НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

До захисту допущено:
Завідувач кафедри
_____ Олена ЧУМАЧЕНКО
«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту» спеціальності 122 «Комп'ютерні науки»

на тему: «Розпізнавання емоцій на людському обличчі з використанням нейронних мереж»

Виконав:
студент IV курсу, групи КІ-91
Бурдейний Артем Олександрович _____

Керівник:
професор, д.ф.-м.н., Купенко Ольга Петрівна _____

Консультант з нормоконтролю:
фах. 1-ї кат., Гончарук Максим Миколайович _____

Консультант з економічного розділу:
доцент, к.е.н., Рощина Надія Василівна _____

Рецензент:
доцент, к.ф.-м.н. Лазаренко Ірина Сергіївна _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки»
Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ:
Завідувач кафедри
_____Олена ЧУМАЧЕНКО
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу студенту
Бурдейному Артему Олександровичу

1. Тема роботи «Розпізнавання емоцій на людському обличчі з використанням нейронних мереж», керівник роботи д.ф.-м.н., професор Купенко Ольга Петрівна, затверджені наказом по університету від «30» травня 2023 р. № 2065-с
2. Термін подання студентом роботи 16.06.2023
3. Вихідні дані до роботи – набір зображень з обличчями людей.
4. Зміст роботи – 1. Огляд предметної області; 2. Математичні основи роботи; 3. Реалізація програмного продукту та аналіз результатів; 4. Функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): механізм роботи алгоритмів, архітектура моделей, візуалізація результатів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	доцент, к.е.н., Рощина Н.В.		

7. Дата видачі завдання _21 лютого 2023 року_

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи	10.04.2023	Виконано
2.	Підготовка першого розділу	19.04.2023	Виконано
3.	Підготовка другого розділу	10.05.2023	Виконано
4.	Розробка програмного продукту	24.05.2023	Виконано
5.	Підготовка презентації доповіді	25.05.2023	Виконано
6.	Підготовка третього розділу	03.06.2023	Виконано
7.	Підготовка економічної частини	05.06.2023	Виконано
8.	Оформлення дипломної роботи	11.06.2023	Виконано

Студент Артем БУРДЕЙНИЙ

Керівник Ольга КУПЕНКО

РЕФЕРАТ

Дипломна робота: 129 с., 73 рис., 19 табл., 1 додаток, 27 джерел

ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, РОЗПІЗНАВАННЯ ЕМОЦІЙ, ГЛИБОКЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ, КОМП'ЮТЕРНИЙ ЗІР

Об'єкт дослідження – фотографії облич людей, що виражають різні емоції.

Предмет дослідження – згорткові нейронні мережі.

Мета роботи складається з декількох пунктів. Дослідити вплив роздільної здатності зображень та факту наявності на них кольору на точність роботи нейронної мережі при розпізнаванні емоцій на обличчі. Дослідити вплив архітектури, складності мережі на якість прогнозів. Дослідити вплив додавання CBAM (Convolutional Block Attention Module) до мережі. Створити програмний продукт з простим графічним інтерфейсом, який класифікуватиме емоцію на фотографії обличчя, яку обере користувач, та виділятиме на цьому фото найважливіші ділянки, що вплинули на результат розпізнавання.

Найкраща модель, що була навчена на підвибірці AffectNet, досягнула точності в 71.5% на тестовій вибірці. На її основі і було створено кінцевий програмний продукт мовою програмування Python.

ABSTRACT

Bachelor thesis: 129 pages, 73 figures, 19 tables, 1 appendix, 27 sources

CONVOLUTIONAL NEURAL NETWORKS, FACIAL EMOTION RECOGNITION, DEEP LEARNING, CLASSIFICATION, COMPUTER VISION

The object of the research is photos of people's faces expressing different emotions.

The subject of the research is convolutional neural networks.

The purpose of the work consists of several points. To investigate the influence of image resolution and the fact of presence of color on the accuracy of a neural network in recognizing facial emotions. To investigate the impact of architecture and network complexity on the quality of predictions. To investigate the impact of adding CBAM (Convolutional Block Attention Module) to the network. To create a software product that features a simple graphical user interface that will classify the emotion in a face on photo selected by the user and highlight the most important areas in the photo that influenced the recognition result.

The best model, trained on the AffectNet subset, achieved 71.5% accuracy on the test set. It was used as the basis for the final software product in the Python programming language.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 8
ВСТУП 9
РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Загальні принципи розпізнавання емоцій на обличчі 11
1.2 Актуальність задачі 13
1.3 Опис та аналіз даних 15
1.4 Постановка задачі 19
1.5 Висновки до розділу 20
РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ 22
2.1 Загальний огляд штучних нейронних мереж. 22
2.1.1 Порівняння методів глибокого та класичного машинного навчань 22
2.1.2 Будова багатошарової нейронної мережі 23
2.1.3 Функції активації 25
2.1.4 Навчання нейронних мереж 27
2.1.5 Перенавчання та методи боротьби з ним 30
2.1.6 Batch Normalization 33
2.1.7 Метрики 35
2.2 Загальний огляд згорткових нейронних мереж 38
2.2.1 Основні поняття згорткових нейронних мереж 38
2.2.2. Згортковий шар 40
2.2.3 Страйд і паддінг 43
2.2.4 Згортка 1×1 45
2.2.5 Шар пулінгу 46
2.2.6 Загальна архітектура згорткових нейронних мереж 47
2.2.7 Transfer learning 49
2.3 Відомі архітектури згорткових нейронних мереж 50
2.3.1 ResNet 50
2.3.2 MobileNet 53
2.4 Convolutional Block Attention Module 57
2.4 Gradient-weighted Class Activation Mapping 61

2.5 Висновки до розділу 2 64
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 65
3.1 Вибір програмних засобів 65
3.2 Препроцесинг та аугментація зображень 67
3.3 Опис навчених моделей та їх результатів 70
3.3.1 Загальний опис датасетів та моделей 70
3.3.2 Custom мережа та її результати 71
3.3.3 MobileNet-v2 та її результати 78
3.3.4 ResNet-50 та її результати 84
3.4 Порівняння та аналіз навчених моделей 90
3.5 Опис кінцевого програмного продукту 92
3.6 Висновки до розділу 3 95
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 96
4.1 Постановка задачі проектування 96
4.2 Обґрунтування функцій програмного продукту 97
4.3 Обґрунтування системи параметрів програмного продукту 100
4.4 Аналіз експертного оцінювання параметрів 103
4.5 Аналіз рівня якості варіантів реалізації функцій 107
4.6 Економічний аналіз варіантів розробки ПП 109
4.7 Вибір кращого варіанту ПП техніко-економічного рівня 115
4.8 Висновки до четвертого розділу 116
ВИСНОВКИ 117
СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ 118
ДОДАТОК А КОД 122

ПЕРЕЛІК СКОРОЧЕНЬ

FER – Facial Emotion Recognition (Розпізнавання емоцій за обличчям)

CNN – Convolutional Neural Network (Згорткова нейронна мережа)

CBAM – Convolutional Block Attention Module

Grad-CAM – Gradient-weighted Class Activation Mapping

AI – Artificial Intelligence (Штучний Інтелект)

BN та Batch Norm – Batch Normalization (Пакетна нормалізація)

HOG – Histogram of Oriented Gradients

ВСТУП

Емоції – це психологічні стани, що викликані нейрофізіологічними змінами в людині і відображають її суб’єктивну реакцію на певний об’єкт, думку чи ситуацію. Емоції грають фундаментальну роль в людському спілкуванні та іншій взаємодії. Здатність правильно розпізнавати емоції дозволяє ефективно комунікувати з іншими. Емоції є величезним джерелом знань, завдяки їх розумінню можна краще розуміти внутрішній стан людини, її мотивацію та ставлення до певних речей, її тривоги і страхи. Обличчя здатне виражати величезний спектр емоцій. Воно є одним з головних способів їх передачі, що робить його основним предметом для аналізу.

Сьогоднішній світ є дуже цифровізованим. Мобільні телефони, комп’ютери та інші технічні пристрої з камерами є в кожному домі. На вулиці та в приміщеннях є безліч камер відеоспостереження. Завдяки цьому стало дуже просто відслідкувати емоційний вираз на обличчі людей. Для людей процес розпізнавання емоцій за виразом обличчя є інтуїтивно зрозумілим та не потребує зусиль. Але навчити машини точно інтерпретувати емоційні сигнали виявилося непростою задачею.

Штучні нейронні мережі стали потужним та ефективним рішенням для вирішення важких задач розпізнавання образів, включно з розпізнаванням емоцій на обличчі. Для багатьох сучасних задач саме нейронні мережі виявились тим, фактично, єдиним засобом, що здатен їх гарно вирішувати. Вони дали можливість моделювати складні відношення і автоматично видобувати складні значимі ознаки із зображень, що відкрило нові можливості для створення ефективних AI систем, в тому числі і систем розпізнавання емоцій.

Метою цієї роботи є розробка моделі, що зможе якісно розпізнавати та класифікувати емоцію на людському обличчі. Ще однією ціллю є визначення впливу різних факторів на якість роботи цієї моделі. Об’єкт дослідження –

фотографії облич людей, шо виражають різні базові емоції. Предмет дослідження – згорткові нейронні мережі, їх різні архітектури та їх застосування для класифікації.

Основна частина роботи складається з чотирьох розділів. В першому будуть розглянуті основні принципи розпізнавання емоцій, різні підходи, актуальність цієї задачі, огляд набору даних, що використовуватиметься у цій роботі, зроблена постановка задачі. У другому розділі розглянуті та описані математичні основи цієї роботи, а саме принципи роботи нейронних мереж, в тому числі згорткових нейронних мереж та їх особливих архітектур. У третьому розділі описані процес обробки даних, архітектура, результати і аналіз побудованих моделей, і нарешті кінцевий програмний продукт, що розпізнаватиме емоції на зображеннях. Четвертий розділ є економічною частиною, де зроблений функціонально-вартісний аналіз програмного продукту.

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні принципи розпізнавання емоцій на обличчі

Розпізнавання емоцій – це технологія, що здатна виявляти і аналізувати настрої та власне емоції з різних джерел. Джерелом інформації можуть слугувати фотографії людей, відео, аудіо їх голосів, тексти, коментарі тощо. Кожен із цих специфічних видів «представлення» людей потребує відповідних специфічних методів роботи з ними. Було досліджено, що 55% емоційної інформації є візуальною, 38% є звуковою, тобто створена голосом, і 7% – вербальна, тобто самі слова та їх зміст [2]. Розпізнавання емоцій за обличчям (Facial Emotion Recognition чи FER) виявляють емоції саме на людському обличчі шляхом аналізу такої візуальної інформації як зображення чи відео. У даній роботі буде розглянуто саме розпізнавання емоцій на обличчі за зображеннями.

Класична задача FER складається з трьох етапів [1]:

1. Виявлення обличчя (face detection). Спершу на вхідному зображені визначається місцезнаходження обличчя чи облич. Кожне з них поміщається в спеціальну обмежувальну рамку (bounding box). Далі аналізуватиметься лише інформація в межах цих рамок, оскільки за їх межами відсутні обличчя, а значить і інформація для аналізу.
2. Виявлення виразу обличчя (facial expression detection). На знайдених обличчях шукаються ключові ознаки, що відповідатимуть за позиції очей, брів, рота та інших рис обличчя чи їх певних частин.
3. Вираз обличчя класифікується до певної емоції. Для цього аналізуються наявність, положення та взаємне розташування ключових ознак, що були знайдені на попередньому кроці. Класифікаційний алгоритм зазвичай побудований на основі AI методів, таких як машинне навчання.

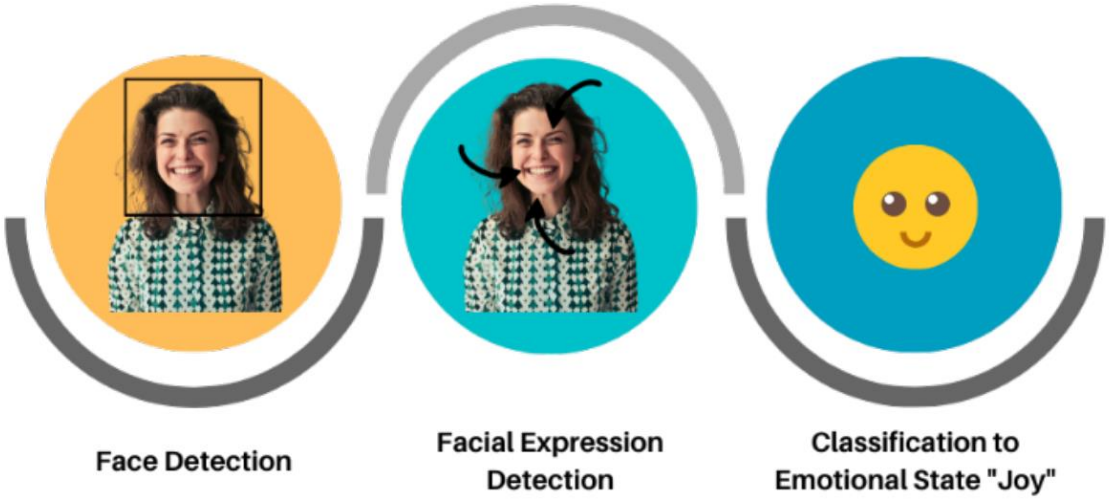

Рисунок 1.1 – Етапи розпізнавання емоцій за обличчям [1]

Описані кроки відповідають методам класифікації виразу обличчя класичними методами машинного навчання. При засновуванні нейронних мереж другий та третій (а іноді і перший) кроки об’єднуються в один етап. Це є можливим, оскільки нейронні мережі здатні самостійно без допоміжних алгоритмів визначати інформативні та цікаві для них ознаки, що можна безпосередньо застосувати для класифікації. Проте були запропоновані методи FER, що поєднують нейронні мережі і класичні методи комп’ютерного зору. Наприклад в [4] емоція розпізнається і на основі ознак видобутих згортковими шарами і на основі дескрипторів, що були створені алгоритмом Dense-SIFT. В [5] схожим чином окрім ознак, що знайдені згортковими шарами, також аналізуються карта HOG (Histogram of Oriented Gradients) та координати 68 основних ключових точок на обличчі.

В залежності від алгоритму можуть розпізнаватися різні емоції. Найзагальнішим варіантом є розпізнавання базових емоцій. До них належать всього 6 емоцій: гнів, огида, страх, радість, сум та здивування [1]. Також до них дуже часто додають сьому «нейтральну» емоцію, тобто коли обличчя не виражає ніяких емоцій. Іноді до базових зараховують презирство, але це непоширений випадок. Іншим варіантом є розпізнавання складених (compound) емоцій [1]. Наприклад здивовано-зляканий, розгнівано-сумний і тому подібне. Ще одним

варіантом FER є розпізнавання ментальних станів, таких як втома, нудьга, закоханість, зацікавленість тощо [1].

Для збільшення точності розпізнавання методи FER часто можуть узагальнюватися до більш складних алгоритмів, що окрім візуальної інформації аналізуватимуть голос людини, її біологічні характеристики такі як серцебиття [1]. Обробка такої різноманітної інформації, що надходить з різних джерел, сприятиме більш повному та всеохоплюючому розумінню певної людини і відповідно її емоційного стану.

1.2 Актуальність задачі

Людина може дуже просто, швидко та точно розпізнати емоції на обличчі інших. Але для комп'ютера ця задача не є тривіальною і є доволі важкою. На сьогоднішній день камери відеоспостереження стали дуже поширеними. Більшість телефонів та комп'ютерів також мають власні камери. Завдяки цьому стало дуже просто моніторити вираз обличчя потрібної людини, що збільшує необхідність появи гарних FER систем. Розробка такої інформаційної системи буде корисною в багатьох сферах. Нижче перераховані основні з них.

1. Персоналізований контент [1]. FER-алгоритми можуть аналізувати емоційну відповідь користувача на певний контент, щоб зрозуміти його смаки та вподобання. Завдяки цим отриманим знанням онлайн додатки по типу YouTube, Instagram, TikTok, Spotify та інші зможуть значно вдосконалити свою систему рекомендацій та покращити персоналізований контент.
2. Ринок та реклама [1] [14]. Аналогічно персоналізованому контенту реклама чи персональні пропозиції магазинів також можуть бути налаштовані для кожної людини індивідуально пропонуючи їй те, що викликає зацікавленість чи радість. Також FER допоможе продавцям вивчити поведінку покупців

виявивши що їм подобається, а що їх навпаки відлякує. FER значно допоможе у вивчені ринку.

3. Охорона здоров’я та медицина [1] [14]. FER-системи можуть бути дуже корисними в медицині, особливо в психології та психіатрії, допомагаючи лікарям ставити діагноз. Вони можуть допомогти виявити такі ментальні порушення як депресія, суїцидальні схильності, аутизм та інші. Також такі системи здатні покращити піклування за пацієнтами, проводити моніторинг їх стану, їх задоволеність лікуванням та наданими умовами.
4. Безпека водія [14]. Використання FER зробить автомобілі значно безпечнішими для водіїв та їх пасажирів, і зменшить кількість аварій. Така система аналізуватиме вираз обличчя водія щоб визначити, чи є воно втомленим, сонним чи п’яним. В позитивному випадку розумна система попередить водія і запропонує йому перепочити чи взагалі не дозволить поїхати.
5. Працевлаштування [1] [14]. FER система допоможе відсіяти незацікавлених кандидатів на робоче місце і сформувати уявлення про характер і настрій кандидатів. Це дозволить роботодавцям краще зрозуміти їх особисті якості та допоможе зробити остаточне рішення щодо прийому кандидатів на роботу.
6. Освіта [1]. Системи розпізнавання емоцій зможуть визначати емоційну відповідь та зацікавленість студентів при навчанні, що дозволить краще адаптувати навчальні програми. FER системи зможуть виміряти ступінь залученості студентів у процес онлайн навчання.
7. Суспільна безпека [1]. FER системи здатні аналізувати емоції натовпу та окремих людей. Це дозволить виявити настрої, що можуть становити потенційну терористичну загрозу, дозволить виявити злочинців, крадіїв та шахраїв, що видають себе специфічною поведінкою та виразом обличчя. Також інтеграція FER в детектори брехні здатні підвищити їх надійність та

точність. Аналіз сцен з місць злочину за допомогою FER систем дозволить краще зрозуміти мотиви злочинців.

1.3 Опис та аналіз даних

Є декілька відомих датасетів облич з різними емоціями. Однією із характеристик цих датасетів є спосіб збору зображень. Деякі датасети були отримані «лабораторним» способом. Тобто була команда акторів, що показували різні емоції. Такі датасети зазвичай мають відносно небагато зображень. Також самі зображення мають багато спільного, а саме однакові освітлення, задній фон і положення голови. До таких датасетів належать, наприклад, CK+ (593 записаних коротких відео) [2], JAFFE (213 чорно-білих фотографій японських жінок) [2] та інші. Інші датасети мають різні зображення, що були взяті з мережі Інтернет. Такі датасети називають «real-world» і вони є дуже різноманітними. Вони мають обличчя в різних позах і нахилах, з різними умовами освітлення, різних рас та віку. Також такі датасети зазвичай мають значно більше зображень ніж їх «лабораторні» аналоги. До таких датасетів належать, наприклад, FER-2013 (майже 36 тис. чорно-білих зображень низької роздільної здатності 48×48 пікселів) [2], AffectNet (420 тис. кольорових зображень, з яких 291.6 тис. фото облич класифіковані до базових емоцій) [17] та інші.

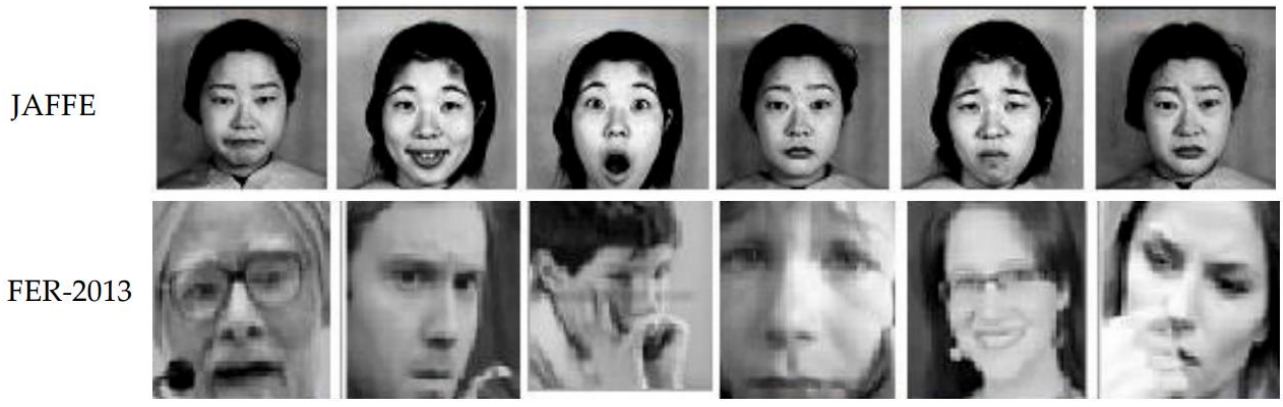

Рисунок 1.2 – Порівняння «лабораторного» і «real-world» датасетів [3]

Було вирішено використовувати саме «real-world» датасет через те, що вони є надзвичайно різноманітними та менш упередженими до умов при яких було зроблені фото, таких як освітлення. Завдяки цьому готовий програмний продукт зможе працювати з фотографіями зробленими в довільних «незафіксованих» умовах. Також будуть вивчені ознаки, що будуть інваріантними до цих умов. Ще однією перевагою є те, що такі датасети мають велику кількість зображень, що дозволить створити більш довершені класифікатори, вивчити різноманітніші ознаки та зменшить проблему перенавчання.

Було вирішено використовувати саме AffectNet датасет, бо на відмінну від іншого популярного набору «real-word» даних FER-2013 він має кольорові фото кращої роздільної здатності. Також його публічно доступні підвибірки є більш збалансованими, тобто різниця між кількістю елементів різних класів є меншою.

AffectNet містить 420299 зображень, що розбиті на декілька класів. Класи та кількість зображень у кожному з них можна побачити нижче.

Neutral	75374
Happy	134915
Sad	25959
Surprise	14590
Fear	6878
Disgust	4303
Anger	25382
Contempt	4250
None	33588
Uncertain	12145
Non-Face	82915
Total	420299

Рисунок 1.3 – Розподіл елементів по класам у AffectNet [17]

Оригінальний повний AffectNet не є загальнодоступним набором даних, але на сайті https://www.kaggle.com/ є публічно доступна підвибірка цього датасету. На момент написання цієї роботи вона містила навчальну множину з 36800

зображень розбитих на 8 емоцій, а саме 5000 екземплярів для всіх емоцій окрім disgust та contempt, що містили по 3800 та 3000 екземплярів відповідно. Тестова множина містила по 500 зображень для кожної з восьми емоцій.

Ця підвибірка (як і весь датасет) має дуже багато хибно промаркованих зображень та дублікатів. Окрім того, емоція contempt (презирство) не належить до базових, часто її дуже важко відрізнити від огиди, радості чи інших емоцій. Тому з початковим датасетом були виконані наступні дії:

1. Вручну видалені більшість хибно класифікованих зображень.
2. Повністю видалений клас contempt.
3. Видалені більшість дублікатів на основі їх косинусної подібності і евклідової відстані. Для цього спершу навчальна і тестова множини були злиті в одну, а всі зображення були приведені до однакової роздільної здатності.

Рисунок 1.4 – Приклади зображень з почищеного датасету

Після цього отриманий датасет було розбито на навчальну, валідаційну і тестові множини (train, validation, test sets). Тестова і валідаційна вибірки містять

приблизно такий самий відносний розподіл елементів по класам, що і навчальна множина. Загалом створений датасет має 21981 зображення, з них 19981 зображення знаходиться в навчальній вибірці, 1000 – у валідаційній вибірці і 1000 – у тестувальній. Він містить 7 класів для кожної з основних емоцій: angry (злість), disgust (огида), fear (страх), happy (радість), neutral (нейтральне обличчя без виражених емоцій), sad (сум), surprised (здивування). Розподіл зображень по класам та приклади фотографій розміщені на діаграмі і в таблиці нижче.

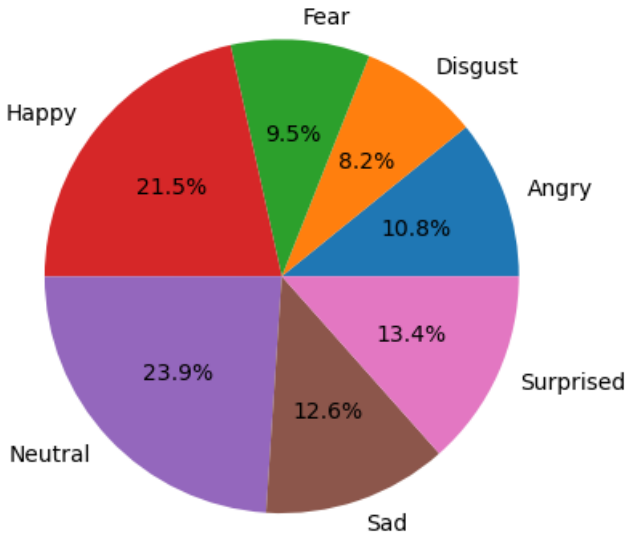

Рисунок 1.5 – Розподіл зображень по класам у тренувальній множині

Таблиця 1.1 – Розподіл зображень по класам і вибіркам

Train set						
Angry	Disgust	Fear	Happy	Neutral	Sad	Surprised
2166	1629	1894	4305	4780	2520	2687
Validation / Test sets						
Angry	Disgust	Fear	Happy	Neutral	Sad	Surprised
108	82	95	215	239	126	135

Усі зображення вже є квадратними, хоча роздільні здатності зображень є різними. Перед використанням датасету для тренування нейронної мережі усі зображення мусять бути приведені до однакової роздільної здатності. Вплив роздільної здатності на якість класифікації нейронними мережами буде досліджено у третьому розділі.

1.4 Постановка задачі

Перед нами стоїть класична задача класифікації, а саме класифікувати фотографію людського обличчя за емоцією, що виражена на ньому. Це завдання складається з двох підзадач.

Спершу ми маємо розробити ефективний алгоритм, що міг би визначати емоцію за фотографією. В даній роботі в якості класифікатора будуть розглянуті нейронні мережі. Вони можуть мати різноманітні архітектурні рішення, і їх треба навчити. Першу підзадачу можна описати наступним чином:

1. Сформувати набір даних, що буде використовуватися для навчання нейронних мереж. Цей датасет складатиметься з трьох множин: навчальної (безпосередньо використовується для навчання і налаштування внутрішніх параметрів і ваг моделі), валідаційної (необхідна для підбору оптимальних зовнішніх гіперпараметрів моделі) і тестової (необхідна для остаточної оцінки якості роботи нейронної мережі). Кожен елемент датасету є парою значень X та Y, де X є зображенням обличчя, а Y – номером класу, номером емоції до якого це зображення належить.
2. Навчити нейронні мережі, що робитимуть відображення $f: X \to Y$ з мінімально можливою похибкою. Навчити означає за допомогою алгоритмів навчання та навчальної вибірки налаштувати оптимальні внутрішні параметри моделі. Треба навчити декілька нейронних мереж, що відрізняються архітектурою. Також необхідно дослідити як вхідні данні X

впливають на результат. А саме дослідити вплив архітектури моделей, роздільної здатності зображення та факту того, чи воно кольорове чи чорно-біле на результат класифікації.

3. Порівняти результати різних архітектур, що були навчені на різних варіаціях зображень X. Обрати той, що покаже найкращий результат на тестовій вибірці.

Тепер необхідно сформувати постановку другої підзадачі. Після того, як ми спроектували та навчили найефективнішу нейронну мережу необхідно створити інтуїтивно зрозумілу та просту програму з GUI. Користувач має мати змогу завантажити довільне фото з обличчям. Існує обмеження, а саме обличчя на фото має бути єдиним. Головним результатом роботи має бути виведена на екран емоція, що присутня на даному користувачем зображені. Окрім цього будуть додатково виведені два зображення, а саме preprocessed зображення обличчя і те ж саме preprocessed зображення з накладеною на нього тепловою картою Grad-CAM, що виділяє «найважливіші» ділянки зображення, що вплинули на отриманий результат класифікації.

1.5 Висновки до розділу

В цьому розділі було описано предметну область поставленої задачі та визначені основні поняття розпізнавання емоцій. Було описано основні кроки які необхідно виконати для розпізнавання емоції на зображені обличчя людини. Наведені деякі приклади робіт присвячених задачі FER.

Була розглянута актуальність задачі розпізнавання емоцій. Показано, що ця задача є надзвичайно актуальною в сучасному дуже цифровізованому світі.

Було описані деякі відомі датасети для FER та обґрунтований зроблений вибір набору даних. Описані дії, що були над ним виконані. Детально описаний

остаточний варіант датасету. Наведений розподіл елементів по класам, показано приклади зображень з кожного класу.

Уточнено поставлену задачу. Визначені кроки, які необхідно розглянути та виконати для створення системи FER. Визначені вимоги до цієї системи.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Загальний огляд штучних нейронних мереж.

2.1.1 Порівняння методів глибокого та класичного машинного навчань

Нейронна мережа – це математична модель, обчислювальна система, один із алгоритмів машинного навчання, що побудований за принципами, які частково ґрунтуються на біологічних нервових системах. Сукупність різних алгоритмів, понять, методів та моделей, що пов'язані з нейронними мережами, виокремлюють у окреме поняття під назвою «глибоке навчання» (deep learning). Глибоке навчання є підмножиною машинного навчання (machine learning), що в свою чергу є підмножиною штучного інтелекту (artificial intelligence) [18]. Хоча нерідко терміни глибоке і машинне навчання використовують окремо, щоб підкреслити відмінність між ними.

