МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

Магістерська робота

ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ

Виконала: студентка групи ПМІМ-22с
спеціальності
122 "Комп'ютерні науки"
(шифр і назва спеціальності)

(підпис) Ромах С.В.
(прізвище та ініціали)

Керівник (підпис) доц. Бернакевич І.Є.
(прізвище та ініціали)

Рецензент (підпис) доц. Дяконюк Л.М.
(прізвище та ініціали)

ДЕКАН
факультету прикладної
математики та інформатики
ЛНУ імені Івана Франка

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *інформаційних систем*

Спеціальність *122 Комп'ютерні науки*

«ЗАТВЕРДЖУЮ»

Завідувач кафедри
проф.Шинкаренко Г.А.
« 7 » *вересня* 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТКИ

Ромах Софії Володимирівни
(прізвище, ім'я, по батькові)

1. Тема роботи *ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ*
керівник роботи *доц. Бернакевич І.Є.*
затверджені Вченою радою факультету від «__» _____ 20 22 р., №

2. Дата подання студентом роботи *12.12.2022*

3. Вихідні дані до роботи
1. Deafness and hearing loss [Електронний ресурс] – Режим доступу: https://www.who.int/health-topics/hearing-loss
2. Sturman D. J. A survey of glove-based input / Sturman D.J., Zeltzer D. – IEEE Computer Graphics and Applications, 1994.
3. Новотарський М. А. Штучні нейронні мережі: обчислення / М. А. Новотарський, Б. Б. Нестеренко. – Київ: Інститут математики НАН України, 2004.
4. Rashid T. Make Your Own Neural Network / Tariq Rashid. – CreateSpace Independent Publishing Platform, 2016.
5. Math.NET [Електронний ресурс] – Режим доступу: https://numerics.mathdotnet.com/

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Опис предметної області
2. Вибір технології для збору даних
3. Постановка завдання
4. Опис методу обробки зображень
5. Розробка нейронної мережі для розпізнавання жестів
6. Вибір інструментальних засобів для реалізації програмного забезпечення
7. Програмна реалізація
8. Аналіз отриманих результатів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1. Діаграма прецедентів (оформлена у вигляді use case діаграми)
2. Будова нейронної мережі (перцептрон, багатошарова нейронна мережа)
3. Функції активації нейронної мережі (сигмоїда, східчаста функція)

4. Діаграми розроблених класів

5. Ілюстрації результатів (знімки екранів під час роботи програми)

6. Консультанти розділів роботи

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

7. Дата видачі завдання *14.09.22*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Аналіз об'єкта дослідження*	*вересень 2022*	
2.	*Вибір технології для збору даних*	*вересень 2022*	
2.	*Постановка завдання*	*вересень 2022*	
3.	*Вибір методів для розробки системи*	*жовтень 2022*	
4.	*Вибір інструментальних засобів для реалізації програмного забезпечення*	*жовтень 2022*	
5.	*Програмна реалізація та обробка результатів*	*жовтень-листопад 2022*	
6.	*Апробація*	*листопад 2022*	
7.	*Оформлення роботи*	*листопад-грудень 2022*	
8.	*Захист роботи*	*грудень 2022*	

Студент _____ *Ромах С.В.*
підпис

Керівник роботи _____ *доц Бернакевич І.Є.*
підпис

РЕФЕРАТ

Актуальність теми

Розробка систем перекладу жестової мови є важливою для того, щоб люди з порушенням слуху могли почувати себе повноцінно у сучасному суспільстві. Система, яка здатна перекладати жестову мову у текст, який зручний для більшої кількості людей, допоможе пришвидшити процес вивчення жестової мови.

Зараз тенденція на системи такого типу все більше зростає, тому напрямок розробки систем розпізнавання жестової мови є вкрай актуальним. Ця актуальність полягає у необхідності створення методів, моделей та алгоритмів для розпізнавання, локалізації, аналізу та відстеження жестів у реальному часі.

Мета роботи

Метою даної роботи було обрано розробку інтелектуальної інформаційної системи для розпізнавання алфавіту української жестової мови за допомогою вебкамери комп'ютера. Ця система забезпечить переклад мови жестів на звичайну мову, таким чином люди з порушенням слуху зможуть використовувати розроблену програму для власних потреб.

Об’єкт дослідження

Об’єктом дослідження даної роботи є методи ідентифікації жестів та алфавіт української жестової мови.

Предмет дослідження

Предметом цього дослідження є інтелектуальна система для розпізнавання жестів, яка за заданим зображенням жесту буде визначати, якій літері це зображення відповідає.

Ключові слова

Інтелектуальна інформаційна система, обробка зображення, нейронна мережа, сигмоїдна функція, EmguCV, Math.NET, бінаризація, перцептрон, функція активації.

ABSTRACT

Topicality

The development of sign language translation systems is important so that people with hearing impairments can feel fully present in modern society. A system that can translate sign language into text that is convenient for more people will help speed up the process of learning sign language.

Currently, the trend towards systems of this type is growing, so the direction of development of sign language recognition systems is extremely relevant. This relevance lies in the need to create methods, models and algorithms for recognition, localization, analysis and tracking of gestures in real time.

Purpose of the research

The goal of this work was to develop an intelligent information system for recognizing the alphabet of the Ukrainian sign language using a computer webcam. This system will translate sign language into normal language, so hearing impaired people can use the developed program for their own needs.

Object of the research

The object of research of this work is the methods of identification of gestures and the alphabet of the Ukrainian sign language.

Subject of the research

The subject of this study is an intelligent system for recognizing gestures, which, based on a given image of a gesture, will determine which letter this image corresponds to.

Keywords

Intelligent information system, image processing, neural network, sigmoid function, EmguCV, Math.NET, binarization, perceptron, activation function.

ЗМІСТ

ВСТУП

Сьогодні більше ніж 5% всього населення на Землі мають проблеми зі слухом. Згідно зі статистикою Всесвітньої організації охорони здоров’я, у світі налічується близько 430 мільйонів людей з такими порушеннями, серед них 34 мільйони — діти. Щорічно ці цифри зростають. За оцінками спеціалістів ВООЗ до 2050 року так чи інакше страждатимуть від втрати слуху понад 900 мільйонів людей. Відомо, що в Україні з такими порушеннями налічується понад 2 мільйони осіб. З них приблизно для 230 тисяч основним або одним з основних способів комунікації є жестова мова [1].

З наведеної статистики зрозуміло, що розробка систем перекладу жестової мови є вкрай важливою для того, щоб люди з порушенням слуху могли почувати себе повноцінно у сучасному суспільстві. Інтелектуальна система, яка здатна перекладати жестову мову у текст, який зручний для більшої кількості людей, допоможе пришвидшити процес вивчення жестової мови. Таким чином людям, які мають проблеми зі слухом, буде простіше порозумітися з людьми, у яких таких проблем немає.

В теперішній час комп'ютерний переклад жестової мови з використанням камери – це один із панівних напрямків в галузі штучного інтелекту та комп'ютерного зору, яка розвивається протягом багатьох років. Останнім часом додатки, які не вимагають безпосереднього контакту між користувачем та пристроєм, набули широкого використання по всьому світу. Практичне застосування технології розпізнавання жестів охоплює додатки розпізнавання мови глухонімих, людино-машинної взаємодії, віртуальної реальності та ін.

Зараз тенденція на системи такого типу все більше зростає, тому напрямок розробки систем розпізнавання жестової мови є вкрай актуальним. Ця актуальність полягає у необхідності створення методів, моделей та алгоритмів для розпізнавання, локалізації, аналізу та відстеження жестів у реальному часі, які будуть використовуватись для розробки систем взаємодії між комп'ютером та людиною.

З прикладної точки зору актуальність теми полягає у потребі створення програмних засобів, які можуть у реальному часі аналізувати та перекладати подані жести, які позначають букву з українського алфавіту.

Власне враховуючи актуальність цієї проблеми, метою даної роботи було обрано розробку інтелектуальної інформаційної системи для розпізнавання алфавіту української жестової мови за допомогою відеокамери комп'ютера. Ця задача є доволі комплексною і умовно її можна поділити на дві основні частини: обробка самого зображення та безпосередньо розпізнавання поданого жесту за допомогою нейронної мережі. Обидві частини є детально описані та програмно реалізовані у цій магістерській роботі.

РОЗДІЛ 1. АНАЛІЗ ОБ'ЄКТА ДОСЛІДЖЕННЯ

1.1 Опис предметної області

Для досягнення мети, яка була описана у вступній частині, потрібно розв’язати такі задачі:

- проаналізувати алфавіт української жестової мови (рис. 1.1) та виділити основні принципи ідентифікації кожного з жестів;
- розробити структуру програмного продукту, яке забезпечить якісне виконання поставленої задачі;
- розробити програмний продукт для опрацювання зображення, виокремлення положення руки з опрацьованого зображення, обробки виокремленого зображення, формування даних, які в подальшому будуть використані для тренування нейронної мережі і, власне, розпізнавання поданого жесту з використанням натренованої нейромережі.

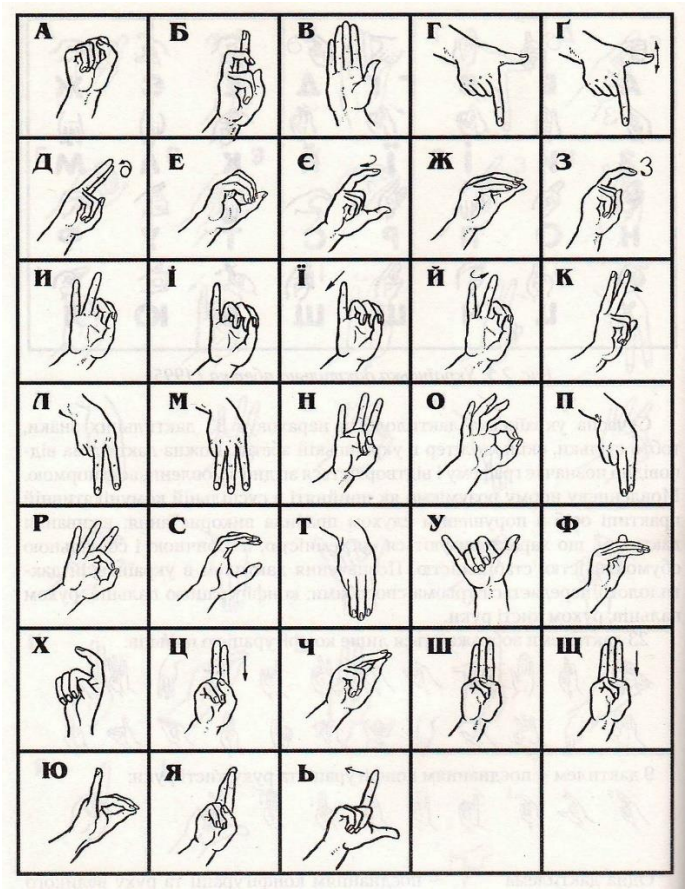


Рис. 1.1. Алфавіт української жестової мови

Об’єктом дослідження даної роботи є методи ідентифікації жестів та алфавіт української жестової мови.

Предметом цього дослідження є інтелектуальна система для розпізнавання жестів, яка за заданим зображенням жесту буде визначати, якій літері це зображення відповідає.

Оскільки нейронна мережа навчається на прикладах, то розроблений у цій роботі програмний продукт можна буде використати для розпізнавання інших мов жестів, окрім української.

1.2 Вибір технології для збору даних

Підготовчим етапом розробки інформаційної системи є аналіз технологій для збору даних і вибір тієї, яка найбільше підходить для виконання роботи. Адже для того, щоб додаток міг розпізнати жест, перш за все потрібно надати йому інформацію про положення долоні та пальців.

Існує дві головні категорії пристроїв для збору даних:

1. контактні;
2. безконтактні.

Щодо контактних пристроїв, які забезпечать отримання інформації про положення руки і пальців, то їх є не так багато. Одним із найзручніших пристроїв для передання жестової інформації до комп’ютера може бути інформаційна рукавиця.

Інформаційна рукавиця (англ. date glove), або провідна рукавиця (англ. wired glove) – це пристрій для введення даних, який базується на жестах. Великий набір сенсорів допомагає отримати інформацію про положення руки, наприклад дані про згин пальців. Після отримання інформації про жест, її можна проаналізувати і перевести у зручний для читання текст. Інформаційні рукавиці вищого класу вміють забезпечувати зворотній зв’язок, який виглядає, як легенький удар струмом, що дозволяє імітувати відчуття дотику. Ця особливість дозволяє використовувати рукавицю не лише як пристрій введення інформації, але як пристрій виведення інформації [2].

Приклади інформаційних рукавиць зображено на рисунку 1.2.

Рис. 1.2 Приклади рукавиць для введення інформації

Ще одним поширеним способом контактного введення даних є технологія захоплення руху (англ. Motion capture).

Захоплення руху – це процес запису руху певного об’єкта або людини. Ця технологія найчастіше використовується під час створення відеоігор, CGI мультфільмів та фільмів.

Принцип захоплення руху полягає у наступному: людина у спеціальному костюмі з датчиками виконує необхідні рухи або симулює певні дії на сцені, яка спеціально обладнана відеокамерами; дані з датчиків фіксуються камерами й надходять до комп’ютера, де вони формуються у єдину тривимірну модель, яка точно відтворює рухи людини. Перевагою цієї технології є те, що вже через кілька хвилин після захоплення рухів можна отримати попередній дійсний результат [3].

Приклад рукавиці, яка розроблена за допомогою технології захоплення руху зображено на рисунку 1.3.

На жаль, контактні пристрої мають значний недолік – вартість. Через складність їх виробництва і вартість сенсорів, ці засоби коштують доволі дорого, відповідно не кожна людина може собі їх дозволити.

Рис. 1.3 Приклад рукавиці для технології захоплення руху

Безконтактними пристроями для збору даних називають такі пристрої, для використання яких не потрібно використовувати спеціальні костюми, як у минулих прикладах(наприклад інформаційні рукавиці). Такими пристроями є відеокамери. Вони захоплюють зображення і надалі воно опрацьовується методами комп'ютерного зору. Перевагою безконтактних пристроїв є їхня доступність. Розглянемо найвідоміші безконтактні пристрої для збору даних.

Kinect – це пристрій, призначений для Xbox 360, який зчитує рухи людини та дозволяє керувати системою без будь-яких додаткових пристроїв. Ця камера також сумісна з операційною системою Windows.

У верхній частині камери Kinect знаходяться такі два сенсори: інфрачервоний сенсор, який сканує простір і монохромний CMOS сенсор, який забезпечує перетворення отриманих даних в 3D проекцію і відстеження переміщення людей всередині неї. Між сенсорами знаходиться RGB камера, яка використовується для розпізнавання людей, зйомки відео і фотознимків.

За допомогою вбудованого сервоприводу Kinect може автоматично повертатись у вертикальній площині, підбираючи оптимальний кут огляду для камери.

Результат обробки зображення за допомогою камери Kinect зображено на рисунку 1.4.

