

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЯВЛЕННЯ
ДЕФЕКТІВ НА МЕТАЛЕВИХ ПОВЕРХНЯХ
SOFTWARE FOR METAL SURFACES
DEFECTS DETECTION

Виконав: студент(ка) 4 курсу, групи КНТ-137
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Козлов В.В.

(прізвище та ініціали)

Керівник Олійник А.О.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

МИНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІПРЕ, ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н., проф.

С.О. Субботін

“ 20 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Козлова Валерія Валентиновича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для виявлення дефектів на металевих поверхнях. Software for Metal Surfaces Defects Detection

керівник проєкту (роботи) Олійник Андрій Олександрович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “ 30 ” березня 2021 року № 103

2. Срок подання студентом проєкту (роботи) 17 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)1. Аналіз предметної області. 2. Проектування архітектури програми. 3. Реалізація компонентів програми. 4. Тестування та експериментальне дослідження програми. 5. Керівництво програміста. 6. Керівництво оператора.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	Олійник А.О., доцент		
Нормоконтроль	Андреєв М. О., асистент		

7. Дата видачі завдання “11” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Розділ 5, Розділ 6, Додатки
7	Захист роботи	9 тиждень	

Студент(ка)

Козлов В.В.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

Олійник А.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 125 с., 15 табл., 60 рис., 2 дод., 32 джерела.

МАШИННЕ НАВЧАННЯ, ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, VGGNET, VGG, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВИЯВЛЕННЯ ДЕФЕКТІВ, PYTHON, TENSORFLOW, KERAS, PYQT, MATPLOTLIB.

Об'єкт дослідження – процес виявлення дефектів на металевих поверхнях.

Предмет дослідження – методи та алгоритми виявлення дефектів на металевих поверхнях.

Мета роботи – розробка програмного забезпечення для виявлення дефектів на металевих поверхнях за допомогою методів в основу яких закладено використання нейромережевих технологій.

Матеріали, методи та технічні засоби: об'єктно орієтоване та процедурне програмування, мова програмування Python, бібліотеки Keras, Tensorflow, PyQt та Matplotlib, персональний комп'ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 10.

Результати. Створено програмне забезпечення для виявлення дефектів на металевих поверхнях. Проведено порівняння ефективності нейромережевих моделей VGG для вирішення задачі виявлення дефектів на металевій поверхні сталевої гарячекатаної смуги.

Висновки. Розроблене програмне забезпечення надає гнучкі можливості навчання нейромережевих моделей сімейства VGGNet, що дозволяє досягти високої точності розпізнавання типів дефектів виявленіх на металевій поверхні під час технологічного процесу.

Галузь використання – виявлення дефектів металевої поверхні під час контролю якості продукції на промислових підприємствах.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
125 pages, 60 figures, 15 tables, 2 appendixes, 32 sources.

MACHINE LEARNING, ARTIFICIAL NEURAL NETWORKS,
CONVOLUTIONAL NEURAL NETWORKS, IMAGE RECOGNITION,
VGGNET, VGG, SOFTWARE, DEFECTS DETECTION, PYTHON,
TENSORFLOW, KERAS, PYQT, MATPLOTLIB.

The object of research is the process of metal surfaces defects detection.

The subject of research is methods and algorithms for metal surfaces defects detection.

The goal of the work is the development of software for metal surfaces defects detection using methods based on the use of neural network technologies.

Materials, methods and technical means: object-oriented and procedural programming, Python programming language, Keras, Tensorflow, PyQt and Matplotlib libraries, personal computer with Intel Core i5 processor under the control of the Microsoft Windows 10 operating system.

Results. The software for metal surfaces defects detection has been created. The efficiency of VGG neural network models is compared to solve the problem of metal surface defects detection on steel hot-rolled strip.

Conclusions. The developed software provides flexible training opportunities for VGGNet family neural network models, which allows to achieve high accuracy of recognition metal surface detected defects types during the technological process.

The field of application is metal surfaces defects detection during quality control of production at the industrial enterprises.

ЗМІСТ

	C.
Перелік скорочень та умовних познак	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Огляд задачі виявлення дефектів на металевих поверхнях	10
1.1.1 Актуальність та необхідність виявлення дефектів	10
1.1.2 Види дефектів	11
1.1.3 Методи та підходи до контролю якості металопродукції.....	12
1.2 Огляд нейронних мереж та їх видів	16
1.2.1 Нейронні мережі.....	16
1.2.2 Види нейронних мереж	19
1.2.3 Види згорткових нейронних мереж	21
1.3 Дослідження архітектур нейромережевих моделей VGGNet сімейства	23
1.3.1 Згорткові шари	24
1.3.2 Агрегувальні шари	26
1.3.3 Повнозв'язні шари	28
1.3.4 Функція активації нейронів	30
1.4 Аналіз існуючих програмних реалізацій	33
1.5 Висновки за розділом 1	34
2 Проектування архітектури програми	36
2.1 Аналіз вимог до програмного забезпечення	36
2.2 Вибір засобів розробки програмного забезпечення	38
2.2.1 Вибір мови програмування	38
2.2.2 Вибір бібліотеки для машинного навчання.....	42
2.2.3 Вибір бібліотеки для побудови графічного інтерфейсу	43
2.2.4 Вибір бібліотеки для відображення графічних даних.....	45
2.3 Розробка прототипів графічного інтерфейсу	46
2.4 Висновки за розділом 2	47
3 Реалізація компонентів програми.....	48

3.1 Реалізована структура програми	48
3.2 Реалізовані класи програми	49
3.3 Реалізація нейромережевих моделей VGG.....	52
3.4 Набір навчальних даних та їх структура	55
3.5 Реалізація графічного інтерфейсу користувача	56
3.6 Інші особливості програмної реалізації.....	58
3.7 Висновки за розділом 3	59
4 Тестування та експериментальне дослідження програми	60
4.1 Тестування програмного забезпечення.....	60
4.2 Експериментальне дослідження програми.....	63
4.3 Висновки за розділом 4	66
5 Керівництво програміста.....	67
5.1 Призначення та умови застосування програми	67
5.2 Вимоги до програмного забезпечення	68
5.3 Характеристики програми.....	68
5.4 Звернення до програми.....	69
5.5 Вхідні та вихідні дані програми	70
5.6 Повідомлення програмісту.....	70
6 Керівництво оператора	71
6.1 Призначення програми	71
6.2 Умови виконання програми	71
6.3 Виконання програми.....	72
6.4 Повідомлення оператору	75
Висновки	76
Перелік джерел посилання	78
Додаток А Текст програми.....	81
Додаток Б Слайди презентації	116

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – Application Programming Interface;
- IDE – Integrated Development Environment;
- IT – Information Technology;
- UI – User Interface;
- БД – база даних;
- ГМК – гірничо-металургійний комплекс;
- ДСТУ – державний стандарт України;
- НМ – штучні нейронні мережі;
- ОЗП – оперативно запам'ятовуючий пристрій;
- ОС – операційна система.

ВСТУП

Сьогодні комп'ютерні засоби застосовуються для модернізації й інтелектуалізації різних процесів діяльності людини. Завдяки впровадженню новітніх технологій спостерігається стрімке зростання обсягів виробництва металевої продукції на заводах гірничу-металургійного комплексу України, що неодмінно впливає на якість виготовленої продукції та наявність різних дефектів на її поверхні.

Існуючі методи виявлення дефектів, які застосовуються під час виробництва, не характеризуються високою швидкістю роботи та є ресурсоємними. Тому необхідно запровадження швидких та досить точних методів в основу яких закладено використання нейромережевих технологій.

Згорткові нейронні мережі, які можуть бути використані для аналізу даних отриманих з датчиків мають певні структурні особливості, на які необхідно звертати увагу під час реалізації відповідної програмної системи. Серед згорткових нейронних мереж значно виділяються нейромережеві моделі сімейства VGGNet, які характеризуються високою точністю обробки й розпізнавання зображень.

Відсутність експериментальних досліджень можливості застосування VGG мереж для виявлення дефектів на металевих поверхнях, а також детального порівняння ефективності їх роботи сприяє тому, що досі використовуються застарілі методи. Розробка програмного забезпечення для оцінки якості продукції за допомогою нейромережевих технологій та подальше порівняння ефективності нейромережевих моделей VGG дозволить змінити існуючі тенденції та сприятиме впровадженню новітніх методів оцінки якості продукції в технологічний процес.

Мета роботи – розробка програмного забезпечення для виявлення дефектів на металевих поверхнях за допомогою методів в основу яких закладено використання нейромережевих технологій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд задачі виявлення дефектів на металевих поверхнях

1.1.1 Актуальність та необхідність виявлення дефектів

Розвиток та використання сучасних комп’ютерних технологій в різних сферах діяльності людини є рушієм технічного прогресу. Щодня все більша кількість новітніх пристройів та алгоритмів стає невід’ємною частиною нашого життя, особливо це стосується сфери інформаційних технологій.

Наряду з впровадженням сучасних технологій в провідних ІТ компаніях, відбувається модернізація й інтелектуалізація виробництва на заводах-гігантах металургійної промисловості України. Серед таких підприємств можна виділити комбінати ПАТ «Запоріжсталь» та ПрАТ «Дніпропрєсталь», які знаходяться в Запоріжжі та є одними з найбільших у країні [1].

Завдяки автоматизації та інтелектуалізації процесу виробництва, що впроваджуються протягом останніх років на різних заводах гірничо-металургійного комплексу України, вже в першому кварталі 2021 року вдалося збільшити обсяг виробництва різних видів продукції в середньому на 1,7 % у порівнянні з першим кварталом минулого року (табл. 1.1) [2].

Таблиця 1.1 – Підсумки роботи ГМК України в 2021 році

Продукція	За 3 місяці 2020, тис. т	За 3 місяці 2021, тис. т	Зсув, %
Агломерат	7 858	8 220	+4,6
Кокс	2 451	2 420	-1,3
Чавун	5 096	5 313	+4,3
Сталь	5 318	5 291	-0,5
Металопрокат	4 670	4 734	+1,4

До того ж за офіційною інформацією, представленою Державною службою статистики України, металургійним підприємствам вдалося

збільшити обсяг реалізованої металургійної продукції в 2020 році до 371 655,4 млн. грн, в порівнянні з 277 547,4 млн. грн в 2014 році [3].

Метал є важливою сировиною для промисловості. Його виготовлення та обробка являє собою складний технологічний цикл. Стрімке збільшення обсягів виготовлення сприяє підвищенню вимог до якості продукції, яка слугує запорукою конкурентоспроможності і прибутковості виробництва.

Під час виготовлення метал з поверхневими дефектами повинен відбраковуватися щоб уникнути подальших помилок, а раннє виявлення має зменшити пошкодження та виробничу вартість виробів. Також важливе значення має правильна класифікація виявлених дефектів, адже це дає можливість усунення причини їх утворення.

Тому можна сказати, що задача вивчення якості металевої поверхні та виявлення дефектів на ній – це одна з найважливіших інженерних задач для металургійної промисловості України [4].

1.1.2 Види дефектів

Уявлення про якість металевої продукції дає стан її поверхні, який оцінюється за відсутністю або наявністю дефектів. Згідно з ДСТУ 2860 [5], дефект – це кожна окрема невідповідність об'єкта встановленим вимогам. Під час виготовлення та обробки металевої продукції, шляхом спеціальної прокатки на станках, можуть з'являтися різні дефекти, що обумовлено особливостями процесу та властивостями сировини.

Відповідно до ДСТУ 2658-94 [6], класифікують 64 види дефектів поверхні металевої продукції після прокату, які розділяють на три групи з урахуванням стадії металургійного виробництва і джерела виникнення (рис. 1.1), а саме:

- дефекти поверхні, обумовлені якістю зливка або литої заготовки;
- дефекти поверхні, що утворились в процесі деформування;
- дефекти поверхні, що утворились під час операцій доробки.



Рисунок 1.1 – Класифікація дефектів металевої поверхні

Одні види дефектів мають суттєві відмінності у зовнішньому вигляді, а інші на перший погляд майже однакові. Типовими дефектами які виникають на виробництві вважають: включення, забруднення, вкатану окалину та подряпини (рис. 1.2) [7].

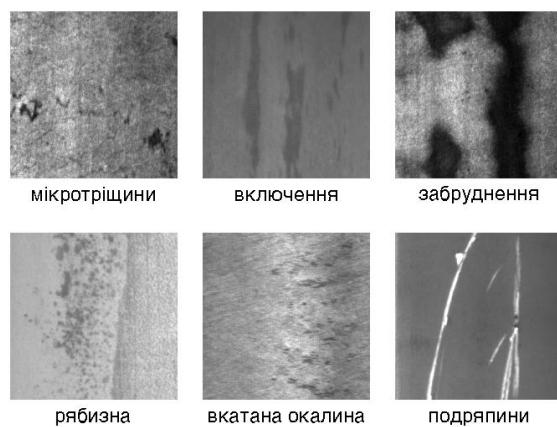


Рисунок 1.2 – Типові дефекти металевої поверхні [8]

1.1.3 Методи та підходи до контролю якості металопродукції

В процесі виробництва застосовують різні підходи до виявлення дефектів та контролю якості металопродукції, які поділяють на два основні види: методи руйнівного та неруйнівного контролю.

При руйнівному методі контролю оцінка якості виконується із застосуванням випробувань на контрольних зразках, що входять до складу виготовлених партій [9]. Зазвичай після завершення таких перевірок зразки стають непридатними для подальшої експлуатації, тому їх переробляють для повторного використання матеріалу або просто викидають. Це викликає додаткові трудові і матеріальні витрати і, крім того, не дає повної впевненості у високій якості всієї партії виробів.

При неруйнівному методі контролю оцінка якості може виконуватися на всіх виробах як при виготовленні, так і в процесі експлуатації без пошкодження зразків та зміни їх фізичних властивостей [9]. Особливістю такого підходу є те, що проведення обстежень може бути повністю автоматизовано, що сприяє зменшенню штату працівників та виключає людський фактор, за рахунок чого значно підвищується надійність контролюваних виробів.

Можливість впровадження автоматизації процесу та інші переваги неруйнівного контролю (табл. 1.2) якості продукції сприяють тому, що на виробництвах найчастіше застосовують саме цей метод.

Таблиця 1.2 – Порівняння руйнівного та неруйнівного контролю

Особливість	Руйнівний контроль	Неруйнівний контроль
Кількість перевірених зразків	Контрольний зразок	Вся партія
Стан зразка після контролю	Непридатний до використання	Придатний до використання
Повний аналіз фізичних властивостей	Так	Ні
Можливість автоматизації	Теоретично так	Так
Складність впровадження	Ні	Так
Матеріальні та трудові витрати	Більші	Менші
Час оцінки та отримання результатів	Довго	Швидко

Серед основних та перспективних підходів до неруйнівного контролю (рис. 1.3) на сьогодні виділяють такі: візуальний огляд, капілярний метод

магнітопорошковий метод, ультразвуковий метод та метод на основі використання нейромережевих технологій.



Рисунок 1.3 – Методи неруйнівного контролю

Найпростішим методом неруйнівного контролю є візуальний огляд, під час якого оцінка якості виробу здійснюється за допомогою неозброєного ока. У рідкісних випадках застосовують деякі оптичні засоби або мікроскопи. Спеціаліст оглядає виготовлений виріб з різних боків, порівнює його з еталонним зразком і виявляє невідповідності та видимі дефекти на поверхнях.

Капілярний метод використовують в тих випадках, коли застосування оптичних пристрій для проведення оцінювання якості неможливе, через недостатню контрастність дефекту на тлі металу. На виріб наносять індикаторну рідину та через деякий час протирають поверхню. Завдяки капілярному ефекту рідина проникає всередину дефекту та виділяє його [10]. Після таких процедур проводять візуальний огляд.

Магнітопорошковий метод ґрунтуються на виявленні магнітних полів, які виникають на поверхні намагніченого виробу в місцях, де є дефекти або включення з іншою магнітною проникливістю. В процесі контролю виріб намагнічують та наносять на поверхню феромагнітний порошок або суспензію. При наявності дефекту утворюється полюсність, а частинки

порошку осідають у вигляді візерунків, виділяючи місцерозташування, форму і довжину дефекту [10].

Ультразвуковий метод контролю використовується для пошуку дефектів шляхом випромінювання та прийняття ультразвукових коливань з подальшим аналізом їх амплітуди, часу приходу, форми та інших параметрів за допомогою спеціального обладнання – ультразвукового дефектоскопа. Принцип роботи цього методу ґрунтуються на властивості ультразвукових хвиль відбиватися від розділу середовищ з різними питомими акустичними опорами та не змінювати траєкторії руху в однорідному матеріалі [11].

Методи в основу яких закладено використання нейромережевих технологій базуються на отриманні даних з пристріїв неруйнівного контролю та їх подальшої обробки й інтерпретації за допомогою штучних нейронних мереж. В якості пристріїв неруйнівного контролю найчастіше використовують камери або різноманітні датчики. Такий підхід дозволяє пришвидшити процес оцінки якості та повністю автоматизувати його.

Кожен з розглянутих методів до виявлення дефектів та оцінювання якості виробу має як значні переваги так і певні обмеження в використанні та недоліки (табл. 1.3).

Таблиця 1.3 – Порівняння методів неруйнівного контролю

Особливість	Візуальний огляд	Капілярний метод	Магнітопорошковий метод	Ультразвуковий метод	Нейромережевий метод
1	2	3	4	5	6
Наявність спеціального обладнання	Hi	Hi	Так	Так	Так
Наявність витратних матеріалів	Hi	Так	Так	Hi	Hi
Залежність від навиків фахівця	Так	Так	Так	Так	Hi
Велика кількість задіяних фахівців	Так	Так	Так	Hi	Hi

Продовження таблиці 1.3

1	2	3	4	5	6
Обмеження по видам матеріалу	Hi	Hi	Так	Hi	Hi
Можливість автоматизації	Hi	Hi	Hi	Так	Так
Складність впровадження	Hi	Hi	Hi	Так	Так
Вартість впровадження	Низька	Середня	Середня	Висока	Висока
Вартість експлуатації методу	Висока	Середня	Середня	Низька	Низька
Час отримання оцінки	Середньо	Довго	Довго	Середньо	Швидко
Точність отриманого результату	Середня	Висока	Висока	Висока	Висока

Розвиток інформаційних технологій, підвищення технічних можливостей обчислювальної техніки та впровадження новітніх алгоритмів сприяють тому, що нейромережі стають потужним і універсальним інструментом для розпізнавання видимих дефектів. Велика кількість переваг нейромережевих методів над іншими методами неруйнівного контролю, а найголовніше точність отриманого результату, є запорукою їх розповсюдження в різних сферах діяльності людини. Саме тому використання нейромережевих методів для виявлення дефектів на металевих поверхнях є одним з найперспективніших напрямків розвитку галузі металургійної промисловості.

1.2 Огляд нейронних мереж та їх видів

1.2.1 Нейронні мережі

Штучні нейронні мережі – це математичні моделі, а також їх програмні або апаратні реалізації, побудовані за принципами подання й обробки інформації у біологічних нейронних мережах – мережах нервових кліток живого організму [12].

НМ використовують для вирішення складних інтелектуальних задач, що не мають точного алгоритмізованого методу вирішення і вимагають аналітичного підходу до виконання обчислень. Найпоширенішими

застосуваннями нейронних мереж є задачі: розпізнавання образів, класифікації, передбачення та кластеризації (рис. 1.4).



Рисунок 1.4 – Застосування нейронних мереж

Для того, щоб зрозуміти, як працюють нейронні мережі, необхідно розглянути їх складові та параметри.

НМ складаються зі штучних нейронів, структура яких запозичена в спрощеної моделі біологічного нейрона (рис. 1.5). За допомогою дендритів біологічний нейрон приймає сигнали з точок з'єднання, які називаються синапсами, оброблює їх, та, в тому випадку, якщо рівень сигналу перевищує деякий поріг посилає його по єдиному аксону на інші нейрони.

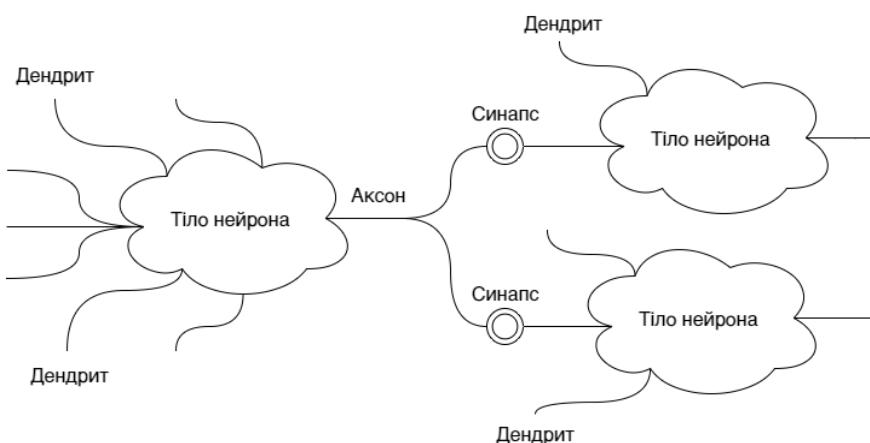


Рисунок 1.5 – Спрощена модель біологічного нейрона

Штучний нейрон (рис. 1.6) є примітивною обчислювальною одиницею, яка відповідає за прийом сигналу з багатьох входів, його подальшу обробку й передачу отриманого результату на єдиний вихід.

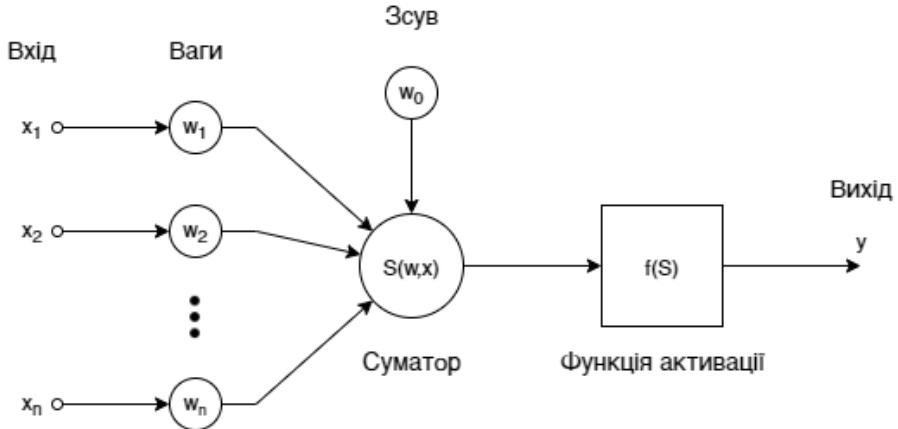


Рисунок 1.6 – Модель штучного нейрона

На вхід штучного нейрона подається набір сигналів, який представлено вектором $x = \{x_i\}, i = 1, 2, \dots, n$, де n – кількість вхідних сигналів. Кожен вхідний сигнал x_i зіставляється з відповідною до нього вагою зв'язку w_i з набору вагових коефіцієнтів $w = \{w_i\}$. Зважені вхідні сигнали та значення зсуву w_0 поєднуються в суматорі $S(w, x)$ та подаються до функції активації $f(S)$, яка повертає одне скалярне значення і подає його на вихід нейрона.

В якості суматора на практиці зазвичай використовується функція зваженої суми (1.1):

$$S(w, x) = \sum_{i=1}^n w_i x_i + w_0 \quad (1.1)$$

Сукупність вагових коефіцієнтів нейронної мережі уособлює мозок усієї системи [13]. Завдяки цьому набору, вхідна інформація обробляється і перетворюється в необхідний результат, а процес навчання НМ зводиться до підбору таких вагових коефіцієнтів, які надавали би якомога краще вихідне значення.

1.2.2 Види нейронних мереж

Будь-яку нейронну мережу утворюють за допомогою об'єднання штучних нейронів в шари та встановлення різних типів зв'язків між ними (рис. 1.7). Зазвичай виділяють три види шарів: входний, приховані та вихідний.

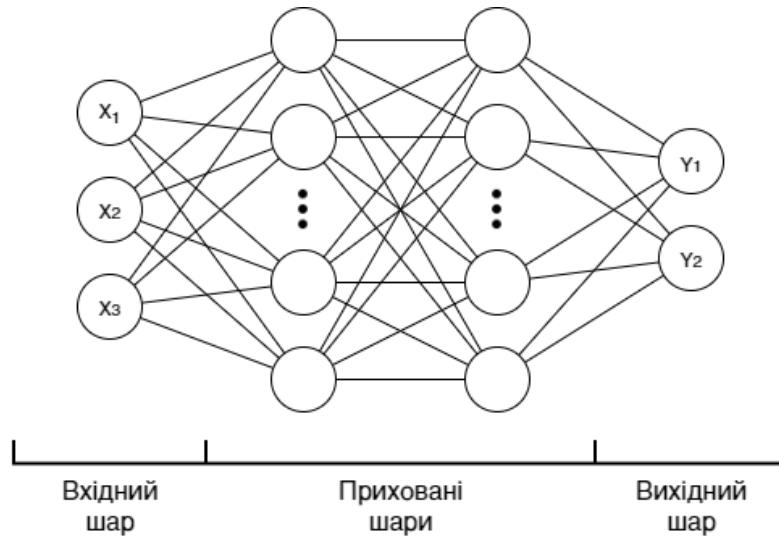


Рисунок 1.7 – Узагальнена структура нейронної мережі

Змінюючи комбінації нейронів в шарах та правила їх взаємодії утворюють різні архітектури мереж (рис. 1.8), тим самим пристосовуючи їх до вирішення різноманітних типів задач.



Рисунок 1.8 – Топологія архітектур нейронних мереж

Нейронні мережі прямого поширення та персепtronи дуже прямолінійні. Вони передають інформацію від входного шару до вихідного. Якщо у такої мережі є достатня кількість прихованих шарів нейронів, вона теоретично здатна знайти зв'язок між входними та вихідними даними. На практиці такі мережі використовуються рідко. Їх часто комбінують з іншими типами мереж для отримання нових архітектур.

В нейронних мережах зі зворотніми або латеральними зв'язками зв'язки між елементами утворюють спрямовану послідовність. Це надає можливість використовувати внутрішню пам'ять для обробки наборів даних довільної довжини [14]. Такі мережі широко використовуються для моделювання послідовних даних при вирішенні задач розпізнавання мовлення і рукописного тексту.

Нейро-нечіткі мережі комбінують методи НМ і систем нечіткої логіки. Вони дозволяють представити досліджувану модель у вигляді таблиці або бази правил «якщо – то». Особливістю таких мереж є можливість інтерпретації отриманої моделі, тому їх використовують в задачах вивчення середовища та взаємодії об'єктів.

Серед глибинних нейронних мереж існує багато архітектур. Яскравим представником яких є згоркові нейронні мережі. Вони можуть приймати входне зображення, призначати важливість різним аспектам або об'єктам на зображені та здатні диференціювати один об'єкт від іншого за рахунок механізмів, подібних зоровій корі [15]. Такі мережі добре пристосовані для вирішення задач розпізнавання, сегментації та класифікації зображень.

Розглянуті види мереж мають різну архітектуру, принцип взаємодії, вагу, час навчання та роботи. Тому необхідно обирати саме такий вид архітектури НМ, який надає якомога більше переваг, для вирішення конкретного типу завдання (табл. 1.4).

Так як при виявленні дефектів на металевих поверхнях за допомогою методів в основу яких закладено використання нейромережевих технологій спираються на аналіз графічної інформації отриманої за допомогою камер, то

необхідно використовувати таку архітектуру, яка пристосована для обробки та класифікації зображень. Тому для вирішення такої задачі слід застосовувати згорктоіві нейронні мережі.

Таблиця 1.4 – Порівняння архітектур нейронних мереж

Особливість	Прямого поширення	Зі зворотніми або лагеральними зв'язками	Нейро-нейткі	Згорткові
Інформація про архітектури в мережі Інтернет	Багато	Середньо	Мало	Багато
Складність реалізації	Легко	Середньо	Складно	Середньо
Вага мережі	Легка	Середня	Середня	Важка
Час навчання	Швидко	Середньо	Середньо	Довго
Час роботи	Швидко	Середньо	Середньо	Довго
Робота з послідовними даними	Hi	Так	Hi	Hi
Класифікація зображень	Теоретично	Теоретично	Hi	Так
Розпізнавання зображень	Hi	Hi	Hi	Так

1.2.3 Види згорткових нейронних мереж

Згорткова нейронна мережа або convolutional neural network – це спеціальна архітектура НМ, яка була запропонована Яном Лекуном в одному з випусків научного журналу «Neural Computation» в 1989 році [16]. Головною особливістю згорткової нейронної мережі є наявність операції згортки [12]. В такій архітектурі кожен мережевий рівень виступає в якості фільтра виявлення на наявність певних ознак або шаблонів, присутніх у вхідних даних які представлено зображенням.

Згорткова нейронна мережа є багатошаровою мережею прямого поширення без зворотних зв'язків. В загальному випадку її структура

складається з таких шарів (рис. 1.9): вхідний (input), згортковий (convolutional), агрегувальний (pooling), згладжувальний (flatten), повнозв'язний (fully connected) та вихідний (output).

