Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Програмний комплекс для інтерактивного навчання та тестування згорткової нейронної мережі»
за освітньою програмою **12 Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи ПЗ1912: _____ (підпис) /Дмитро ЧЕРКАС/

Керівник: _____ (підпис) /Валентин РАЗНОСІЛІН/

Нормоконтролер: _____ (підпис) /Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент: _____ (підпис)

Дніпро – 2023 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis
bachelor

on the topic: «A software package for interactive training and testing of convolutional neural network»

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ1912: ___ _____ /Dmytro CHERKAS/
(посада) (підпис)

Scientific Supervisor: ___ _____ /Valentin RAZNOSILIN/
(посада) (підпис)

Normative controller: ___ _____ / Svitlana VOLKOVA/
(посада) (підпис)

Supervisors:
Economic part ___ _____ / Mykola GNENNIYN/
(посада) (підпис)

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ____КІТ
_____ Вадим ГОРЯЧКІН
_____ 202_ р.

ЗАВДАННЯ

На кваліфікаційну роботу ____Бакалавр_____
студенту Черкасу Дмитру Анатолійовичу_____

1. Тема дипломної роботи: Програмний комплекс для інтерактивного навчання та тестування згорткової нейронної мережі
   Керівник роботи: Разносілін Валентин В'ячеславович
   затверджені наказом 1196 ст від 07.12.2022 року

2. Строк подання студентом роботи 22.06. 2023 року

3. Вихідні дані до дипломної роботи: _____.

4. Зміст пояснювальної записки (перелік питань до розробки): реферат, вступ, аналіз сучасного стану предметної області, проектування, тестування та відлагодження, опис програмного продукту, висновки, бібліографічний список

5. Перелік демонстраційного матеріалу:
   5.1. доповідь;
   5.2. презентація;
   5.3. демонстраційне відео.

6. Консультанти:

Розділ	Консультант	Завдання видав	Завдання прийняв
Техніко-економічні розрахунки	Микола ГНЕНИЙ	Разносілін Валентин В'ячеславович	Черкас Дмитро Анатолійович

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Вступ	12.09.22 – 26.10.22	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	27.10.22 – 04.03.23	
3	аліз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	05.03.23 – 31.04.23	
4	Постановка задачі, технічне завдання	01.05.23 – 07.05.23	30%
5	Техніко-економічні показники	08.05.23 – 15.05.23	
6	Розробка інструментальних засобів дослідження	16.05.23 – 21.05.23	
7	Виконання досліджень	22.05.23 – 28.05.23	60%
8	Оформлення тез доповідей	29.05.23 – 01.06.23	
9	Оформлення статті у фаховий журнал	02.06.23 – 06.06.23	
10	Оформлення пояснювальної записки	07.06.23 – 11.06.23	
11	Розробка демонстраційних матеріалів	12.06.23 – 18.06.23	100%
12	Подання кваліфікаційної роботи до кафедри	22.06.23	
13	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	28.06.23	

Студент _____ /Дмитро ЧЕРКАС/

Керівник роботи _____ /Валентин РАЗНОСІЛІН/

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра 131 с., 40 рис., 14 табл., 4 додатки, 18 джерел.

Об’єктом дослідження є згорткові нейронні мережі. Предметом дослідження є прогнозування швидкості та ефективності навчання загорткових нейронних мереж на заздалегідь оброблених вхідних даних.

Метою дослідження в контексті даної роботи є виявлення найбільш ефективного алгоритму подання даних у згорткову нейронну мережу підходящої для аналізування набору зображень, та її реалізація за допомогою .Net фреймворка на мові програмування C# без участі бібліотек, які заточені під роботу із нейронними мережами.

Методи дослідження: навчання загорткової нейронної мережі за стохастичним методом, та алгоритмом зворотного поширення помилки, які як тільки знаходять помилку – одразу оновлюють ваги мережі. При розробці програмної реалізації була застосована методологія об’єктно-орієнтованого проектування, та моделювання на основі UML.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, бібліографічного списку та чотирьох додатків.

Вступ описує суть, мету та актуальність роботи (2 сторінки).

Перший розділ: аналіз сучасного стану предметної області (5 сторінок).

Другий розділ: проектування (29 сторінок).

Третій розділ: тестування та відлагодження (14 сторінок).

Четвертий розділ: опис програмного продукту (7 сторінок).

Додатки: технічне завдання, специфікація, листи затвердження, текст програми.

Ключові слова: згорткова нейронна мережа; нейрон; ваги нейронної мережі; шар нейронної мережі.

ВСТУП

Актуальність роботи. Нейронні мережі на сьогоднішній день є областю, що дуже швидко розвивається. Зумовлюється це тим, що вони здатні навчитися робити будь-що, у будь-яких умовах. Згорткові нейронні мережі (Convolutional Neural Networks – CNNs) [1], є потужним інструментом для обробки зображень облич, оскільки вони можуть автоматично виявляти та відрізняти різноманітні риси, такі як: форму обличчя, контури очей, контури рота, тощо. Використання загорткових нейронних мереж у задачах аналізування зображення, на сьогодні є найефективнішим методом. Однак, ефективність та швидкість навчання згорткових нейронних мереж значно залежить від вхідних даних, а також їх порядку подання. Використання великого обсягу різноманітних зображень облич допомагає нейронним мережам краще навчитися, однак тоді процес навчання буде довшим. Подання даних у правильному порядку, наприклад, послідовність облич, в залежності від їх місця розташування, може покращити, або погіршити швидкість навчання та здатність мережі розпізнавати обличчя з високою точністю. Вивчення впливу різних вхідних даних та порядку їх подання на ефективність та швидкість навчання згорткових нейронних мереж є необхідним кроком у вдосконаленні цих систем.

Мета роботи. Створення ПЗ для знаходження оптимальної кількості зображень для навчання згорткової нейронної мережі, розкиду її атрибутів та порядку подання вхідних даних.

Експлуатаційне призначення. Дослідження допоможе розробникам проаналізувати їх набори вхідних даних, та створити вибірку, яка буде навчатися найефективнішим методом. Хоч, попередньо, користувачем досліджень будуть розробники ПЗ, однак вже для їх продуктів кінцевими користувачами будуть інші люди із багатого списку сфер, наприклад: безпека, медицина, реклама, розваги тощо.

Робота відповідає структурі:

Таблиця 3.1 – Структура пояснювальної записки

№ з/п	Елемент (частина) пояснювальної записки	Орієнтовний обсяг, стор.
1.	Титульний аркуш	1
2.	Завдання на кваліфікаційну роботу	2
3.	Відомість кваліфікаційної роботи	1-2
4.	Реферат	1
5.	Зміст	1-2
6.	Перелік умовних позначень, символів, одиниць, скорочень і термінів (за потреби)	1
7.	Вступ	1-2
8.	Розділи роботи, що розкривають її зміст	20-35* 50-70**
9.	Висновки та рекомендації	1-2
10.	Перелік посилань	1-3
11.	Додатки	

* - для бакалаврської роботи, ** - для магістерської роботи

1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ

Нейронні мережі – область яка розвивається дуже стрімко, і за останні декілька років були сильно популяризовані. DLSS [2], Midjourney [3], DeepFake [4], StyleGAN [https://en.wikipedia.org/wiki/Deepfake

5], хоч всі ці технології відносно нові, а про них чули вже майже всі. Всіх їх об’єднує те, що для навчання і роботи вони використовують згорткові нейронні мережі. Навіть представити майбутнє життя без цих, і подібних їм, технологій дуже важко, тому розвиток цієї сфери є необхідним кроком задля поліпшення всіх сфер життєдіяльності де використовуються ці нейронні мережі.

1.1. Огляд сучасних методів та моделей згорткових нейронних мереж.

Сучасні методи та моделі згорткових нейронних мереж займають важливе місце в галузі машинного навчання та комп'ютерного зору. Вони здатні ефективно аналізувати зображення, виявляючи патерни зображень та виконувати завдання класифікації, виявлення об'єктів, сегментації, тощо.

Першими і найвпливовішими моделями можна виділити французьку LeNet-5 [6] та українську AlexNet [7]. LeNet-5 була розроблена для розпізнавання написаних чисел у поштових індексах і автоматизації роботи поштових відділень, а AlexNet першою показала потужність глибоких згорткових нейронних мереж для класифікації зображень на великому наборі даних ImageNet [8]. Ці дві моделі дали поштовх для розвитку цієї галузі, після чого було розроблено багато інших моделей, що покращили точність та ефективність згорткових нейронних мереж, таких як: VGG [9], що є покращеною версією AlexNet, в якій замінено великі фільтри згортки на кілька послідовних фільтрів меншого розміру; GoogLeNet [10], що ще більше модифікував архітектуру в AlexNet, додавши також фільтри згортки 1x1 щоб взяти поточну кількість вимірів, і лінійно їх змішати в меншу. Завдяки цьому

ефективність навчання сильно росте; ResNet [11], яка почала застосовувати нелінійну передачу зображення та підсумування оброблюваних зображень; DenseNet [12], яка при передачі оброблених зображень у наступний шар не підсумовує їх, а конкатенує у єдиний тензор, зменшуючи кількість операцій майже втричі; EfficientNet [13], яка пропонує новий метод масштабування, який рівномірно масштабує роздільну здатність зображень, глибину та ширину із фіксованими пропорціями між ними.

Для навчання та оптимізації згорткових нейронних мереж часто використовуються функції втрат, такі як Cross-Entropy [14] або Back Propagation [15], а також методи оптимізації, такі як стохастичний градієнтний спуск з моментом [16], Adam [17] та RMSprop [18].

Cross-Entropy бере за помилку не тільки помилковий вихід нейронної мережі, а і вихід який частино вірний, але мережа у ньому не впевнена. Back Propagation бере кожну помилку, або групу помилок, і змінює ваги нейронної мережі відносно цієї помилки.

Стохастичний градієнтний спуск з моментом який використовується для пришвидшення пошуку необхідних ваг шляхом використання тренувального набору даних, який вибирається випадковим чином. Adam модифікує метод стохастичного градієнтного спуску, додаючи динамічну зміну швидкості навчання, залежно від розвитку нейронної мережі. У RMSprop замість повної суми оновлень ваг буде використовуватися середнє значення декількох ітерацій.

Отже, згорткові нейромережі розвиваються дуже стрімко, і не збираються зупинятися. Для кожної нової моделі чи метода через певний час знайдеться більш досконала альтернатива.

1.2. Аналіз ефективності згорткових нейронних мереж у розпізнаванні осіб та визначенні їх характеристик.

Кожний тип нейронної мережі має свої сильні та слабкі сторони. Безперечно, що найсильнішою стороною згорткових нейромереж є робота із зображенням та виділенні основних характеристик об’єктів у ньому. Тому вони виявляються особливо ефективними у завданнях комп'ютерного зору, розпізнавання об'єктів, розпізнавання облич та їх ознак, тощо.

Що саме робить згорткові нейронні мережі настільки ефективними у розпізнаванні осіб та визначенні їх характеристик?

По-перше, здатність до локального сприйняття. Згорткові шари в мережі дозволяють моделі фокусуватися на локальних деталях зображень, використовуючи для цього фільтри згортки. Це дозволяє моделі виявляти локальні особливості, такі як контури облич, форми та текстури.

По-друге, вирішення розмірності. Завдяки використанню пулінгу у певних шарах мережі, вони здатні зменшити фактичний розмір вхідного зображення, а отже і кількість параметрів, підвищуючи обчислювальну ефективність нейромережі.

По-третє, загальність розпізнавання. Виявлення ключових ознак зображення дозволяє нейромережі розпізнавати і ідентифікувати обличчя незалежно від певних факторів, таких як розміщення, розмір, нахил або освітлення.

По-четверте, навчання на великому обсязі даних. Для успішного застосування згорткових нейронних мереж у розпізнаванні осіб необхідні великі набори даних. Більшість сучасних популярних мереж використовуються після попереднього навчання на великій кількості зображень, що дозволяє моделям запам’ятати всі можливі ознаки облич, що майже унеможливлює негативний результат роботи нейромережі.

По-п’яте, використання рекурентних мереж. У деяких випадках рекурентні нейронні мережі можуть бути використанні у парі із згортковими шарами для аналізу послідовностей облич, або визначення емоцій на основі зміни в обличчі з часом.

По-шосте, визначення характеристик. Згорткові нейронні мережі можуть бути використані для визначення різних характеристик осіб, таких як вік, стать, етнічність, тощо. Це досягається за допомогою додаткових шарів після основних згорткових шарів, та нейронів, відповідаючих за вихідні данні.

Отже, згорткові нейронні мережі виявляються дуже ефективними для розпізнавання осіб та визначення їх характеристик. Однак, успішність моделі значною мірою залежить від якості та розміру навчального набору даних, який використовується для навчання мережі та архітектури збудованої нейромережі.

1.3. Обмеження та перешкоди, з якими стикаються при використанні згорткових нейронних мереж.

При використанні згорткових нейронних мереж у розпізнаванні осіб та визначенні їх характеристик можуть виникати певні обмеження та перешкоди:

По-перше, необхідність великих наборів даних. Згорткові нейронні мережі потребують великого обсягу навчальних даних для ефективного навчання. Однак, отримання великого та всеосяжного набору даних облич може бути викликом, особливо для спеціалізованих завдань, де потрібно враховувати спецефічні фактори.

По-друге, розмір обчислювального ресурсу. Згорткові нейронні мережі можуть бути занадто вимогливими щодо обчислювального ресурсу, особливо якщо маємо справу з великими та глибокими мережами та великими за обсягами зображеннями. Для ефективного навчання та використання таких моделей треба мати доступ до потужних обчислювальних ресурсів, таких як відеокарти або серверні процесори.

По-третє, недостатність даних певних категорій. У деяких випадках можуть бути обмеження зі збору даних для певних категорій. Це може призвести до нерівномірності та недостатньої представленості цих категорій у навчальному наборі даних, що може негативно вплинути на ефективність мережі.

По-четверте, небажані зміни в обличчях. Згорткові нейронні мережі які навчалися на обмеженій вибірці, або мають недостатньо продуману архітектуру можуть бути чутливі до незначних змін у позиції обличчя, освітлення, тощо.

Висновки до пункту 1

Для викреслення всіх вище описаних обмежень, потрібно щоб були три наступні компоненти: час, великий розмір обчислювального ресурсу, та великий набір даних. При нестачі хоч одного із цих компонентів, навантаження перейде у інші, що вимусить зробити нейромережу із певними обмеженнями.

2. ПРОЕКТУВАННЯ

2.1. Зовнішнє проектування

2.1.1. Функціональне призначення

Функціональним призначенням програмного комплексу є полегшення дослідницької роботи у сфері навчання згорткових нейронних мереж шляхом генерації і використання певної вибірки зображень із заданими параметрами.

2.1.2. Експлуатаційне призначення

Експлуатаційним призначенням програмного комплексу є зменшення часових витрат на навчання згорткової нейронної мережі, шляхом дослідження впливу виборок із обмеженими параметрами без втрачання точності її роботи.

2.1.3. Функціональні вимоги

Програмний комплекс повинен надавати можливість:

- Генерувати зображення “обличчя” розміром 100x100 пікселів за заданими параметрами;
- Обирати граничні параметри облич при їх генерації;
- Повертати параметри облич до базових;
- Зберігати генеровані обличчя на жорсткий диск;
- Зберігати файл конфігурацій параметрів облич на жорсткий диск;
- Сортувати обличчя за заданими параметрами;
- Окремо обирати вибірки для навчання і для тестування згорткової нейронної мережі;
- Налаштовувати модифікатор навчання нейромережі;
- Конфігурувати ядра згортки;
- Зберігати конфігурацію ваг нейронної мережі до файлу;
- Зчитувати конфігурацію ваг нейронної мережі із файлу;

- Отримувати візуалізацію процесу навчання згорткової нейронної мережі;
- Зберігати результати навчання мережі у .csv файл;
- Моделювати навчання згорткової мережі;
- Тестувати обране зображення згортковою мережею.

2.1.4. Вхідні та вихідні дані

Вхідними даними є:

- Кількість зображень для генерування;
- Параметр що регулює створення обличчя, або множини випадково розташованих геометричних фігур (не обличчя);
- Граничні значення:
    1. Куту нахилу облич (°);
    2. Позиції обличчя по горизонталі (у пікселях);
    3. Позиції обличчя по вертикалі (у пікселях);
    4. Розміру обличчя по горизонталі (у процентному відношенні);
    5. Розміру обличчя по вертикалі (у процентному відношенні);
    6. Змінної, що регулює верхню частину форми обличчя від ромбовидної до овальної (у процентному відношенні);
    7. Змінної, що регулює нижню частину форми обличчя від ромбовидної до овальної (у процентному відношенні);
    8. Розміру вух по горизонталі (у процентному відношенні);
    9. Розміру вух по вертикалі (у процентному відношенні);
    10. Зміщення вух по контуру обличчя (у процентному відношенні);
    11. Розміру очей по горизонталі (у пікселях);
    12. Розміру очей по вертикалі (у пікселях);
    13. Зміщення очей від центру обличчя по горизонталі (у пікселях);

14. Зміщення очей від центру обличчя по вертикалі (у пікселях);
15. Розміру носа по горизонталі (у пікселях);
16. Розміру носа по вертикалі (у пікселях);
17. Зміщення носа від центру обличчя по вертикалі (у пікселях);
18. Розміру рота по горизонталі (у пікселях);
19. Розміру рота по вертикалі (у пікселях);
20. Зміщення рота від центру обличчя по вертикалі (у пікселях).

- Кількість зображень для сортування їх за параметрами;
- Критерії сортування (3 параметри);
- Файл конфігурацій облич, який буде відсортований;
- Файл конфігурацій облич, що буде створений для збереження результатів сортування;
- Кількість облич для навчання нейронної мережі;
- Шлях до папки із обличчями для навчання нейронної мережі;
- Файл конфігурацій облич для навчання;
- Файл ваг нейронної мережі;
- Кількість облич для тестування нейронної мережі;
- Шлях до папки із обличчями для тестування нейронної мережі;
- Файл конфігурацій облич для тестування;
- Модифікатор навчання згорткової нейронної мережі;
- Масив ядер згортки (матриці 3х3 елементи).

Вихідними даними є:

- Масив зображень облич розміром 100х100 пікселів;
- Файл конфігурації генерованих облич;
- Відсортований файл конфігурацій облич;
- Текстове повідомлення про завершення сортування облич;
- Графіки ефективності роботи нейронної мережі у вигляді таблиць та

файлів;

- Поточна кількість оброблених нейронною мережею облич;
- Загальна кількість облич, що будуть оброблятися;
- Візуальне зображення поточного стану кількості оброблених облич, щодо їх максимальної кількості;
- Файл ваг нейронної мережі;
- Файл типу .csv із збереженням історії навчання нейронної мережі.

2.1.5. Опис зовнішнього інформаційного середовища

Для функціонування системи потрібна операційна система Windows 10 та новіше, наявність стандартних бібліотек та бібліотеки .Net 4.5 та новіше.

Кількість зображень для генерування, кількість зображень для сортування їх за параметрами, кількість облич для навчання нейронної мережі, кількість облич для тестування нейронної мережі, модифікатор навчання згорткової нейронної мережі та масив ядер згортки вводяться з клавіатури в числове поле з контролем введення.

Параметр що регулює створення обличчя, або множини випадково розташованих геометричних фігур та усі граничні значення регулюються за допомогою повзунків.

Критерії сортування обираються за допомогою натискання лівої кнопки миші на відповідному пункті списку, представленому на формі.

Файли конфігурацій, файли ваг нейронної мережі та шляхи до папок із обличчями вводяться з клавіатури в текстове поле з контролем введення або за допомогою інтерактивного відкриття файлів.

Специфікація функціональних вимог виконана у вигляді діаграм прецедентів (рис. 2.1.1 – рис. 2.1.3).

Рисунок 2.1.1 – Діаграма прецендентів программи створення облич

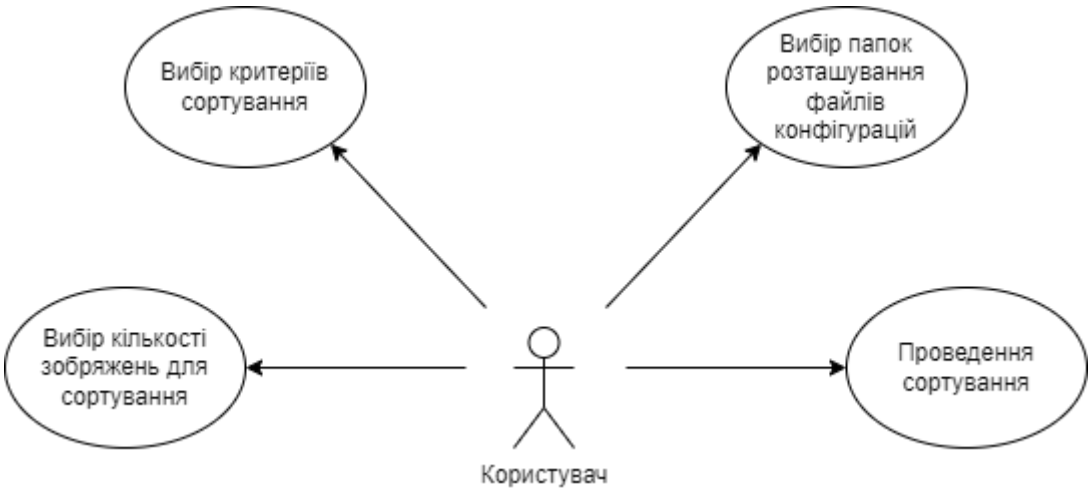

Рисунок 2.1.2 – Діаграма прецендентів программи сортування облич

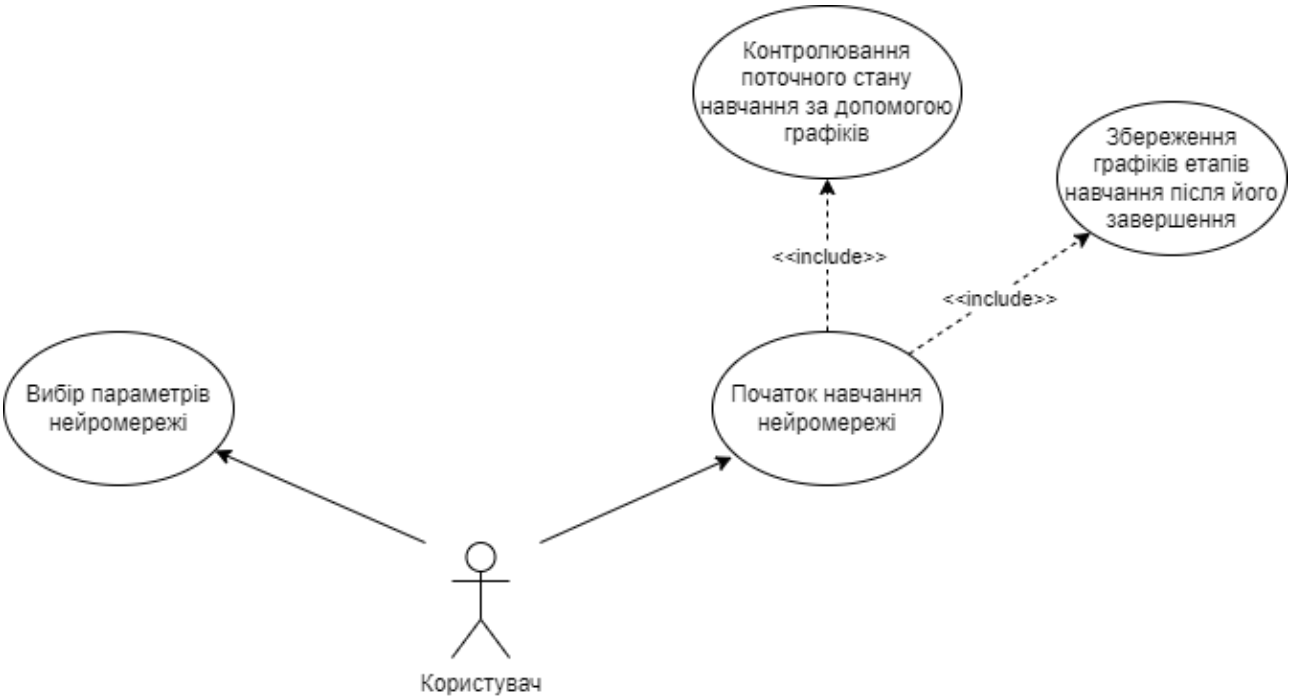

Рисунок 2.1.3 – Діаграма прецендентів программи навчання нейронної мережі

2.2. Внутрішнє проектування

2.2.1. Аналіз зовнішніх специфікацій систем

2.2.1.1. Моделювання словника системи

Розглянувши математичні моделі та сценарії взаємодії користувача із системою, можна виділити базові сутності.

Ідентифіковані сутності: Генератор облич, Конфігуратор облич, Сортувальник облич, Нейронна мережа, Конфігуратор нейронної мережі, Конфігуратор ядер згортки.