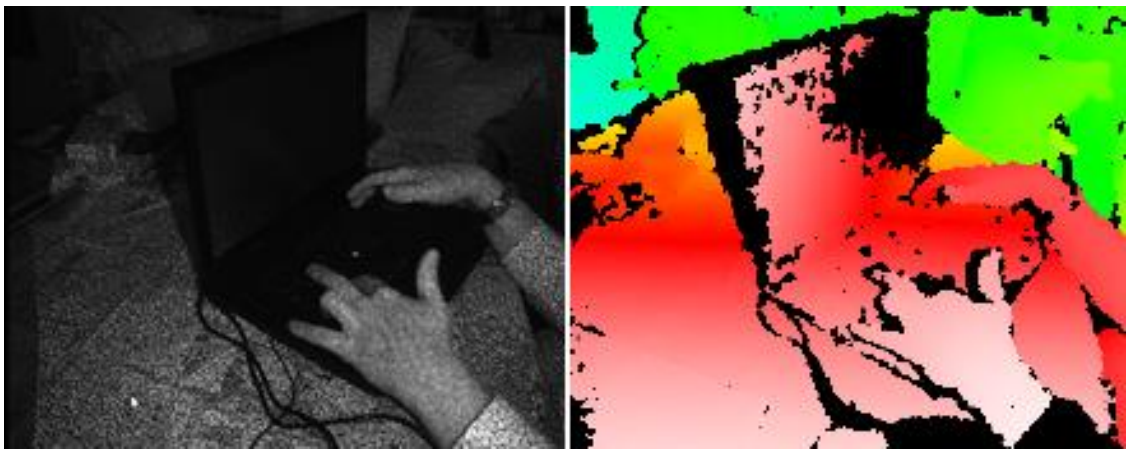


Рис. 1.4 Результат обробки відео камерою Kinect

Розглянемо ще один пристрій – вебкамеру комп'ютера. Цей пристрій є системою, яка складається з однієї камери, яка переважно є вбудованою у ноутбук або комп'ютер. Вона не може надати інформацію про глибину зображення, таким чином не буде змоги сформувати 3D зображення руки. Проте така камера дозволить точно розпізнати контур руки. А для задачі, яку потрібно вирішити в цій роботі, цього абсолютно достатньо. Адже даних про контур долоні достатньо для того, щоб з високою точністю розпізнати жест.

Перевагами однокамерної системи є те, вона дешевша ніж інші контактні пристрої, чи навіть ніж Kinect камера, а також ця камера вбудована у велику кількість пристроїв – телефони, комп'ютери, телевізори. Приклад такої камери можна побачити на рисунку 1.5.

Рис. 1.5 Приклад веб-камери, вбудованої у ноутбук

Враховуючи всі вищеперелічені переваги однокамерної системи, для реалізації завдання зчитування зображення було обрано звичайну вебкамеру комп'ютера.

1.3 Висновки до розділу

Оскільки важливою частиною реалізації інтелектуальної системи для розпізнавання жестів є саме робота із зображенням, потрібно відповідально підійти до вибору пристрою, який зможе забезпечити найточніші результати.

Після аналізу технологій, які призначені для зчитування рухів та жестів, було вирішено вибрати вебкамеру як засіб зчитування жестів. Адже для потреб нашої інформаційної системи вона підійде найкраще.

Таким чином обраний пристрій буде основною складовою інформаційної системи. Завдяки ньому ми зможемо зчитати зображення з камери і вже пізніше виконувати необхідні перетворення для досягнення бажаного результату.

РОЗДІЛ 2. ПОСТАНОВКА ЗАВДАННЯ ТА МЕТОДИ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ДЛЯ РОЗПІЗНАВАННЯ МОВИ ЖЕСТІВ

2.1 Постановка завдання

Метою даної системи є обробка зображення таким чином, щоб його можна було передати нейронній мережі на тренування і відповідно розпізнавання жесту.

Умовно інтелектуальну систему можна поділити на такі основні процеси:

- захоплення зображення;
- обробка зображення;
- аналіз зображення.

Розглянемо детальніше кожен з цих процесів.

Захоплення зображення – це перше завдання нашої системи, воно має виконуватись з самого початку. На цьому етапі пристрій введення, тобто веб-камера, захоплює жест і передає зображення на опрацювання. Вхідним елементом є жест (положення руки), а вихідним – захоплене зображення, тобто зображення з локалізованою рукою.

Опрацювання зображення – наступне завдання інформаційної системи. На цьому етапі зображення опрацьовується так, щоб було чітко видно контур руки, для того щоб у майбутньому його проаналізувати. Вхідним елементом є захоплене зображення, а вихідним – опрацьоване зображення, тобто зображення з чітко вираженим контуром руки для подальшого його аналізу.

І останній, найскладніший та найважливіший етап – це процес аналізу жестів. Саме на цьому етапі відбувається навчання нейронної мережі та розпізнавання зображення. Вхідним елементом є опрацьоване зображення, а вихідним – текстове значення жесту.

Для того, щоб краще розуміти, як функціонує інтелектуальна інформаційна система для розпізнавання мови жестів загалом, та як взаємодіють користувачі між

собою у цій системі, варто розглянути діаграму прецедентів, яка зображена на рисунку 2.1.

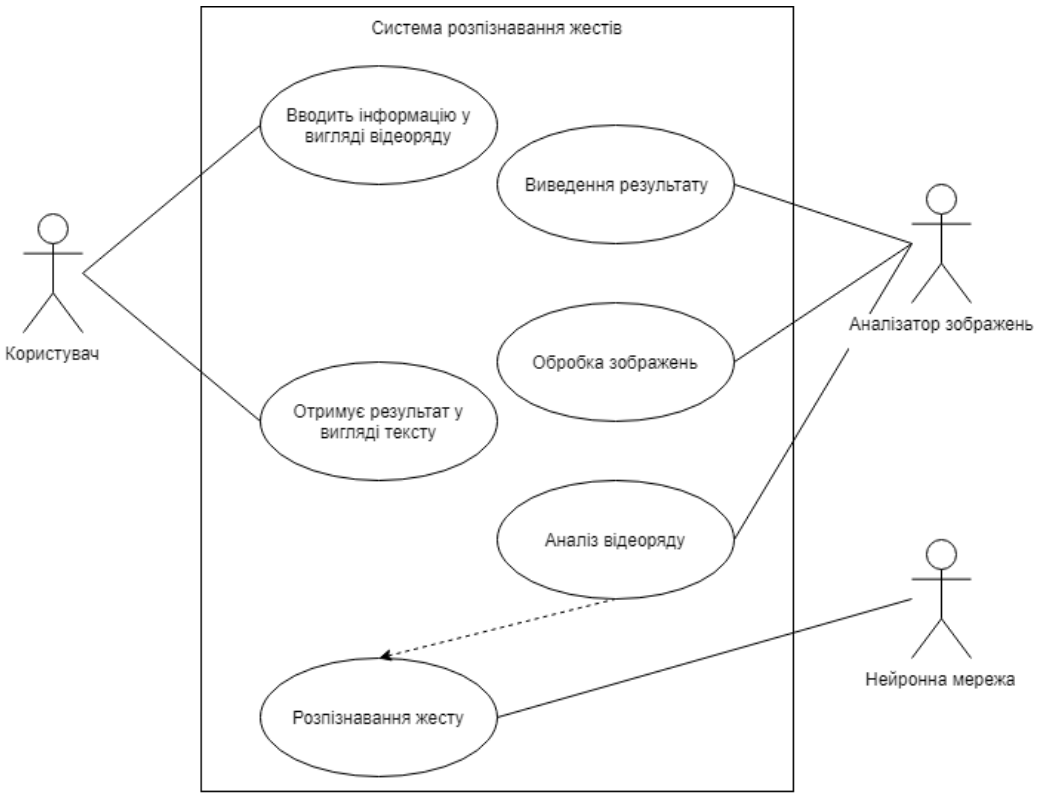


Рис. 2.1 Діаграма прецедентів

На рисунку 2.1 бачимо, що діаграма містить таких осіб:

- користувач – це особа, яка використовує нашу інтелектуальну інформаційну систему для виконання конкретної дії. Користувачу доступна можливість введення інформації у вигляді зображення для його подальшої обробки, аналізу та отримання остаточного результату у вигляді текстового відповідника поданого жесту;
- аналізатор зображень – це частина інтелектуальної інформаційної системи, яка призначена для аналізу зображення. Це програмний модуль, який приймає вхідне зображення, опрацьовує його за деякими алгоритмами. Після цього аналізатор виводить результат на екран, і його може бачити користувач. Також тут відбувається аналіз зображеня, що тісно пов’язано з навчанням нейронної мережі;
- нейронна мережа – це сукупність нейронів, які взаємодіють один з одним. Вони можуть приймати, обробляти та створювати дані. Якщо

розглядати нейронну мережу з точки зору машинного навчання, то вона є сукупністю методів розпізнавання образів, методів кластеризації, дискримінантного аналізу. В інформаційній системі нейронна мережа відповідальна за розпізнавання жесту, що, власне, і є метою всієї інтелектуальної інформаційної системи.

2.2 Метод обробки зображення

Для того, щоб зображення можна було використовувати у нейронній мережі, його потрібно підготувати до цього. В нашому випадку потрібно виконати деякі перетворення, які з вхідного кольорового зображення передадуть у нейронну мережу контури цього жесту. Для виконання поставленої задачі було проаналізовано наявні методи та обрано метод бінаризації зображення.

Процес бінаризації є перетворенням кольорового зображення в зображення в градаціях сірого або двоколірне чорно-біле. Основний параметр такого перетворення це поріг, зі значенням якого порівнюється яскравість кожного пікселя. Після виконання порівняння пікселю присвоюється одне з двох значень: 0 – «точка належить до об'єкта» або 1 – «інша область, точка не належить до об’єкта» [5].

Весь процес бінаризації описується наступними кроками:

- На вході маємо оригінальне кольорове зображення;
- Трансформація зображення у кольори градації сірого;
- Аналіз кожного пікселя і присвоєння йому значення 0 або 1;
- На виході отримуємо бінарне зображення.

Основною метою бінаризації є суттєве зменшення кількості інформації, з якою потрібно працювати. Вдала бінаризація значно спрощує майбутню роботу із зображенням [5].

Виділяють дві групи методів бінаризації:

- глобальні (порогові);
- локальні (адаптивні).

У глобальних (порогових) методах зображення опрацьовується в повному обсязі. Під час виконання методу у нас є поріг бінаризації, за допомогою якого відбувається розподіл на чорне і біле, причому важливо, що величина порогу залишається сталою протягом всього процесу бінаризації. До глобальних методів бінаризації відносяться [6]:

- за нижнім порогом;
- за верхнім порогом;
- з подвійним обмеженням;
- неповна порогова обробка;
- багаторівневе граничне перетворення.

З перелічених методів одним з найпростіших є саме бінаризація за нижнім порогом, де береться до уваги лише одне значення порогу [6]:

$$F'(m,\ n)=\begin{cases}0,\ F(m,n)\geq t,\\ 1,\ F(m,n)<t\end{cases} \tag{2.1}$$

Якщо в формулі 2.1 для точки зображення, яке ми аналізуємо, виконується перша умова, то ця точка є точкою об'єкта, а якщо виконується друга умова, то ця точка є точкою фону.

У певних випадках можна застосовувати варіант методу бінаризації з верхнім порогом, в результаті якого отримується негатив вихідного зображення. Такий метод називається бінаризацією з верхнім порогом і зображається такою формулою:

$$F'(m,\ n)=\begin{cases}0,\ F(m,n)\leq t,\\ 1,\ F(m,n)>t\end{cases} \tag{2.2}$$

Якщо потрібно виділити певні області зображення, в яких значення яскравості пікселів можуть змінюватися в межах певного діапазону, то можна застосувати метод бінаризації з подвійним обмеженням. Такий метод описується наступною формулою:

$$F'(m,\ n)=\begin{cases}0,\ F(m,n)\geq t1,\\ 1, t1<F(m,n)\leq t2\\ 0,\ F(m,n)>t2\end{cases} \tag{2.3}$$

Якщо потрібно отримати якомога простіше для подальшого аналізу зображення, то слід використати алгоритм неповної порогової обробки, в результаті виконання якого із зображення вилучається фон зі всіма його деталями.

Формула для неповної порогової бінаризації записана нижче:

$$F'(m,\ n)=\begin{cases} F(m,n),\ F(m,n) > t, \\ 0,\ F(m,n) \le t \end{cases} \tag{2.4}$$

У тому разі, якщо потрібно мати зображення, яке містить в собі частини з різною яскравістю, можна використати метод багаторівневого порогового перетворення. Проте таке зображення вже не буде вважатись бінарним (чорно-білим).

Формула такого перетворення записана нижче:

$$F'(m,\ n)=\begin{cases} 1,\ F(m,n)\ \epsilon\ D1, \\ 2,\ F(m,n)\ \epsilon\ D2, \\ \dots \\ n, F(m,n)\ \epsilon\ Dn \\ 0,\ \text{в усіх інших випадках} \end{cases} \tag{2.5}$$

Результат алгоритму порогової бінаризації представлений на рисунках 2.2, 2.3.

Рис. 2.2 Вхідне зображення

Рис. 2.3 Зображення після перетворення методом порогової бінаризації

2.3 Нейронна мережа для розпізнавання жесту

Після обробки зображення можна починати тренування нейронної мережі та згодом, коли буде достатньо прикладів, можна буде розпізнавати подані жести.

Спочатку розглянемо, що таке нейронна мережа та яким чином з її допомогою можна виконати поставлене завдання – розпізнати жест.

Штучна нейронна мережа – це сукупність нейронів, які взаємодіють один з одним. Вони можуть приймати, обробляти та створювати дані. Нейромережа зазвичай складається з трьох або більше шарів: вхідного шару, прихованого шару (або шарів) та вихідного шару.

2.3.1 Перцептрон

Для початку розглянемо тип штучного нейрона під назвою перцептрон. Цей термін був розроблений в 1950-х і 1960-х роках вченим Френком Розенблатом, який черпав інформацію у роботах Уоррена Мак-Калоха і Вальтера Піттса [7].

Перцептрон приймає декілька дійкових входів і створює єдиний двійковий вихід. Ця проста схема зображена на рисунку 2.4.

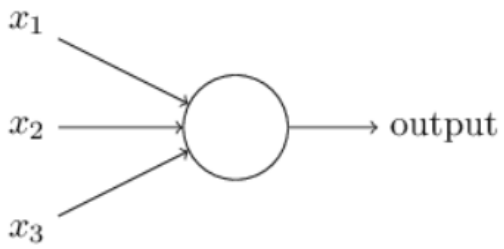


Рис. 2.4 Перцептрон з трьома входами

У наведеному прикладі перцептрон має три входи x_1, x_2, x_3, але може бути як більше, так і менше вхідних даних. Розенблат запропонував просте правило для обчислення результату. Він представив ваги w_1, w_2, w_3 – дійсні числа, що виражають важливість відповідних вхідних даних для виходу. Вихід нейрона, 0 або 1, визначається за допомогою порівняння зваженої суми $\sum_j w_j x_j$ з деяким пороговим значенням. Цей поріг є дійсним числом, як і вагові коефіцієнти. Для сформульованого правила запишемо формулу

$$\text{вихід} = \begin{cases} 0, \sum_j w_j x_j \ \leq \text{поріг} \\ 1, \sum_j w_j x_j > \text{поріг} \end{cases} \tag{2.6}$$

Це основна математична модель перцептрона. Тобто це пристрій який приймає рішення зважуючи докази.

Тепер розглянемо трохи складніший приклад перцентрона на рисунку 2.5.

