МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра прикладної математики та моделювання складних систем

«До захисту допущено»
Завідувач кафедри
_____ Ігор КОПЛИК
(підпис)
____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

зі спеціальності 113 Прикладна математика,
освітньо-професійної програми Наука про дані та моделювання складних систем
на тему: «Система аналізу емоційного стану людини через розпізнавання облич та міміки на зображеннях»

Здобувача групи ПМ.м-21 Прокопенка Микити Едуардовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Микита ПРОКОПЕНКО
(підпис)

Керівник канд. фіз.- мат. наук, доцент, Уляна ШВЕЦЬ _____
(підпис)

Суми – 2023

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет **електроніки та інформаційних технологій**
Кафедра **прикладної математики та моделювання складних систем**
Рівень вищої освіти другий (магістр)

Галузь знань **11 Математика та статистика**
Спеціальність **113 Прикладна математика**
Освітня програма **освітньо-професійна** «Наука про дані та моделювання складних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМтаМСС
Ігор КОПЛИК_____
«___»_____2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Прокопенко Микита Едуардович

1. Тема роботи Система аналізу емоційного стану людини через розпізнавання облич та міміки на зображеннях

Керівник роботи кандидат фізико-математичних наук, доцент, Швець Уляна Станіславівна

затверджую наказом по факультету ЕлІТ від «07» листопада 2023 р. № 1237-VI

2. Термін подання роботи студентом «16» грудня 2023 р.

3. Вихідні данні до роботи: набір зображень облич людей з різними виразами обличчя та емоційними станами

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити): створення ефективної моделі, яка здатна точно визначати емоційний стан осіб на зображеннях.

5. Перелік графічного матеріалу

1) Види базових емоцій.

2) Аналіз емоційного стану за допомогою технологій розпізнавання обличчя.

3) Архітектура простої нейронної мережі.

4) Процес навчання нейронної мережі.

5) Спрощена ілюстрація мережі CNN.

6) Архітектура CNN.

7) Ілюстрація операції згортання.

8) Операція згортання.

9) Операція об’єднання.

10) Алгоритм роботи моделі.

11) Графік точності на тренувальному та валідаційному наборах під час тренування.

12) Матриця плутанини класифікації емоцій.

13) Результати роботи моделі на валідаційних даних.

14) Результати роботи моделі на зображеннях особистих виразів обличчя та емоцій.

6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» листопада 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання роботи	Примітка
1	Літературний огляд понять аналізу емоційного стану людини та теоретичних основ архітектури нейронних мереж, процесу навчання та методів оптимізації градієнта для них.	06.11.2023 – 19.11.2023	
2	Формування бази даних для розробки моделі на основі отриманих даних із інших джерел.	20.11.2023 – 26.11.2023	
3	Програмна реалізація моделі системи аналізу емоційного стану людини через розпізнавання облич з використанням алгоритмів оптимізації градієнта. Отримання результатів.	27.11.2023 – 06.12.2023	
4	Тестування моделі та її вдосконалення. Аналіз результатів.	07.12.2023 – 11.12.2023	
5	Написання тексту кваліфікаційної роботи.	12.12.2023 – 16.12.2023	

Здобувач вищої освіти _____ Микита ПРОКОПЕНКО

Керівник роботи _____ Уляна ШВЕЦЬ

АНОТАЦІЯ

Звіт містить: 62 с., 14 рис., 2 табл., 22 джерела, 3 додатки.

Мета роботи: створення ефективної моделі, що здатна точно визначати емоційний стан осіб на зображеннях.

Об’єкт досліджень: фотографії людей з різними емоційними станами.

Предмет дослідження: алгоритми оптимізації градієнта, які використовуються для покращення ефективності та точності моделі.

У роботі здійснено детальний аналіз основних концепцій нейронних мереж та ключових алгоритмів оптимізації градієнта, таких як SGD, SGD з моментом, RMSprop та Adam. Був проведений порівняльний аналіз ефективності чотирьох алгоритмів оптимізації градієнта. На базі цього була створена модель системи, спрямована на аналіз емоційного стану людини через розпізнавання облич. Одержані результати свідчать про успішність використання згорткових нейронних мереж у поєднанні з зазначеними алгоритмами оптимізації градієнта. Збільшення обсягу тренувального датасету та повних проходів через весь тренувальний набір дозволило покращити точність моделі.

Ключові слова: КОМП’ЮТЕРНИЙ ЗІР, НЕЙРОННА МЕРЕЖА, АЛГОРИТМ ОПТИМІЗАЦІЇ ГРАДІЄНТА, CNN.

ЗМІСТ

ВСТУП

У сучасному світі, де високорозвинені технології вже не просто частина нашого повсякдення, але й визначальний аспект соціального, емоційного та психологічного взаємодії, виникає важлива потреба в розробці та впровадженні технологічних рішень, спрямованих на розуміння та інтерпретацію людських емоцій. Моя робота зосереджена на створенні системи, яка застосовує передові технології розпізнавання облич та міміки для об'єктивного визначення емоційного стану особи.

У контексті цього дослідження, нейронні мережі, зокрема згортокові (CNN), стають ключовим інструментом для аналізу та інтерпретації великої кількості інформації, отриманої з зображень облич. Враховуючи комплексність виразів та субтільність емоційного вираження, використання таких алгоритмів стає важливим етапом у побудові системи, яка здатна точно та об'єктивно аналізувати емоційний стан.

Крім технічних аспектів, у цій роботі приділяється увага практичному застосуванню одержаних результатів. Створення моделі, яка здатна визначати емоційний стан, має широкі перспективи у різних сферах: від розваг та віртуальної взаємодії до медичних досліджень і психологічної допомоги. Розвиток подібних технологій може відкрити нові горизонти для розуміння та сприйняття емоцій як в індивідуальному, так і в колективному контексті, надаючи інструмент для покращення якості життя та взаєморозуміння в суспільстві.

1. ОСНОВНІ ПОНЯТТЯ АНАЛІЗУ ЕМОЦІЙНОГО СТАНУ ЛЮДИНИ

1.1 Значення аналізу емоційного стану

Аналіз емоційного стану людини є однією з важливих галузей сучасних досліджень, оскільки емоції відіграють вирішальну роль у житті кожної людини. Емоції впливають на наше сприйняття світу, міжособистісні стосунки, прийняття рішень та загальний стан психіки. Розуміння та аналіз емоційного стану має безліч застосувань у різних галузях життя.

Емоції, особливий вид психічних процесів, дозволяють живим істотам швидко та ефективно реагувати на зовнішні впливи, що є важливим для їх виживання. Емоції також відіграють ключову роль у взаємодії людини з самою собою та навколишнім світом. Їх прояв може відбуватися через голосові вирази, рухи, міміку, пози та вегетативні реакції, але найбільш виразною є обличчя людини. Аналізуючи вираз обличчя, можна здобути значну частину інформації щодо емоційного стану особи. Передача та розуміння емоційного виразу через обличчя відіграє невід'ємну роль у міжособистісних відносинах, забезпечуючи важливий елемент невербальної комунікації [1].

Основна частина систем автоматичного розпізнавання виразу обличчя здійснює класифікацію безпосередньо за основними емоціями, які представлені на рис. 1.1.

У сучасному світі із зростанням інтересу до штучного інтелекту, машинного навчання та обробки зображень виникають нові можливості в галузі аналізу емоційного виразу обличчя. Системи розпізнавання обличчя на основі штучного інтелекту можуть виявляти та аналізувати емоційні вирази в реальному часі. Це важливий крок у напрямку розширення можливостей технологій для покращення спілкування та розвитку нових додатків у сферах медицини, робототехніки, безпеки та багатьох інших галузях.

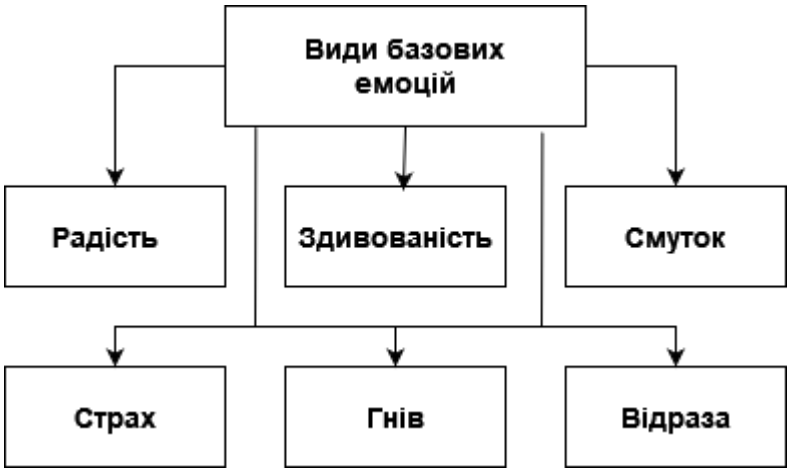

Рис. 1.1 – Види базових емоцій

Збільшена увага до аналізу емоційного виразу обличчя свідчить про його значущість у сучасному світі, який вимагає більш глибокого розуміння та використання наших емоцій для покращення нашого життя та взаємодії з технологіями. Дослідження цієї галузі продовжують надавати нові можливості для розвитку інноваційних рішень та застосувань [2].

1.2 Застосування в аналізі емоційного стану

Емоційний вираз обличчя та міміка є важливою складовою нашого сприйняття світу та способом вираження наших емоцій. Вони відіграють важливу роль у спілкуванні, психології та науці. Розуміння та аналіз цих аспектів дозволяє нам краще розуміти себе та інших, покращувати міжособистісні відносини, сприяти ефективному спілкуванню та розвивати технології для використання в різних галузях життя.

Однією з важливих сфер є медицина, де аналіз емоцій може допомогти у визначенні психічних розладів, таких як депресія, тривожність чи посттравматичний стрес. Відомо, що розпізнавання емоційних реакцій може бути важливим критерієм для діагностики та відслідковування психічного стану пацієнтів. Наприклад, здатність відслідковувати зміни у виразах обличчя та міміці

пацієнта може допомогти вчасно розпізнати симптоми психічних розладів та призначити відповідне лікування.

Для психології аналіз емоційного стану є важливим інструментом для вивчення людської психіки та поведінки. Він допомагає виявити, як емоції впливають на наші рішення, мотивацію та міжособистісні відносини. Зокрема, аналіз міміки та виразів обличчя може допомогти вивчити, які конкретні зміни відбуваються в обличчі людини під впливом різних емоцій, що може призвести до кращого розуміння психологічних механізмів, що стоять за ними.

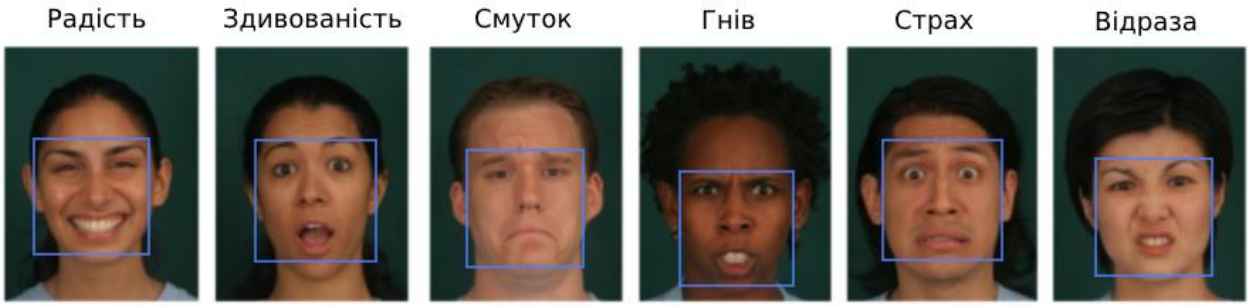

Рис. 1.2 – Аналіз емоційного стану за допомогою технологій розпізнавання облич

У сфері безпеки аналіз емоційного стану може бути корисним для виявлення підозрілої або загрозливої поведінки. Наприклад, системи відеоспостереження можуть виявляти можливі загрози, аналізуючи емоційні вирази на обличчях людей, що може бути корисним у публічних місцях або транспортних вузлах для забезпечення загальної безпеки.

Реклама та маркетинг – це інша галузь, де аналіз емоцій грає важливу роль. Вивчення реакцій споживачів на рекламу дозволяє покращити ефективність рекламних кампаній та підвищити продажі товарів і послуг.

Аналіз емоційного стану через розпізнавання обличчя та міміки відкриває широкий спектр можливостей у різних сферах життя та досліджень. Це допомагає розуміти та аналізувати емоційні реакції людини, що є важливим кроком до зрозуміння людської природи та створення більш ефективних методів взаємодії з навколишнім світом [3,4].

2. НЕЙРОННІ МЕРЕЖІ

2.1 Огляд нейронних мереж

Нейронна мережа (Neural Network) – це математична модель або алгоритм, що імітує структуру та функції людського мозку для розв'язання задач машинного навчання. Вона складається зі з’єднаних нейронів, організованих у шари, і використовується для вирішення різних завдань, таких як класифікація, регресія, генерація контенту, розпізнавання об'єктів і багато інших [5].

