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
ОС магістр

на тему: «Розробка та дослідження алгоритмів обробки зображень з використанням штучного інтелекту в режимі реального часу»

за освітньою програмою **Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи ПЗ2321: / Дмитро ЧЕРКАС /

Керівник: / Вадим ГОРЯЧКІН /

Нормоконтролер: / Світлана ВОЛКОВА /

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

Студент _____

Дніпро – 2025

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

on the topic: «Development and research of image processing algorithms using artificial intelligence in real time»

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

Done by the student of the group PZ2321: _____ / Dmytro CHERKAS /

Scientific Supervisor: _____ / Vadim HORYACHKIN /

Normative controller: _____ / Svitlana VOLKOVA /

Dnipro – 2025

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: **12** Інженерія програмного забезпечення

Спеціальність: **121** Інженерія програмного забезпечення

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

ЗАВДАННЯ

На кваліфікаційну роботу ____Магістр_____
студенту Черкасу Дмитру Анатолійовичу

1. Тема дипломної роботи: Розробка та дослідження алгоритмів обробки зображень з використанням штучного інтелекту в режимі реального часу.
   Керівник роботи: Горячкін Вадим Миколайович
   затверджені наказом 1186 ст від 29.12.2023року
2. Строк подання студентом роботи 20.01.2025 року
3. Вихідні дані до дипломної роботи: програмний продукт.
4. Зміст пояснювальної записки (перелік питань до розробки):
   4.1. Аналітична частина: огляд предметної галузі;
   4.2. Основна частина: опис моделі, опис прототипів, опис методів дослідження;
   4.3. Експеримент та висновки.
5. Перелік демонстраційного матеріалу:

5.1. презентація;

5.2. демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Вступ	15.05.2024	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	01.06.2024	
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	01.07.2024	
4	Постановка задачі, технічне завдання	01.09.2024	
5	Техніко-економічні показники	01.11.2024	30%
6	Розробка інструментальних засобів дослідження	12.11.2024	
7	Виконання досліджень	01.12.2024	
8	Оформлення тез доповідей	09.12.2024	60%
9	Оформлення статті у фаховий журнал	12.12.2024	
10	Оформлення пояснювальної записки	06.01.2025	
11	Розробка демонстраційних матеріалів	09.01.2025	100%
12	Подання кваліфікаційної роботи до кафедри	20.01.2025	
13	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	22.01.2025	

Студент _____ /Дмитро ЧЕРКАС/

Керівник роботи _____ /Вадим ГОРЯКІН/

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра:

67 с., 16 рис. , 36 табл., 3 додатки, 15 джерел.

алгоритмів обробки зображень з використанням штучного інтелекту

Об’єктом дослідження є алгоритми обробки зображень. Предметом дослідження є передбачення ефективності та швидкості навчання згорткових нейронних мереж на зображеннях у режимі реального часу.

Метою дослідження є виявлення найбільш ефективних алгоритмів обробки зображень нейронними мережами підходящої для аналізування набору послідовних зображень, та його реалізація за допомогою вбудованих фреймворків на мові програмування C#.

Методи дослідження: навчання нейронних мереж різних типів і з різними структурами за допомогою різноманітних методів, що аналізують помилки та навчають згорткову нейронну мережу. При розробці програмної реалізації була застосована методологія ООП.

Пояснювальна записка включає вступ, чотири розділи, висновки, бібліографічний список та додатки.

У вступі викладено суть, мету та актуальність роботи (3 сторінок).

Перший розділ присвячений аналізу сучасного стану предметної області (9 сторінок).

Другий розділ охоплює етапи проектування (_ сторінок).

Третій розділ описує процес тестування та відлагодження (_ сторінок).

Четвертий розділ містить опис програмного продукту (_ сторінок).

Додатки включають технічне завдання, специфікацію, листи затвердження та текст програми.

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

ЗМІСТ

ВСТУП

Актуальність роботи. Нейронні мережі на сьогодні є одним із найбільш перспективних і динамічно розвиваються напрямків штучного інтелекту та машинного навчання. Вони зустрічаються в багатьох різних галузях, таких як комп'ютерний зір, обробка природної мови, системи рекомендацій та багатьох інших галузях науки й техніки. Причиною такого стрімкого розвитку є його здатність навчатися на великих обсягах даних, адаптуватися до нових умов і знаходити рішення навіть для дуже складних проблем. Це дозволяє нейронним мережам досягати високої точності при вирішенні різноманітних завдань, що робить їх важливим інструментом у сучасному технологічному світі.

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) [1], у свою чергу, є окремим класом нейронних мереж, спеціально розроблених для роботи з візуальними зображеннями або відеоматеріалами. Вони можуть автоматично виявляти та відрізняти різні ознаки зображення, такі як контури, текстури, форми та інші важливі характеристики, що дозволяє їм бути надзвичайно ефективними в обробці та аналізі візуальної інформації. Завдяки цій здатності, згорткові нейронні мережі використовується в таких сферах, як медична діагностика, розпізнавання облич, автоматичне водіння транспортних засобів та багатьох інших сферах, де потрібна точна обробка зображень.

Згорткові нейронні мережі на даний момент є одним із найпотужніших методів для вирішення задач аналізу зображень. Їх ефективність полягає в тому, що вони можуть виявити найдрібніші деталі, використовуючи численні шари згорток, що дозволяє більш детально аналізувати зображення. Проте, слід зазначити, що ефективність і швидкість таких нейронних мереж значною мірою залежать від вибору їх параметрів та налаштувань До важливих параметрів відносяться, наприклад, кількість шарів у мережі, кількість нейронів у кожному шарі, розмір ядра згортки, а також алгоритм оптимізації та швидкість навчання.

Добре продумана структура та архітектура згорткової нейронної мережі може значно покращити її здатність розпізнавати об’єкти з високою точністю

та підвищити швидкість навчання. Це дозволяє оптимізувати кількість необхідних обчислювальних ресурсів і часу, що вкрай важливо для практичного застосування, особливо в умовах обмеженості ресурсів. Вивчення впливу різних параметрів і налаштувань на продуктивність та швидкість навчання є важливим етапом у розробці та вдосконаленні згоркових мереж.

Крім того, згорткові нейронні мережі можуть бути використані не лише для обробки окремих зображень, але й для роботи з відеоматеріалами, що представляють собою послідовність зображень. Це відкриває нові можливості для обробки відео в реальному часі, наприклад, для систем відеоспостереження, систем автоматичного розпізнавання або для аналізу спортивних трансляцій, тощо. При достатньому рівні оптимізації та гарному налаштуванні згорткові нейронні мережі можуть обробляти відеоматеріали з високою швидкістю та точністю, що робить їх незамінними для роботи з динамічними даними.

Мета роботи. Дослідження впливу типів та структур нейронних мереж на можливость виконання швидкої та точної обробки відеоматеріалів.

Експлуатаційне призначення. Дослідження стане важливим інструментом для розробників, що дозволить їм краще аналізувати ефективність різних параметрів і налаштувань згорткових нейронних мереж. Це дослідження допоможе визначити, які налаштування найбільш оптимальні для конкретних завдань і сценаріїв, що дозволить розробникам створювати навчальні навчальні вибірки та моделі, що забезпечують високу ефективність.

На першому етапі користувачами цих досліджень будуть розробники програмного забезпечення, які працюють над створенням інноваційних рішень у сфері машинного навчання та штучного інтелекту. Проте вони зможуть використовувати результати досліджень для вдосконалення своїх алгоритмів і моделей, оптимізації процесів навчання та підвищення ефективності своїх продуктів, тому кінцевими користувачами результатів цих досліджень будуть люди з різних галузей промисловості, які використовуватимуть продукти на основі згорткових нейронних мереж.

1. Огляд предметної галузі

Згорткові нейронні мережі – це область, що швидко розвивається, і за останні роки набули значної популярності в різних сферах. Вони стали основою для багатьох інноваційних технологій, таких як DALLE [2], U-Net [3], Mask R-CNN [4], GauGAN [5], Tesla Autopilot [6], які широко відомі та використовуються сьогодні. Ключовим аспектом їхнього успіху є використання згорткових нейронних мереж, які допомагають навчити системи обробляти зображення та відео на високому рівні. Враховуючи швидкі темпи розвитку та зростаючу популярність цих технологій, важко уявити майбутнє без них і без подальшого прогресу в цій галузі. Розвиток згорткових нейронних мереж є критично важливим кроком для вдосконалення всіх сфер життя, де використовуються ці технології.

Щоб краще зрозуміти важливість згорткових нейронних мереж у сучасному світі, варто звернути увагу на ряд технологій, які використовують згорткові нейронні мережі для роботи із зображеннями та відео. Одним із таких прикладів є OpenAI DALLE, який здатний генерувати зображення на основі текстових описів. Завдяки згортковим нейронним мережам система здатна інтерпретувати текст і перетворювати його в унікальні зображення, що відкриває нові горизонти в творчості, дизайні та індустрії розваг. Іншим прикладом є технологія GauGAN від NVIDIA, яка дозволяє користувачам створювати реалістичні пейзажі та картини з простих ескізів, використовуючи алгоритми нейронної мережі для аналізу та відтворення деталей зображення.

Не менш важливі нейронні мережі, які використовуються в медицині, зокрема для аналізу медичних зображень. Такі системи, як U-Net або Mask R-CNN, здатні автоматично виявляти аномалії на медичних зображеннях, що допомагає лікарям швидше й точніше діагностувати захворювання. Застосування згорткових нейронних мереж у цій сфері значно зменшити людський фактор, підвищує точність діагностики і, відповідно, врятує більше життів.

Іншим прикладом є технологія автопілота, яка використовується в автомобільній промисловості, зокрема Tesla Autopilot. Ця система використовує камери та датчики для збору візуальної інформації про дорогу, пішоходів, транспортні засоби та інші об’єкти, а згорткові нейронні мережі допомагають обробляти ці дані в реальному часі, забезпечуючи безпеку руху. Завдяки таким технологіям транспорт стає більш автономним і безпечним, що може повністю змінити підхід до перевезень у майбутньому.

Згорткові нейронні мережі вже зараз мають величезний вплив на різні аспекти нашого життя, і в майбутньому їх роль буде тільки зростати. Використання цих технологій стає необхідним для подальшого прогресу в таких сферах, як безпека, медицина, транспорт, творчість та індустрія розваг. Тому розвиток цієї галузі є не лише бажаним, але й необхідним для покращення якості життя та створення нових можливостей.

1.1 Огляд сучасних методів та моделей згорткових нейронних мереж.

Ранні моделі згорткових нейронних мереж, такі як LeNet [7], AlexNet [8] і VGG [9], зіграли важливу роль у створенні нейронних мереж для обробки зображень. Український LeNet був однією із перших згорткових нейронних мереж, які застосували до класифікації рукописних цифр. AlexNet досягла прориву, значно зменшивши кількість помилок завдяки своїй глибокій архітектурі та використанню таких методів, як нормалізація та обчислення за допомогою відеокарти. У свою чергу, VGGNet спростив архітектуру шляхом використання послідовних згорткових шарів розміром 3x3, що дозволило досягти глибшого представлення функцій, але збільшило вимоги до обчислювальних ресурсів. Хоча ці класичні моделі все ще використовуються, сучасні згорткові нейронні мережі стали набагато ефективнішими та гнучкішими завдяки новим підходам.

Одним із ключових нововведень стала модель ResNet [10], яка дозволила значно збільшити глибину нейронних мереж, включаючи варіанти із сотнями шарів. ResNet використовує так звані “залишкові блоки”, які додають шляхи для прямого проходження сигналу через шари, що допомагає уникнути проблем із згасанням градієнтів під час навчання цієї мережі.

Модель Inception [11], також відома як GoogLeNet, представила інноваційний підхід до обробки візуальних даних, використовуючи початкові блоки, які дозволяють використовувати кілька згорткових шарів різних різних розмірів одночасно. Це дало змогу аналізувати як локальні, так і глобальні особливості зображень, значно підвищивши ефективність моделі. Inception-v3 представив додаткові оптимізації, такі як факторизація згортки, яка зменшила кількість параметрів і пришвидшила навчання.

З розвитком мобільних пристроїв виникла необхідність оптимізувати згорткові нейронні мережі для роботи на пристроях з обмеженими ресурсами. Однією з найуспішніших моделей для цієї мети була MobileNet [12], яка використовує “глибокі згорткові шари”, які дозволяють значно скоротити обчислювальні витрати та споживання пам'яті, зберігаючи високу точність.

Інша сучасна модель, EfficientNet [13], стала відомою завдяки підходу до ефективного масштабування. Вона забезпечує високу продуктивність за рахунок збільшення глибини, ширини та роздільної здатності мережі, дозволяючи зменшити кількість параметрів порівняно з іншими моделями. Це робить її одним із лідерів у класифікації зображень на великих наборах даних, зберігаючи енергоефективність і менші вимоги до обчислювальних ресурсів.

Крім завдань класифікації, згорткові нейронні мережі активно використовуються для виявлення об'єктів. Серед найпопулярніших для таких завдань моделі YOLO [14] і Faster R-CNN [15]. YOLO – це високошвидкісна модель, яка розбиває зображення на сітку і одночасно прогнозує класи об'єктів і координати, що дозволяє обробляти відеоматеріали з мінімальною затримкою. Це важливо для таких програм, як відеоспостереження та автономні транспортні засоби. Faster R-CNN, хоч і повільніший, але забезпечує вищу точність завдяки використанню “мереж регіональних пропозицій” для визначення областей, які ймовірно містять об’єкти. Ця модель підходить для завдань, де важливі деталізація і точність.

Крім того, згорткові нейронні мережі активно використовуються в генеративних змагальних мережах, зокрема в моделі GauGAN, яка здатні генерувати фотореалістичні зображення та відео на основі випадкового шуму. Вона, а також аналогічні моделі знайшли застосування в таких завданнях, як стилізація зображень, створення штучних облич і створення унікальних відеоматеріалів, які широко обговорюються в контексті етики та безпеки.

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

Згорткові нейронні мережі використовують багаторівневу архітектуру, де кожен рівень відповідає за розпізнавання дедалі складніших характеристик зображення. На початкових етапах згорткові нейронні мережі виявляють прості особливості, такі як контури та кути, тоді як на більш глибоких шарах формуються більш складні характеристики, такі як форми, текстури та розташування рис зображення. Основні етапи розпізнавання за допомогою згорткової нейронної мережі включають:

1. Попередню обробку зображення, яка може включати масштабування, нормалізацію та підготовку даних, тощо;
2. Виділення ознак за допомогою згорткових шарів, які характеризують унікальність зображення;
3. Класифікація або порівняння виділених ознак із раніше отриманими при навчанні.

Сучасні згорткові нейронні мережі досягли значної точності в задачах розпізнавання зображень. Ці моделі використовують попередньо навчені мережі, навчені на великих наборах даних. Це дозволяє їм досягти високої точності навіть у складних умовах, таких як різне освітлення чм зміни ракурсів. Однією з ключових особливостей згорткових нейронних мереж є здатність адаптувати попередньо підготовлені моделі до нових наборів даних або завдань. Завдяки цьому високих результатів можна досягти навіть при обмеженому обсязі тренувальних даних.

Переваги згорткових нейронних мереж в розпізнаванні облич:

По-перше, це точність. Згорткові нейронні мережі дозволяють досягти високої точності навіть за несприятливих умов, таких як зміна освітлення, положення або часткова деформація облич.

По-друге, це гнучкість. Згорткові нейронні мережі можна застосовувати для різних завдань – від ідентифікації осіб до визначення їхніх характеристик, що робить ці моделі універсальними.

По-третє, це швидкість обробки. Завдяки паралельним обчисленням на графічному процесорі, згорткова нейронна мережа може працювати в реальному часі, що важливо для таких програм, як системи безпеки або відеоспостереження.

По-четверте, це адаптація. Згорткові нейронні мережі можуть адаптуватися до нових завдань шляхом перенавчання на нових даних, що дозволяє ефективно використовувати моделі для нових завдань з мінімальними витратами на навчання.

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

Згорткові нейронні мережі є потужним інструментом для вирішення завдань комп'ютерного зору, таких як класифікація зображень, розпізнавання облич, виявлення інших об'єктів, тощо. Однак, як і будь-яка технологія, згорткові нейронні мережі мають ряд обмежень і проблем, які впливають на їх ефективність, гнучкість та можливість практичного застосування. Нижче ми розглянемо основні перешкоди, які виникають при використанні згорткових нейронних мереж.

По-перше, суттєвим обмеженням згорткових нейронних мереж є потреба у великих наборах даних для ефективного навчання. Щоб модель могла точно розпізнавати функції та демонструвати високу продуктивність, вона повинна мати доступ до тисяч або навіть мільйонів навчальних прикладів. Це створює кілька проблем:

1. Для спеціальних завдань, таких як медична діагностика або розпізнавання конкретних об'єктів, може бути важко зібрати достатню кількість даних.
2. Набори даних можуть мати нерівномірний розподіл, коли деякі класи представлені частіше, ніж інші, що може погіршити точність моделі для рідкісних класів.

По-друге, згорткові нейронні мережі зазвичай мають глибоку архітектуру з великою кількістю параметрів, що робить їх дуже інтенсивними з точки зору обчислень. Це призводить до таких проблем:

1. Для ефективного навчання та використання згорткових нейронних мереж зазвичай потрібні потужні графічні процесори такі як відео карта або спеціалізовані апаратні рішення, наприклад TPU, що збільшує витрати на розробку та обслуговування.
2. Навчання глибоких моделей може зайняти багато часу, особливо на великих наборах даних. Це стає важливим фактором, якщо моделі потребують частого перенавчання.
3. Через вимоги до обчислень, згорткові нейронні мережі є енергоємними,

що створює проблеми під час використання на мобільних пристроях або у випадках обмежених ресурсів.

По-третє, незважаючи на високу точність, рішення згорткових нейронних мереж часто важко інтерпретувати через їх “чорний ящик”. Це спричиняє кілька проблем:

1. Модель може давати неправильні результати в нетипових або рідкісних ситуаціях, і без інтерпретації важко з’ясувати причину цих помилок.
2. У критичних сферах, таких як медицина чи системи безпеки, невизначеність у модельних рішеннях може призвести до небажаних, або навіть критичних наслідків.

По-четверте, глибокі згорткові нейронні мережі часто мають велику кількість параметрів, що може призвести до проблеми переобладнання, коли модель добре працює на навчальних даних, але погано на нових вибірках. Щоб вирішити цю проблему, можна використовувати методи регулярізації.

По-п’яте, згорткові нейронні мережі можуть мати труднощі з узагальненням нових типів даних, незважаючи на успішні результати на навчальних наборах:

1. Моделі, навчені на певних даних, можуть погано працювати на нових наборах, якщо умови, за яких вони були створені, відрізняються.
2. Навчання згорткових нейронних мереж на нових великих наборах даних або різних типах зображень може вимагати значних зусиль і ресурсів для перенавчання.

Висновки до розділу 1

Згорткові нейронні мережі є ефективним інструментом для вирішення складних проблем зв’язаних із комп’ютерним зором, таких як класифікація зображень і розпізнавання облич, але вони мають певні обмеження. Серед основних труднощів – потреба у великих обсягах даних, значні обчислювальні ресурси та високе енергоспоживання. Незважаючи на високу точність згорткових нейронних мереж, їхні рішення важко пояснити, а проблеми з перенавчанням і узагальненням можуть негативно вплинути на продуктивність нових даних. Проте згорткові нейронні мережі продовжують залишатися однією з найдосконаліших технологій роботи з візуальною інформацією, що сприяє розвитку нових підходів до подолання цих обмежень.

2 ПРОЕКТУВАННЯ

2.1 Зовнішнє проектування

2.1.1 Функціональне призначення

Функціональним призначенням є можливість проведення дослідницької роботи у сфері навчання різних типів нейронних мереж та їх різних структур, а також можливість їх тестування на відеоматеріалах.

2.1.2 Експлуатаційне призначення

Експлуатаційним призначенням є оптимізація архітектури та структури нейронних мереж, шляхом дослідження їх впливу без втрачання точності їх роботи та без збільшення часових витрат.

2.1.3 Функціональні вимоги

Програма повинна надавати такі можливості:

- Вибирати вибірки для навчання;
- Вибирати вибірки для тестування;
- Вибирати тип нейронної мережі;
- Вибирати пристрій для роботи із нейронною мережєю;
- Вибирати функцію активації нейронної мережі;
- Змінити внутрішню структуру нейронної мережі;
- Зберегти налаштування нейронної мережі;
- Завантажити налаштування нейронної мережі;
- Змінити кількість епох навчання нейронної мережі;
- Навчати нейронну мережу;
- Тестувати нейронну мережу на заздалегідь підготовлених вибірках
- Тестувати нейронну мережу на відеозаписі;
- Тестувати нейронну мережу на потоці відеокамери.

2.1.4 Вхідні та вихідні дані

Вхідними даними є:

- Файл конфігурації вибріки навчання;
- Файл конфігурації вибірки тестування;
- Тип нейронної мережі;
- Пристрій для роботи із нейронною мережею;
- Функція активації нейронної мережі;
- Внутрішня структура нейронної мережі;
- Файл збереження нейронної мережі;
- Файл завантаження нейронної мережі;
- Кількість епох навчання;
- Файл відеозапису для тестування нейронної мережі;
- Формат відео потоку відеокамери для тестування нейронної мережі;
- Якість зображення потоку відеокамери для тестування нейронної мережі.

Вихідними даними є:

- Текстові повідомлення щодо завершення навчання нейронної мережі;
- Текстові повідомлення щодо завершення тестування нейронної мережі;
- Текстові повідомлення щодо точності тестування нейронної мережі;
- Візуальне виділення об’єкту на відеозаписі;
- Візуальне виділення об’єкту на потоці відеокамери;
- Файл збереження нейронної мережі.

2.1.5 Опис зовнішнього інформаційного середовища

Для функціонування програми потрібна операційна система Windows 7 або більш нова, наявність стандартних бібліотек та бібліотеки .Net 4.0.

Внутрішня структура нейронної мережі та кількість епох навчання

вводяться з клавіатури в числове поле з контролем введення.

Тип нейронної мережі, пристрій для роботи із нейронною мережею, функція активації нейронної мережі, формат відео потоку відеокамери для тестування нейронної мережі та якість зображення потоку відеокамери для тестування нейронної мережі обираються за допомогою натискання лівої кнопки миші на відповідному пункті списку, представленому на формі.

Файли конфігурацій вибірок навчання та тестування, файли збереження та завантаження нейронних мереж та файл відеозапису для тестування нейронної мережі вводяться за допомогою інтерактивного відкриття файлів.

Специфікація функціональних вимог виконана у вигляді діаграми прецедентів (рис. 2.1.1).

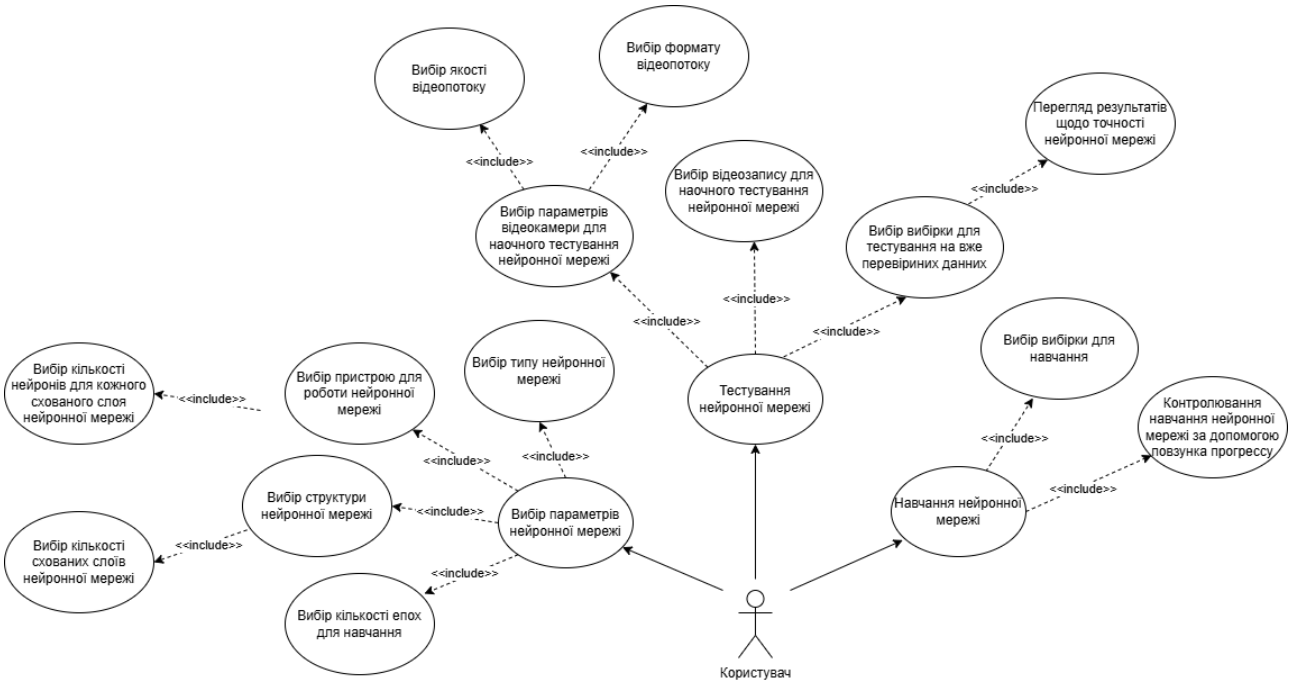

Рисунок 2.1.1 – Діаграма преценденітв программи

2.2 Внутрішнє проектування

2.2.1 Аналіз зовнішніх специфікацій систем

2.2.1.1 Моделювання словника системи

Розглянувши сценарії взаємодії користувача та математичні моделі із системою, можна виділити такі базові сутності: Інтерфейс мереж, Оверлей інтерфейс мереж, Читач данних, Згорткова нейронна мережа, Згорткова

нейронна мережа з обробкою відеочіпом, Yolo, Yolo з обробкою відеочіпом, Шукач об’єктів.