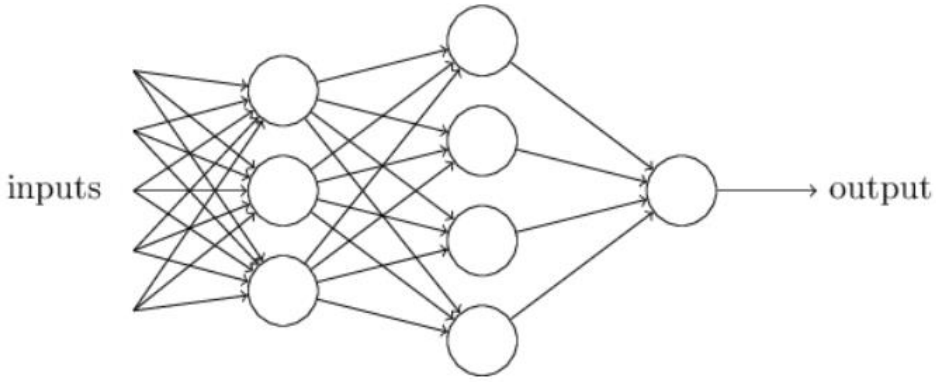


Рис. 2.5 Двошаровий перцептрон

У цій мережі перший стовпець є першим шаром перцептронів, який приймає три рішення на основі зважування вхідних даних. А в другому шарі кожен з перцептронів приймає рішення зважуючи результати першого рівня. Ще складніші

рішення може приймати третій шар. Таким чином, багатошарова мережа перцептронів може брати участь у прийнятті складних рішень.

Для кращого розуміння перепишемо формулу 2.6 у простіший вигляд замінивши значення порогу на – b :

$$\text{вихід} = \begin{cases} 0, \text{якщо } w \cdot x + b \leq 0 \\ 1, \text{якщо } w \cdot x + b > 0 \end{cases} \qquad (2.7)$$

2.3.2 Сигмоїдні нейрони

Нам потрібно розробити алгоритми для нейронної мережі які будуть розв’язувати поставлену задачу – розпізнавати жести. Для цього вхідними даними в нейронну мережу можуть бути піксельні дані з обробленого зображення жесту. Нам потрібно, щоб мережа вивчала ваги та зміщення для того, щоб вихідні дані мережі могли правильно класифікувати жест. Щоб побачити, як мережа буде навчатись, ми будемо вносити зміни у певну вагу і відповідно буде змінюватись вихідний сигнал. Схематично ця процедура зображена на рисунку 2.6.

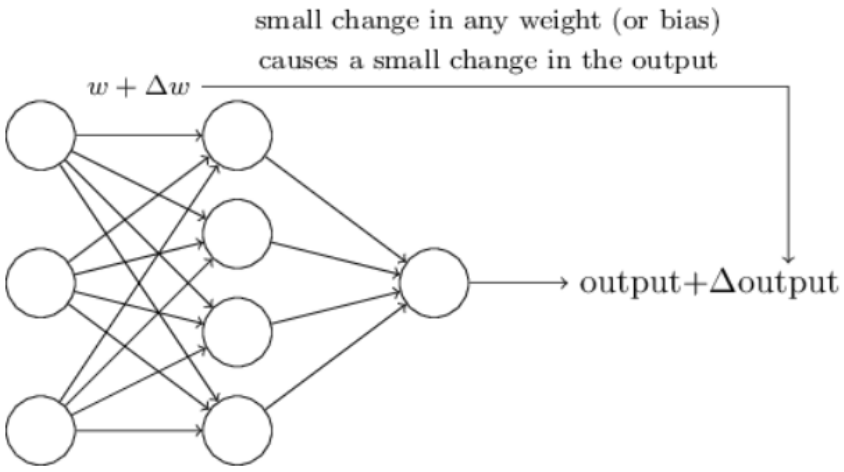


Рис. 2.6 Навчання мережі за допомогою зміни ваг

Проте насправді навіть незначна зміна ваги тут може спричинити вагомі зміни вихідного сигналу. Тому для нашої задачі краще використати сигмоїдний нейрон. Такий тип нейронів схожий до перцептронів, проте вони модифіковані таким чином, що невеликі зміни в їхніх вагах спричиняють незначну зміну їхнього виходу. Це важливий факт, завдяки якому мережа сигмоїдних нейронів може навчатись.

Сам сигмовидний нейрон можна зобразити так само як перцептрон (рис. 2.4). Але тут вхідні дані можуть приймати будь-які значення між 0 і 1. Так, наприклад, 0.674 може бути входом для сигмовидного нейрона. Ваги й зміщення b є такі самі як для перцептрона, проте вихід може набувати значень, які описуються функцією:

$$p(z) = \frac{1}{1+e^{-z}} \tag{2.8}$$

де

$$z = \sum_j w_j x_j - b \tag{2.9}$$

На рисунку 2.7 зображена сигмоїдна функція $p(z)$, яка називається функцією активації.

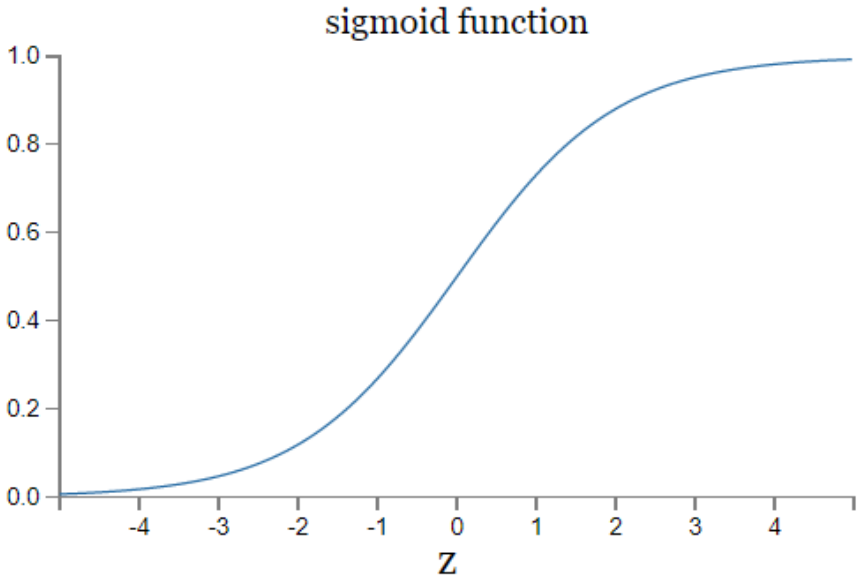


Рис. 2.7 Сигмоїдна функція

Ця функція є згладженою версією східчастої функції (рис. 2.8), яка є функцією перцептрона. Саме ця згладженість і є ключовим фактором навчання нейромережі. Адже це свідчить про те, що невеликі зміни Δw_j у вагах і зміщенні b спричинять незначні зміни Δвихід на виході нейрона.

Взагалі рекомендовано, щоб функція активації задовольняла такі властивості:

- нелінійність;
- неперервна диференційовність;

- область визначення;
- монотонність;
- гладкість функції з монотонною похідною;
- наближення до тотожної функції в початку координат.

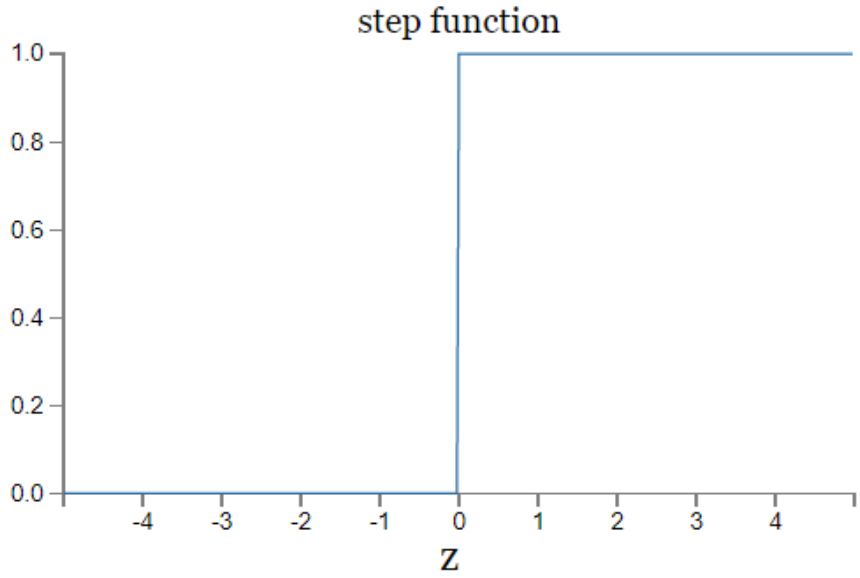


Рис. 2.8 Ступінчаста функція

2.3.3 Архітектура нейронних мереж

Тепер розглянемо детальніше структуру нейронної мережі, яка буде використовуватись для розпізнавання жестів. Розглянемо мережу, зображену на рисунку 2.9. Вона має два приховані шари – ті, які не є ні входами, ні виходами. Такі багатошарові мережі називають багатошаровими перцептронами або MPL.

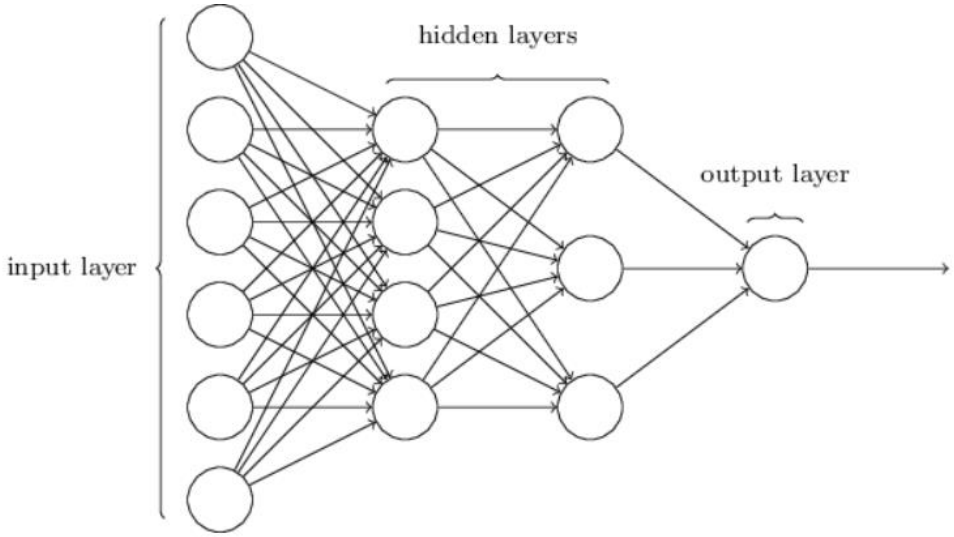


Рис. 2.9 Чотирирівнева мережа

Структура вхідного та вихідного шарів у мережі є простою. Наприклад, уявімо, що нам потрібно визначити чи зображено жест, який відповідає літері «а». Нехай розмір зображення становить 64x64 пікселі й воно представлене в градаціях сірого, тоді нам потрібно $4096 = 64 * 64$ вхідних нейронів зі значенням інтенсивності від 0 до 1. Вихідний рівень міститиме лише один нейрон, для якого, якщо вихідне значення менше за 0.5, то зображення не відповідає літері «а», а якщо вихідне значення більше за 0.5, то жест відповідає літері «а».

Проєктування прихованих шарів може відбуватись різними способами й неможливо описати цей процес кількома простими емпіричними правилами [8].

До цього часу ми обговорювали нейронні мережі, де вихідні дані з одного рівня використовуються як вхідні дані для наступного рівня. Такі мережі називаються нейронними мережами *прямого зв'язку*. Це означає, що в мережі немає петель – інформація завжди передається вперед і не повертається назад.

Однак існують інші моделі штучних нейронних мереж, в яких можливі петлі зворотного зв'язку. Ці моделі називають *рекурентними нейронними мережами*.

Для реалізації поставленого завдання, тобто розпізнавання жесту, буде використовуватись нейромережа прямого зв'язку.

Важливим алгоритмом, який використовується для навчання нейронної мережі, є метод зворотного поширення помилки. Це ітеративний градієнтний алгоритм, при виконанні якого похибка між фактичною відповіддю і передбаченою мінімізується.

Похідна цієї похибки розраховується по кожній вазі і згодом ці диференціали множаться на значення швидкості навчання. Після цього отриманий результат віднімається від відповідних ваг (2.10, 2.11, 2.12).

$$w_1 = w_1 - \left(\eta \cdot \frac{\partial(err)}{\partial(w_1)}\right) \tag{2.10}$$

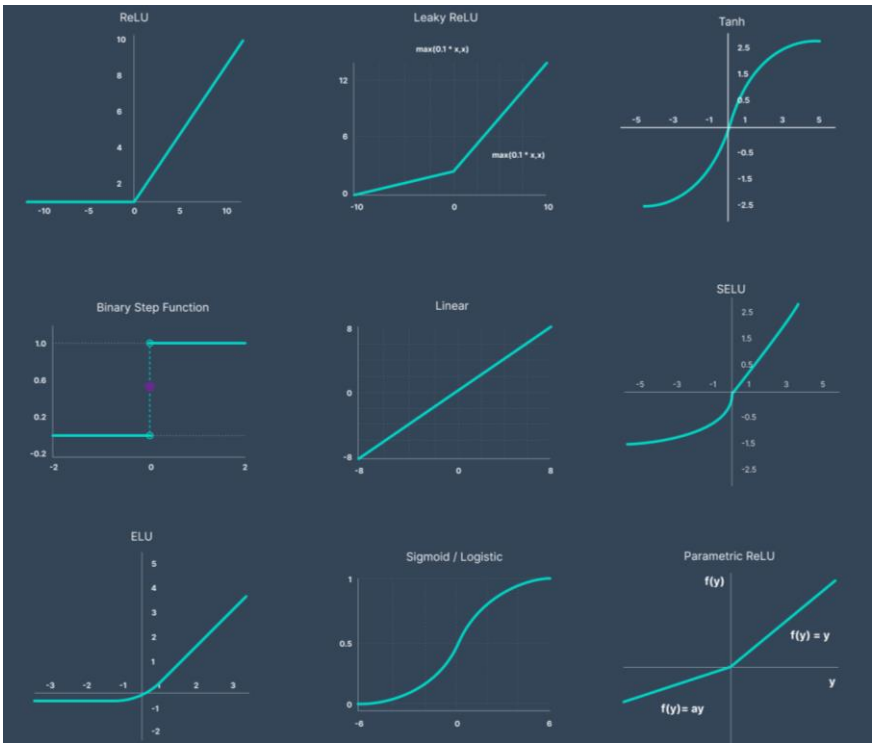
$$w_2 = w_2 - \left(\eta \cdot \frac{\partial(err)}{\partial(w_2)}\right) \tag{2.11}$$

$$w_3 = w_3 - \left(\eta \cdot \frac{\partial(err)}{\partial(w_3)}\right) \tag{2.12}$$

Деякі параметри нейронної мережі налаштовуються вручну, а саме – швидкість навчання, функція активації, функція втрати. Розглянемо коротко кожен з цих параметрів.

Надзвичайно важливим параметром є швидкість навчання мережі (learning rate). Це параметр, який контролює те, наскільки потрібно змінити модель відповідно до помилки щоразу, коли змінюються вагові коефіцієнти моделі. Зазвичай це значення знаходиться в межах від 0.0 до 1.0. Якщо ця величина занадто мала, то для отримання оптимальних результатів недостатньо буде навіть тривалого процесу навчання. Проте, якщо значення швидкості занадто високе, то ми дуже швидко тримаємо відповіді, проте їхня точність буде не високою, тобто в результаті отримаємо неоптимальний набір ваг, або процес навчання буде нестабільним.

