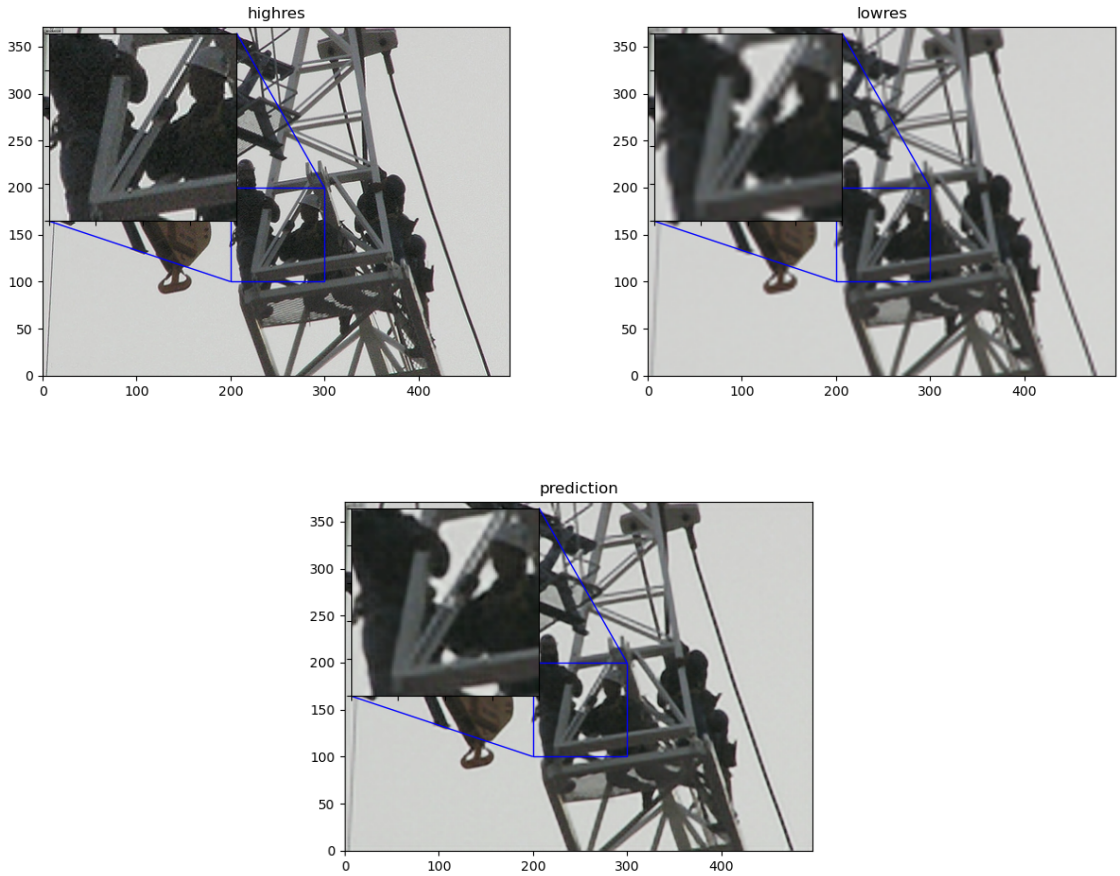

Рисунок 3.2 – Результат роботи нейронної мережі (оригінальне зображення, версія у низькій роздільній здатності, та результат роботи нейронної мережі)

3.2 Інтеграція гібридного підходу у мобільний додаток iOS

3.2.1 Конвертація моделі TensorFlow у формат Core ML

Пакет coremltools використовується для перетворення моделей машинного навчання із сторонніх бібліотек у формат Core ML. [23] Пакет для Python містить допоміжні інструменти для перетворення моделей з навчальних бібліотек, зокрема TensorFlow 2.x.

Процес конвертації готової моделі є неймовірно простим, потрібно завантажити Keras модель методом `keras.models.load_model`, визначити назви вхідних та вихідних параметрів мережі та викликати метод конвертації, як показано на Лістингу 3.6.

Лістинг 3.6 – Конвертація моделі TensorFlow Keras у формат Core ML

```
from tensorflow import keras
import coremltools

model_name = "image_upscaling_model_2"
model = keras.models.load_model(model_name)

print("[INFO] converting model")
coreml_model = coremltools.converters.convert(model,
    input_names="image",
    image_input_names="image",
    image_scale=1/255.0,
    is_bgr=True)

output = model_name + ".mlmodel"
print("[INFO] saving model as {}".format(output))
coreml_model.save(output)
```

3.2.2 Створення проєкту та інтеграція моделі

У середовищі розробки Xcode було потрібно створити iOS проєкт зі стандартними налаштуваннями. Після цього, при додаванні файлу моделі у проєкт, середовище розробки самостійно завантажує усю інформацію про модель, таку як кількість та типи шарів, а також назви та форму вхідних та вихідних параметрів, як показано на Рисунку 3.3.

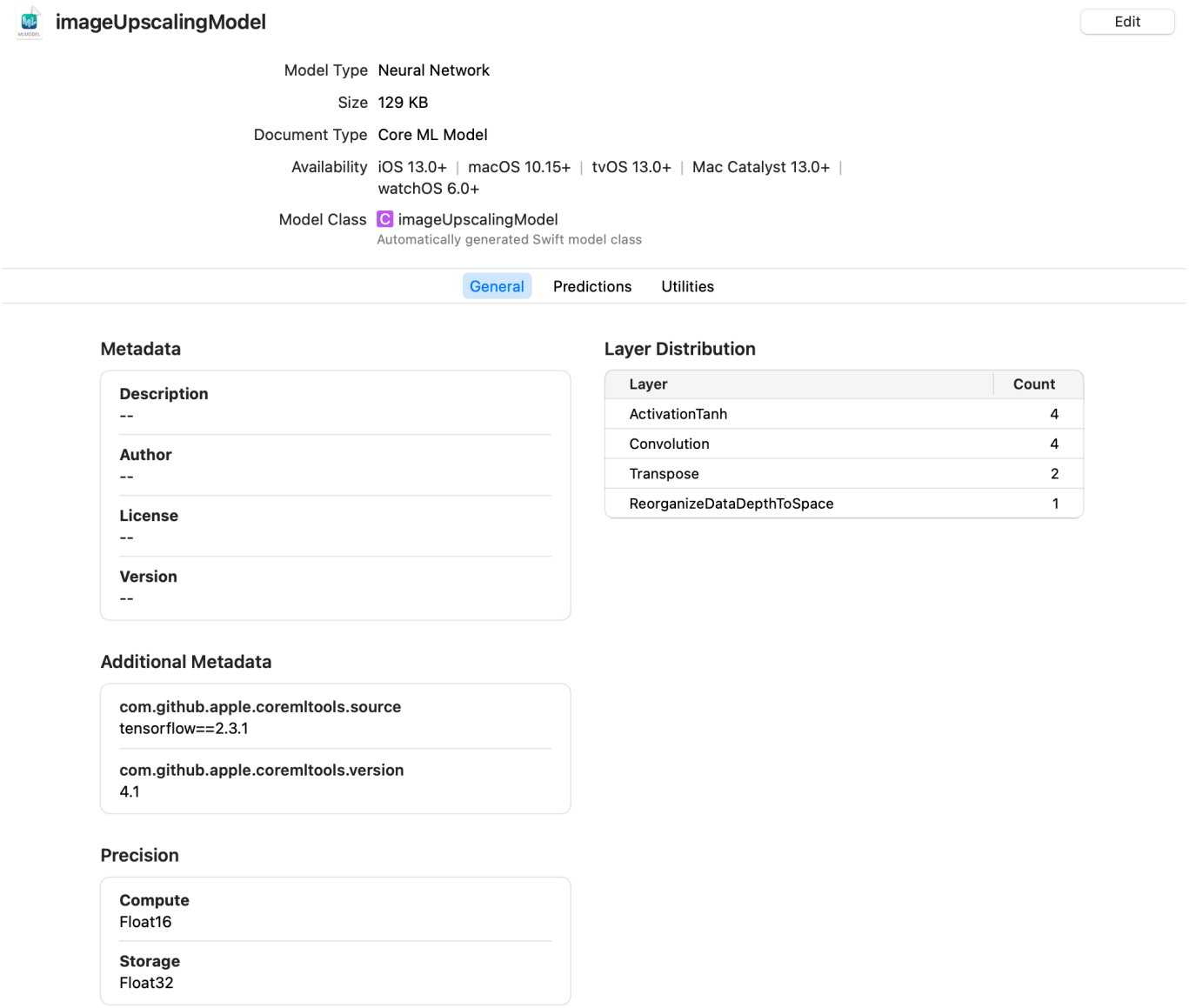

Рисунок 3.3 – Завантажені шари мережі, версії TensorFlow та coremltools

На цьому ж рисунку видно клас моделі `imageUpscalingModel`, автоматично згенерований середовищем розробки, який включає у себе всі вхідні та вихідні поля моделі у вигляді класів `imageUpscalingModelInput` та `imageUpscalingModelOutput` відповідно для легкого її використання.

Для передачі зображення на вхід моделі, потрібно конвертувати його в кольоровий простір YUV. Ця операція є вбудованою у фреймворк TensorFlow на мові Python для персональних комп'ютерів, але на мобільному пристрої на мові Swift його доведеться реалізувати самостійно.

Існує декілька стандартів конвертації між RGB та YUV, зокрема BT.470 для SDTV та новіший BT.709 для HDTV.

Y′UV сигнали зазвичай створюються з джерела RGB (червоного, зеленого та синього). Зважені значення R, G і B підсумовуються, щоб отримати Y′, міру загальної яскравості. U та V обчислюються як масштабовані відмінності між Y′ та значеннями B і R.

Для конвертації до стандарту BT.470 маємо наступну матрицю: [30]

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix},$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}.$$

Зауважте, що для малих значень Y' можна отримати негативні значення R, G або B, тому на практиці ми закріплюємо результати RGB до інтервалу [0,1].

Для HDTV (телебачення високої роздільної здатності) ATSC вирішив змінити базові значення WR і WB порівняно з попередньо вибраними значеннями в системі SDTV. Для HDTV ці значення надаються у [30]. Це рішення додатково вплинуло на матрицю для перетворення Y′UV до RGB, тому її значення членів також дещо відрізняються. Як наслідок, із SDTV та HDTV є, як правило, два різних представлення Y′UV для будь-якої трійки RGB: SDTV-Y′UV і HDTV-Y′UV. Це в деталях означає, що при безпосередньому перетворенні між SDTV і HDTV інформація про яскравість

(Y′) приблизно однакова, але представлення інформації каналу кольоровості (U & V) потребує перетворення.

Для перетворення з RGB до YUV BT.709 матриці конвертації виглядатимуть так: [29]

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Згідно з [31] саме останній стандарт варто використовувати у сучасних задачах, тому він і буде реалізований. Функції конвертації між RGB та YUV на мові Swift показано у Лістингу 3.7. Процес зводиться до створення матриць з необхідними коефіцієнтами та перемноження їх. Операції ці будуть проводитися дуже швидко, тому що на низькому платформенному рівні реалізована оптимізована версія алгоритму перемноження матриць.