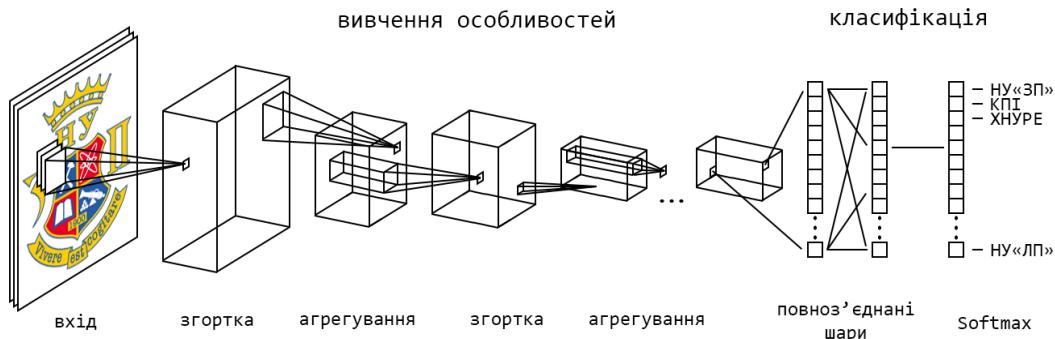


Рисунок 1.9 – Загальна архітектура згорткової нейронної мережі

Існує велика кількість видів архітектур згорткових нейронних мереж: LeNet, AlexNet, ZFNet, GoogleNet, VGGNet та інші. Вони різняться між собою розміром вхідного зображення, глибиною кольору, кількістю згорткових шарів та розмірами ядер згортки (табл. 1.5).

Таблиця 1.5 – Порівняння архітектур згорткових нейронних мереж

Особливість	LeNet-5	AlexNet	ZFNet	GoogleNet	VGG-16
Рік представлення	1989	2012	2013	2014	2014
Розмір зображення	32x32	227x227	224x224	224x224	224x224
Глибина кольору	1	3	3	3	3
Кількість згорткових шарів	3	5	5	-	13
Наявність inception модуля	Hi	Hi	Hi	9 модулів	Hi
Розмір ядер згортки	5x5	11x11	7x7	5x5, 3x3 та 1x1	3x3
Функція активації нейронів	Tanh	ReLU	ReLU	ReLU	ReLU
Функція агрегування	Average	Maximum	Maximum	Maximum, Average	Maximum
Загальна кількість шарів	5	8	8	22	16
Складність реалізації	Легко	Середньо	Середньо	Складно	Середньо
Час навчання	Швидко	Довго	Довго	Середньо	Довго
Точність на наборі даних ImageNet	-	83,6%	88,3%	93,3%	92,7%

Зважаючи на те, що нейромережеві моделі VGG сімейства можуть працювати з кольоровими зображеннями, мають просту архітектуру, легкі в

реалізації та не поступаються в точності класифікації та розпізнавання більш складним архітектурам, доцільно використовувати саме їх для вирішення поставленої задачі.

1.3 Дослідження архітектур нейромережевих моделей VGGNet сімейства

Сімейство моделей VGGNet є підвидом згорткових нейронних мереж. Його архітектура була запропонована вченими Оксфордського університету Кареном Сімоняном та Ендрю Зіссерманом в статті «Very Deep Convolutional Networks for Large-Scale Image Recognition» 2014 року [17]. Ефективність та перспективність використання саме VGG архітектури для вирішення задачі класифікації та локалізації зображень доведена в змаганні ILSVRC [18], де моделі сімейства VGGNet досягли одного з найкращих результатів точності.

Сімейство моделей VGGNet складається з шести архітектур. Переважна відмінність між ними полягає в кількості застосованих згорткових шарів та розмірів використованого ядра згортки (рис. 1.10).

VGG-11	VGG-11 (LRN)	VGG-13	VGG-16-1	VGG-16	VGG-19
11 вагових шарів	11 вагових шарів	13 вагових шарів	16 вагових шарів	16 вагових шарів	19 вагових шарів
Вхідний шар (224 × 224 RGB зображення)					
Згортка 3×3×64	Згортка 3×3×64 LRN	Згортка 3×3×64 Згортка 3×3×64	Згортка 3×3×64 Згортка 3×3×64	Згортка 3×3×64 Згортка 3×3×64	Згортка 3×3×64 Згортка 3×3×64
Максимізаційне агрегування					
Згортка 3×3×128	Згортка 3×3×128	Згортка 3×3×128 Згортка 3×3×128	Згортка 3×3×128 Згортка 3×3×128	Згортка 3×3×128 Згортка 3×3×128	Згортка 3×3×128 Згортка 3×3×128
Максимізаційне агрегування					
Згортка 3×3×256 Згортка 3×3×256	Згортка 3×3×256 Згортка 3×3×256	Згортка 3×3×256 Згортка 3×3×256	Згортка 3×3×256 Згортка 3×3×256 Згортка 1×1×256	Згортка 3×3×256 Згортка 3×3×256 Згортка 3×3×256	Згортка 3×3×256 Згортка 3×3×256 Згортка 3×3×256 Згортка 3×3×256
Максимізаційне агрегування					
Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512 Згортка 1×1×512	Згортка 3×3×512 Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512 Згортка 3×3×512 Згортка 3×3×512
Максимізаційне агрегування					
Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512 Згортка 1×1×512	Згортка 3×3×512 Згортка 3×3×512 Згортка 3×3×512	Згортка 3×3×512 Згортка 3×3×512 Згортка 3×3×512 Згортка 3×3×512
Повноз'єднаний шар ×4096					
Повноз'єднаний шар ×4096					
Повноз'єднаний шар ×1000					
Нормалізована експоненціальна функція (Softmax)					

Рисунок 1.10 – Архітектура моделей VGGNet

В якості вхідних даних кожна з моделей приймає кольорове зображення розміру 224x224 пікселів. Вхідне зображення проходить через послідовність згортальних шарів, в яких використовуються фільтри з дуже маленьким рецептивним полем, та агрегувальних шарів, де здійснюється просторова агрегація. На виході зі стеку згортальних та агрегувальних шарів виділені особливості зображення передаються на три повнозв'язних шари. В результаті застосування функції softmax представляється набір значень ймовірності приналежності зображення до того чи іншого класу. Для навчання мережі використовується метод зворотного поширення помилки.

1.3.1 Згорткові шари

Головним структурним елементом будь-якої згорткової нейронної мережі є згорткові шари, в основу яких закладено потужну математичну операцію згортки (convolution) [19].

Процес згортки зображення виконується за допомогою фільтрів або ядер згортки, які представляються матрицею вагів. Зазвичай ядро згортки є квадратною матрицею $n * n$, де n – непарне число, яке характеризує розміри рецептивного поля нейрона. Протягом проходу фільтр переміщується над двовимірним зображенням з заданим значенням кроку. На кожному етапі виконується поелементна операція множення елементів фільтра з тією частиною вхідних даних, над якою він знаходиться, а потім підсумовуються отримані значення і записуються у відповідний елемент вихідної матриці.

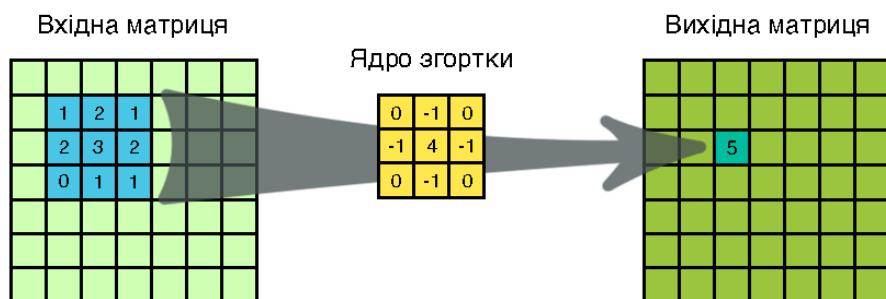


Рисунок 1.11 – Операція згортки

При обробці крайніх елементів матриці виникають певні труднощі. Крайні елементи ніколи не потрапляють в центр ядра згортки. Наприклад, для лівого верхнього пікселя не існує сусідніх лівих та верхніх елементів. Щоб уникнути таких ситуацій зазвичай використовують нульове доповнення (padding) [20], додаючи до країв матриці нульові значення (рис. 1.12).

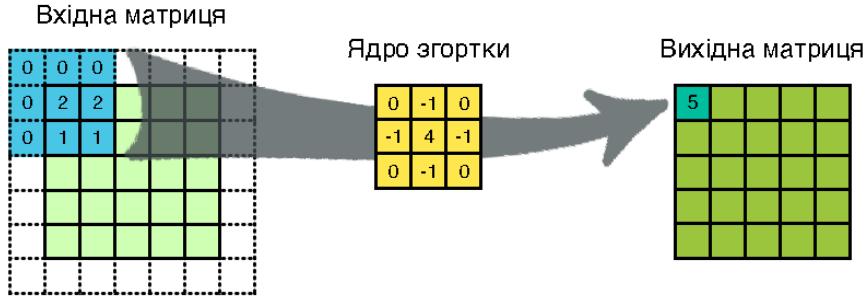


Рисунок 1.12 – Операція згортки з нульовим доповненням

Нехай $x_{i,j}$ – це елемент області рецептивного поля вхідної матриці, $w_{i,j}$ – ваговий множник ядра згортки, а n розмірність самого ядра, тоді операція згортки має такий вигляд (1.2):

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,n} \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{bmatrix} = \sum_{i=1}^n \sum_{j=1}^n x_{i,j} w_{i,j} \quad (1.2)$$

Вихідну матрицю називають картою активації. Великі значення в карті активації означають те, що рецептивне поле активує використаний фільтр, тобто особливість, яка визначається ядром згортки, наявна у вхідних даних.

Якщо кількість колірних каналів на зображення або розмірність вхідних даних дорівнює m (звичайне RGB зображення складається з трьох колірних каналів), то для отримання однієї карти активації необхідно використати m двовимірних фільтрів.

Всі фільтри на певній ітерації взаємодіють з однаковими вхідними даними та працюють незалежно один від одного. Збільшення кількості різних фільтрів допомагає отримати більше карт активації. Це допомагає виявляти на зображеннях різні шаблони, що облегшує процес подальшого аналізу даних (рис. 1.13). Кожен згортковий шар мережі представляє собою набір фільтрів особливостей зображення, рівень абстракції яких стає складнішим з кожним наступним шаром.

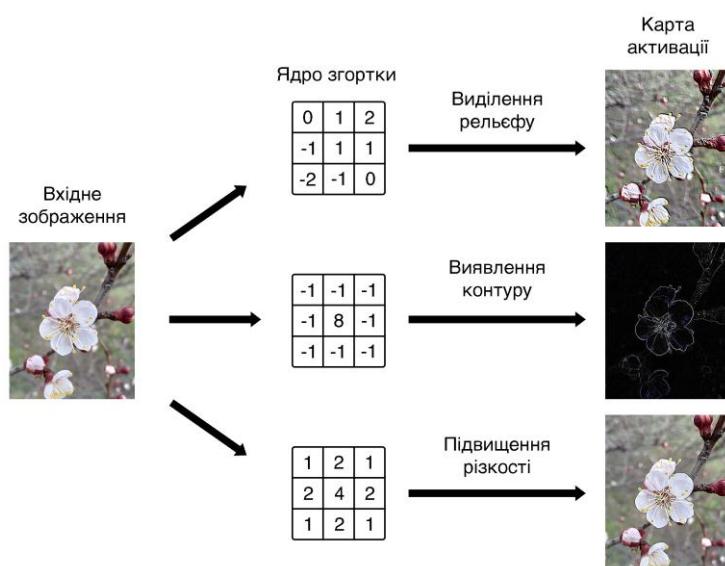


Рисунок 1.13 – Особливості зображення виявлені різними ядрами згортки

В сімействі моделей VGGNet використовуються фільтри з маленьким рецептивним полем розміру 3×3 . Перевагою таких фільтрів є те що вони мають меншу кількість параметрів для навчання та при поєднанні у три шари можуть досягати такого ж розміру рецептивного поля як і фільтри 7×7 . Тому використання таких маленьких фільтрів сприяє збільшенню глибини мережі, що значно підвищує її нелінійні можливості.

1.3.2 Агрегувальні шари

За стеком згорткових шарів розташовують агрегувальні шари, які обробляють отриманий набір виділених ознак. Мета використання

агрегувального шару – зменшення розмірності карт активації, отриманих на попередньому етапі.

Основна ідея полягає в тому, що після операції згортки отримується необхідний перелік ознак, який відображене в картах активації. Для подальшої обробки настільки докладне зображення стає вже не потрібним і його розмірність зменшують, фільтруючи надлишкові деталі [21].

Агрегувальний шар діє незалежно дляожної карти активації і зменшує її просторовий розмір. Важливою особливістю є те, що в процесі сканування, ядро агрегувального шару ніколи не потрапляє на вже відскановану ділянку.

Зазвичай для агрегації будь-якої карти активації використовують ядро розміром 2×2 та кроком 2. Карта активації поділяється на осередки 2×2 елементів, з яких вибираються такі значення, що відповідають агрегувальній функції. Використання такого ядра допомагає зменшити розміри карт активації отриманих після згортки в два рази.

Існує дві основні агрегувальні функції: усереднювання та максимізаційне агрегування (рис. 1.14).

Усереднювання (average pooling) використовується для отримання усередненого значення активації з досліджуваної області (1.3):

$$x_{i,j} = \frac{1}{n^2} \sum_{k=0}^n \sum_{l=0}^n x_{(i+k)(j+l)}, \quad (1.3)$$

де $x_{i,j}$ – рівень активації нейрона з індексами (i, j) ;

n – розмір використаного ядра.

Максимізаційне агрегування (max pooling) використовується для отримання максимального значення активації, що спостерігалося в досліджуваній області (1.4):

$$x_{i,j} = \max_{k=0, l=0}^n \{x_{(i+k)(j+l)}\} \quad (1.4)$$

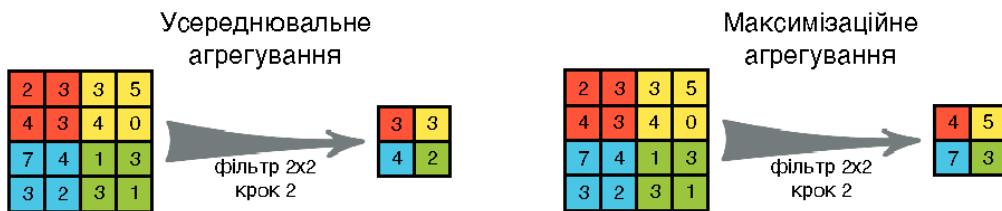


Рисунок 1.14 – Приклади агрегування

В сімействі моделей VGGNet в якості агрегувальної функції використовується максимізаційне агрегування з ядром розміру 2×2 та кроком 2. Використання такої функції дозволяє чітко виділяти виявлені ознаки на зображенні і передавати їх на інші шари, акцентуючи увагу на знайдених деталях.

1.3.3 Повнозв’язні шари

Завершальним кроком в мережевих моделях сімейства VGGNet, як і в інших згорткових нейронних мережах, є класифікація вхідного зображення за допомогою виявлених ознак. Класифікаційна частина мережі складається зі згладжувального та декількох повнозв’язних шарів [21].

На виході зі згорткових та агрегувальних шарів повертається набір карт активацій нейронів, який представляється у вигляді матриці значень. За допомогою згладжувального шару матрицю розтягають у векторний набір нейронів, який і стає вхідними даними для повнозв’язних шарів (рис. 1.15).

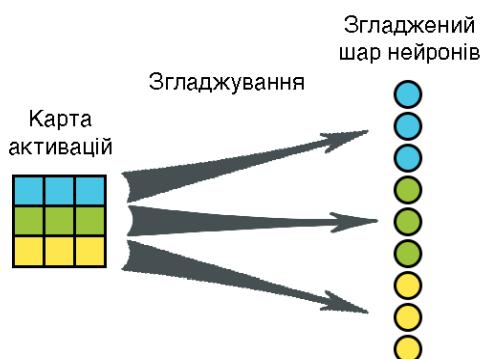


Рисунок 1.15 – Згладжування карти активацій

В якості повнозв'язних шарів використовується модель багатошарового персептрона, на вхід якого подається отриманий вектор з характеристиками виявлених особливостей на зображені. На вихідному шарі багатошарового персептрона знаходиться розподіл ймовірностей по класах для вхідного зображення, а кількість нейронів в ньому відповідає кількості класів, на які мережа може класифікувати його. Таким чином зображення класифікують вибравши найбільш ймовірний клас за допомогою нейрона з найбільшим значенням активації.

Узагальнену структуру повнозв'язних шарів згорткової нейронної мережі наведено на рисунку 1.16.

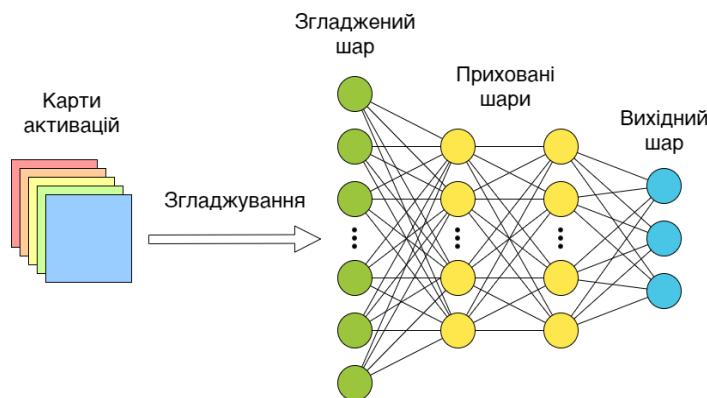


Рисунок 1.16 – Узагальнена структура повнозв'язних шарів

В сімействі моделей VGGNet використовується згладжувальний шар та багатошаровий персепtron з двома прихованими шарами, які містять по 4096 нейронів, та одним вихідним шаром з 1000 нейронами. Така структура обумовлена тим, що моделі VGG створювалися для вирішення задачі класифікації зображень на 1000 класів.

В залежності від складності вирішуваної задачі та кількості наявних класів у вибірці, змінюють структуру використованого багатошарового персептрона. Так, якщо у задачі, яка вирішується, зображення можна віднести до одного з шести класів, то у вихідному шарі слід використовувати 6 нейронів замість 1000. Тобто варто будувати мережу так, щоб на вихідному шарі кількість нейронів дорівнювала кількості класів.

1.3.4 Функція активації нейронів

Після надходження вхідного сигналу та розрахунку зваженої суми за допомогою суматора на виході нейрона формується деяке скалярне значення яке може бути будь-яким в діапазоні $(-\infty; +\infty)$. Так як нейрон не знає порогу, при перевищенні якого він має бути активований, тому отримане значення необхідно нормалізувати. Для цього використовують функцію активації.

Функція активації – це математичне перетворення, яке використовується в штучних нейронних мережах для нормалізації даних. Вона перетворює вироблене нейроном значення в необхідний обмежений числовий діапазон тим самим вказуючи зовнішнім зв'язкам активований нейрон чи ні [22].

Існує достатня кількість функцій активації. Головна їх відмінність – це діапазон отримуваних значень. Виділяють такі основні функції: порогова, лінійна, сигмоїдна, гіперболічний тангенс та ReLU.

Порогова функція є найпростішою функцією активації та використовується лише для того, щоб показати активовано нейрон чи ні, перетворюючи його значення на 1 та 0 (1.5). Так як в задачах класифікації не можна зі 100% впевненостю стверджувати до якого класу належить досліджуваний об'єкт, то таку функцію зазвичай не використовують:

$$f(x) = \begin{cases} 1, & x \geq a \\ 0, & x < a \end{cases}, \quad (1.5)$$

де a – деяке порогове значення.

Лінійна функція використовується в тих випадках, коли необхідно протестувати нейронну мережу та передати значення без нормалізації (1.6). На практиці використання лінійної функції обмежено, адже її похідна є постійним числом, що впливає на процес навчання нейронної мережі методом зворотного поширення помилки, а використання ясукупності шарів

з лінійними функціями активації призводить до того, що багатошаровий персепtron стає еквівалентним одношаровому:

$$f(x) = x \quad (1.6)$$

Сигмоїдна або логістична функція є найбільш популярною за використанням функцією активації в нейронних мережах. Вона доволі часто використовується в задачах класифікації, оскільки приймає значення в діапазоні $(0; 1)$ та має гладкий вид (1.7) [23]:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1.7)$$

Гіперболічний тангенс повертає значення в діапазоні $(-1; 1)$ та використовується в тих випадках, коли вихідні значення можуть бути від'ємними (1.8). Основним недоліком гіперболічного тангенса, як і сигмоїдної функції є те, що розрахунок значення є доволі витратною операцією:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (1.8)$$

ReLU є нелінійною функцією активації, яка має простий математичний вигляд (1.9) та вимагає менше обчислювальних ресурсів, що суттєво прискорює час навчання глибинної нейронної мережі [24]:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (1.9)$$

Графічний вигляд проаналізованих функцій активації наведено на рисунку 1.17.

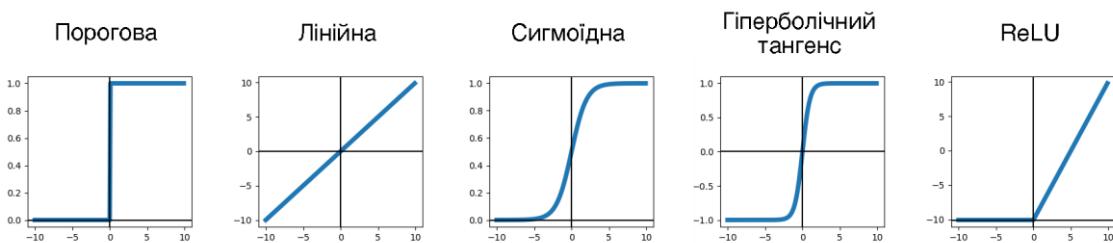


Рисунок 1.17 – Функції активації

В якості функцій активації нейронів в сімействі моделей VGGNet після кожного зі згорткових та агрегувальних шарів застосовується функція ReLU. У порівнянні з іншими функціями активації (табл. 1.6) вона володіє усіма перевагами нелінійності, вимагає менше обчислювальних ресурсів, швидко розраховується та надає можливість виключати неактивовані нейрони з подальших розрахунків.

Таблиця 1.6 – Порівняння функцій активації

Особливість	Функція активації				
	Порогова	Лінійна	Сигмоїдна	Гіперболічний тангенс	ReLU
Діапазон значень	{0, 1}	($-\infty$; $+\infty$)	(0; 1)	(-1; 1)	[0; $+\infty$)
Нелінійність	Так	Ні	Так	Так	Так
Не захоплює від'ємні значення	Так	Ні	Так	Ні	Так
Швидкість розрахунку значення	Швидко	Швидко	Довго	Довго	Швидко
Відкидання не активованих нейронів	Так	Ні	Ні	Ні	Так
Ріномірність розподілення значень	Ні	Так	Ні	Ні	Так

Для вихідного шара в нейронних мережах сімейства VGG використовують функцію активації softmax (1.10):

$$f(x_i) = \frac{x_i}{\sum_{k=1}^K x_k}, \quad (1.10)$$

де x_i – значення активації відповідного нейрона;

K – кількість класів.

Softmax – це підвид логістичної функції активації, який використовується для розрахунку не одного значення, а цілого вектора при вирішенні задачі багатокласової класифікації.

Особливість використання функції softmax полягає в тому, що кожен нейрон вихідного шару буде видавать значення ймовірності приналежності об'єкта до певного класу, а сума значень всіх нейронів буде дорівнювати одиниці [25].

1.4 Аналіз існуючих програмних реалізацій

Серед програмного забезпечення яке можна знайти у відкритому доступі в мережі Інтернет не знайдено жодної програми, яка може використовуватися для вирішення задачі виявлення дефектів на металевих поверхнях. Це пов'язано з тим, що розробка такого типу програм достатньо ресурсоємний процес який вимагає багато технічних та матеріальних витрат. Компаніям, які задіяні в металургійній промисловості не вигідно поширювати свої наробки та тим паче виставляти їх у вільний доступ, надаючи можливість своїм конкурентам користуватися розробленими технологіями.

Проте в мережі Інтернет можна знайти програмну реалізацію різних нейронних мереж, які навчено для вирішення деяких типових задач комп'ютерного зору. Сімейство моделей VGGNet не є винятком. Моделі VGG представлено в різних бібліотеках і фреймворках машинного навчання для різних мов програмування.

Так моделі VGG16 та VGG19 представлена в пакеті прикладних програм для вирішення задач технічних обчислень Matlab та у відкритій нейромережевій бібліотеці Keras, а моделі VGG11 та VGG13 наведено у фреймворці машинного навчання PyTorch [26-28].

Використання реалізованих нейромережевих моделей з бібліотек та фреймворків для машинного навчання спрощує процес розробки, адже

можна використати вже навчену модель для вирішення простої задачі класифікації.

Моделі сімейства VGGNet створювалися та налаштовувалися для участі в змаганнях ILSVRC, де необхідно було віднести зображення з бази даних ImageNet [29] до одного з 1000 класів, тому в описаних вище реалізаціях всі моделі навчені тільки для вирішення цієї задачі. Проте, якщо необхідно пристосувати нейронну мережу для вирішення вузькоспеціалізованої задачі, то виникають труднощі, пов’язані зі зміною параметрів шарів та її перенавчанням.

Для звичайного користувача такий підхід ускладнюється тим, що для використання моделей необхідно володіти навичками програмування, адже взаємодія з ними в бібліотеках та фреймворках не передбачає наявності графічного інтерфейса, а всі налаштування необхідно виконувати вручну за допомогою програмного кода чи консолі.

Зважаючи на відсутність аналогів у вільному доступі в мережі Інтернет та особливості виявленіх програмних реалізацій, можна стверджувати, що вільне програмне забезпечення для виявлення дефектів на металевих поверхнях за допомогою нейромережевих технологій з простим та інтуїтивно зрозумілим графічним інтерфейсом користувача є перспективною та затребованою технологією серед невеликих компаній металургійної промисловості країни.

1.5 Висновки за розділом 1

Розвиток та використання сучасних комп’ютерних технологій сприяє модернізація та інтелектуалізації промислового та металургійного виробництва. Пришвидшення темпів та обсягів виробництва сировини спонукає впровадження новітніх засобів контролю якості продукції. Це сприяє тому, що однією з найважливіших інженерних задач стає задача вивчення якості металевої поверхні та виявлення дефектів на ній.

Методи виявлення дефектів в основу яких закладено використання нейромережевих технологій набувають широкої популярності в використанні. Протягом виконання аналізу предметної області проведено дослідження різних видів нейронних мереж. Так як при виявленні дефектів спираються на аналіз графічної інформації отриманої за допомогою камер то для вирішення такої задачі слід застосовувати згоркові нейронні мережі, в особливості нейромережеві моделі VGG.

Проведено докладний аналіз архітектури та основних властивостей нейронних моделей сімейства VGGNet, завдяки чому отримано необхідні теоретичні та практичні знання про процес розпізнавання зображень за допомогою згоркових нейронних мереж.

В результаті аналізу існуючих аналогів та особливостей виявленіх програмних реалізацій, можна зробити висновок, що розроблюване програмне забезпечення для вирішення задачі виявлення дефектів на металевих поверхнях за допомогою нейромережевих моделей VGG є актуальною та затребованої технологією.

Для досягнення мети роботи необхідно вирішити такі завдання:

- виконати аналіз вимог;
- обрати програмні засоби;
- виконати програмну реалізацію;
- провести тестування та експериментальне дослідження програми;
- розробити програмну документацію.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМИ

2.1 Аналіз вимог до програмного забезпечення

Невід'ємною частиною проєктування будь-якого програмного забезпечення є аналіз вимог до нього. Виходячи з проведеного аналізу предметної області та загальних вимог до сучасних комп'ютерних програм складено функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення, реалізація яких сприятиме можливості його використання як основного засобу контролю якості технологічного процесу під час виготовлення металевих виробів (рис. 2.1).



Рисунок 2.1 – Вимоги до програмного забезпечення

Для забезпечення можливості виявлення дефектів на металевій поверхні програмний засіб повинен надавати можливість завантаження зображення, вибору необхідної нейронної моделі та організації процесу класифікації з подальшим виведенням інформації про тип знайденого дефекта.

Оскільки виробництво та обробка металу є складним технологічним процесом а кількість виготовленої продукції та види матеріалів змінюються з

кожним роком, то програмний засіб повинен надавати можливість навчання та перенавчання нейромережевих моделей на нових або оновлених навчальних наборах даних.

Процес навчання вибраної мережі має супроводжуватися докладною статистичною та графічною інформацією про її поточні характеристики.

Програмне забезпечення повинне мати інтуїтивно зрозумілий графічний інтерфейс та високу швидкість відклику на дії користувача, бути стійким до виникаючих позаштатних ситуацій.

Програмна документація має надавати інструкцію з запуску та використання програми, описувати основні вимоги до апаратної та програмної частини використовуваного пристрою.

На рисунку 2.2 наведено Use Case діаграму розроблюваного програмного забезпечення. Вона відображає відносини між акторами і прецедентами, що дозволяє описати систему на концептуальному рівні, демонструючи основні варіанти використання програми та її реакції на дії користувача.

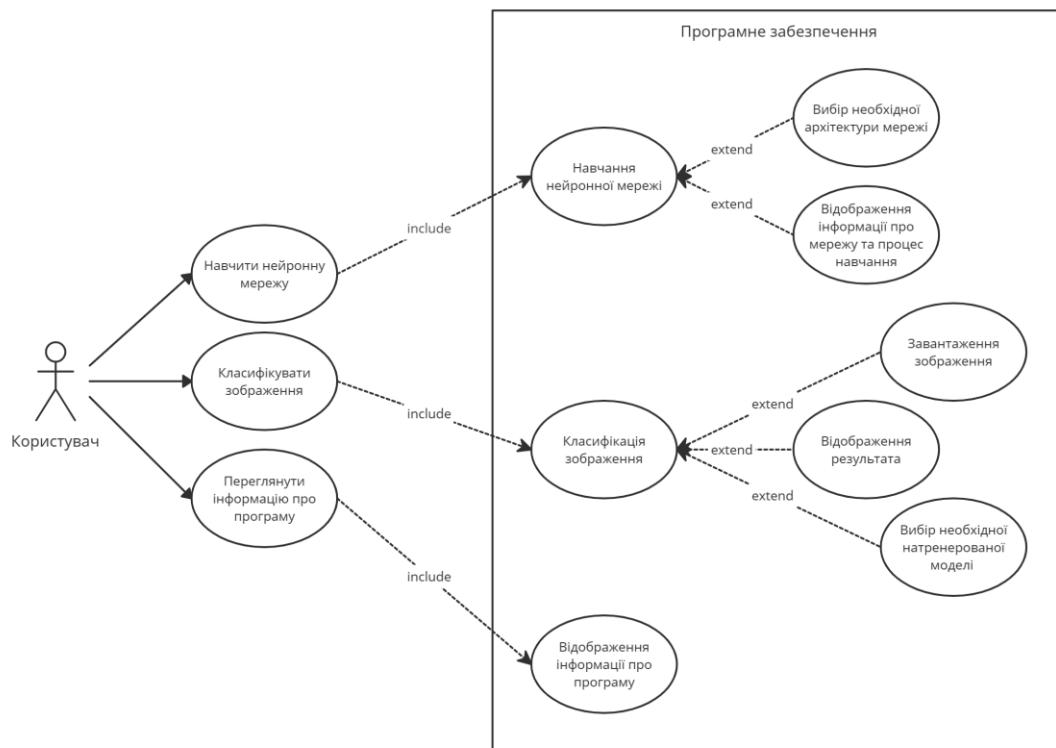


Рисунок 2.2 – Use Case діаграма програми

В такій діаграмі актором є користувач та його дії, а прецедентом – функціональна частина розробленого програмного забезпечення. Використання такої діаграми дозволяє виконати специфікацію зовнішніх вимог до системи.