Про функцію активації вже було згадано в цьому розділі. Вона визначає те, яка інформація буде передаватись до наступних шарів, тобто чи потрібно активовувати нейрон, чи ні. Найкраще брати нелінійні функції активації, як, наприклад, сигмоїдна функція. На рисунку 2.10 зображено різні приклади функцій активації.



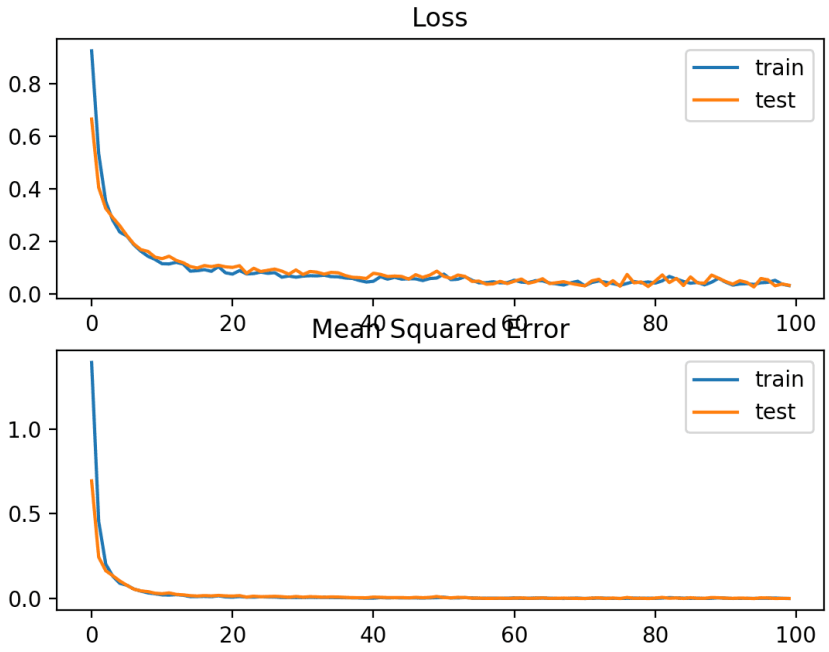
2.10. Приклади функцій активації

І останній важливий параметр – це функція втрат. Вона вимірює те, наскільки нейронна мережа добре підходить для нашої вибірки. Функція втрат може залежати від значення ваг та зміщення.

Найпоширенішими є такі функції втрат:

- квадратична (середньоквадратична);
- крос-ентропія;
- експоненціальна.

Найчастіше використовується перша з перелічених – квадратична (середньоквадратична) функція, приклад зображений на рисунку 2.11.



2.11 Графік середньоквадратичної функції втрат

2.4 Вибір інструментальних засобів для реалізації програмного забезпечення

Додаток, який потрібно розробити – це програма для робочого столу на базі операційної системи Windows. Для виконання поставлених задач буде використовуватись мова програмування C# на основі фреймворку .NET. Сам інтерфейс користувача розробляється за допомогою Windows Forms .NET.

Для роботи із зображенням та його аналізом використовується бібліотека

EmguCV, яка є своєрідною версією OpenCV для мови .NET. Матричні операції та обчислення виконуються за допомогою бібліотеки Math.NET. Розглянемо кожен з цих інструментів детальніше.

2.4.1 EmguCV

Emgu CV — це кросплатформна обгортка .NET бібліотеки для обробки зображень OpenCV. Тобто з допомогою цієї бібліотеки можна викликати функції OpenCV, використовуючи .NET-сумісні мови: C#, F#, Visual Basic. Обгортка може бути скомпільована за допомогою Visual Studio та Unity, і може працювати на Windows, Linux, Mac OS, iOS та Android.

OpenCV (Open Source Computer Vision Library) — це бібліотека комп'ютерного зору та машинного навчання з відкритим кодом. Її було створено для забезпечення загальної інфраструктури для додатків комп'ютерного зору та для прискорення використання машинного сприйняття в комерційних продуктах. OpenCV має ліцензію від Apache 2, тому користувачам легко її використовувати та змінювати код [9].

Ця бібліотека містить велику кількість алгоритмів, які використовуються для розпізнавань зображень зокрема. Саме тому було обрано саме її для обробки та аналізу зображення, на якому людина показує певний жест.

Простори імен бібліотеки EmguCV, які використовуються для реалізації поставленого завдання [10]:

1. Emgu.CV – обгортка над функціями OpenCV. Основні класи:
   - CvInvoke – базові функції;
   - VideoCapture – захоплення зображення з камери або відеофайлу;
   - Image<TColor, TDepth> – обгортка для IplImage OpenCV;
   - Mat – еквівалент cv::Mat;
2. Emgu.CV.CvEnum – OpenCV Enumeration. Основні класи:
   - CapProp – ідентифікатор властивості CV Capture;
   - Inter – типи інтерполювання, можливі значення: Nearest, Linear, Cubic, Area;

- ThresholdType – можливі значення: Binary, BinaryInv, Trunc, ToZero, ToZeroInv, Mask, Otsu, Triangle;
- RetrType – режим пошуку контуру, можливі значення: External, List, Ccomp, Tree, Floodfill;
- BorderType – тип для функції CopyMakeBorder;

3. Emgu.CV.Structure – обгортка над структурами OpenCV. Основні структури:
   - Gray;
   - MCvScalar;
4. Emgu.CV.Util – колекція збірок, які використовуються проектами Emgu.CV. Основні класи:
   - VectorOfVectorOfPoint;
   - VectorOfPoint;

Власне на рисунках 2.2 і 2.3 ми бачимо результати обробки зображення з використанням бібліотеки EmguCV. Завдяки методам, які в ній реалізовані, можна виділити контур руки, змінити колір зображення та інше.

2.4.2 Math.NET

Math.NET Numerics надає методи та алгоритми для чисельних обчислень у науці, техніці та повсякденному використанні. Основні можливості бібліотеки включають:

- лінійну алгебру;
- спеціальні функції;
- випадкові числа;
- ймовірнісні моделі;
- інтегрування;
- регресію;
- інтерполяцію;
- оптимізаційні проблеми.

Math.NET Numerics є частиною ініціативи Math.NET і є результатом злиття dnAnalytics з Math.NET Iridium, замінивши обидва. Доступний безкоштовно за ліцензією MIT. Він орієнтований на Microsoft .NET 5.0, .NET 4.6.1 і вище, а також .NET Standard 2.0 і вище. На додаток до суто керованої реалізації він також підтримує оптимізацію вбудованого обладнання [11].

2.5 Висновки до розділу

У даному розділі сформульовано задачу, яку має вирішувати інтелектуальна система. Спочатку тут було детально описано метод обробки зображення – бінаризацію, який буде виокремлювати контур долоні. Пізніше у цьому розділі було описано основну частину магістерської роботи – нейронну мережу, як вона працює та які завдання виконує.

Крім цього описано всі технології та програмні інструменти, з допомогою яких буде розроблятись продукт. Було обрано самі такі технології, тому що вони надають широкий набір інструментів, які допоможуть нам створити програмний продукт для розпізнавання жестів.

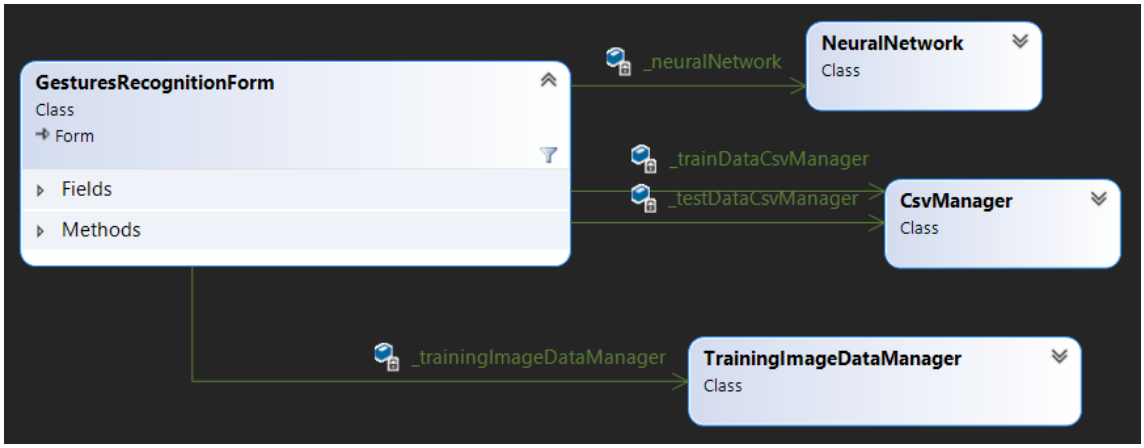
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ

3.1 Програмна реалізація

Для складної інформаційної системи було розроблено відповідну програму з використанням технологій, які були описані в попередньому розділі. Ця програма складається з кількох самостійних частин – аналізатора зображень, менеджера файлів з даними для навчання та нейронна мережа, які відображені в коді такими класами:

- GesturesRecognitionForm
- CsvManager
- TrainingImageDataManager
- NeuralNetwork

Далі детально розглянемо реалізацію кожної з перелічених компонент. На рисунку 3.1 зображено зв’язки між цими класами.



3.1 Зв’язки між програмними компонентами

3.1.1 Допоміжні класи

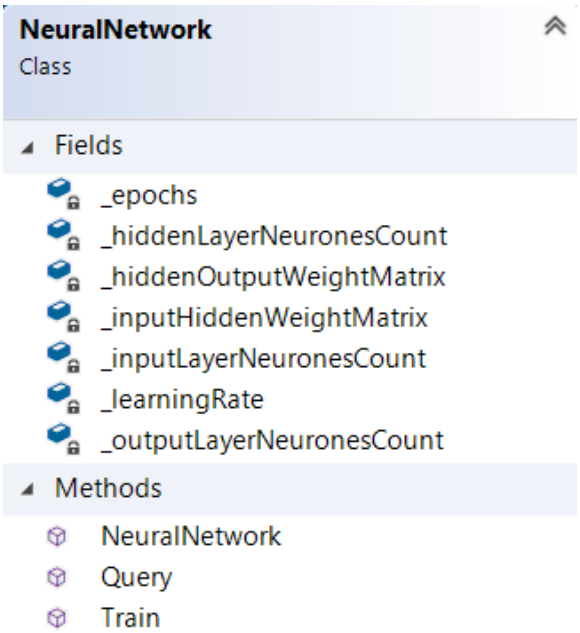
Пропоную спочатку розглянути допоміжні класи та їхні основні функції. Одним з таких класів є CsvManager. Код знаходиться у додатку А.1. Призначенням цього класу є збереження даних для тренування у файл з розширенням CSV. Клас тримає приватну інформацію про шлях збереження та ім’я файлу. Цей процес відбувається з використанням методу AppendLineToFile – який для кожного

навчального зображення додає один рядок у файл. Згодом цей файл буде зчитаний за допомогою методу AppendLineToFile і переданий у метод Train нейронної мережі. Тобто використовуючи цей файл нейронна мережа буде навчатись.

Наступним допоміжним класом є TrainingImageDataManager. Код до цього класу наведений в додатку А.2. Цей клас відповідає за збереження тренувальних фотографій на комп'ютер. Приватним полем цього класу є шлях, куди ми хочемо зберегти зображення. Клас містить лише один метод – SaveImageToFile, який приймає такі параметри: саме зображення, шлях, куди його зберегти і формат, у кому ми хочемо зберегти це зображення. Це можуть бути різні формати: JPG, PNG та інші.

3.1.2 Клас нейронної мережі

Найважливішим класом програми є саме той, в якому відбувається навчання нейромережі, та розпізнавання жестів. Цей клас називається NeuralNetwork (код до нього у додатку А.3). На рисунку 3.2 зображено діаграму цього класу.



4.2 Діаграма класу NeuralNetwork

Він містить такі приватні поля:

- _inputLayerNeuronesCount – це показник, який відповідає за кількість нейронів на вхідному шарі, у нашому випадку це буде 4096, оскільки розмір зображення, яке ми передаємо становить 64x64 пікселі;

- _hiddenLayerNeuronesCount – кількість нейронів на прихованому шарі;
- _outputLayerNeuronesCount – кількість нейронів на вихідному шарі;
- _learningRate – швидкість навчання нейромережі;
- _epochs – кількість епох, за які буде відбуватись навчання нейронної мережі.

В констукторі відбувається ініціалізація цих полів, а також ініціалізація матриць ваг, які спочатку генеруються випадковим чином. Розміри вхідної матриці ваг задаються як hiddenLayerNeuronesCount x inputLayerNeuronesCount. Відповідно розміри вихідної матриці – outputLayerNeuronesCount x hiddenLayerNeuronesCount.

При ініціалізації початкових значень не варто використовувати великі числа, адже використання функції активації для таких значень може спричинити те, що мережа не буде здатна навчатись на кращих значеннях. Таким чином, ваги обираються з нормального розподілу з центром в нулі і з стандартним відхиленням, для якого величина обернено пропорційна квадратному кореню з кількості вхідних нейронів.

У методі, який відповідає за тренування нейронної мережі спочатку відбувається розрахунок вихідних сигналів для заданого прикладу. А потім відбувається порівняння результатів із бажаною відповіддю і на основі цього оновлюються ваги.