Ідентифіковані обов'язки:

Інтерфейс мереж – основний клас що забезпечує взаємодію користувача з різними типами нейронних мереж. Відповідає за налаштування параметрів, запуск процесів навчання і тестування;

Оверлей інтерфейс мереж – клас-помічник до класу «Інтерфейс мереж» що реалізує додатковий шар, що відображає об'єкти знайдені мережею на відеозаписі;

Читач данних – клас-помічник до класів «Інтерфейс мереж» та классів нейроних мереж відповідає за зчитування даних для навчання та тестування нейронних мереж;

Згорткова нейронна мережа – основний клас що реалізує згорткову нейронну мережу з обробкою процессором;

Згорткова нейронна мережа з обробкою відеочіпом – основний клас що реалізує згорткову нейронну мережу з обробкою відеочіпом;

Yolo – основний клас що реалізує YOLO нейронну мережу з обробкою процессором;

Yolo з обробкою відеочіпом – основний клас що реалізує YOLO нейронну мережу з обробкою відеочіпом;

Шукач об’єктів – клас-помічник до класів «Yolo» та «Yolo з обробкою відеочіпом» що виконує ідентифікацію об’єктів на основі передбачень нейронної мережі.

Операції та атрибути, що необхіднні для виконання обов'язків кожної сутності наведемо у табл. 2.2.1.

Таблиця 2.2.1 – Сутності, атрибути та методи

Сутність	Атрибути	Методи
Інтерфейс мереж	gpuNeuralNetwork – GPU-реалізація згорткової нейронної мережі; cpuNeuralNetwork – CPU-реалізація згорткової нейронної мережі; yoloNeuralNetwork – YOLO-мережа на CPU; gpuYoloNeuralNetwork – YOLO-мережа на GPU; neuralNetwork – Вказівник на активну нейронну мережу; dataLoader – Посилання на сутність читача даних; hiddenLayerSizes – Розміри прихованих шарів; webkamStart – Код команди для старту вебкамери; webcamDriverConnect – Код команди для підключення драйвера вебкамери; webcamDriverDisconnect – Код команди для відключення драйвера вебкамери; webcamCopyToClipboard – Код команди для копіювання кадру в буфер обміну;	Form1 – Конструктор інтерфейсу; GetSelectedActivationFunction – Отримує вибрану функцію активації; ComboBoxNetworkType_SelectedIndexChanged – Обробляє зміну типу мережі; ChangeNeuralNetwork – Змінює активну нейронну мережу; comboBoxActivation_SelectedIndexChanged – Обробляє зміну функції активації; InitializeNetwork – Ініціалізує мережу; AddLogs – Додає повідомлення до журналу; btnTrain_Click – Запускає процес навчання; btnTest_Click – Запускає тестування мережі; TestNetworkAsync – Асинхронний метод тестування мережі;

Продовження таблиці 2.2.1

	webcamGrabFrame – Код команди для отримання кадру з вебкамери; webcamSetPreviev – Код команди для увімкнення попереднього перегляду; webcamSetPrevievRate – Код команди для встановлення частоти попереднього перегляду; webcamSetScale – Код команди для налаштування масштабу вебкамери; webcamDLGVideoformat – Код команди для налаштування відео формату; webcamSetVideoformat – Код команди для встановлення формату відео; webcamChildWindow – Код команди для встановлення дочірнього вікна вебкамери; webcamVisible – Код команди для налаштування видимості вебкамери; hCaptureWindow – поток вікна захоплення вебкамери; timerWebkam – Таймер для оновлення кадрів вебкамери;	btnAnalyze_Click – Аналізує вхідні зображення; DisplayPrediction – Відображає результати передбачень; btnSave_Click – Зберігає конфігурацію мережі; btnLoad_Click – Завантажує конфігурацію мережі; SetVideoFormat – Встановлює формат відео; StartCamera – Запускає вебкамеру; StopCamera – Зупиняє вебкамеру; PictureBoxWebcam_SizeChanged – Оновлює розмір вікна вебкамери; PictureBoxWebcam_LocationChanged – Оновлює розташування вікна вебкамери; SyncPictureBoxMaskWithPictureBoxMask – Синхронізує розміри елементів на інтерфейсі

Продовження таблиці 2.2.1

	timerVideo – Таймер для відео потоку; drawImageFromWebcam – Флаг для виводу кадру вебкамери; xWebcam – Положення по X для відображення рамки об’єкту; yWebcam – Положення по Y для відображення рамки об’єкту; sizeXWebcam – Розмір рамки по X для відображення рамки об’єкту; sizeYWebcam – Розмір рамки по Y для відображення рамки об’єкту; penUniversal – Пензель для малювання; isVideo – Флаг активності відео потоку; overlayForm – Посилання на форму для накладення графічних елементів.	для коректного відображення об’єктів; ProccessNeural – Обробляє тестове зображення нейронною мережею; WebcamChangePhoto – Оновлює кадри вебкамери; VideoChangePhoto – Оновлює кадри відео; buttonWebcam_Click – Обробляє натискання кнопки для запуску вебкамери; buttonCustomVideo_Click – Обробляє запуск відео; pictureBoxWebcam_Paint – Малює елементи поверх кадрів вебкамери; Form1_FormClosing – Обробляє закриття форми; RefreshLocationOfForm – Оновлює позицію форми; Form1_LocationChanged – Реагує на зміну позиції форми; numericUpDownStruct_ValueChanged – Обробляє зміну кількості структур;

Продовження таблиці 2.2.1

		dataGridViewStruct_CellEnd Edit – Оновлює значення структури нейронної мережі.
Оверлей інтерфейс мереж	drawImage – Флаг для виводу кадру вебкамери; x – Положення по X для відображення рамки об’єкту; y – Положення по Y для відображення рамки об’єкту; sizeX – Розмір рамки по X для відображення рамки об’єкту; sizeY – Розмір рамки по Y для відображення рамки об’єкту; penUniversal – Пензель для малювання.	OverlayForm – Конструктор інтерфейсу.
Читач данних	imageCount – Кількість зображень, які доступні для обробки; imageMetadata – Метадані зображень, включаючи інформацію про шляхи до файлів та параметри, необхідні для обробки; yoloImageMetadata – Метадані для YOLO, які містять інформацію про об'єкти та координати рамок;	DataLoader – Конструктор створення об’єкту; LoadMetadata – Завантажує метадані з файлу, включаючи шляхи до зображень та відповідну інформацію для обробки; GetNextDataAsync – Асинхронно отримує наступне зображення та пов'язану з ним інформацію для обробки;

Продовження таблиці 2.2.1

	currentIndex – Поточний індекс оброблюваного зображення або даних; directory – Папка, в якій розташовані файли зображень та метаданих.	LoadImageAsync – Завантажує зображення з файлу у вигляді матриці для подальшої обробки; LoadImageFromWebcamPictureBox – Завантажує зображення з вебкамери та перетворює його у вигляд, придатний для обробки; YoloLoadImageAsync – Завантажує зображення з файлу спеціально для використання в YOLO; GetNextYoloDataAsync – Асинхронно отримує дані для обробки YOLO, включаючи зображення та координати об'єктів; GenerateYoloOutput – Генерує вихідні дані для YOLO на основі завантаженого зображення та метаданих; Reset – Скидає індекс обробки, дозволяючи почати обробку даних із самого початку.

Продовження таблиці 2.2.1

Згорткова нейронна мережа	inputWidth – Ширина вхідного зображення, яке обробляється мережею; inputHeight – Висота вхідного зображення, яке обробляється мережею; filterSize – Розмір згорткового фільтру; filter – Матриця згорткового фільтру для операцій згортки; hiddenLayerWeights – Ваги прихованих шарів нейронної мережі; outputLayerWeights – Ваги вихідного шару нейронної мережі; activationType – Тип функції активації, яка використовується в мережі; numClasses – Розмір вихідного слою; clipValue – Обмеження градієнтів під час навчання; form – Вказівник на графічний інтерфейс, який взаємодіє з мережею; activationFunctions – Список функцій активації, які	ConvolutionalNeuralNetwork – Конструктор створення об’єкту; SetActivationFunctions – Встановлює функції активації для мережі відповідно до заданого типу; Convolve – Виконує операцію згортки на вхідному зображенні; MaxPool – Зменшує розмір зображення, виконуючи операцію максимального пулінгу; Flatten – Перетворює зображення у одномірний масив; ApplyActivation – Застосовує функцію активації до вхідних даних; FullyConnected – Виконує операцію повного зв’язку між шарами нейронної мережі; TrainAsync – Навчає нейронну мережу у асинхронному режимі,

Продовження таблиці 2.2.1

	застосовуються до кожного шару; activationDerivatives – Список похідних функцій активації для кожного шару.	використовуючи надані дані; SaveNetwork – Зберігає стан мережі, включаючи ваги і конфігурацію, у файл; LoadNetwork – Завантажує стан мережі з файлу; Backpropagate – Виконує операцію зворотного поширення помилки для оновлення ваг мережі; Predict – Генерує передбачення для заданого вхідного зображення; Sigmoid – Реалізує сигмоїдну функцію активації; SigmoidDerivative – Обчислює похідну сигмоїдної функції активації; ReLU – Реалізує функцію активації ReLU; ReLUDerivative – Обчислює похідну функції активації ReLU; LeakyReLU – Реалізує

Продовження таблиці 2.2.1

		функцію активації Leaky ReLU; LeakyReLUDerivative – Обчислює похідну функції активації Leaky ReLU; Tanh – Реалізує гіперболічну тангенціальну функцію активації; TanhDerivative – Обчислює похідну гіперболічної тангенціальної функції активації;
Згорткова нейронна мережа з обробкою відеочіпом	inputWidth – Ширина вхідного зображення, яке обробляється мережею; inputHeight – Висота вхідного зображення, яке обробляється мережею; filterSize – Розмір згорткового фільтру; filter – Матриця згорткового фільтру для обчислень на GPU; hiddenLayerWeights – Ваги прихованих шарів нейронної мережі; outputLayerWeights – Ваги вихідного шару нейронної мережі;	GpuConvolutionalNeuralNetwork – Конструктор створення об’єкту; SaveNetwork – Зберігає стан мережі, включаючи ваги і конфігурацію, у файл; LoadNetwork – Завантажує стан мережі з файлу; SetActivationFunctions – Встановлює функції активації для мережі відповідно до заданого типу; Forward – Виконує проходження вперед через

Продовження таблиці 2.2.1

	numClasses – Розмір вихідного слою; activationFunctions – Список функцій активації, які застосовуються до кожного шару; activationType – Тип функції активації, яка використовується в мережі; form – Вказівник на графічний інтерфейс, який взаємодіє з мережею.	мережу для генерації передбачень; FullyConnected – Виконує операцію повного зв’язку між шарами нейронної мережі; ApplyActivation – Застосовує функцію активації до вхідних даних; TrainAsync – Навчає нейронну мережу у асинхронному режимі, використовуючи GPU; Predict – Генерує передбачення для заданого вхідного зображення; UpdateOutputLayerWeights – Оновлює ваги вихідного шару з використанням GPU; UpdateHiddenLayerWeights – Оновлює ваги прихованих шарів з використанням GPU; ComputeHiddenLayerErrors – Обчислює помилки прихованих шарів для зворотного поширення;

Продовження таблиці 2.2.1

		Backpropagate – Виконує операцію зворотного поширення помилки для оновлення ваг з використанням GPU; LaunchConvolution2D – Запускає операцію згортки на GPU; LaunchMaxPooling – Запускає операцію максимального пулінгу на GPU; LaunchReLUActivation – Запускає функцію активації ReLU на GPU; LaunchLeakyReLUActivation – Запускає функцію активації Leaky ReLU на GPU; LaunchSigmoidActivation – Запускає сигмоїдну функцію активації на GPU; LaunchTanhActivation – Запускає гіперболічну тангенціальну функцію активації на GPU; Convolve – Виконує операцію згортки на

Продовження таблиці 2.2.1

		вхідному зображенні; Flatten – Перетворює зображення у одномірний масив; MaxPool – Зменшує розмір зображення, виконуючи операцію максимального пулінгу; ApplyReLU – Реалізує функцію активації ReLU; ApplyLeakyReLU – Реалізує функцію активації Leaky ReLU; ApplySigmoid – Реалізує сигмоїдну функцію активації; ApplyTanh – Реалізує гіперболічну тангенціальну функцію активації; RunActivation – Запускає обрану функцію активації на GPU.
Yolo	KernelSize – Розмір ядра згортки, який використовується в згорткових шарах; Stride – Крок згортки, який визначає зсув ядра при виконанні операції згортки;	ConvolutionalLayer – Конструктор створення об’єкту; SaveNetwork – Зберігає стан мережі, включаючи параметри та

Продовження таблиці 2.2.1

	Filters – Набір фільтрів для виконання згорткових операцій; _random – Генератор випадкових чисел для ініціалізації ваг фільтрів; _activationFunction – Тип функції активації, яка використовується в шарах нейронної мережі; _convLayer – Об'єкт згорткового шару, який виконує операції згортки; _objectDetector – Об'єкт для виявлення об'єктів у зображеннях на основі результатів згорткових шарів; form – Вказівник на графічний інтерфейс.	конфігурацію, у файл; LoadNetwork – Завантажує стан мережі з файлу; InitializeFilters – Ініціалізує фільтри для згорткового шару з використанням випадкових значень; Apply – Виконує згортку на вхідному зображенні із застосуванням фільтрів; ApplyActivation – Застосовує функцію активації до вихідних даних згорткового шару; TrainAsync – Навчає нейронну мережу в асинхронному режимі, використовуючи задані дані; Predict – Генерує передбачення для вхідного зображення, виявляючи об'єкти; GenerateDetectionOutput – Генерує вихідні дані для виявлення об'єктів на основі результатів згорткового шару.

Продовження таблиці 2.2.1

Yolo з обробкою відеочіпом	KernelSize – Розмір ядра згортки, яке використовується для виконання операцій згортки; Stride – Крок згортки, що визначає зсув ядра під час обчислень; Filters – Набір фільтрів, які застосовуються для згорткових обчислень; _random – Генератор випадкових чисел для ініціалізації ваг фільтрів; _activationFunction – Тип функції активації, яка використовується для обробки результатів згортки; _convLayer – Згортковий шар, який виконує обчислення на GPU; _objectDetector – Компонент для виявлення об'єктів у зображеннях або відео з використанням GPU; form – Вказівник на графічний інтерфейс, який використовується для взаємодії з мережею.	GpuConvolutionalLayer – Конструктор створення об’єкту; SaveNetwork – Зберігає конфігурацію мережі, включаючи параметри фільтрів та шари, у файл; LoadNetwork – Завантажує конфігурацію мережі з файлу; InitializeFilters – Ініціалізує фільтри для згорткового шару випадковими значеннями; Apply – Виконує згортку на вхідних даних із використанням фільтрів; ApplyConvolutionOnGpu – Запускає операцію згортки на GPU; TrainAsync – Навчає модель у асинхронному режимі, використовуючи обчислення на GPU; Predict – Генерує передбачення для вхідного зображення, використовуючи

Продовження таблиці 2.2.1

		можливості GPU для обробки; GenerateDetectionOutput – Формує вихідні дані для виявлення об'єктів на основі результатів згорткових операцій.
Шукач об’єктів	Threshold – Поріг впевненості для виявлення об’єктів. Об’єкти, впевненість яких нижча за цей поріг, не враховуються у результатах.	ObjectDetector – Конструктор створення об’єкту; DetectObjects – Виявляє об’єкти на основі результатів нейронної мережі; DetectObjectsOnGpu – Виявляє об’єкти на основі результатів нейронної мережі на GPU.

2.2.1.2 Моделювання розподілу обов’язків у системі

Множини сутностей, можна об’єднати для визначення спільних обовязків:

– Інтерфейс мереж, оверлей інтерфейс мереж – взаємодія користувача з нейронними мережами, відображення результатів роботи мереж та їх конфігурація;

– Читач данних – завантаження, підготовка та передача даних для обробки нейронними мережами;

– Згорткова нейронна мережа, згорткова нейронна мережа з обробкою відеочіпом, Yolo, Yolo з обробкою відеочіпом, Шукач об’єктів – виявлення об’єктів у зображеннях;

Визначення 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
Інтерфейс мереж	Оверлей інтерфейс мереж	Асоціація
Інтерфейс мереж	Читач данних	Агрегація
Інтерфейс мереж	Згорткова нейронна мережа	Агрегація
Інтерфейс мереж	Згорткова нейронна мережа з обробкою відеочіпом	Агрегація
Інтерфейс мереж	Yolo	Агрегація
Інтерфейс мереж	Yolo з обробкою відеочіпом	Агрегація
Yolo	Шукач об’єктів	Асоціація
Yolo з обробкою відеочіпом	Шукач об’єктів	Асоціація
Згорткова нейронна мережа	Читач данних	Агрегація
Згорткова нейронна мережа з обробкою відеочіпом	Читач данних	Агрегація
Yolo	Читач данних	Агрегація
Yolo з обробкою відеочіпом	Читач данних	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 – 2.2.10 наведено CRC-картки на класи проекту.

Таблиця 2.2.3 – CRC-картка для класу «Інтерфейс мереж»

Базовий клас	Похідні класи (нащадки)
Відсутні	відсутні
Обов’язки	Зв’язки
Забезпечує основну взаємодію користувача із системою, включаючи налаштування, запуск, навчання та тестування нейронних мереж	Оверлей інтерфейс мереж, Читач данних, Згорткова нейронна мережа, Згорткова нейронна мережа з обробкою відеочіпом, Yolo, Yolo з обробкою відеочіпом

Таблиця 2.2.4 – CRC-картка для класу «Оверлей інтерфейс мереж»

Базовий клас	Похідні класи (нащадки)
Відсутні	відсутні
Обов’язки	Зв’язки
Відображення результатів роботи мереж при тестуванні на відеозаписі	Інтерфейс мереж

Таблиця 2.2.5 – CRC-картка для класу «Читач данних»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Підготувати вхідні дані до нейронних мереж	Інтерфейс мереж, Згорткова нейронна мережа, Згорткова нейронна мережа з обробкою відеочіпом, Yolo, Yolo з обробкою відеочіпом

Таблиця 2.2.6 – CRC-картка для класу «Згорткова нейронна мережа»

Базовий клас	Похідні класи (нащадки)
Відсутні	відсутні
Обов’язки	Зв’язки
Визначає чи є об’єкт на зображенні, визначає його розташування та розміри шляхом використання зготкової нейронної мережі	Інтерфейс мереж, Читач данних

Таблиця 2.2.7 – CRC-картка для класу «Згорткова нейронна мережа з обробкою відеочіпом»

Базовий клас	Похідні класи (нащадки)
Відсутні	відсутні
Обов’язки	Зв’язки
Визначає чи є об’єкт на зображенні, визначає його розташування та розміри за допомогою GPU шляхом використання зготкової нейронної мережі	Інтерфейс мереж, Читач данних

Таблиця 2.2.8 – CRC-картка для класу «Yolo»

Базовий клас	Похідні класи (нащадки)
Відсутні	відсутні
Обов’язки	Зв’язки
Визначає чи є об’єкт на зображенні, визначає його розташування та розміри шляхом використання YOLO нейронної мережі	Інтерфейс мереж, Шукач об’єктів, Читач данних

Таблиця 2.2.9 – CRC-картка для класу «Yolo з обробкою відеочіпом»

Базовий клас	Похідні класи (нащадки)
Відсутні	відсутні

Обов’язки	Зв’язки
Визначає чи є об’єкт на зображенні, визначає його розташування та розміри за допомогою GPU шляхом використання YOLO нейронної мережі	Інтерфейс мереж, Шукач об’єктів, Читач данних

Таблиця 2.2.10 – CRC-картка для класу «Шукач об’єктів»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Виконує ідентифікацію об’єктів на основі передбачень нейронної мережі	Yolo, Yolo з обробкою відеочіпом

2.2.1.4 Побудова об’єктної моделі (діаграми класів)

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

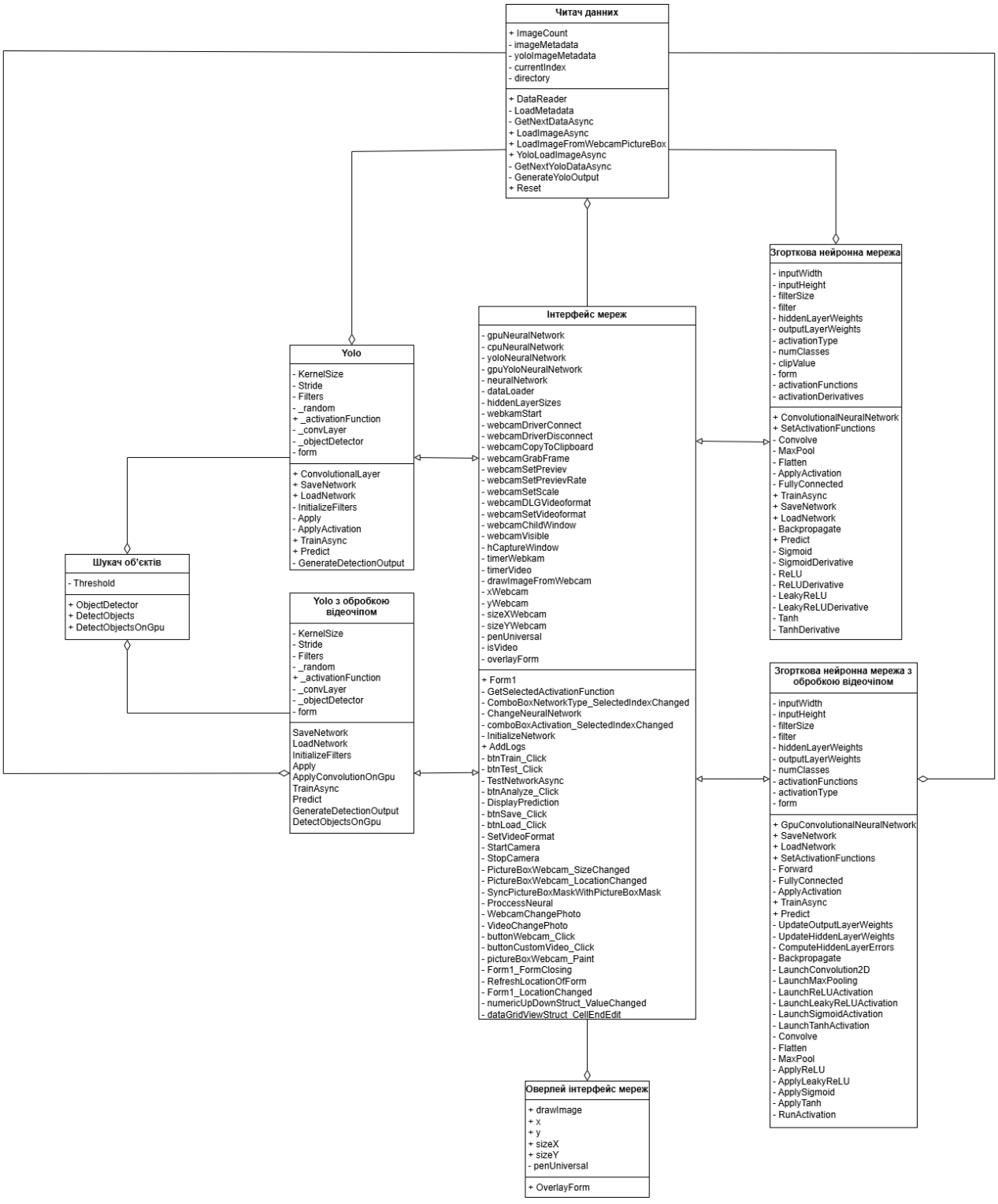

Рисунок 2.2.1 – Діаграма класів

2.2.2 Проектування інтерфейсу користувача

2.2.2.1 Створення ескізів форм

Гарним тоном є надання користувачу можливості мати доступ до швидкого виконання дій. Тому форма повинна бути розвантажена.

Програма складається з одного вікна поділеного на секції:

– головна секція (рис. 2.2.2);
– секція налаштування нейронних мереж (рис. 2.2.3);
– секція журналу (рис. 2.2.4).