Лістинг 3.7 – Функція конвертації між RGB та YUV

```
func toYpCbCr() -> YpCbCr {
    var yp: [NSNumber] = []
    var cb: [NSNumber] = []
    var cr: [NSNumber] = []
    for yCo in 0 ..< Int(size.height) {
        for xCo in 0 ..< Int(size.width) {
            guard let color = self[xCo, yCo] else { continue
}
            var red: CGFloat = 0
            var green: CGFloat = 0
            var blue: CGFloat = 0
            var alpha: CGFloat = 0
```

```
            color.getRed(&red, green: &green, blue: &blue,
alpha: &alpha)
            let colorMatrix = simd_float3(Float(red),
Float(green), Float(blue))
            let conversionMatrix = simd_float3x3(rows: [
                simd_float3(0.2126, 0.7152, 0.0722),
                simd_float3(-0.09991, -0.33609, 0.436),
                simd_float3(0.615, -0.55861, -0.05639)
            ])
            let res = conversionMatrix * colorMatrix
            yp.append(NSNumber(value: res.x))
            cb.append(NSNumber(value: res.y))
            cr.append(NSNumber(value: res.z))
        }
    }
    return YpCbCr(yp: yp, cb: cb, cr: cr)
}

func toRGB(width: Int, height: Int) -> UIImage? {
    var pixels: [PixelData] = []

    for index in yp.indices {
        let colorMatrix = simd_float3(Float(truncating:
yp[index]), Float(Double(truncating: cb[index])),
Float(Double(truncating: cr[index])))
        let conversionMatrix = simd_float3x3(rows: [
            simd_float3(1.000, 0.000, 1.28033),
            simd_float3(1.000, -0.21482, -0.38059),
            simd_float3(1.000, 2.12798, 0.000)
        ])
```

```
        let res = conversionMatrix * colorMatrix
        pixels.append(PixelData(a: 255, r: UInt8(truncating:
NSNumber(value: res.x * 255)), g: UInt8(truncating:
NSNumber(value: res.y * 255)), b: UInt8(truncating:
NSNumber(value: res.z * 255))))
    }

    return UIImage(pixels: pixels, width: width, height:
height)
}
```

Оскільки у платформі iOS також немає конструктора для створення зображення з окремих пікселів, що був би доступний розробнику, його доведеться написати самому (це конструктор і використовується у кінці лістингу 3.7), він наведений на Лістингу 3.8.

Лістинг 3.8 – Створення зображення для відображення в системі iOS

```
extension UIImage {
    convenience init?(pixels: [PixelData], width: Int,
height: Int) {
        guard width > 0 && height > 0, pixels.count == width
* height else { return nil }
        var data = pixels
        guard let providerRef = CGDataProvider(data:
Data(bytes: &data, count: data.count *
MemoryLayout<PixelData>.size) as CFData)
            else { return nil }
        guard let cgim = CGImage(
            width: width,
            height: height,
```

```
            bitsPerComponent: 8,
            bitsPerPixel: 32,
            bytesPerRow: width *
MemoryLayout<PixelData>.size,
            space: CGColorSpaceCreateDeviceRGB(),
            bitmapInfo: CGBitmapInfo(rawValue:
CGImageAlphaInfo.premultipliedFirst.rawValue),
            provider: providerRef,
            decode: nil,
            shouldInterpolate: true,
            intent: .defaultIntent)
        else { return nil }
        self.init(cgImage: cgim)
    }
```

Наступним кроком буде конвертувати отримане зображення формату YUV у багатовимірний масив, який і буде подано на вхід моделі. У цьому нам допоможе клас `MLMultiArray`, вбудований в Core ML. Це тип колекції машинного навчання, який зберігає числові значення в масиві з кількома вимірами.

Кожен вимір у мультимасиві, як правило, є значущим. Наприклад, модель може мати вхід, який приймає зображення як багатовимірний масив пікселів із трьома вимірами, C x H x W. Перший вимір, C, представляє кількість колірних каналів, а другий і третій виміри, H і W , представляють висоту та ширину зображення відповідно. Кількість вимірів і розмір кожного виміру визначають форму мультимасиву. [32] У нашому випадку це буде 1x100x100. Функцію перетворення зображення YUV до масиву, який буде передано на вхід нейронної мережі, наведено на Лістингу 3.9.

Лістинг 3.9 – Перетворення зображення з YUV до масиву для нейронної мережі

```
func upscalerInput(from imageYpCbCr: YpCbCr) throws ->
(ypInput: imageUpscalingModelInput, cbInput:
imageUpscalingModelInput, crInput: imageUpscalingModelInput)
{
    let ypInputArray = try MLMultiArray(shape:
Constants.coremlInputShape, dataType:
MLMultiArrayDataType.float32)
    let cbInputArray = try MLMultiArray(shape:
Constants.coremlInputShape, dataType:
MLMultiArrayDataType.float32)
    let crInputArray = try MLMultiArray(shape:
Constants.coremlInputShape, dataType:
MLMultiArrayDataType.float32)
    for idx in imageYpCbCr.yp.indices {
        let inputIndex = [NSNumber(value: 0),
NSNumber(value: idx / Constants.inputDimension),
NSNumber(value: idx % Constants.inputDimension),
NSNumber(value: 0)]
        ypInputArray[inputIndex] = imageYpCbCr.yp[idx]
        cbInputArray[inputIndex] = imageYpCbCr.cb[idx]
        crInputArray[inputIndex] = imageYpCbCr.cr[idx]
    }
    return (imageUpscalingModelInput(input_1: ypInputArray),
imageUpscalingModelInput(input_1: cbInputArray),
imageUpscalingModelInput(input_1: crInputArray))
}
```

Після успішного перетворення зображення на масив, потрібно отримати передбачення з моделі для кожного кольорового каналу, та перетворити масив

отриманий від нейронної мережі на зображення, що можна буде продемонструвати користувачеві. Цей процес показано на Лістингу 3.10.

Для досягнення мети потрібно використати клас `imageUpscalingModel`, про який згадувалося вище, а також клас конфігурації `MLModelConfiguration`. Цей клас репрезентує налаштування для створення або оновлення моделі машинного навчання. [34] Конфігурацію моделі використовують, щоб:

– встановити або замінити параметри моделі;

– визначити, який пристрій використовує модель для прогнозування, наприклад графічний процесор;

– обмежити модель використанням певної категорії обчислювальних пристроїв, наприклад ЦП.

Нам підходять налаштування за замовчуванням, де система автоматично вибирає між ГП та нейронним двигуном у залежності від типу моделі.

Лістинг 3.10 – Отримання передсказання з моделі

```
func upscale(image inputImage: UIImage) -> Result<UIImage,
Error>  {
    do {
        let inputYpCbCr = try preProcess(inputImage:
inputImage)
        let input = try upscalerInput(from: inputYpCbCr)
        let outputYp = try
imageUpscalingModel(configuration:
MLModelConfiguration()).prediction(input: input.ypInput)
        let outputCb = try
imageUpscalingModel(configuration:
MLModelConfiguration()).prediction(input: input.cbInput)
        let outputCr = try
imageUpscalingModel(configuration:
MLModelConfiguration()).prediction(input: input.crInput)
```

```
        let outputImageYpCbCr = try
imageYpCbCr(upscaledYpOutput: outputYp, upscaledCbOutput:
outputCb, upscaledCrOutput: outputCr, inputImage:
inputYpCbCr)
        let resultImage = try postProcess(outputYpCbCr:
outputImageYpCbCr, inputImage: inputImage)
        return .success(resultImage)
    } catch {
        return .failure(error)
    }
}
```

3.2.3 Пост-обробка за допомогою алгоритму Unsharp Masking

Як було описано у Розділі 1.2.3, алгоритм нерізного маскування (Unsharp Masking) самостійно є досить слабким для покращення якості зображення, але він дуже корисний при пост-обробці, у чому і полягає суть гібридного підходу, що досліджується у цій дисертації. Для мобільного пристрою таких підхід є особливо оптимальним через те, що такий алгоритм є простим у реалізації та пропонує гарну швидкодію, надаючи можливість спростити нейронну мережу і зменшити навантаження на ту частину системи.

На платформі iOS ви можете додавати ефекти до зображень, застосовуючи фільтри Core Image до об’єктів CIImage. [34] Деталі реалізації показано на Лістингу 3.11.

Обробка CIImage відбувається в об'єкті CIContext. Створення CIContext є дорогим, тому краще створити його під час початкового налаштування та повторно використовуйте його у своїй програмі. Наступним кроком є завантаження зображення для обробки. Об'єкт CIImage сам по собі не є зображенням для відображення, а скоріше даними зображення. Щоб відобразити його, необхідно перетворити його на інший тип, наприклад UIImage.

CIFilter являє собою одну операцію або рецепт для певного ефекту. Щоб обробити об’єкт CIImage, потрібно пропустити його через об’єкти CIFilter. Ви можете зробити підклас CIFilter або черпати з існуючої бібліотеки вбудованих фільтрів. Ми будемо використовувати `CIUnsharpMask`.

Також, одним з етапів пост-обробки є підгонка зображення під оригінальне співвідношення сторін, щоб була можливість використовувати систему з будь-якими зображеннями, а не лише квадратними як часто роблять для технічних демо.

Лістинг 3.11 – Використання фільтру CIUnsharpMask та пост-обробка для правильного співвідношення сторін зображення

```
func unsharpMask() -> UIImage? {
    let context = CIContext()
    let currentFilter = CIFilter(name: "CIUnsharpMask")
    let beginImage = CIImage(image: self)
    currentFilter?.setValue(beginImage, forKey:
kCIInputImageKey)
    guard let image = currentFilter?.outputImage,
          let cgimg = context.createCGImage(image, from:
image.extent) else {
        return nil
    }
    return UIImage(cgImage: cgimg)
}

func postProcess(outputYpCbCr: YpCbCr, inputImage: UIImage)
throws -> UIImage {
    let maxDimension = max(inputImage.size.width,
inputImage.size.height)
    let heightScale = inputImage.size.height / maxDimension
    let widthScale = inputImage.size.width / maxDimension
```

```
    let size = CGSize(width:
CGFloat(Constants.outputDimension) * widthScale, height:
CGFloat(Constants.outputDimension) * heightScale)
    guard let resultImage = outputYpCbCr.toRGB(width:
Constants.outputDimension, height:
Constants.outputDimension)?.resizedImage(with:
size)?.unsharpMask() else {
        throw UpscalerError.postprocessFailure
    }
    return resultImage
}
```

3.3 Графічний інтерфейс користувача та функціональні можливості додатку

Застосунок включає у себе одну основну сторінку, яка надає необхідний функціонал та меню вибору джерела фото (галерея або камера). На рисунку 3.4 зображено графічний інтерфейс користувача додатку с позначеними елементами.

Головна сторінка додатку складається з наступних елементів:

1) кнопка активації меню вибору джерела фото;
2) меню вибору джерела фото;
3) блок відображення початкового зображення;
4) блок відображення результуючого зображення.

Меню дозволяє вибрати джерело фотографії (галерея або камера). При будь-якому виборі можна завантажити лише одне зображення за раз для обробки, для всіх процедур по вибору або створенню фото використовуються нативні компоненти і процедури системи iOS.

Основну частину займають блоки відображення початкового та результуючого зображень в однаковому фізичному розмірі та з однаковим співвідношенням сторін для більш легкого візуального порівняння і оцінки результатів.