Основні характеристики нейронних мереж:

1. *Архітектура:* Нейронні мережі можуть мати різні архітектури, включаючи одношарові та багатошарові. У багатошарових мережах нейрони розташовані у вхідному, прихованому та вихідному шарах.
2. *Зв'язки між нейронами:* Кожен нейрон з'єднаний з іншими нейронами через зв'язки, які мають ваги. Ваги визначають силу зв'язку та його вплив на вихід нейрона.
3. *Функції активації:* Кожен нейрон використовує функцію активації для регулювання виходу відповідно до вхідних сигналів. Популярні функції активації включають сигмоїду, гіперболічний тангенс, ReLU тощо.
4. *Навчання:* Нейронні мережі можуть навчатися на основі вхідних та вихідних даних. Процес навчання включає оптимізацію ваг зв'язків так, щоб мережа правильно вирішувала поставлені завдання.
5. *Функція втрат:* Це метрика, яка визначає різницю між прогнозованим і правильним вихідними даними. Під час навчання мережа намагається мінімізувати цю функцію.

Нейронні мережі використовуються в різних галузях, включаючи комп'ютерний зір, обробку природної мови, рекомендації, медицину та інші. Застосування нейронних мереж в різних галузях дозволяє їм ефективно вирішувати різноманітні завдання завдяки їхній здатності вивчати складні залежності в даних.

2.1.1 Архітектура нейронної мережі

Архітектура нейронної мережі визначається структурою та організацією її нейронів та шарів. Існують різні типи архітектур, включаючи одношарові та багатошарові мережі. Основні компоненти архітектури нейронної мережі включають вхідний шар, приховані шари і вихідний шар [6].

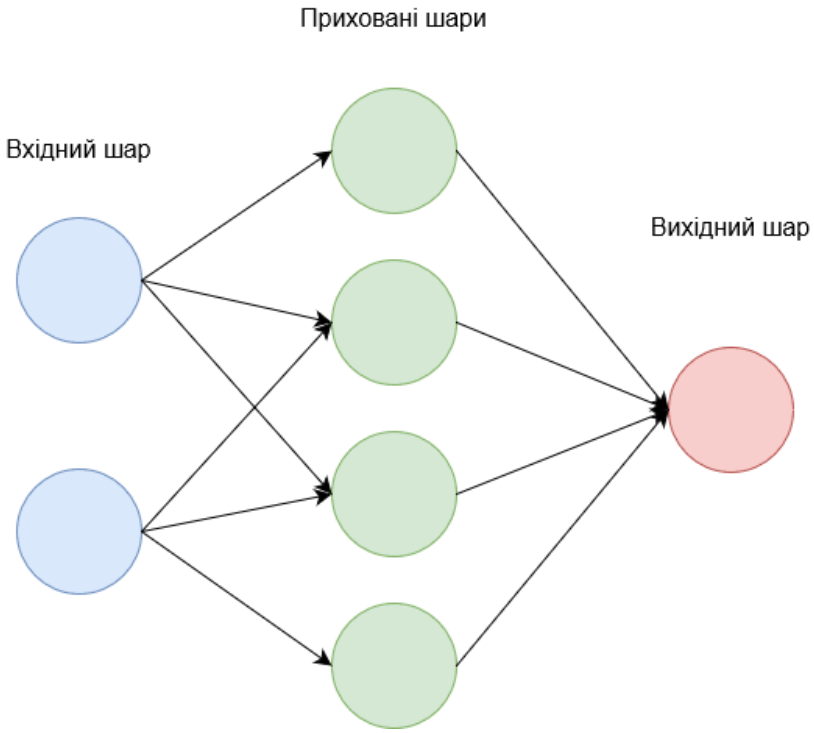

Рис. 2.1. – Архітектура простої нейронної мережі

1. *Вхідний шар (Input Layer):* Це перший шар нейронної мережі, який отримує вхідні дані. Кількість нейронів у вхідному шарі визначається кількістю функціональних або ознак вхідних даних.
2. *Приховані шари (Hidden Layers):* Це шари між вхідним і вихідним шарами. Кожен прихований шар складається з нейронів, і їх кількість може варіюватися залежно від конкретної архітектури мережі. Мережі з одним або більше прихованими шарами називають багатошаровими мережами.
3. *Вихідний шар (Output Layer):* Це останній шар, який видає результати або прогнози. Кількість нейронів у вихідному шарі зазвичай визначається кількістю класів або вихідних значень, які мережа має передбачити.

4. *Зв'язки між нейронами:* Кожен нейрон в одному шарі з’єднаний з кожним нейроном в наступному шарі. Кожен зв'язок між нейронами має вагу, яка визначає важливість цього зв'язку.
5. *Функції активації:* Кожен нейрон використовує функцію активації для визначення свого виходу на основі зважених вхідних сигналів.
6. *Функція втрат:* Це метрика, яка визначає різницю між прогнозованими і правильними вихідними даними. Вона використовується під час процесу навчання для корекції ваг зв’язків.

Наприклад, у простій багатошаровій мережі може бути один вхідний шар, кілька прихованих шарів і один вихідний шар. Однак в глибоких нейронних мережах (глибоке навчання) може бути багато прихованих шарів, що дозволяє моделі вивчати більш складні залежності в даних. Архітектура мережі визначається завданням, яке вона повинна вирішити, і характеристиками вхідних та вихідних даних.

2.1.2 Зв’язки між нейронами

Зв’язки між нейронами у нейронній мережі визначають, як інформація передається від одного нейрона до іншого. Кожен зв'язок між нейронами має вагу, яка визначає його силу або важливість. Загальною метою є навчання мережі таким чином, щоб ваги забезпечували ефективну обробку вхідних даних та правильне вирішення завдань машинного навчання [7].

Основні аспекти зв’язків між нейронами включають:

1. *Вага зв'язку (Weight):* Кожен зв’язок між нейронами має вагу, яка визначає його силу. Вага впливає на величину та напрям передачі сигналу від одного нейрона до іншого. Під час навчання мережі ваги змінюються так, щоб вирішувати конкретне завдання.
2. *Суматор (Summation):* Сигнали, які надходять від попередніх нейронів, множаться на їхні ваги і додаються в суматорі. Це визначає важливість різних вхідні сигналів для певного нейрона.

3. *Функція активації (Activation Function):* Сумарний вихід із суматора передається через функцію активації, яка визначає вихід нейрона на основі отриманих сигналів. Функції активації можуть додавати нелінійність в мережу, що дозволяє їй вирішувати більш складні завдання.
4. *Зворотний зв'язок (Backpropagation):* Під час навчання мережі використовується алгоритм зворотнього поширення помилок, який використовує градієнт функції втрати для корекції ваг зв'язків. Останнє дозволяє мережі адаптуватися до навчальних даних та покращити її можливості вирішувати поставлені завдання.

Зв'язки між нейронами та їх ваги важливі для успішної роботи нейронної мережі. Під час навчання ці ваги оптимізуються, щоб мережа могла вивчати представлення та робити точні прогнози або класифікації.

2.1.3 Функції активації в нейронних мережах

Функції активації в нейронних мережах відірають ключову роль у регулюванні виходу нейрона відповідно до вхідних сигналів. Вони додають нелінійність в модель, дозволяючи мережі вирішувати більш складні завдання. До популярних функції активації належать такі:

1) Сигмоїда (Sigmoid): $\sigma(x) = \frac{1}{1+e^{-x}}$.

   Функція сигмоїди приймає будь-яке вхідне значення і перетворює його до діапазону між 0 і 1. Зазвичай використовується у вихідному шарі для завдань бінарної класифікації.

2) Гіперболічний тангенс: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$.

   Гіперболічний тангенс подібний до сигмоїди, але виводить значення у діапазоні від -1 до 1. Використовується в прихованих шарах нейронних мереж.

3) Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$.

   Функція ReLU повертає вхідне значення, якщо воно більше 0, і 0, якщо

воно менше або рівне 0. Реалізує нелінійність, легко обчислюється і сприяє прискоренню збіжності під час навчання.

4) Leaky ReLU: $f(x) = \begin{cases} x, & \text{якщо } x > 0 \\ \alpha x \end{cases}$

Ліквідаційна версія ReLU, де додається параметр нещільності, який навчається разом з іншими параметрами нейронної мережі.

5) Softmax: $softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$.

Використовується у вихідному шарі для розподілу ймовірностей за різними класами у задачах класифікації. Приводить до суми ймовірностей, рівної 1.

Ці функції активації допомагають моделі нейронної мережі вивчати та адаптуватися до нелінійних залежностей у даних, що є важливим для багатьох завдань машинного навчання. Вибір конкретної функції активації може впливати на продуктивність нейронної мережі в розв'язанні конкретних завдань.

2.1.4 Процес навчання нейронної мережі

Процес навчання нейронної мережі можна розділити на кілька етапів, які описують, як мережа вивчає внутрішні представлення та оптимізує ваги зв’язків для вирішення конкретного завдання машинного навчання [8]. Основні етапи процесу навчання включають:

- Пряме поширення (прямий перехід).
- Розрахунок функції втрат.
- Зворотне поширення (зворотне розповсюдження/зворотне поширення).

Цей ітераційний процес працює над вдосконаленням внутрішніх представлень та оновленням ваг зв’язків так, щоб мережа краще вирішувала поставлені завдання машинного навчання. Повторення цих етапів протягом кількох епох дозволяє мережі адаптуватися до вхідних даних і вдосконалювати свої прогнози чи класифікації.

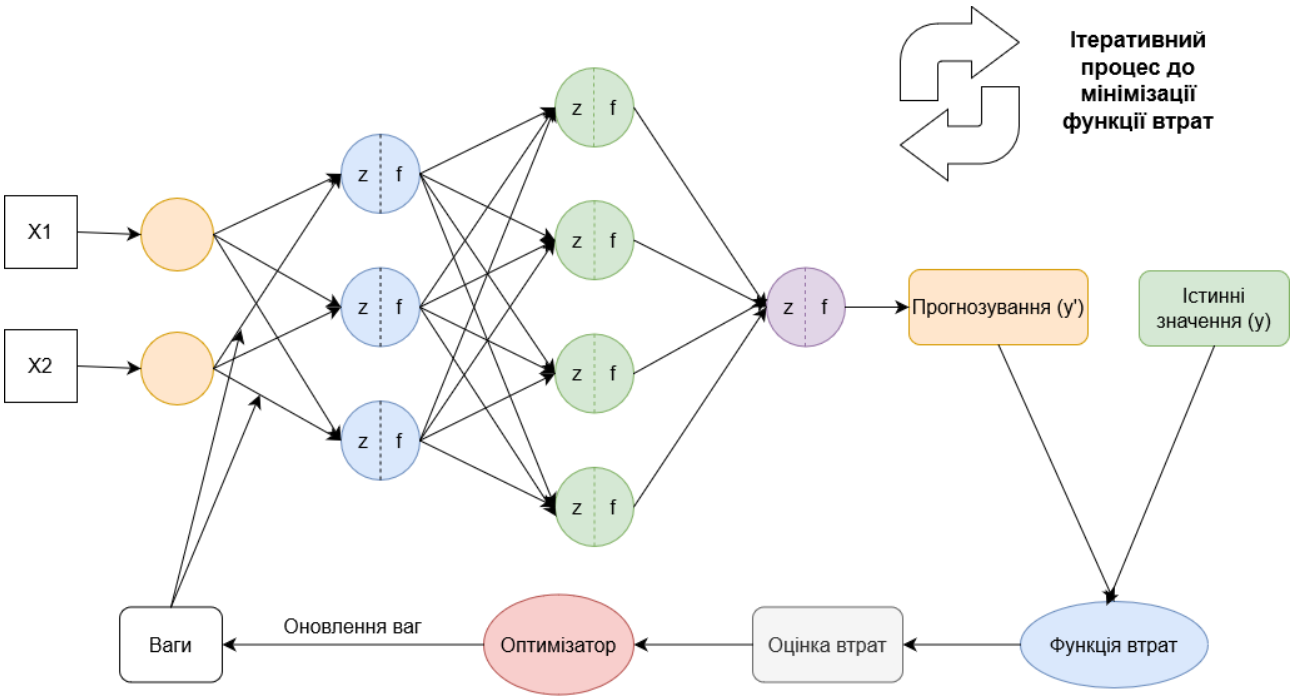

Рис. 2.2. – Процес навчання нейронної мережі

Пряме поширення в нейронних мережах є процесом передачі інформації від вхідних даних до виходу мережі через всі шари. Цей процес включає кроки, які описані нижче.

На початку вхідні дані подаються у вхідний шар мережі, де кожен вхідний вузол (нейрон) приймає значення одного вхідного параметра. Далі, кожен вихідний вузол у кожному прихованому та вихідному шарі обчислює зважену суму вхідних даних, де ваги визначають важливість кожного входу для відповідного нейрона. Ця сума вагованих вхідних значень є лінійною комбінацією вхідних даних для кожного нейрона.

