Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Виконала: студентка 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Артеменко І. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф.КН
Паночишин Ю.М.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., проф. каф. КСУ
Биков М. М.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Артеменко Ірині Володимирівні

1. Тема роботи: Програмний модуль розпізнавання мови жестів на основі згорткової нейронної мережі.
керівник роботи: Паночишин Юрій Миколайович, к.т.н., доц.
затверджені наказом вищого навчального закладу "20" березня 2025 року №97

2. Термін подання студентом роботи 30 травня 2025 р

3. Вихідні дані до роботи :
Формат вхідних файлів зображень – jpeg; кількість жестів для розпізнавання - 24; обсяг набору даних для навчання та тестування – не менше 1000 зображень; використання об'єктноорієнтованої мови програмування.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
проаналізувати існуючі методи розпізнавання мови жестів та вибрати найбільш ефективний;
вибрати нейронну мережу, спроектувати її структуру та метод навчання
розробити алгоритм роботи модуля розпізнавання мови жестів;
спроектувати та реалізувати модуль розпізнавання мови жестів за допомогою згорткової нейронної мережі;
протестувати роботу програми розпізнавання мови жестів на основі згорткової нейронної мережі.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Схема алгоритму роботи програми
Структура нейронної мережі
Кодування англійських літер мовою жестів
Результати роботи програми

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Паночишин Ю.М., к.т.н., доц. каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	05.05.25	07.05.25	
2	Обгрунтування методу розв'язання задачі, проектування програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі	08.05.25	12.05.25	
3	Розробка алгоритму роботи програмного модуля розпізнавання мови жестів	13.05.25	16.05.25	
4	Програмна реалізація модуля розпізнавання мови жестів	17.05.25	29.05.25	
5	Аналіз результатів тестування модуля розпізнавання мови жестів	30.05.25	01.05.25	
6	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студентка _____ **Артеменко І. В.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Паночишин Ю.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Артеменко І. В. Програмний модуль розпізнавання мови жестів на основі згорткової нейронної мережі. Бакалаврська кваліфікаційна робота скаладається з 78 сторінок формату А4, на яких є 15 рисунків, 2 таблиці, список використаних джерел містить 22 найменувань.

Метою роботи є підвищення достовірності розпізнавання мови жестів.

Розроблений програмний модуль призначений для розпізнавання мови жестів з використанням згорткової нейронної мережі. Із таких проаналізованих архітектур: LeNet, AlexNet, ZF Net, GoogLeNet, VGGNet, ResNet, було обрано згорткову нейромережу VGGNet як найбільш перспективну. для вирішення задачі розпізнавання мови жестів. Але було прийнято рішення використовувати власні структури згорткових нейронних мереж, але побудовані на основі структури VGG16. Модуль розроблено на мові програмування Python з використанням бібліотек TensorFlow, Keras, Sklearn, Numpy, Pandas, Matplotlib. Для навчання та тестування модуля взято набір даних мови жестів MNIST (Sign Language MNIST), який містить 27455 навчальних зображень та 7172 тестових зображення. Розроблений програмний модуль розпізнавання мови жестів на основі згорткової нейронної мережі має достовірність розпізнавання 96,3%, що порівняно з аналогом збільшено на 5,8%.

Ключові слова: мова жестів, згорткова нейронна мережа, VGGNet, достовірність розпізнавання.

ABSTRACT

Artemenko I. V. Software module for sign language recognition based on a convolutional neural network. The bachelor thesis consists of 78 pages of A4 format, on which there are 15 figures, 2 tables, the list of used sources contains 22 names.

The aim of the work is to increase the reliability of sign language recognition.

The developed software module is designed for sign language recognition using a convolutional neural network. Of the following analyzed architectures: LeNet, AlexNet, ZF Net, GoogLeNet, VGGNet, ResNet, the VGGNet convolutional neural network was chosen as the most promising. to solve the problem of sign language recognition. But it was decided to use our own structures of convolutional neural networks, but built on the basis of the VGG16 structure. The module was developed in the Python programming language using the TensorFlow, Keras, Sklearn, Numpy, Pandas, Matplotlib libraries. For training and testing the module, the MNIST sign language dataset was taken, which contains 27455 training images and 7172 test images. The developed software module for sign language recognition based on a convolutional neural network has a recognition reliability of 96.3%, which is increased by 5.8% compared to the analogue.

Keywords: sign language, convolutional neural network, VGGNet, recognition accuracy.

ЗМІСТ

ВСТУП

Актуальність досліджень. Розпізнавання жестів рук має широкий спектр застосувань. Мова жестів є основним способом спілкування понад 70 мільйонів глухих людей з неглухими. Здатність автоматично перетворювати жести рук у текст може полегшити спілкування глухих з неглухими. У цій роботі досліджується проблема ідентифікації двадяти чотирьох жестів рук, що представляють англійські літери мовою жестів.

Крім того, оскільки віртуальна реальність набирає все більшої популярності, здатність ретельно розпізнавати, який жест рукою використовує користувач, має вирішальне значення для забезпечення спілкування з комп’ютером. Взаємодія рук є основним компонентом в індустрії VR/AR (VR - Virtual Reality - віртуальна реальність, AR - Augmented Reality - доповнена реальність), і відстеження рук є важливим для такого виду віртуальної взаємодії. Для вирішення цієї проблеми останнім часом все частіше використовують штучні нейронні мережі. Вони приймають на вхід набір зображень рук з різним фоном і видають назву жесту рук.

Порушення мовлення обмежує здатність людини до усного та слухового спілкування. Покращення комунікації між глухими та широкою публікою може бути досягнуто за допомогою детектора жестової мови в режимі реального часу. Нещодавні дослідження сприяли досягненню прогресу в процесах розпізнавання рухів та жестів за допомогою методів глибокого навчання (DL) та комп'ютерного зору. Але розробка статичних та динамічних моделей розпізнавання жестової мови (SLR) все ще залишається складною галуззю досліджень. Складність полягає в отриманні відповідної моделі, яка вирішує проблеми безперервних знаків, незалежних від людини, яка говорить жестами. Різна швидкість, тривалість та багато інших факторів мови жестів ускладнюють створення моделі з високою точністю та безперервністю. Це дослідження в основному зосереджено на розпізнаванні жестової мови з

використанням модифікованого глибокого навчання та гібридного підходу до оптимізації. Зокрема, просторові та геометричні ознаки витягуються за допомогою згорткових нейромереж (наприклад, Visual Geometry Group 16 - VGG16), а ознаки руху витягуються за допомогою підходу оптичного потоку.

У цій роботі ставиться завдання побудови програмного модуля на основі моделі згорткової нейронної мережі, яка перетворює мову жестів на англійську абетку.

Метою дослідження є підвищення достовірності розпізнавання мови жестів.

Об'єктом дослідження є процес комп'ютеризованого розпізнавання мови жестів на основі нейромережі.

Предметом дослідження є алгоритми та програмні засоби розпізнавання мови жестів на основі згорткової нейронної мережі та достовірність їх роботи.

Задачі дослідження:

1. Проаналізувати відомі методи розпізнавання мови жестів та обрати напрямок досліджень;
2. Обгрунтувати вибір архітектури згорткової нейромережі;
3. Розробити структуру згорткової нейронної мережі для розпізнавання мови жестів;
4. Розробити алгоритм роботи програмного модуля розпізнавання мови жестів на основі згорткової нейромережі;
5. Здійснити програмну реалізацію модуля розпізнавання мови жестів на основі згорткової нейромережі;
6. Провести тестування програмного модуля розпізнавання мови жестів на основі згорткової нейромережі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ

1.1 Постановка задачі

Перетворення мови жестів у текст можна звести до завдання класифікації образів , коли візуальний образ жеста руки потрібно віднести до певного класу, якому відповідає латинська (англійська) літера.

Як вхідні дані будемо використовувати MNIST-датасет [1] мови жестів від Kaggle, який поширюється під ліцензією CC0: Public Domain. Кодування англійських літер мовою жестів представлено на рис. 1.1.

Рисунок 1.1 – Кодування англійських літер мовою жестів

У наборі даних MNIST-датасет мови жестів для літер J та Z немає репрезентацій (Причина: жести J та Z використовують рухи). Зображення є чорно-білими. Значення пікселів лежать у діапазоні від 0 до 255. Кожне зображення містить 784 пікселі. Мітки кодуються цифрами, які варіюються від 0 до 25 відповідно від A до Z. Дані містять навчальний і тестовий набори, кожна група яких включає зображення, що містять 784 пікселя і відповідну мітку. Завдання полягає в розробці програми, яка буде по зображенню жеста визначати яку саме літеру англійського алфавіта він кодує і виводити цю літеру.

1.2 Огляд відомих методів розпізнавання мови жестів

Наша задача є окремим випадком загальної задачі розпізнавання образів. Розглянемо які є методи розпізнавання образів і оберемо той, що найбільше підходить для нашої задачі.

Розпізнавання образів охоплює різні методи, що використовуються для ідентифікації та класифікації закономірностей у даних (рис. 1.2). Ці методи можна загалом розділити на статистичні, структурні та нейромережеві підходи [2]. Крім того, поширеними методами є зіставлення шаблонів та нечіткі моделі.

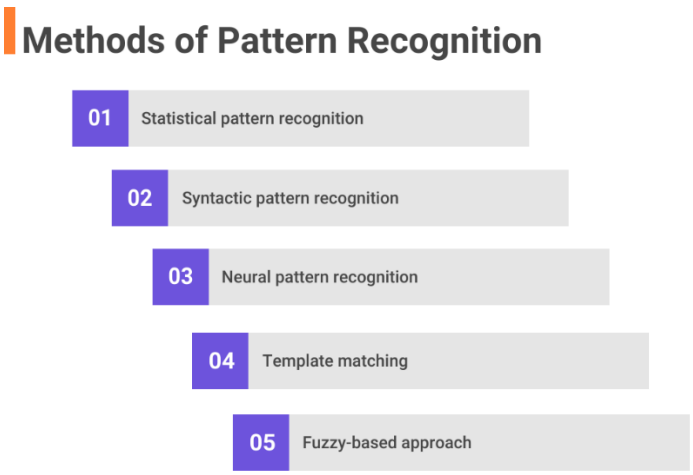

Рисунок 1.2 – Методи розпізнавання образів

Ось детальніший огляд різних методів:

1. Статистичне розпізнавання образів:

Цей підхід до розпізнавання образів використовує історичні статистичні дані, які навчаються на основі закономірностей та прикладів. Метод збирає спостереження та обробляє їх для визначення моделі. Ця модель потім узагальнює зібрані спостереження та застосовує правила до нових наборів даних або прикладів.

Концепція: Використовує математичні та статистичні методи для класифікації даних на основі розподілів ймовірностей.

Приклади: Байєсівська класифікація, лінійний дискримінантний аналіз (LDA), метод опорних векторів (SVM).

Кроки: Вилучення ознак, оцінка моделі та класифікація.

2. Структурне розпізнавання образів:

Структурне розпізнавання образів включає складні шаблони, які можна ідентифікувати за допомогою ієрархічного підходу. Шаблони встановлюються на основі того, як примітиви (наприклад, літери в слові) взаємодіють один з одним. Прикладом цього може бути те, як примітиви об'єднуються в слова та речення. Такі навчальні зразки дозволять розробити граматичні правила, які демонструють, як речення будуть читатися в майбутньому.

Концепція: Зосереджується на зв'язках між ознаками для розпізнавання закономірностей, особливо корисно для складних структур.

Підхід: Ієрархічний підхід з категоризацією на підкласи.

3. Розпізнавання образів на основі нейронних мереж:

Цей метод використовує штучні нейронні мережі (ШНМ) [3,4] та навчається на складних та нелінійних співвідношеннях вхід/вихід, адаптується до даних та виявляє закономірності. Найпопулярнішим та найефективнішим методом у нейронних мережах є метод прямого зв'язку. У цьому методі

навчання відбувається шляхом надання зворотного зв'язку до вхідних шаблонів. Це дуже схоже на те, як люди навчаються на своєму минулому досвіді та помилках. Модель на основі ШНМ оцінюється як найдорожчий метод розпізнавання шаблонів порівняно з іншими методами через обчислювальні ресурси, що задіяні в процесі.

Концепція: Використовує штучні нейронні мережі для розпізнавання образів.

Перевага: Більш гнучке та ефективне в класифікації, особливо з методами глибокого навчання.

4. Зіставлення шаблонів:

Зіставлення шаблонів є одним з найпростіших з усіх підходів до розпізнавання образів. Тут подібність між двома сутностями визначається шляхом зіставлення зразка з еталонним шаблоном. Такі методи зазвичай використовуються в цифровій обробці зображень, де невеликі ділянки зображення зіставляються зі збереженим зображенням шаблону. Деякі з реальних прикладів включають обробку медичних зображень, розпізнавання облич та навігацію роботів.

Концепція: Порівнює цільовий образ зі збереженим шаблоном для визначення подібності.

Застосування: Використовується при роботі з подібними типами об'єктів, такими як фігури або криві.

5. Нечіткі моделі:

У нечіткому підході набір шаблонів розділяється на основі подібності в ознаках шаблонів. Коли унікальні ознаки шаблону правильно виявлені, дані можна легко класифікувати у цей відомий простір ознак. Навіть людська зорова система іноді не розпізнає певні компоненти, незважаючи на тривале сканування об'єктів. Те саме стосується і цифрового світу, де алгоритми не можуть визначити точну природу об'єкта. Отже, нечіткий підхід спрямований

на класифікацію об'єктів на основі кількох подібних ознак у виявлених шаблонах.

Концепція: Включає нечітку логіку для обробки неточних або невизначених даних.

Застосування: Корисний для реальних задач розпізнавання, де поширена нечіткість.

6. Гібридні моделі:

Гібридний підхід використовує комбінацію вищезазначених методів, щоб скористатися перевагами всіх цих методів. Він використовує кілька класифікаторів для виявлення шаблонів, де кожен класифікатор навчається на певному просторі ознак. Висновок робиться на основі результатів, накопичених усіма класифікаторами.

Концепція: Поєднує різні методи розпізнавання образів, щоб використовувати їхні сильні сторони.

Інші підходи:

Навчання з учителем, без учителя та напівнавчання з учителем:

Це парадигми навчання, які можна застосовувати в розпізнаванні образів.

Глибоке навчання [5].:

Підмножина машинного навчання, особливо корисна для складних завдань розпізнавання образів.

Процес розпізнавання образів складається з таких етапів:

1. Збір даних: Збір необхідних даних для навчання та тестування.
2. Вибір ознак: Визначення найбільш релевантних ознак для розпізнавання образів.
3. Вибір моделі: Вибір відповідного алгоритму або моделі для завдання.
4. Навчання: Навчання моделі на даних.

5. Оцінювання та уточнення: Оцінювання продуктивності моделі та внесення коригувань.

Таким чином, огляд літературних джерел показав, що найбільш перспективним для задачі розпізнавання мови жестів є метод на основі штучних нейронних мереж, оскільки він більш гнучкий та ефективний в класифікації зображень, особливо, при використанні згорткових нейромереж.

1.3 Обгрунтування вибору аналогів розробленого модуля розпізнавання мови жестів на основі згорткової нейронної мережі

У статті [6] розпізнавання мови жестів виконано за допомогою PCA (аналізу головних компонентів). У статті також пропонується розпізнавання за допомогою нейронних мереж. Дані вони отримали за допомогою 3-мегапіксельної камери, тому якість була низькою. Для кожного знаку вони зробили 15 зображень та зберегли їх у своїй базі даних. Невеликий набір даних був однією з причин, чому їхні результати були незадовільними. Вони виконали сегментацію та розділили RGB на компоненти, а також виконали простий аналіз пікселів на контурах. Вони зазначили, що хоча поєднання алгоритму контурів з PCA дало хороший результат, кращий результат можна отримати за допомогою нейронних мереж.

Розпізнавання жестової мови, виконане в статті [7], використовує згорткову нейронну мережу (ЗНМ) [8,9]. Для автоматизації процесу розпізнавання жестової мови було виконано два кроки: вилучення ознак та класифікація дій. Перший крок виконується за допомогою ЗНМ, а наступний — за допомогою традиційної штучної нейронної мережі. Набір даних містить загалом 20 різних італійських жестів, які були сформовані 27 людьми в різних середовищах. Відео записувалися за допомогою Microsoft Kinect. Для розробки було використано загалом 6600 зображень, з яких 4600 було використано для навчання, а решта — для тестування. Зображення були

попередньо оброблені для обрізання верхньої частини руки та тулуба. Для всіх жестів модель повинна вивчити лише одну сторону. Для вилучення ознак та класифікації використовується максимальне об'єднання (max-pooling). Він складається з 2 ЗНМ, вихід яких надходить до традиційної ШНМ. Традиційна ШНМ поєднує вихід двох ЗНМ. Для доповнення даних використовувався центральний процесор, а для навчання моделі — графічний процесор. Масштабування, обертання та зміщення виконувалися як частина аугментації (розширення) даних. Навчання з використанням цієї моделі призвело до достовірності 91,7%.