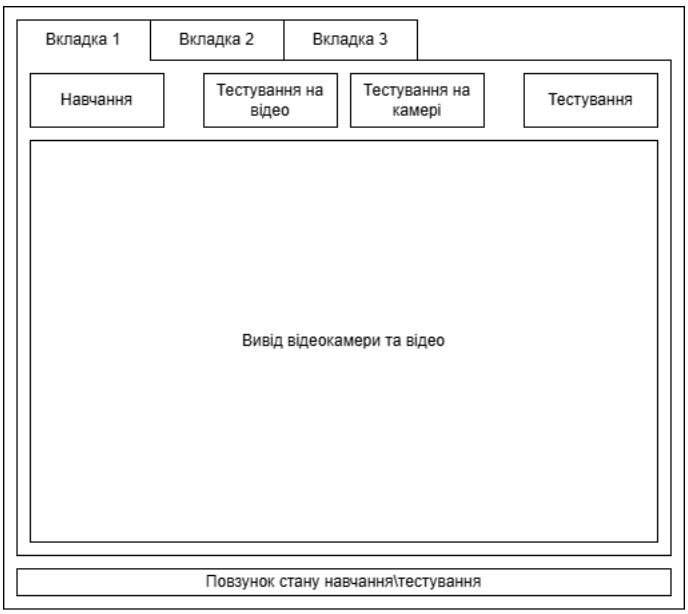

Рисунок 2.2.2 – Ескіз головної секції

Головна секція (рис. 2.2.2) має сім пунктів меню:

1. Кнопки зміни вкладок;
2. Кнопка навчання нейронної мережі на підготовлених даних;
3. Кнопка тестування нейронної мережі на підготовлених даних;
4. Кнопка тестування нейронної мережі на відеозаписі;
5. Кнопка тестування нейронної мережі на відеокамері;
6. Область виводу відеокамери та відеозаписів;
7. Повзунок стану навчання\тестування.

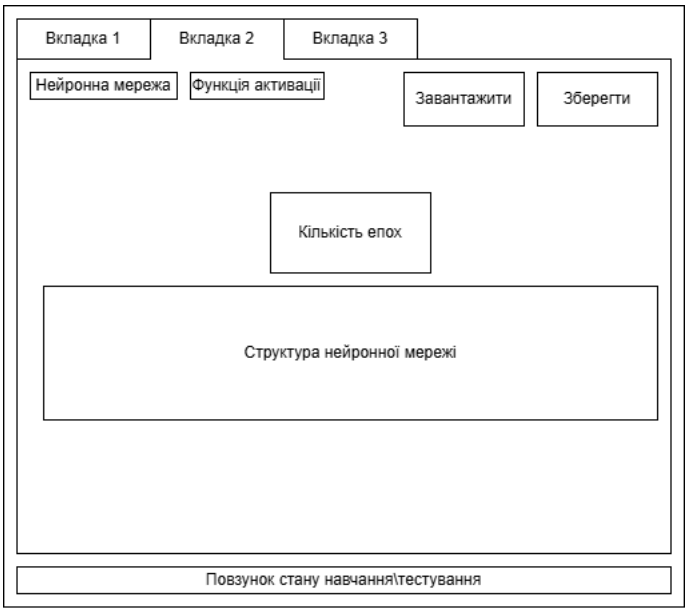

Рисунок 2.2.3 – Секція налаштування нейронних мереж

Секція налаштування нейронних мереж (рис. 2.2.3) має вісім пунктів меню:

1. Кнопки зміни вкладок;
2. Випадаючий список вибору типу нейронної мережі;
3. Випадаючий список вибору функції активації;
4. Кнопка збереження нейронної мережі;
5. Кнопка завантаження нейронної мережі;
6. Поле введення кількості епох для навчання нейронної мережі;
7. Поля налаштування структури нейронної мережі;
8. Повзунок стану навчання\тестування.

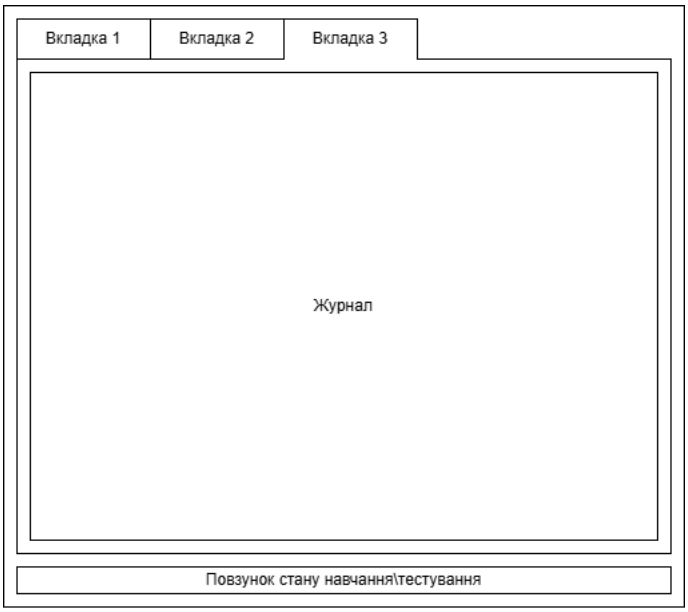

Рисунок 2.2.4 – Секція журналу

Секція журналу (рис. 2.2.4) має три пункти меню:

1. Кнопки зміни вкладок;
2. Поле журналу;
3. Повзунок стану навчання\тестування.

2.2.3 Проектування динаміки системи

Об'єктами виступають користувач та класи, які реалізують функціонал системи.

Можна виділити по одному варіанту використання на кожну нейронну мережу, а саме:

- вибір параметрів згорткової нейронної мережі, та робота із нею;
- вибір параметрів згорткової нейронної мережі з обробкою відеочіпом, та робота із нею;
- вибір параметрів YOLO нейронної мережі, та робота із нею;
- вибір параметрів YOLO нейронної мережі з обробкою відеочіпом, та робота із нею.

Сценарій «вибір параметрів згорткової нейронної мережі, та робота із нею»: спочатку користувач вибирає згорткову нейронну мережу, її параметри, запускає навчання, запускає тестування. Діаграма послідовності представлена

на рис. 2.2.5.

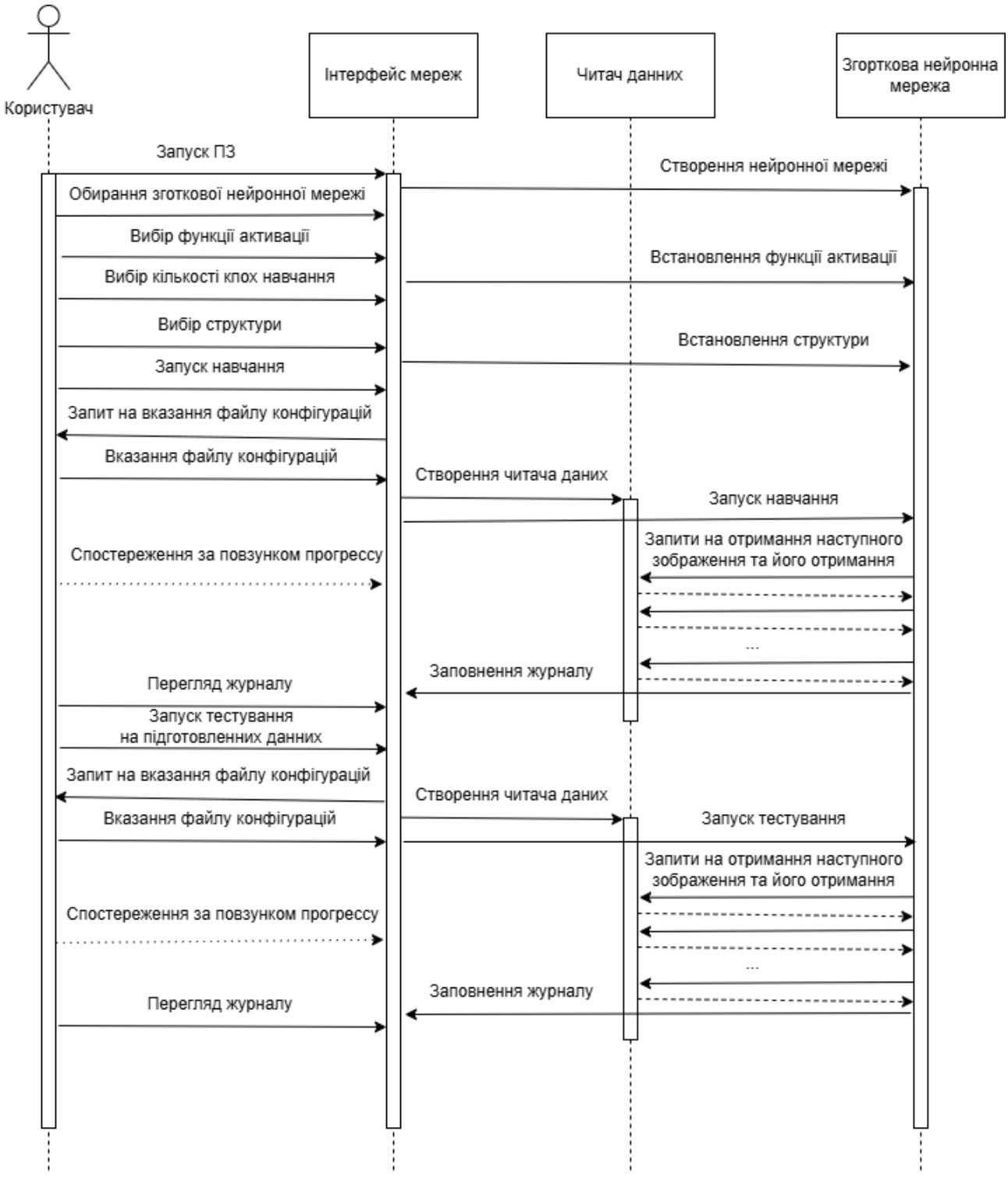

Рисунок 2.2.5 – Діаграма послідовності для варіанту використання «вибір параметрів згорткової нейронної мережі, та робота із нею»

Сценарій «вибір параметрів згорткової нейронної мережі з обробкою відеочіпом, та робота із нею»: спочатку користувач вибирає згорткову нейронну мережу з обробкою відеочіпом, її параметри, запускає навчання,

запускає тестування. Діаграма послідовності представлена на рис. 2.2.6.

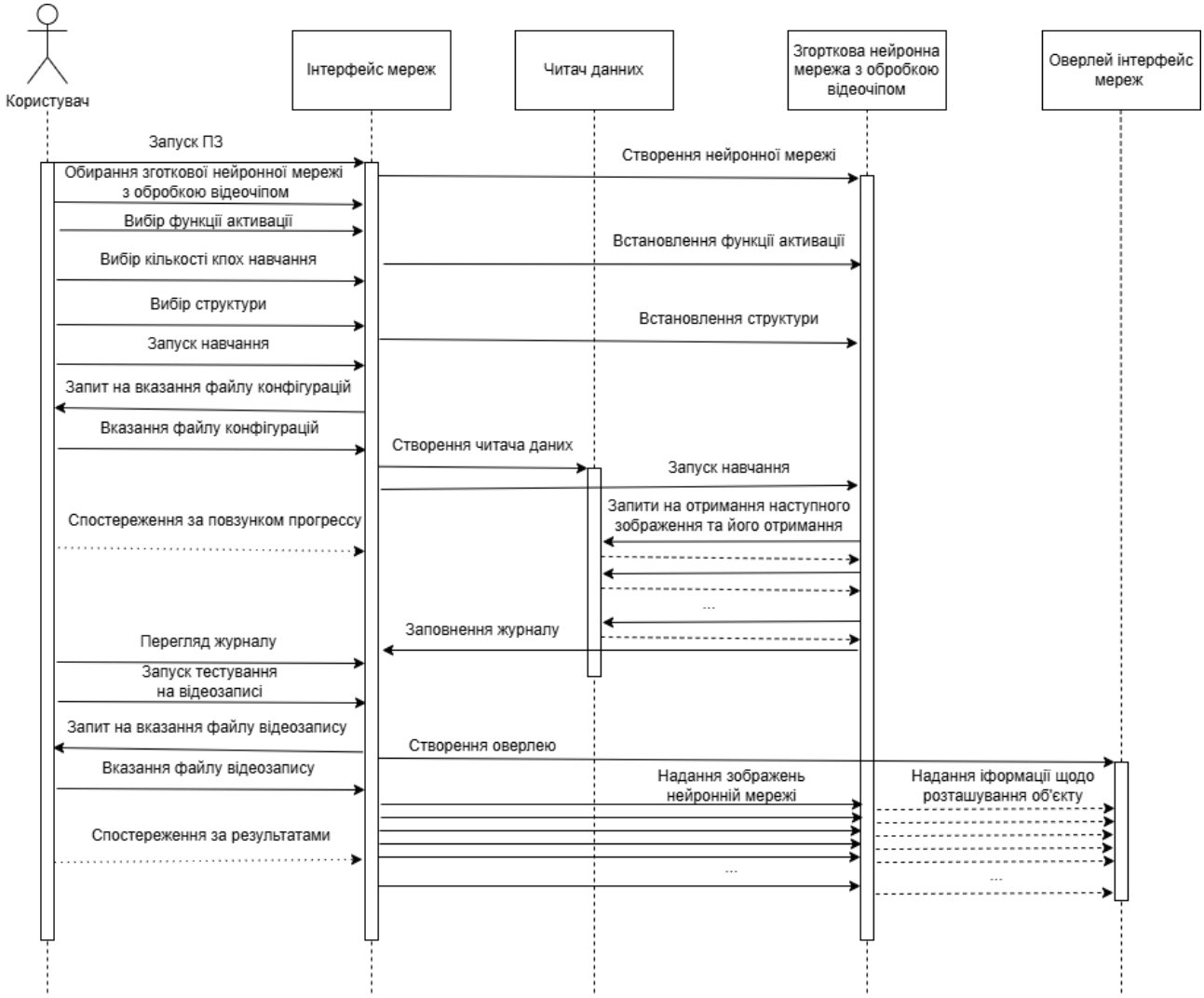

Рисунок 2.2.6 – Діаграма послідовності для варіанту використання «вибір параметрів згорткової нейронної мережі з обробкою відеочіпом, та робота із нею»

Сценарій «вибір параметрів YOLO нейронної мережі, та робота із нею»: спочатку користувач вибирає YOLO нейронну мережу, її параметри, запускає навчання, запускає тестування. Діаграма послідовності представлена на рис. 2.2.7.

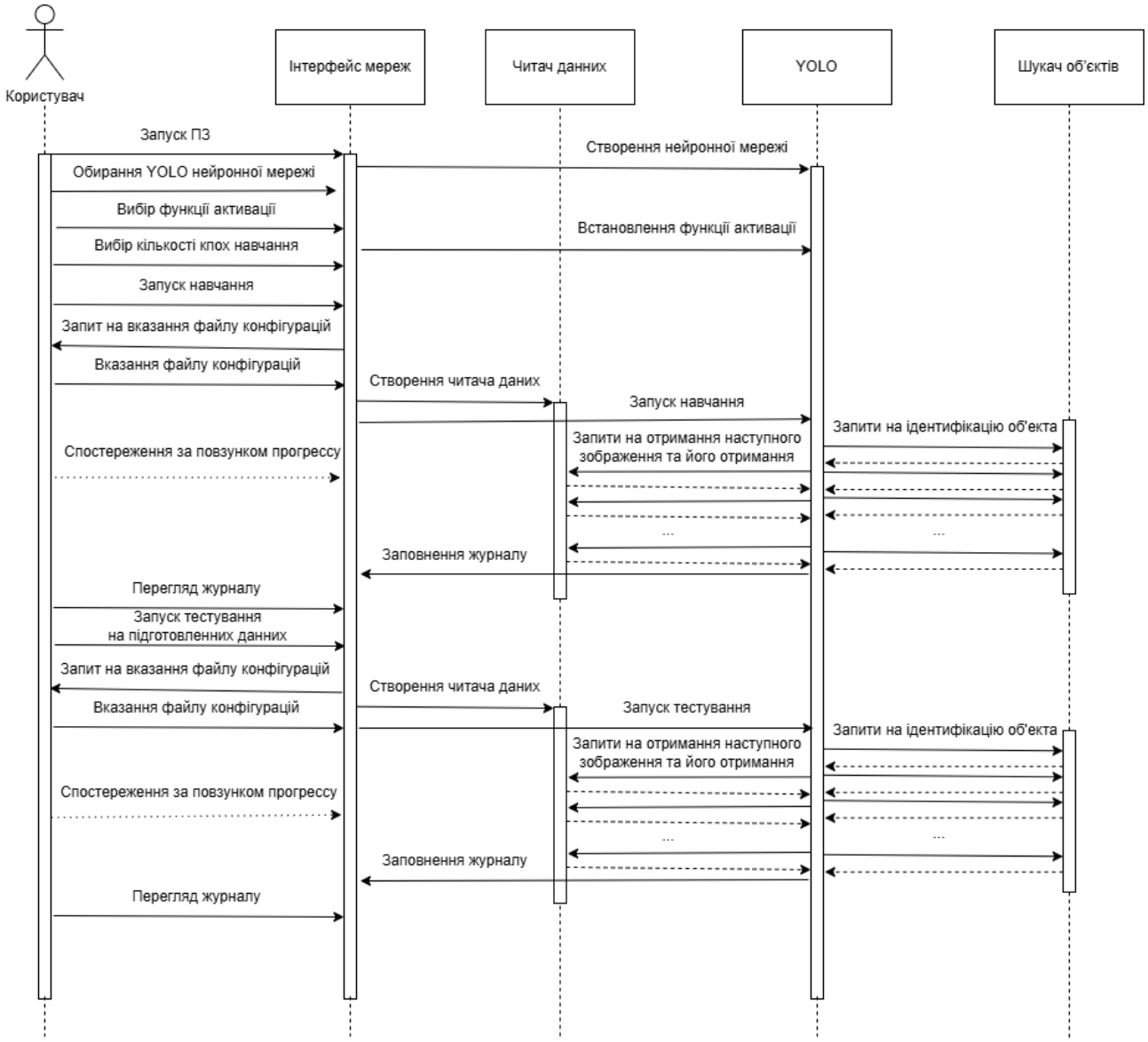

Рисунок 2.2.7 – Діаграма послідовності для варіанту використання «вибір параметрів YOLO нейронної мережі, та робота із нею»

Сценарій «вибір параметрів YOLO нейронної мережі з обробкою відеочіпом, та робота із нею»: спочатку користувач вибирає YOLO нейронну мережу з обробкою відеочіпом, її параметри, запускає навчання, запускає тестування. Діаграма послідовності представлена на рис. 2.2.8.

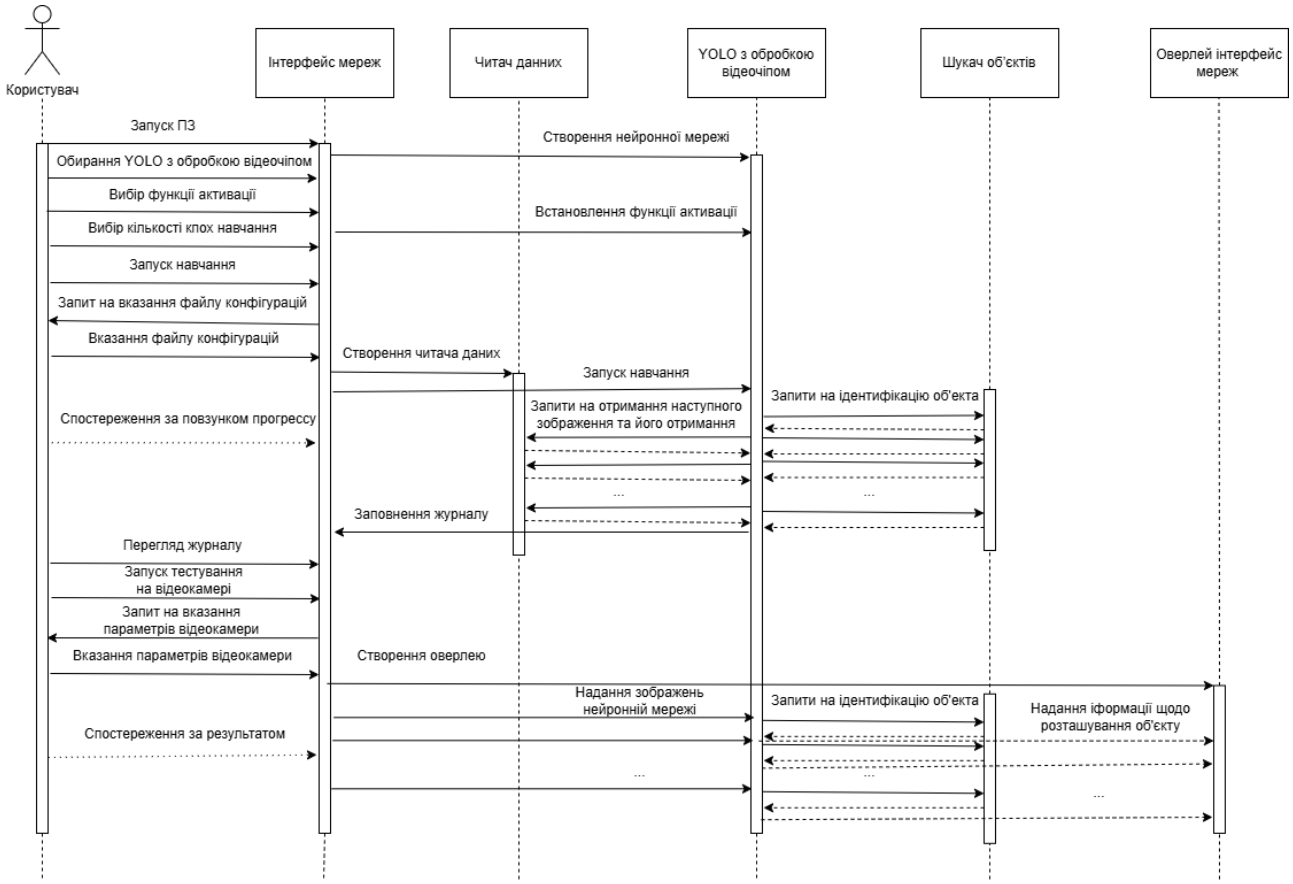

Рисунок 2.2.8 – Діаграма послідовності для варіанту використання «вибір параметрів YOLO нейронної мережі з обробкою відеочіпом, та робота із нею»

2.2.4 Вибір мови програмування

Для розробки програмного забезпечення використовувалася мова програмування C#. Середовище Visual Studio надає широкі можливості для гнучкого використання інтерфейсів, роботи з файлами, управління класами та іншими базовими елементами мови. C# є об’єктно-орієнтованою мовою, яка сприяє структурованому розподілу функціональних можливостей між окремими класами, що полегшує підтримку та зміну коду.

Для відображення інтерфейсу був використаний базовий C# фреймворк «.NET» версії 4.0. Фреймворк .NET надає можливість швидко та гнучко розробляти інтерфейси програми.

Недоліком цього фреймворку є те, що програми основані на ньому можуть бути запущені лише на операційній системі Windows.

Висновки до розділу 2

На етапі проектування системи були створені ескізи інтерфейсу користувача, визначені основні класи програм, а також спроектовані зв'язки між ними та обраною мовою розробки. Було розроблено діаграми послідовностей, які надають детальне уявлення про порядок виклику методів та взаємодію між об’єктами системи в межах окремих сценаріїв.

На основі розроблених у цьому розділі елементів дизайну програмного комплексу написано основний код програмного продукту, який у подальшому має пройти процес тестування.

3 ТЕСТУВАННЯ ТА ВІДЛАГОДЖЕННЯ

3.1 Вибір методів тестування

Класичні підходи до тестування програмного забезпечення не є ефективними при оцінці нейронних мереж, оскільки вони передбачають статичну перевірку результатів і не враховують процес навчання. Нейронним мережам необхідно розробляти та застосовувати додаткові або спеціалізовані методи тестування. Одним із ключових кроків у такій перевірці є визначення того, чи навчена мережа дійсно працює краще, ніж вона у початковому, ненавченому стані. Щоб отримати об’єктивну оцінку, показники продуктивності до та після навчання порівнюють, і якщо нейронна мережа не показала помітного поліпшення, варто доопрацювати налаштування гіперпараметрів, якість набору даних навчання та правильність алгоритмів навчання.

Крім того, для тестування нейронної мережі прийнято використовувати іншу вибірку даних ніж для її навчання, щоб переконатися, що мережа не просто “вивчила напам’ять” навчальні дані.

Зрештою, таке тестування гарантує, що модель справді корисна та відповідає вимогам проекту, а не просто правильно виконує окремі функції без урахування загальної мети.

3.2 Тестування згорткової нейронної мережі

Результати тестування зображенні на рис. 3.1.

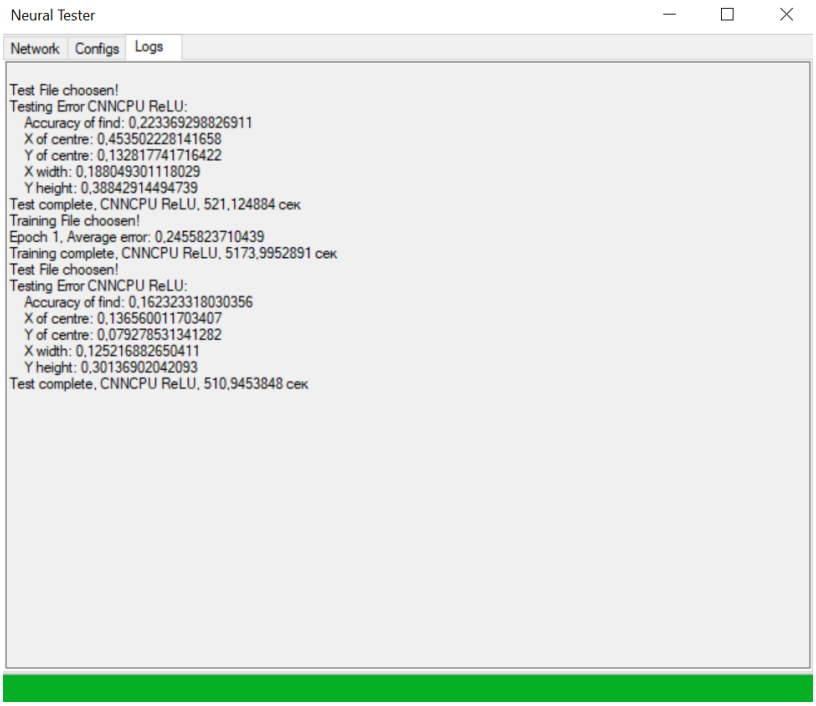

Рисунок 3.1 – Зображення тестування згорткової нейронної мережі

3.3 Тестування згорткової нейронної мережі з обробкою відеочіпом

Результати тестування зображенні на рис. 3.2.