Отримані лінійні комбінації подаються через функцію активації, яка надає нелінійність та визначає вихід кожного нейрона. Функція активації може бути різною, наприклад, ReLU, сигмоїда, тангенс гіперболічний, залежно від того, яка підходить для конкретної задачі мережі.

Загальний результат прямого поширення - вихід мережі, який представляє собою фінальні прогнози, згенеровані мережею на основі вхідних даних та її

внутрішніх параметрів (ваги та зміщення). Цей процес відбувається одноразово для кожного вхідного прикладу під час навчання чи використання мережі для прогнозування.

Розрахунок функції втрат є етапом процесу навчання нейронної мережі, де вимірюється різниця між прогнозованими значеннями та фактичними даними. Основні етапи цього процесу можна описати так:

1) Прогнозовані значення, одержані в результаті прямого поширення, порівнюють з фактичними (істинними) значеннями. Це порівняння здійснюється для кожного вхідного прикладу в мережі.
2) Різниця між прогнозованими та фактичними значеннями вимірюється за допомогою функції втрат. Вибір конкретної функції втрат залежить від типу задачі, що розв’язує мережа. Наприклад, для задач регресії часто використовують середньоквадратичну помилку (MSE), а для задач класифікації - кросс-ентропію.

Мета навчання – мінімізувати значення функції втрат для покращення точності та якості прогнозів мережі.

Зворотне поширення – це ключовий процес у навчанні нейронних мереж, який використовується для оновлення ваг та зміщень мережі з метою мінімізації функції втрат.

Після того, як було обчислено значення функції втрат для прогнозованих та фактичних даних, використовуючи пряме поширення, починається обчислення похідних (градієнтів) цієї функції втрати відносно всіх ваг та зміщень мережі. Це виконується за допомогою правила ланцюга (chain rule) диференціювання функції втрати по параметрах мережі.

Отримані градієнти (похідні) починають передаватися вздовж мережі від виходу до входу. Це означає, що градієнти обчислюються спочатку для виходу мережі і поширюються назад через всі шари мережі. Під час проходження градієнтів від виходу до входу, вони коригують ваги та зміщення в кожному шарі, забезпечуючи оптимізацію цих параметрів таким чином, щоб мінімізувати значення функції втрати.

Зворотне поширення дозволяє мережі використовувати інформацію про похідні функції втрати для коригування внутрішніх параметрів (ваг та зміщень) таким чином, щоб покращити її прогнозуючі здібності. Цей процес ітераційно повторюється під час навчання мережі, допоки значення функції втрати не стає задовільним або не досягає певної точки.

2.2 Типи нейронних мереж

Існує багато типів нейронних мереж, кожен з яких використовується для різних задач та має свої особливості. Кілька основних типів [6]:

1. *Персептрон:*
    - Проста форма нейронної мережі з одним або декількома шарами.
    - Використовується для бінарної класифікації.
2. *Згорткові нейронні мережі (CNN):*
    - Ефективно обробляють зображення та відео завдяки використанню згорткових шарів для виділення ознак та підсумовування їх у відповідних шарах.
    - Широко використовується у комп’ютерному зорі, розпізнаванні образів та обробці зображень.
3. *Рекурентні нейронні мережі (RNN):*
    - Мережі, що мають зв’язки між нейронами, що дозволяють враховувати послідовний контекст даних.
    - Використовується для обробки послідовних даних, таких як текст, мова, часові ряди.
4. *Довго-короткострокова пам'ять (LSTM):*
    - Спеціалізована форма RNN з механізмами для керування та зберігання інформації відносно довгих залежностей у послідовних даних.
5. *Мережі з генеративними моделюваннями (GAN):*
    - Складаються з генератора та дискримінатора, що співпрацюють для генерації реалістичних даних та їх відмінності від реальних.

6. *Автокодери:*
    - Мережі, які використовують для зменшення розмірності даних, зберігаючи при цьому їх основні ознаки.
7. *Мережі на основі уваги (Transformer):*
    - Ефективно обробляють послідовні дані, роблять акцент на важливих елементах послідовності.

Ці типи нейронних мереж використовують для різних завдань, починаючи від класифікації та прогнозування до обробки зображень та роботи зі складними послідовними даними.

2.3 Згортокові нейронні мережі (CNN)

Згорткові нейронні мережі CNN – це потужний тип нейронної мережі, що широко застосовують для розпізнавання зображень. Він складається з послідовності згорткових та об’єднуючих шарів, які витягують відповідні функції з вхідного зображення. Потім слідують один або декілька пов’язаних шарів, які використовують ці функції для прогнозування. Щоб застосувати CNN для розпізнавання зображень, спочатку його навчають на великому наборі даних з позначеними зображеннями об’єктів. Під час навчання CNN вивчає пов’язувати витягнуті характеристики з правильними мітками через процес зворотного поширення та оптимізації [9]. Після навчання CNN можна використовувати для прогнозування нових, невидимих зображень, передаючи їх через мережу та вибираючи мітку з найвищою прогнозованою ймовірністю.

На рисунку 2.3 представлено спрощену схему згорткової нейронної мережі (CNN), призначеної для класифікації зображень [10]. Вхід до мережі – це зображення автомобіля, яке вводиться через вхідний шар. Потім зображення проходить через перший прихований шар, де застосовуються набори фільтрів. Кожен фільтр вирізняє конкретні особливості зображення, такі як краї чи текстури. Результати першого прихованого шару передаються до наступного, де також використовуються додаткові фільтри для виділення вищого рівня ознак. Цей

процес повторюється через кілька прихованих шарів до останнього, який створює набір функцій і передає їх на вихідний рівень. Вихідний рівень формує розподіл ймовірностей для трьох класів: автомобіль, автобус і літак. Мережа приймає рішення на основі класу з найвищою ймовірністю. Ваги фільтрів у кожному шарі налаштовуються за допомогою процесу, відомого як зворотне поширення, який коригує ваги для мінімізації різниці між прогнозованим та фактичним виходом.

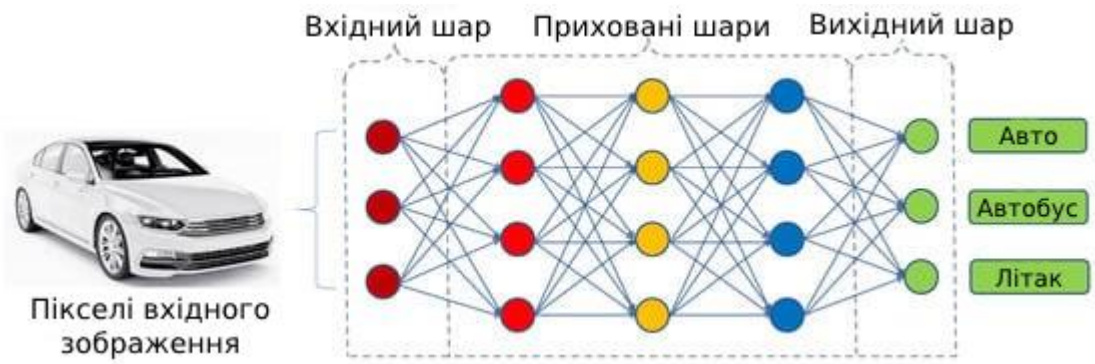

Рис. 2.3 – Спрощена ілюстрація мережі CNN [11]

2.3.1 Архітектура CNN

CNN, як правило, має три рівні: згортковий рівень, рівень об’єднання та повністю зв’язаний рівень.

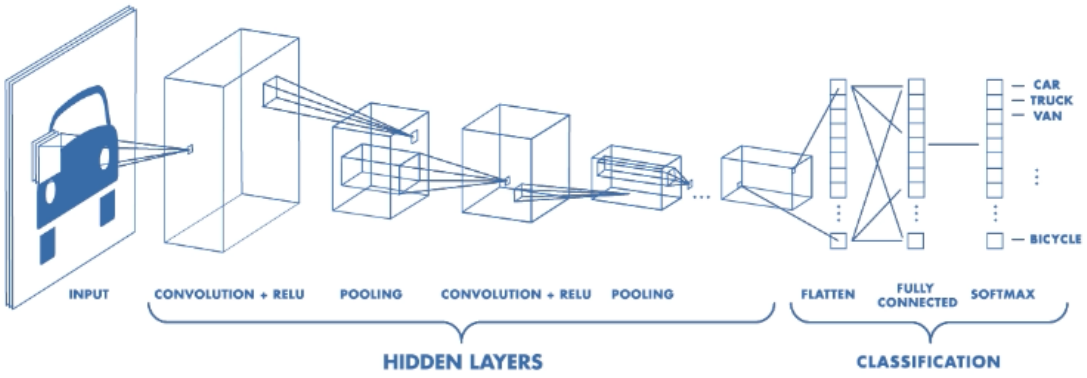

Рис. 2.4 – Архітектура CNN [12]

Згортковий рівень є ключовим елементом у структурі CNN і відповідає за значну частину обчислювального завдання мережі [13].

Цей рівень реалізує скалярний добуток між двома матрицями, де одна матриця є набором параметрів, який можна навчити, відомий як ядро, а інша матриця представляє обмежену частину сприймаючого поля. Розміри ядра просторово менше, ніж зображення, але його глибина охоплює всі три кольорові канали (RGB), якщо вони присутні. Це означає, що висота і ширина ядра є просторово обмеженими, але воно проникає у глибину всіх каналів зображення [10].

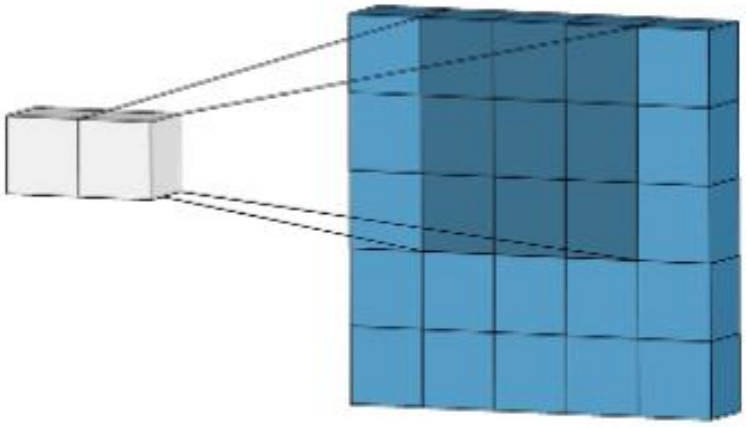

Рис. 2.5 – Ілюстрація операції згортання

Під час прямого проходу ядро ковзає по висоті та ширині зображення, створюючи представлення зображення цієї сприймаючої області. Це створює двовимірне представлення зображення, відоме як карта активації, яка дає відповідь ядра на кожну просторову позицію зображення. Розмір ковзання ядра називають кроком.

Якщо ми маємо вхід розміром $W \times W \times D$ і D_{out} кількістю ядер із просторовим розміром F із кроком S і розміром заповнення P, тоді розмір

вихідного об’єму можна визначити за такою формулою (формула зроткового шару):

$$W_{out} = \frac{W-F+2P}{S} + 1. \quad (2.1)$$

Це дасть вихідний обсяг розміром $W_{out} \times W_{out} \times D_{out}$ [13].

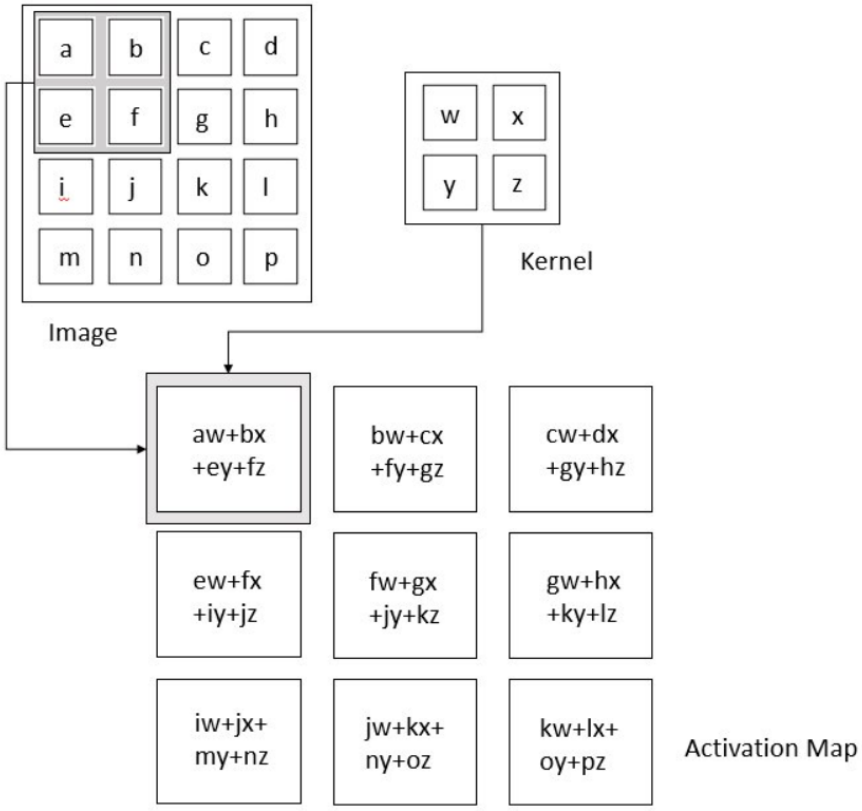

Рис. 2.6 – Операція згортання

Рівень об’єднання заміщує вихідні дані мережі в певних областях, обчислюючи загальну статистику для найближчих виходів [14]. Це сприяє зменшенню просторового розміру представлення, що призводить до зменшення обчислювальних витрат і кількості параметрів ваги. Операція об’єднання виконується над кожним фрагментом представлення окремо.