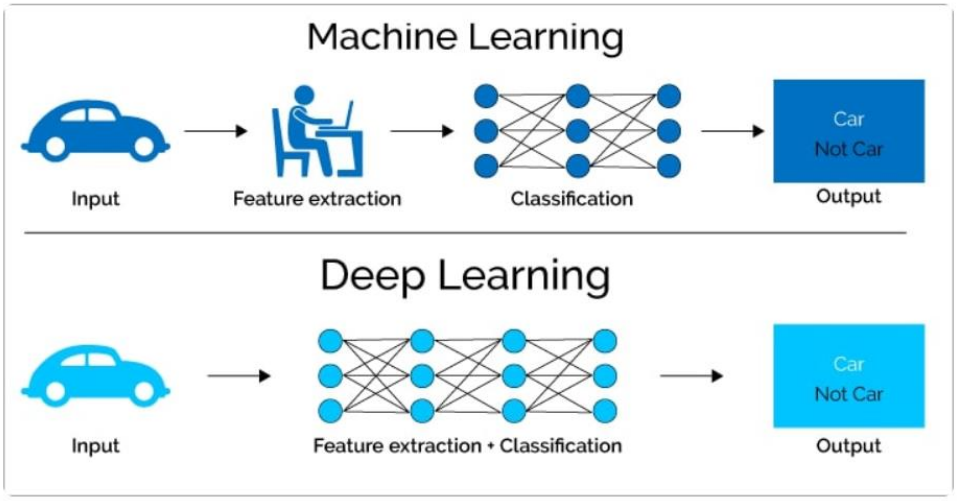

Рисунок 2.1 – Порівняння традиційного машинного та глибокого навчань [18]

Традиційні методи машинного навчання вимагають, щоб завчасно людьми чи іншими алгоритмами були виділені репрезентативні ознаки кожного об'єкта (наприклад виділити ключові точки обличчя з зображення). І після цього

алгоритм машинного навчання навчається на цих виділений структурованих ознаках. З іншого боку глибоке навчання використовує нейронні мережі, що здатні самостійно автоматично виділяти необхідні цікаві для них ознаки без залучення участі людини [18]. Саме тому нейронні мережі краще підходять для роботи з неструктурованим даними, такими як відео, аудіо та зображення, в тому числі і зображення облич.

2.1.2 Будова багатошарової нейронної мережі

Як було вище сказано, принципи, на яких побудовані штучні нейронні мережі (далі нейронні мережі), частково залучені у біологічних нервових мереж. Нейронні мережі складаються з великої кількості обчислювальних вузлів – нейронів, які поєднуються у складні архітектури. Не зважаючи на величезну кількість різноманітних мереж і задач, що вони виконують, структура кожного нейрона у них є майже незмінною.

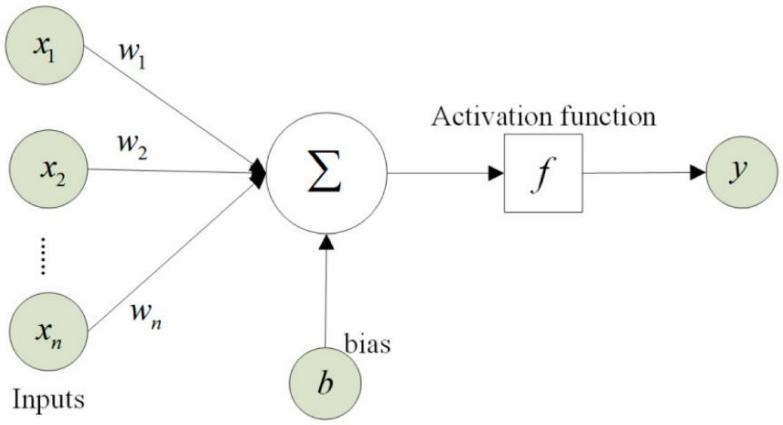

Рисунок 2.2 – Структура нейрона

На вхід нейрона подається вхідний сигнал, що представлений числовим вектором x. Потім рахується зважена сума усіх складових цього сигналу. До цієї суми також може додаватися зміщення (bias), але це не обов’язково. Наступним етапом, що в деяких випадках пропускається, є застосування певної нелінійної

функції, яку називають функцією активації, до зваженої суми. Результат активаційної функції є виходом нейрона, що подається на входи до інших нейронів. Наведені пояснення можна зобразити наступним чином:

$$y = f(w_1 x_1 + \cdots + w_n x_n + b) \tag{2.1}$$

де f – функція активація (може бути відсутня);

x_k – k-та координата вхідного сигналу;

w_k – вага k-го зв'язку, що асоційований з k-тою координатою входу;

b – зміщення (може бути відсутнє).

Нейрони організовані в шари. Нейрони одного шару не поєднані між собою, вони поєднані з нейронами безпосередньо наступного та попереднього шарів. Нейрони отримують сигнал від попереднього шару, кожен нейрон шару обробляє сигнал та передає його нейронам наступного шару. Функція активації є однаковою серед нейронів одного шару, але може відрізнятися від активацій інших шарів. Вхідним шаром (Input layer) називається вхідний сигнал у мережу. Вхідний шар не виконує ніяких операцій над сигналом. Вихідним шаром (Output layer) – останній шар мережі, що повертає результат. Прихованими шарами (Hidden layers) називають шари, що розташовані між вхідним та вихідним.

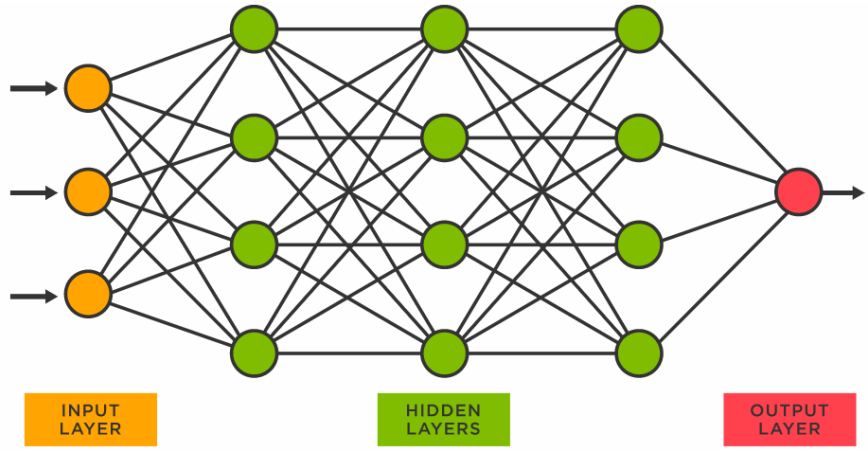

Рисунок 2.3 – Структура нейронної мережі

Якщо кожен нейрон певного шару поєднаний з усіма нейронами попереднього шару, то такий шар називають повнозв'язним (fully connected layer). Багатошарова нейронна мережа (чи її окрема частина), що складається лише з повнозв'язних шарів, називається багатошаровим персептроном (multi-layer perceptron або MLP). Операцію, що виконує один повнозв'язний шар, можна нескладно представити наступним виразом:

$$y_i = f_i\left(W_i^T y_{i-1} + b_i\right) \tag{2.2}$$

де y_i та y_{i-1} – вихід i-го шару та вихід попереднього $i-1$ шару відповідно;

f_i – функція активації і-го шару;

W_i – матриця ваг і-го шару. Ця матриця має розмірність $\mathrm{n} \times m$, де n – кількість вхідних ознак, а m – кількість вихідних ознак, тобто кількість нейронів в шарі;

b_i – вектор зміщення для і-го шару. Розмірність $\mathrm{m} \times 1$.

2.1.3 Функції активації

Існує широкий вибір різноманітних активаційних функцій, їх вибір залежить від поставленої задачі, архітектури нейронної мережі та розташування в ній шару, активаційна функція якого розглядається, та від вхідних даних. Розглянемо деякі з активаційних функцій.

Сигмоїдна функція активації (sigmoid або logistic) задається наступним рівнянням:

$$y(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Раніше вважалася основною функцією активацій для прихованих шарів, але сьогодні у багатьох архітектурах її витіснила інша функція активації ReLU. Особливістю сигмоїди є те, що вона відображає вхідне значення на інтервал (0, 1), а тому є ідеальним варіантом, якщо вихід нейрона слід інтерпретувати як ймовірність.

Функція активації softmax діє на всі нейрони шару глобально, а не ізольовано на кожен нейрон. Використовується зазвичай лише у вихідному шарі.

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.4}$$

де y_i – вихід і-го нейрону;

x_k – зважена сума k-го нейрону;

n – загальна кількість нейронів у цьому шарі.

Усі виходи шару з функцією softmax можна інтерпретувати як ймовірності, їх сума дорівнюватиме одиниці, а значення кожного y_i лежить на інтервалі (0, 1).

Функція активації ReLU (Rectified Linear Unit) задається наступним виразом:

$$\mathrm{y}(x) = max(0, x) \tag{2.5}$$

Функція ReLU має багато переваг як активаційна функція внутрішніх шарів порівняно з сигмоїдною функцією активації:

- Обчислюється дуже швидко, для цього необхідне лише одне порівняння з нулем;
- Похідна ReLU в точці рахується дуже просто. Вона дорівнюватиме 0 якщо x менший нуля і 1 – якщо більший;

- Не має межі насичення при $x > 0$, а тому менш вразлива до проблеми зникаючих градієнтів.

2.1.4 Навчання нейронних мереж

Для того, щоб під час навчання зрозуміти наскільки гарно працює нейронна мережа, використовують функцію втрат (loss function). Ця функція порівнює істинне значення цільової змінної навчального прикладу з результатом роботи нейромережі і повертає число, що відображає якість передбачення. Для задачі багатокласової класифікації найпоширенішим варіантом є перехресна ентропія (cross entropy). Вона визначається наступним чином:

$$Q = -\sum_{i} y_i \log(\hat{y}_i) \tag{2.6}$$

де y_i – це справжнє значення;

$\hat{y}_i$ – спрогнозоване мережею значення;

i – номер виходу мережі.

Навчання нейронних мереж включає в себе підбір оптимальних параметрів, наприклад ваги зв’язків та зміщення. Найпоширенішими способами навчання є градієнтний спуск та його модифікації. Для цього намагаються мінімізувати функцію втрат ітеративно змінюючи значення параметрів мережі. Спершу параметри ініціалізуються невеликими випадковими значеннями, зміщення ініціалізуються нулями. Потім відбувається прямий прохід (forward propagation), де мережа робить передбачення на вхідних даних. Також обчислюється функція втрат. Наступний крок – це зворотнє розповсюдження похибки (back propagation). На цьому етапі розраховуються похідні від функції втрат по всім параметрам

мережі, які необхідно навчити. Потім функції ваг коригуються в залежності від значення цього градієнту.

Якщо розмір датасету є великим, то тоді використовують стохастичний градієнтний спуск та його модифікації. Під час навчання крок оновлення параметрів базується не на всьому датасеті, а лише на його невеликій частині. Через мережу проганяють не весь датасет, а лише його частину, що називається мінібатч (minibatch), розмір якого не перевищує декількох сотень елементів. Потім беруть наступний мінібатч з нових елементів і виконують аналогічні дії до тих пір, поки не буде переглянуто весь датасет. Через те, що крок оновлення параметрів вираховується на основі невеликої підвибірки, то сама траєкторія оновлення ваг стає більш хаотичною і випадковою, хоча в середньому все ще прямує до локального мінімуму функції втрат. Завдяки тому, що градієнтний крок вираховується на мінібатчі датасету, а не на всій вибірці, то збіг до локального мінімуму займає значно менше часу, бо крок обраховується значно швидше. Також значно економніше витрачається оперативна пам’ять, бо ми там тримаємо лише мінібатч, а не всю вибірку. Ще одним плюсом є те, що завдяки наявності хаотичності у траєкторії оновлення ваг ми можемо «вискочити» з локальних не найкращих мінімумів.

Однією модифікацій градієнтного спуску є оптимізаційний алгоритм Adam (Adaptive Moment Estimation). Було емпірично показано, що в більшості ситуацій на багатьох датасетах він виявляється найефективнішим оптимізаційним алгоритмом [6].

В цьому алгоритмі ми зберігаємо минулі значення градієнтів та їх квадратів за допомогою експоненційно зваженого ковзного середнього (Exponentially Weighted Moving Average).

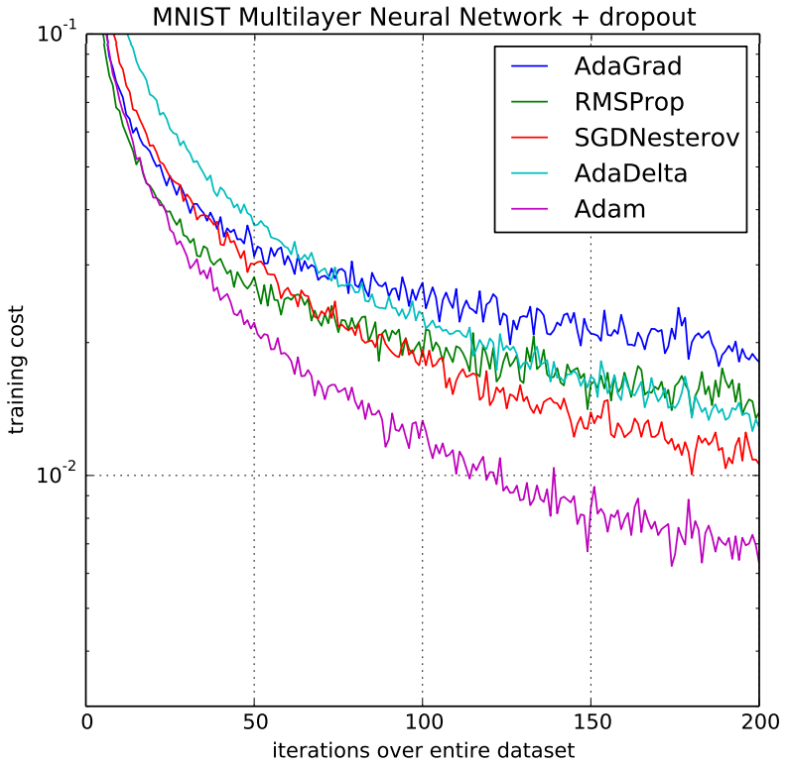

Рисунок 2.4 – Порівняння Adam з іншими оптимізаторами на MNIST [6]

Кроки алгоритму [6]:

1. Ініціалізуємо $V_{dw} = S_{dw} = \vec{0}$;
2. Обчислюємо градієнти функції втрат Q відносно параметрів мережі, тобто $\frac{\partial Q}{\partial w}$ під час back propagation;
3. Обчислюємо $V_{dw} = \beta_1 V_{dw} + (1 - \beta_1)\frac{\partial Q}{\partial w}$ та $S_{dw} = \beta_2 S_{dw} + (1 - \beta_2)\left(\frac{\partial Q}{\partial w}\right)^2$ де β_1 та β_2 – гіперпараметри, дійсні числа, зазвичай 0.9 та 0.999 відповідно;
4. Цей крок є необов'язковим. Через те, що ковзні середні ініціалізовані нулями, вони нешвидко збільшуватимуть свої значення. Тому використовують bias correction, щоб штучно збільшити значення.

   $V_{dw}^{corr} = \frac{V_{dw}}{1-\beta_1{}^t}$, $S_{dw}^{corr} = \frac{S_{dw}}{1-\beta_2{}^t}$, де t – номер ітерації;
5. Оновлюємо ваги та зміщення за наступним правилом:

$$w = w - \alpha \frac{V_{dw}^{corr}}{\sqrt{S_{dw}^{corr}} + \epsilon} \tag{2.7}$$

де ϵ – дуже мале число, що додається для уникнення ділення на нуль;

Якщо не виконували 4-й крок, то замість скоригованих значень підставляємо оригінальні V_{dw} та S_{dw}.

V_{dw} є моментом і діє схоже на фізичний імпульс. Він накопичує минулі градієнти. Завдяки цьому напрямок і швидкість зміни ваг не може різко змінитись, це дозволяє не застрягати в невеликих локальних мінімумах. Також завдяки імпульсу швидше проходяться ділянки «плато», де градієнти за усіма напрямками є дуже малими, і загалом прискорюється збіжність. В знаменнику S_{dw} значно покращує збіжність, якщо поверхня є надто розтягнутою вздовж певних осей. Це ковзне середнє дозволяє адаптувати свою швидкість навчання для кожного параметра, що усуває «коливальний» характер руху градієнта і пришвидшує збіжність.

2.1.5 Перенавчання та методи боротьби з ним

Перенавчання (overfitting) – це звичне явище, яке часто виникає при навчанні нейронних мереж та інших моделей машинного навчання. Воно полягає в тому, що модель починає надто сильно «налаштовуватися» на вхідні дані. Вона намагається ідеально відповідати вхідним даним, що спричиняє надлишкове ускладнення моделі, «підгонку» під шум та погіршення узагальнюваної здатності мережі. Перенавчена модель буде дуже добре описувати та робити прогнози на тренувальній вибірці, але на тестовій, яку модель не бачила під час навчання, вона покаже погані результати.

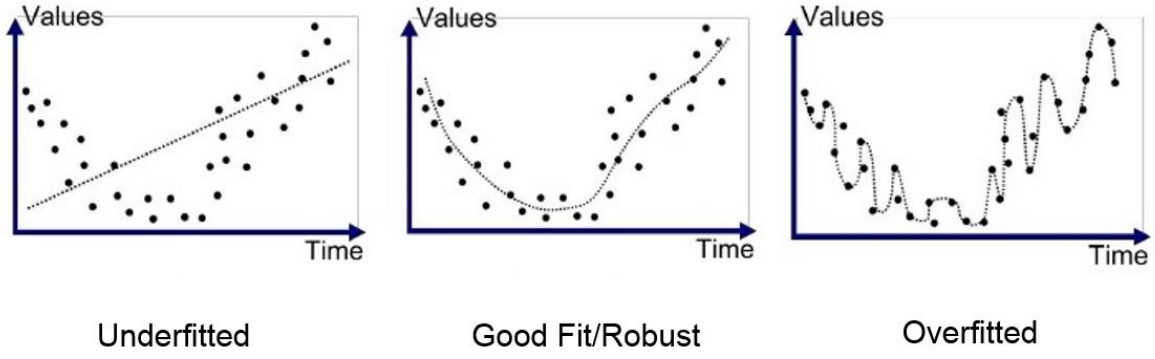

Рисунок 2.5 – Порівняння недонавченої, навченої та перенавченої моделі [19]

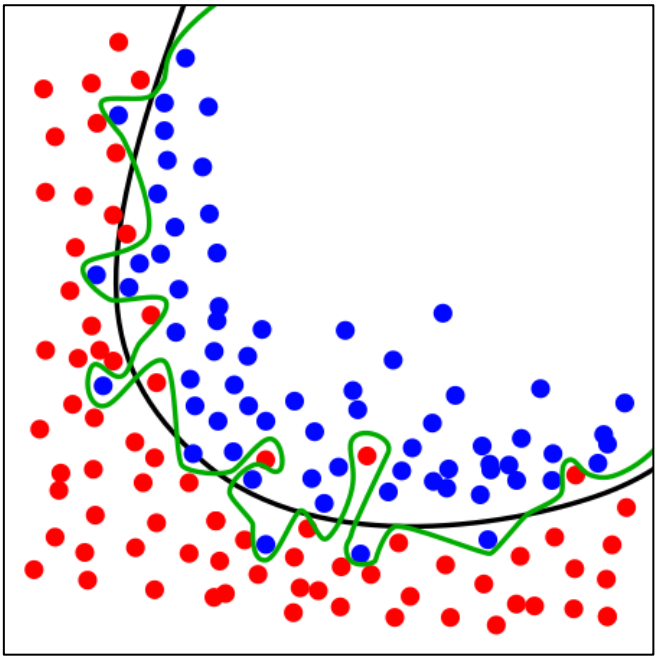

Рисунок 2.6 – Гарно навчена модель (чорна лінія) та перенавчена (зелена) [20]

Існує декілька методів, що здатні зменшити ефект перенавчання. Ці методи ще називаються регуляризацією, або такими, що мають регуляризаційний ефект. Ось деякі з них, які можна застосувати до нейронних мереж [19]:

1. Аугментація даних (Data augmentation).

Цей метод полягає в додаванні незначного шуму до даних, що все ще залишає їх інформативними, впізнаваними та доволі подібними. Наприклад це відображення, поворот, зміна яскравості зображення, заміна слів їх синонімами

у тексті, додавання фонового шуму до аудіо і тому подібне. Завдяки цьому дані стають значно різноманітнішими без явного збільшення розміру датасету. Через це модель не може ідеально «підігнати» себе до навчальної вибірки, бо сама вибірка є мінливою. І тому модель вимушена навчатися на загальних ознаках, які гарно узагальнюють датасет.

2. L2 регуляризація (L2 regularization або weight decay).

Цей метод полягає у додаванні до функції втрат нового члену, що штрафує значення ваг. Функція втрат виглядатиме наступним чином:

$$Q_{L2} = Q + \frac{\lambda}{2}\left|\left|w\right|\right|^2 \tag{2.8}$$

де Q – звичайна функція втрат;

λ – гіперпараметр, що регулює як сильно штрафувати величину параметрів;

$\left|\left|w\right|\right|^2$ – квадрат L2 (евклідової) норми вектору, що складається з усіх параметрів, що ми регуляризуємо. Тобто сума квадратів усіх параметрів.

Завдяки цьому регуляризаційному прийому значення ваг будуть меншими за модулем. Через це моделі будуть значно простішими, кожна навчена функція вноситиме менший вклад у результат, що зменшує можливість просто запам’ятати вхідні дані. Через невеликі ваги прогноз на новому зразку даних, що незначно відрізняється від даних у навчальній вибірці, лише трохи відрізнятиметься від прогнозів на подібних даних.

3. Дропаут (Dropout).

Цей метод випадковим чином обнуляє виходи деяких нейронів. Зазвичай реалізується як новий додатковий шар у нейромережі. Цей шар спершу незалежно обнуляє деякі складові вхідного сигналу з ймовірністю p (це число є гіперпараметром), а потім масштабує результат домножуючи його на $\frac{1}{1-p}$ [22]. Це

масштабування гарантує, що дропаут не змінить очікуване значення сигналу і не спричинить його затухання через обнулення деяких координат вхідного вектору.

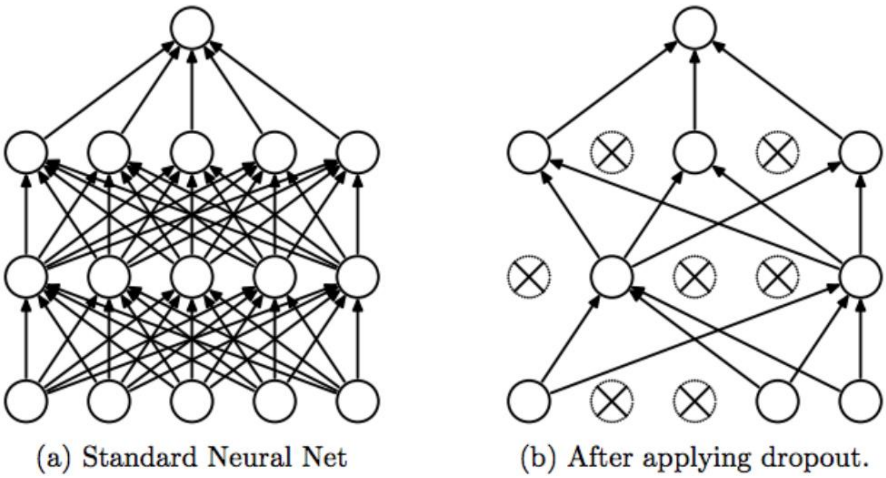

Рисунок 2.7 – Приклад застосування дропауту [21]

Шар дропауту працює лише під час навчання мережі. Під час фази прогнозування шар дропауту вимикається і не виконує жодних дій пропускаючи крізь себе інформацію без змін [22].

Дропаут змушує нейронну мережу вивчати більш загальні ознаки, що залишаються корисними, навіть якщо деякі складові сигналу будуть вилучені. Дропаут унеможливлює надто сильну коадаптацію різних нейронів. Також дропаут спричиняє те, що через випадкове вилучення нейронів сигнал після дропауту буде зашумленим, що, як і при аугментації даних, матиме регуляризаційний ефект.

2.1.6 Batch Normalization

Пакетна нормалізація (batch normalization) – це техніка, що дозволяє покращити навчання та кінцеву продуктивність і якість прогнозів мережі. Також пакетна нормалізація має незначний регуляризаційний ефект, але вона не розглядається як метод регуляризації як такий.

Зазвичай нормалізація даних робиться лише один раз перед першим шаром мережі, що збільшує стабільність мережі, уніфікує вхідні дані, масштабує ознаки до подібних діапазонів та загалом покращує збіжність до мінімуму функції втрат. При застосуванні пакетної нормалізації подібні дії робитимуться і в прихованих шарах нейронної мережі. Вона реалізується як окремний шар, що модифікує вхідні сигнали. Таких шарів може бути декілька в мережі. Шар пакетної нормалізації виконує наступні дії [23]:

1. Для вхідного мінібатчу рахуємо середнє μ та дисперсію σ^2 для кожної ознаки серед усіх елементів мінібатчу;
2. Центруємо і масштабуємо елементи мінібатчу, щоб вони мали нульове середнє та одиничну дисперсію;

$$y_{norm} = \frac{y - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.9}$$

де y_{norm} – нормований сигнал;

y – вхідний сигнал;

ϵ – мале число для уникнення ділення на нуль.

3. Можливо для глибоких шарів буде краще, якщо дані матимуть інакші середнє та дисперсію. Тому знову масштабуємо і зсуваємо до бажаних значень середнього та дисперсії;

$$\tilde{y} = \gamma \cdot y_{norm} + \beta \tag{2.10}$$

де γ та β – бажані значення дисперсії та середнього відповідно. Є параметрами, що також навчаються мережею як і усі інші параметри та ваги. Ці параметри ініціалізуються значеннями 1 та 0 відповідно.

Під час тестування мережі шар пакетної нормалізації працюватиме інакше, бо тепер дані зазвичай попадають в мережу не мінібатчами, а по одному. Тому запам’ятовується середнє і дисперсія навчального датасету, а нові значення додаються до цієї статистики за допомогою експоненційно зваженого ковзного середнього [23].

Не зважаючи на те, що пакетна нормалізація показала чудові емпіричні результати, сьогодні все ще чітко не зрозуміло, чому вона працює настільки добре. Поширеним поясненням є те, що вона зменшує внутрішній коваріаційний зсув. Це явище полягає в тому при навчанні попередніх шарів, розподіл їх виходу буде змінюватися, що змусить наступний шар постійно налаштовуватись на новий розподіл попереднього шару. Завдяки пакетній нормалізації середнє і дисперсія розподілу попереднього шару фіксуються, що полегшує навчання наступних шарів [23].

Пакетна нормалізація зазвичай використовується перед функцією активації одразу після зваженої суми, тому що тоді результат самої активаційної функції не буде викривленим, а також тому, що на її вхід подаватимуться нормалізовані дані, що покращить стабільність цієї функції. Однак варіант, де пакетну нормалізацію розміщують після функції активації, також є поширеним.

2.1.7 Метрики

Для оцінки якості навчання мережі використовують не тільки функцію втрат, а й так звані метрики (metrics). Ці функції не використовуються для навчання, вони необхідні лише щоб оцінити якість роботи моделі і повернути результат, що є більш зрозумілим і наочним для людини ніж значення функції втрат. Метрики дозволяють порівняти різні моделі акцентуючи увагу на різних речах в залежності від обраної метрики. На вхід метрики при класифікації поступає два вектори: перший відповідає істинному значенню, а другий – спрогнозованому

мережею. Розглянемо основні метрики, що використовуються при багатокласовій класифікації. Позначатимемо далі істинний вектор як y^{true}, спрогнозований як y^{pred}, кількість елементів у вибірці як n.

Accuracy (точність) – найпростіша й найбільш інтуїтивно зрозуміла метрика, що показує частку правильних класифікацій. Її можна обрахувати наступним чином [24]:

$$\text{accuracy}(y^{true}, y^{pred}) = \frac{1}{n}\sum_{i=0}^{n-1} \mathbb{1}\left(y_i^{true} = y_i^{pred}\right) \tag{2.11}$$

де $\mathbb{1}$ – індикаторна функція.

Проблема accuracy, що вона може давати викривлені результати, якщо датасет помітно незбалансований, тобто якщо кількість елементів одних класів сильно відрізняється від кількості у інших класах. Щоб врахувати це слід використовувати інші метрики.

Top-k accuracy – метрика, що є узагальненням звичайного accuracy. Також рахує частку правильних класифікацій, але тепер класифікація вважатиметься правильною, якщо істинне значення є серед перших k спрогнозованих значень з найбільшими ймовірностями [24]. Top-1 accuracy аналогічна звичайній accuracy.

Confusion matrix [24] – особлива метрика, бо повертає одразу матрицю чисел, що власне так і називається. Елемент матриці a_{ij} дорівнюватиме кількості елементів класу i, що були класифіковані до класу j. Тобто рядки відповідають істинним класам, а стовпчики – спрогнозованим. Елементи на головній діагоналі a_{ii} відповідають правильним класифікаціям.

Наступні метрики розглядають кожен клас незалежно і повертатимуть окремих n значень для кожного класу відповідно. При цьому елементи поточного

класу k вважатимуться positive, а усі інші об’єднуються у negative. Тоді класифіковані об’єкти можна розбити на 4 групи:

1. TP (True Positive) – елементи є positive (належать до класу k) і були правильно класифіковані як positive.
2. TN (True Negative) – елементи є negative (не належать до класу k) і були правильно класифіковані як negative.
3. FP (False Positive) – елементи є negative (не належать до класу k) і були хибно класифіковані як positive (належать до класу k).
4. FN (False Negative) – елементи є positive (належать до класу k) і були хибно класифіковані як negative (не належать до класу k).