Ідентифіковані обов'язки:

Генератор облич – основний клас для генерації облич розміру 100x100 пікселів і їх файлу конфігурацій;

Конфігуратор облич – клас-помічник до класу «Генератор облич» для завдання користувачем основних параметрів генеруємих облич;

Сортувальник облич – основний клас для сортування файлів конфігурацій облич;

Нейронна мережа – основний клас що виконує навчання нейронної мережі і зберігає його результати;

Конфігуратор нейронної мережі – клас-помічник до класу «Нейронна мережа» що дозволяє користувачу вибирати параметри нейронної мережі;

Конфігуратор ядер згортки – клас-помічник до класу «Конфігуратор нейронної мережі» що дозволяє користувачу редагувати ядра згортки;

Атрибути та операції, необхідні для виконання обов'язків кожної сутності-класу наведемо у табл. 2.2.1.

Таблиця 2.2.1 – Сутності, атрибути та методи

Сутність	Атрибути	Методи
Генератор облич	rand – змінна для роботи генератора випадкових чисел settings – вказівник на клас «Конфігуратор облич»; g – змінна для графічного моделювання обличчя; image – змінна для зберігання графічної моделі обличчя; path – змінна для шляху збереження облич на комп’ютері; currentFace – змінна що зберігає структуру параметрів поточно-сгенерованого обличчя; hindrance – Параметр що регулює створення обличчя, або множини випадково розташованих геометричних фігур (не	drawEyes – малює очі; drawNose – малює носа; drawMounth – малює рота; drawCounterFaceAndEars – малює контури обличчя та вуха; button1_Click – при натисканні кнопки запускає процес малювання обличчя; pictureBox1_Paint – викликається при запиті на оновлення обличчя, створює фіксовані параметри поточного обличчя і передає їх далі до малювання частин

Продовження таблиці 2.2.1

	обличчя); positionMinX – нижня гранична точка генерації горизонтального положення центра обличчя; positionMaxX – верхня гранична точка генерації горизонтального положення центра обличчя; positionMinY – нижня гранична точка генерації вертикального положення центра обличчя; positionMaxY – верхня гранична точка генерації вертикального положення центра обличчя; angleMin – нижня гранична точка куту нахилу обличчя; angleMax – верхня гранична точка куту нахилу обличчя; sizeMinX – нижня гранична точка розміру обличчя по горизонталі; sizeMaxX – верхня гранична точка розміру обличчя по горизонталі; sizeMinY – нижня гранична точка розміру обличчя по вертикалі;	обличчя; button3_Click – при натисканні кнопки запускає процес малювання облич і зберігання їх на комп’ютері разом із їх файлом конфігурацій; SettingsButton_Click – при натисканні кнопки викликає форму іншого класу для редагування граничних значень параметрів обличчя; setSettingToZero – скидає всі граничні точки параметрів обличчя до базових.

Продовження таблиці 2.2.1

	sizeMaxY – верхня гранична точка розміру обличчя по вертикалі; topDistortionMin – нижня гранична точка змінної, що регулює верхню частину форми обличчя від ромбовидної до овальної; topDistortionMax – верхня гранична точка змінної, що регулює верхню частину форми обличчя від ромбовидної до овальної; bottomDistortionMin – нижня гранична точка змінної, що регулює нижню частину форми обличчя від ромбовидної до овальної; bottomDistortionMax – верхня гранична точка змінної, що регулює нижню частину форми обличчя від ромбовидної до овальної; earSizeMinY – нижня гранична точка розміру вух по вертикалі; earSizeMaxY – верхня гранична точка розміру вух по вертикалі;	

Продовження таблиці 2.2.1

	earSizeMinX – нижня гранична точка розміру вух по горизонталі; earSizeMaxX – верхня гранична точка розміру вух по горизонталі; earShiftMin – нижня гранична точка зміщення вух по контуру обличчя; earShiftMax – верхня гранична точка зміщення вух по контуру обличчя; eyeSizeMinX – нижня гранична точка розміру очей по горизонталі; eyeSizeMaxX – верхня гранична точка розміру очей по горизонталі; eyeSizeMinY – нижня гранична точка розміру очей по вертикалі; eyeSizeMaxY – верхня гранична точка розміру очей по вертикалі; eyeShiftMinX – нижня гранична точка зміщення очей від центру обличчя по горизонталі; eyeShiftMaxX – верхня гранична точка зміщення очей від центру	

Продовження таблиці 2.2.1

	обличчя по горизонталі; eyeShiftMinY – нижня гранична точка зміщення очей від центру обличчя по вертикалі; eyeShiftMaxY – верхня гранична точка зміщення очей від центру обличчя по вертикалі; noseSizeMinX – нижня гранична точка розміру носа по горизонталі; noseSizeMaxX – верхня гранична точка розміру носа по горизонталі; noseSizeMinY – нижня гранична точка розміру носа по вертикалі; noseSizeMaxY – верхня гранична точка розміру носа по вертикалі; noseShiftMin – нижня гранична точка зміщення носа від центру обличчя по вертикалі; noseShiftMax – верхня гранична точка зміщення носа від центру обличчя по вертикалі; mounthSizeMinX – нижня гранична точка розміру рота по горизонталі;	

Продовження таблиці 2.2.1

	mounthSizeMaxX – верхня гранична точка розміру рота по горизонталі; mounthSizeMinY – нижня гранична точка розміру рота по вертикалі; mounthSizeMaxY – верхня гранична точка розміру рота по вертикалі; mounthShiftMin – нижня гранична точка зміщення рота від центру обличчя по вертикалі; mounthShiftMax – верхня гранична точка зміщення рота від центру обличчя по вертикалі.	
Конфігуратор облич	mainForm – вказівник на клас «Генератор облич».	Settings – викликається при створенні класу, завантажує поточні параметри обличчя до себе; trackBar_Scroll – оновлює параметри обличчя.
Сортувальник облич	pathConfig – шлях до початкового файлу конфігурацій; pathOutput – шлях до вихідного файлу конфігурацій; openFileDialogConfig – змінна	button3_Click – при натисканні кнопки викликає діалог відкриття початкового файлу конфігурацій;

Продовження таблиці 2.2.1

	для виклику діалогу відкриття початкового файлу конфігурацій; openFileDialogOutput – змінна для виклику діалогу відкриття вихідного файлу конфігурацій.	button2_Click – при натисканні кнопки викликає діалог відкриття вихідного файлу конфігурацій; button1_Click – при натисканні кнопки починає сортування; decomposeElements – зберігає данні із файлу конфігурацій у вигляді структури; swapConfigElements – змінює місце розташування двох елементів структури; getValue – отримання конкретного параметра структури за індексом; randomDistribution – функція випадкового перемішування елементів структури; shellSort – сортування Шелла; getSmallerGroupForSorting – використовується при

Продовження таблиці 2.2.1

		сортуванні за декількома критеріями, виділяє підгрупи після сортування по кожній категорії та викликає сортування цих підгруп.
Нейронна мережа	lastNumber – порядковий номер останньої точки на графіку; lastX – похибка по X останньої точки на графіку; lastY – похибка по Y останньої точки на графіку; lastDeltaHindrance – похибка по точності вгадування обличчя останньої точки на графіку; lastHindranceAccuracy – вірогідність вгадування облич для останньої точки на графіку; testAfter – параметр визначає кількість елементів навчання, після котрих програма буде проганяти процедуру перевірки навченості нейронної мережі; convolutionMatrix – матриці сгортки; convolutionNumber – кількість	setConvolutionMatrix – встановлення значень ядер згортки у базові; decomposeElements – зберігає данні із файлу конфігурацій у вигляді структури; createMatrtixFromImage – розбиває зображення на матрицю такого-ж розміру; downsampling – функція що зменшує фактичний розмір зображення за певними алгоритмами; convolution – проходження матриці зображення через ядро згортки; convolutionsUnion – об’єднання результатів матриць після

Продовження таблиці 2.2.1

	матриць сгортки; pictureSize – розмір картинки downsamplingMultiplier – потужність даунсемплінгу зображення; firstNeuronLayerSize – розмір першого шару нейронної мережі; secondNeuronLayerSize – розмір другого шару нейронної мережі; dStep – множник швидкості навчання; n – кількість зображень для навчання; nTest – кількість зображень для тестування; folderBrowserDialog – змінна для виклику діалогу відкриття папки із зображеннями для навчання; folderBrowserDialogTest – змінна для виклику діалогу відкриття папки із зображеннями для тестування; openFileDialogConfig – змінна для виклику діалогу відкриття файлу конфігурацій для навчання;	проходження через ядра згортки; getNodesFromMatrix – розбиває двомірний масив у одномірний; activationFunc – функція активації; activationFuncDx – дискримінант функції активації; getNewNodes – отримання значень наступного шару нейронної мережі; generateRandomMatrix – створення матриць ваг нейронної мережі випадковим чином; readWeightsFromFile – зчитування ваг нейронної мережі із файлу; saveWeightsToFile – запис ваг нейронної мережі до файлу; neuralEducation – функція зворотного поширення помилки;

Продовження таблиці 2.2.1

	openFileDialogTest – змінна для виклику діалогу відкриття файлу конфігурацій для тестування; saveFileDialogSaveLoad – змінна для виклику діалогу відкриття або створення файлу ваг нейронної мережі; settingsSelected – змінна фляжок, що підтверджує успішне введення усіх параметрів нейронної мережі.	neuralNetworkWork – функція навчання нейронної мережі; neuralNetworkTest – функція тестування нейронної мережі; enterConfigs – зчитування файлів конфігурацій нейронної мережі; neuralStart – запуск роботи нейронної мережі; button1_Click – при натисканні кнопки перевіряє правильність вводу параметрів нейронної мережі та запускає її.
Конфігуратор нейронної мережі	mainForm – вказівник на клас «Нейронна мережа».	button1_Click – зберігає параметри на надсилає їх до класу «Нейронна мережа»; button2_Click – відкриває діалог вибору файлу конфігурацій зображень для навчання; button3_Click – відкриває

Продовження таблиці 2.2.1

		діалог вибору або створення файлу збереження дуг; button4_Click – відкриває діалог вибору папки збереження зображень для навчання; button5_Click – відкриває діалог вибору файлу конфігурацій зображень для тестування; button6_Click – відкриває форму редагування ядер згортки; button7_Click – відкриває діалог вибору папки збереження зображень для тестування; textBox1_TextChanged – перевіряє правильність вводу числового значення «dStep».
Конфігуратор ядер згортки	current – поточне ядро згортки; number – кількість ядер згортки.	dataGridView1_CellEndEdit – перевіряє правильність вводу числового значення елемента ядра згортки;

Продовження таблиці 2.2.1

		toolStripButtonRight_Click – переміщення індексу ядра згортки що редагується вправо; toolStripButtonLeft_Click – переміщення індексу ядра згортки що редагується вліво; toolStripButtonMinus_Click – видалення поточного ядра згортки; toolStripButtonPlus_Click – додавання ядра згортки; swapMatrices – перенесення матриці ядер згортки, що відбувається при видаленні елементу; dataGridRefresh – оновлення поточного ядра згортки.

2.2.1.2. Моделювання розподілу обов'язків у системі

Множини класів, які працюють спільно для досягнення деякої поведінки:

– Генератор облич, Конфігуратор облич – створення облич за певними критеріями;
– Нейронна мережа, Конфігуратор нейронної мережі – конфігурація та запуск нейронної мережі;

– Конфігуратор нейронної мережі, Конфігуратор ядер згортки – більш глибоке конфігурування нейронної мережі.

Визначення примітивних типів виконаємо на етапі побудови діаграми класів. Результат моделювання різних видів зв’язків подано у табл. 2.2.2.

Таблиця 2.2.2 – Моделювання залежностей

Клас, котрий зв’язується	Клас із котрим зв’язується	Тип зв’язку
Генератор облич	Конфігуратор облич	Асоціація
Нейронна мережа	Конфігуратор нейронної мережі	Агрегація
Конфігуратор нейронної мережі	Конфігуратор ядер Згортки	Агрегація

2.2.1.3. Визначення призначень об’єктів за допомогою CRC карток

Класи можуть бути специфіковані у вигляді CRC-карток. У таблицях 2.2.3 – 2.2.8 наведено CRC-картки на класи проекту.

Таблиця 2.2.3 – CRC-картка для класу «Генератор облич»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Генерує обличчя і зберігає їх у файл	Конфігуратор облич

Таблиця 2.2.4 – CRC-картка для класу «Конфігуратор облич»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Конфігурує параметри облич	Генератор облич

Таблиця 2.2.5 – CRC-картка для класу «Сортувальник облич»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Сортує файли конфігурацій	відсутні

Таблиця 2.2.6 – CRC-картка для класу «Нейронна мережа»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Запускає нейронну мережу і зберігає результати її роботи	Конфігуратор нейронної мережі

Таблиця 2.2.7 – CRC-картка для класу «Конфігуратор нейронної мережі»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Дозволяє користувачу конфігурувати нейронну мережу	Нейронна мережа, Конфігуратор ядер згортки

Таблиця 2.2.8 – CRC-картка для класу «Конфігуратор ядер згортки»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Дозволяє користувачу конфігурувати ядра згортки	Конфігуратор нейронної мережі

2.2.1.4. Побудова об’єктної моделі (діаграми класів)

Заключним результатом моделювання представимо у вигляді діаграми класів (рис. 2.2.1).

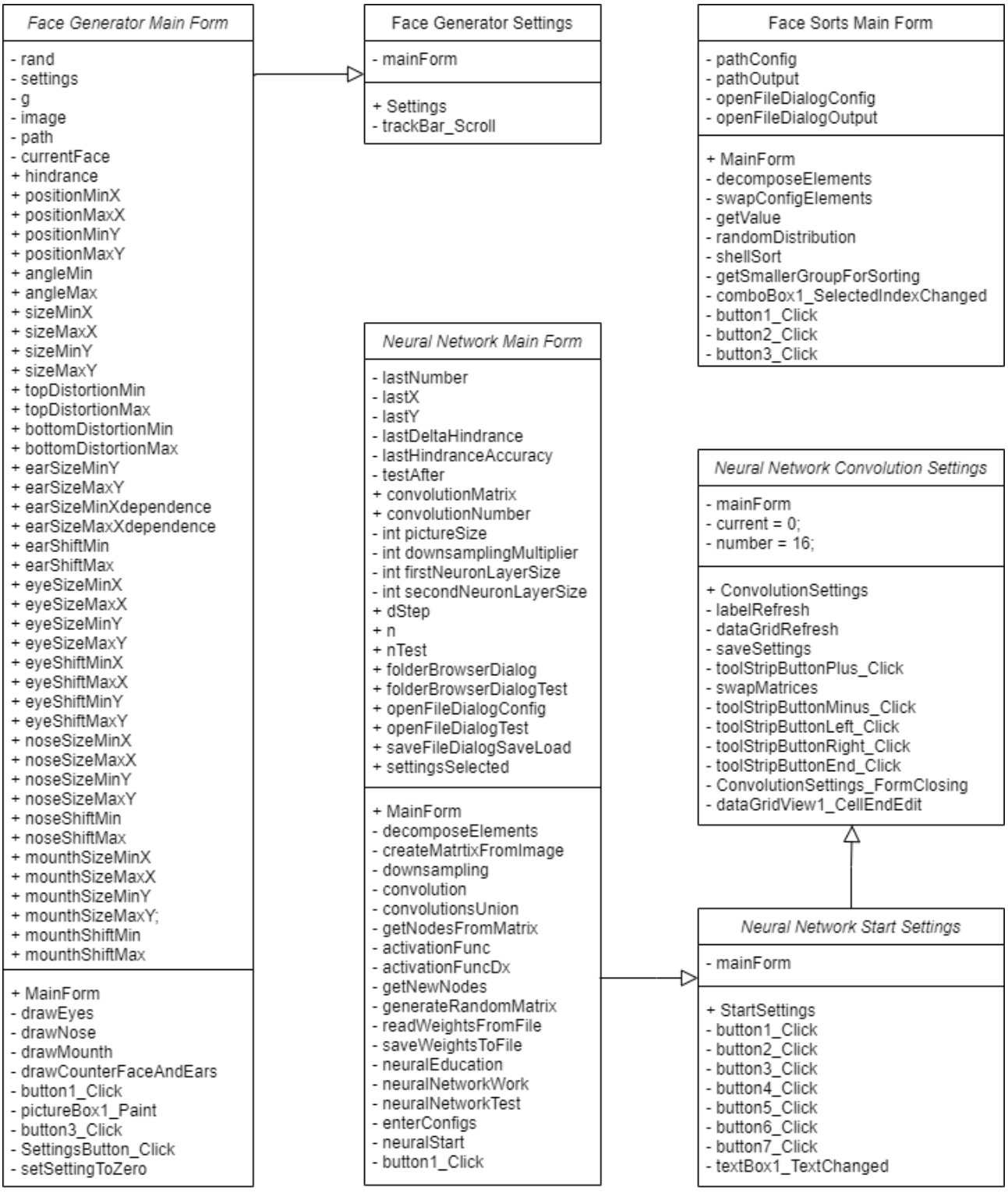

Рисунок 2.2.1 – Діаграма класів

Класи пов’язані агрегацією: Генератор облич та Конфігуратор облич, Нейронна мережа та Конфігуратор нейронної мережі, Конфігуратор нейронної

мережі та Конфігуратор ядер згортки, тому що другий клас є доповненням першого.

2.2.2. Проектування інтерфейсу користувача

2.2.2.1. Створення ескізів форм

Користувачу бажано мати доступ до швидкого виконання дій. Тому головні форми повинні бути розвантажені.

Програмний продукт складається з наступних вікон:

– головне вікно програми створення облич (рис. 2.2.2);
– вікно з налаштуванням облич (рис. 2.2.3);
– головне вікно програми сортувальника (рис. 2.2.4);
– головне вікно програми нейронної мережі (рис. 2.2.5);
– вікно налаштування нейронної мережі (рис. 2.2.6);
– вікно налаштування ядер згортки (рис. 2.2.7).

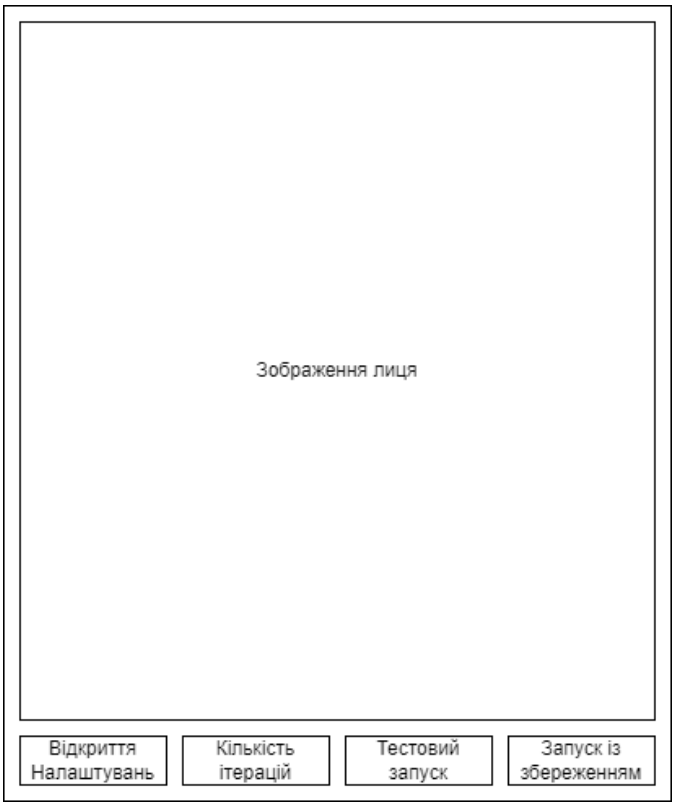

Рисунок 2.2.2 – Ескіз головного вікна програми створення облич

Головне вікно програми створення облич (рис. 2.2.2) має п’ять пунктів меню:

1. Ескіз лиця;
2. Кнопка виклику налаштувань;
3. Кількість зображень для створення;
4. Кнопка тестового запуску генерації зображень;
5. Кнопка запуску генерації зображень із їх збереженням на жорсткому диску.

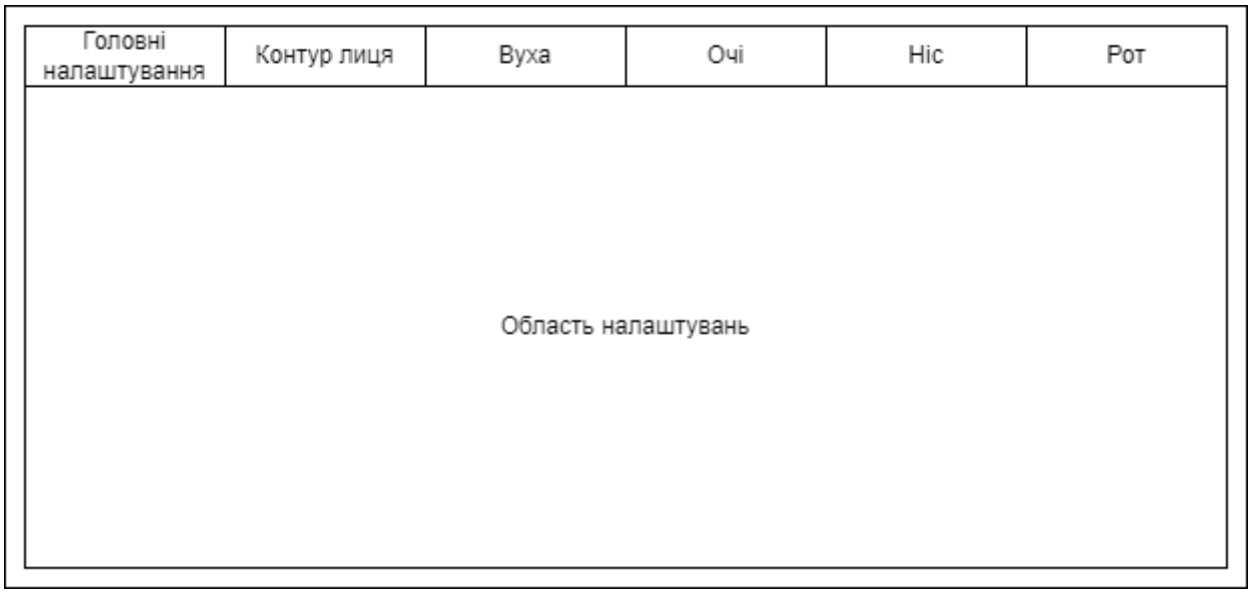

Рисунок 2.2.3 – Ескіз головного вікна з налаштуванням облич

Вікно з налаштуванням облич (рис. 2.2.3) має сім пунктів меню:

1. Область налаштувань вибраної вкладки;
2. Вкладка налаштувань – Головні налаштування;
3. Вкладка налаштувань – Контур лиця;
4. Вкладка налаштувань – Вуха;
5. Вкладка налаштувань – Очі;
6. Вкладка налаштувань – Ніс;
7. Вкладка налаштувань – Рот.

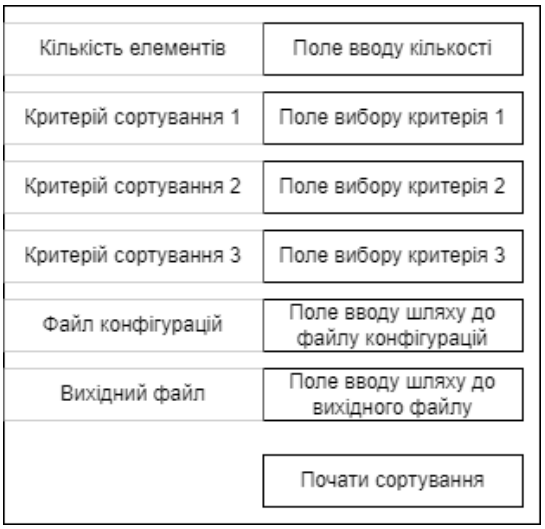

Рисунок 2.2.4 – Ескіз головного вікна програми сортувальника

Головне вікно програми сортувальника (рис. 2.2.4) має тринадцять пунктів меню:

1. Шість описів сусідніх полей;
2. Поле вводу кількості елементів для сортування;
3. Три поля вибору критеріїв сортування;
4. Поле вводу шляху до файлу конфігурацій;
5. Поле вводу шляху до вихідного файлу;
6. Кнопка початку сортування.

Рисунок 2.2.5 – Ескіз головного вікна програми нейронної мережі

Головне вікно програми нейронної мережі (рис. 2.2.5) має вісім пунктів меню:

1. Чотири графіки навчання нейронної мережі;
2. Повзунок прогресу навчання нейронної мережі;
3. Поточне зображення на обробці у нейронної мережі;
4. Загальна кількість зображень для обробки нейронною мережею;
5. Кнопка початку роботи нейронної мережі.