Існує кілька видів функцій об’єднання, таких як середнє значення прямокутної області, норма L2 прямокутної області та зважене середнє на основі

відстані від центрального пікселя. Проте найбільш популярним є максимальне об’єднання, яке вибирає максимальний результат з сусідніх областей [15].

Рис.2.7 – Операція об’єднання [12]

Якщо у нас є карта активації розміром $W \times W \times D$, об’єднуюче ядро просторового розміру F і крок S, тоді розмір вихідного об’єму можна визначити за такою формулою:

$$W_{out} = \frac{W-F}{S} + 1. \quad (2.2)$$

Це дасть вихідний обсяг розміром $W_{out} \times W_{out} \times D$.

У всіх випадках об’єднання забезпечує певну інваріантність положень, що означає, що об’єкт буде розпізнаним незалежно від того, де він з’являється на кадрі.

Повністю підключений рівень (FC) представляє собою шар нейронів, в якому кожен нейрон з’єднаний з усіма нейронами попереднього та наступного шарів, що відрізняє його від звичайного FCNN (Fully Connected Neural Network). Це означає, що кожен вхідний сигнал з попереднього шару переходить до кожного нейрона у цьому шарі, і, відповідно, кожен нейрон цього шару з’єднаний з кожним нейроном наступного шару. Цей зв’язок можна обчислити за допомогою множення матриць і додавання зміщення [16].

Рівень FC допомагає вирівнювати та виражати складні зв’язки між вхідними

та вихідними сигналами. Однак важливо враховувати, що за наявності великої кількості нейронів у FC-шарі може збільшитися кількість параметрів мережі, що може призвести до перенавчання та великої обчислювальної складності.

Поза основними рівнями, у спеціалізованих типах CNN також використовуються додаткові рівні для вирішення конкретних завдань. Наприклад, у рекурентних CNN (RCNN) може бути доданий рекурентний рівень для захоплення часових залежностей у вхідній послідовності. Для розв'язання проблеми довготривалих залежностей у вхідній послідовності у довгострокових короткочасних пам'ятях (LSTM) CNN може використовувати рівень LSTM.

2.3.2 Оптимізація градієнта у CNN

Оптимізація градієнта у згорткових нейронних мережах є ключовим етапом у навчанні моделей машинного навчання. Цей процес спрямований на покращення ефективності та точності моделі шляхом виправлення її параметрів. Оптимізація градієнта важлива для досягнення високої продуктивності та здатності моделі вирішувати завдання з високою точністю [17].

Основні цілі оптимізації градієнта в CNN:

1. Мінімізація функції втрат: зменшення різниці між прогнозованими та фактичними значеннями моделі. Мінімізація функції втрат є ключовим завданням для забезпечення точності та навчання моделі.
2. Швидше навчання (збіжність): забезпечення ефективного та швидкого навчання моделі. Оптимізація градієнта спрямована на прискорення процесу збіжності до оптимальних значень параметрів.
3. Стійкість до збурень: зменшення впливу шумів та аномалій в даних, що може виникнути через непередбачувані зміни в градієнтах. Модель повинна бути стійкою до випадкових аномалій.
4. Збільшення точності: підвищення точності моделі на відомих та нових даних. Модель повинна бути загальним інструментом для вирішення завдань.
5. Уникнення перенавчання: забезпечення того, що модель буде ефективною не

лише на тренувальних, але й на тестових та нових даних. Перенавчання повинно бути мінімізовано.

Процес навчання використовує методи оптимізації градієнта на кожному кроці ітеративного оновлення параметрів моделі. Ці методи впроваджуються на кожній ітерації, коли модель адаптує свої параметри для мінімізації функції втрат. Під час кожного проходження через тренувальні дані градієнти обчислюються, і параметри оновлюються в напрямку зменшення втрат.

Ці методи оптимізації варіюються від класичних, таких як градієнтний спуск, до більш складних, таких як Adam. Вибір конкретного методу може залежати від завдань моделі, розміру даних, архітектури мережі та інших факторів. Ретельний вибір методу оптимізації може визначити швидкість та точність навчання моделі.

3. АЛГОРИТМИ ОПТИМІЗАЦІЇ ГРАДІЄНТА

Глибокі нейронні мережі, зокрема згорткові нейронні мережі (CNN), відіграють ключову роль у багатьох галузях, таких як класифікація зображень, аналіз мікровиразів, розпізнавання облич та виявлення об’єктів. Хоча базові CNN є контрольованими алгоритмами навчання, їхні високі здатності до виділення особливостей також породжують виклики. Це призвело до розвитку вдосконалених моделей, які комбінують CNN з алгоритмами неконтрольованого навчання, такими як CSFL.

У процесі навчання CNN визначає мінімальне значення функції втрат за допомогою градієнтної оптимізації спуску (GD). Існують вдосконалені алгоритми, які поділяються на дві групи. Перша включає метод імпульсу та прискорення Нестерова для усунення шуму та стабілізації кривої конвергенції [18]. Друга група використовує адаптивні методи, такі як AdaGrad та RMSprop, що коригують крок наступної ітерації з урахуванням кумулятивних змін параметрів. Алгоритми, як AdaDelta та Adam, є комбінацією ідей і визнані за свою ефективність.

Adam, розроблений Дідеріком П. Кінгмою, об'єднав переваги різних методів [21]. Проте, існуючі алгоритми недостатньо приділяють увагу поточним і останнім градієнтам. Відсутність уваги до цього може призвести до ігнорування важливості градієнтів у процесі оптимізації.

У цій роботі розглянуті різні методи градієнтної оптимізації такі як SGD, SGD з моментом, Adam, RMSprot, Adagrad, Adadelta.

3.1 SGD

Стохастичний градієнтний спуск (SGD) представляє собою стохастичне наближення до оптимізації градієнтного спуску та є ітераційним методом для мінімізації чи максимізації цільової функції. Основна ідея полягає в тому, щоб виявляти мінімуми чи максимуми, використовуючи ітераційний підхід та апдейт

градієнта цільової функції після перевірки лише одного чи кількох навчальних прикладів.

SGD має переваги над методами батч-градієнтного спуску, оскільки він обчислює негативний градієнт цільової функції лише за допомогою обраної випадкової підмножини навчальних прикладів. Він є ефективнішим для великої кількості даних, оскільки не потребує враховування всіх прикладів на кожному кроці.

Однією з переваг SGD є його можливість динамічно коригувати оцінку матриць першого та другого порядку градієнта для кожного параметра згідно з функцією витрат. Це дозволяє зменшити ризик потрапляння моделі до локального оптимуму. Враховуючи ці переваги, можна стверджувати, що використання SGD допомагає зменшити витрати на обчислення та призводить до швидкої збіжності.

Налаштування набору даних представлено як $S.\ x_i \in R^n$, де n-вимірний вектор.$y_i \in \{1, m-1\}$ – категорія i-ї навчальної вибірки. Тоді SGD можна деталізувати так.

Спочатку призначається нульовий вектор значенню ваги W_1, а потім випадковим чином обирається навчальна вибірка (x_{i_t}, y_{i_t}) із усього навчального набору, де $i_t \in \{1, \dots, m\}$ є індексом обраної навчальної вибірки на t-й ітерації. Цільова функція є

$$\min(W) = \frac{\lambda}{2}\|W\|^2 + f\left(W, \left(x_{i_t}, y_{i_t}\right)\right). \tag{3.1}$$

Потім обчислюється градієнт за формулою (3.2), яку можна виразити через

$$\nabla_t = \lambda W_t - \alpha_t y_{i_t} x_{i_t}, \tag{3.2}$$

де $\alpha_t = \begin{cases} 1, & y_{i_t}\langle W_t, x_{i_t}\rangle < 1 \\ 0\,. \end{cases}$

Оновлена формула матриці W виглядає так:

$$W_{t+1} = W_t - \eta_t \alpha_t, \tag{3.3}$$

де $\eta_t = \frac{1}{\lambda t}$.

Тоді оновлену вагову матрицю W, враховуючи формули (3.2) і (3.3), можна отримати за допомогою

$$W_{t+1} = \left(1 - \frac{1}{t}\right) W_t + y_{i_t} x_{i_t}. \tag{3.4}$$

На практиці формула (3.4) використовується для знаходження мінімумів або максимумів шляхом ітерації.

3.2 SGD з моментом

Стохастичний градієнтний спуск (SGD) з моментом – це вдосконалена версія базового SGD, яка використовує концепцію «моменту» або інерції для прискорення оптимізаційного процесу та стабілізації швидкості збіжності.

Ключова ідея полягає в тому, що SGD з моментом використовує експоненціально зважений ковзний середній (EWMA) градієнту, щоб зберігати інформацію про попередні градієнти та їх напрямки. Це допомагає уникнути значних коливань у випадкових градієнтах та дозволяє більш стабільно оновлювати параметри моделі.

Для того щоб зрозуміти, як працює SGD з моментом, необхідно зрозуміти концепцію експоненціально зваженого ковзного середнього (EWMA). Це була техніка, за допомогою якої намагалися знайти тенденцію в даних часових рядів. Формула EWMA така:

$$V_t = \beta V_{t-1} + (1 - \beta)\theta_t. \tag{3.5}$$

У формулі (3.5) β є ваговий коефіцієнт, яка присвоюється минулим значенням градієнта. Значення β: $0 < \beta < 1$. Якщо значення бета дорівнює 0,5, це означає, що $1/1-0{,}5 = 2$, тобто це свідчить про те, що розраховане середнє значення було отримане з попередніх 2 вимірювань.

Величина V_t залежить від β. Чим вище значення β, тим більше ми намагаємося отримати середнє значення більшої кількості попередніх даних і навпаки.

Тепер у SGD з моментом використовуємо ту саму концепцію EWMA. У цьому випадку вводиться термін «швидкість» (V), який відстежує зміни градієнта з метою досягнення глобальних мінімумів. Зміна ваг позначається за допомогою формули:

$$W_{t+1} = W_t - V_t, \tag{3.6}$$

де

$$V_t = \beta V_{t-1} + \eta \Delta W_t. \tag{3.7}$$

У формулі 3.7 частина β відповідає за обчислення імпульсу та є корисною для визначення попередньої швидкості (V_{t-1}). Імпульс враховує попередні градієнти та їхні напрямки, дозволяючи зберігати інформацію про тенденції для досягнення стійкої та прискореної збіжності в оптимізаційному процесі. Також сприяє прискоренню, оскільки враховує попередні зміни градієнта та забезпечує більш плавні оновлення ваг [18].

3.3 RMSprop

Метод RMSprop (Root Mean Square Propagation) є адаптивним алгоритмом оптимізації, що регулює швидкість навчання для кожного параметра

індивідуально, використовуючи інформацію про середній квадрат градієнта.

На кожному кроці алгоритм використовує експоненційно згладжене середнє квадрату градієнтів для кожного параметра. Позначимо градієнт для параметра θ на кроці часу t як g_t, тоді середній квадрат градієнта обчислюється за формулою:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) g_t^2, \quad (3.8)$$

де β – це параметр експоненційного згладжування, який знаходиться в інтервалі (0, 1). Таким чином, алгоритм обчислює ваговий середній квадрат градієнта, де β визначає вагу оновлення попереднього середнього значення.

Оновлення параметра θ на кожному кроці здійснюється згідно з формулою:

$$\theta_{t+1} = \theta - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t. \quad (3.9)$$

Отже, RMSprop адаптивно регулює швидкість навчання для кожного параметра, що дозволяє ефективніше оптимізувати функцію витрат, особливо в умовах нерівномірної швидкості збіжності по різних напрямках.

Переваги RMSprop:

1. *Адаптивність швидкості навчання:* RMSprop адаптує швидкість навчання для кожного параметра окремо. Це дозволяє ефективно працювати з різноманітними параметрами, особливо тоді, коли вони мають різні шкали.
2. *Зменшення впливу великих градієнтів:* Алгоритм використовує експоненціально зважене середнє для квадратів градієнтів, що дозволяє автоматично зменшувати вплив великих градієнтів на швидкість навчання.
3. *Врахування історії градієнтів:* RMSprop використовує історію квадратів градієнтів, що дозволяє більш ефективно адаптувати швидкість навчання до змін у функції втрат.
4. *Зменшення необхідності в ручному налаштуванні гіперпараметрів:*

Алгоритм включає параметр γ (зазвичай близький до 0,9), який дозволяє автоматично регулювати вагу історії градієнтів.