У статті [10] для розпізнавання американської жестової мови було використано згорткові нейронні мережі. Використаний набір даних зображень складається зі статичних жестів жестової мови, знятих на RGB-камеру. Набір даних для навчання та тестування складався в середньому із 100 зображень на кожен клас з 24 літер, тобто разом 2400 зображень. Із них 80 % кожного класу було взято для навчання і 20% для тестування. Попередню обробку було виконано над зображеннями, які потім слугували очищеними вхідними даними. У статті представлені результати, отримані шляхом перенавчання та тестування цього набору даних жестів жестової мови на моделі згорткової нейронної мережі з використанням Inception v3 [11]. Модель складається з кількох вхідних даних згорткового фільтра, які обробляються на одному й тому ж вході. Отримана достовірність тестування складала 90,5%. У цій статті також розглядаються різні спроби розпізнавання жестової мови за допомогою машинного навчання та даних глибини зображень.

Обидві цих статті [6, 10] описують програмні реалізації розроблених в них методів, а тому можуть бути взяті за аналоги. Як бачимо, достовірність розпізнавання зображень жестової мови складає трохи більше 90% (відповідно 91,7% та 90,5%), що є недостатнім. Звідси і випливає мета бакалаврської кваліфікаційної роботи - підвищення достовірності розпізнавання мови жестів.

1.4 Висновок до розділу 1

У розділі описано детальну постановку задачі розпізнавання мови жестів. Було розглянуто 5 методів розпізнавання образів. Як найбільш перспективний для задачі розпізнавання мови жестів було обрано метод на основі штучних нейронних мереж, а згорткові нейромережі визначено як найбільш ефективні для роботи із зображеннями. Крім цього, було обгрунтовано вибір аналогів розробленого модуля розпізнавання мови жестів. Обидва аналоги використовують згорткові нейромережі різних типів і мають достовірність розпізнавання зображень жестової мови відповідно 91,7% та 90,5%, що є недостатнім. Звідси і випливає мета бакалаврської кваліфікаційної роботи - підвищення достовірності розпізнавання мови жестів.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1 Аналіз роботи згорткової нейронної мережі для розпізнавання мови жестів

Згорткові нейронні мережі (ЗНМ [8,9]) схожі на звичайні нейронні мережі: вони складаються з нейронів, які мають вагу та зсуви, що можуть навчатися. Кожен нейрон отримує певні вхідні дані, виконує точковий добуток і, за бажанням, слідує за ним з нелінійністю. Вся мережа виражає єдину диференційовану функцію оцінки: від пікселів вхідного зображення на одному кінці до оцінок імовірностей класів на іншому. І мережа ще має функцію втрат (наприклад, SVM/Softmax) на останньому (повністю зв'язаному) шарі, і всі поради/підказки, які існують для навчання звичайних нейронних мереж, застосовуються і для згорткових.

Так що ж змінюється? Архітектура ЗНМ передбачає, що вхідні дані є зображеннями, що дозволяє нам закодувати певні властивості в архітектурі. Це робить функцію прямого поширення більш ефективною для реалізації і значно зменшує кількість параметрів у мережі.

Огляд архітектури

Нейронні мережі отримують вхідні дані (один вектор) і перетворюють їх через ряд прихованих шарів. Кожен прихований шар складається з набору нейронів, де кожен нейрон повністю пов'язаний з усіма нейронами попереднього шару, і де нейрони в одному шарі функціонують повністю незалежно і не мають спільних зв'язків. Останній повністю пов'язаний шар називається "вихідним шаром" і в налаштуваннях класифікації він представляє оцінки класів.

Звичайні нейронні мережі погано масштабуються до повних зображень. У CIFAR-10 зображення мають лише розмір 32x32x3 (32 по ширині, 32 по

висоті, 3 кольорові канали), тому один повністю зв'язаний нейрон у першому прихованому шарі звичайної нейронної мережі мав би 32*32*3 = 3072 вагових коефіцієнти. Ця кількість все ще здається керованою, але очевидно, що ця повністю зв'язана структура не масштабується до більших зображень. Наприклад, зображення більш солідного розміру, наприклад, 200x200x3, призведе до того, що нейрони матимуть 200*200*3 = 120 000 ваг. Більше того, ми майже напевно захочемо мати кілька таких нейронів, тож параметри швидко складатимуться! Очевидно, що така повна зв'язність є марнотратством і величезна кількість параметрів швидко призвела б до перенастроювання.

3D-об'єми нейронів. Згорткові нейронні мережі використовують той факт, що вхідні дані складаються з зображень, і вони обмежують архітектуру більш розумним чином. Зокрема, на відміну від звичайної нейронної мережі, шари ЗНМ мають нейрони, розташовані у 3 вимірах: ширина, висота, глибина. (Слово глибина тут відноситься до третього виміру об'єму активації, а не до глибини повної нейронної мережі, яка може відноситися до загальної кількості шарів у мережі). Наприклад, вхідні зображення в CIFAR-10 є вхідним об'ємом активацій, і цей об'єм має розміри 32x32x3 (ширина, висота, глибина відповідно). В ЗНМ [9] нейрони в шарі з'єднані лише з невеликою областю шару перед ним, а не з усіма нейронами повністю. Більше того, кінцевий вихідний шар для CIFAR-10 матиме розміри 1x1x10, тому що до кінця роботи архітектури ЗНМ ми зведемо повне зображення до єдиного вектора оцінок класів, розташованих уздовж виміру глибини (рис. 2.1).

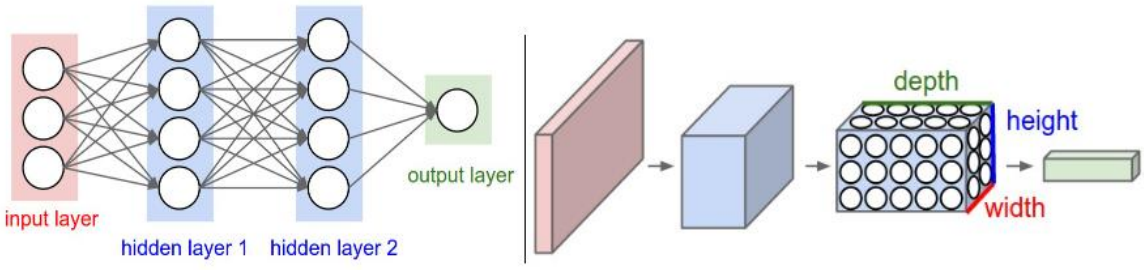

Рисунок 2.1 – Порівняння звичайної нейромережі та згорткової

На рис. 2.1 ліворуч: звичайна 3-шарова нейронна мережа. Праворуч: ЗНМ розташовує свої нейрони у трьох вимірах (ширина, висота, глибина), як показано на одному з шарів. Кожен шар ЗНМ перетворює тривимірний вхідний об'єм у тривимірний вихідний об'єм активацій нейронів. У цьому прикладі червоний вхідний шар містить зображення, тому його ширина і висота будуть розмірами зображення, а глибина буде дорівнювати 3 (червоний, зелений, синій канали).

ЗНМ складається з шарів. Кожен шар має простий API: Він перетворює вхідний 3D-об'єм на вихідний 3D-об'єм за допомогою деякої диференційованої функції, яка може мати або не мати параметрів.

Розглянемо шари, що використовуються для побудови ConvNet.

Проста ConvNet - це послідовність шарів, і кожен шар ConvNet перетворює один об'єм активацій в інший за допомогою диференційованої функції. Ми використовуємо три основні типи шарів для побудови архітектури ConvNet: Згортковий шар, об'єднувальний шар і повністю з'єднаний шар (Convolutional Layer, Pooling Layer, and Fully-Connected Layer) (так само, як і у звичайних нейронних мережах). Ми комбінуємо ці шари, щоб сформувати повну архітектуру ConvNet [8].

Приклад архітектури. Простий ConvNet для класифікації CIFAR-10 може мати архітектуру [INPUT - CONV - RELU - POOL - FC]. Більш детально:

− INPUT [32x32x3] містить вихідні значення пікселів зображення, в даному випадку зображення шириною 32, висотою 32 і з трьома колірними каналами R,G,B.

− Шар CONV обчислюватиме вихід нейронів, які з'єднані з локальними областями на вході, кожен з яких обчислюватиме точковий добуток між своєю вагою та невеликою областю, до якої він з'єднаний у вхідному об'ємі. Це може призвести до об'єму [32x32x12], якщо ми вирішили використати 12 фільтрів.

− Шар RELU застосує поелементну функцію активації, наприклад, max(0,x) з нульовим порогом. Це залишить розмір об'єму незмінним ([32x32x12]).

− Шар POOL виконує операцію дискретизації вздовж просторових розмірів (ширина, висота), що призводить до об'єму типу [16x16x12].

− Шар FC (тобто повністю пов'язаний) обчислюватиме бали класу, що призведе до об'єму розміром [1x1x10], де кожне з 10 чисел відповідає балу класу, як, наприклад, серед 10 категорій CIFAR-10. Як і у випадку зі звичайними нейронними мережами, і як випливає з назви, кожен нейрон у цьому шарі буде з'єднаний з усіма числами в попередньому шарі.

Таким чином, ConvNets перетворює вихідне зображення шар за шаром від початкових значень пікселів до кінцевих оцінок класів. Деякі шари містять параметри, а інші ні. Зокрема, шари CONV/FC виконують перетворення, які є функцією не лише активацій у вхідному об'ємі, але й параметрів (ваг та зсувів нейронів). З іншого боку, шари RELU/POOL реалізують фіксовану функцію. Параметри в шарах CONV/FC навчатимуться за допомогою градієнтного спуску, щоб оцінки класів, які обчислює ConvNet, відповідали міткам у навчальній вибірці для кожного зображення.

Таким чином:

− Архітектура ConvNet у найпростішому випадку є списком шарів, які перетворюють об'єм зображення у вихідний об'єм (наприклад, отримують бали класу) – рис. 2.2,

− Існує декілька різних типів шарів (наприклад, CONV/FC/RELU/POOL є найпопулярнішими),

− Кожен шар приймає вхідний 3D-об'єм і перетворює його на вихідний 3D-об'єм за допомогою диференційованої функції,

− Кожен шар може мати або не мати параметрів (наприклад, CONV/FC мають, RELU/POOL не мають),

– Кожен шар може мати або не мати додаткових гіперпараметрів (наприклад, CONV/FC/POOL мають, RELU не має).

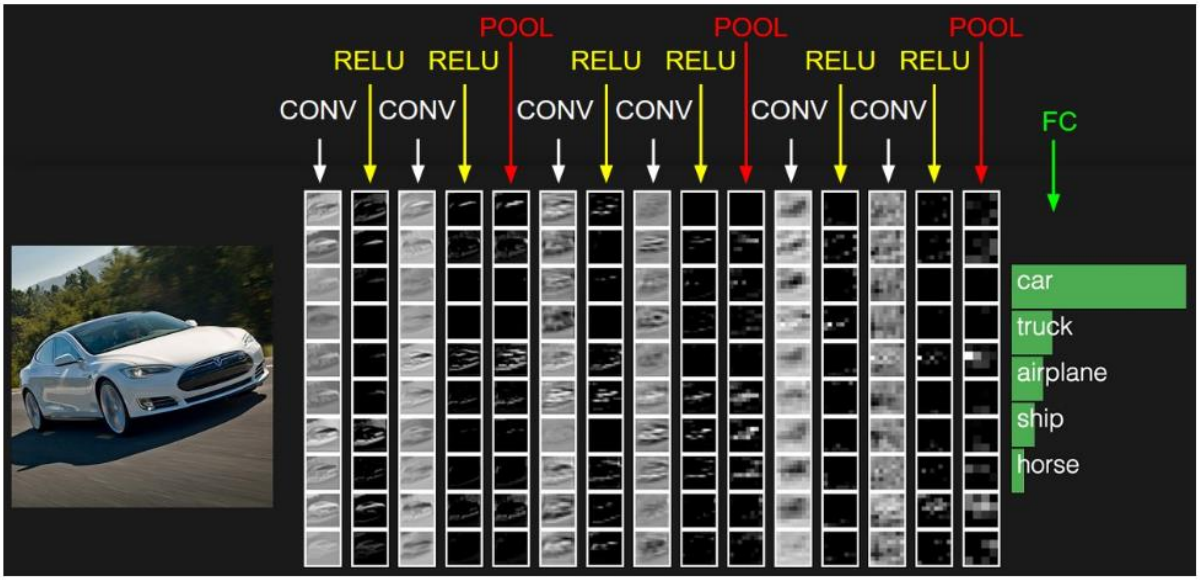

Рисунок 2.2 – Приклад архітектури ЗНМ

На рис. 2.2 початковий шар зберігає пікселі необробленого зображення (ліворуч), а останній шар зберігає оцінки класів (праворуч). Останній шар містить оцінки для кожного класу, але тут ми візуалізуємо лише відсортовані 5 найкращих оцінок і роздруковуємо етикетки для кожної з них. Архітектура, показана тут, є зменшеною VGG-мережею [12].

Опишемо окремі шари та деталі їхніх гіперпараметрів і зв'язків між ними.

2.1.1. Згортковий шар.

Згортковий шар - це основний будівельний блок згорткової мережі, який виконує більшу частину обчислювальної роботи.

Параметри шару CONV складаються з набору фільтрів, що навчаються. Кожен фільтр є невеликим у просторі (по ширині та висоті), але поширюється на всю глибину вхідного об'єму. Наприклад, типовий фільтр на першому шарі ConvNet може мати розмір 5x5x3 (тобто 5 пікселів завширшки і заввишки, і 3 - тому що зображення має глибину 3, колірні канали). Під час прямого проходу

ми ковзаємо (точніше, згортаємо) кожен фільтр по ширині і висоті вхідного об'єму і обчислюємо добутки точок між входами фільтра і вхідним об'ємом у будь-якій позиції. Пересуваючи фільтр по ширині та висоті вхідного об'єму, ми створюємо двовимірну карту активації, яка показує реакції цього фільтра в кожній просторовій позиції. Інтуїтивно мережа запам'ятає фільтри, які активуються, коли вона бачить якусь візуальну особливість, наприклад, край певної орієнтації або пляму певного кольору на першому шарі, або, врешті-решт, цілі стільникові або колесоподібні патерни на вищих шарах мережі. Тепер ми матимемо цілий набір фільтрів у кожному шарі CONV (наприклад, 12 фільтрів), і кожен з них створюватиме окрему двовимірну карту активації. Ми складемо ці карти активації вздовж виміру глибини і отримаємо вихідний об'єм.

Локальне підключення. При роботі з вхідними даними високої розмірності, такими як зображення, як ми бачили вище, непрактично з'єднувати нейрони з усіма нейронами в попередньому об'ємі. Замість цього ми з'єднаємо кожен нейрон лише з локальною областю вхідного об'єму. Просторова протяжність цієї зв'язності є гіперпараметром, який називається рецептивним полем нейрона (еквівалентно розміру фільтра). Протяжність зв'язку вздовж осі глибини завжди дорівнює глибині вхідного об'єму. З'єднання є локальними у двовимірному просторі (по ширині та висоті), але завжди повними по всій глибині вхідного об'єму.

Просторове розташування. Ми пояснили зв'язок кожного нейрона шару Conv з вхідним об'ємом, але ми ще не обговорили, скільки нейронів є у вихідному об'ємі і як вони розташовані. Три гіперпараметри контролюють розмір вихідного об'єму: глибина, крок і нульове заповнення:

1) По-перше, глибина вихідного об'єму є гіперпараметром: вона відповідає кількості фільтрів, які ми хотіли б використати, кожен з яких навчається шукати щось своє на вході. Наприклад, якщо перший згортковий шар отримує на вході сире зображення, то різні нейрони вздовж виміру

глибини можуть активуватися за наявності різних орієнтованих країв або кольорових плям. Ми будемо називати набір нейронів, які дивляться на одну і ту ж область вхідного зображення, стовпчиком глибини (дехто також віддає перевагу терміну "волокно").