Далі в цій роботі під цими чотирма позначеннями матимуться на увазі кількість елементів у відповідній множині.

Precision (точність) – для класу k показує частку елементів класу k серед усіх елементів, що було класифіковані як клас k. Обраховується для кожного класу окремо наступним чином [24]:

$$\text{Precision}_k = \frac{TP}{TP + FP} \tag{2.12}$$

Recall (чутливість) – для класу k показує частку істинних елементів класу k, що були класифіковані як k. Обчислюється для кожного класу окремо наступним чином [24]:

$$\text{Recall}_k = \frac{TP}{TP + FN} \tag{2.13}$$

Precision та Recall є взаємопов’язаними метриками. Оскільки вони рахуються для кожного класу окремо, то їх можна використовувати для

незбалансованих вибірок. Ці дві метрики дадуть повну вичерпну інформацію тільки якщо використовуються разом.

f1-score – метрика, що об’єднує precision та recall шляхом їх середнього гармонічного. Використовується саме середнє гармонічно, бо воно сильніше штрафує малі значення. Обчислюється для кожного класу наступним чином [24]:

$$f1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.14}$$

Є декілька способів усереднити n різних значень для precision, recall та f1-score в одне значення.

Micro averaging – рахує метрики глобально, підраховуючи загальну сумарну кількість TF, FP та FN результатів для всіх класів [24]. Цей метод надає більшу вагу більшим класам, а тому краще підійде, якщо розміри класів приблизно однакові.

Macro averaging – підраховує метрики для кожного класу окремо, а потім рахує їх середнє арифметичне [24]. Надає усім класам однакову вагу незалежно від їх розміру, тому гарно підходить для незбалансованих вибірок, де усі класи є однаково важливими.

Weighted averaging – працює аналогічно macro averaging, але тепер ми самі обираємо вагу «важливості» для кожного класу [24].

2.2 Загальний огляд згорткових нейронних мереж

2.2.1 Основні поняття згорткових нейронних мереж

Згорткові нейронні мережі (convolutional neural networks або CNN) – тип нейронних мереж, що особливо ефективний при обробці зображень, часових рядів, тексту, аудіо, відео та інших типів даних, що структуровані у часі чи

просторі. Найчастіше згорткові нейромережі використовуються саме для роботи з зображеннями. Для того, щоб мережу можна було назвати згортковою достатньо, щоб у ній був присутній принаймні один згортковий шар. Згорткові мережі працюють з зображенням як з матрицею пікселів, а не розтягнутим у вектор пікселів зображенням.

Зображення буде представлене як багатовимірний масив чисел. Кожне число відповідатиме яскравості пікселя, де 0 – нульова яскравість, а 1 – максимальна яскравість. Такий масив називається тензором, що є математичним узагальненням матриці, коли кількість її вимірів більша за 2. Є дві конвенції щодо форми цього тензора: (H, W, C) або (C, H, W), де H та W – висота та ширина зображення у пікселях, а C – кількість кольорових каналів. Чорно-білі зображення мають один канал, а кольорові RGB зображення – три канали (Red, Green та Blue). Перший варіант (H, W, C) є більш звичним і поширеним, але тут всюди буде використано саме другу конвенцію (C, H, W), бо саме вона застосована у фреймворці PyTorch [16], що використовується у цій роботі. Тому, наприклад, кольорова фотографія з роздільною здатністю 224×224 пікселя буде представлена тензором з формою $(3, 224, 224)$.

Архітектура та особливості CNN були натхненні тим, як працює зорова кора головного мозку. Цей клас нейронних мереж має декілька суттєвих переваг порівняно зі звичайними нейромережами:

1. Здатні вивчати просторові інваріантні ознаки, які залишаються стійкими при зміні положення об’єктів на зображені;
2. Здатні працювати з зображеннями великої роздільної здатності, в яких десятки, сотні тисяч чи навіть мільйони пікселів;
3. Використовують значно меншу кількість параметрів ніж багатошарові персептрони, щоб досягти схожого результату. Це відбувається за допомогою частково зв’язних шарів та поділу параметрів.

2.2.2. Згортковий шар

Згортковий шар (convolutional layer) – це основний шар у згортковій нейронній мережі. У згортковій мережі їх зазвичай доволі багато. Його особливістю є те, що кожен нейрон цього шару пов’язаний лише з невеликою кількістю нейронів або пікселів попереднього шару, що утворюють його власне зазвичай квадратне рецептивне поле, а не з усіма нейронами попереднього шару, як це є в повнозв’язних шарах.

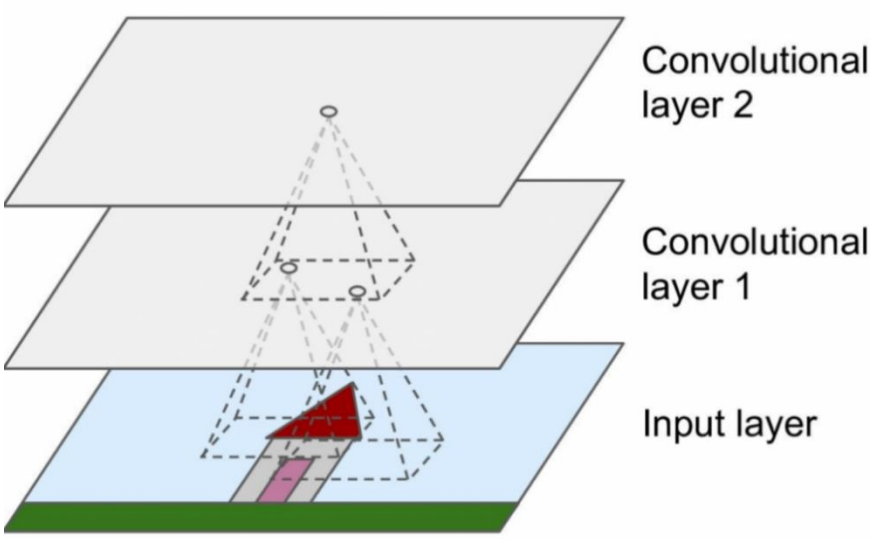

Рисунок 2.8 – Рецептивне поле нейронів у згорткових шарах [25]

Усі нейрони одного шару мають однакові розміри рецептивних полів. Більш того, ваги зв’язків, що поєднують нейрони з їх рецептивними полями також є однаковими серед усіх нейронів одного шару. Ці ваги утворюють матрицю, що називається ядром (kernel) або фільтром (filter). Розміри фільтрів зазвичай є непарними числами (наприклад 3×3), бо тоді є центральний піксель, а також тоді легше обробляти краї зображень.

Операцію, що виконує згортковий шар можна представити наступним рисунком:

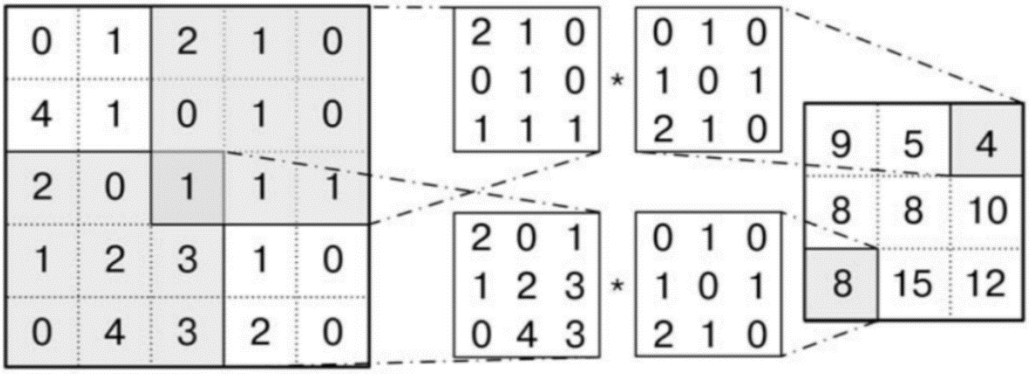

Рисунок 2.9 – Операція згортки над матрицею зображення

Фактично ядро (у даному випадку 3×3) рухається по зображенню розміром 5×5. В кожній окремій позиції ядра вираховується значення нейрона наступного шару шляхом скалярного добутку (або іншими словами поелементне множення) його рецептивного поля на ядро. Потім до отриманого зображення (тобто до кожного його пікселя) може додаватися зміщення (bias). Ця операція, цей дуже специфічний скалярний добуток називається згорткою, хоча формально кажучи правильніше це називати взаємною кореляцією (cross-correlation), бо ми не перевертаємо фільтр. Результатом згортки буде інша матриця чисел, що називається картою ознак (feature map), у якій більші значення відповідають тим рецептивним полям які більше схожі на фільтр. Отже згортка шукатиме ті ознаки, які схожі на фільтр.

До цього було розглянуто випадок, коли на вхід згорткового шару подається лише одна карта ознак чи зображення з одним каналом. Насправді ж згорткові шари працюють одразу з декількома вхідними картами ознак і здатні також повертати декілька карт ознак. Ці карти ознак складаються одна на одну вздовж осі каналів. Працює тут все аналогічно як і у розглянутому вище випадку. Нехай на вхід згорткового шару подається C_{in} карт ознак розміру $H \times W$. Тоді розмір одного фільтра буде (C_{in}, H_f, W_f) де H_f, W_f – висота і ширина фільтра у пікселях. Рецептивне поле тепер також буде мати глибину C_{in}. В результаті згортки отримаємо єдину карту ознак. Якщо бажаємо, щоб виходом була не одна карта

ознак, а C_{out} карт ознак, то тоді треба застосувати C_{out} різних фільтрів з різними вагами, але того самого розміру.

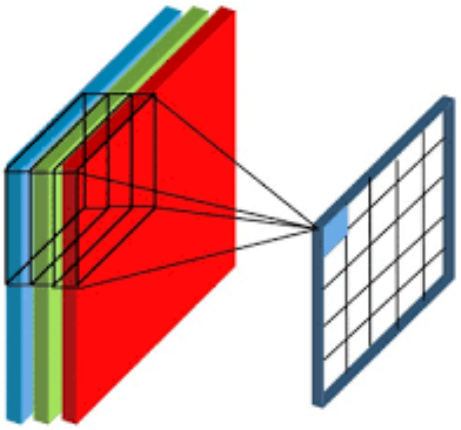

Рисунок 2.10 – Згортка одного фільтра з трьома картами ознак

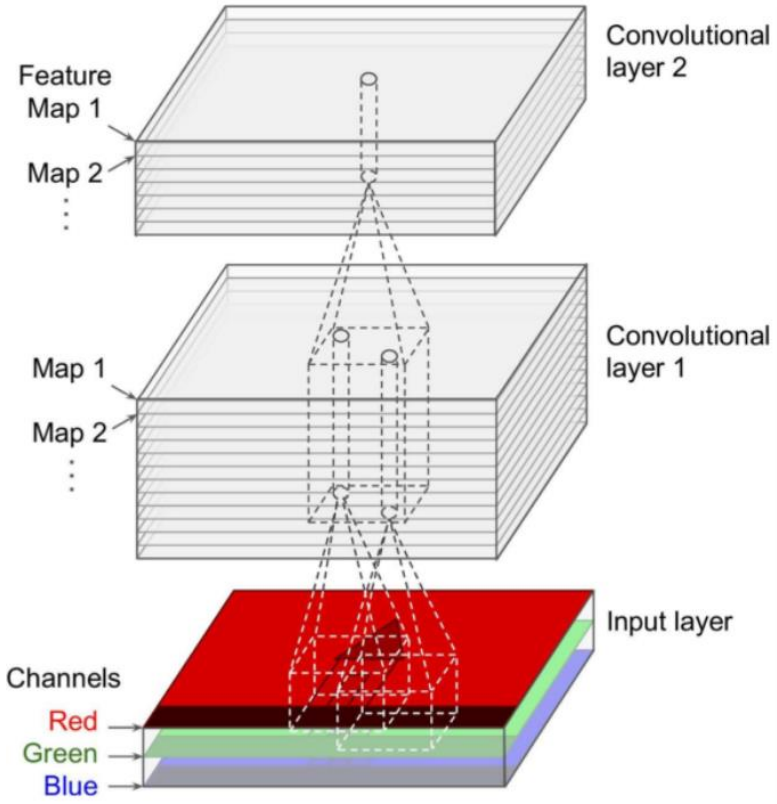

Рисунок 2.11 – Стопка карт ознак [25]

Припустимо на вхід згорткового шару подається тензор (C_{in}, H, W), де C_{in} – кількість вхідних карт ознак, H, W – розмір цих карт. Хочемо щоб повернулось

C_{out} карт ознак, тобто хочемо застосувати C_{out} різних фільтрів. Тоді операція, що виконується згортковим шаром, може бути описана наступним чином [16]:

$$\text{out}\left(C_{out_j}\right) = \text{bias}\left(C_{out_j}\right) + \sum_{k=0}^{C_{in}-1} \text{filter}\left(C_{out_j}, k\right) * \text{fmap}(k) \qquad (2.15)$$

де out – вихідні карти ознак;

C_{out_j} – j-та вихідна карта ознак;

bias – зміщення;

k – номер вхідної карти ознак;

filter – фільтр, ядро згортки;

* – операція взаємної кореляції;

fmap – вхідні карти ознак.

Згортковий шар виконує лінійну операцію, після нього застосовується функція активації, як і у випадку повнозв'язних шарів. Ця функція застосовується незалежно до кожного значення кожної карти ознак. Найпоширенішою активаційною функцією у згорткових шарах є ReLU та її модифікації. Зазвичай декілька згорткових шарів з функціями активацій слідують один за одним. Перші шари вивчають прості ознаки, тоді як наступні шари вивчають все більш складні і складні ознаки, що утворені з простіших.

2.2.3 Страйд і паддінг

Згортковий шар має також два важливих гіперпараметра – страйд та паддінг.

Страйд (stride) – число чи пара чисел, що визначає розмір кроку фільтру, що рухається по вхідному об'єму карт ознак, по вертикалі та по горизонталі. Це відстань у пікселях між двома послідовними рецепторними полями.

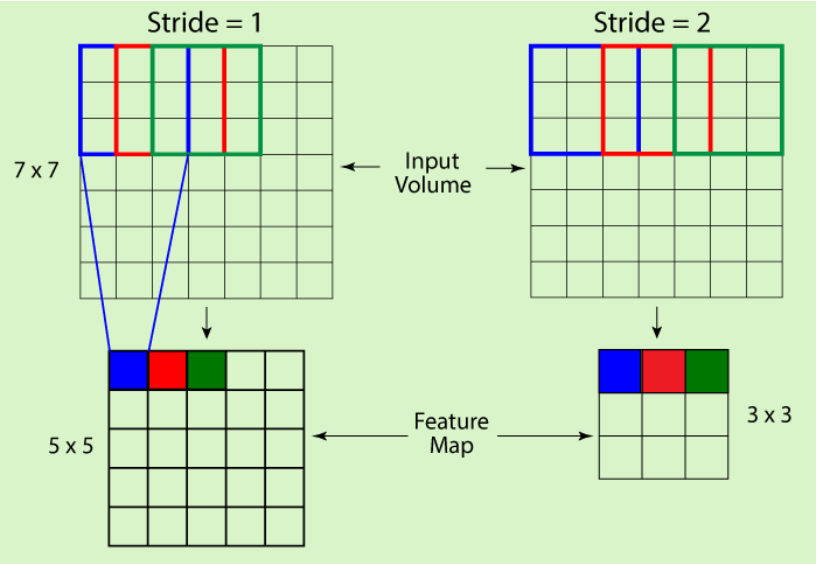

Рисунок 2.12 – Ілюстрація роботи згортки з різними страйдами

Паддінг (padding) – це техніка додавання додаткових рядків і стовпців пікселів по краям карт ознак перед застосуванням згортки. Ці додаткові пікселі зазвичай заповнені нулями і утворюють наче рамку. Ця техніка необхідна, бо інакше б кожен згортковий шар зменшував розміри карт ознак. Завдяки такому штучному розширенню карт ознак після згортки зберігається оригінальний розмір карт ознак. Згортка без паддінгу називається valid згорткою. Згортка з таким паддінгом, що зберігає розмір карт ознак, називається same згорткою.

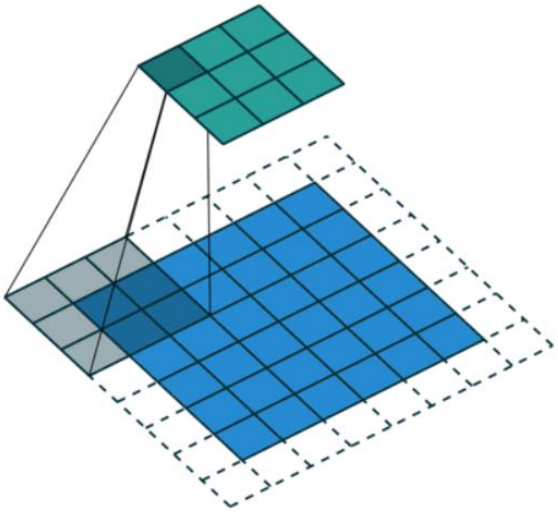

Рисунок 2.13 – Одиничний паддінг [26]

Розмірність вихідної карти ознак можна визначити з формули [16]:

$$n_{out} = \left[\frac{n_{in} + 2p - k}{s}\right] + 1 \quad (2.16)$$

де n_{out} – вихідна розмірність;

[ ] - ціла частина;

n_{in} – розмірність вхідних карт ознак;

p – паддінг, кількість додаткових пікселів що доповнюють зображення;

k – розмір фільтра згортки;

s – страйд.

2.2.4 Згортка 1×1

Особливим випадком є згортка з розміром фільтра 1×1. Цей фільтр обробляє лише по одному значенню в однакових позиціях з кожної карти ознак. Така згортка не здатна захопити просторові ознаки, але вона здатна перетворити та змінити відношення між ознаками по каналам. Цілі згортки 1×1:

- Змінити кількість карт ознак не змінюючи їх просторовий розмір при цьому витративши відносно невелику кількість параметрів;
- Для зменшення параметрів у наступному згортковому шарі. Якщо карт ознак дуже багато, то навіть невеликі фільтри 3×3 чи 5×5 матимуть багато параметрів. В такому випадку може бути доцільно зменшити вхідну кількість карт ознак, що значно зменшить кількість параметрів та обчислень у фільтрах наступного згорткового шару;
- Згортка 1×1 в поєднанні з наступним згортковим шаром 3×3 чи 5×5 діє як єдиний потужний згортковий шар, що може захоплювати значно складніші образи.

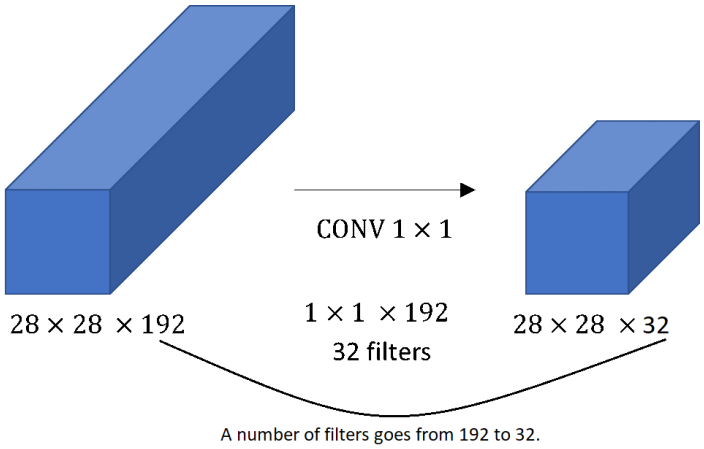

Рисунок 2.14 – Застосування згортки 1×1 для зменшення числа карт ознак

2.2.5 Шар пулінгу

Шар об’єднання або пулінгу (pooling / subsampling layer) – це особливий шар, що, як і згорткові шари, зустрічається у CNN. Він використовується з метою зменшення розмірів карт ознак шляхом об’єднання виходів кількох нейронів, об’єднання кількох пікселів однієї карти ознак в одне значення. Через пулінг втрачається деяка кількість інформації. З іншого боку зменшуються розміри карт ознак, зменшується перенавчання і кількість параметрів та збільшується стійкість мережі до невеликих зсувів і поворотів зображення. Так само як і в згортковому шарі є рухоме вікно певного розміру, що рухається по картам ознак з певним страйдом. В кожній окремій позиції цього вікна відбувається певне агрегування значень карти ознак в межах вікна. Пулінг діє на кожну карту ознак окремо. На відмінну від згортки, пулінг впливає лише на розміри карт ознак, він не змінює їх кількість. Пулінг не має ніяких параметрів, яким необхідне навчання. Існує два види пулінгу: максимальний та середній.

Максимальний пулінг (max pooling) - агрегує значення у вікні шляхом вибору лише одного максимального значення у цьому вікні. Інші значення втрачаються.

Середній пулінг (average pooling) – агрегує значення у вікні шляхом взяття їх середнього.

Максимальний пулінг використовується значно частіше за середній. Страйд зазвичай дорівнює розміру вікна пулінгу. Нижче наведено зображення дії максимального і середнього пулінгу з розміром вікна 2×2 і страйдом 2.

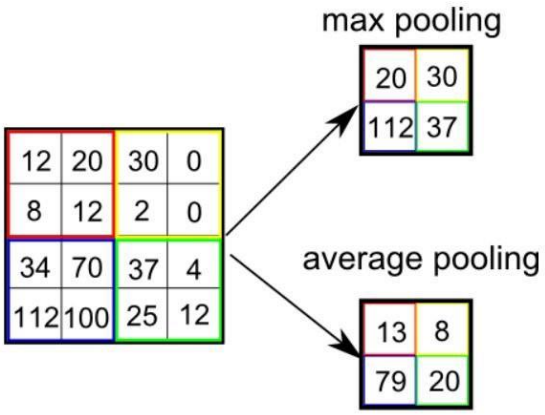

Рисунок 2.15 – Застосування різних пулінгів

Також виділяють глобальний середній і глобальний максимальний пулінги (global average / max pooling). Такий пулінг полягає в тому, розмір вікна встановлюється рівним розміру карт ознак. І тому кожна карта ознак стискається до одного числа. Тобто, якщо на вхід подається тензор карт ознак розмірності (C_{in}, H, W), то після глобального пулінгу повернеться тензор $(C_{in}, 1,1)$. Середній глобальний пулінг використовується значно частіше за максимальний. Глобальний пулінг, якщо використовується, то зазвичай лише один раз в кінці усіх згорткових шарів перед повнозв'язними.

2.2.6 Загальна архітектура згорткових нейронних мереж

Останніми шарами згорткової мережі зазвичай є один чи декілька повнозв'язних шарів. По-перше, повнозв'язний шар слугує вихідним шаром. По-друге повнозв'язні шари дозволяють поєднувати значення з карт ознак, навіть

якщо ці ознаки розташовані далеко одна від одно. Це дозволяє вивчати дуже складні образи, такі як наприклад «собака» чи «сумне обличчя». Згорткові шари необхідні для виявлення різних ознак, тоді як в кінці мережі повнозв'язні шари поєднують знайдені ознаки для пошуку складних образів.

Входом повнозв'язних шарів є вектор, тоді як виходом згорткових шарів та пулінгу є багатовимірні тензори. Тому безпосередньо перед першим повнозв'язним шаром розташовується розгладжуючий шар (flatten layer), який розтягує вхідний тензор у одновимірний вектор.

Загалом згорткова нейронна мережа виглядатиме наступним чином:

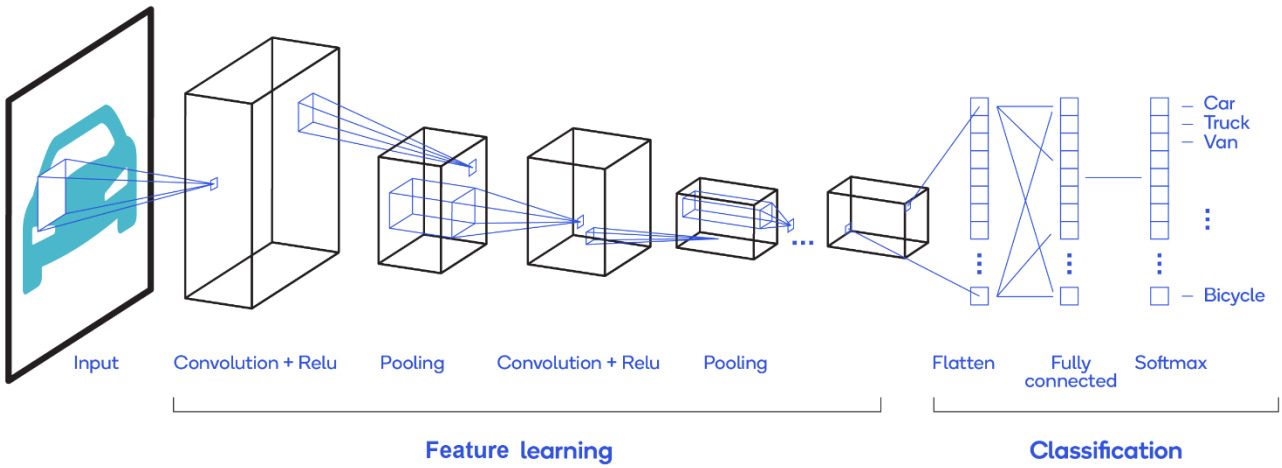

Рисунок 2.16 – Типова структура CNN.

Спершу слідує «згорткова» частина, що складається з комбінації різних згорткових шарів та шарів пулінгу. Потім остання карта ознак розтягується flatten шаром, за цим слідує декілька прихованих повнозв'язних шарів (цілком можливо, що прихованих повнозв'язних шарів не буде). Останнім вихідним шаром буде повнозв'язний шар, що у випадку багатокласової класифікації матиме функцію активації softmax та число нейронів, що дорівнюватиме кількості класам, що необхідно розрізняти.

2.2.7 Transfer learning

Трансферне навчання (transfer learning) – це метод навчання, в якому вже навчена модель машинного навчання використовується як початкове наближення, відправна точка для навчання нашої власної моделі. Ми переносимо знання з іншої навченої моделі у нашу.

Зазвичай такі готові моделі навчені на дуже великих загальних датасетах, наприклад на ImageNet. Цей набір даних містить 1.28 мільйона тренувальних зображень, 50 тис. валідаційних та 100 тис. тестових. Зображення розбиті на 1000 різних категорій. Навчені на такому датасеті моделі здатні розпізнавати дуже широкий клас різних складних ознак, і ці знання можна використати для власних цілей.

У випадку згорткових мереж зазвичай роблять наступне:

1. Ініціалізується вже навчена на загальному датасеті модель;
2. Останні повнозв’язні шари замінюються власними, щоб адаптувати модель для власного датасету, у якого кількість класів може бути іншою;
3. Відбувається донавчання моделі саме для нашої специфічної задачі.

Донавчання є необхідним, бо пренавчена мережа навчена розпізнавати широкий спектр різних ознак, тоді як для нашої задачі можливо потрібні більш специфічні ознаки. Третій крок може відбуватися по різному. Може навчатися вся мережа одразу. Іншим варіантом є заморозка згорткових шарів і навчання лише повнозв’язних.

Трансферне навчання значно полегшує та прискорює навчання нейронних мереж, особливо якщо мережа є складною, а навчальний набір даних невеликим.

2.3 Відомі архітектури згорткових нейронних мереж

2.3.1 ResNet

ResNet (Residual Network) – глибока згорткова нейронна мережа, що була представлена в 2015 році [7]. Немає єдиного набору цифр, який б міг описати якість роботи цієї мережі на ImageNet, бо було опубліковано багато робіт, де використовуються різні техніки навчання та різне програмне забезпечення, і всі вони показують трохи різні результати. Тому для ResNet та інших архітектур тут будуть наведені характеристики пренавчених на ImageNet моделей доступних у фреймворці PyTorch. Всього є декілька «видів» класичної ResNet: ResNet-18, ResNet-34, ResNet-50, ResNet-101 та ResNet-152, де цифра означає загальну кількість шарів у відповідній мережі. Їх характеристики наведені у таблиці нижче. Приставка -ntr (new training recipe) у назві моделі означає, що ця модель була навчена з використанням більш складної та просунутої навчальної стратегії.

Таблиця 2.1 – Характеристики ResNet та їх результати на ImageNet [15]

Модель	Top-1 acc %	Top-5 acc %	К-ть парам.	GFLOPS
ResNet-18	69.758	89.078	11.7M	1.81
ResNet-34	73.314	91.42	21.8M	3.66
ResNet-50	76.13	92.862	25.6M	4.09
ResNet-50-ntr	80.858	95.434	25.6M	4.09
ResNet-101	77.374	93.546	44.5M	7.8
ResNet-101-ntr	81.886	95.78	44.5M	7.8
ResNet-152	78.312	94.046	60.2M	11.51
ResNet-152-ntr	82.284	96.002	60.2M	11.51

Навчання глибоких нейронних мереж, у яких десятки чи тим більше сотні шарів, може стати дуже складною задачею через проблему зникаючого градієнту. Ця проблема полягає в тому, що при зворотному розповсюджені помилки градієнти стають меншими з кожним шаром. Через це перші шари мережі навчаються дуже повільно, або в найгіршому випадку не навчаються взагалі. Розробники ResNet запропонували революційну ідею використання пропускних зв'язків (skip / residual connections), що практично повністю усувала проблему зникаючого градієнту та дозволяла навчати мережі необмеженої глибини [7]. ResNet складається з багатьох залишкових блоків, будова яких наведена нижче.