5. *Ефективність на практиці:* RMSprop добре працює для різних завдань машинного навчання і глибокого навчання та часто дає гарні результати на практиці.

Загалом, RMSprop є ефективним алгоритмом оптимізації, який дозволяє автоматично регулювати швидкість навчання, зменшуючи його для параметрів з великими градієнтами і збільшуючи для параметрів з малими градієнтами [20].

3.4 Adagrad

Adagrad – це алгоритм оптимізації на основі градієнта, який адаптує швидкість навчання для кожного параметра, враховуючи історію градієнтів для кожного параметра. Цей метод дозволяє виконувати менші оновлення (з низькою швидкістю навчання) для параметрів, які часто зустрічаються, і більші оновлення (з високою швидкістю навчання) для параметрів, які рідко зустрічаються. Такий підхід робить Adagrad ефективним для роботи з розрідженими даними, де деякі параметри можуть мати значення для лише обмеженої кількості навчальних прикладів.

Оновлення Adagrad для кожного параметра θ_i на кроці часу *t* виглядає так:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} g_{t,i}, \tag{3.10}$$

де $g_{t,i}$ – градієнт цільової функції відносно параметра θ_i на кроці часу *t*; G_t - діагональна матриця, де кожен діагональний елемент $G_{t,ii}$ є сумою квадратів градієнтів відносно θ_i на кроці часу *t*; ϵ - член згладжування, що уникає ділення на нуль (зазвичай порядку $1e^{-8}$).

Однією з головних переваг Adagrad є те, що він усуває необхідність ручного налаштування швидкості навчання. Більшість реалізацій використовують значення за замовчуванням 0,01 і залишають його без змін.

Основною «слабкістю» Adagrad є накопичення квадратів градієнтів у знаменнику: оскільки кожен доданий член додатний, накопичена сума продовжує зростати під час навчання. Це, у свою чергу, призводить до того, що швидкість навчання зменшується і зрештою стає нескінченно малою, після чого алгоритм більше не може отримувати нову інформацію. Наступні алгоритми спрямовані на усунення цієї вади.

3.5 Adadelta

Adadelta є розширенням Adagrad, призначений для зменшення агресивного монотонного зниження швидкості навчання. На відміну від Adagrad, який накопичує всі попередні квадратичні градієнти, Adadelta обмежує вікно накопичених попередніх градієнтів до фіксованого розміру w.

Замість неефективного зберігання w попередніх квадратів градієнтів, сума градієнтів рекурсивно визначається як експоненційно згладжене середнє всіх попередніх квадратів градієнтів. Поточне середнє $E[g^2]$ на кроці часу t тоді залежить (за допомогою параметра γ, схожого на імпульс) лише від попереднього середнього та поточного градієнта:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2. \tag{3.11}$$

Значення параметра γ встановлюється таким самим, як і момент імпульсу, зазвичай приблизно 0,9.

Для ясності, оновлення SGD можна переписати у термінах вектора оновлення параметрів:

$$\Delta\theta_t = -\eta g_{t,t}; \theta_{t+1} = \theta_t + \Delta\theta_t. \quad (3.12)$$

Отже, вектор оновлення параметрів Adagrad, який ми отримали раніше, набуває вигляду:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t+\epsilon}} g_{t.} \quad (3.13)$$

Тепер ми просто замінимо діагональну матрицю G_t із спадаючим середнім квадратів минулих градієнтів $E[g^2]_t$:

$$\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t+\epsilon}} g_t. \quad (3.14)$$

Оскільки знаменник є лише критерієм середньоквадратичної (RMS) помилки градієнта, ми можемо замінити його скороченим критерієм:

$$\theta_t = -\frac{\eta}{RMS[g]_t} g_t. \quad (3.15)$$

Одиниці в цьому оновленні (як і в SGD або Adagrad) не збігаються, тобто оновлення повинно мати ті самі гіпотетичні одиниці, що і параметр. Щоб усвідомити це, спочатку вони визначають ще одне експоненційно зглажене середнє, але цього разу не для квадратів градієнтів, а для квадратів оновлень параметрів:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2. \quad (3.16)$$

Таким чином, маємо таку середньоквадратичну помилки оновлень параметрів:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}. \quad (3.17)$$

Оскільки $RMS[\Delta\theta]_t$ невідомий, ми наближаємо його до RMS оновлень параметрів до попереднього кроку часу. Заміна швидкості навчання η у попередньому правилі оновлення на $RMS[\Delta\theta]_{t-1}$ нарешті дає правило оновлення Adadelta:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t, \quad (3.18)$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t. \quad (3.19)$$

З Adadelta нам навіть не потрібно встановлювати швидкість навчання за замовчуванням, оскільки її було вилучено з правила оновлення [21].

Переваги Adadelta:

1. *Адаптивність швидкості навчання*: Adadelta автоматично адаптує швидкість навчання для кожного параметра окремо. Це дозволяє ефективно працювати з різноманітними параметрами, особливо тоді, коли вони мають різні шкали.
2. *Зменшення впливу великих градієнтів:* Як і в Adagrad, в Adadelta використовується історія квадратів градієнтів для адаптивного зменшення впливу великих градієнтів на швидкість навчання.
3. *Відсутність необхідності в глобальному гіперпараметрі:* Adadelta не вимагає налаштування глобального гіперпараметра, такого як швидкість навчання (learning rate). Він самостійно регулюється на основі історії градієнтів.
4. *Відсутність необхідності в обчисленні оберненої матриці гауссіана*: На відміну від методів, які використовують обернену матрицю гауссіана, оскільки він уникає потреби у цьому обчисленні, це призводить до підвищення швидкості розрахунків.

5. *Ефективність на практиці:* Adadelta добре працює на практиці для різних завдань машинного навчання і глибокого навчання, часто демонструючи гарні результати.

Загалом, Adadelta є алгоритмом оптимізації, який дозволяє автоматично регулювати швидкість навчання, а також враховувати історію градієнтів для кожного параметра.

3.6 Adam

Adam (Adaptive Moment Estimation) – це алгоритм оптимізації, який обчислює адаптивні швидкості навчання для кожного параметра. Він поєднує в собі ідеї імпульсу та адаптивної швидкості навчання з таких методів, як Adadelta та RMSprop. Ключові компоненти Адама включають експоненціально спадаючі середні значення попередніх градієнтів (m_t) і градієнти попереднього квадрата (v_t), подібно до імпульсу та RMSprop. Метод обчислює ці середні значення так:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{3.20}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{3.21}$$

де g_t – градієнт у часі t; β_1 і β_2 – темпи розпаду; m_t і v_t – оцінки першого моменту (середнє) і другого моменту (нецентрована дисперсія) градієнтів відповідно.

Щоб протидіяти зміщенням у бік нуля, особливо на початкових етапах часу, коли m_t і v_t ініціалізуються як вектори нулів, обчислюються оцінки першого та другого моментів з поправкою на зсув:

$$\widehat{m}_t = \frac{m_t}{1-\beta_2^t}, \tag{3.22}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{3.23}$$

Потім параметри оновлюються з використанням оцінок, виправлених на зсув, у правилі оновлення Адама [22]:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t. \quad (3.24)$$

Алгоритм Adam часто виявляється ефективним на практиці та має декілька переваг, які вигідно вирізняють його серед інших алгоритмів адаптивного навчання:

1. *Ефективність*: Adam часто працює добре в різноманітних завданнях машинного навчання і глибокого навчання, демонструючи ефективність на практиці.
2. *Адаптивність до різних параметрів*: Алгоритм адаптується до різних швидкостей навчання для кожного параметра, що дозволяє ефективно працювати з різноманітними функціями втрат.
3. *Зменшення впливу шуму в градієнтах*: Використання експоненціально зважених середніх дозволяє зменшити вплив шуму в градієнтах на оновлення параметрів.

Відсутність необхідності в ручному налаштуванні гіперпараметрів: Adam включає параметри за замовчуванням, які часто виявляються дієвими без додаткового налаштування.

4. МОДЕЛЬ СИСТЕМИ АНАЛІЗУ ЕМОЙЦІЙНОГО СТАНУ ЛЮДИНИ ЧЕРЕЗ РОЗПІЗНАВАННЯ ОБЛИЧ

4.1 Опис набору даних

Для розробки та навчання моделі системи аналізу емоційного стану використані набори даних, що містять зображення облич людей з різними виразами обличчя та емоційними станами, такими як: радість, смуток, здивованість, гнів, відраза, страх. Датасет складається з 2956 зображень. Кожне зображення анотоване з вказівкою емоційного стану, що надає можливість системі навчатися належним чином класифікувати та розпізнавати різні емоційні вирази.

Додатково, для оптимізації та тестування моделі використаний набір зображень облич, які не входять до тренувальної вибірки. Цей набір включає в себе реальні умови використання системи, що дозволило визначити рівень точності та ефективності розробленої моделі в реальних умовах.

4.2 Побудова моделі

Задача моєї роботи полягає в розробці моделі системи для аналізу емоційного стану людини через розпізнавання облич за допомогою алгоритмів оптимізації градієнта. Передбачається створення системи, яка визначатиме емоційний стан особи на основі зображень обличчя, використовуючи нейронні мережі та алгоритми оптимізації градієнта.

Алгоритм роботи моделі представлений на рис 4.1.

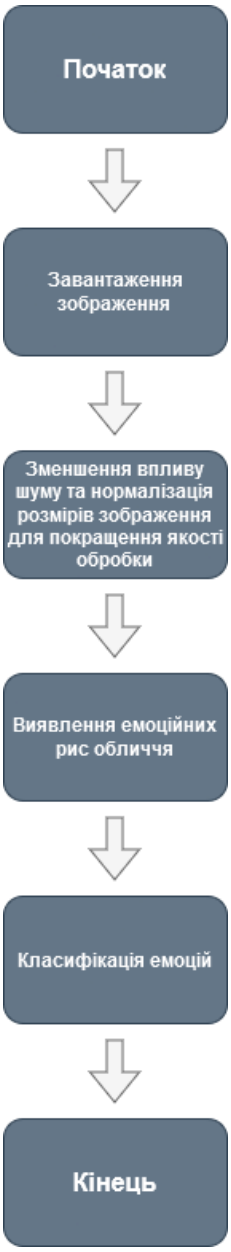

Рис. 4.1 – Алгоритм роботи моделі

Для початку порівняємо точність та ефективність згорткової нейронної мережі, яка використовувала різні алгоритми оптимізації градієнта. Мета цього експерименту полягає в оцінці впливу вибору алгоритму оптимізації на якість класифікації емоцій та швидкість обробки зображень облич у контексті аналізу емоційного стану.

Для цього модель навчалася на тренувальному наборі даних, використовуючи алгоритми оптимізації градієнта, такі як: SGD, SGD з моментом,

RMSprop та Adam. Після завершення навчання проводилося тестування на тестовому наборі для оцінки точності класифікації.

Порівняння точності знаходжень помилки при використанні описаних методів представлено в таблиці 4.1.

Таблиця 4.1

Порівняння ефективності алгоритмів оптимізації градієнта

Алгоритм	**Точність, %**	**Похибка**
SGD	~28	~1,89
SGD з моментом	~24	~2,36
RMSprop	~28	~2,35
Adam	~34	~1,68

Як видно з наведених даних (табл. 4.1), найкращим за показниками відсотка правильних виявлень і мінімальним показником похибки є алгоритм Adam. Тому для створення моделі системи аналізу емоційного стану людини через розпізнавання облич будемо використовувати згорткову нейронну мережу з використанням Adam.

З метою збільшення точності моделі було збільшено обсяг даних та кількість повних проходів через весь тренувальний набір даних.

Результати точності та помилки наведені в таблиці 4.2.

Таблиця 4.2

Ефективність моделі

Алгоритм	**Точність, %**	**Похибка**
Adam	~86	~0,51

Модель має точність розпізнавання 86 %, що є дуже непоганим результатом.

На рис. 4.2 представлений графік точності на тренувальному на валідаційному наборах, які були правильно класифіковані моделлю під час тренування.