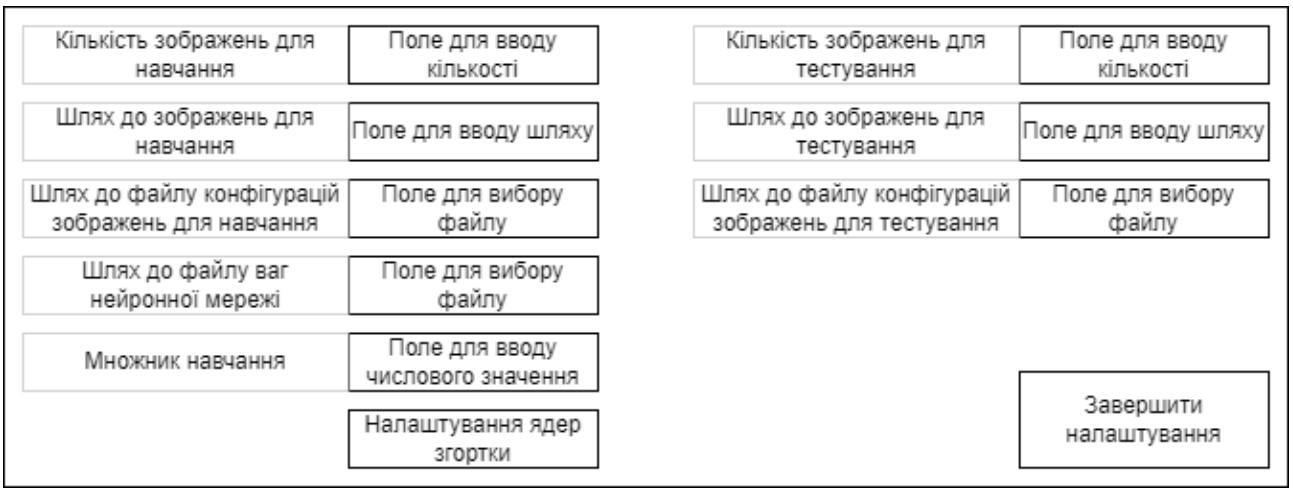

Рисунок 2.2.6 – Ескіз вікна налаштування нейронної мережі

Вікно програми налаштування нейронної мережі (рис. 2.2.6) має вісімнадцять пунктів меню:

1. Вісім описів сусідніх полей;
2. Поля для вводу кількості зображень для навчання і тестування;
3. Поля для вводу шляху до зображень для навчання і тестування;
4. Чотири поля для вибору файлів;
5. Поле для вводу множника навчання;
6. Кнопка для виклику форми налаштування ядер;
7. Кнопка завершення налаштування.

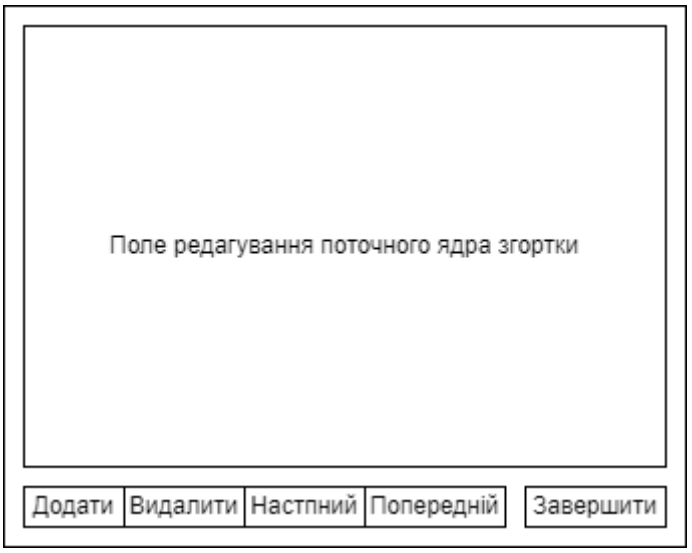

Рисунок 2.2.7 – Ескіз вікна налаштування ядер згортки

Вікно налаштування ядер згортки (рис. 2.2.7) має шість пунктів меню:

1. Поле редагування поточного ядра згортки;
2. Кнопка додавання ядра згортки;
3. Кнопка видалення ядра згортки;
4. Кнопка переходу поточного ядра згортки на наступне;
5. Кнопка переходу поточного ядра згортки на попереднє;
6. Кнопка завершення редагування ядер згортки.

2.2.3. Проектування динаміки системи

В якості об'єктів виступають користувач програмного комплексу та класи, що реалізують поведінку системи.

Для кожної програми у програмному комплексі можна виділити по одному варіанту використання, а саме:

- обирання параметрів лиця та створення зображень;
- обирання параметрів сортування та запуск сортування;
- обирання параметрів нейронної не межі та її запуск.

Сценарій «обирання параметрів лиця та створення зображень»: спочатку користувач за бажанням редагує параметри лиця, а потім запускає створення

зображень. Діаграма послідовності представлена на рис. 2.2.8.

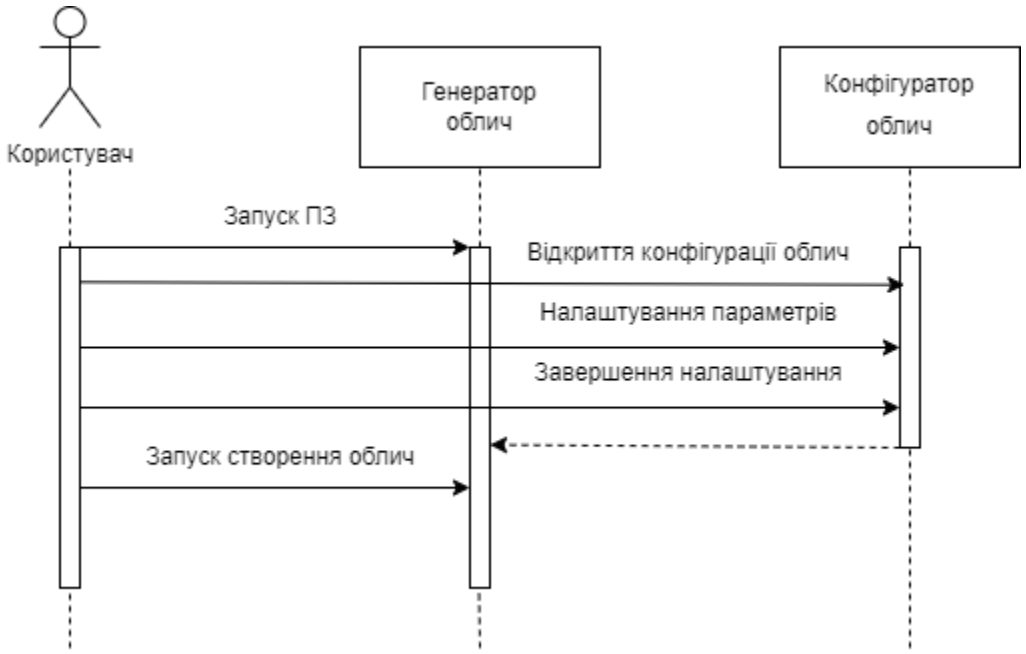

Рисунок 2.2.8 – Діаграма послідовності для варіанту використання «обирання параметрів лиця та створення зображень»

Сценарій «обирання параметрів сортування та запуск сортування»: спочатку користувач вказує параметри сортування, а потім запускає його. Діаграма послідовності представлена на рис. 2.2.9.

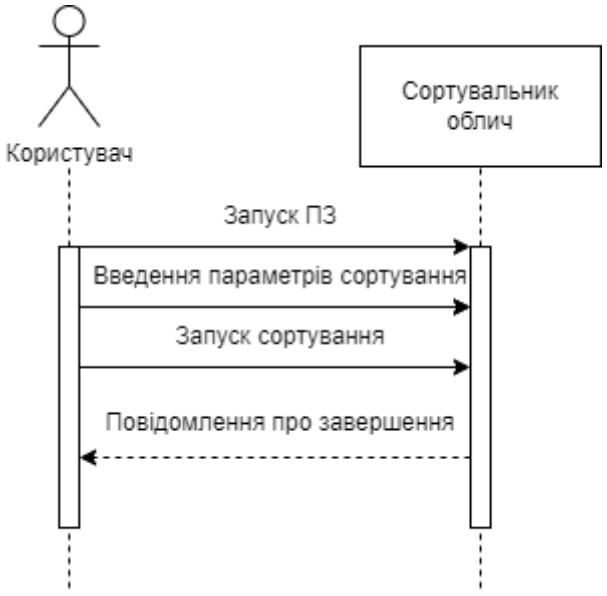

Рисунок 2.2.9 – Діаграма послідовності для варіанту використання «обирання параметрів сортування та запуск сортування»

Сценарій «обирання параметрів нейронної не межі та її запуск»: спочатку користувач обирає параметри нейронної мережі, параметри ядер згортки, а потім запускає нейронну мережу. Діаграма послідовності представлена на рис. 2.2.10.

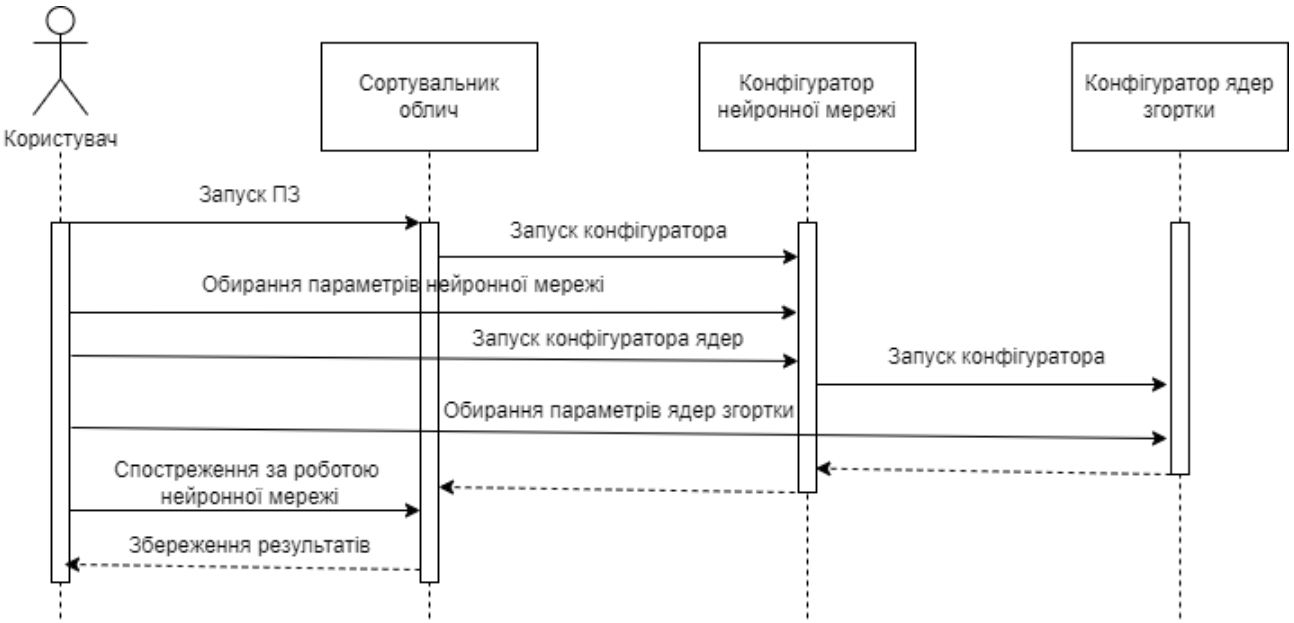

Рисунок 2.2.10 – Діаграма послідовності для варіанту використання «обирання параметрів нейронної не межі та її запуск»

2.2.4. Вибір мови програмування

Для створення ПЗ була використана мова програмування C#. Мова C# у Visual Studio IDE надає можливість гнучкої роботи із інтерфейсами, файлами, класами і іншими базовими елементами мови програмування. C# це об’єктно-орієнтована мова програмування, що дозволяє розподілити обов’язки програми між її класами.

Для відображення інтерфейсу була використана базова бібліотека «.NET» версії 4.5. Бібліотека .NET надає можливість швидко розробляти інтерфейси програми, що пришвидшує її розробку.

Недоліком .NET є те, що додатки основані на ньому можуть бути запущені лише на Windows.

Висновки до пункту 2

На стадії проектування системи були спроектовані ескізи інтерфейсу користувача, основі класи програми, зв’язки між ними та була обрана мова проектування. На основі розроблених у цьому розділі деталей проектування програмного комплексу, був написаний основний код програмного продукту, який має пройти тестування.

3. ТЕСТУВАННЯ ТА ВІДЛАГОДЖЕННЯ

3.1. Тестування програми створення зображень облич

Якщо порівнювати методи чорної та білої скриньки то можна виділити такі їх особливості:

Метод чорної скриньки створений для того, щоб перевіряти неправильно реалізовані методи, помилки інтерфейсів та помилки при використанні граничних значень. Однак тестування чорною скринькою не покриває усі варіанти вхідних даних, та не дозволяє виділяти код програми, що не виконається ні у якому разі.

Метод білої скриньки створений для тестування на будь-якому етапі розробки програми, у тому числі ранньому, також дозволяє перевірити майже всі шляхи виконання програми. Однак цей метод може опинитися надмірним, так як зачасту розробники проводять тестування деяких функцій на моменті від лагодження.

Програма створення зображень облич має складний інтерфейс із великою кількістю параметрів, для яких слід перевірити роботу програми на граничних значеннях, тому найкращим варіантом тестування цієї програми буде метод чорної скриньки.

Для тестування методом чорної скриньки слід виділити класи еквівалентності (табл. 3.1.1):

Таблиця 3.1.1 – Класи еквівалентності програми створення облич

Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
Кількість зображень для створення	Значення є цілим числом (обмеження типу int) у границях від 1 до 1000000 (1)	Значення не є цілим числом (обмеження типу int) (2)

Продовження таблиці 3.1.1

		Значення є цілим числом (обмеження типу int) що знаходиться не у границях від 1 до 1000000 (3)
Hindrance	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (4)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (5)	
Angle	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (6)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (7)	
Position X	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (8)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (9)	
Position Y	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (10)	

Продовження таблиці 3.1.1

	Значення на графічному елементі trackbar виставлено у крайнє праве положення (11)	
Size X	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (12)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (13)	
Size Y	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (14)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (15)	
Top Distortion	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (16)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (17)	
Bottom Distortion	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (18)	

Продовження таблиці 3.1.1

	Значення на графічному елементі trackbar виставлено у крайнє праве положення (19)	
Ears Size X	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (20)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (21)	
Ears Size Y	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (22)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (23)	
Ears Shift	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (24)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (25)	
Eyes Size X	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (26)	

Продовження таблиці 3.1.1

	Значення на графічному елементі trackbar виставлено у крайнє праве положення (27)	
Eyes Size Y	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (28)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (29)	
Eyes Shift X	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (30)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (31)	
Eyes Shift Y	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (32)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (33)	
Nose Size X	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (34)	

Продовження таблиці 3.1.1

	Значення на графічному елементі trackbar виставлено у крайнє праве положення (35)	
Nose Size Y	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (36)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (37)	
Nose Shift	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (38)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (39)	
Mouth Size X	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (40)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (41)	
Mouth Size Y	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (42)	

Продовження таблиці 3.1.1

	Значення на графічному елементі trackbar виставлено у крайнє праве положення (43)	
Mouth Shift	Значення на графічному елементі trackbar виставлено у крайнє ліве положення (44)	
	Значення на графічному елементі trackbar виставлено у крайнє праве положення (45)	

Нижче представлена таблиця еквівалентних розбиттів (табл. 3.1.2.).

Таблиця 3.1.2. – Еквівалентне розбиття програми створення облич

Клас, що покривається	Тест	Вхід	Очікуваний вихід
2	1	n = “a”	Програма не дозволить ввести такі значення
3	2	n = “0”	Програма не дозволить ввести такі значення
1, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44	3	n = “1”, усі trackbar переведені у крайнє ліве положення	Програма створить лице
1, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45	4	n = “1”, усі trackbar переведені у крайнє праве положення	Програма створить лице

Результати тестування приведені на рисунках 3.1.1-3.1.4.

Рисунок 3.1.1 – Перший тест еквівалентності до програми створення облич

Рисунок 3.1.2 – Другий тест еквівалентності до програми створення облич

Рисунок 3.1.3 – Третій тест еквівалентності до програми створення облич

Рисунок 3.1.4 – Четвертий тест еквівалентності до програми створення облич

3.2. Тестування програми сортування облич

На відміну від попередньої програми, програма сортування облич є невеликою, не має важких інтерфейсів, і великої кількості параметрів. Тому найкращим методом для її тестування буде метод білої скриньки.

Метод 1 – метод сортування Шелла. Код методу:

```
int n = count / 2;
while (n >= 1)
{
    for (int i = n; i < count; i++)
    {
        int j = i;
        while ((j >= n) && (getValue(ref faceConfigs, from + j, ref param) < getValue(ref faceConfigs, from + j - n, ref param)))
        {
            swapConfigElements(ref faceConfigs, from + j, from + j - n);
            j = j - n;
        }
    }
    n = n / 2;
}
```

Таблиця 3.2.1 – Комбінаторне покриття рішень для метода сортування Шелла

	n >= 1 //1		i < count //2		(j >= n) && (getValue(ref faceConfigs, from + j, ref param) < getValue(ref faceConfigs, from + j - n, ref param)) //3	
	+	-	+	-	+	-
T1		*				
T2	*		*	*	*	*
T3	*		*	*		*

Інші умови не виконуються, бо:

– Якщо у //1, буде негативний результат, то програма не дійде до //2 та //3.

– Умови //2 і //3, не можуть бути при першій ітерації від’ємними через структуру методу, також ці умови є умовами циклу, тому після певної кількості позитивних результатів з’явиться і негативний результат.

На тестування методу сортування Шелла потрібно зробити всього 2 тести відповідно до табл. 3.2.1:

Тест 1:

Вхід: count = 1, faceConfig = { 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
Вихід: Програма не провела сортування масиву через те, що у ньому є лише один елемент (див. рис. 3.2.1).

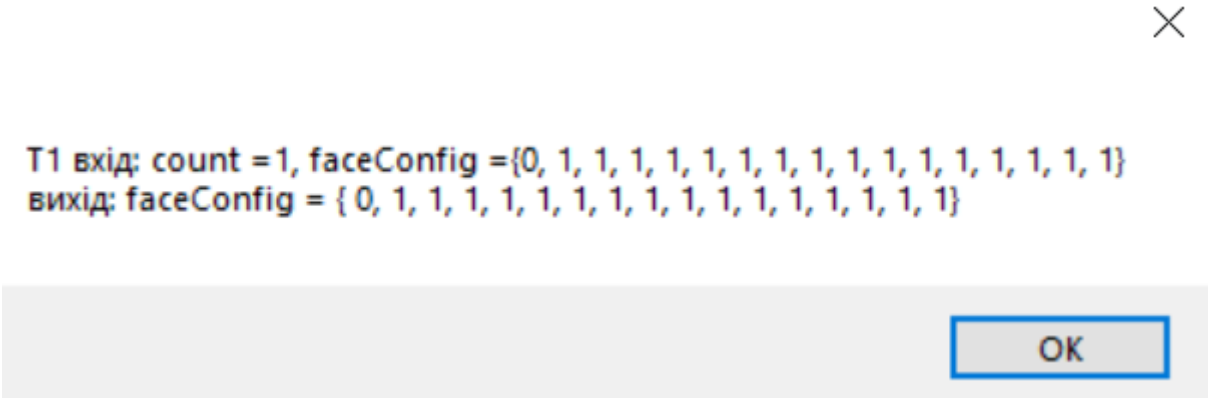

Рисунок 3.2.1 – Результат першого теста для метода сортування Шелла

Тест 2:

Вхід: count = 2, faceConfig = { {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1} , {1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}};
Вихід: Програма не провела сортування масиву через те, що масив вже відсортований. (див. рис. 3.2.2).

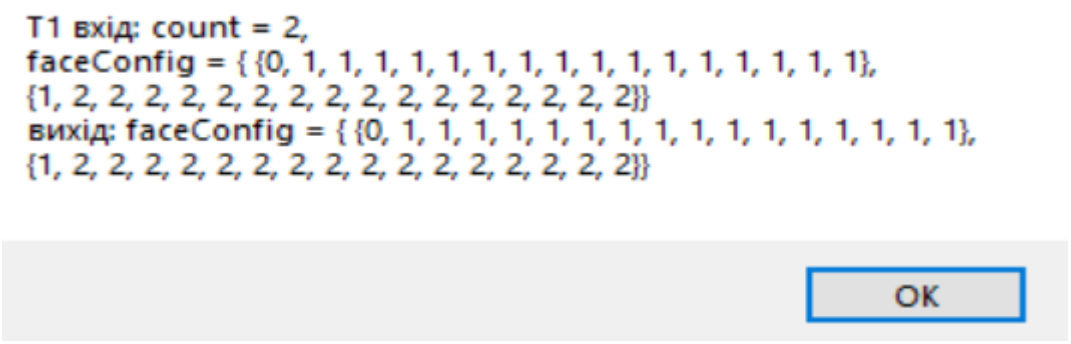

Рисунок 3.2.2 – Результат другого теста для метода сортування Шелла

Тест 3:

Вхід: count = 2, faceConfig = { {1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2} , {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};
Вихід: Програма провела сортування масиву. (див. рис. 3.2.3).

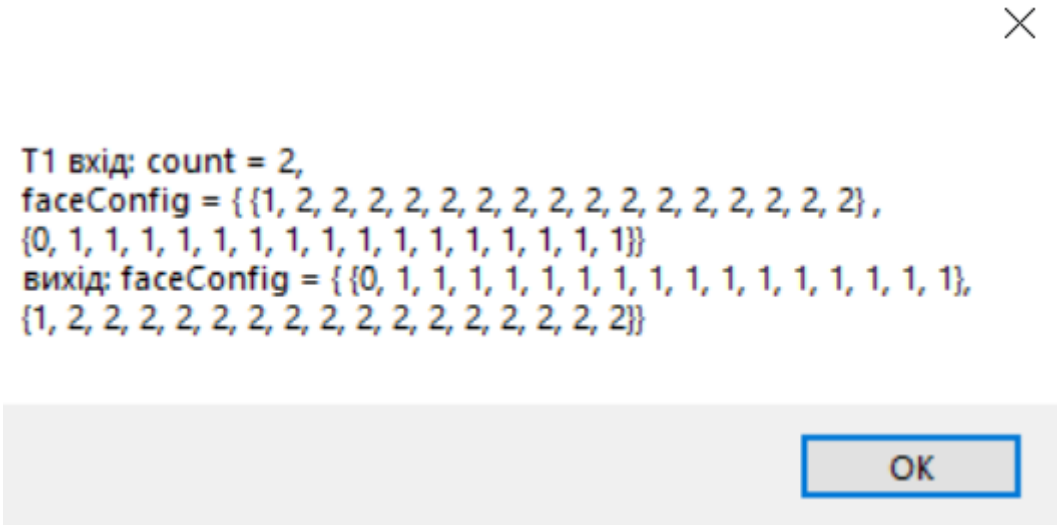

Рисунок 3.2.3 – Результат третього теста для метода сортування Шелла

3.3. Тестування нейронної мережі

Тестування нейронної мережі білою і чорною скринькою зазвичай не проводяться. Для нейронної мережі тестування проходить у форматі перевірки на адекватність: звичайний запуск програми та перевірка декількох сценаріїв роботи на отримання очікуваних результатів.

Тест 1 – тестування за випадково-відсортованими параметрами. Очікуваний вихід – всі чотири відстежуємі параметри повинні покращуватись із часом. Результати тестування зображенні на рис. 3.3.1-3.3.4.