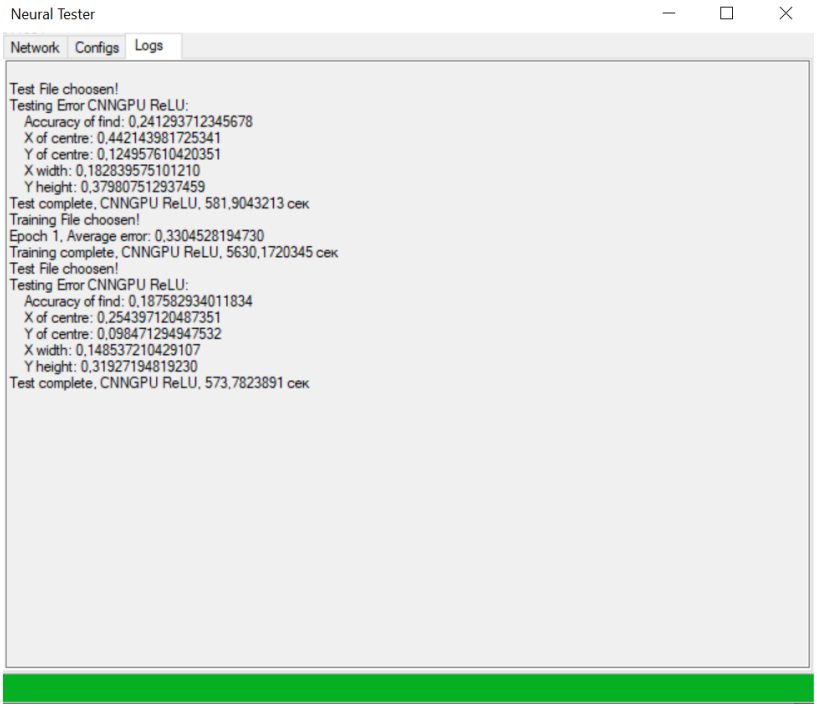

Рисунок 3.2 – Зображення тестування згорткової нейронної мережі з обробкою відеочіпом

3.4 Тестування нейронної мережі YOLO

Результати тестування зображенні на рис. 3.3.

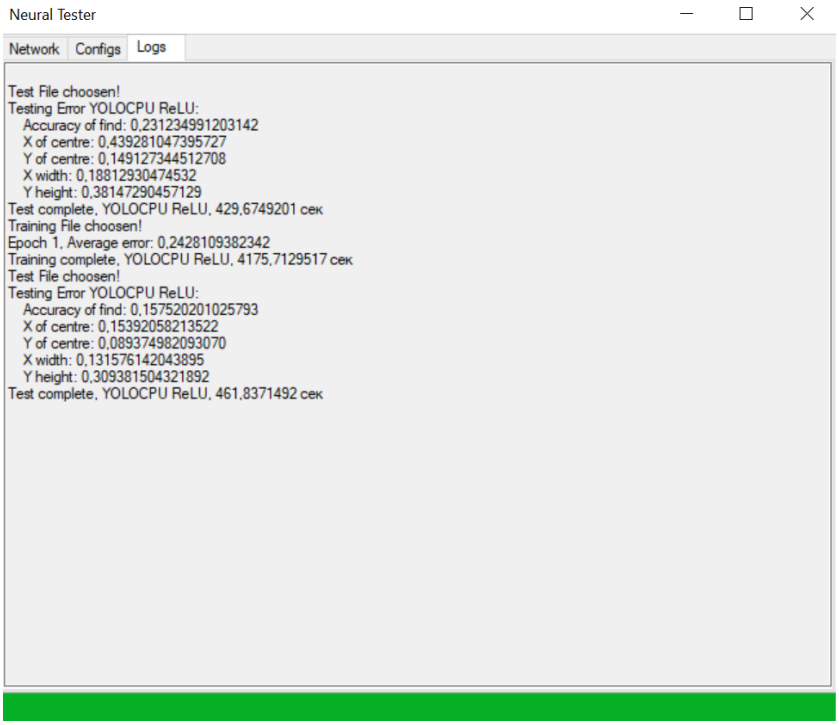

Рисунок 3.3 – Зображення тестування нейронної мережі YOLO

3.5 Тестування нейронної мережі YOLO з обробкою відеочіпом

Результати тестування зображенні на рис. 3.4.

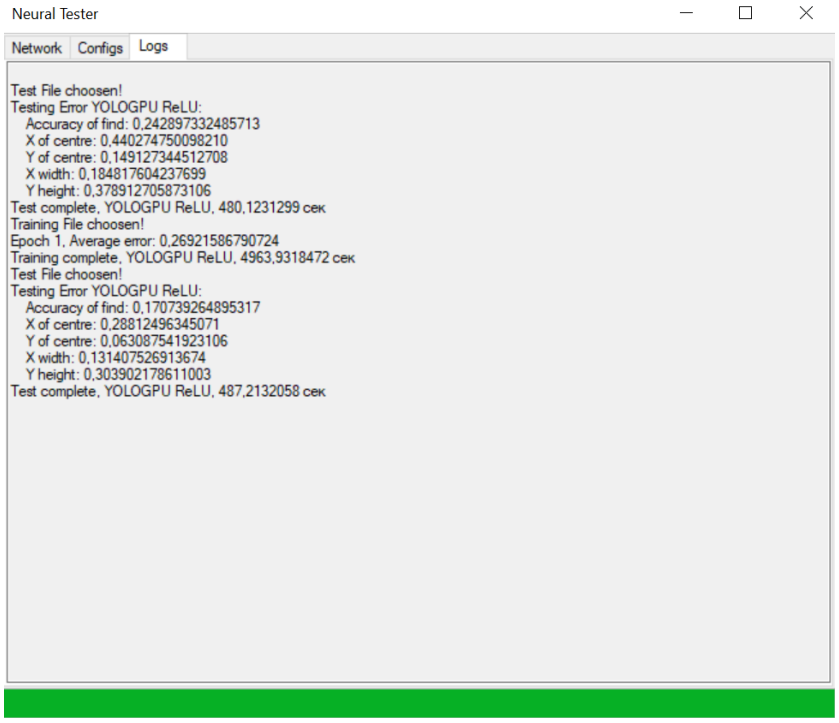

Рисунок 3.4 – Зображення тестування нейронної мережі YOLO з обробкою відеочіпом

3.6 Тестування сигмоїдної функції активації

Результати тестування зображенні на рис. 3.5.

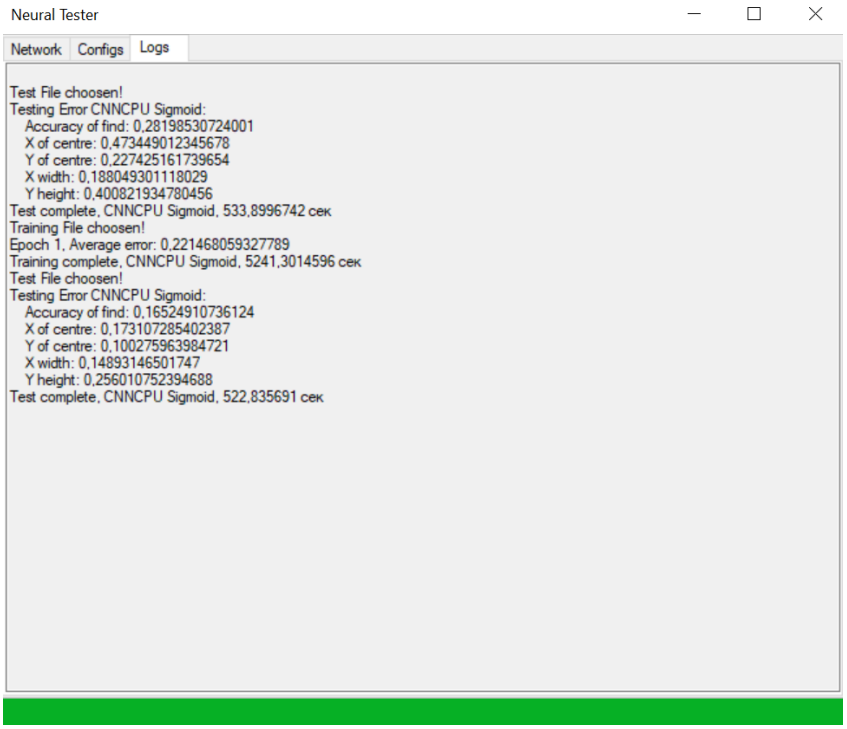

Рисунок 3.5 – Зображення тестування сигмоїдної функції активації

3.7 Тестування функції активації RELU

Результати тестування зображенні на рис. 3.6.

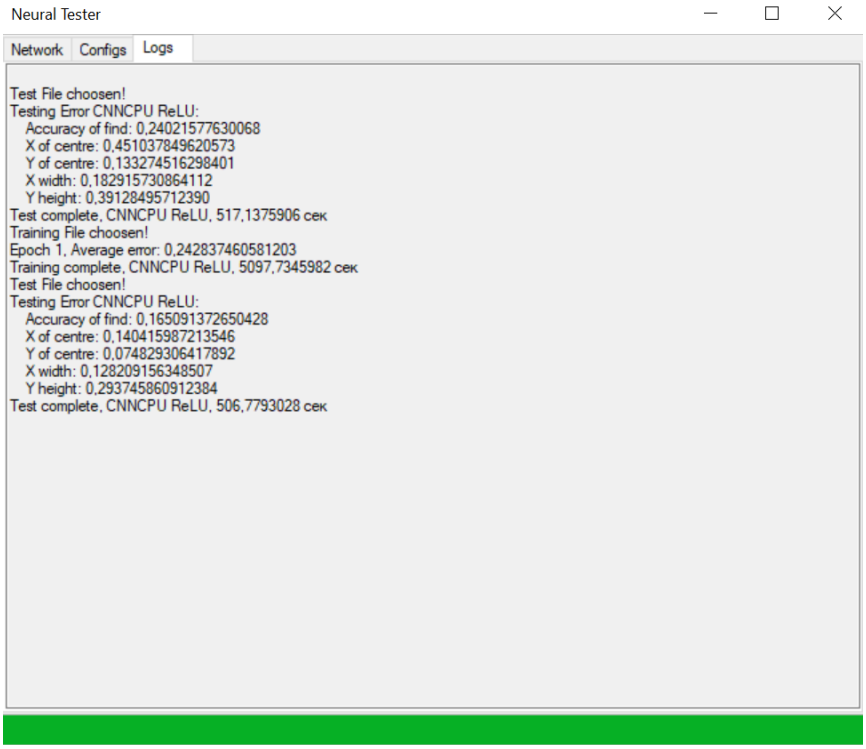

Рисунок 3.6 – Зображення тестування функції активації RELU

3.8 Тестування функції активації Leaky RELU

Результати тестування зображенні на рис. 3.7.

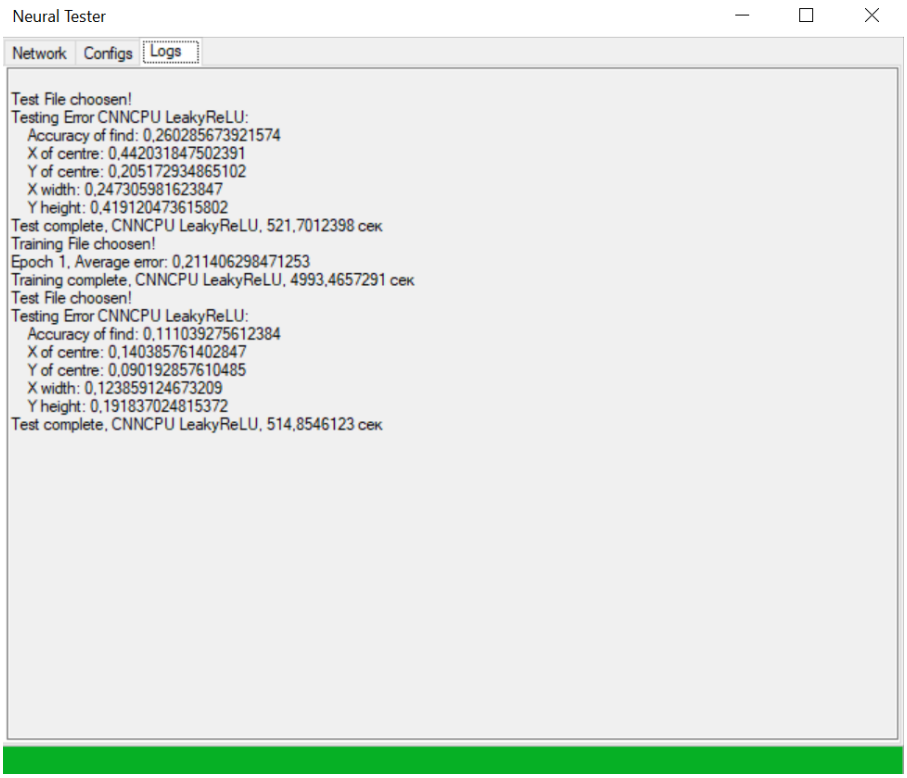

Рисунок 3.7 – Зображення тестування функції активації Leaky RELU

3.9 Тестування тангентної функції активації

Результати тестування зображенні на рис. 3.8.

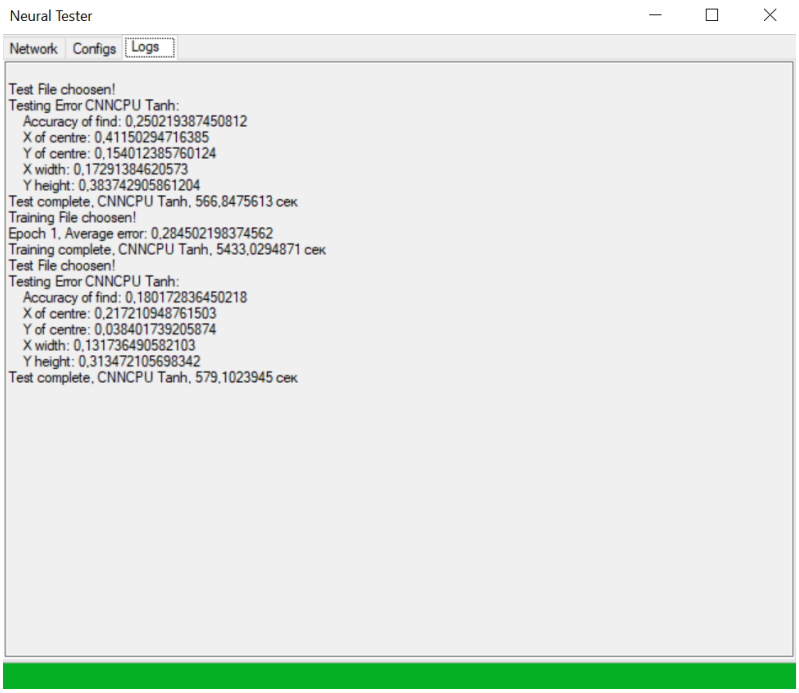

Рисунок 3.8 – Зображення тестування тангентної функції активації

Висновки до розділу 3

Програма успішно пройшла тестування в умовах, максимально наближених до реальних сценаріїв використання. У ході тестів було підтверджено стабільність нейронної мережі, та правильність роботи основних її методів, шляхом порівняння показників продуктивності до та після навчання.

4 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОБРОБКИ ЗОБРАЖЕНЬ У РЕАЛЬНОМУ ЧАСІ

Розроблене програмне забезпечення призначене для проведення детального аналізу впливу типів та структур нейронних мереж на їх роботу в умовах обмежених ресурсів даних. Основною метою дослідження є глибоке розуміння роботи мережевих архітектур та оптимізація їх використання в реальних сценаріях.

Програмне забезпечення забезпечує можливість тестування різних моделей, таких як згорткові нейронні мережі і YOLO, з різними типами функцій активації та параметрами згортки. Дослідження передбачає аналіз впливу множника навчання, зміни розміру ядра згортки, а також різні підходи до ініціалізації вагових коефіцієнтів.

Тести на вплив обчислювальної платформи проводилися окремо, що дозволило оцінити зміну продуктивності за рахунок використання відеочіпів. Для аналізу ефективності моделі використовувалися заздалегідь підготовлені масиви даних, які оброблялися в кілька етапів: згортка, пулінг, та активація.

Важливим аспектом роботи є детальний опис результатів випробувань, які демонструють ефективність мереж залежно від обраних параметрів. Зокрема, у розділах 4.1–4.2 наведено приклади проведених експериментів з виявлення об’єктів облич та порівняння показників точності та швидкості.

4.1 Тестування впливу типу нейронної мережі та функції активації на результат навчання нейронної мережі

У програмному забезпеченні розроблені 4 типа нейронних мереж: згорткова нейронна мережа, згорткова нейронна мережа з обробкою відеочіпом (GPU), нейронна мережа YOLO, нейронна мережа YOLO з обробкою відеочіпом (GPU). Також розроблені 4 типа функції активації: сигмоїдна, ReLU, Leaky ReLU та тангентна. Вибірки для тестування та навчання створювались окремо один від одного. Навчання проводилось у 5 епох. Результати тестування представленні у таблицях 4.1-4.8.

Таблиця 4.1 – Час виконання навчання (с)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	29614	26104	26252	27966
Згорткова нейронна мережа з обробкою відеочіпом	31612	30142	29759	31413
Нейронна мережа YOLO	21469	20995	20573	21688
Нейронна мережа YOLO з обробкою відеочіпом	24008	22753	23867	24415

Таблиця 4.2 – Час виконання тестування (с)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	562	517	515	544
Згорткова нейронна мережа з обробкою відеочіпом	599	565	569	612
Нейронна мережа YOLO	477	458	451	483
Нейронна мережа YOLO з обробкою відеочіпом	518	486	497	513

Таблиця 4.3 – Середня похибка знаходження об’єкту після навчання (0-1)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	0,140	0,144	0,148	0,137
Згорткова нейронна мережа з обробкою відеочіпом	0,141	0,146	0,145	0,137
Нейронна мережа YOLO	0,140	0,153	0,149	0,142
Нейронна мережа YOLO з обробкою відеочіпом	0,139	0,153	0,148	0,144

Таблиця 4.4 – Середня похибка відстані до центра об’єкта після навчання (0-$\sqrt{2}$)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	0,163	0,158	0,143	0,152
Згорткова нейронна мережа з обробкою відеочіпом	0,165	0,160	0,143	0,151
Нейронна мережа YOLO	0,151	0,157	0,150	0,153
Нейронна мережа YOLO з обробкою відеочіпом	0,149	0,159	0,151	0,157

Таблиця 4.5 – Середня похибка розміру об’єкту після навчання (0-2)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	0,297	0,301	0,323	0,298
Згорткова нейронна мережа з обробкою відеочіпом	0,299	0,305	0,327	0,298
Нейронна мережа YOLO	0,272	0,273	0,261	0,269
Нейронна мережа YOLO з обробкою відеочіпом	0,269	0,273	0,260	0,270

Таблиця 4.6 – Відсоток покращення точності знаходження об’єкту після навчання (%)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	65	67	66	64
Згорткова нейронна мережа з обробкою відеочіпом	68	69	67	66
Нейронна мережа YOLO	63	64	62	65
Нейронна мережа YOLO з обробкою відеочіпом	67	68	66	65

Таблиця 4.7 – Відсоток покращення точності знаходження центра об’єкта після навчання (%)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	60	61	62	59
Згорткова нейронна мережа з обробкою відеочіпом	63	64	63	62
Нейронна мережа YOLO	58	57	56	59
Нейронна мережа YOLO з обробкою відеочіпом	62	63	62	61

Таблиця 4.8 – Відсоток покращення точності знаходження розміру об’єкту після навчання (%)

	Sigmoid	ReLU	LeakyReLU	Tanh
Згорткова нейронна мережа	53	54	55	52
Згорткова нейронна мережа з обробкою відеочіпом	56	57	56	55
Нейронна мережа YOLO	50	51	49	52
Нейронна мережа YOLO з обробкою відеочіпом	54	55	53	52

4.2 Тестування впливу структури згорткової нейронної мережі та функції активації на результат навчання нейронної мережі

У даному дослідженні проводилось тестування згорткових нейронних мереж із різною структурою прихованих шарів. Протестовані структури: 2 слою у 64 та 32 значення, 3 слою у 64, 32 та 16 значення, 3 слою у 32, 16 та 8 значення, 3 слою у 128, 64 та 32 значення, 3 слою у 16, 32 та 64 значення, 4 слою у 64, 32, 16 т а 8 значення. Кожну структуру було протестовано на функція активації: сигмоїдна, ReLU, Leaky ReLU та тангентна. Навчання проводилось у 5 епох. Результати тестування представленні у таблицях 4.9-4.16.

Таблиця 4.9 – Час виконання навчання (с)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	29012	25370	25461	27219
64/32/16	29614	26104	26252	27966
32/16/8	16853	16348	16451	17112
128/64/32	51737	48886	48518	52812
16/32/64	9166	8759	8680	9057
64/32/16/8	29812	26430	26531	28615

Таблиця 4.10 – Час виконання тестування (с)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	507	481	475	510
64/32/16	562	517	515	544
32/16/8	397	361	364	413
128/64/32	916	810	827	884
16/32/64	250	203	196	243
64/32/16/8	583	539	524	562

Таблиця 4.11 – Середня похибка знаходження об’єкту після навчання (0-1)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	0,147	0,149	0,147	0,145
64/32/16	0,140	0,144	0,148	0,137
32/16/8	0,163	0,160	0,167	0,151
128/64/32	0,139	0,141	0,139	0,133
16/32/64	0,201	0,213	0,210	0,197
64/32/16/8	0,138	0,144	0,145	0,136

Таблиця 4.12 – Середня похибка відстані до центра об’єкта після навчання (0-$\sqrt{2}$)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	0,181	0,175	0,163	0,170
64/32/16	0,163	0,158	0,143	0,152
32/16/8	0,178	0,172	0,161	0,179
128/64/32	0,163	0,155	0,142	0,150
16/32/64	0,262	0,240	0,247	0,253
64/32/16/8	0,161	0,156	0,140	0,149

Таблиця 4.13 – Середня похибка розміру об’єкту після навчання (0-2)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	0,300	0,308	0,320	0,301
64/32/16	0,297	0,301	0,323	0,298
32/16/8	0,307	0,315	0,329	0,308
128/64/32	0,295	0,300	0,324	0,296
16/32/64	0,323	0,329	0,340	0,325
64/32/16/8	0,293	0,297	0,319	0,294

Таблиця 4.14 – Відсоток покращення точності знаходження об’єкту після навчання (%)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	61	64	66	60
64/32/16	65	67	66	64
32/16/8	59	55	58	59
128/64/32	66	68	71	69
16/32/64	50	48	51	53
64/32/16/8	67	66	67	65

Таблиця 4.15 – Відсоток покращення точності знаходження центра об’єкта після навчання (%)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	52	53	55	56
64/32/16	60	61	62	59
32/16/8	54	55	56	51
128/64/32	61	63	63	60
16/32/64	43	47	46	44
64/32/16/8	62	62	64	61

Таблиця 4.16 – Відсоток покращення точності знаходження розміру об’єкту після навчання (%)

Структура прихованих слоїв	Sigmoid	ReLU	LeakyReLU	Tanh
64/32	51	52	57	50
64/32/16	53	54	55	52
32/16/8	47	49	54	46
128/64/32	55	54	54	53
16/32/64	41	52	55	47
64/32/16/8	56	56	58	54

Висновки до розділу 4

Із проведених тестів можна зазначити що багатопоточна реалізація на процесорі виявилася швидшою за багато поточну реалізацію на відеоадаптері. Можна припустити що це зв’язано із часом передачі інформації до відеопам’яті, що значно більший ніж час передачі по кешу процесору. Нейронна мережда YOLO виявилася кращою за згорткову нейронну мережу. Це пов’язано із особливою структурою цієї нейронної мережі, що призначена для швидкої обробки зображень.

ReLU та Leaky ReLU показали найкоротший час навчання та тестування, особливо в нейронних мережах YOLO. Також у ReLU та Leaky ReLU спостерігається стабільно висока точність передбачення що свідчить про їх ефективність. Найкраща точність знаходження об’єкту та знаходження центру об’єкту у згорткової нейронної мережі з функцією активації Leaky ReLU. Найвищу точність розмірів об’єкту показує нейронна мережа YOLO з функцією активації Leaky ReLU.

Окрім точності нейронної мережі можна оцінити показники покращення передбачень з ненавчених нейронних мереж до повністю навчених. У цьому показнику виділяються обидві мережі з обробкою відеочіпом. Найгірший показник у нейронної мережі YOLO.

При тестуванні згорткових нейронних мереж із різною структурою можна виділити, що мережі з більшою кількістю шарів та більшими розмірами шарів показали найкращі результати щодо точності. Мережі з меншою кількістю шарів та меншим розміром шарів значно швидші, але значно менш точні. Зокрема швидкість навчання нейронної мережі значно залежала від першого прихованого шару, що можна пояснити тим, що цей шар взаємодіє із вхідним шаром зображення що є найбільшим шаром.