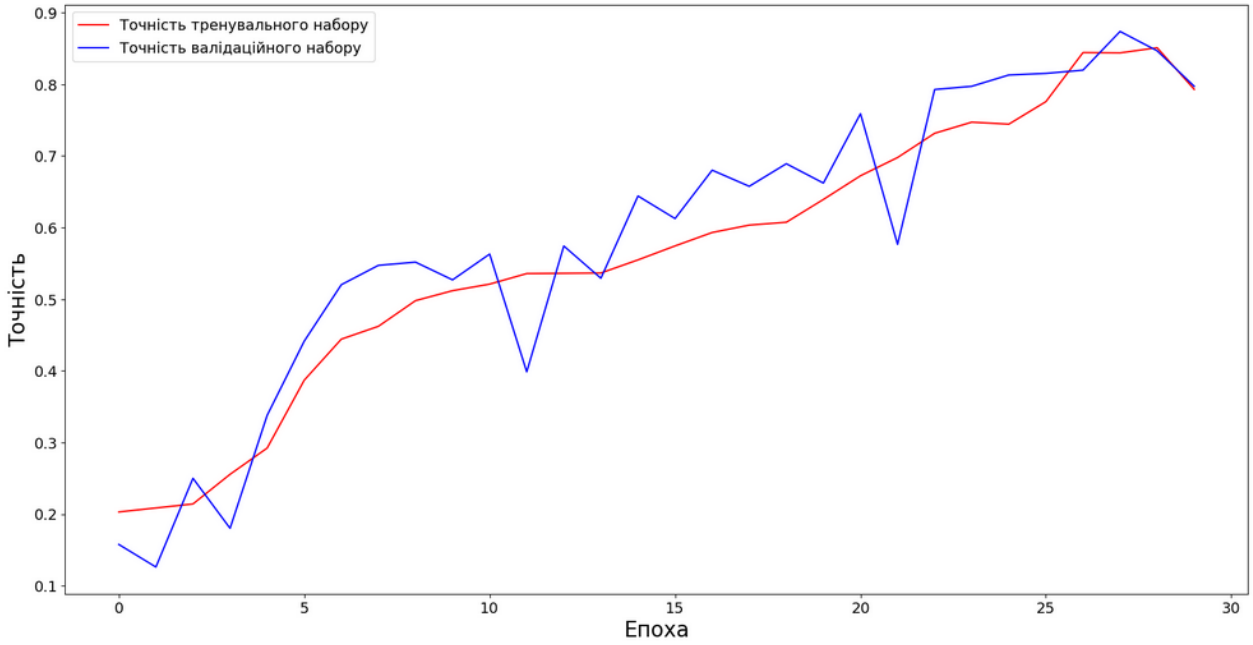

Рис. 4.2 – Графік точності на тренувальному та валідаційному наборах під час тренування

Далі візуалізуємо матрицю плутанини, яка використовується для оцінки ефективності класифікаційної моделі на наборі даних, для якого відомі справжні класи. Кожен рядок цієї таблиці представляє справжній клас, а кожен стовпчик – передбачений клас моделі. Матриця плутанини наведена на рис. 4.3.

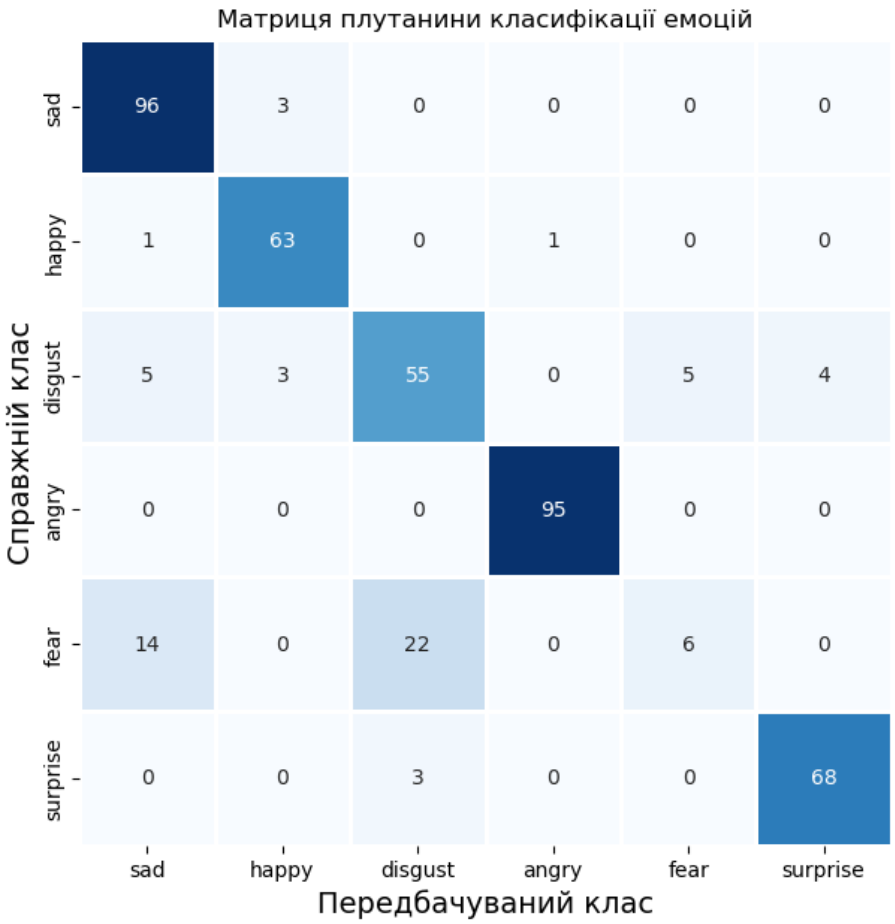

Рис. 4.3 – Матриця плутанини класифікації емоцій

Матриця плутанини дозволяє зрозуміти, які типи помилок припускається модель. У цьому разі модель плутає таку емоцію як страх зі смутком та відразою, оскільки вищезазначені емоції можуть мати подібні вирази обличчя, що ускладнює їхню розпізнаваність. Для вирішення цього питання можна спробувати налаштувати модель, додавши більше тренувальних даних з різноманітними виразами обличчя та виразами емоцій.

На рис. 4.4 наведені результати роботи моделі на валідаційних даних.

Рис. 4.4 – Результати роботи моделі на валідаційних даних

Одержані результати моделі, яка визначає емоції на зображеннях, представлені у вигляді карти емоцій. На зазначеному рисунку (рис. 4.4) відображені прогнозовані емоційні стани для кожного обличчя. Модель здатна ефективно розпізнавати деякі емоції, такі як радість, відразу та злість, але може мати труднощі з інтерпретацією емоцій, що мають схожі вирази.

На рис. 4.5 представлені результати роботи моделі на зображення особистих виразів обличчя та емоцій.

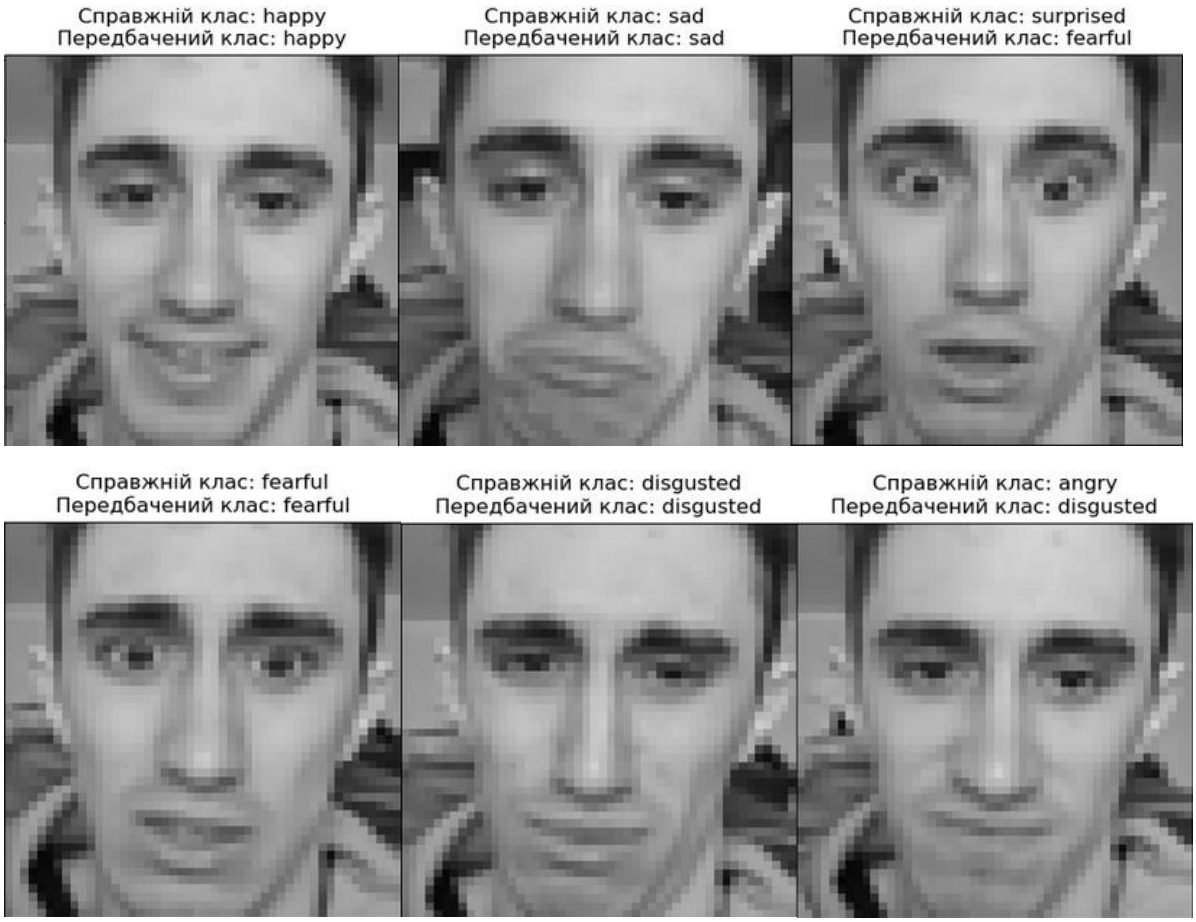

Рис. 4.5 – Результати роботи моделі на зображення особистих виразів обличчя та емоцій

На зазначеному рисунку (рис. 4.5) відображені прогнозовані емоційні стани автора для кожного виразу обличчя та емоції. Аналізуючи одержані результати роботи моделі, можна зробити висновок, що вона демонструє «неточність» у виявленні справжніх емоцій (як психічного процесу) через їх імітацію автором. Це пояснюється тим, що реальна емоція, окрім певних змін міміки (підняття кутків губ, розширення очей, напруження лобу тощо) ще супроводжується певними фізіологічними змінами, такими, наприклад, як розширення або звуження зіниць очей тощо.

Отже, результат, де модель правильно визначає лише частину емоцій при використанні зображень автора та імітації емоцій, може бути обумовлений відмінністю між виразами, що виникають внаслідок природної емоційної реакції, та намаганнями їх зімітувати.

Важливо зазначити, що результати можуть бути покращені через додаткове тренування на різноманітних даних та враховуючи індивідуальні особливості виразів обличчя.

У цілому, незважаючи на виявлені обмеження, отримані результати вказують на потенціал системи у розвитку та застосуванні в областях, де важливо враховувати емоційний стан людини за допомогою аналізу обличчя.

ВИСНОВКИ

1. У проведених експериментах з порівнянням алгоритмів оптимізації градієнта (SGD, SGD з моментом, Adam, RMSprop) на основі моделі згорткової нейронної мережі (CNN) виявлено, що Adam продемонстрував вищу точність на рівні 34 %, порівняно з іншими алгоритмами. Це свідчить про ефективність Adam у зборі та оптимізації параметрів мережі.
2. Додавання обсягу тренувального датасету та повних проходів через весь тренувальний набір даних виявилося ключовим кроком у покращенні точності моделі. Ці дії підвищили точність моделі до 86 %, що значно відрізняється від стартового значення.
3. Подальший аналіз результатів передбачень моделі, побудованої на збільшеному датасеті, включає в себе високий рівень впевненості в правильних класифікаціях та кращі показники на матриці плутанини. Це свідчить про вдалі адаптації моделі до розпізнавання широкого спектру емоцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. E. Fox. Emotion Science: Cognitive and Neuroscientific Approaches to Understanding Human Emotions. Red Globe Press, 2008. - 456 с.
2. C. R. Combei, V. Reggi. Appraisal, Sentiment and Emotion Analysis in Political Discourse. Routledge, 2023. - 180 с.
3. Emotion Analysis [Електронний ресурс]. URL: https://medium.com/analytics-vidhya/emotion-analysis-d7a9d68b17ed
4. E. Kim, R. Klinger. A survey on Sentiment and Emotion Analysis for Computational Literary Studies. ZFDG, 2018. - 38 с.
5. Chen J. What Is a Neural Network?. Investopedia. URL: https://www.investopedia.com/terms/n/neuralnetwork.asp (date of access: 23.11.2023).
6. What are Neural Network Architectures? | H2O.ai. H2O.ai | The fastest, most accurate AI Cloud Platform. URL: https://h2o.ai/wiki/neural-network-architectures/ (date of access: 23.11.2023).
7. Takyar A. What are neural networks?. LeewayHertz - AI Development Company. URL: https://www.leewayhertz.com/what-are-neural-networks/ (date of access: 23.11.2023).
8. B. M. Wilamowski, "Neural network architectures and learning algorithms," in IEEE Industrial Electronics Magazine, vol. 3, no. 4, pp. 56-63, Dec. 2009, doi: 10.1109/MIE.2009.934790.
9. Shen, L.; Lin, Z.; Huang, Q. Relay backpropagation for effective learning of deep convolutional neural networks. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part VII 14. Springer: Berlin/Heidelberg, Germany, 2016; pp. 467–482.
10. Pandian, J.A.; Kanchanadevi, K.; Kumar, V.D.; Jasińska, E.; Goňo, R.; Leonowicz, Z.; Jasiński, M. A five convolutional layer deep convolutional neural network for plant leaf disease detection. Electronics 2022, 11, 1266.