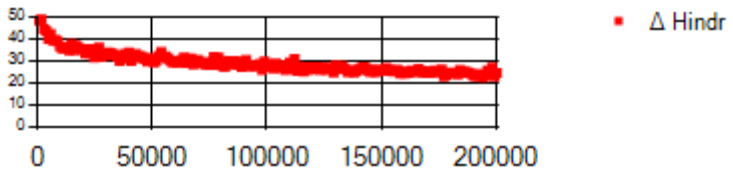

Рисунок 3.3.1 – Діаграма модуля помилки вгадування облич для першого тесту

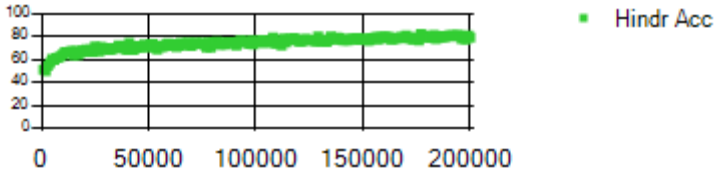

Рисунок 3.3.2 – Діаграма точності вгадування облич для першого тесту

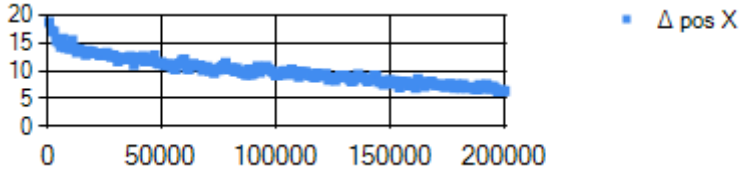

Рисунок 3.3.3 – Діаграма модуля помилки вгадування X координати для першого тесту

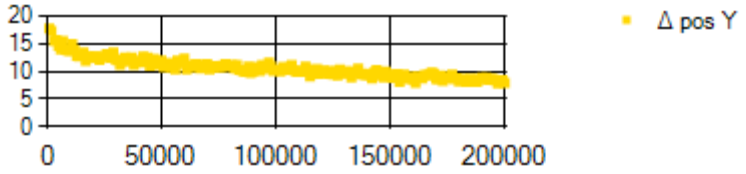

Рисунок 3.3.4 – Діаграма модуля помилки вгадування Y координати для першого тесту

Тест 1 – тестування за параметрами відсортованими так, що спочатку йдуть обличчя, а потім зображення що не є обличчями. Очікуваний вихід – точність вгадування облич майже відразу приблизиться до 100%, потім по середині випробування, коли зміниться тип зображень, точність впаде, і знову повернеться до максимальної. Модуль помилки таких вгадувань буде зворотній до точності вгадувань. Модулі помилок X та Y координат будуть поступово зменшуватись, а після зміни типу зображень, коли програма перестає перевіряти її, помилки зменшуються до мінімальних. Результати тестування зображенні на рис. 3.3.5-3.3.8.

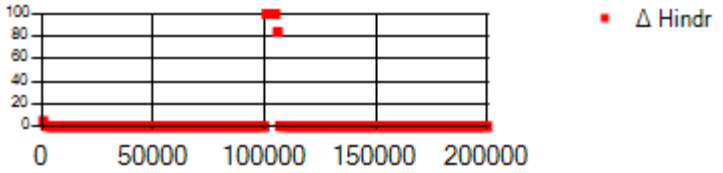

Рисунок 3.3.5 – Діаграма модуля помилки вгадування облич для другого тесту

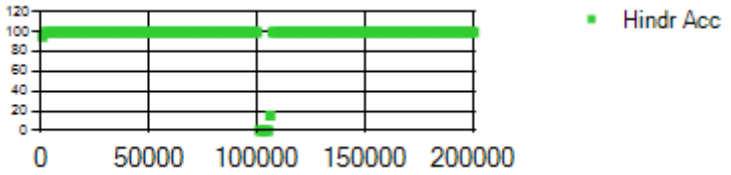

Рисунок 3.3.6 – Діаграма точності вгадування облич для другого тесту

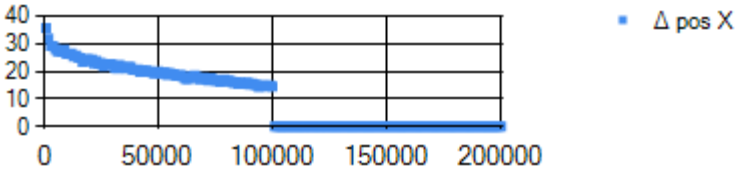

Рисунок 3.3.7 – Діаграма модуля помилки вгадування X координати для другого тесту

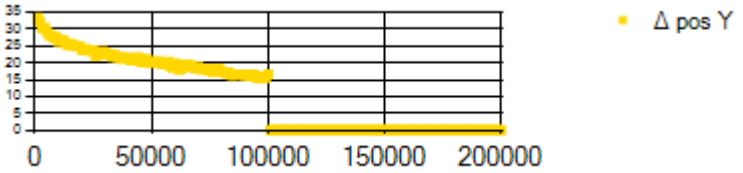

Рисунок 3.3.8 – Діаграма модуля помилки вгадування Y координати для другого тесту

Висновки до пункту 3

Програмний комплекс успішно пройшов тестування, де кожна програма була протестована своїм методом. Із недоліків можна побачити що на максимально викручених параметрах програма створення облич створює обличчя, які мінімально схожі із людськими. У всьому іншому поведінка програм була цілком очікуваною і правильною.

4. ОПИС ПРОГРАМНОГО ПРОДУКТУ

Створюваний програмний комплекс призначений для полегшення тестування нейронних мереж у умовах обмеженості вхідних даних. Також програмний комплекс дозволяє аналізувати вплив множника навчання та модифікація ядер згорток. Приклади тестування нейронних мереж приведені у розділах 4.1-4.3.

4.1. Вплив множника навчання на результат навчання нейронної мережі

Множник навчання – параметр у нейронних мережах який визначає на скільки ваги нейронної мережі будуть змінюватись в залежності від середньої похибки. Для тестування була вибрана вибірка відсортована випадковим чином, яка була протестована на чотирьох різних множниках навчання, а саме: 0.5, 0.1, 0.02, 0.004. Результати тестування представленні на рис. 4.1.1-4.1.4.

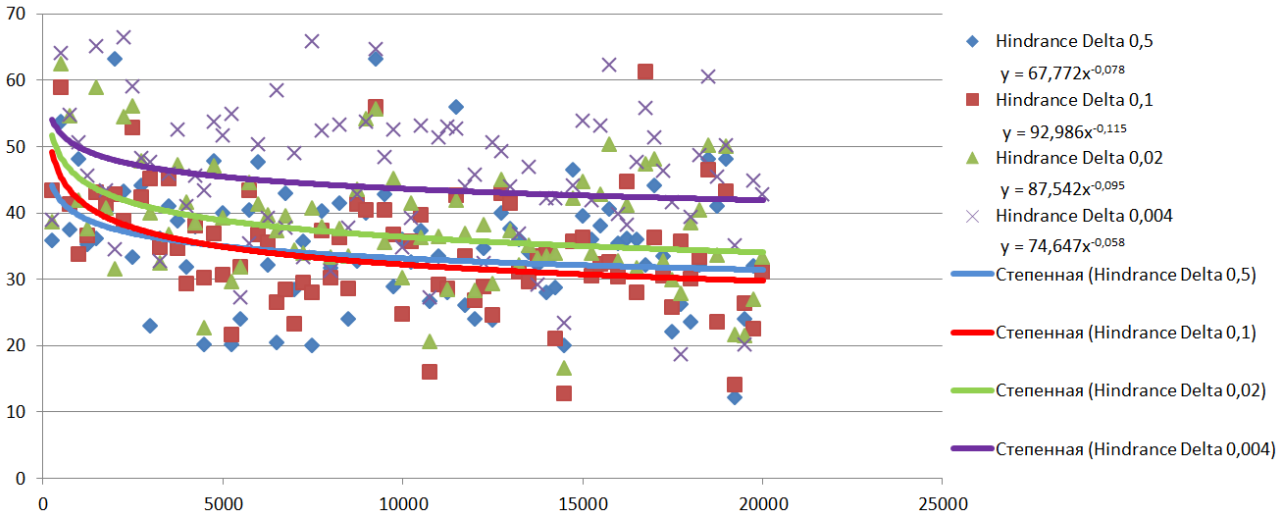

Рисунок 4.1.1 – Діаграма модулів помилки вгадування облич

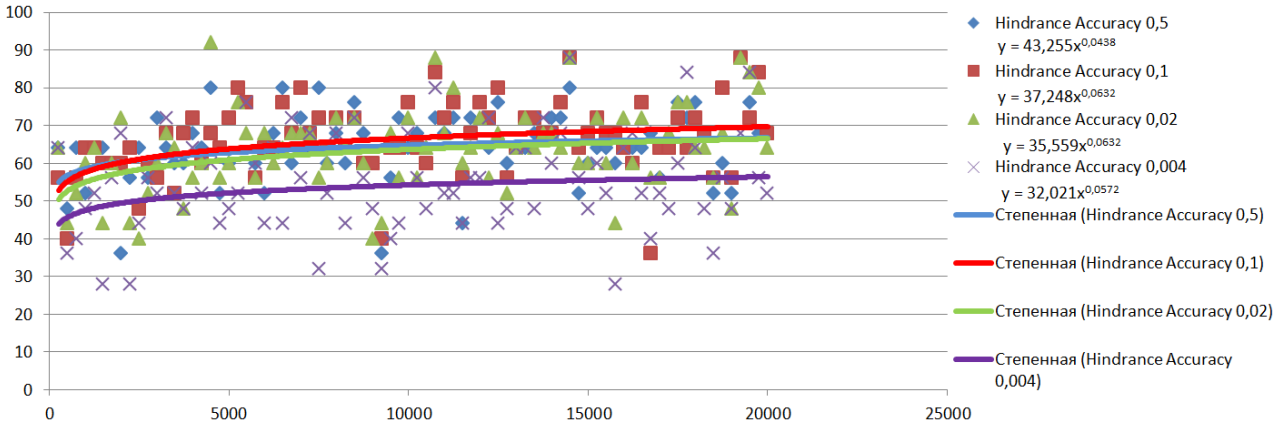

Рисунок 4.1.2 – Діаграма точності вгадування облич

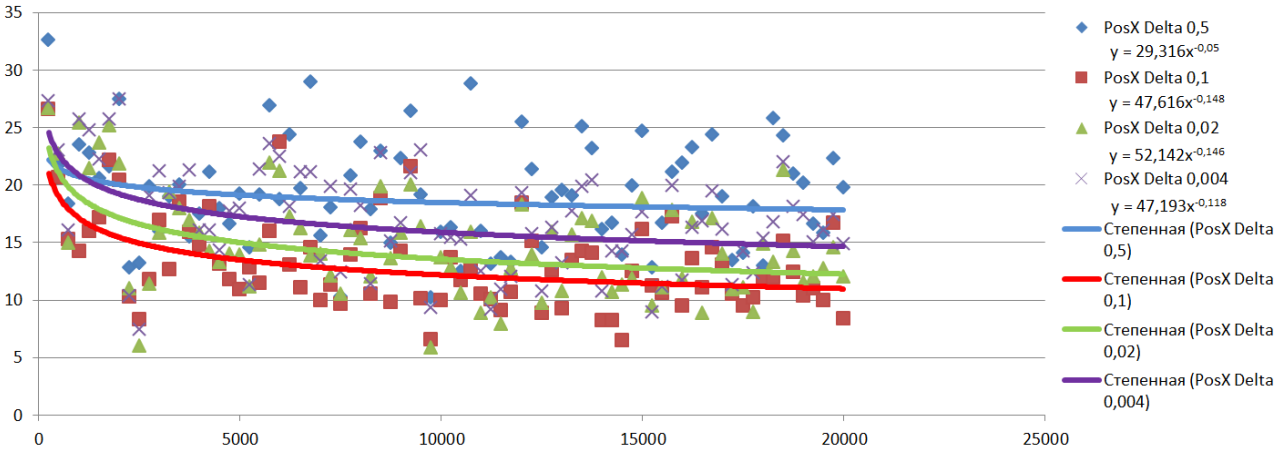

Рисунок 4.1.3 – Діаграма модуля помилки вгадування X координати

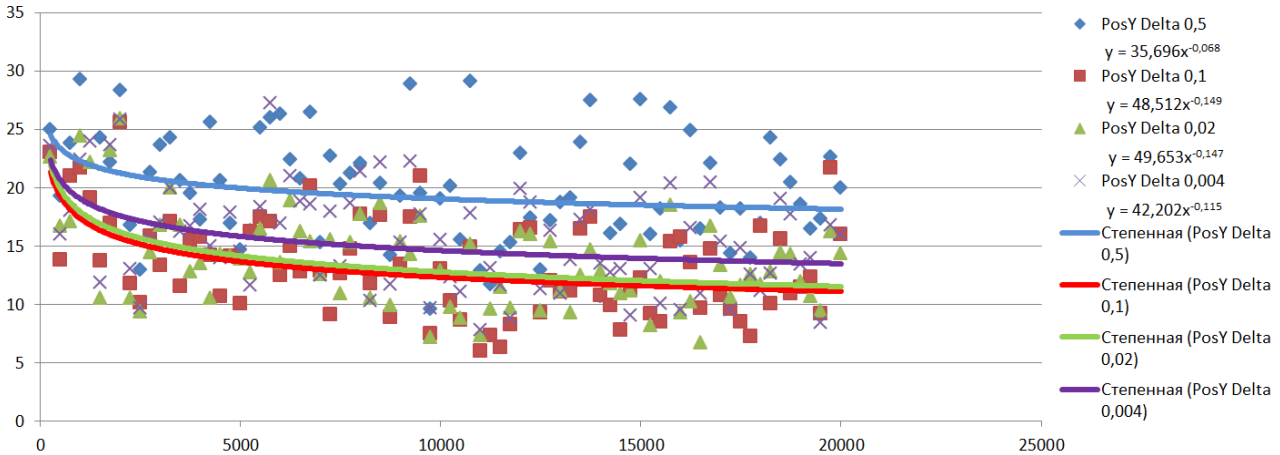

Рисунок 4.1.4 – Діаграма модуля помилки вгадування Y координати

4.2. Результати навчання нейронної мережі на вибірках із однаковим діапазоном розкиду параметрів облич

Для тестування та навчання нейронної мережі були створені вибірки із маленьким, середнім, та великим розкидом параметрів створення облич. Вибірки для навчання та тестування створюються окремо один від одного, і при виборі виборок обираються вибірки із однаковим розкидом параметрів. Результати тестування представленні на рис. 4.2.1-4.2.4.

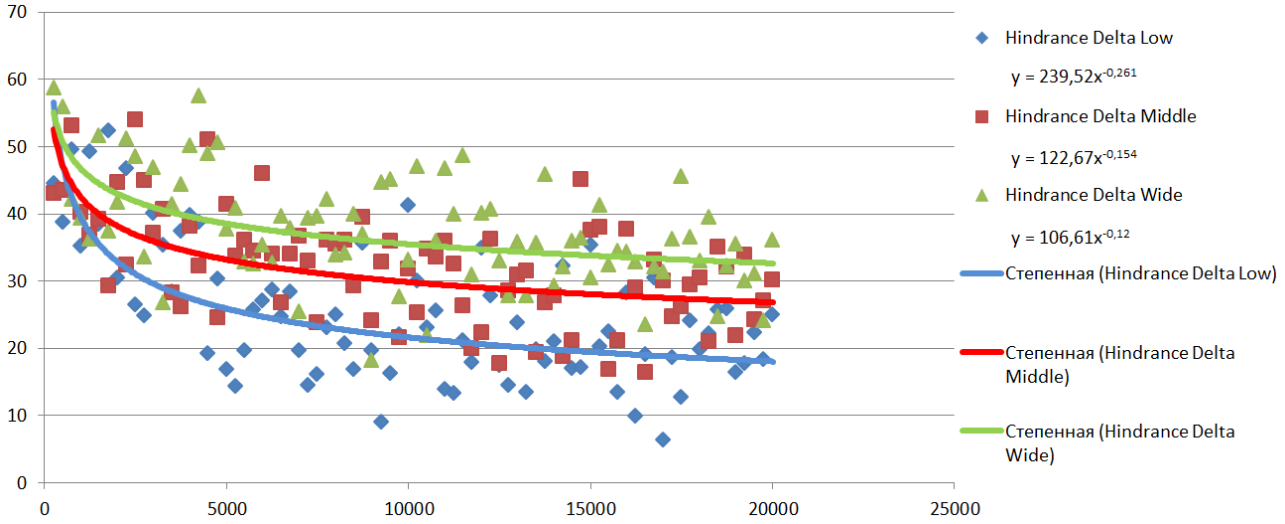

Рисунок 4.2.1 – Діаграма модулів помилки вгадування облич

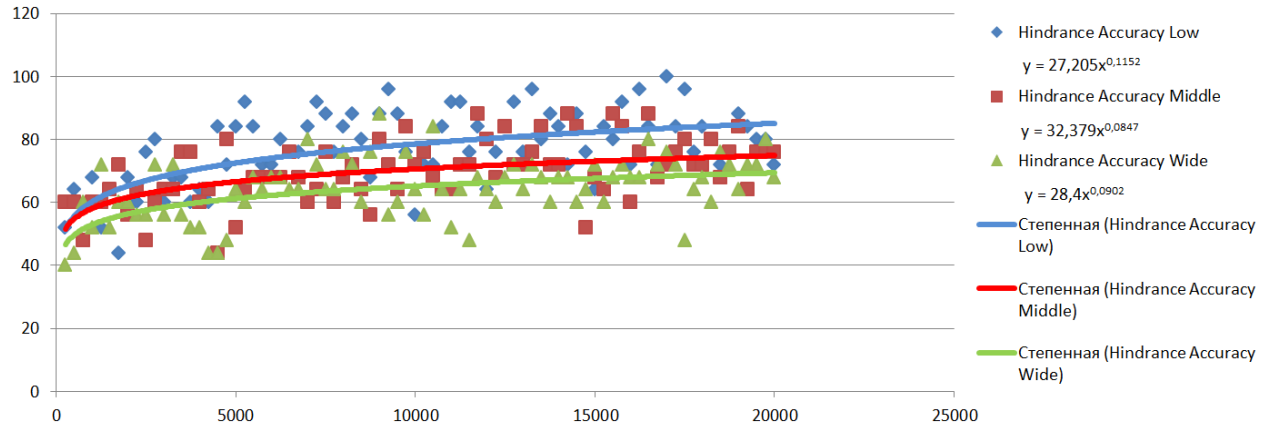

Рисунок 4.2.2 – Діаграма точності вгадування облич

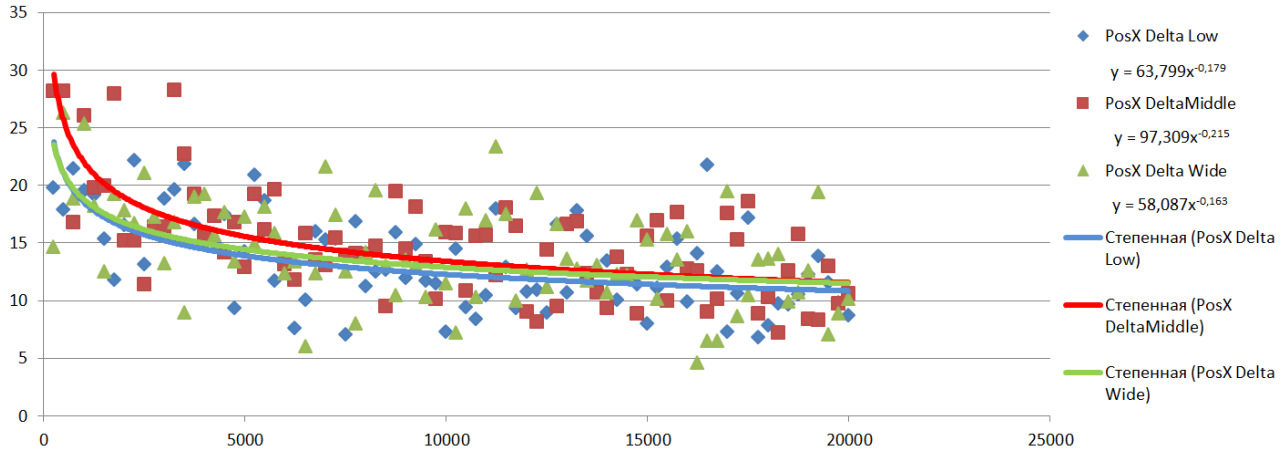

Рисунок 4.2.3 – Діаграма модуля помилки вгадування X координати

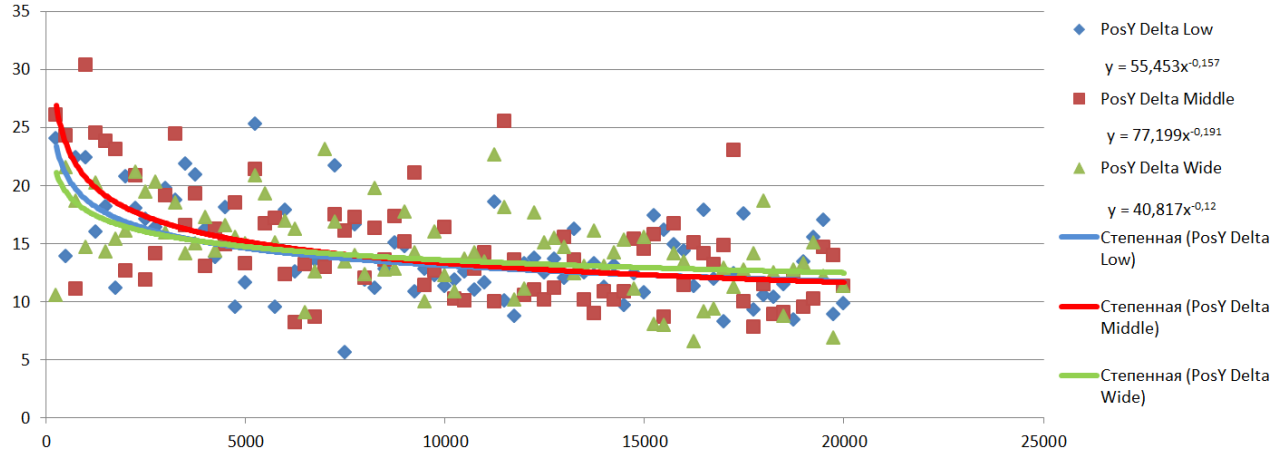

Рисунок 4.2.4 – Діаграма модуля помилки вгадування Y координати

4.3. Результати навчання нейронної мережі на вибірках із різним діапазоном розкиду параметрів облич

Для тестування та навчання були обрані вибірки із попереднього розділу, однак при їх виборі у нейронній мережі вибираються вибірки із різними розкидами параметрів. Результати тестування представленні на рис. 4.3.1-4.3.4.