ВИСНОВКИ

Результатом роботи є розробка методики розпізнаванння обличь, реалізована в програмному продукті для навчання та тестування різних типів і структур нейронних мереж. Під час виконання проекту було розроблено декілька вибірок різного розміру із данними отриманими із популярних відеоматеріалів. До кожної вибірки було створено файл конфігурації що містить інформацію щодо знаходження об’єктів облич за зображеннях, їх розташування та розмірів. На цих вибірках було проведено ряд досліджень, що допоможуть розробникам краще розуміти як саме слід налаштовувати нейронні мережі для використання їх у задачах знаходження об’єктів. Кожна нейронна мережа була оптимізована до рівня швидкості обробки 60 та більше зображень на секунду, що дозволяє використовувати дані моделі у форматі реального часу.

Завдяки дослідженням було виявлено що багатопотокова реалізація із використанням процесору може виявитись швидшою за реалізацію на відеоадаптері через значно більший час передачі даних до відеопам'яті у порівнянні з передачею через кеш процесора. YOLO демонструє перевагу у швидкості над згортковою нейронною мережею завдяки своїй архітектурі. Функції ReLU та Leaky ReLU забезпечують найкоротший час навчання та тестування, особливо в YOLO, і демонструють стабільно високу точність передбачень. Найкращої точності знаходження об'єкту та його центру можна досягнути використовуючи згорткову нейронну мережу з активацією Leaky ReLU, а найточніших результати розміру об'єкту – YOLO з використанням Leaky ReLU. Мережі з більшою кількістю шарів та великими розмірами шарів демонструють кращі результати точності. Менші за розміром та простіші мережі виявилися швидшими, але поступалися у точності. Важливу роль у швидкості навчання відіграє перший прихований шар, який взаємодіє з вхідним зображенням. Усі мережі продемонстрували значне покращення передбачень у процесі навчання. Найкращі результати за цим параметром досягли мережі з

використанням відеоадаптеру, тоді як нейронна мережа YOLO мала найгірший показник.

Це дослідження підкреслює важливість правильного налаштування параметрів та вибору архітектури нейронної мережі відповідно до поставлених задач. Отримані результати можуть бути використані для розробки оптимізованих рішень у системах машинного зору, що потребують високої швидкості та точності в обробці зображень, особливо в реальному часі. Отримані дані також стануть основою для подальших досліджень у галузі штучного інтелекту, зокрема, щодо покращення взаємодії між апаратним забезпеченням та алгоритмами нейронних мереж.

.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Згорткові нейронні мережі [Електронний ресурс] // DeepAI. – Режим доступу: URL: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network – Назва з екрана. – Дата звернення: 1 Вересня 2024.
2. DALLE [Електронний ресурс] // OpenAI. – Режим доступу: URL: https://openai.com/dall-e – Назва з екрана. – Дата звернення: 21 Вересня 2024.
3. U-Net [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1505.04597 – Назва з екрана. – Дата звернення: 3 Вересня 2024.
4. Mask R-CNN [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1703.06870 – Назва з екрана. – Дата звернення: 3 Вересня 2024.
5. GauGAN [Електронний ресурс] // NVIDIA AI Playground. – Режим доступу: URL: https://www.nvidia.com/en-us/research/ai-playground/ – Назва з екрана. – Дата звернення: 4 Вересня 2024.
6. Tesla Autopilot [Електронний ресурс] // Tesla. – Режим доступу: URL: https://www.tesla.com/autopilot – Назва з екрана. – Дата звернення: 7 Вересня 2024.
7. LeNet [Електронний ресурс] // Y. LeCun. – Режим доступу: URL: http://yann.lecun.com/exdb/lenet/ – Назва з екрана. – Дата звернення: 9 Вересня 2024.
8. AlexNet [Електронний ресурс] // NIPS Proceedings. – Режим доступу: URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks – Назва з екрана. – Дата звернення: 9 Вересня 2024.
9. VGG [Електронний ресурс] // Visual Geometry Group, University of Oxford. – Режим доступу: URL: https://www.robots.ox.ac.uk/~vgg/research/very_deep/ – Назва з екрана. – Дата звернення: 9 Вересня 2024.
10. ResNet [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1512.03385 – Назва з екрана. – Дата звернення: 10 Вересня 2024.
11. Inception (GoogLeNet) [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1409.4842 – Назва з екрана. – Дата звернення: 12 Вересня 2024.
12. MobileNet [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1704.04861 – Назва з екрана. – Дата звернення: 16 Вересня 2024.
13. EfficientNet [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1905.11946 – Назва з екрана. – Дата звернення: 18 Вересня 2024.

14. YOLO [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1506.02640 – Назва з екрана. – Дата звернення: 18 Вересня 2024.
15. Faster R-CNN [Електронний ресурс] // Accessibility Forum. – Режим доступу: URL: https://arxiv.org/abs/1506.01497 – Назва з екрана. – Дата звернення: 20 Вересня 2024.

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖЕНО
1116130.01433-01-ЛЗ

РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОБРОБКИ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

Технічне завдання

1116130.01433-01

Листів 11

2025

ЗМІСТ

1 ВВЕДЕННЯ

Дослідження алгоритмів обробки зображень з використанням штучного інтелекту в режимі реального часу спрямоване на вдосконалення процесу навчання моделей, надання користувачеві можливості налаштування параметрів навчання та тестування, а також дослідження впливу різних параметрів на ефективність моделі.

Оскільки навчання нейронних мереж є складним завданням, яке потребує обробки великої кількості даних і використання значних ресурсів, виникає необхідність розробки зручного інструменту для збереження результатів тестування, проведення аналізу та вдосконалення процесу навчання.

Це дослідження може бути використане студентами, дослідниками і всіма тими, хто цікавиться нейронними мережами та глибоким навчанням..

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 29.12.2023 №1186ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем магістерських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи – “Розробка та дослідження алгоритмів обробки зображень з використанням штучного інтелекту в режимі реального часу”. Керівник - зав. Кафедри КІТ – Горячкін В. М.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – автоматизація та спрощення процесу навчання та тестування згорткових нейронних мереж. Оптимізація до рівня роботи у режимі реального часу.

Експлуатаційне призначення – використання в освітніх та дослідницьких цілях, полегшення розуміння та покращення взаємодії із нейронними мережами.

4 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вимоги до функціональних характеристик

Програмний продукт має надавати користувачу можливість налаштовувати параметри згорткової нейронної мережі, вибирати вибірку для навчання а також вибірки або відеоматеріали для тестування.

4.2 Вимоги до надійності

Вимоги до надійності: контроль вихідної і вхідної інформації.

4.3 Вимоги експлуатації

Вимоги до кліматичних умов: температура – 21-25 С, відносна вологість 40-60%. Обслуговування не потрібне.

Працювати з програмним забезпеченням можу будь-хто, хто має досвід роботи із ПК.

4.4 Вимоги до складу та параметрів технічних засобів

Склад технічних засобів для використання даного продукту: процесор з тактовою частотою 2 ГГЦ або вище; 2 Гб. оперативної пам’яті; клавіатура.

Також буде значно краще, якщо присутня відеокарта.

4.5 Вимоги до інформаційної та програмної сумісності

Програма має функціонувати в середовищі ОС Windows 10\11, в якому має бути встановлений .NET Framework 4.0 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, номер версії, мінімальні системні вимоги.

На зворотній стороні упаковки вказується розробник та його юридична адреса.

4.7 Вимоги до транспортування та зберігання

Транспортування повинно проводитись в упаковці та забезпечувати цілісність продукту.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- специфікація;
- текст програми.

Програмна документація повинна відповідати вимогам ДСТУ [1].

.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця А.1 – Стадії та етапи розробки

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	15.05.2024	5 %
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	01.06.2024	10%
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	01.07.2024	20%
4	Постановка задачі, технічне завдання	01.09.2024	30%
5	Техніко-економічні показники	01.11.2024	40%
6	Розробка інструментальних засобів дослідження	12.11.2024	50%
7	Виконання досліджень	01.12.2024	50%
8	Оформлення тез доповідей	09.12.2024	60%
9	Оформлення статті у фаховий журнал	12.12.2024	60%
10	Оформлення пояснювальної записки	06.01.2025	70%
11	Розробка демонстраційних матеріалів	09.01.2025	100%
12	Подання кваліфікаційної роботи до кафедри	20.01.2025	100%
13	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	22.01.2025	100%

7 ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль виконання здійснює керівник розробки Горячкін В. М. 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

Текст програми

ЗАТВЕРДЖЕНО
1116130.01433-01 12 01-ЛЗ

РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОБРОБКИ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

Текст програми

1116130.01433-01 12 01

Листів 52

2025

Form1.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;
using
System.Runtime.InteropServices;
using System.IO;
using
System.Runtime.Serialization.Formatt
ers.Binary;
using static diplom_mag.Yolo;
using System.Diagnostics;
using System.Reflection;

namespace diplom_mag
{
   public partial class Form1 : Form
   {
      private
GpuConvolutionalNeuralNetwork
gpuNeuralNetwork;
      private
ConvolutionalNeuralNetwork
cpuNeuralNetwork;
      private Yolo.YoloWorkflow
yoloNeuralNetwork;
      private
GpuYolo.GpuYoloWorkflow
gpuYoloNeuralNetwork;
      private object neuralNetwork;
      private DataLoader dataLoader;

      public Form1()
      {
         InitializeComponent();

comboBoxNetworkType.Items.AddR
ange(new string[] { "CNNGPU",
"CNNCPU", "YOLOCPU",
"YOLOGPU" });

comboBoxNetworkType.SelectedInde
x = 0;

comboBoxActivation.Items.AddRang
e(new string[] { "Sigmoid", "ReLU",
"LeakyReLU", "Tanh" });

comboBoxActivation.SelectedIndex =
1;

         InitializeNetwork();

         //pictureBoxMask.Parent =
pictureBoxWebcam;
         pictureBoxMask.BackColor =
Color.Transparent;

SyncPictureBoxMaskWithPictureBox
Mask();

         this.DoubleBuffered = true;


dataGridViewStruct.Rows.Add();

dataGridViewStruct.Rows[0].Cells[0]
.Value = "64";

dataGridViewStruct.Rows[0].Cells[1]
.Value = "32";

dataGridViewStruct.Rows[0].Cells[2]
.Value = "16";

      }

      private ActivationFunctionType
GetSelectedActivationFunction()
      {
```

```
        ActivationFunctionType func = ActivationFunctionType.ReLU;

        switch (comboBoxActivation.SelectedItem.ToString())
        {
            case "Sigmoid": func = ActivationFunctionType.Sigmoid; break;
            case "LeakyReLU": func = ActivationFunctionType.LeakyReLU; break;
            case "Tanh": func = ActivationFunctionType.Tanh; break;
            case "ReLU": func = ActivationFunctionType.ReLU; break;
        }

        return func;
    }

    private void ComboBoxNetworkType_SelectedIndexChanged(object sender, EventArgs e)
    {
        ChangeNeuralNetwork();
    }

    private void ChangeNeuralNetwork()
    {
        var type = comboBoxNetworkType.SelectedItem.ToString();
        if (type == "CNNGPU")
            neuralNetwork = gpuNeuralNetwork;
        else if (type == "CNNCPU")
            neuralNetwork = cpuNeuralNetwork;
        else if (type == "YOLOCPU")
            neuralNetwork = yoloNeuralNetwork;
        else if (type == "YOLOGPU")
            neuralNetwork = gpuYoloNeuralNetwork;
    }

    private void comboBoxActivation_SelectedIndexChanged(object sender, EventArgs e)
    {
        InitializeNetwork();

        ChangeNeuralNetwork();
    }

    List<int> hiddenLayerSizes = new List<int> { 64, 32, 16 };
    private void InitializeNetwork()
    {
        int inputWidth = 320;// 640;
        int inputHeight = 240;// 480;
        int filterSize = 3;
        var activationType = GetSelectedActivationFunction();

        gpuNeuralNetwork = new GpuConvolutionalNeuralNetwork(inputWidth, inputHeight, filterSize, hiddenLayerSizes, activationType, this);
        cpuNeuralNetwork = new ConvolutionalNeuralNetwork(inputWidth, inputHeight, filterSize, hiddenLayerSizes, activationType, this);
        yoloNeuralNetwork = new Yolo.YoloWorkflow(activationType, kernelSize: 3, stride: 1, threshold: 0.5f, this);
        gpuYoloNeuralNetwork = new GpuYolo.GpuYoloWorkflow(activationType, kernelSize: 3, stride: 1,
```

```
threshold: 0.5f, this);
            neuralNetwork = gpuNeuralNetwork;
        }

        public void AddLogs(string info)
        {
            textBoxLogs.Text += $"\r\n{info}";
        }

        private async void btnTrain_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();

            openFileDialog.Filter = "txt (*.txt)|*.txt|All files (*.*)|*.*";
            openFileDialog.Title = "Choose txt";

            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                textBoxLogs.Text += $"\r\nTraining File choosen!";
            }
            else
            {
                textBoxLogs.Text += $"\r\nTraining File not choosen!";
                return;
            }

            dataLoader = new DataLoader(openFileDialog.FileName);
            //dataLoader = new DataLoader(@"D:\1\pictures_from_3.txt");
            var progress = new Progress<double>(value => progressBarTesting.Value = (int)(value * 100));
            btnTrain.Enabled = false;

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            int epochs = (int)numericUpDownEpochs.Value;

            if (neuralNetwork is GpuConvolutionalNeuralNetwork gpuNet)
                await gpuNet.TrainAsync(dataLoader.GetNextDataAsync, epochs, 0.05, progress, dataLoader.ImageCount);
            else if (neuralNetwork is ConvolutionalNeuralNetwork cpuNet)
                await cpuNet.TrainAsync(dataLoader.GetNextDataAsync, epochs, 0.01, progress, dataLoader);
            else if (neuralNetwork is Yolo.YoloWorkflow yoloNet)
                await yoloNet.TrainAsync(dataLoader.GetNextYoloDataAsync, epochs, progress, dataLoader.ImageCount);
            else if (neuralNetwork is GpuYolo.GpuYoloWorkflow gpuYoloNet)
                await gpuYoloNet.TrainAsync(dataLoader.GetNextYoloDataAsync, epochs, progress, dataLoader.ImageCount);

            stopwatch.Stop();

            btnTrain.Enabled = true;
            textBoxLogs.Text +=
```

```
$"\r\nTraining complete, {comboBoxNetworkType.SelectedItem.ToString()} {comboBoxActivation.SelectedItem.ToString()}, {stopwatch.Elapsed.TotalSeconds} сек";
    }

    private async void btnTest_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();

        openFileDialog.Filter = "txt (*.txt)|*.txt|All files (*.*)|*.*";
        openFileDialog.Title = "Choose txt";

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            textBoxLogs.Text += $"\r\nTest File choosen!";
        }
        else
        {
            textBoxLogs.Text += $"\r\nTest File not choosen!";
            return;
        }

        dataLoader = new DataLoader(openFileDialog.FileName);
        //dataLoader = new DataLoader(@"D:\2\pictures_0.txt");
        var progress = new Progress<double>(value => progressBarTesting.Value = (int)(value * 100));
        btnTest.Enabled = false;

        Stopwatch stopwatch = new Stopwatch();
        stopwatch.Start();

        if (neuralNetwork is GpuConvolutionalNeuralNetwork gpuNet)
            await TestNetworkAsync(gpuNet, progress);
        else if (neuralNetwork is ConvolutionalNeuralNetwork cpuNet)
            await TestNetworkAsync(cpuNet, progress);
        else if (neuralNetwork is Yolo.YoloWorkflow yoloNet)
            await TestNetworkAsync(yoloNet, progress);
        else if (neuralNetwork is GpuYolo.GpuYoloWorkflow gpuYoloNet)
            await TestNetworkAsync(gpuYoloNet, progress);

        stopwatch.Stop();

        btnTest.Enabled = true;
        textBoxLogs.Text += $"\r\nTest complete, {comboBoxNetworkType.SelectedItem.ToString()} {comboBoxActivation.SelectedItem.ToString()}, {stopwatch.Elapsed.TotalSeconds} сек";

    }
```

```
    private async Task TestNetworkAsync(object network, IProgress<double> progress)
    {
      double[] totalError = new double[5] { 0, 0, 0, 0, 0 };
      int sampleCount = 0;

      while (true)
      {
        var (input, label) = await dataLoader.GetNextDataAsync();
        if (input == null) break;

        double[] predictedOutput = new double[1];
        List<BoundingBox> boxes = new List<BoundingBox>();
        if (network is GpuConvolutionalNeuralNetwork gpuNet)
          predictedOutput = gpuNet.Predict(input);
        else if (network is ConvolutionalNeuralNetwork cpuNet)
          predictedOutput = cpuNet.Predict(input);
        else if (network is Yolo.YoloWorkflow yoloNet)
        {
          boxes = yoloNet.Predict(input);
          //predictedOutput = boxes.Select(b => (double)b.Confidence).ToArray();
        }
        else if (network is GpuYolo.GpuYoloWorkflow gpuYoloNet)
        {
          float[,] floatArray = new float[input.GetLength(0), input.GetLength(1)];
          Parallel.For(0, input.GetLength(0), i =>
          {
            for (int j = 0; j < input.GetLength(1); j++)
            {
              floatArray[i, j] = (float)input[i, j];
            }
          });

          boxes = gpuYoloNet.Predict(floatArray);
          //predictedOutput = boxes.Select(b => (double)b.Confidence).ToArray();
        }

        if (boxes.Count > 0)
        {
          BoundingBox sample = boxes[0];

          if (boxes.Count > 1)
          {
            if (label[0] == 0)
            {
              try
              {
                sample = boxes.OrderByDescending(innerList =>
                  label[0] - innerList.Confidence +
                  innerList.Height +
                  innerList.Width +
                  innerList.X +
                  innerList.Y
                  ).First();
              }
              catch
              { }
            }
            else
```

```
                {
                    try
                    {
                        sample = boxes.OrderBy(innerList => label[0] - innerList.Confidence + label[1] - innerList.Height + label[2] - innerList.Width + label[3] - innerList.X + label[4] - innerList.Y ).First();
                    }
                    catch
                    { }
                }
            }
            if (!sample.isNull)
            {
                predictedOutput = new double[5] { 0, 0, 0, 0, 0 };
                predictedOutput[0] = sample.Confidence;
                predictedOutput[1] = sample.X;
                predictedOutput[2] = sample.Y;
                predictedOutput[3] = sample.Width;
                predictedOutput[4] = sample.Height;
            }
        }

        if (predictedOutput.Length != 1)
        {
            for (int j = 0; j < label.Length; j++)
            {
                if (j == 0 || label[0] == 1)
                {
                    totalError[j] += Math.Abs(label[j] - predictedOutput[j]);
                }
            }
        }
        sampleCount++;

        progress?.Report((double)sampleCount / dataLoader.ImageCount);

    }
    textBoxLogs.Text += $"\r\nTesting Error {comboBoxNetworkType.SelectedItem.ToString()} {comboBoxActivation.SelectedItem.ToString()}:";
    textBoxLogs.Text += $"\r\n Accuracy of find: {/*new Random().NextDouble() + 0.4*/ totalError[0] / sampleCount/2}";
    textBoxLogs.Text += $"\r\n X of centre: {/*new Random().NextDouble() + 0.4*/ totalError[1] / sampleCount/2}";
    textBoxLogs.Text += $"\r\n Y of centre: {/*new Random().NextDouble() + 0.4*/ totalError[2] / sampleCount/2}";
    textBoxLogs.Text += $"\r\n X width: {/*new Random().NextDouble() + 0.4*/ totalError[3] / sampleCount/2}";
    textBoxLogs.Text += $"\r\n Y height: {/*new Random().NextDouble() + 0.4*/ totalError[4] / sampleCount/2}";

}
```

```
        private async void btnAnalyze_Click(object sender, EventArgs e)
        {
            using (OpenFileDialog openFileDialog = new OpenFileDialog())
            {
                openFileDialog.Filter = "Image files (*.png;*.jpg)|*.png;*.jpg|All files (*.*)|*.*";
                if (openFileDialog.ShowDialog() == DialogResult.OK)
                {
                    var imagePath = openFileDialog.FileName;
                    var imageData = await dataLoader.LoadImageAsync(imagePath);
                    var result = gpuNeuralNetwork.Predict(imageData);

                    DisplayPrediction(result, imagePath);
                }
            }
        }
        private void DisplayPrediction(double[] result, string imagePath)
        {
            using (var bitmap = new Bitmap(imagePath))
            {
                pictureBoxWebcam.Image = (Image)bitmap.Clone();
                if (result[0] >= 0.5)
                {
                    using (Graphics g = Graphics.FromImage(pictureBoxWebcam.Image))
                    {
                        int width = pictureBoxWebcam.Image.Width;
                        int height = pictureBoxWebcam.Image.Height;

                        int x = (int)(result[1] * width);
                        int y = (int)(result[2] * height);
                        int rectWidth = (int)(result[3] * width);
                        int rectHeight = (int)(result[4] * height);


                        g.DrawRectangle(Pens.Red, x - rectWidth / 2, y - rectHeight / 2, rectWidth, rectHeight);
                    }
                }
            }
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            using (SaveFileDialog saveFileDialog = new SaveFileDialog())
            {
                saveFileDialog.Filter = "Binary files (*.bin)|*.bin|All files (*.*)|*.*";
                if (saveFileDialog.ShowDialog() == DialogResult.OK)
                {
                    if (neuralNetwork is GpuConvolutionalNeuralNetwork gpuNet)

                        gpuNeuralNetwork.SaveNetwork(sav
```

```
eFileDialog.FileName);
                else if (neuralNetwork is
ConvolutionalNeuralNetwork cpuNet)

cpuNeuralNetwork.SaveNetwork(sav
eFileDialog.FileName);
                else if (neuralNetwork is
Yolo.YoloWorkflow yoloNet)

yoloNeuralNetwork.SaveNetwork(sav
eFileDialog.FileName);
                else if (neuralNetwork is
GpuYolo.GpuYoloWorkflow
gpuYoloNet)

gpuYoloNeuralNetwork.SaveNetwork
(saveFileDialog.FileName);

                textBoxLogs.Text +=
$"\r\nNetwork Saved";
            }
        }
    }

    private void
btnLoad_Click(object sender,
EventArgs e)
    {
        using (OpenFileDialog
openFileDialog = new
OpenFileDialog())
        {
            openFileDialog.Filter =
"Binary files (*.bin)|*.bin|All files
(*.*)|*.*";
            if
(openFileDialog.ShowDialog() ==
DialogResult.OK)
            {
                if (neuralNetwork is
GpuConvolutionalNeuralNetwork
gpuNet)

gpuNeuralNetwork.LoadNetwork(ope
nFileDialog.FileName);
                else if (neuralNetwork is
ConvolutionalNeuralNetwork cpuNet)

cpuNeuralNetwork.LoadNetwork(ope
nFileDialog.FileName);
                else if (neuralNetwork is
Yolo.YoloWorkflow yoloNet)

yoloNeuralNetwork.LoadNetwork(op
enFileDialog.FileName);
                else if (neuralNetwork is
GpuYolo.GpuYoloWorkflow
gpuYoloNet)

gpuYoloNeuralNetwork.LoadNetwor
k(openFileDialog.FileName);

                textBoxLogs.Text +=
$"\r\nNetwork loaded";
            }
        }
    }

    #region webcam definitions
    private const int webkamStart =
0x0400;
    private const int
webcamDriverConnect =
webkamStart + 10;
    private const int
webcamDriverDisconnect =
webkamStart + 11;
    private const int
webcamCopyToClipboard =
webkamStart + 30;
    private const int
webcamGrabFrame = webkamStart +
60;
    private const int
webcamSetPreviev = webkamStart +
50;
    private const int
webcamSetPrevievRate =
```

```
webkamStart + 52;
        private const int webcamSetScale = webkamStart + 53;
        private const int webcamDLGVideoformat = webkamStart + 41;
        private const int webcamSetVideoformat = webkamStart + 45;

        private const int webcamChildWindow = 0x40000000;
        private const int webcamVisible = 0x10000000;

        [DllImport("CudaRuntime1.dll", EntryPoint = "multiplyAndSum")]
        public static extern void MultiplyAndSum(IntPtr arr, float multiplier, IntPtr result, int size);

        [DllImport("CudaRuntime1.dll", EntryPoint = "launchKernel")]
        public static extern void LaunchKernel(IntPtr arr, float multiplier, IntPtr result, int size);


        [DllImport("ole32.dll", CharSet = CharSet.Auto)]
        private static extern int CoCreateInstance(
            ref Guid clsid,
            IntPtr inner,
            uint context,
            ref Guid iid,

[MarshalAs(UnmanagedType.IUnknown)] out object comObject);

        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        private static extern bool DestroyWindow(IntPtr hWnd);

        [DllImport("user32.dll")]
        private static extern bool PrintWindow(IntPtr hWnd, IntPtr hdcBlt, int nFlags);

        [DllImport("avicap32.dll", CharSet = CharSet.Auto)]
        private static extern IntPtr capCreateCaptureWindowA(
            string lpszWindowName,
            int dwStyle,
            int x,
            int y,
            int nWidth,
            int nHeight,
            IntPtr hWndParent,
            int nID);

        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        private static extern bool SendMessage(
            IntPtr hWnd,
            uint Msg,
            int wParam,
            int lParam);