Користувач при роботі з програмою ставить собі за мету виконання таких функцій: навчити нейронну мережу, подивитися статистику її навчання, класифікувати зображення та переглянути інформацію про програму. Взаємодіючи з графічним інтерфейсом програми (натискаючи на кнопки та інші елементи), саме він посилає сигнали програмному засобу.

Програмний засіб, отримуючи результати взаємодії користувача з графічним інтерфейсом, виконує необхідні дії над системою та викликає різні модулі: запускає програмний код, відкриває нові вікна програми, змінює елементи інтерфейсу, навчає мережі, завантажує та класифікує зображення.

2.2 Вибір засобів розробки програмного забезпечення

Процес розробки програмного забезпечення варто починати з вибору мови програмування, що зумовлює основні характеристики програми та можливості її застосування у відповідній галузі, та використовуваних існуючих бібліотек, які сприяють можливості повторного використання коду й спрощення процесу розробки. Такий підхід є найпопулярнішою практикою створення програмних засобів, адже зникає необхідність реалізації великої кількості аналогічних рішень, що дозволяє поглибитися в процес реалізації індивідуальних особливостей продукту.

2.2.1 Вибір мови програмування

Мова програмування є формальною мовою, що призначена для запису комп’ютерних програм. Мова програмування визначає набір лексичних,

синтаксичних та семантичних правил, що визначають зовнішній вигляд програми і дії, які виконує комп'ютер під її керуванням.

На даний момент з популярних мов програмування (рис. 2.3), які можуть бути використані для вирішення завдань машинного навчання можна виділити такі: C++, C#, R, Java та Python.

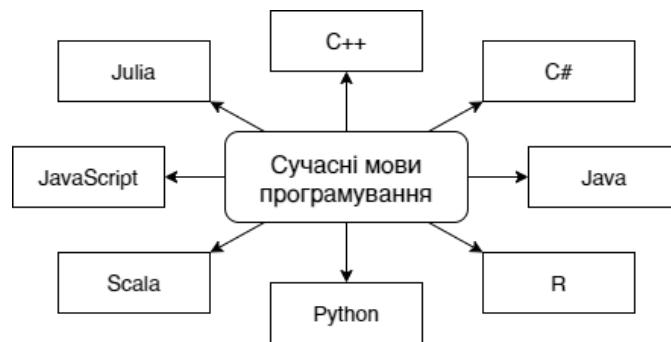


Рисунок 2.3 – Сучасні мови програмування

C++ є однією з найпізнаваніших мов програмування у світі. Це строго типізовані мова програмування високого рівня з підтримкою об'єктно-орієнтованої, узагальненої та процедурної парадигми програмування. Вона широко використовується для розробки прикладних програм, драйверів пристройів та ігор, а її основні переваги: швидкість виконання програмного коду, підтримка великої кількості сторонніх бібліотек та наявність докладної технічної документації. Останнім часом C++ починає закріплювати свої позиції в якості мови програмування для роботи з нейронними мережами завдяки анонсованій підтримці бібліотек машинного навчання TensorFlow, LightGBM та Turi Create.

Мова програмування C# (C Sharp), як і C++ є представником сімейства мов програмування C. Це строго типізовані об'єктно-орієнтована мова програмування. Її мета полягає в забезпеченні продуктивності роботи програмістів, що виражається в балансі між простотою, виразністю і швидкістю роботи. Вона підтримується компанією Microsoft та в основному використовується для створення, комп'ютерних ігор, веб-застосунків та програм під ОС Windows. Компанія Microsoft намагається просувати C# в

сферу машинного навчання та комп’ютерного зору додаючи нові бібліотеки, однією із яких є пакет для машинного навчання ML.NET.

Мова програмування R фактично підтримує процедурну й об’єктно-орієнтовану парадигми програмування та широко використовуються для статистичної обробки та аналізу даних. R найчастіше використовують за допомогою інтерфейса командного рядка або в якості допоміжних модулей для програм написаних на інших мовах.

Java є строго типізованою об’єктно-орієнтованою мовою програмування, яку використовують для розробки мобільних додатків під ОС Android та клієнт-серверних застосунків. Java розроблена як вільно пов'язана мова програмування, яка може працювати на будь-якій платформі. Використання цієї мови програмування в галузі машинного навчання зумовлено наявністю бібліотек SMILE та H2O, які можуть бути використані як бібліотеки для локальних та хмарних розрахунків.

Мова програмування Python – це інтерпретована високорівнева мова програмування з динамічною типізацією та підтримкою основних парадигм програмування: процедурного, функціонального, об’єктно-орієнтованого тощо [30]. За допомогою Python створюють веб-застосунки, комп’ютерні програми, оброблюють дані, розроблюють та навчають нейронні мережі. Використанню мови програмування Python в галузі машинного навчання сприяє наявність навчальних матеріалів, технічної документації та великої кількості потужних бібліотек.

Кожна з розглянутих мов програмування має певні переваги та недоліки, що зумовлює сферу її використання для розроблюваного програмного забезпечення (табл. 2.1).

Таблиця 2.1 – Порівняння мов програмування

Особливість	C++	C#	R	Java	Python
1	2	3	4	5	6
Рік розробки	1983	2001	1993	1995	1991
Інтерпретованість	Hi	Hi	Так	Hi	Так

Продовження таблиці 2.1

1	2	3	4	5	6
Динамічна типізація	Hi	Hi	Так	Hi	Так
Легкий інтуїтивний синтаксис	Hi	Середньо	Середньо	Середньо	Так
Підтримка об'єктно орієнтованої парадигми	Так	Так	Так	Так	Так
Підтримка процедурної парадигми	Так	Hi	Так	Hi	Так
Швидкість роботи	Швидко	Швидко	Середньо	Швидко	Середньо
Скільки пам'яті займає програма	Багато	Багато	Мало	Багато	Мало
Керування пам'яттю	Вручну	Вручну	Авто	Авто	Авто
Створення та стилізація графічного інтерфейсу	Так	Так	Hi	Так	Так
Бібліотеки для аналізу даних	Мало	Мало	Багато	Мало	Багато
Бібліотеки для машинного навчання	Мало	Мало	Середньо	Середньо	Багато
Легко встановити додаткову бібліотеку	Hi	Так	Hi	Hi	Так
Технічна документація та навчальні посібники	Багато	Багато	Середньо	Середньо	Багато

Відповідно до проведеного порівняльного аналізу мов програмування можна сказати, що легкий та інтуїтивний синтаксис, велика кількість бібліотек та програмної документації, набір попередньо налаштованих інструментів для впровадження алгоритмів обробки даних, створення та навчання штучних нейронних мереж виділяють Python серед конкурентів і роблять його ідеальним та універсальним інструментом для розробки програм в основі яких лежить використання машинного навчання. Тому прийнято рішення, що для реалізації розроблюваного програмного забезпечення варто використовувати мову програмування Python.

Серед існуючих інтегрованих середовищ розробки для створення програм, за допомогою Python, можна виділити IDE PyCharm, яке було розроблено міжнародною компанією JetBrains. PyCharm – це вільно розповсюджуване крос-платформне середовище розробки, яке сумісне з ОС Windows, Linux та Mac OS. Воно надає необхідний набір інструментів для аналізу та рефакторингу коду, запуску та тестування програм, використання різних систем контролю версій. Отже, враховуючи всі наявні функціональні можливості та відсутність аналогів, для розробки програмного забезпечення доцільним є використання середовища PyCharm.

2.2.2 Вибір бібліотеки для машинного навчання

Популярність мови програмування Python в сфері науки о даних обумовлена наявністю великої кількості спеціалізованих бібліотек для машинного навчання (рис. 2.4) які використовують як професіонали, так і початківці. Одні бібліотеки надають зручний прикладний програмний інтерфейс, інші взагалі написано на самій мові Python. Серед найчастіше використовуваних можна виділити такі: Scikit-learn, PyTorch, Keras та Tensorflow.

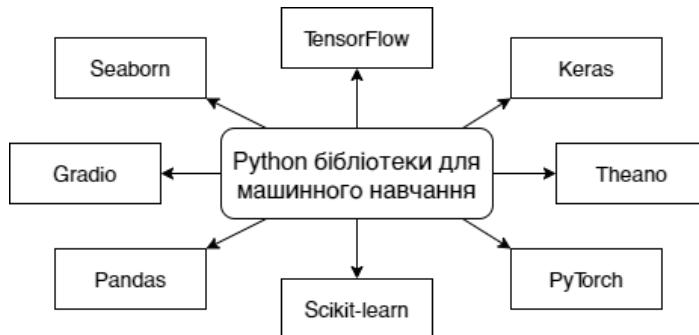


Рисунок 2.4 – Бібліотеки машинного навчання в Python

Бібліотека Scikit-learn надає достатню кількість інструментів для обробки й аналізу даних, зокрема для створення простих моделей машинного навчання. Її часто використовують під час навчання учнів та студентів для демонстрації основ машинної обробки даних під час вирішення типових задач: навчання з учителем, лінійна регресія, кластеризація тощо.

Бібліотека PyTorch, що створена в Facebook і написана на Python, використовується для глибинного навчання. Вона являє собою простий API та надає можливості проводити розрахунки за допомогою ядер графічного процесора.

Бібліотека Keras є специфікацією прикладного програмного інтерфейсу, яка описує, як система глибокого навчання повинна реалізовувати певну частину, пов’язану з визначенням моделі та її навчанням. Вона містить реалізацію широко вживаних нейромережевих

моделей та їх складових частин: специфікації шарів, активаційні, цільові та передавальні функції, методи оцінювання якості навчання та оптимізації. Використовується бібліотека тільки у зв'язці з Tensorflow.

Розроблена компанією Google бібліотека Tensorflow є найпопулярнішою бібліотекою машинного навчання в сучасному світі [31]. Вона реалізовує багатонивесість за допомогою використання тензорів для організації процесу обчислення різних операцій на одних вхідних даних та дозволяє проводити навчання нейромережевих моделей на центральному та графічному процесорі. Tensorflow використовують для створення та навчання складних нейронних мереж з подальшим їх впровадженням в різні програмні засоби.

В результаті порівняльного аналізу вищезазначених бібліотек для машинного навчання (табл. 2.2) можна зробити висновок, що під час розробки програмного забезпечення для створення та навчання згорткових нейронних мереж варто використовувати бібліотеки TensorFlow та Keras.

Таблиця 2.2 – Порівняння бібліотек для машинного навчання

Особливість	Scikit-learn	PyTorch	Tensorflow + Keras
Простота взаємодії з бібліотекою	Легко	Середньо	Легко
Є готові методи для навчання моделей	Так	Hi	Так
Можливість навчання на центральному процесорі	Так	Так	Так
Можливість навчання на графічному процесорі	Hi	Так	Так
Розгортання моделі на мобільному пристрої	Hi	Hi	Так
Підтримка глибинного навчання	Hi	Так	Так
Підтримка згорткових нейронних мереж	Hi	Так	Так
Технічна документація та навчальні посібники	Багато	Мало	Багато

2.2.3 Вибір бібліотеки для побудови графічного інтерфейсу

Для підвищення зручності використання програмного забезпечення будь-яка система має надавати можливість створення зручного графічного інтерфейсу, через який буде відбуватися взаємодія кінцевого користувача з

програмним кодом. В мові програмування Python представлено низку бібліотек, які використовуються для побудови графічного інтерфейсу користувача (рис. 2.5). На практиці найчастіше обирають Tkinter та PyQt.

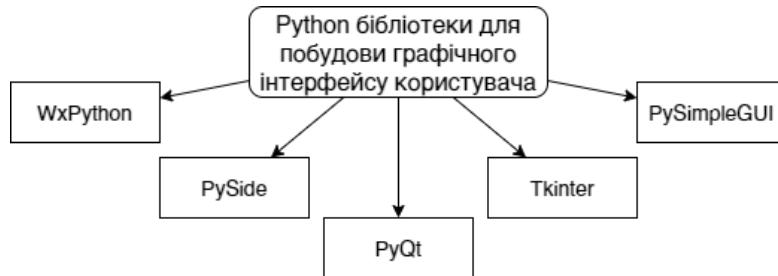


Рисунок 2.5 – Бібліотеки для побудови графічного інтерфейсу

Бібліотека Tkinter містить базовий набір структурних компонентів для побудови графічного інтерфейсу та використовується для швидкого створення простих програм. Вона вбудована в стандартний набір пакетів Python та не вимагає встановлення додаткових бібліотек для роботи. Створювані графічні елементи мають стандартний дизайн, проте його складно відредактувати та налаштувати під свої потреби.

Бібліотека PyQt є кросплатформною реалізацією бібліотеки Qt. Програмування графічного інтерфейсу за допомогою Qt базується навколо сигналів і слотів для зв’язку між об’єктами, що забезпечує гнуучкість розробки. PyQt використовують для створення інтерфейсу в складних сучасних програмах, адже він надає необхідний набір вбудованих інструментів для створення і редагування графічних компонентів.

Зважаючи на виявлені переваги PyQt та недоліки Tkinter (табл. 2.3) стає зрозумілим, що в розроблюваному програмному забезпеченні для побудови графічного інтерфейсу варто використовувати бібліотеку PyQt.

Таблиця 2.3 – Порівняння бібліотек для графічного інтерфейсу

Особливість	Tkinter	PyQt
1	2	3
Вбудована в Python	Так	Ні

Продовження таблиці 2.3

1	2	3
Вільно розповсюджувана бібліотека	Так	Так
Швидкість та легкість створення інтерфейсу	Так	Ні
Однаковий вигляд розробленого інтерфейсу на різних платформах	Ні	Так
Широкий набір інструментів для редагування стилів компонентів	Ні	Так
Кількість вбудованих віджетів та контейнерів	Мало	Багато
Віджети реалізовані як класи Python	Ні	Так
Можливість створення користувацьких компонентів, спадкування	Ні	Так
Функціональні можливості інтерфейсу розробленої програми	Менше	Більше
Технічна документація та навчальні посібники	Мало	Багато

2.2.4 Вибір бібліотеки для відображення графічних даних

Процес навчання нейронної мережі завжди супроводжується статистичною інформацією про її стан. Інформація про точність та витрати під час навчання та валідації краще засвоюється людиною у графічному вигляді. Для побудови графіків в елементах графічного інтерфейсу PyQt можна використати PyQtGraph або Matplotlib.

Бібліотека PyQtGraph поставляється разом з PyQt та дозволяє будувати графіки на стандартних елементах графічного інтерфейсу. Вона призначена для використання в наукових, інженерних та математичних, додатках і написана мовою програмування Python, завдяки чому швидко виконується. Серед недоліків PyQtGraph можна виділити такі: складність реалізації та відсутність детальної програмної документації.

Бібліотека Matplotlib стала стандартним засобом для візуалізації даних на Python. Вона використовується для побудови двовимірної та тривимірної графіки, надає набір простих та гнучких параметрів, які легко можна налаштувати відповідно до потреб програми. Для Matplotlib існує багато навчальних посібників з побудови графіків, а її функціонал значно перевищує можливості PyQtGraph.

Тому, можна зробити висновок, що для візуалізації графічних даних доцільним є використання саме бібліотеки Matplotlib.

2.3 Розробка прототипів графічного інтерфейсу

Перед розробкою графічного інтерфейсу користувача необхідно розробити його прототипи. Під час прототипування графічного інтерфейсу прийнято використовувати вайрфрейми.

Згідно з проведеним аналізом вимог, функціональну частину програми можна умовно розділити на дві частини: навчання моделі та класифікація зображення. В такому випадку для реалізації інтуїтивно зрозумілого графічного інтерфейсу користувача необхідно створити такі вікна: головне вікно програми, вікно зі списком всіх моделей, вікно навчання моделі з її основними характеристиками, вікно для класифікації зображень з датчиків та вікно з інформацією про програму (рис. 2.6).

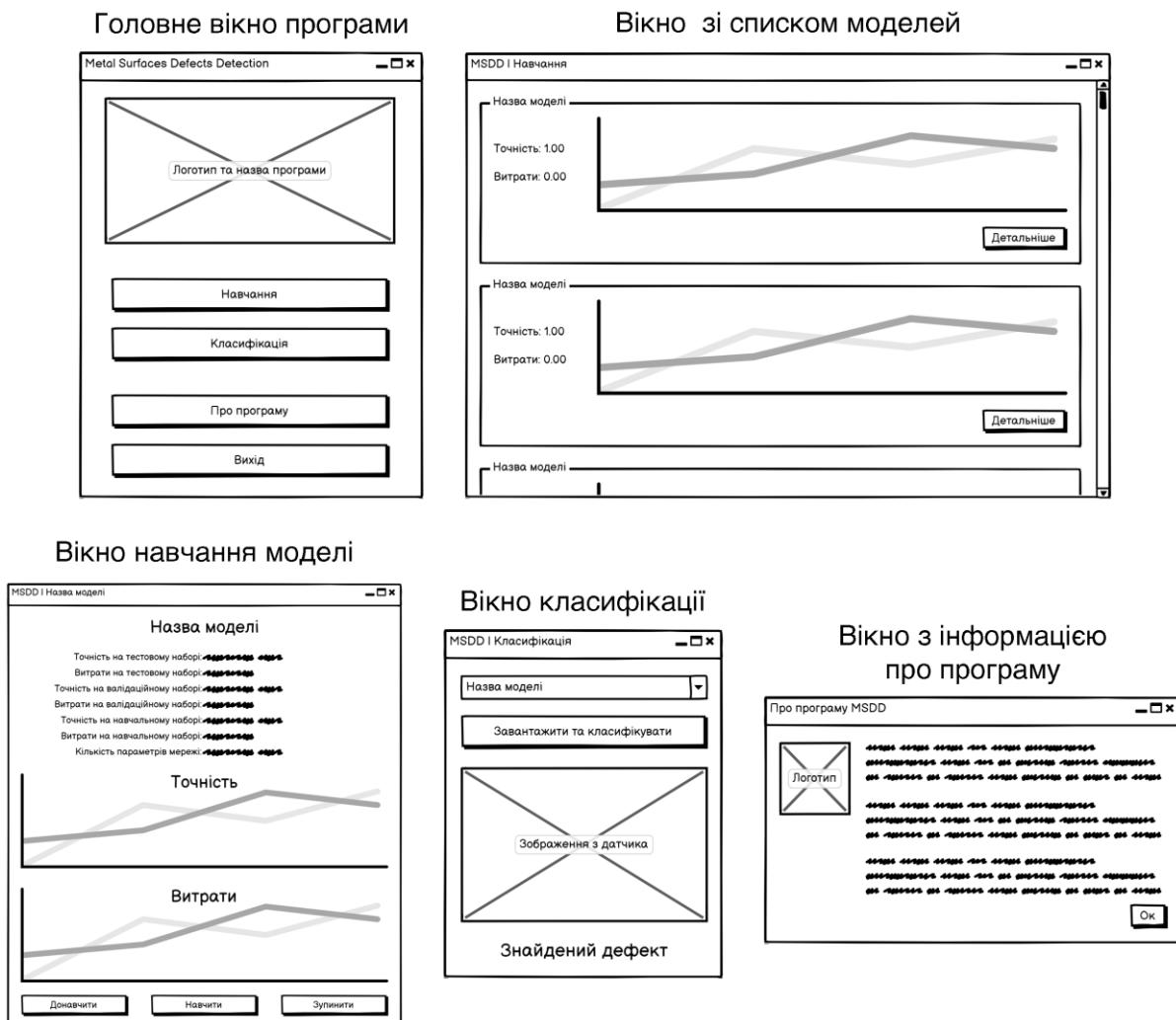


Рисунок 2.6 – Вайрфрейми вікон програми

2.4 Висновки за розділом 2

Відповідно до поставленої мети роботи було виконано аналіз вимог до розроблюваного програмного забезпечення, визначено основні програмні засоби, які варто використовувати під час розробки програми, виконано прототипування графічного інтерфейсу користувача.

Згідно з визначеними функціональними та нефункціональними вимогами програма має надавати можливості навчання моделей згорткових нейронних мереж та їх вибору для подальшого використання в задачі класифікації зображень. Програмне забезпечення повинно бути стійким до виникаючих позаштатних ситуацій, мати інтуїтивно зрозумілий графічний інтерфейс та швидкий відклик на дії користувача.

В якості інструментальних засобів обрано мову програмування Python, інтегроване середовище розробки PyCharm, бібліотеки Keras, TensorFlow, PyQt та Matplotlib. Такий стек технологій дозволить реалізувати сучасний програмний засіб, що відповідає всім поставленим вимогам і з високою ефективністю вирішує задачу виявлення дефектів на металевих поверхнях.

3 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ПРОГРАМИ

3.1 Реалізована структура програми

Розроблене програмне забезпечення складається з 1 проекта, 8 папок, 22 виконуваних файлів з 21 програмним класом. Загальну структуру проекта наведено на рисунку 3.1.

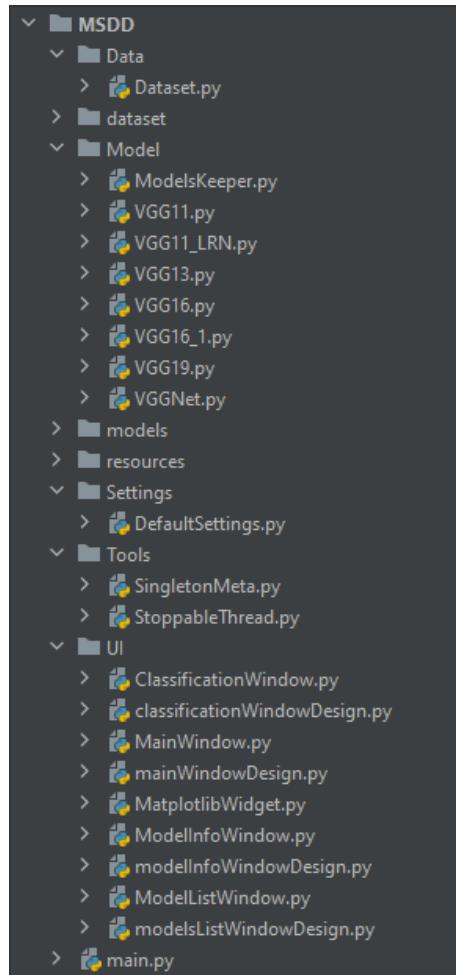


Рисунок 3.1 – Загальна структура розробленої програми.

Кореневою папкою проекта з програмним забезпечення є папка зі скороченою назвою програми «MSDD». В ній зберігаються всі програмні файли, набір даних для навчання моделей, статистична інформація та збережені ваги навчених нейромережевих моделей VGG.

В папці Data зберігаються файли, які використовуються для організації та підготовки навчального набору даних.

В папці dataset зберігається набір даних який використовується для навчання та тестування створюваних згорткових нейронних мереж.

В папці Model зберігаються файли, які відповідають за створення, компіляцію, навчання, тестування та збереження нейромережевих моделей. Вони уособлюють модель даних і не залежать від графічного інтерфейсу.

В папці models зберігаються статистичні дані про навчені моделі та файли з їх вагами. При розпізнаванні зображення, і створенні нейромережової моделі її ваги за замовчанням завантажуються з цієї папки.

В папці resources зберігаються файли з ресурсами, які необхідні для роботи програми. Серед ресурсів програми знаходиться логотип та деякі програмні зображення.

В папці Settings знаходиться файл, що реалізує основні налаштування програми. Серед основних параметрів можна виділити такі: шлях до файлів з моделями та наборів даних, деяка статична інформація.

В папці Tools зберігається реалізація допоміжних класів проекта, які допомагають спростити процес його розробки, реалізуючи можливості використання шаблону програмування Singleton та організації багатонитевих процесів.

В папці UI розташовано файли які відповідають за графічний інтерфейс користувача, організацію передачі даних від моделей до інтерфейсу та додавання віджетів з графіками Matplotlib до компонентів програми.

Особливу увагу варто приділити файлу «main.py». З нього починається виконання програми та запуск головного вікна графічного інтерфейсу користувача.

3.2 Реалізовані класи програми

Кожен розроблений програмний клас розміщено в окремому файлі з розширенням «.py». Назва практично кожного класу відповідає назві файлу, в

якому його записано. Вийнятком є класи, що реалізують графічний інтерфейсу користувача. Їх назва завжди однаєва – це «Ui_mainWindow», а файли в яких вони зберігаються в кінці назви мають допис «Design.py».

Клас Dataset відповідає за підготовку наборів даних для навчання приводячи розмір використовуваних зображень до специфікації мережі, нормалізуючи значення пікселів на зображені та групуючи їх за класовою ознакою. Серед методів класу можна виділити такі:

- get_train_data – отримати набір даних для тренування;
- get_valid_data – отримати набір даних для валідації;
- get_test_data – отримати набір даних для тестування;
- get_classes – отримати всі класи які представлено в наборі даних.

Клас VGG11 відповідає за створення та налаштування архітектури моделі VGG11.

Клас VGG11_LRN відповідає за створення та налаштування архітектури моделі VGG11 LRN.

Клас VGG13 відповідає за створення та налаштування архітектури моделі VGG13.

Клас VGG16 відповідає за створення та налаштування архітектури моделі VGG16.

Клас VGG16_1 відповідає за створення та налаштування архітектури моделі VGG16 з доданими ядрами згортки розміром 1×1 .

Клас VGG19 відповідає за створення та налаштування архітектури моделі VGG19.

Всі класи VGG містять один основний статичний метод – create_empty_model. Він дозволяє створити та повернути архітектуру выбраної VGG мережі.

Клас VGGNet уособлює згорткову нейронну модель взагалом. Він відповідає за завантаження однієї з VGG архітектур з вище зазначених класів, її компіляцію, навчання тестування та використання в програмі. Серед основних методів класу можна виділити такі:

- `__init__` – конструктор класу;
- `set_new_model_and_compile` – встановлення нової архітектурної моделі та її подальша компіляція;
- `fit_model` – навчання скомпільованої нейромережової моделі;
- `load_model` – завантаження моделі з пам'яті пристрою;
- `save_model` – збереження моделі в пам'ять пристрою;
- `predict` – передбачення наявності та типу дефекта.

Клас `ModelKeeper` є контейнером для використовуваних нейронних моделей та відповідає за можливість доступу до них з будь-якої частини програми. Клас реалізовано за допомогою шаблона проєктування `Singleton`. Основні методи `ModelKeeper`:

- `get_models` – отримати словник зі всіма моделями;
- `get_model_by_name` – отримати певну модель за назвою.

Клас `DefaultSettings` відповідає за встановлення та зберігання налаштувань програми.

Клас `SingletonMeta` реалізує шаблон програмування `Singleton` і використовується як базовий клас для `ModelKeeper`.

Клас `StoppableThread` використовується для реалізації багатонитевості в програмі та можливості зупиняти поток виконання, чого за замовчанням не було передбачено мовою програмування Python.

Клас `MatplotlibWidget` реалізує графічний віджет для PyQt графічного інтерфейсу, який може містити графіки побудовані за допомогою `Matplotlib`.

Клас `mainWindowDesign.Ui_mainWindow` реалізовує інтерфейс головного вікна програми.

Клас `MainWindow` реалізує взаємодію програмних функцій з головним вікном програми та прив'язку натискань на кнопки до методів класу.

Клас `modelListWindowDesign.Ui_mainWindow` реалізовує інтерфейс вікна зі списком моделей.

Клас `ModelListWindow` реалізовує взаємодію програмних функцій об'єкта класу `ModelKeeper` та вікна зі списком моделей.

Клас `modelInfoWindowDesign.Ui_mainWindow` реалізовує інтерфейс вікна навчання моделі.

Клас `ModelInfoWindow` реалізує взаємодію програмних функцій з вікном навчання моделі.

Клас `classificationWindowDesign.Ui_mainWindow` реалізовує інтерфейс вікна класифікації даних.

Клас `ClassificationWindow` реалізовує взаємодію програмних функцій об'єкта класу `VGGNet` та вікна класифікації.

Взаємодію розроблених класів між собою відображене в діаграмі класів програми (рис. 3.2).