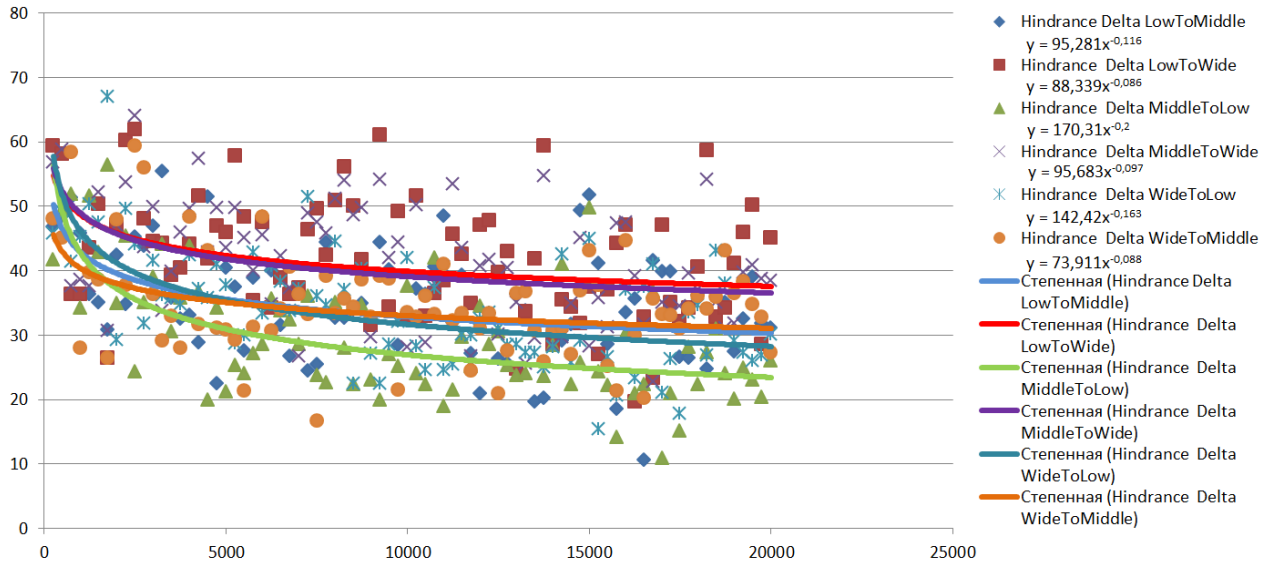

Рисунок 4.3.1 – Діаграма модулів помилки вгадування облич

Рисунок 4.3.2 – Діаграма точності вгадування облич

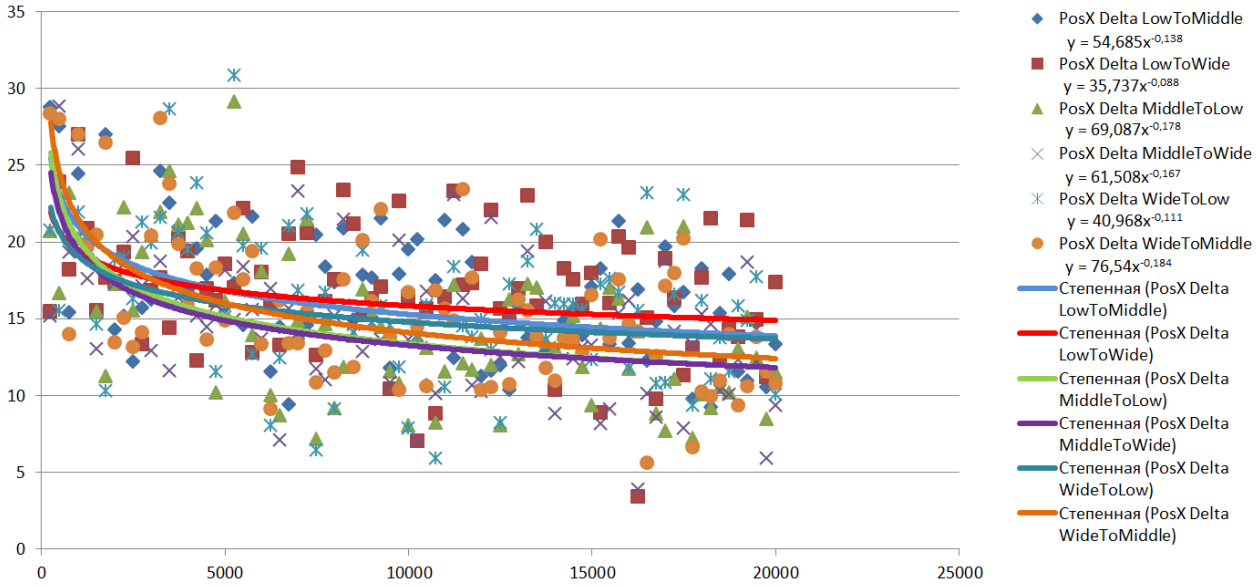

Рисунок 4.3.3 – Діаграма модуля помилки вгадування X координати

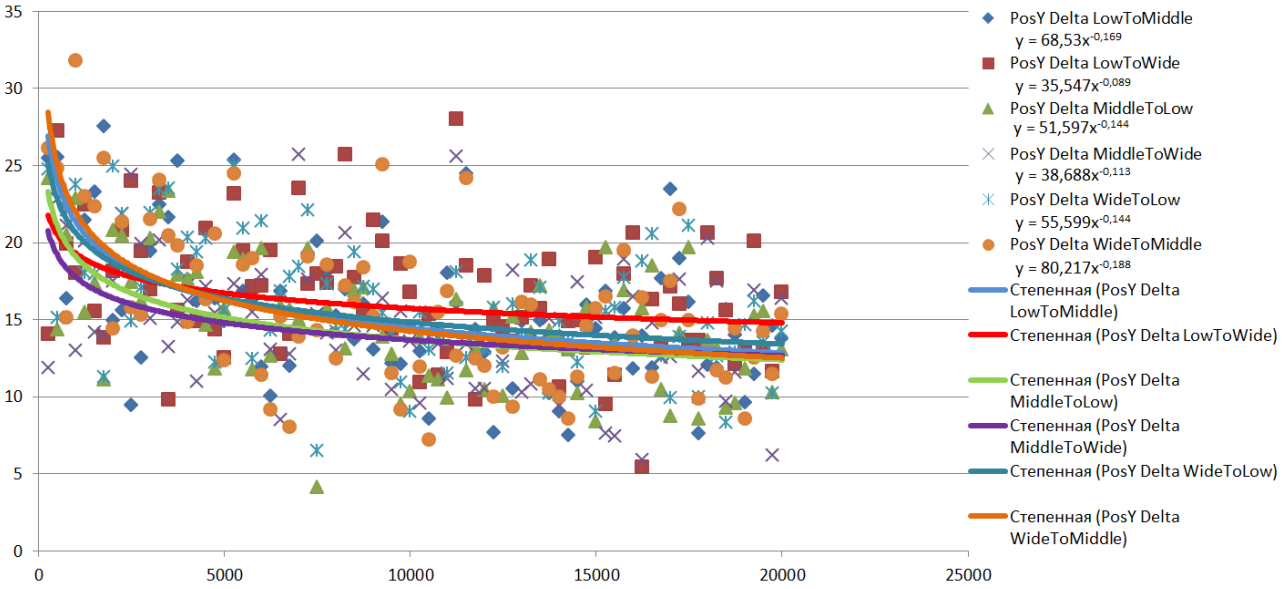

Рисунок 4.3.4 – Діаграма модуля помилки вгадування Y координати

Висновки до пункту 4

Із проведених тестів можна зазначити що найкращими множниками навчання є 0.1 та 0.02. При виставленні більшого множника навчання нейронна мережа на початку швидше навчається, що робить його хорошим вибором для невеликих вибірок, але така нейронна мережа не може визначати невеличкі деталі, що є основою облич, тому все ж таки не підходить для роботи із ними.

При виставленні меншого множника нейронна мережа теоретично видасть кращий результат через своє вміння підмічати деталі, однак швидкість навчання значно зменшується, тож для відносно невеликих вибірок він не підходить.

Тестування на вибірках із однаковим діапазоном розкиду показало, що чим менша варіація облич, тим нейронна мережа швидше навчається та отримує кращий фінальний результат.

Тестування на вибірках із різним діапазоном розкиду параметрів облич показало що при навчанні на більш широкій вибірці точність вгадування облич, що були створенні із більш обмеженими параметрами, зростала. Точність вгадування розташування обличчя була найкращою при навчанні на середній вибірці облич.

ВИСНОВКИ

Результатом роботи є розроблений програмний комплекс для інтерактивного навчання і тестування загорткової нейронної мережі. Під час виконання дипломної роботи були написані три додатки на основі фреймворку .Net, на мові C#. Програмні продукти були успішно протестовані, вихідні зображення при створенні їх на граничних параметрів – проаналізовані. Додатки дозволяють генерувати обличчя із певним розкидом параметрів, сортувати їх за будь-якими параметрами і проводити навчання і тестування загорткової нейронної мережі.

При проведенні дослідження за допомогою програмного комплексу було виділено, що: для відносно невеликої вибірки розміром 20000 елементів для навчання найкращим множником навчання є множник діапазоном від 0.1 до 0.02; При використанні множника більшого за рекомендований, згорткова нейронна мережа з початку буде показувати кращі результати, однак при продовженні навчання нейронної мережі результати швидко перестануть покращуватись; При використанні множника меншого за рекомендований, згортковій нейронній мережі знадобиться значно більші кількості зображень та часу для навчання, однак результат буде кращим; Якщо при використанні нейронної мережі заздалегідь відомі приблизні показники оброблюваних облич, то для навчання їх розпізнавати слід використовувати вибірку або з саме таким розкидом параметрів облич, або із трохи більшим, що може покращити не тільки швидкість навчання, зменшуючи кількість зображень для навчання, а і фінальну точність загорткової нейронної мережі; Якщо згорткова нейронна мережа буде використовуватися для задач знаходження положення об’єктів, то слід використовувати більш вузький діапазон зображень, що може пришвидшити навчання і позитивно вплинути на результати навчання на вибірці середнього розміру.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Convolutional neural network [Virtual Resource] / BOZ / Mako001 / Abductive. – Edit: 2023 – 30 May – Access Mode : URL :

https://en.wikipedia.org/wiki/Convolutional_neural_network

2. Deep learning super sampling [Virtual Resource] / Олександр Кравчук / Olexa Riznyk. – Edit: 2022 – 5 May – Access Mode : URL :

https://uk.wikipedia.org/wiki/Deep_learning_super_sampling

3. Midjourney [Virtual Resource] / Kat Alpatova / Pvladko / Віщун. – Edit: 2023 – 9 May – Access Mode : URL :

https://uk.wikipedia.org/wiki/Midjourney

4. Deepfake [Virtual Resource] / Treetoes023 / Johnuniq / GeogSage / Doctor Astrocyte. – Edit: 2023 – 29 May – Access Mode : URL :

https://en.wikipedia.org/wiki/Deepfake

5. StyleGAN [Virtual Resource] / Richienb / Cosmia Nebula / Spideyunlimited / Grondilu. – Edit: 2023 – 26 May – Access Mode : URL :

https://en.wikipedia.org/wiki/StyleGAN

6. LeNet [Virtual Resource] / Swadim / Tohaomg. – Edit: 2022 – 3 November – Access Mode : URL :

https://uk.wikipedia.org/wiki/LeNet

7. AlexNet [Virtual Resource] / Swadim / Alex Blokha. – Edit: 2023 – 8 March – Access Mode : URL :

https://uk.wikipedia.org/wiki/AlexNet

8. ImageNet [Virtual Resource] / Materialscientist / Jaleks / Muhali / Thatsme314 / Glane23 / Jordi Burguet Castell. – Edit: 2023 – 27 March – Access Mode : URL :

https://en.wikipedia.org/wiki/ImageNet

9. VGG Very Deep Convolutional Networks (VGGNet) – What you need to know [Virtual Resource] / Gaudenz Boesch. – 2021 – 6 October – Access Mode : URL :

https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

10. Deep Learning: GoogLeNet Explained [Virtual Resource] / Richmond Alake. – 2020 – 23 December – Access Mode : URL :

https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765

11. Residual neural network [Virtual Resource] / LunarLullaby / Speedboys / Kaarelh / Olexa Riznyk. – Edit: 2023 – 19 May – Access Mode : URL :

https://en.wikipedia.org/wiki/Residual_neural_network

12. DenseNet [Virtual Resource] / likebupt / PeterCLu / v-chmccl. – 2021 – 11 April – Access Mode : URL :

https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/densenet?view=azureml-api-2

13. EfficientNet [Virtual Resource] / Tan. – 2020 – 6 June – Access Mode : URL :

https://paperswithcode.com/method/efficientnet

14. A Gentle Introduction to Cross-Entropy for Machine Learning [Virtual Resource] / Jason Brownlee. – 2019 – 21 October – Access Mode : URL :

https://machinelearningmastery.com/cross-entropy-for-machine-learning/

15. Backpropagation [Virtual Resource] / Jarble / Kaltenmeyer / Lineality / Sunsetsilk / SamuelRiv / Streded. – Edit: 2023 – 25 May – Access Mode : URL :

https://en.wikipedia.org/wiki/Backpropagation

16. Стохастичний градієнтний спуск [Virtual Resource] / Olexa Riznyk / SoloWay / Vlasenko D / MobyVan / Geohem / Iguanakostya2. – Edit: 2022 – 13

November – Access Mode : URL :

https://uk.wikipedia.org/wiki/Стохастичний_градієнтний_спуск

17. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning [Virtual Resource] / Jason Brownlee. – 2017 – 3 July – Access Mode : URL :

https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

18. Understanding RMSprop — faster neural network learning [Virtual Resource] / Vitaly Bushaev. – 2018 – 2 September – Access Mode : URL:

https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a

ДОДАТКИ

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖЕНО
1116130.01315-01-ЛЗ

ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ІНТЕРАКТИВНОГО НАВЧАННЯ ТА ТЕСТУВАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Технічне завдання

1116130.01315-01

Листів 11

2023

ЗМІСТ

1. ВВЕДЕННЯ

Програмний комплекс для інтерактивного навчання та тестування загорткової нейронної мережі, призначений для генерації виборок зображень облич із певним розбігом параметрів, сортування їх за певними параметрами та навчання і тестування нейронної мережі з ціллю дослідження впливу певних особливостей створеної вибірки на навчання та тестування загорткової нейронної мережі.

Причиною виникнення продукту є недостатня кількість досліджень у сфері навчання і тестування згорткових нейронних мереж.

Область застосування – інженери програмного забезпечення.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 07.12.22 №1209ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи – “ Програмний комплекс для інтерактивного навчання та тестування загорткової нейронної мережі ”. Керівник – Разносілін В. В.

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення: Продукт що розробляється дозволяє інженерам полегшити навчання роботи із згортковими нейронними мережами, дозволяючи швидко і інтерактивно створювати вибірки і аналізувати поведінку загорткової нейронної мережі на неї.

Експлуатаційне призначення: Нейронні мережі стрімко розвиваються, але велика кількість інженерів не проводить дослідження щодо швидкості і ефективності навчання нейронних мереж. Продукт допомагає інтерактивно тестувати створену згорткову нейронну мережу, що дозволить зрозуміти, які дані із вибірки є надмірними, а яких не вистачає.

4. ВИМОГИ ДО ПРОГРАМИ

4.1. Вимоги до функціональних характеристик

Можливість створювати зображення облич із певними параметрами, можливість вибору цих параметрів, можливість вибору параметрів сортування, можливість сортувати створенні обличчя за вказаними параметрами, можливість модифікувати ядра згортки, можливість модифікувати множник навчання.

4.2. Вимоги до надійності

Вимоги до надійності наступні: забезпечення стійкого функціонування програми; контроль вхідної і вихідної інформації; наявність архівної копії тексту програми на зовнішньому носії.

4.3. Умови експлуатації

Вимоги до кліматичних умов: температура – 21-25 С, відносна вологість 40-60%. Обслуговування не потрібне. Для роботи із ПЗ достатньо однієї людини, що має досвід роботи із ПК, і бажає проводити дослідження у сфері навчання і тестування згорткових нейронних мереж.

4.4. Вимоги до складу і параметрів технічних засобів

Склад технічних засобів: процесор з тактовою частотою 2 ГГЦ або вище; 6 Мб. місця на накопичувачі; 2 Гб. оперативної пам’яті.

4.5. Вимоги до інформаційної і програмної сумісності

Програма має функціонувати під управлінням ОС Windows 10\11.

Програма написана на мові програмування C#. Перевагою цієї мови програмування є те, що вона об’єктно-орієнтована, тобто код можна поділити на логічні частини. Також ця мова має великий рівень безпеки і створення для написання додатків на операційну систему Windows. На системі має бути встановлений .NET Framework 4.5 або вище.

4.6. Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинно бути вказана назва продукту, мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

4.7. Вимоги до транспортування і зберігання

Транспортування повинно проводитись в упаковці. Умови зберігання повинні забезпечувати безпеку продукту.

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

1. специфікація;
2. текст програми.

Програмна документація повинна відповідати вимогам ДСТУ [1].

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця А.1 – Стадії та етапи розробки

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Вступ	12.09.22 – 26.10.22	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	27.10.22 – 04.03.23	
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	05.03.23 – 31.04.23	
4	Постановка задачі, технічне завдання	01.05.23 – 07.05.23	30%
5	Техніко-економічні показники	08.05.23 – 15.05.23	
6	Розробка інструментальних засобів дослідження	16.05.23 – 21.05.23	
7	Виконання досліджень	22.05.23 – 28.05.23	60%
8	Оформлення тез доповідей	29.05.23 – 01.06.23	
9	Оформлення статті у фаховий журнал	02.06.23 – 06.06.23	
10	Оформлення пояснювальної записки	07.06.23 – 11.06.23	
11	Розробка демонстраційних матеріалів	12.06.23 – 18.06.23	100%
12	Подання кваліфікаційної роботи до кафедри	22.06.23	
13	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	28.06.23	

7. ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль виконання здійснює керівник розробки Разносілін В. В. Прийом здійснюється уповноваженою комісією.

8. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Б

Специфікація

ЗАТВЕРДЖЕНО
1116130.01315-01-ЛЗ

ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ІНТЕРАКТИВНОГО НАВЧАННЯ ТА ТЕСТУВАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Специфікація

1116130.01315-01

Листів 2

2023

Специфікації

Таблиця 5.2. – Специфікації

Позначення	Найменування	Примітка
	Документація	
1116130.01315-01-ЛЗ	Лист затвердження	
1116130.01315-01	Технічне завдання	
1116130.01315-01-ЛЗ	Лист затвердження	
1116130.01315-01	Специфікація	
1116130.01315-01 12 01-ЛЗ	Лист затвердження	
1116130.01315-01 12 01	Текст програми	

ДОДАТОК В

Листи затвердження

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
_____ Анатолій РАДКЕВИЧ
07.12.2022

ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ІНТЕРАКТИВНОГО НАВЧАННЯ ТА ТЕСТУВАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01315-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
07.12.22
Керівник розробки
_____ Валентин РАЗНОСІЛІН
07.12.22
Виконавець
_____ Дмитро ЧЕРКАС
07.12.22
Нормконтролер
_____ Світлана ВОЛКОВА
07.12.22

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
_____ Анатолій РАДКЕВИЧ
07.12.2022

ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ІНТЕРАКТИВНОГО НАВЧАННЯ ТА ТЕСТУВАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Специфікація
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01315-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
07.12.22
Керівник розробки
_____ Валентин РАЗНОСІЛІН
07.12.22
Виконавець
_____ Дмитро ЧЕРКАС
07.12.22
Нормконтролер
_____ Світлана ВОЛКОВА
07.12.22

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
_____ Анатолій РАДКЕВИЧ
07.12.2022

ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ІНТЕРАКТИВНОГО НАВЧАННЯ ТА ТЕСТУВАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01315-01 12 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
07.12.22
Керівник розробки
_____ Валентин РАЗНОСІЛІН
07.12.22
Виконавець
_____ Дмитро ЧЕРКАС
07.12.22
Нормконтролер
_____ Світлана ВОЛКОВА
07.12.22

2023

ДОДАТОК Г

Текст програми

ЗАТВЕРДЖЕНО
1116130.01315-01 12 01-ЛЗ

ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ІНТЕРАКТИВНОГО НАВЧАННЯ ТА ТЕСТУВАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Текст програми

1116130.01315-01 12 01

Листів 55

2023

Програма 1 – Створення облич:

Mainform.cs:

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.IO;

namespace faceConturGenerator
{
   public partial class MainForm : Form
   {
      private Random rand = new Random();
      private Settings settings;
      private Graphics g;
      Bitmap image;
      private string path = "E:\\diploma\\New\\test\\";

      private struct Current
      {
         public int hindrance { set; get; }
         public int angle { set; get; }
         public int posX { set; get; }
         public int posY { set; get; }
         public int sizeX { set; get; }
         public int sizeY { set; get; }
         public int topDistortion { set; get; }
         public int bottomDistortion { set; get; }
         public int sizeEarsY { set; get; }
         public int sizeEarsX { set; get; }
         public int shiftEars { set; get; }
         public int eyeSizeX { set; get; }
         public int eyeSizeY { set; get; }
         public int eyeShiftX { set; get; }
         public int eyeShiftY { set; get; }
         public int noseSizeX { set; get; }
         public int noseSizeY { set; get; }
         public int noseShift { set; get; }
         public int mounthSizeX { set; get; }
         public int mounthSizeY { set; get; }
         public int mounthShift { set; get; }
      }
      private Current currentFace;

      //Field for hindrance image
      public bool hindrance = false;
      //generating center of the face (15; 85)
      public int positionMinX = 25;
      public int positionMaxX = 75;
      public int positionMinY = 25;
      public int positionMaxY = 75;
      //angle of face (-15;15)
      public int angleMin = -15;
      public int angleMax = 15;

      //Face counter:
      //size (10; 45), (0; 50)
      public int sizeMinX = 25;
      public int sizeMaxX = 45;
      public int sizeMinYindependence = 0;
      public int sizeMaxYindependence = 50;
      //top distortion (0; 100)
      public int topDistortionMin = 100;
      public int topDistortionMax = 100;
```

```
        //bottom distortion (0; 100)
        public int bottomDistortionMin =
50;
        public int bottomDistortionMax =
100;

        //Ears:
        //ear Size (20; 35), (60; 100)
        public int earSizeMinY = 20;
        public int earSizeMaxY = 35;
        public int
earSizeMinXdependence = 60;
        public int
earSizeMaxXdependence = 100;
        //ear shift (35; 65)
        public int earShiftMin = 35;
        public int earShiftMax = 75;

        //Eyes:
        //eyes Size (20; 50), (15; 40)
        public int eyeSizeMinX = 20;
        public int eyeSizeMaxX = 50;
        public int eyeSizeMinY = 15;
        public int eyeSizeMaxY = 40;
        //eyes shift (15; 40), (10; 50)
        public int eyeShiftMinX = 15;
        public int eyeShiftMaxX = 40;
        public int eyeShiftMinY = 10;
        public int eyeShiftMaxY = 50;

        //Nose:
        //size (15; 30), (15; 40)
        public int noseSizeMinX = 15;
        public int noseSizeMaxX = 30;
        public int noseSizeMinY = 15;
        public int noseSizeMaxY = 40;
        //shift (-10; 15)
        public int noseShiftMin = -10;
        public int noseShiftMax = 15;

        //Mounth:
        //size (30; 65), (15; 30)
        public int mounthSizeMinX = 30;
        public int mounthSizeMaxX = 65;
        public int mounthSizeMinY = 15;
        public int mounthSizeMaxY = 30;
        //shift (30; 65)
        public int mounthShiftMin = 30;
        public int mounthShiftMax = 65;

        public MainForm()
        {
            InitializeComponent();
            pictureBox1.Image = new
Bitmap(pictureBox1.Width,
pictureBox1.Height);
            g =
Graphics.FromImage(pictureBox1.Ima
ge);
        }

        private void drawEyes()
        {
            int W = pictureBox1.Width;
            int H = pictureBox1.Height;
            int posX = currentFace.posX;
            int posY = currentFace.posY;
            int sizeX = currentFace.sizeX;
            int sizeY = currentFace.sizeY;

            if (currentFace.hindrance == 1)
            {
                posX =
rand.Next(positionMinX,
positionMaxX);
                posY =
rand.Next(positionMinY,
positionMaxY);
            }

            int eyeSizeX =
rand.Next(eyeSizeMinX,
```

```
eyeSizeMaxX);
            int eyeSizeY =
rand.Next(eyeSizeMinY,
eyeSizeMaxY);
            int eyeShiftX =
rand.Next(eyeShiftMinX,
eyeShiftMaxX);
            int eyeShiftY =
rand.Next(eyeShiftMinY,
eyeShiftMaxY);

            currentFace.eyeSizeX =
eyeSizeX;
            currentFace.eyeSizeY =
eyeSizeY;
            currentFace.eyeShiftX =
eyeShiftX;
            currentFace.eyeShiftY =
eyeShiftY;

            g.DrawEllipse(Pens.Black, W *
posX / 100 - W / 2 - ((eyeSizeX +
eyeShiftX) * sizeX / 2) / 100,
                                    H * posY
/ 100 - H / 2 - ((eyeSizeY + eyeShiftY)
* sizeY / 2) / 100,
                                    eyeSizeX
* sizeX / 200, eyeSizeY * sizeY / 200);

            if (currentFace.hindrance == 1)
            {
                posX =
rand.Next(positionMinX,
positionMaxX);
                posY =
rand.Next(positionMinY,
positionMaxY);
            }

            g.DrawEllipse(Pens.Black, W *
posX / 100 - W / 2 + (eyeShiftX *
sizeX / 2) / 100,
                                    H * posY
/ 100 - H / 2 - ((eyeSizeY + eyeShiftY)
* sizeY / 2) / 100,
                                    eyeSizeX
* sizeX / 200, eyeSizeY * sizeY / 200);
        }

        private void drawNose()
        {
            int W = pictureBox1.Width;
            int H = pictureBox1.Height;
            int posX = currentFace.posX;
            int posY = currentFace.posY;
            int sizeX = currentFace.sizeX;
            int sizeY = currentFace.sizeY;

            if (currentFace.hindrance == 1)
            {
                posX =
rand.Next(positionMinX,
positionMaxX);
                posY =
rand.Next(positionMinY,
positionMaxY);
            }

            int noseSizeX =
rand.Next(noseSizeMinX,
noseSizeMaxX);
            int noseSizeY =
rand.Next(noseSizeMinY,
noseSizeMaxY);
            int noseShift =
rand.Next(noseShiftMin,
noseShiftMax);

            currentFace.noseSizeX =
noseSizeX;
            currentFace.noseSizeY =
noseSizeY;
```

```
            currentFace.noseShift =
noseShift;

            g.DrawEllipse(Pens.Black, W *
posX / 100 - W / 2 - (noseSizeX/2 *
sizeX / 2) / 100,
                                        H * posY
/ 100 - H / 2 - ((noseSizeY/2 -
noseShift) * sizeY / 2) / 100,

noseSizeX * sizeX / 200, noseSizeY *
sizeY / 200);
        }

        private void drawMounth()
        {
            int W = pictureBox1.Width;
            int H = pictureBox1.Height;
            int posX = currentFace.posX;
            int posY = currentFace.posY;
            int sizeX = currentFace.sizeX;
            int sizeY = currentFace.sizeY;

            if (currentFace.hindrance == 1)
            {
                posX =
rand.Next(positionMinX,
positionMaxX);
                posY =
rand.Next(positionMinY,
positionMaxY);
            }

            int mounthSizeX =
rand.Next(mounthSizeMinX,
mounthSizeMaxX);
            int mounthSizeY =
rand.Next(mounthSizeMinY,
mounthSizeMaxY);
            int mounthShift =
rand.Next(mounthShiftMin,
mounthShiftMax);

            currentFace.mounthSizeX =
mounthSizeX;
            currentFace.mounthSizeY =
mounthSizeY;
            currentFace.mounthShift =
mounthShift;

            g.DrawEllipse(Pens.Black, W *
posX / 100 - W / 2 - (mounthSizeX / 2
* sizeX) / 100,
                                        H * posY
/ 100 - H / 2 - ((mounthSizeY / 2 -
mounthShift) * sizeY / 2) / 100,

mounthSizeX * sizeX / 100,
mounthSizeY * sizeY / 200);
        }

        private void
drawCounterFaceAndEars()
        {
            int W = pictureBox1.Width;
            int H = pictureBox1.Height;
            int posX = currentFace.posX;
            int posY = currentFace.posY;
            int sizeX = currentFace.sizeX;
            int sizeY = currentFace.sizeY;

            if (currentFace.hindrance == 1)
            {
                posX =
rand.Next(positionMinX,
positionMaxX);
                posY =
rand.Next(positionMinY,
positionMaxY);
            }

            currentFace.topDistortion =
```

```
rand.Next(topDistortionMin,
topDistortionMax);
        currentFace.bottomDistortion =
rand.Next(bottomDistortionMin,
bottomDistortionMax);

        float topDistortion =
currentFace.topDistortion / 100f;
        float bottomDistortion =
currentFace.bottomDistortion / 100f;