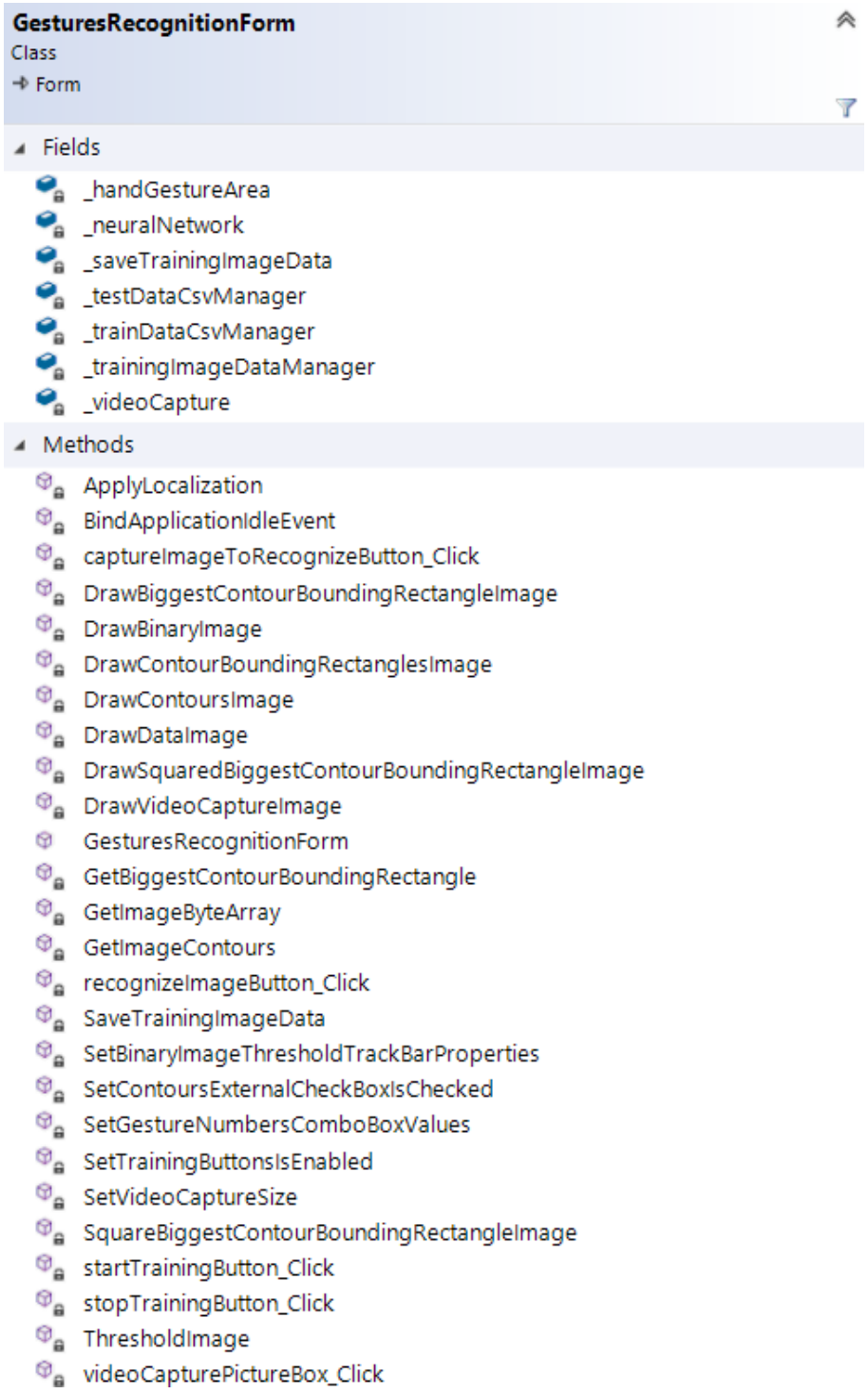
Помилка обчислюється як різниця між бажаним значенням, яке було надано самим тренувальним прикладом і фактичним значенням виходу. В коді це різниця між матрицями targetMatrix та finalLayerOutputSignals, яка розраховується поелементно.

Далі потрібно порахувати зворотне поширення помилок для вузлів у прихованому шарі і згодом оновити значення ваг зв’язків між вихідним і прихованим шарами, а потім між прихованим і вхідним.

3.1.3 Основний клас програми

Тепер розглянемо GesturesRecognitionForm – основний клас програми. Код наведений у додатку А.4. Він містить об’єкти усіх попередньо описаних класів

(композиція, зв’язок «has a»). Об’єкти класів CsvManager, TrainingImageDataManager та NeuralNetwork є приватними полями класу GesturesRecognitionForm. GesturesRecognitionForm також містить методи для аналізу зображення. І з класової діаграми 3.3 бачимо, що цей клас не тільки відповідає за функціонування елементів на формі, а й за саму обробку зображень.



3.3 Діаграма класу GesturesRecognitionForm

Розглянемо детально усі етапи роботи програми. Спочатку потрібно зчитати зображення з вебкамери комп’ютера. Для цього створюється об’єкт класу VideoCapture. Цей клас був описаний в попередньому розділі (2.4.1).

Взагалі, зчитування повного зображення – досить ресурсозатратна задача, тому було вирішено розміщувати жест у визначеному наперед прямокутнику. Тобто одразу на оригінальному відеопотоці буде зображений прямокутник, в який користувач має помістити руку. І до уваги буде братись лише те зображення, яке розміщене безпосередньо в цьому прямокутнику.

На рисунку 3.4 бачимо виділений прямокутник, в який поміщена рука.

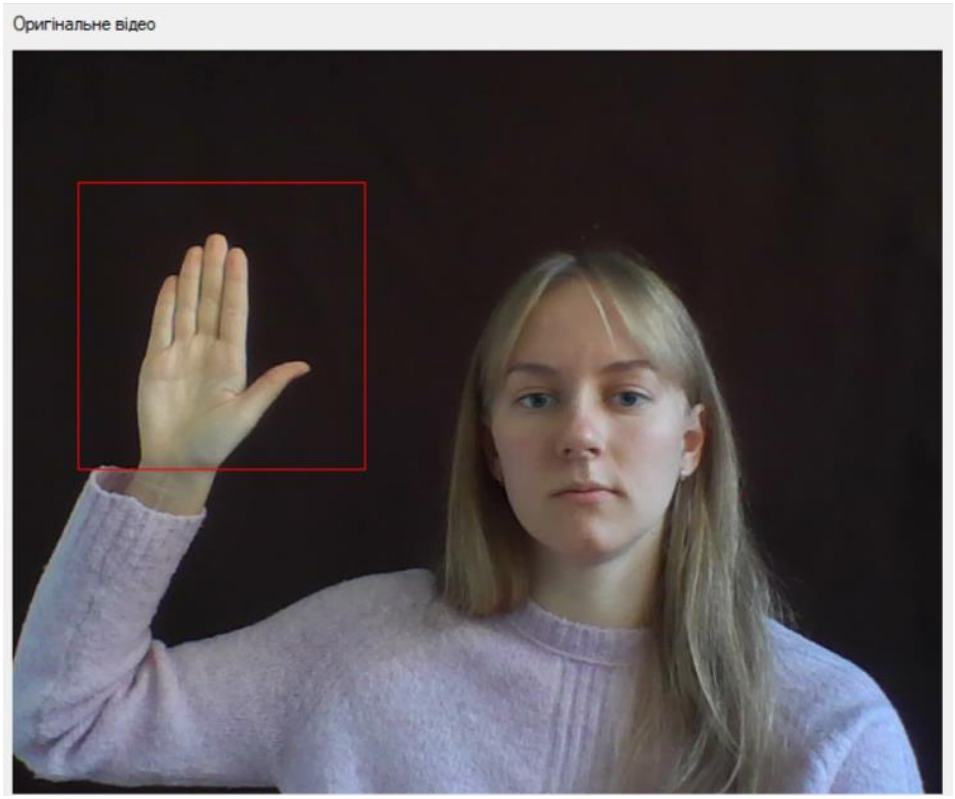


Рис. 3.4 Виділений прямокутник для жесту

Наступним етапом потрібно опрацювати зображення, яке виділене в прямокутнику так, щоб можна було його передати у нейронну мережу для навчання. Для цього спочатку необхідно виконати бінаризацію зображення.

Для проведення бінаризації застосовується метод бібліотеки EmguCV Threshold. Він приймає поріг бінаризації, значення якого береться з елементу на формі. Візуалізація цього зображення виконується за допомогою методу DrawBinaryImage. Результат бінаризації зображення наведено на рисунку 3.5.

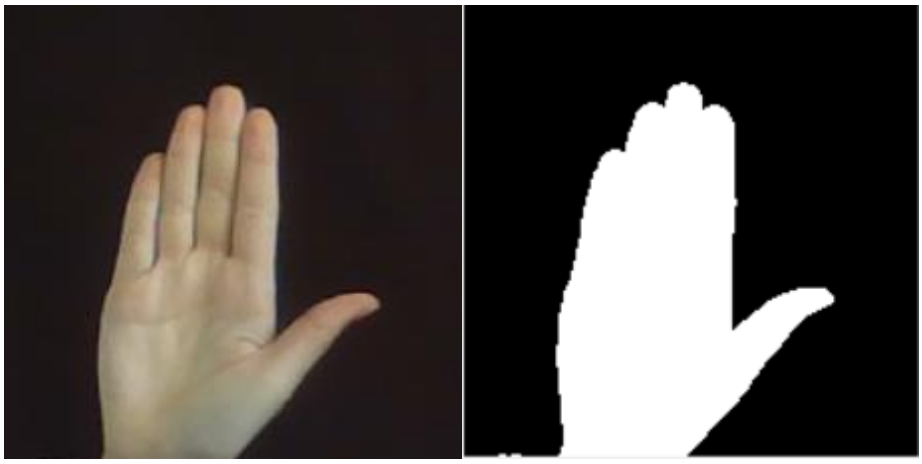


Рис. 3.5 Бінаризація зображення

Після того, як ми отримали бінаризоване зображення потрібно виділити його контури, адже саме контури потрібні для того, щоб потім сформувати дані, які передадуться у нейронну мережу. Для цього використовується метод бібліотеки EmguCV DrawContours. Параметрами цього методу є колір контуру та його товщина. Результат роботи методу продемонстровано на рисунку 3.6.

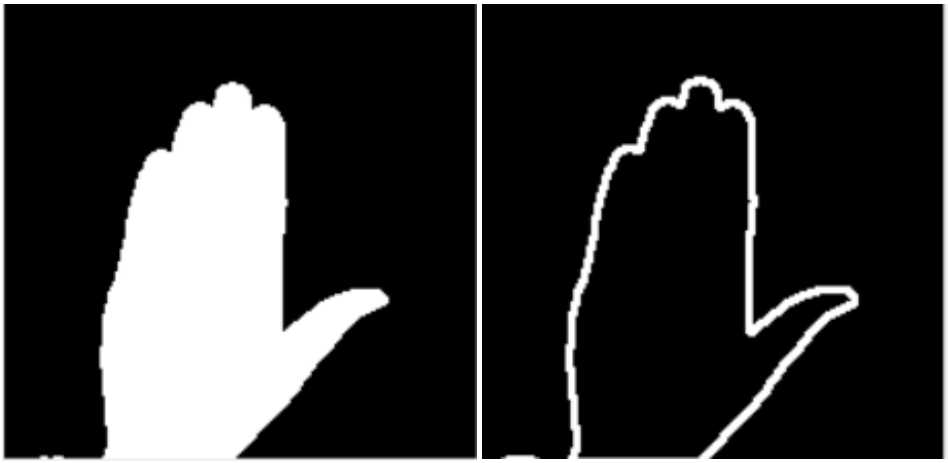


Рис. 3.6 Виділення контурів

Тепер слід локалізувати усі контури, які знаходяться на зображенні, адже очевидно, що їх може бути декілька. Тому для кожного замкнутого контуру, який знайдений на зображені, викликається метод BoundingRectangle, який навколо кожного з них обмальовує прямокутник. Результат роботи методу продемонстровано на рисунку 3.7. У нашому прикладі лише один контур і відповідно лише для нього є намальований прямокутник.

Рис. 3.7 Локалізація всіх знайдених контурів

Після того, як ми локалізували усі контури на зображенні слід виділити найбільший з них, щоб працювати конкретно з ним. В коді за це відповідає метод DrawBiggestContourBoundingRectangleImage. Для попереднього прикладу ця процедура дасть такий самий результат, як і звичайний виклик методу BoundingRectangle, проте для випадку, коли локалізовано кілька контурів, буде інший результат. Його можна спостерігати на рисунку 3.8.

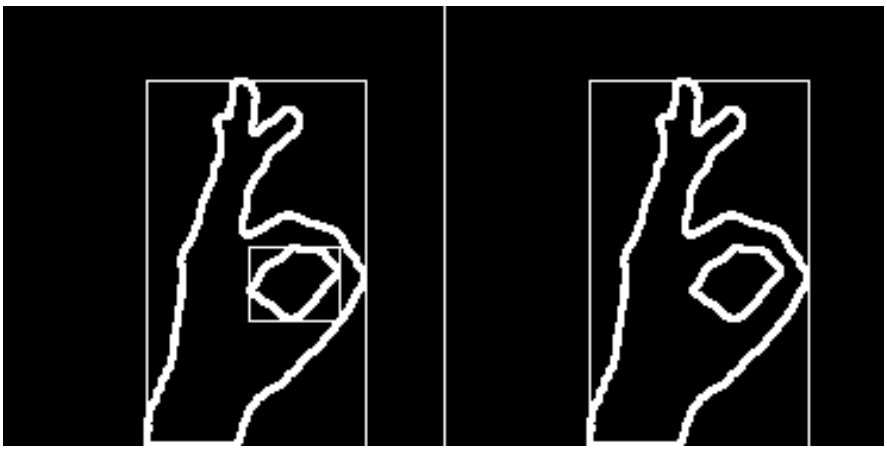


Рис. 3.8 Виділення найбільшого контуру

Також є метод, який відповідає за відсікання зайвого простору із зображення. В результаті роботи функції на рисунку залишається лише контур долоні – решта зайвого простору навколо ігнорується. В коді для цього викликається метод DrawSquaredBiggestContourBoundingRectangleImage. Результат виконання цього методу продемонстровано на рисунку 3.9.

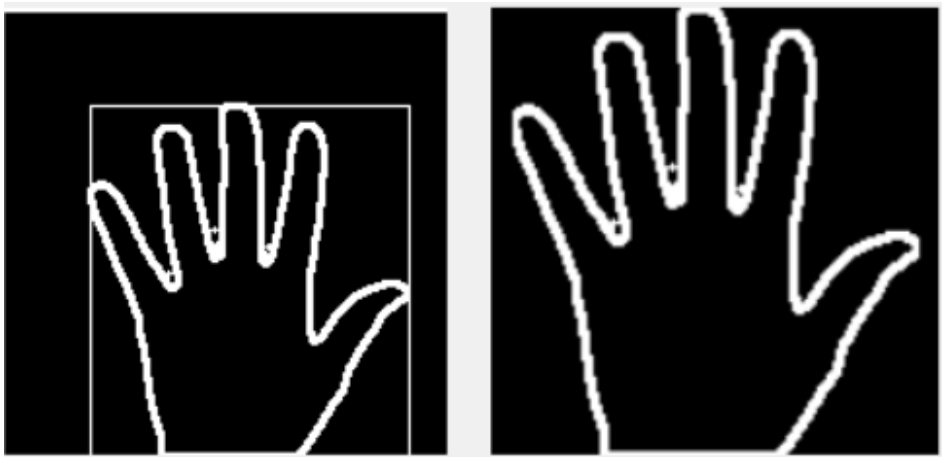


Рис. 3.9 Результат відсікання зайвого простору

При використанні нейронної мережі буде використовуватись зображення розміром 64 на 64 пікселі, тому потрібно виконати необхідні перетворення останнього зображення. А саме – викликати метод Resize бібліотеки EmguCV. В коді також реалізований метод DrawDataImage, який виводить масштабоване зображення для користувача. Результат зміни розміру продемонстровано на рисунку 3.10.

Рис. 3.10 Контур для навчання нейронної мережі

На цьому етапі обробка зображення завершена. Тобто всі необхідні операції щодо вхідного зображення вже виконані і воно готове для передачі у нейронну мережу. В коді є метод SaveTrainingImageData, який використовує допоміжні сервіси TrainingImageDataManager і CsvManager для збереження зображення у форматі csv. І власне в такому вигляді нейронна мережа отримує зображення, і на цих даних вона здатна навчатись.

Після цього користувач може натиснути на кнопку Захопити зображення і воно з’явиться у вікні нижче. Це продемонстровано на рисунку 3.11.