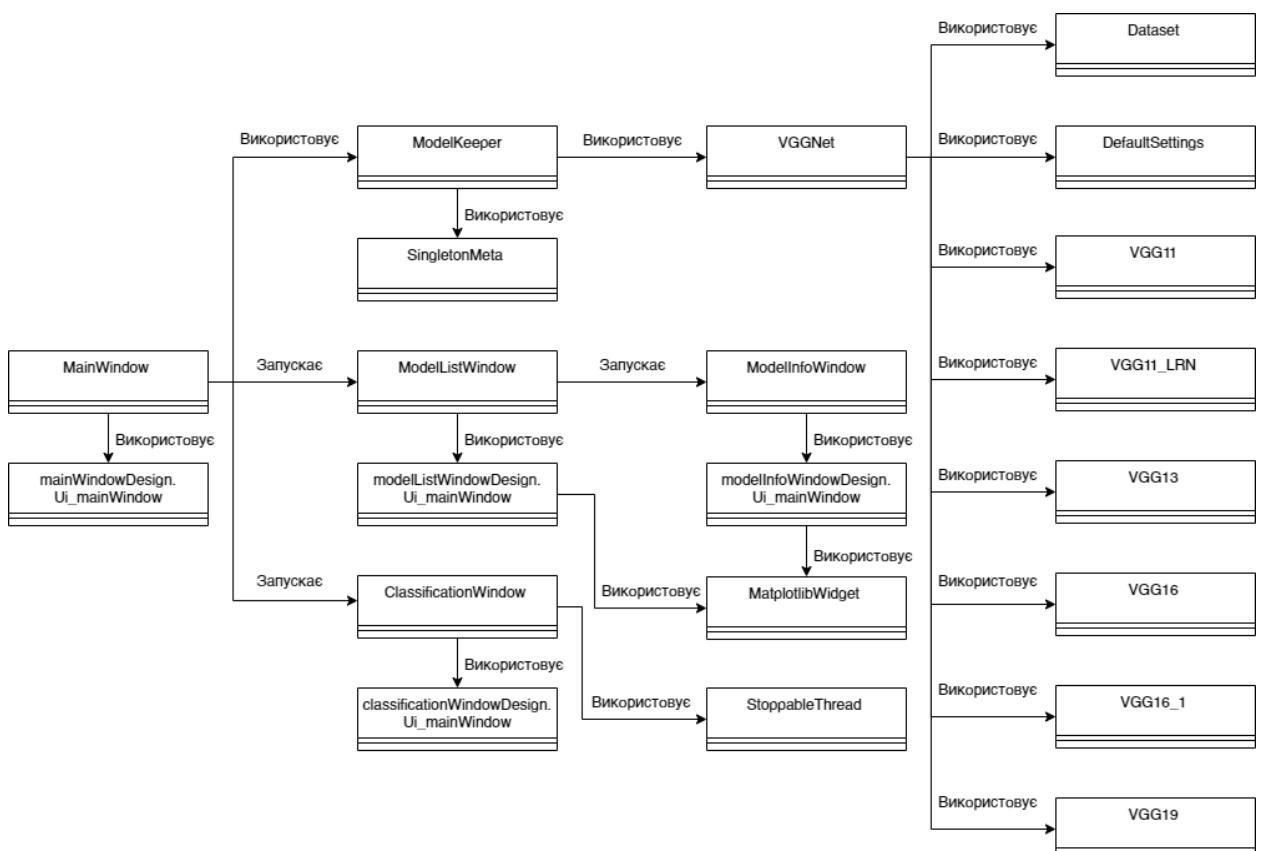


Рисунок 3.2 – Діаграма класів програми

3.3 Реалізація нейромережевих моделей VGG

Основною функціональною частиною програми є шість згорткових нейронних мереж сімейства `VGGNet`. Їх архітектура реалізована за

допомогою інструментів для побудови нейронних мереж взятих з API Keras та імплементованих в бібліотеку Tensorflow. Структура шарів кожної спроєктованої VGG мережі відповідає вимогам визначеним в специфікації сімейства VGGNet [17], а їх короткий опис та основні використані параметри наведено в таблиці 3.1.

Таблиця 3.1 – Використані шари для побудови архітектур VGGNet

Назва шару	Опис	Основні параметри
tf.keras.layers.Conv2D	Створює ядро згортки, яке спільно з входом шару створює тензор вихідних даних	Кількість фільтрів на виході Розмір ядра згортки Тип вирівнювання Тип функції активації Форма вхідного тензора
tf.keras.layers.Lambda	Надає можливість використання довільних виразів в побудованій послідовній моделі	Залежить від довільного виразу, який використовується
tf.keras.layers.MaxPool2D	Агрегувальний шар зменшує вибірку вводу за його просторовими розмірами, приймаючи максимальні значення	Розмір агрегувального фільтра Крок Тип вирівнювання
tf.keras.layers.Flatten	Згладжує вхідний шар у одновимірний тензор	-
tf.keras.layers.Dense	Додавання повнозв'язаного шару	Кількість нейронів в шарі Тип функції активації

В якості структурної організації моделей використано послідовну модель tensorflow.keras.Sequential. Вона підходить для простого стеку шарів, де кожен шар має рівно один вхідний і один вихідний тензор. Така структура повністю відповідає будові всіх нейромережевих моделей VGG.

Під час дослідження архітектури моделей VGGNet сімейства було визначено що всі вони схожі за своєю структурою, тому в якості демонстрації доцільно навести спроєктовану архітектуру однієї з моделей, наприклад VGG19 (рис. 3.3).

Навчання згорткових нейронних мереж VGGNet сімейства виконується протягом 20 епох на тренувальному наборі даних. Після завершення кожної епохи якість моделі перевіряється на наборі даних для

валідації. Для забезпечення стабільності роботи програми та протидії перенавчанню мережі використано такі технології:

- `tf.keras.callbacks.EarlyStopping` – для зупинки навчання моделі, якщо відстежувана метрика перестала вдосконалюватися або змінюватися;
- `tf.keras.callbacks.ModelCheckpoint` – для збереження найкращої моделі за час навчання або моделі після закінчення кожної епохи;
- `tf.keras.callbacks.Callback` – для відстежування параметрів моделі під час навчання та відповідної зміни інформації в графічному інтерфейсі.

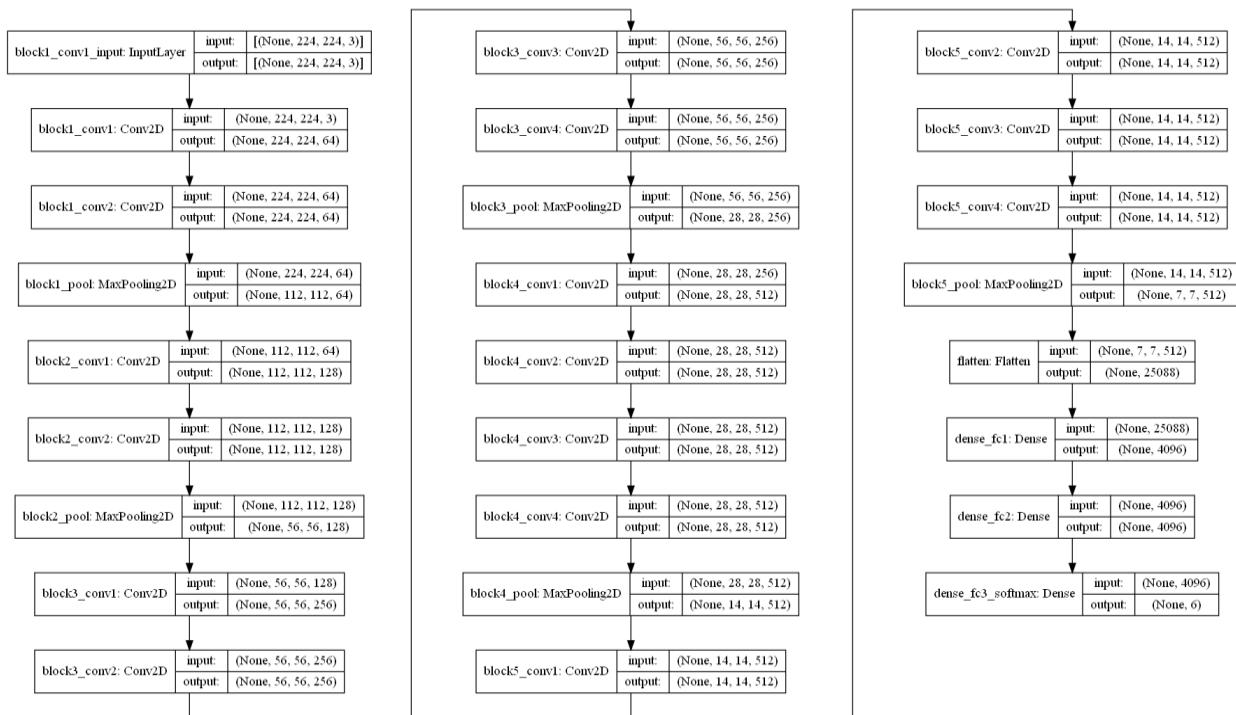


Рисунок 3.3 – Спроектована архітектура мережі VGG19

Під час навчання інформація про модель та її ваги зберігається в папці з назвою моделі в каталозі `models`. Для збереження інформації про статистику навчання використовується JSON файл. Ваги мережі зберігаються в форматі HDF5 представлена бібліотекою Keras. Так як процес збереження вагів моделі займає багато часу, то для стабілізації роботи програми спочатку дані зберігаються в тимчасовому файлі а вже потім замінюють файл з попередньою версією вагів.

3.4 Набір навчальних даних та їх структура

Для навчання VGG моделей використано набір даних з БД NEU Metal Surface Defects Databse який містить шість типів типових поверхневих дефектів металевої поверхні сталевої гарячекатаної смуги які доволі часто виникають під час технологічного процесу [32]. Набір включає в себе 1800 RGB зображень з такими дефектами: мікротріщини, включення, забруднення, рябизна, вкатана окалина та подряпини.

Для підготовки наборів даних для навчання використовується клас Dataset. Його робота заснована на використанні можливостей бібліотеки Tensorflow, а саме класу tf.keras.preprocessing.image.ImageDataGenerator. Він дозволяє генерувати партії даних тензорних зображень із збільшенням набору даних у режимі реального часу. Кількість та назви класів розраховуються автоматично. Для цього необхідно, щоб структура набору даних (рис. 3.4) відповідала таким правилам:

- зображення кожного з типів дефектів збережено в папці з назвою відповідного дефекту;
- всі папки знаходяться в одному каталозі, шлях до якого і подається в якості одного з параметрів.

Для генерації набору даних і подальшої його передачі в модель використовується метод flow_from_directory.

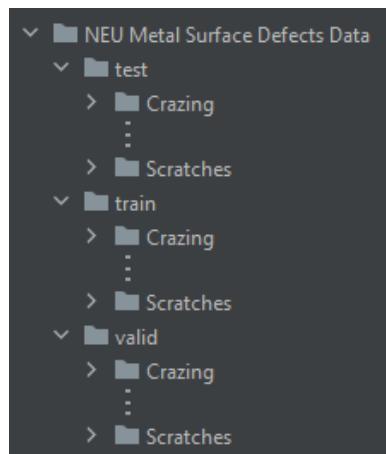


Рисунок 3.4 – Структура відредагованого набору даних

3.5 Реалізація графічного інтерфейсу користувача

Візуально зрозумілий та привабливий інтерфейс сприяє більш приємній та ефективній роботі з програмою. Він має бути реалізований інтуїтивно зрозумілим для користувача – непрофесіонала в комп’ютерній галузі.

При розробці графічного інтерфейсу за мету було визначено створення функціонального, проте простого, неперенавантаженого зайвою інформацією, інтерфейсу. Інтерфейс, створений за такими принципами сприятиме зручному використанню програми, та підвищить конкурентоздатність системи.

В процесі розробки графічного користувацького інтерфейсу перевага була віддана спокійним, однотонним відтінкам, які привертають увагу користувача. Для полегшення процесу візуального сприйняття, основні елементи управління мають закруглені кути, так як вони не просто приємні для погляду, а й полегшують сприйняття графіки та обробку інформації людиною.

На рисунку 3.5 наведено зображення головного вікна розробленого програмного забезпечення.

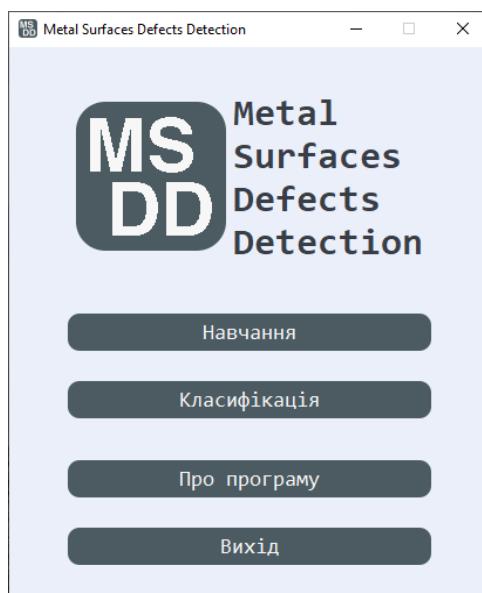


Рисунок 3.5 – Інтерфейс головного вікна програми

Зображення інтерфейсу вікна зі списком всіх моделей, які використовуються в програмі наведено на рисунку 3.6.



Рисунок 3.6 – Інтерфейс вікна зі списком всіх моделей

Одне з вікон навчання нейромережової моделі VGG з переліком її основних характеристик наведено на рисунку 3.7.

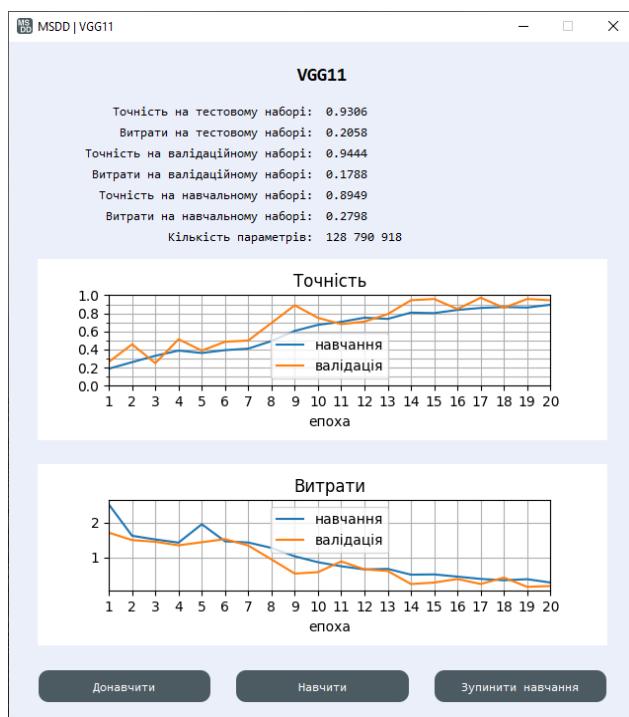


Рисунок 3.7 – Інтерфейс вікна навчання моделі

Зображення інтерфейсу вікна розпізнавання та попереднього завантаження зображення наведено на рисунку 3.8.



Рисунок 3.8 – Інтерфейс вікна розпізнавання зображень

На рисунку 3.9 наведено зображення вікна з основною інформацією про програму.

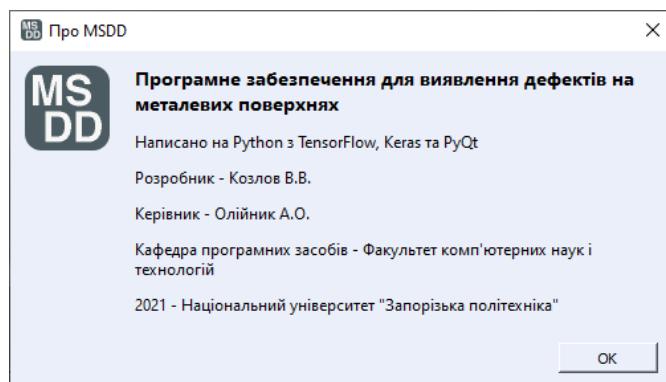


Рисунок 3.9 – Вікно з інформацією про програму

3.6 Інші особливості програмної реалізації

Процес завантаження, навчання моделі та класифікації за її допомогою є ресурсоємним. Для того, щоб графічний інтерфейс не зависав

було прийнято рішення виконувати вищезазначені процеси в окремих потоках за допомогою стандартних засобів організації багатонитевості мовою Python.

Для того, щоб навчити модель розпізнавати зображення на інших тренувальних даних, необхідно попередньо підготовлений за правилами зазначеними в пункті 3.4 набір даних розмістити в папці Data папки dataset та перезапустити програму.

Якщо відбувається навчання моделі і користувач хоче зупинити його або закрити вікно, то стандартні засоби TensorFlow не надають бажаної швидкодії, адже для того щоб зупинити навчання необхідно дочекатися закінчення епохи, що інколи може тривати до 40 хвилин. Для вирішення цієї проблеми реалізовано клас StoppableThread, який дозволяє дуже швидко зупинити виконання потоку навчання, при цьому зберегти в пам'яті пристрою ваги нейронної мережі з минулої епохи.

Програма розроблена таким чином, щоб її можна було легко масштабувати за допомогою додавання інших глибинних нейронних мереж, шляхом додавання оголошення таких моделей в класі ModelsKeeper.

3.7 Висновки за розділом 3

Отже, за результатами проведеної роботи було обрано набір навчальних даних та оглянуто основні правила його будови. Реалізовано основну структуру програмного забезпечення та розроблено необхідні програмні класи.

Спроектовано та впроваджено всі 6 нейромережевих моделей сімейства VGGNet за допомогою інструментів для побудови нейронних мереж взятих з API Keras та імплементованих в бібліотеку Tensorflow.

Відповідно до визначених функціональних та нефункціональних вимог спроектовано та розроблено графічний інтерфейс користувача в мінімалістичному та лаконічному дизайні.

4 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Тестування програмного забезпечення

Для перевірки дієздатності розробленої програмної системи та її відповідності до висунутих функціональних вимог необхідно виконати функціональне тестування.

Функціональне тестування – це процес тестування програмного забезпечення який не залежить від вихідного коду і ґрунтуються на принципі дослідження «чорної скриньки». Метою тестування є перевірка функціональних можливостей програми шляхом надання необхідних вхідних даних та подальшої перевірки отриманого результату.

Враховуючи визначені вимоги до розроблюваного програмного забезпечення було створено набір тест-кейсів та проведено тестування основних можливостей програми за ними (табл. 4.1 – 4.4).

Таблиця 4.1 – Тест-кейс перегляду списку моделей

Короткий опис	Перевірка можливості перегляду списку моделей			
Передумови	Відкрито головне вікно програми			
Кроки	№	Покрокові дії	Очікуваний результат	Статус
	1	Натиснути на кнопку навчання	Відкрито вікно зі списком всіх моделей.	Пройдено
	2	Перемістити положення повзунка	Відбувається гортання списку	Пройдено
	3	Закрити вікно зі списком моделей	Головне вікно програми стало активним	Пройдено

Таблиця 4.2 – Тест-кейс перегляду інформації про модель

Короткий опис	Перевірка можливості перегляду інформації про модель			
Передумови	Відкрито головне вікно програми			
Кроки	№	Покрокові дії	Очікуваний результат	Статус
	1	Натиснути на кнопку навчання	Відкрито вікно зі списком всіх моделей.	Пройдено
	2	Вибрати зі списку необхідну модель та натиснути на кнопку детальніше	Відкрито вікно з інформацією про вибрано модель	Пройдено
	3	Закрити вікно з інформацією про модель	Вікно зі списком моделей стало активним	Пройдено

Таблиця 4.3 – Тест-кейс процес навчання моделі

Короткий опис	Перевірка процесу навчання моделі		
Передумови	Відкрито головне вікно програми		
Кроки	№	Покрокові дії	Очікуваний результат
	1	Натиснути на кнопку навчання	Відкрито вікно зі списком всіх моделей
	2	Вибрати зі списку необхідну модель та натиснути на кнопку детальніше	Відкрито вікно з інформацією про вибрано модель
	3	Натиснути на кнопку навчити	Якщо модель не навчена почався процес навчання, про що свідчить повідомлення в назві моделі. Якщо модель попередньо навчена відображене діалогове вікно. Попередній прогрес навчання моделі скинуто
	4	Натиснути на кнопку зупинити навчання	Навчання моделі зупинено
	5	Натиснути кнопку донавчити	Почався процес донавчання, про що свідчить повідомлення в назві моделі
	6	Закрити вікно	Процес донавчання зупинено. Збережено прогрес останньої пройденої епохи. Вікно зі списком моделей стало активним

Таблиця 4.4 – Тест-кейс розпізнавання зображень

Короткий опис	Перевірка можливості розпізнавання зображень		
Передумови	Відкрито головне вікно програми		
Кроки	№	Покрокові дії	Очікуваний результат
	1	Натиснути на кнопку класифікація	Відкрито вікно класифікації зображень
	2	Натиснути на комбобокс з моделями	Відкрито список моделей, які можна використати для розпізнавання зображень
	3	Вибрати необхідну модель	Список закрито. В комбобокс зазначено назву вибраної моделі
	4	Натиснути на кнопку завантажити та класифікувати зображення	Відкрито вікно вибору файлу з пам'яті пристрою
	5	Вибрати зображення з пам'яті пристрою	Вибране зображення відображено у вікні програми. Відбувається класифікація зображення. Результат розпізнавання виведено у вікні класифікації
	6	Закрити вікно	Головне вікно програми активне

Перелік інших перевірок виконаних під час тестування програмного забезпечення було оформлено у вигляді контрольного списку (табл. 4.5).

Таблиця 4.5 – Контрольний список тестування

Вимога	Статус
Відкриття вікон програми	
Відкриття головного вікна програми	Пройдено
Відкриття вікна зі списком моделей	Пройдено
Відкриття вікна з інформацією про модель	Пройдено
Відкриття вікна класифікації	Пройдено
Відкриття вікна з інформацією про програму	Пройдено
Можливості головного вікна програми	
Реакція графічних елементів на дії користувача	Пройдено
Відображення назви програми	Пройдено
Відображення логотипа програми	Пройдено
Фіксований розмір вікна	Пройдено
Можливості вікна зі списком моделей	
Можливість гортання списку моделей	Пройдено
Коректне відображення назв неромережевих моделей	Пройдено
Відображення графіка точності для кожної з моделей	Пройдено
Реакція графічних елементів на дії користувача	Пройдено
Розмір вікна можна змінювати	Пройдено
Можливості вікна з інформацією про модель	
Реакція графічних елементів на дії користувача	Пройдено
Можливість навчання моделі	Пройдено
Можливість зупинити навчання моделі	Пройдено
Коректне відображення назви неромережової моделі	Пройдено
Відображення графіка точності та витрат	Пройдено
Можливості вікна класифікації	
Вибір та завантаження зображення	Пройдено
Коректне відображення зображення	Пройдено
Можливість вибору моделі зі списку	Пройдено
Інше	
Найвні всі моделі сімейства VGGNet	Пройдено
Читабельний розмір шрифтів графічних елементів	Пройдено
Основні елементи керування підписано українською мовою	Пройдено
Висока швидкість відклику на дії користувача	Пройдено
Наявність повідомлення користувачу під час виходу з програми	Пройдено

За отриманими результатами проведеного функціонального тестування можна стверджувати, що розроблене програмне забезпечення повністю відповідає визначеним функціональним вимогам.

4.2 Експериментальне дослідження програми

Навчанняожної нейромережевої моделі сімейства VGGNet виконувалося впродовж 20 епох за допомогою можливостей програмних бібліотек Keras та TensorFlow. В результаті проведеного навчання було збережено ваги всіх отриманих мереж, історію зміни точності та витрат. Навчені моделі доступні для донавчання та класифікації зображень, а графіки зміни точності наведено на рисунку 4.1.

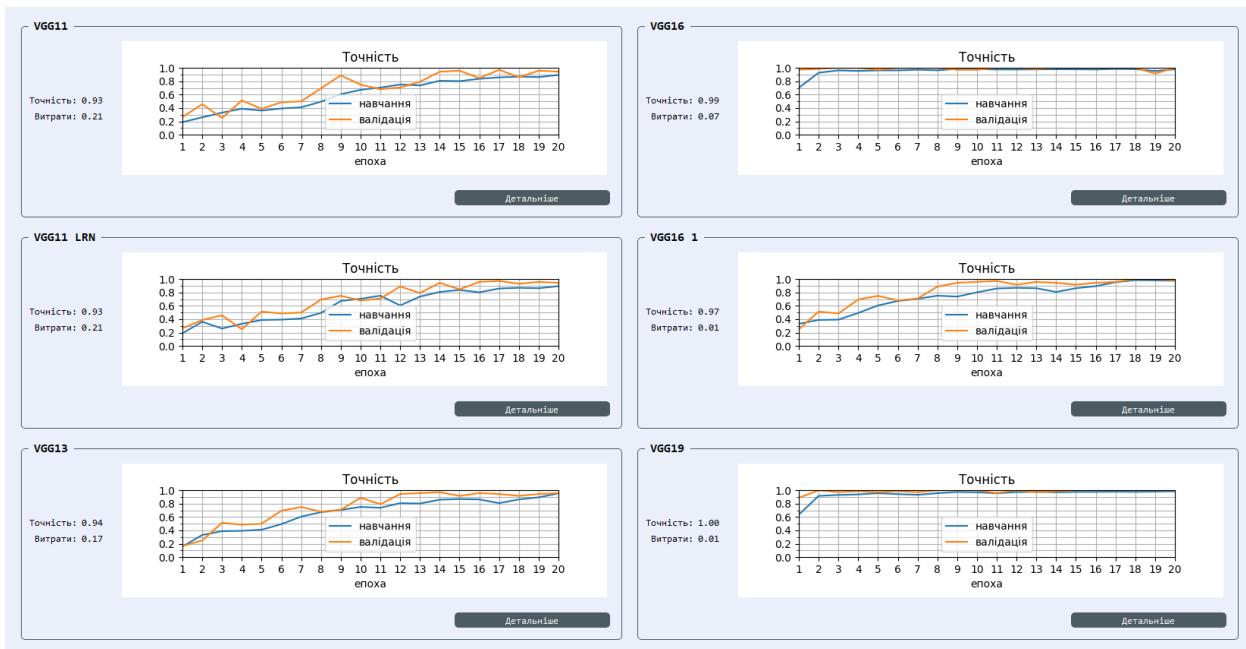


Рисунок 4.1 – Графіки зміни точності всіх моделей VGG під час навчання

Всі нейромережеві моделі VGG досягли точності розпізнавання зображення більше ніж 89%, що є достатньо високим результатом.

Значно виділяються стрімким зростанням точності графіки моделей VGG16 та VGG19. Це обумовлено тим, що для їх навчання було використано попередньо підготовлені ваги ядер згортки, які навчалися протягом 3 тижнів на наборі даних ImageNet та можуть добре узагальнювати зображення і находити необхідні ознаки на ньому.

Детальне порівняння результатів навчання та роботи реалізованих моделей сімейства VGGNet наведено в таблиці 4.6.

Таблиця 4.6 – Результати навчання та використання моделей

Параметр	VGG11	VGG11 LRN	VGG13	VGG16	VGG16 1	VGG19
Точність на навчальному наборі даних, %	89,49	89,49	95,41	97,95	97,64	98,55
Точність на валідаційному наборі даних, %	94,44	94,44	95,83	100	98,61	100
Точність на тестовому наборі даних, %	93,06	93,06	94,44	98,61	97,22	100
Витрати на навчальному наборі даних	0.2798	0.2798	0.1732	0.0598	0.0571	0.0500
Витрати на валідаційному наборі даних	0.1788	0.1788	0.1562	0.0231	0.0042	0.0027
Витрати на тестовому наборі даних	0.2058	0.2058	0.1656	0.0719	0.0120	0.0141
Кількість згорткових шарів	8	8	10	13	13	16
Загальна кількість шарів	11	11	13	16	16	19
Кількість параметрів	128790918	128790918	128975430	134285126	129566534	139594822
Розмір файлу з вагами, КБ	503133	503134	503858	524610	506178	545359
Розмір файлу з моделлю загалом, КБ	1509367	1509382	1511553	1458790	1518504	1479542
Загальний час навчання, год	4 год 19 хв	4 год 41 хв	6 год 45 хв	12 год 17 хв	9 год 20 хв	13 год 53 хв
Можливість використання вагів ядер згортки мережі натренованої на БД ImageNet	Hi	Hi	Hi	Так	Hi	Так
Загальний час навчання з використанням натренованих вагів ядер згортки, год	-	-	-	3 год 10 хв	-	3 год 50 хв
Кількість епох	20	20	20	20	20	20
Епоха на якій перевищено 75% точності на навчальному наборі	12	11	10	2	8	2

Згідно з отриманими результатами можна стверджувати, що можлива досягнута точність розпізнавання зображень залежить від кількості згорткових шарів в моделях VGG. Так найменша модель VGG11 з 8 згортковими шарами під час навчання впродовж 20 епох змогла досягти точності на навчальному наборі даних в 89,49%, на тестовому в 93,06%, а найбільша модель VGG19 – на навчальному наборі даних в 98,55%, на тестовому в 100%.

Використання локальної нормалізації відповідей після першого згорткового шару в нейронній мережі VGG11 LRN значно не вплинуло на якість отриманих результатів, адже значення точності після 20 епох не

відрізняється від точності моделі VGG11. Проте час навчання VGG11 LRN склав 4 год 41 хв, що на 22 хв довше часу навчання VGG11.

Використання однічних ядер згортки в нейромережевій моделі VGG16 1 зменшило розмір файлу з вагами мережі на 19 МБ, проте й вплинуло на зменшення точності на навчальному наборі даних на 0,31%, на тестовому – на 1,39% у порівнянні з VGG16.

Більша кількість згорткових шарів сприяє зросту кількості параметрів для навчання, що впливає на розмір файлу з вагами мережі в пам'яті пристрою. Так модель VGG11 з 8 згортковими шарами може бути записана в файл розміром 491 МБ, VGG16 з 13 згортковими шарами – в файл розміром 512 МБ, а VGG19 з 16 згортковими шарами – в файл розміром 533 МБ.