        Point f1 = new Point(W * posX
/ 100 - W / 2, H * posY / 100 - sizeY /
2 - H / 2),
              f2 = new Point(W * posX /
100 + sizeX / 2 - W / 2, H * posY / 100
- H / 2),
              f3 = new Point(W * posX /
100 - W / 2, H * posY / 100 + sizeY / 2
- H / 2),
              f4 = new Point(W * posX /
100 - sizeX / 2 - W / 2, H * posY / 100
- H / 2),
              g1 = new Point((int)((f1.X
+ f2.X) / 2 + topDistortion * ((f2.X -
f1.X) / 2)),
                 (int)((f1.Y + f2.Y) / 2 +
topDistortion * ((f1.Y - f2.Y) / 2))),
              g2 = new Point((int)((f1.X
+ f4.X) / 2 + topDistortion * ((f4.X -
f1.X) / 2)),
                 (int)((f1.Y + f4.Y) / 2 +
topDistortion * ((f1.Y - f4.Y) / 2))),
              g3 = new Point((int)((f3.X
+ f2.X) / 2 + bottomDistortion * ((f2.X
- f3.X) / 2)),
                 (int)((f3.Y + f2.Y) / 2 +
bottomDistortion * ((f3.Y - f2.Y) / 2))),
              g4 = new Point((int)((f3.X
+ f4.X) / 2 + bottomDistortion * ((f4.X
- f3.X) / 2)),
                 (int)((f3.Y + f4.Y) / 2 +
bottomDistortion * ((f3.Y - f4.Y) / 2)));

        //top right
        Point xy0 = f1;
        for(float t=0; t<=1; t+=0.01f)
        {
           Point xy = new Point((int)((1
- t) * (1 - t) * f1.X + 2 * (1 - t) * t *
g1.X + t * t * f2.X),
              (int)((1 - t) * (1 - t) * f1.Y
+ 2 * (1 - t) * t * g1.Y + t * t * f2.Y));
           g.DrawLine(Pens.Black, xy0,
xy);
           xy0 = xy;
        }

        //top left
        xy0 = f1;
        for (float t = 0; t <= 1; t +=
0.01f)
        {
           Point xy = new Point((int)((1
- t) * (1 - t) * f1.X + 2 * (1 - t) * t *
g2.X + t * t * f4.X),
              (int)((1 - t) * (1 - t) * f1.Y
+ 2 * (1 - t) * t * g2.Y + t * t * f4.Y));
           g.DrawLine(Pens.Black, xy0,
xy);
           xy0 = xy;
        }

        //bottom right
        xy0 = f3;
        for (float t = 0; t <= 1; t +=
0.01f)
        {
           Point xy = new Point((int)((1
- t) * (1 - t) * f3.X + 2 * (1 - t) * t *
g3.X + t * t * f2.X),
              (int)((1 - t) * (1 - t) * f3.Y
```

```
+ 2 * (1 - t) * t * g3.Y + t * t * f2.Y));
            g.DrawLine(Pens.Black, xy0,
xy);
            xy0 = xy;
         }

         //bottom left
         xy0 = f3;
         for (float t = 0; t <= 1; t +=
0.01f)
         {
            Point xy = new Point((int)((1
- t) * (1 - t) * f3.X + 2 * (1 - t) * t *
g4.X + t * t * f4.X),
               (int)((1 - t) * (1 - t) * f3.Y
+ 2 * (1 - t) * t * g4.Y + t * t * f4.Y));
            g.DrawLine(Pens.Black, xy0,
xy);
            xy0 = xy;
         }

         if (currentFace.hindrance == 1)
         {
            f1.X =
rand.Next(positionMinX,
positionMaxX);
            f1.Y =
rand.Next(positionMinY,
positionMaxY);
            f2.X =
rand.Next(positionMinX,
positionMaxX);
            f2.Y =
rand.Next(positionMinY,
positionMaxY);
            f3.X =
rand.Next(positionMinX,
positionMaxX);
            f3.Y =
rand.Next(positionMinY,
positionMaxY);
            f4.X =
rand.Next(positionMinX,
positionMaxX);
            f4.Y =
rand.Next(positionMinY,
positionMaxY);
         }

         //Ears
         int sizeEarsY =
rand.Next(earSizeMinY,
earSizeMaxY);
         int sizeEarsX =
rand.Next(earSizeMinXdependence,
earSizeMaxXdependence);
         int shiftEars =
rand.Next(earShiftMin, earShiftMax);

         currentFace.sizeEarsY =
sizeEarsY;
         currentFace.sizeEarsX =
sizeEarsX;
         currentFace.shiftEars =
shiftEars;

         float p0 = shiftEars / 100.0f;
         float p = (shiftEars +
sizeEarsY) / 100.0f;

         Point fe1 = new Point((int)((1 -
p0) * (1 - p0) * f1.X + 2 * (1 - p0) * p0
* g1.X + p0 * p0 * f2.X),
                  (int)((1 - p0) * (1 -
p0) * f1.Y + 2 * (1 - p0) * p0 * g1.Y +
p0 * p0 * f2.Y)),
            fe2 = new Point((int)((1 -
p) * (1 - p) * f1.X + 2 * (1 - p) * p *
g1.X + p * p * f2.X),
                  (int)((1 - p) * (1 -
p) * f1.Y + 2 * (1 - p) * p * g1.Y + p *
p * f2.Y)),
```

```
            fe3 = new Point((int)((1 -
p0) * (1 - p0) * f1.X + 2 * (1 - p0) * p0
* g2.X + p0 * p0 * f4.X),
                    (int)((1 - p0) * (1 -
p0) * f1.Y + 2 * (1 - p0) * p0 * g2.Y +
p0 * p0 * f4.Y)),
            fe4 = new Point((int)((1 -
p) * (1 - p) * f1.X + 2 * (1 - p) * p *
g2.X + p * p * f4.X),
                    (int)((1 - p) * (1 -
p) * f1.Y + 2 * (1 - p) * p * g2.Y + p *
p * f4.Y)),
            ge1 = new Point(fe1.X +
sizeEarsX * (fe2.Y - fe1.Y) / 100,
                    fe1.Y -
sizeEarsX * (fe2.X - fe1.X) / 80),
            ge2 = new Point(fe2.X +
sizeEarsX * (fe2.Y - fe1.Y) / 100,
                    fe2.Y -
sizeEarsX * (fe2.X - fe1.X) / 80),
            ge3 = new Point(fe3.X -
sizeEarsX * (fe4.Y - fe3.Y) / 100,
                    fe3.Y +
sizeEarsX * (fe4.X - fe3.X) / 80),
            ge4 = new Point(fe4.X -
sizeEarsX * (fe4.Y - fe3.Y) / 100,
                    fe4.Y +
sizeEarsX * (fe4.X - fe3.X) / 80);


        //right ear
        xy0 = fe1;
        for (float t = 0; t <= 1; t +=
0.01f)
        {
            Point xy = new
Point((int)(Math.Pow((1 - t), 3) * fe1.X
+ 3 * Math.Pow((1 - t), 2) * t * ge1.X
                    + 3 * (1 - t)
* t * t * ge2.X + t * t * t * fe2.X),
(int)(Math.Pow((1 - t), 3) * fe1.Y + 3 *
Math.Pow((1 - t), 2) * t * ge1.Y
                    + 3 * (1 - t)
* t * t * ge2.Y + t * t * t * fe2.Y));
            g.DrawLine(Pens.Black, xy0,
xy);
            xy0 = xy;
        }

        //left ear
        xy0 = fe3;
        for (float t = 0; t <= 1; t +=
0.01f)
        {
            Point xy = new
Point((int)(Math.Pow((1 - t), 3) * fe3.X
+ 3 * Math.Pow((1 - t), 2) * t * ge3.X
                    + 3 * (1 - t)
* t * t * ge4.X + t * t * t * fe4.X),

(int)(Math.Pow((1 - t), 3) * fe3.Y + 3 *
Math.Pow((1 - t), 2) * t * ge3.Y
                    + 3 * (1 - t)
* t * t * ge4.Y + t * t * t * fe4.Y));
            g.DrawLine(Pens.Black, xy0,
xy);
            xy0 = xy;
        }
    }

    private void button1_Click(object
sender, EventArgs e)
    {
        pictureBox1.Refresh();
        pictureBox1.Invalidate();
    }

    private void
pictureBox1_Paint(object sender,
PaintEventArgs e)
    {
```

```
            g.Clear(Color.White);
            int angle = rand.Next(angleMin,
angleMax);
            int posX =
rand.Next(positionMinX,
positionMaxX);
            int posY =
rand.Next(positionMinY,
positionMaxY);
            int sizeX =
rand.Next(sizeMinX, sizeMaxX);
            int sizeY =
rand.Next(sizeMinYindependence,
sizeMaxYindependence) * sizeX / 100
+ sizeX;

            currentFace.angle = angle;
            currentFace.posX = posX;
            currentFace.posY = posY;
            currentFace.sizeX = sizeX;
            currentFace.sizeY = sizeY;
            if(hindrance)
currentFace.hindrance = 1;
            else currentFace.hindrance = 0;


g.TranslateTransform(pictureBox1.Wi
dth / 2, pictureBox1.Height / 2);
            g.RotateTransform(angle);

            drawCounterFaceAndEars();
            drawEyes();
            drawNose();
            drawMounth();

            g.RotateTransform(-angle);
            g.TranslateTransform(-
pictureBox1.Width / 2, -
pictureBox1.Height / 2);
            if(image != null)
image.Dispose();
            image = new
Bitmap(pictureBox1.Image, new
Size(pictureBox2.Width,
pictureBox2.Height));
            pictureBox2.Image = image;
            pictureBox2.Refresh();
        }

        private void button3_Click(object
sender, EventArgs e)
        {
            FileInfo configFile = new
FileInfo(path + "config.txt");

            if (configFile.Exists)
            {
                configFile.Delete();
            }

            using (StreamWriter
streamWriter =
configFile.CreateText())
            {
                int number =
Int32.Parse(numericUpDown1.Value.T
oString());
                for (int i = 0; i < number;
i++)
                {
                    pictureBox1.Refresh();
                    pictureBox1.Invalidate();

                    string saveTxt = i + ": " +
currentFace.hindrance + " " +
currentFace.angle + " " +
currentFace.posX + " "
                        + currentFace.posY + "
" + currentFace.sizeX + " " +
currentFace.sizeY + " "
                        +
currentFace.topDistortion + " " +
```

```
currentFace.bottomDistortion + " " +
currentFace.sizeEarsY + " "
                    + currentFace.sizeEarsX
+ " " + currentFace.shiftEars + " " +
currentFace.eyeSizeX + " "
                    + currentFace.eyeSizeY
+ " " + currentFace.eyeShiftX + " " +
currentFace.eyeShiftY + " "
                    +
currentFace.noseSizeX + " " +
currentFace.noseSizeY + " " +
currentFace.noseShift + " "
                    +
currentFace.mounthSizeX + " " +
currentFace.mounthSizeY + " " +
currentFace.mounthShift + " \r\n";


streamWriter.Write(saveTxt);

pictureBox1.Image.Save(path + i +
".png");
                }
            }
        }

        private void
SettingsButton_Click(object sender,
EventArgs e)
        {
            if (settings != null)
settings.Close();
            settings = new Settings(this);
            settings.Show();
        }

        private void setSettingToZero()
        {
            positionMinX = positionMaxX
= positionMinY = positionMaxY = 50;
            angleMin = angleMax = 0;
            sizeMaxX = sizeMinX = 35;
            sizeMinYindependence =
sizeMaxYindependence = 25;
            topDistortionMin =
topDistortionMax = 100;
            bottomDistortionMin =
bottomDistortionMax = 75;
            earSizeMinY = earSizeMaxY =
27;
            earSizeMinXdependence =
earSizeMaxXdependence = 80;
            earShiftMin = earShiftMax =
55;
            eyeSizeMinX = eyeSizeMaxX
= 35;
            eyeSizeMinY = eyeSizeMaxY
= 28;
            eyeShiftMinX = eyeShiftMaxX
= 28;
            eyeShiftMinY = eyeShiftMaxY
= 30;
            noseSizeMinX =
noseSizeMaxX = 22;
            noseSizeMinY =
noseSizeMaxY = 28;
            noseShiftMin = noseShiftMax
= 3;
            mounthSizeMinX =
mounthSizeMaxX = 48;
            mounthSizeMinY =
mounthSizeMaxY = 22;
            mounthShiftMin =
mounthShiftMax = 48;
        }

        private void
pictureBox2_DoubleClick(object
sender, EventArgs e)
        {
            setSettingToZero();
            pictureBox1.Refresh();
```

```
        }
    }
}
```

Settings.cs:

```
using System;
using System.Windows.Forms;

namespace faceConturGenerator
{
    public partial class Settings : Form
    {
        private MainForm mainForm;
        public Settings(MainForm form1)
        {
            InitializeComponent();
            mainForm = form1;

            if (mainForm.hindrance)
trackBar1.Value = 1;
            else trackBar1.Value = 0;
            trackBar2.Value =
mainForm.angleMin;
            trackBar7.Value =
mainForm.angleMax;
            trackBar3.Value =
mainForm.positionMinX;
            trackBar4.Value =
mainForm.positionMaxX;
            trackBar6.Value =
mainForm.positionMinY;
            trackBar5.Value =
mainForm.positionMaxY;

            trackBar14.Value =
mainForm.sizeMinX;
            trackBar15.Value =
mainForm.sizeMaxX;
            trackBar13.Value =
mainForm.sizeMinYindependence;
            trackBar12.Value =
mainForm.sizeMaxYindependence;
            trackBar10.Value =
mainForm.topDistortionMin;
            trackBar11.Value =
mainForm.topDistortionMax;
            trackBar8.Value =
mainForm.bottomDistortionMin;
            trackBar9.Value =
mainForm.bottomDistortionMax;

            trackBar23.Value =
mainForm.earSizeMinXdependence;
            trackBar22.Value =
mainForm.earSizeMaxXdependence;
            trackBar21.Value =
mainForm.earSizeMinY;
            trackBar20.Value =
mainForm.earSizeMaxY;
            trackBar17.Value =
mainForm.earShiftMin;
            trackBar16.Value =
mainForm.earShiftMax;

            trackBar29.Value =
mainForm.eyeSizeMinX;
            trackBar28.Value =
mainForm.eyeSizeMaxX;
            trackBar27.Value =
mainForm.eyeSizeMinY;
            trackBar26.Value =
mainForm.eyeSizeMaxY;
            trackBar24.Value =
mainForm.eyeShiftMinX;
            trackBar25.Value =
mainForm.eyeShiftMaxX;
            trackBar18.Value =
mainForm.eyeShiftMinY;
            trackBar19.Value =
mainForm.eyeShiftMaxY;

            trackBar33.Value =
mainForm.noseSizeMinX;
```

```
            trackBar32.Value =
mainForm.noseSizeMaxX;
            trackBar31.Value =
mainForm.noseSizeMinY;
            trackBar30.Value =
mainForm.noseSizeMaxY;
            trackBar35.Value =
mainForm.noseShiftMin;
            trackBar34.Value =
mainForm.noseShiftMax;

            trackBar39.Value =
mainForm.mounthSizeMinX;
            trackBar38.Value =
mainForm.mounthSizeMaxX;
            trackBar37.Value =
mainForm.mounthSizeMinY;
            trackBar36.Value =
mainForm.mounthSizeMaxY;
            trackBar41.Value =
mainForm.mounthShiftMin;
            trackBar40.Value =
mainForm.mounthShiftMax;
        }

        private void
trackBar1_Scroll(object sender,
EventArgs e)
        {
            if (trackBar1.Value == 0)
mainForm.hindrance = false;
            else mainForm.hindrance =
true;
        }

        private void
trackBar2_Scroll(object sender,
EventArgs e)
        {
            mainForm.angleMin =
trackBar2.Value;
            if (trackBar7.Value <
trackBar2.Value)
            {
                trackBar7.Value =
trackBar2.Value;
                mainForm.angleMax =
trackBar7.Value;
            }
        }

        private void
trackBar3_Scroll(object sender,
EventArgs e)
        {
            mainForm.positionMinX =
trackBar3.Value;
            if (trackBar4.Value <
trackBar3.Value)
            {
                trackBar4.Value =
trackBar3.Value;
                mainForm.positionMaxX =
trackBar4.Value;
            }
        }

        private void
trackBar4_Scroll(object sender,
EventArgs e)
        {
            mainForm.positionMaxX =
trackBar4.Value;
            if (trackBar3.Value >
trackBar4.Value)
            {
                trackBar3.Value =
trackBar4.Value;
                mainForm.positionMinX =
trackBar3.Value;
            }
        }
```

```
        private void
trackBar6_Scroll(object sender,
EventArgs e)
        {
            mainForm.positionMinY =
trackBar6.Value;
            if (trackBar5.Value <
trackBar6.Value)
            {
                trackBar5.Value =
trackBar6.Value;
                mainForm.positionMaxY =
trackBar5.Value;
            }
        }

        private void
trackBar5_Scroll(object sender,
EventArgs e)
        {
            mainForm.positionMaxY =
trackBar5.Value;
            if (trackBar6.Value >
trackBar5.Value)
            {
                trackBar6.Value =
trackBar5.Value;
                mainForm.positionMinY =
trackBar6.Value;
            }
        }

        private void
trackBar7_Scroll(object sender,
EventArgs e)
        {
            mainForm.angleMax =
trackBar7.Value;
            if (trackBar7.Value <
trackBar2.Value)
            {
                trackBar2.Value =
trackBar7.Value;
                mainForm.angleMin =
trackBar2.Value;
            }
        }

        private void
trackBar14_Scroll(object sender,
EventArgs e)
        {
            mainForm.sizeMinX =
trackBar14.Value;
            if (trackBar15.Value <
trackBar14.Value)
            {
                trackBar15.Value =
trackBar14.Value;
                mainForm.sizeMaxX =
trackBar15.Value;
            }
        }

        private void
trackBar15_Scroll(object sender,
EventArgs e)
        {
            mainForm.sizeMaxX =
trackBar15.Value;
            if (trackBar15.Value <
trackBar14.Value)
            {
                trackBar14.Value =
trackBar15.Value;
                mainForm.sizeMinX =
trackBar14.Value;
            }
        }

        private void
```

```
trackBar13_Scroll(object sender,
EventArgs e)
      {

mainForm.sizeMinYindependence =
trackBar13.Value;
         if (trackBar12.Value <
trackBar13.Value)
         {
            trackBar12.Value =
trackBar13.Value;

mainForm.sizeMaxYindependence=
trackBar12.Value;
         }
      }

      private void
trackBar12_Scroll(object sender,
EventArgs e)
      {

mainForm.sizeMaxYindependence =
trackBar12.Value;
         if (trackBar12.Value <
trackBar13.Value)
         {
            trackBar13.Value =
trackBar12.Value;

mainForm.sizeMinYindependence =
trackBar13.Value;
         }
      }

      private void
trackBar10_Scroll(object sender,
EventArgs e)
      {
         mainForm.topDistortionMin =
trackBar10.Value;
         if (trackBar11.Value <
trackBar10.Value)
         {
            trackBar11.Value =
trackBar10.Value;
            mainForm.topDistortionMax
= trackBar11.Value;
         }
      }

      private void
trackBar11_Scroll(object sender,
EventArgs e)
      {
         mainForm.topDistortionMax =
trackBar11.Value;
         if (trackBar11.Value <
trackBar10.Value)
         {
            trackBar10.Value =
trackBar11.Value;
            mainForm.topDistortionMin
= trackBar10.Value;
         }
      }

      private void
trackBar8_Scroll(object sender,
EventArgs e)
      {

mainForm.bottomDistortionMin =
trackBar8.Value;
         if (trackBar9.Value <
trackBar8.Value)
         {
            trackBar9.Value =
trackBar8.Value;

mainForm.bottomDistortionMax =
trackBar9.Value;
```

```
        }
    }

    private void
trackBar9_Scroll(object sender,
EventArgs e)
    {

mainForm.bottomDistortionMax =
trackBar9.Value;
        if (trackBar9.Value <
trackBar8.Value)
        {
            trackBar8.Value =
trackBar9.Value;

mainForm.bottomDistortionMin =
trackBar8.Value;
        }
    }

    private void
trackBar23_Scroll(object sender,
EventArgs e)
    {

mainForm.earSizeMinXdependence =
trackBar23.Value;
        if (trackBar22.Value <
trackBar23.Value)
        {
            trackBar22.Value =
trackBar23.Value;

mainForm.earSizeMaxXdependence =
trackBar22.Value;
        }
    }

    private void
trackBar22_Scroll(object sender,
EventArgs e)
    {

mainForm.earSizeMaxXdependence =
trackBar22.Value;
        if (trackBar22.Value <
trackBar23.Value)
        {
            trackBar23.Value =
trackBar22.Value;

mainForm.earSizeMinXdependence =
trackBar23.Value;
        }
    }

    private void
trackBar21_Scroll(object sender,
EventArgs e)
    {
        mainForm.earSizeMinY =
trackBar21.Value;
        if (trackBar20.Value <
trackBar21.Value)
        {
            trackBar20.Value =
trackBar21.Value;
            mainForm.earSizeMaxY =
trackBar20.Value;
        }
    }

    private void
trackBar20_Scroll(object sender,
EventArgs e)
    {
        mainForm.earSizeMaxY =
trackBar20.Value;
        if (trackBar20.Value <
trackBar21.Value)
        {
```

```
            trackBar21.Value =
trackBar20.Value;
            mainForm.earSizeMinY =
trackBar21.Value;
        }
    }

    private void
trackBar17_Scroll(object sender,
EventArgs e)
    {
        mainForm.earShiftMin =
trackBar17.Value;
        if (trackBar16.Value <
trackBar17.Value)
        {
            trackBar16.Value =
trackBar17.Value;
            mainForm.earShiftMax =
trackBar16.Value;
        }
    }

    private void
trackBar16_Scroll(object sender,
EventArgs e)
    {
        mainForm.earShiftMax =
trackBar16.Value;
        if (trackBar16.Value <
trackBar17.Value)
        {
            trackBar17.Value =
trackBar16.Value;
            mainForm.earShiftMin =
trackBar17.Value;
        }
    }

    private void
trackBar29_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeSizeMinX =
trackBar29.Value;
        if (trackBar28.Value <
trackBar29.Value)
        {
            trackBar28.Value =
trackBar29.Value;
            mainForm.eyeSizeMaxX =
trackBar28.Value;
        }
    }

    private void
trackBar28_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeSizeMaxX =
trackBar28.Value;
        if (trackBar28.Value <
trackBar29.Value)
        {
            trackBar29.Value =
trackBar28.Value;
            mainForm.eyeSizeMinX =
trackBar29.Value;
        }
    }

    private void
trackBar27_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeSizeMinY =
trackBar27.Value;
        if (trackBar26.Value <
trackBar27.Value)
        {
            trackBar26.Value =
trackBar27.Value;
```

```
            mainForm.eyeSizeMaxY =
trackBar26.Value;
        }
    }

    private void
trackBar26_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeSizeMaxY =
trackBar26.Value;
        if (trackBar26.Value <
trackBar27.Value)
        {
            trackBar27.Value =
trackBar26.Value;
            mainForm.eyeSizeMinY =
trackBar27.Value;
        }
    }

    private void
trackBar24_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeShiftMinX =
trackBar24.Value;
        if (trackBar25.Value <
trackBar24.Value)
        {
            trackBar25.Value =
trackBar24.Value;
            mainForm.eyeShiftMaxX =
trackBar25.Value;
        }
    }

    private void
trackBar25_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeShiftMaxX =
trackBar25.Value;
        if (trackBar25.Value <
trackBar24.Value)
        {
            trackBar24.Value =
trackBar25.Value;
            mainForm.eyeShiftMinX =
trackBar24.Value;
        }
    }

    private void
trackBar18_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeShiftMinY =
trackBar18.Value;
        if (trackBar19.Value <
trackBar18.Value)
        {
            trackBar19.Value =
trackBar18.Value;
            mainForm.eyeShiftMaxY =
trackBar19.Value;
        }
    }

    private void
trackBar19_Scroll(object sender,
EventArgs e)
    {
        mainForm.eyeShiftMaxY =
trackBar19.Value;
        if (trackBar19.Value <
trackBar18.Value)
        {
            trackBar18.Value =
trackBar19.Value;
            mainForm.eyeShiftMinY =
trackBar18.Value;
```