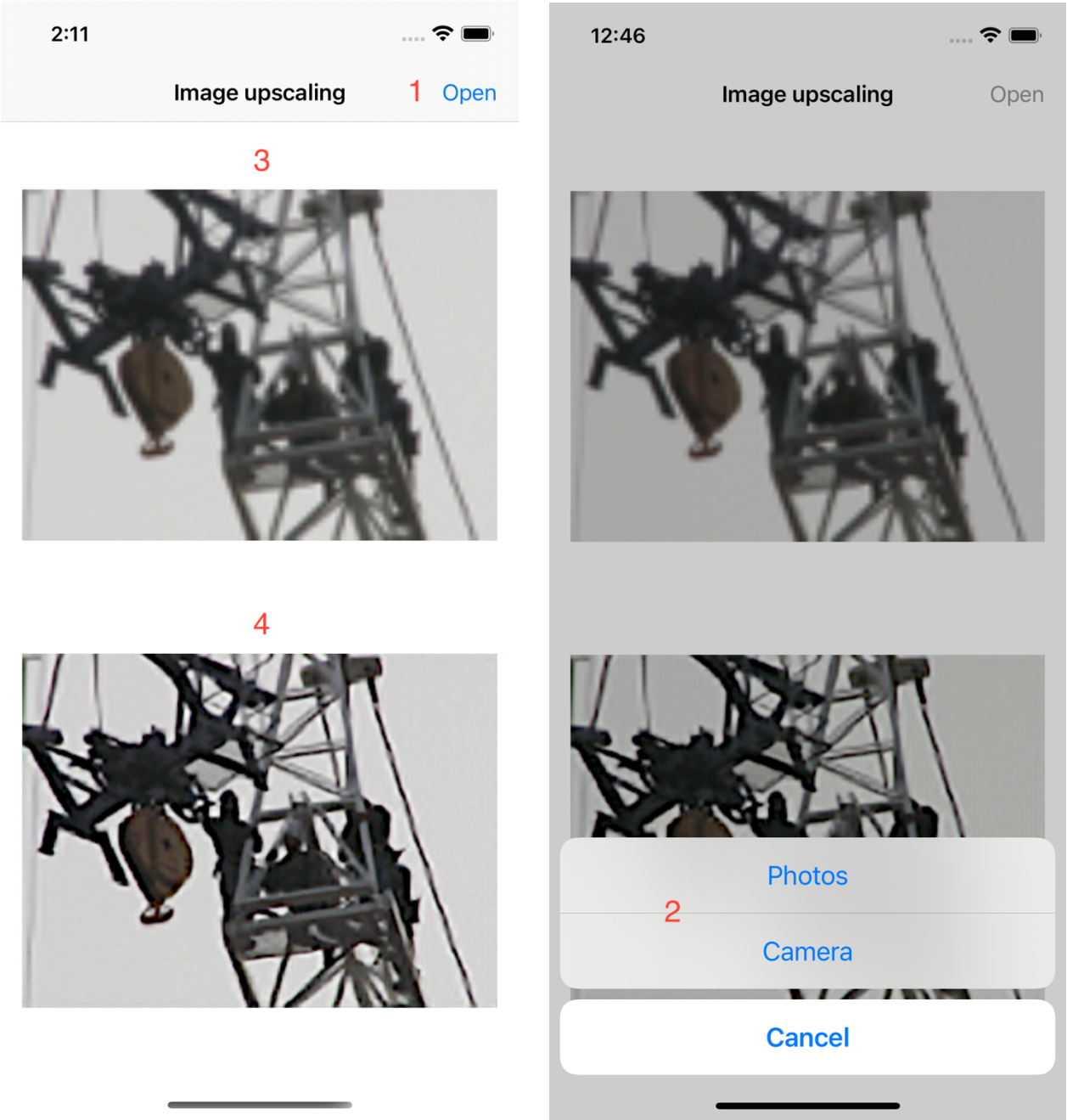

Рисунок 3.4 – Графічний інтерфейс додатку

3.4 Експериментальні дослідження

Навчання було проведено на декількох датасетах, описаних у розділі 2.4. Проте, лише Flickr30k датасет показав добрі результати у контексті вирішуваної задачі.

Також було проведено експерименти з кількістю епох навчання, від 25 до 100. 50 епох є золотою серединою за якістю результатів та швидкістю тренування при швидкій розробці прототипу на звичайному комп’ютері. Це можна побачити на

Рисунку 3.5, де на графіку зображено статистику тренування з помилкою для кожної кількості епох.

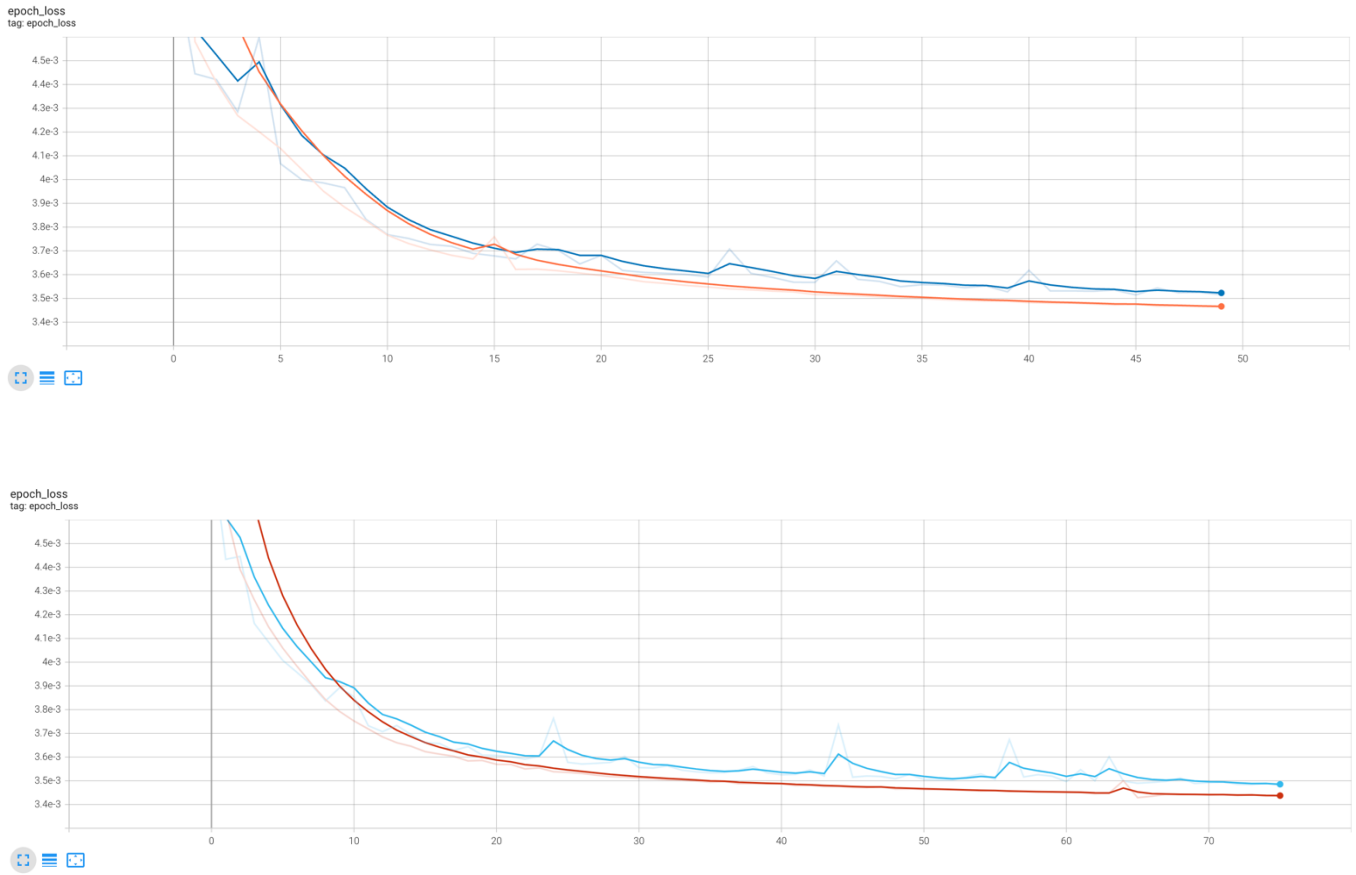

Рисунок 3.5 – Графік результатів тренування

При тренуванні на 50 епох помилка наближається до 3.5e-3, у випадку 3.4e-3, при цілі 1e-3.

Проте, усі подальші результати буде показано при тренування зі 100 епохами, оскільки у реальному використанні тренування проходило б на спеціально оптимізованому сервері для досягнення найкращих результатів.

Для порівняння різних версій фреймворку TensorFlow та оцінки швидкості тренування, було проведено тренування мережі з версією 2.4 alpha3 на Apple M1, 2.4 на AMD Radeon M5300 GPU та Intel Core i7 9750H CPU. Версія 2.4 підтримує графічні процесори на операційній системі macOS, а також використовує Neural Engine та графічні ядра процесорів сімейства Apple M1. Попередні версії можуть

використовувати тільки CPU. Характеристики процесу навчання наведено на Таблиці 3.1.

Таблиця 3.1 – Порівняння навчання на різних системах та версіях фреймворку TensorFlow

Критерій / Конфігурація	**TensorFlow 2.4 alpha3, Apple M1**	**TensorFlow 2.4 alpha3, AMD Radeon M5300 GPU**	**TensorFlow 2.3.1 stable, Intel Core i7 9750H CPU**
Час навчання (год)	5:43	8:37	17:11
Початкова помилка	4.5	4.5	4.5
Кінцева помилка	3.4e-3	3.4e-3	3.4e-3
Оптимальна кількість епох для навчання	100	100	100

Візуальна інспекція результатів підтверджує ці дані (Рисунок 3.6). Результати після тренування у 100 епох мають нижчий показник шуму та чистіші границі об’єктів, але це мінорна різниця.

Навіть у альфа версії фреймворк TensorFlow дуже добре оптимізований як під GPU на macOS, так і під нові процесори лінійки M1, що мають у своєму складі не лише графічний процесор, а й прискорювач для машинного навчання Neural Engine.

Таким чином, у майбутньому, особливо на системах на чіпі Apple швидкодія навчання TensorFlow буде ще кращою ніж на традиційних x86 системах.

Нажаль, фреймворк PyTorch на даний момент не пропонує такої функціональності взагалі, а також розробники не мають планів щодо підтримки цих систем на чіпі.