2) По-друге, ми повинні вказати крок, з яким ми пересуваємо фільтр. Якщо крок дорівнює 1, ми пересуваємо фільтри по одному пікселю за раз. Якщо крок дорівнює 2 (або рідко 3 і більше, хоча на практиці це трапляється рідко), то фільтри пересуваються на 2 пікселі за раз, коли ми їх пересуваємо. Це призведе до менших просторових об'ємів вихідного зображення.

3) Іноді буває зручно доповнити вхідний об'єм нулями навколо межі. Розмір цього нульового заповнення є гіперпараметром. Приємною особливістю нульового заповнення є те, що воно дозволяє нам контролювати просторовий розмір вихідних об'ємів (найчастіше, ми будемо використовувати його для точного збереження просторового розміру вхідного об'єму, щоб ширина і висота вхідного і вихідного об'ємів були однаковими).

Ми можемо обчислити просторовий розмір вихідного об'єму як функцію розміру вхідного об'єму (W), розміру рецептивного поля нейронів шару Conv (F), кроку, з яким вони застосовуються (S), та кількості нульового заповнення (P) на межі. Правильна формула для обчислення кількості нейронів, що "поміщаються", має вигляд (W-F+2P)/S+1. Наприклад, для входу 7x7 і фільтра 3x3 з кроком 1 і pad 0 ми отримаємо вихід 5x5. З кроком 2 ми отримаємо вихід 3x3.

Підсумок. Підсумовуючи, шар згортки:

- Приймає об'єм розміром W1×H1×D1
- Вимагає чотирьох гіперпараметрів:
    1) Кількість фільтрів K,
    2) їх просторова протяжність F,
    3) крок S,
    4) кількість нульового заповнення P.

- Створюється об'єм розміром W2×H2×D2 де
    1) W2=(W1-F+2P)/S+1
    2) H2=(H1-F+2P)/S+1 (тобто ширина і висота обчислюються однаково за симетрією)
    3) D2=K
- Спільне використання параметрів вводить F·F·D1 ваг на фільтр, загалом (F·F·D1)·K ваг і K зсувів.
- У вихідному об'ємі d-й зріз глибини (розміром W2×H2) є результатом виконання коректної згортки d-го фільтра з вхідним об'ємом з кроком S а потім зміщений d-м зсувом.

Загальноприйняте налаштування гіперпараметрів має вигляд F=3, S=1, P=1. Однак існують загальні домовленості та емпіричні правила, які мотивують ці гіперпараметри..

Демонстрація згортки [13]. Нижче на рис. 2.3 наведено демонстраційну версію шару CONV. Оскільки 3D-об'єми важко візуалізувати, всі об'єми (вхідний об'єм (синім), вагові об'єми (червоним), вихідний об'єм (зеленим)) візуалізуються з кожним зрізом глибини, складеним у рядки. Вхідний об'єм має розміри W1=5, H1=5, D1=3 а параметри шару CONV K=2, F=3, S=2, P=1. Тобто ми маємо два фільтри розміром 3×3 і вони накладені з кроком 2. Таким чином, розмір вихідного об'єму має просторовий розмір (5 - 3 + 2)/2 + 1 = 3. Крім того, зауважте, що до вхідного об'єму додано прокладку P=1 застосовується до вхідного об'єму, що робить зовнішню межу вхідного об'єму нульовою.

Реалізація у вигляді множення матриці. Зауважимо, що операція згортки по суті виконує скалярний добуток між фільтрами та локальними областями вхідних даних.

Шар об'єднання (пулінгу).

Зазвичай між послідовними шарами Conv в архітектурі ConvNet періодично вставляється шар об’єднання. Його функція полягає в

поступовому зменшенні просторового розміру представлення, щоб зменшити кількість параметрів і обчислень у мережі, а отже, також контролювати переналаштування рис. 2.4.

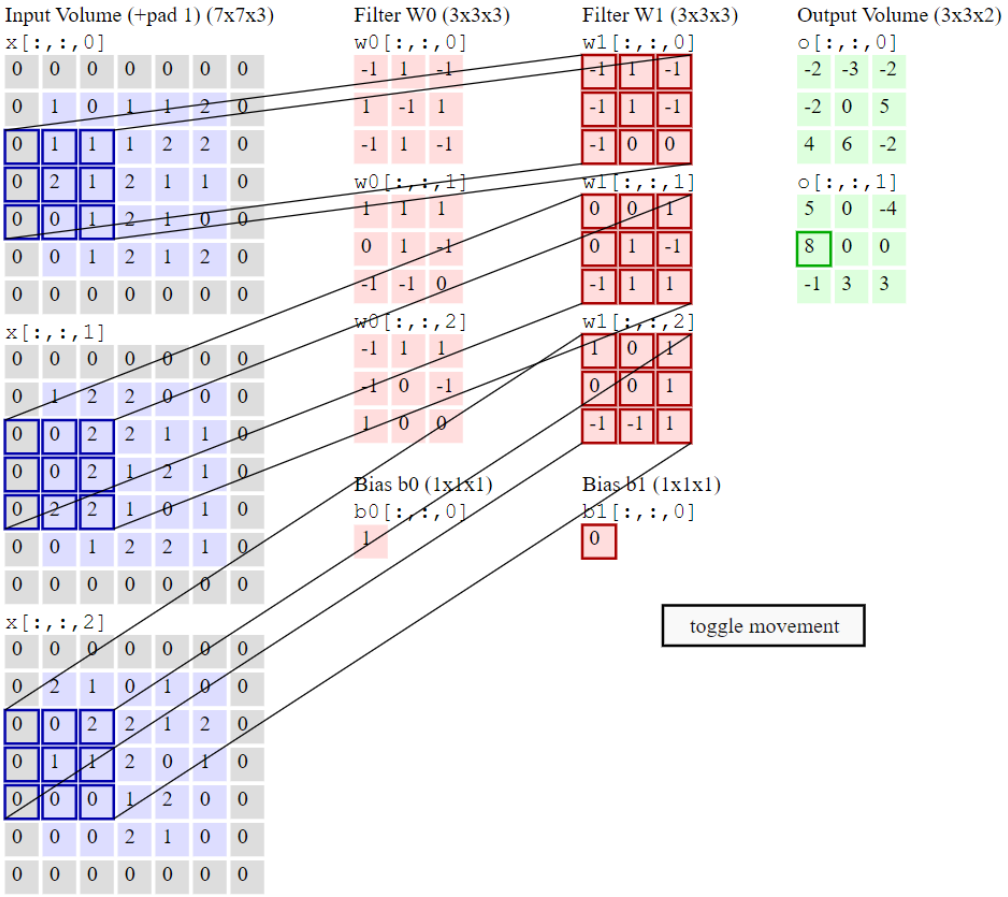

Рисунок 2.3 – Демонстрація виконання операціїзгортки

Шар об’єднання працює незалежно з кожним фрагментом глибини вхідних даних і змінює його просторовий розмір за допомогою операції MAX. Найпоширенішою формою є шар об’єднання з фільтрами розміром 2x2, застосованими з кроком у 2, що зменшує кожен фрагмент глибини у вхідних даних в 2 рази по ширині та висоті, відкидаючи 75% активацій. Кожна операція MAX у цьому випадку прийматиме максимум понад 4 числа (невелика область 2x2 у певній глибині фрагмента). Розмір глибини залишається незмінним. Загалом, шар об’єднання:

- Приймає об’єм розміром W1×H1×D1;

- Потрібні два гіперпараметри:
  - їх просторова протяжність F,
  - крок S;
- Утворює об’єм розміром W2×H2×D2 де:
  - W2=(W1−F)/S+1,
  - H2=(H1−F)/S+1,
  - D2=D1;
- Вводить нульові параметри, оскільки обчислює фіксовану функцію вхідних даних;
- Для шарів об’єднання не прийнято доповнювати вхід за допомогою нульового доповнення.

Варто зазначити, що на практиці існує лише два типові варіанти максимального рівня об’єднання: шар об’єднання з F=3, S=2 (також називається об’єднанням, що перекривається), і частіше F=2, S=2. Об’єднання розмірів із більшими рецептивними полями є надто руйнівним.

Загальне об'єднання. На додаток до максимального об’єднання блоки об’єднання також можуть виконувати інші функції, наприклад об’єднання середнього значення або навіть об’єднання за нормою L2. Раніше об’єднання середніх значень часто використовувалося, але останнім часом воно втратило популярність порівняно з операцією об’єднання максимальних значень, яка на практиці показала свою ефективність.

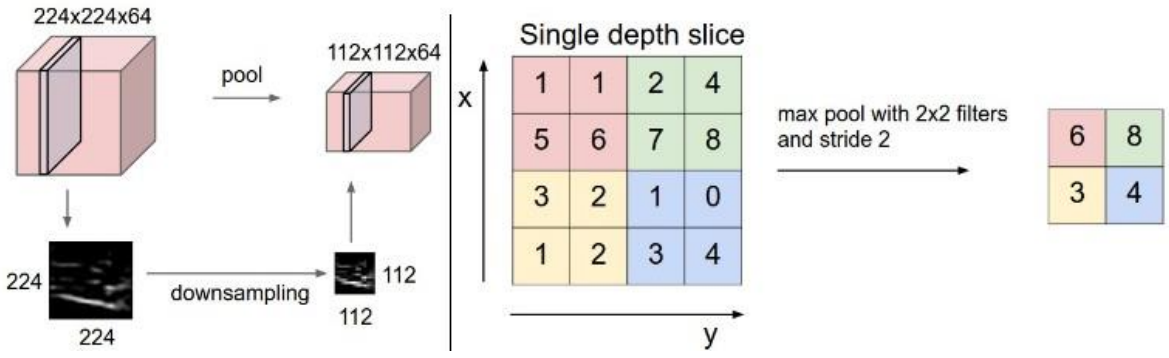

Рисунок 2.4 –Виконання об’єднання (пулінгу)

На рис. 2.4 показано, що шар об'єднання зменшує об'єм просторово, незалежно в кожному зрізі глибини вхідного об'єму. Ліворуч: У цьому прикладі вхідний об'єм розміром [224x224x64] об'єднується з фільтром розміром 2, кроком 2, у вихідний об'єм розміром [112x112x64]. Зверніть увагу, що глибина об'єму зберігається. Праворуч: Найпоширенішою операцією зменшення об'єму є max, що призводить до максимального об'єднання, тут показано з кроком 2. Тобто, кожен max береться від 4 чиесел (маленький квадрат 2x2).

Шар нормалізації.

Багато типів шарів нормалізації було запропоновано для використання в архітектурах ConvNet, іноді з наміром реалізації схем гальмування, які спостерігаються в біологічному мозку. Однак з тих пір ці шари втратили прихильність, оскільки на практиці було показано, що їхній внесок є мінімальним, якщо взагалі є.

Повнозв'язаний шар.

Нейрони в повнопов’язаному шарі мають повні зв’язки з усіма активаціями на попередньому рівні, як це видно у звичайних нейронних мережах. Таким чином, їх активацію можна обчислити за допомогою множення матриці з подальшим зміщенням.

Перетворення повнозв'язаних шарів на згорткові шари.

Варто зазначити, що єдина різниця між шарами FC і CONV полягає в тому, що нейрони в шарі CONV підключені лише до локальної області на вході, і що багато нейронів у тому CONV мають спільні параметри. Однак нейрони в обох шарах все ще обчислюють скалярний добуток, тому їх функціональна форма ідентична. Таким чином, виявляється, що можна конвертувати шари FC у CONV:

Архітектури ConvNet.

Згорткові мережі зазвичай складаються лише з трьох типів шарів: CONV, POOL (припускається максимальний пул, якщо не вказано інше) і FC (скорочення від fully-connected). Також можна вважати функцію активації RELU як шар, який застосовує поелементну нелінійність. Розглянемо як вони зазвичай складаються разом, щоб утворити цілі ConvNets.

Шаблони шарів.

Найпоширеніша форма архітектури ConvNet складається з кількох шарів CONV-RELU, після яких йдуть шари POOL і повторюється цей шаблон, доки зображення не буде об'єднано просторово до невеликого розміру. У якийсь момент прийнято переходити до повнозв'язних шарів. Останній повнозв'язний шар містить вихідні дані, наприклад оцінки класу. Іншими словами, найпоширеніша архітектура ConvNet відповідає шаблону:

INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

де * вказує на повторення, а POOL? вказує на додатковий рівень об'єднання. Крім того, N >= 0 (і зазвичай N <= 3), M >= 0, K >= 0 (і зазвичай K < 3). Наприклад, ось деякі поширені архітектури ConvNet, які дотримуються цього шаблону:

- INPUT -> FC, реалізує лінійний класифікатор. Тут N = M = K = 0.
- INPUT -> CONV -> RELU -> FC
- INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC. Тут ми бачимо, що між кожним шаром POOL є один шар CONV.
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC. Тут ми бачимо два шари CONV, розташовані перед кожним шаром POOL. Це, як правило, гарна ідея для великих і глибоких мереж, оскільки кілька складених рівнів CONV можуть розвинути більш складні характеристики вхідного обсягу перед деструктивною операцією об'єднання.

Таким чином, в згорткових нейромережах існує можливість створювати їх різні структури шляхом комбінування різних наборів трьох типів шарів – згорткового, пулінгового та повнозв'язного.

2.2 Розробка структури згорткової нейронної мережі для розпізнавання мови жестів

У сфері згорткових мереж існує кілька архітектур, які мають назву. Найпоширенішими є:

- LeNet. Перші успішні програми згорткових мереж були розроблені Янном Лекуном у 1990-х роках. З них найвідомішою є архітектура LeNet, яка використовувалася для читання поштових індексів, цифр тощо.
- AlexNet. Першою роботою, яка популяризувала згорткові мережі в комп'ютерному баченні, була AlexNet, розроблена Алексом Крижевським, Іллею Суцкевером і Джеффом Гінтоном. AlexNet було подано на конкурс ImageNet ILSVRC у 2012 році та значно перевершило друге місце (помилка 5 найпопулярніших 16% у порівнянні з другим із помилкою 26%). Мережа мала архітектуру, дуже схожу на LeNet, але була глибшою, більшою та містила згорткові шари, накладені один на одного (раніше зазвичай було мати лише один рівень CONV, за яким слідував шар POOL).
- ZF Net. Переможцем ILSVRC 2013 стала Convolutional Network (Згорточна мережа) від Метью Зейлера та Роба Фергуса. Він став відомий як ZFNet (скорочення від Zeiler & Fergus Net). Це було вдосконалення AlexNet шляхом налаштування гіперпараметрів архітектури, зокрема шляхом розширення розміру середніх згорткових шарів і зменшення кроку та розміру фільтра на першому шарі.
- GoogLeNet. Переможцем ILSVRC 2014 стала Convolutional Network від Szegedy et al. від Google. Його основним внеском стала розробка початкового модуля, який різко зменшив кількість параметрів у мережі (4M, порівняно з AlexNet з 60M). Крім того, у цьому документі використовується Average Pooling замість Fully Connected шарів у верхній частині ConvNet, усуваючи велику кількість параметрів, які, здається, не мають особливого

значення. Існує також кілька наступних версій GoogLeNet, остання з яких Inception-v4.

- VGGNet. Друге місце в ILSVRC 2014 посіла мережа Карена Симоняна та Ендрю Зіссермана, яка стала відомою як VGGNet. Його основний внесок полягав у тому, щоб показати, що глибина мережі є критично важливим компонентом для високої продуктивності. Їх остання найкраща мережа містить 16 рівнів CONV/FC і, що привабливо, має надзвичайно однорідну архітектуру, яка виконує лише згортки 3x3 і об'єднання 2x2 від початку до кінця. Їх попередньо навчена модель доступна для використання в Caffe. Недоліком VGGNet є те, що він дорожчий для оцінки та використовує набагато більше пам'яті та параметрів (140 МБ). Більшість цих параметрів знаходяться на першому повнозв'язному шарі, і з тих пір було виявлено, що ці повнозв'язні шари FC можна видалити без зниження продуктивності, що значно зменшує кількість необхідних параметрів.

- ResNet. Залишкова (residual) мережа, розроблена Kaiming He та ін. був переможцем ILSVRC 2015. Він відрізняється спеціальними з'єднаннями пропуску та інтенсивним використанням пакетної нормалізації. В архітектурі також відсутні повнозв'язні шари в кінці мережі.

В цій роботі для вирішення задачі розпізнавання мови жестів будемо використовувати власні структури згорткових нейронних мереж, але побудовані на основі структури. VGG16. Тому давайте розглянемо структуру та архітектуру згорткової нейронної мережі VGG16, яка представлена на рис. 2.5.