Різниця в розмірі файлів з вагами моделей не перевищує 50 МБ і є достатньо малою для сучасних обчислювальних пристройів. Проте при використанні більшої кількості параметрів збільшується й час, необхідний для навчання й оптимізації мережі. При навчанні впродовж 20 епох на наборі даних, який містить 1800 зображень час навчання мережі VGG11 склав 4 год 19 хв, мережі VGG13 – 6 год 45 хв, VGG16 – 12 год 17 хв, VGG19 – 13 год 53 хв. За допомогою використання попередньо підготовлених вагів ядер згортки для мережі VGG16 вдалося зменшити час навчання до 3 год 10 хв, а для мережі VGG19 – до 3 год 50 хв.

При вирішенні задачі виявлення великої кількості різних типів дефектів на сучасних підприємствах використовують для навчання мільйони зображень. При навчанні моделей на такому наборі даних, час навчання мережі значно більше й може сягати декілька тижнів або місяців.

При виборі використовуваної моделі завжди варто зважувати її переваги та недоліки. Якщо необхідно швидко впровадити таку систему спочатку варто навчити найпростішу модель визначати типові види дефектів, а вже потім вдосконалювати систему впроваджуючи більш складні моделі.

Порівняння основних факторів, які впливають на вибір моделі наведено на рисунку 4.2.

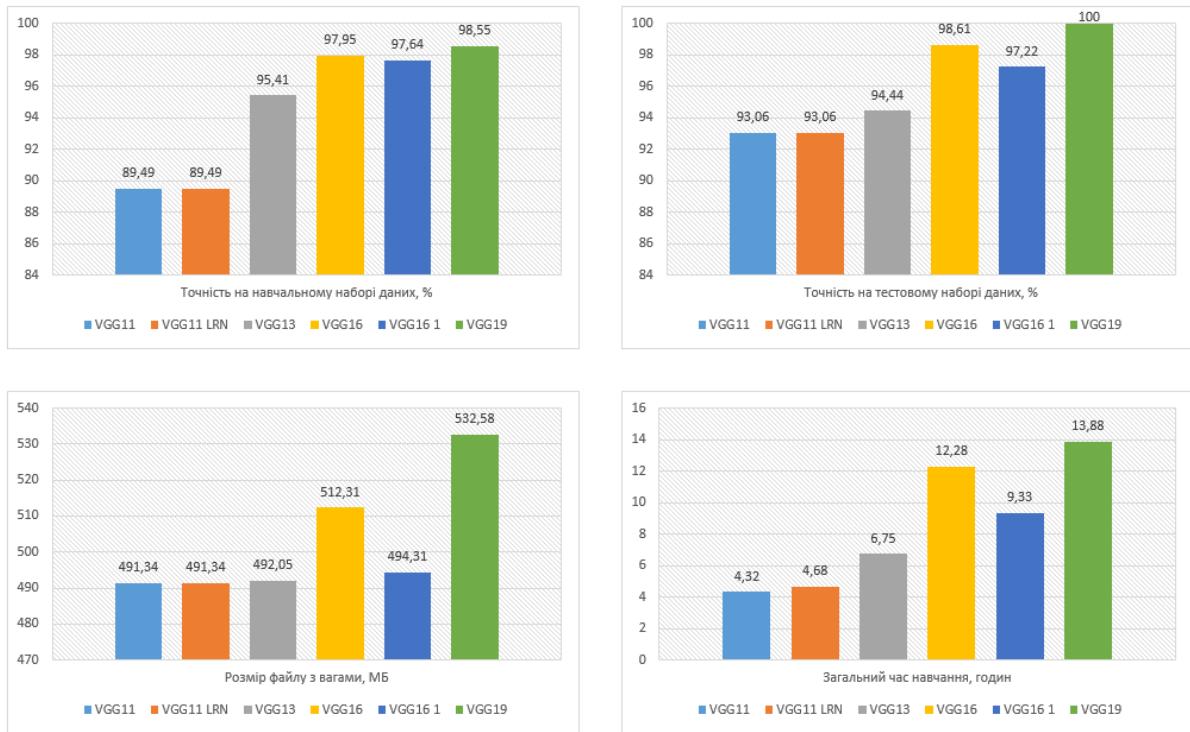


Рисунок 4.2 –Порівняння точності, розміру файлів та часу навчання

4.3 Висновки за розділом 4

За результатами проведеної роботи успішно виконано функціональне тестування розробленого програмного забезпечення та проведено докладне експериментальне дослідження.

Було виконано тестування на наборі тест-кейсів та контрольному списку. Згідно з проведеним тестуванням програмне забезпечення повністю відповідає всім висунитим до нього функціональним вимогам.

Згідно з проведеним експериментальним дослідженням, можна стверджувати, що нейромережеві моделі сімейства VGGNet можуть бути використання для вирішення задачі виявлення дефектів на металевих поверхнях. Це підтверджено на теорії та підкріплено отриманими практичними результатами.

Визначено, що особливу увагу при виборі моделей необхідно приділяти показникам точності, часу навчання та розміру файлу з налаштуваннями моделі в пам'яті використовуваного пристрою.

5 КЕРІВНИЦТВО ПРОГРАМІСТА

5.1 Призначення та умови застосування програми

Програмне забезпечення призначено для вирішення задачі виявлення дефектів на металевих поверхнях за допомогою нейромережевих моделей сімейства VGGNet.

Воно надає можливість користувачу навчати нейромережеві моделі VGG на визначеному наборі навчальних даних, обирати необхідну модель та за її допомогою виконувати розпізнавання зображення металевої поверхні з подальшою класифікацією виявлених дефектів.

Програмне забезпечення розроблено мовою програмування Python з використанням бібліотек Keras, Tensorflow, PyQt та Matplotlib. В якості інтегрованого середовища розробки використовується PyCharm.

Як апаратно-технічні засоби для експлуатації програмного забезпечення повинні використовуватися IBM-сумісний комп’ютер з характеристиками не нижче:

- процесор Intel Core i3-5xxx;
- ОЗП 4 ГБ;
- відеокарта з об’ємом пам’яті 2 ГБ;
- жорсткий диск об’ємом 64 ГБ;
- маніпулятор типу «миша» і / або клавіатура.

Як програмні засоби для належного функціонування програмного забезпечення повинні використовуватися:

- ОС Windows 10 та новіше;
- Python версії 3.8.8;
- система керування пакетами pip версії 21.0.1 та новіше;
- бібліотека Tensorflow версії 2.4.1;
- бібліотека Keras-Preprocessing версії 1.1.2;
- бібліотека Matplotlib версії 3.4.1;
- бібліотека PyQt версії 5.15.2.

5.2 Вимоги до програмного забезпечення

Вимоги до функціональних характеристик програми :

- можливість вибору необхідної нейромережевої моделі з переліку;
- навчання мережі на заданому навчальному наборі даних;
- перегляд основної інформації про мережу та графіку її навчання;
- можливість вибору нейромережевої моделі для класифікації;
- класифікація завантаженого зображення.

Нефункціональні вимоги до програми:

- інтуїтивно зрозумілий графічний інтерфейс;
- швидкість реакції на дії користувача;
- стійкість до позаштатних ситуацій.

5.3 Характеристики програми

Обсяг пам'яті, який займає програма – 3,5 ГБ. Обсяг пам'яті залежить від кількості навчених моделей та розміру даних для навчання.

Програма має міститися в пам'яті комп'ютера та запускатися у разі необхідності. Може стабільно працювати в фоновому режимі.

Структуру папок програмного забезпечення наведено на рисунку 5.1.

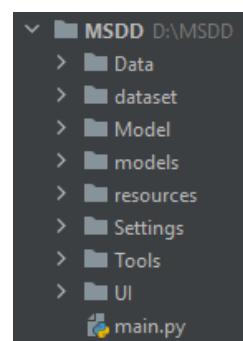


Рисунок 5.1 – Структура програмного забезпечення

Програмне забезпечення складається з таких файлів:

- Dataset.py – містить клас Dataset;

- ModelsKeeper.py – містить клас ModelsKeeper;
- VGG11.py – містить клас VGG11;
- VGG11_LRN.py – містить клас VGG11_LRN;
- VGG13.py – містить клас VGG13;
- VGG16.py – містить клас VGG16;
- VGG16_1.py – містить клас VGG16_1;
- VGG19.py – містить клас VGG19;
- VGGNet.py – містить клас VGGNet;
- DefaultSettings.py – містить клас DefaultSettings;
- SingletonMeta.py – містить клас SingletonMeta;
- StoppableThread.py – містить клас StoppableThread;
- ClassificationWindow.py – містить клас ClassificationWindow;
- classificationWindowDesign.py – дизайн вікна класифікації;
- MainWindow.py – містить клас MainWindow;
- mainWindowDesign.py – дизайн головного вікна програми;
- MatplotlibWidget.py – містить клас MatplotlibWidget;
- ModelInfoWindow.py – містить клас ModelInfoWindow;
- modelInfoWindowDesign.py – дизайн вікна навчання моделі;
- ModelListWindow.py – містить клас ModelListWindow;
- modelsListWindowDesign.py – дизайн вікна зі списком моделей;
- main.py – файл з функцією main;
- icon.png – файл зі значком програми;
- placeholder.png – файл з пустим зображенням для вікна класифікації;
- файли з навчальними даними;
- файли з вагами моделей та даними про їх навчання.

5.4 Звернення до програми

Звертання до програми виконується двома способами:

- за допомогою файлу проекта з назвою main.py;

– за допомогою командного рядка (без аргументів командного рядка).

Після виклику програми відкривається графічний інтерфейс користувача. Далі необхідно слідувати наданим програмою інструкціям.

5.5 Вхідні та вихідні дані програми

Вхідними даними для роботи програми є набір даних для навчання який представлено набором папок із зображеннями, файли зі статистичною інформацією про процес навчання моделі та файли з вагами навчених нейронних мереж. Варто зауважити, що результати взаємодії користувача з графічним інтерфейсом (натискання кнопок, гортання списку, вибір об'єктів з комбобокс) також вважаються одним з видів вхідних даних.

Вихідними даними програмного забезпечення є інформація про знайдені типи дефектів на зображенні, збережені та модифіковані файли з даними про навчені моделі та їх вагами, які зберігаються в пам'яті пристрою.

5.6 Повідомлення програмісту

Під час роботи програми в консолі виводяться повідомлення програмних бібліотек, які дублюють інформацію, що відображається в графічному інтерфейсі користувача.

Бібліотеки Keras.Preprocessing та TensorFlow виводять повідомлення з додатковою інформацією про кількість екземплярів в навчальному наборі даних, опис архітектури запущеної нейромережевої моделі та інформацію про процес навчання та розпізнавання зображень.

6 КЕРІВНИЦТВО ОПЕРАТОРА

6.1 Призначення програми

Програмний продукт призначено для вирішення задачі виявлення дефектів на металевих поверхнях за допомогою нейромережевих моделей сімейства VGGNet. Програма надає користувачу можливість:

- виконувати навчання нейромережової моделі VGG на визначеному наборі навчальних даних;
- обирати необхідну модель зі списку;
- переглядати основні характеристики, графіки точності та витрат мережі під час навчання;
- виконувати розпізнавання зображення металевої поверхні з подальшою класифікацією виявлених дефектів.

6.2 Умови виконання програми

Як апаратно-технічні засоби для експлуатації програмного забезпечення повинні використовуватися IBM-сумісний комп’ютер з характеристиками не нижче:

- процесор Intel Core i3-5xxx;
- ОЗП 4 Гб;
- відеокарта з об’ємом пам’яті 2 ГБ;
- жорсткий диск об’ємом 64 ГБ;
- маніпулятор типу «миша» і / або клавіатура.

Як програмні засоби для належного функціонування програмного забезпечення повинні використовуватися:

- ОС Windows 10 та новіше;
- Python версії 3.8.8;
- система керування пакетами pip версії 21.0.1 та новіше;
- бібліотека Tensorflow версії 2.4.1;

- бібліотека Keras-Preprocessing версії 1.1.2;
- бібліотека Matplotlib версії 3.4.1;
- бібліотека PyQt версії 5.15.2.

6.3 Виконання програми

Для початку роботи необхідно запустити саму програму. Зробити це можна за допомогою файлу проекта з назвою main.py або за допомогою командного рядка (виклик системи без передачі жодних аргументів).

В результаті виконаних дій програму буде запущено, а користувач побачить на екрані головне вікно програмного засобу (рис. 6.1).

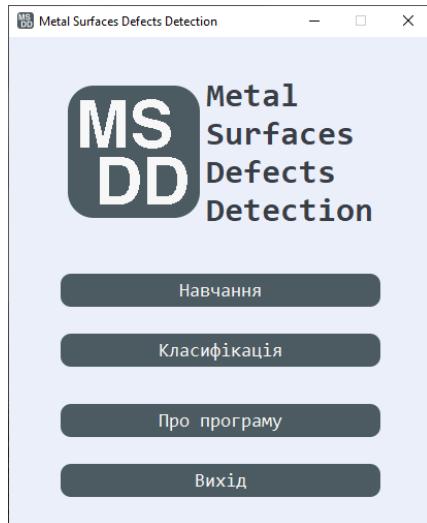


Рисунок 6.1 – Головне вікно програми

На головному вікні розташовано назву, логотип програми та чотири функціональні кнопки які відповідають за такі функції:

- кнопка навчання – для відкриття вікна зі списком моделей та їх подальшого навчання на наборі даних;
- кнопка класифікація – для відкриття вікна розпізнавання зображень;
- кнопка про програму – для відкриття вікна з інформацією про програму;
- кнопка вихід – для закриття головного вікна та виходу з програми.

Після натискання на кнопку навчання відкриється вікно зі списком всіх нейромережевих моделей сімейства VGGNet (рис. 6.2).

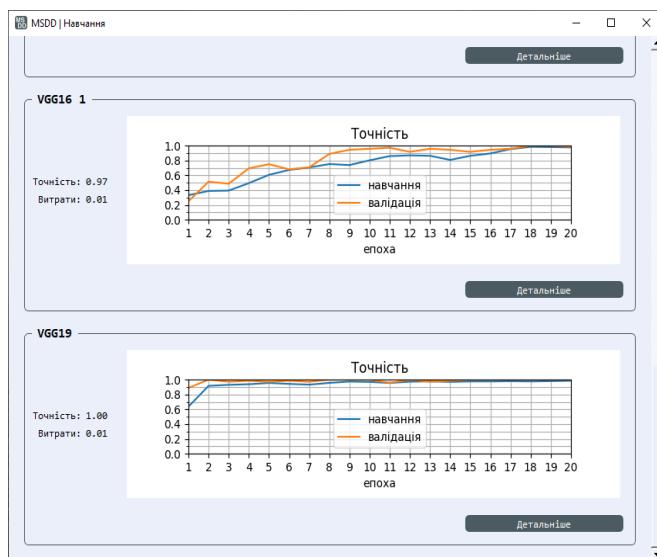


Рисунок 6.2 – Вікно зі списком всіх моделей

Користувач може переглянути короткі відомості про всі моделі та графіки зміни точності під час навчання гортаючи список. Після вибору бажаної моделі зі списку та натискання на кнопку детальніше відкриється вікно навчання з інформацією про вибрану нейромережеву модель (рис. 6.3).

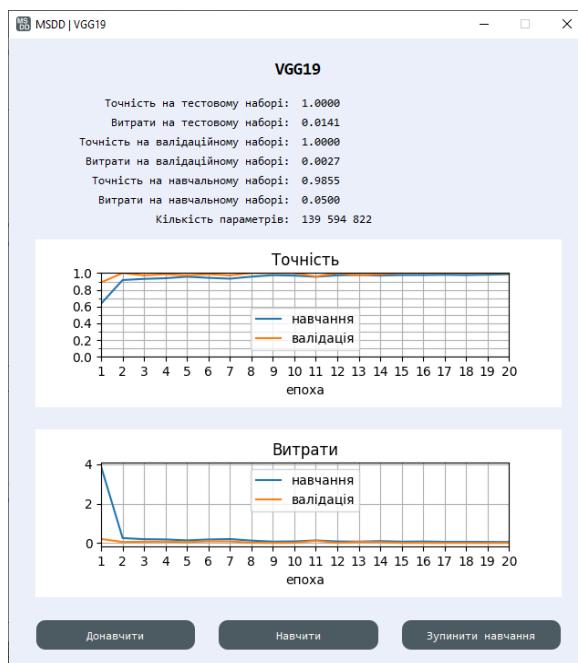


Рисунок 6.3 – Вікно навчання моделі VGG19

У вікні навчання моделі наведено основну статистичну інформацію про неї та відображені графіки зміни точності та витрат, що спостерігалися під час навчання.

Для навчання моделі на нових даних необхідно натиснути на кнопку навчити. В такому випадку вже навчена модель буде видалена з системи і процес навчання почнеться з самого початку. Для продовження навчання моделі на вибраному наборі даних необхідно натиснути на кнопку донаучити. Щоб зупинити процес навчання необхідно натиснути на кнопку зупинити навчання.

Для відкриття вікна розпізнавання зображень та виявлення дефектів на металевих поверхнях необхідно в головному вікні програми натиснути на кнопку класифікація. В результаті виконаних дій буде відкрито вікно розпізнавання зображень (рис. 6.4).

Для виявлення дефектів необхідно в комбобоксі вибрати одну з нейромережевих моделей VGG і натиснути на кнопку завантажити та класифікувати зображення. У відкритому вікні необхідно вибрати бажане зображення та натиснути кнопку відкрити. Після завершення процесу розпізнавання зображення буде відображенено тип виявленого дефекту.



Рисунок 6.4 – Вікно розпізнавання зображень

Для відкриття вікна з інформацією про програму (рис. 6.5) необхідно в головному вікні програми натиснути на кнопку про програму.

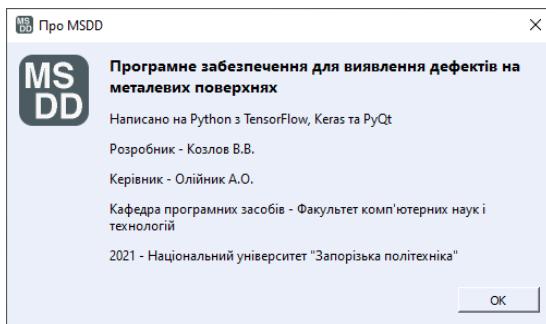


Рисунок 6.5 – Вікно з інформацією про програму

Для виходу з програми необхідно натиснути на кнопку вихід або на значок хрестика в правому верхньому куті головного вікна програми.

6.4 Повідомлення оператору

Для допомоги користувачу та виключення ймовірності втрати вже навченої нейромережової моделі під час процесу навчання на нових даних, передбачено спеціальне діалогове вікно. Воно пропонує користувачу підтвердити свій намір натиснувши на кнопку yes (рис. 6.6).

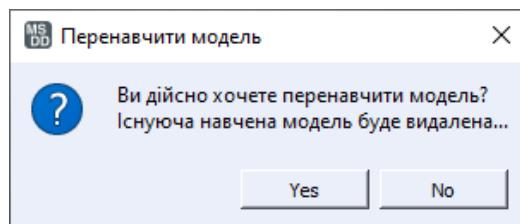


Рисунок 6.6 – Вікно підтвердження наміру перенавчання моделі

При закритті головного вікна програми відображається діалогове вікно з питанням про намір виходу з програми (рис. 6.7).

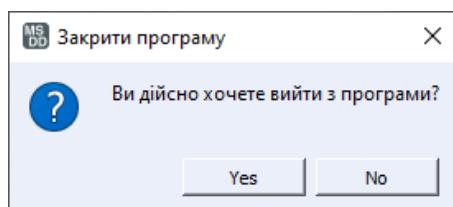


Рисунок 6.7 – Вікно попередження при закритті програми

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи бакалавра вирішено актуальну інженерну задачу виявлення дефектів на металевих поверхнях за допомогою методів в основу яких закладено використання нейромережевих технологій.

Проведено детальний аналіз предметної області, та існуючих алгоритмів. Визначено, що для вирішення такої задачі слід застосовувати згорктоі нейронні мережі, в особливості нейромережеві моделі VGG. Виконано дослідження архітектури та основних властивостей нейронних мереж сімейства VGGNet.

Відповідно до поставленої мети роботи було виконано аналіз вимог до програмного забезпечення та визначено основні програмні засоби. В якості інструментальних засобів обрано мову програмування Python, бібліотеки Keras, TensorFlow, PyQt та Matplotlib.

Реалізовано основну структуру програмного забезпечення та розроблено необхідні програмні класи. Спроєктовано та впроваджено всі нейромережеві моделі сімейства VGGNet. Обрано набір навчальних даних та оглянуто основні правила його будови. Спроєктовано та розроблено графічний інтерфейс користувача. Виконано тестування на наборі тест-кейсів та контролльному списку. Проведено експериментальне дослідження.

Вперше проведено порівняння ефективності впровадження та використання всіх нейромережевих моделей сімейства VGGNet для вирішення задачі виявлення дефектів на металевій поверхні сталової гарячекатаної смуги. Результати порівняння дозволяють стверджувати, що нейромережеві моделі VGG можуть досягати точності розпізнавання дефектів на навчальних даних в 98,55%, займаючи не більше 550 МБ вільного місця на використовуваному пристрої. Визначено, що при виявленні різних типів дефектів на металевій поверхні найбільшу точність має мережа VGG19 з 16 згортковими шарами.

Практична цінність розробленого програмного забезпечення полягає в тому, що воно надає гнучкі можливості навчання нейромережевих моделей сімейства VGGNet та їх подальшого вибору для виявлення дефектів на металевих поверхнях. Програма може використовуватися як для розпізнавання дефектів на виробництві, так і під час проведення навчання та підвищення кваліфікації персоналу.

В подальшому буде додано можливість роботи з програмним забезпеченням за допомогою командного рядка, що дозволить виконувати одночасну обробку великої кількості вхідних даних. Окрім цього буде додано нові екземпляри до навчальної вибірки, що дозволить збільшити кількість типів дефектів на металевих поверхнях, які може виявляти програма. Також планується впровадження системи спостерігання за діями користувача для подальшого аналізу результатів його роботи.

Основні положення роботи було представлено в тезах доповіді науково-практичної конференції Тиждень науки – 2021.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 100 найбільших приватних компаній України – Forbes.ua [Електронний ресурс]. – Режим доступу : <https://forbes.ua/ratings/100-naybilshikh-privatnikh-kompaniy-ukraini-02112020-408>
2. Підсумки роботи ГМК України у першому кварталі 2021 р. [Електронний ресурс]. – Режим доступу : <https://www.ukrmetprom.org/pidsumki-roboti-gmk-ukraini-u-pershomu/>
3. Обсяг реалізованої промислової продукції за видами діяльності [Електронний ресурс]. – Режим доступу : http://www.ukrstat.gov.ua/operativ/operativ2007/pr/orp/orp_u/arh_orp_u.html
4. Дубровин, В.И. Метод выделения профилей волнистости и шероховатости профилограмм металлических поверхностей с помощью вейвлет-анализа / В. И. Дубровин, П. А. Каморкин, Ю. В. Твердохлеб // Адаптивні системи автоматичного управління. – 2015. – № 1. – С. 26-31.
5. Надійність техніки. Терміни та визначення : ДСТУ 2860-94. – [Чинний від 1996-01-01]. – К. : Держстандарт України, 1996. – 96 с. – (Національний стандарт України).
6. Прокат чорних металів. Терміни та визначення дефектів поверхні : ДСТУ 2658-94. – [Чинний від 1995-07-01]. – К. : Держстандарт України, 1995. – 70 с. – (Національний стандарт України).
7. Левченко, Г.В. Технологія якості і сертифікація : навчальний посібник для студентів спеціальності 7.090404 – Обробка металів тиском / Г.В. Левченко, В.М. Самохвал. – Дніпродзержинськ : ДДТУ, 2009. – 117 с.
8. NEU surface defect database [Electronic resource]. – Access mode : http://faculty.neu.edu.cn/yunhyan/NEU_surface_defect_database.html
9. Герасимова, Н.С. Методы испытания и контроля качества металлов : учебное пособие по дисциплине «Специальные главы материаловедения» / Н.С. Герасимова. – Калуга : КФ МГТУ им. Н.Э. Баумана, 2019. – 44 с.
10. Неразрушающий контроль и диагностика : справочник / [В.В.

Клюев, Ф.Р. Соснин, А.В. Ковалев и др.]; под ред. В.В. Клюева. – М. : Машиностроение, 2003. – 656 с.

11. Выборнов, Б.И. Ультразвуковая дефектоскопия. 2-е изд. / Б.И. Выборнов. – М. : Металлургия, 1985. – 256 с.

12. Субботін, С.О. Нейронні мережі: теорія та практика: навч. посіб. / С.О. Субботін. – Житомир : Вид. О.О. Євенюк, 2020. – 184 с.

13. Нейронные сети для начинающих. Часть 1 / Хабр [Электронный ресурс]. – Режим доступа : <https://habr.com/ru/post/312450/>

14. The Neural Network Zoo – The Asimov Institute [Electronic resource]. – Access mode : <https://www.asimovinstitute.org/neural-network-zoo/>

15. Comprehensive Guide to Convolutional Neural Networks [Electronic resource]. – Access mode : <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

16. Backpropagation applied to handwritten zip code recognition / [Y. Lecun, B. Boser, JS. Denker et al.] // Neural Computation. – 1989. – Vol. 1, № 4. – P. 541–551.

17. Very Deep Convolutional Networks [Electronic resource]. – Access mode : <https://arxiv.org/abs/1409.1556/>

18. ImageNet [Electronic resource]. – Access mode : <http://www.image-net.org/challenges/LSVRC/>

19. Convolution [Electronic resource]. – Access mode : <http://www.songho.ca/dsp/convolution/convolution.html>

20. Как устроена сверточная нейронная сеть: архитектуры и параметры [Электронный ресурс]. – Режим доступа : <https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/>

21. AI, практический курс. Обзор нейронных сетей для классификации изображений / Блог компании Intel / Хабр [Электронный ресурс]. – Режим доступа : <https://habr.com/ru/company/intel/blog/415811/>

22. Полное руководство по функциям активации [Электронный

ресурс]. – Режим доступа : <https://www.machinelearningmastery.ru/complete-guide-of-activation-functions-34076e95d044/>

23. Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLu, tahn [Электронный ресурс]. – Режим доступа : <https://neurohive.io/ru/osnovy-data-science/activation-functions/>

24. Функции активации в искусственных нейронных сетях [Электронный ресурс]. – Режим доступа : <https://aisimple.ru/8-funkcii-aktivacii-v-iskusstvennyh-nejronnyh-setyah.html>

25. Softmax. Softmax-это логистическая функция для... [Электронный ресурс]. – Режим доступа : <https://congyuzhou.medium.com/softmax-3408fb42d55a>

26. VGG-16 convolutional neural network [Electronic resource]. – Access mode : <https://www.mathworks.com/help/deeplearning/ref/vgg16.html>

27. VGG16 and VGG19 [Electronic resource]. – Access mode : <https://keras.io/api/applications/vgg/>

28. vgg-nets | PyTorch [Electronic resource]. – Access mode : https://pytorch.org/hub/pytorch_vision_vgg/

29. ImageNet [Electronic resource]. – Access mode : <https://www.image-net.org/>

30. Welcome to Python.org [Electronic resource]. – Access mode : <https://www.python.org/>

31. TensorFlow [Electronic resource]. – Access mode : <https://www.tensorflow.org/>

32. NEU surface defect database [Electronic resource]. – Access mode : <https://drive.google.com/open?id=0B5OUtBsSxu1Bdjh4dk1SeGYtNFU>

ДОДАТОК А
ТЕКСТ ПРОГРАМИ

A.1 Файл main.py

```
import os
import sys
from PyQt5.QtWidgets import QApplication

from Model.ModelsKeeper import ModelsKeeper

from UI.MainWindow import MainWindow

def main():
    ModelsKeeper()

    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

    main()
```

A.2 Файл Dataset.py

```

def get_train_data(self):
    return self.__trainData

def get_valid_data(self):
    return self.__validData

def get_test_data(self):
    return self.__testData

def get_num_of_classes(self):
    return len(self.__trainData.class_indices)

def get_classes(self):
    return {value: key for key, value in self.__trainData.class_indices.items()}

```

A.3 Файл ModelsKeeper.py

```

from Tools.SingletonMeta import SingletonMeta
from Model.VGGNet import VGGNet
from Settings.DefaultSettings import DefaultSettings

class ModelsKeeper(metaclass=SingletonMeta):

    def __init__(self):
        self.models = {
            "VGG11": VGGNet("VGG11", DefaultSettings.VGG11_PATH),
            "VGG11_LRN": VGGNet("VGG11_LRN", DefaultSettings.VGG11_LRN_PATH),
            "VGG13": VGGNet("VGG13", DefaultSettings.VGG13_PATH),
            "VGG16": VGGNet("VGG16", DefaultSettings.VGG16_PATH),
            "VGG16_1": VGGNet("VGG16_1", DefaultSettings.VGG16_1_PATH),
            "VGG19": VGGNet("VGG19", DefaultSettings.VGG19_PATH),
        }

        for name, model in self.models.items():
            if model.is_saved_fitted_model():
                model.load_model_info()

    def get_models(self):
        return self.models

    def get_model_by_name(self, model_name):
        if model_name in self.models.keys():
            return self.models[model_name]
        else:
            return None

```

A.4 Файл VGGNet.py

```

from datetime import datetime
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np
import json
import os