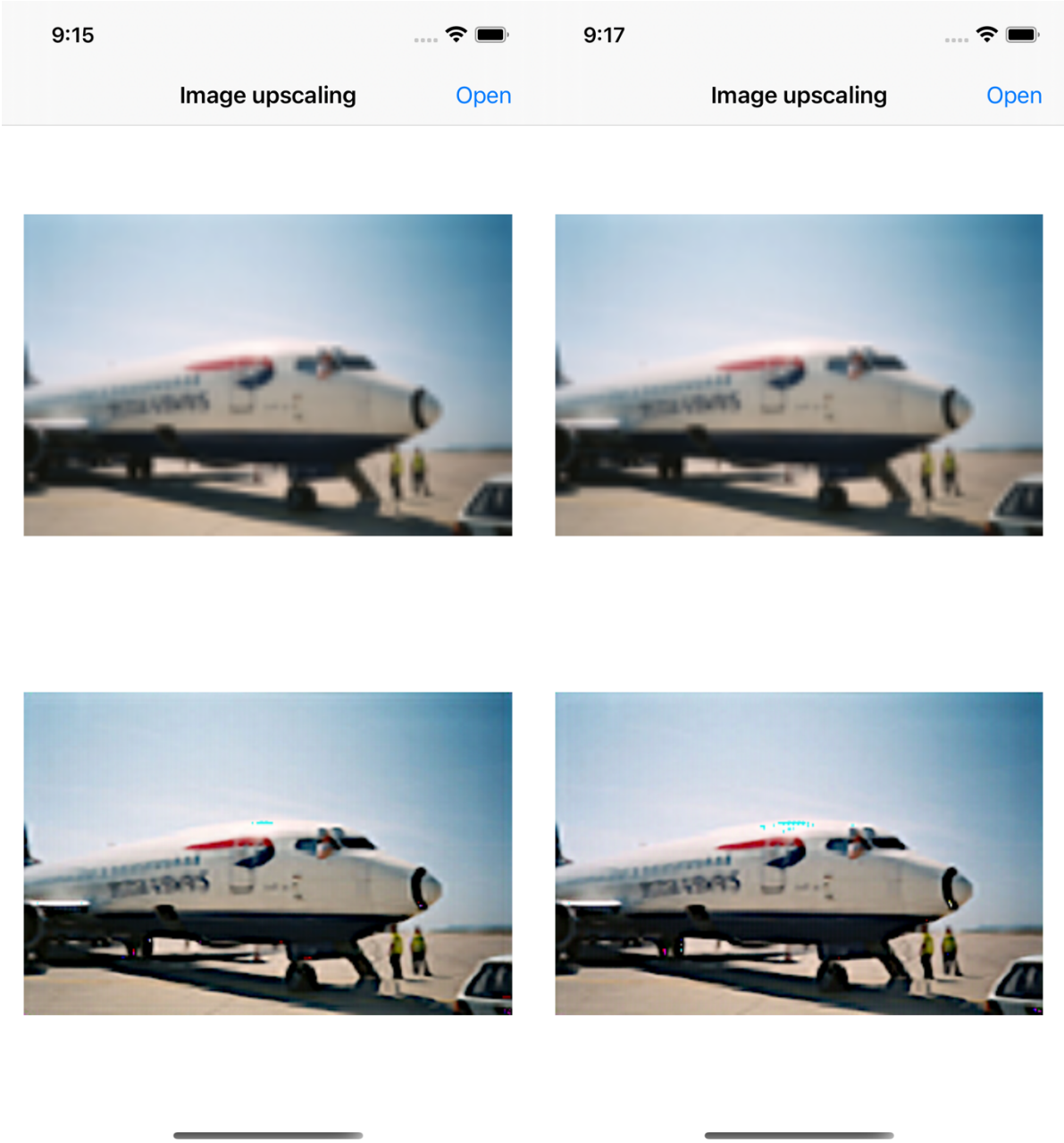

Рисунок 3.6 – Порівняння 50 епох навчання (справа) з 100 епохами (зліва)

Ще одним експериментом було порівняння результату роботи тільки нейронної мережі з гібридним підходом (після пост-обробки Unsharp Masking). Результат можна побачити на Рисунку 3.7, очевидно що гібридний підхід виправдав себе, додаючи контрасту та виділяючи чіткі границі об’єктів.

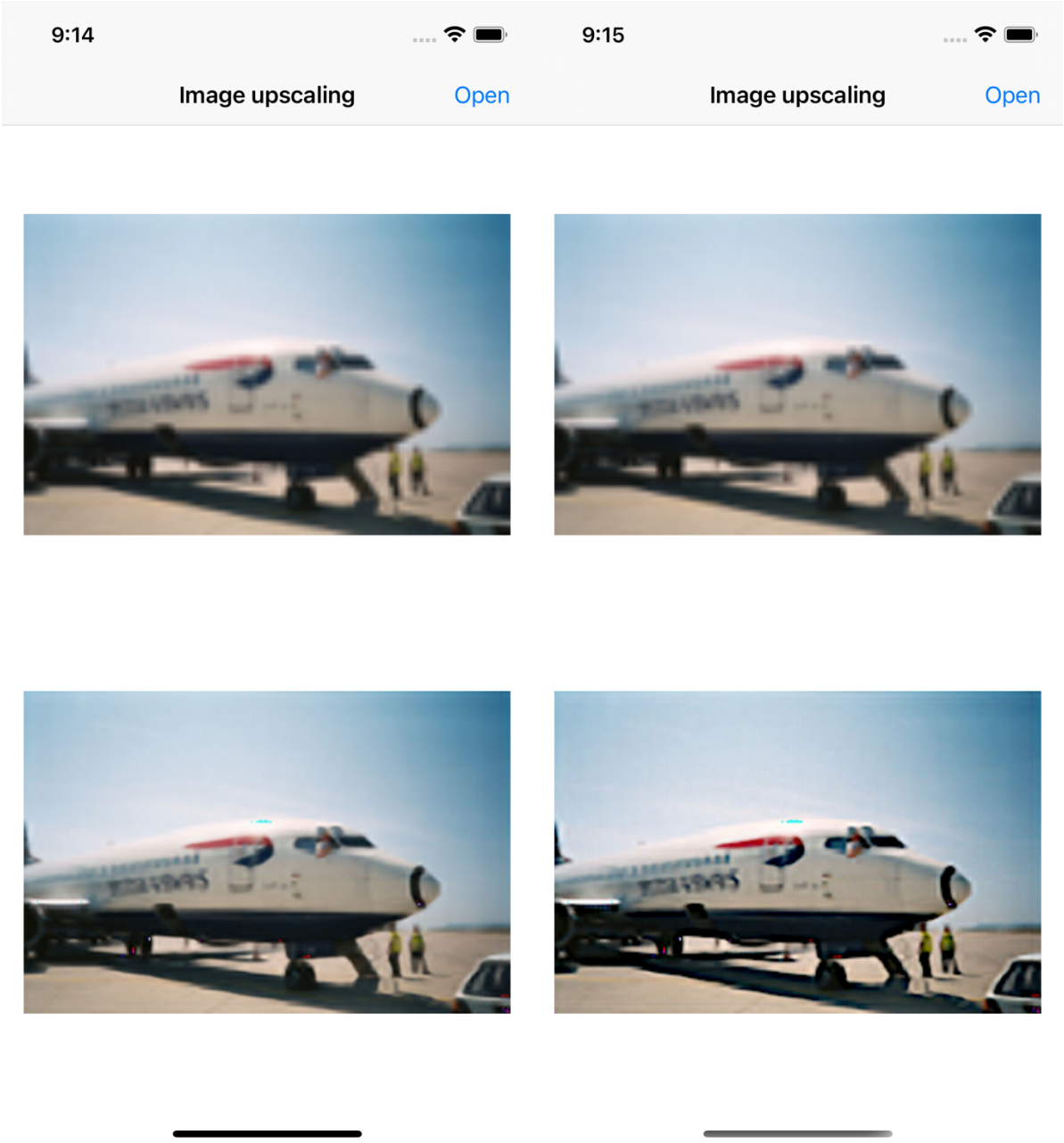

Рисунок 3.7 – Порівняння результату роботи нейронної мережі (справа) з гібридним підходом (зліва)

3.5 Оцінка ефективності запропонованого рішення

3.5.1 Аналіз результатів роботи системи на різних вхідних зображеннях

Після навчання моделі на 30000 зображень з датасету Flickr30k, було проведено ряд досліджень щодо результатів роботи системи як на валідаційних зображеннях з датасету, так і на сторонніх для репрезентації реальних умов.

Використовувався коефіцієнт збільшення 3, тобто зображення розміром 100x100 перетворювались у зображення розміром 300x300 зі збереженням оригінальних пропорцій.

Перше протестоване зображення, з яким можна ознайомитися на Рисунку 3.8, має дуже сильний контраст між об’єктами на ньому, тому система впоралась дуже добре. Також варто зазначити, що це є ідеальною ситуацією для гібридного підходу, оскільки висока контрастність вхідного зображення дає шанс на кращий результат алгоритму Unsharp Masking.