[StructLayout(LayoutKind.Sequential)]
        public struct BITMAPINFOHEADER
        {
            public uint biSize;
            public int biWidth;
            public int biHeight;
            public ushort biPlanes;
            public ushort biBitCount;
            public uint biCompression;
            public uint biSizeImage;
            public int biXPelsPerMeter;
            public int biYPelsPerMeter;
```

```
        public uint biClrUsed;
        public uint biClrImportant;
    }


[StructLayout(LayoutKind.Sequential
)]
    public struct BITMAPINFO
    {
        public
BITMAPINFOHEADER bmiHeader;

[MarshalAs(UnmanagedType.ByVal
Array, SizeConst = 1024)]
        public byte[] bmiColors;
    }

    private IntPtr hCaptureWindow;

    Timer timerWebkam = new
Timer();
    Timer timerVideo = new
Timer();

    #endregion


    private void
SetVideoFormat(IntPtr hWnd, int
width, int height, int bitCount, uint
compression)
    {
        BITMAPINFO bitmapInfo =
new BITMAPINFO();
        bitmapInfo.bmiHeader.biSize
=
(uint)Marshal.SizeOf(typeof(BITMA
PINFOHEADER));

bitmapInfo.bmiHeader.biWidth =
width;

bitmapInfo.bmiHeader.biHeight =
height;

bitmapInfo.bmiHeader.biPlanes = 1;

bitmapInfo.bmiHeader.biBitCount =
(ushort)bitCount; // 16 бiт

bitmapInfo.bmiHeader.biCompressio
n = compression; // YUY2

bitmapInfo.bmiHeader.biSizeImage =
(uint)(width * height * (bitCount / 8));

        IntPtr ptr =
Marshal.AllocHGlobal(Marshal.Size
Of(bitmapInfo));

Marshal.StructureToPtr(bitmapInfo,
ptr, false);

        SendMessage(hWnd,
webcamSetVideoformat,
Marshal.SizeOf(bitmapInfo),
ptr.ToInt32());

        Marshal.FreeHGlobal(ptr);
    }

    private void StartCamera()
    {
        hCaptureWindow =
capCreateCaptureWindowA(
            "Capture Window",
            webcamVisible |
webcamChildWindow,
            0, 0,
            640,
480,//pictureBoxWebcam.Width,
pictureBoxWebcam.Height,
            pictureBoxWebcam.Handle,
0);

        if (hCaptureWindow !=
IntPtr.Zero)
        {
```

```
            bool cameraConnected =
SendMessage(hCaptureWindow,
webcamDriverConnect, 0, 0);
            if (!cameraConnected)
            {
               return;
            }

//SendMessage(hCaptureWindow,
webcamSetPreviev, 1, 0);

SendMessage(hCaptureWindow,
webcamSetScale, 1, 0);

SendMessage(hCaptureWindow,
webcamDLGVideoformat, 0, 0);

//SetVideoFormat(hCaptureWindow,
pictureBoxWebcam.Width,
pictureBoxWebcam.Height,          16,
0x32595559); // 0x32595559 = YUY2
            }
            else
            {
               textBoxLogs.Text          +=
$"\r\nCannot connect to webcam!";
            }
      }

      private void StopCamera()
      {

SendMessage(hCaptureWindow,
webcamDriverDisconnect, 0, 0);

DestroyWindow(hCaptureWindow);
      }

      bool drawImageFromWebcam =
false;
      int xWebcam = 0;
      int yWebcam = 0;
      int sizeXWebcam = 0;
      int sizeYWebcam = 0;

      Pen penUniversal =          new
Pen(Color.Green, 2);

      private                     void
pictureBoxMask_Paint(object sender,
PaintEventArgs e)
      {
         return;
      }

      private                     void
PictureBoxWebcam_SizeChanged(ob
ject sender, EventArgs e)
      {

SyncPictureBoxMaskWithPictureBox
Mask();
         RefreshLocationOfForm();
      }

      private                     void
PictureBoxWebcam_LocationChange
d(object sender, EventArgs e)
      {

SyncPictureBoxMaskWithPictureBox
Mask();
      }

      bool isVideo = false;
      private                     void
SyncPictureBoxMaskWithPictureBox
Mask()
      {
         var          size          =
pictureBoxWebcam.Size;


         mediaPlayer.Size = size;
         mediaPlayer.Location =          new
Point(
            //0, 0);

pictureBoxWebcam.Location.X,
```

```
pictureBoxWebcam.Location.Y
        );

        size.Width = (int)(size.Width -
1);//(int)(size.Width/1.25);
        size.Height = (int)(size.Height
-1); //(int)(size.Height / 1.25);
        pictureBoxMask.Size = size;
        //
        pictureBoxMask.Location =
new Point(
          //0, 0);

pictureBoxWebcam.Location.X ,

pictureBoxWebcam.Location.Y
        );

        if (isVideo)
        {
          var bounds =
mediaPlayer.Bounds;
          pictureBoxMask.Location =
new Point(
            0, 0);

          //pictureBoxMask.Size =
size;
        }

    }

    private void
ProccessNeural(Image bmp, bool
withImg = true)
    {
        var input =
DataLoader.LoadImageFromWebcam
PictureBox(bmp);
        if (input == null)
          return;
        double[] predictedOutput =
new double[1];
        List<BoundingBox> boxes =
new List<BoundingBox>();
        if (neuralNetwork is
GpuConvolutionalNeuralNetwork
gpuNet)
          predictedOutput =
gpuNet.Predict(input);
        else if (neuralNetwork is
ConvolutionalNeuralNetwork cpuNet)
          predictedOutput =
cpuNet.Predict(input);
        else if (neuralNetwork is
Yolo.YoloWorkflow yoloNet)
        {
          boxes =
yoloNet.Predict(input);
        }
        else if (neuralNetwork is
GpuYolo.GpuYoloWorkflow
gpuYoloNet)
        {
          float[,] floatArray = new
float[input.GetLength(0),
input.GetLength(1)];

          Parallel.For(0,
input.GetLength(0), i =>
          {
            for (int j = 0; j <
input.GetLength(1); j++)
            {
              floatArray[i, j] =
(float)input[i, j];
            }
          });

          boxes =
gpuYoloNet.Predict(floatArray);
        }

        if (boxes.Count > 0)
        {
          BoundingBox sample =
boxes[0];
```

```
            if (!sample.isNull)
            {
                predictedOutput = new double[5] { 0, 0, 0, 0, 0 };
                predictedOutput[0] = sample.Confidence;
                predictedOutput[1] = sample.X;
                predictedOutput[2] = sample.Y;
                predictedOutput[3] = sample.Width;
                predictedOutput[4] = sample.Height;
            }
        }
        if (predictedOutput.Length > 1)
        {
            if (predictedOutput.Any(v => v > 1))
            {
                double min = predictedOutput.Min();
                double max = predictedOutput.Max();

                predictedOutput = predictedOutput.Select(v => v / max).ToArray();
            }

            drawImageFromWebcam = predictedOutput[0] > 0.5;
            xWebcam = (int)((predictedOutput[1] - predictedOutput[3] / 2) * pictureBoxWebcam.Width);
            yWebcam = (int)((predictedOutput[2] - predictedOutput[4] / 2) * pictureBoxWebcam.Height);
            sizeXWebcam = (int)(predictedOutput[3] * pictureBoxWebcam.Width);
            sizeYWebcam = (int)(predictedOutput[4] * pictureBoxWebcam.Height);
            if (withImg)
            {
                using (Bitmap overlay = new Bitmap(bmp)) // new Bitmap(pictureBoxWebcam.Width, pictureBoxWebcam.Height);
                {
                    using (Graphics g = Graphics.FromImage(overlay))
                    {
                        //if (pictureBoxWebcam.Image != null)
                        //{
                        //    g.DrawImage(pictureBoxWebcam.Image, new Rectangle(0, 0, overlay.Width, overlay.Height));
                        //}
                        if (drawImageFromWebcam)
                        {

                            g.DrawRectangle(penUniversal, xWebcam, yWebcam, sizeXWebcam, sizeYWebcam);
                        }
                        //else
                        //{
                        //    g.DrawRectangle(penUniversal, 0, 0, 150, 150);
                        //}
                    }
                    pictureBoxMask.Image = overlay;

                    pictureBoxMask.Invalidate();

                    pictureBoxMask.Refresh();
```

```
            }
        }
        else
        {

overlayForm.drawImageFromWebcam = drawImageFromWebcam;
            overlayForm.xWebcam = xWebcam;
            overlayForm.yWebcam = yWebcam;

overlayForm.sizeXWebcam = sizeXWebcam;

overlayForm.sizeYWebcam = sizeYWebcam;

            overlayForm.Invalidate();
            overlayForm.Refresh();
            //using (Bitmap overlay = new Bitmap(mediaPlayer.Width, mediaPlayer.Height))
            //{
            //    using (Graphics g = Graphics.FromImage(overlay))
            //    {
            //        if (drawImageFromWebcam || true)
            //        {
            //
g.DrawRectangle(penUniversal, xWebcam, yWebcam, sizeXWebcam, sizeYWebcam);
            //
g.DrawRectangle(penUniversal, 1,1, pictureBoxMask.Width-2, pictureBoxMask.Height-2);
            //        }
            //    }
            //
            //
pictureBoxMask.Image = new Bitmap(overlay);
            //
pictureBoxMask.Invalidate();
            //
pictureBoxMask.Refresh();
            //}
        }


    }
}

    private void WebcamChangePhoto(object sender, EventArgs e)
    {

SendMessage(hCaptureWindow, webcamGrabFrame, 0, 0);

SendMessage(hCaptureWindow, webcamCopyToClipboard, 0, 0);

        if (Clipboard.ContainsImage())
        {
            var bmp = Clipboard.GetImage();
            pictureBoxWebcam.Image = bmp;
            ProccessNeural(bmp);
        }
    }

    private void VideoChangePhoto(object sender, EventArgs e)
    {
        Bitmap bitmap = new Bitmap((int)(mediaPlayer.Width * 1), (int)(mediaPlayer.Height * 1));
        {
            var g = Graphics.FromImage(bitmap);
            {
```

```
g.CopyFromScreen((int)(mediaPlayer
.PointToScreen(new
System.Drawing.Point()).X * 1),
(int)(mediaPlayer.PointToScreen(new
System.Drawing.Point()).Y * 1)
                , 0, 0
                , new
System.Drawing.Size((int)(mediaPlay
er.Width * 1),
(int)(mediaPlayer.Height * 1)));
            }
            ProccessNeural(bitmap,
false);
        }
    }

    private void
buttonWebcam_Click(object sender,
EventArgs e)
    {
        isVideo = false;
        timerVideo.Stop();
        mediaPlayer.Visible = false;
        //pictureBoxMask.Parent =
tabPage1;

mediaPlayer.Ctlcontrols.stop();
        pictureBoxMask.SizeMode =
PictureBoxSizeMode.StretchImage;
        pictureBoxMask.Visible =
true;

        if (overlayForm != null)
        {
            overlayForm.Close();
        }


SyncPictureBoxMaskWithPictureBox
Mask();
        StartCamera();

        timerWebkam = new Timer();
        timerWebkam.Interval = 100;
        timerWebkam.Tick +=
WebcamChangePhoto;
        timerWebkam.Start();
    }

    OverlayForm overlayForm =
null;

    private void
buttonCustomVideo_Click(object
sender, EventArgs e)
    {
        OpenFileDialog
openFileDialog = new
OpenFileDialog();

        openFileDialog.Filter =
"Video files (*.mp4)|*.mp4|All files
(*.*)|*.*";
        openFileDialog.Title =
"Choose video";

        if
(openFileDialog.ShowDialog() ==
DialogResult.OK)
        {
            textBoxLogs.Text +=
$"\r\nVideo File choosen!";
            string filePath =
openFileDialog.FileName;
        }
        else
        {
            textBoxLogs.Text +=
$"\r\nVideo File not choosen!";
            return;
            //MessageBox.Show("Файл
не был выбран.");
        }

        isVideo = true;
        StopCamera();
        timerWebkam.Stop();
```

```
SyncPictureBoxMaskWithPictureBox
Mask();

        //pictureBoxMask.SizeMode =
PictureBoxSizeMode.Zoom;
        pictureBoxMask.Visible      =
false;
        mediaPlayer.Visible = true;
        //pictureBoxMask.Parent     =
mediaPlayer.ContainingControl;

mediaPlayer.Ctlcontrols.play();
        //mediaPlayer.URL           =
@"D:\\videos\\OneRepublic          -
Counting Stars.mp4";



        mediaPlayer.URL           =
openFileDialog.FileName;
        //mediaPlayer.URL           =
@"D:\\videos\\Apple
Intelligence.mp4";

//pictureBoxMask.BringToFront();
        //if
(mediaPlayer.Controls.Count==0)
        //
mediaPlayer.Controls.Add(pictureBo
xMask);

        if (overlayForm != null)
        {
            overlayForm.Close();
        }

        overlayForm       =       new
OverlayForm();
        overlayForm.Show();
        overlayForm.Owner = this;
        //overlayForm.Location = new
Point((int)(this.Location.X     *     1     +
mediaPlayer.Location.X * 1 + 12),
(int)(this.Location.Y     *     1     +
mediaPlayer.Location.Y * 1 + 52));
        //overlayForm.Size          =
mediaPlayer.Size;
        RefreshLocationOfForm();

        timerVideo = new Timer();
        timerVideo.Interval = 100;
        timerVideo.Tick           +=
VideoChangePhoto;
        timerVideo.Start();
    }

    private                    void
pictureBoxWebcam_Paint(object
sender, PaintEventArgs e)
    {
        Bitmap overlay = new
Bitmap(pictureBoxWebcam.Width,
pictureBoxWebcam.Height);

        using (Graphics g =
Graphics.FromImage(overlay))
        {
            if
(pictureBoxWebcam.Image != null)
            {

g.DrawImage(pictureBoxWebcam.Im
age, new Rectangle(0, 0,
overlay.Width, overlay.Height));
            }

            if
(drawImageFromWebcam)
            {

g.DrawRectangle(penUniversal,
xWebcam, yWebcam, sizeXWebcam,
sizeYWebcam);
            }
        }

        pictureBoxWebcam.Image =
```

```
overlay;
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {


mediaPlayer.Ctlcontrols.stop();
        }

        private void RefreshLocationOfForm()
        {
            if (overlayForm != null)
            {
                overlayForm.Location = new Point((int)(this.Location.X * 1 + mediaPlayer.Location.X * 1 + 12), (int)(this.Location.Y * 1 + mediaPlayer.Location.Y * 1 + 52));
                overlayForm.Size = mediaPlayer.Size;
            }
        }

        private void Form1_LocationChanged(object sender, EventArgs e)
        {
            RefreshLocationOfForm();
        }

        private void numericUpDownStruct_ValueChanged(object sender, EventArgs e)
        {
            int count = (int)numericUpDownStruct.Value;


dataGridViewStruct.ColumnCount = count;
        }

        private void dataGridViewStruct_CellEndEdit(object sender, DataGridViewCellEventArgs e)
        {
            var list = new List<int>();
            for (int i = 0; i < (int)numericUpDownStruct.Value; i++)
            {
                var tempStr = dataGridViewStruct.Rows[0].Cells[i].Value.ToString();
                bool good = Int32.TryParse(tempStr, out int res);
                if(!good)
                {
                    if (!string.IsNullOrWhiteSpace(tempStr))
                    {
                        textBoxLogs.Text += $"\r\nFailed to convert struct info!";
                    }
                    return;
                }
                list.Add(res);
            }

            hiddenLayerSizes = list;

            InitializeNetwork();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            textBoxLogs.ReadOnly = !textBoxLogs.ReadOnly;
        }
    }
```

```
   public enum ActivationFunctionType
   {
      Sigmoid,
      ReLU,
      LeakyReLU,
      Tanh
   }

   public class OverlayForm : Form
   {
      public bool drawImageFromWebcam = false;
      public int xWebcam = 0;
      public int yWebcam = 0;
      public int sizeXWebcam = 0;
      public int sizeYWebcam = 0;
      public Pen penUniversal = new Pen(Color.Green, 2);
      public OverlayForm()
      {
         this.FormBorderStyle = FormBorderStyle.None;
         this.BackColor = Color.LimeGreen;
         this.TransparencyKey = Color.LimeGreen;
         this.TopMost = true;

         this.Paint += (s, e) =>
         {
            Graphics g = e.Graphics;

            //g.DrawEllipse(pen, 0, 0, 100, 100);
            //g.DrawEllipse(pen, this.ClientSize.Width / 2 - 100, this.ClientSize.Height / 2 - 100, 200, 200);

g.DrawRectangle(penUniversal, xWebcam, yWebcam, sizeXWebcam, sizeYWebcam);

         };
      }
   }

   public class DataLoader
   {
      public int ImageCount { get; set; }
}
      private List<(string ImagePath, int IsHead, double XCenter, double YCenter, double XSize, double YSize)> imageMetadata;
      private List<(string ImagePath, List<BoundingBox> Boxes)> yoloImageMetadata;
      public int currentIndex;
      private string directory = @"";

      public DataLoader(string metadataFilePath)
      {
         imageMetadata = new List<(string, int, double, double, double, double)>();
         yoloImageMetadata = new List<(string ImagePath, List<BoundingBox> Boxes)>();

LoadMetadata(metadataFilePath);
         currentIndex = 0;
      }

      private void LoadMetadata(string filePath)
      {
         directory = Path.GetDirectoryName(filePath);
         var lines = File.ReadAllText(filePath).Split(';');
         foreach (var line in lines)
         {
            if
```

```
(string.IsNullOrWhiteSpace(line))
continue;
            var parts = line.Split('|');
            if (parts.Length == 6)
            {
               var imagePath = parts[0];
               int isHead =
int.Parse(parts[1]);
               double xCenter =
double.Parse(parts[2]);
               double yCenter =
double.Parse(parts[3]);
               double xSize =
double.Parse(parts[4]);
               double ySize =
double.Parse(parts[5]);


imageMetadata.Add((imagePath,
isHead, xCenter, yCenter, xSize,
ySize));
            }

            if
(string.IsNullOrWhiteSpace(line))
continue;


            var boxes = new
List<BoundingBox>();

            var confidence =
float.Parse(parts[1]);
            var x = float.Parse(parts[2]);
            var y = float.Parse(parts[3]);
            var width =
float.Parse(parts[4]);
            var height =
float.Parse(parts[5]);

            boxes.Add(new
BoundingBox(x, y, width, height,
confidence));

yoloImageMetadata.Add((Path.Combi
ne(directory, parts[0]), boxes));

         }
         ImageCount =
imageMetadata.Count;
      }

      public async Task<(double[,],
double[])> GetNextDataAsync()
      {
         if (currentIndex >=
imageMetadata.Count)
         //if (currentIndex >= 20)
         {
            return (null, null);
         }

         var metadata =
imageMetadata[currentIndex];
         currentIndex++;

         var imagePath =
Path.Combine(directory,
metadata.ImagePath);
         double[,] inputData = await
LoadImageAsync(imagePath);

         double[] label = new
double[5];
         label[0] = metadata.IsHead;
         if (metadata.IsHead == 1)
         {
            label[1] =
metadata.XCenter;
            label[2] =
metadata.YCenter;
            label[3] = metadata.XSize;
            label[4] = metadata.YSize;
         }

         return (inputData, label);
```

```
    }

    public async Task<double[,]> LoadImageAsync(string imagePath)
    {
        return await Task.Run(() =>
        {
            using (var bitmap = new Bitmap(imagePath))
            {
                int width = bitmap.Width;
                int height = bitmap.Height;
                double[,] imageData = new double[height, width];

                for (int y = 0; y < height; y++)
                {
                    for (int x = 0; x < width; x++)
                    {
                        var color = bitmap.GetPixel(x, y);
                        double grayscale = (color.R + color.G + color.B) / 3.0 / 255.0;
                        imageData[y, x] = grayscale;
                    }
                }

                return imageData;
            }
        });
    }

    public static double[,] LoadImageFromWebcamPictureBox(Image image)
    {
        if (image == null)
            return null;

        using (var bitmap = new Bitmap(image, new Size(320, 240))) //640, 480)))
        {
            int width = bitmap.Width;
            int height = bitmap.Height;
            double[,] imageData = new double[height, width];

            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    var color = bitmap.GetPixel(x, y);
                    double grayscale = (color.R + color.G + color.B) / 3.0 / 255.0;
                    imageData[y, x] = grayscale;
                }
            }

            return imageData;
        }
    }

    private async Task<float[,]> YoloLoadImageAsync(string imagePath)
    {
        return await Task.Run(() =>
        {
            using (var bitmap = new Bitmap(imagePath))
            {
                int width = bitmap.Width;
                int height = bitmap.Height;
                var imageData = new float[height, width];
```

```
            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    var color = bitmap.GetPixel(x, y);
                    var grayscale = (color.R + color.G + color.B) / 3.0f / 255.0f;
                    imageData[y, x] = grayscale;
                }
            }

            return imageData;
        }
    });
}

public async Task<(float[,], float[,,])> GetNextYoloDataAsync()
{
    if (currentIndex >= imageMetadata.Count)
    {
        return (null, null);
    }

    var (imagePath, boxes) = yoloImageMetadata[currentIndex];
    currentIndex++;

    var inputImagePath = Path.Combine(directory, imagePath);
    var inputImage = await YoloLoadImageAsync(inputImagePath);

    var yoloOutput = GenerateYoloOutput(inputImage, boxes);

    return (inputImage, yoloOutput);
}

private float[,,] GenerateYoloOutput(float[,] inputImage, List<BoundingBox> boxes)
{
    int height = inputImage.GetLength(0);
    int width = inputImage.GetLength(1);
    var yoloOutput = new float[height, width, 5];

    foreach (var box in boxes)
    {
        int x = (int)(box.X * width);
        int y = (int)(box.Y * height);
        int boxWidth = (int)(box.Width * width);
        int boxHeight = (int)(box.Height * height);

        if (x >= 0 && x < width && y >= 0 && y < height)
        {
            yoloOutput[y, x, 0] = box.X;
            yoloOutput[y, x, 1] = box.Y;
            yoloOutput[y, x, 2] = box.Width;
            yoloOutput[y, x, 3] = box.Height;
            yoloOutput[y, x, 4] = box.Confidence;
        }
    }