11. Convolutional Neural Networks: A survey, Moez Krichen published by MDPI, Computers 2023, 12, 151.
12. Facial Emotion Recognition For Autism Spectrum Disorder Using Mobilenetv2, Samiat Bola-Matanmi, Bournemouth University, 2022
13. Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville published by MIT Press, 2016
14. Galanis, N.I.; Vafiadis, P.; Mirzaev, K.G.; Papakostas, G.A. Convolutional Neural Networks: A Roundup and Benchmark of Their Pooling Layer Variants. Algorithms 2022, 15, 391.
15. Zheng, T.; Wang, Q.; Shen, Y.; Lin, X. Gradient rectified parameter unit of the fully connected layer in convolutional neural networks. Knowl.-Based Syst. 2022, 248
16. S. Ruder, "An overview of gradient descent optimization algorithms," 2016, https://arxiv.org/abs/1609.04747.
17. I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in Proceedings of the 30 th International Conference on Machine Learning, pp. 1139–1147, Atlanta, Georgia, USA, June 2013.
18. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, The MIT Press, Cambridge, MA, USA, 2016.
19. D. Kingma and J. Ba, "Adam: a method for stochastic optimization," 2014, https://arxiv.org/abs/1412.6980.
20. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159. Retrieved from http://jmlr.org/papers/v12/duchi11a.html
21. Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved from http://arxiv.org/abs/1212.5701
22. Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13

ДОДАТОК А

Підготовка даних

```
# Вказуємо шлях до директорії з підготовленими даними
data_path = 'testing_data'
# Отримуємо список піддиректорій (наборів даних) у директорії з підготовленими даними
data_dir_list = os.listdir(data_path)
# Встановлюємо розміри зображень та кількість каналів
img_rows=256
img_cols=256
num_channel=1
# Встановлюємо кількість епох навчання
num_epoch=30
# Ініціалізуємо порожній список для зберігання зменшених зображень
img_data_list=[]
# Перебираємо кожен набір даних у директорії з підготовленими даними
for dataset in data_dir_list:
    # Отримуємо список файлів зображень у поточному наборі даних
    img_list=os.listdir(data_path+'/'+ dataset)
    # Виводимо повідомлення про завантаження зображень з поточного набору даних
    print ('Loaded the images of dataset-'+'{}\n'.format(dataset))
    # Перебираємо кожне зображення у поточному наборі даних
    for img in img_list:
        # Зчитуємо зображення за допомогою OpenCV
        input_img=cv2.imread(data_path + '/'+ dataset + '/'+ img )
        # Змінюємо розмір зображення до вказаних розмірів (48x48 пікселів)
        input_img_resize=cv2.resize(input_img,(48,48))
        # Додаємо зменшене зображення до списку
        img_data_list.append(input_img_resize)
img_data = np.array(img_data_list) # Конвертуємо список зображень в масив NumPy
img_data = img_data.astype('float32') # Конвертуємо дані зображень до типу float32
img_data = img_data/255 # Нормалізуємо дані зображень, масштабуючи значення пікселів до діапазону [0, 1]
```

```
img_data.shape # Виводимо форму отриманого масиву зображень
num_classes = 7 # Визначаємо кількість класів (емоцій), що визначаються
num_of_samples = img_data.shape[0] # Отримуємо кількість зразків (зображень) у даних
labels = np.ones((num_of_samples,),dtype='int64') # Ініціалізуємо масив міток, всі мітки встановлені в 1
# Задаємо мітки для кожного класу вручну заздалегідь, відповідно до розподілу зображень
labels[0:740]=0 #741
labels[741:1155]=1 #415
labels[1156:1636]=2 #480
labels[1637:2238]=3 #603
labels[2239:2490]=4 #252
labels[2491:2955]=5 #465
# Створюємо список імен класів, що відповідають їхнім числовим міткам
names = ['angry','disgusted','fearful','happy','sad','surprised']
# Визначаємо функцію для отримання імені класу за його числовою міткою
def getLabel(id):
    return ['angry','disgusted','fearful','happy','sad','surprised'][id]
```

ДОДАТОК Б

Створення моделі нейронної мережі CNN для порівняння алгоритмів оптимізації градієнта

```
# Функція активації ReLU
def relu(x):
    return np.maximum(0, x)
# Функція активації Softmax
def softmax(x):
    exp_x = np.exp(x - np.max(x))
    return exp_x / exp_x.sum(axis=0, keepdims=True)
# Функція для завантаження та обробки даних
def load_data(directory):
    data = []
    labels = []
    # Отримання списку класів (емоцій)
    classes = os.listdir(directory)
    for class_name in classes:
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            for image_name in os.listdir(class_path):
                image_path = os.path.join(class_path, image_name)
                # Завантаження та обробка зображення
                image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
                image = cv2.resize(image, (48, 48))
                image = image / 255.0  # Нормалізація значень пікселів до [0, 1]
                data.append(image)
                labels.append(classes.index(class_name))
    return np.array(data), np.array(labels)
def compute_loss_gradient_softmax(label, dense_output_softmax):
    # Ініціалізуємо градієнт втрат для softmax
    loss_gradient_softmax = np.zeros_like(dense_output_softmax)
```

```
    # Обчислюємо градієнт відносно виходу softmax тільки для вірного класу
    loss_gradient_softmax[label] = dense_output_softmax[label] - 1
    return loss_gradient_softmax

# Функція для згорткового шару
def convolution(image, kernel):
    image_height, image_width = image.shape
    kernel_height, kernel_width = kernel.shape
    # Розмір вихідного зображення після згортки
    output_height = image_height - kernel_height + 1
    output_width = image_width - kernel_width + 1

    # Створення вихідного зображення

    output_image = np.zeros((output_height, output_width))

    # Проходження ядром по зображенню та виконання операції згортки

    for i in range(output_height):

        for j in range(output_width):

            output_image[i, j] = np.sum(image[i:i+kernel_height, j:j+kernel_width] * kernel)

    return output_image

# Функція для максимального пулінгу

def max_pooling(feature_map, pool_size):

    height, width = feature_map.shape

    # Розмір відсіченого зображення після пулінгу

    output_height = height // pool_size

    output_width = width // pool_size

    # Створення вихідного зображення

    output_image = np.zeros((output_height, output_width))

    # Проходження вікном пулінгу по зображенню та вибір максимального значення

    for i in range(0, height - pool_size + 1, pool_size):

        for j in range(0, width - pool_size + 1, pool_size):

            output_image[i//pool_size, j//pool_size] = np.max(feature_map[i:i+pool_size, j:j+pool_size])

    return output_image

# Створення моделі

def create_model():

    # Параметри першого згорткового шару

    conv1_filters = 32
```

```
    conv1_kernel_size = (3, 3)
    conv1_weights = np.random.rand(*conv1_kernel_size)
    # Параметри другого згорткового шару
    conv2_filters = 64
    conv2_kernel_size = (3, 3)
    conv2_weights = np.random.rand(*conv2_kernel_size)
    # Параметри третього згорткового шару
    conv3_filters = 128
    conv3_kernel_size = (3, 3)
    conv3_weights = np.random.rand(*conv3_kernel_size)
    # Параметри четвертого згорткового шару
    conv4_filters = 256
    conv4_kernel_size = (3, 3)
    conv4_weights = np.random.rand(*conv4_kernel_size)
    # Розмір вікна максимального пулінгу
    pool_size = 2
    # Створення моделі
    model = dict()
    # Перший згортковий шар
    model['conv1'] = {'filters': conv1_filters, 'kernel_size': conv1_kernel_size, 'weights': conv1_weights,
'activation': 'relu'}
    # Максимальний пулінг
    model['pool1'] = {'pool_size': pool_size}
    # Другий згортковий шар
    model['conv2'] = {'filters': conv2_filters, 'kernel_size': conv2_kernel_size, 'weights': conv2_weights,
'activation': 'relu'}
    # Максимальний пулінг
    model['pool2'] = {'pool_size': pool_size}
    # Третій згортковий шар
    model['conv3'] = {'filters': conv3_filters, 'kernel_size': conv3_kernel_size, 'weights': conv3_weights,
'activation': 'relu'}
    # Максимальний пулінг
    model['pool3'] = {'pool_size': pool_size}
```

```
    # Четвертий згортковий шар
    model['conv4'] = {'filters': conv4_filters, 'kernel_size': conv4_kernel_size, 'weights': conv4_weights, 'activation': 'relu'}
    model['pool4'] = {'pool_size': pool_size}
    # Повнозв'язаний шар 1
    model['dense'] = {'units': 64, 'activation': 'relu'}
    # Повнозв'язаний шар 2
    model['dense2'] = {'units': 128, 'activation': 'relu'}
    # Повнозв'язаний шар 3 (вихідний)
    model['dense3'] = {'units': 10, 'activation': 'softmax'}  # Припустимо, що у нас 10 класів
    return model
def compute_loss_gradient_softmax(y_true, dense_output_softmax):
    # Ініціалізуємо градієнт втрат для softmax
    loss_gradient_softmax = np.zeros_like(dense_output_softmax)
    # Обчислюємо градієнт відносно виходу softmax тільки для вірних класів
    loss_gradient_softmax = compute_loss_gradient_softmax(label, dense3_output)
    return loss_gradient_softmax
def evaluate_model(X, y, model):
    correct_predictions = 0
    total_samples = len(X)
    for i in range(total_samples):
        image = X[i]
        label = y[i]
        # Операції згорткового шару, пулінгу та повнозв'язаного шару
        conv1_output = relu(convolution(image, model['conv1']['weights']))
        pool1_output = max_pooling(conv1_output, model['pool1']['pool_size'])
        conv2_output = relu(convolution(pool1_output, model['conv2']['weights']))
        pool2_output = max_pooling(conv2_output, model['pool2']['pool_size'])
        conv3_output = relu(convolution(pool2_output, model['conv3']['weights']))
        pool3_output = max_pooling(conv3_output, model['pool3']['pool_size'])
        conv4_output = relu(convolution(pool3_output, model['conv4']['weights']))
        pool4_output = max_pooling(conv4_output, model['pool4']['pool_size'])
```

```
        # Останнє зображення після згорткового шару
        last_conv_output = pool4_output
        pool4_output_size = last_conv_output.shape[0] * last_conv_output.shape[1]
        flatten_output = last_conv_output.flatten()
        dense_output = relu(np.dot(flatten_output, model['dense']['units']))
        dense2_output = relu(np.dot(dense_output, model['dense2']['units']))
        dense3_output = softmax(np.dot(dense2_output, model['dense3']['units']))
        # Обчислення передбачення
        predicted_label = np.argmax(dense3_output)
        # Порівняння з справжнім значенням
        if predicted_label == label:
            correct_predictions += 1
    # Обчислення точності
    accuracy = correct_predictions / total_samples
    return accuracy
# Завантаження та обробка даних
data, labels = load_data('prepared_data')
# Розбиття на тренувальний та тестовий набір
X_train, X_val, y_train, y_val = train_test_split(data, labels, test_size=0.2, random_state=42)
# Створення та компіляція моделі
model = create_model()
# Цикл тренування
num_epochs = 50
learning_rate = 0.001
beta1 = 0.9
beta2 = 0.999
epsilon = 1e-8
m_conv1 = np.zeros_like(model['conv1']['weights'])
v_conv1 = np.zeros_like(model['conv1']['weights'])
t_conv1 = 0
m_conv2 = np.zeros_like(model['conv2']['weights'])
```

```
v_conv2 = np.zeros_like(model['conv2']['weights'])
t_conv2 = 0
m_conv3 = np.zeros_like(model['conv3']['weights'])
v_conv3 = np.zeros_like(model['conv3']['weights'])
t_conv3 = 0
m_conv4 = np.zeros_like(model['conv4']['weights'])
v_conv4 = np.zeros_like(model['conv4']['weights'])
t_conv4 = 0
m_dense = np.zeros_like(model['dense']['units'])
v_dense = np.zeros_like(model['dense']['units'])
t_dense = 0
m_dense2 = np.zeros_like(model['dense2']['units'])
v_dense2 = np.zeros_like(model['dense2']['units'])
t_dense2 = 0
m_dense3 = np.zeros_like(model['dense3']['units'])
v_dense3 = np.zeros_like(model['dense3']['units'])
t_dense3 = 0
for epoch in range(num_epochs):
    # Передача даних через мережу та оновлення ваг
    for image, label in zip(X_train, y_train):
        # Операції згорткового шару, пулінгу та повнозв'язаного шару
        conv1_output = relu(convolution(image, model['conv1']['weights']))
        pool1_output = max_pooling(conv1_output, model['pool1']['pool_size'])
        conv2_output = relu(convolution(pool1_output, model['conv2']['weights']))
        pool2_output = max_pooling(conv2_output, model['pool2']['pool_size'])
        conv3_output = relu(convolution(pool2_output, model['conv3']['weights']))
        pool3_output = max_pooling(conv3_output, model['pool3']['pool_size'])
        conv4_output = relu(convolution(pool3_output, model['conv4']['weights']))
        pool4_output = max_pooling(conv4_output, model['pool4']['pool_size'])
        pool4_output_size = pool4_output.shape[0] * pool4_output.shape[1]
        flatten_output = pool4_output.flatten()
```

```
dense_output = relu(np.dot(flatten_output, model['dense']['units']))
dense2_output = relu(np.dot(dense_output, model['dense2']['units']))
dense3_output = softmax(np.dot(dense2_output, model['dense3']['units']))
# Обчислення градієнтів за параметрами
loss_gradient_softmax = compute_loss_gradient_softmax(label, dense3_output)
gradient_dense3 = np.outer(dense2_output, loss_gradient_softmax)
gradient_dense2 = np.outer(dense_output, np.dot(model['dense3']['units'], loss_gradient_softmax))
gradient_dense = np.outer(flatten_output, np.dot(model['dense2']['units'], gradient_dense2))
gradient_pool4 = np.dot(model['dense']['units'], np.dot(model['dense2']['units'], gradient_dense2))
gradient_conv4 = gradient_pool4.reshape(pool4_output.shape)
gradient_pool3 = relu(convolution(gradient_conv4, np.flip(model['conv4']['weights'])))
gradient_conv3 = gradient_pool3.reshape(pool3_output.shape)
gradient_pool2 = relu(convolution(gradient_conv3, np.flip(model['conv3']['weights'])))
gradient_conv2 = gradient_pool2.reshape(pool2_output.shape)
gradient_pool1 = relu(convolution(gradient_conv2, np.flip(model['conv2']['weights'])))
gradient_conv1 = gradient_pool1.reshape(pool1_output.shape)
# Оновлення вагових коефіцієнтів за алгоритмом Adam для згорткового шару 1
t_conv1 += 1
m_conv1 = beta1 * m_conv1 + (1 - beta1) * gradient_conv1
v_conv1 = beta2 * v_conv1 + (1 - beta2) * (gradient_conv1 ** 2)
m_hat_conv1 = m_conv1 / (1 - beta1 ** t_conv1)
v_hat_conv1 = v_conv1 / (1 - beta2 ** t_conv1)
model['conv1']['weights'] -= learning_rate * m_hat_conv1 / (np.sqrt(v_hat_conv1) + epsilon)
# Оновлення вагових коефіцієнтів за алгоритмом Adam для згорткового шару 2
t_conv2 += 1
m_conv2 = beta1 * m_conv2 + (1 - beta1) * gradient_conv2
v_conv2 = beta2 * v_conv2 + (1 - beta2) * (gradient_conv2 ** 2)
m_hat_conv2 = m_conv2 / (1 - beta1 ** t_conv2)
v_hat_conv2 = v_conv2 / (1 - beta2 ** t_conv2)
model['conv2']['weights'] -= learning_rate * m_hat_conv2 / (np.sqrt(v_hat_conv2) + epsilon)
# Оновлення вагових коефіцієнтів за алгоритмом Adam для згорткового шару 3
```

```
t_conv3 += 1
m_conv3 = beta1 * m_conv3 + (1 - beta1) * gradient_conv3
v_conv3 = beta2 * v_conv3 + (1 - beta2) * (gradient_conv3 ** 2)
m_hat_conv3 = m_conv3 / (1 - beta1 ** t_conv3)
v_hat_conv3 = v_conv3 / (1 - beta2 ** t_conv3)
model['conv3']['weights'] -= learning_rate * m_hat_conv3 / (np.sqrt(v_hat_conv3) + epsilon)
# Оновлення вагових коефіцієнтів за алгоритмом Adam для згорткового шару 4
t_conv4 += 1
m_conv4 = beta1 * m_conv4 + (1 - beta1) * gradient_conv4
v_conv4 = beta2 * v_conv4 + (1 - beta2) * (gradient_conv4 ** 2)
m_hat_conv4 = m_conv4 / (1 - beta1 ** t_conv4)
v_hat_conv4 = v_conv4 / (1 - beta2 ** t_conv4)
model['conv4']['weights'] -= learning_rate * m_hat_conv4 / (np.sqrt(v_hat_conv4) + epsilon)
# Оновлення вагових коефіцієнтів за алгоритмом Adam для повнозв'язаного шару 1
t_dense += 1
m_dense = beta1 * m_dense + (1 - beta1) * gradient_dense
v_dense = beta2 * v_dense + (1 - beta2) * (gradient_dense ** 2)
m_hat_dense = m_dense / (1 - beta1 ** t_dense)
v_hat_dense = v_dense / (1 - beta2 ** t_dense)
model['dense']['units'] -= learning_rate * m_hat_dense / (np.sqrt(v_hat_dense) + epsilon)
# Оновлення вагових коефіцієнтів за алгоритмом Adam для повнозв'язаного шару 2
t_dense2 += 1
m_dense2 = beta1 * m_dense2 + (1 - beta1) * gradient_dense2
v_dense2 = beta2 * v_dense2 + (1 - beta2) * (gradient_dense2 ** 2)
m_hat_dense2 = m_dense2 / (1 - beta1 ** t_dense2)
v_hat_dense2 = v_dense2 / (1 - beta2 ** t_dense2)
model['dense2']['units'] -= learning_rate * m_hat_dense2 / (np.sqrt(v_hat_dense2) + epsilon)
# Оновлення вагових коефіцієнтів за алгоритмом Adam для повнозв'язаного шару 3
t_dense3 += 1
m_dense3 = beta1 * m_dense3 + (1 - beta1) * gradient_dense3
v_dense3 = beta2 * v_dense3 + (1 - beta2) * (gradient_dense3 ** 2)
```

```
    m_hat_dense3 = m_dense3 / (1 - beta1 ** t_dense3)
    v_hat_dense3 = v_dense3 / (1 - beta2 ** t_dense3)
    model['dense3']['units'] -= learning_rate * m_hat_dense3 / (np.sqrt(v_hat_dense3) + epsilon)
# Оцінка моделі на валідаційному наборі
val_accuracy = evaluate_model(X_val, y_val, model)
print(f'Epoch {epoch + 1}/{num_epochs}, Validation Accuracy: {val_accuracy}')
```