```
        }
    }

    private void
trackBar33_Scroll(object sender,
EventArgs e)
    {
        mainForm.noseSizeMinX =
trackBar33.Value;
        if (trackBar32.Value <
trackBar33.Value)
        {
            trackBar32.Value =
trackBar33.Value;
            mainForm.noseSizeMaxX =
trackBar32.Value;
        }
    }

    private void
trackBar32_Scroll(object sender,
EventArgs e)
    {
        mainForm.noseSizeMaxX =
trackBar32.Value;
        if (trackBar32.Value <
trackBar33.Value)
        {
            trackBar33.Value =
trackBar32.Value;
            mainForm.noseSizeMinX =
trackBar33.Value;
        }
    }

    private void
trackBar31_Scroll(object sender,
EventArgs e)
    {
        mainForm.noseSizeMinY =
trackBar31.Value;
        if (trackBar30.Value <
trackBar31.Value)
        {
            trackBar30.Value =
trackBar31.Value;
            mainForm.noseSizeMaxY =
trackBar30.Value;
        }
    }

    private void
trackBar30_Scroll(object sender,
EventArgs e)
    {
        mainForm.noseSizeMaxY =
trackBar30.Value;
        if (trackBar30.Value <
trackBar31.Value)
        {
            trackBar31.Value =
trackBar30.Value;
            mainForm.noseSizeMinY =
trackBar31.Value;
        }
    }

    private void
trackBar35_Scroll(object sender,
EventArgs e)
    {
        mainForm.noseShiftMin =
trackBar35.Value;
        if (trackBar34.Value <
trackBar35.Value)
        {
            trackBar34.Value =
trackBar35.Value;
            mainForm.noseShiftMax =
trackBar34.Value;
        }
    }
```

```
        private void
trackBar34_Scroll(object sender,
EventArgs e)
        {
            mainForm.noseShiftMax =
trackBar34.Value;
            if (trackBar34.Value <
trackBar35.Value)
            {
                trackBar35.Value =
trackBar34.Value;
                mainForm.noseShiftMin =
trackBar35.Value;
            }
        }

        private void
trackBar39_Scroll(object sender,
EventArgs e)
        {
            mainForm.mounthSizeMinX =
trackBar39.Value;
            if (trackBar38.Value <
trackBar39.Value)
            {
                trackBar38.Value =
trackBar39.Value;
                mainForm.mounthSizeMaxX
= trackBar38.Value;
            }
        }

        private void
trackBar38_Scroll(object sender,
EventArgs e)
        {
            mainForm.mounthSizeMaxX =
trackBar38.Value;
            if (trackBar38.Value <
trackBar39.Value)
            {
                trackBar39.Value =
trackBar38.Value;
                mainForm.mounthSizeMinX
= trackBar39.Value;
            }
        }

        private void
trackBar37_Scroll(object sender,
EventArgs e)
        {
            mainForm.mounthSizeMinY =
trackBar37.Value;
            if (trackBar36.Value <
trackBar37.Value)
            {
                trackBar36.Value =
trackBar37.Value;
                mainForm.mounthSizeMaxY
= trackBar36.Value;
            }
        }

        private void
trackBar36_Scroll(object sender,
EventArgs e)
        {
            mainForm.mounthSizeMaxY =
trackBar36.Value;
            if (trackBar36.Value <
trackBar37.Value)
            {
                trackBar37.Value =
trackBar36.Value;
                mainForm.mounthSizeMinY
= trackBar37.Value;
            }
        }

        private void
```

```
trackBar41_Scroll(object sender,
EventArgs e)
      {
         mainForm.mounthShiftMin =
trackBar41.Value;
         if (trackBar40.Value <
trackBar41.Value)
         {
            trackBar40.Value =
trackBar41.Value;
            mainForm.mounthShiftMax
= trackBar40.Value;
         }
      }

      private void
trackBar40_Scroll(object sender,
EventArgs e)
      {
         mainForm.mounthShiftMax =
trackBar40.Value;
         if (trackBar40.Value <
trackBar41.Value)
         {
            trackBar41.Value =
trackBar40.Value;
            mainForm.mounthShiftMin =
trackBar41.Value;
         }
      }
   }
}
```

Програма 2 – сортування облич:

MainForm.cs:

```
using System;
using System.Windows.Forms;
using System.IO;

namespace FaceSorts
{
   public partial class MainForm :
Form
   {
      private struct FaceConfig
      {
         public int indexOfPicture { set;
get; }
         public int hindrance { set; get; }
         public int angle { set; get; }
         public int posX { set; get; }
         public int posY { set; get; }
         public int sizeX { set; get; }
         public int sizeY { set; get; }
         public int topDistortion { set;
get; }
         public int bottomDistortion {
set; get; }
         public int sizeEarsY { set; get;
}
         public int sizeEarsX { set; get;
}
         public int shiftEars { set; get; }
         public int eyeSizeX { set; get; }
         public int eyeSizeY { set; get; }
         public int eyeShiftX { set; get;
}
         public int eyeShiftY { set; get;
}
         public int noseSizeX { set; get;
}
         public int noseSizeY { set; get;
}
         public int noseShift { set; get; }
         public int mounthSizeX { set;
get; }
         public int mounthSizeY { set;
get; }
         public int mounthShift { set;
get; }
      }
```

```
        string pathConfig = "";
        string pathOutput = "";

        OpenFileDialog
openFileDialogConfig;
        OpenFileDialog
openFileDialogOutput;

        public MainForm()
        {
            InitializeComponent();
            openFileDialogConfig = new
OpenFileDialog();
            openFileDialogOutput = new
OpenFileDialog();
            openFileDialogConfig.Filter =
"txt files (*.txt)|*.txt";

openFileDialogConfig.FilterIndex = 2;
            openFileDialogOutput.Filter =
"txt files (*.txt)|*.txt";

openFileDialogOutput.FilterIndex = 2;
            comboBox1.SelectedIndex = 0;
            comboBox2.SelectedIndex = 0;
            comboBox3.SelectedIndex = 0;
        }

        private void
decomposeElements(ref FaceConfig[]
faceConfigs, int n, string configText)
        {
            int index = 0;
            for (int i = 0; i < n; i++)
            {
                int nextIndex =
configText.IndexOf(':', index);
                string element =
configText.Substring(index, nextIndex
- index);
faceConfigs[i].indexOfPicture =
Convert.ToInt32(element);
                index = nextIndex + 2;

                nextIndex =
configText.IndexOf(' ', index);
                element =
configText.Substring(index, nextIndex
- index);
                faceConfigs[i].hindrance =
Convert.ToInt32(element);
                index = nextIndex + 1;

                nextIndex =
configText.IndexOf(' ', index);
                element =
configText.Substring(index, nextIndex
- index);
                faceConfigs[i].angle =
Convert.ToInt32(element);
                index = nextIndex + 1;

                nextIndex =
configText.IndexOf(' ', index);
                element =
configText.Substring(index, nextIndex
- index);
                faceConfigs[i].posX =
Convert.ToInt32(element);
                index = nextIndex + 1;

                nextIndex =
configText.IndexOf(' ', index);
                element =
configText.Substring(index, nextIndex
- index);
                faceConfigs[i].posY =
Convert.ToInt32(element);
                index = nextIndex + 1;

                nextIndex =
```

```
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].sizeX =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].sizeY =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].topDistortion
= Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);

faceConfigs[i].bottomDistortion =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].sizeEarsY =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].sizeEarsX =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].shiftEars =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].eyeSizeX =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].eyeSizeY =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
```

```
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].eyeShiftX =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].eyeShiftY =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].noseSizeX =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].noseSizeY =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].noseShift =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].mounthSizeX
= Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].mounthSizeY
= Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf('\r', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].mounthShift =
Convert.ToInt32(element);
            index = nextIndex + 2;
        }
    }

    private void
swapConfigElements(ref FaceConfig[]
faceConfigs, int element1, int
element2)
    {
        int temp =
faceConfigs[element1].indexOfPicture;

faceConfigs[element1].indexOfPicture
```

```
=
faceConfigs[element2].indexOfPicture;

faceConfigs[element2].indexOfPicture
= temp;

        temp =
faceConfigs[element1].hindrance;

faceConfigs[element1].hindrance =
faceConfigs[element2].hindrance;

faceConfigs[element2].hindrance =
temp;

        temp =
faceConfigs[element1].angle;
        faceConfigs[element1].angle =
faceConfigs[element2].angle;
        faceConfigs[element2].angle =
temp;

        temp =
faceConfigs[element1].posX;
        faceConfigs[element1].posX =
faceConfigs[element2].posX;
        faceConfigs[element2].posX =
temp;

        temp =
faceConfigs[element1].posY;
        faceConfigs[element1].posY =
faceConfigs[element2].posY;
        faceConfigs[element2].posY =
temp;

        temp =
faceConfigs[element1].sizeX;
        faceConfigs[element1].sizeX =
faceConfigs[element2].sizeX;
        faceConfigs[element2].sizeX =
temp;

        temp =
faceConfigs[element1].sizeY;
        faceConfigs[element1].sizeY =
faceConfigs[element2].sizeY;
        faceConfigs[element2].sizeY =
temp;

        temp =
faceConfigs[element1].topDistortion;

faceConfigs[element1].topDistortion =
faceConfigs[element2].topDistortion;

faceConfigs[element2].topDistortion =
temp;

        temp =
faceConfigs[element1].bottomDistortio
n;

faceConfigs[element1].bottomDistortio
n =
faceConfigs[element2].bottomDistortio
n;

faceConfigs[element2].bottomDistortio
n = temp;

        temp =
faceConfigs[element1].sizeEarsY;

faceConfigs[element1].sizeEarsY =
faceConfigs[element2].sizeEarsY;

faceConfigs[element2].sizeEarsY =
temp;

        temp =
faceConfigs[element1].sizeEarsX;
```

```
faceConfigs[element1].sizeEarsX =
faceConfigs[element2].sizeEarsX;

faceConfigs[element2].sizeEarsX =
temp;

        temp =
faceConfigs[element1].shiftEars;

faceConfigs[element1].shiftEars =
faceConfigs[element2].shiftEars;

faceConfigs[element2].shiftEars =
temp;

        temp =
faceConfigs[element1].eyeSizeX;

faceConfigs[element1].eyeSizeX =
faceConfigs[element2].eyeSizeX;

faceConfigs[element2].eyeSizeX =
temp;

        temp =
faceConfigs[element1].eyeSizeY;

faceConfigs[element1].eyeSizeY =
faceConfigs[element2].eyeSizeY;

faceConfigs[element2].eyeSizeY =
temp;

        temp =
faceConfigs[element1].eyeShiftX;

faceConfigs[element1].eyeShiftX =
faceConfigs[element2].eyeShiftX;

faceConfigs[element2].eyeShiftX =
temp;

        temp =
faceConfigs[element1].eyeShiftY;

faceConfigs[element1].eyeShiftY =
faceConfigs[element2].eyeShiftY;

faceConfigs[element2].eyeShiftY =
temp;

        temp =
faceConfigs[element1].noseSizeX;

faceConfigs[element1].noseSizeX =
faceConfigs[element2].noseSizeX;

faceConfigs[element2].noseSizeX =
temp;

        temp =
faceConfigs[element1].noseSizeY;

faceConfigs[element1].noseSizeY =
faceConfigs[element2].noseSizeY;

faceConfigs[element2].noseSizeY =
temp;

        temp =
faceConfigs[element1].noseShift;

faceConfigs[element1].noseShift =
faceConfigs[element2].noseShift;

faceConfigs[element2].noseShift =
temp;

        temp =
faceConfigs[element1].mounthSizeX;
```

```
faceConfigs[element1].mounthSizeX =
faceConfigs[element2].mounthSizeX;

faceConfigs[element2].mounthSizeX =
temp;

        temp =
faceConfigs[element1].mounthSizeY;

faceConfigs[element1].mounthSizeY =
faceConfigs[element2].mounthSizeY;

faceConfigs[element2].mounthSizeY =
temp;

        temp =
faceConfigs[element1].mounthShift;

faceConfigs[element1].mounthShift =
faceConfigs[element2].mounthShift;

faceConfigs[element2].mounthShift =
temp;
      }

      private int getValue(ref
FaceConfig[] faceConfigs, int element,
ref int param)
      {
         switch (param)
         {
            case 0:
               return
faceConfigs[element].hindrance;
            case 1:
               return
faceConfigs[element].angle;
            case 2:
               return
faceConfigs[element].posX;
            case 3:
               return
faceConfigs[element].posY;
            case 4:
               return
faceConfigs[element].sizeX;
            case 5:
               return
faceConfigs[element].sizeY;
            case 6:
               return
faceConfigs[element].topDistortion;
            case 7:
               return
faceConfigs[element].bottomDistortion
;
            case 8:
               return
faceConfigs[element].sizeEarsY;
            case 9:
               return
faceConfigs[element].sizeEarsX;
            case 10:
               return
faceConfigs[element].shiftEars;
            case 11:
               return
faceConfigs[element].eyeSizeX;
            case 12:
               return
faceConfigs[element].eyeSizeY;
            case 13:
               return
faceConfigs[element].eyeShiftX;
            case 14:
               return
faceConfigs[element].eyeShiftY;
            case 15:
               return
faceConfigs[element].noseSizeX;
            case 16:
               return
```

```
faceConfigs[element].noseSizeY;
            case 17:
               return
faceConfigs[element].noseShift;
            case 18:
               return
faceConfigs[element].mounthSizeX;
            case 19:
               return
faceConfigs[element].mounthSizeY;
            case 20:
               return
faceConfigs[element].mounthShift;
         }
         return 0;
      }

      private void
randomDistribution(ref FaceConfig[]
faceConfigs, int n)
      {
         Random random = new
Random();
         for (int i = 0; i < n; i++)
         {
            swapConfigElements(ref
faceConfigs, i, random.Next(0, n));
         }
      }

      private void shellSort(ref
FaceConfig[] faceConfigs, int from, int
count, ref int param)
      {
         int n = count / 2;
         while (n >= 1)
         {
            for (int i = n; i < count; i++)
            {
               int j = i;
               while ((j >= n) &&
(getValue(ref faceConfigs, from + j, ref
param) < getValue(ref faceConfigs,
from + j - n, ref param)))
               {

swapConfigElements(ref faceConfigs,
from + j, from + j - n);
                  j = j - n;
               }
            }
            n = n / 2;
         }
      }

      private void
getSmallerGroupForSorting(ref
FaceConfig[] faceConfigs, int start, int
n, int param, int lastParam1)
      {
         int from = start;
         int lastValue = getValue(ref
faceConfigs, 0, ref lastParam1);
         for (int i = start + 1; i < n; i++)
         {
            if(lastValue != getValue(ref
faceConfigs, i, ref lastParam1))
            {
               if (i - from != 1)
               {
                  if (i - from == 2)
                  {
                     if (getValue(ref
faceConfigs, from, ref param) >
getValue(ref faceConfigs, from + 1, ref
param))
                     {

swapConfigElements(ref faceConfigs,
from, from + 1);
                     }
                  }
```

```
                else
                {
                    shellSort(ref
faceConfigs, from, i - from, ref param);
                }
                if(start ==0 &&
comboBox3.SelectedIndex != 0)
                {

getSmallerGroupForSorting(ref
faceConfigs, from, i,
comboBox3.SelectedIndex - 1, param);
                }
            }
            from = i;
            lastValue = getValue(ref
faceConfigs, i, ref lastParam1);
        }
      }
      if (n - 1 - from != 1)
      {
        if (n - 1 - from == 2)
        {
          if (getValue(ref
faceConfigs, from, ref param) >
getValue(ref faceConfigs, from + 1, ref
param))
          {

swapConfigElements(ref faceConfigs,
from, from + 1);
          }
        }
        else
        {
          shellSort(ref faceConfigs,
from, n - 1 - from, ref param);
        }
        if (start == 0 &&
comboBox3.SelectedIndex != 0)
        {

getSmallerGroupForSorting(ref
faceConfigs, from, n - 1,
comboBox3.SelectedIndex - 1, param);
        }
      }
    }

    private void button1_Click(object
sender, EventArgs e)
    {
      if(pathConfig == "" ||
pathOutput == "")
      {
        MessageBox.Show("Please,
enter pathes");
        return;
      }
      int param;
      int n;
      FaceConfig[] faceConfigs;

      using (StreamReader
streamReader = new
StreamReader(openFileDialogConfig.
OpenFile()))
      {
        string configText =
streamReader.ReadToEnd();
        n =
(int)numericUpDown1.Value;
        faceConfigs = new
FaceConfig[n];
        decomposeElements(ref
faceConfigs, n, configText);
        streamReader.Close();
      }

      param =
comboBox1.SelectedIndex;
      if (param != 21)
```

```
        {
            shellSort(ref faceConfigs, 0,
n, ref param);

            if
(comboBox2.SelectedIndex != 0)
            {

getSmallerGroupForSorting(ref
faceConfigs, 0, n,
comboBox2.SelectedIndex - 1, param);
            }
        }
        else
        {
            randomDistribution(ref
faceConfigs, n);
        }

        using (StreamWriter
streamWriter = new
StreamWriter(openFileDialogOutput.Fi
leName, false))
        {
            for (int i = 0; i < n; i++)
            {
                string saveTxt =
faceConfigs[i].indexOfPicture + ": " +
faceConfigs[i].hindrance + " " +
faceConfigs[i].angle + " "
                    + faceConfigs[i].posX +
" " + faceConfigs[i].posY + " " +
faceConfigs[i].sizeX + " " +
faceConfigs[i].sizeY + " "
                    +
faceConfigs[i].topDistortion + " " +
faceConfigs[i].bottomDistortion + " " +
faceConfigs[i].sizeEarsY + " "
                    +
faceConfigs[i].sizeEarsX + " " +
faceConfigs[i].shiftEars + " " +
faceConfigs[i].eyeSizeX + " "
                    +
faceConfigs[i].eyeSizeY + " " +
faceConfigs[i].eyeShiftX + " " +
faceConfigs[i].eyeShiftY + " "
                    +
faceConfigs[i].noseSizeX + " " +
faceConfigs[i].noseSizeY + " " +
faceConfigs[i].noseShift + " "
                    +
faceConfigs[i].mounthSizeX + " " +
faceConfigs[i].mounthSizeY + " " +
faceConfigs[i].mounthShift + " \r\n";


streamWriter.Write(saveTxt);
            }
            streamWriter.Close();
        }
        MessageBox.Show("Sort
Ready!");
    }

    private void button3_Click(object
sender, EventArgs e)
    {

openFileDialogConfig.ShowDialog();
        pathConfig =
openFileDialogConfig.FileName;
        button3.Text = pathConfig;
    }

    private void button2_Click(object
sender, EventArgs e)
    {

openFileDialogOutput.ShowDialog();
        pathOutput =
openFileDialogOutput.FileName;
        button2.Text = pathOutput;
```