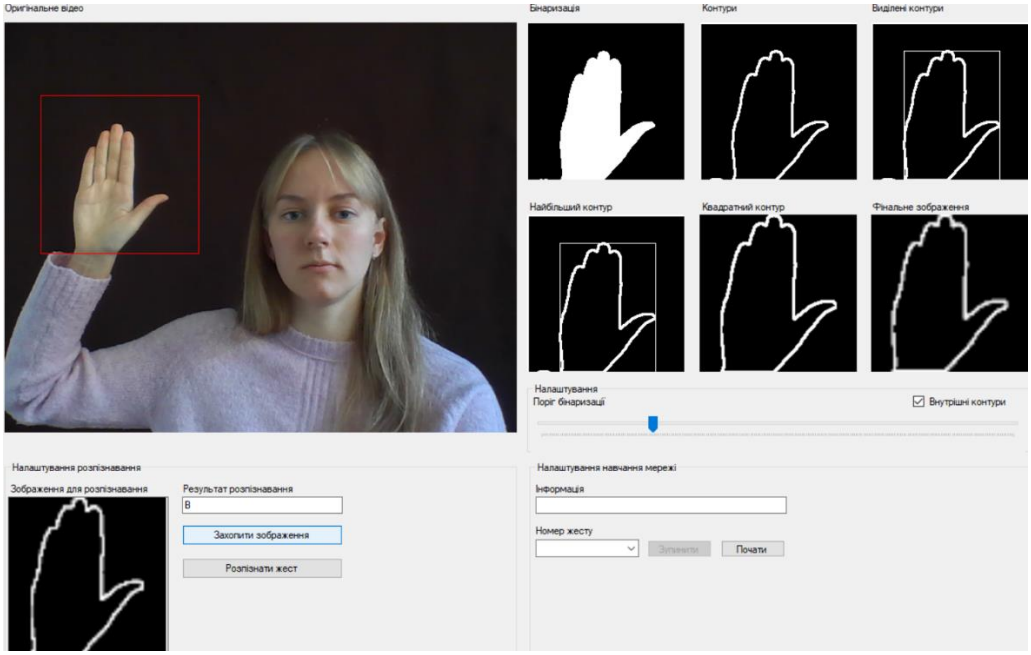


Рис. 3.11 Захоплення жесту для подальшого навчання/розпізнавання

Тепер ми можемо вказати якій літері відповідає вказаний жест і почати навчати нейромережу з таких прикладів. Для цього в полі Інформація вводимо літеру і нижче номер (ідентифікатор, який потрібен для запису зображення у таблицю csv), який їй відповідає і натискаємо Почати тренування, після цього програма автоматично буде захоплювати зображення кожні 2 секунди й користувачеві потрібно змінювати положення руки, щоб система з різних ракурсів могла потім розпізнати жест. В коді викликається метод Train об’єкта _neuralNetwork. Тобто це виглядає так:

```
_neuralNetwork.Train(_testDataCsvManager.ReadAllLines());
```

І останній етап виконання програми – це саме розпізнавання жесту(рис. 3.11). Після натискання на кнопку Розпізнати жест програма звертається до методу Query об’єкта _neuralNetwork. Це виглядає наступним чином:

```
_neuralNetwork.Query(GetImageByteArray(imageToRecognize));
```

3.2 Результати роботи системи

В результаті роботи розроблено додаток, який здатний обробити зображення жесту, який належить алфавіту української жестової мови, передати його у

нейронну мережу для навчання, провести тренування, а згодом розпізнати показаний жест з допомогою раніше навченої нейромережі.

Для наглядності програма видає результат на кожній ітерації, починаючи з необробленого відео та завершуючи результуючим зображенням.

Загальний інтерфейс користувача зображено на рисунку 3.12.

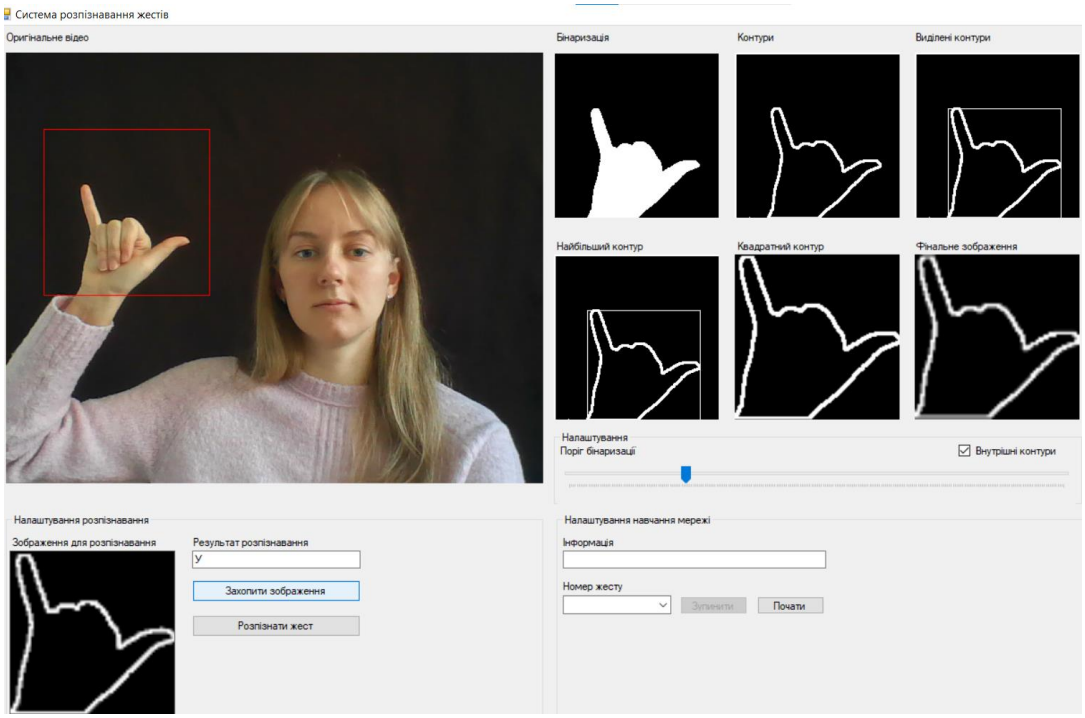


Рис. 3.12 Інтерфейс користувача

Увесь процес завжди починається із захоплення відео. Приклад кадру з оригінального, необробленого відеопотоку зображене на рисунку 3.13.

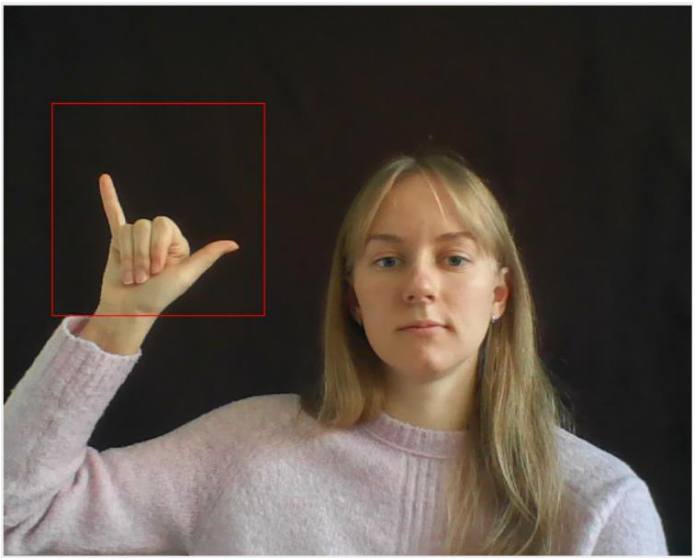


Рис. 3.13 Оригінальний кадр

Після захоплення відео з використанням однокамерної системи, оригінальне зображення бінаризується, після цього виділяються його контури, локалізується найбільший з них, відсікається частина зображення, на якій немає жесту, і, зрештою, це зображення перетворюється у потрібний розмір розміром 64x64 пікселі.

Кожна з перелічених ітерацій послідовно зображена на рисунку 3.14.

Рис. 3.14 Всі етапи обробки зображення

На цьому ж рисунку бачимо, що задається так званий поріг бінаризації, який відповідає за те, щоб контури були достатньо чіткими. Його потрібно налаштувати вручну, в залежності від освітлення приміщення та фону. Для того, що це зробити, потрібно переміщати контролер вправо або вліво і, власне, спостерігати за результатами, чи зображення стає чіткішим, чи навпаки.

Якщо підібрати цей поріг некоректно, то можна отримати погані результати з нечітким контуром зображення, або великою кількістю перешкод на фоні, як це бачимо на рисунку 3.15 та 3.16.

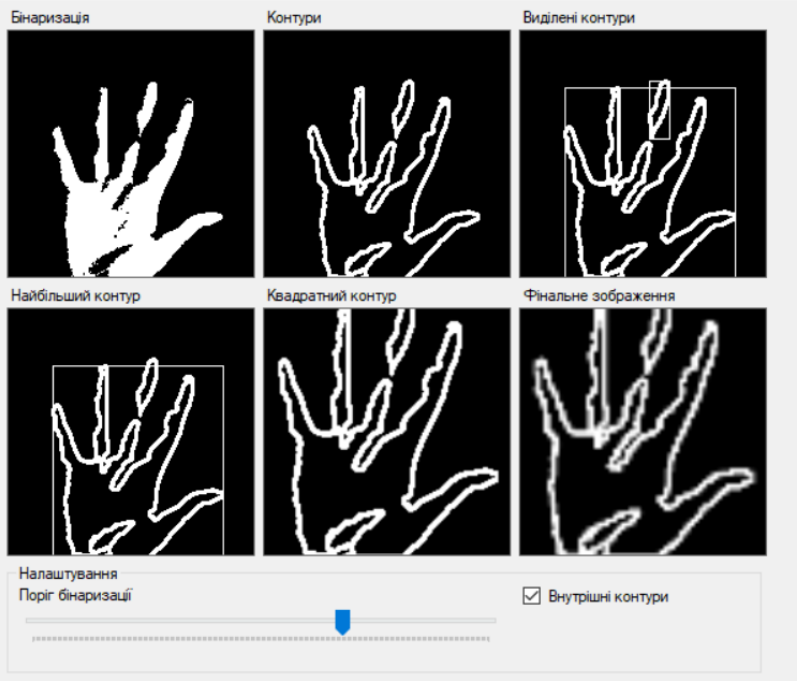


Рис. 3.15 Результат погано підібраного порогу бінаризації – поріг зависокий

Рис. 3.16 Результат погано підібраного порогу бінаризації – поріг занизький

Після того, як ми налаштували поріг бінаризації таким чином, щоб контур був достатньо чітким, потрібно натиснути на кнопку Захопити зображення, після цього воно з'явиться у вікні нижче.

Тепер можна тренувати нейромережу на таких прикладах, для цього на користувацькому інтерфейсі є секція, яка називається Налаштування навчання мережі (рис. 3.17). Після введення необхідної інформації для тренування,

користувач має натиснути кнопку Почати – після чого почнеться процес навчання нейромережі. Кожне зображення буде збережене у раніше визначеній директорії і відповідний рядок буде додано у CSV таблицю. Кожне таке зображення, перетворене у цифри, буде додаватись у раніше згадану таблицю новим рядком з ідентифікатором на початку цього рядка.

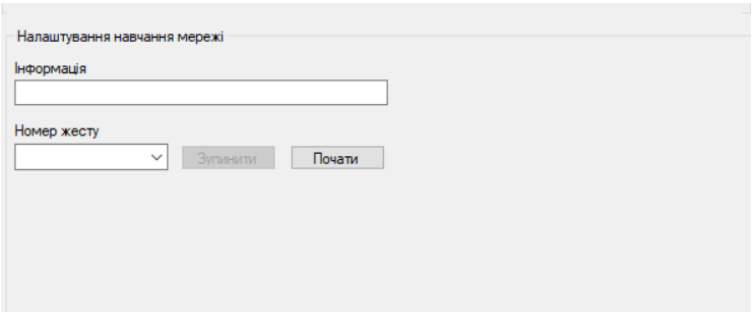


Рис. 3.17 Секція для навчання нейронної мережі

Після успішного тренування можна переходити до процесу розпізнавання. Для цього користувачеві потрібно показати жест, який він хоче розпізнати, натиснути на кнопку Захопити зображення та згодом Розпізнати зображення. І тоді в полі Результат розпізнавання з’явиться літера, якій відповідає показаний жест. Секція з захопленням та розпізнаванням зображення показана на рисунку 3.18.

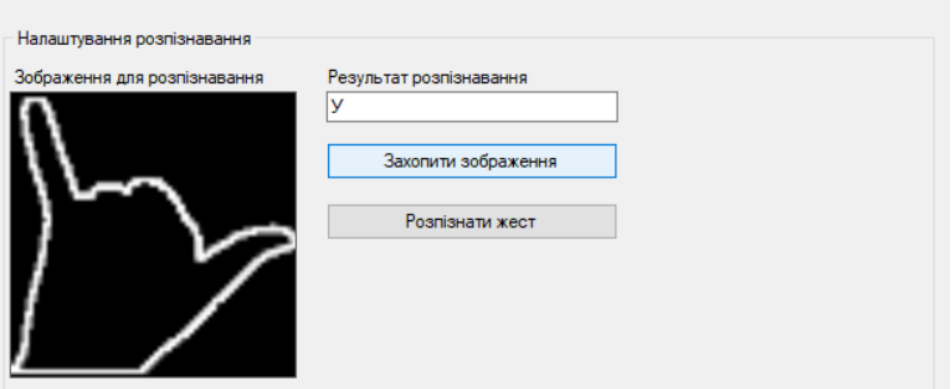


Рис. 3.18 Секція для розпізнавання жесту

Таким чином розроблено програму, яка повноцінно виконує процеси навчання нейронної мережі, а також успішно розпізнає показані жести. Ще кілька прикладів розпізнавання зображений на рисунках 3.19 і 3.20. Як бачимо, в усіх випадках система вдало розпізнала показаний жест. Можна це перевірити у таблиці, яка показує відповідність жестів та літер (рис. 1.1).