ДОДАТОК В

Створення моделі нейронної мережі CNN

```
Y = np_utils.to_categorical(labels, num_classes)

x,y = shuffle(img_data,Y, random_state=2)

# Розділяємо набір даних на навчальний і тестовий

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=2)

# Зберігаємо тестовий набір як окрему змінну для подальшого використання

x_test=X_test

# Визначаємо новий клас, який успадковує від Conv2D

class StandardizedConv2DWithOverride(layers.Conv2D):

    # Перевизначаємо операцію згортки

    def convolution_op(self, inputs, kernel):

        # Використовуємо функцію moments для обчислення середнього значення та дисперсії ядра

        mean, var = tf.nn.moments(kernel, axes=[0, 1, 2], keepdims=True)

        # Використовуємо стандартну операцію згортки з нормалізованим ядром

        return tf.nn.conv2d(

            inputs,

            (kernel - mean) / tf.sqrt(var + 1e-10), # Використовуємо стандартизацію для отримання нормалізованого ядра

            padding="SAME",

            strides=list(self.strides),

            name=self.__class__.__name__)

# Inputs

input_layer = Input((48,48,3)) # Визначення вхідного шару розмірністю 48x48 пікселів та 3 канали (RGB)

# Перший блок згорткових шарів

f1=StandardizedConv2DWithOverride(32, kernel_size=3, strides=1, padding='same', activation='relu')(input_layer)

f1=BatchNormalization()(f1)

f=StandardizedConv2DWithOverride(32, kernel_size=3, strides=1, padding='same', activation='relu')(f1)

f=StandardizedConv2DWithOverride(32, kernel_size=3, strides=1, padding='same', activation='relu')(f)
```

```
f=MaxPooling2D(2,2)(f)

f2=Conv2D(32, kernel_size=1, strides=2, padding='same', activation='relu')(f)

# Другий блок згорткових шарів з різними розмірами ядер

f1=StandardizedConv2DWithOverride(32, kernel_size=5, strides=1, padding='same',
activation='relu')(f1)

f1=StandardizedConv2DWithOverride(32, kernel_size=5, strides=2, padding='same',
activation='relu')(f1)

# Злиття двох блоків

f=concatenate([f,f1])

f=BatchNormalization()(f)

# Третій блок згорткових шарів

f1=StandardizedConv2DWithOverride(64, kernel_size=3, strides=1, padding='same',
activation='relu')(f)

f=StandardizedConv2DWithOverride(64, kernel_size=3, strides=1, padding='same',
activation='relu')(f1)

f=StandardizedConv2DWithOverride(64, kernel_size=3, strides=1, padding='same',
activation='relu',name='BeforeFinal_Layer')(f)

f=MaxPooling2D(2,2)(f)

f3=Conv2D(32, kernel_size=1, strides=2, padding='same', activation='relu')(f)

# Четвертий блок згорткових шарів

f1=StandardizedConv2DWithOverride(64, kernel_size=5, strides=1, padding='same',
activation='relu')(f1)

f1=StandardizedConv2DWithOverride(64, kernel_size=5, strides=2, padding='same',
activation='relu')(f1)

# Злиття з третім блоком

f=concatenate([f,f1])

f=BatchNormalization()(f)

# П'ятий блок згорткових шарів

f1=StandardizedConv2DWithOverride(128, kernel_size=3, strides=1, padding='same',
activation='relu')(f)

f=Conv2D(128, kernel_size=3, strides=1, padding='same', activation='relu')(f1)

f=Conv2D(128, kernel_size=3, strides=1, padding='same', activation='relu')(f)

f=MaxPooling2D(2,2)(f)

f4=Conv2D(32, kernel_size=1, strides=2, padding='same', activation='relu')(f)
```

```
# Шостий блок згорткових шарів
f1=StandardizedConv2DWithOverride(128, kernel_size=5, strides=1, padding='same', activation='relu')(f1)
f1=StandardizedConv2DWithOverride(128, kernel_size=5, strides=2, padding='same', activation='relu')(f1)
f1=BatchNormalization()(f1)
# Злиття з попереднім блоком
f=concatenate([f,f1])
# Сьомий блок згорткових шарів
f1=Conv2D(256, kernel_size=3, strides=1, padding='same', activation='relu')(f)
f=Conv2D(256, kernel_size=3, strides=1, padding='same', activation='relu')(f1)
f=Conv2D(256, kernel_size=3, strides=1, padding='same', activation='relu')(f)
f=MaxPooling2D(2,2)(f)
# Восьмий блок згорткових шарів
f1=Conv2D(512, kernel_size=3, strides=2, padding='same', activation='relu')(f1)
f1=BatchNormalization()(f1)
# Злиття двох блоків перед останнім шаром
f=concatenate([f,f1])
f=StandardizedConv2DWithOverride(512, kernel_size=3, strides=1, padding='same', activation='relu', name='Final_Layer')(f)
f=BatchNormalization()(f)
# Останній блок згорткових шарів
f= Flatten()(f)
f=Dropout(rate=0.3)(f)
f=BatchNormalization()(f)
f=Dense(512, activation='relu')(f)
f=Dropout(rate=0.32)(f)
f=BatchNormalization()(f)
# Вихідний шар
output_layer=Dense(7, activation='softmax')(f)
# Визначення моделі, використовуючи вхідний та вихідний шари
model = Model(
    inputs=[input_layer],
```

```
    outputs=[output_layer])
# Визначення кількості епох та швидкості навчання
epochs = 30
learning_rate = 1e-3
# Визначення оптимізатора (Adam) та компіляція моделі
opt = Adam(learning_rate=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=None, amsgrad=False)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
# Визначення зворотніх викликів для використання під час тренування
callbacks = [ModelCheckpoint('TESTING_model.hdf5',monitor="val_accuracy", verbose=1,
save_best_only=True),
        ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=20, verbose=1, min_lr=1e-
6),
        EarlyStopping(monitor='val_accuracy', restore_best_weights=True, patience=100)]
# Тренування моделі з використанням навчальних та валідаційних даних
history = hist = model.fit(X_train, y_train, batch_size=7, epochs=epochs, verbose=1,
validation_data=(X_test, y_test),callbacks=callbacks)
```