VGG — це не окрема модель, а сімейство моделей, які є подібними, але мають різні конфігурації. Кожна конфігурація визначає кількість шарів та розмір кожного шару. Конфігурації наведено в таблиці 2.1 та позначено літерами, хоча останнім часом їх просто називають кількістю шарів з вагами в моделі, наприклад, конфігурація "A" має 11 шарів з вагами, тому вона відома як VGG11.

Таблиця 2.1 – Конфігурації різних модифікацій згорткової нейромережі VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Нижче на рис. 2.5 наведено архітектуру конфігурації "D", також відомої як VGG16, для кольорового зображення 224x224.

Вхідні дані для шару conv1 – це зображення RGB фіксованого розміру 224 x 224. Зображення проходить через стек згорткових (конволюційних) шарів, де використовуються фільтри з дуже малим рецептивним полем: 3×3 (що є найменшим розміром для врахування поняття ліворуч/праворуч, вгору/вниз, центр). В одній з конфігурацій також використовуються згорткові фільтри 1×1, які можна розглядати як лінійне перетворення вхідних каналів (з

подальшою нелінійністю). Крок згортки фіксований на рівні 1 пікселя; просторове доповнення вхідних даних шару conv. таке, що просторова роздільна здатність зберігається після згортки, тобто доповнення становить 1 піксель для шарів conv. 3×3. Просторове об'єднання здійснюється п'ятьма шарами максимального об'єднання, які йдуть після деяких шарів conv. (не всі шари conv. мають максимальне об'єднання). Макс-пул виконується у вікні розміром 2×2 пікселя з кроком 2.

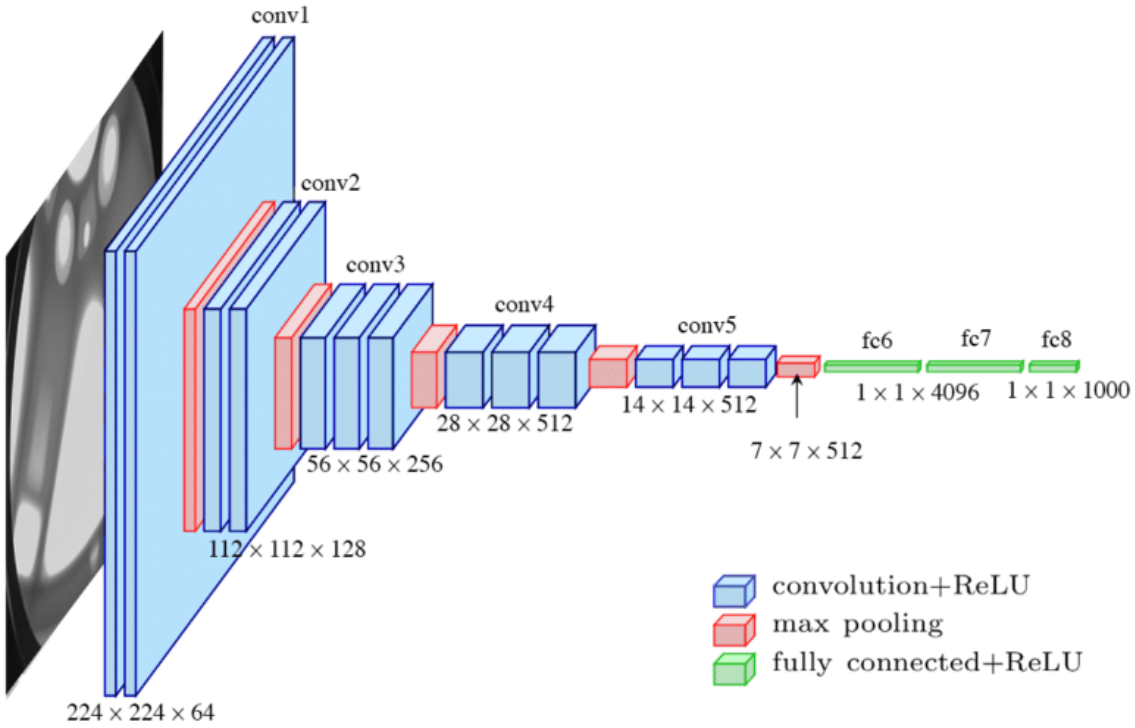

Рисунок 2.5 – Структура згорткової нейронної мережі VGG16

Три повністю зв'язані (FC) шари слідують за стеком згорткових шарів (який має різну глибину в різних архітектурах): перші два мають по 4096 каналів кожен, третій виконує 1000-канальну класифікацією ILSVRC і, таким чином, містить 1000 каналів (по одному для кожного класу). Останній шар - це Soft-max шар. Конфігурація повністю зв'язаних шарів однакова у всіх мережах.

Усі приховані шари оснащені нелінійністю випрямлення (ReLU). Також зазначається, що жодна з мереж (крім однієї) не містить локальної нормалізації

відгуку (LRN), така нормалізація не покращує продуктивність набору даних ILSVRC, але призводить до збільшення споживання пам'яті та часу обчислення.

Інші поширені варіанти VGG - це VGG11, VGG13 та VGG19, які відповідають конфігураціям "A", "B" та "E" у табл. 2.1. Конфігурації "A-LRN" та "C" - такі ж, як "D", але з меншими розмірами фільтрів у деяких згорткових шарах - використовуються рідко.

Часто для різних задач обробки зображень використовують попередньо навчені моделі.

Зазвичай початкові ваги мережі ініціалізують випадковим чином - дотримуючись певної схеми ініціалізації ваг - а потім навчають модель. Використання попередньо навченої моделі означає, що деякі - потенційно всі - ваги моделі не ініціалізуються випадковим чином, а беруться з копії моделі, яка вже була навчена на певному завданні. Завдання, на якому модель була попередньо навчена, не обов'язково має відповідати поточному завданню, на якому користувач хоче використовувати попередньо навчену модель. Наприклад, модель, яка була навчена класифікувати зображення, потім може бути використана для виявлення об'єктів на зображенні.

Теорія полягає в тому, що ці попередньо навчені моделі вже вивчили високорівневі ознаки зображень, які будуть корисними для іншого завдання. Це означає, що не потрібно вивчати їх з нуля, коли ми використовуємо попередньо навчену модель для нашого завдання, що призводить до швидшої збіжності нашої моделі. Ми також можемо розглядати попередньо навчену модель як дуже хороший набір ваг для ініціалізації нашої моделі, і використання попередньо навчених моделей зазвичай призводить до покращення продуктивності порівняно з випадковою ініціалізацією наших ваг.

Акт використання попередньо навченої моделі зазвичай відомий як трансферне навчання, оскільки ми вчимося переносити знання з одного

завдання на інше. Його також називають тонким налаштуванням, оскільки ми точно налаштовуємо наші параметри, навчені на одному завданні, на нове, наступне завдання. Терміни "трансферне навчання" та "точне налаштування" використовуються взаємозамінно в машинному навчанні.

Також іноді застосовується техніка, яка називається дискримінативним тонким налаштуванням, яку спочатку запровадили для покращення трансферного навчання класифікації тексту, але потім використовували для завдань комп'ютерного зору.

2.4 Розробка алгоритму роботи програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі

Відповідно до мети роботи та постановки задачі було розроблено алгоритм програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі, представлений на рис. 2.6.

Першим кроком в програмному модулі розпізнавання мови жестів буде імпорт необхідних модулів і пакетів (блок 1). Це такі бібліотеки як TensorFlow, Keras, NumPy, Pandas, Matplotlib. А другим кроком буде підготовка даних: зчитування CSV-файлу (_sign_mnisttrain.csv) з навчальною вибіркою за допомогою Pandas та перемішування всіх навчальних даних (блок 2).

Далі переходимо до нормалізації та пакетної обробки (блок 3). Нормалізація вхідних даних (діапазон яскравостей 0…255 приводиться до діапазона 0…1) важлива для швидшого збігатння алгоритму навчання. А групування навчальних даних у пакети зменшує час, необхідний для навчання моделі.

Потім відбувається бінаризація міток (блок 4) - номер класа кодується одинично-позиційним кодом). Після цього йде відокремлення (блок 5) від навчальних прикладів даних валідації (перевірки). Дані валідації допоможуть контролювати якість навчання на кожній епосі.

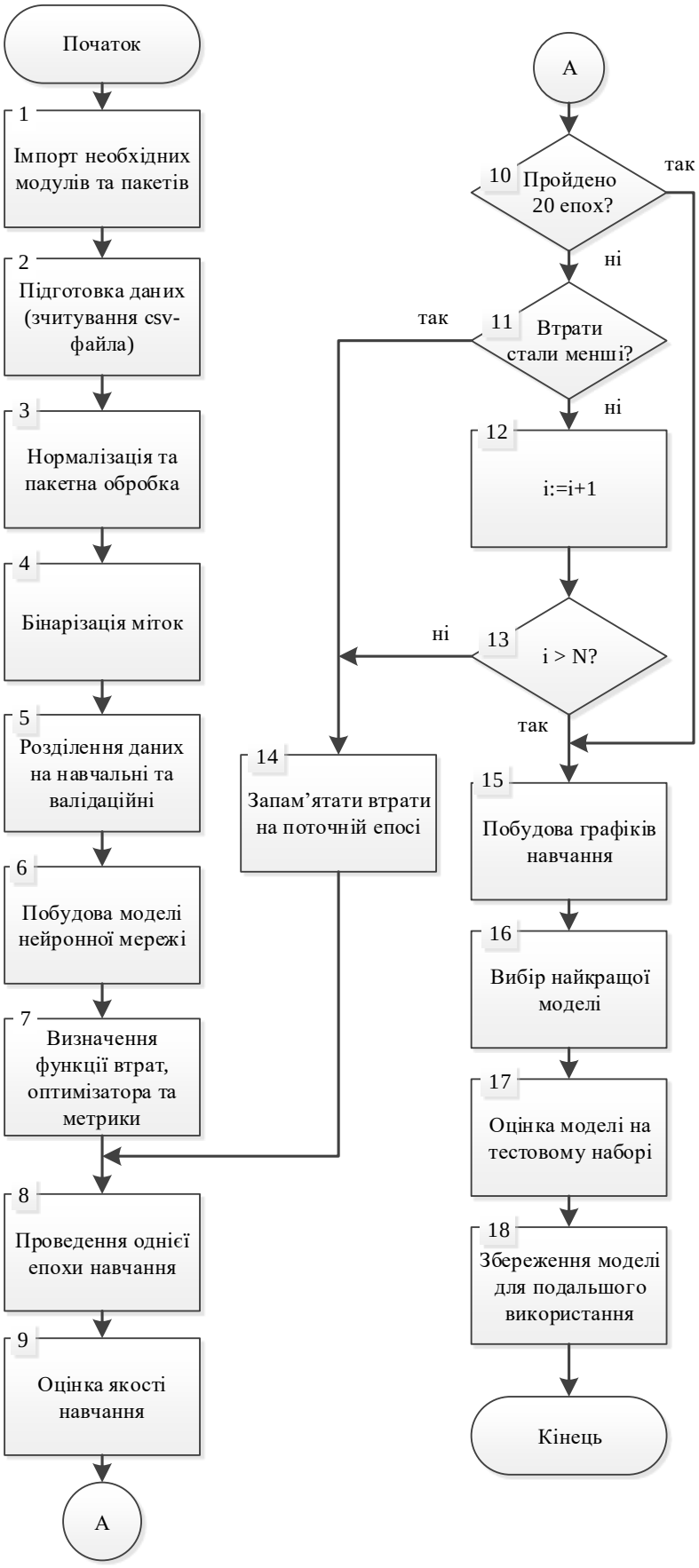

Рисунок 2.6 – Схема алгоритму роботи програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі

Далі виконуємо побудову моделі (блок 6) згорткової нейронної мережі, тобто її структури. Потім визначаємо вид функції втрат, вид оптимізатора (метод навчання) та метрику, яка використовується для оцінки достовірності навчання (блок 7).

Після цтого проводимо одну епоху навчання (блок 8), після якої на валідаційному наборі згідно обраної функції втрат та метрики визначаємо якість процесу навчання (блок9). Умовний блок 10 визначає чи вже проведено потрібну кількість епох навчання (у даному випадку 20). Якщо так, то ідем на блок 15, а якщо ні, то дивимось чи втрати (похибка) навчання стали менші відносно попередньої епохи (блок 11). Якщо втрати зменшились, то запам’ятовуємо їх значення (блок 14) та проводимо наступну епоху навчання. Якщо втрати збільшились , то значення лічильника і збільшуємо на 1 (блок 12). Сукупність блока 12 і 13 дозволяє нам закінчити процес навчання раніше, якщо протягом N епох підряд похибка навчання не збільшується. Це дозволяє уникнути процесу перенавчання нейромережі, коли похибка від епохи до епохи не зменшується, а збільшується.

Коли навчання завершено, будуються графіки залежності похибки і метрики якості навчання від номеру епохи навчання (блок 15). На основі цих графіків обирається епоха, на якій модель нейромережі мала найкращі показники похибки і достовірності навчання і її ваги запам’ятовуються як найкраща модель (блок 16).

Далі проводиться оцінка якості моделі на тестовому наборі даних (блок 17) шляхом визначення втрат і метрики достовірності. Ці параметри на тестовому наборі характеризують наскількі достовірно модель нейромережі розпізнає невідомі зображення. Після цього отримана навчена нейромережа зберігається (блок 18) для подальшого використання при розпізнавання мови жестів.

Згідно розробленого алгоритму у п.3.3 роботи описано програмну реалізацію модуля розпізнавання мови жестів на основі згорткової нейронної мережі.

2.5 Висновок до розділу 2

У розділі було проаналізовано роботу згорткової нейронної мережі для розпізнавання мови жестів. Із таких проаналізованих архітектур: LeNet, AlexNet, ZF Net, GoogLeNet, VGGNet, ResNet, було обрано згорткову нейромережу VGGNet як найбільш перспективну. для вирішення задачі розпізнавання мови жестів. Але було прийнято рішення використовувати власні структури згорткових нейронних мереж, але побудовані на основі структури VGG16. Також було розроблено алгоритм роботи програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору мови програмування та спеціалізованих бібліотек

Для реалізації програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі було обрано мову програмування Python [21].

Python — це мова програмування, яка широко використовується в інтернет-додатках, розробці програмного забезпечення, науці про дані і машинному навчанні (ML). Розробники використовують Python, тому що він ефективний, простий у вивченні та працює на різних платформах. Програми мовою Python можна завантажити безкоштовно, вони сумісні з усіма типами систем та підвищують швидкість розробки.

Мова Python має такі переваги:

– Розробники можуть легко читати та розуміти програми на Python, оскільки мова має базовий синтаксис, схожий на синтаксис англійської.

– Python допомагає розробникам бути більш продуктивними, оскільки вони можуть писати програми на Python, використовуючи менше рядків коду, ніж іншими мовами.

– Python має велику стандартну бібліотеку, що містить багаторазові коди практично для будь-якого завдання. В результаті розробникам не потрібно писати код із нуля.

– Розробники можуть легко поєднувати Python з іншими популярними мовами програмування: Java, C та C++.

– Активна спільнота Python складається з мільйонів розробників з усього світу. У разі виникнення проблем співтовариство допоможе в їх вирішенні.

– Крім того, в Інтернеті є безліч корисних ресурсів для вивчення Python. Наприклад, відеоролики, навчальні посібники, документація та посібники для розробників.

– Python можна переносити на різні операційні системи: Windows, MacOS, Linux і Unix.

Мова Python має кілька стандартних прикладів використання при розробці додатків, серед яких:

- Веб-розробка на стороні сервера.

Веб-розробка на стороні сервера включає складні серверні функції, за допомогою яких веб-сайти відображають інформацію для користувача. Наприклад, веб-сайти повинні взаємодіяти з базами даних та іншими веб-сайтами, а також захищати дані під час їх надсилання по мережі.

Python корисний при написанні серверного коду, оскільки він пропонує безліч бібліотек, що складаються із попередньо написаного коду для складних серверних функцій. Також розробники використовують широкий спектр платформ Python, які надають всі необхідні інструменти для більш швидкого та простого створення інтернет-додатків. Наприклад, розробники можуть створити «скелет» інтернет-програми за лічені секунди, тому що їм не потрібно писати код з нуля. Потім можна протестувати за допомогою інструментів тестування платформи незалежно від зовнішніх інструментів тестування.

- Автоматизація за допомогою скриптів Python.