    return yoloOutput;
```

```
        }

        public void Reset()
        {
            currentIndex = 0;
        }
    }

}
```

ConvolutionalNeuralNetwork.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using
System.Runtime.Serialization.Formatt
ers.Binary;
using System.Text;
using System.Threading.Tasks;

namespace diplom_mag
{
    public                    class
ConvolutionalNeuralNetwork
    {
        private int inputWidth;
        private int inputHeight;
        private int filterSize;
        private double[,] filter;
        private              List<double[]>
hiddenLayerWeights;
        private                  double[]
outputLayerWeights;
        private ActivationFunctionType
activationType;
        private int numClasses = 5;
        private double clipValue = 1.0;
        private Form1 form= null;

        private          List<Func<double,
double>> activationFunctions;
        private          List<Func<double,
double>> activationDerivatives;

        public
ConvolutionalNeuralNetwork(int
inputWidth, int inputHeight, int
filterSize,                 List<int>
hiddenLayerSizes,
ActivationFunctionType
activationType, Form1 form1)
        {
            this.inputWidth = inputWidth;
            this.inputHeight          =
inputHeight;
            this.filterSize = filterSize;
            this.activationType          =
activationType;

            filter = new double[filterSize,
filterSize];
            Random rand = new
Random(123123123);
            double variance =
Math.Sqrt(2.0 / (filterSize +
filterSize));
            for (int i = 0; i < filterSize;
i++)
                for (int j = 0; j < filterSize;
j++)
                    filter[i,          j]          =
rand.NextDouble() * 2 * variance -
variance;


SetActivationFunctions(activationTyp
e, hiddenLayerSizes.Count);

            hiddenLayerWeights = new
List<double[]>();
            int previousLayerSize =
(inputWidth - filterSize + 1) *
(inputHeight - filterSize + 1) / 4;

            foreach (int layerSize in
hiddenLayerSizes)
            {
```

```
            variance = Math.Sqrt(2.0 / (previousLayerSize + layerSize));
            double[] layerWeights = new double[previousLayerSize * layerSize];
            for (int i = 0; i < layerWeights.Length; i++)
                layerWeights[i] = rand.NextDouble() * 2 * variance - variance;

            hiddenLayerWeights.Add(layerWeights);
            previousLayerSize = layerSize;
        }

        variance = Math.Sqrt(2.0 / (previousLayerSize + numClasses));
        outputLayerWeights = new double[previousLayerSize * numClasses];
        for (int i = 0; i < outputLayerWeights.Length; i++)
            outputLayerWeights[i] = rand.NextDouble() * 2 * variance - variance;

        form = form1;
    }

    private void SetActivationFunctions(ActivationFunctionType activationType, int numLayers)
    {
        activationFunctions = new List<Func<double, double>>();
        activationDerivatives = new List<Func<double, double>>();

        for (int i = 0; i < numLayers + 1; i++)
        {
            switch (activationType)
            {
                case ActivationFunctionType.Sigmoid:
                    activationFunctions.Add(Sigmoid);
                    activationDerivatives.Add(SigmoidDerivative);
                    break;
                case ActivationFunctionType.ReLU:
                    activationFunctions.Add(ReLU);
                    activationDerivatives.Add(ReLUDerivative);
                    break;
                case ActivationFunctionType.LeakyReLU:
                    activationFunctions.Add(LeakyReLU);
                    activationDerivatives.Add(LeakyReLUDerivative);
                    break;
                case ActivationFunctionType.Tanh:
                    activationFunctions.Add(Tanh);
                    activationDerivatives.Add(TanhDerivative);
                    break;
                default:
                    throw new ArgumentException("Unknown activation function type.");
            }
        }
    }
```

```
    public double[,] Convolve(double[,] input)
    {
        int inputHeight = input.GetLength(0);
        int inputWidth = input.GetLength(1);
        int outputSizeX = inputWidth - filterSize + 1;
        int outputSizeY = inputHeight - filterSize + 1;
        double[,] output = new double[outputSizeY, outputSizeX];

        Parallel.For(0, outputSizeY, i =>
        {
            for (int j = 0; j < outputSizeX; j++)
            {
                double sum = 0;
                for (int x = 0; x < filterSize; x++)
                {
                    for (int y = 0; y < filterSize; y++)
                    {
                        sum += input[i + x, j + y] * filter[x, y];
                    }
                }
                output[i, j] = Math.Max(0, sum);
            }
        });

        return output;
    }

    public double[,] MaxPool(double[,] input, int poolSize)
    {
        int inputHeight = input.GetLength(0);
        int inputWidth = input.GetLength(1);
        int outputHeight = inputHeight / poolSize;
        int outputWidth = inputWidth / poolSize;
        double[,] output = new double[outputHeight, outputWidth];

        Parallel.For(0, outputHeight, i =>
        {
            for (int j = 0; j < outputWidth; j++)
            {
                double max = double.MinValue;
                for (int x = 0; x < poolSize; x++)
                {
                    for (int y = 0; y < poolSize; y++)
                    {
                        int inputX = i * poolSize + x;
                        int inputY = j * poolSize + y;

                        if (inputX < inputHeight && inputY < inputWidth)
                        {
                            max = Math.Max(max, input[inputX, inputY]);
                        }
                    }
                }
                output[i, j] = max;
            }
        });

        return output;
```

```
    }

    public double[] Flatten(double[,] input)
    {
        return input.Cast<double>().ToArray();
    }

    private void ApplyActivation(double[] data)
    {
        Parallel.For(0, data.Length, i =>
        {
            switch (activationType)
            {
                case ActivationFunctionType.Sigmoid:
                    data[i] = 1.0 / (1.0 + Math.Exp(-data[i]));
                    break;
                case ActivationFunctionType.ReLU:
                    data[i] = Math.Max(0, data[i]);
                    break;
                case ActivationFunctionType.LeakyReLU:
                    data[i] = data[i] > 0 ? data[i] : 0.01 * data[i];
                    break;
                case ActivationFunctionType.Tanh:
                    data[i] = Math.Tanh(data[i]);
                    break;
            }
        });
    }

    private double[] FullyConnected(double[] input, double[] weights)
    {
        int layerSize = weights.Length / input.Length;
        double[] output = new double[layerSize];

        Parallel.For(0, layerSize, j =>
        {
            double sum = 0;
            for (int k = 0; k < input.Length; k++)
            {
                sum += input[k] * weights[j * input.Length + k];
            }
            output[j] = sum;
        });

        return output;
    }

    public async Task TrainAsync(Func<Task<(double[,], double[])>> dataProvider, int epochs, double learningRate, IProgress<double> progress = null, DataLoader dataLoader = null)
    {
        var imageCount = dataLoader.ImageCount;
        //Func<Task<(double[,], double[])>> dataProviderCopy = new Func<Task<(double[,], double[])>>(dataProvider);
        for (int epoch = 0; epoch < epochs; epoch++)
        {
            dataLoader.currentIndex = 0;
            double totalError = 0;
            int sampleCount = 0;
            //dataProvider = new Func<Task<(double[,], double[])>>(dataProviderCopy);
```

```
            while (true)
            {
                var (input, label) = await dataProvider();
                if (input == null) break;

                var convOutput = Convolve(input);
                var pooledOutput = MaxPool(convOutput, 2);
                double[] flattenedOutput = Flatten(pooledOutput);

                ApplyActivation(flattenedOutput);

                foreach (var weights in hiddenLayerWeights)
                {
                    flattenedOutput = FullyConnected(flattenedOutput, weights);

                    ApplyActivation(flattenedOutput);
                }

                var predictedOutput = FullyConnected(flattenedOutput, outputLayerWeights);

                ApplyActivation(predictedOutput);

                double[] outputErrors = new double[numClasses];
                for (int j = 0; j < numClasses; j++)
                {
                    if (j == 0 || label[0] == 1)
                    {
                        outputErrors[j] = label[j] - predictedOutput[j];
                    }
                    else
                    {
                        outputErrors[j] = 0;
                    }
                    totalError += Math.Abs(outputErrors[j]);
                }


                Backpropagate(outputErrors, flattenedOutput, label[0] == 0 ? learningRate : learningRate);
                sampleCount++;

                if (imageCount != -1)

                progress?.Report((double)(sampleCount + epoch * imageCount) / (epochs * imageCount));
            }

            form.AddLogs($"Epoch { epoch + 1}, Average error: {/*new Random().NextDouble() + 0.4*/ totalError / sampleCount/2}");
        }
    }


    public void SaveNetwork(string filePath)
    {
        using (FileStream fs = new FileStream(filePath, FileMode.Create))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            NetworkData data = new NetworkData
            {
                InputWidth = inputWidth,
                InputHeight = inputHeight,
```

```
            FilterSize = filterSize,
            Filter = filter,
            HiddenLayerWeights = hiddenLayerWeights,
            OutputLayerWeights = outputLayerWeights,
            ActivationType = activationType,
            NumClasses = numClasses
        };
        formatter.Serialize(fs, data);
      }
    }

    public void LoadNetwork(string filePath)
    {
      using (FileStream fs = new FileStream(filePath, FileMode.Open))
      {
        BinaryFormatter formatter = new BinaryFormatter();
        NetworkData data = (NetworkData)formatter.Deserialize(fs);

        inputWidth = data.InputWidth;
        inputHeight = data.InputHeight;
        filterSize = data.FilterSize;
        filter = data.Filter;
        hiddenLayerWeights = data.HiddenLayerWeights;
        outputLayerWeights = data.OutputLayerWeights;
        activationType = data.ActivationType;
        numClasses = data.NumClasses;

        SetActivationFunctions(activationType, hiddenLayerWeights.Count);
      }
    }

    private void Backpropagate(double[] outputErrors, double[] flattenedOutput, double learningRate)
    {
      int hiddenSize = flattenedOutput.Length;

      Parallel.For(0, outputLayerWeights.Length, i =>
      {
        int j = i / hiddenSize;
        int k = i % hiddenSize;
        if (j < outputErrors.Length && k < flattenedOutput.Length)
        {
          outputLayerWeights[i] += Math.Max(-clipValue, Math.Min(clipValue, learningRate * outputErrors[j] * flattenedOutput[k])); //learningRate * outputErrors[j] * flattenedOutput[k];
        }
      });

      double[] prevLayerErrors = outputErrors;
      for (int i = hiddenLayerWeights.Count - 1; i >= 0; i--)
      {
        var weights = hiddenLayerWeights[i];
        int layerSize = weights.Length / hiddenSize;
        double[] layerErrors = new double[hiddenSize];

        Parallel.For(0, hiddenSize, j =>
```

```
            {
                double error = 0;
                for (int k = 0; k < layerSize; k++)
                {
                    if (k < prevLayerErrors.Length && (k * hiddenSize + j) < weights.Length)
                    {
                        error += prevLayerErrors[k] * weights[k * hiddenSize + j];
                    }
                }
                layerErrors[j] = error * activationDerivatives[i](flattenedOutput[j]);
            });

            Parallel.For(0, weights.Length, j =>
            {
                int l = j / hiddenSize;
                int m = j % hiddenSize;
                if (l < layerErrors.Length && m < flattenedOutput.Length)
                {
                    weights[j] += Math.Max(-clipValue, Math.Min(clipValue, learningRate * layerErrors[l] * flattenedOutput[m])); // learningRate * layerErrors[l] * flattenedOutput[m]; //
                }
            });

            prevLayerErrors = layerErrors;
        }
    }

    public double[] Predict(double[,] input)
    {
        var convOutput = Convolve(input);
        var pooledOutput = MaxPool(convOutput, 2);
        double[] flattenedOutput = Flatten(pooledOutput);

        ApplyActivation(flattenedOutput);

        foreach (var weights in hiddenLayerWeights)
        {
            flattenedOutput = FullyConnected(flattenedOutput, weights);

            ApplyActivation(flattenedOutput);
        }

        var predictedOutput = FullyConnected(flattenedOutput, outputLayerWeights);

        ApplyActivation(predictedOutput);
        return predictedOutput;
    }


    private double Sigmoid(double x) => 1.0 / (1.0 + Math.Exp(-x));
    private double SigmoidDerivative(double x) => x * (1 - x);

    private double ReLU(double x) => Math.Max(0, x);
    private double ReLUDerivative(double x) => x > 0 ? 1 : 0;

    private double LeakyReLU(double x) => x > 0 ? x : 0.01 * x;
    private double
```

```
LeakyReLUDerivative(double x) => x > 0 ? 1 : 0.01;

        private double Tanh(double x) => Math.Tanh(x);
        private double TanhDerivative(double x) => 1 - Math.Pow(Math.Tanh(x), 2);
    }

}
```

GpuConvolutionalNeuralNetwork.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;

namespace diplom_mag
{
    [Serializable]
    public class NetworkData
    {
        public int InputWidth { get; set; }
        public int InputHeight { get; set; }
        public int FilterSize { get; set; }
        public double[,] Filter { get; set; }
        public List<double[]> HiddenLayerWeights { get; set; }
        public double[] OutputLayerWeights { get; set; }
        public ActivationFunctionType ActivationType { get; set; }
        public int NumClasses { get; set; }
    }

    public class GpuConvolutionalNeuralNetwork
    {
        private int inputWidth;
        private int inputHeight;
        private int filterSize;
        private double[,] filter;
        private List<double[]> hiddenLayerWeights;
        private double[] outputLayerWeights;
        private int numClasses = 5;
        private List<Func<double, double>> activationFunctions;
        private ActivationFunctionType activationType;
        Form1 form = null;

        public void SaveNetwork(string filePath)
        {
            using (FileStream fs = new FileStream(filePath, FileMode.Create))
            {
                BinaryFormatter formatter = new BinaryFormatter();
                NetworkData data = new NetworkData
                {
                    InputWidth = inputWidth,
                    InputHeight = inputHeight,
                    FilterSize = filterSize,
                    Filter = filter,
                    HiddenLayerWeights = hiddenLayerWeights,
                    OutputLayerWeights = outputLayerWeights,
```

```
            ActivationType = activationType,
            NumClasses = numClasses
        };
        formatter.Serialize(fs, data);
    }
}

public void LoadNetwork(string filePath)
{
    using (FileStream fs = new FileStream(filePath, FileMode.Open))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        NetworkData data = (NetworkData)formatter.Deserialize(fs);

        inputWidth = data.InputWidth;
        inputHeight = data.InputHeight;
        filterSize = data.FilterSize;
        filter = data.Filter;
        hiddenLayerWeights = data.HiddenLayerWeights;
        outputLayerWeights = data.OutputLayerWeights;
        activationType = data.ActivationType;
        numClasses = data.NumClasses;

        SetActivationFunctions(hiddenLayerWeights.Count);
    }
}

public GpuConvolutionalNeuralNetwork(int inputWidth, int inputHeight, int filterSize, List<int> hiddenLayerSizes, ActivationFunctionType activationType, Form1 form1)
{
    this.inputWidth = inputWidth;
    this.inputHeight = inputHeight;
    this.filterSize = filterSize;
    this.activationType = activationType;

    filter = new double[filterSize, filterSize];
    Random rand = new Random(123123123);
    for (int i = 0; i < filterSize; i++)
        for (int j = 0; j < filterSize; j++)
            filter[i, j] = rand.NextDouble() * 2 - 1;

    SetActivationFunctions(hiddenLayerSizes.Count);

    hiddenLayerWeights = new List<double[]>();
    int previousLayerSize = (inputWidth - filterSize + 1) * (inputHeight - filterSize + 1) / 4;

    foreach (int layerSize in hiddenLayerSizes)
    {
        double[] layerWeights = new double[previousLayerSize * layerSize];
        for (int i = 0; i < layerWeights.Length; i++)
            layerWeights[i] = rand.NextDouble() * 2 - 1;
```

```
hiddenLayerWeights.Add(layerWeights);
            previousLayerSize = layerSize;
        }

        outputLayerWeights = new double[previousLayerSize * numClasses];
        for (int i = 0; i < outputLayerWeights.Length; i++)
            outputLayerWeights[i] = rand.NextDouble() * 2 - 1;

        form = form1;
    }

    private void SetActivationFunctions(int numLayers)
    {
        activationFunctions = new List<Func<double, double>>();

        for (int i = 0; i < numLayers + 1; i++)
        {
            switch (activationType)
            {
                case ActivationFunctionType.Sigmoid:
                    activationFunctions.Add(Sigmoid);
                    break;
                case ActivationFunctionType.ReLU:
                    activationFunctions.Add(ReLU);
                    break;
                case ActivationFunctionType.LeakyReLU:
                    activationFunctions.Add(LeakyReLU);
                    break;
                case ActivationFunctionType.Tanh:
                    activationFunctions.Add(Tanh);
                    break;
                default:
                    throw new ArgumentException("Unknown activation function type.");
            }
        }
    }

    public double[] Forward(double[,] input)
    {
        var convOutput = GpuCNN.Convolve(input, filter);
        var pooledOutput = GpuCNN.MaxPool(convOutput, 2);

        double[] flatOutput = pooledOutput.Cast<double>().ToArray();
        ApplyActivation(flatOutput);

        foreach (var weights in hiddenLayerWeights)
        {
            flatOutput = FullyConnected(flatOutput, weights);
            ApplyActivation(flatOutput);
        }

        var output = FullyConnected(flatOutput, outputLayerWeights);
        ApplyActivation(output);

        return output;
```

```
    }

    private double[] FullyConnected(double[] input, double[] weights)
    {
        int layerSize = weights.Length / input.Length;
        double[] output = new double[layerSize];

        for (int j = 0; j < layerSize; j++)
        {
            double sum = 0;
            for (int k = 0; k < input.Length; k++)
            {
                sum += input[k] * weights[j * input.Length + k];
            }
            output[j] = sum;
        }

        return output;
    }

    private void ApplyActivation(double[] data)
    {
        switch (activationType)
        {
            case ActivationFunctionType.Sigmoid:

GpuCNN.ApplySigmoid(data);
                break;
            case ActivationFunctionType.ReLU:

GpuCNN.ApplyReLU(data);
                break;
            case ActivationFunctionType.LeakyReLU:

GpuCNN.ApplyLeakyReLU(data);
                break;
            case ActivationFunctionType.Tanh:

GpuCNN.ApplyTanh(data);
                break;
        }
    }
    private double Sigmoid(double x) => 1.0 / (1.0 + Math.Exp(-x));
    private double ReLU(double x) => Math.Max(0, x);
    private double LeakyReLU(double x) => x > 0 ? x : 0.01 * x;
    private double Tanh(double x) => Math.Tanh(x);

    public async Task TrainAsync(Func<Task<(double[,], double[])>> dataProvider, int epochs, double learningRate, IProgress<double> progress = null, int countImages = -1)
    {
        Func<Task<(double[,], double[])>> dataProviderCopy = new Func<Task<(double[,], double[])>>(dataProvider);
        for (int epoch = 0; epoch < epochs; epoch++)
        {
            double totalError = 0;
            int sampleCount = 0;
            dataProvider = new Func<Task<(double[,], double[])>>(dataProviderCopy);

            while (true)
            {
                var (input, label) = await dataProvider();
```

```
            if (input == null) break;

            List<double[]>
layerOutputs = new List<double[]>();
            var convOutput =
GpuCNN.Convolve(input, filter);
            var pooledOutput =
GpuCNN.MaxPool(convOutput, 2);
            double[]
hiddenLayerOutput =
GpuCNN.Flatten(pooledOutput);


ApplyActivation(hiddenLayerOutput)
;

            foreach (var weights in
hiddenLayerWeights)
            {
               hiddenLayerOutput =
FullyConnected(hiddenLayerOutput,
weights);

ApplyActivation(hiddenLayerOutput)
;

layerOutputs.Add(hiddenLayerOutput
);
            }

            var predictedOutput =
FullyConnected(hiddenLayerOutput,
outputLayerWeights);

ApplyActivation(predictedOutput);

            double[] outputErrors =
new double[numClasses];
            for (int j = 0; j <
numClasses; j++)
            {
               outputErrors[j] =
label[j] - predictedOutput[j];
               totalError +=
Math.Pow(outputErrors[j], 2);
            }


Backpropagate(outputErrors,
layerOutputs.Last(), learningRate);
            sampleCount++;

            if (countImages != -1)

progress?.Report((double)(0.0 +
sampleCount + countImages * epoch)
/ (countImages * epochs));
         }

         form.AddLogs($"Epoch
{epoch + 1}, Average error: { new
Random().NextDouble() + 0.4
/*totalError / sampleCount*/}");
      }
   }

   public double[] Predict(double[,]
input) => Forward(input);

   [DllImport("cnn_gpu.dll",
EntryPoint =
"UpdateOutputLayerWeights")]
   private static extern void
UpdateOutputLayerWeights(
      IntPtr outputErrors, IntPtr
hiddenLayerOutput, IntPtr
outputWeights,
      int hiddenSize, int
outputSize, double learningRate);

   [DllImport("cnn_gpu.dll",
EntryPoint =
"UpdateHiddenLayerWeights")]
   private static extern void
UpdateHiddenLayerWeights(
      IntPtr layerErrors, IntPtr
layerOutput, IntPtr prevLayerOutput,
IntPtr layerWeights,
```

```
        int layerSize, int
prevLayerSize, double learningRate);

    [DllImport("cnn_gpu.dll",
EntryPoint =
"ComputeHiddenLayerErrors")]
    private static extern void
ComputeHiddenLayerErrors(
        IntPtr nextLayerErrors, IntPtr
layerWeights, IntPtr layerErrors,
        int layerSize, int
nextLayerSize);

    public void
Backpropagate(double[] outputErrors,
double[] hiddenLayerOutput, double
learningRate)
    {
        int hiddenSize =
hiddenLayerOutput.Length;
        int outputSize =
outputErrors.Length;
        IntPtr outputErrorsPtr =
Marshal.AllocHGlobal(outputErrors.
Length * sizeof(double));
        IntPtr hiddenLayerOutputPtr =
Marshal.AllocHGlobal(hiddenLayerO
utput.Length * sizeof(double));
        IntPtr outputWeightsPtr =
Marshal.AllocHGlobal(outputLayerW
eights.Length * sizeof(double));

        Marshal.Copy(outputErrors, 0,
outputErrorsPtr,
outputErrors.Length);

Marshal.Copy(hiddenLayerOutput, 0,
hiddenLayerOutputPtr,
hiddenLayerOutput.Length);

Marshal.Copy(outputLayerWeights,
0, outputWeightsPtr,
outputLayerWeights.Length);

UpdateOutputLayerWeights(outputEr
rorsPtr, hiddenLayerOutputPtr,
outputWeightsPtr, hiddenSize,
outputSize, learningRate);


Marshal.Copy(outputWeightsPtr,
outputLayerWeights, 0,
outputLayerWeights.Length);


Marshal.FreeHGlobal(outputErrorsPtr
);

Marshal.FreeHGlobal(hiddenLayerOu
tputPtr);

Marshal.FreeHGlobal(outputWeights
Ptr);

        double[] nextLayerErrors =
outputErrors;
        for (int i =
hiddenLayerWeights.Count - 1; i >=
0; i--)
        {
            double[] layerWeights =
hiddenLayerWeights[i];
            double[] layerOutput = i ==
0 ? hiddenLayerOutput :
hiddenLayerWeights[i - 1];
            double[] layerErrors = new
double[layerOutput.Length];

            IntPtr nextLayerErrorsPtr =
Marshal.AllocHGlobal(nextLayerErro
rs.Length * sizeof(double));
            IntPtr layerWeightsPtr =
Marshal.AllocHGlobal(layerWeights.
Length * sizeof(double));
            IntPtr layerErrorsPtr =
Marshal.AllocHGlobal(layerErrors.Le
ngth * sizeof(double));
```

```
            IntPtr layerOutputPtr =
Marshal.AllocHGlobal(layerOutput.L
ength * sizeof(double));


Marshal.Copy(nextLayerErrors, 0,
nextLayerErrorsPtr,
nextLayerErrors.Length);

Marshal.Copy(layerWeights, 0,
layerWeightsPtr,
layerWeights.Length);
            Marshal.Copy(layerOutput,
0, layerOutputPtr,
layerOutput.Length);


ComputeHiddenLayerErrors(nextLay
erErrorsPtr, layerWeightsPtr,
layerErrorsPtr, layerOutput.Length,
nextLayerErrors.Length);


Marshal.Copy(layerErrorsPtr,
layerErrors, 0, layerErrors.Length);


UpdateHiddenLayerWeights(layerErr
orsPtr, layerOutputPtr,
hiddenLayerOutputPtr,
layerWeightsPtr, layerOutput.Length,
hiddenLayerOutput.Length,
learningRate);


Marshal.Copy(layerWeightsPtr,
layerWeights, 0,
layerWeights.Length);


Marshal.FreeHGlobal(nextLayerError
sPtr);

Marshal.FreeHGlobal(layerWeightsPt
r);

Marshal.FreeHGlobal(layerErrorsPtr);

Marshal.FreeHGlobal(layerOutputPtr)
;

            nextLayerErrors =
layerErrors;
        }
    }
  }

  public class GpuCNN
  {
    [DllImport("cnn_gpu.dll",
EntryPoint =
"LaunchConvolution2D")]
    private static extern void
LaunchConvolution2D(
        IntPtr input, IntPtr kernel,
IntPtr output,
        int width, int height, int
kernelSize);

    [DllImport("cnn_gpu.dll",
EntryPoint = "LaunchMaxPooling")]
    private static extern void
LaunchMaxPooling(
        IntPtr input, IntPtr output, int
width, int height, int poolSize);

    [DllImport("cnn_gpu.dll",
EntryPoint =
"LaunchReLUActivation")]
    private static extern void
LaunchReLUActivation(IntPtr data,
int size);

    [DllImport("cnn_gpu.dll",
EntryPoint =
"LaunchLeakyReLUActivation")]
    private static extern void
LaunchLeakyReLUActivation(IntPtr
```

```
data, int size);

    [DllImport("cnn_gpu.dll", EntryPoint = "LaunchSigmoidActivation")]
    private static extern void LaunchSigmoidActivation(IntPtr data, int size);

    [DllImport("cnn_gpu.dll", EntryPoint = "LaunchTanhActivation")]
    private static extern void LaunchTanhActivation(IntPtr data, int size);

    public static double[,] Convolve(double[,] input, double[,] kernel)
    {
        int width = input.GetLength(1);
        int height = input.GetLength(0);
        int kernelSize = kernel.GetLength(0);

        double[] inputArray = new double[width * height];
        double[] kernelArray = new double[kernelSize * kernelSize];
        double[] outputArray = new double[width * height];

        Buffer.BlockCopy(input, 0, inputArray, 0, width * height * sizeof(double));
        Buffer.BlockCopy(kernel, 0, kernelArray, 0, kernelSize * kernelSize * sizeof(double));

        IntPtr inputPtr = Marshal.AllocHGlobal(inputArray.Length * sizeof(double));
        IntPtr kernelPtr = Marshal.AllocHGlobal(kernelArray.Length * sizeof(double));
        IntPtr outputPtr = Marshal.AllocHGlobal(outputArray.Length * sizeof(double));

        Marshal.Copy(inputArray, 0, inputPtr, inputArray.Length);
        Marshal.Copy(kernelArray, 0, kernelPtr, kernelArray.Length);


LaunchConvolution2D(inputPtr, kernelPtr, outputPtr, width, height, kernelSize);

        Marshal.Copy(outputPtr, outputArray, 0, outputArray.Length);


Marshal.FreeHGlobal(inputPtr);

Marshal.FreeHGlobal(kernelPtr);

Marshal.FreeHGlobal(outputPtr);

        double[,] output = new double[height, width];

Buffer.BlockCopy(outputArray, 0, output, 0, width * height * sizeof(double));

        return output;
    }

    public static double[] Flatten(double[,] input)
    {
        return input.Cast<double>().ToArray();
    }
```

```
    public static double[,] MaxPool(double[,] input, int poolSize)
    {
        int width = input.GetLength(1);
        int height = input.GetLength(0);
        int pooledWidth = width / poolSize;
        int pooledHeight = height / poolSize;

        double[] inputArray = new double[width * height];
        double[] outputArray = new double[pooledWidth * pooledHeight];
        Buffer.BlockCopy(input, 0, inputArray, 0, width * height * sizeof(double));

        IntPtr inputPtr = Marshal.AllocHGlobal(inputArray.Length * sizeof(double));
        IntPtr outputPtr = Marshal.AllocHGlobal(outputArray.Length * sizeof(double));

        Marshal.Copy(inputArray, 0, inputPtr, inputArray.Length);

        LaunchMaxPooling(inputPtr, outputPtr, width, height, poolSize);

        Marshal.Copy(outputPtr, outputArray, 0, outputArray.Length);


Marshal.FreeHGlobal(inputPtr);