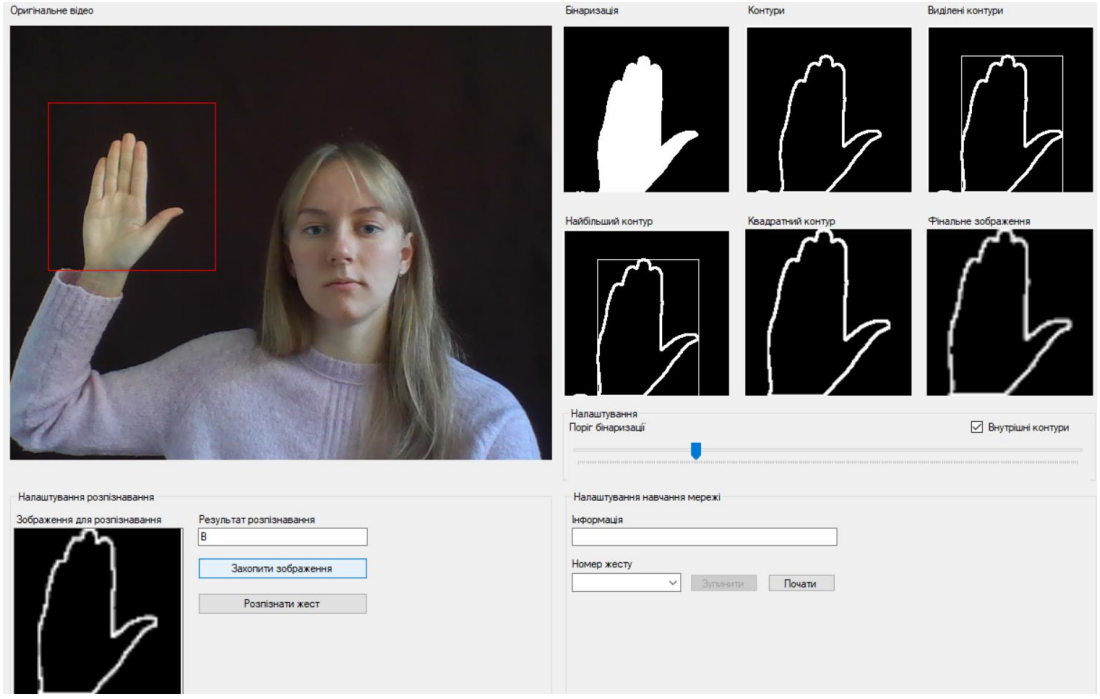


Рис. 3.19 Приклад розпізнавання літери В

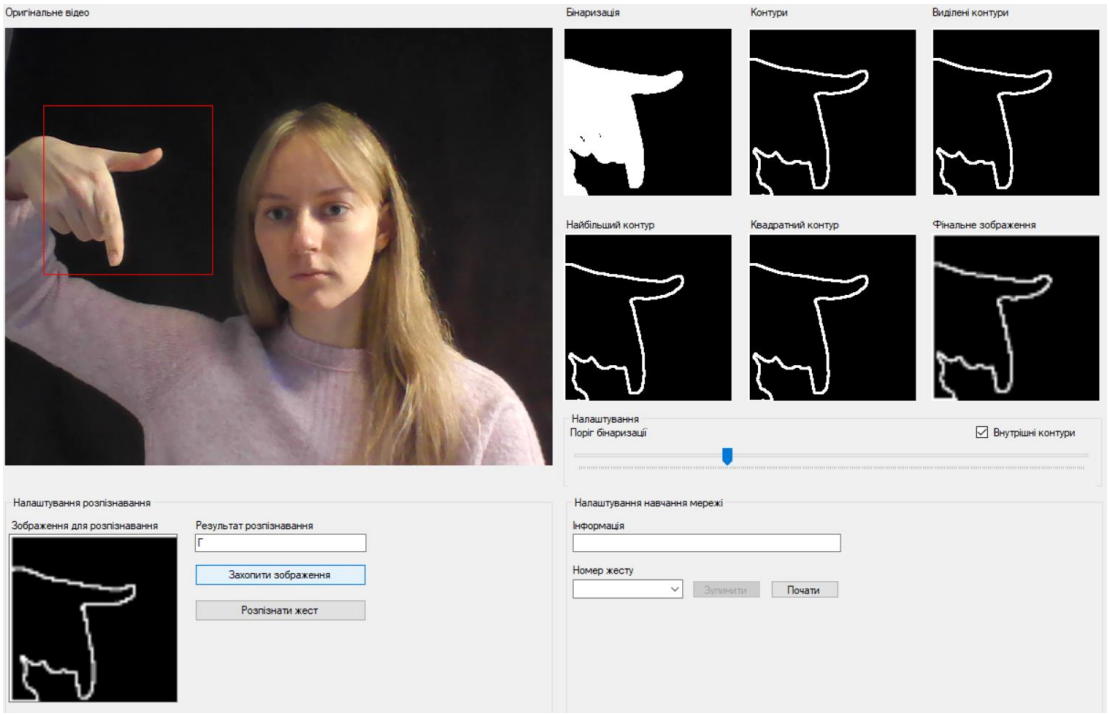


Рис. 3.20 Приклад розпізнавання літери Г

3.3 Висновки до розділу

У цьому розділі пояснено програмну реалізацію інтелектуальної інформаційної системи для розпізнавання мови жестів. Для кожного етапу розробки наведені приклади коду і знімки екрана з результатами.

Тут було детально описано, які етапи обробки зображення повинні бути

виконані для успішного навчання нейронної мережі. Крім цього, тут було показано, як виглядає сама нейронна мережа, як навчається, та як безпосередньо розпізнає жести. Також показано приклади різних жестів та їх перетворення, процес тренування та розпізнавання зображень.

У результаті представлено користувацький інтерфейс, який дозволяє зручно тренувати нейронну мережі, а також розпізнавати зображення жестів.

ВИСНОВКИ

У результаті виконання даної магістерської роботи реалізовано інтелектуальну інформаційну систему для розпізнавання мови жестів.

Оскільки проблема розуміння мови жестів сьогодні є вкрай актуальною, саме це було обрано напрямком розвитку інформаційної системи. В першому розділі детально описано предметну область, а також проведено аналіз технологій для збору даних. В результаті аналізу вирішено, що найкращим вибором буде звичайна вебкамера комп'ютера, оскільки цей пристрій є доступний практично кожному.

В наступному розділі було пояснено метод, який використовується для обробки зображення і виділення контуру долоні, а саме – бінаризацію. Пізніше було коротко викладено інформацію про інструменти, які застосовуються для побудови інформаційної системи. До цих інструментів належать такі технології: мова програмування C# (фреймворк .NET), Windows Forms .NET, EmguCV та Math.NET.

Також було продемостровано те, як працює додаток. Наведено достатньо знимків, щоб зрозуміти кожен етап обробки зображення, формування даних для навчання, тренування нейронної мережі та безпосередньо розпізнавання поданих жестів. Весь важливий функціонал також підкріплено прикладами коду і детальними поясненнями.

У підсумку розроблений програмний продукт здатний полегшити спілкування між людьми, які володіють мовою жестів та людьми, які її не знають. Адже додаток опрацьовує жест таким чином, що на виході користувач отримує літеру українського алфавіту, яка відповідає показаному жесту. Таким чином, один із співрозмовників може показувати жест, а інший читати цей жест у звичному для себе вигляді.

СПИСОК ЛІТЕРАТУРИ

1. Deafness and hearing loss [Електронний ресурс] – Режим доступу: https://www.who.int/health-topics/hearing-loss
2. Sturman D. J. A survey of glove-based input / Sturman D.J., Zeltzer D. - IEEE Computer Graphics and Applications, 1994. – 302 с.
3. Menache A. Understanding Motion Capture for Computer Animation 2nd Edition / Alberto Menache. – Morgan Kaufmann, 2010. – 276 с.
4. Borenstein G. Making Things See: 3D Vision with Kinect, Processing, Arduino, and MakerBot / Greg Borenstein. – Maker Media, Inc, 2012. – 436 с.
5. Фурман Я. А. Цифровые методы обработки и распознавания бинарных изображений / Я. А. Фурман, А. Н. Юрьев, В. В. Яншин – Красноярск: Изд-во Краснояр, 1992. – 248 с.
6. Фёдоров А. Бинаризация черно-белых изображений: состояние и перспективы развития / А. Фёдоров. – Москва: Техносфера, 2006. – 616 с.
7. Новотарський М. А. Штучні нейронні мережі: обчислення / М. А. Новотарський, Б. Б. Нестеренко. – Київ: Інститут математики НАН України, 2004. – 22-25 с.
8. Rashid T. Make Your Own Neural Network / Tariq Rashid. – CreateSpace Independent Publishing Platform, 2016. – 62 с.
9. OpenCV [Електронний ресурс] – Режим доступу: https://opencv.org/about/
10. EmguCV [Електронний ресурс] – Режим доступу: https://www.emgu.com/wiki/files/4.6.0/document/html/R_Project_Emgu_CV_Documentation.htm
11. Math.NET [Електронний ресурс] – Режим доступу: https://numerics.mathdotnet.com/

ДОДАТОК А. КОД ПРОГРАМИ

А.1 Клас CsvManager

```
using System.IO;
using System.Text;

namespace GestureRecognition.Managers
{
    public class CsvManager
    {
        private readonly string _csvPath;
        private readonly string _fileName;

        public CsvManager(string csvPath, string fileName)
        {
            _csvPath = csvPath;
            _fileName = fileName;
        }

        public void AppendLineToFile(string line)
        {
            var stringBuilder = new StringBuilder();
            stringBuilder.AppendLine(line);
            File.AppendAllText($@"{_csvPath}\{_fileName}.csv",
                stringBuilder.ToString());
        }

        public string[] ReadAllLines()
        {
            return File.ReadAllLines($@"{_csvPath}\{_fileName}.csv");
        }
    }
}
```

А.2 Клас TrainingImageDataManager

```
using System;
using Emgu.CV;
using Emgu.CV.Structure;

namespace GestureRecognition.Managers
{
    public class TrainingImageDataManager
    {
        private readonly string _imageDataPath;

        public TrainingImageDataManager(string imageDataPath)
        {
            _imageDataPath = imageDataPath;
        }

    public void SaveImageToFile(Image<Gray, byte> imageToSave, string fileName,
string imageFormat)
        {
            imageToSave.Save(string.Format(@"{0}\{1}_{2}.{3}",
_imageDataPath,
                DateTime.Now.Ticks.ToString(),
                fileName,
                imageFormat));
        }
    }
}
```

А.3 Клас NeuralNetwork

```
using System;
using System.Collections.Generic;
using System.Linq;
using MathNet.Numerics;
using MathNet.Numerics.Distributions;
using MathNet.Numerics.LinearAlgebra;

namespace GestureRecognition.NeuralNetwork
{
    public class NeuralNetwork
    {
        private readonly int _inputLayerNeuronesCount; // кількість нейронів
на вхідному шарі
        private readonly int _hiddenLayerNeuronesCount; // кількість
нейронів на прихованому шарі
        private readonly int _outputLayerNeuronesCount; // кількість
нейронів на вихідному шарі
        private readonly double _learningRate; // швидкість навчання
        private readonly int _epochs; // кількість епох

        private Matrix<double> _inputHiddenWeightMatrix;
```

```
        private Matrix<double> _hiddenOutputWeightMatrix;

        public NeuralNetwork(int inputLayerNeuronesCount,
            int hiddenLayerNeuronesCount,
            int outputLayerNeuronesCount,
            double learningRate,
            int epochs)
        {
            _inputLayerNeuronesCount = inputLayerNeuronesCount;
            _hiddenLayerNeuronesCount = hiddenLayerNeuronesCount;
            _outputLayerNeuronesCount = outputLayerNeuronesCount;
            _learningRate = learningRate;
            _epochs = epochs;

            var inputHiddenWeightMatrixNormalDistribution = new Normal(0,
Math.Pow(hiddenLayerNeuronesCount, -0.5));
            var hiddenOutputWeightMatrixNormalDistribution = new Normal(0,
Math.Pow(outputLayerNeuronesCount, -0.5));

            _inputHiddenWeightMatrix = Matrix<double>
                .Build
                .Random(hiddenLayerNeuronesCount, inputLayerNeuronesCount,
inputHiddenWeightMatrixNormalDistribution);
            _hiddenOutputWeightMatrix = Matrix<double>
                .Build
                .Random(outputLayerNeuronesCount, hiddenLayerNeuronesCount,
hiddenOutputWeightMatrixNormalDistribution);
        }

        // метод для тренування нейромережі
        public void Train(string[] trainInputLines)
        {
            for (int e = 0; e < _epochs; e++)
            {
                foreach (var trainInputLine in trainInputLines)
                {
                    var trainInputLineDoubleArray = Array
                            .ConvertAll(trainInputLine.Split(','),
double.Parse)
                            .ToArray();

                    var inputMatrix = Matrix<double>
                        .Build
                        .Dense(1, _inputLayerNeuronesCount,
trainInputLineDoubleArray.Skip(1).Select(x => (x / 255 * 0.99) + 0.01).ToArray())
                        .Transpose();

                    var targetMatrix = Matrix<double>
                        .Build
                        .Dense(1, _outputLayerNeuronesCount, (rowIndex,
columnIndex) => columnIndex == trainInputLineDoubleArray.First() ? 0.99 : 0.01)
                        .Transpose();
```

```
                        var hiddenLayerInputSignals =
_inputHiddenWeightMatrix.Multiply(inputMatrix);
                        var hiddenLayerOutputSignals = Matrix<double>
                            .Build
                            .Dense(hiddenLayerInputSignals.RowCount, 1,
(rowIndex, columnIndex) =>
SpecialFunctions.Logistic(hiddenLayerInputSignals[rowIndex, columnIndex]));

                        var finalLayerInputSignals =
_hiddenOutputWeightMatrix.Multiply(hiddenLayerOutputSignals);
                        var finalLayerOutputSignals = Matrix<double>
                            .Build
                            .Dense(finalLayerInputSignals.RowCount, 1,
(rowIndex, columnIndex) =>
SpecialFunctions.Logistic(finalLayerInputSignals[rowIndex, columnIndex]));

                        var outputLayerErrors = targetMatrix -
finalLayerOutputSignals;
                        var hiddenLayerErrors =
_hiddenOutputWeightMatrix.Transpose().Multiply(outputLayerErrors);

                        _hiddenOutputWeightMatrix += _learningRate * (
                                Matrix<double>
                                .Build
                                .Dense(_outputLayerNeuronesCount, 1, (rowIndex,
columnIndex) => outputLayerErrors[rowIndex, columnIndex] *
finalLayerOutputSignals[rowIndex, columnIndex] * (1 -
finalLayerOutputSignals[rowIndex, columnIndex]))
                            ).Multiply(hiddenLayerOutputSignals.Transpose());

                        _inputHiddenWeightMatrix += _learningRate * (
                                Matrix<double>
                                .Build
                                .Dense(_hiddenLayerNeuronesCount, 1, (rowIndex,
columnIndex) => hiddenLayerErrors[rowIndex, columnIndex] *
hiddenLayerOutputSignals[rowIndex, columnIndex] * (1 -
hiddenLayerOutputSignals[rowIndex, columnIndex]))
                            ).Multiply(inputMatrix.Transpose());
                    }
                }
            }

            // метод для розпізнавання вхідного зображення
            public Matrix<double> Query(List<byte> inputArray)
            {
                var inputMatrix = Matrix<double>
                    .Build
                    .Dense(1, _inputLayerNeuronesCount, inputArray.Select(x =>
(x / 255 * 0.99) + 0.01).ToArray())
                    .Transpose();
```

```
                var hiddenLayerInputSignals =
_inputHiddenWeightMatrix.Multiply(inputMatrix);
                var hiddenLayerOutputSignals = Matrix<double>
                    .Build
                    .Dense(hiddenLayerInputSignals.RowCount, 1, (rowIndex,
columnIndex) => SpecialFunctions.Logistic(hiddenLayerInputSignals[rowIndex,
columnIndex]));

                var finalLayerInputSignals =
_hiddenOutputWeightMatrix.Multiply(hiddenLayerOutputSignals);
                var finalLayerOutputSignals = Matrix<double>
                    .Build
                    .Dense(finalLayerInputSignals.RowCount, 1, (rowIndex,
columnIndex) => SpecialFunctions.Logistic(finalLayerInputSignals[rowIndex,
columnIndex]));

                return finalLayerOutputSignals;
            }
        }
    }
```