```

```

import shutil

from Model.VGG11 import VGG11
from Model.VGG11_LRN import VGG11_LRN
from Model.VGG13 import VGG13
from Model.VGG16 import VGG16
from Model.VGG16_1 import VGG16_1
from Model.VGG19 import VGG19

from Data.Dataset import Dataset

class VGGNet:

    def __init__(self, name, path):
        self.name = name
        self.path = path

        self.modelPath = path + "model_weights.h5"
        self.dataPath = path + "data.json"

        self.model = None
        self.classes = {}
        self.fitHistory = {"loss": [], "accuracy": [],
                           "val_loss": [], "val_accuracy": []}
        self.evaluateResults = {"loss": None, "accuracy": None}
        self.parametersCount = None

        self.lastChangeTime = datetime.now()

    def is_saved_fitted_model(self):
        if not os.path.exists(self.modelPath) or not os.path.exists(self.dataPath):
            return False
        else:
            return True

    def set_new_model(self, num_of_classes):
        self.clear_model_variables()
        self.model = VGGNet.get_empty_model_architecture_by_name(self.name,
                                                               num_of_classes)

        self.lastChangeTime = datetime.now()

        if os.path.exists(self.path):
            shutil.rmtree(self.path)

    def set_new_model_and_compile(self, num_of_classes):
        self.set_new_model(num_of_classes)
        self.compile_model()

    def compile_model(self):
        self.model.compile(loss=tf.keras.losses.categorical_crossentropy,
                           optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                           metrics=['accuracy'])

        print(self.model.summary())
        self.parametersCount = self.model.count_params()

    def fit_model(self, dataset):
        self.classes = dataset.get_classes()

        early_stopping = tf.keras.callbacks.EarlyStopping(

```

```

        monitor='val_accuracy',
        mode='auto',
        min_delta=0,
        verbose=1,
        patience=3,
        restore_best_weights=False)

# Save model after each epoch
#
#
# filepath = self.path + "model_" + \
#     datetime.now().strftime("%Y%d%m_%H%M%S%f") + \
#     "_epoch_{epoch:04d}_acc_{val_accuracy:.4f}.h5"
# model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
#     filepath=filepath,
#     save_weights_only=False,
#     monitor='val_accuracy',
#     mode='auto',
#     save_best_only=False,
#     verbose=1)

class HistoryCallback(tf.keras.callbacks.Callback):
    def __init__(self, outer_class):
        super().__init__()
        self.outerClass = outer_class

    def on_epoch_end(self, epoch, logs=None):
        self.outerClass.fitHistory['loss'].append(logs['loss'])
        self.outerClass.fitHistory['accuracy'].append(logs['accuracy'])
        self.outerClass.fitHistory['val_loss'].append(logs['val_loss'])
        self.outerClass.fitHistory['val_accuracy'].\
            append(logs['val_accuracy'])

        self.outerClass.evaluateResults = self.model.\
            evaluate(dataset.get_test_data(), return_dict=True)

        self.outerClass.save_model(self.model)

        self.outerClass.lastChangeTime = datetime.now()

# Stop fitting, but it works only after end of epoch
#
# class TerminateOnFlag(tf.keras.callbacks.EarlyStopping):
#     def __init__(self, outer_class):
#         super().__init__()
#         self.outerClass = outer_class
#
#     def on_batch_end(self, batch, logs=None):
#         if self.outerClass.stopTrainingImmediately is True:
#             self.model.stop_training = True
#             self.outerClass.stopTrainingImmediately = False

callbacks = [early_stopping, HistoryCallback(self)]

self.model.fit(dataset.get_train_data(),
               batch_size=32,
               epochs=20,
               validation_data=dataset.get_valid_data(),
               verbose=1,
               shuffle=True,
               callbacks=callbacks)

```

```

def save_model(self, model=None):
    current_time = datetime.now().strftime("%Y%d%m_%H%M%S%f")
    temp_model_path = self.path + current_time + ".h5"
    temp_swap_model_path = self.path + current_time + "swap.h5"
    temp_data_path = self.path + current_time + ".json"
    temp_swap_data_path = self.path + current_time + "swap.json"

    if not os.path.exists(self.path):
        os.mkdir(self.path)

    if model is not None:
        model.save_weights(temp_model_path)
    else:
        self.model.save_weights(temp_model_path)

    data = {"classes": self.classes,
            "fitHistory": self.fitHistory,
            "evaluateResults": self.evaluateResults,
            "parametersCount": self.parametersCount}

    with open(temp_data_path, 'w') as outfile:
        json.dump(data, outfile, indent=4)

    if os.path.exists(self.modelPath):
        os.rename(self.modelPath, temp_swap_model_path)

    if os.path.exists(self.dataPath):
        os.rename(self.dataPath, temp_swap_data_path)

    os.rename(temp_model_path, self.modelPath)
    os.rename(temp_data_path, self.dataPath)

    if os.path.exists(temp_swap_model_path):
        os.remove(temp_swap_model_path)

    if os.path.exists(temp_swap_data_path):
        os.remove(temp_swap_data_path)

def predict(self, img_path):
    test_image = image.load_img(img_path, target_size=(224, 224))
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    test_image = test_image.astype('float32') / 255

    result = self.classes[np.argmax(self.model.predict(test_image))]
    if result in Dataset.DEFECT_TYPES_TRANSLATE:
        return Dataset.DEFECT_TYPES_TRANSLATE[result]
    else:
        return result

def load_model_info(self):
    self.lastChangeTime = datetime.now()
    with open(self.dataPath) as json_file:
        data = json.load(json_file)
        self.classes = {int(key): value for key, value in
                       data["classes"].items()}
        self.fitHistory = data["fitHistory"]
        self.evaluateResults = data["evaluateResults"]
        self.parametersCount = data["parametersCount"]

def load_model(self):
    self.load_model_info()

```

```

        self.model = VGGNet.get_empty_model_architecture_by_name(self.name,
                                                               len(self.classes))
        self.compile_model()
        self.model.load_weights(self.modelPath)

    def clear_model_variable(self):
        self.model = None

    def clear_model_variables(self):
        self.clear_model_variable()

        self.classes = {}
        self.fitHistory = {"loss": [], "accuracy": [],
                           "val_loss": [], "val_accuracy": []}
        self.evaluateResults = {"loss": None, "accuracy": None}
        self.parametersCount = None

    @staticmethod
    def get_empty_model_architecture_by_name(model_name, num_of_classes):
        if model_name == "VGG11":
            return VGG11.create_empty_model(num_of_classes)
        elif model_name == "VGG11_LRN":
            return VGG11_LRN.create_empty_model(num_of_classes)
        elif model_name == "VGG13":
            return VGG13.create_empty_model(num_of_classes)
        elif model_name == "VGG16":
            return VGG16.create_empty_model(num_of_classes)
        elif model_name == "VGG16_1":
            return VGG16_1.create_empty_model(num_of_classes)
        elif model_name == "VGG19":
            return VGG19.create_empty_model(num_of_classes)
        else:
            return None

```

A.5 Файл VGG11.py

```

import tensorflow as tf

class VGG11:

    @staticmethod
    def create_empty_model(num_of_classes):
        model = tf.keras.Sequential(name="vgg11")

        model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same',
                                       activation='relu', input_shape=(224, 224, 3), name="block1_conv1"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block1_pool"))

        model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',
                                       activation='relu', name="block2_conv1"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block2_pool"))

        model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                       activation='relu', name="block3_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                       activation='relu', name="block3_conv2"))

```

```

model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                    name="block3_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block4_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block4_conv2"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                    name="block4_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block5_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block5_conv2"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                    name="block5_pool"))

model.add(tf.keras.layers.Flatten(name="flatten"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
                               name="dense_fc1"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
                               name="dense_fc2"))
model.add(tf.keras.layers.Dense(units=num_of_classes, activation='softmax',
                               name="dense_fc3_softmax"))

return model

```

A.6 Файл VGG11_LRN

```

import tensorflow as tf

class VGG11_LRN:

    @staticmethod
    def create_empty_model(num_of_classes):
        model = tf.keras.Sequential(name="vgg11_LRN")

        model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same',
                                       activation='relu', input_shape=(224, 224, 3), name="block1_conv1"))
        model.add(tf.keras.layers.Lambda(tf.nn.local_response_normalization,
                                       name="block1_lrn"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block1_pool"))

        model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',
                                       activation='relu', name="block2_conv1"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block2_pool"))

        model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                       activation='relu', name="block3_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                       activation='relu', name="block3_conv2"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block3_pool"))

        model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                       activation='relu', name="block4_conv1"))

```

```
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block4_conv2"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
    name="block4_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block5_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block5_conv2"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
    name="block5_pool"))

model.add(tf.keras.layers.Flatten(name="flatten"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc1"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc2"))
model.add(tf.keras.layers.Dense(units=num_of_classes, activation='softmax',
    name="dense_fc3_softmax"))

return model
```

A.7 Файл VGG13.py

```
import tensorflow as tf

class VGG13:

    @staticmethod
    def create_empty_model(num_of_classes):
        model = tf.keras.Sequential(name="vgg13")

        model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same',
                                        activation='relu', input_shape=(224, 224, 3), name="block1_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same',
                                        activation='relu', name="block1_conv2"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block1_pool"))

        model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',
                                        activation='relu', name="block2_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',
                                        activation='relu', name="block2_conv2"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block2_pool"))

        model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                        activation='relu', name="block3_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                        activation='relu', name="block3_conv2"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                           name="block3_pool"))

        model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                        activation='relu', name="block4_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                        activation='relu', name="block4_conv2"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
```

```

        name="block4_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block5_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block5_conv2"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
    name="block5_pool"))

model.add(tf.keras.layers.Flatten(name="flatten"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc1"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc2"))
model.add(tf.keras.layers.Dense(units=num_of_classes, activation='softmax',
    name="dense_fc3_softmax"))

return model

```

A.8 Файл VGG16.py

```

import tensorflow as tf

class VGG16:

    @staticmethod
    def create_empty_model(num_of_classes):
        conv_model = tf.keras.applications.vgg16.VGG16(weights='imagenet',
            include_top=False, input_shape=(224, 224, 3))

        for layer in conv_model.layers:
            layer.trainable = False

        x = tf.keras.layers.Flatten(name="flatten")(conv_model.output)
        x = tf.keras.layers.Dense(4096, activation='relu', name="dense_fc1")(x)
        x = tf.keras.layers.Dense(4096, activation='relu', name="dense_fc2")(x)
        output = tf.keras.layers.Dense(num_of_classes, activation='softmax',
            name="dense_fc3_softmax")(x)

        model = tf.keras.models.Model(inputs=conv_model.input, outputs=output,
            name="vgg16")

        return model

    @staticmethod
    def create_empty_model_2(num_of_classes):
        model = tf.keras.models.Sequential(name="vgg16")

        model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same',
            activation='relu', input_shape=(224, 224, 3), name="block1_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same',
            activation='relu', name="block1_conv2"))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
            name="block1_pool"))

        model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',
            activation='relu', name="block2_conv1"))
        model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',

```

```
activation='relu', name="block2_conv2"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                   name="block2_pool"))

model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                 activation='relu', name="block3_conv1"))
model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                 activation='relu', name="block3_conv2"))
model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
                                 activation='relu', name="block3_conv3"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                   name="block3_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block4_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block4_conv2"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block4_conv3"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                   name="block4_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block5_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block5_conv2"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
                                 activation='relu', name="block5_conv3"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
                                   name="block5_pool"))

model.add(tf.keras.layers.Flatten(name="flatten"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
                               name="dense_fc1"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
                               name="dense_fc2"))
model.add(tf.keras.layers.Dense(units=num_of_classes, activation='softmax',
                               name="dense_fc3_softmax"))

return model
```

A.9 Файл VGG16_1.py

```

model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',
    activation='relu', name="block2_conv1"))
model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same',
    activation='relu', name="block2_conv2"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
    name="block2_pool"))

model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
    activation='relu', name="block3_conv1"))
model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3, padding='same',
    activation='relu', name="block3_conv2"))
model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=1, padding='same',
    activation='relu', name="block3_conv3"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
    name="block3_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block4_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block4_conv2"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=1, padding='same',
    activation='relu', name="block4_conv3"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
    name="block4_pool"))

model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block5_conv1"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=3, padding='same',
    activation='relu', name="block5_conv2"))
model.add(tf.keras.layers.Conv2D(filters=512, kernel_size=1, padding='same',
    activation='relu', name="block5_conv3"))
model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='same',
    name="block5_pool"))

model.add(tf.keras.layers.Flatten(name="flatten"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc1"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc2"))
model.add(tf.keras.layers.Dense(units=num_of_classes, activation='softmax',
    name="dense_fc3_softmax"))

return model

```

A.10 Файл VGG19.py

```

import tensorflow as tf

class VGG19:

    @staticmethod
    def create_empty_model(num_of_classes):
        conv_model = tf.keras.applications.vgg19.VGG19(weights='imagenet',
            include_top=False, input_shape=(224, 224, 3))

        for layer in conv_model.layers:
            layer.trainable = False

```



```

model.add(tf.keras.layers.Flatten(name="flatten"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc1"))
model.add(tf.keras.layers.Dense(units=4096, activation='relu',
    name="dense_fc2"))
model.add(tf.keras.layers.Dense(units=num_of_classes, activation='softmax',
    name="dense_fc3_softmax"))

# draw vgg19 structure plot
# from tensorflow.keras.utils import plot_model
# plot_model(model, to_file='D:\\vgg19_plot.png', show_shapes=True,
#             show_layer_names=True)

return model

```

A.11 Файл DefaultSettings.py

```

import os
from pathlib import Path

class DefaultSettings:

    PROJECT_PATH = str(Path(__file__).parent.parent)

    DATASET_PATH = PROJECT_PATH + os.sep + "dataset" + os.sep + "Data" + os.sep

    MODELS_PATH = PROJECT_PATH + os.sep + "models" + os.sep

    VGG11_PATH = MODELS_PATH + "VGG11" + os.sep
    VGG11_LRN_PATH = MODELS_PATH + "VGG11_LRN" + os.sep
    VGG13_PATH = MODELS_PATH + "VGG13" + os.sep
    VGG16_PATH = MODELS_PATH + "VGG16" + os.sep
    VGG16_1_PATH = MODELS_PATH + "VGG16_1" + os.sep
    VGG19_PATH = MODELS_PATH + "VGG19" + os.sep

    RESOURCES_PATH = PROJECT_PATH + os.sep + "resources" + os.sep

    ABOUT_PROGRAM = '<h3><strong>Програмне забезпечення для виявлення дефектів на ' \
                    '<strong>металевих поверхнях</strong></h3><p />' \
                    '<p>Написано на Python з TensorFlow, Keras та PyQt</p><p />' \
                    '<p>Розробник - Козлов В.В.</p>' \
                    '<p>Керівник - Олійник А.О.</p><p />' \
                    '<p>Кафедра програмних засобів - Факультет комп'ютерних наук ' \
                    '& i технологій</p>' \
                    '<p>2021 - Національний університет "Запорізька політехніка"</p>'


```

A.12 Файл SingletonMeta.py

```

class SingletonMeta(type):

    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:

```

```

        instance = super().__call__(*args, **kwargs)
        cls._instances[cls] = instance
    return cls._instances[cls]

```

A.13 Файл StoppableThread.py

```

import sys
import trace
import threading

class StoppableThread(threading.Thread):
    def __init__(self, *args, **keywords):
        threading.Thread.__init__(self, *args, **keywords)
        self.killed = False

    def start(self):
        self.__run_backup = self.run
        self.run = self.__run
        threading.Thread.start(self)

    def __run(self):
        sys.settrace(self.globaltrace)
        self.__run_backup()
        self.run = self.__run_backup

    def globaltrace(self, frame, why, arg):
        if why == 'call':
            return self.localtrace
        else:
            return None

    def localtrace(self, frame, why, arg):
        if self.killed:
            if why == 'line':
                raise SystemExit()
        return self.localtrace

    def kill(self):
        self.killed = True

```

A.14 Файл ClassificationWindow.py

```

from PyQt5 import QtWidgets, QtGui
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMainWindow
from PyQt5.QtCore import pyqtSlot, Qt, QSize

from Model.ModelsKeeper import ModelsKeeper
from Settings.DefaultSettings import DefaultSettings
from UI.classificationWindowDesign import Ui_mainWindow

from Tools.StoppableThread import StoppableThread

```

```

class ClassificationWindow(QMainWindow, Ui_mainWindow):

    def __init__(self):
        self.model = None
        self.loadModelThread = None
        self.predictionThread = None

        super(ClassificationWindow, self).__init__()
        self.setupUi(self)

        icon = QIcon()
        icon.addFile(DefaultSettings.RESOURCES_PATH + "icon.png", QSize(),
                     QIcon.Normal, QIcon.Off)
        self.setWindowIcon(icon)

        self.setFixedSize(self.size())
        self.setWindowFlags(self.windowFlags() & ~Qt.WindowMaximizeButtonHint)

        self.image.setPixmap(QtGui.QPixmap(DefaultSettings.RESOURCES_PATH +
                                         "placeholder.png"))

        models_keeper = ModelsKeeper()
        for name, model in models_keeper.get_models().items():
            if model.is_saved_fitted_model():
                self.model_select.addItem(name)

        if self.model_select.count() > 0:
            self.model_select.setCurrentIndex(self.model_select.count() - 1)

        self.classify_b.clicked.connect(self.on_classify_b_click)
        self.model_select.currentIndexChanged.connect(
            self.on_model_select_current_index_changed)
        self.on_model_select_current_index_changed()

    @pyqtSlot()
    def on_model_select_current_index_changed(self):
        if self.loadModelThread is not None and self.loadModelThread.is_alive():
            self.loadModelThread.kill()
            self.loadModelThread.join()

        if self.model is not None:
            self.model.model = None

        self.model = ModelsKeeper().get_model_by_name(
            self.model_select.currentText())

        if self.model is not None:
            self.loadModelThread = StoppableThread(target=self.load_model)
            self.loadModelThread.start()

    @pyqtSlot()
    def on_classify_b_click(self):
        if self.model is not None:

            self.image.setPixmap(QtGui.QPixmap(DefaultSettings.RESOURCES_PATH +
                                              "placeholder.png"))
            self.image.repaint()

            self.prediction.setText("передбачення")
            self.prediction.repaint()

            image_path, _ = QtWidgets.QFileDialog.getOpenFileName(self,

```

```

'Завантажити зображення', "", "Images (*.png *.jpg *.bmp)")
if image_path != "":
    self.image.setPixmap(QtGui.QPixmap(image_path))
    self.image.repaint()

    self.prediction.setText("* передбачення...")
    self.prediction.repaint()

    self.classify_b.setEnabled(False)

    self.predictionThread = StoppableThread(self.classify(image_path))
    self.predictionThread.start()

def load_model(self):
    self.model.load_model()

def classify(self, image_path):
    if self.loadModelThread is not None and self.loadModelThread.is_alive():
        self.loadModelThread.join()

    res_class = self.model.predict(image_path)
    self.prediction.setText(res_class)
    self.prediction.repaint()

    self.classify_b.setEnabled(True)

def closeEvent(self, event):
    if self.predictionThread is not None and self.predictionThread.is_alive():
        self.predictionThread.kill()
        self.predictionThread.join()

    if self.loadModelThread is not None and self.loadModelThread.is_alive():
        self.loadModelThread.kill()
        self.loadModelThread.join()

    if self.model is not None:
        self.model.clear_model_variable()

```

A.15 Файл classificationWindowDesign.py

```

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_mainWindow(object):
    def setupUi(self, mainWindow):
        mainWindow.setObjectName("mainWindow")
        mainWindow.resize(346, 500)
        mainWindow.setStyleSheet("background-color: #eaeffa;")
        self.main = QtWidgets.QWidget(mainWindow)
        self.main.setObjectName("main")
        self.main_layout = QtWidgets.QVBoxLayout(self.main)
        self.main_layout.setContentsMargins(20, 20, 20, 20)
        self.main_layout.setObjectName("main_layout")
        self.model_select = QtWidgets.QComboBox(self.main)
        font = QtGui.QFont()
        font.setFamily("Consolas")
        font.setPointSize(10)
        self.model_select.setFont(font)
        self.model_select.setStyleSheet("QComboBox {\n"

```

```

        " color: #4c5b61;\n"
        " background-color: #f7f7f7;\n"
        " border-radius: 10px;\n"
        " border: 2px solid;\n"
        " border-color: #4c5b61;\n"
        " spacing: 40px;\n"
        " padding-left: 5px;\n"
        " height: 28px;\n"
        " margin-bottom: 10px;\n"
    "}\n"
"\n"
"QComboBox:hover:!pressed {\n"
"    background-color: #f0f0f0;\n"
"}\n"
"\n"
"QComboBox::drop-down:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}\n"
"\n"
"QComboBox::drop-down:pressed {\n"
"    background-color: #393e46;\n"
"}\n"
"\n"
"QComboBox::drop-down {\n"
"    subcontrol-origin: padding;\n"
"    subcontrol-position: top right;\n"
"    width: 20px;\n"
"    border-left: 2px solid #4c5b61;\n"
"    border-top-right-radius: 5px;\n"
"    border-bottom-right-radius: 5px;\n"
"    background-color: #4c5b61;\n"
"}\n"
"\n"
"QComboBox QAbstractItemView{\n"
"    background-color: #f7f7f7;\n"
"    border: 2px solid #4c5b61;\n"
"    border-radius: 1px;\n"
"\n"
"    selection-background-color: lightgray;\n"
"    selection-color: #4c5b61;\n"
"    selection-padding-left: 5px;\n"
"\n"
"    outline: none;\n"
"}\n"
"")

        self.model_select.setObjectName("model_select")
        self.main_layout.addWidget(self.model_select)
        self.classify_b = QtWidgets.QPushButton(self.main)
        self.classify_b.setStyleSheet("QPushButton {\n"
        "    color: #f7f7f7;\n"
        "    background-color: #4c5b61;\n"
        "    border-radius: 10px;\n"
        "    margin: 10px 0px 20px 0px;\n"
        "    height: 32px;\n"
        "    font: 10pt \\'Consolas\\';\n"
    "}\n"
"\n"
"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"

```

```

"\n"
"QPushButton:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}\n"
""")
    self.classify_b.setObjectName("classify_b")
    self.main_layout.addWidget(self.classify_b)
    self.image = QtWidgets.QLabel(self.main)
    sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Ignored,
        QtWidgets.QSizePolicy.Ignored)
    sizePolicy.setHorizontalStretch(0)
    sizePolicy.setVerticalStretch(0)
    sizePolicy.setHeightForWidth(self.image.sizePolicy().hasHeightForWidth())
    self.image.setSizePolicy(sizePolicy)
    self.image.setScaledContents(True)
    self.image.setAlignment(QtCore.Qt.AlignCenter)
    self.image.setObjectName("image")
    self.main_layout.addWidget(self.image)
    self.prediction = QtWidgets.QLabel(self.main)
    self.prediction.setMinimumSize(QtCore.QSize(0, 32))
    self.prediction.setMaximumSize(QtCore.QSize(16777215, 32))
    self.prediction.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 12pt;\n"
"    font-weight: bold;\n"
"}")
    self.prediction.setAlignment(QtCore.Qt.AlignCenter)
    self.prediction.setObjectName("prediction")
    self.main_layout.addWidget(self.prediction)
    mainWindow.setCentralWidget(self.main)

    self.retranslateUi(mainWindow)
    QtCore.QMetaObject.connectSlotsByName(mainWindow)

def retranslateUi(self, mainWindow):
    _translate = QtCore.QCoreApplication.translate
    mainWindow.setWindowTitle(_translate("mainWindow", "MSDD | Класифікація"))
    self.model_select.setPlaceholderText(_translate("mainWindow",
                                                    "Виберіть модель"))
    self.classify_b.setText(_translate("mainWindow",
                                      "Завантажити та класифікувати зображення"))
    self.prediction.setText(_translate("mainWindow", "передбачення"))

```

A.16 Файл MainWindow.py

```

from PyQt5 import QtGui
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMainWindow, QMessageBox
from PyQt5.QtCore import pyqtSlot, Qt, QSize

from Settings.DefaultSettings import DefaultSettings
from UI.mainWindowDesign import Ui_mainWindow
from UI.ModelListWindow import ModelListWindow
from UI.ClassificationWindow import ClassificationWindow

class MainWindow(QMainWindow, Ui_mainWindow):

    def __init__(self):

```

```

super(MainWindow, self).__init__()
self.setupUi(self)

icon = QIcon()
icon.addFile(DefaultSettings.RESOURCES_PATH + "icon.png", QSize(),
             QIcon.Normal, QIcon.Off)
self.setWindowIcon(icon)

self.setFixedSize(self.size())
self.setWindowFlags(self.windowFlags() & ~Qt.WindowMaximizeButtonHint)

self.info_logo.setPixmap(QtGui.QPixmap(DefaultSettings.RESOURCES_PATH +
                                         "icon.png"))

self.training_b.clicked.connect(self.on_training_b_click)
self.classification_b.clicked.connect(self.on_classification_b_click)
self.about_b.clicked.connect(self.on_about_b_click)
self.exit_b.clicked.connect(self.on_exit_b_click)

def closeEvent(self, event):
    close_dialog_result = QMessageBox.question(self, 'Закрити програму',
                                                'Ви дійсно хочете вийти з програми?',
                                                QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

    if close_dialog_result == QMessageBox.Yes:
        event.accept()
    else:
        event.ignore()

@pyqtSlot()
def on_training_b_click(self):
    self.modelListWindow = ModelListWindow()
    self.modelListWindow.setWindowModality(Qt.ApplicationModal)
    self.modelListWindow.show()

@pyqtSlot()
def on_classification_b_click(self):
    self.classificationWindow = ClassificationWindow()
    self.classificationWindow.setWindowModality(Qt.ApplicationModal)
    self.classificationWindow.show()

@pyqtSlot()
def on_about_b_click(self):
    QMessageBox.about(self, "Про MSDD", DefaultSettings.ABOUT_PROGRAM)

    # Information about Qt
    # QMessageBox.aboutQt(self)

@pyqtSlot()
def on_exit_b_click(self):
    self.close()

```

A.17 Файл mainWindowDesign.py

```

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_mainWindow(object):
    def setupUi(self, mainWindow):

```

```

mainWindow.setObjectName("mainWindow")
mainWindow.resize(410, 475)
mainWindow.setStyleSheet("background-color: #eaeffa;")
self.main = QtWidgets.QWidget(mainWindow)
self.main.setObjectName("main")
self.main_layout = QtWidgets.QVBoxLayout(self.main)
self.main_layout.setContentsMargins(20, 20, 20, 20)
self.main_layout.setObjectName("main_layout")
self.info_layout = QtWidgets.QHBoxLayout()
self.info_layout.setObjectName("info_layout")
spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
                                  QtWidgets.QSizePolicy.Minimum)
self.info_layout.addItem(spacerItem)
self.info_logo = QtWidgets.QLabel(self.main)
self.info_logo.setText("")
self.info_logo.setPixmap(QtGui.QPixmap("D:/MSDD/resources/icon.png"))
self.info_logo.setObjectName("info_logo")
self.info_layout.addWidget(self.info_logo)
self.info_text = QtWidgets.QLabel(self.main)
self.info_text.setStyleSheet("QLabel {\n"
"    color: #393e46;\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 24pt;\n"
"    font-weight: bold;\n"
"}\n"
""")
self.info_text.setObjectName("info_text")
self.info_layout.addWidget(self.info_text)
spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
                                   QtWidgets.QSizePolicy.Minimum)
self.info_layout.addItem(spacerItem1)
self.main_layout.addLayout(self.info_layout)
self.training_b = QtWidgets.QPushButton(self.main)
font = QtGui.QFont()
font.setFamily("Consolas")
font.setPointSize(14)
font.setBold(False)
font.setItalic(False)
font.setWeight(50)
self.training_b.setFont(font)
self.training_b.setStyleSheet("QPushButton {\n"
"    color: #f7f7f7;\n"
"    background-color: #4c5b61;\n"
"    border-radius: 10px;\n"
"    margin: 20px 30px 10px 30px;\n"
"    height: 32px;\n"
"    font: 14pt \\'Consolas\\';\n"
"}\n"
"\n"
"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"
"\n"
"QPushButton:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}")
self.training_b.setObjectName("training_b")
self.main_layout.addWidget(self.training_b)
self.classification_b = QtWidgets.QPushButton(self.main)
self.classification_b.setStyleSheet("QPushButton {\n"
"    color: #f7f7f7;\n"
"
```

```

        "background-color: #4c5b61;\n"
        "border-radius: 10px;\n"
        "margin: 10px 30px 10px 30px;\n"
        "height: 32px;\n"
        "font: 14pt \\'Consolas\\';\n"
    "}\n"
"\n"
"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"
"\n"
"QPushButton:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}")
    self.classification_b.setObjectName("classification_b")
    self.main_layout.addWidget(self.classification_b)
    self.about_b = QtWidgets.QPushButton(self.main)
    self.about_b.setStyleSheet("QPushButton {\n"
        "color: #f7f7f7;\n"
        "background-color: #4c5b61;\n"
        "border-radius: 10px;\n"
        "margin: 20px 30px 10px 30px;\n"
        "height: 32px;\n"
        "font: 14pt \\'Consolas\\';\n"
    "}\n"
"\n"
"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"
"\n"
"QPushButton:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}")
    self.about_b.setObjectName("about_b")
    self.main_layout.addWidget(self.about_b)
    self.exit_b = QtWidgets.QPushButton(self.main)
    self.exit_b.setStyleSheet("QPushButton {\n"
        "color: #f7f7f7;\n"
        "background-color: #4c5b61;\n"
        "border-radius: 10px;\n"
        "margin: 10px 30px 10px 30px;\n"
        "height: 32px;\n"
        "font: 14pt \\'Consolas\\';\n"
    "}\n"
"\n"
"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"
"\n"
"QPushButton:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}")
    self.exit_b.setObjectName("exit_b")
    self.main_layout.addWidget(self.exit_b)
    mainWindow.setCentralWidget(self.main)

    self.retranslateUi(mainWindow)
    QtCore.QMetaObject.connectSlotsByName(mainWindow)

```

```

def retranslateUi(self, mainWindow):
    _translate = QtCore.QCoreApplication.translate
    mainWindow.setWindowTitle(_translate("mainWindow",
                                         "Metal Surfaces Defects Detection"))
    self.info_text.setText(_translate("mainWindow", "Metal\n"
                                         "Surfaces\n"
                                         "Defects\n"
                                         "Detection"))
    self.training_b.setText(_translate("mainWindow", "Навчання"))
    self.classification_b.setText(_translate("mainWindow", "Класифікація"))
    self.about_b.setText(_translate("mainWindow", "Про програму"))
    self.exit_b.setText(_translate("mainWindow", "Вихід"))