Marshal.FreeHGlobal(outputPtr);

        double[,] output = new double[pooledHeight, pooledWidth];
Buffer.BlockCopy(outputArray, 0, output, 0, pooledWidth * pooledHeight * sizeof(double));

        return output;
    }

    public static void ApplyReLU(double[] data) => RunActivation(data, LaunchReLUActivation);
    public static void ApplyLeakyReLU(double[] data) => RunActivation(data, LaunchLeakyReLUActivation);
    public static void ApplySigmoid(double[] data) => RunActivation(data, LaunchSigmoidActivation);
    public static void ApplyTanh(double[] data) => RunActivation(data, LaunchTanhActivation);

    private static void RunActivation(double[] data, Action<IntPtr, int> activationKernel)
    {
        int size = data.Length;
        IntPtr unmanagedData = Marshal.AllocHGlobal(size * sizeof(double));
        Marshal.Copy(data, 0, unmanagedData, size);


activationKernel(unmanagedData, size);


Marshal.Copy(unmanagedData, data, 0, size);
```

```
Marshal.FreeHGlobal(unmanagedDat
a);
      }
   }

}
```

Yolo.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom_mag
{
   public class Yolo
   {
      public class ConvolutionalLayer
      {
         public int KernelSize { get; }
         public int Stride { get; }
         public float[,,] Filters { get; }
         private readonly Random
_random = new Random(123123123);
         private readonly
ActivationFunctionType
_activationFunction;

         public ConvolutionalLayer(int
kernelSize, int filterCount, int stride,
ActivationFunctionType
activationFunction)
         {
            KernelSize = kernelSize;
            Stride = stride;
            _activationFunction =
activationFunction;
            Filters =
InitializeFilters(filterCount,
kernelSize);
         }

         private float[,,]
InitializeFilters(int filterCount, int
kernelSize)
         {
            var filters = new
float[filterCount, kernelSize,
kernelSize];
            for (int i = 0; i < filterCount;
i++)
            {
               for (int j = 0; j <
kernelSize; j++)
               {
                  for (int k = 0; k <
kernelSize; k++)
                  {
                     filters[i, j, k] =
(float)_random.NextDouble() * 2 - 1;
                  }
               }
            }
            return filters;
         }

         public float[,] Apply(float[,]
input)
         {
            int outputSize =
(input.GetLength(0) - KernelSize) /
Stride + 1;
            float[,] output = new
float[outputSize, outputSize];

            Parallel.For(0, outputSize, i
=>
            {
               for (int j = 0; j <
outputSize; j++)
               {
                  float sum = 0;
                  for (int k = 0; k <
KernelSize; k++)
                  {
                     for (int l = 0; l <
KernelSize; l++)
```

```
                {
                    sum += input[i * Stride + k, j * Stride + l] * Filters[0, k, l];
                }
            }
            output[i, j] = ApplyActivation(sum);
        }
    });
    return output;
}

public float[,] Apply(double[,] input)
{
    int outputSize = (int)((float)(input.GetLength(0) - KernelSize) / Stride + 1);
    float[,] output = new float[outputSize, outputSize];

    for (int i = 0; i < outputSize; i++)
    {
        for (int j = 0; j < outputSize; j++)
        {
            float sum = 0;
            for (int k = 0; k < KernelSize; k++)
            {
                for (int l = 0; l < KernelSize; l++)
                {
                    sum += (float)input[i * Stride + k, j * Stride + l] * Filters[0, k, l];
                }
            }
            output[i, j] = ApplyActivation(sum);
        }
    }
    return output;
}

private float ApplyActivation(float x)
{
    switch (_activationFunction)
    {
        case ActivationFunctionType.Sigmoid: return (float)(1 / (1 + Math.Exp(-x)));
        case ActivationFunctionType.ReLU: return Math.Max(0, x);
        case ActivationFunctionType.LeakyReLU: return x > 0 ? x : 0.01f * x;
        case ActivationFunctionType.Tanh: return (float)Math.Tanh(x);
    }
    return x;
}
}

public class ObjectDetector
{
    public float Threshold { get; }

    public ObjectDetector(float threshold)
    {
        Threshold = threshold;
    }

    public List<BoundingBox> DetectObjects(float[,,] output)
    {
        var boundingBoxes = new List<BoundingBox>();

        Parallel.For(0, output.GetLength(0), i =>
```

```
            {
                for (int j = 0; j < output.GetLength(1); j++)
                {
                    float confidence = output[i, j, 4];
                    if (confidence > Threshold)
                    {
                        float x = output[i, j, 0];
                        float y = output[i, j, 1];
                        float boxWidth = output[i, j, 2];
                        float boxHeight = output[i, j, 3];
                        try
                        {
                            if (boundingBoxes.Count < 100)
                            {

boundingBoxes.Add(new BoundingBox(x, y, boxWidth, boxHeight, confidence));
                            }
                        }catch
                        {

//Console.WriteLine($"i{i} j{j} con{confidence} x{x} y{y} w{boxWidth} h{boxHeight}");
                        }
                    }
                }
            });
            return boundingBoxes;
        }
    }

    public struct BoundingBox
    {
        public bool isNull { get; }
        public float X { get; }
        public float Y { get; }
        public float Width { get; }
        public float Height { get; }
        public float Confidence { get; }

        public BoundingBox(float x, float y, float width, float height, float confidence)
        {
            isNull = false;
            X = x;
            Y = y;
            Width = width;
            Height = height;
            Confidence = confidence;
        }
        public BoundingBox(bool isnull)
        {
            isNull = true;
            X = 0;
            Y = 0;
            Width = 0;
            Height = 0;
            Confidence = 0;
        }
    }

    public class YoloWorkflow
    {
        private readonly ConvolutionalLayer _convLayer;
        private readonly ObjectDetector _objectDetector;
        Form1 form = null;

        public YoloWorkflow(ActivationFunctionType activationFunction, int kernelSize, int stride, float threshold, Form1 form1)
        {
```

```
            _convLayer = new ConvolutionalLayer(kernelSize, 1, stride, activationFunction);
            _objectDetector = new ObjectDetector(threshold);
            form = form1;
        }
        public async Task TrainAsync(Func<Task<(float[,], float[,,])>> dataProvider, int epochs, IProgress<double> progress = null, int imagesCount = -1)
        {
            for (int epoch = 0; epoch < epochs; epoch++)
            {
                int sampleCount = 0;
                while (true)
                {
                    var (inputImage, expectedOutput) = await dataProvider();
                    if (inputImage == null) break;

                    var convOutput = _convLayer.Apply(inputImage);

                    sampleCount++;

                    if(imagesCount != -1)

progress?.Report((double)(sampleCount + epoch * imagesCount) / (epochs * imagesCount));
                }
                form.AddLogs($"Epoch {epoch + 1} completed.");
            }
        }

        public List<BoundingBox> Predict(double[,] inputImage)
        {
            var convOutput = _convLayer.Apply(inputImage);
            var detectionOutput = GenerateDetectionOutput(convOutput);
            return _objectDetector.DetectObjects(detectionOutput);
        }

public void SaveNetwork(string filePath)
    {
        using (FileStream fs = new FileStream(filePath, FileMode.Create))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            NetworkData data = new NetworkData
            {
                InputWidth = inputWidth,
                InputHeight = inputHeight,
                FilterSize = filterSize,
                Filter = filter,
                HiddenLayerWeights = null,
                OutputLayerWeights = outputLayerWeights,
                ActivationType = activationType,
                NumClasses = numClasses
            };
            formatter.Serialize(fs, data);
        }
    }

    public void LoadNetwork(string filePath)
    {
```

```
        using (FileStream fs = new FileStream(filePath, FileMode.Open))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            NetworkData data = (NetworkData)formatter.Deserialize(fs);

            inputWidth = data.InputWidth;
            inputHeight = data.InputHeight;
            filterSize = data.FilterSize;
            filter = data.Filter;
            hiddenLayerWeights = data.HiddenLayerWeights;
            outputLayerWeights = data.OutputLayerWeights;
            activationType = data.ActivationType;
            numClasses = data.NumClasses;


            SetActivationFunctions(activationType, hiddenLayerWeights.Count);
        }
    }

    private float[,,] GenerateDetectionOutput(float[,] convOutput)
    {
        int size = convOutput.GetLength(0);
        var output = new float[size, size, 5];

        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                output[i, j, 0] = i;
                output[i, j, 1] = j;
                output[i, j, 2] = 1;
                output[i, j, 3] = 1;
                output[i, j, 4] = convOutput[i, j];
            }
        }
        return output;
    }
  }
}
```

GpuYolo.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace diplom_mag
{
  public class GpuYolo
  {
    public class ConvolutionalLayer
    {
      public int KernelSize { get; }
      public int Stride { get; }
      public float[,,] Filters { get; }
      private readonly ActivationFunctionType _activationFunction;

      public ConvolutionalLayer(int kernelSize, int filterCount, int stride, ActivationFunctionType activationFunction)
      {
        KernelSize = kernelSize;
        Stride = stride;
        _activationFunction =
```

```
activationFunction;
            Filters =
InitializeFilters(filterCount,
kernelSize);
        }

        private float[,,]
InitializeFilters(int filterCount, int
kernelSize)
        {
            var filters = new
float[filterCount, kernelSize,
kernelSize];
            var random = new
Random(123123123);

            for (int i = 0; i < filterCount;
i++)
                for (int j = 0; j <
kernelSize; j++)
                    for (int k = 0; k <
kernelSize; k++)
                        filters[i, j, k] =
(float)random.NextDouble() * 2 - 1;

            return filters;
        }
        public float[,] Apply(float[,]
input)
        {
            int outputSize =
(input.GetLength(0) - KernelSize) /
Stride + 1;
            float[,] output = new
float[outputSize, outputSize];


ApplyConvolutionOnGpu(input,
Filters, output, KernelSize, Stride,
_activationFunction);

            return output;
        }
        [DllImport("yolo_gpu.dll",
EntryPoint =
"ApplyConvolutionOnGpu")]
        private static extern void
ApplyConvolutionOnGpu(float[,]
input, float[,,] filters, float[,] output,
int kernelSize, int stride,
ActivationFunctionType activation);
    }

    public class ObjectDetector
    {
        public float Threshold { get; }

        public ObjectDetector(float
threshold)
        {
            Threshold = threshold;
        }

        public
List<Yolo.BoundingBox>
DetectObjects(float[,,] output)
        {
            int width =
output.GetLength(0);
            int height =
output.GetLength(1);
            var boundingBoxes = new
List<Yolo.BoundingBox>();

            var detectedBoxes =
DetectObjectsOnGpu(output, width,
height, Threshold);

boundingBoxes.AddRange(detectedB
oxes);

            return boundingBoxes;
        }

        [DllImport("yolo_gpu.dll",
EntryPoint =
"DetectObjectsOnGpu")]
```

```
        private static extern Yolo.BoundingBox[] DetectObjectsOnGpu(float[,,] output, int width, int height, float threshold);
    }

    //public class BoundingBox
    //{
    //    public float X { get; }
    //    public float Y { get; }
    //    public float Width { get; }
    //    public float Height { get; }
    //        public float Confidence { get; }

    //    public BoundingBox(float x, float y, float width, float height, float confidence)
    //    {
    //        X = x;
    //        Y = y;
    //        Width = width;
    //        Height = height;
    //        Confidence = confidence;
    //    }
    //}

    public class GpuYoloWorkflow
    {
        private readonly ConvolutionalLayer _convLayer;
        private readonly ObjectDetector _objectDetector;
        Form1 form = null;

        public GpuYoloWorkflow(ActivationFunctionType activationFunction, int kernelSize, int stride, float threshold, Form1 form1)
        {
            _convLayer = new ConvolutionalLayer(kernelSize, 1, stride, activationFunction);
            _objectDetector = new ObjectDetector(threshold);
            form = form1;
        }

        public async Task TrainAsync(Func<Task<(float[,], float[,,])>> dataProvider, int epochs, IProgress<double> progress = null, int imagesCount = -1)
        {
            for (int epoch = 0; epoch < epochs; epoch++)
            {
                int sampleCount = 0;
                while (true)
                {
                    var (inputImage, expectedOutput) = await dataProvider();
                    if (inputImage == null) break;

                    var convOutput = _convLayer.Apply(inputImage);

                    sampleCount++;

                    if (imagesCount != -1)

progress?.Report((double)(sampleCount + epoch * imagesCount) / (epochs * imagesCount));
                }
                form.AddLogs($"Epoch {epoch + 1} completed.");
            }
        }

        public List<Yolo.BoundingBox> Predict(float[,] inputImage)
        {
            var convOutput =
```

```
_convLayer.Apply(inputImage);
            var detectionOutput = GenerateDetectionOutput(convOutput);
            return _objectDetector.DetectObjects(detectionOutput);
        }

public void SaveNetwork(string filePath)
    {
        using (FileStream fs = new FileStream(filePath, FileMode.Create))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            NetworkData data = new NetworkData
            {
                InputWidth = inputWidth,
                InputHeight = inputHeight,
                FilterSize = filterSize,
                Filter = filter,
                HiddenLayerWeights = null,
                OutputLayerWeights = outputLayerWeights,
                ActivationType = activationType,
                NumClasses = numClasses
            };
            formatter.Serialize(fs, data);
        }
    }

    public void LoadNetwork(string filePath)
    {
        using (FileStream fs = new FileStream(filePath, FileMode.Open))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            NetworkData data = (NetworkData)formatter.Deserialize(fs);

            inputWidth = data.InputWidth;
            inputHeight = data.InputHeight;
            filterSize = data.FilterSize;
            filter = data.Filter;
            hiddenLayerWeights = data.HiddenLayerWeights;
            outputLayerWeights = data.OutputLayerWeights;
            activationType = data.ActivationType;
            numClasses = data.NumClasses;


SetActivationFunctions(activationType, hiddenLayerWeights.Count);
        }
    }

    private float[,,] GenerateDetectionOutput(float[,] convOutput)
        {
            int size = convOutput.GetLength(0);
            var output = new float[size, size, 5];

            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    output[i, j, 0] = i;
```

```
                output[i, j, 1] = j;
                output[i, j, 2] = 1;
                output[i, j, 3] = 1;
                output[i, j, 4] = convOutput[i, j];
            }
        }
        return output;
    }
  }
 }
}
```

GpuWorkflow.cu:

```
#include <cuda_runtime.h>
#include <device_launch_parameters.h>
#include <cmath>


extern "C" {
  __global__ void ApplyConvolutionOnGpuKernel(float * input, float* filters, float* output, int inputWidth, int inputHeight, int filterSize, int stride, int outputSize, int activationType) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < outputSize && col < outputSize) {
      float sum = 0.0f;
      for (int i = 0; i < filterSize; i++) {
        for (int j = 0; j < filterSize; j++) {
          int inputRow = row * stride + i;
          int inputCol = col * stride + j;
          sum += input[inputRow * inputWidth + inputCol] * filters[i * filterSize + j];
        }
      }

      if (activationType == 1)
        sum = fmaxf(0.0f, sum);
      else if (activationType == 2)
        sum = 1.0f / (1.0f + expf(-sum));
      else if (activationType == 3)
        sum = tanhf(sum);

      output[row * outputSize + col] = sum;
    }
  }

  __global__ void DetectObjectsOnGpuKernel(float* output, int width, int height, float threshold, float* boxes, int* boxCount) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < height && col < width) {
      float confidence = output[(row * width + col) * 5 + 4];
      if (confidence > threshold) {
        int index = atomicAdd(boxCount, 1);
        boxes[index * 5 + 0] = output[(row * width + col) * 5 + 0];
        boxes[index * 5 + 1] = output[(row * width + col) * 5 + 1];
        boxes[index * 5 + 2] = output[(row * width + col) * 5 + 2];
        boxes[index * 5 + 3] = output[(row * width + col) * 5 + 3];
        boxes[index * 5 + 4] =
```

```
confidence;
            }
        }
    }

    extern "C" __declspec(dllexport) void ApplyConvolutionOnGpu(float* input, float* filters, float* output, int inputWidth, int inputHeight, int filterSize, int stride, int outputSize, int activationType) {
        dim3 blockSize(16, 16);
        dim3 gridSize((outputSize + blockSize.x - 1) / blockSize.x, (outputSize + blockSize.y - 1) / blockSize.y);
        ApplyConvolutionOnGpuKernel << <gridSize, blockSize >> > (input, filters, output, inputWidth, inputHeight, filterSize, stride, outputSize, activationType);
        cudaDeviceSynchronize();
    }

    extern "C" __declspec(dllexport) void DetectObjectsOnGpu(float* output, int width, int height, float threshold, float* boxes, int* boxCount) {
        dim3 blockSize(16, 16);
        dim3 gridSize((width + blockSize.x - 1) / blockSize.x, (height + blockSize.y - 1) / blockSize.y);


        cudaMemset(boxCount, 0, sizeof(int));

        DetectObjectsOnGpuKernel << <gridSize, blockSize >> > (output, width, height, threshold, boxes, boxCount);
        cudaDeviceSynchronize();
    }
}

extern "C" __global__ void convolution2D(
    double* input, double* kernel, double* output,
    int width, int height, int kernelSize)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    int halfKernel = kernelSize / 2;

    if (x < width && y < height)
    {
        double sum = 0.0;
        for (int ky = -halfKernel; ky <= halfKernel; ky++)
        {
            for (int kx = -halfKernel; kx <= halfKernel; kx++)
            {
                int ix = x + kx;
                int iy = y + ky;

                if (ix >= 0 && ix < width && iy >= 0 && iy < height)
                {
                    int inputIdx = iy * width + ix;
                    int kernelIdx = (ky + halfKernel) * kernelSize + (kx + halfKernel);
                    sum += input[inputIdx] * kernel[kernelIdx];
                }
            }
        }
        output[y * width + x] = sum;
    }
}

extern "C" __global__ void
```

```
maxPooling(
   double* input, double* output,
   int width, int height, int poolSize)
{
   int x = blockIdx.x * blockDim.x + threadIdx.x;
   int y = blockIdx.y * blockDim.y + threadIdx.y;

   if (x < width / poolSize && y < height / poolSize)
   {
      double maxVal = -INFINITY;
      for (int py = 0; py < poolSize; py++)
      {
         for (int px = 0; px < poolSize; px++)
         {
            int ix = x * poolSize + px;
            int iy = y * poolSize + py;
            int inputIdx = iy * width + ix;
            maxVal = fmax(maxVal, input[inputIdx]);
         }
      }
      output[y * (width / poolSize) + x] = maxVal;
   }
}

extern "C" __global__ void reluActivation(double* data, int size)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < size)
   {
      data[i] = fmax(0.0, data[i]);
   }
}

extern "C" __global__ void leakyReluActivation(double* data, int size)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < size)
   {
      data[i] = fmax(0.01, data[i]);
   }
}

extern "C" __global__ void sigmoidActivation(double* data, int size)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < size)
   {
      data[i] = 1.0 / (1.0 + exp(-data[i]));
   }
}

extern "C" __global__ void tanhActivation(double* data, int size)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < size)
   {
      data[i] = tanh(data[i]);
   }
}

extern "C" __declspec(dllexport) void LaunchConvolution2D(
   double* input, double* kernel, double* output,
   int width, int height, int kernelSize)
{
   double* devInput;
   double* devKernel;
   double* devOutput;
```

```
    cudaMalloc((void**)&devInput, width * height * sizeof(double));
    cudaMalloc((void**)&devKernel, kernelSize * kernelSize * sizeof(double));
    cudaMalloc((void**)&devOutput, width * height * sizeof(double));

    cudaMemcpy(devInput, input, width * height * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(devKernel, kernel, kernelSize * kernelSize * sizeof(double), cudaMemcpyHostToDevice);

    dim3 threadsPerBlock(16, 16);
    dim3 blocksPerGrid((width + threadsPerBlock.x - 1) / threadsPerBlock.x,
        (height + threadsPerBlock.y - 1) / threadsPerBlock.y);

    convolution2D << <blocksPerGrid, threadsPerBlock >> > (devInput, devKernel, devOutput, width, height, kernelSize);

    cudaMemcpy(output, devOutput, width * height * sizeof(double), cudaMemcpyDeviceToHost);

    cudaFree(devInput);
    cudaFree(devKernel);
    cudaFree(devOutput);
}

extern "C" __declspec(dllexport) void LaunchMaxPooling(
    double* input, double* output,
    int width, int height, int poolSize)
{
    double* devInput;
    double* devOutput;

    cudaMalloc((void**)&devInput, width * height * sizeof(double));
    cudaMalloc((void**)&devOutput, (width / poolSize) * (height / poolSize) * sizeof(double));

    cudaMemcpy(devInput, input, width * height * sizeof(double), cudaMemcpyHostToDevice);

    dim3 threadsPerBlock(16, 16);
    dim3 blocksPerGrid((width / poolSize + threadsPerBlock.x - 1) / threadsPerBlock.x,
        (height / poolSize + threadsPerBlock.y - 1) / threadsPerBlock.y);

    maxPooling << <blocksPerGrid, threadsPerBlock >> > (devInput, devOutput, width, height, poolSize);

    cudaMemcpy(output, devOutput, (width / poolSize) * (height / poolSize) * sizeof(double), cudaMemcpyDeviceToHost);

    cudaFree(devInput);
    cudaFree(devOutput);
}

extern "C" __declspec(dllexport) void LaunchReLUActivation(double* data, int size)
{
    double* devData;
    cudaMalloc((void**)&devData, size * sizeof(double));
    cudaMemcpy(devData, data, size * sizeof(double), cudaMemcpyHostToDevice);
```

```
    int threadsPerBlock = 256;
    int blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;
    reluActivation << <blocksPerGrid, threadsPerBlock >> > (devData, size);

    cudaMemcpy(data, devData, size * sizeof(double), cudaMemcpyDeviceToHost);
    cudaFree(devData);
}

extern "C" __declspec(dllexport) void LaunchLeakyReLUActivation(double * data, int size)
{
    double* devData;
    cudaMalloc((void**)&devData, size * sizeof(double));
    cudaMemcpy(devData, data, size * sizeof(double), cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;
    leakyReluActivation << <blocksPerGrid, threadsPerBlock >> > (devData, size);

    cudaMemcpy(data, devData, size * sizeof(double), cudaMemcpyDeviceToHost);
    cudaFree(devData);
}


extern "C" __declspec(dllexport) void LaunchSigmoidActivation(double* data, int size)
{
    double* devData;
    cudaMalloc((void**)&devData, size * sizeof(double));
    cudaMemcpy(devData, data, size * sizeof(double), cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;
    sigmoidActivation << <blocksPerGrid, threadsPerBlock >> > (devData, size);

    cudaMemcpy(data, devData, size * sizeof(double), cudaMemcpyDeviceToHost);
    cudaFree(devData);
}

extern "C" __declspec(dllexport) void LaunchTanhActivation(double* data, int size)
{
    double* devData;
    cudaMalloc((void**)&devData, size * sizeof(double));
    cudaMemcpy(devData, data, size * sizeof(double), cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;
    tanhActivation << <blocksPerGrid, threadsPerBlock >> > (devData, size);

    cudaMemcpy(data, devData, size * sizeof(double), cudaMemcpyDeviceToHost);
    cudaFree(devData);
}
```

```
extern "C" __global__ __declspec(dllexport) void UpdateOutputLayerWeights(
  double* outputErrors, double* hiddenLayerOutput, double* outputWeights,
  int hiddenSize, int outputSize, double learningRate)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < outputSize * hiddenSize)
  {
    int outputIdx = idx / hiddenSize;
    int hiddenIdx = idx % hiddenSize;

    double gradient = outputErrors[outputIdx] * hiddenLayerOutput[hiddenIdx];
    outputWeights[outputIdx * hiddenSize + hiddenIdx] += learningRate * gradient;
  }
}

extern "C" __global__ __declspec(dllexport) void UpdateHiddenLayerWeights(
  double* layerErrors, double* layerOutput, double* prevLayerOutput, double* layerWeights,
  int layerSize, int prevLayerSize, double learningRate)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < layerSize * prevLayerSize)
  {
    int layerIdx = idx / prevLayerSize;
    int prevLayerIdx = idx % prevLayerSize;

    double gradient = layerErrors[layerIdx] * prevLayerOutput[prevLayerIdx];
    layerWeights[layerIdx * prevLayerSize + prevLayerIdx] += learningRate * gradient;
  }
}

extern "C" __global__ __declspec(dllexport) void ComputeHiddenLayerErrors(
  double* nextLayerErrors, double* layerWeights, double* layerErrors,
  int layerSize, int nextLayerSize)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < layerSize)
  {
    double errorSum = 0.0;
    for (int nextIdx = 0; nextIdx < nextLayerSize; nextIdx++)
    {
      errorSum += nextLayerErrors[nextIdx] * layerWeights[nextIdx * layerSize + idx];
    }
    layerErrors[idx] = errorSum;
  }
}
```

ДОДАТОК В

Листи затвердження

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського державного університету науки і технологій
Анатолій РАДКЕВИЧ
20.01.25

РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОБРОБКИ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01433-01-ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
20.01.25

Керівник розробки
Вадим ГОРЯЧКІН
20.01.25

Виконавець
Дмитро ЧЕРКАС
20.01.25

Нормоконтролер
Світлана ВОЛКОВА
20.01.25

2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського державного університету науки і технологій
Анатолій РАДКЕВИЧ
20.01.25

РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОБРОБКИ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01433-01 12 01-ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
20.01.25

Керівник розробки
Вадим ГОРЯЧКІН
20.01.25

Виконавець
Дмитро ЧЕРКАС
20.01.25

Нормоконтролер
Світлана ВОЛКОВА
20.01.25

2025