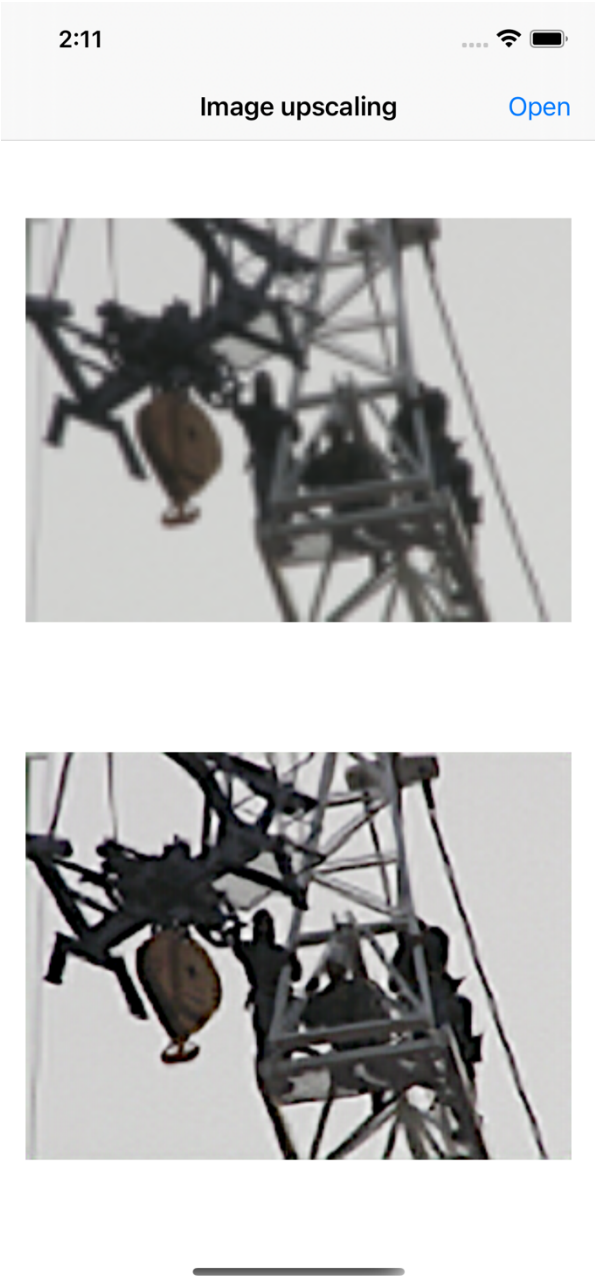

Рисунок 3.8 – Результат роботи системи

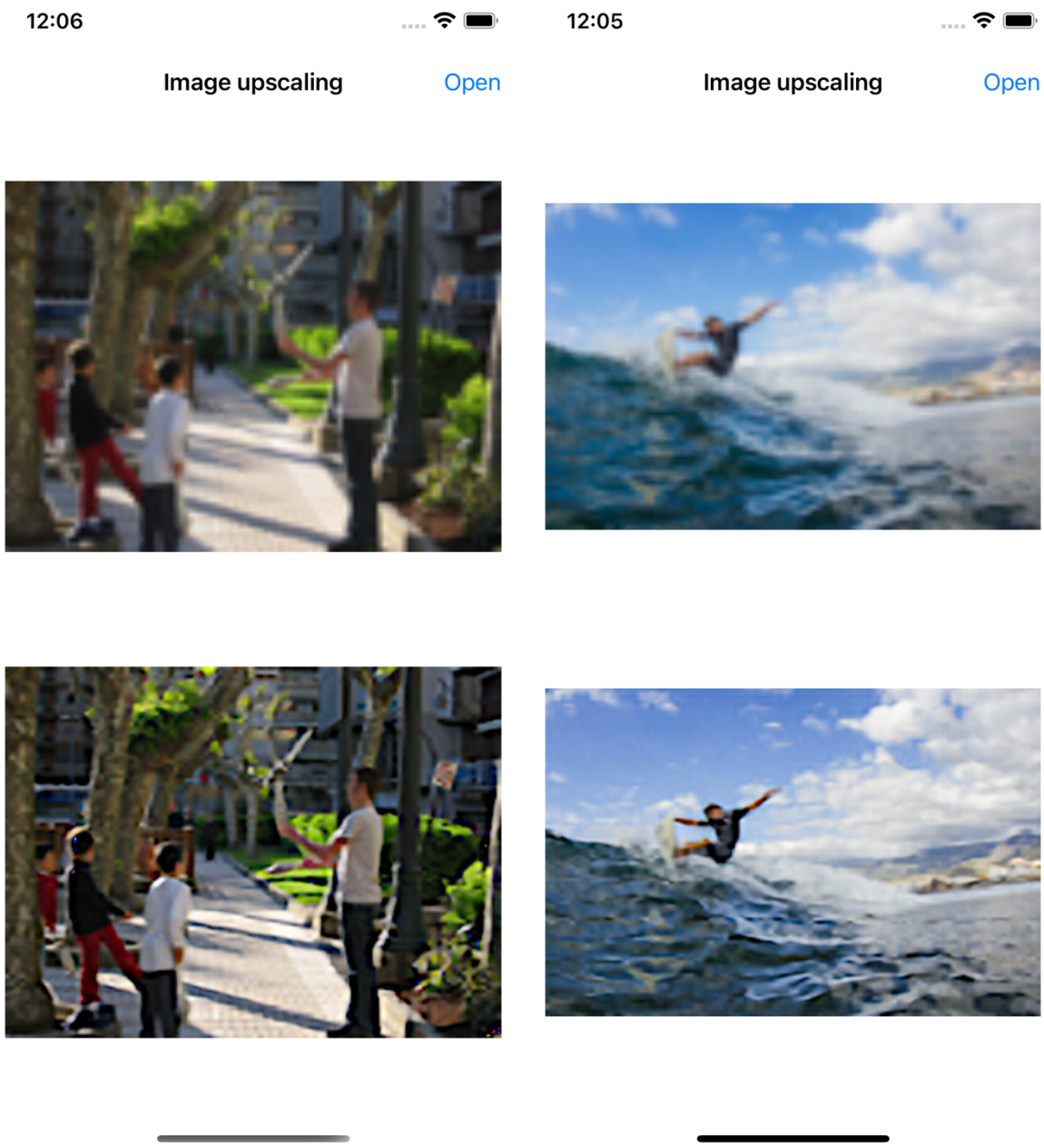

Рисунок 3.9 – Результат роботи системи

На Рисунку 3.9 показано результат роботи системи на зображеннях з більшою кількістю кольорів та сильнішим контрастом. Результат роботи системи у цьому випадку є ще більш вражаючим.

На першому зображенні зокрема видно більш чіткі лінії складних форм навколо людей стали чіткими, а також інформація про колір була збережена та передана добре, без артефактів та інших проблем. Варто звернути увагу на тіні, які зберегли свою текстуру, отримавши вищу роздільну здатність.

На зображенні зліва видно ті ж самі характеристики покращення, на хвилях особливо можна помітити ефект пост-обробки Unsharp Masking, що дає лініям хвиль чіткий контур.

3.5.2 Аналіз швидкодії системи

Тестування швидкодії системи було проведено на пристрої Apple iPhone 13 Pro з операційною системою iOS 15.1. У Таблиці 3.2 можна побачити час на обробку фотографії, приведено час обробки як просто нейронною мережею, так і повним гібридним методом з пост-обробкою.

Таблиця 3.2 – Порівняння швидкодії нейронної мережі та гібридного підходу

Фото / Вид системи	**Нейронна мережа**	**Гібридний підхід**
Фото 1 (монтажні працівники на крані)	313мс	318мс
Фото 2 (парк)	311мс	317мс
Фото 3 (серфер)	312мс	317мс

З цих даних можна зробити висновок, що гібридний підхід лише зменшує швидкодію на 1.5-1.9% в залежності від зображення, тому він є оправданим, враховуючи отримані візуальні результати.

Інший висновок, що випливає з цього дослідження це те, що у поточному стані система не готова до обробки відео у реальному часі, оскільки швидкодія буде на рівні 3-3.5 кадри на секунду, що неприпустимо для такої задачі. Проте, це і не було ціллю дослідження, фокус було встановлено на обробку одиночних зображень.

На більш старих смартфонах швидкодія буде нижчою, але не сильно, оскільки нейронний прискорювач вбудований у всі смартфони компанії Apple за останні 5 років.

3.5.3 Варіанти покращення точності роботи системи

Після аналізу результатів роботи системи можна виділити декілька моментів, які можна покращити, зокрема:

– швидкодія частини нейронної мережі для можливості використання системи для обробки відео у реальному часі;

– більша точність нейронної мережі при роботі з більшими факторами масштабування, наприклад x4 або x5.

Щоб досягти цього, доступні такі кроки:

– провести тренування з більшою кількістю тренувальних зображень. Це дозволить збільшити вибірку можливих сцен та видів об’єктів;

– провести тренування на більшій вхідній роздільній здатності зображень. Це дозволить зібрати більше інформації про колір та його співвідношення на початковому та покращеному зображенні;

– змінити архітектуру системи, врахувавши досвід нейронних мереж, що націлені на обробку у реальному часі та швидкодію загалом;

– оптимізувати процес конвертації у YUV та назад, або уникнути цієї конвертації цілком, оскільки ця операція є доволі «дорогою» з точки зору швидкодії.

Висновки розділу

У цьому розділі було описано реалізацію підходу, описану і Розділі 2, а також зазначено та проаналізовано отримані результати.

У підрозділах описуються процеси та реалізація сортування датасету, створення моделі нейронної мережі, тренування мережі, а також алгоритми пост-обробки та всі допоміжні методи, що допомагають працювати з зображеннями та є необхідними для функціонування системи. Також було проведено аналіз конфігурацій систем для тренування та оптимальних параметрів системи для досягнення прийнятного часу тренування.

Наприкінці розділу проведено детальний візуальний аналіз результатів, визначено виправданість гібридного підходу, а також виміряно швидкодію на реальному пристрої та зроблено висновки щодо потенціального застосування системи та можливих недоліків та покращення роботи системи.

4 СТАРТАП-ПРОЄКТ

4.1 Опис основної ідеї проєкту

У даному розділі проаналізовано та подано зміст ідеї стартап-проєкту, розібрано можливі напрямки застосування, вигоди користувача та порівняно розроблений програмний засіб з існуючими аналогами.

Зміст ідеї – програмні засоби для допомоги у обробці зображень з метою покращення роздільної здатності зображення.

Таблиця 4.1 – Опис ідеї стартап-проєкту

Зміст ідеї стартап-проєкту	**Напрямки застосування**	**Вигоди для користувача**
Покращення роздільної здатності зображення	Мобільний додаток	Дає простий у використанні інструментарій для покращення якості зображення прямо на смартфоні
	Бібліотека для використання в інших програмних продуктах	Є основою додатків для роботи з фото, або основою для інших підходів, алгоритмів, моделей нейронних мереж, тощо

Для порівняння пропонованого стартап-проєкту з пропозиціями конкуруючих засобів було виконано такі кроки:

– визначено техніко-економічні властивості пропонованого стартап-проєкту;

– визначено приблизне коло конкурентів на ринку та визначено техніко-економічні властивості продуктів, які пропонуються;

– виконано порівняльний аналіз видобутих даних та відсортовано відносно сильних, нейтральних та слабких сторін.

Результати виконаної у цьому напрямі роботи подано у Таблиці 4.2.

Таблиця 4.2 – Порівняння характеристик з конкурентними продуктами

	Пропонований проект	**AI Image Enlarger**	**Enlarger – AI Photo Upscale**
Сильна сторона пропонованого проекту	Робота локально на пристрої, фотографія не покидає пристрою і не потрібно чекати обробки на сервері	Обробка повністю на сервері, мобільний додаток просто є клієнтом і не робить нічого для процесу покращення	Обробка повністю на сервері, мобільний додаток просто є клієнтом і не робить нічого для процесу покращення
Нейтральна сторона пропонованого проекту	Відсутність іншої функціональності як фільтрів, тощо	Відсутність іншої функціональності як фільтрів, тощо	Відсутність іншої функціональності як фільтрів, тощо
Слабка сторона пропонованого продукту	На даному етапі обмежено вибором одного множника покращення роздільної здатності	Вибір між 4 множниками покращення роздільної здатності	На даному етапі обмежено вибором одного множника покращення роздільної здатності

4.2 Технологічний аудит ідеї проекту

В даному розділі аудитовано технології, від яких залежить реалізація ідеї проєкту з надання оцінки якості новинних матеріалів.

Проаналізовано такі складові:

– яка технологія є базовою у продукті?
– чи доступна ця технологія, чи вона вимагає її винайдення чи покращення?
– чи легко ці технології доступні?

Результати такого аналізу у застосуванні до пропонованого проєкту наведено у Таблиці 4.3.

Таблиця 4.3 – Технологічна здійсненність

	Ідея пропонованого проєкту	**Технологія, від якої залежить реалізація**	**Наявність технології**	**Доступність технології**
1	Покращення роздільної здатності зображення	Нейронні мережі для покращення роздільної здатності зображення	Наявна, але потребує доробки для ефективного застосування на мобільному пристрої	Доступна
2		Алгоритми пост-обробки для покращення контрасту	Наявна	Доступна
3		Процесори для обробки машинного навчання у мобільному пристрої	Наявна	Доступна

З отриманих результатів технологічного аудиту та аналізу конкурентів пропонованого проекту можемо зробити висновок, що реалізація проекту з технічної точки зору реальна на базі існуючих та доступних технологій.

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Для коректного планування напрямків розвитку проекту з урахуванням поведінки ринку, пропозицій конкуруючих гравців та запитів активних та потенційних споживачів необхідно у тому числі визначити ринкові можливості та ринкові загрози.

Першим етапом у такому аналізі є вияснення об’ємів попиту на пропонований продукт та динаміки ринку. Такий аналіз наведено в Таблиці 4.4.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проєкту

	Показник ринку	**Значення показника ринку**
1	Кількість головних гравців	2
2	Загальний обсяг продаж	Конкуренти є сервісами з підпискою, тому обсяг продаж невідомий
3	Динаміка ринку	Відсутня (ринок ще не є сформованим)
4	Обмеження для виходу на ринок	Відсутня
5	Вимоги до сертифікації або стандартизації	Відсутні

Враховуючи пустоту ринку можемо зробити висновок, що хоч фінансово ринок і є невизначеним і невідомим, він все ж є достатньо привабливим для входження.