Мова скриптів — це мова програмування, яка автоматизує завдання, які зазвичай виконують люди. Програмісти широко використовують скрипти Python для автоматизації багатьох повсякденних завдань, серед яких:

1) Одночасне перейменування великої кількості файлів,
2) Перетворення файлу на інший тип файлу,
3) Видалення повторюваних слів у текстовому файлі,
4) Виконання базових математичних операцій,

5) Надсилання повідомлень електронної пошти,
6) Завантаження контенту,
7) Виконання базового аналізу журналів,
8) Пошук помилок у кількох файлах.

- Наука про дані та машинне навчання.

Наука про дані отримує цінну інформацію з даних, а машинне навчання (ML) дозволяє комп'ютерам автоматично вчитися на даних і робити точні прогнози. Фахівці з роботи з даними використовують Python для вирішення наступних завдань:

1) Виправлення та видалення неправильних даних (очищення даних),
2) Вилучення та вибір різних характеристик даних,
3) За допомогою маркування даних можна додавати даним їхні значні імена,
4) Пошук статистичної інформації у даних,
5) Візуалізація даних за допомогою діаграм та графіків: лінійних діаграм, стовпчастих діаграм, гістограм та кругових діаграм.

Фахівці з роботи з даними використовують бібліотеки Python ML для моделей машинного навчання та створення класифікаторів, які точно класифікують дані. Класифікатори на основі Python використовуються в різних областях і застосовуються для виконання таких завдань, як класифікація зображень, тексту та мережевого трафіку, розпізнавання мови та розпізнавання осіб. Фахівці роботи з даними також використовують Python для глибокого навчання, що є передовою технікою машинного навчання.

- Розробка програмного забезпечення.

Розробники програмного забезпечення часто використовують Python для різних завдань розробки та програмних додатків, серед яких:

1) Відстеження помилок у програмному коді,
2) Автоматичне складання програмного забезпечення,

3) Управління програмними проектами,

4) Розробка прототипів програмного забезпечення,

5) Розробка настільних додатків з використанням бібліотек графічного інтерфейсу користувача (ДПІ),

6) Розробка ігор: від простих текстових ігор до складних ігор.

- Автоматизація тестування програмного забезпечення.

Тестування програмного забезпечення — це процес перевірки відповідності фактичних результатів програмного забезпечення очікуваним результатам, який дозволяє переконатися, що програмне забезпечення не містить помилок.

Розробники використовують середовища модульного тестування Python (Unittest, Robot та PyUnit) для тестування написаних функцій.

Тестувальники програмного забезпечення використовують Python для написання прикладів тестів для різних сценаріїв. Наприклад, мова застосовується для тестування інтерфейсу інтернет-додатку, декількох програмних компонентів і нових функцій.

Розробники можуть використовувати кілька інструментів автоматичного запуску тестових скриптів. Ці інструменти відомі як інструменти безперервної інтеграції/безперервного розгортання (CI/CD). Тестувальники та розробники програмного забезпечення використовують інструменти CI/CD для автоматизації процесу тестування. Інструмент CI/CD автоматично запускає тестові скрипти Python і повідомляє про результати тестування щоразу, коли розробники вносять нові зміни коду.

Оскільки ми будемо створювати та навчати штучні нейронні мережі, які використовують дуже багато операцій множення векторів та матриць, то у цій роботі стануть у пригоді такі бібліотек: TensorFlow [23], Keras [23], Sklearn [23], Numpy [23], Pandas [23], Matplotlib [23].

TensorFlow — відкрита програмна бібліотека для машинного навчання, розроблена Google для вирішення завдань побудови та тренування нейронної

мережі з метою автоматичного знаходження та класифікації образів, досягаючи якості людського сприйняття.

Існує три окремі частини, які визначають робочий процес TensorFlow, а саме: попередня обробка даних, побудова моделі та навчання моделі для прогнозування. Фреймворк вводить дані у вигляді багатовимірного масиву, який називається тензорами, та виконує їх двома різними способами. Основний метод полягає у побудові обчислювального графа, який визначає потік даних для навчання моделі. Другий, і часто більш інтуїтивний метод, полягає у використанні швидкого виконання, яке дотримується принципів імперативного програмування та негайно оцінює операції.

Keras — відкрита нейромережна бібліотека, написана мовою Python. Спроєктовану для уможливлення швидких експериментів з мережами глибокого навчання, її зосереджено на тому, щоби вона була зручною в користуванні, модульною та розширюваною.

Keras містить численні втілення широко вживаних нейромережних будівельних блоків, таких як шари, цільові та передавальні функції, оптимізувальники та безліч інструментів для спрощення роботи із зображеннями та текстом, щоби спрощувати кодування, потрібне для написання коду для глибоких нейромереж. Її код розміщено на GitHub, а до форумів спільнотної підтримки належать сторінка питань GitHub та канал Slack.

На додачу до стандартних нейронних мереж, Keras містить підтримку згорткових та рекурентних нейронних мереж. Вона підтримує інші поширені службові шари, такі як виключення, пакетне унормовування та агрегування

Scikit-learn (також відома як sklearn або scikits.learn) — це безкоштовна програмна бібліотека машинного навчання для мови програмування Python, яка надає функціональність для створення та тренування різноманітних алгоритмів класифікації, регресії та кластеризації, таких як лінійна регресія, випадковий ліс, градієнтний бустинг, і працює у зв'язці з бібліотеками NumPy

та SciPy. Scikit-learn є однією з найбільш популярних бібліотек машинного навчання.

NumPy (скорочено від Numerical Python) — це бібліотека з відкритим кодом для мови Python. Вона має такі можливості:

- підтримка багатовимірних масивів (включаючи матриці);
- підтримка математичних функцій високого рівня, які призначені для роботи з багатовимірними масивами.

Pandas — це бібліотека Python, яка використовується для маніпулювання та аналізу даних. Вона надає структури даних, такі як серії та кадри даних, які необхідні для роботи з позначеними або реляційними даними. Pandas широко використовується в науці про дані для аналізу, очищення та маніпулювання даними, і стала фундаментальним інструментом для аналізу реальних даних. Вона дозволяє користувачам виконувати різні операції, такі як: обчислення статистичних показників, визначення кореляцій, обробка відсутніх даних, фільтрація та сортування даних, а також групування та агрегування даних. Бібліотека має відкритий вихідний код та ліцензована за ліцензією BSD. Назва «Pandas» походить від «Panel Data» та «Python Data Analysis».

Matplotlib - комплексна бібліотека для створення статичних, інтерактивних та анімованих візуалізацій на мові Python.

3.2 Опис набору даних зображень мови жестів для навчання та тестування модуля

Датасет MNIST мови жестів [1] відноситься до серії відомих датасетів MNIST, таких як набір зображень MNIST рукописних цифр та набір даних Fashion-MNIST. Датасет MNIST мови жестів дотримується того ж формату CSV з мітками та значеннями пікселів в окремих рядках. База даних літер Американської мови жестів про жести рук представляє багатокласову задачу з 24 класами літер (за винятком J та Z, які вимагають руху).

Формат набору даних побудовано таким чином, щоб максимально відповідати класичному MNIST. Кожен навчальний та тестовий випадок представляє мітку (0-25) як однозначну карту для кожної літери алфавіту A-Z (і немає випадків для 9=J або 25=Z через необхідність рухи жестів). Дані навчання (27 455 зображень) та тестові дані (7172 зображень) приблизно вдвічі менші за стандартний MNIST, але в іншому схожі з рядком заголовка з міткою pixel1,pixel2….pixel784, який представляє одне зображення розміром 28x28 пікселів зі значеннями градації сірого від 0 до 255. Вихідні дані зображення жесту руки представляли кількох користувачів, які повторюють жест на різних фонах. Дані жестової мови MNIST отримані шляхом значного розширення невеликої кількості (1704) кольорових зображень, включених як не обрізані навколо області руки. Для створення нових даних було використано конвеєр зображень на основі ImageMagick, який включав обрізання лише до області руки, масштабування сірого, зміну розміру, а потім створення щонайменше 50+ варіацій для збільшення кількості. Стратегія модифікації та розширення включала фільтри («Mitchell», «Robidoux», «Catrom», «Spline», «Hermite»), а також 5% випадкової пікселізації, +/- 15% яскравості/контрасту та, нарешті, поворот на 3 градуси. Через крихітний розмір зображень ці модифікації ефективно змінюють роздільну здатність та розділення класів цікавими та контрольованими способами.

Надійний алгоритм візуального розпізнавання може забезпечити не лише нові орієнтири, які кидають виклик сучасним методам машинного навчання, таким як згорткові нейронні мережі, але й може прагматично допомогти глухим та слабочуючим людям краще спілкуватися за допомогою програм комп'ютерного зору. Національний інститут глухоти та інших комунікаційних розладів (NIDCD) вказує на те, що 200-річна американська жестова мова є повноцінною, складною мовою (жести літер є лише частиною), але є основною мовою для багатьох глухих.

3.3 Програмна реалізація модуля розпізнавання мови жестів на основі згорткової нейронної мережі

Першим кроком в програмному модулі розпізнавання мови жестів на основі згорткової нейронної мережі буде імпорт необхідних модулів та пакетів

```
from sklearn.preprocessing import LabelBinarizer
from tensorflow import keras
from keras.utils import plot_model

import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
```

Потім іде завантаження навчальних даних:

```
# loading the training data (X+y)
train_df = pd.read_csv('data/alphabet/sign_mnist_train.csv')
```

Далі йде підготовка даних. Зчитування CSV-файлу (_sign_mnisttrain.csv) з навчальною вибіркою за допомогою pandas та перемішування всіх навчальних даних. Розділення пікселів зображення та міток дозволяє нам застосовувати методи попередньої обробки, специфічні для окремих ознак.

```
train_df = pd.read_csv('data/alphabet/sign_mnist_train.csv')
train_df = train_df.sample(frac=1, random_state=42)
X, y = train_df.drop('label', axis=1), train_df['label']
```

Далі йде нормалізація та пакетна обробка. Нормалізація вхідних даних дуже важлива, особливо при використанні для навчання алгоритму

градієнтного спуску, який, ймовірно, швидше буде збігатися при нормалізації даних. Групування навчальних даних у пакети зменшує час, необхідний для навчання моделі.

```
X = X/255.0
X = tf.reshape(X, [-1, 28, 28, 1])
```

Приклад зображення із навчальних даних після застосування попередньої обробки показано на рис. 3.1

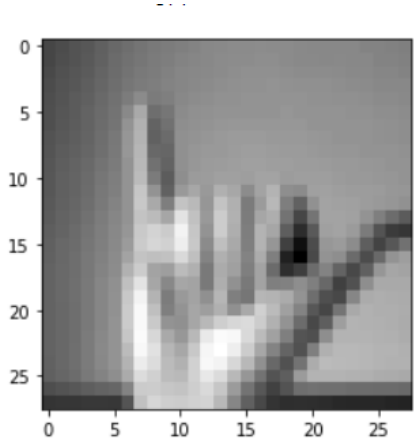

Рисунок 3.1 - Приклад зображення із навчальних даних після застосування попередньої обробки

Далі виконується бінаризація міток, приклад якої показано на рис. 3.2.

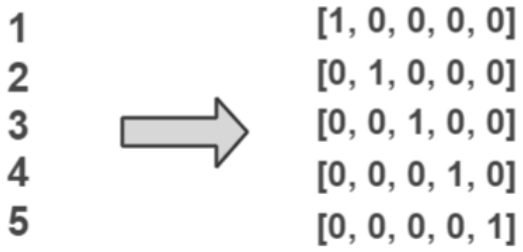

Рисунок 3.2 - Приклад бінаризації міток

LabelBinarizer з бібліотеки Scikit-Learn бінаризує мітки за принципом «один проти всіх» та повертає вектори, закодовані за одним параметром.

```
label_binarizer = LabelBinarizer()
y = label_binarizer.fit_transform(y)
```

Далі йде відокремлення даних валідації (перевірки). Дані валідації допоможуть у виборі найкращої моделі. Якщо ми використовуватимемо тут тестові дані, ми виберемо модель, яка є занадто оптимістичною на основі тестових даних.

```
X_train, X_valid = X[:25000], X[25000:]
y_train, y_valid = y[:25000], y[25000:]
```

Потім виконуємо побудову моделі. Модель згорткової нейронної мережі можна визначити наступним чином. Зазвичай вибір структури згорткової нейронної мережі під час її побудови складається з таких етапів:

– Вибір набору згорткових та пулінгових шарів.
– Збільшення кількості нейронів у більш глибоких згорткових шарах.
– Додавання набору повнозв’язних шарів після згорткових і пулінгових шарів.

```
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(32, (5, 5), padding='same', activation='relu', input_shape=(28, 28
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Conv2D(64, (5, 5), padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Conv2D(128, (5, 5), padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dense(24, activation='softmax'))
```

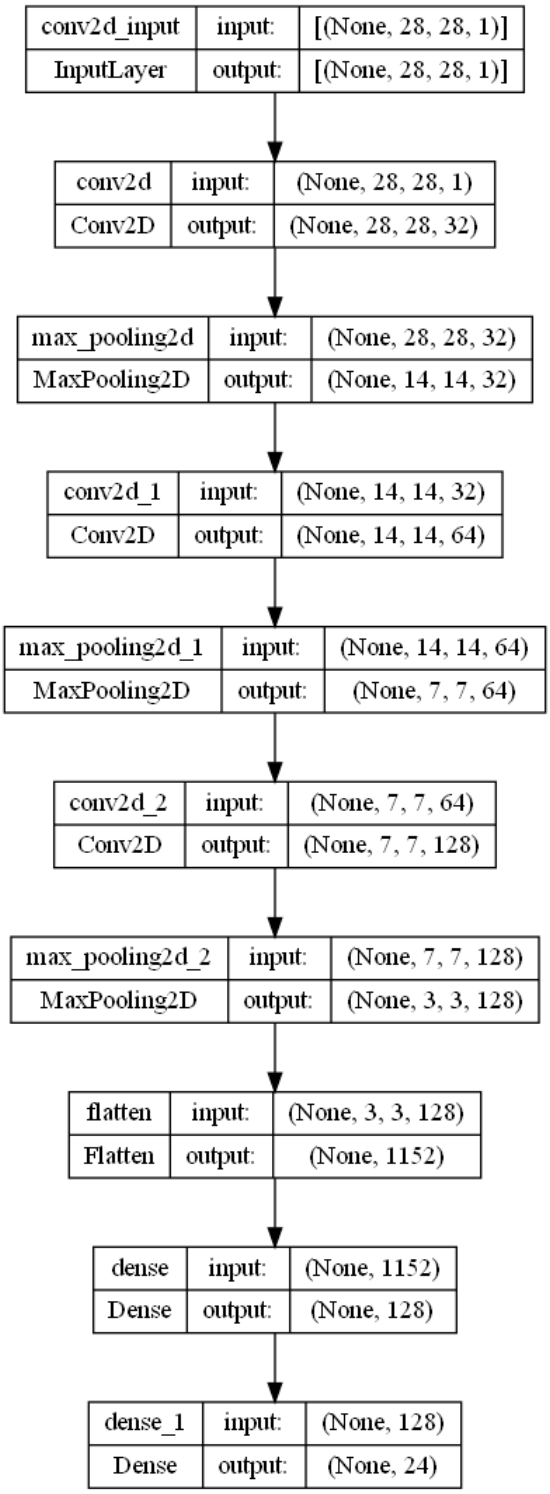

Рисунок 3.3 – Приклад моделі згорткової нейронної мережі

Тепер нам необхідно визначити ключові елементи у рамках побудови моделі:

– loss - Визначає функцію втрат, яку ми прагнемо мінімізувати. Оскільки наші мітки приведені до унітарного вигляду, ми можемо

вибрати як функцію втрат категоріальну перехресну ентропію (categorical cross entropy).

– optimizer - Цей алгоритм знаходить найкращі ваги, які мінімізують функцію втрат. Adam – один із таких алгоритмів, який добре працює в більшості випадків.

Ми можемо вказати будь-які метрики, які будуть задіяні в оцінці нашої моделі. Для побудови нашої моделі ми вибрали достовірність (accuracy).

Визначення функції втрат, оптимізатора та метрик:

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Далі треба організувати запам’ятовування контрольних точок процесу навчання нейромережі.

ModelCheckpoint - Зберігає найкращу модель, знайдену в процесі навчання в кожну епоху, перевіряючи показники моделі на валідаційних (перевірочних) даних.