```

A.18 Файл MatplotlibWidget.py

```

from matplotlib.figure import Figure
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from PyQt5 import QtWidgets

class MatplotlibWidget(QtWidgets.QWidget):

    def __init__(self, parent=None):
        QtWidgets.QWidget.__init__(self, parent)

        self.vbl = QtWidgets.QVBoxLayout()
        self.setLayout(self.vbl)

        self.fig = Figure()
        self.plot = self.fig.add_subplot()
        self.canvas = FigureCanvasQTAgg(self.fig)
        FigureCanvasQTAgg.setSizePolicy(self.canvas, QtWidgets.QSizePolicy.Expanding,
                                        QtWidgets.QSizePolicy.Expanding)
        FigureCanvasQTAgg.updateGeometry(self.canvas)
        self.vbl.addWidget(self.canvas)

    def make_plot(self, ox, oy, x_label="", y_label="", title ""):
        self.fig.clear()
        self.plot = self.fig.add_subplot(111)
        self.fig.subplots_adjust(bottom=0.3, top=0.8)
        self.plot.grid(True)

        for key, value in oy.items():
            self.plot.plot(ox, value, label=key)

        self.plot.set_title(title, fontsize=12)
        self.plot.set_xlabel(x_label, fontsize=10)
        self.plot.set_ylabel(y_label, fontsize=10)

        if len(ox) > 1:
            self.plot.set_xlim(1, ox[-1])
            self.plot.set_xticks([i for i in range(1, ox[-1]+1)])

        if title == "Точність":
            self.plot.legend(loc='lower center', fontsize=10)
            self.plot.set_ylim(0.0, 1.0)
            self.plot.set_yticks([i / 10.0 for i in range(0, 11, 2)])
            self.plot.set_yticks([i / 10.0 for i in range(0, 11, 1)], minor=True)
            self.plot.grid(True, which='both')

```

```

        elif title == "Витрати":
            self.plot.legend(loc='upper center', fontsize=10)
        else:
            self.plot.legend(fontsize=10)

        self.canvas.draw()

# Another approach to create MatplotlibWidget
#
#
# class MplCanvas(FigureCanvasQTAgg):
#     def __init__(self):
#         self.fig = Figure()
#         self.ax = self.fig.add_subplot(111)
#         FigureCanvasQTAgg.__init__(self, self.fig)
#         FigureCanvasQTAgg.setSizePolicy(self, QtWidgets.QSizePolicy.Expanding,
#                                         QtWidgets.QSizePolicy.Expanding)
#         FigureCanvasQTAgg.updateGeometry(self)
#
#
# class MplWidget(QtWidgets.QWidget):
#     def __init__(self, parent=None):
#         QtWidgets.QWidget.__init__(self, parent)
#         self.canvas = MplCanvas()
#         self.vbl = QtWidgets.QVBoxLayout()
#         self.vbl.addWidget(self.canvas)
#         self.setLayout(self.vbl)

```

A.19 Файл ModelInfoWindow.py

```

from datetime import datetime

from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMainWindow, QMessageBox
from PyQt5.QtCore import pyqtSlot, QTimer, Qt, QSize

from Data.Dataset import Dataset
from Settings.DefaultSettings import DefaultSettings
from UI.modelInfoWindowDesign import Ui_mainWindow

from Model.ModelsKeeper import ModelsKeeper

from Tools.StoppableThread import StoppableThread

class ModelInfoWindow(QMainWindow, Ui_mainWindow):

    def __init__(self, parent_window, model_name):

        self.parentWindow = parent_window
        self.modelName = model_name
        self.model = None
        self.lastInfoChange = datetime.now()
        self.isModelChanged = False

        self.trainThread = None

        super(ModelInfoWindow, self).__init__()

```

```

    self.setupUi(self)

    icon = QIcon()
    icon.addFile(DefaultSettings.RESOURCES_PATH + "icon.png", QSize(),
                 QIcon.Normal, QIcon.Off)
    self.setWindowIcon(icon)

    self.setFixedSize(self.size())
    self.setWindowFlags(self.windowFlags() & ~Qt.WindowMaximizeButtonHint)

    self.setWindowTitle(self.windowTitle() + self.modelName)

    self.model = ModelsKeeper().get_model_by_name(self.modelName)

    self.model_name_value.setText(self.modelName)

    self.set_model_info()

    self.stop_b.setEnabled(False)

    self.train_b.clicked.connect(self.on_train_b_click)
    self.retrain_b.clicked.connect(self.on_retrain_b_click)
    self.stop_b.clicked.connect(self.on_stop_b_click)

    self.timer = QTimer()
    self.timer.timeout.connect(self.check_model_for_updates)
    self.timer.start(1000)

def update_model_info(self):
    self.isModelChanged = True
    self.set_model_info()

def set_model_info(self):
    self.lastInfoChange = datetime.now()

    if self.model is not None and self.model.is_saved_fitted_model():

        self.test_acc_value.setText(f'{self.model.evaluateResults["accuracy"]:.4f}')
        self.test_loss_value.setText(
            f'{self.model.evaluateResults["loss"]:.4f}')
        self.val_acc_value.setText(
            f'{self.model.fitHistory["val_accuracy"][-1]:.4f}')
        self.val_loss_value.setText(
            f'{self.model.fitHistory["val_loss"][-1]:.4f}')
        self.train_acc_value.setText(
            f'{self.model.fitHistory["accuracy"][-1]:.4f}')
        self.train_loss_value.setText(
            f'{self.model.fitHistory["loss"][-1]:.4f}')
        self.params_value.setText(
            '{},.format(self.model.parametersCount).replace(',', ' '))'

        self.matplotlib_accuracy_plot.make_plot(
            range(1, len(self.model.fitHistory["accuracy"]) + 1),
            {"навчання": self.model.fitHistory["accuracy"],
             "валідація": self.model.fitHistory["val_accuracy"]},
            x_label="епоха", title="Точність")

        self.matplotlib_loss_plot.make_plot(
            range(1, len(self.model.fitHistory["loss"]) + 1),
            {"навчання": self.model.fitHistory["loss"],
             "валідація": self.model.fitHistory["val_loss"]},
            x_label="епоха", title="Втрати")

```

```

        x_label="епоха", title="Витрати")
else:
    self.test_acc_value.setText("")
    self.test_loss_value.setText("")
    self.val_acc_value.setText("")
    self.val_loss_value.setText("")
    self.train_acc_value.setText("")
    self.train_loss_value.setText("")
    self.params_value.setText("")

    self.matplotlib_accuracy_plot.make_plot(
        [],
        {"навчання": [],
         "валідація": []},
        x_label="епоха", title="Точність")

    self.matplotlib_loss_plot.make_plot(
        [],
        {"навчання": [],
         "валідація": []},
        x_label="епоха", title="Витрати")

@pyqtSlot()
def on_train_b_click(self):
    self.model_name_value.setText(self.modelName + "* навчання...")
    self.model_name_value.repaint()

    self.stop_b.setEnabled(True)
    self.train_b.setEnabled(False)
    self.retrain_b.setEnabled(False)

    self.trainThread = StoppableThread(target=self.train)
    self.trainThread.start()

def train(self):
    try:
        if self.model.is_saved_fitted_model():
            dataset = Dataset(DefaultSettings.DATASET_PATH)

            self.model.load_model()
            self.model.fit_model(dataset)
        else:
            self.retrain()
    finally:
        self.on_train_end()

@pyqtSlot()
def on_retrain_b_click(self):
    if self.model is not None and self.model.is_saved_fitted_model():
        dialog_result = QMessageBox.question(self, 'Перенавчити модель',
                                              'Ви дійсно хочете перенавчити модель?\n'
                                              'Існуюча навчена модель буде видалена...', QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
        if dialog_result == QMessageBox.No:
            return

        self.model_name_value.setText(self.modelName + "* навчання...")
        self.model_name_value.repaint()

        self.stop_b.setEnabled(True)
        self.train_b.setEnabled(False)
        self.retrain_b.setEnabled(False)

```

```

        self.trainThread = StoppableThread(target=self.retrain)
        self.trainThread.start()

    def retrain(self):
        try:
            dataset = Dataset(DefaultSettings.DATASET_PATH)

            self.model.set_new_model_and_compile(dataset.get_num_of_classes())
            self.model.fit_model(dataset)
        finally:
            self.on_train_end()

    def cancel_training(self):
        if self.trainThread is not None and self.trainThread.is_alive():
            self.model_name_value.setText(self.modelName + " * зупинка навчання...")
            self.model_name_value.repaint()
            self.trainThread.kill()
            self.trainThread.join()

    def on_train_end(self):
        self.check_model_for_updates()

        self.model.clear_model_variables()
        if self.model.is_saved_fitted_model():
            self.model.load_model_info()

        self.model_name_value.setText(self.modelName)
        self.model_name_value.repaint()

        self.train_b.setEnabled(True)
        self.retrain_b.setEnabled(True)
        self.stop_b.setEnabled(False)

    @pyqtSlot()
    def on_stop_b_click(self):
        self.cancel_training()

    @pyqtSlot()
    def check_model_for_updates(self):
        if self.lastInfoChange <= self.model.lastChangeTime:
            self.update_model_info()

    def closeEvent(self, event):
        self.timer.stop()
        self.cancel_training()

        if self.isModelChanged:
            self.parentWindow.update_scroll_area_content()

```

A.20 Файл modelInfoWindowDesign.py

```

from PyQt5 import QtCore, QtGui, QtWidgets

from UI.MatplotlibWidget import MatplotlibWidget

class Ui_mainWindow(object):
    def setupUi(self, mainWindow):
        mainWindow.setObjectName("mainWindow")

```

```

        mainWindow.resize(635, 683)
        mainWindow.setStyleSheet("background-color: #eaeffa;")
        self.main = QtWidgets.QWidget(mainWindow)
        self.main.setObjectName("main")
        self.main_layout = QtWidgets.QVBoxLayout(self.main)
        self.main_layout.setContentsMargins(20, 20, 20, 20)
        self.main_layout.setObjectName("main_layout")
        self.model_name_value = QtWidgets.QLabel(self.main)
        self.model_name_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 14pt;\n"
"    font-weight: bold;\n"
"    margin-bottom: 10px;\n"
"}")
        self.model_name_value.setAlignment(QtCore.Qt.AlignCenter)
        self.model_name_value.setObjectName("model_name_value")
        self.main_layout.addWidget(self.model_name_value)
        self.statistic_layout = QtWidgets.QGridLayout()
        self.statistic_layout.setObjectName("statistic_layout")
        self.train_loss_label = QtWidgets.QLabel(self.main)
        self.train_loss_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.train_loss_label.setAlignment(
            QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.train_loss_label.setObjectName("train_loss_label")
        self.statistic_layout.addWidget(self.train_loss_label, 5, 0, 1, 1)
        self.test_acc_label = QtWidgets.QLabel(self.main)
        self.test_acc_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.test_acc_label.setAlignment(
            QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.test_acc_label.setObjectName("test_acc_label")
        self.statistic_layout.addWidget(self.test_acc_label, 0, 0, 1, 1)
        self.val_acc_label = QtWidgets.QLabel(self.main)
        self.val_acc_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.val_acc_label.setAlignment(
            QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.val_acc_label.setObjectName("val_acc_label")
        self.statistic_layout.addWidget(self.val_acc_label, 2, 0, 1, 1)
        self.val_loss_label = QtWidgets.QLabel(self.main)
        self.val_loss_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.val_loss_label.setAlignment(
            QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.val_loss_label.setObjectName("val_loss_label")
        self.statistic_layout.addWidget(self.val_loss_label, 3, 0, 1, 1)
        self.train_acc_label = QtWidgets.QLabel(self.main)
        self.train_acc_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.train_acc_label.setAlignment(
            QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)

```

```

        self.train_acc_label.setObjectName("train_acc_label")
        self.statistic_layout.addWidget(self.train_acc_label, 4, 0, 1, 1)
        self.test_loss_label = QtWidgets.QLabel(self.main)
        self.test_loss_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.test_loss_label.setAlignment(
            QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.test_loss_label.setObjectName("test_loss_label")
        self.statistic_layout.addWidget(self.test_loss_label, 1, 0, 1, 1)
        self.params_label = QtWidgets.QLabel(self.main)
        self.params_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.params_label.setAlignment(
            QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.params_label.setObjectName("params_label")
        self.statistic_layout.addWidget(self.params_label, 6, 0, 1, 1)
        self.test_acc_value = QtWidgets.QLabel(self.main)
        self.test_acc_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.test_acc_value.setText("")
        self.test_acc_value.setObjectName("test_acc_value")
        self.statistic_layout.addWidget(self.test_acc_value, 0, 1, 1, 1)
        self.test_loss_value = QtWidgets.QLabel(self.main)
        self.test_loss_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.test_loss_value.setText("")
        self.test_loss_value.setObjectName("test_loss_value")
        self.statistic_layout.addWidget(self.test_loss_value, 1, 1, 1, 1)
        self.val_acc_value = QtWidgets.QLabel(self.main)
        self.val_acc_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.val_acc_value.setText("")
        self.val_acc_value.setObjectName("val_acc_value")
        self.statistic_layout.addWidget(self.val_acc_value, 2, 1, 1, 1)
        self.val_loss_value = QtWidgets.QLabel(self.main)
        self.val_loss_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.val_loss_value.setText("")
        self.val_loss_value.setObjectName("val_loss_value")
        self.statistic_layout.addWidget(self.val_loss_value, 3, 1, 1, 1)
        self.train_acc_value = QtWidgets.QLabel(self.main)
        self.train_acc_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
        self.train_acc_value.setText("")
        self.train_acc_value.setObjectName("train_acc_value")
        self.statistic_layout.addWidget(self.train_acc_value, 4, 1, 1, 1)
        self.train_loss_value = QtWidgets.QLabel(self.main)
        self.train_loss_value.setStyleSheet("QLabel {\n"

```

```

        "     font-family: \'Consolas\';\n"
        "     font-size: 10pt;\n"
    "}")
        self.train_loss_value.setText("")
        self.train_loss_value.setObjectName("train_loss_value")
        self.statistic_layout.addWidget(self.train_loss_value, 5, 1, 1, 1)
        self.params_value = QtWidgets.QLabel(self.main)
        self.params_value.setStyleSheet("QLabel {\n"
        "     font-family: \'Consolas\';\n"
        "     font-size: 10pt;\n"
    "}")
        self.params_value.setText("")
        self.params_value.setObjectName("params_value")
        self.statistic_layout.addWidget(self.params_value, 6, 1, 1, 1)
        self.main_layout.addLayout(self.statistic_layout)
        self.matplotlib_accuracy_plot = MatplotlibWidget(self.main)
        self.matplotlib_accuracy_plot.setMinimumSize(QtCore.QSize(0, 200))
        self.matplotlib_accuracy_plot.setMaximumSize(QtCore.QSize(16777215, 200))
        self.matplotlib_accuracy_plot.setStyleSheet("margin-top: 10px;\n"
"margin-bottom: 10px;")
        self.matplotlib_accuracy_plot.setObjectName("matplotlib_accuracy_plot")
        self.main_layout.addWidget(self.matplotlib_accuracy_plot)
        self.matplotlib_loss_plot = MatplotlibWidget(self.main)
        self.matplotlib_loss_plot.setMinimumSize(QtCore.QSize(0, 200))
        self.matplotlib_loss_plot.setMaximumSize(QtCore.QSize(16777215, 200))
        self.matplotlib_loss_plot.setStyleSheet("margin-top: 10px;\n"
"margin-bottom: 10px;")
        self.matplotlib_loss_plot.setObjectName("matplotlib_loss_plot")
        self.buttons_layout = QtWidgets.QHBoxLayout()
        self.buttons_layout.setObjectName("buttons_layout")
        self.train_b = QtWidgets.QPushButton(self.main)
        self.train_b.setStyleSheet("QPushButton {\n"
        "     color: #f7f7f7;\n"
        "     background-color: #4c5b61;\n"
        "     border-radius: 10px;\n"
        "     margin: 10px 10px 0px 10px;\n"
        "     height: 32px;\n"
        "     font: 10pt \'Consolas\';\n"
    "}\n"
    "\n"
    "QPushButton:pressed {\n"
    "     background-color: #393e46;\n"
    "     font: bold;\n"
    "}\n"
    "\n"
    "QPushButton:hover:!pressed {\n"
    "     background-color: #424b53;\n"
    "}\n"
    ""))
        self.train_b.setObjectName("train_b")
        self.buttons_layout.addWidget(self.train_b)
        self.retrain_b = QtWidgets.QPushButton(self.main)
        self.retrain_b.setStyleSheet("QPushButton {\n"
        "     color: #f7f7f7;\n"
        "     background-color: #4c5b61;\n"
        "     border-radius: 10px;\n"
        "     margin: 10px 10px 0px 10px;\n"
        "     height: 32px;\n"
        "     font: 10pt \'Consolas\';\n"
    "}\n"
    "\n"

```

```

"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"
"\n"
"QPushButton:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}\n"
"")

        self.retrain_b.setObjectName("retrain_b")
        self.buttons_layout.addWidget(self.retrain_b)
        self.stop_b = QtWidgets.QPushButton(self.main)
        self.stop_b.setStyleSheet("QPushButton {\n"
"    color: #f7f7f7;\n"
"    background-color: #4c5b61;\n"
"    border-radius: 10px;\n"
"    margin: 10px 10px 0px 10px;\n"
"    height: 32px;\n"
"    font: 10pt \\"Consolas\\';\n"
"}\n"
"\n"
"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"
"\n"
"QPushButton:hover:!pressed {\n"
"    background-color: #424b53;\n"
"}\n"
"")

        self.stop_b.setObjectName("stop_b")
        self.buttons_layout.addWidget(self.stop_b)
        self.main_layout.addLayout(self.buttons_layout)
        mainWindow.setCentralWidget(self.main)

        self.retranslateUi(mainWindow)
        QtCore.QMetaObject.connectSlotsByName(mainWindow)

def retranslateUi(self, mainWindow):
    _translate = QtCore.QCoreApplication.translate
    mainWindow.setWindowTitle(_translate("mainWindow",
        "MSDD | "))
    self.model_name_value.setText(_translate("mainWindow",
        "Назва моделі"))
    self.train_loss_label.setText(_translate("mainWindow",
        "Витрати на навчальному наборі: "))
    self.test_acc_label.setText(_translate("mainWindow",
        "Точність на тестовому наборі: "))
    self.val_acc_label.setText(_translate("mainWindow",
        "Точність на валідаційному наборі: "))
    self.val_loss_label.setText(_translate("mainWindow",
        "Витрати на валідаційному наборі: "))
    self.train_acc_label.setText(_translate("mainWindow",
        "Точність на навчальному наборі: "))
    self.test_loss_label.setText(_translate("mainWindow",
        "Витрати на тестовому наборі: "))
    self.params_label.setText(_translate("mainWindow",
        "Кількість параметрів: "))
    self.train_b.setText(_translate("mainWindow", "Донаучити"))
    self.retrain_b.setText(_translate("mainWindow", "Навчити"))
    self.stop_b.setText(_translate("mainWindow",
        "Зупинити навчання"))

```

A.21 Файл ModelListWindow.py

```

from PyQt5 import QtWidgets, QtCore
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QMainWindow
from PyQt5.QtCore import pyqtSlot, Qt, QSize

from Settings.DefaultSettings import DefaultSettings
from UI.modelsListWindowDesign import Ui_mainWindow
from UI.ModelInfoWindow import ModelInfoWindow
from UI.MatplotlibWidget import MatplotlibWidget

from Model.ModelsKeeper import ModelsKeeper

class ModelListWindow(QMainWindow, Ui_mainWindow):

    def __init__(self):
        super(ModelListWindow, self).__init__()
        self.setupUi(self)

        icon = QIcon()
        icon.addFile(DefaultSettings.RESOURCES_PATH + "icon.png", QSize(),
                     QIcon.Normal, QIcon.Off)
        self.setWindowIcon(icon)

        self.update_scroll_area_content()

    def update_scroll_area_content(self):
        scroll_bar_value = self.scroll_area.verticalScrollBar().value()

        self.scroll_area_widget_content = QtWidgets.QWidget()
        self.scroll_area_widget_content.setGeometry(QtCore.QRect(0, 0, 850, 625))
        self.scroll_area_widget_content.setObjectName("scroll_area_widget_content")

        self.scroll_area_widget_content_layout = QtWidgets.QVBoxLayout(
            self.scroll_area_widget_content)
        self.scroll_area_widget_content_layout.setContentsMargins(20, 0, 20, 20)
        self.scroll_area_widget_content_layout.setObjectName(
            "scroll_area_widget_content_layout")

        for name, model in ModelsKeeper().get_models().items():
            q_group_box = QtWidgets.QGroupBox(self.scroll_area_widget_content)
            q_group_box.setTitle(name)
            q_group_box.setStyleSheet("QGroupBox {\n"
                                     "border: 1px solid #4c5b61;\n"
                                     "border-radius: 5px;\n"
                                     "margin-top: 20px;\n"
                                     "height: 250px;\n"
                                     "font: 12pt \\'Consolas\\';\n"
                                     "font-weight: bold;\n"
                                     "}\n"
                                     "\n"
                                     "QGroupBox::title {\n"
                                     "subcontrol-origin: margin;\n"
                                     "left: 10px;\n"
                                     "padding: 10px 3px 0 3px;\n"
                                     "}")
            q_group_box_layout = QtWidgets.QGridLayout(q_group_box)

```

```

q_group_box_layout.setContentsMargins(10, 10, 10, 10)

q_group_box_info = QtWidgets.QFormLayout()
q_group_box_info.setLabelAlignment(QtCore.Qt.AlignRight |
    QtCore.Qt.AlignTrailing | QtCore.Qt.AlignVCenter)
q_group_box_info.setFormAlignment(QtCore.Qt.AlignLeading |
    QtCore.Qt.AlignLeft | QtCore.Qt.AlignVCenter)

accuracy_label = QtWidgets.QLabel(q_group_box)
accuracy_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
q_group_box_info.setWidget(0, QtWidgets.QFormLayout.LabelRole,
                           accuracy_label)

accuracy_value = QtWidgets.QLabel(q_group_box)
accuracy_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
q_group_box_info.setWidget(0, QtWidgets.QFormLayout.FieldRole,
                           accuracy_value)

loss_label = QtWidgets.QLabel(q_group_box)
loss_label.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
q_group_box_info.setWidget(1, QtWidgets.QFormLayout.LabelRole,
                           loss_label)

loss_value = QtWidgets.QLabel(q_group_box)
loss_value.setStyleSheet("QLabel {\n"
"    font-family: \\'Consolas\\';\n"
"    font-size: 10pt;\n"
"}")
q_group_box_info.setWidget(1, QtWidgets.QFormLayout.FieldRole,
                           loss_value)

q_group_box_layout.addLayout(q_group_box_info, 0, 0, 1, 1)

matplotlib_plot = MatplotlibWidget(q_group_box)

q_group_box_layout.addWidget(matplotlib_plot, 0, 1, 1, 3)

info_b = QtWidgets.QPushButton(q_group_box)
info_b.setStyleSheet("QPushButton {\n"
"    color: #f7f7f7;\n"
"    background-color: #4c5b61;\n"
"    border-radius: 5px;\n"
"    margin: 5px;\n"
"    height: 20px;\n"
"    font: 10pt \\'Consolas\\';\n"
"}\n"
"\n"
"QPushButton:pressed {\n"
"    background-color: #393e46;\n"
"    font: bold;\n"
"}\n"
"\n"
"QPushButton:hover:!pressed\n"
")
```

```

"{}\\n"
"    background-color: #424b53;\\n"
"}")
q_group_box_layout.addWidget(info_b, 1, 3, 1, 1)

self.scroll_area_widget_content_layout.addWidget(q_group_box)

info_b.setText("Детальніше")
accuracy_label.setText("Точність:")
loss_label.setText("Витрати:")

if model.is_saved_fitted_model():
    accuracy_value.setText(f'{model.evaluateResults["accuracy"]:.2f}')
    loss_value.setText(f'{model.evaluateResults["loss"]:.2f}')

    matplotlib_plot.make_plot(range(1,
                                    len(model.fitHistory["accuracy"]) + 1),
                               {"навчання": model.fitHistory["accuracy"],
                                "валідація": model.fitHistory["val_accuracy"]},
                               x_label="епоха", title="Точність")
else:
    accuracy_value.setText("")
    loss_value.setText("")

matplotlib_plot.make_plot([], {
    "навчання": [],
    "валідація": []},
    x_label="епоха", title="Точність")

info_b.clicked.connect(lambda state, x=name: self.on_info_b_click(x))

spacer_item = QtWidgets.QSpacerItem(20, 210, QtWidgets.QSizePolicy.Minimum,
                                   QtWidgets.QSizePolicy.Expanding)
self.scroll_area_widget_content_layout.addItem(spacer_item)

self.scroll_area.setWidget(self.scroll_area_widget_content)

self.scroll_area.verticalScrollBar().setValue(scroll_bar_value)

@pyqtSlot()
def on_info_b_click(self, name):
    self.modelInfoWindow = ModelInfoWindow(self, name)
    self.modelInfoWindow.setWindowModality(Qt.ApplicationModal)
    self.modelInfoWindow.show()

```

A.22 Файл modelsListWindowDesign.py

```

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_mainWindow(object):
    def setupUi(self, mainWindow):
        mainWindow.setObjectName("mainWindow")
        mainWindow.resize(850, 625)
        mainWindow.setStyleSheet("background-color: #eaeffa;")
        self.main = QtWidgets.QWidget(mainWindow)
        self.main.setObjectName("main")
        self.main_layout = QtWidgets.QVBoxLayout(self.main)
        self.main_layout.setContentsMargins(0, 0, 0, 0)

```

```
self.main_layout.setObjectName("main_layout")
self.scroll_area = QtWidgets.QScrollArea(self.main)
self.scroll_area.setFrameShape(QtWidgets.QFrame.NoFrame)
self.scroll_area.setWidgetResizable(True)
self.scroll_area.setObjectName("scroll_area")
self.scroll_area_widget_content = QtWidgets.QWidget()
self.scroll_area_widget_content.setGeometry(QtCore.QRect(0, 0, 850, 625))
self.scroll_area_widget_content.setObjectName("scroll_area_widget_content")
self.scroll_area_widget_content_layout = QtWidgets.QVBoxLayout(
    self.scroll_area_widget_content)
self.scroll_area_widget_content_layout.setContentsMargins(20, 0, 20, 20)
self.scroll_area_widget_content_layout.setObjectName(
    "scroll_area_widget_content_layout")
self.scroll_area.setWidget(self.scroll_area_widget_content)
self.main_layout.addWidget(self.scroll_area)
mainWindow.setCentralWidget(self.main)

self.retranslateUi(mainWindow)
QtCore.QMetaObject.connectSlotsByName(mainWindow)

def retranslateUi(self, mainWindow):
    _translate = QtCore.QCoreApplication.translate
    mainWindow.setWindowTitle(_translate("mainWindow", "MSDD | Навчання"))
```