```
        }

        private void
comboBox1_SelectedIndexChanged(o
bject sender, EventArgs e)
        {
            if(comboBox1.SelectedIndex
== 21)
            {
                comboBox2.Enabled = false;
                comboBox3.Enabled = false;
            }
            else
            {
                comboBox2.Enabled = true;
                comboBox3.Enabled = true;
            }
        }
    }
}
```

Програма 2 – нейронна мережа:

MainForm:

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.IO;
using System.Threading;

namespace FaceNeuralNetwork
{
    public partial class MainForm :
Form
    {
        private struct FaceConfig
        {
            public int indexOfPicture { set;
get; }
            public int hindrance { set; get; }
            public int posX { set; get; }
            public int posY { set; get; }
        }

        private double lastNumber = 0;
        private double lastX = 0;
        private double lastY = 0;
        private double lastDeltaHindrance
= 0;
        private double
lastHindranceAccuracy = 0;
        private int testAfter = 250;

        public double[,,]
convolutionMatrix = new double[32, 3,
3];
        public int convolutionNumber =
16;

        private const int pictureSize =
100;
        private const int
downsamplingMultiplier = 2;
        private const int
firstNeuronLayerSize = (pictureSize /
downsamplingMultiplier) *
(pictureSize /
downsamplingMultiplier);
        private const int
secondNeuronLayerSize = 400;

        public double dStep = 0.02;
        public int n = 100;
        public int nTest = 100;
        public FolderBrowserDialog
folderBrowserDialog = new
FolderBrowserDialog();
        public FolderBrowserDialog
folderBrowserDialogTest = new
FolderBrowserDialog();
        public OpenFileDialog
openFileDialogConfig = new
OpenFileDialog();
```

```
        public OpenFileDialog
openFileDialogTest = new
OpenFileDialog();
        public SaveFileDialog
saveFileDialogSaveLoad = new
SaveFileDialog();
        public bool settingsSelected =
false;

        private void
setConvolutionMatrix()
        {
            // 0    1   0
            // 1    1   0
            // 0    1   0
            convolutionMatrix[0, 0, 1] = 1;
            convolutionMatrix[0, 1, 0] = 1;
            convolutionMatrix[0, 1, 1] = 1;
            convolutionMatrix[0, 2, 1] = 1;

            // 0    1   0
            // 1    1   1
            // 0    0   1
            convolutionMatrix[1, 0, 1] = 1;
            convolutionMatrix[1, 1, 0] = 1;
            convolutionMatrix[1, 1, 1] = 1;
            convolutionMatrix[1, 1, 2] = 1;
            convolutionMatrix[1, 2, 2] = 1;

            // 1    0   1
            // 1    1   1
            // 0    1   0
            convolutionMatrix[2, 0, 0] = 1;
            convolutionMatrix[2, 0, 2] = 1;
            convolutionMatrix[2, 1, 0] = 1;
            convolutionMatrix[2, 1, 1] = 1;
            convolutionMatrix[2, 1, 2] = 1;
            convolutionMatrix[2, 2, 1] = 1;

            // 1    0   1
            // 1    1   0
            // 1    1   0
            convolutionMatrix[3, 0, 0] = 1;
            convolutionMatrix[3, 0, 2] = 1;
            convolutionMatrix[3, 1, 0] = 1;
            convolutionMatrix[3, 1, 1] = 1;
            convolutionMatrix[3, 2, 0] = 1;
            convolutionMatrix[3, 2, 1] = 1;

            // 0    1   0
            // 0    1   1
            // 0    1   0
            convolutionMatrix[4, 0, 1] = 1;
            convolutionMatrix[4, 1, 2] = 1;
            convolutionMatrix[4, 1, 1] = 1;
            convolutionMatrix[4, 2, 1] = 1;

            // 0    1   0
            // 1    1   1
            // 1    0   0
            convolutionMatrix[5, 0, 1] = 1;
            convolutionMatrix[5, 1, 0] = 1;
            convolutionMatrix[5, 1, 1] = 1;
            convolutionMatrix[5, 1, 2] = 1;
            convolutionMatrix[5, 2, 0] = 1;

            // 1    1   1
            // 0    1   0
            // 0    0   1
            convolutionMatrix[6, 0, 0] = 1;
            convolutionMatrix[6, 0, 1] = 1;
            convolutionMatrix[6, 0, 2] = 1;
            convolutionMatrix[6, 1, 1] = 1;
            convolutionMatrix[6, 2, 2] = 1;

            // 1    0   1
            // 0    1   1
            // 0    1   1
            convolutionMatrix[7, 0, 0] = 1;
            convolutionMatrix[7, 0, 2] = 1;
            convolutionMatrix[7, 1, 2] = 1;
            convolutionMatrix[7, 1, 1] = 1;
```

```
        convolutionMatrix[7, 2, 2] = 1;
        convolutionMatrix[7, 2, 1] = 1;

        // 1    0   1
        // 1    1   0
        // 0    1   0
        convolutionMatrix[8, 0, 0] = 1;
        convolutionMatrix[8, 0, 2] = 1;
        convolutionMatrix[8, 1, 0] = 1;
        convolutionMatrix[8, 1, 1] = 1;
        convolutionMatrix[8, 2, 1] = 1;

        // 1    1   1
        // 1    1   0
        // 0    1   0
        convolutionMatrix[9, 0, 0] = 1;
        convolutionMatrix[9, 0, 1] = 1;
        convolutionMatrix[9, 0, 2] = 1;
        convolutionMatrix[9, 1, 0] = 1;
        convolutionMatrix[9, 1, 1] = 1;
        convolutionMatrix[9, 2, 1] = 1;

        // 1    1   1
        // 1    0   0
        // 0    1   0
        convolutionMatrix[10, 0, 0] =
1;
        convolutionMatrix[10, 0, 1] =
1;
        convolutionMatrix[10, 0, 2] =
1;
        convolutionMatrix[10, 1, 0] =
1;
        convolutionMatrix[10, 2, 1] =
1;

        // 1    1   1
        // 1    1   0
        // 0    0   1
        convolutionMatrix[11, 0, 0] =
1;
        convolutionMatrix[11, 0, 1] =
1;
        convolutionMatrix[11, 0, 2] =
1;
        convolutionMatrix[11, 1, 0] =
1;
        convolutionMatrix[11, 1, 1] =
1;
        convolutionMatrix[11, 2, 2] =
1;

        // 1    1   0
        // 0    1   1
        // 1    1   0
        convolutionMatrix[12, 0, 0] =
1;
        convolutionMatrix[12, 0, 1] =
1;
        convolutionMatrix[12, 1, 1] =
1;
        convolutionMatrix[12, 1, 2] =
1;
        convolutionMatrix[12, 2, 0] =
1;
        convolutionMatrix[12, 2, 1] =
1;

        // 1    1   1
        // 0    1   1
        // 0    1   0
        convolutionMatrix[13, 0, 0] =
1;
        convolutionMatrix[13, 0, 1] =
1;
        convolutionMatrix[13, 0, 2] =
1;
        convolutionMatrix[13, 1, 2] =
1;
        convolutionMatrix[13, 1, 1] =
1;
        convolutionMatrix[13, 2, 1] =
```

```
1;

            // 1    1   1
            // 0    0   1
            // 0    1   0
            convolutionMatrix[14, 0, 0] =
1;
            convolutionMatrix[14, 0, 1] =
1;
            convolutionMatrix[14, 0, 2] =
1;
            convolutionMatrix[14, 1, 2] =
1;
            convolutionMatrix[14, 2, 1] =
1;

            // 1    1   1
            // 0    1   1
            // 1    0   0
            convolutionMatrix[15, 0, 0] =
1;
            convolutionMatrix[15, 0, 1] =
1;
            convolutionMatrix[15, 0, 2] =
1;
            convolutionMatrix[15, 1, 2] =
1;
            convolutionMatrix[15, 1, 1] =
1;
            convolutionMatrix[15, 2, 0] =
1;
        }

        public MainForm()
        {
            InitializeComponent();
            setConvolutionMatrix();

folderBrowserDialog.SelectedPath =
"E:\\diploma\\New\\";
folderBrowserDialogTest.SelectedPath
= "E:\\diploma\\New\\";
            openFileDialogConfig.Filter =
"txt files (*.txt)|*.txt";

openFileDialogConfig.FilterIndex = 2;
            openFileDialogTest.Filter = "txt
files (*.txt)|*.txt";
            openFileDialogTest.FilterIndex
= 2;
            saveFileDialogSaveLoad.Filter
= "txt files (*.txt)|*.txt";

saveFileDialogSaveLoad.FilterIndex =
2;
        }

        private void
decomposeElements(ref FaceConfig[]
faceConfigs, int n, string configText)
        {
            int index = 0;
            for (int i = 0; i < n; i++)
            {
                int nextIndex =
configText.IndexOf(':', index);
                string element =
configText.Substring(index, nextIndex
- index);

faceConfigs[i].indexOfPicture =
Convert.ToInt32(element);
                index = nextIndex + 2;

                nextIndex =
configText.IndexOf(' ', index);
                element =
configText.Substring(index, nextIndex
- index);
                faceConfigs[i].hindrance =
Convert.ToInt32(element);
```

```
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].posX =
Convert.ToInt32(element);
            index = nextIndex + 1;

            nextIndex =
configText.IndexOf(' ', index);
            element =
configText.Substring(index, nextIndex
- index);
            faceConfigs[i].posY =
Convert.ToInt32(element);

            nextIndex =
configText.IndexOf('\r', index);
            index = nextIndex + 2;
         }
      }

      private double[,]
createMatrtixFromImage(ref Bitmap
image)
      {
         double[,] result = new
double[image.Height, image.Width];
         for (int i = 0; i < image.Height;
i++)
         {
            for (int j = 0; j <
image.Width; j++)
            {
               result[i, j] = 255 -
image.GetPixel(j, i).R;
            }
         }
         return result;
      }

      private double[,]
downsampling(double[,] matrix, int n,
int step)
      {
         int count = n / step;
         double[,] result = new
double[count, count];
         for(int i=0; i<count; i++)
         {
            for(int j =0; j< count; j++)
            {
               double summ = 0;
               for (int k = 0; k < step;
k++)
               {
                  for (int p = 0; p < step;
p++)
                  {
                     summ += matrix[i *
step + k, j * step + p];
                  }
               }
               result[i, j] = summ /
Math.Pow(step, 2);
            }
         }
         return result;
      }

      private double[,]
convolution(double[,] matrix, int n, int
number)
      {
         double[,] result = new double[n,
```

```
n];
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    double summ = 0;
                    for (int k = 0; k < 3; k++)
                    {
                        if (i + k - 1 < 0 || i + k -
1 == n) continue;
                        for (int p = 0; p < 3;
p++)
                        {
                            if (i + p - 1 < 0 || i + p
- 1 == n) continue;
                            summ += matrix[i + k
- 1, i + p - 1] *
convolutionMatrix[number, k, p];
                        }
                    }
                    result[i, j] = summ / 9;
                }
            }
            return result;
        }

        private double[,]
convolutionsUnion(double[][,] matrix,
int n, int number)
        {
            double[,] result = new double[n,
n];
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    double average = 0;
                    for (int p = 0; p < number;
p++)
                    {
                        average += matrix[p][i,
j];
                    }
                    result[i, j] = average /
(number * 60);
                }
            }
            return result;
        }

        private double[]
getNodesFromMatrix(double[,] matrix,
int n)
        {
            double[] result = new double[n
* n];
            for(int i=0; i< n; i++)
            {
                for(int j=0; j<n; j++)
                {
                    result[i * n + j] = matrix[i,
j];
                }
            }
            return result;
        }

        //Sigmoid:
        //1 / (1 + e^-x)
        private double
activationFunc(double x)
        {
            return 1.0 / (1 + Math.Exp(-x));
        }

        private double
activationFuncDx(double fx)
        {
            return fx * (1 - fx);
        }

        private double[]
```

```
getNewNodes(double[] oldNodes,
double[,] nodesMatrix, int n, int newN,
bool bias)
        {
            double[] result;
            if(bias)
            {
                result = new double[newN];
            }
            else
            {
                result = new double[newN +
1];
                result[newN] = 1;
            }

            int rows = n;
            if (bias) rows++;

            for (int i = 0; i < newN; i++)
            {
                for (int j = 0; j < rows; j++)
                {
                    result[i] += oldNodes[j] *
nodesMatrix[i, j];
                }
                result[i] =
activationFunc(result[i]);
            }
            return result;
        }

        private double[,]
generateRandomMatrix(int n, int m, int
p)
        {
            Random rand;
            if (p > 0)
            {
                rand = new Random(p);
            }
            else
            {
                rand = new Random();
            }

            double[,] result = new double[n,
m];

            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    result[i, j] = rand.Next(-
1000, 1000) / 1000.0;
                }
            }
            return result;
        }

        private void
readWeightsFromFile(double[,]
nodesMatrix, double[,] nodesMatrix2,
string text)
        {
            int index = 0;
            for (int i = 0; i <
secondNeuronLayerSize; i++)
            {
                for (int j = 0; j <
firstNeuronLayerSize; j++)
                {
                    int nextIndex =
text.IndexOf(' ', index);
                    string element =
text.Substring(index, nextIndex -
index);
                    nodesMatrix[i,j] =
Convert.ToDouble(element);
                    index = nextIndex + 1;
                }
                int newIndex =
```

```
text.IndexOf('\n', index);
            index = newIndex + 1;
         }

         for (int i = 0; i < 3; i++)
         {
            for (int j = 0; j <
secondNeuronLayerSize + 1; j++)
            {
               int nextIndex =
text.IndexOf(' ', index);
               string element =
text.Substring(index, nextIndex -
index);
               nodesMatrix2[i, j] =
Convert.ToDouble(element);
               index = nextIndex + 1;
            }
            int newIndex =
text.IndexOf('\n', index);
            index = newIndex + 1;
         }
      }

      private void
saveWeightsToFile(double[,]
nodesMatrix, double[,] nodesMatrix2)
      {
         using (StreamWriter
streamWriter = new
StreamWriter(saveFileDialogSaveLoad
.FileName, true))
         {
            for (int i = 0; i <
secondNeuronLayerSize; i++)
            {
               for (int j = 0; j <
firstNeuronLayerSize; j++)
               {

streamWriter.Write(Math.Round(nodes
Matrix[i, j], 4).ToString() + " ");
               }
               streamWriter.Write("
\r\n");
            }

            for (int i = 0; i < 3; i++)
            {
               for (int j = 0; j <
secondNeuronLayerSize + 1; j++)
               {

streamWriter.Write(Math.Round(nodes
Matrix2[i, j], 4).ToString() + " ");
               }
               streamWriter.Write("
\r\n");
            }

            streamWriter.Close();
         }
      }

      private double[]
neuralEducation(double[] errors,
double[] layer, double[] pastLayer,
double[,] nodesMatrix, int n, int m,
bool bias)
      {
         double[] resultLayerErrors =
new double[m];
         double[] weightErrorsDelta =
new double[n];

         for (int i = 0; i < n; i++)
         {
            weightErrorsDelta[i] =
errors[i] * activationFuncDx(layer[i]);
         }

         if (bias) m++;
```

```
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    nodesMatrix[i, j] =
nodesMatrix[i, j] - pastLayer[j] *
weightErrorsDelta[i] * dStep;
                }
            }

            if (bias) m--;
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    resultLayerErrors[j] +=
(nodesMatrix[i, j] *
weightErrorsDelta[i]);
                }
            }

            weightErrorsDelta = null;
            return resultLayerErrors;
        }

        private void
neuralNetworkWork(FaceConfig[]
faceConfigs, FaceConfig[]
faceConfigs2, double[,] nodesMatrix,
double[,] nodesMatrix2)
        {
            Bitmap image;
            double[] errors = new double[3]
{ 0, 0, 0 };
            double[,] workMatrix;
            double[][,] convMatrices;
            double[] layer1;
            double[] layer2;
            double[] result;
            for (int i = 0; i < n; i++)
            {
                image = new
Bitmap(folderBrowserDialog.Selected
Path + "\\" +
faceConfigs[i].indexOfPicture +
".png");
                if(image.Width !=
pictureSize || image.Height !=
pictureSize) image = new
Bitmap(image, new Size(pictureSize,
pictureSize));
                workMatrix =
createMatrtixFromImage(ref image);
                image.Dispose();

                convMatrices = new
double[convolutionNumber][,];
                for (int j = 0; j <
convolutionNumber; j++)
                {
                    convMatrices[j] =
convolution(workMatrix, pictureSize,
j);
                }
                workMatrix =
convolutionsUnion(convMatrices,
pictureSize, convolutionNumber);
                workMatrix =
downsampling(workMatrix,
pictureSize, downsamplingMultiplier);

                layer1 =
getNodesFromMatrix(workMatrix,
pictureSize / downsamplingMultiplier);
                layer2 =
getNewNodes(layer1, nodesMatrix,
firstNeuronLayerSize,
secondNeuronLayerSize, false);
                result =
getNewNodes(layer2, nodesMatrix2,
secondNeuronLayerSize, 3, true);
```

```
            errors[0] += result[0] -
faceConfigs[i].hindrance;
            if (faceConfigs[i].hindrance
== 0)
            {
               errors[1] += (result[1] -
faceConfigs[i].posX / 100.0);
               errors[2] += (result[2] -
faceConfigs[i].posY / 100.0);
            }
            errors =
neuralEducation(errors, result, layer2,
nodesMatrix2, 3,
secondNeuronLayerSize, true);
            neuralEducation(errors,
layer2, layer1, nodesMatrix,
secondNeuronLayerSize,
firstNeuronLayerSize, false);
            errors = new double[3] { 0, 0,
0 };
            Action act = () => {
label2.Text = "" + i;
progressBar1.Value = i; };
            Invoke(act);

            if ((i % testAfter) == 0 && i
!= 0)
            {
               int nSteps = n / testAfter;
               int startIndex = nTest * ((i
/ testAfter) - 1) / nSteps;
               int finalIndex = nTest * (i /
testAfter) / nSteps;

neuralNetworkTest(startIndex,
finalIndex, faceConfigs2, nodesMatrix,
nodesMatrix2);
               lastNumber = i;
            }
         }
            neuralNetworkTest(nTest * (n /
testAfter - 1) / (n / testAfter), nTest,
faceConfigs2, nodesMatrix,
nodesMatrix2);
            lastNumber = n;


saveWeightsToFile(nodesMatrix,
nodesMatrix2);
            Thread.Sleep(150);
         }

         private void
neuralNetworkTest(int startIndex, int
finalIndex, FaceConfig[] faceConfigs,
double[,] nodesMatrix, double[,]
nodesMatrix2)
         {
            double
hindranceAccuracyForChart = 0;
            double hindranceDeltaForChart
= 0;
            double posXDeltaForChart = 0;
            double posYDeltaForChart = 0;

            Bitmap image;
            double[,] workMatrix;
            double[][,] convMatrices;
            double[] layer1;
            double[] layer2;
            double[] result;
            for (int i = startIndex; i <
finalIndex; i++)
            {
               image = new
Bitmap(folderBrowserDialogTest.Selec
tedPath + "\\" +
faceConfigs[i].indexOfPicture +
".png");
               if (image.Width !=
```

```
pictureSize || image.Height != pictureSize) image = new Bitmap(image, new Size(pictureSize, pictureSize));
            workMatrix = createMatrtixFromImage(ref image);
            image.Dispose();

            convMatrices = new double[convolutionNumber][,];
            for (int j = 0; j < convolutionNumber; j++)
            {
                convMatrices[j] = convolution(workMatrix, pictureSize, j);
            }
            workMatrix = convolutionsUnion(convMatrices, pictureSize, convolutionNumber);
            workMatrix = downsampling(workMatrix, pictureSize, downsamplingMultiplier);
            layer1 = getNodesFromMatrix(workMatrix, pictureSize / downsamplingMultiplier);
            layer2 = getNewNodes(layer1, nodesMatrix, firstNeuronLayerSize, secondNeuronLayerSize, false);
            result = getNewNodes(layer2, nodesMatrix2, secondNeuronLayerSize, 3, true);

            if (Math.Abs(result[0] - faceConfigs[i].hindrance) < 0.5)
            {

hindranceAccuracyForChart += 1;
            }
            hindranceDeltaForChart += Math.Abs(faceConfigs[i].hindrance - result[0]);
            if (faceConfigs[i].hindrance == 0)
            {
                posXDeltaForChart += Math.Abs(faceConfigs[i].posX - result[1] * 100);
                posYDeltaForChart += Math.Abs(faceConfigs[i].posY - result[2] * 100);
            }
        }
        lastHindranceAccuracy = hindranceAccuracyForChart * 100 / (finalIndex - startIndex);
        lastDeltaHindrance = hindranceDeltaForChart * 100 / (finalIndex - startIndex);
        lastX = posXDeltaForChart / (finalIndex - startIndex);
        lastY = posYDeltaForChart / (finalIndex - startIndex);
    }

    private void enterConfigs(ref FaceConfig[] faceConfigs, ref FaceConfig[] faceConfigs2, ref double[,] nodesMatrix, ref double[,] nodesMatrix2)
    {
        using (StreamReader streamReader = new StreamReader(openFileDialogConfig.OpenFile()))
        {
            string configText = streamReader.ReadToEnd();
            decomposeElements(ref faceConfigs, n, configText);
```

```
            streamReader.Close();
        }
        using (StreamReader
streamReader = new
StreamReader(openFileDialogTest.Ope
nFile()))
        {
            string configText =
streamReader.ReadToEnd();
            decomposeElements(ref
faceConfigs2, nTest, configText);
            streamReader.Close();
        }
        using (StreamReader
streamReader = new
StreamReader(saveFileDialogSaveLoa
d.OpenFile()))
        {
            string configText =
streamReader.ReadToEnd();
            if (configText == "")
            {
                nodesMatrix =
generateRandomMatrix(secondNeuron
LayerSize, firstNeuronLayerSize, 6);
                nodesMatrix2 =
generateRandomMatrix(3,
secondNeuronLayerSize + 1, 6);
            }
            else
            {

readWeightsFromFile(nodesMatrix,
nodesMatrix2, configText);
            }
            streamReader.Close();
        }
    }

    private void neuralStart()
    {
        Action act = () => {

chart1.Series[0].Points.Clear();

chart2.Series[0].Points.Clear();

chart3.Series[0].Points.Clear();

chart4.Series[0].Points.Clear();
        };
        Invoke(act);

        FaceConfig[] faceConfigs =
new FaceConfig[n];
        FaceConfig[] faceConfigs2 =
new FaceConfig[nTest];
        double[,] nodesMatrix = new
double[secondNeuronLayerSize,
firstNeuronLayerSize];
        double[,] nodesMatrix2 = new
double[3, secondNeuronLayerSize +
1];
        enterConfigs(ref faceConfigs,
ref faceConfigs2, ref nodesMatrix, ref
nodesMatrix2);

        Thread thread = new Thread(()
=> neuralNetworkWork(faceConfigs,
faceConfigs2, nodesMatrix,
nodesMatrix2));
        thread.Start();

        string resultPath =
saveFileDialogSaveLoad.FileName.Re
place(".txt", "Education.csv");
        using (StreamWriter
streamWriter = new
StreamWriter(resultPath, false))
        {

streamWriter.Write("i;Hindrance
```

```
Accuracy;Hindrance Delta;PosX
Delta;PosY Delta\r\n");
            while (thread.IsAlive)
            {
               if (lastNumber != 0)
               {
                  act = () => {

chart1.Series[0].Points.AddXY(lastNu
mber, lastHindranceAccuracy);

chart2.Series[0].Points.AddXY(lastNu
mber, lastX);

chart3.Series[0].Points.AddXY(lastNu
mber, lastY);

chart4.Series[0].Points.AddXY(lastNu
mber, lastDeltaHindrance);
                  };
                  Invoke(act);

streamWriter.Write(lastNumber + ";" +
Math.Round(lastHindranceAccuracy,
3) + ";"
                     +
Math.Round(lastDeltaHindrance, 3) +
";" + Math.Round(lastX, 3) + ";" +
Math.Round(lastY, 3) + "\r\n");
                  lastNumber = 0;
               }
               act = () => {
                  progressBar1.Maximum
= n;
                  label1.Text = "/ " + n;
                  Thread.Sleep(100);
                  Refresh();
               };
               Invoke(act);
            }
            streamWriter.Close();
         }


         act = () => {
            string path =
saveFileDialogSaveLoad.FileName.Re
place(".txt", "Chart");
            chart1.SaveImage(path +
"HindranceAccuracy.png",
System.Drawing.Imaging.ImageFormat
.Png);
            chart2.SaveImage(path +
"PosX.png",
System.Drawing.Imaging.ImageFormat
.Png);
            chart3.SaveImage(path +
"PosY.png",
System.Drawing.Imaging.ImageFormat
.Png);
            chart4.SaveImage(path +
"DeltaHindrance.png",
System.Drawing.Imaging.ImageFormat
.Png);


MessageBox.Show("Ready!");
            progressBar1.Value = 0;
            label1.Text = "/ n";
            label2.Text = "0";

            button1.Enabled = true;
         };
         Invoke(act);
      }

      private void button1_Click(object
sender, EventArgs e)
      {
         StartSettings startSettings =
new StartSettings(this);
         startSettings.ShowDialog();
```

```
            if (!settingsSelected) return;
            settingsSelected = false;

            Thread thread = new Thread(() => neuralStart());
            thread.Start();

            button1.Enabled = false;
        }
    }
}
```

StartSettings.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FaceNeuralNetwork
{
    public partial class StartSettings : Form
    {
        private MainForm mainForm;
        public StartSettings(MainForm form)
        {
            mainForm = form;
            InitializeComponent();
            //textBox1.Text = "" + mainForm.dStep;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = "" + mainForm.dStep;
            if (mainForm.folderBrowserDialog.SelectedPath == "") return;
            if (mainForm.openFileDialogConfig.FileName == "") return;
            if (mainForm.saveFileDialogSaveLoad.FileName == "") return;
            if (mainForm.folderBrowserDialogTest.SelectedPath == "") return;
            if (mainForm.openFileDialogTest.FileName == "") return;
            mainForm.n = (int)numericUpDown1.Value;
            mainForm.nTest = (int)numericUpDown2.Value;
            mainForm.settingsSelected = true;
            Close();
        }

        private void button4_Click(object sender, EventArgs e)
        {

            mainForm.folderBrowserDialog.ShowDialog();
            button4.Text = mainForm.folderBrowserDialog.SelectedPath;
        }

        private void button2_Click(object sender, EventArgs e)
        {
```

```
mainForm.openFileDialogConfig.ShowDialog();
            button2.Text = mainForm.openFileDialogConfig.FileName;
        }

        private void button3_Click(object sender, EventArgs e)
        {

mainForm.saveFileDialogSaveLoad.ShowDialog();
            button3.Text = mainForm.saveFileDialogSaveLoad.FileName;
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            if (double.TryParse(textBox1.Text, out double result)) mainForm.dStep = result;
        }

        private void button7_Click(object sender, EventArgs e)
        {

mainForm.folderBrowserDialogTest.ShowDialog();
            button7.Text = mainForm.folderBrowserDialogTest.SelectedPath;
        }

        private void button5_Click(object sender, EventArgs e)
        {

mainForm.openFileDialogTest.ShowDialog();
            button5.Text = mainForm.openFileDialogTest.FileName;
        }

        private void button6_Click(object sender, EventArgs e)
        {
            ConvolutionSettings convolutionSettings = new ConvolutionSettings(mainForm);

convolutionSettings.ShowDialog();
        }
    }
}
```

ConvolutionSettings:

```
using System;
using System.Windows.Forms;

namespace FaceNeuralNetwork
{
    public partial class ConvolutionSettings : Form
    {
        private MainForm mainForm;
        private int current = 0;
        private int number = 16;

        public ConvolutionSettings(MainForm mainForm)
        {
            this.mainForm = mainForm;
            number = mainForm.convolutionNumber;
            InitializeComponent();
```

```
            dataGridView1.Rows.Add();
            dataGridView1.Rows.Add();
            dataGridView1.Rows.Add();
            dataGridRefresh();
            labelRefresh();
        }

        private void labelRefresh()
        {
            toolStripLabel1.Text = (current
+ 1) + " / " + number;
        }

        private void dataGridRefresh()
        {
            for (int i=0; i< 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {

dataGridView1.Rows[i].Cells[j].Value
=
mainForm.convolutionMatrix[current,
i, j];
                }
            }
    }

        private void saveSettings()
        {
            mainForm.convolutionNumber
= number;
        }

        private void
toolStripButtonPlus_Click(object
sender, EventArgs e)
        {
            if(number < 32)
            {
                number++;
                current = number - 1;
                dataGridRefresh();
                labelRefresh();
            }
            else
            {
                MessageBox.Show("The
maximum number of matrices\n\thas
already been reached!");
            }
        }

        private void swapMatrices(int
changebleElem)
        {
            for (int i = changebleElem; i <
number - 1; i++)
            {
                for (int k = 0; k < 3; k++)
                {
                    for (int p = 0; p < 3; p++)
                    {

mainForm.convolutionMatrix[i, k, p] =
mainForm.convolutionMatrix[i + 1, k,
p];
                    }
                }
            }

            for (int k = 0; k < 3; k++)
            {
                for (int p = 0; p < 3; p++)
                {

mainForm.convolutionMatrix[number -
1, k, p] = 0;
                }
            }
        }
```

```
        private void
toolStripButtonMinus_Click(object
sender, EventArgs e)
        {
            if (number > 1)
            {
                swapMatrices(current);
                number--;
                if(current == number)
current--;
                dataGridRefresh();
                labelRefresh();
            }
            else
            {
                MessageBox.Show("The
minimum number of matrices\n\t\thas
already been reached!");
            }
        }

        private void
toolStripButtonLeft_Click(object
sender, EventArgs e)
        {
            if (current > 0)
            {
                current--;
                dataGridRefresh();
                labelRefresh();
            }
        }

        private void
toolStripButtonRight_Click(object
sender, EventArgs e)
        {
            if (current < number - 1)
            {
                current++;
                dataGridRefresh();
                labelRefresh();
            }
        }

        private void
toolStripButtonEnd_Click(object
sender, EventArgs e)
        {
            saveSettings();
            Close();
        }

        private void
ConvolutionSettings_FormClosing(obj
ect sender, FormClosingEventArgs e)
        {
            saveSettings();
        }

        private void
dataGridView1_CellEndEdit(object
sender, DataGridViewCellEventArgs
e)
        {
            double result;
            if
(dataGridView1.Rows[e.RowIndex].Ce
lls[e.ColumnIndex].Value != null &&
Double.TryParse(dataGridView1.Rows
[e.RowIndex].Cells[e.ColumnIndex].V
alue.ToString(), out result))
            {

mainForm.convolutionMatrix[current,
e.RowIndex, e.ColumnIndex] = result;
            }
        }
    }
}
```