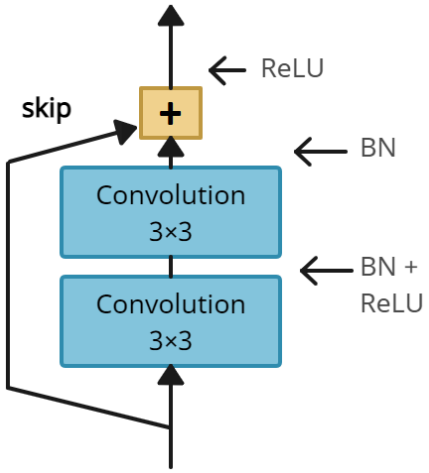

Рисунок 2.17 – Residual Block

Залишковий блок складається з двох згорткових шарів, після кожного з яких є batch normalization та активація ReLU. Автори ResNet запропонували перед активацією другого шару до тензору додавати вхідний сигнал першого шару [7]. Таким чином цей вхідний сигнал наче пропускає одразу два згорткових шари і додається до виходу другого шару. Це виражається наступним виразом [7]:

$$y = \mathrm{ReLU}(H(x) + x) \tag{2.17}$$

де y – вихідний тензор залишкового блоку;

$H(\,)$ - перетворення, що здійснюють два згорткових шари;

x – вхідний тензор до залишкового блоку.

ResNet складається з великої кількості таких блоків, що складені разом в одну велику мережу. Потім йде глобальний середній пулінг і одразу повнозв'язний вихідний шар з активацією softmax. Якщо в залишковому блоці зменшується розмір і подвоюється кількість карт ознак, то skip connection не просто переносить сигнал до виходу другого шару, а й також додатково робить відповідні зміни карт ознак за допомогою згортки 1×1 зі страйдом 2 і подальшим batch normalization.

У мережах ResNet-50, 101 та 152 структура залишкового блоку трохи інакша. Ці блоки називаються Bottleneck через їх особливу будову [7]. Замість двох згорткових шарів там три. Перший - це згортка 1×1, що зменшує кількість карт ознак. Другий – це звичайна згортка 3×3. І третій – це знову згортка 1×1, що збільшує кількість карт до початкової кількості. Застосування згорток 1×1 необхідно, щоб зменшити кількість параметрів та обчислень у залишковому Bottleneck блоці. Skip connection не змінився, єдине, що він тепер пропускає три шари, а не два. Детальна будова ResNet з різною кількістю шарів у таблиці нижче.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				

Рисунок 2.18 – Будова різних ResNet [7]

Skip connections значно покращують навчання та усувають проблему зникаючих градієнтів, бо не дозволяють їм затухати. Якщо деякий шар дуже сильно зменшує градієнти, то вони оминуть його за допомогою skip connection, і навчання більш ранніх шарів не зупиниться. Градієнти можуть пропускати будь-які шари, що викликають їх затухання. Окрім того, завдяки пропускному зв'язку дуже просто навчити residual block функції ідентичності, що принаймні не погіршує результат. А при подальшому навчанні residual block навчиться кориснішим функціям і покращить результати.

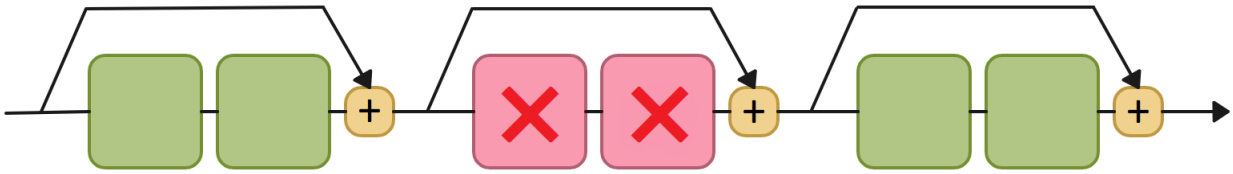

Рисунок 2.19 – Skip connection пропускає шари, що заважають навчанню

2.3.2 MobileNet

MobileNet – мережа, що спеціально розроблена для мобільних пристроїв, що мають обмежені обчислювальні ресурси, і була представлена в 2017 році [8]. Її удосконалені версії MobileNet-v2 [9] та MobileNet-v3 [10] були представлені відповідно у 2018 та 2019 роках. Їх результати можна побачити нижче. Як і у випадку ResNet -ntr (new training recipe) означатиме, що відповідна модель була навчена з використанням більш складної та просунутої навчальної стратегії. MobileNet-v1 не була реалізована в PyTorch, бо її наступниці в усьому її перевершують. Згідно оригінальної статті MobileNet-v1 має top-1 accuracy 70.6% та 4.2 мільйони параметрів [8].

Таблиця 2.2 – Характеристики MobileNet та їх результати на ImageNet [15]

Модель	Top-1 acc %	Top-5 acc %	К-ть парам.	GFLOPS
MobileNet-v2	71.878	90.286	3.5M	0.3
MobileNet-v2-ntr	72.154	90.822	3.5M	0.3
MobileNet-v3	74.042	91.34	5.5M	0.22
MobileNet-v3-ntr	75.274	92.566	5.5M	0.22

Головною особливістю MobileNet є заміна звичайної згортки 3×3 на особливу згортку, що має назву depth-wise separable convolution [8]. Така заміна трохи погіршує якість прогнозів, але в багато разів зменшує кількість параметрів та кількість обчислень при прогнозуванні.

Depth-wise separable згортка складається з двох етапів: depth-wise згортки та pointwise згортки.

Нехай надійшов тензор (C_{in}, H_{in}, W_{in}), де C_{in} – кількість вхідних карт ознак, H_{in} та W_{in} – їх розміри. Спершу виконується depth-wise згортка. Вона застосовує C_{in} різних двовимірних фільтрів $1 \times k \times k$ до кожного каналу зображення окремо. В результаті отримуємо тензор розмірністю $(C_{in}, H_{out}, W_{out})$. Depth-wise згортка не змінює кількість карт ознак, бо діє на кожну з них незалежно.

Далі використовується pointwise згортка. Нехай на вхід подається тензор після depth-wise згортки $(C_{in}, H_{out}, W_{out})$. До цього тензору застосовуються C_{out} різних фільтрів розмірності $C_{in} \times 1 \times 1$ зі страйдом 1 за звичайними правилами згортки. Результатом буде тензор розмірності $(C_{out}, H_{out}, W_{out})$. Pointwise згортка не змінює розміри карт знак, вона лише змінює їх кількість.

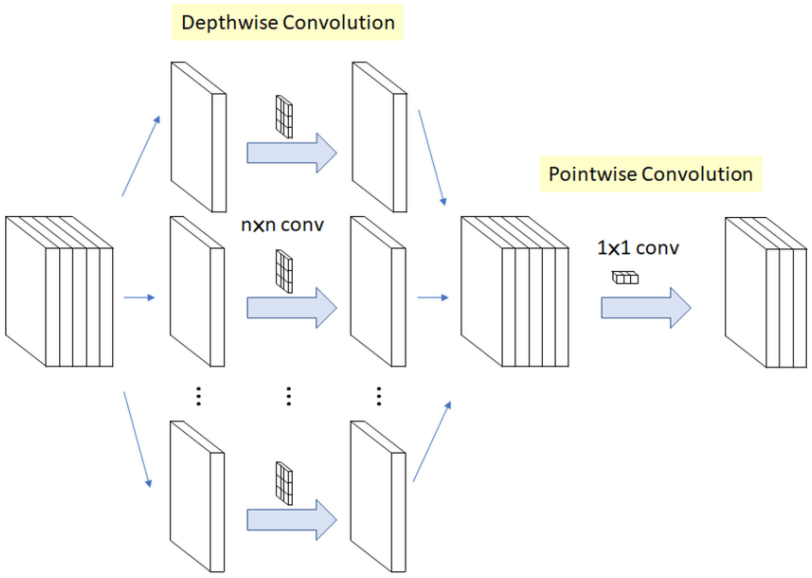

Рисунок 2.20 – Depth-wise separable convolution

Авторами MobileNet було підраховано, що ступінь скорочення обчислень в depth-wise separable згортці становить $\frac{1}{C_{out}} + \frac{1}{k^2}$, де k – розмір фільтра depth-wise згортки [8].

MobileNet-v1 використовує 13 блоків depth-wise separable згортки, за якими слідують глобальний середній пулінг та вихідний повнозв'язний шар softmax з 1000 нейронами. У кожному блоці після кожної згортки використовується batch normalization та активація ReLU [8].

В MobileNet-v2 використовуються основні ідеї попередниці, а саме depth-wise separable згортка. Також було внесено додаткові зміни, що ще більше прискорили мережу, зменшили кількість параметрів і одночасно збільшили її точність.

Головним структурним елементом MobileNet-v2 є inverted residual block [9]. Входом і виходом цього блоку є тензори з відносно невеликою кількістю карт ознак. Спершу застосовується pointwise 1×1 згортка, що збільшує кількість карт ознак у t разів (в оригінальній статті всюди $t = 6$) [9]. Вона має назву expansion layer. Це робиться для того, щоб збагатити тензор новими ознаками. Потім застосовується depth-wise 3×3 згортка, що тепер обробить більшу кількість

цікавих характеристик ніж зазвичай, а отже і навчиться виділяти цікавіші ознаки. Потім знову застосовується pointwise 1×1 згортка, що стискає кількість карт ознак до попередньої. Вона називається projection layer. Автори MobileNet-v2 емпіричним шляхом показали, що можливо стиснути збагачений тензор до тензора з меншою розмірністю без втрати цих цікавих ознак [9].

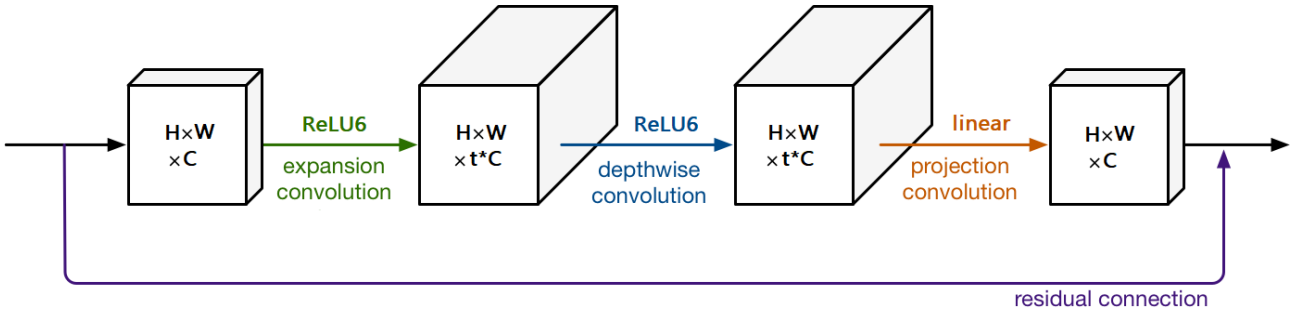

Рисунок 2.21 – Inverted residual block

Після кожної згортки застосовується batch normalization, а після першої pointwise та depth-wise згорток – ще активація ReLU6. Вона відрізняється від звичайної ReLU лише тим, що усі при x $>$ 6 вона поверне 6, а не x. Таким чином ReLU6 можна представити як $min(max(x, 0), 6)$. Застосування цієї активації було аргументоване тим, що ReLU6 більш стійка при обчисленнях з низькою точністю (наприклад при використанні чисел з фіксованою точкою) [9]. Друга pointwise згортка (bottleneck layer) не використовує функцію активації, для того щоб зберегти знайдені цікаві ознаки (інакше б ReLU6 обнулив значну частину з них). Також у inverted residual block використано skip connection з метою, що вже описана у розділі 2.3.1 про ResNet.

MobileNet-v2 складається з 17-ти inverted residual блоків. За ними слідує expansion layer з batch norm та ReLU6, глобальний середній пулінг, dropout з ймовірністю 0.2, та повнозв'язний шар softmax з 1000 нейронами [9].

MobileNet-v3 має схожу архітектуру на MobileNet-v2, він також використовує inverted residual block. Кількість цих блоків дорівнює 15, а також вони мають інші параметри [10]. Основні відмінності [10]:

- Функцією активації є swish, що виражається як $x \cdot \sigma(x)$, де $\sigma(x)$ – сигмоїда. Проте обчислення сигмоїди потребує багато ресурсів, тому її замінили аналогом, що є обчислювально менш затратним, а саме $x\frac{\mathrm{ReLU6}(x+3)}{6}$. Експериментально було показано, що ця активаційна функція показує кращі результати за ReLU чи ReLU6.
- В деяких блоках одразу після depth-wise згортки виконується додаткова дія “Squeeze-and-Excite”. Цей крок акцентує увагу на інформативних картах ознак і пригнічує непотрібні. Спершу застосовуємо глобальний середній пулінг до тензору, отримавши по одному значенню для кожної ознаки. Потім застосовується невелика двошарова повнозв’язна мережа з сигмоїдною функцією активації в кінці, що повертає ступінь важливості кожної карти ознак. Потім кожна карта ознак тензору множиться на відповідне їй отримане число інформативності. “Squeeze-and-Excite” дуже схожий на Channel Attention, що застосовується в CBAM, котрий буде детально розглянутий у наступному розділі.
- Останні шари були перероблені для оптимізації часу обчислень. Останній extension layer був винесений після глобального середнього пулінгу, а останні шари depth-wise та projection були видалені.

2.4 Convolutional Block Attention Module

Convolutional Block Attention Module, або скорочено CBAM – це особливий обчислювальний блок у згорткових нейронних мережах, що був запропонований у 2018 році [11]. Він здатен покращити прогнози моделей шляхом акцентування уваги на цікавих інформативних ознаках, а також на інформативних ділянках зображення. Цих блоків може бути декілька у мережі, і кожен CBAM може бути вставлений після будь-якого згорткового шару. Нижче зображено загальну будову CBAM.

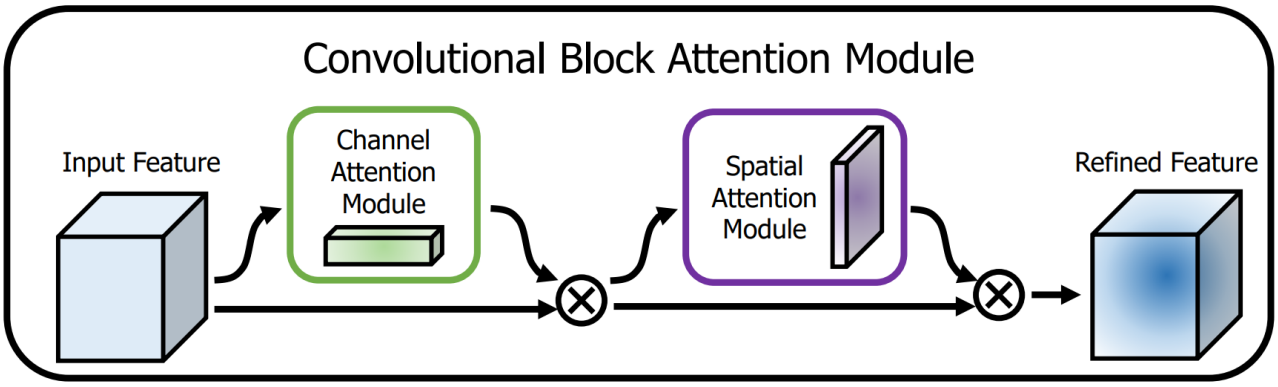

Рисунок 2.22 – Структура CBAM [11]

CBAM складається з двох кроків. Нехай на вхід подається тензор карт ознак розмірністю (C, H, W). Спершу на основі цього тензору обчислюється карта важливості ознак (channel attention map) M_c розмірністю $(C, 1, 1)$, що множиться на вхідний тензор. Потім на основі результату попередньої дії знаходиться карта, що вказує цікаві ділянки (spatial attention map) M_s розмірністю $(1, H, W)$ і знову множиться поелементно. Цей процес можна описати наступним чином [11]:

$$F' = M_c(F) \cdot F \tag{2.18}$$

$$F'' = M_s(F') \cdot F' \tag{2.19}$$

де M_c та M_s – channel attention map та spatial attention map відповідно;

F – вхідний тензор карт ознак;

F'' – вихідний результуючий тензор того самого розміру;

$\cdot$ – поелементне множення.

Channel Attention Module обчислює channel attention map на основі внутрішніх взаємозв'язків між різними картами ознак. Цей модуль акцентує увагу на тому «що» є важливим [11]. В наступному абзаці описано як обчислити channel attention map [11].

Спершу до вхідного тензору застосовується одночасно глобальний середній та глобальний максимальний пулінги повертаючи одразу два тензори F_{avg}^{C} та F_{max}^{C} відповідно, розмірності $(C, 1,1)$ кожен. Називатимемо їх далі дескрипторами. Автори CBAM стверджують та доводять експериментально, що застосування окрім середнього також максимального пулінгу дозволяє зібрати інші важливі додаткові характеристики. Потім ці два дескриптори незалежно один від одного пропускаються через двошаровий персептрон. Його єдиний прихований шар має зменшену в ratio разів кількість нейронів для зменшення кількості параметрів. Цей персептрон не змінює розмір дескрипторів і дозволяє вивчити зв'язки між ознаками. Персептрон той самий і для F_{avg}^{C} і для F_{max}^{c}, він має спільні ваги. Потім вихідні вектори ознак персептрону поелементно додаються один до одного і нарешті до суми застосовується сигмоїда, щоб стиснути значення до діапазону $(0,1)$. Результатом буде тензор channel attention map. Наведений вище опис дій можна описати наступним виразом [11]:

$$M_c(F) = \sigma\left(\mathrm{MLP}\big(\mathrm{AvgPool}(F)\big) + \mathrm{MLP}\big(\mathrm{MaxPool}(F)\big)\right) \qquad (2.20)$$

де σ – сигмоїдна функція активації;

MLP – двошаровий персептрон.

Spatial Attention Module обчислює spatial attention map вивчаючи просторові зв'язки. Цей модуль акцентує увагу на тому «де» знаходяться інформативні ділянки [11]. В наступному абзаці описано як обчислити spatial attention map [11].

Ми також спершу застосовуємо глобальні середній та максимальний пулінги, але використовуємо їх вздовж осі каналів. Тобто ми усереднюємо і шукаємо найбільше значення не в кожній карті ознак, а в кожній просторовій позиції агрегуючи значення з усіх каналів у цій позиції. Результатом буде дві 2D карти F_{avg}^{S} та F_{max}^{S} розмірністю $(1, H, W)$ кожна. Потім ми їх конкатенуємо по осі

каналів отримавши тензор розмірності $(2, H, W)$. Після цього застосовується одна звичайна згортка з одним фільтром, що повертає одну карту ознак, що і буде spatial attention map. Автори використовували фільтр 7×7 та сигмоїду як функцію активації.

Наведені вище дії можна описати наступним виразом [11]:

$$M_s(F) = \sigma\left(f^{7\times7}([\text{AvgPool}(F),\ \text{MaxPool}(F)])\right) \tag{2.21}$$

де σ – сигмоїда;

$f^{7\times7}$ – згортка з фільтром 7×7;

[ ] – конкатенація.

Авторами було експериментально доведено, що найефективніший спосіб це послідовно спершу застосувати Channel Attention Module і потім Spatial Attention Module [11]. Описані дії у Channel Attention Module та Spatial Attention Module наочно зображено на рисунку нижче.

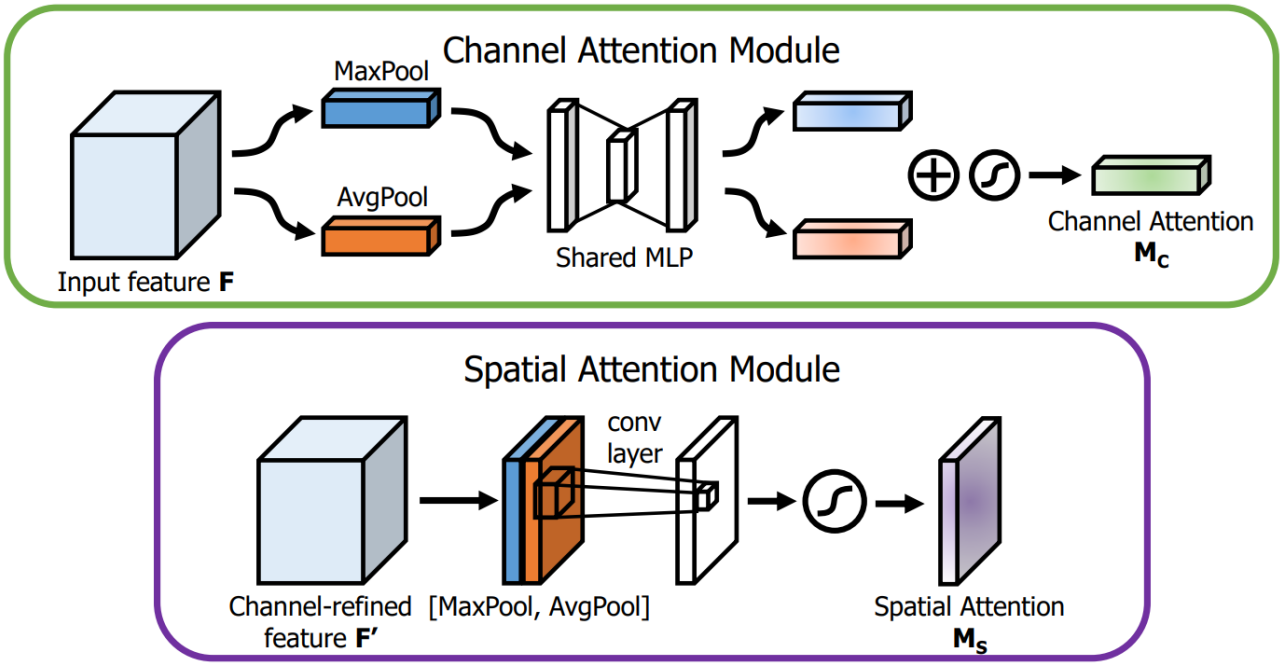

Рисунок 2.23 – Структура головних модулів CBAM [11]

CBAM можна вбудувати в будь-які згорткові мережі, в тому числі і в ті, що описані у розділі 2.3. Авторами було експериментально показано, що CBAM значно покращує якість роботи нейронних мереж та допомагає зосередити увагу на ключових ділянках зображення [11].

2.4 Gradient-weighted Class Activation Mapping

Grad-CAM (Gradient-weighted Class Activation Mapping) – метод запропонований у 2016 році, що дає візуальне пояснення рішень, які зробила нейронна мережа [12]. Цей метод виділяє частини вхідного зображення, які були важливими і мали позитивний вплив на результат. Grad-CAM може працювати з будь-якою CNN. Він не модифікує структуру мережі, а тому може вбудовуватися у вже навчені готові моделі не змінюючи їх та не змушуючи заново навчатись [12]. Цей метод базується на принципі, що карти ознак, навіть ті, що знаходяться дуже глибоко і далеко від вхідного зображення, зберігають просторову інформацію та взаємне розташування різних ознак як і на вхідному зображені.

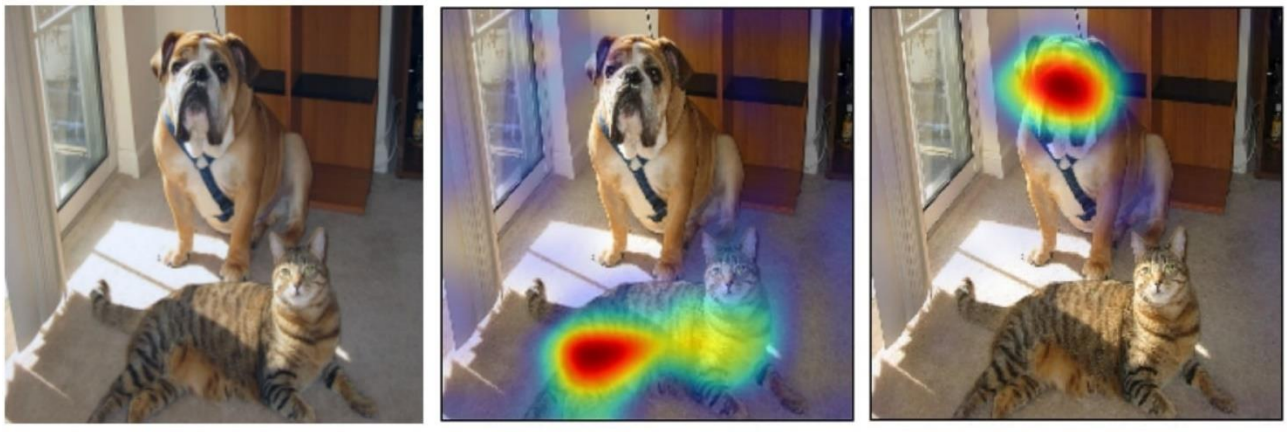

Рисунок 2.24 – Приклад роботи Grad-CAM [12]

Зазвичай Grad-CAM вивчатиме карти ознак саме останнього згорткового шару [12], оскільки саме ці ознаки безпосередньо впливають на результат

класифікації. Хоча, звісно, можна аналізувати більш ранні карти ознак, але вони будуть значно менш інформативними. Цей метод аналізуватиме як і карту ознак, так і градієнти, що проходитимуть крізь неї при back propagation. На рисунку нижче зображено роботу Grad-CAM і далі описано покроково як він працює.

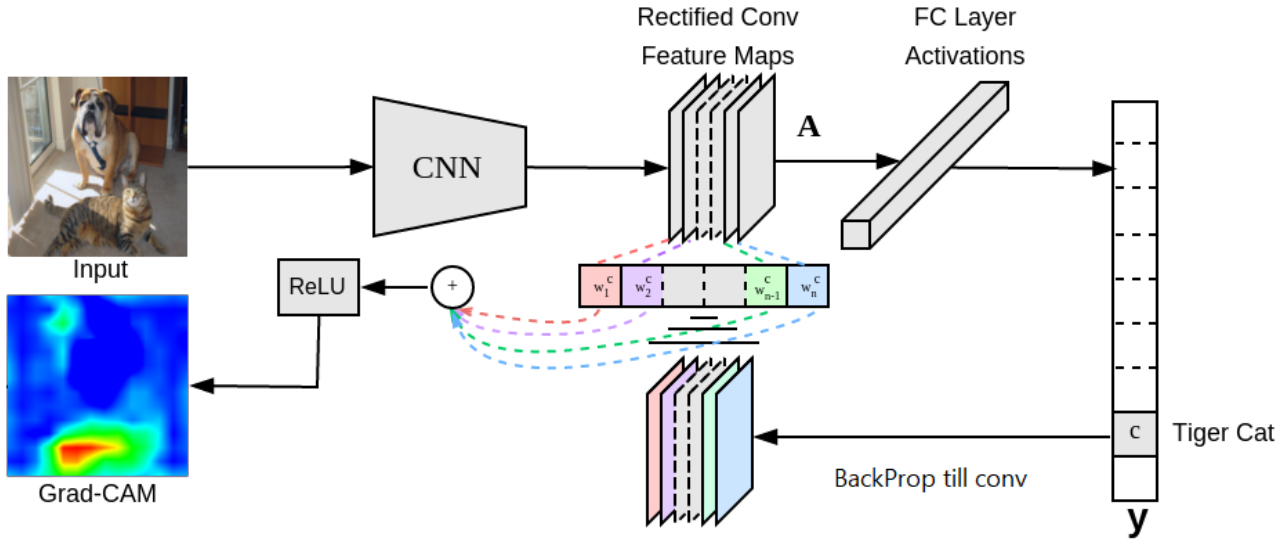

Рисунок 2.25 – Робота Grad-CAM [12]

Кроки цього методу [12]:

1. Беремо вже навчену згорткову мережу;
2. Беремо зображення та виконуємо feedforward, тобто пропускаємо зображення через мережу, щоб його класифікувати. В результаті отримаємо вектор y. Карти ознак, що аналізуємо, і що є результатом k-го згорткового шару, зберігаємо. Назвемо тензор, що містить ці карти, як A_k і нехай він матиме розмірність (C, H, W);
3. Обираємо цільовий клас c (на рисунку 2.25 це tiger cat), який досліджуємо. Зазвичай це клас, до якого належить зображення з найбільшою ймовірністю;
4. Виконуємо back propagation крок. Спершу встановлюємо градієнт рівний 0 для всіх класів окрім цільового c, для якого він дорівнює 1. Потім обчислюємо градієнти вектору y по відношенню до тензора карт ознак A_k.

Цей градієнт буде тензором такої ж розмірності, як і сам A_k, тобто матиме форму (C, H, W);

5. Агрегуємо інформацію з отриманих градієнтів для кожної відповідної ознаки. Для цього застосовуємо глобальний середній пулінг по просторовим ознакам градієнту отримавши тензор форми $(C, 1,1)$, що назвемо α_k^c. Кроки 4 та 5 можуть бути записані наступним виразом:

$$\alpha_k^c = \text{GlobAvgPool}\left(\frac{\partial y^c}{\partial A_k}\right) \tag{2.22}$$

Отриманий тензор α_k^c показує важливість кожної карти ознаки в A_k щодо класу c;

6. Множимо кожну карту ознак з A_k на відповідне їй значення важливості з α_k^c. Отримані зважені карти додаємо одна до одної вздовж осі каналів. Тобто, загалом, ми виконуємо зважену суму цих карт ознак. В результаті отримуємо тензор форми $(1, H, W)$;

7. Отримана карта вже в принципі буде виділяти важливі частини зображення. Але вона матиме як позитивні, так і негативні значення, тобто і ті ділянки, що мали позитивний внесок для отримання класу c, так і ті що навпаки мали негативний внесок. Тоді як нас цікавлять тільки ті ділянки, що мали позитивний вплив. Тому до отриманої карти застосовуємо функцію ReLU, що обнулить усі негативні значення. Кроки 6 та 7 можна записати у наступний вираз.

$$Map^c = \text{ReLU}\left(\sum \alpha_k^c A_k\right) \tag{2.23}$$

8. Отримана карта Grad-CAM матиме ті самі просторові розміри, що і тензор A_k, а тому буде значно меншою за вхідне зображення. Тому збільшуємо

карту до розмірів вхідного зображення (upsampling) використовуючи білінійну інтерполяцію. Потім вхідне фото та карта можуть накладатись об’єднуючись в єдине зображення.

2.5 Висновки до розділу 2

У другому розділі було детально оглянуто математичні методи, що використовуються у даній роботі.

Були пояснені та викладені основні поняття та принципи роботи нейронних мереж, їх навчання. Розглянуто метрики, методи регуляризації та покращення роботи мережі.

Були детально розглянуті згорткові нейронні мережі. Описано їх загальну архітектуру та будову їх специфічних шарів, таких як згорткові та пулінг. Описане трансферне навчання.

Після цього були описані деякі існуючі ефективні архітектури глибоких згорткових мереж, а саме ResNet і MobileNet. Детально описано їх будову та причини, чому вони є дуже ефективними.

Потім було описано механізм уваги, та його реалізацію у вигляді CBAM модуля, що значно покращує ефективність мереж.

В кінці було описано спосіб візуалізації найцікавіших та найінформативніших ділянок зображень за допомогою методу Grad-CAM, що може бути вбудований у вже навчені моделі без їх змін.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Вибір програмних засобів

Мовою програмування було обрано Python версії 3.10. Взагалі Python є мовою «за замовчуванням» для проєктів з машинного навчання, і вважається найкращим вибором в тому числі для розробки нейронних мереж. На це є декілька вагомих причин.

1. Простота мови. Python має зрозумілий та простий синтаксис, що значно полегшує та пришвидшує розробку продуктів на ньому.
2. Великий вибір бібліотек. Python має дуже багато різних бібліотек та фреймворків, що мають готові реалізовані алгоритми та методи для машинного навчання, математики, роботи з масивами, для графіків, обробки зображень, комп'ютерного зору та багато чого іншого. Це дуже прискорює розробку в тому числі і нейронних мереж.
3. Велика спільнота та популярність. Python є дуже популярною мовою, що має велику кількість розробників та спеціалістів. Тому в інтернеті можна знайти багато гайдів чи обговорень, що допоможуть створити власний продукт.

Проте ця мова також має принаймні один вагомий мінус, а саме швидкість. Python є значно повільнішим за інші мови, такі як, наприклад, C++. Але цей недолік частково компенсується тим, що багато бібліотек та фреймворків Python реалізують свої основні функції мовами C та C++, завдяки чому вони працюють майже так само швидко, як і програми, що повністю написані на цих мовах. До таких належать, наприклад, PyTorch, TensorFlow, NumPy та OpenCV.

Середовищем розробки було обрано Jupyter Notebook. Це інтерактивне середовище, що дозволяє програмувати на кількох мовах, в тому числі і Python.

Він реалізований як веб-інтерфейс в якому можна редагувати файли, що містять не тільки код, а й результати роботи коду, звичайний текст, зображення і графіки.

В процесі розробки були використані наступні фреймворки та бібліотеки.

1. PyTorch. Це open-source фреймворк створений для роботи і розробки Deep Learning проектів. Розроблений компанією Meta AI (колишня Facebook AI). Це ключова і головна бібліотека, що використовується в даній роботі. За її допомогою тут були сформовані та навчені різні нейронні мережі. PyTorch є інтуїтивно зрозумілим і дуже гнучким, він реалізований на C++, що також збільшує його швидкість. Також він може навчати нейронні мережі на GPU використовуючи платформу CUDA розроблену Nvidia. Це дозволяє значно пришвидшити навчання завдяки перевазі GPU над CPU при паралельних обчисленнях та операціях.
2. OpenCV. Це open-source бібліотека написана на C++. Забезпечує дуже широкий набір інструментів та методів для задач комп'ютерного зору та обробки зображень.
3. Matplotlib. Ця бібліотека використовується для візуалізації результатів та інформації, малювання графіків та діаграм.
4. Scikit-Learn. Це бібліотека для машинного навчання. Пропонує багато методів препроцесингу, класифікації, регресії, кластеризації, метрик та багато чого іншого.
5. Dlib. Бібліотека написана на C++ загального призначення, має інтерфейс Python. Має інструменти для роботи з потоками, зображеннями, структурами даних, машинним навчанням.

Також використовуватимуться декілька допоміжних бібліотек, а саме os (для доступу до файлової системи), time (для вимірювання часу навчання), PIL (для роботи з зображеннями), NumPy (для векторизованих обчислень), tkinter (для створення графічного інтерфейсу).

3.2 Препроцесинг та аугментація зображень

Препроцесинг зображень (image preprocessing) – це попередня обробка фото перед тим як подати їх до нейронної мережі чи іншого алгоритму. Це необхідний крок, що необхідний для покращення якості зображення (для нейронної мережі, а не людини) та його інформативності. Усі зображення мають різні кольорові палітри, освітлення, положення голови. Завдяки препроцесингу ми уніфікуємо усі зображення та видаляємо зайві ділянки, що робить фото більш подібними, і що дозволяє нейронній мережі зосередитись на інформативних ділянках та вивчити ознаки, що інваріантні до освітлення, положення голови і т. д. А подальша аугментація дозволяє «збільшити» розмір датасету без кардинальної зміни самих зображень.

Препроцесинг та аугментація буде виконуватися як було запропоновано в статті [13]. Авторами було експериментально показано, що саме такі дії значно покращують навчання та підвищують точність моделей при розпізнаванні емоцій. Виконуються наступні кроки.

1. Спершу за допомогою бібліотеки Dlib на обличчі визначаються 68 ключових точок. Для їх пошуку використовується вже готова навчена модель shape_predictor_68_face_landmarks. Під капотом вона використовує HOG та метод опорних векторів (support vector machine).

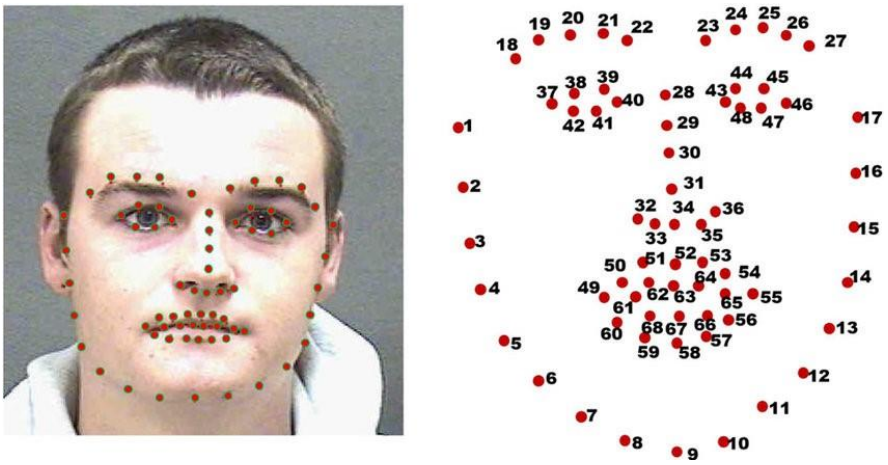

Рисунок 3.1 – 68 ключових точок на обличчі [27]

2. Вирівнювання обличчя. Для цього обчислюються центри лівого (точки 37-42) та правого ока (точки 43-48). Після цього зображення повертається навколо свого центру на такий кут, щоб центри очей були на одному горизонтальному рівні. Кут повороту визначається з наступної формули [13]:

$$\text{angle} = \arctan\frac{dY}{dX} = \arctan\frac{\sum_{i=43}^{48} y_n - \sum_{i=37}^{42} y_n}{\sum_{i=43}^{48} x_n - \sum_{i=37}^{42} x_n} \tag{3.1}$$

де x_n – x-координата n-ї точки,

y_n – y-координата n-ї точки.

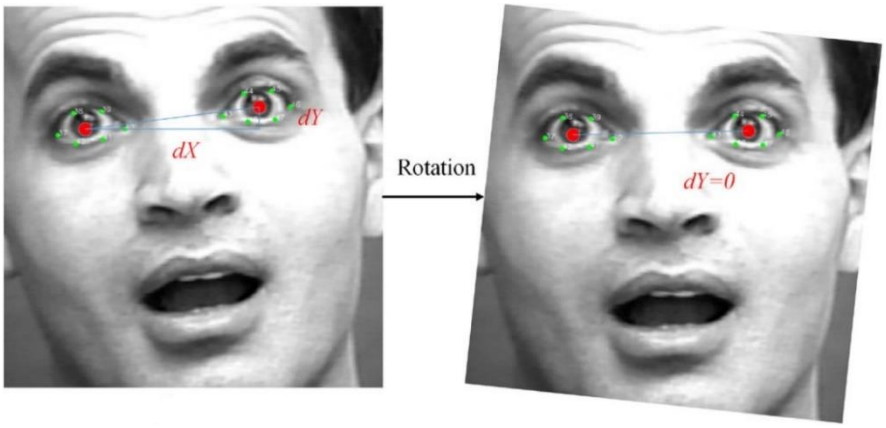

Рисунок 3.2 – Обертання зображення [13]

3. Обрізання. З вирівняного зображення видаляються усі зайві ділянки залишаючи мінімум фону та неінформативних частин голови. Спершу визначається відстань між центрами очей за наступною формулою [13]:

$$d = \frac{\sum_{i=43}^{48} x_n - \sum_{i=37}^{42} x_n}{6} \tag{3.2}$$

Потім визначається верхня горизонтальна межа обрізання. Відстань від неї до лінії, що сполучає очі становить $0.6d$. В статті [13] інші межі обрізаного зображеннями визначаються положеннями 1-ї, 9-ї та 17-ї ключових точок.

Проте мною було помічено, якщо обличчя зображено не в фас, а в три чверті чи в профіль, то тоді часто обрізаються потрібні ділянки обличчя. Для того, щоб цього уникнути, опорною точкою для лівої межі слугуватиме не x_1, а $min(x_1, x_2, x_3, x_{18}, x_{37})$, а для правої – не x_{17}, а $max(x_{15}, x_{16}, x_{17}, x_{27}, x_{46})$. Для нижньої межі без змін.

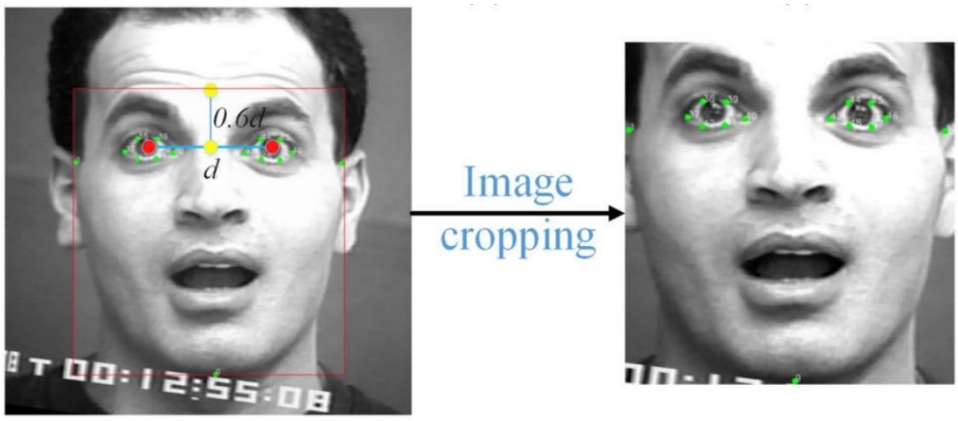

Рисунок 3.3 – Обрізання зображення [13]

4. Вирівнювання гістограми. Контраст та яскравість різних зображень є різною. Тому щоб зменшити цю різницю був застосований метод вирівнювання гістограми (histogram equalization) до кожного вже повернутого та обрізаного зображення.

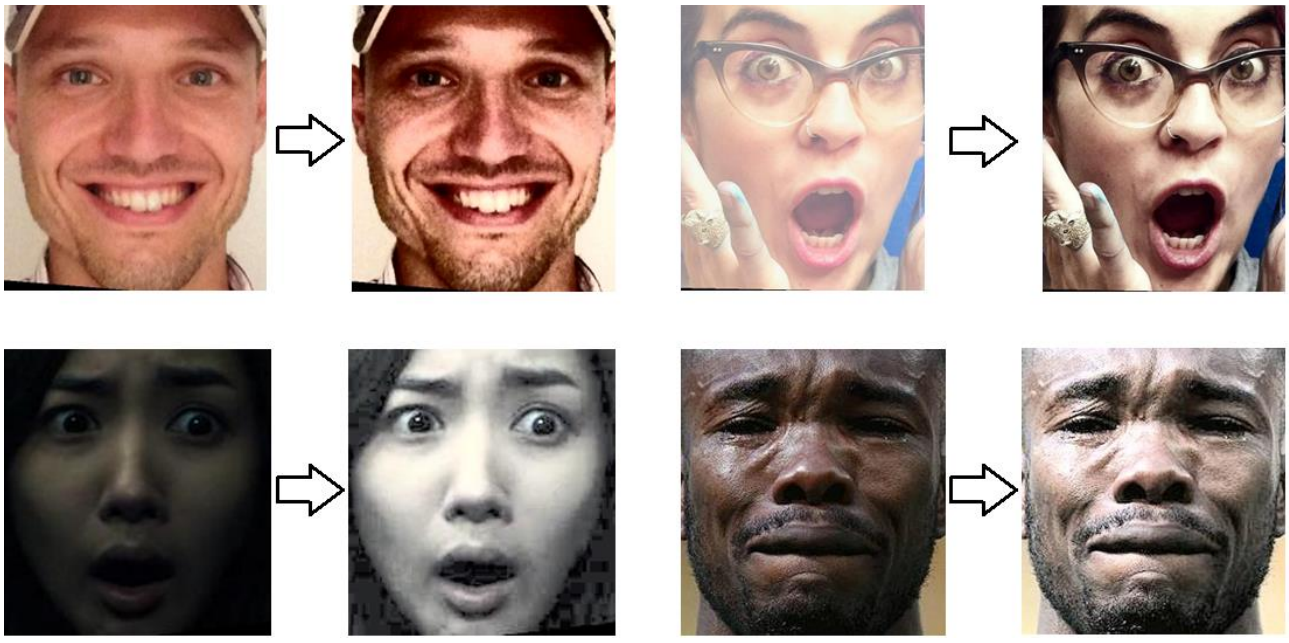

Рисунок 3.4 – Результат застосування histogram equalization

5. Нормалізація. Після цього зображення були приведені до однієї роздільної здатності. Потім увесь датасет зображень стандартизується за допомогою z-score normalization, завдяки чому він тепер має нульове середнє та одиничну дисперсію. Нормалізується кожен канал незалежно від інших каналів.

Аугментація даних також буде відбуватися як описано в статті [13]. Вона полягатиме в наступному:

1. Випадкове перевертання зображення по горизонталі (random horizontal flipping) з ймовірністю 50%.
2. Випадковий поворот на кут в межах $(-2^\circ, 2^\circ)$. Кут є невеликим, щоб не сильно спотворити зображення.

3.3 Опис навчених моделей та їх результатів

3.3.1 Загальний опис датасетів та моделей

Всього було створено 4 набори даних зображень. А саме датасет з 64×64 чорно-білими зображеннями, датасет з 64×64 кольоровими RGB зображеннями, датасет з 224×224 чорно-білими зображеннями, та датасет з 224×224 кольоровими RGB зображеннями. Розмір саме в 224 пікселя був обраний тому, що таку роздільну здатність мають зображення, що подаються на пренавчені ResNet та MobileNet. Самі зображення та розбиття на train/val/test підмножини в усіх чотирьох наборах даних є однаковими. Як було сказано в пункті 3.2, після ресамплінгу зображень до певної роздільної здатності, весь датасет стандартизується. Для цього були обчислені середні значення (mean) та стандартне відхилення (std) кожного кольорового каналу усіх зображень тренувального набору. Отримані числа на тренувальному наборі використовуватимуться для стандартизації даних і з власне тренувального датасету, і з валідаційного датасету, і з тестового.

Були отримані наступні значення. У випадку кольорових зображень маємо три числа для кожної статистики для Red, Green та Blue каналів відповідно.

Таблиця 3.1 – Статистичні величини досліджуваних датасетів

	64 × 64 gray	224 × 224 gray	64 × 64 color	224 × 224 color
Mean	0.5006	0.5006	[0.5933, 0.4675, 0.4137]	[0.5943, 0.4672, 0.4130]
Std	0.2871	0.2904	[0.2954, 0.2836, 0.2751]	[0.3001, 0.2875, 0.2789]

Також було використано три різні архітектури нейронних мереж: звичайна відносно невелика кастомна CNN, ResNet-50 та MobileNet-v2. Кожна з цих трьох архітектур була навчена на кожному з чотирьох датасетів. Це дало нам 12 різних навчених моделей. Потім до кожної з трьох архітектур були додані CBAM блоки, і навчені на тих датасетах, на яких були показані найкращі результати звичайними немодифікованими архітектурами. Це дало нам ще 3 моделі. Тому загалом було навчено 15 моделей.

3.3.2 Custom мережа та її результати

Першою архітектурою є звичайна невелика CNN. Вона складається з чотирьох однотипних блоків. Кожен блок має наступний вигляд:

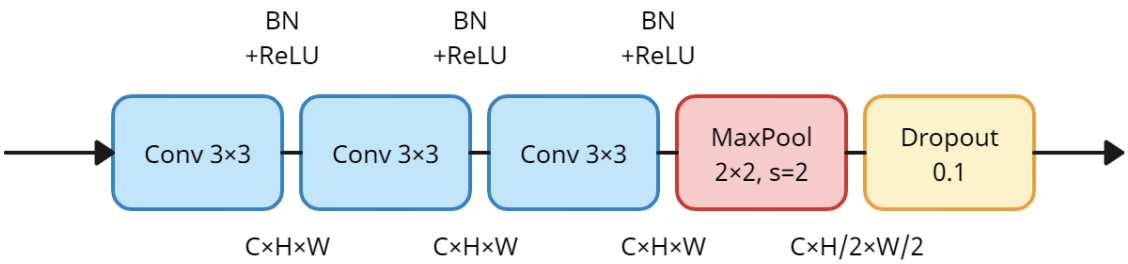

Рисунок 3.5 – Custom Block CNN

Загалом же ця нейронна мережа має наступний вигляд. В останньому Custom Block не використовуються пулінг і dropout. Перше для уникнення надмірного зменшення розміру карт ознак, а друге тому що буде застосовано dropout після всіх згорткових шарів.

Таблиця 3.2 – Будова Custom мережі для зображень 64×64

Layer	Output size
Input image	$1/3 \times 64 \times 64$
Custom Block ($C = 16,\ H = W = 64$)	$16 \times 32 \times 32$
Custom Block ($C = 32,\ H = W = 32$)	$32 \times 16 \times 16$
Custom Block ($C = 64,\ H = W = 16$)	$64 \times 8 \times 8$
Custom Block ($C = 128,\ H = W = 8$)	$128 \times 8 \times 8$
Dropout 0.3	$128 \times 8 \times 8$
Flatten	8192
Fully connected + Softmax	7

Для чорно-білого 64×64 зображення з одним каналом ця мережа матиме 548103 параметри. Для трьох-канального RGB зображення мережа матиме 548391 параметр. Загалом 12 згорткових шарів, 1 повнозв'язний та 3 пулінги.

Для зображення 224×224 мережа матиме трохи інший, але дуже схожий вигляд. Аналогічно в останньому Custom Block не використовуються пулінг і dropout. Перші два згорткові шари зменшують розміри карт ознак. Вони також використовують Batch Normalization та ReLU у вказаному порядку.

Таблиця 3.3 – Будова Custom мережі для зображень 224×224

Layer	Output size
Input image	$1/3 \times 224 \times 224$
Conv 16, 3×3, stride = 2	$16 \times 112 \times 112$
Conv 16, 3×3, stride = 2	$16 \times 56 \times 56$
Custom Block ($C = 16,\ H = W = 56$)	$16 \times 28 \times 28$
Custom Block ($C = 32,\ H = W = 28$)	$32 \times 14 \times 14$
Custom Block ($C = 64,\ H = W = 14$)	$64 \times 7 \times 7$
Custom Block ($C = 128,\ H = W = 7$)	$128 \times 7 \times 7$
Dropout 0.3	$128 \times 7 \times 7$
Flatten	6272
Fully connected + Softmax	7

Для чорно-білого 224×224 зображення з одним каналом ця мережа матиме 539367 параметри. Для трьох-канального RGB зображення мережа матиме 539655 параметр. Загалом 14 згорткових шарів, 1 повнозв'язний та 3 пулінги.

При навчанні усіх 5-ти моделей Custom CNN (4 без CBAM, 1 з CBAM) використовувалися наступні параметри:

- Функція втрат Cross Entropy;
- Оптимізатор Adam з параметрами за замовчуванням $\beta_1 = 0.9$ та $\beta_2 = 0.999$;
- Початкова швидкість навчання (Learning rate) становить 0.001. Виконується експоненційне затухання з параметром $\gamma = 0.939$. Це значення було обрано таким чином, щоб швидкість навчання зменшувалася вдвічі кожні 11 епох;
- Використовується L2 регуляризація (weight decay) з параметром $\lambda = 0.001$;

- Навчання триває протягом 40-ка епох. Зберігається та модель, шо показала найкращий результат на валідаційній вибірці протягом цих 40-ка епох;
- Розмір батча 32 елементи.

Для 4-х Custom CNN були отримані наступні результати:

	precision	recall	f1-score	support
angry	0.65	0.45	0.54	108
disgust	0.62	0.63	0.63	82
fear	0.57	0.46	0.51	95
happy	0.87	0.92	0.89	215
neutral	0.67	0.72	0.69	239
sad	0.65	0.62	0.63	126
surprised	0.58	0.68	0.63	135
accuracy			0.69	1000
macro avg	0.66	0.64	0.65	1000
weighted avg	0.68	0.69	0.68	1000

Рисунок 3.6 – Метрики на тестовому датасеті для Custom 64 gray

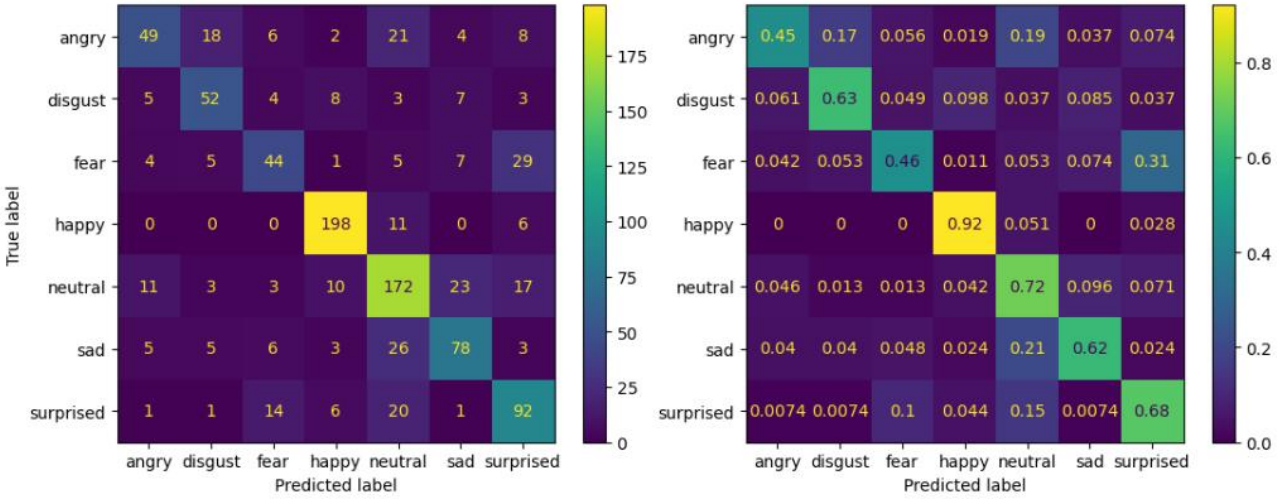

Рисунок 3.7 – Confusion matrix на тестовому датасеті для Custom 64 gray

	precision	recall	f1-score	support
angry	0.65	0.56	0.60	108
disgust	0.82	0.61	0.70	82
fear	0.58	0.51	0.54	95
happy	0.86	0.92	0.89	215
neutral	0.63	0.77	0.70	239
sad	0.65	0.65	0.65	126
surprised	0.67	0.58	0.62	135
accuracy			0.70	1000
macro avg	0.69	0.66	0.67	1000
weighted avg	0.70	0.70	0.70	1000

Рисунок 3.8 – Метрики на тестовому датасеті для Custom 64 color

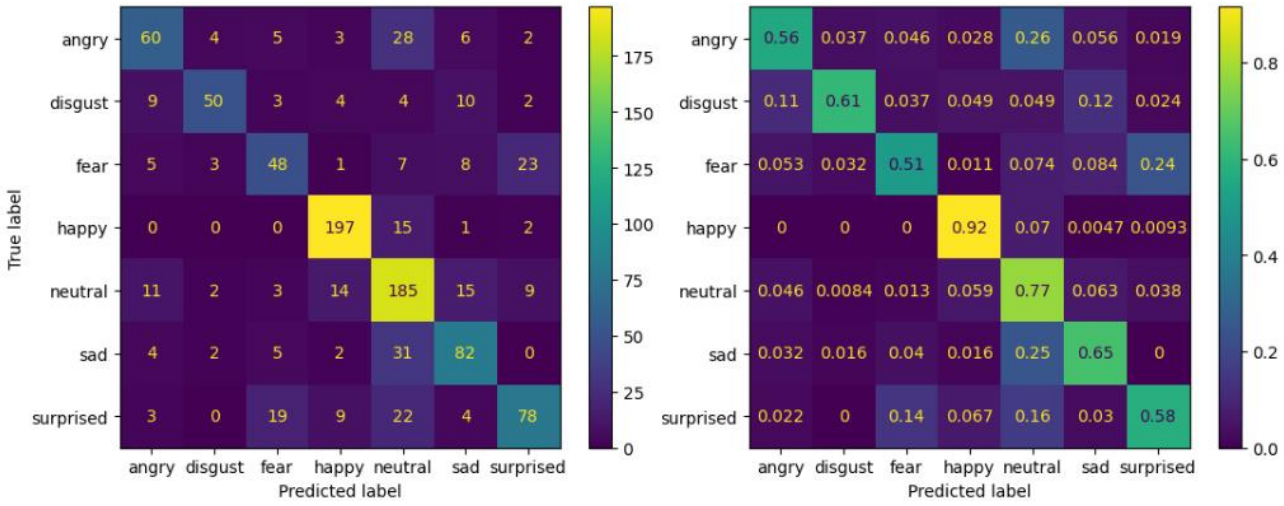

Рисунок 3.9 – Confusion matrix на тестовому датасеті для Custom 64 color

	precision	recall	f1-score	support
angry	0.69	0.56	0.62	108
disgust	0.68	0.63	0.66	82
fear	0.61	0.45	0.52	95
happy	0.90	0.88	0.89	215
neutral	0.66	0.77	0.71	239
sad	0.64	0.61	0.63	126
surprised	0.60	0.70	0.64	135
accuracy			0.70	1000
macro avg	0.68	0.66	0.67	1000
weighted avg	0.70	0.70	0.70	1000

Рисунок 3.10 – Метрики на тестовому датасеті для Custom 224 gray

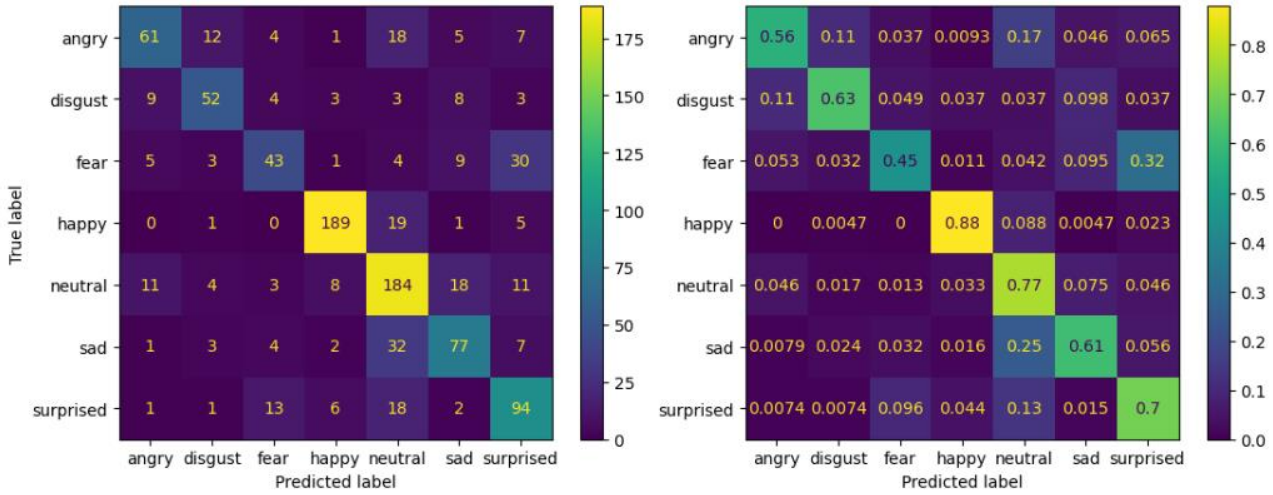

Рисунок 3.11 – Confusion matrix на тестовому датасеті для Custom 224 gray

	precision	recall	f1-score	support
angry	0.66	0.62	0.64	108
disgust	0.73	0.60	0.66	82
fear	0.59	0.51	0.54	95
happy	0.87	0.90	0.89	215
neutral	0.69	0.75	0.72	239
sad	0.61	0.68	0.64	126
surprised	0.68	0.64	0.66	135
accuracy			0.71	1000
macro avg	0.69	0.67	0.68	1000
weighted avg	0.71	0.71	0.71	1000

Рисунок 3.12 – Метрики на тестовому датасеті для Custom 224 color

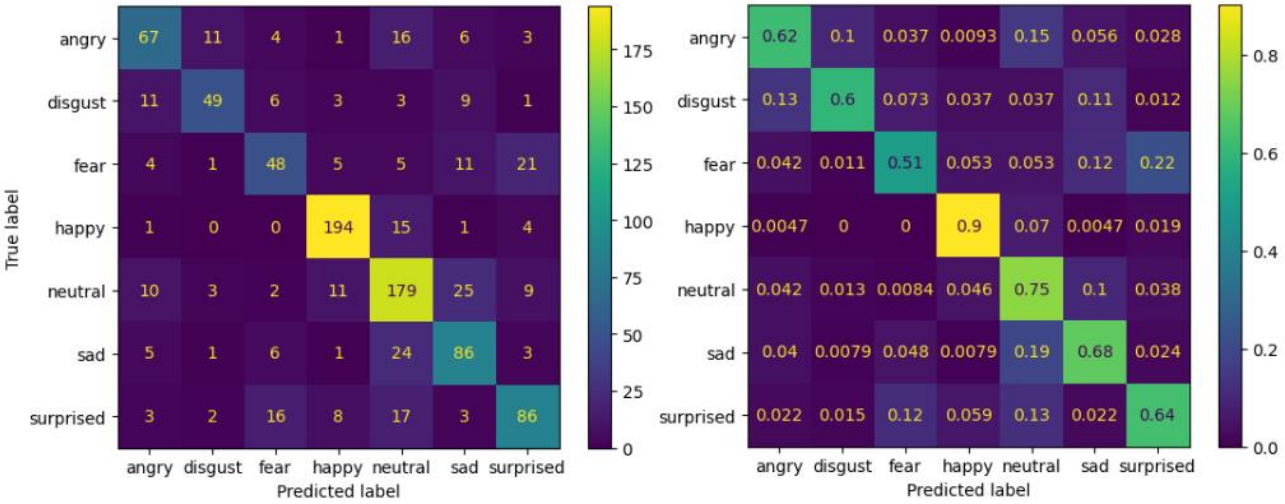

Рисунок 3.13 – Confusion matrix на тестовому датасеті для Custom 224 color

Таблиця 3.4 – Результати Custom мережі

Custom Model		64×64		224×224	
		Acc %	Loss	Acc %	Loss
gray	Val	72.4	0.805	72.9	0.802
	Test	**68.5**	**0.838**	**70.0**	**0.859**
color	Val	71.3	0.848	71.9	0.803
	Test	**70.0**	**0.820**	**70.9**	**0.814**

Найкращою виявилася Custom модель навчена на кольорових 224×224 зображеннях. Тепер додамо до неї 5 модулів CBAM. Вони будуть розміщені в кожному з чотирьох Custom Block-ів. В перших трьох він буде розміщений безпосередньо перед шаром MaxPool. В останньому блоці буде розміщений після функції активації третього згорткового шару перед дропаутом 0.3. П'ятий блок буде розміщено після двох перших згорткових шарів, що зменшують розміри зображення. Оскільки кількість карт ознак в усіх шарах є невеликою, ми встановимо значення $\text{ratio} = 4$ в Channel Attention, на відмінну від значення 16, що використовували автори в [11]. Загалом ця мережа матиме 551153 параметри. Отримаємо наступне:

	precision	recall	f1-score	support
angry	0.74	0.71	0.73	108
disgust	0.69	0.54	0.60	82
fear	0.59	0.60	0.59	95
happy	0.88	0.91	0.89	215
neutral	0.69	0.78	0.73	239
sad	0.56	0.52	0.54	126
surprised	0.64	0.60	0.62	135
accuracy			0.71	1000
macro avg	0.68	0.67	0.67	1000
weighted avg	0.70	0.71	0.70	1000

Рисунок 3.14 – Метрики на тестовому датасеті для Custom + CBAM 224 color

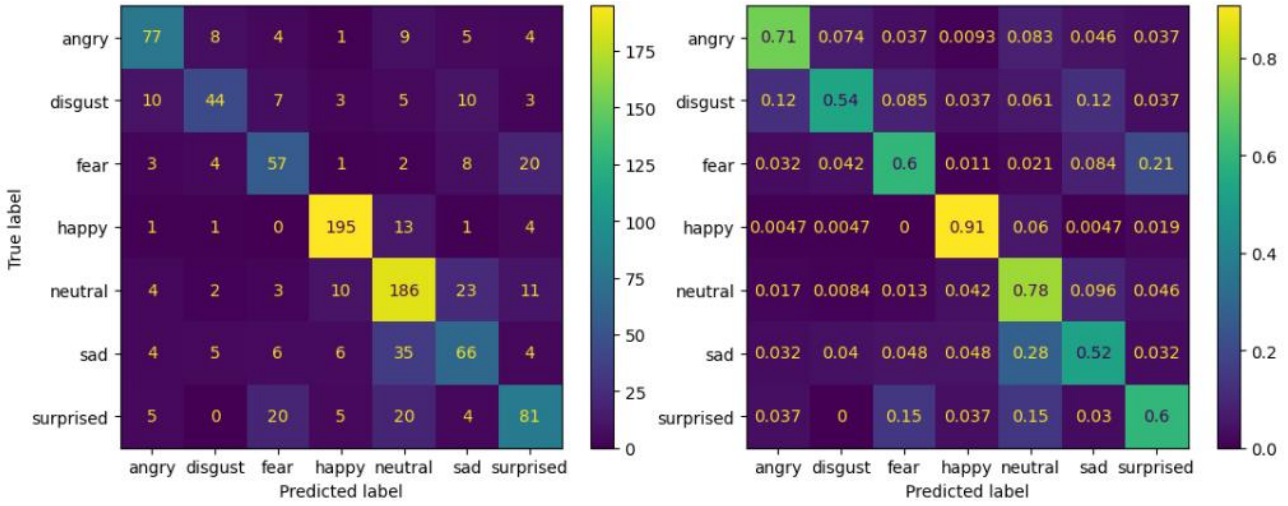

Рисунок 3.15 – Confusion matrix на тестовому датасеті для Custom+CBAM

Таблиця 3.5 – Результати Custom + CBAM моделі

		Acc %	Loss
Custom + CBAM 224 color	Validation	71.3	0.806
	Test	**71.2**	**0.807**

Як бачимо, додавання CBAM блоку покращило точність моделі.

3.3.3 MobileNet-v2 та її результати

Далі були проведені 5 аналогічних експериментів з MobileNet. Було обрано MobileNet-v2. Ваги було ініціалізовано пренавченими вагами, що були навчені на ImageNet з використанням просунутої навчальної стратегій. Кількість параметрів в чотирьох «базових» моделях однакова і становить 2232839.

Вхідними зображеннями пренавченої MobileNet можуть бути лише трьох-канальні RGB зображення 224×224, оскільки саме на таких зображеннях з ImageNet ця мережа була навчена. Тому чорно-білі датасети і датасети з 64×64

зображеннями потребують додаткової обробки, щоб зробити їх придатними для входу у мережу. Чорно-білі зображення були перетворені в RGB шляхом дублювання їх єдиного каналу. В результаті чорно-білі одноканальні зображення стали трьохканальними RGB при цьому залишаючись сірими з абсолютно ідентичним кольоровими каналами. Роздільна здатність зображень 64×64 була збільшена до необхідних 224×224 пікселя (upscaling) використовуючи бікубічну інтерполяцію.

При навчанні використовувалися аналогічні параметри як і в Custom моделі.

Були отримані наступні результати:

	precision	recall	f1-score	support
angry	0.68	0.66	0.67	108
disgust	0.66	0.70	0.68	82
fear	0.63	0.42	0.51	95
happy	0.84	0.92	0.88	215
neutral	0.68	0.74	0.71	239
sad	0.70	0.65	0.67	126
surprised	0.63	0.61	0.62	135
accuracy			0.71	1000
macro avg	0.69	0.67	0.68	1000
weighted avg	0.70	0.71	0.70	1000

Рисунок 3.16 – Метрики на тестовому датасеті для MobileNet 64 gray

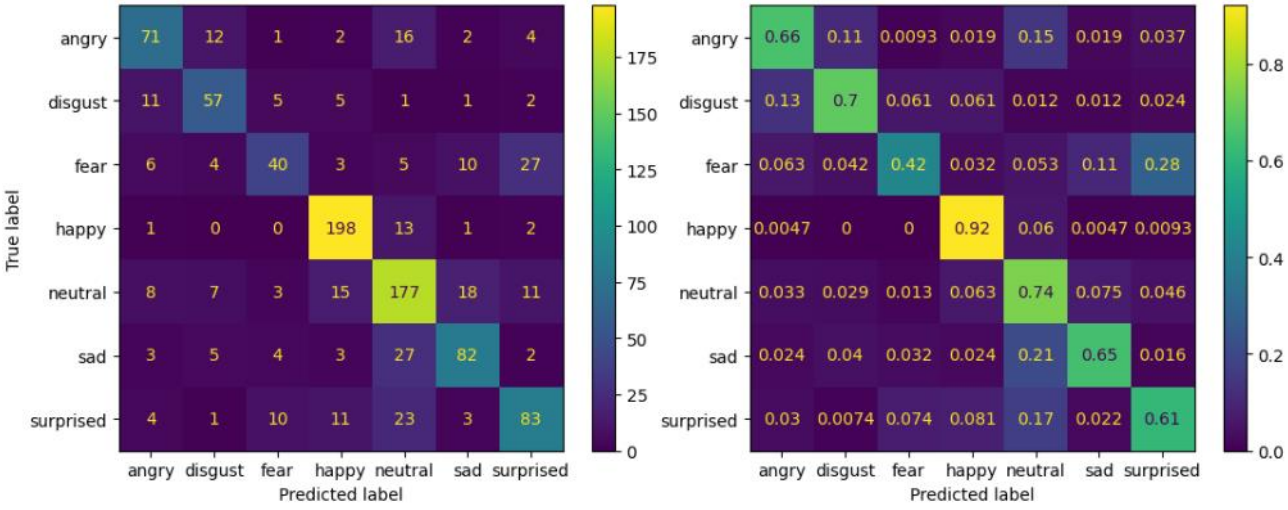

Рисунок 3.17 – Confusion matrix на тестовому датасеті для MobileNet 64 gray

	precision	recall	f1-score	support
angry	0.60	0.63	0.61	108
disgust	0.70	0.49	0.58	82
fear	0.67	0.40	0.50	95
happy	0.87	0.90	0.89	215
neutral	0.65	0.84	0.73	239
sad	0.71	0.60	0.65	126
surprised	0.70	0.69	0.69	135
accuracy			0.71	1000
macro avg	0.70	0.65	0.66	1000
weighted avg	0.71	0.71	0.70	1000

Рисунок 3.18 – Метрики на тестовому датасеті для MobileNet 64 color

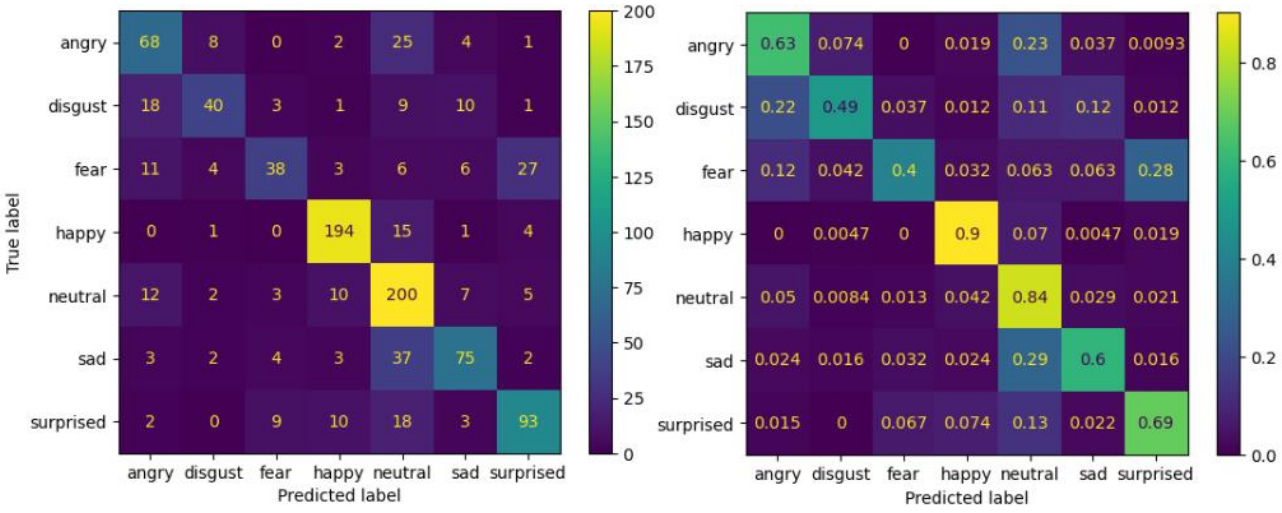

Рисунок 3.19 – Confusion matrix на тестовому датасеті для MobileNet 64 color

	precision	recall	f1-score	support
angry	0.67	0.57	0.62	108
disgust	0.66	0.70	0.68	82
fear	0.64	0.40	0.49	95
happy	0.87	0.90	0.89	215
neutral	0.69	0.74	0.72	239
sad	0.75	0.67	0.71	126
surprised	0.58	0.73	0.64	135
accuracy			0.71	1000
macro avg	0.69	0.67	0.68	1000
weighted avg	0.71	0.71	0.71	1000

Рисунок 3.20 – Метрики на тестовому датасеті для MobileNet 224 gray

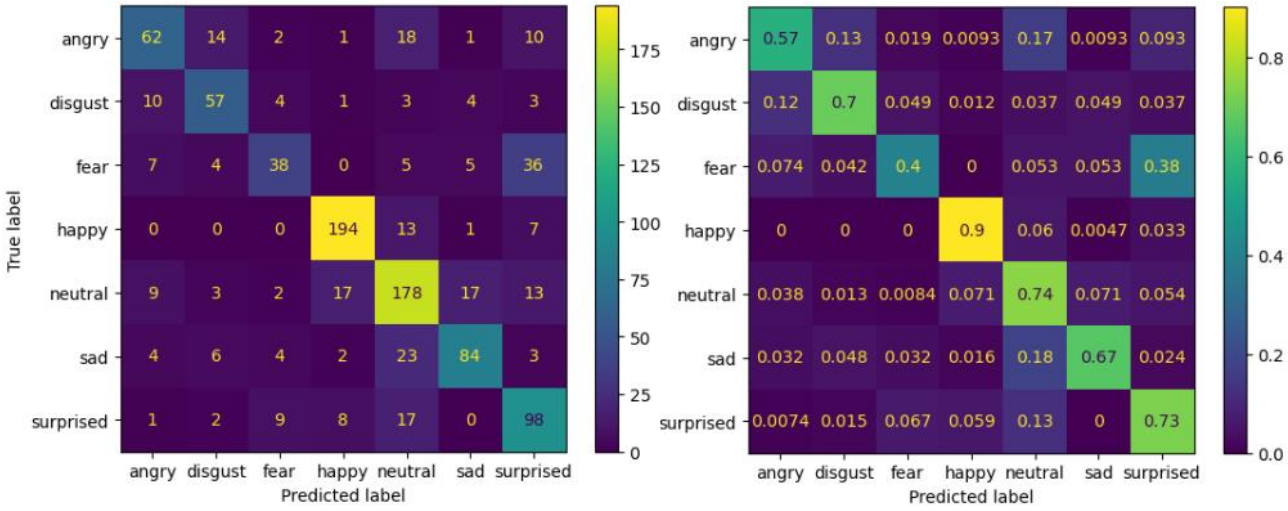

Рисунок 3.21 – Confusion matrix на тестовому датасеті для MobileNet 224 gray

	precision	recall	f1-score	support
angry	0.70	0.55	0.61	108
disgust	0.79	0.67	0.72	82
fear	0.60	0.39	0.47	95
happy	0.89	0.89	0.89	215
neutral	0.66	0.82	0.74	239
sad	0.70	0.67	0.68	126
surprised	0.60	0.67	0.63	135
accuracy			0.71	1000
macro avg	0.70	0.67	0.68	1000
weighted avg	0.72	0.71	0.71	1000

Рисунок 3.22 – Метрики на тестовому датасеті для MobileNet 224 color

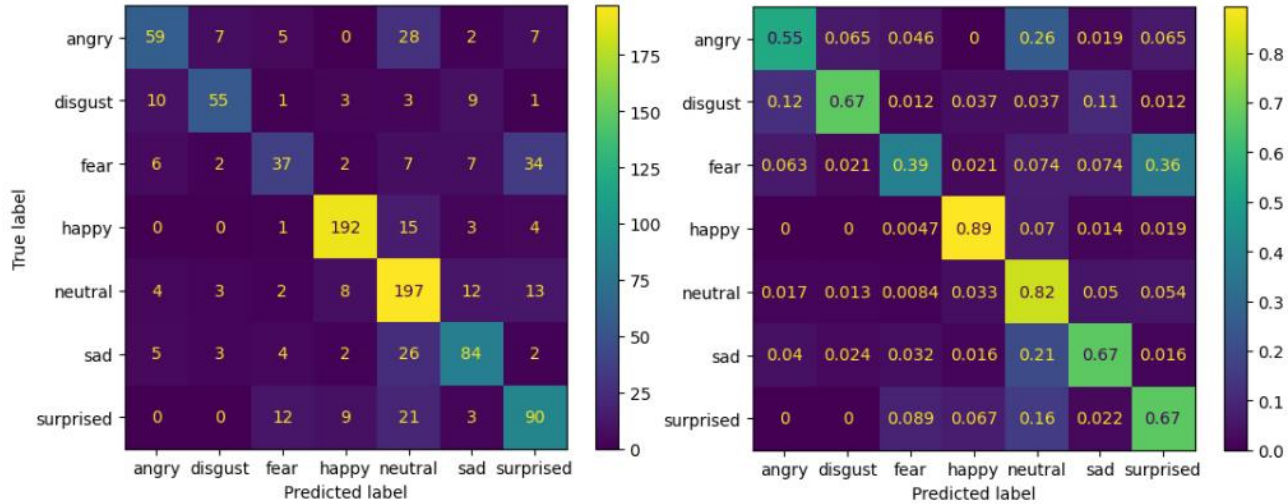

Рисунок 3.23 – Confusion matrix на тестовому датасеті для MobileNet 224 color

Таблиця 3.6 – Результати MobileNet-v2

MobileNet-v2		64 × 64		224 × 224	
		Acc %	Loss	Acc %	Loss
gray	Val	71.3	0.804	72.8	0.800
	Test	**70.8**	**0.819**	**71.1**	**0.809**
color	Val	73.0	0.780	74.1	0.776
	Test	**70.8**	**0.818**	**71.4**	**0.827**

Найкращий результат знову мала модель навчена на датасеті 224 color. Саме на ньому буде навчена модифікована модель MobileNet-v2 + CBAM.

Тепер вставимо CBAM в мережу MobileNet-v2. CBAM було вставлено в кожен inverted residual block. А саме, CBAM було застосовано до всіх карт ознак, що утворюються після depth-wise згортки. Це зображено нижче.

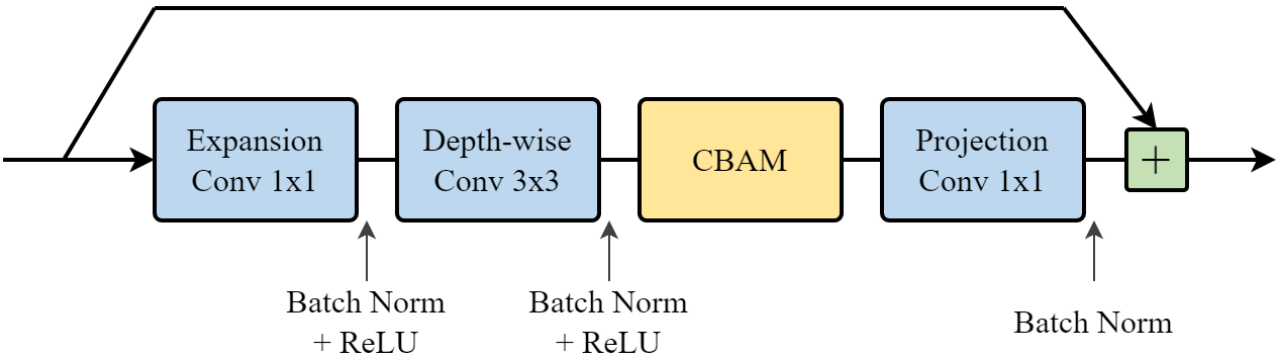

Рисунок 3.24 – Inverted residual block + CBAM

Причиною вставки цього модуля саме в це місце є те, що це дозволяє акцентувати увагу на необхідному саме в збагачених ознаками тензорах. Як було сказано в пункті 2.3.2 присвяченому MobileNet, стискаюча projection згортка не погіршує якість добутих ознак, а отже не погіршить і результат CBAM на збагачених картах ознак. Окрім того, саме в цьому місці виконується дія “Squeeze-and-Excite”, що була додана вже в MobileNet-v3 і що має таку ж мету, як

і Channel Attention в CBAM, що лише зайвий раз доводить доцільність обрання саме цього місця.

Всього вставлено 17 CBAM модулів в кожен з 17-ти Inverted residual блоків. Перші блоки мають невелику кількість карт ознак, тому використовувати оригінальне значення $\text{ratio} = 16$ [11] в Channel Attention було б надлишковим. Тому в першому Inverted residual block CBAM має $\text{ratio} = 4$, другий, третій та четвертий мають CBAM зі значенням $\text{ratio} = 8$, і усі інші 13 блоків мають CBAM з оригінальним значенням $\text{ratio} = 16$, що використовувався авторами статті CBAM.

Ця модель MobileNet-v2 + CBAM, як було сказано вище, була навчена на кольорових зображеннях 224×224. Вона має 2805257 параметри. Отримані наступні результати:

	precision	recall	f1-score	support
angry	0.70	0.60	0.65	108
disgust	0.73	0.63	0.68	82
fear	0.65	0.56	0.60	95
happy	0.91	0.87	0.89	215
neutral	0.64	0.77	0.70	239
sad	0.63	0.70	0.66	126
surprised	0.70	0.64	0.67	135
accuracy			0.71	1000
macro avg	0.71	0.68	0.69	1000
weighted avg	0.72	0.71	0.71	1000

Рисунок 3.25 – Метрики на тестовому датасеті для MobileNet+CBAM

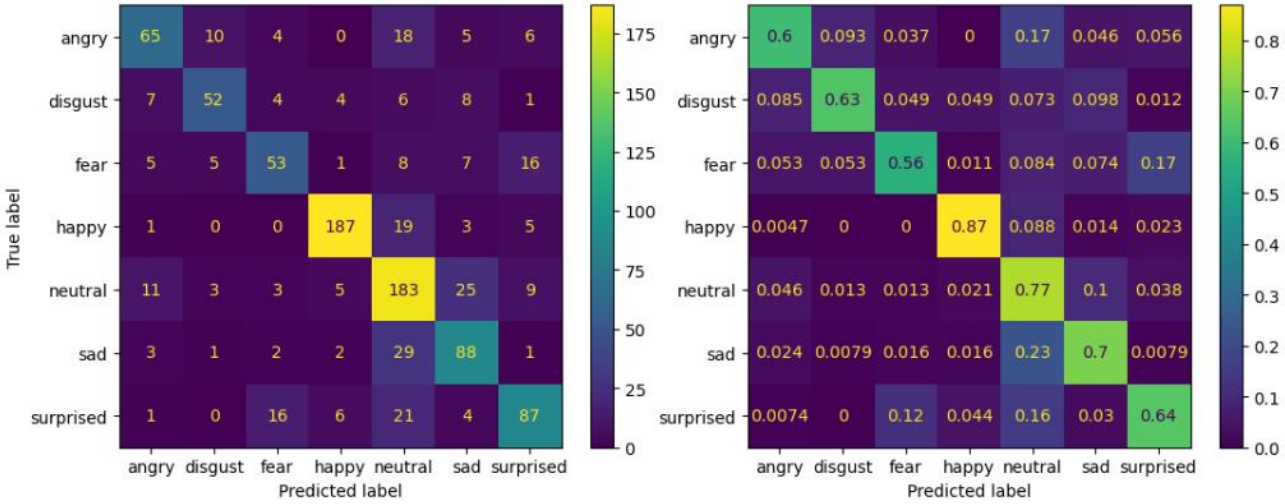

Рисунок 3.26 – Confusion matrix на тестовому датасеті для MobileNet+CBAM

Таблиця 3.7 – Результати MobileNet-v2 + CBAM

		Acc %	Loss
MobileNet-v2 + CBAM 224 color	Validation	74.3	0.769
	Test	**71.5**	**0.801**

Результати на тестовій множині виявились кращими ніж у звичайних MobileNet-v2. Це свідчить про те, що додавання CBAM покращує якість роботи цієї архітектури.

3.3.4 ResNet-50 та її результати

Далі були проведені аналогічні вже останні 5 експериментів для архітектури ResNet-50. Усе аналогічно як і у випадку MobileNet-v2: ваги ініціалізовані пренавченими вагами, що були навчені на ImageNet з використанням більш просунутих навчальних стратегій. Чорно-білі зображення та зображення 64×64

були приведені до необхідного «вхідного формату» ResNet шляхом дублювання каналів і ресамплінгу з застосуванням бікубічної інтерполяції.

Для додаткового необхідного регуляризаційного ефекту після глобального пулінгу був доданий шар дропауту з ймовірністю 0.25. Для чотирьох експериментів ця модель ResNet-50 має 23522375 параметрів. Параметри навчання такі самі, як і в попередніх моделях. Були отримані наступні результати:

	precision	recall	f1-score	support
angry	0.72	0.53	0.61	108
disgust	0.57	0.60	0.58	82
fear	0.59	0.49	0.54	95
happy	0.87	0.89	0.88	215
neutral	0.61	0.76	0.68	239
sad	0.63	0.59	0.61	126
surprised	0.67	0.59	0.63	135
accuracy			0.68	1000
macro avg	0.67	0.64	0.65	1000
weighted avg	0.68	0.68	0.68	1000

Рисунок 3.27 – Метрики на тестовому датасеті для ResNet 64 gray

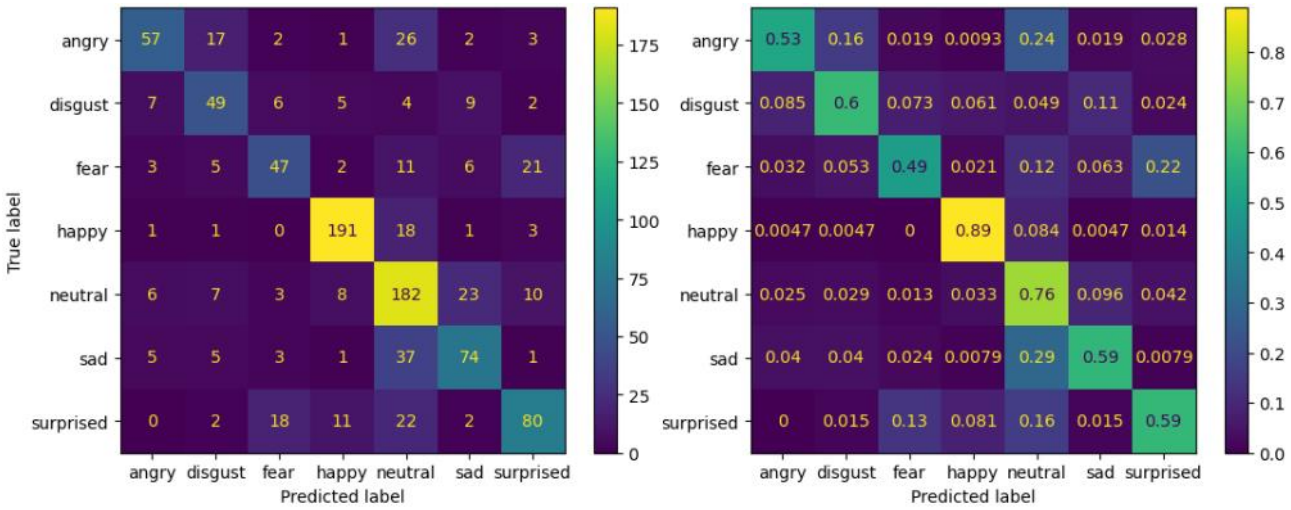

Рисунок 3.28 – Confusion matrix на тестовому датасеті для ResNet 64 gray

	precision	recall	f1-score	support
angry	0.58	0.56	0.57	108
disgust	0.66	0.59	0.62	82
fear	0.63	0.39	0.48	95
happy	0.86	0.89	0.87	215
neutral	0.64	0.78	0.70	239
sad	0.61	0.57	0.59	126
surprised	0.69	0.67	0.68	135
accuracy			0.69	1000
macro avg	0.67	0.64	0.65	1000
weighted avg	0.68	0.69	0.68	1000

Рисунок 3.29 – Метрики на тестовому датасеті для ResNet 64 color

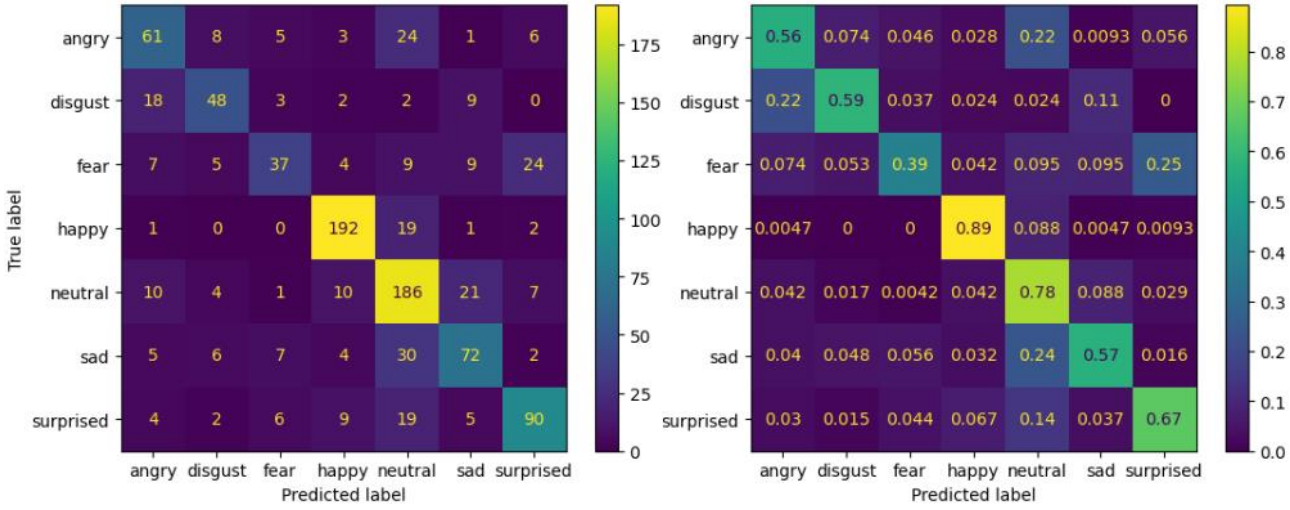

Рисунок 3.30 – Confusion matrix на тестовому датасеті для ResNet 64 color

	precision	recall	f1-score	support
angry	0.66	0.55	0.60	108
disgust	0.64	0.61	0.62	82
fear	0.58	0.37	0.45	95
happy	0.87	0.90	0.89	215
neutral	0.63	0.81	0.71	239
sad	0.71	0.54	0.61	126
surprised	0.58	0.64	0.61	135
accuracy			0.69	1000
macro avg	0.67	0.63	0.64	1000
weighted avg	0.69	0.69	0.68	1000

Рисунок 3.31 – Метрики на тестовому датасеті для ResNet 224 gray

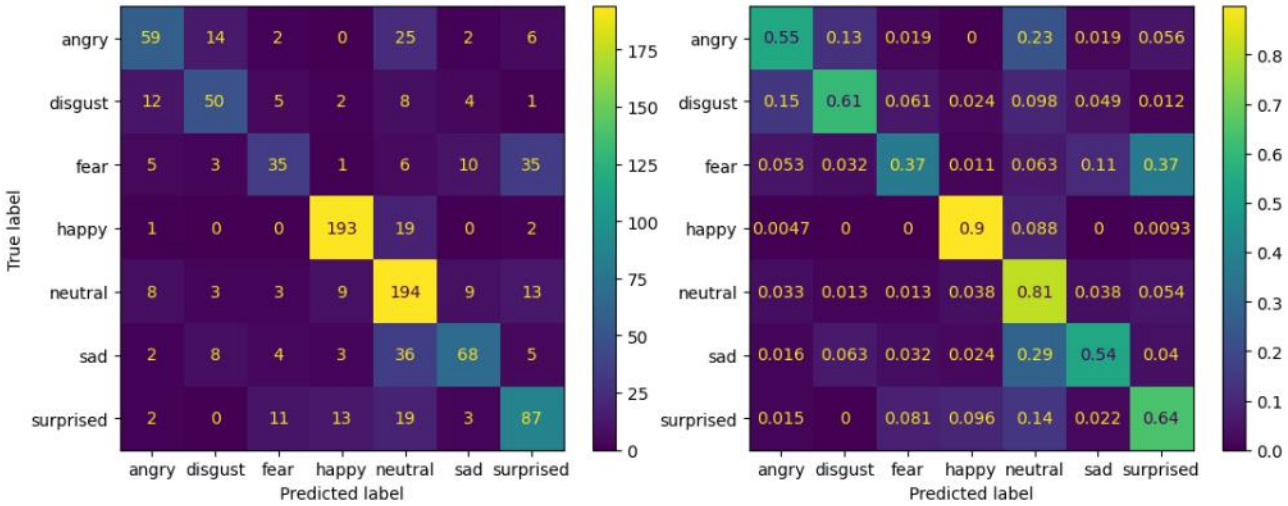

Рисунок 3.32 – Confusion matrix на тестовому датасеті для ResNet 224 gray

	precision	recall	f1-score	support
angry	0.60	0.65	0.62	108
disgust	0.62	0.66	0.64	82
fear	0.58	0.53	0.55	95
happy	0.86	0.89	0.88	215
neutral	0.69	0.72	0.70	239
sad	0.70	0.66	0.68	126
surprised	0.65	0.57	0.61	135
accuracy			0.70	1000
macro avg	0.67	0.67	0.67	1000
weighted avg	0.70	0.70	0.70	1000

Рисунок 3.33 – Метрики на тестовому датасеті для ResNet 224 color

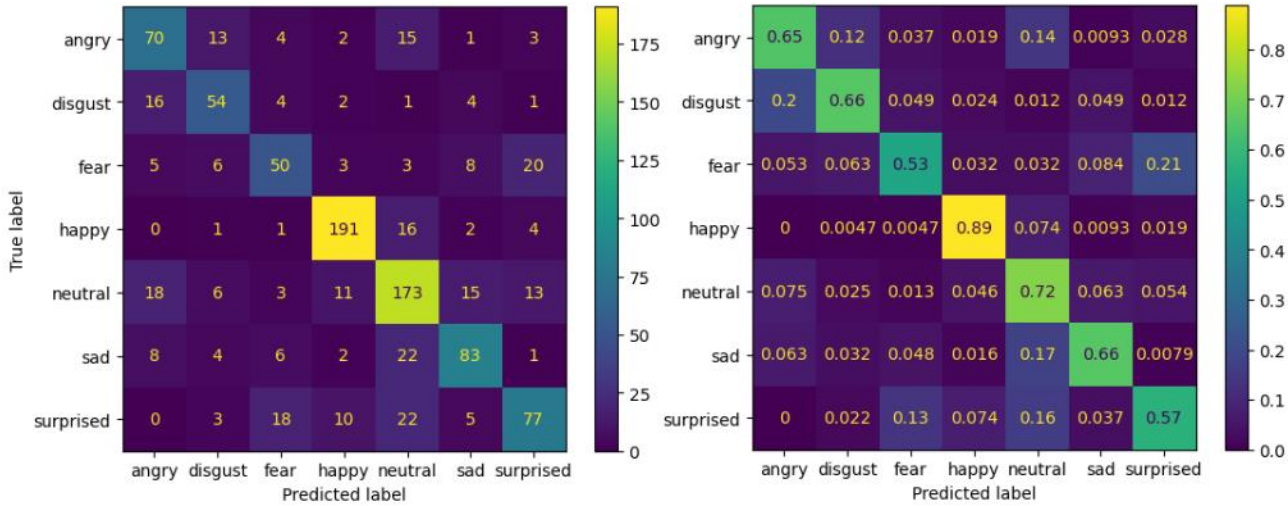

Рисунок 3.34 – Confusion matrix на тестовому датасеті для ResNet 224 color

Таблиця 3.8 – Результати ResNet-50

ResNet-50		64 × 64		224 × 224	
		Acc %	Loss	Acc %	Loss
gray	Val	70.4	0.842	71.2	0.811
	Test	**68.0**	**0.937**	**68.6**	**0.889**
color	Val	70.6	0.834	72.3	0.796
	Test	**68.6**	**0.863**	**69.8**	**0.854**

Найкращий результат знову мала модель навчена на датасеті 224 color. Саме на ньому буде навчена модифікована модель ResNet-50 + CBAM.

Тепер вставимо CBAM в архітектуру ResNet-50. Він буде вставлений в кожен Bottleneck блок таким чином, як і запропоновано в статті [11]. Це зображено на рисунку нижче.

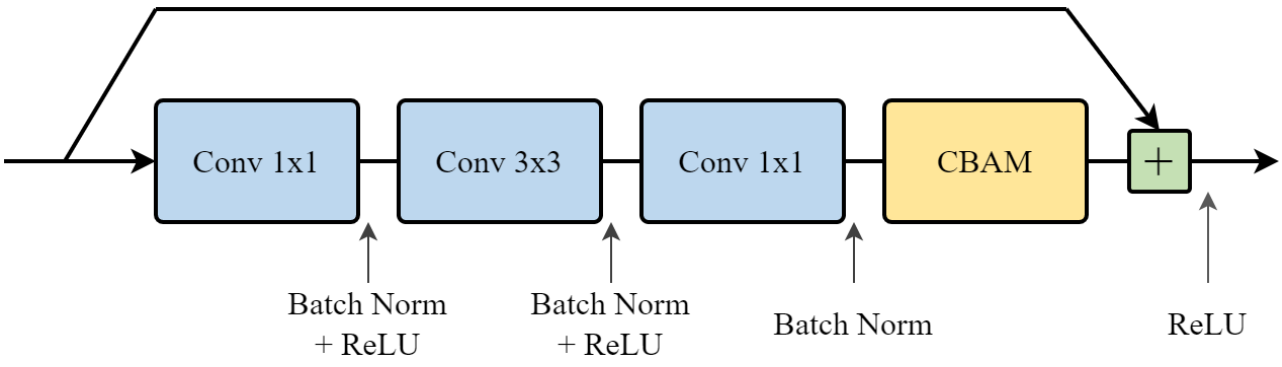

Рисунок 3.35 – Bottleneck Residual Block + CBAM

Всього вставлено 16 CBAM модулів в кожен з 16-ти залишкових блоків. В усіх модулях Channel Attention має оригінальне значення $\text{ratio} = 16$. Аналогічно, як і попередніх 4-х ResNet моделях, був доданий дропаут з ймовірністю 0.25. Загалом ResNet-50 з вбудованим CBAM матиме 26038887 параметри. Результати:

	precision	recall	f1-score	support
angry	0.53	0.62	0.57	108
disgust	0.64	0.66	0.65	82
fear	0.59	0.58	0.58	95
happy	0.89	0.86	0.87	215
neutral	0.68	0.72	0.70	239
sad	0.69	0.60	0.64	126
surprised	0.67	0.64	0.65	135
accuracy			0.69	1000
macro avg	0.67	0.67	0.67	1000
weighted avg	0.70	0.69	0.69	1000

Рисунок 3.36 – Метрики на тестовому датасеті для ResNet+CBAM

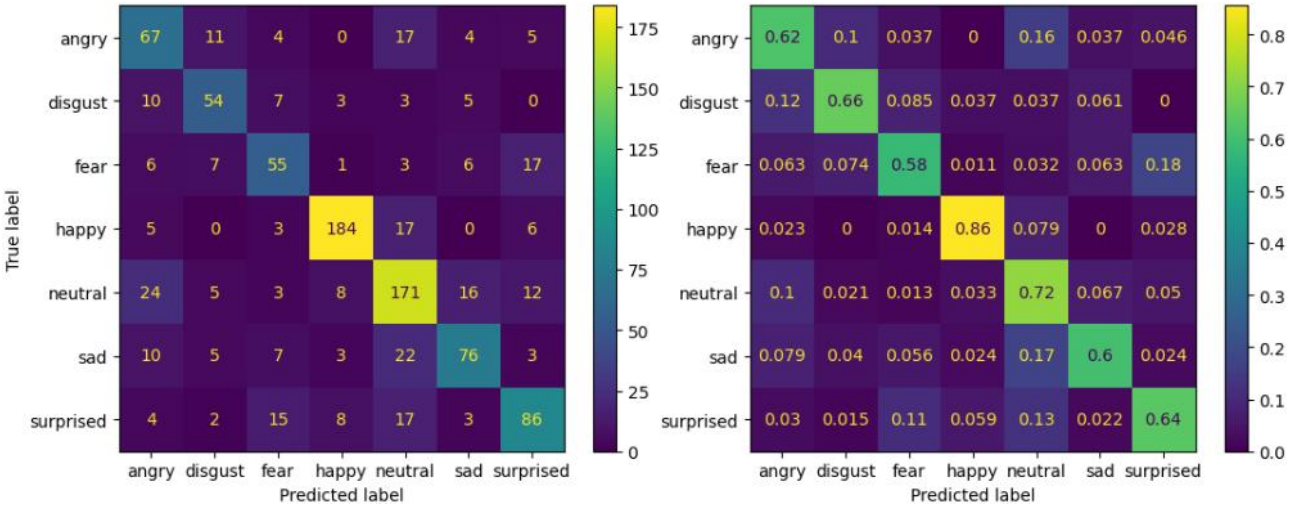

Рисунок 3.37 – Confusion matrix на тестовому датасеті для ResNet+CBAM

Таблиця 3.9 – Результати ResNet-50 + CBAM

		Acc %	Loss
MobileNet-v2 + CBAM	Validation	71.7	0.837
224 color	Test	**69.3**	**0.881**

CBAM не покращив роботу мережі ResNet на нашому датасеті, звичайна ResNet-50 показала кращі результати на датасеті 224 color.

3.4 Порівняння та аналіз навчених моделей

Accuracy у відсотках усіх 15-ти моделей на тестовій множині було зведено в єдину таблицю.

Таблиця 3.10 – Точність моделей на тестових підвибірках

	64 gray	64 color	224 gray	224 color	224 color + CBAM
Custom	68.5	70.0	70.0	70.9	71.2
MobileNet-v2	70.8	70.8	71.1	71.4	71.5
ResNet-50	68.0	68.6	68.6	69.8	69.3

Проаналізуємо отримані результати.

Спершу зробимо висновки щодо то, чи впливає роздільна здатність і наявність кольору на якість класифікації зображень. В усіх трьох архітектурах збільшення роздільної здатності з 64×64 до 224×224 сприяє збільшенню точності роботи нейронних мереж. Аналогічно наявність кольору також призводить до покращення прогнозів. Тому, базуючись на цих емпіричних доказах, робимо висновок, що в задачі розпізнавання емоцій на обличчі висока роздільна здатність зображень та наявність кольору сприятиме збільшенню точності моделі та покращенню класифікації. Також можна зробити висновок, що це збільшення не є дуже великим: при значному збільшені роздільної здатності майже в 4 рази точність зросла в середньому лише на 0.85%, а при додаванні кольору – в середньому на 0.75%. Тому вимоги щодо кольоровості і високої здатності зображень не є критичними. Моделі, що навчені на чорно-білих малих зображеннях, все ще здатні виділяти цікаві ознаки і якісно виконувати задачу, хоч і трохи гірше за аналогічні моделі, що навчені на кольорових зображеннях вищої роздільної здатності. Тепер дамо відповідь на питання чому ці характеристики збільшують точність. З роздільною здатністю інтуїтивно зрозуміло – зображення

високої якості мають значно більше деталей, і самі ці деталі є більш чіткими і краще відрізняються одна від одної. Завдяки цьому нейронні мережі зможуть навчитися якісно виділяти більше цікавих інформативних ознак, які б інакше було б неможливо побачити чи розрізнити серед інших. Щодо кольору відповідь не є такою очевидною. Колір не несе жодної емоційної інформації: емоція на обличчі визначається виключно виразом на ньому, і колір не здатен надати жодної підказки, що допомогла б безпосередньо визначити емоцію. Тому логічно було б припустити, що як на кольорових, так і на чорно-білих зображеннях точність моделей буде приблизно однаковою. Але ми бачимо, що це не так, і що на кольорових датасетах були показані кращі результати. Це можна пояснити тим, що хоч колір і не несе емоційної інформації, він допомагає краще і точніше виявити й розрізнити загальні деталі на фото, що можливо і мають схожі яскравості, але мають різні кольори (наприклад очі, губи і волосся можуть мати подібний колір на чорно-білих фото, але на кольорових матимуть різне забарвлення, що дозволить їх краще розрізнити). Усі три моделі показали найкращі результати на датасеті 224 color.

Тепер зробимо висновки щодо того, як впливає архітектура мережі на точність. Перехід від простої архітектури, що має невелика мережа Custom, до складнішої архітектури MobileNet-v2 помітно збільшив точність. Це і очікувалося, бо MobileNet-v2 є значно глибшою і обробляє більше карт ознак, а тому здатна вивчити більшу кількість якісніших ознак, що сприятиме збільшенню точності класифікації. Але перехід від простої Custom архітектури до ResNet-50, що є ще більшою ніж MobileNet-v2 не просто не покращив результати, а навіть погіршив їх. Причина цього, на мою думку, криється в розмірі датасету і кількості параметрів у моделі. Наш ResNet-50 має 23.5 млн параметри, що вимагає дуже багато навчальних даних, для того щоб правильно навчити усі ці ваги і отримати модель з великою узагальнюючою здатністю. Проте наш навчальний датасет містить всього 19981 зображення, що є недостатнім, щоб навчити ResNet-50. Різноманітність датасету є недостатньою і

через це така велика модель стає дуже схильною до перенавчання і не здатна гарно узагальнювати навіть при використанні дуже жорсткої регуляризації. На ImageNet, що містить понад 1.2 млн зображень, ResNet-50 показує значно кращі результати ніж MobileNet-v2. Отже, найкращий результат показала архітектура MobileNet-v2, що має компроміс між потенційною узагальнюючою здатністю, своїм розміром і кількістю зображень у навчальній вибірці, чого не має інша досліджувана архітектура ResNet-50.

І, нарешті, зробимо висновки щодо впливу додавання модуля уваги CBAM до моделей. Для MobileNet-v2 і особливо Custom додавання цього attention механізму покращило результати. Для MobileNet-v2 ця зміна принесла дуже мале покращення мабуть тому, що немодифікована MobileNet-v2 вже знаходиться близько до тої межі, коли подальше ускладнення моделі не покращить результат через дисбаланс складності моделі і різноманітності навчальної вибірки. Це наочно видно на прикладі ResNet-50, де додавання CBAM навіть погіршило результат. Отже, для задачі розпізнавання емоцій, за умови що модель не є переускладненою і завеликою, додавання CBAM покращить точність класифікації. Інакше покращення не буде, а можливо буде навіть погіршення точності.

Найгіршу точність має модель ResNet-50 навчена на датасеті 64 gray. Найкращий результат має модель MobileNet-v2 + CBAM навчена на датасеті 224 color.

3.5 Опис кінцевого програмного продукту

В кінцевий продукт з GUI буде імплементована модель, що показала найкращий результат, а саме MobileNet-v2 + CBAM навчена на RGB зображеннях з роздільною здатністю 224×244 пікселі. GUI було реалізовано за допомогою бібліотеки tkinter.

При запуску додатку відкриється вікно і у користувача буде можливість обрати зображення обличчя, що має знаходитися локально на його комп’ютері. Фотографія може бути будь-якої форми та роздільної здатності. Але є дві вимоги щодо неї: на ній обов’язково має бути людське обличчя, і воно має бути єдиним. Для обрання фото користувач має натиснути кнопку “Select image”.

Після цього обране фото буде зображено у вікні. Після цього користувач має натиснути на кнопку “Recognize emotion”. Якщо натиснути на цю кнопку до того, як обрати зображення, то буде виведена фраза “No image :(”. Якщо ж зображення було обране, то результатом роботи програми буде три речі:

1. Розпізнана емоція на обличчі;
2. Оброблене вхідне фото як було описано в розділі 3.2;
3. Теплова карта Grad-CAM для розпізнаної в першому результаті емоції, що була накладена зверху на preprocessed фото. Це накладення було реалізовано як $0.7 \cdot \text{Preprocessed image} + 0.3 \cdot \text{Heatmap}$.

Після розпізнавання одного фото можна обрати наступне і так далі. Для виходу необхідно закрити програму як звичайне вікно натиснувши на хрестик зверху праворуч.

Нижче наведені результати роботи програмного продукту для декількох необроблених фотографій з тестової множини.

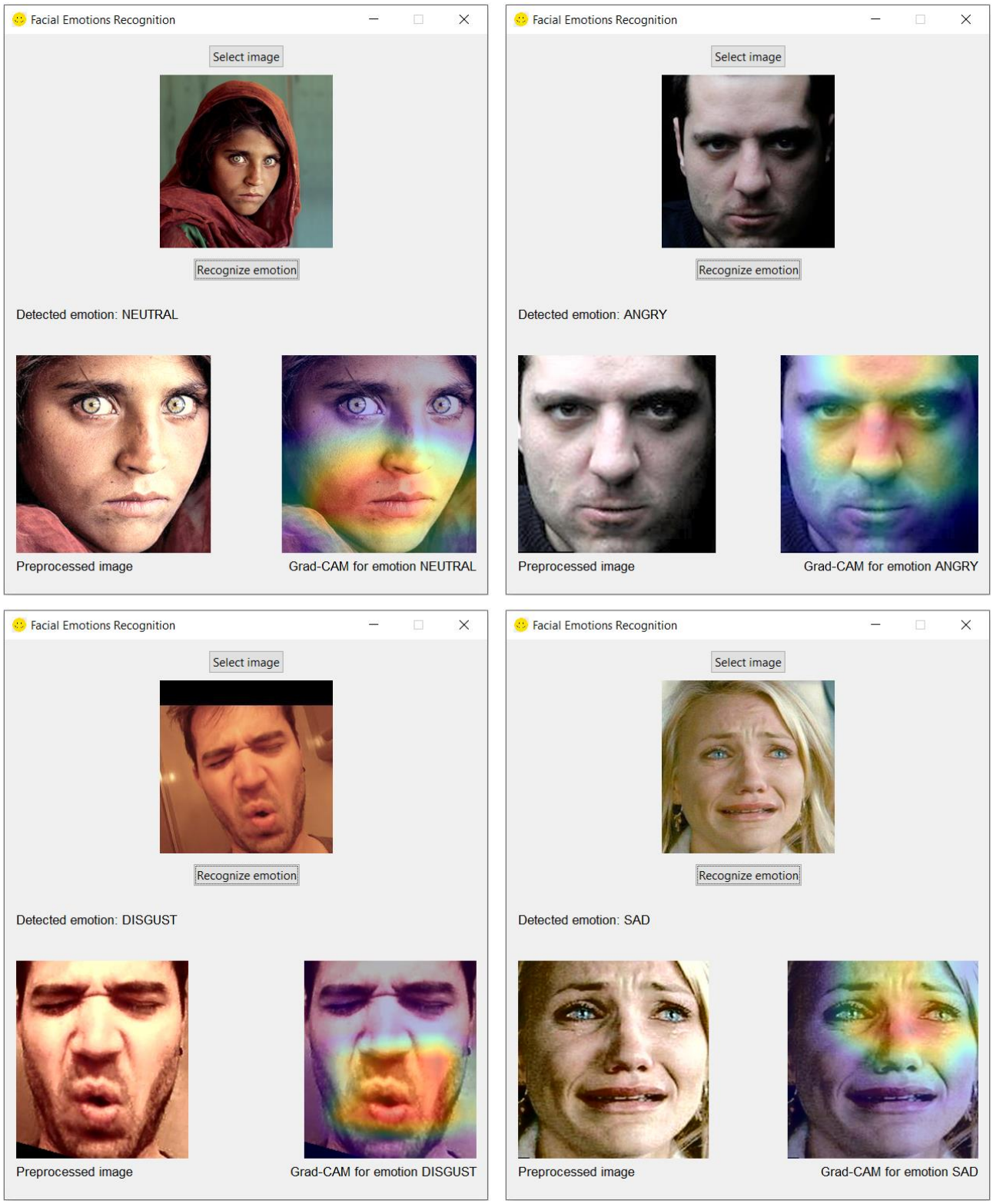

Рисунок 3.38 – Результати роботи додатку

3.6 Висновки до розділу 3

У цьому розділі було детально розглянуто саме практичну частину даної роботи розпізнавання емоцій на людському обличчі.

Було описано процес попередньої обробки зображень, що складається з повороту і обрізання зображення, застосування вирівнювання гістограми, приведення до спільної роздільної здатності і нормалізації.

Було навчено 15 різних моделей 3-х різних архітектур: невелика Custom згорткова мережа, MobileNet-v2 та ResNet-50. Було визначено, що висока роздільна здатність фото і наявність кольору незначно, але все ж покращують точність класифікації. Найкращі результати були показані саме на кольорових зображеннях великої роздільної здатності. Серед архітектур найкращий результат показала MobileNet-v2, бо вона має баланс між своєю складністю і розміром навчального датасету. Додавання CBAM збільшує точність за умови, що нейронна мережа ще не є заскладною для розміру датасету і його різноманітності.

Найкращий результат показала модель MobileNet-v2 + CBAM на кольоровому датасеті зображень 224×224 пікселі досягнувши accuracy у 71.5% на тестовій вибірці.

Цю модель було вбудовано в додаток з інтуїтивно зрозумілим інтерфейсом. Цей програмний продукт був детально описаний, він дозволяє користувачеві обрати зображення обличчя на своєму пристрої і побачити як результат класифіковану емоцію, оброблене обличчя та теплову карту Grad-CAM, що виділяє на ньому ті ділянки, які мали найбільший внесок у класифікацію.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту для розпізнавання емоцій на людському обличчі.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи розпізнавання емоцій на людському . Оскільки рішення

стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

1. функціонування на персональних комп'ютерах із стандартним набором компонентів;
2. зручність та зрозумілість для користувача;
3. швидкість обробки даних та доступ до інформації в реальному часі;
4. можливість зручного масштабування та обслуговування;
5. мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який дозволяє класифікувати емоції. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – вибір основного фреймворку глибокого навчання.

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python.

б) C++.

Функція F_2.

а) PyTorch.

б) TensorFlow.

Функція F_3:

а) Jupyter Notebook.

б) Visual Studio.

Варіанти реалізації основних функцій наведені у морфологічній карті системи нижче.

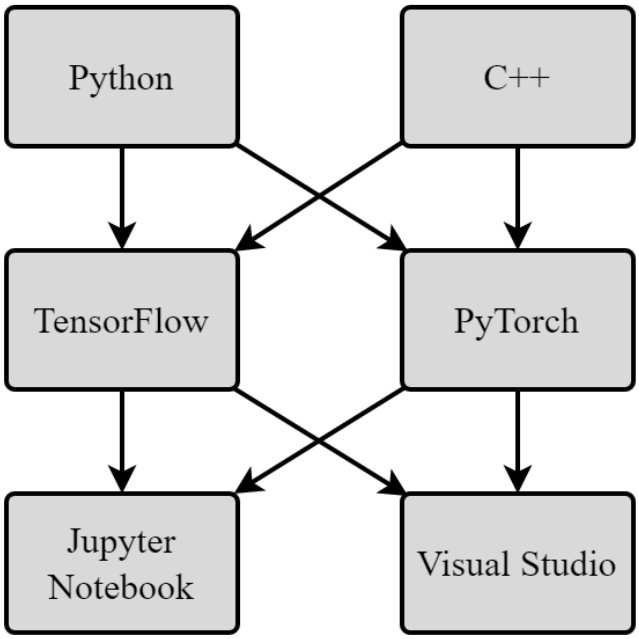

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*Python*	Швидка розробка програми, доступність	Низька швидкість виконання програми

		бібліотек, кросплатформеність	
	C++	Велика швидкість виконання коду	Складний синтаксис, мала кількість бібліотек для глибокого навчання, великий час розробки
F_2	*TensorFlow*	Велика екосистема, простіше розгортання на сервері, кращий для масштабних промислових проєктів, багато готових реалізацій	Складніший для навчання, мала гнучкість, проблеми з налагодженням і дебагінгом
	PyTorch	Велика гнучкість, простий дебагінг, більш пітоністичний і простий для розуміння, кращий для наукових проєктів	Менше готових реалізованих рішень та інструментів, гірший при розгортанні
F_3	*Jupyter Notebook*	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері, інтуїтивно простий	Мало готових реалізованих інструментів, що могли б полегшити розробку
	Visual Studio	Багато готових реалізованих фіч, підсвітки синтаксису, автозаповнення і т. п.	Підтримує одночасно лише одну мову програмування, громіздкий

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу даємо швидкості розробки та простоті. Python має величезну кількість бібліотек і готових реалізованих функцій, що зроблять життя комфортним та приємним.

Функція F_2:

Обидва варіанти задовольняють моїм вимогам, вони є повністю взаємозамінними в моєму випадку.

Функція F_3:

Оскільки мовою програмування було обрано Python, то було обрано варіант Jupyter Notebook.

Отже, будемо розглядати такі варіанти реалізації програмного забезпечення:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія мови програмування;

X2 – об’єм пам’яті для обчислень та збереження даних;

X3 – час навчання на даних;

X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час витрачений на навчання	X3	годин	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	10000

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

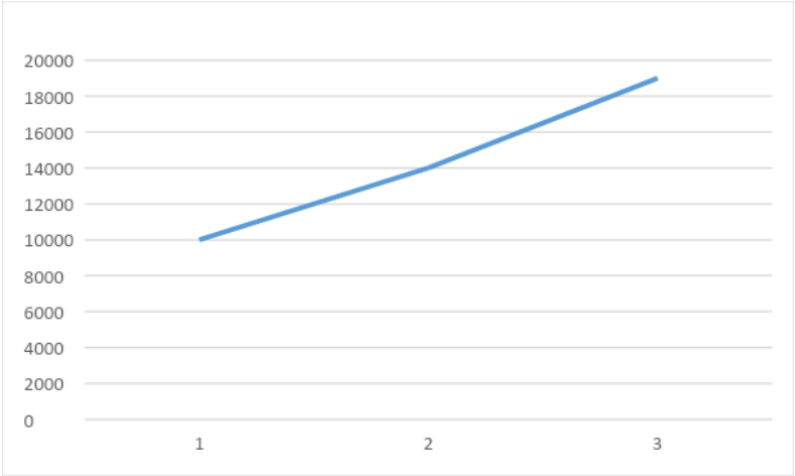

Рисунок 4.2 – X1, швидкодія мови програмування

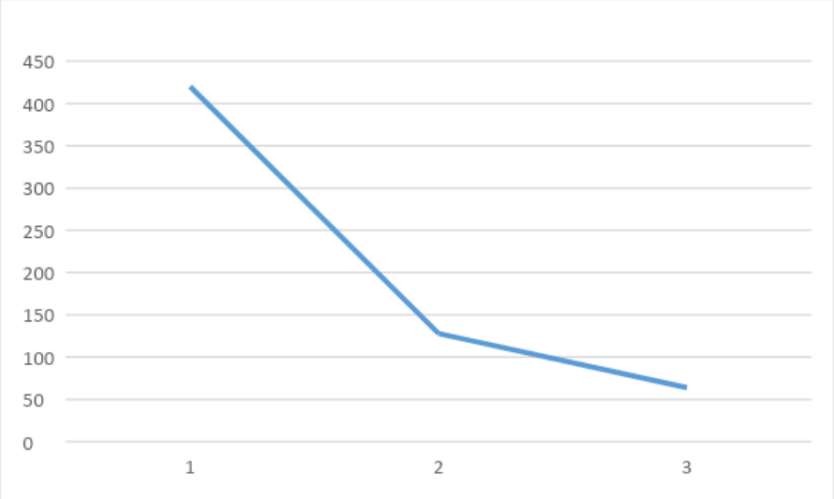

Рисунок 4.3 – X2, об’єм пам’яті

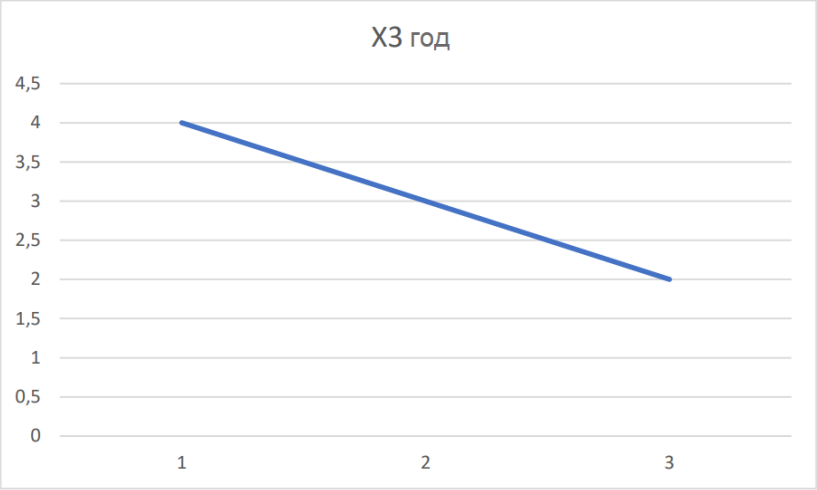

Рисунок 4.4 – X3, час навчання

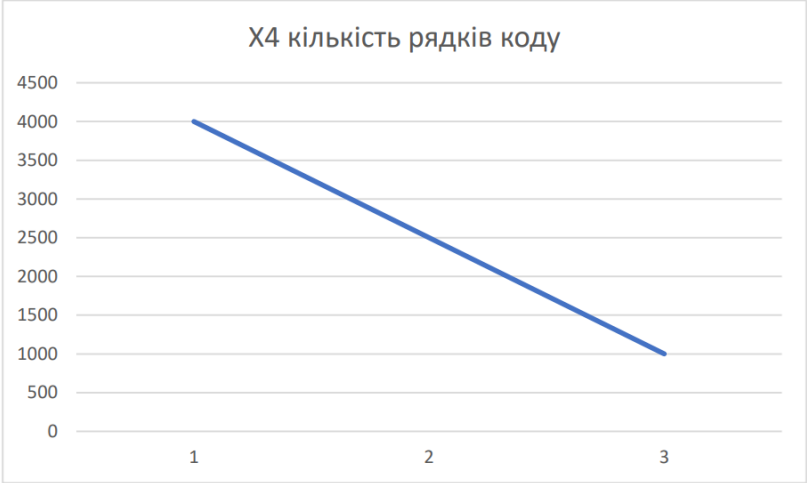

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретної цілі, що була поставлена вище.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці нижче.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта			Δ_i^2

			1	2	3	4	5	6	7	Сума рангів R_i	Відхи-лення Δ_i	
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3.5	12.25
X2	Об’єм пам’яті	Mb	2	1	3	1	2	1	2	12	-12.5	156.25
X3	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5.5	30.25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3.5	12.25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \qquad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24.5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 211. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0.86 > W_k = 0.67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	>	>	>	>	>	0.5
X1 і X3	<	>	<	=	<	<	>	<	0,5
X1 і X4	>	=	<	>	<	=	>	>	1.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	>	<	>	>	<	>	<	>	1.5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 5%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 5%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	1.5	0,5	1.5	4.5	0.28	16.25	0.275	59.125	0.274
X2	0.5	1	0.5	0.5	2.5	0.16	9.25	0.157	34.125	0.156
X3	1,5	1.5	1	1.5	5.5	0.34	21.25	0.360	77.875	0.361
X4	0.5	1.5	0.5	1	3.5	0.22	12.25	0.208	44.875	0.209
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм пам’яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	А	Х1	20000	7	0.274	1.918
F_2	А	Х2	1028	2	0.156	0.312
	Б	Х3	64	2	0.361	0.722
F_3	А	Х4	965	6	0.209	1.254

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 1.918 + 0.312 + 1.254 = 3.484$;

$K_{K2} = 1.918 + 0.722 + 1.254 = 3.894.$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення. Тому обираємо його, тобто PyTorch.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$\mathrm{T_O} = \mathrm{T_P} \cdot \mathrm{K_П} \cdot \mathrm{K_{СК}} \cdot \mathrm{K_М} \cdot \mathrm{K_{СТ}} \cdot \mathrm{K_{СТ.М}}, \qquad (4.13)$$

де $\mathrm{T_P}$ – трудомісткість розробки ПП;

$\mathrm{K_П}$ – поправочний коефіцієнт;

$\mathrm{K_{СК}}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ = 50 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 50 \cdot 1.8 \cdot 0.9 = 81$$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =28 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 28 \cdot 0.9 \cdot 0.8 = 20.16$$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I$ = (81 + 20.16 + 4.8 + 20.16) · 8 = 1008.96 людино-годин.

$Т_{II}$ = (81 + 20.16 + 6.91 + 20.16) · 8 = 1025.84 людино-годин.

Найбільш високу трудомісткість має варіант II.

В розробці бере участь єдиний програміст-аналітик з окладом 24600 грн. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів у місяці;

t – кількість робочих годин в день.

$$С_Ч = \frac{24600}{21 \cdot 8} = 146.43 \ \ грн. \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_Ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 146.43 \cdot 1008.96 \cdot 1.2 = 177290.416$ грн.

II. $С_{ЗП} = 146.43 \cdot 1025.84 \cdot 1.2 = 180256.501$ грн.

Відрахування на єдиний соціальний внесок на момент написання роботи становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 177290.416 \cdot 0.22 = 39003.89$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 180256.501 \cdot 0.22 = 39656.43$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 24600 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 24600 \cdot 0.2 = 59040$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 59040 \cdot (1 + 0.2) = 70848$ грн.

Відрахування на соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 70848 \cdot 0.22 = 15586.56$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.1 \cdot 0.25 \cdot 25000 = 6875$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.1 \cdot 25000 \cdot 0.08 = 2200$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$
$$= 1677.6 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 \cdot 0.3 \cdot 0.9 \cdot 4.7996 = 2173.99 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 25000 \cdot 0.67 = 16750 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$C_{ЕКС}$ = 70848 + 15586.56 + 6875 + 2200 + 2173.99+ 16750 = 114433.55 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ}$ = 114433.55 / 1677.6 = 68.21 грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М-Г} \cdot T, \quad (4.18)$$

I. $C_М$ = 68.21 · 1008.96 = 68821.16 грн.

II. $C_М$ = 68.21 · 1025.84 = 69972.55 грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0{,}67, \quad (4.19)$$

I. $C_Н$ = 177290.416 · 0.67 = 118784.58 грн.

II. $C_Н$ = 180256.501 · 0.67 = 120771.86 грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н, \quad (4.20)$$

I. $C_{ПП}$ = 177290.42 + 39003.89 + 68821.16 + 118784.58 = 403900.05 грн.

II. $C_{ПП}$ = 180256.50 + 39656.43 + 69972.55 + 120771.86 = 410657.34 грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР\,j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$К_{ТЕР1} = 3.484 / 403900.05 = 8.626 \cdot 10^{-6}$,

$К_{ТЕР2} = 3.894 / 410657.34 = 9.482 \cdot 10^{-6}$.

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 9.482 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 9.482 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Вибір фреймворку – PyTorch;
- Мова програмування – Python;
- Середовище розробки – Jupyter Notebook.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

4.8 Висновки до четвертого розділу

У даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу обрано варіант реалізації програмного продукту.

ВИСНОВКИ

Ця дипломна робота є дослідженням на тему застосування штучних нейронних мереж для розпізнавання емоцій на людському обличчі за фотографією. Метою було дослідити вплив роздільної здатності зображень, наявності у них кольору, архітектури мережі на якість роботи моделі та точність класифікацію, та побудувати програмний продукт з графічним інтерфейсом, що буде здатен розпізнавати емоції. Ці поставлені задачі були виконані.

Була досліджена література за темою і оглянута предметна область. Було досліджені математичні основи роботи згорткових нейронних мереж. Були навчені 15 різних моделей, на основі результатів яких було зроблено висновки щодо впливу перерахованих у попередньому абзаці факторів та роботу моделі. Точність найкращої моделі – 71.5%, що, враховуючи складність задачі та відносно невеликий розмір навчальної вибірки, є дуже гарним результатом. Було створено інтуїтивно зрозумілий GUI для наочного представлення роботи найкращої моделі.

СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ

1. Vemou K. Facial Emotion Recognition [Електронний ресурс] / K. Vemou, A. Horvath // EDPS TechDispatch. – 2021. – Режим доступу до ресурсу: https://edps.europa.eu/system/files/2021-05/21-05-26_techdispatch-facial-emotion-recognition_ref_en.pdf.

2. Mellouk W. Facial emotion recognition using deep learning: review and insights [Електронний ресурс] / W. Mellouk, W. Handouzi // Procedia Computer Science. – 2020. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S1877050920318019.

3. Khan A. Facial Emotion Recognition Using Conventional Machine Learning and Deep Learning Methods: Current Achievements, Analysis and Remaining Challenges [Електронний ресурс] / Amjad Rehman Khan // Information. – 2022. – Режим доступу до ресурсу: https://www.mdpi.com/2078-2489/13/6/268

4. Connie T. Facial Expression Recognition Using a Hybrid CNN– SIFT Aggregator [Електронний ресурс] / T. Connie, M. Al-Shabi, W. Cheah // Springer International Publishing. – 2017. – Режим доступу до ресурсу: https://arxiv.org/ftp/arxiv/papers/1608/1608.02833.pdf.

5. Yi H. Research on facial expression recognition based on Multimodal data fusion and neural network [Електронний ресурс] / H. Yi, W. Xubin, L. Zhengyu. – 2021. – Режим доступу до ресурсу: https://arxiv.org/ftp/arxiv/papers/2109/2109.12724.pdf

6. Kingma D. Adam: A Method for Stochastic Optimization [Електронний ресурс] / D. Kingma, J. Ba // ICLR. – 2014. – Режим доступу до ресурсу: https://arxiv.org/pdf/1412.6980.pdf.

7. Deep Residual Learning for Image Recognition [Електронний ресурс] / K.He, X. Zhang, S. Ren, J. Sun // CVPR. – 2015. – Режим доступу до ресурсу: https://arxiv.org/pdf/1512.03385.pdf.

8. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Електронний ресурс] / [A. Howard, M. Zhu, B. Chen та ін.] // Google Inc.. – 2017. – Режим доступу до ресурсу: https://arxiv.org/pdf/1704.04861.pdf.

9. MobileNetV2: Inverted Residuals and Linear Bottlenecks [Електронний ресурс] / [M. Sandler, A. Howard, M. Zhu та ін.] // IEEE/CVF. – 2018. – Режим доступу до ресурсу: https://arxiv.org/pdf/1801.04381.pdf.

10. Searching for MobileNetV3 [Електронний ресурс] / [A. Howard, M. Sandler, G. Chu та ін.] // IEEE/CVF. – 2019. – Режим доступу до ресурсу: https://arxiv.org/pdf/1905.02244.pdf.

11. CBAM: Convolutional Block Attention Module [Електронний ресурс] / S.Woo, J. Park, J. Y. Lee, I. S. Kweon // ECCV. – 2018. – Режим доступу до ресурсу: https://arxiv.org/pdf/1807.06521.pdf.

12. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization [Електронний ресурс] / [R. R. Selvaraju, M. Cogswell, A. Das та ін.] // IEEE International Conference on Computer Vision (ICCV). – 2017. – Режим доступу до ресурсу: https://arxiv.org/pdf/1610.02391.pdf.

13. Facial expression recognition with convolutional neural networks via a new face cropping and rotation strategy [Електронний ресурс] / [Y. Jin, R. Han, M. Akram та ін.] // The Visual Computer. – 2020. – Режим доступу до ресурсу: https://www.researchgate.net/publication/330301581_Facial_expression_recognition_with_convolutional_neural_networks_via_a_new_face_cropping_and_rotation_strategy.

14. Modawal A. How facial recognition and emotion detection helps businesses [Електронний ресурс] / Archana Modawal // Softweb Solutions Inc.. – 2022. – Режим

доступу до ресурсу: https://www.softwebsolutions.com/resources/benefits-of-facial-recognition.html.

15. Models and pre-trained weights – Torchvision 0.15 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/vision/stable/models.html.

16. Conv2d - PyTorch 2.0 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.

17. AffectNet – Mohammad H. Mahoor, PhD [Електронний ресурс] – Режим доступу до ресурсу: http://mohammadmahoor.com/affectnet/.

18. Wolfewicz A. Deep Learning vs. Machine Learning – What's The Difference? [Електронний ресурс] / Arne Wolfewicz // Levity. – 2023. – Режим доступу до ресурсу: https://levity.ai/blog/difference-machine-learning-deep-learning.

19. Yan W. How to Handle Overfitting in Deep Learning? [Електронний ресурс] / Wai Yan // Medium. – 2018. – Режим доступу до ресурсу: https://medium.com/@waiyan.nn18/what-is-over-fitting-how-to-avoid-57cd72fa7e8.

20. Overfitting [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Overfitting.

21. Budhiraja A. Dropout in (Deep) Machine learning [Електронний ресурс] / Amar Budhiraja // Medium. – 2016. – Режим доступу до ресурсу: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

22. Dropout – PyTorch 2.0 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html.

23. Huber J. Batch normalization in 3 levels of understanding [Електронний ресурс] / Johann Huber // Towards Data Science. – 2020. – Режим доступу до ресурсу: https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338.

24. Metrics and scoring: quantifying the quality of predictions – scikit-learn 1.2.2 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org/stable/modules/model_evaluation.html.

25. Géron A. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow / Aurélien Géron. –Sebastopol, CA, USA: O'Reilly, 2019. – 484 с.

26. Convolution arithmetic [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md.

27. Elmahmudi A. A framework for facial age progression and regression using exemplar face templates [Електронний ресурс] / A. Elmahmudi, H. Ugail // Springer. – 2020. – Режим доступу до ресурсу: https://www.researchgate.net/publication/343699139_A_framework_for_facial_age_progression_and_regression_using_exemplar_face_templates.

ДОДАТОК А КОД

Кінцевий програмний продукт:

```
from tkinter import Tk, ttk
from tkinter.filedialog import askopenfilename
from PIL import Image, ImageTk

import numpy as np
import matplotlib.pyplot as plt
import dlib
import cv2

import torch
from torchvision.transforms import Compose, ToTensor, Normalize
from torch import nn
import torch.nn.functional as F

selected_image = None

device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)

class ChannelAttention(nn.Module):
    def __init__(self, in_channels, ratio):
        super().__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)
        self.sigmoid = nn.Sigmoid()

        self.mlp = nn.Sequential(
            nn.Conv2d(in_channels, in_channels//ratio, 1, padding=0,
bias=False),
            nn.ReLU(),
            nn.Conv2d(in_channels//ratio, in_channels, 1, padding=0, bias=False)
        )
        nn.init.kaiming_normal_(self.mlp[0].weight, mode='fan_in',
nonlinearity='relu')
        nn.init.xavier_normal_(self.mlp[-1].weight)
    def forward(self, x):
        avg_desc = self.avg_pool(x)
        max_desc = self.avg_pool(x)

        avg_desc = self.mlp(avg_desc)
        max_desc = self.mlp(max_desc)

        scale = self.sigmoid(avg_desc + max_desc)
        return scale * x

class SpatialAttention(nn.Module):
    def __init__(self, kernel_size):
        super().__init__()
```

```
        self.conv = nn.Conv2d(2, 1, kernel_size, padding='same', bias=False)
        self.sigmoid = nn.Sigmoid()

        nn.init.xavier_normal_(self.conv.weight)
    def forward(self, x):
        avg_map = torch.mean(x, dim=1, keepdim=True)
        max_map, _ = torch.max(x, dim=1, keepdim=True)

        feature_map = torch.cat([avg_map, max_map], dim=1)
        feature_map = self.conv(feature_map)
        scale = self.sigmoid(feature_map)
        return scale * x

model = torch.load('best-model.pt')
model.to(device)
model.eval()

def predict(model, image):
    loader = Compose([ToTensor(),
                      Normalize([0.5943, 0.4672, 0.4130], [0.30008, 0.28751,
0.27885])])
    tensor = loader(image).unsqueeze(0).to(device)
    with torch.inference_mode():
        logits = model(tensor)
        answer = logits.argmax().cpu().item()
    return answer

class GradCAM:
    def __init__(self, model):
        self.model = model.eval()
        self.feature_layer = self.model.features[18]
        self.feature_map = None
        self.gradients = None

    def features_hook(self, module, input, output):
        self.feature_map = output
    def gradient_hook(self, module, input, output):
        self.gradients = output[0]

    def generate_map(self, image, target, orig_img):
        handle_features =
self.feature_layer.register_forward_hook(self.features_hook)
        handle_gradients =
self.feature_layer.register_full_backward_hook(self.gradient_hook)

        loader = Compose([ToTensor(),
                          Normalize([0.5943, 0.4672, 0.4130], [0.30008, 0.28751,
0.27885])])
        tensor = loader(image).unsqueeze(0).to(device)

        out = self.model(tensor)
        grad_hot = torch.nn.functional.one_hot(torch.tensor([target]),
7).to(device)

        self.model.zero_grad()
        self.feature_layer.zero_grad()
        out.backward(gradient=grad_hot)

        handle_features.remove()
        handle_gradients.remove()

        imp_weights = F.adaptive_avg_pool2d(self.gradients, 1)
        cam = imp_weights * self.feature_map
```

```
        cam = cam.sum(dim=1, keepdim=True)
        cam = F.relu(cam)

        cam = cam * (1/cam.max())
        cam = F.interpolate(cam, size=(orig_img.shape[0], orig_img.shape[1]),
mode='bilinear', align_corners=False)
        return np.squeeze(cam.detach().clone().cpu().numpy())

def create_heatmap(model, image, emotion_num, orig_img):
    g_cam = GradCAM(model)
    heatmap = g_cam.generate_map(image, emotion_num, orig_img)

    heatmap = plt.get_cmap('jet')(heatmap)[:,:,:3]
    result = (0.7 * orig_img/255 + 0.3 * heatmap)
    result = Image.fromarray((result*255).astype('uint8'), mode='RGB')
    return result

def preprocess_image(orig_img):
    predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
    if orig_img.format != 'RGB':
        orig_img = orig_img.convert('RGB')

    # Rotate
    image = np.array(orig_img)

    detector = dlib.get_frontal_face_detector()
    faces = detector(image)

    try:
        landmarks = predictor(image, faces[0])
    except:
        rect = dlib.rectangle(0, 0, image.shape[0], image.shape[1])
        faces.append(rect)
        landmarks = predictor(image, faces[0])

    x_sum_lefteye = x_sum_righteye = y_sum_lefteye = y_sum_righteye = 0
    for i_left in range(36, 42):
        x = landmarks.part(i_left).x
        y = landmarks.part(i_left).y
        x_sum_lefteye += x
        y_sum_lefteye += y

    for i_right in range(42, 48):
        x = landmarks.part(i_right).x
        y = landmarks.part(i_right).y
        x_sum_righteye += x
        y_sum_righteye += y

    angle = np.rad2deg( np.arctan((y_sum_righteye - y_sum_lefteye) /
(x_sum_righteye - x_sum_lefteye)) )

    im_rotated = orig_img.rotate(angle, resample = Image.Resampling.BICUBIC)

    #Crop
    image_rotated = np.array(im_rotated)
    faces = detector(image_rotated)

    try:
        landmarks = predictor(image_rotated, faces[0])
    except:
        rect = dlib.rectangle(0, 0, image_rotated.shape[0],
image_rotated.shape[1])
        faces.append(rect)
```

```
        landmarks = predictor(image_rotated, faces[0])

    x_sum_lefteye = x_sum_righteye = y_sum_lefteye = y_sum_righteye = 0
    for i_left in range(36, 42):
        x = landmarks.part(i_left).x
        y = landmarks.part(i_left).y
        x_sum_lefteye += x
        y_sum_lefteye += y

    for i_right in range(42, 48):
        x = landmarks.part(i_right).x
        y = landmarks.part(i_right).y
        x_sum_righteye += x
        y_sum_righteye += y

    d = (x_sum_righteye - x_sum_lefteye) / 6
    y_mean = (y_sum_lefteye + y_sum_righteye) / 12

    top_border = max(0, y_mean - 0.6*d)
    left_border = min(landmarks.part(0).x, landmarks.part(1).x,
landmarks.part(2).x,
                      landmarks.part(17).x, landmarks.part(36).x,)
    right_border = max(landmarks.part(16).x, landmarks.part(15).x,
landmarks.part(14).x,
                       landmarks.part(26).x, landmarks.part(45).x)
    bottom_border = min(image_rotated.shape[1], landmarks.part(8).y)

    im_crop = im_rotated.crop((left_border, top_border, right_border,
bottom_border))

    #Histogram equalization
    hsv_image = cv2.cvtColor(np.array(im_crop), cv2.COLOR_BGR2YCrCb)
    hsv_image[:,:,0] = cv2.equalizeHist(hsv_image[:,:,0])
    equalized_image = cv2.cvtColor(hsv_image, cv2.COLOR_YCrCb2BGR)
    res_im = Image.fromarray(equalized_image, mode='RGB')
    return res_im

def select_image():
    filepath = askopenfilename(filetypes=[("Image files",
"*.png;*.jpg;*.jpeg")])
    if filepath:
        img = Image.open(filepath)
        global selected_image
        selected_image = img
        photo = img.resize((int((224 / img.size[1]) * img.size[0]), 224),
resample=Image.LANCZOS)  # Resize the image as desired
        photo = ImageTk.PhotoImage(photo)
        image_label.configure(image=photo)
        image_label.image = photo  # Keep a reference to avoid garbage
collection

def classify():
    if selected_image is None:
        text_class['text'] = 'No image :('
        return
    image = preprocess_image(selected_image)
    image_show = image.resize((int((256 / image.size[1]) * image.size[0]), 256),
resample=Image.LANCZOS)
    image_show = ImageTk.PhotoImage(image_show)
    prep_img.configure(image=image_show)
    prep_img.image = image_show

    image_cl = image.resize((224, 224), resample=Image.LANCZOS)
```

```
    emotions = {
        0: 'ANGRY',
        1: 'DISGUST',
        2: 'FEAR',
        3: 'HAPPY',
        4: 'NEUTRAL',
        5: 'SAD',
        6: 'SURPRISED'
    }
    emotion_num = predict(model, image_cl)
    text_class['text'] = f'Detected emotion: {emotions[emotion_num]}'

    image_gradcam = create_heatmap(model, np.array(image_cl), emotion_num,
np.array(image))
    image_gradcam = image_gradcam.resize((int((256 / image_gradcam.size[1]) *
image_gradcam.size[0]), 256), resample=Image.LANCZOS)
    image_gradcam= ImageTk.PhotoImage(image_gradcam)
    gradcam_img.configure(image=image_gradcam)
    gradcam_img.image = image_gradcam

    prep_desc['text'] = 'Preprocessed image'
    gradcam_desc['text'] = f'Grad-CAM for emotion {emotions[emotion_num]}'


window = Tk()
window.title("Facial Emotions Recognition")
window.iconbitmap("icon.ico")
window.geometry("625x725")
window.resizable(False, False)

select_button = ttk.Button(window, text="Select image", command=select_image)
select_button.place(relx=0.5, rely=0.04, anchor="center")
clssify_button = ttk.Button(window, text="Recognize emotion", command=classify)
clssify_button.place(relx=0.5, rely=0.42, anchor="center")

image_label = ttk.Label()
image_label.place(relx=0.5, rely=0.07, anchor="n")

text_class = ttk.Label(text="", font=(None, '10'))
text_class.place(relx=0.02, rely=0.5, anchor="w")

prep_img = ttk.Label()
prep_img.place(relx=0.02, rely=0.75, anchor="w")

gradcam_img = ttk.Label()
gradcam_img.place(relx=0.98, rely=0.75, anchor="e")

prep_desc = ttk.Label(font=(None, '10'))
prep_desc.place(relx=0.02, rely=0.95, anchor="w")

gradcam_desc = ttk.Label(font=(None, '10'))
gradcam_desc.place(relx=0.98, rely=0.95, anchor="e")

window.mainloop()
```

В усіх 15-ти навчений моделях код дуже схожий. Тому буде показано лише код, завдяки якому було навчено найкращу модель MobileNet-v2+CBAM на датасеті 224 color. В інших моделях все аналогічно. Модель навчена в Colab.

MobileNet-v2 + CBAM:

```
import torch
import torchvision
from torchvision.transforms import Compose, Grayscale, ToTensor,
RandomHorizontalFlip, RandomRotation, Normalize, Resize
from torch import nn
from torch.utils.data import DataLoader, Subset
from torchvision import transforms, models
import torch.nn.functional as F

from PIL import Image
import numpy as np
import time

from google.colab import drive
drive.mount('/content/drive')
!unzip '/content/drive/MyDrive/224_color.zip'

device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

train_tranform = Compose([
    RandomHorizontalFlip(),
    RandomRotation(2, interpolation=Image.BICUBIC),
    ToTensor(),
    Normalize(mean=[0.5943, 0.4672, 0.4130],
              std=[0.30008, 0.28751, 0.27885])
])

test_transform = Compose([
    ToTensor(),
    Normalize(mean=[0.5943, 0.4672, 0.4130],
              std=[0.30008, 0.28751, 0.27885])
])
train_data = torchvision.datasets.ImageFolder('/content/train', transform =
train_tranform)
val_data = torchvision.datasets.ImageFolder('/content/train', transform =
test_transform)

train_dataloader = DataLoader(train_data, batch_size=32, shuffle=True)
val_dataloader = DataLoader(val_data, batch_size=32, shuffle=True)

class ChannelAttention(nn.Module):
    def __init__(self, in_channels, ratio):
        super().__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)
        self.sigmoid = nn.Sigmoid()

        self.mlp = nn.Sequential(
            nn.Conv2d(in_channels, in_channels//ratio, 1, padding=0,
bias=False),
            nn.ReLU(),
```

```
            nn.Conv2d(in_channels//ratio, in_channels, 1, padding=0, bias=False)
        )
        nn.init.kaiming_normal_(self.mlp[0].weight, mode='fan_in',
nonlinearity='relu')
        nn.init.xavier_normal_(self.mlp[-1].weight)
    def forward(self, x):
        avg_desc = self.avg_pool(x)
        max_desc = self.avg_pool(x)

        avg_desc = self.mlp(avg_desc)
        max_desc = self.mlp(max_desc)

        scale = self.sigmoid(avg_desc + max_desc)
        return scale * x

class SpatialAttention(nn.Module):
    def __init__(self, kernel_size):
        super().__init__()
        self.conv = nn.Conv2d(2, 1, kernel_size, padding='same', bias=False)
        self.sigmoid = nn.Sigmoid()

        nn.init.xavier_normal_(self.conv.weight)
    def forward(self, x):
        avg_map = torch.mean(x, dim=1, keepdim=True)
        max_map, _ = torch.max(x, dim=1, keepdim=True)

        feature_map = torch.cat([avg_map, max_map], dim=1)
        feature_map = self.conv(feature_map)
        scale = self.sigmoid(feature_map)
        return scale * x

model10 = models.mobilenet_v2(weights='MobileNet_V2_Weights.IMAGENET1K_V2')
model10.classifier = nn.Sequential(
    nn.Dropout(p=0.2, inplace=False),
    nn.Linear(in_features=1280, out_features=7, bias=True)
)
for block in range(1, 18):
    if block == 1:
        r = 4
    elif block >= 2 and block <= 4:
        r = 8
    else:
        r = 16
    model10.features[block].conv = nn.Sequential(
        *model10.features[block].conv[:-2],
        ChannelAttention(model10.features[block].conv[-3][0].in_channels, r),
        SpatialAttention(7),
        *model10.features[block].conv[-2:]
    )
model10.to(device)

pytorch_total_params = sum(p.numel() for p in model10.parameters() if
p.requires_grad)
print(pytorch_total_params)

def train_loop(dataloader, model, loss_fn, optimizer, start):
    model.train()
    size = len(dataloader.dataset)
    for batch, data in enumerate(dataloader):
        X, y = data[0].to(device), data[1].to(device)
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)
```

```
        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

def test_loop(dataloader, model, loss_fn, loss, acc):
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    loss.append(test_loss)
    acc.append(correct)
    print(f"Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f}")

epochs = 40
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model10.parameters(), weight_decay=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.939)

loss_val = []
acc_val = []

best_acc = 0
start = time.time()
for t in range(epochs):
    print(f"\nEpoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model10, loss_fn, optimizer, start)

    print("Results:")
    print("Validation ", end = '')
    test_loop(val_dataloader, model10, loss_fn, loss_val, acc_val)

    if acc_val[-1] > best_acc:
        best_acc = acc_val[-1]
        torch.save(model10, '/content/best-model10.pt')

    scheduler.step()
    end = time.time()
    print("Time:", round(end - start, 2), 'sec')
print("Done!")
end = time.time()
print("final for", epochs, "epochs:", round(end - start, 2), "sec")
```