EarlyStopping - Перериває навчання, коли протягом зазначеної кількості епох перестає спостерігатися прогрес (зменшення похибки мережі). Контрольні точки:

```
save_best_cb = keras.callbacks.ModelCheckpoint('models/initial-end-to-end',
save_best_only = True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience = 5)
```

Пошук патернів. Припасування даних:

```
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid,
y_valid), callbacks=[save_best_cb, early_stopping_cb])
```

Аналіз навчання моделі. Об'єкт history містить відомості про функцію втрат і зазначені метрики, отримані під час навчання моделі. Цю інформацію можна використовувати для формування кривих навчання та доступу до процесу навчання та показників моделі у кожну епоху.

Збереження історії:

```
with open('models/intial-end-to-end-history', 'wb') as history_file:
    pickle.dump(history.history, history_file)
```

Найкраща модель. Тепер нам потрібно виокремити кращу модель, отриману під час навчання, оскільки модель, сформована наприкінці навчання, не обов'язково буде найкращою. Найкраща модель:

```
best_model = keras.models.load_model('models/initial-end-to-end')
```

Показники на тестовому наборі. Запуск на тестовому наборі:

```
def evaluate_model(model, X_test, y_test, label_binarizer):
    X_test_reshape = tf.reshape(X_test, [-1, 28, 28, 1])
    y_test_labels = label_binarizer.transform(y_test)
    results = model.evaluate(X_test_reshape, y_test_labels)
    print(f'Loss: {results[0]:.3f} Accuracy: {results[1]:.3f}')

results = evaluate_model(best_model, test_df.drop('label', axis=1), test_df['label'], label_binarizer)
```

Достовірність: 94%.

3.4 Оптимізація гіперпараметрів згорткової нейронної мережі

Коли справа доходить до оптимізації гіперпараметрів, існує ціла низка варіантів, як можна побудувати нашу згорткову нейронну мережу. Деякі з найбільш важливих гіперпараметрів, які необхідно оптимізувати, включають:

1. Кількість згорткових і пулінгових пар.

Цей гіперпараметр визначає кількість пар Conv2D і MaxPooling2D, які ми об’єднуємо разом у рамках нашої згорткової нейронної мережі.

По мірі того, як ми залучаємо все більше таких пар шарів, мережа стає все глибше і глибше, збільшує здатність моделі ідентифікувати більш складні шаблони зображень.

Але розміщення занадто великої кількості пар шарів негативно відображається на показниках моделей (розміри вхідного зображення починають швидко зменшуватися за тим вимірюванням, як воно поглиблюється в нейронну мережу), а також збільшується час, необхідний для навчання, оскільки кількість налаштовуваних параметрів моделі різко збільшується.

2. Фільтри.

Фільтри визначають кількість вихідних карт ознак. Матричний фільтр як раз і представляє собою операцію згортки, яка дозволяє знайти одні і те ж шаблони в різних частинах зображення. У більшості випадків збільшення кількості фільтрів у послідовних шарах дає хороші результати.

3. Розмір фільтра.

У якості розміру фільтра прийнято брати непарні числа, щоб ми могли отримати центральну позицію. Одна з основних проблем з фільтрами парного розміру полягає в тому, що вони вимагають асимметричного заповнення.

Замість використання одного згорточного шару, що складається з фільтрів більшого розміру, наприклад (7x7, 9x9), ми можемо використовувати кілька згорточних шарів з фільтрами меншого розміру, щоб, скоріше всього, покращити показники моделі, оскільки більш глибокі мережі можуть вичвляти більш складні шаблони.

4. Дропаут.

Дропаут (проріджування або виключення) слугує як регулятор і запобігає перенавчанню моделі. Шар випадання нейронів зводить нанівець

внесок деяких нейронів до наступного шару та залишає інші незмінними. Коефіцієнт дропаута визначає ймовірність того, що вклад конкретного нейрона буде відмінений.

На ранніх епохах ми можемо зіткнутися з тим, що втрати від навчання більші, ніж втрати від валідації, оскільки деякі нейрони можуть бути відключені під час навчання, але для валідації використовується повна мережа з усіма нейронами.

5. Аугментація (розширення) даних

За допомогою розширення даних (data augmentation) ми можемо створювати трохи змінені копії наявних зображень (рис. 3.4) і використовувати їх для моделі навчання. Такі зображення зі зміненою орієнтацією допомагають моделі ідентифікувати об'єкти з різною орієнтацією. Наприклад, ми можемо ввести невеликий поворот, масштабування та зміщення зображень (рис. 3.4).

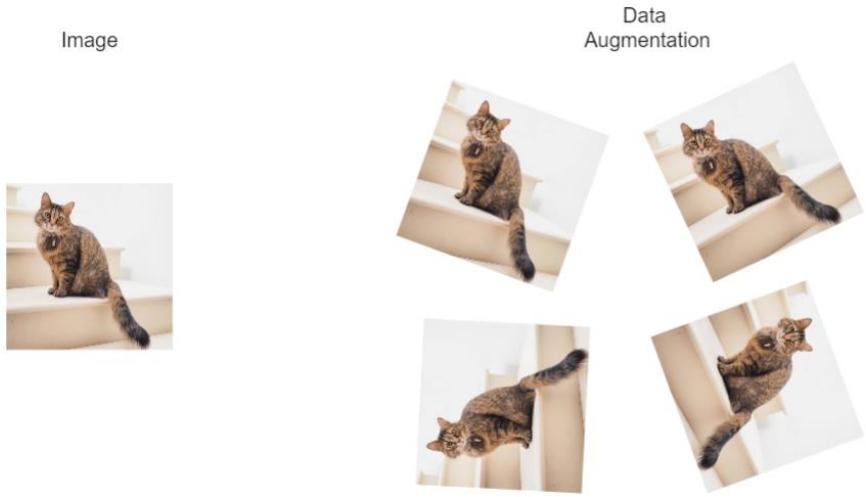

Рисунок 3.4 - Аугментація (розширення) даних

Аугментація (розширення) даних:

```
data_augmentation = keras.models.Sequential()
data_augmentation.add(keras.layers.RandomRotation(0.1, fill_mode='nearest', input_shape=(28, 28, 1)))
data_augmentation.add(keras.layers.RandomZoom((0.15, 0.2), fill_mode='nearest'))
data_augmentation.add(keras.layers.RandomTranslation(0.1, 0.1, fill_mode='nearest'))
```

Інші гіперпараметри, які також можна спробувати налаштувати:

1) Пакетна нормалізація – зміна розміру пакета навчальних даних (вона нормалізує вхідні дані шару),

2) Глибина мережі. Глибші мережі краще працюють. Наприклад, заміна одного шару згортки з розміром фільтра (5X5) двома послідовними шарами згортки з розміром фільтра (3X3),

3) Кількість нейронів у повнозв'язному шарі та кількість повнозв'язних шарів,

4) Заміна шару MaxPooling шаром згортки з кроком > 1,

5) Оптимізатори,

6) Швидкість навчання оптимізатора.

Після лптимізації всіх гіперпараметрів, було отримано кінцеву модель, структура якої наведена на рис. 3.5.

Проведемо оцінку кінцевої моделі:

```
def evaluate_model(model, X_test, y_test, label_binarizer):
    # label_binarizer: Used while preprocessing the train data
    X_test_reshape = tf.reshape(X_test, [-1, 28, 28, 1])
    y_test_labels = label_binarizer.transform(y_test)
    results = model.evaluate(X_test_reshape, y_test_labels)
    print(f'Loss: {results[0]:.3f} Accuracy: {results[1]:.3f}')

best_model = keras.models.load_model('models/experiment-dropout-0/')
evaluate_model(best_model, X_test, y_test, label_binarizer)
```

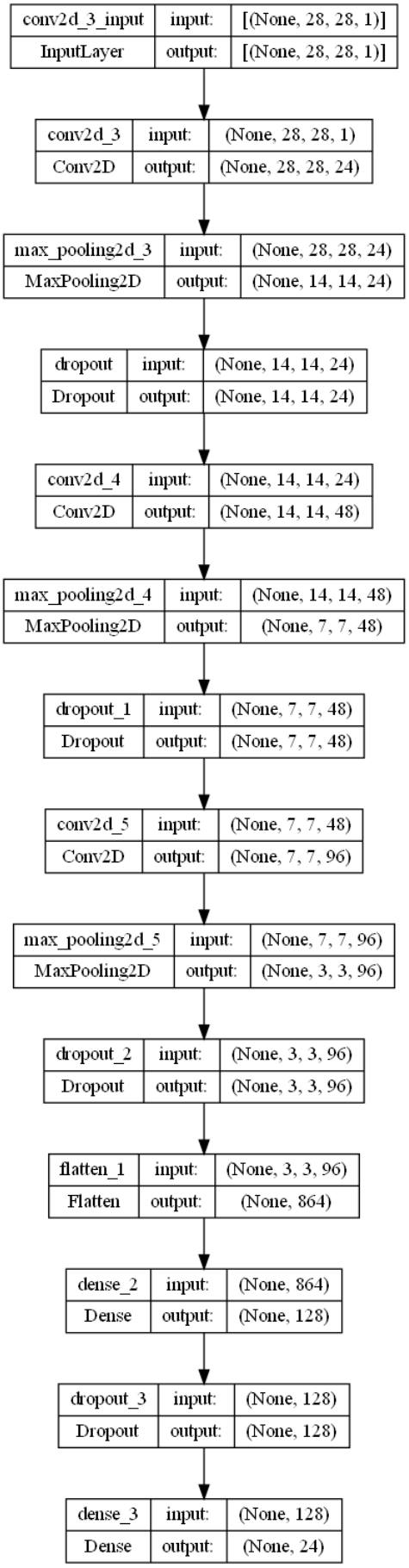

Рисунок 3.5 - Найкраща модель після оптимізації гіперпараметрів

Результати оцінки показали, що кінцева (найкраща) модель має достовірність на тестовій вибірці 96%.

3.5 Висновок до розділу 3

У розділі було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек TensorFlow, Keras, Sklearn, Numpy, Pandas, Matplotlib для програмної реалізації модуля розпізнавання мови жестів на основі згорткової нейронної мережі. Проведено аналіз набору даних мови жестів MNIST (Sign Language MNIST) для навчання та тестування модуля, який містить 27 455 навчальних зображень та 7172 тестових зображення. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання мови жестів на основі згорткової нейронної мережі, що складається з імпорту бібліотек, завантаження набору даних із зображеннями, формування валідаційної вибірки, побудови моделі запропонованої нейромережі, навчання нейромережі, оптимізації нейромережі, тестування результатів роботи розробленої нейромережі.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

4.1 Тестування програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі

Програмний модуль створено як Jupiter Notebook, що дає змогу виконувати код програми послідовно у вигляді окремих блоків коду, спостерігаючи результати виконання кожного блоку на екрані.

У п. 3.3 описано процес програмної реалізації модуля та послідовність блоків коду, із яких складається загальний код програми.

Для тестування програми запускались послідовно ці блоки коду і спостерігались результати їх виконання.

Після завантаження бібліотек, навчальних даних, їх нормалізації, бінарізації міток, побудови моделі нейронної мережі, модуль виводить структуру побудованої ЗНМ у вигляді такої таблиці:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        832

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        51264

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 128)         204928

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 128)        0
 2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 128)               147584

 dense_1 (Dense)             (None, 24)                3096

=================================================================
Total params: 407,704
Trainable params: 407,704
Non-trainable params: 0
```

Потім запускається процес навчання і виводиться для кожної епохи такі параметри: номер епохи/кількість епох, тривалість епохи навчання - step (у секундах і мілісекундах), похибка навчання – loss, достовірність на навчальній вибірці – accuracy, похибка на валідаційній вибірці - val_loss, достовірність на валідаційній вибірці - val_accuracy:

```
Epoch 1/20
782/782 [==============================] - 26s 21ms/step - loss: 0.8774 - acc
uracy: 0.7326 - val_loss: 0.0267 - val_accuracy: 0.9935
Epoch 2/20
782/782 [==============================] - 14s 18ms/step - loss: 0.0051 - acc
uracy: 0.9994 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 3/20
782/782 [==============================] - 12s 16ms/step - loss: 5.9076e-04 -
accuracy: 1.0000 - val_loss: 4.7529e-04 - val_accuracy: 1.0000
...
Epoch 19/20
782/782 [==============================] - 11s 15ms/step - loss: 2.2815e-07 -
accuracy: 1.0000 - val_loss: 2.9071e-07 - val_accuracy: 1.0000
Epoch 20/20
782/782 [==============================] - 11s 15ms/step - loss: 1.5183e-07 -
accuracy: 1.0000 - val_loss: 1.6801e-07 - val_accuracy: 1.0000
```

Історія зміни таких параметрів як похибка навчання – loss, достовірність на навчальній вибірці – accuracy, похибка на валідаційній вибірці - val_loss, достовірність на валідаційній вибірці - val_accuracy записується в пам’ять для того, щоб в подальшому можна було б побудувати графіки їх зміни та вибрати оптимальну модель із кількох досліджуваних:

```
'loss': [0.8774073719978333,
  0.005074513144791126,
  0.0005907623562961817,
  0.035671062767505646,
  0.00012915894330944866,
  6.119009776739404e-05,
  3.5120148822898045e-05,
  2.134285750798881e-05,
  1.3638154086947907e-05,
  8.999433703138493e-06,
  5.808890364278341e-06,
  3.864049176627304e-06,
  2.576162614786881e-06,
```

```
 1.7137577970061102e-06,
 1.1716660992533434e-06,
 7.670482204957807e-07,
 5.091210937280266e-07,
 3.424441104016296e-07,
 2.2814701594597864e-07,
 1.518342855888477e-07],
'accuracy': [0.7325999736785889,
 0.9993600249290466,
 1.0,
 0.9892799854278564,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0],
'val_loss': [0.02665344439446926,
 0.0010351891396567225,
 0.0004752879322040826,
 0.00024525265325792134,
 0.00010227379971183836,
 5.5580920161446556e-05,
 3.293823101557791e-05,
 2.091940950776916e-05,
 1.3808692528982647e-05,
 8.840480404614937e-06,
 5.666528068104526e-06,
 3.9937822293723e-06,
 2.64862956100842e-06,
 1.8130501757696038e-06,
 1.2420367738741334e-06,
 8.05462889275077e-07,
 5.323349228092411e-07,
 4.007931124760944e-07,
 2.907113696437591e-07,
 1.6800919411252835e-07],
'val_accuracy': [0.9934827089309692,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
```

```
1.0,
1.0,
1.0,
1.0,
1.0,
1.0,
1.0,
1.0,
1.0]}
```

На основі історії зміни чотирьох зазначених параметрів програма будує графіки їх зміни, представлені на рис. 4.1

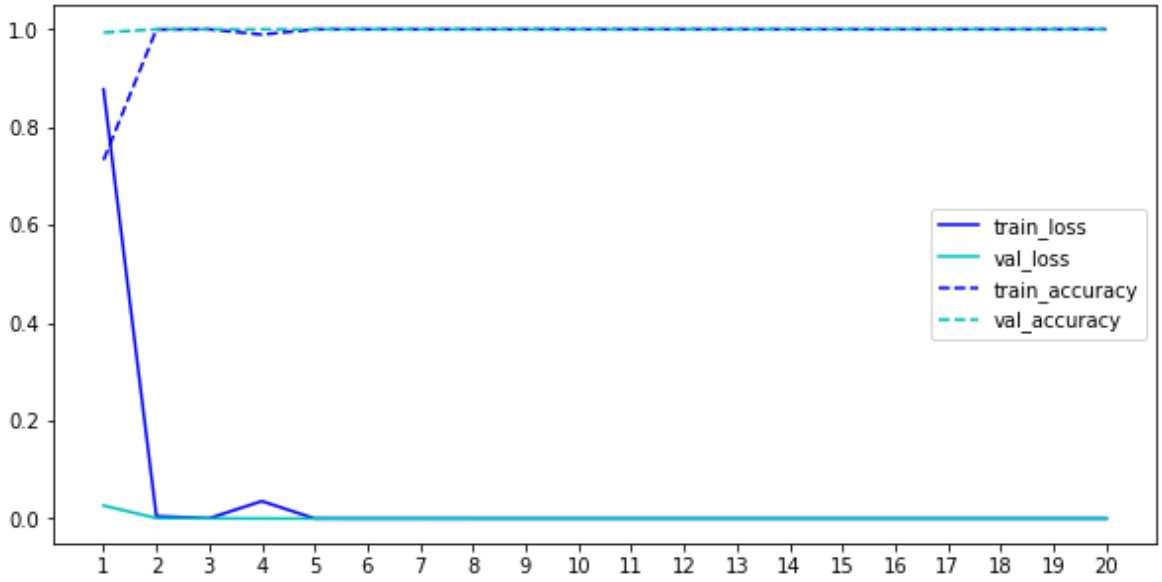

Рисунок 4.1 - Графіки залежності параметрів loss, accuracy, val_loss, val_accuracy від номеру епохи навчання

Тобто процес навчання даного варіанта структури нейромережі завершено. Треба оцінити її продуктивність на тестовій вибірці. Тому далі завантажується навчальна вибірка і проганяється через мережу і після цього виводиться похибка мережі і достовірність її роботи (accuracy) на тестових даних:

```
225/225 [==============================] - 2s 8ms/step - loss: 87.9655 - accu
racy: 0.9405
[87.96553802490234, 0.9404628872871399]
```

Як бачимо, достовірність даного варіанту структури нейромережі Accuracy: 94%.

Далі є блок коду, що дозволяє перетворювати випадковий набір зображень у речення. Наприклад, на рис. 4.2 показано результат розпізнавання послідовності зображень речення "sign language".

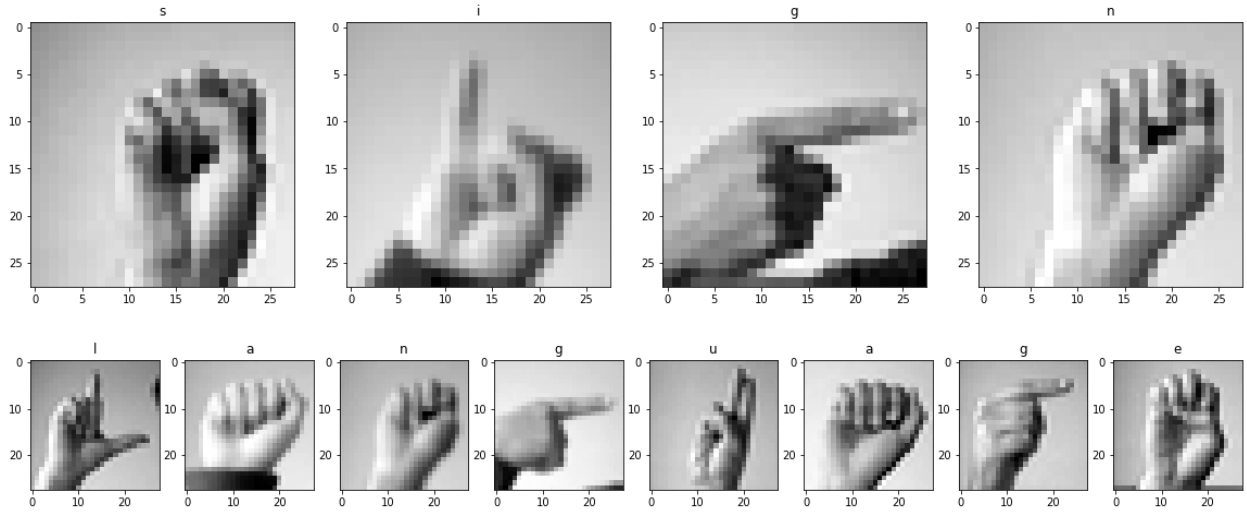

Рисунок 4.2 - Результат розпізнавання послідовності зображень речення "sign language"

При цьому ще виводиться час виконання розпізнавання кожного зображення із послідовності та друкується речення, яке треба було розпізнати і розпізнане речення. Тобто якщо мережа зробила помилку, то це можна буде побачити:

```
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 76ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 44ms/step
The actual sentence is "sign language"
The predicted sentence is "sign language"
```

Потім можна проводити точне наладтування структури згорткової нейромережі шляхом оптимізації її гіперпараметрів. Ми змінювали такі гіперпараметри як Convolution and Max Pooling Pairs, Filters in the convolution layers, Filter Shape, Dropout. Було досліджено 38 варіантів структур нейромереж та виявлено, що найкращою виявилася структура згідно з рис. 3.7. Для цієї найкращої нейромережі були отримані такі результати:

```
225/225 [==============================] - 1s 4ms/step - loss: 0.1247 - accur
acy: 0.9632
Loss: 0.125 Accuracy: 0.963
```

Як бачимо, достовірність найкращого варіанту структури нейромережі Accuracy: 96,3%.

4.2 Аналіз результатів роботи програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі

Метрика оцінки якості роботи програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі.

Як метрику оцінки якості роботи програмного модуля ми будемо використовувати достовірність. Достовірність – це відношення правильно класифікованих екземплярів до їхньої загальної кількості.

Як було зазначено у п. 1.3 за аналоги були взяті роботи [6, 10], де описано програмні реалізації розроблених в них методів. Для порівняння аналогів та розробленого модуля зведемо їхні параметри до табл. 4.1

У табл. 4.1 крім основного показника – достовірність розпізнавання мови жестів також наведено і інші показники. Хоча аналог 1 має кращу достовірність розпізнавання, будемо порівнювати з аналогом 2, оскільки він так само як і розроблений модуль, розпізнає мову жестів американського англійського [21].

Таблиця 4.1 - Порівняння аналогів та розробленого модуля розпізнавання мови жестів

	Вид жестової мови	Вид згорткової нейромережі	Кількість зображень навчальної вибірки	Кількість зображень тестової вибірки	Достовір-ність розпізна-вання, %
Аналог1 [6]	Італійський	2xCNN+ повнозв'язна НМ	4600	2000	91,7
Аналог2 [10]	Американ-ський англійський	Inception v3	1920	480	90,5
Розроб-лений модуль	Американ-ський англійський	Модифікована VGG	27455	7172	96,3

Із табл. 4.1 видно, що достовірність розпізнавання мови жестів на тестовому наборі для запропонованого модуля має величину 96,3%, а для найближчого за функціональним призначенням Аналога 2 [10] – 90,5%. Тобто точність розпізнавання мови жестів розробленого програмного модуля вища на 5,8% (96,3-90,5=5,8), ніж у програми аналога.

Таким чином, можна зробити висновок, що розроблений програмний модуль розпізнавання мови жестів на основі згорткової нейронної мережі має порівняно з аналогом збільшену на 5,8% достовірність розпізнавання мови жестів на тестовому наборі. Тобто мета роботи досягнута – достовірність розпізнавання мови жестів підвищена.

Також у табл. 4.1 наведено такв дані як вид згорткової нейронної мережі, що лежить в основі розробки та кількість зображень мови жестів у навчальній та тестовій вибірках тренувального набору даних.

4.3 Висновок до розділу 4

У розділі в результаті тестування програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі було доведено його працездатність та відповідність поставленому завданню. Було розглянуто інформацію, яку виводить програмний модуль в процесі своєї роботи: структуру згорткової нейромережі, параметри ходу процесу навчання, значення та графіки зміни точності навчання від епохи до епохи, оцінку достовірності нейромережі на тестовому наборі та перетворення послідовності зображень жестової мови у речення. Розроблений програмний модуль розпізнавання мови жестів на основі згорткової нейронної мережі має достовірність розпізнавання 96,3%, що порівняно з аналогом збільшено на 5,8%. Тобто мета роботи досягнута – достовірність розпізнавання мови жестів підвищена.

ВИСНОВКИ

У роботі було розглянуто задачу розпізнавання мови жестів. Було розглянуто 5 методів розпізнавання образів. Як найбільш перспективний для задачі розпізнавання мови жестів було обрано метод на основі штучних нейронних мереж, а згорткові нейромережі визначено як найбільш ефективні для роботи із зображеннями. Крім цього, було обгрунтовано вибір аналогів розробленого модуля розпізнавання мови жестів. Обидва аналоги використовують згорткові нейромережі різних типів і мають достовірність розпізнавання зображень жестової мови відповідно 91,7% та 90,5%, що є недостатнім. Звідси і випливає мета бакалаврської кваліфікаційної роботи - підвищення достовірності розпізнавання мови жестів.

Було проаналізовано роботу згорткової нейронної мережі для розпізнавання мови жестів. Із таких проаналізованих архітектур: LeNet, AlexNet, ZF Net, GoogLeNet, VGGNet, ResNet, було обрано згорткову нейромережу VGGNet як найбільш перспективну. для вирішення задачі розпізнавання мови жестів. Але було прийнято рішення використовувати власні структури згорткових нейронних мереж, але побудовані на основі структури VGG16. Також було розроблено алгоритм роботи програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі.

Було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек TensorFlow, Keras, Sklearn, Numpy, Pandas, Matplotlib для програмної реалізації модуля розпізнавання мови жестів на основі згорткової нейронної мережі. Проведено аналіз набору даних мови жестів MNIST (Sign Language MNIST) для навчання та тестування модуля, який містить 27455 навчальних зображень та 7172 тестових зображення. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання мови жестів на основі згорткової нейронної мережі, що складається з імпорту бібліотек, завантаження набору даних із зображеннями, формування валідаційної

вибірки, побудови моделі запропонованої нейромережі, навчання нейромережі, оптимізації нейромережі, тестування результатів роботи розробленої нейромережі.

У результаті тестування програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі було доведено його працездатність та відповідність поставленому завданню. Було розглянуто інформацію, яку виводить програмний модуль в процесі своєї роботи: структуру згорткової нейромережі, параметри ходу процесу навчання, значення та графіки зміни точності навчання від епохи до епохи, оцінку достовірності нейромережі на тестовому наборі та перетворення послідовності зображень жестової мови у речення. Розроблений програмний модуль розпізнавання мови жестів на основі згорткової нейронної мережі має достовірність розпізнавання 96,3%, що порівняно з аналогом збільшено на 5,8%. Тобто мета роботи досягнута – достовірність розпізнавання мови жестів підвищена

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Sign Language MNIST [Електронний ресурс] – Режим доступу: https://www.kaggle.com/datasets/datamunge/sign-language-mnist

2 Кутковецький В. Я. Розпізнавання образів : навчальний посібник / В. Я. Кутковецький. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2017. – 420 с. ISBN 978-966-336-384-4

3 Руденко О.В. Штучні нейронні мережі: Навчальний посібник / О.В.Руденко, Є.В.Бодянський. - Харків : ТОВ «Компанія СМІТ», 2006. — 404 с. - ISBN 966-8630-73-X.

4 Любунь З. М. Основи теорії нейромереж: текст лекцій / З. М. Любунь. – Львів : Видавничий центр ЛНУ ім. Івана Франка, 2006.

5 Machine Learning with PyTorch and Scikit-Learn . Book Code Repository. Paperback: 770 pages. Publisher: Packt Publishing. Language: English. ISBN-10: 1801819319 I.SBN-13: 978-1801819312

6 Zafar Ahmed Ansari and Gaurav Harit, “Nearest Neighbour Classification of Indian Sign Language Gestures using Kinect Camera”, in Sadhana, Vol. 41, No. 2, February 2016, pp. 161-182

7 Lionel Pigou et al, "Sign Language Recognition using Convolutional Neural Networks", presented at the Ghent University, ELIS, Belgium

8 The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3) [Online]. Available: https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

9 Adrian Rosebrock (2017, March 20). ImageNet: VGGNet, ResNet, Inception, and Xception with Keras [Online] Available: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

10 Das, A., Gawde, S., Suratwala, K., & Kalbande, D. (2018). Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images.

2018 International Conference on Smart City and Emerging Technology (ICSCET). doi:10.1109/icscet.2018.8537248

11 Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261, 2016.

12 Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

13 Alex Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", in NIPS, 2012, pp. 1106â€“1114

14 Python [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Python

15 TensorFlow [Електронний ресурс] – Режим доступу: https://www.tensorflow.org

16 Keras [Електронний ресурс] – Режим доступу: https://keras.io/

17 Scikit-learn. Machine Learning in Python [Електронний ресурс] – Режим доступу: https://scikit-learn.org/stable/

18 Pandas. Python Data Analysis Library [Електронний ресурс] – Режим доступу:https://pandas.pydata.org/

19 NumPy [Електронний ресурс] – Режим доступу: https://numpy.org/

20 Matplotlib: Visualization with Python [Електронний ресурс] – Режим доступу: https://matplotlib.org/

21 Brandon Garcia and Sigberto Viesca, "Real-time American Sign Language Recognition with Convolutional Neural Networks", in Convolutional Neural Networks for Visual Recognition at Stanford University, 2016

22 Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп'ютерні науки» / Укладачі А.А.Яровий, О.В.Сілагін, І.В.Богач. – Вінниця: ВНТУ, 2022. – 47 с.

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль розпізнавання мови жестів на основі згорткової нейронної мережі

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 33,59 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Паночишин Ю.М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Артеменко І.В.
(підпис) (прізвище, ініціали)

Додаток Б (обов'язковий)

Лістинг програми

Фрагмент коду

```
# Importing required libraries

from sklearn.preprocessing import LabelBinarizer
from tensorflow import keras
from keras.utils import plot_model

import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle

# loading the training data (X+y)
train_df = pd.read_csv('data/alphabet/sign_mnist_train.csv')
train_df = train_df.sample(frac=1, random_state=42) # Shuffiling the entire
dataset
X, y = train_df.drop('label', axis=1), train_df['label'] # Split the dataset
into X, y
X.shape, y.shape
np.unique(X.dtypes), y.dtype
label_binarizer = LabelBinarizer() # Binarize labels in a one-vs-all fashion
(return one-hot encoded vectors)

y = label_binarizer.fit_transform(y)
X = X/255.0 # Normalizing the training data and converting the data type to
float
np.unique(X.dtypes)

# Converting the 1-D array of 784 pixels to (28, 28, 1) Image
# (28, 28) represents the spatial dimensions of the image & 1 specifies that
the image is grayscale
X = tf.reshape(X, [-1, 28, 28, 1])
X.shape, y.shape

# Generating a validation set

X_train, X_valid = X[:25000], X[25000:]
y_train, y_valid = y[:25000], y[25000:]
X_train[0].dtype
X_train[0].shape
plt.imshow(X[0], cmap='gray'), y[0]

# Defining the CNN

model = keras.models.Sequential()
model.add(keras.layers.Conv2D(32, (5, 5), padding='same', activation='relu',
input_shape=(28, 28, 1)))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Conv2D(64, (5, 5), padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(keras.layers.Conv2D(128, (5, 5), padding='same',
activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dense(24, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
save_best_cb = keras.callbacks.ModelCheckpoint('models/initial-end-to-end',
save_best_only=True) # Saves the best model so far
early_stopping_cb = keras.callbacks.EarlyStopping(patience=5) # Interupts
training when there is no progress

# The model is same is 'models/initial-end-to-end'
# The history object is 'models/initial-end-to-end-history'

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid,
y_valid), callbacks=[save_best_cb, early_stopping_cb])
history.history # Contains the training related information for each epoch

# Saving the history object

# with open('models/intial-end-to-end-history', 'wb') as history_file:
#     pickle.dump(history.history, history_file)
h = np.load('models/intial-end-to-end-history', allow_pickle=True)
h

best_model = keras.models.load_model('models/initial-end-to-end') # Model
with best set of parameters not necessarily the model at the last epoch of
training

fig, ax = plt.subplots(figsize=(10, 5))
n_epochs = len(h['loss'])
ax.plot(range(1, n_epochs+1), h['loss'], color='b', label='train_loss')
ax.plot(range(1, n_epochs+1), h['val_loss'], color='c', label='val_loss')
ax.plot(range(1, n_epochs+1), h['accuracy'], color='b',
label='train_accuracy', linestyle='--')
ax.plot(range(1, n_epochs+1), h['val_accuracy'], color='c',
label='val_accuracy', linestyle='--')
ax.set_xticks(range(1, n_epochs+1))
ax.legend()

# Training Loss Correction

fig, ax = plt.subplots(figsize=(10, 5))
n_epochs = len(h['loss'])

# Shift training loss by 0.5 as training loss is measured during the epoch
and validation loss is measured after the epoch

x_loss = np.arange(n_epochs+1)-0.5
ax.plot(x_loss[x_loss >= 0], h['loss'], color='b', label='train_loss')
ax.plot(range(1, n_epochs+1), h['val_loss'], color='r', label='val_loss')
ax.plot(range(1, n_epochs+1), h['accuracy'], color='b',
label='train_accuracy', linestyle='--')
```

```
ax.plot(range(1, n_epochs+1), h['val_accuracy'], color='r',
label='val_accuracy', linestyle='--')
ax.set_xlim(0, n_epochs)
ax.set_xticks(range(1, n_epochs+1))
ax.legend()