Наступним необхідно визначити категорії потенційних клієнтів та означити характерні їм риси і їх вимоги до продукту (Таблиця 4.5).

Таблиця 4.5 – Категорії потенційних клієнтів пропонованого проекту

	Потреба	**Аудиторія**	**Відмінності**	**Вимоги**
1	Продовження життя старого телефону з поганою якістю фото	Технічно необізнані люди, що хочу зекономити гроші	Не готові платити відразу	Простота та доступність інтерфейсу, наочний результат роботи, низька ціна
2	Покращення роздільної здатності нового телефону з метою друку тощо	Технічно обізнані люди, що займаються творчістю за допомогою смартфону	Схильні придбати інструмент, яким будуть коритуватися постійно	Наочний результат роботи

Наступним кроком аналізу ринку є підведення загрожуючих та сприяючих факторів поведінки ринку відносно пропонованого проекту. Результати наявні в Таблиці 4.6.

Таблиця 4.6 – Ринкові фактори

	Фактор	**Вид**	**Зміст**	**Пропоноване рішення**
1	Низький попит, неготовність платити	Загроза	Продукт не є життєво необхідним	Проведення рекламної та інформаційної кампаній
2	Популярність системи серед створювачів контенту	Можливість	Продукт підходить не тільки прямому споживачеві	Доповнення функціоналу для покращення відповідності системи до вимог нових категорій споживачів

Також важливим є знання пропозицій та конкуренції на ринку. Такий аналіз має бути виконаний через призму існуючих видів конкуренцій, та визначити у тому числі шлях просування продукту. Аналіз конкуренції наведено в Таблицях 4.7 та 4.8.

Таблиця 4.7 – Аналіз конкуренції

Особливості конкуренції	**Прояв особливості**	**Вплив на пропонований продукт**
Конкуренція – монополістична, чиста монополія, олігополія	чиста	Просування продукту, укладання договорів пакетних підписок
Рівень конкурентної боротьби – локальний, національний, глобальний	глобальний	Просування продукту по всьому світу
Галузь – внутрішньогалузева, міжгалузева	Внутрішньогалузева	Покращувати точність, зручність, тощо
За товарами – між бажаннями, товарно родова, товарно-видова	Товарно-видова	Покращувати точність, зручність, тощо
Конкурентна перевага – цінова, нецінова	нецінова	Додавати параметри покращення, які відсутні у конкурентів

Продовження таблиці 4.7

Інтенсивність – марочна, немарочна	марочна	Додавання загального функціоналу, недоступного конкурентам

Таблиця 4.8 – Галузева конкуренція, проаналізована за М. Портером.

Складові аналізу	**Прямі конкуренти у галузі**	**Потенційні конкуренти**	**Постачальники**	**Клієнти**	**Товари-замінники**
	AI Image Enlarger	Enlarger – AI Photo Upscale	Концентрація постачальників	Контроль якості	Ціна
Висновки	Може вийти на ринок, якщо буде додана функціональність різних факторів покращення	Може вийти на ринок, якщо покращиться швидкодія	Постачальник и підлаштовуються під ринок	Клієнти обирають якісніші додатки	Обмеження для роботи через товари замінники

На основі аналізу конкуренції, характеристик ідеї проекту, вимог споживачів, ринкового середовища можемо визначити, що продукт є конкурентоспроможним.

4.4 Розробка ринкової програми стартапу

За результатами аналізу цільових груп потенційних споживачів, яким пропонується дана система було обрано ринок індивідуальних користувачів та професіоналів, що створюють контент на смартфоні та сформовано базову стратегію розвитку (Таблиця 4.9).

Таблиця 4.9 – Базова стратегія розвитку

	Обрана альтернатива розвитку проекту	**Стратегія охоплення ринку**	**Ключові конкурентоспроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку**
1	Орієнтація на ринок індивідуальних користувачів	Стратегія концентрованого маркетингу	Технічно необізнані люди, які хочуть продовжити життя смартфона через покращення якості фото, які він робить	Концентрація
2	Орієнтація на ринок медіа-професіоналів та компанії	Стратегія концентрованого маркетингу	Технічно обізнані люди та компанії, які використовуватимуть продукт для заробляння грошей	Вертикальна інтеграція

Наступним кроком виберемо стратегію конкурентної поведінки на ринку (Таблиця 4.10).

Таблиця 4.10 – Стратегія конкурентної поведінки

Чи є пропонований проект «першопрохідцем» на ринку?	Чи буде дана компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде дана компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Ні	Шукати нових споживачів	Працювати над оптимізацією роботи за допомогою впровадження централізованої бази даних та додавання аналітичних метрик, які дозволять прийняти рішення раніше	Стратегія заняття конкурентної ніші

Остаточним кроком розробки ринкової стратегії пропонованого проекту є стратегія позиціонування, яка полягає у формуванні ринкової позиції, яка є для

проекту є ідентифікуючою серед користувачів. Стратегія позиціонування зазначена у таблиці 4.11.

Таблиця 4.11 – Стратегія позиціонування

Вимоги до товару від цільової аудиторії	**Базова стратегія розвитку**	**Ключові конкурентоспроможні позиції стартап- проекту**	**Вибір асоціацій, які мають сформувати комплексну позицію власного проекту**
Динаміччність та гнучкість роботи. Легка інтеграція (для користувачів бібліотеки). Легкість використання.	Концентрація	Технологічно освідченні люди потребують доступного способу покращення фотографій на смартфоні.	Легкість використання.

4.5 Розроблення маркетингової програми проекту

У таблиці 4.12 наведено попередній аналіз конкурентоспроможності пропонованого продукту на ринку.

Далі розроблена трирівнева маркетингова модель продукту, наведена у таблиці 4.13.

Таблиця 4.12 – Ключові переваги концепції потенційного товару

	Потреба	**Вигода**	**Перевага (існуюча або майбутня)**
1	Динамічна робота з будь-якими зображеннями	Надійність продукту	Основний конкурент обмежений у наборі фотографій до сцен з людьми
2	Робота на смартфоні локально	Швидкість роботи, можливість обробки фото оффлайн	Основні конкуренти потребують підключення до сервера для роботи

Таблиця 4.13 – Трирівнева модель продукту

Рівні продукту			
I. Товар за задумом	Покращення роздільної здатності зображення локально на смартфоні		
II. Товар у реальному виконанні	Властивість/характеристика	М/Нм	Вр/Тх/Тл/Ор
	Можливість оптимізації витрат часу	М	Тл
	Можливість оптимізації ергономіки	Нм	Е
II. Товар у реальному виконанні	Властивість/характеристика	М/Нм	Вр/Тх/Тл/Ор
II. Товар у реальному виконанні	Відповідність актуальним технологіям	М	Тх
	Відповідає вимогам ДСТУ ISO/IEC 25030:2015 Програмна інженерія. Вимоги щодо якості та оцінювання програмного продукту (SQuaRE).		
	Пакування: публікація в App Store		
II. Товар у реальному виконанні	Марка: CoreML Super Resolution		
III. Товар з підкріпленням	Потенційний користувач може ознайомитись з роботою системи використовуючи випробувальний період у 30 днів.		
Захист від копіювання	Випуск у Apple App Store, копіювання неможливе		

Розшифровка скорочень таблиці 4.13:

– М/Нм – монотонні або немонотонні;

– Вр/Тх/Тл/Е/Ор – вартісні, технічні, технологічні, ергономічні або органолептичні(останній – для продуктів харчування);

– захист інтелектуальної власності продукту є важливим моментом публікування та комерційності діяльності. Такий захист може бути досягнуто за допомогою DRM, захисту ідеї, ноу-хау, патентування, тощо.

Наступним етапом є визначення системи збуту, зазначене у таблиці 4.14.

Таблиця 4.14 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	**Функції збуту, які має виконувати постачальник товару**	**Глибина каналу збуту**	**Оптимальна система збуту**
Певний початковий період користування продуктом має бути безкоштовним	Легкість у сплаті послуг, легкість у встановленні	Розробник – магазин – користувач	Проведення збуту через посередника необхідне, оскільки додаток є залежним від системи розповсюдження платформи

Наступним та останнім кроком у розробці маркетингової програми є концепція маркетингових комунікацій, спираючись на визначену специфічність планованих клієнтів та основу позиціонування.

Таблиця 4.15 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	**Канали комунікацій, що використовуют ь цільові клієнти**	**Ключові позиції, обрані для позиціонуванн я**	**Завдання рекламного повідомленн я**	**Концепція рекламног о звернення**
Підписуються на програмний продукт	Інтернет	Робота з будь-якими фотографіями	Довести, що додаток надасть очікуване покращення якості фото	-

Висновки до розділу

Пропонований продукт є конкурентоздатним, оскільки має істотні переваги над існуючими конкурентами. Було визначено шляхи для подальшого розвитку, шляхи збуту та описано маркетингову стратегію. Основною цільовою аудидторією було обрано людей, які хочуть продовжити життя свого смартфона за рахунок покращення якості результатів фото з камери.

ВИСНОВКИ

Було проаналізовано проблему покращення роздільної якості зображення загалом, а також існуючі підходи до вирішення цієї проблеми, описані у наукових роботах та науково-популярних публікаціях.

Було проаналізовано стан використання машинного навчання на мобільних пристроях, додатки та сервіси, що можуть бути конкурентоспроможними у вирішенні поставленої задачі, та визначено, що ця технологія ще недостатньо вживана у реальних додатках, проте має величезний потенціал через розповсюдженість прискорювачів для машинного навчання у сучасних смартфонах.

Було проаналізовано технології, що є доступними у сфері мобільних пристроїв та можуть використовуватися для вирішення задачі дослідження, зокрема фреймворки машинного навчання та клієнтські фреймворки для взаємодії з моделями нейронних мереж на мобільних пристроях.

Було розроблено гібридний підхід до обробки зображень з метою покращення роздільної здатності зображень, що включає в себе методи нейронних мереж та машинного навчання, а також пост-обробку простими традиційними алгоритмами для додаткового покращення результату.

На прикладі розробленого підходу та тестового додатку для системи iOS було доведено, що прості у плані обчислювальної складності гібридні алгоритми є конкурентоспроможними з існуючими традиційними алгоритмами та існуючими нейронними мережами, направленими на покращення роздільної здатності зображень, особливо у контексті менш потужних мобільних пристроїв.

На прикладі розробленого додатку було доведено, що створений підхід здатний до повноцінної роботи з будь-яким одиночним зображенням, при цьому демонструючи високу швидкодію, проте у поточному вигляді не підходить для обробки відео у реальному часі, що не було метою дослідження, проте мало місце як експеримент у межах дослідження. Також було показано, що гібридна складова алгоритму не сповільнює виконання алгоритму значним чином, тобто у контексті

обмежених ресурсів мобільного пристрою у реальному житті цей підхід є обґрунтованим.