А.4 Клас GesturesRecognition

```
    namespace GestureRecognition
    {
        public partial class GesturesRecognitionForm : Form
        {
            private VideoCapture _videoCapture;
            private Rectangle _handGestureArea;
            private bool _saveTrainingImageData;
            private TrainingImageDataManager _trainingImageDataManager;
            private CsvManager _trainDataCsvManager;
            private NeuralNetwork.NeuralNetwork _neuralNetwork;
            private CsvManager _testDataCsvManager;

            public GesturesRecognitionForm()
            {
                InitializeComponent();
                ApplyLocalization();
                SetBinaryImageThresholdTrackBarProperties();

                _videoCapture = new VideoCapture();
                _handGestureArea = new
Rectangle(Constants.HAND_GESTURE_AREA_LOCATION_X,
                    Constants.HAND_GESTURE_AREA_LOCATION_Y,
                    Constants.HAND_GESTURE_AREA_WIDTH,
                    Constants.HAND_GESTURE_AREA_HEIGHT);
                _trainingImageDataManager = new TrainingImageDataManager(@"");
                _trainDataCsvManager = new CsvManager(@"", "");
                _testDataCsvManager = new CsvManager(@"", "");
```

```
                _neuralNetwork = new NeuralNetwork.NeuralNetwork(4096, 200, 10,
0.1, 5);
                _saveTrainingImageData = false;

                SetGestureNumbersComboBoxValues();
                SetTrainingButtonsIsEnabled();
                SetContoursExternalCheckBoxIsChecked();
                SetVideoCaptureSize();
                BindApplicationIdleEvent();
            }
            private void ApplyLocalization()
            {
                this.Text =
GestureRecognitionLocalization.HandGesturesRecognitionFormTitleText;
                this.videoCaptureLabel.Text =
GestureRecognitionLocalization.VideoCaptureLabelText;
                this.binaryImageLabel.Text =
GestureRecognitionLocalization.BinaryImageLabelText;
                this.contoursLabel.Text =
GestureRecognitionLocalization.ContoursLabelText;
                this.contourBoundingRectanglesLabel.Text =
GestureRecognitionLocalization.ContourBoundingRectanglesLabelText;
                this.biggestContourBoundingRectangleLabel.Text =
GestureRecognitionLocalization.BiggestContourBoundingRectangleLabelText;
                this.squaredBiggestContourBoundingRectangleLabel.Text =
GestureRecognitionLocalization.SquaredBiggestContourBoundingRectangleLabel;
                this.dataImageLabel.Text =
GestureRecognitionLocalization.DataImageLabelText;
                this.optionsGroupBox.Text =
GestureRecognitionLocalization.OptionsGroupBoxText;
                this.binaryImageThresholdTrackBarLabel.Text =
GestureRecognitionLocalization.BinaryImageThresholdTrackBarLabelText;
                this.areContoursExternalCheckBox.Text =
GestureRecognitionLocalization.AreContoursExternalCheckBoxText;
                this.recognitionOptionsGroupBox.Text =
GestureRecognitionLocalization.RecognitionOptionsGroupBoxText;
                this.imageToRecognizeLabel.Text =
GestureRecognitionLocalization.ImageToRecognizeLabelText;
                this.recognitionResultLabel.Text =
GestureRecognitionLocalization.RecognitionResultLabelText;
                this.captureImageToRecognizeButton.Text =
GestureRecognitionLocalization.CaptureImageToRecognizeButtonText;
                this.recognizeImageButton.Text =
GestureRecognitionLocalization.RecognizeImageButtonText;
                this.trainingOptionsGroupBox.Text =
GestureRecognitionLocalization.TrainingOptionsGroupBoxText;
                this.infoLabel.Text =
GestureRecognitionLocalization.InfoLabelText;
                this.gestureNumberLabel.Text =
GestureRecognitionLocalization.GestureNumberLabelText;
                this.stopTrainingButton.Text =
GestureRecognitionLocalization.StopTrainingButtonText;
```

```
                this.startTrainingButton.Text =
GestureRecognitionLocalization.StartTrainingButtonText;
            }
            private void SetBinaryImageThresholdTrackBarProperties()
            {
                binaryImageThresholdTrackBar.Minimum =
Constants.MIN_COLOR_VALUE;
                binaryImageThresholdTrackBar.Maximum =
Constants.MAX_COLOR_VALUE;
                binaryImageThresholdTrackBar.Value =
Constants.DEFAULT_THRESHOLD_VALUE;
            }

            private void SetGestureNumbersComboBoxValues()
            {
                for (int i = 0; i < 10; i++)
                {
                    gestureNumberComboBox.Items.Add(i);
                }
            }

            private void SetTrainingButtonsIsEnabled()
            {
                startTrainingButton.Enabled = !_saveTrainingImageData;
                stopTrainingButton.Enabled = _saveTrainingImageData;
            }
            private void SetContoursExternalCheckBoxIsChecked()
            {
                areContoursExternalCheckBox.Checked = true;
            }

            private void SetVideoCaptureSize()
            {
                _videoCapture.Set(CapProp.FrameWidth,
videoCapturePictureBox.Width);
                _videoCapture.Set(CapProp.FrameHeight,
videoCapturePictureBox.Height);
            }

            private void BindApplicationIdleEvent()
            {
                Application.Idle += delegate
                {
                    var videoCaptureImageMatrix = _videoCapture.QueryFrame();
                    DrawVideoCaptureImage(videoCaptureImageMatrix);

                    using (var grayImage = videoCaptureImageMatrix.ToImage<Gray,
byte>())
                    {
                        grayImage.ROI = _handGestureArea;
                        using (var binaryImage = new Image<Gray, byte>(new
Size()
```

```
                            {
                                Width = Constants.HAND_GESTURE_AREA_WIDTH,
                                Height = Constants.HAND_GESTURE_AREA_HEIGHT
                            }))
                            {
                                ThresholdImage(grayImage, binaryImage);
                                DrawBinaryImage(binaryImage);

                                using (var contoursImage = new Image<Gray, byte>(new
Size()
                                {
                                    Width = Constants.HAND_GESTURE_AREA_WIDTH,
                                    Height = Constants.HAND_GESTURE_AREA_HEIGHT
                                }))
                                {
                                    var contours = GetImageContours(binaryImage);
                                    DrawContoursImage(contoursImage, contours);

                                    using (var contourBoundingRectanglesImage = new
Image<Gray, byte>(contoursImage.Data))
                                    {
                                        DrawContourBoundingRectanglesImage(contours,
contourBoundingRectanglesImage);

                                        using (var
biggestContourBoundingRectangleImage =
                                               new Image<Gray,
byte>(contoursImage.Data))
                                        {
                                            var biggestContourBoundingRectangle =
contours.Size > 0
                                                ?
GetBiggestContourBoundingRectangle(contours)
                                                : binaryImage.ROI;


DrawBiggestContourBoundingRectangleImage(biggestContourBoundingRectangleImage,
                                                biggestContourBoundingRectangle);

                                            using (var copyMakeBorderInputImage =
new Image<Gray, byte>(contoursImage.Data))
                                            {
                                                using (var
squaredBiggestContourBoundingRectangleMat = new Mat())
                                                {

SquareBiggestContourBoundingRectangleImage(biggestContourBoundingRectangle,
                                                           copyMakeBorderInputImage,
squaredBiggestContourBoundingRectangleMat);

                                                    using (var
squaredBiggestContourBoundingRectangleImage =
```

```
squaredBiggestContourBoundingRectangleMat.ToImage<Gray, byte>())
                                                    {
DrawSquaredBiggestContourBoundingRectangleImage(
squaredBiggestContourBoundingRectangleImage);
                                                        using (var dataImage =
squaredBiggestContourBoundingRectangleImage
.Resize(Constants.OUTPUT_DATA_IMAGE_SIDE_SIZE,
Constants.OUTPUT_DATA_IMAGE_SIDE_SIZE, Inter.Linear))
                                                        {
                                                            if
(_saveTrainingImageData)
                                                            {
SaveTrainingImageData(dataImage);
                                                            }

DrawDataImage(dataImage);
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        };
                }
                private void DrawVideoCaptureImage(Mat videoCaptureImageMatrix)
                {
                    CvInvoke.Rectangle(videoCaptureImageMatrix, _handGestureArea,
new MCvScalar(0, 0, 255));
                    videoCapturePictureBox.Image =
videoCaptureImageMatrix.ToBitmap();
                }

                private void ThresholdImage(Image<Gray, byte> grayImage, Image<Gray,
byte> binaryImage)
                {
                    CvInvoke.Threshold(grayImage,
                        binaryImage,
                        binaryImageThresholdTrackBar.Value,
                        Constants.MAX_COLOR_VALUE,
                        ThresholdType.Binary);
                }
```

```
            private void DrawBinaryImage(Image<Gray, byte> binaryImage)
            {
                binaryImagePictureBox.Image = binaryImage.ToBitmap();
            }

            private VectorOfVectorOfPoint GetImageContours(Image<Gray, byte>
binaryImage)
            {
                var contours = new VectorOfVectorOfPoint();
                CvInvoke.FindContours(binaryImage,
                    contours,
                    null,
                    areContoursExternalCheckBox.Checked ? RetrType.External :
RetrType.Ccomp,
                    ChainApproxMethod.ChainApproxSimple);
                return contours;
            }

            private void DrawContoursImage(Image<Gray, byte> contoursImage,
VectorOfVectorOfPoint contours)
            {
                CvInvoke.DrawContours(contoursImage, contours, -1, new
MCvScalar(255, 255, 255), thickness: 2);
                contoursPictureBox.Image = contoursImage.ToBitmap();
            }

            private void
DrawContourBoundingRectanglesImage(VectorOfVectorOfPoint contours, Image<Gray,
byte> boundingRectanglesImage)
            {
                for (int i = 0; i < contours.Size; i++)
                {
                    using (VectorOfPoint contour = contours[i])
                    {

boundingRectanglesImage.Draw(CvInvoke.BoundingRectangle(contour), new Gray(255));
                    }
                }

                contourBoundingRectanglesPictureBox.Image =
boundingRectanglesImage.ToBitmap();
            }

            private Rectangle
GetBiggestContourBoundingRectangle(VectorOfVectorOfPoint contours)
            {
                var biggestContourBoundingRectangle =
CvInvoke.BoundingRectangle(contours[0]);

                for (int i = 0; i < contours.Size; i++)
                {
```

```
                    using (VectorOfPoint contour = contours[i])
                    {
                        var contourBoundingRectangle =
CvInvoke.BoundingRectangle(contour);

                        if (biggestContourBoundingRectangle.Width *
biggestContourBoundingRectangle.Height < contourBoundingRectangle.Width *
contourBoundingRectangle.Height &&
                            contourBoundingRectangle.Width !=
Constants.HAND_GESTURE_AREA_WIDTH
                            && contourBoundingRectangle.Height !=
Constants.HAND_GESTURE_AREA_HEIGHT)
                        {
                            biggestContourBoundingRectangle =
contourBoundingRectangle;
                        }
                    }
                }
                return biggestContourBoundingRectangle;
            }
            private void DrawBiggestContourBoundingRectangleImage(Image<Gray,
byte> biggestBoundingRectangleImage, Rectangle biggestBoundingRectangle)
            {
                biggestBoundingRectangleImage.Draw(biggestBoundingRectangle, new
Gray(255));
                biggestContourBoundingRectanglePictureBox.Image =
biggestBoundingRectangleImage.ToBitmap();
            }
            private static void
SquareBiggestContourBoundingRectangleImage(Rectangle boundingRectangle,
Image<Gray, byte> copyMakeBorderInputImage, Mat boundingRectangleMat)
            {
                copyMakeBorderInputImage.ROI = boundingRectangle;

                CvInvoke.CopyMakeBorder(copyMakeBorderInputImage,
                    boundingRectangleMat,
                    boundingRectangle.Width > boundingRectangle.Height ?
(boundingRectangle.Width - boundingRectangle.Height) / 2 : 0,
                    boundingRectangle.Width > boundingRectangle.Height ?
(boundingRectangle.Width - boundingRectangle.Height) / 2 : 0,
                    boundingRectangle.Width < boundingRectangle.Height ?
(boundingRectangle.Height - boundingRectangle.Width) / 2 : 0,
                    boundingRectangle.Width < boundingRectangle.Height ?
(boundingRectangle.Height - boundingRectangle.Width) / 2 : 0,
                    BorderType.Constant);
            }

        private void DrawSquaredBiggestContourBoundingRectangleImage(Image<Gray,
byte> squaredBoundingRectangleImage)
            {
                squaredBiggestContourBoundingRectanglePictureBox.Image =
squaredBoundingRectangleImage
```

```
                    .Resize(Constants.HAND_GESTURE_AREA_WIDTH,
Constants.HAND_GESTURE_AREA_HEIGHT, Inter.Linear)
                    .ToBitmap();
            }
            private void DrawDataImage(Image<Gray, byte>
squaredBoundingRectangleImage)
            {
                dataImagePictureBox.Image = squaredBoundingRectangleImage
                    .Resize(dataImagePictureBox.Width,
dataImagePictureBox.Height, Inter.Linear)
                    .ToBitmap();
            }
            private void videoCapturePictureBox_Click(object sender,
System.EventArgs e)
            {
                var mouseEventArgs = e as MouseEventArgs;
                _handGestureArea.X = (mouseEventArgs.X < _handGestureArea.Width
/ 2) ? 0 :
                    (videoCapturePictureBox.Width - mouseEventArgs.X <
_handGestureArea.Width / 2) ?
                        videoCapturePictureBox.Width - _handGestureArea.Width -
1 :
                        mouseEventArgs.X - _handGestureArea.Width / 2 - 1;

                _handGestureArea.Y = (mouseEventArgs.Y < _handGestureArea.Height
/ 2) ? 0 :
                    (videoCapturePictureBox.Height - mouseEventArgs.Y <
_handGestureArea.Height / 2) ?
                        videoCapturePictureBox.Height - _handGestureArea.Height
- 1 :
                        mouseEventArgs.Y - _handGestureArea.Height / 2 - 1;
            }
            private void captureImageToRecognizeButton_Click(object sender,
System.EventArgs e)
            {
                imageToRecognizePictureBox.Image = dataImagePictureBox.Image;
            }
            private void stopTrainingButton_Click(object sender,
System.EventArgs e)
            {
                _saveTrainingImageData = false;
                SetTrainingButtonsIsEnabled();
            }
            private void startTrainingButton_Click(object sender,
System.EventArgs e)
            {
                _saveTrainingImageData = true;
                SetTrainingButtonsIsEnabled();
                _neuralNetwork.Train(_testDataCsvManager.ReadAllLines());
            }
            private void recognizeImageButton_Click(object sender,
System.EventArgs e)
```

```
            {
                if (imageToRecognizePictureBox.Image != null)
                {
                    using (var imageToRecognize = new Image<Gray,
byte>(imageToRecognizePictureBox.Image.ToString())
                                    .Resize(Constants.OUTPUT_DATA_IMAGE_SIDE_SIZE,
Constants.OUTPUT_DATA_IMAGE_SIDE_SIZE, Inter.Linear))
                    {
                        var queryResult =
_neuralNetwork.Query(GetImageByteArray(imageToRecognize));
                    }
                }
            }
            private void SaveTrainingImageData(Image<Gray, byte> dataImage)
            {
                _trainingImageDataManager.SaveImageToFile(dataImage,
gestureNumberComboBox.SelectedItem.ToString(), "jpg");

                try
                {
                    _trainDataCsvManager.AppendLineToFile(string.Format("{0},
{1}", gestureNumberComboBox.SelectedItem.ToString(),
                        string.Join(", ", GetImageByteArray(dataImage).Select(b
=> b.ToString())));
                    infoTextBox.Text = string.Empty;
                }
                catch (IOException)
                {
                    infoTextBox.Text = "Bad";
                }
            }
            private List<byte> GetImageByteArray(Image<Gray, byte> imageOpenCv)
            {
                var arrayByte = new List<byte>();

                for (int v = 0; v < imageOpenCv.Height; v++)
                {
                    for (int u = 0; u < imageOpenCv.Width; u++)
                    {
                        arrayByte.Add(imageOpenCv.Data[v, u, 0]);
                    }
                }
                return arrayByte;
            }
        }
    }
```