# Defining a function the get the training and validation plots representing
the accuracy and loss at each epoch

def get_train_val_plots(h, yticks=None, figsize=(10, 5)):
    # h: Any dictionary like the history.history

    fig, ax = plt.subplots(figsize=figsize)
    n_epochs = len(h['loss'])
    x_loss = np.arange(n_epochs+1)-0.5

    ax.plot(x_loss[x_loss >= 0], h['loss'], color='b', label='train_loss')
    ax.plot(range(1, n_epochs+1), h['val_loss'], color='r', label='val_loss')
    ax.plot(range(1, n_epochs+1), h['accuracy'], color='b',
label='train_accuracy', linestyle='--')
    ax.plot(range(1, n_epochs+1), h['val_accuracy'], color='r',
label='val_accuracy', linestyle='--')
    ax.set_xlim(0, n_epochs)
    ax.set_xticks(range(1, n_epochs+1))
    if yticks is not None:
        ax.set_yticks(yticks)
    ax.legend()

get_train_val_plots(h)

test_df = pd.read_csv('data/alphabet/sign_mnist_test.csv') # Load the test
data

X_test, y_test = test_df.drop('label', axis=1), test_df['label']

X_test = tf.reshape(X_test, [-1, 28, 28, 1])

y_test = label_binarizer.transform(y_test)

best_model.evaluate(X_test, y_test)

# Preprocesses the input and evaluates the model

def evaluate_model(model, X_test, y_test, label_binarizer):
    X_test_reshape = tf.reshape(X_test, [-1, 28, 28, 1])
    y_test_labels = label_binarizer.transform(y_test)
    results = model.evaluate(X_test_reshape, y_test_labels)
    print(f'Loss: {results[0]:.3f} Accuracy: {results[1]:.3f}')

results = evaluate_model(best_model, test_df.drop('label', axis=1),
test_df['label'], label_binarizer)

test_df = pd.read_csv('data/alphabet/sign_mnist_test.csv') # Load the test
data

X_test, y_test = test_df.drop('label', axis=1), test_df['label']

X_test = tf.reshape(X_test, [-1, 28, 28, 1])
```

```
d = {chr(ord('a') + i):i for i in range(26)}
d_rev = {i:chr(ord('a') + i) for i in range(26)}
d[' '] = d_rev[' '] = ' '

sentence = 'sign language'

for i in sentence:
    print(d[i], end=' ')

best_model.predict(tf.reshape(X_test[0], [-1, 28, 28, 1]))

images_taken = []
result = ''

for i in sentence:
    if i != ' ':
        char_index = np.random.choice(y_test[y_test==ord(i)-ord('a')].index)
        images_taken.append(char_index)
        y_pred = best_model.predict(tf.reshape(X_test[char_index], [-1, 28,
28, 1]))
        result += d_rev[label_binarizer.inverse_transform(y_pred)[0]]
    else:
        result += ' '
print(result)

# Visualizing the test images
images_taken_dup = list(reversed(images_taken))
for word in sentence.split():
    fig, ax = plt.subplots(1, len(word), figsize=(20, 20))
    for i in range(len(word)):
        ax[i].imshow(X_test[images_taken_dup.pop()], cmap='gray')
        ax[i].set_title(word[i])

def test_on_sentence(model, sentence, X_test, y_test, label_binarizer,
figsize=(20, 20)):
    # Random images are taken from X_test along with the corresponding labels
in y_test
    # based on the letters in the sentence.
    # These images are fed to the model and its output is printed

    sentence = sentence.lower()

    d = {chr(ord('a') + i):i for i in range(26)}
    d_rev = {i:chr(ord('a') + i) for i in range(26)}
    d[' '] = d_rev[' '] = ' '


    images_taken = []
    result = ''

    X_test_reshape = tf.reshape(X_test, [-1, 28, 28, 1])


    for i in sentence:
        if i != ' ':
            char_index = np.random.choice(y_test[y_test==ord(i)-
ord('a')].index)
```

```
            images_taken.append(char_index)
            y_pred = model.predict(tf.reshape(X_test_reshape[char_index], [1,
28, 28, 1]))
            result += d_rev[label_binarizer.inverse_transform(y_pred)[0]]
        else:
            result += ' '

    print(f'The actual sentence is "{sentence}"')
    print(f'The predicted sentence is "{result}"')

    images_taken.reverse()
    for word in sentence.split():
        fig, ax = plt.subplots(1, len(word), figsize=figsize)
        for i in range(len(word)):
            ax[i].imshow(X_test_reshape[images_taken.pop()], cmap='gray')
            ax[i].set_title(word[i])

test_on_sentence(best_model, 'sign language', test_df.drop('label', axis=1),
test_df['label'], label_binarizer)
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖ»

Виконала: студентка 4 курсу,групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Артеменко І. В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф.КН
Паночишин Ю.М.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

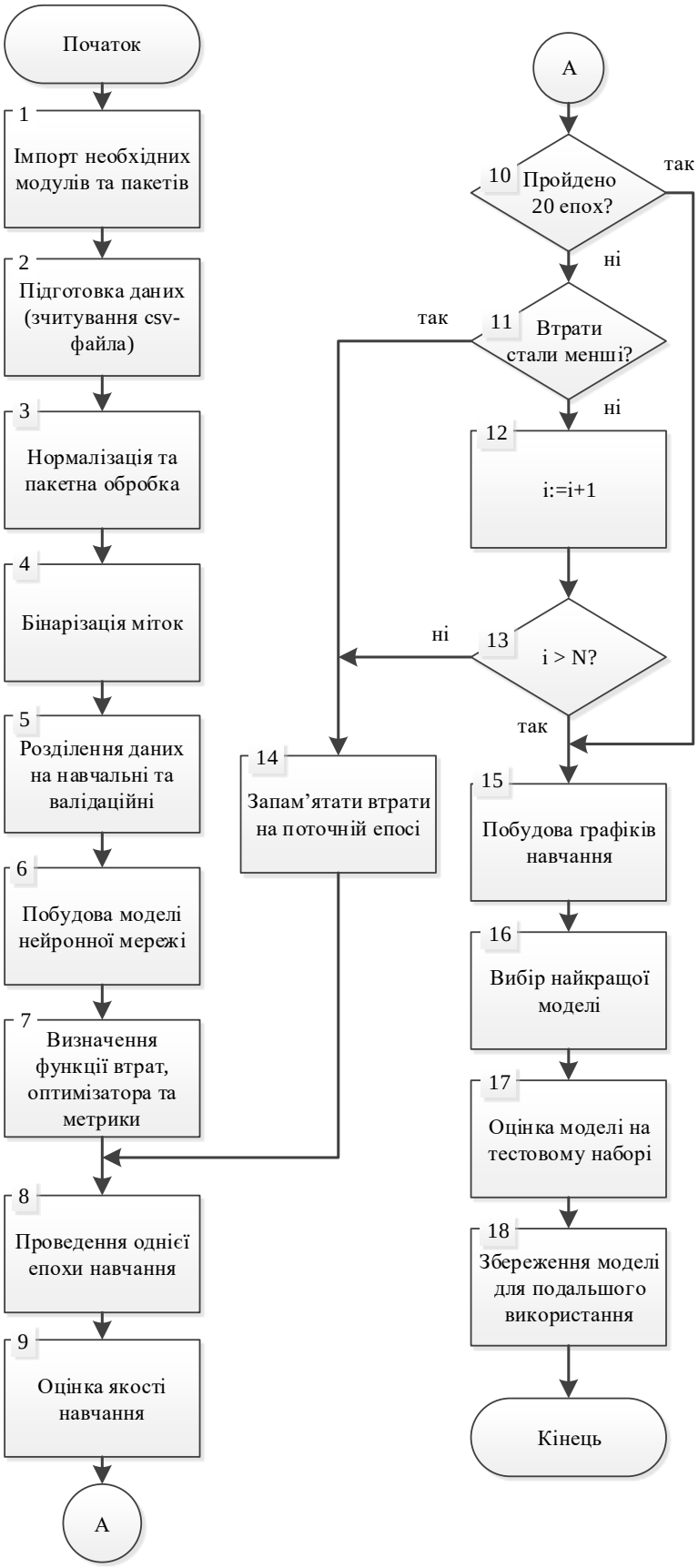

Рисунок В.1– Схема алгоритму роботи програмного модуля розпізнавання мови жестів на основі згорткової нейронної мережі

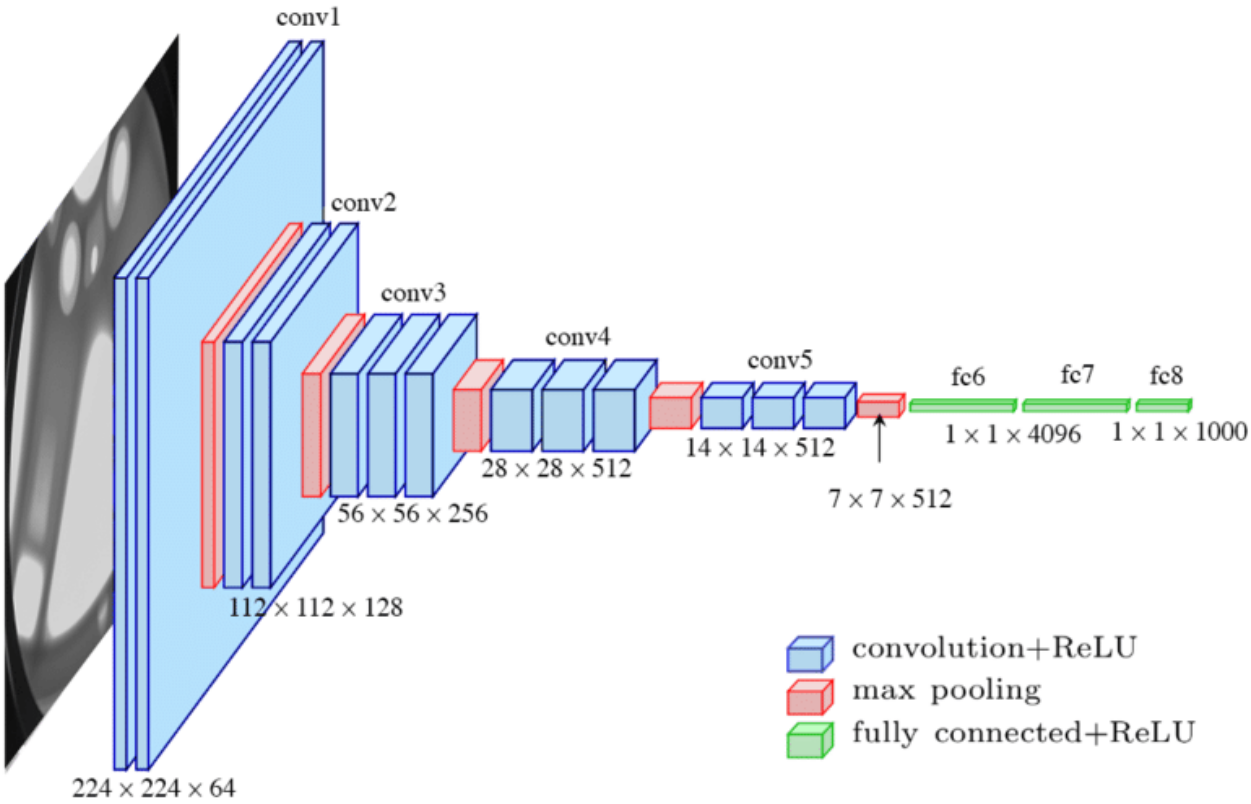

Рисунок В.2– Структура згорткової нейронної мережі VGG16

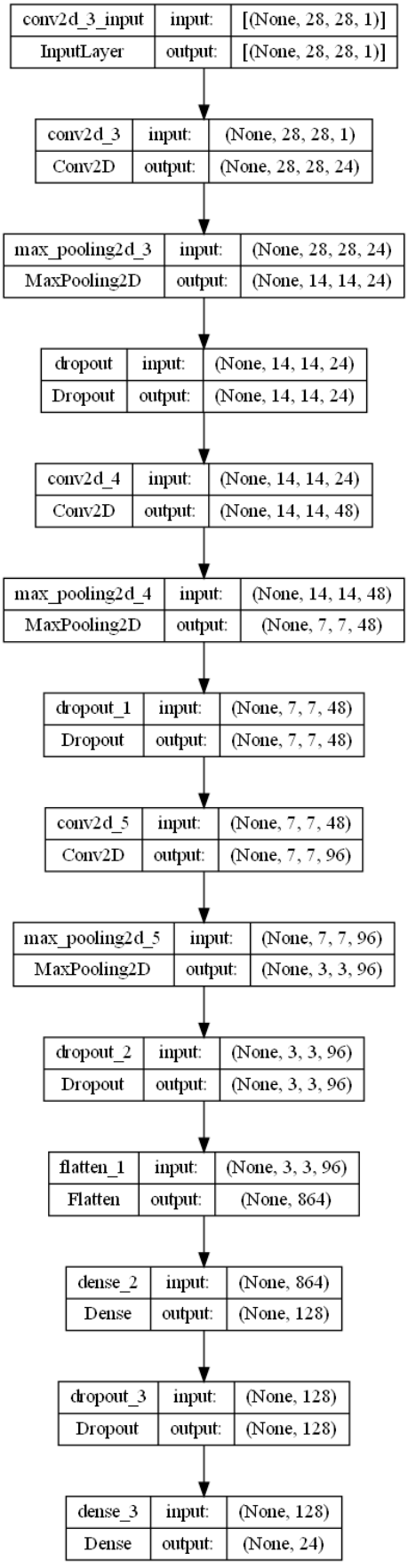

Рисунок В.3 - Найкраща із розроблених моделей після оптимізації гіперпараметрів

Рисунок В.4 – Кодування англійських літер мовою жестів

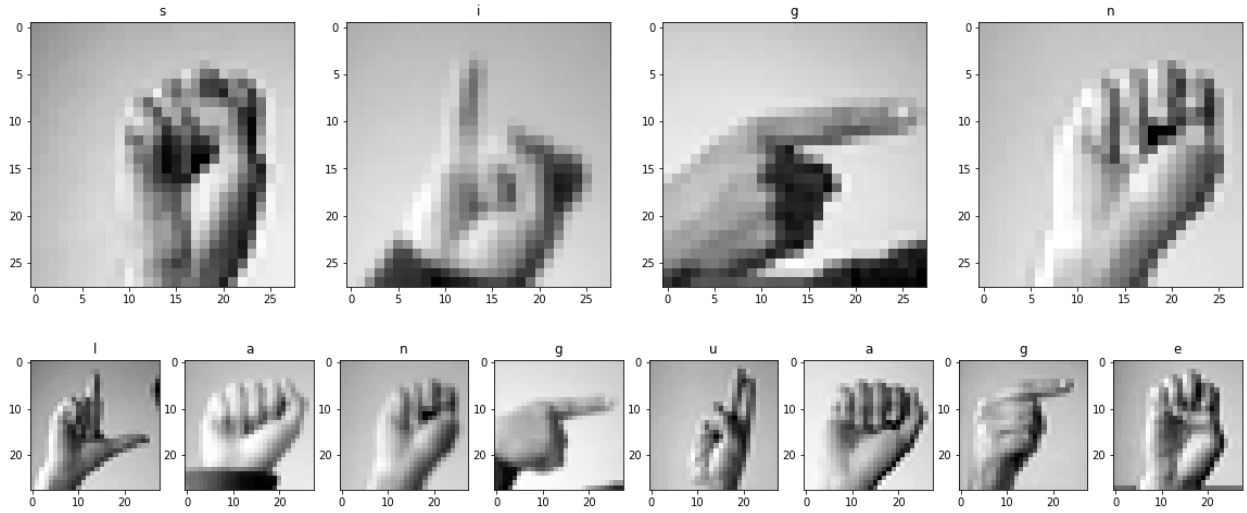

Рисунок В.5 - Результат розпізнавання послідовності зображень речення "sign language"

Таблиця В.1 - Порівняння аналогів та розробленого модуля розпізнавання мови жестів

	Вид жестової мови	Вид згорткової нейромережі	Кількість зображень навчальної вибірки	Кількість зображень тестової вибірки	Достовір-ність розпізна-вання, %
Аналог1 [д2]	Італійський	2xCNN+ повнозв’язна НМ	4600	2000	91,7
Аналог2 [д]	Американ-ський англійський	Inception v3	1920	480	90,5
Розроб-лений модуль	Американ-ський англійський	Модифікована VGG	27455	7172	96,3