Опис алгоритму та архітектури нейронної мережі було опубліковано в статті «Гібридний метод обробки зображень на конволюційних нейронних мережах» в 38 номері (2021 рік) друкованого видання «Адаптивні Системи Автоматичного Управління». [1]

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Федоряка М. Г. Гібридний метод обробки зображень на конволюційних нейронних мережах / М. Г. Федоряка, К. Ю. Мелкумян. // Друковане видання «Адаптивні Системи Автоматичного Управління». Міжвідомчий науково-технічний збірник. – 2021. – №38. – С. 72–76.

2) Anna Díez Manjarrés. Edge-Preserving Image Upscaling : дис. канд. техн. наук / Anna Díez Manjarrés. – Irvine, California, 2009. – 70 с.

3) Archana J.N. A Review on the Image Sharpening Algorithms Using Unsharp Masking / Archana J.N, Aishwarya .P. // International Journal of Engineering Science and Computing,. – 2016. – №7.

4) L. Ying. A wavelet based image sharpening algorithm / L. Ying, N.T Ming, L.B Keat // International Conference on Computer Science and Software Engineering (CSSE 2008) / L. Ying, N.T Ming, L.B Keat. – Wuhen, Hubei, China, 2008. – С. 1053–1056.

5) Salonika Kansaall. Image Sharpening using Unsharp Masking and Wavelet Transform / Salonika Kansaall, Gurpreet Kaur. // International Journal of Advance Research in Computer Science and Management Studies. – 2014. – №2.

6) F. Russo, Image Enhancement Technique combining Sharpening and Noise Reduction // IEEE Transactions on Instrumentation and Measurement, – 2002, –Volume 51, С. 824-828.

7) C. A. Glasbey. A review of image-warping methods / C. A. Glasbey, K. V. Mardia. // Journal of Applied Statistics. – 1998. – №25. – С. 155–171.

8) Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network / Christian Ledig, Lucas Theis, Ferenc Huszar та ін.]. // Cornell Univercity, Computer Vision and Pattern Recognition. – 2016. – №1.

9) Second-order Attention Network for Single Image Super-Resolution / Tao Dai, Jianrui Cai, Yongbing Zhang та ін.]. // ECCV. – 2014. – №4. – С. 372–386.

10) Jin Yamanaka. Fast and Accurate Image Super Resolution by Deep CNN with Skip Connection and Network in Network / Jin Yamanaka, Shigesumi Kuwashima, Takio Kurita. // Hiroshima University, Hiroshima. – 2017.

11) Deconvolutional Networks / Zeiler, M.D., Krishnan, D., Taylor, G.W. // Computer Vision and Pattern Recognition, – 2010. С. 2528–2535.

12) Watson A. Deep Learning Techniques for Super-Resolution in Video Games / Alexander Watson. // Department of Computing and Informatics, Bournemouth University. – 2012.

13) Neural Supersampling for Real-time Rendering / Lei Xiao, Salah Nouri, Matt Chapman та ін.]. // ACM Trans. Graph.. – 2020. – №39. – С. 142:1–12.

14) PanNet: A deep network architecture for pan-sharpening / Junfeng Yang, Xueyang Fu, Yuwen Hu та ін.]. // IEEE International Conference on Computer Vision. – 2017. – С. 1753–1761.

15) Apple's Core ML 2 vs. Google's ML Kit: What's the difference? [Електронний ресурс] // Venture Beat. – 2018. – Режим доступу до ресурсу: https://venturebeat.com/2018/06/05/apples-core-ml-2-vs-googles-ml-kit-whats-the-difference/.

16) ML KIT ON IOS AND HOW IT PERFORMS AGAINST CORE ML [Електронний ресурс] // Xmartlabs. – 2018. – Режим доступу до ресурсу: https://blog.xmartlabs.com/2018/06/21/ML-Kit-CoreML/.

17) PyTorch vs TensorFlow — spotting the difference [Електронний ресурс] // Towards Data Science. – 2017. – Режим доступу до ресурсу: https://realpython.com/pytorch-vs-tensorflow/#who-uses-pytorch.

18) PyTorch vs TensorFlow for Your Python Deep Learning Project [Електронний ресурс] // Real Python. – 2021. – Режим доступу до ресурсу: https://realpython.com/pytorch-vs-tensorflow/.

19) 2020 Stack Overflow Developer Survey [Електронний ресурс] // Stack Overflow. – 2020. – Режим доступу до ресурсу: https://insights.stackoverflow.com/survey/2020#technology-other-frameworks-libraries-and-tools-professional-developers3.

20) Dual Temporal Scale Convolutional Neural Network for Micro-Expression Recognition / Min Peng, Chongyang Wang, Tong Chen та ін.]. // Frontiers in Psychology. – 2017.

21) Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [Електронний ресурс] / Sumit Saha // Towards Data Science. – 2018. – Режим доступу до ресурсу: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

22) Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network / Wenzhe Shi, Jose Caballero, Ferenc Huszár та ін.]. // Cornell Univercity, Computer Vision and Pattern Recognition. – 2016. – №1.

23) Core ML Tools [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/apple/coremltools.

24) BSDS500 (Berkeley Segmentation Dataset 500) [Електронний ресурс] // Papers with code – Режим доступу до ресурсу: https://paperswithcode.com/dataset/bsds500#:~:text=Berkeley%20Segmentation%20Data%20Set%20500%20(BSDS500)%20is%20a%20standard%20benchmark,interior%20boundaries%20and%20background%20boundaries..

25) Urban100 [Електронний ресурс] // Papers with code – Режим доступу до ресурсу: https://paperswithcode.com/dataset/urban100.

26) DIV2K dataset: DIVerse 2K resolution high quality images as used for the challenges [Електронний ресурс] – Режим доступу до ресурсу: https://data.vision.ee.ethz.ch/cvl/DIV2K/.

27) Intel Image Classification [Електронний ресурс] // Kaggle – Режим доступу до ресурсу: https://www.kaggle.com/puneet6060/intel-image-classification.

28) Flickr30k Dataset [Електронний ресурс] // Kaggle – Режим доступу до ресурсу: https://www.kaggle.com/nunenuh/flickr30k.

29) BT.470 : Conventional television systems [Електронний ресурс] // International Communications Union. – 1998. – Режим доступу до ресурсу: https://www.itu.int/rec/R-REC-BT.470-6-199811-S/en.

30) World Analogue Television Standards and Waveforms [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://web.archive.org/web/20190227232952/http://www.radios-tv.co.uk/Pembers/World-TV-Standards/Colour-Standards.html#SECAM.

31) YUV Video Subtypes [Електронний ресурс] // Microsoft. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/windows/win32/directshow/yuv-video-subtypes?redirectedfrom=MSDN.

32) MLMultiArray [Електронний ресурс] // Apple. – 2017. – Режим доступу до ресурсу: https://developer.apple.com/documentation/coreml/mlmultiarray.

33) MLModelConfiguration [Електронний ресурс] // Apple. – 2018. – Режим доступу до ресурсу: https://developer.apple.com/documentation/coreml/mlmodelconfiguration.

34) Processing an Image Using Built-in Filters [Електронний ресурс] // Apple – Режим доступу до ресурсу: https://developer.apple.com/documentation/coreimage/processing_an_image_using_built-in_filters.

ДОДАТОК А

Код розробленого програмного забезпечення

Частина інтерфейсу користувача, що відповідає за вибір зображень

```
import UIKit
import PhotosUI

final class ImagePicker: ImageProvider {
    private var completion: Completion?

    func image(with completion: @escaping Completion) {
        guard self.completion == nil else {
            completion(.failure(ImageProviderError.startFailure))
            return
        }
        self.completion = completion

        var configuration = PHPickerConfiguration()
        configuration.filter = .images
        configuration.selectionLimit = 1

        let picker = PHPickerViewController(configuration:
configuration)
        picker.delegate = self
        UIViewController.topmostViewContoller?.present(picker,
animated: true)
    }
}

extension ImagePicker: PHPickerViewControllerDelegate {
    func picker(_ picker: PHPickerViewController, didFinishPicking
results: [PHPickerResult]) {
        guard let itemProvider = results.first?.itemProvider else {
            self.completion = nil
            picker.dismiss(animated: true)
            return
        }
        image(from: itemProvider) { result in
            picker.dismiss(animated: true) {
                self.completion?(result)
                self.completion = nil
            }
        }
    }
```

```
    private func image(from itemProvider: NSItemProvider, completion:
@escaping (Result<UIImage, Error>) -> Void) {
        guard itemProvider.canLoadObject(ofClass: UIImage.self) else {
            completion(.failure(ImageProviderError.internalError))
            return
        }

        itemProvider.loadObject(ofClass: UIImage.self) { image, error in
            if let image = image as? UIImage {
                DispatchQueue.main.async { completion(.success(image))
}
            } else {
                DispatchQueue.main.async                              {
completion(.failure(ImageProviderError.internalError)) }
            }
        }
    }
}
```