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ



Рисунок Б.1 – Слайд презентації № 1

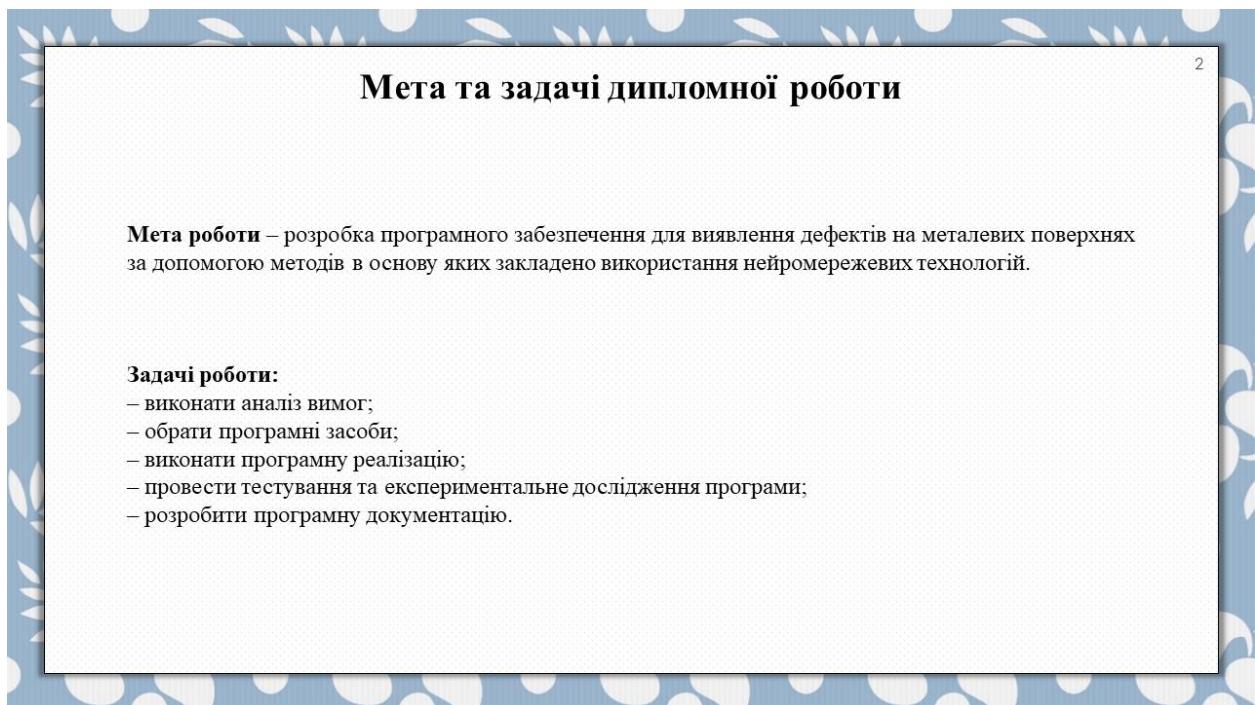


Рисунок Б.2 – Слайд презентації № 2

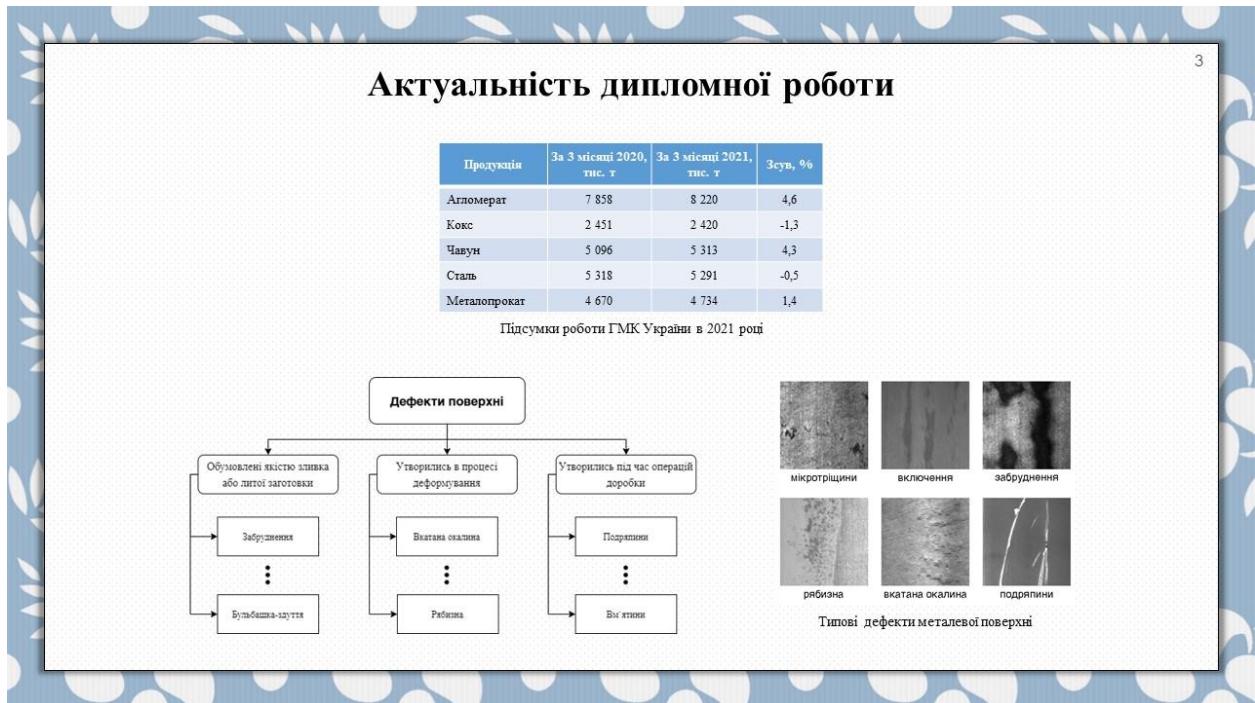
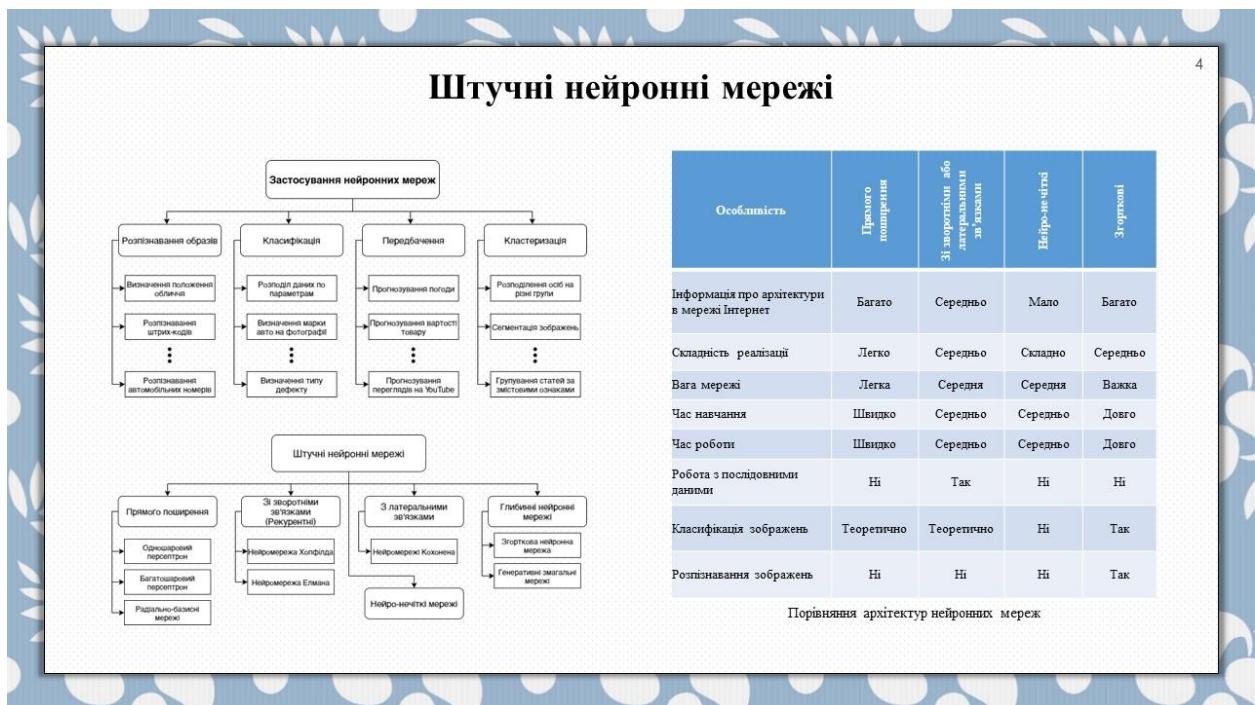


Рисунок Б.3 – Слайд презентації № 3



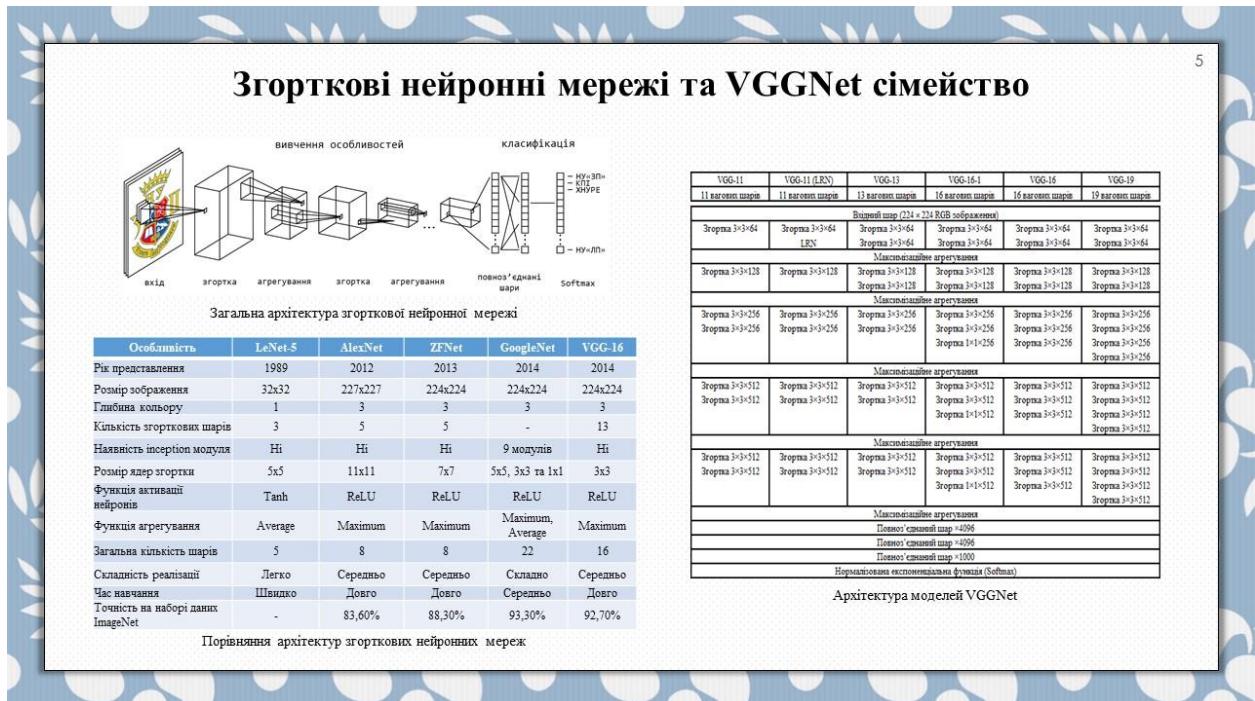


Рисунок Б.5 – Слайд презентації № 5

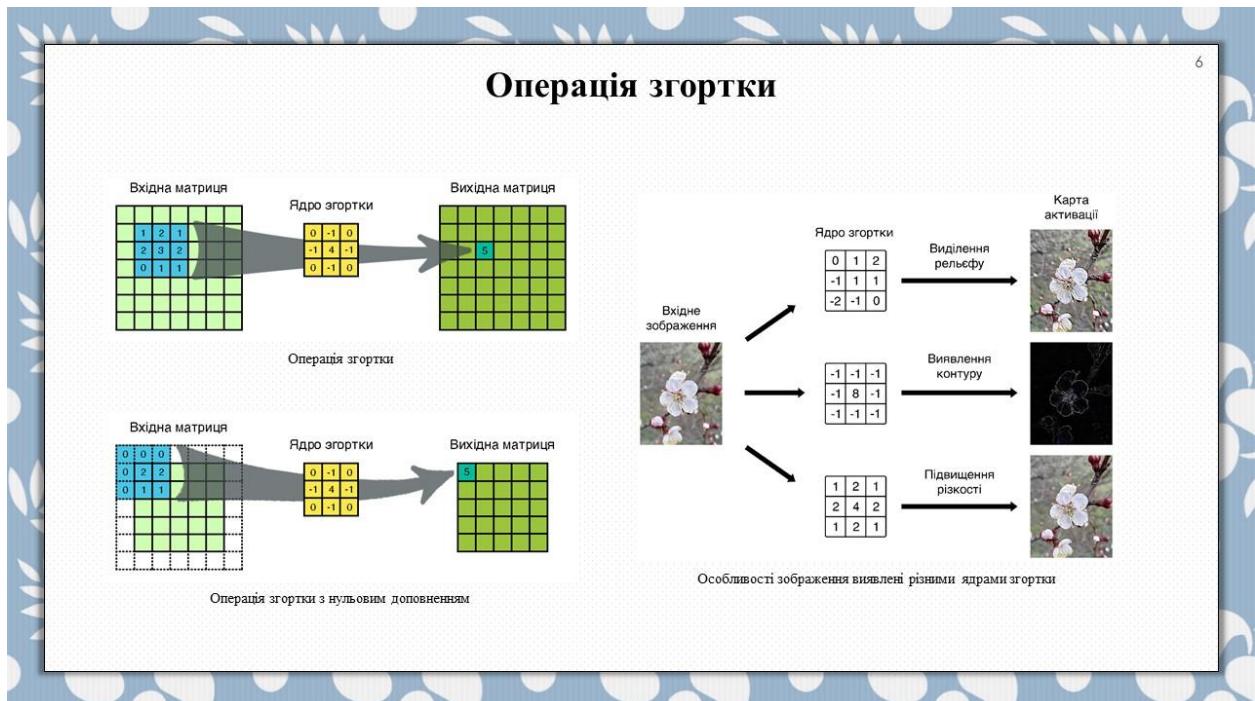


Рисунок Б.6 – Слайд презентації № 6

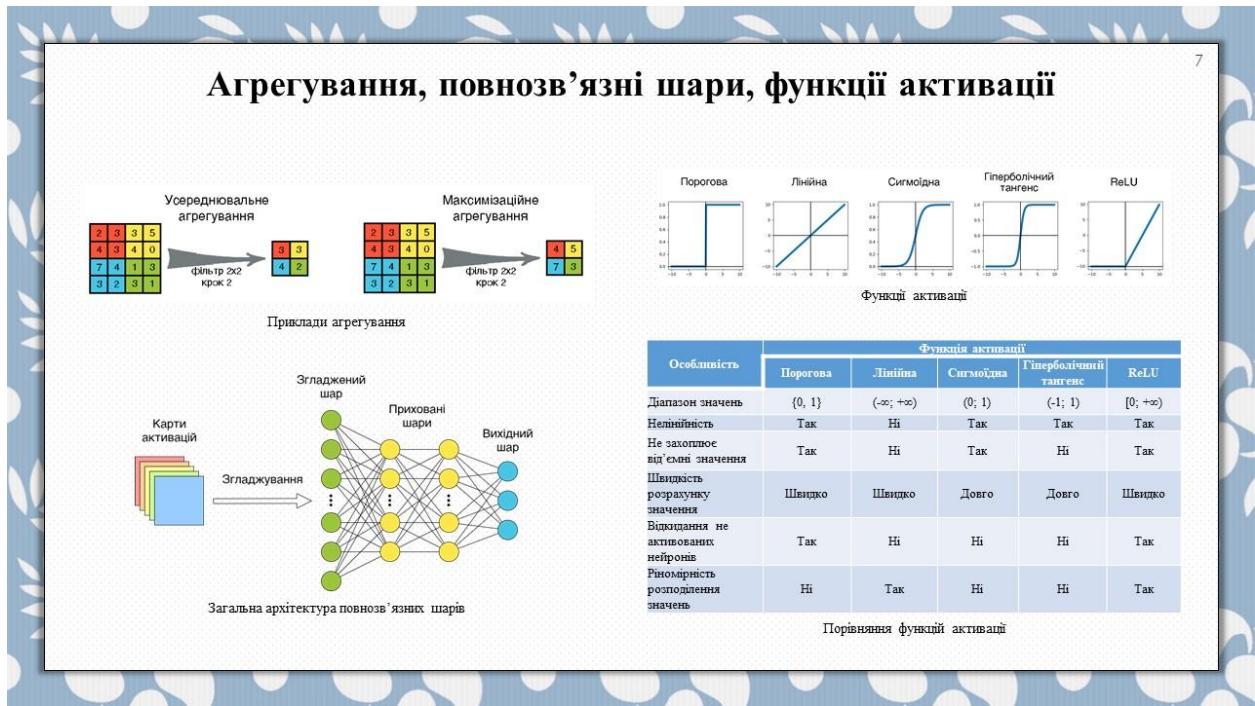


Рисунок Б.7 – Слайд презентації № 7



Рисунок Б.8 – Слайд презентації № 8



Рисунок Б.9 – Слайд презентації № 9



Рисунок Б.10 – Слайд презентації № 10

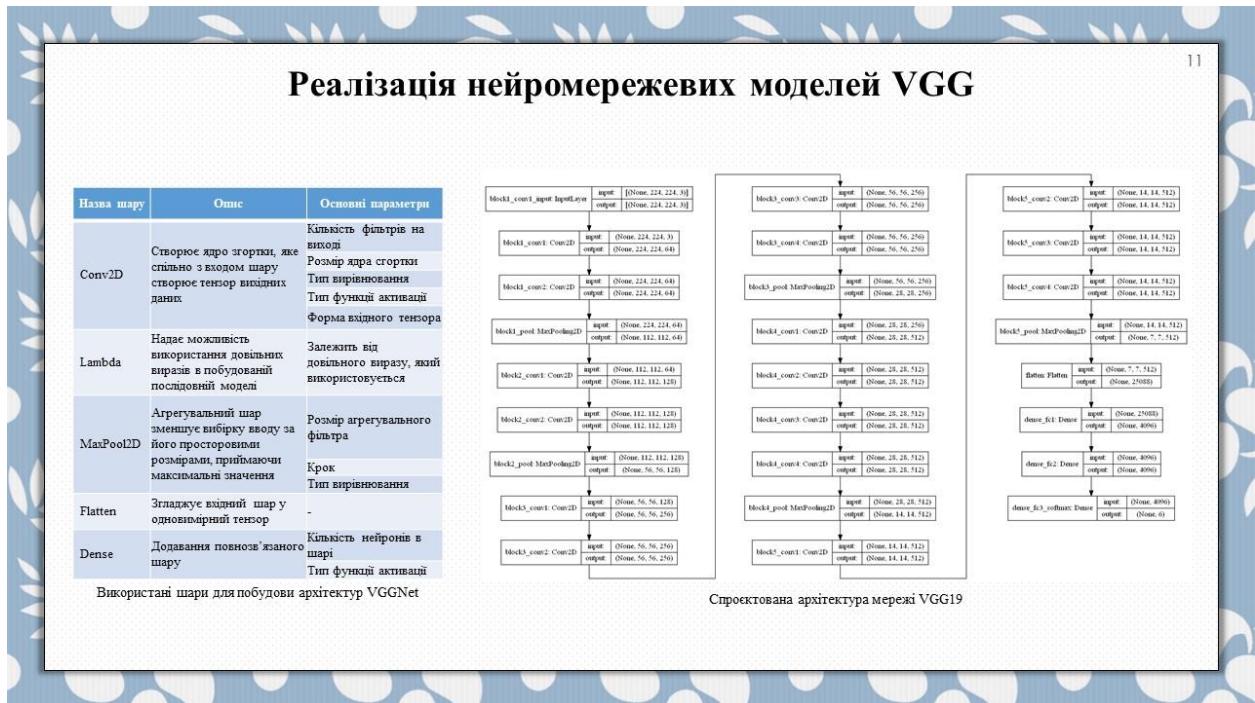


Рисунок Б.11 – Слайд презентації № 11

12

Тестування

Вимога	Статус
Відкриття вікон програми	Прайдено
Відкриття головного вікна програми	Прайдено
Відкриття вікна з списком моделей	Прайдено
Відкриття вікна з інформацією про модель	Прайдено
Відкриття вікна класифікації	Прайдено
Відкриття вікна з інформацією про програму	Прайдено
Можливості головного вікна програми	Прайдено
Реакція графічних елементів на дії користувача	Прайдено
Відображення назви програми	Прайдено
Відображення логотипа програми	Прайдено
Фіксований розмір вікна	Прайдено
Можливості вікна зі списком моделей	Прайдено
Можливість гортання списку моделей	Прайдено
Коректне відображення назв неромережевих моделей	Прайдено
Відображення графіка точності для кожної з моделей	Прайдено
Реакція графічних елементів на дії користувача	Прайдено
Розмір вікна може змінюватися	Прайдено
Можливості вікна з інформацією про модель	Прайдено
Реакція графічних елементів на дії користувача	Прайдено
Можливість навчання моделі	Прайдено
Можливість зупинки навчання моделі	Прайдено
Коректне відображення назви неромережевої моделі	Прайдено
Відображення графіка точності та виграт	Прайдено
Можливості вікна класифікації	Прайдено
Вибр та завантаження зображення	Прайдено
Коректне відображення зображення	Прайдено
Можливість вибору моделі зі списку	Прайдено
Інше	
Найвиї всі моделі сімейства VGGNet	Прайдено
Читабельний розмір пріорітетів графічних елементів	Прайдено
Основні елементи керування підписано українською мовою	Прайдено
Висока швидкість відповіді на дії користувача	Прайдено
Наявність повідомлення користувачу під час виходу з програми	Прайдено

Контрольний список тестування

Рисунок Б.12 – Слайд презентації № 12

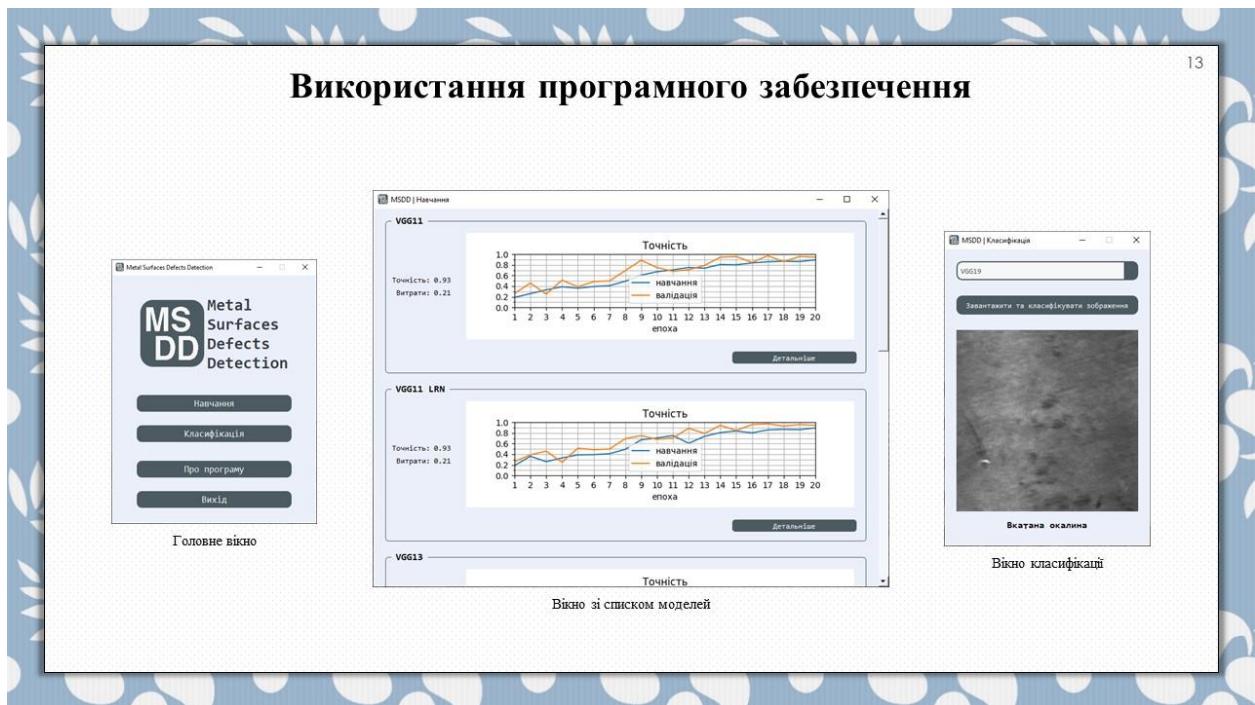


Рисунок Б.13 – Слайд презентації № 13



Рисунок Б.14 – Слайд презентації № 14



Рисунок Б.15 – Слайд презентації № 15

16

Експериментальне дослідження програми

Параметр	VGG11	VGG11 LRN	VGG13	VGG16	VGG16 1	VGG19
Точність на навчальному наборі даних, %	89,49	89,49	95,41	97,95	97,64	98,55
Точність на валідаційному наборі даних, %	94,44	94,44	95,83	100	98,61	100
Точність на тестовому наборі даних, %	93,06	93,06	94,44	98,61	97,22	100
Витрати на навчальному наборі даних	0.2798	0.2798	0.1732	0.0598	0.0571	0.0500
Витрати на валідаційному наборі даних	0.1788	0.1788	0.1562	0.0231	0.0042	0.0027
Витрати на тестовому наборі даних	0.2058	0.2058	0.1656	0.0719	0.0120	0.0141
Кількість згорткових шарів	8	8	10	13	13	16
Загальна кількість шарів	11	11	13	16	16	19
Кількість параметрів	128790918	128790918	128975430	134285126	129566534	139594822
Розмір файлу з вагами, КБ	503133	503134	503858	524610	506178	545359
Розмір файлу з моделлю загалом, КБ	1509367	1509382	1511553	1458790	1518504	1479542
Загальний час навчання, год	4 год 19 хв	4 год 41 хв	6 год 45 хв	12 год 17 хв	9 год 20 хв	13 год 53 хв
Можливість використання вагів ядер згортки мережі натренованої на БД ImageNet	Ні	Ні	Ні	Так	Ні	Так
Загальний час навчання з використанням натренованих вагів ядер згортки, год	-	-	-	3 год 10 хв	-	3 год 50 хв
Кількість епох	20	20	20	20	20	20
Епоха на якій перевищено 75% точності на навчальному наборі	12	11	10	2	8	2

Результати навчання та використання моделей

Рисунок Б.16 – Слайд презентації № 16

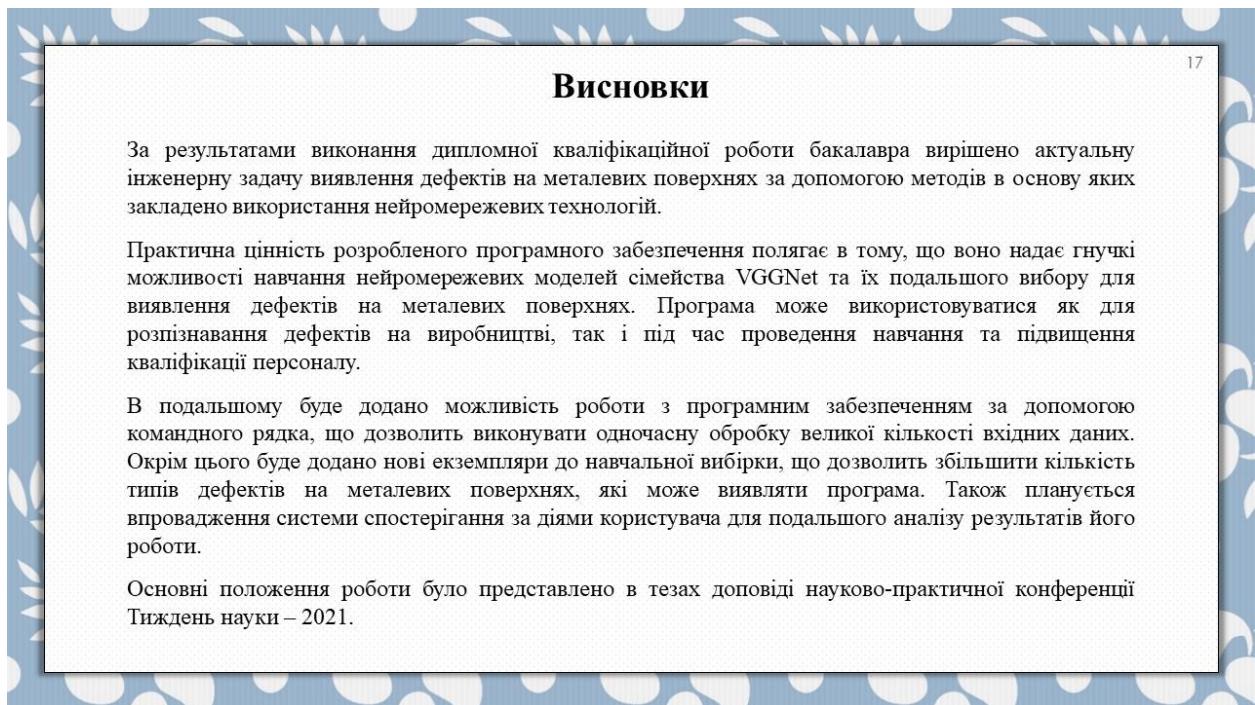


Рисунок Б.17 – Слайд презентації № 17

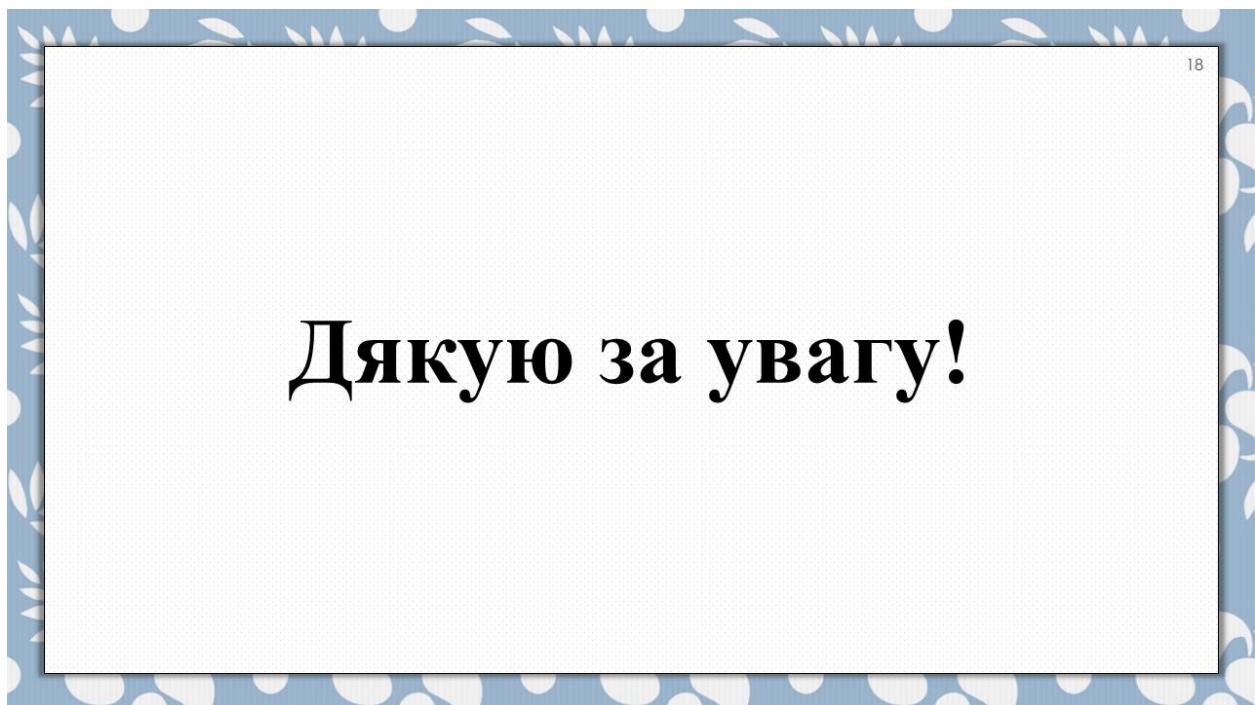


Рисунок Б.18 – Слайд презентації № 18