Головний елемент архітектури – процесор та препроцесор зображень

```
import Foundation
import CoreML
import UIKit
import CoreImage

enum UpscalerError: Swift.Error {
    case preprocessFailure
    case postprocessFailure
}

class ImageUpscaler {
    struct Constants {
        static let inputDimension = 100
        static let scaleFactor = 3
        static let outputDimension = inputDimension * scaleFactor
        static let inputSize = CGSize(width: inputDimension, height:
inputDimension)
        static let outputSize = CGSize(width: outputDimension, height:
outputDimension)
        static let coremlInputShape = [1, NSNumber(value:
Constants.inputDimension), NSNumber(value: Constants.inputDimension),
1]
    }
```

```
    func upscale(image inputImage: UIImage, completion: @escaping (Result<UIImage, Error>) -> Void) {
        DispatchQueue.global().async {
            let result = self.upscale(image: inputImage)
            DispatchQueue.main.async { completion(result) }
        }
    }
}

private extension ImageUpscaler {
    func upscale(image inputImage: UIImage) -> Result<UIImage, Error> {
        do {
            let inputYpCbCr = try preProcess(inputImage: inputImage)
            let input = try upscalerInput(from: inputYpCbCr)
            let outputYp = try imageUpscalingModel(configuration: MLModelConfiguration()).prediction(input: input.ypInput)
            let outputCb = try imageUpscalingModel(configuration: MLModelConfiguration()).prediction(input: input.cbInput)
            let outputCr = try imageUpscalingModel(configuration: MLModelConfiguration()).prediction(input: input.crInput)
            let outputImageYpCbCr = try imageYpCbCr(upscaledYpOutput: outputYp, upscaledCbOutput: outputCb, upscaledCrOutput: outputCr, inputImage: inputYpCbCr)
            let resultImage = try postProcess(outputYpCbCr: outputImageYpCbCr, inputImage: inputImage)
            return .success(resultImage)
        } catch {
            return .failure(error)
        }
    }

    func preProcess(inputImage: UIImage) throws -> YpCbCr {
        guard let converted = inputImage.resizedImage(with: Constants.inputSize)?.toYpCbCr() else {
            throw UpscalerError.preprocessFailure
        }
        return converted
    }

    func upscalerInput(from imageYpCbCr: YpCbCr) throws -> (ypInput: imageUpscalingModelInput, cbInput: imageUpscalingModelInput, crInput: imageUpscalingModelInput) {
        let ypInputArray = try MLMultiArray(shape: Constants.coremlInputShape, dataType: MLMultiArrayDataType.float32)
        let cbInputArray = try MLMultiArray(shape: Constants.coremlInputShape, dataType: MLMultiArrayDataType.float32)
```

```
        let crInputArray = try MLMultiArray(shape:
Constants.coremlInputShape, dataType: MLMultiArrayDataType.float32)
        for idx in imageYpCbCr.yp.indices {
            let inputIndex = [NSNumber(value: 0), NSNumber(value: idx /
Constants.inputDimension), NSNumber(value: idx %
Constants.inputDimension), NSNumber(value: 0)]
            ypInputArray[inputIndex] = imageYpCbCr.yp[idx]
            cbInputArray[inputIndex] = imageYpCbCr.cb[idx]
            crInputArray[inputIndex] = imageYpCbCr.cr[idx]
        }
        return (imageUpscalingModelInput(input_1: ypInputArray),
imageUpscalingModelInput(input_1: cbInputArray),
imageUpscalingModelInput(input_1: crInputArray))
    }

    func imageYpCbCr(upscaledYpOutput: imageUpscalingModelOutput,
upscaledCbOutput: imageUpscalingModelOutput, upscaledCrOutput:
imageUpscalingModelOutput, inputImage: YpCbCr) throws -> YpCbCr {
        var upscaledYp: [NSNumber] = []
        var upscaledCb: [NSNumber] = []
        var upscaledCr: [NSNumber] = []
        for idx in 0..<Constants.outputDimension *
Constants.outputDimension {
            let outputIndex = [NSNumber(value: 0), NSNumber(value: idx
/ Constants.outputDimension), NSNumber(value: idx %
Constants.outputDimension), NSNumber(value: 0)]
            upscaledYp.append(NSNumber(value:
upscaledYpOutput.Identity[outputIndex].doubleValue))
            upscaledCb.append(NSNumber(value:
upscaledCbOutput.Identity[outputIndex].doubleValue))
            upscaledCr.append(NSNumber(value:
upscaledCrOutput.Identity[outputIndex].doubleValue))
        }
        return YpCbCr(yp: upscaledYp, cb: upscaledCb, cr: upscaledCr)
    }

    func postProcess(outputYpCbCr: YpCbCr, inputImage: UIImage) throws
-> UIImage {
        let maxDimension = max(inputImage.size.width,
inputImage.size.height)
        let heightScale = inputImage.size.height / maxDimension
        let widthScale = inputImage.size.width / maxDimension
        let size = CGSize(width: CGFloat(Constants.outputDimension) *
widthScale, height: CGFloat(Constants.outputDimension) * heightScale)
        guard let resultImage = outputYpCbCr.toRGB(width:
Constants.outputDimension, height:
```

```
Constants.outputDimension)?.resizedImage(with:     size)?.unsharpMask()
else {
            throw UpscalerError.postprocessFailure
        }
        return resultImage
    }
}
```

Конвертер форматів зображень

```
import UIKit
import simd

public struct PixelData {
    var a: UInt8
    var r: UInt8
    var g: UInt8
    var b: UInt8
}

struct YpCbCr {
    let yp, cb, cr: [NSNumber]

    func toRGB(width: Int, height: Int) -> UIImage? {
        var pixels: [PixelData] = []

        for index in yp.indices {
            let colorMatrix = simd_float3(Float(truncating: yp[index]),
Float(Double(truncating:     cb[index])),     Float(Double(truncating:
cr[index])))
            let conversionMatrix = simd_float3x3(rows: [
                simd_float3(1.000, 0.000, 1.28033),
                simd_float3(1.000, -0.21482, -0.38059),
                simd_float3(1.000, 2.12798, 0.000)
            ])
            let res = conversionMatrix * colorMatrix
            pixels.append(PixelData(a:   255,   r:   UInt8(truncating:
NSNumber(value:  res.x  *  255)),  g:  UInt8(truncating:  NSNumber(value:
res.y * 255)), b: UInt8(truncating: NSNumber(value: res.z * 255))))
        }

        return UIImage(pixels: pixels, width: width, height: height)
    }

extension UIImage {
    convenience init?(pixels: [PixelData], width: Int, height: Int) {
```

```
        guard width > 0 && height > 0, pixels.count == width * height
else { return nil }
        var data = pixels
        guard let providerRef = CGDataProvider(data: Data(bytes: &data,
count: data.count * MemoryLayout<PixelData>.size) as CFData)
            else { return nil }
        guard let cgim = CGImage(
            width: width,
            height: height,
            bitsPerComponent: 8,
            bitsPerPixel: 32,
            bytesPerRow: width * MemoryLayout<PixelData>.size,
            space: CGColorSpaceCreateDeviceRGB(),
            bitmapInfo:                          CGBitmapInfo(rawValue:
CGImageAlphaInfo.premultipliedFirst.rawValue),
            provider: providerRef,
            decode: nil,
            shouldInterpolate: true,
            intent: .defaultIntent)
        else { return nil }
        self.init(cgImage: cgim)
    }

    func resizedImage(with size: CGSize) -> UIImage? {
        guard let image = cgImage else { return nil }

        if (image.colorSpace?.model == .rgb) {
            let bytesPerPixel = 4;
            let bytesPerRow = bytesPerPixel * Int(size.width);
            let bitsPerComponent = 8;
            let context = CGContext(data: nil,
                                    width: Int(size.width),
                                    height: Int(size.height),
                                    bitsPerComponent: bitsPerComponent,
                                    bytesPerRow: bytesPerRow,
                                    space: image.colorSpace!,
                                    bitmapInfo:
image.bitmapInfo.rawValue)
            context?.interpolationQuality = .high
            context?.draw(image, in: CGRect(origin: .zero, size: size))

            guard let scaledImage = context?.makeImage() else { return
nil }

            return UIImage(cgImage: scaledImage)
        } else if (image.colorSpace?.model == .monochrome) {
```

```
            let context = CGContext(data: nil,
                                    width: Int(size.width),
                                    height: Int(size.height),
                                    bitsPerComponent:
image.bitsPerComponent,
                                    bytesPerRow: Int(size.width),
                                    space: image.colorSpace!,
                                    bitmapInfo:
image.bitmapInfo.rawValue)
            context?.interpolationQuality = .high
            context?.draw(image, in: CGRect(origin: .zero, size: size))

            guard let scaledImage = context?.makeImage() else { return
nil }

            return UIImage(cgImage: scaledImage)
        } else {
            assertionFailure("The colorspace is not supported yet!")
            return nil
        }
    }

    subscript (x: Int, y: Int) -> UIColor? {
        guard x >= 0 && x < Int(size.width) && y >= 0 && y <
Int(size.height),
            let cgImage = cgImage,
            let provider = cgImage.dataProvider,
            let providerData = provider.data,
            let data = CFDataGetBytePtr(providerData) else {
            return nil
        }

        let numberOfComponents = 4
        let pixelData = ((Int(size.width) * y) + x) * numberOfComponents

        let r = CGFloat(data[pixelData]) / 255.0
        let g = CGFloat(data[pixelData + 1]) / 255.0
        let b = CGFloat(data[pixelData + 2]) / 255.0
        let a = CGFloat(data[pixelData + 3]) / 255.0

        return UIColor(red: r, green: g, blue: b, alpha: a)
    }

    func toYpCbCr() -> YpCbCr {
        var yp: [NSNumber] = []
        var cb: [NSNumber] = []
```

```
        var cr: [NSNumber] = []
        for yCo in 0 ..< Int(size.height) {
            for xCo in 0 ..< Int(size.width) {
                guard let color = self[xCo, yCo] else { continue }
                var red: CGFloat = 0
                var green: CGFloat = 0
                var blue: CGFloat = 0
                var alpha: CGFloat = 0
                color.getRed(&red, green: &green, blue: &blue, alpha:
&alpha)
                let colorMatrix = simd_float3(Float(red), Float(green),
Float(blue))
                let conversionMatrix = simd_float3x3(rows: [
                    simd_float3(0.2126, 0.7152, 0.0722),
                    simd_float3(-0.09991, -0.33609, 0.436),
                    simd_float3(0.615, -0.55861, -0.05639)
                ])
                let res = conversionMatrix * colorMatrix
                yp.append(NSNumber(value: res.x))
                cb.append(NSNumber(value: res.y))
                cr.append(NSNumber(value: res.z))
            }
        }
        return YpCbCr(yp: yp, cb: cb, cr: cr)
    }
```

Алгоритм пост-обробки зображень Unsharp Masking

```
    func unsharpMask() -> UIImage? {
        let context = CIContext()
        let currentFilter = CIFilter(name: "CIUnsharpMask")
        let beginImage = CIImage(image: self)
        currentFilter?.setValue(beginImage, forKey: kCIInputImageKey)
        guard let image = currentFilter?.outputImage,
              let    cgimg    =    context.createCGImage(image,    from:
image.extent) else {
            return nil
        }
        return UIImage(cgImage: cgimg)
    }
}
```

Модель головного екрану додатку

```
import Foundation
import UIKit

protocol ViewModelInput {
    func pickImage()
    func getCameraImage()
}

protocol ViewModelOutput {
    var isLoading: Observable<Bool> { get }
    var inputImage: Observable<UIImage?> { get }
    var resultImage: Observable<UIImage?> { get }
    var errorMessage: Observable<String> { get }
}

protocol ViewModelProtocol: ViewModelInput, ViewModelOutput { }

class ViewModel: ViewModelProtocol {
    private let scanner = ImageScanner()
    private let picker = ImagePicker()
    private let upscaler = ImageUpscaler()

    var isLoading: Observable<Bool> = Observable(false)
    var inputImage: Observable<UIImage?> = Observable(nil)
    var resultImage: Observable<UIImage?> = Observable(nil)
    var errorMessage: Observable<String> = Observable("")

    func pickImage() {
        picker.image(with: processImageResult)
    }

    func getCameraImage() {
        scanner.image(with: processImageResult)
    }
}

private extension ViewModel {
    func processImageResult(result: Result<UIImage, Error>) {
        switch result {
        case let .success(image):
            let scaleFactor = max(image.size.height, image.size.width)
/ CGFloat(ImageUpscaler.Constants.inputDimension)
```

```
            inputImage.value = image.resizedImage(with: CGSize(width:
image.size.width / scaleFactor, height: image.size.height /
scaleFactor))
            isLoading.value = true
            upscaler.upscale(image: image) { [weak self] outputResult
in
                guard let self = self else { return }
                self.isLoading.value = false
                switch outputResult {
                case let .success(outputImage):
                    self.resultImage.value = outputImage
                case let .failure(error):
                    self.errorMessage.value                          =
error.localizedDescription
                }
            }
        case let .failure(error):
            errorMessage.value = error.localizedDescription
        }
    }
}
```

ДОДАТОК Б

Перевірка на співпадання

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1009257042

Дата проверки:
19.11.2021 10:01:18 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
19.11.2021 10:13:37 EET

ID пользователя:
76913

Название файла: IT-з02мп_Федоряка_ПЗ

Количество страниц: 65 Количество слов: 12577 Количество символов: 93375 Размер файла: 129.29 KB ID файла: 1009286296

5.1% Совпадения

Наибольшее совпадение: 1.49% с Интернет-источником (https://blog.csdn.net/generaljing/article/details/108300093)

3.29% Источники из Интернета | 12 ..... Страница 67

2.62% Источники из Библиотеки | 143 ..... Страница 67

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников