МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ПАЛЬЧИК Вікторія Олександрівна

Метод класифікації дерев’яних виробів з використанням штучного інтелекту / The Method of Wooden Products Classification Using Artificial Intelligence

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп’ютерні науки

Кваліфікаційна робота

Виконав студент групи
КНм-21
В.О. Пальчик
_____

Науковий керівник:
д.т.н., доцент В.С. Коваль
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
Завідувач кафедри
_____ М.П. Комар

ТЕРНОПІЛЬ - 2022

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «магістр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ М.П. Комар
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Пальчик Вікторія Олександрівна
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Метод класифікації дерев'яних виробів з використанням штучного інтелекту / The Method of Wooden Products Classification Using Artificial Intelligence
керівник роботи д.т.н., доцент В.С. Коваль
затверджені наказом по університету від 31 грудня 2021 року № 606.

2. Строк подання студентом закінченої кваліфікаційної роботи 16 листопада 2022 року.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити
- дослідити існуючі рішення задачі класифікації деревини;
- дослідити теоретичні основи нейронних мереж;
- дослідити теоретичні основи багатошарових нейронних мереж;
- дослідити теоретичні основи згорткових нейронних мереж;
- провести аналіз існуючих засобів реалізації і обрати засоби розробки;
- запропонувати спосіб класифікації дерев'яних виробів на основі нейронних мереж;
- провести огляд архітектур штучних нейронних мереж для запропонованого методу провести порівняльний аналіз;
- розробити архітектуру нейромережі для класифікації дерев'яних виробів;
- провести експериментальні дослідження запропонованого методу.

5. Перелік графічного матеріалу у роботі
- алгоритм роботи методу класифікації дерев'яних виробів;
- Схема роботи методу класифікації дерев'яних виробів
- Архітектура UNET для класифікації дерев'яних виробів.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 11 жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз існуючих рішень класифікації виробів	10.2021 р. – 03.2022 р.	
2	Розроблення методу розпізнавання дефектів та класифікації дерев’яних зображень	03.2022 р. – 05.2022 р.	
3	Програмна реалізація	05.2022 р. – 11.2022 р.	
4	Повне завершення та представлення кваліфікаційної роботи на кафедру	05.12.2022 р.	

Студент _____ В.О. Пальчик
підпис

Керівник роботи _____ д.т.н., доцент В.С. Коваль
підпис

РЕЗЮМЕ

Кваліфікаційна робота на тему «Метод класифікації дерев’яних виробів з використанням штучного інтелекту» на здобуття освітнього ступеня «Магістр» зі спеціальності 122 «Комп’ютерні науки» освітньої програми «Комп’ютерні науки» написана обсягом в 88 сторінки і містить 36 ілюстрації, 1 таблицю, 4 додатки та 55 використаних джерел.

Метою даної кваліфікаційної роботи є розробка методу класифікації дерев’яних виробів з використанням штучного інтелекту.

Методи досліджень: аналіз літературних джерел та узагальнення інформації, огляд існуючих рішень класифікації дерев’яних дефектів, аналіз нейронних мереж, методи розробки нейромереж для задач семантичної сегментації, методи машинного навчання, проведення та аналіз результатів експериментальних досліджень.

Результати дослідження: Розроблено нейронну мережу яка виконує класифікацію дефектів дерев’яних виробів, та задачі семантичної сегментації, що дозволяє чітко виявити межі дефектів на дерев’яних виробах. Проведено експериментальні дослідження оцінки ефективності та якості запропонованого методу.

Отримані результати дозволять розробити програмне забезпечення для дефектування дерев’яних виробів.

Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, UNET НЕЙРОМЕРЕЖА, КЛАСИФІКАЦІЯ ДЕРЕВ’ЯНИХ ВИРОБІВ, НЕЙРОННА МЕРЕЖА, СЕМАНТИЧНА СЕГМЕНТАЦІЯ.

ABSTRACT

The qualification work on the topic "The method of classifying wooden products using artificial intelligence" for obtaining the Master's degree in the specialty 122 "Computer Science" of the educational program "Computer Science" is written in the volume of 88 pages and contains 36 illustrations, 1 table, 4 appendices and 55 references.

The purpose of this qualification work is to develop a method of classifying wooden products using artificial intelligence.

Research methods: analysis of literary sources and generalization of information, review of existing solutions for the classification of wood defects, analysis of neural networks, methods of developing neural networks for semantic segmentation tasks, machine learning methods, conducting and analyzing the results of experimental studies.

Research results: A neural network was developed that performs the classification of defects of wooden products and tasks of semantic segmentation, which allows clearly identifying the boundaries of defects on wooden products. Experimental studies were conducted to assess the effectiveness and quality of the proposed method.

The obtained results will make it possible to develop software for defecting wooden products.

Keywords: ARTIFICIAL INTELLIGENCE, UNET NEURAL NETWORK, CLASSIFICATION OF WOODEN PRODUCTS, NEURAL NETWORK, SEMANTIC SEGMENTATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN – згорткова нейронна мережа

FCNN – повністю згорткова нейронна мережа

R-CNN - Семантична сегментація на основі регіонів

ШІ – Штучний інтелект

ШНМ – Штучні нейронні мережі

SVM - керований алгоритм навчання, класифікований за методами класифікації.

ASIC - спеціалізована інтегральна схема

GPU – Графічний процесор

CPU – Центральний процесор

TPU – Тензорний професор

ВСТУП

Актуальність теми. Паралельно з індустріальним розвитком, цифрові технології, все більше проникають в науку, техніку і виробництво. Технічний прогрес вимагає вдосконалення алгоритмів, методів і способів опрацювання даних на виробництві.

Досягнення в області комп'ютерного зору та машинного навчання змінили методології багатьох наукових дисциплін. Вони також створили нову дослідницьку сферу в галузі науки про деревину під назвою ідентифікація деревини на основі комп'ютерного зору, яка постійно просувається до мети побудови автоматизованих систем ідентифікації деревини для задоволення потреб деревообробної промисловості та ринку. Ідентифікація деревини на основі комп'ютерного зору все ще є актуальною в науці про деревину і досі незнайома багатьом анатомам деревини.

Ця галузь на основі комп'ютерного зору постійно вдосконалюється шляхом побудови автоматизованих систем ідентифікації, що задовольняє потреби деревообробної промисловості. Водночас, ідентифікація деревини проводиться групою методів, які на основі комп'ютерного зору дозволяють вивчати властивості деревини. Паралельно з індустріальним розвитком, цифрові технології, все більше проникають в науку, техніку і виробництво. Технічний прогрес вимагає вдосконалення алгоритмів, методів і способів опрацювання даних на виробництві. Враховуючи що, в даний момент, штучні нейроні мережі набули великої популярності через їх ефективність, стає доцільним використання штучний нейроних мереж у вирішенні завдань, таких як, класифікація дерев'яних виробів на виробництві. Їх використання дозволить вирішити питання якості продукції, автоматизувати і покращити ефективніть роботи та використання ресурсів виробництва.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розробка методу класифікації дерев'яних виробів з використанням штучного інтелекту.

Для досягнення поставленої мети неохідно виконати наступні завдання:

- дослідити існуючі рішення задачі класифікації деревини;
- дослідити теоретичні основи нейронних мереж;
- дослідити теоретичні основи багатошарових нейронних мереж;
- дослідити теоретичні основи згорткових нейронних мереж;
- провести аналіз існуючих засобів реалізації і обрати засоби розробки;
- запропонувати спосіб класифікації дерев’яних виробів на основі нейронних мереж;
- провести огляд архітектур штучних нейронних мереж для запропонованого методу;
- дослідити існуючі функції втрат та метрики для нейронних мереж;
- провести порівняльний аналіз архітектур FCN I UNET ;
- розробити алгоритм роботи методу класифікації дерев'яних виробів;
- розробити структурні схеми роботи методу класифікації дерев'яних виробів;
- розробити архітектуру нейромережі для класифікації дерев'яних виробів;
- провести експерементальні дослідження запропонованого методу.

Об’єкт дослідження – процеси класифікації дерев'яних дефектів.

Предмет дослідження - метод класифікації дерев’яних виробів з використанням штучного інтелекту.

Методи дослідження. Для вирішення поставлених задач в роботі використовувалися наступні методи:

- аналіз літературних джерел та узагальнення інформації;
- огляд існуючих рішень класифікації дерев’яних дефектів;
- аналіз нейронних мереж;

- методи розробки нейромереж для задач семантичної сегментації;
- методи машинного навчання;
- проведення та аналіз результатів експериментальних досліджень.

Наукова новизна одержаних результатів. Розроблено метод дефектування дерев’яних виробів, який на основі використання UNET архітектури нейромережі та функцій втрат дозволяє більш точно класифіккувати дефекти у порівнянні з відомими рішеннями. На відміну від відомих рішень, цей метод може виконувати обробку зображень на малих датасетах і за допомогою комбінацій функцій, що нівелюють погано збалансований набір вхідних даних, досягти хороших результатів навчання.

Практичне значення отриманих результатів. Розроблено нейронну мережу яка виконує класифікацію дефектів дерев’яних виробів, та задачі семантичної сегментації, що дозволяє чітко виявити межі дефектів на дерев’яних виробах. Проведено експерементальні дослідження оцінки ефективності та якості запропонованого методу. Отримані результати дозволять розробити програмне забезпечення для дефектування дерев’яних виробів.

Публікації та апробація. Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах:

Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі». 2022.

Школа-семінар молодих вчених і студентів "Комп’ютерні інформаційні технології (CIT’2022)", 2022р.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ КЛАСИФІКАЦІЇ ВИРОБІВ

1.1 Огляд існуючих підходів рішення задачі

Швидкий метод виявлення дефектів деревини необхідний сучасній промисловості з обробки деревини, щоб покращити рівень використання деревини та збільшити дохід.

В даний час для виявлення дефектів на дерев'яному шпоні використовується багато видів технологій, включаючи ультразвукову технологію з повітряним зв'язком [3], технологію хвиль напруги [4], 3D лазерну технологію [5], комп'ютерну томографію [6] і технологію комп'ютерного зору. [7].

Ультразвукова технологія з повітряним зв'язком — це безконтактний ультразвуковий метод вимірювання, який виявляє дефекти всередині дерев'яного шпону на основі коливань щільності дефектів усередині деревини. Однак ультразвукова технологія сприйнятлива до зовнішнього середовища і нестабільна, що призводить до неприйнятної для промислових вимог ефективності виявлення [8]. Технологія хвилі напруги може визначити розмір і форму внутрішніх дефектів на основі сегментованих променів поширення хвилі напруги. Однак для отримання сигналів хвилі напруги датчики повинні бути міцно прикріплені до поверхні деревини, а відстань між датчиками має бути фіксованою, що обмежує ефективність виявлення дефектів і робить цей метод недостатньо гнучким для промислового використання [9] .

3D-лазерну технологію можна використовувати для точного вимірювання форми дерев'яних поверхонь і визначення дефектів за допомогою технології 3D-сканера за площею та об'ємом [10]. Однак деякі дефекти не можуть бути точно виявлені 3D-лазерним сканером, якщо їх форма дуже мала.

Рентгенівське випромінювання на основі комп'ютерної томографії використовується для виділення сучків деревини на основі щільності та

вологості, так що точність найбільше залежить від вологості, що призведе до роз'єднання товстої деревини на сучки [6].

Технологія комп'ютерного зору є ефективним методом роботи з зображеннями та відео та використовується для виявлення різноманітних дефектів [11,12]. Наприклад, Boardman et al. [13] використовували колориметричні методи для виявлення дефектів шпону чорного горіха. Момін та ін. [14] використовували колірну модель насиченості тону (HIS) у поєднанні з медіанним розмиттям, морфологічними операторами та перетворенням вододілу для виявлення фракцій доку. Однак колірна модель HIS не може бути розширена для виявлення всіх дефектів деревини через різноманітність дефектів. Dawood та ін. [15] запропонували інтегровану модель, що підтримується регресійним аналізом, щоб передбачити глибину відколу. Однак продуктивність інтегрованої моделі була чутливою до вхідних даних, і тому її надійність була низькою.

З розвитком комп'ютерного обладнання, особливо високопродуктивних графічних процесорів (GPU), поєднання технології комп'ютерного зору та технології глибокого навчання має такі переваги, як висока швидкість і низька вартість, і широко використовується для виявлення дефектів [16]. Парк та ін. [17] запропоновано метод автоматичного візуального контролю дефектів на поверхні виробів з деревини на основі згорткової нейронної мережі (CNN), який був ефективним алгоритмом виявлення дефектів деревини, але не був введений у реальне застосування. Юнг та ін. [18] використовували три різні архітектури CNN для класифікації звичайної деревини та чотири типи зображень дефектів. Порівнюючи продуктивність трьох моделей CNN, глибокий CNN досяг високої точності класифікації 99,8% для виявлення дефектів, але його швидкість роботи була повільною, оскільки мережа була глибшою. Він та ін. [19] використовували змішану повністю згорнуту нейронну мережу (Mix-FCN) для автоматичного визначення та класифікації дефектів деревини за допомогою зображень поверхні деревини. Однак Mix-FCN

використовував усі обчислювальні ресурси для всієї деревини, незважаючи на те, що більшість зображень деревини були без дефектів, що було неефективним з точки зору часу. Урбонас та ін. [20] застосували попередньо навчену модель нейронної мережі ResNet152 у поєднанні з швидшим R-CNN для виявлення дефектів поверхні дерев'яних панелей і досягли точності 96,1%. Однак цей метод виводить лише рамку, що містить дефекти деревини, а не реальні форми дефектів, що вплинуло на точність розрізу дефектів.

Маска R-CNN була розроблена як покращена версія більш швидкої R-CNN, яка може виявляти дефекти та виводити форми дефектів; він широко використовується в області виявлення цілей і досягає чудової продуктивності в багатьох додатках. У масці R-CNN зображення обробляється серією шарів згортки та пропускається через три різні гілки мереж [21]. Однак небагато досліджень застосовували його для виявлення дефектів деревини. Хуан та ін. [22] повідомили про метод перевірки складання на основі моделі маски R-CNN для ідентифікації та малювання дефектів з кожної частини зображення та досягли точності класифікації понад 86,5%. Ху та ін. [23] використовували маску R-CNN у поєднанні з генеративною змагальною мережею (GAN) для ідентифікації та класифікації дефектів шпону, і точність досягла 98,4%. Незважаючи на те, що точність виявлення маски R-CNN висока, низька швидкість виявлення є її незамінним недоліком [24]. Тому необхідно розробити методику з урахуванням точності та швидкості виявлення дефектів деревини для промислового виробництва.

У цьому дослідженні перед маскою R-CNN була додана мережа огляду, щоб зменшити час роботи маски R-CNN. Мережа погляду була використана, щоб відфільтрувати всі дефектні зображення та прийняти якомога менше звичайних зображень за дефектні. Одночасно мережа погляду повинна використовувати якомога менше обчислювальних ресурсів під час класифікації. Хоча цілі мережі glance були зрозумілі, було важко знайти найбільш підходящу архітектуру та параметри для їх задоволення. Таким чином, для побудови

структури та визначення параметрів мережі погляду була використана технологія пошуку архітектури нейронної мережі (NAS).

NAS — це нова технологія, яка створює різні типи структур на основі інтелектуальних алгоритмів [25, 26, 27]. Наразі більшість архітектур нейронних мереж розробляються вручну та ретельно проектуються експертами-людьми, що є трудомістким і схильним до помилок процесом. Тому NAS було розроблено для створення нових структур для більш точної цілі [28, 29, 30]. Однак NAS використовується лише для підвищення точності нейронної мережі, ігноруючи продуктивність побудованої архітектури в реальному часі, що ускладнює адаптацію до промислового виробництва.

У цьому дослідженні було застосовано покращений метод, який поєднує технологію NAS і маску R-CNN для виявлення дефектів на поверхнях дерев'яного шпону та виведення форми та типу дефектів. Запропонований алгоритм інтегрує оптимізовану багатоканальну маску R-CNN і мережу огляду на основі NAS для отримання швидкого сканування вхідного зображення для відрізнення дефектних зображень дерев'яного шпону від усіх зображень дерев'яного шпону. На відміну від стандартної технології NAS, в мережі glance швидкість нейронної мережі розглядається за допомогою значення операцій з плаваючою комою (FLOP). Потім мережа огляду, створена NAS, витягує та виводить ознаки перших відбитків дефектних зображень на наступну багатоканальну маску R-CNN для подальшого виявлення. Щоб мати найкращу комбінацію функцій, введених у багатоканальний маску R-CNN, використовується генетичний алгоритм для оптимізації вибору функцій для досягнення кращих результатів виявлення. Запропонована інтегрована модель може забезпечити точність виявлення та кращу продуктивність у реальному часі порівняно з традиційною маскою R-CNN, у якій відсутня мережа огляду та багатоканальна структура.

Зокрема, інтегрована модель використовується як остаточний класифікатор для ідентифікації дефектів на деревині поверхні шпону, а конкретні внески цієї роботи такі:

Мережа огляду розроблена для швидкого сканування зображення, щоб визначити, чи підходить зображення для подальшого виявлення, що значно скорочує час виявлення для промислового використання. Крім того, вперше створено нову функцію відповідного значення, яка використовує FLOP мережі, щоб покращити продуктивність мережі огляду в реальному часі.

Генетичний алгоритм використовується для визначення вибору функцій багатоканальної маски вхідних каналів R-CNN, щоб досягти вищої точності виявлення.

1.2 Штучний інтелект

ШІ є однією з найновіших галузей науки та техніки. Інтенсивна робота в напрямку ШІ почалася після Другої світової війни, а сама назва була придумана в 1956 році. Разом з молекулярною біологією штучний інтелект регулярно згадується як «галузь, у якій я б найбільше хотів працювати» вченими з інших дисциплін.

В даний час ШІ охоплює величезну різноманітність підполей, починаючи від загальних (навчання та сприйняття) до конкретних, таких як гра в шахи, доведення математичних теорем, написання віршів, водіння автомобіля на людній вулиці та діагностика захворювань. ШІ актуальний для будь-якого інтелектуального завдання.

Штучний інтелект як наука включає кілька підходів і методів,утому числі:

- машинне навчання (поглиблене навчання та навчання з підкріпленням);
- машинне мислення (машинне мислення, що охоплює такі процеси, як планування, представлення знань і міркування, пошук і оптимізація);
- робототехніка (контроль, сприйняття, датчики та виконавчі механізми, а також інтеграція всіх інших технологій у кібер -фізичні системи).

1.2.1 Нейронні мережі

Математична модель нейронної мережі — це концептуальна модель біологічної нейронної мережі, що складається з різним чином пов’язаних між собою шарів штучних нейронів, які організовують загальну систему діяльності та функціонально впливають на роботу один одного. В архітектурі більшості моделей штучних нейронних мереж активність нейрона визначається шляхом перетворення зовнішнього загального сигналу інших нейронів у цей нейрон. З моменту свого створення штучні нейронні мережі еволюціонували від традиційних методів повністю індивідуальним чином, часто повністю змінюючи уявлення про теми та проблеми в машинному навчанні та теорії розпізнавання образів, з великим впливом на теорію та моделі. Методологічні підходи до цих галузей знань.

При створенні функціональної моделі біологічного нейрона важливе значення мають три основні складові. По-перше, синапси нейрона моделюються як ваги. Сила зв'язку між входом і нейроном відзначається значенням ваги. Негативні значення ваги відображають гальмівні зв'язки, а позитивні - збуджувальні. Наступні два компоненти моделюють реальну активність всередині клітини нейрона. Суматор підсумовує всі входи, модифіковані відповідними вагами. Ця активність називається лінійною комбінацією. Нарешті, функція активації контролює амплітуду виходу нейрона. Прийнятний діапазон виходу зазвичай знаходиться між 0 і 1, або -1 і 1. Математично цей процес описується на рисунку 1.1.

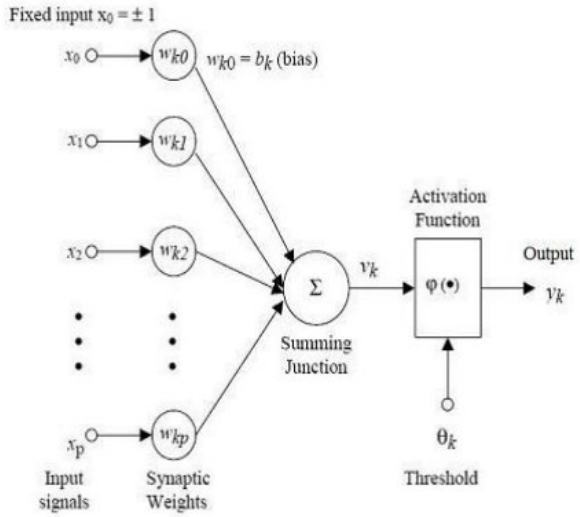


Рисунок 1.1 – Математича модель нейрона [51]

З цієї моделі інтервальну активність нейрона можна показати формулою (1.1) [31]:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j \qquad (1.1)$$

Таким чином, вихід нейрона, Yk, буде результатом деякої функції активації від значення Vk.

Якщо в нейронній мережі не використовується функція активації, то вихідний сигнал буде просто простою лінійною функцією, яка є поліномом першого ступеня. Хоча лінійне рівняння є простим і легким для вирішення, але їх складність обмежена, і вони не мають здатності навчатися і розпізнавати складні відображення з даних. Нейронна мережа без функції активації в більшості випадків діє як лінійна регресійна модель з обмеженою продуктивністю і потужністю. Бажано, щоб нейронна мережа не просто навчалася і обчислювала лінійну функцію, а виконувала більш складні

завдання, такі як моделювання складних типів даних, таких як зображення, відео, аудіо, мова, текст і т.д. Саме тому використання функції активації та методи штучних нейронних мереж, такі як глибоке навчання, має сенс для складних, багатовимірних та нелінійних наборів даних, де модель містить декілька прихованих шарів. Вибір тієї чи іншої функції активації залежить залежить від структури, набору даних і призначення нейромережі. Наразі існує широкий консенсус щодо того, які функції є корисними в різних сценаріях.

Бінарна ступінчаста функція (Binary Step Function) - це найпростіша функція активації, яка існує, і її можна реалізувати за допомогою простих операторів if-else на мові Python. При створенні бінарного класифікатора зазвичай використовуються бінарні функції активації. Але бінарна функція кроку не може бути використана у випадку багатокласової класифікації в цільовому об'єкті. Крім того, градієнт бінарної функції кроку дорівнює нулю, що може викликати перешкоду при зворотному поширенні кроку, тобто якщо ми обчислюємо похідну f(x) по відношенню до x, то вона дорівнює нулю. Математично бінарну ступінчасту функцію (1.2-1.3) можна визначити як [50]:

$$f(x) = 1, x >= 0 \quad (1.2)$$

$$f(x) = 0, x < 0 \quad (1.3)$$

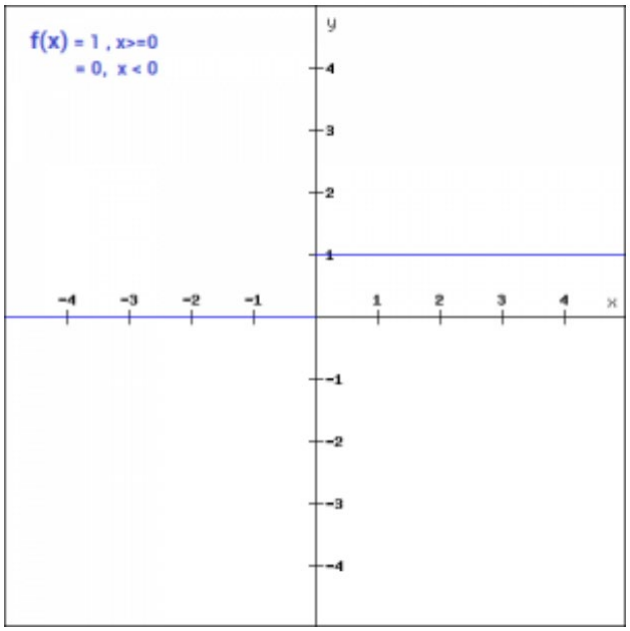


Рисунок 1.2 - Binary Step Function [50]

На рисунку 1.2 зображено графік Binary Step Function.

Лінійна активаційна функція прямо пропорційна вхідному сигналу. Основним недоліком бінарної ступеневої функції було те, що вона мала нульовий градієнт, оскільки в бінарній ступеневій функції відсутня складова x. Для того, щоб усунути цей недолік, можна використати лінійну функцію, яка може бути визначена за (1.4) [50]:

$$F(x) = ax \qquad (1.4)$$

Значенням змінної a може бути будь-яка константна величина.

Граф лінійної функції зображено на рисунку 1.3.

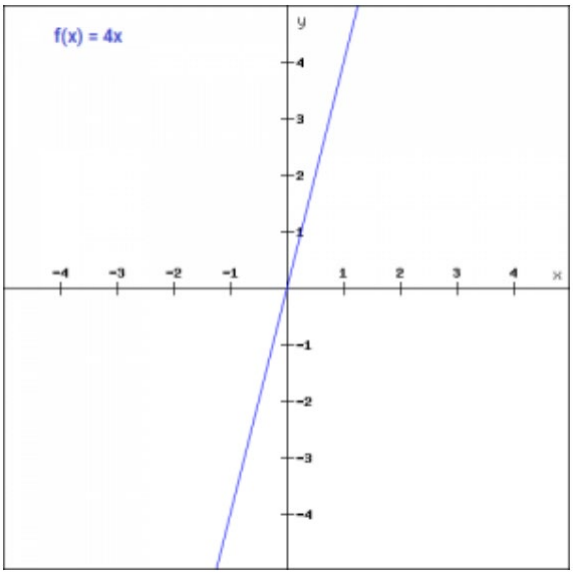


Рисунок 1.3 - Лінійна активаційна функція [50]

Тут похідна функції f(x) не дорівнює нулю, а дорівнює значенню використаної константи. Градієнт не дорівнює нулю, а є постійною величиною, яка не залежить від вхідного значення x, що означає, що ваги та зміщення будуть оновлюватися на кожному кроці регресії, хоча коефіцієнт оновлення буде однаковим. Використання лінійної функції не має великої переваги, оскільки нейронна мережа не зможе покращити похибку через однакове значення градієнта для кожної ітерації. Крім того, мережа не зможе ідентифікувати складні закономірності в даних. Тому лінійні функції ідеально підходять там, де потрібна інтерпретованість і для простих завдань.

Сигмоїдна функція активації є монотонно зростаючою, абсолютно диференційованою сигмоїдальною нелінійною функцією. Це найбільш широко використовувана функція активації, оскільки вона є нелінійною функцією. Сигмоїдна функція перетворює значення в діапазоні від 0 до 1. її можна визначити за (1.5) [50]:

$$f(x) = 1/e - x \qquad (1.5)$$

Сигмоїдна функція є неперервно диференційованою і гладкою S-подібною функцією. Похідна функції має вигляд (1.6) [50]:

$$f'(x) = 1 - sigmoid(x) \quad (1.6)$$

Також сигмоїдна функція не є симетричною відносно нуля, що означає, що знаки всіх вихідних значень нейронів будуть однаковими. Цю проблему можна вирішити шляхом масштабування сигмоїдної функції (рисунок 1.4).

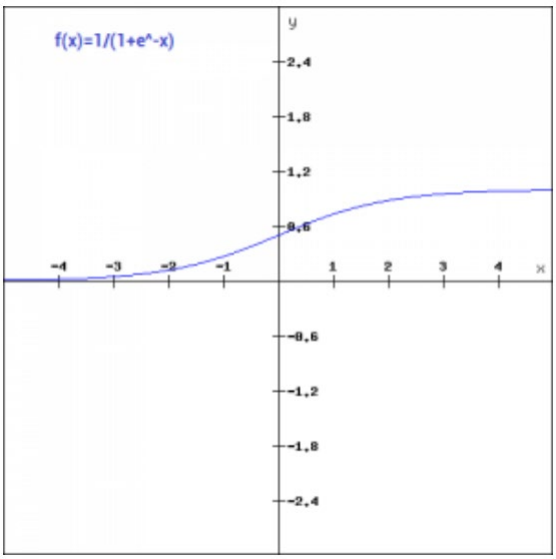


Рисунок 1.4 - Сигмоїдна функція [50]

Сигмоїдна функція була введена через негнучкість класифікаторів, заснованих на порогових функціях передачі, що дозволяє перейти від суворої однобітної логіки до більш гнучкої поведінки та адаптивної параметризації нейронних мереж.

Функція гіперболічного тангенса (Tanh) схожа на сигмоїдну функцію, але вона симетрична відносно початку координат. Це призводить до різних знаків виходів з попередніх шарів, які будуть подаватися на вхід наступного шару. Вона може бути визначена за (1.7) [50]:

$$(x) = 2sigmoid(2x) - 1 \qquad (1.7)$$

Функція Tanh є неперервною та диференційованою, її значення лежать в діапазоні від -1 до 1. У порівнянні з сигмоїдною функцією градієнт функції Tanh більш крутий. Функція Tanh є кращою за сигмоїдну функцію, оскільки її градієнти не обмежені у зміні в певному напрямку, а також вона має нульовий центр.

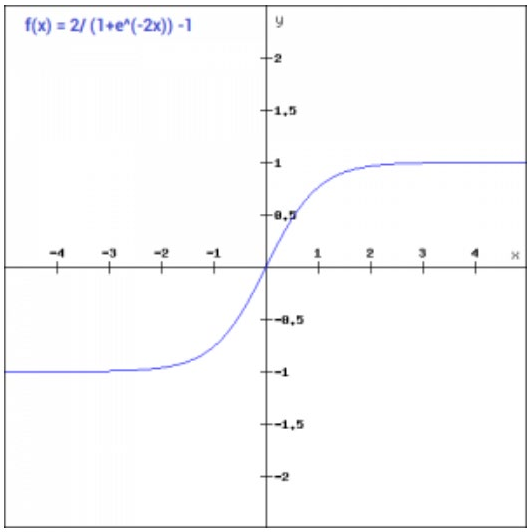


Рисунок .1.5 - Tahn функція [50]

Функція ReLU — це модифікована лінійна функція, яка зараз вважається простішим і ефективнішим з точки зору обчислень варіантом функції активації. Оскільки похідна цієї функції дорівнює 0 або 1, використання цієї функції запобігає росту та згасання градієнта, зменшує ваги та позитивно впливає на обчислювальну потужність нейронної мережі. Перевагою використання функції ReLU є те, що всі нейрони не активуються одночасно. Це означає, що нейрон буде деактивований тільки тоді, коли результат лінійного перетворення дорівнює нулю. Математично це можна виразити за (1.8) [50]:

$$f(x) = max(0, x) \qquad (1.8)$$

ReLU є більш ефективною, ніж інші функції, оскільки не всі нейрони активуються одночасно, а певна кількість нейронів активується одночасно. У деяких випадках значення градієнта дорівнює нулю, завдяки чому ваги та зміщення не оновлюються на кроці зворотного поширення при навчанні нейронної мережі. На рисунку 1.6 зображено граф активаційної функції ReLU.

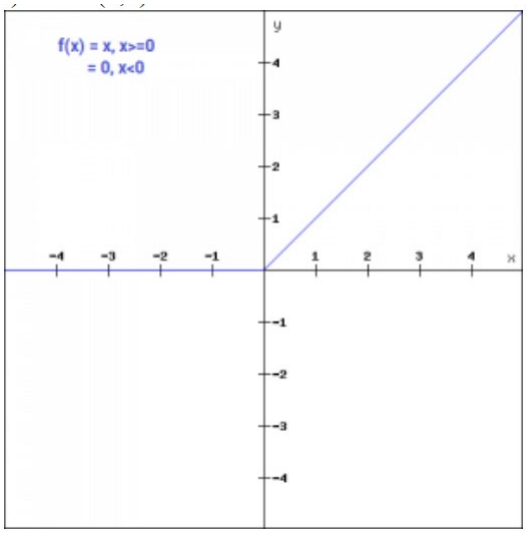


Рисунок 1.6 - функція активації ReLU [50]

Сьогодні існує цілий набір різних модифікацій ReLU, які стосуються надійності цієї функції при проходженні через нейрони з великими градієнтами: Leaky ReLU, Parametric ReLU, Randomized ReLU.

1.3 Вибір засобів розробки

1.3.1 Постановка задачі дослідження

Щоб реалізувати спеціалізовану нейронну мережу для дефектування дерев’яних виробів, необхідно вирішити ряд проблем.

Одним із класів проблем, які вирішують нейронні мережі, є семантична сегментація зображень. У таких завданнях необхідно класифікувати кожен піксель вхідного зображення відповідно до попередньо створеного списку можливих класів.

Традиційні алгоритми сегментації зображень зосереджені на кластеризації з використанням додаткової інформації про контури об’єктів на зображенні. Однак дослідження в області машинного навчання зробили ці алгоритми застарілими. На сьогодні для вирішення задачі семантичної сегментації існує три методи:

1. Семантична сегментація на основі регіонів (Region-based semantic segmentation)

Метод сегментації по області, як правило, слідує принципу "сегментації після розпізнавання", який спочатку виділяє області довільної форми із зображення і розпізнає об’єкти на них, після чого виконується сегментація на цих областях. Під час тестування передбачення на основі цих областей перетворюються на піксельні передбачення, шляхом надання класу пікселю за найбільшим значенням сегментованого передбачення області, що містить цей піксель [37]. Regions with CNN feature (RCNN) [38] є однією з репрезентативних робіт для методів сегментації на областях. Він виконує семантичну сегментацію на основі результатів виявлення об'єктів. Зокрема, RCNN спочатку використовує вибірковий пошук [39] для вилучення великої кількості пропозицій об'єктів, а потім обчислює ознаки CNN для кожного з них. Після цього він класифікує кожну область, використовуючи векторні машини SVM. У порівнянні з традиційними структурами ШНМ, які в основному призначені для класифікації зображень, RCNN може вирішувати більш складні завдання, такі як виявлення об'єктів і сегментація зображень. Крім того, RCNN може бути побудована на основі будь-яких CNN-архітектур, таких як AlexNet [40], VGG [41], GoogLeNet [42] та ResNet [43].

2. Сегментація на основі згорткових нейронних мереж(FCN-based semantic segmentation)

Ключова ідея методів на основі FCN [38,39,40] полягає в тому, що вони вивчають відображення від пікселя до пікселя, без розпізнавання областей з можливими об’єктами на них. FCN є розширенням класичного CNN. Основна ідея полягає в тому, щоб класичний CNN приймав на вхід зображення довільного розміру. FCN мають лише згорткові та об'єднуючі шари, які дають їм можливість робити прогнози на входах довільного розміру. Цей тип мереж зазвичай використовуються для локальних, а не глобальних завдань наприклад для задач семантичної сегментації [38], розпізнавання об'єктів [41]. Проте для задачі класифікації зображень [37] FCN не підходить. Оскільки FCN складаються з шарів згортки, pooling та upsampling шари, залежно від вибраної функції втрат, вони можуть бути натреновані для отримання результатів у форматі end-to-end.

3. Слабо контрольована семантична сегментація(Weakly supervised semantic segmentation)

Більшість методів семантичної сегментації спираються на велику кількість зображень із попіксельними масками сегментації. Однак, ручне анотування цих масок є досить трудомістким і комерційно дорогим процесом. Проте є метод, який призначений для виконання семантичної сегментації за допомогою анотованих обмежувальних рамок на об’єкті інтересу або просто міток класів зображеннь.

Наприклад в роботі [53] використовують анотації обмежувальних рамок на областях інтересу для тренування з покроковим покращенням по піксельно передбачених масок.

Одним з основних обмежень використання такого типу мереж є ігнорування локалізації об'єкта на зображеннях. Для покращення ефективності локалізації деякі підходи [58, 59, 60, 61] пропонують змінити поняття об'єкту інтересу, включити елементи у функцію втрат [58, 59], використовуючи

попередньо навченої нейромережі для зовнішнього контролю як модуль для знаходження об’єктів [60, 61]. Іншим перспективним способом покращення ефективності сегментації є використання додаткових слабо контрольованих зображень, таких як зображення з веб ресурсів, для навчання ШНМ [62, 63].

На основі вищезазначених підходів, було вирішено створити повністю згортову нейронну мережу для вирішення задачі семантичної сегментації зображення. Оскільки це дозволяє отримувати результати прогнозу у форматі, який можна використовувати одразу. Проблема виявлення дефектів на дерев’яних виробах відноситься до задач локального типу, що вирішуються за допомогою таких мереж.

1.3.2 Огляд існуючих засобів для реалізації

1. Javascript є однією з найпоширеніших мов програмування, вона є високорівневою, динамічно типізованою, гнучкою та дотримується багатьох парадигм. Популярний для створення продуктів на основі машинного навчання з використанням таких фреймворків, як Tensorflow.js. Використовуючи Javascript, можна створити продукт повністю від серверної та клієнтської сторони до створення та навчання нейронних мереж. Можливість вибрати один із кількох популярних фреймворків машинного навчання. Фреймворк Brain.js може використовувати прискорення GPU та інтегрувати навчену модель в інший фреймворк Node.js у браузері. Machinelearn.js — це еквівалент бібліотеки ScikitLearn на Python із кластеризацією даних, факторизацією та утилітами для навчання з репетиторами та без репетиторів.

2. R є однією з основних мов для data science. R є однією з основних мов для науки про дані. Він забезпечує чудові функції візуалізації, що є важливим для вивчення даних перед подачею їх на автоматизоване навчання, а також для оцінки результатів алгоритму навчання. Багато R-пакетів для машинного навчання є готовими, і багато сучасних методів статистичного навчання реалізовано в R як частину їх розробки. Корисність R для data science випливає з великої, активної та зростаючої екосистеми пакетів сторонніх

розробників: tidyverse для загальної діяльності з аналізу даних; h2o, ranger, xgboost та інші для швидкого й масштабованого машинного навчання; iml, pdp, vip та інші для інтерпретації машинного навчання.

3. Python став лінгва франка для багатьох програм обробки даних. Він поєднує потужність мов програмування загального призначення з простотою використання предметно-спеціальних мов сценаріїв, таких як MATLAB або R. У Python є бібліотеки для завантаження даних, візуалізації, статистики, обробки природної мови, обробки зображень та інші. Цей широкий набір інструментів надає дата аналітикам великий набір універсальних та спеціалізованих можливостей. Однією з основних переваг використання Python є можливість працювати безпосередньо з кодом за допомогою інших інструментів, таких як Terminal або Jupyter Notebook. Машинне навчання та аналіз даних - це принципово ітераційні процеси, в яких аналіз управляється даними. У цих процесах важливо мати інструменти, що забезпечують швидку ітерацію та просту взаємодію. Будучи мовою програмування загального призначення, Python також дозволяє створювати складні графічні інтерфейси (GUI), веб-служби та інтеграцію в існуючі системи.

Після аналізу всієї наведеної вище інформації про мови програмування для машинного навчання було обрано мову програмування Python з наступних причин:

- простота використання;
- Можливість використання таких фреймворків, як TensorFlow [29] та PyTorch [30].
- Кількість документів та прикладів реалізації доступних в Інтернеті.
- Широка підтримка популярних платформ машинного навчання у хмарних середовищах.

4. TensorFlow - це безплатна бібліотека з відкритим вихідним кодом для швидких чисельних обчислень. Вона була створена і підтримується компанією Google і була випущена під ліцензією Apache 2.0 з відкритим

вихідним кодом. API номінально призначений для мови програмування Python, хоча є доступ до базового API C++.

На відміну від інших числових бібліотек, призначених для використання в глибокому навчанні, таких як Theano, TensorFlow був розроблений для використання як в дослідженнях і розробках, так і в виробничих системах.

Він може працювати як на однопроцесорних системах і графічних процесорах, так і на мобільних пристроях і великомасштабних розподілених системах з сотень машин.

TensorFlow забезпечує стабільний Python (для версії 3.7 на всіх платформах) та C API; і без гарантії зворотної сумісності API: C ++, Go, Java, JavaScript та Swift (достроковий випуск). Сторонні пакети доступні для C#, Haskell, Julia, R, Scala, Rust, OCaml та Crystal. "Нова мовна підтримка повинна бути побудована на версії API C. Однак, ще не всі функціональні можливості доступні в C". Ще кілька функціональних можливостей надає API Python.

5. Keras — це високорівневий API глибокого навчання, який дозволяє легко створювати, навчати, оцінювати та запускати всі види нейронних мереж. Він був розроблений Франсуа Шолле в рамках дослідницького проекту ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) [47] та випущений як проект з відкритим вихідним кодом у березні 2015 року. Він швидко набув популярності завдяки своїй простоті у використанні, гнучкості та красивому дизайну. Для виконання важких обчислень, необхідних для нейронних мереж, keras-team покладається на обчислювальний сервер. На даний момент можна вибрати одну з трьох популярних відкритих бібліотек глибокого навчання: TensorFlow, Microsoft Cognitive Toolkit (CNTK) або Theano.

Крім того, з кінця 2016 року були випущені інші реалізації. Тепер можна запускати Keras на Apache MXNet, Apple Core ML, Javascript або Typescript (для запуску коду Keras у веб-браузері) або PlaidML (який може працювати на всіх видах графічних пристроїв, не лише на Nvidia). Крім того,

сам TensorFlow тепер поставляється в комплекті з власною реалізацією Keras під назвою tf.keras. Він підтримує лише TensorFlow як бекенд, але має перевагу в тому, що пропонує деякі дуже корисні додаткові функції (див. рисунок 1.7): наприклад, він підтримує Data API TensorFlow, що дозволяє досить легко завантажувати та ефективно обробляти дані. З цієї причини у цій роботі буде використовуватись tf.keras. Також бібліотека підтримує згорткові і рекурентні мережі.

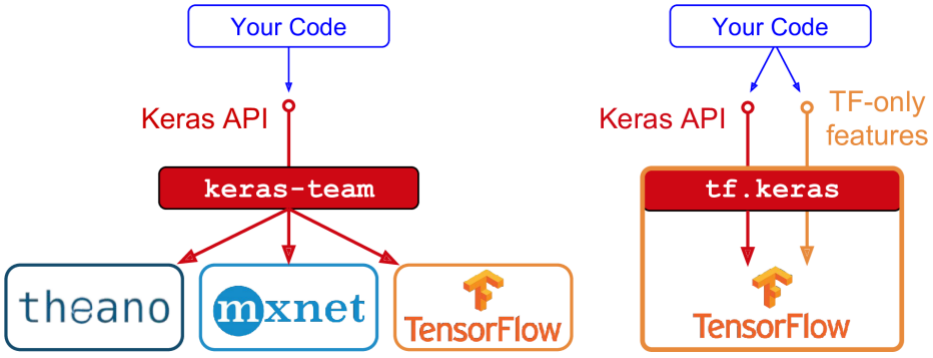


Рисунок 1.7 – Дві реалізації Keras: keras-team (ліворуч) і tf.keras (праворуч) [48]

Keras використовує різні методи оптимізації, щоб зробити високорівневий API нейронної мережі простішим і ефективнішим. Він підтримує наступні функції:

- Послідовний, простий і розширений API;
- Мінімальна структура - легко досягти результату без будь-яких надмірностей;
- Підтримує кілька платформ і серверних модулів;
- Це зручна структура, яка працює як на CPU, так і на GPU;
- Висока масштабованість обчислень.

6. Google Colaboratory — це безкоштовне середовище для Jupyter Notebook, яке повністю працює в хмарі та зберігає записники на Google Drive. Google досить агресивно займається дослідженнями ШІ. Протягом багатьох років Google розробляв фреймворк AI під назвою TensorFlow та інструмент розробки під назвою Colaboratory. Сьогодні TensorFlow має відкритий код, а з 2017 року Google зробив Colaboratory безкоштовним для загального використання. Colaboratory тепер відомий як Google Colab або просто Colab. Ще одна приваблива функція, яку Google пропонує розробникам — використання GPU. Colab підтримує графічний процесор і є абсолютно безкоштовним.

Впровадження Colab полегшило навчання та розробку програм машинного навчання. Colab підтримує багато популярних бібліотек машинного навчання, які можна легко завантажити у свій блокнот.

В результаті дослідження була обрана платформа Google Colab для спрощення навчання нейромережі, адже він безкоштовний, має досить великий ресурс і може працювати автономно.

1.3.3 Апаратне середовище для тренування моделі

Залежно від типу обладнання, яке ви оберете, час навчання може бути скорочено. Це ще раз демонструє важливість цього питання, перш ніж ми почнемо розробляти модель та навчати її. Поява прискорення у навчанні пов'язані з тим, як апаратна платформа обробляє вхідні дані. У період бурхливого розвитку в галузі машинного навчання була розроблена та запущена у виробництво платформа нового типу, унікальна у своєму роді. Його використання повністю орієнтоване на швидке навчання нейронних мереж. Цей тип платформи можна класифікувати як ASIC (спеціалізована інтегральна схема). Можливі такі варіанти:

1. Навчання ЦП (CPU) - найпростіший і найдоступніший спосіб навчання нейронної мережі, але й найповільніший. Це пов’язано з тим, що кількість паралельних обчислень, які може виконувати ЦП, невелика.
2. Навчання на графічних процесорах (GPU) - найпопулярніший спосіб навчання нейронних мереж, з високими можливостями паралельних обчислень та великою кількістю комерційних графічних процесорів від Nvidia та AMD.
3. Навчання на тензорі TPU - це найшвидший спосіб навчання, але за цінами на рівні підприємств та дослідницьких інститутів, а не для середнього ентузіаста машинного навчання. Однак є можливість розпочати навчання TPU за допомогою платформи Google Colab, яка надає віртуальні машини з TPU. Обмеження в цьому випадку полягають у тому, де зберігати дані для навчання моделі, тобто в хмарному сховищі Google або Google Диску, а також в обмеженні часу безвідмовної роботи віртуальної машини до 12 годин для обмеження зловживань. платформи.

Висновки до розділу 1

1. Проведено огляд існуючих підходів рішень задачі дефектування дерев’яних виробів.

2. Отримана та досліджена значна кількість інформації щодо будови та використання систем класифікації дерев’яних виробів за допомогою штучного інтелекту.

3. Проаналізовано існуючі засоби розробки для реалізації методу. На основі цих досліджень було вибрано мову програмування Python, оскільки вона має великий набір інструмендів для реалізації даного методу та підтримуючих бібліотек та вибрано апаратне середовище для розробки методу класифікації дерев’яних виробів.

4. На основі проведених досліджень, що були виконані у цьому розділі, було зроблено постановку задачі дослідження

2 РОЗРОБЛЕННЯ МЕТОДУ РОЗПІЗНАВАННЯ ДЕФЕКТІВ ТА КЛАСИФІКАЦІЇ ДЕРЕВ’ЯНИХ ВИРОБІВ З ДОПОМОГОЮ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Нейронна мережа — це математична модель взаємопов'язаних штучних нейронів, організованих у єдину структуру і які впливають одне одного. Ця концептуальна модель була розроблена на основі того, як працюють клітини мозку.

Завдання нейронної мережі полягає в тому, щоб перетворити вхідний вектор у вихідний вектор. Для задач сегментації вхідним вектором є зображення, а вихідним вектором його маска. Теоретично пов'язані нейронні мережі прямого поширення можна використовувати для завдань комп'ютерного зору, але використовувати цю архітектуру для зображень недоцільно. мережа. Крім того, пошук шуканих ознак здійснюється лише на тій локальній частині зображення, що знаходилася у процесі навчання.

2.1 Запропонований спосіб класифікації дерев’яних виробів

2.1.1 Автоенкодер

Автокодер (англ. Autoencoder, autoencoder, AE) - нейронна мережа, яка копіює вхідні дані на вихід. Його архітектура схожа на перцептрон. Автокодери стискають вхідні дані, щоб представити їх у прихованому просторі, а потім реконструюють вихідні дані з цього представлення. Мета полягає в тому, щоб отримати зворотний зв’язок на вихідному рівні, найближчому до вхідного. Примітною особливістю автокодерів є однакова кількість нейронів на вході та виході.

Автокодер складається з двох частин:

Кодер: відповідає за стиснення прихованого простору введення. Виражається функцією кодування $h = f(x)$;

Декодер: спрямований на відновлення вхідних даних із прихованого простору. Він представлений функцією декодування $h = f(x)$.

Таким чином, автокодер описується функцією (1.9):

$$g(f(x)) = r, \tag{2.1}$$

де r відповідає вихідному x у вхідних даних.

Основним принципом роботи та навчання мережі автокодера є отримання відповіді на вихідному рівні, найближчому до вхідного. Накладіть обмеження на проміжні рівні автокодера, щоб гарантувати, що рішення не стане відношенням ідентичності. Прихований шар або менший за розміром, ніж вхідний і вихідний шари, або існує штучне обмеження на кількість одночасно активних нейронів у прихованому шарі, що називається розрідженою активацією. Ці обмеження змушують модель шукати узагальнення і залежності даних, що надходять на вхідні рівні мережі, і виконувати їх стиснення. Таким чином, ШНМ автоматично навчається витягувати загальні ознаки, закодовані у значеннях ваги ШНМ, із вхідних даних. Таким чином, коли мережа навчається на іншому наборі вхідних зображень, нейронна мережа може самостійно вивчати функції високого рівня, такі як дуги та прямі лінії. На рисунку 2.1 показана схематична структура автокодера.

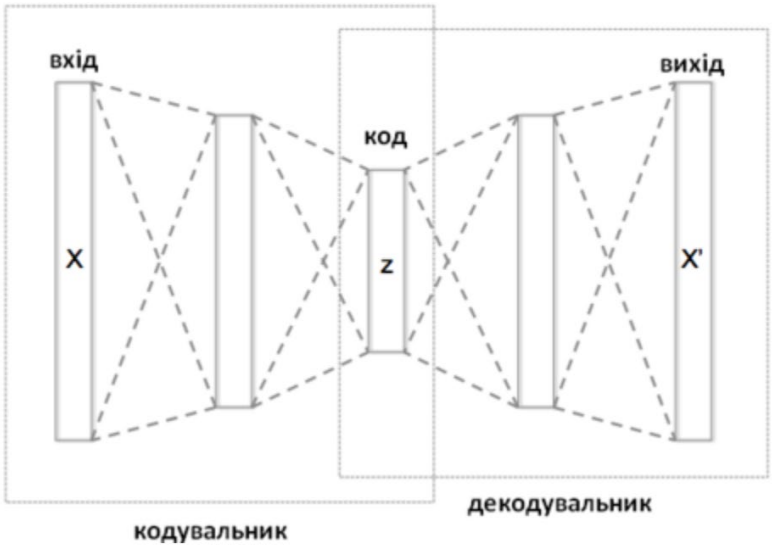


Рисунок 2.1 - Схематична структура автоенкодеру

Автоенкодери найчастіше використовуються як каскадні елементи для навчання глибоких мереж. Автоенкодери також можна використовувати для попереднього навчання абстрактної глибокої мережі в неконтрольованому навчанні. І тому шари поступово тренуються поперемінно. До кожного нового шару під час навчання підключається окремий вихідний шар. Це наближає мережу архітектурно до автоэнкодеру, після якого на вхід мережі подається один пакет даних. Параметри всіх шарів такої мережі оптимізуються за допомогою зворотного розповсюдження помилок. Потім один шар автоенкодера відключається і додається новий шар, що відповідає наступному ненавченому шару мережі. Той самий набір даних знову надходить на вхід мережі, а перший навчений шар мережі залишається незмінним, виступаючи в ролі 52-константного перетворення для входу наступного шару автоенкодера, що навчається. Тому навчання проводиться всім шарів мережі, крім останнього шару. Останній шар також можна навчити шляхом зворотного поширення помилки в режимі наставника.

З архітектурної точки зору найпростішим варіантом автоенкодера є повнозв'язкова нейронна мережа із прямим зв'язком. Насправді це схоже на багатошаровий перцептрон з вхідним шаром, вихідним шаром і одним або декількома прихованими шарами, що їх з'єднують. Різниця між автоенкодерами і багатошаровими персептронами полягає в тому, що в автоенкодерах вихідний шар має ту ж кількість вузлів, що і вхідні шари, і замість навчання прогнозуванню цільового значення змінної Y при заданому вході X автоэнкодер тренується. За визначенням автоенкодер завжди складається з 53 двох основних блоків, енкодера і декодера, які можна визначити як функції (2.2), (2.3)

$$\varphi: X \to F;\ \psi: F \to X \tag{2.2}$$

$$\varphi, \psi = argmin_{\varphi,\psi} \|X - (\psi \circ \varphi)X\|^2 \tag{2.3}$$

Коли кількість прихованих шарів дорівнює 1, автокодер приймає деякі вхідні дані x $\in \mathbb{R}^d$ та виконує відображення $z \in \mathbb{R}^p$ (2.4):

$$z = \sigma_1(Wx + b). \tag{2.4}$$

У цьому формулюванні σ є функцією активації, такою як сигмоїдна функція або ReLU. Після цього z відображається на реконструкцію x' з тими самими розмірами, що й x (2.5):

$$x' = \sigma_2(W_z' + b'). \tag{2.5}$$

Автокодер також навчено мінімізувати помилку реконструкції, наприклад, середньоквадратичну помилку (2.6):

$$L(x, x') = \|x - x'\|^2. \quad (2.6)$$

Якщо розмірність простору ознак нижча, ніж розмірність вхідного простору X, тоді вектор ознак φ() можна розглядати як стиснуте представлення вхідних даних. Крім того, якщо розмірність прихованого шару більша, ніж розмірність вхідного рівня мережі, автокодер теоретично може навчитися тим самим функціям і стати марним. Однак на практиці автокодери можуть навчитися отримувати корисні функції навіть у цьому випадку.

2.2 Архітектури штучних нейронних мереж

2.2.1 Архітектура FCN

Повністю згортковамережа (англ. Fully Convolutional Network) [44] – це архітектура в основі якої є CNN мережа, в якій з'єднувальні шари, що виконують функцію класифікації ознак, замінені на upsampling шари, які локалізують ознаки (рисунок 2.2).

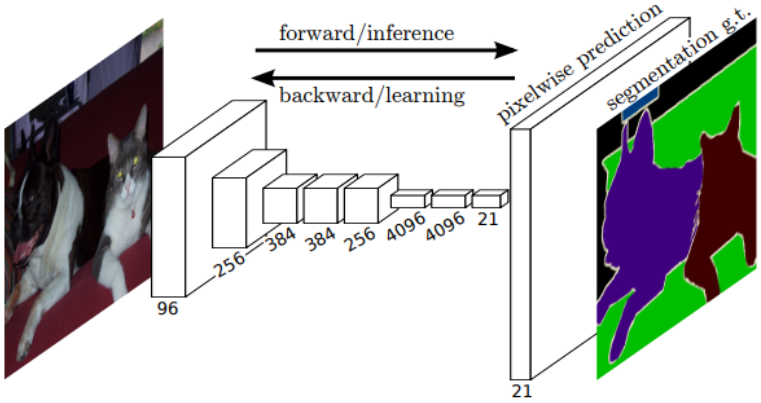


Рисунок 2.2 - архітектура FCN [44]

FCN є розширенням класичного CNN. Основна ідея полягає в тому, щоб класичний CNN приймав на вхід зображення довільного розміру. FCN мають лише згорткові та об'єднуючі шари, які дають їм можливість робити прогнози на входах довільного розміру. Цей тип мереж зазвичай використовуються для локальних, а не глобальних завдань, наприклад, для задач семантичної

сегментації [29], розпізнавання об'єктів [40]. Проте для задачі класифікації зображень [37] FCN не підходить. Оскільки FCN складаються з шарів згортки, pooling та upsampling шари, залежно від вибраної функції втрат, вони можуть бути натреновані для отримання результатів у форматі end-to-end, тобто може отримувати фінальний результат без будь-яких постобробок.

Однією з основних особливостей згорткових нейронних мереж є те, що вони не вимагають попередньої обробки даних. Для обробки зображень за допомогою мереж згортання це означає, що мережа сама створює фільтри, які для традиційних підходів повинен розробляти експерт. Перевага полягає в тому, що такі мережі не залежать від апріорних знань предметної галузі. Згорткова нейронна мережа між входом та виходом має прихований шар. Зазвичай вони складаються зі згорткових шарів, що поєднують шарів, шарів нормалізації та повнозв'язкових шарів. Дані досліджень зорової кори мозку були використані ідеї згорткових нейронних мереж. Згорткові шари моделюють реакцію нейронів мозку на візуальні сигнали. Один згортковий нейрон може обробляти інформацію лише у межах свого рецептивного поля. Рецептивні поля всіх нейронів накладаються на зображення.

2.2.2 Архітектура SegNet

Модель SegNet [33] є типовим автокодувальником, заснованим на

згортковій нейронній мережі. Схема цієї моделі представлена на Рисунок 2.3 . Мережа складається з блоків, в кожному блоці присутні згортки і пулінги (субдискретизуючі шари) або шари, що підвищують дискретизацію (апсемплінг шари), а також активаційні шари ReLU [36] та шари нормалізації батч-нормалізації (BatchNor). Архітектура повністю симетрична, крім шару м'якого максимуму (Softmax) наприкінці декодера. Цей шар перетворює кожен піксель вихідний матриці на ціле число, що показує клас даного пікселя. Головна відмінність SegNet від звичайного згорткового автокодувальника в тому, що його апсемплінг шари декодера інформаційно з'єднані з відповідними пулінг шарами енкодера. Тобто, апсемплінг шари мережі не навчаються, а отримують

необхідну інформацію про те, як підвищити розміри та як відновити стислу (втрачену) топологію від відповідних пулінг шарів, які зберігають індекси активованих пікселів (пікселів з найбільшим значенням у вікні).

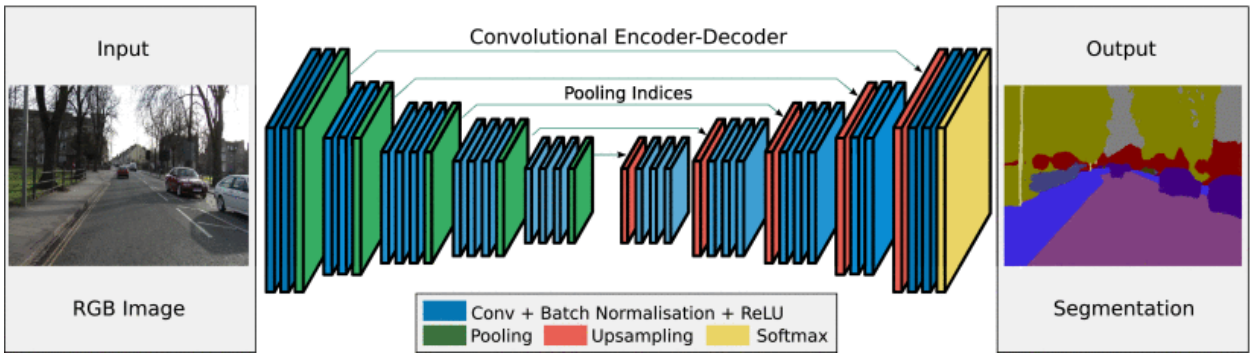


Рисунок 2.3 - Архітектура SegNet[33]

SegNet використовується в декодер Anti-Pooling (див. рисунок 2.4), карта характеристик піддається дискретизації, і цілісність високочастотних деталей зберігається під час сегментації. Кодер не використовує повністю підключений рівень (згортка, як FCN), тому це легковажна мережа з меншою кількістю параметрів.

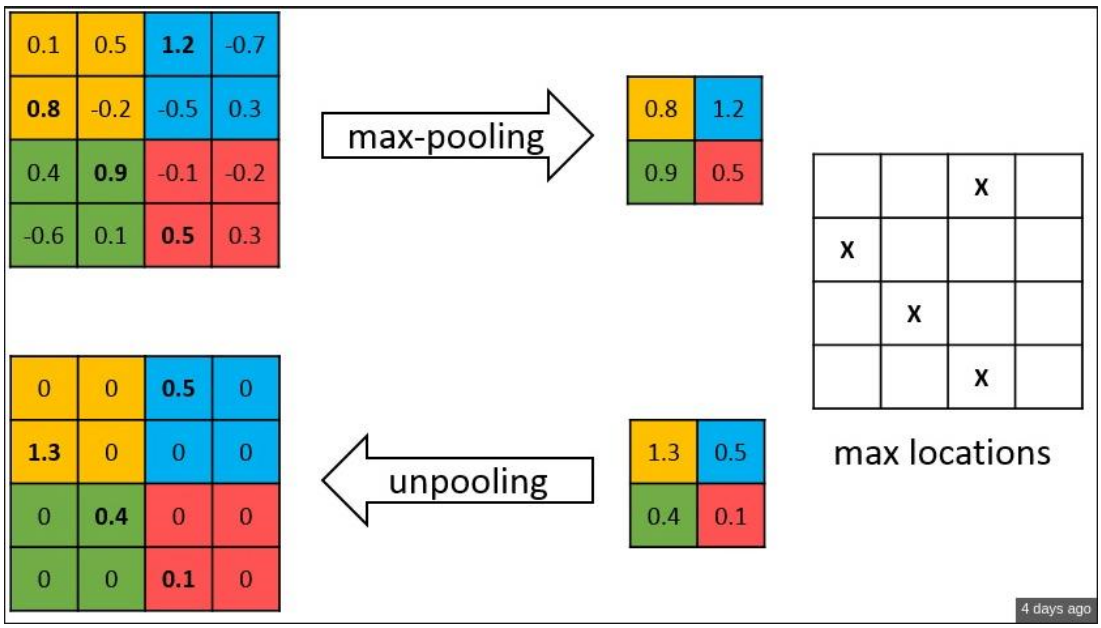


Рисунок 2.4 - Anti-Pooling

Як показано на малюнку вище, індекси кожного найбільшого рівня пулу в кодері зберігаються для подальшого використання декодері для видалення відповідних карт ознак. Хоча це допомагає підтримувати цілісність високочастотної інформації, воно також ігнорує сусідню інформацію при видаленні з пулів карток з низьким розширенням.

2.2.3 Архітектура U-Net

Архітектура UNET є одним із найяскравіших представників моделей кодувальника/декодера сегментації, незважаючи на те, що були опубліковані мережі для сегментації біологічних зображень [32]. Спочатку UNET був розроблений Олафом Роннебергером, Філіппом Фішером і Томасом Броксом і застосований до біомедичної візуалізації у 2015 році [32]. Що стосується архітектури нейронної мережі, то вона складається з одією половини, що звужується (згортки), і другої половини, що розширюється (розгортки). Архітектура UNET зображена на рисунку 2.5. Пропонована мережа складається з 4 блоків кодування, середнього блоку та 4 блоків декодування. Блок кодувальника складається з двох шарів згортки (позначених фіолетовими стрілками на зображенні) з вікнами згортки розміром 3 на 3 пікселя та функцією ReLu для активації, а також вікна розміром 2 на 2 пікселі. шар пулу (позначений червоною стрілкою). Після проходження всіх чотирьох блоків кодування, що зменшують розмірність зображення в 2 рази, зображення зменшується до розміру 32x32 пікселя і використовується 512 функціональних каналів. Після проходження проміжних блоків розпочинається процес декодування.

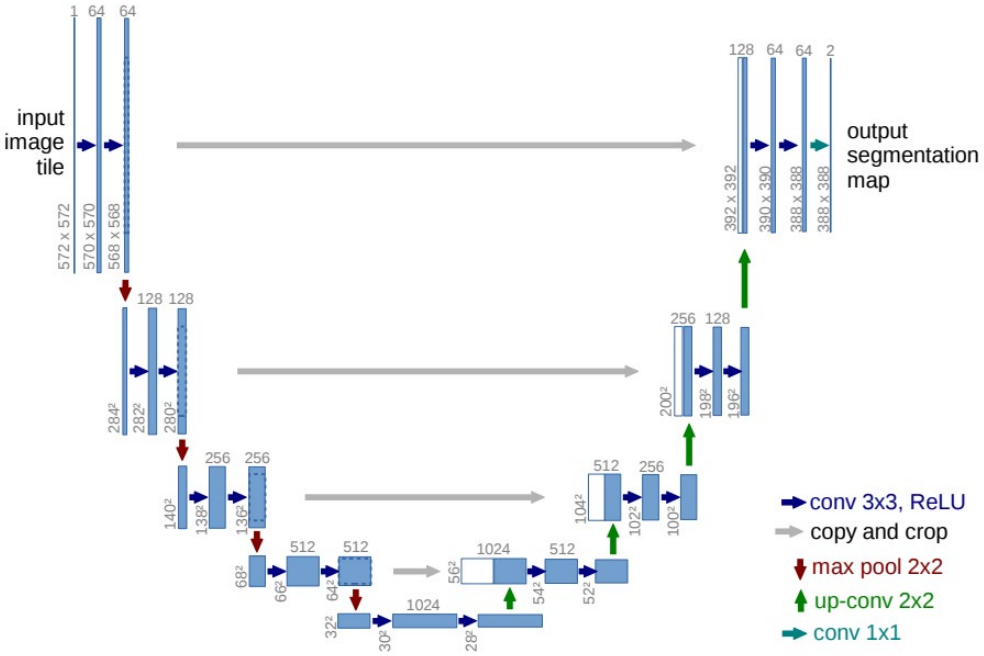


Рисунок 2.5 - Архітектура UNET [32]

Блок декодування також складається з двох послідовних згорткових шарів, за якими слідує шар розгортки (позначений зеленим), який подвоює розмір зображення. Після проходження останнього блоку декодування вихідне зображення набуває тих самих розмірів, що і вхідне. Останній являє собою шар згортки з вікном розміром 1 на 1 піксель, що використовує функцію сигмовидную активацію, яка створює вихідну маску для зображення. Загалом в архітектурі, запропонованій Олафом Роннебергером, використовувалося 23 згорткові шари. З'єднання між блоками симетричного кодера та декодера (позначені сірими стрілками на зображенні) використовуються для відновлення дрібнозернистої інформації.

2.3 Вибір метрик та функцій втрат

Сегментацію зображення можна визначити як завдання класифікації на рівні пікселів. Зображення складається з різних пікселів, і ці пікселі, згруповані разом, визначають різні елементи зображення. Метод класифікації цих пікселів

на елементи а називається семантичною сегментацією зображення. Вибір функції втрати/об’єктивності надзвичайно важливий під час розробки архітектур глибокого навчання на основі сегментації складних зображень, оскільки вони стимулюють процес навчання алгоритму.

Метрики використовуються в завданнях машинного навчання для оцінки якості моделі та порівняння різних алгоритмів.. Для фіксованої параметризованої архітектури алгоритму машинного навчання необхідно знайти безліч параметрів, при яких функція помилки (при виборі якої відштовхуються від типу конкретної завдання) має найменше значення. При цьому необхідно стежити, щоб у процесі пошуку мінімуму узагальнююча здатність алгоритму зростала.

Оптимізовані функції завдання машинного навчання діляться на три категорії:

Функції втрат (Loss functions/objectives) дозволяють обчислити помилку алгоритму на кожному конкретному об'єкті. Вибираються зазвичай безперервними і майже всюди диференційованими, щоб застосовувати градієнтні методи. Середнє значення функції втрат на датасеті (функціонал якості алгоритму) оптимізується у процесі навчання.

Регуляризатори (Regularizers) — вводяться у функціонал якості для того, щоб зробити оптимізаційне завдання навчання коректним, процес навчання є стійким, та отримати якісніше рішення. Найчастіше регуляризація допомагає уникнути перенавчання. На відміну від функції втрат, для обчислення значення регулятора часто не потрібно використання інформації про об'єкти з датасета.

Метрики якості (Metrics) — функції, якими визначається (валідується) якість навченої моделі (як у всьому датасете загалом, і окремих моделях), але відбувається безпосередньої оптимізації. Використовуються для контролю перенавчання та для порівняння різних моделей.

2.3.1 Intersection over Union (IoU, Jaccard Index)

Метрика Intersection over Union (IoU), також відома як Jaccaed index – також відомий як індекс Жаккара, є одним із найбільш часто використовуваних показників у семантичній сегментації. На рисунку2.6 зображено наочну фомулу обрахунку метрики.

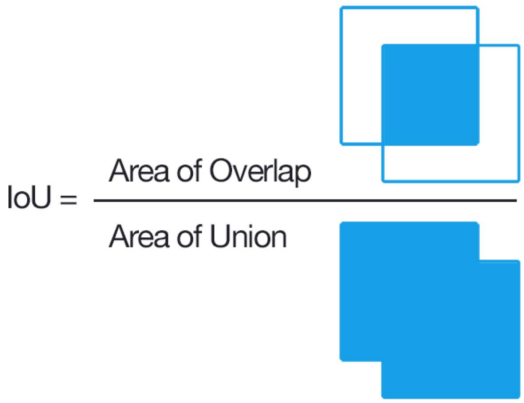


Рисунок 2.6 - Наочна формула обрахунку IoU

IoU — це площа перекриття між прогнозованою сегментацією та базовою правдою, поділена на площу об’єднання між прогнозованою сегментацією та базовою правдою, як показано на зображенні ліворуч. Цей показник коливається від 0–1 (0–100%), де 0 означає відсутність накладання, а 1 означає ідеальне накладання сегментації.

Формально, для двох непустих множин A і B, функція IoU оприділяється як (2.7):

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.7}$$

Для того, щоб підрахувати IoU, необхідно вміти обчислювати внутрішній обсяг об'єктів, що розглядаються. У випадку з полігональними моделями найчастіше вдаються до оцінки обсягу методом Монте-Карло. Також для коректного обчислення IoU необхідно, щоб порівнювані моделі мали однаковий

масштаб та орієнтацію. На рисунку 2.7 зображено приклад оцінки якості метрики Mean IoU.

Рисунок 2.7 - приклад оцінки якості метрики IoU

У задачах комп'ютерного зору і 3D ML, IoU часто використовують при оцінці того, наскільки коректно знайдений прямокутник, що обмежує або обмежує паралелепіпед (bounding box), тобто. як метрика якості алгоритму, але останнім часом з'явилися різні модифікації IoU, які можуть використовуватися і як функція втрат [49] .

2.3.2 Binary cross entropy

Binary cross entropy використовується для вимірювання різниці між двома розподілами ймовірностей. Він використовується як показник подібності, щоб визначити, наскільки один розподіл випадкових подій близький до іншого, і використовується як для класифікації (у більш загальному сенсі), так і для сегментації.

Таким чином, Binary cross entropy (BCE) намагається виміряти відмінності інформаційного вмісту між фактичною та прогнозованою масками зображення. Він більш загально заснований на розподілі Бернуллі та найкраще працює з рівномірним розподілом даних між класами. Іншими словами, маски зображення з дуже сильним дисбалансом класів (наприклад, при виявленні дуже маленьких рідкісних пухлин на рентгенівських зображеннях) можуть бути неадекватно оцінені BCE.

Це пов’язано з тим, що BCE однаково обробляє як позитивні (1), так і негативні (0) зразки в масці зображення. Оскільки може бути нерівномірний розподіл пікселів, які представляють даний об’єкт (скажімо, автомобіль із першого зображення вище), і решту зображення, втрата BCE може не ефективно відображати продуктивність моделі глибокого навчання.

Binary cross entropy визначається як (2.8) [34]:

$$L_{BCE(y,p)} = -(y\log(p) + (1-y)\log(1-p)) \qquad (2.8)$$

де y – реальне значення класу 0 або 1;

p – передбачене значення класу від 0 до 1.

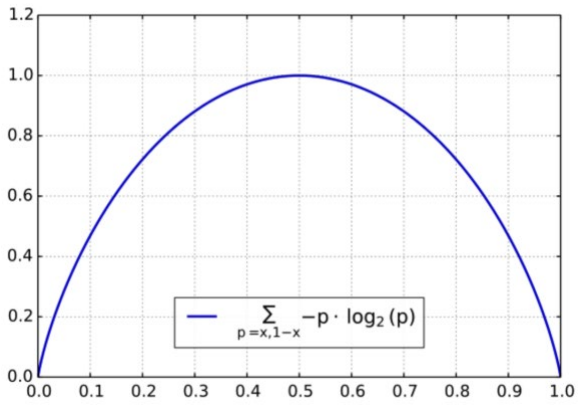


Рисунок 2.8 - Граф функції втрат Binary cross entropy[34]

З рисунку 2.8 ентропія визначається на осі Y, а ймовірність події – на осі X.

2.3.3 Коефіціїєнт Dice (F1 Score) I Dice loss

Коли виконується завдання семантичної сегментації, потрібно оцінити модель або під час навчання, тобто на етапах перевірки, або після навчання, тобто на етапах тестування. Потрібно обчислювати метрику, яка є рівнянням між основною істиною та прогнозованою маскою. І, дивлячись на значення цих

показників, ми можемо сказати, добре навчається модель чи ні. Таким чином, рівняння коефіцієнта Dice, яке можна використовувати як метрику, дорівнює подвоєному перетину між основною істинністю та прогнозованою маскою, поділеною на суму основної істинності та прогнозованої маски.

Коефіцієнт Dice визначається за (2.9):

$$Dice = \frac{2 * |A \cap B|}{|A| + |B|} \tag{2.9}$$

де A — передбачуваний набір пікселів,

B — основна правда.

Коефіцієнт Dice дорівнює 1, коли обидва поля A і B порожні. Оцінка таблиці лідерів є середнім значенням коефіцієнтів Dice для кожного зображення в тестовому наборі. Наочна формула обрахунку коефіцієнта Dice зоражено на рисунку 2.9

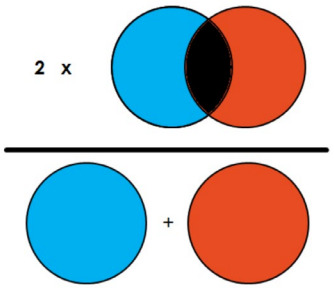


Рисунок 2.9 - Наочна формула обрахунку коефіцієнта Dice

Коефіцієнт Дайс широко використовується в завданнях комп'ютерного зору для обчислення подібності між двома зображеннями. Пізніше у 2016 році її також адаптували як функцію втрат, відому як Dice Loss.

Dice loss визначається за формулою (2.10) [34]:

$$L_{Dice} = 1 - \frac{2yp + 1}{y + p + 1} \quad (2.10)$$

де y – реальне значення класу 0 або 1;

p – передбачене значення класу від 0 до 1.

1 додається в чисельнику та знаменнику, щоб гарантувати, що функція не є невизначеною в крайових сценаріях, наприклад, коли $y = p = 0$.

2.3.4 Точність (accuracy)

Матриця помилок — це матриця N x N, яка використовується для оцінки продуктивності моделі класифікації, де N — кількість цільових класів. Матриця порівнює фактичні цільові значення з прогнозованими моделлю машинного навчання. Це дає цілісне уявлення про те, наскільки добре працює модель класифікації та які помилки вона допускає. На рисунку 2.10 наочно зображено матрицю помилок.

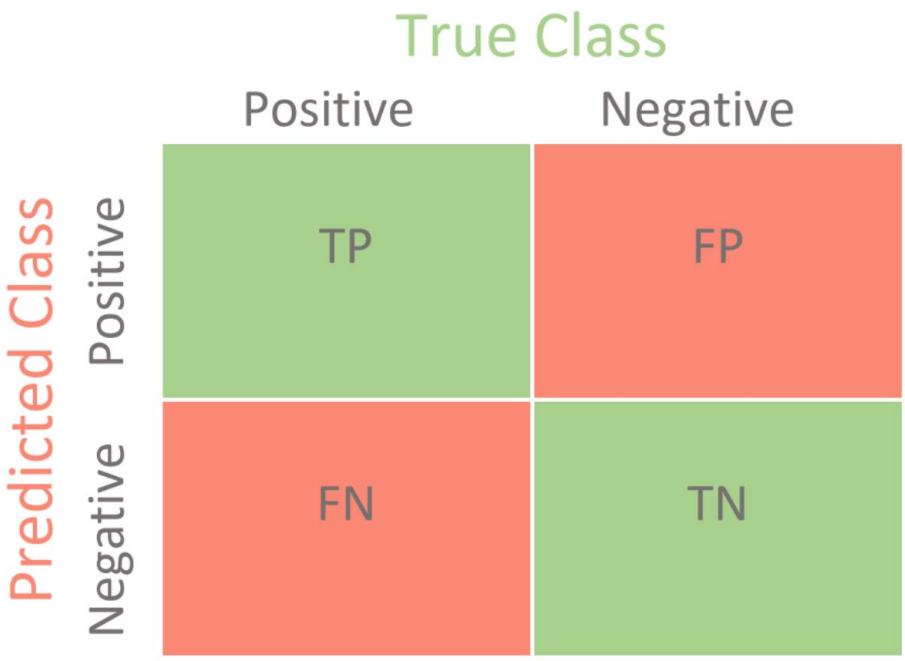


Рисунок 2.10 - Матриця помилок

Цільова змінна має два значення: позитивне(Positive) або негативне(Negative) Стовпці представляють фактичні значення цільової змінної Рядки представляють прогнозовані значення цільової змінної. У таблиці 2.1 зображенно розшифрування значень TP FP FN TN.

Таблиця 2.1 – Матриця помилок

		Реальність	
		+	-
Прогноз		True Positive (істинно-позитивне рішення): прогноз відповідає дійсності, результат є позитивним, як і було передбачено ML-моделлю	False Positive (хибнопозитивне рішення): помилка 1-го роду, ML-Фактичне значення було негативним, але модель передбачила позитивне

			значення
		False Negative (хибнонегативне рішення): помилка 2-го роду - Фактичне значення було позитивним, але модель передбачила негативне значення	True Negative (істинно-негативне рішення): результат негативний, ML-прогноз збігся з реальністю

Точність (accuracy) — інтуїтивно зрозуміла, очевидна й майже невикористана метрика — частка правильних відповідей алгоритму. Точність є одним з показників для оцінки моделей класифікації. Неформально, точність – це частка прогнозів, які наша модель отримала правильно. Формально точність має таке визначення (2.11):

$$acurracy = \frac{number\ of\ correct\ predictions}{total\ number\ predictions} \quad (2.11)$$

Для двійкової класифікації точність також можна розрахувати з точки зору позитивних і негативних показників таким чином (2.12) [34]:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.12)$$

де TP = True Positives,

TN = True Negatives,

FP = False Positives,

FN = False Negatives.

Ця метрика може давати результати, що вводять в оману, якщо представлення класу на зображенні невелике. Це в основному пов'язано з

похибкою вимірювань, які повідомляють про те, наскільки добре виявлено негативні випадки (тобто області, в яких відсутній клас).

2.4 Порівнняльний аналіз архітектур FCN I U-Net

В результаті досліджень [35], було побудовано наступні графіки, що порівнюють між собою зазначені вище моделі за такими характеристиками, як точність та функції втрат при наборі даних 256x256 пікселів та 512x512 (див. рисунок 2.11).

Для оцінки точності роботи архітектур використовуються метрики Mean IoU та точність натренованої моделі (accuracy).

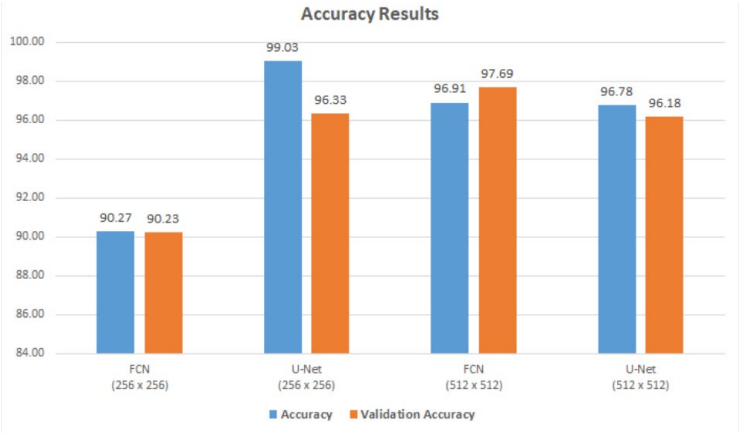


Рисунок 2.11 - Результати оцінки точності UNET архітектури та FCN[35]

На рисунку 2.11 представлено графіки точності архітектур UNET і FCN при наборах даних, розмірами 256x256 та 512x512 пікселів. З графіку можна зробити висновок, що при наборі даних розміром 256 x 256 точність UNET архітектури значно вища (99%) ніж точність архітектури FCN(90%). При наборі даних розміром 512 x 512 пікселів точність перевірки архітектури UNET показала 96%, а точність архітектури FCN - 97%.

Рисунок 2.12 - Результати функції втрат UNET архітектури та FCN[35]

Остаточні результати функції втрат показано на рисунку 2.12. Блакитний колір показує значення втрат під час тренування, а помаранчевий під час тестування моделі. Аналогічно до результатів точності, UNET показав кращі результати з набором даних 256 x 256, а FCN – з набором даних 512 x 512 відповідно до результатів втрат.

На основі вище описаних даних, було прийнято рішення використати UNET модель, так як вона має досить високу точність. При створенні даних розміром 256x256 пікселів - оригінальні зображення будуть відмасштабовані до цього розміру, після чого зображення будуть подаватись на вхід нейронної мережі. Щодо метрик, було обрано метрики Binary cross entropy і Dice а точність нейромережі буде вимірюватись коефіцієнтом Dice.

2.5 Алгоритм методу класифікації дерев’яних виробів

Так як в основі даної роботи лежать процеси машинного навчання, то першою підсистемою буде система глибокого навчання за допомогою вищеописаних продуктів, а саме: TensorFlow та Keras, а весь процес навчання

буде проходити на базі Google Colaboratory, використовуючи для даних цілей GPU.

Задачами методу будуть процеси навчання нейромережі, використовуючи дані з структурованого набору даних (датасету), що містить в собі зображення необхідних об’єктів та їх масок, що будуть потім розпізнаватися вже в готовому продукті.

Алгоритм для реалізації цієї підсистеми буде виглядати наступним чином:

1. побудова набору даних (ручна розмітка зображень)
2. Створення записника в Google Colaboratory з усіма необхідними інструкціями для виконання процесу навчання.
3. Завантаження датасету на Google Drive для подальшого використання в процесі навчання.
4. побудова моделі нейронної мережі
5. Запуск процесу навчання.
6. Тестування нейронної мережі
7. Оцінка швидкості навчання та її аналіз.
8. Оцінка та аналіз точності навчання.

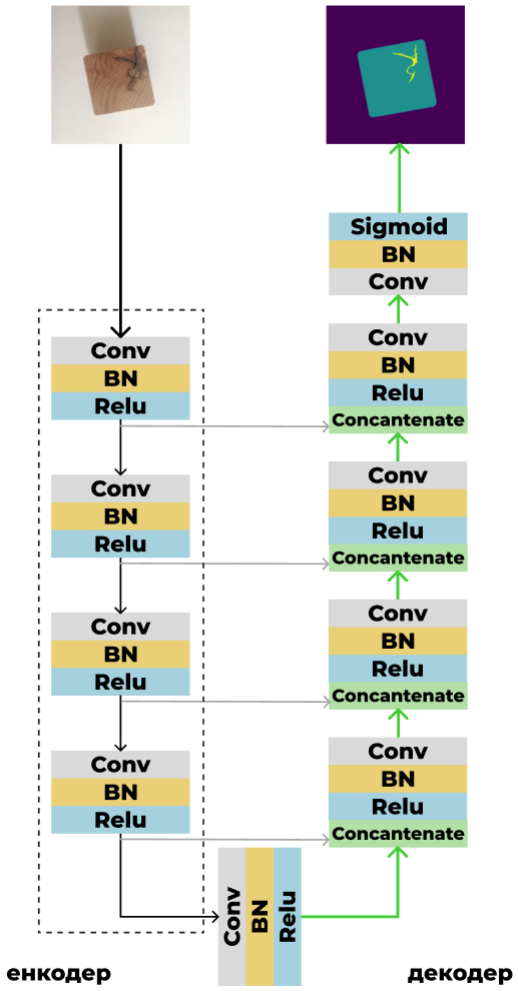


Рисунок 2.13 - приклад роботи методу класифікації дерев'яних виробів

З рисунку 2.13 видно роботу методу класифікації, а саме на вхід енкодеру подається зображення, а на виході отримуємо маску зображення. Алгоритм роботи методу зображений у Додатку А.

Висновки до розділу 2

В ході досліджень у другому розділі роботи, було отримано наступні основні результати:

1. Запропоновано модель для класифікації дерев’яних виробів, а саме використання архітектури автоенкодеру для побудови нейронної мережі.
2. Досліджено архітектури штучних нейронних мереж FCN, SegNet та U-Net, Проведено пошук та аналіз готових рішень а також досліджено типи нейронних мереж, а саме згорткову нейронну мережу і архітектуру UNET .
3. Проаналізовано метрики та функції втрат нейромереж, що дозволило більш якісно проводити процес навчання.
4. Проведено порівняльний аналіз архітектур нейромереж FCN I U-Net, який показав, що UNET нейромережа є на 9% точніша від FCNN для зображень 256x256 пікселів. Виходячи із зазначеного показника, архітектуру UNET більш доцільно обирати для вирішення задачі семантичної сегментації зображень при дефектування дерев’яних виробів.
5. Розроблено алгоритм методу класифікації дерев’яних виробів, який описує роботу методу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Структурна схема системи розпізнавання дефектів дерев’яних виробів

В ході роботи було побудовано дві структурні схеми систем для виявлення дефектів на дерев'яних виробах: для навчання мережі (рисунок . 3.1) та для її використання (рисунок . 3.2).

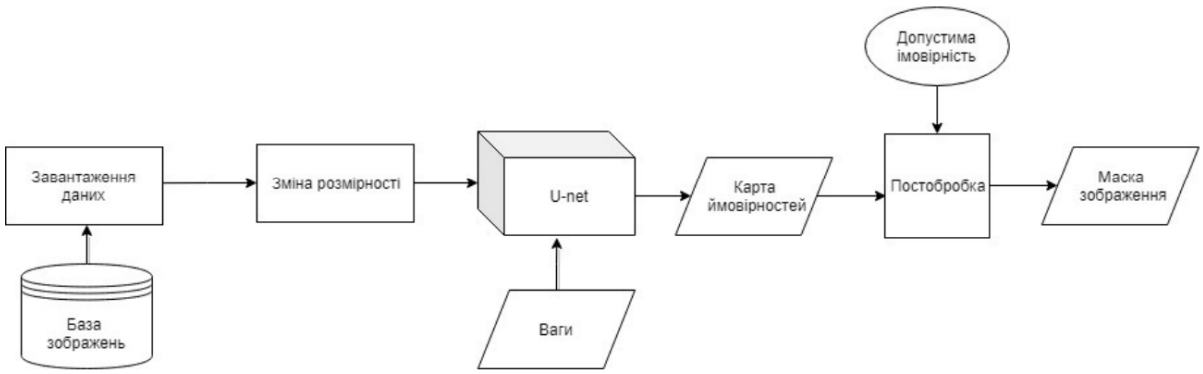


Рисунок 3.1 - модуль навчання моделі

Для навчання нейронної мережі модуль завантаження витягує зображення та відповідні маски зі сховища файлів. Потім зображення та маска стискаються до заданого розміру та передаються на вхід модуля сегментації даних. Модуль Data Segmentation виконує однакові операції по модифікації зображення і маски. Потім зображення надсилається на вхід мережі на основі архітектури U-net. Мережа обробляє зображення та виводить маску у вигляді карти ймовірностей. Згенерована маска порівнюється з існуючою маскою, обчислюється помилка, і на її основі мережа змінює ваги. Схема роботи методу зображена у Додатку Б.

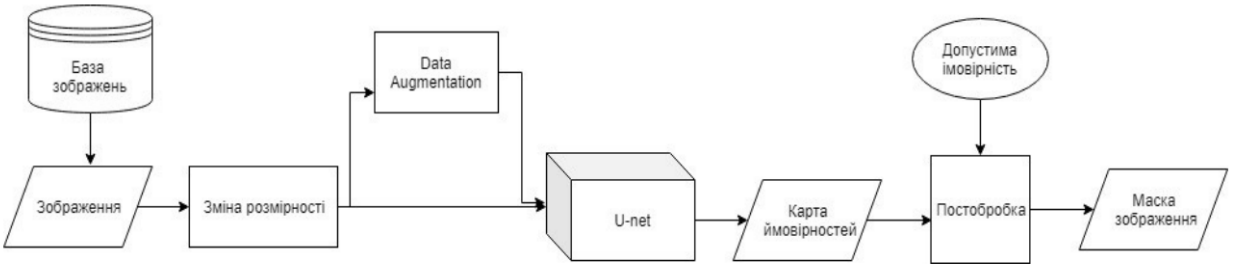


Рисунок 3.2 - модуль передбачень модель

У системі прогнозування створюються маски сегментації зображення, тому вводяться лише зображення. Результатом роботи мережі є карта ймовірностей, де пікселі зображення належать сегменту пошуку. Для отримання маски модуль постобробки порівнює кожне значення карти ймовірностей з допустимими значеннями. Якщо значення більше, піксель вважається належним об'єкту інтересу. Результатом є маска сегментації зображення.

3.2 Опис засобів розробки

3.2.1 Підготовка датасету

В цілому задача розпізнавання образів на зображеннях можна розділити по групах: класифікація (коли потрібно визначити якому з класу належить об’єкт), локалізація (коли потрібно визначити місцезнаходження об’єкта на зображенні та найти його координати), сегментація (коли потрібно визначити яка частина зображення належить одному чи іншому класу).

Набір даних для роботи системи класифікації було сформовано вручну з допомогою фотоапарату CODAK-25783. За допомогою сервісу анотацій зображень Supervisely проведено розмітку отриманих раніше зображеннях а саме: виділення меж виробу і сегментація дефектів. Було присвоєно два класи:

wood cube (помаранчеві лінії) і crack (червоні лінії). В кінцевому результаті було отримано датасет із зображеннями виробів, а також з масками анотованих зображень у форматі PNG (рисунок .1).

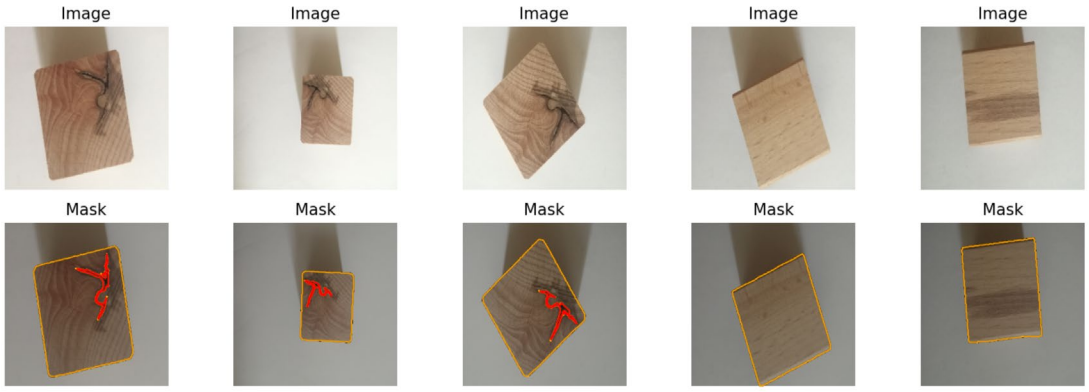


Рисунок 3.3 - Приклад розмітки зображень

З рисунку 3.3 показано приклади зображень і їхні маски які були вручну розмічені за допомогою сервісу Supervisely.

Існує декілька способів обробки зображень нейроннимим мережами. Наприклад, якщо потрібно обробити велике зображення у високій якості то для його обробки потрібно розділити зображення на менші області і оброботити кожну з них, при цьому збільшиться час для обробки цього зображення. У випадку якщо потрібно обробити невелике зображення і при цьому не важлива деталізація отриманого результату, то можна зменшити зображення до потрібного розміру і пропустити його через нейронну мережу. У цій роботі буде розглянуто другий підхід – оригінальні зображення будуть відмасштабовані до розміру 256x256 пікселів, після чого зображення будуть подаватись на вхід нейронної мережі, а на виході отримувати багатоканальні сегментовані зображення.

3.2.2 Побудова архітектури нейронної мережі

Щодо архітектури нейронної мережі, то вона складається зі звужуючої першої частини і розширяючої другої частини (див. рисунок 3.4). Особливістю згорткової мережі UNET є застосування skip connections [14] (див. рисунок 3.4). Ця техніка часто застосовується у глибоких нейромережах і позитивно впливає на збіжності моделі. Завдяки цим міжшаровим з'єднанням інформація низького рівня спільно використовується вхідними та вихідними шарами, а в архітектурі UNET ці зв'язки використовуються для того, щоб передати ознаки отримані в енкодері в декодер. Це допомагає відновити просторову інформацію, яка була втрачена в результаті операції згортки.

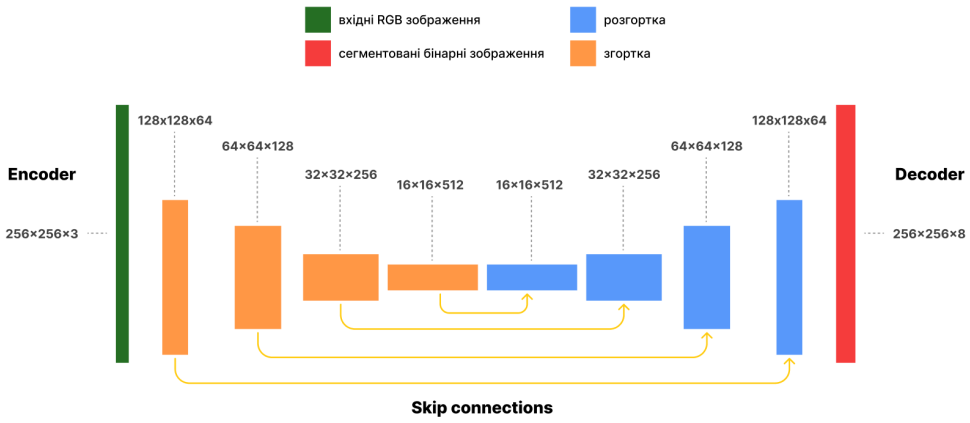


Рисунок 3.4 - Архітектура UNET нейронної мережі

Блок згортки складається з комбінації згорткових шарів, пакетної нормалізації та активаційної функції LeakyRelu (див. рисунок 3.5). Друга частина нейромережі складається з комбінації Transpose згорткових шарів, шарів пакетної нормалізації, dropout шарів і активаційної функції Relu (рисунок 3.5). Dropout шари є інструметами для запобігання перенавчання мережі. Шари пакетної нормалізації застосовуються для того щоб підвищити продуктивність мережі і робить роботу мережі стабільною. Згорткові Transpose шари застосовуються для того щоб провести протилежну операцію згортки і збільшити розмірність вихідних даних.

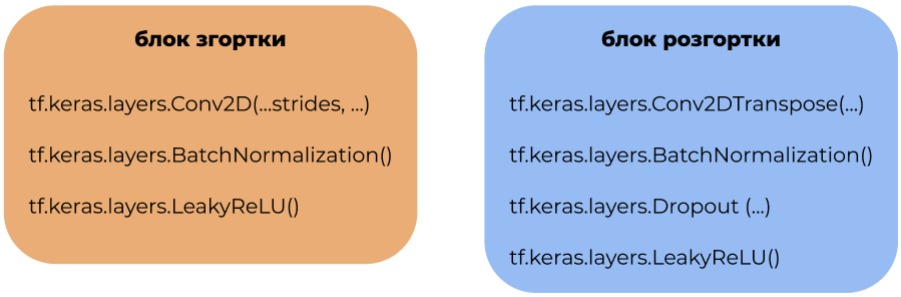


Рисунок 3.5 - Блок згортки і розготрки архітектури

У запропонованановому способу, пулінг шари для зменшненшення розмірності у оригінальній архітектурі UNET будуть замнінені на параметр strides у самому згортковому шарі. Побудована архітектура для класифікації дерев'яних виробів зображена у Додатку В.

3.2.3 Налаштування конвеєру даних

Щоб використовувати дані з набору даних, необхідно створити конвеєр, який готує дані для введення моделі. Основними вимогами цього конвеєра є можливість підготовки даних у групах (пакети англійською мовою), можливість відправлення цих даних у декілька моделей одночасно та поділ набору даних на дані для навчання моделі та дані для тестування.

Для цього завдання обсяг даних для однієї групи еквівалентний 16 парам зображень і відповідним сегментованим маскам. 16 ядер графічного процесора, які обслуговують 8 моделей, що навчаються, повинні обслуговувати 16 * 16 = 256 пар зображень на крок градієнтного спуску. Набір даних поділяється під час конвеєра, тому 80% даних використовуються для навчання моделі, а 20% для оцінки моделі. Поділ між 95% і 5% може призвести до більш неточних результатів оцінки моделі, навіть якщо є більше даних для навчання моделі. Це

число еквівалентне тисячам зображень через невелику кількість зображень (1560 зображень), і такий поділ був розумним. Розрахуйте кількість кроків, які модель робить за одну епоху, використовуючи таку формулу (3.1):

$$steps = \frac{N_{images}}{N_{minibatch} * N_{files} * (T * N_{train\ files} + (1 - T) * N_{test\ files})} \quad (3.1)$$

Де N_{files} – кількість зображень з яких складається датасет;

N_{train} – кількість зображень для тренування моделі;

$N_{test\ files}$ – кількість зображень для тестування моделі;

N_{images} – загальна кількість зображень;

$N_{minibatch}$ – кількість зображень у міні групі.

У цій формулі steps – це кількість кроків навчання чи тестування. T - параметр, що дорівнює 0, якщо на даному етапі модель тестується, і 1, якщо на даному етапі вона навчається.

Розробнику доступний інструмент за допомогою якого можна організувати складні вхідні потоки даних. Доступні методи такі як: послідовна обробка, паралельна обробка, кешування, передвибірка, векторізація даних та інші. Набір цих інструментів дозволяє створити оптимізований вхідний конвеєр даних. Таким чином можна балансувати між споживанням ресурсів процесора, оперативної пам’яті або сховища. На рисунку 3.6 показано схематично налаштування конвеєру даних.

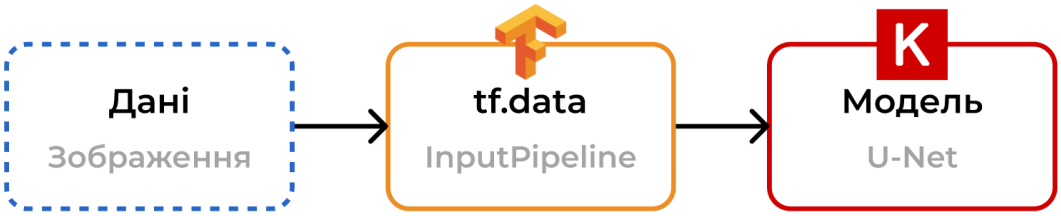


Рисунок 3.6 - налаштування конвеєру даних

Спочатку оголошуються змінні, які задають число класу – всі об’єкти які присутні на сцені і додатковий клас для позначення заднього фону. Потім задаються кольори класів за допомогою яких будуть підсвічуватись об’єкти на зображеннях, фіксовані розмори вхідного зображення, які будуть використовуватись у нейронній мережі, і розмір вихідного зображення яке буде формуватись після фінальної обробки даних.

Далі оголошуються функції load_images та augmentate_images. load_images приймає на вхід шлях до масок та зображень.

За допомогою функцій у Tensorflow завантажується файл з зображенням та перетворюється у необхідний формат, змінюється розмір та нормалізуються значення. Аналогічним чином завантажується маска зображення.

Зображення з масками представляють собою одновимірні масиви, у яких регіони з об’єктами відповідних класів помічені різними значеннями від 1 до 2. В результаті формуються бінарні зображення, розмірність яких відповідає кількості класів об’єктів.

Функція augmentate_images приймає трьохканалні RGB зображення і багатоканальну маску, яка була сформована на попередньому етапі, і проводить аугментацію даних. це дозволить збільшити варіативність вибірки, що в кінцевому результаті позитивно вплине на якість роботи нейронної мережі. До зображень застосовувалася комбінація методів обробки зображень:

- обертання;
- зсув;
- обрізання;
- масштабування;
- відображення щодо осі.

Комбінація методів та ступінь інтенсивності в процесі обробки зображень вибираються випадковим чином. Завдання сегментації вимагає, щоб зображення та його маска відповідали один одному, тому для зображення та маски використовуються один і той самий метод та інтенсивність. Приклад аугментації зображень показаний на рисунку 3.7.

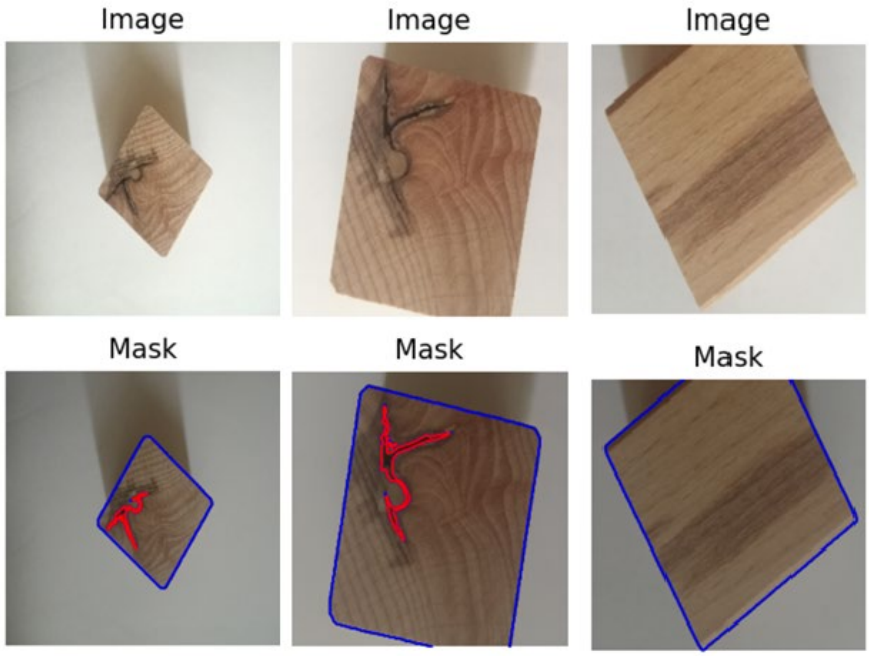


Рисунок 3.7 - Приклад аугментації зображень

Застосовується функція central_cropp, яка витягає центральний фрагмент з випадковим значенням масштабу, і функція random_flip, що виконує випадкове відображення зображень по горизонталі. В кінці цього конвеєру

встановлюється вихідний розмір зображень і масок. Після цього як функції обробки були визначеними, йде загрузка даних з диску. За допомогою функцій tensorflow формуються набори даних з зображень та масок, та об’єднуються для паралельної обробки.

Прочитавши пару зображень та маску, далі йде перетворення їх у тензор розміру, що відповідає збереженому розміру. Тобто 256x256x3 для зображення і 256x256xC для масок, C - це кількість класів для сегментації. Кількість паралельних обробок набору даних дорівнює кількості ядер GPU. За допомогою функції load_images завантажуються дані в пам’ять. Після цього , за допомогою функції repeat збільшується об’єм даних у 60 разів простим копіюванням. І до всього набору даних застосовується функція augmentate_images. В результаті кожне зображення у вибірці буде унікальним. Таким чином було штучно збільшено невеликий об’єм даних, розмічених вручну, у 60 разів. Після цього за допомогою функцій take І skip було розділено набір даних на навчальний і тестовий та кешовано їх у пам'яті. Щоб витрачати час на підготовку наступної групи зображень після навчання попереднього зображення, наступна група зображень для навчання моделі готується заздалегідь.

3.2.4 Функції втрат та метрики

Алгоритми глибокого навчання використовують метод стохастичного градієнтного спуску (рисунок 3.8). Це метод для знаходження мінімума або максимума цільової функції, а цільова функція це математичне представлення того наскільки добре нейромережа виконує свою роботу.

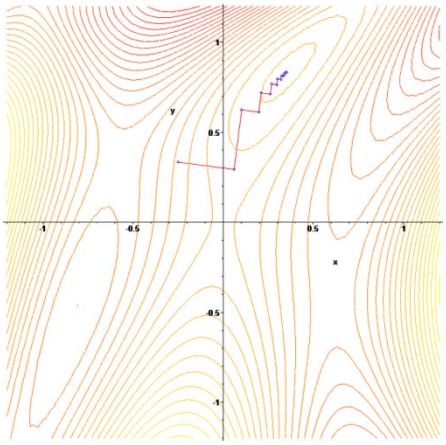


Рисунок 3.8 - Граф градієнтного спуску

Для вирішення задач сегментації можуть застосовуватись різні функції або їх комбінації. Одними з популярних є функції Binary cross entropy і Dice (рисунки 3.9, 3.10)[15]. Якщо нейронна мережа дає відповідь близький до правильного то значення функції втрат буде малим. В іншому випадку значення цільової функції буде великим. Правильний вибір функції втрат впливає на швидкість і якість навчання моделі. Для системи розпізнавання дефектів на дерев'яних виробах було застосовано комбінацію функцій Binary cross entropy і Dice.

Binary cross entropy (3.2):

$$-(y\log(p) + (1-y)\log(1-p) \tag{3.2}$$

де y – реальне значення класу 0 або 1;

p – передбачене значення класу від 0 до 1.

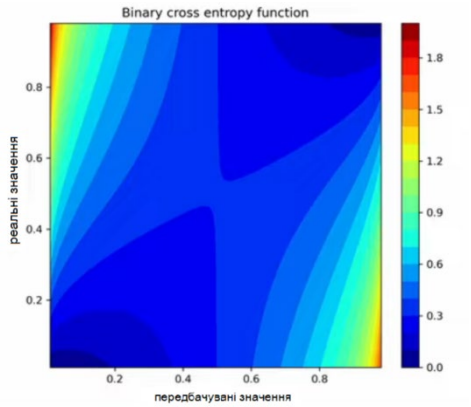


Рисунок 3.9 - Граф функції втрат Binary cross entropy

Dice (3.3):

$$1 - \frac{2yp + 1}{y + p + 1} \tag{3.3}$$

де y – реальне значення класу 0 або 1;

p – передбачене значення класу від 0 до 1.

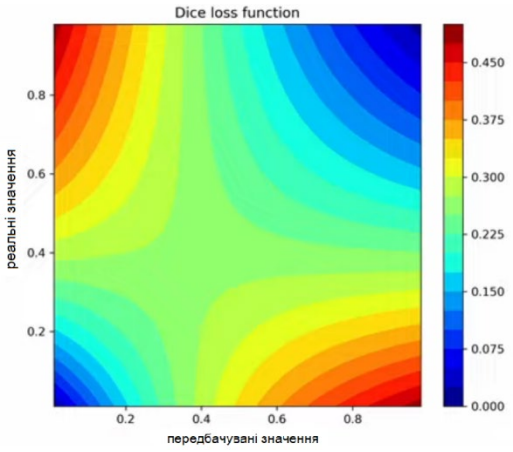


Рисунок 3.10 - Граф функції втрат Dice

Функція Binary cross entropy дає високу збіжність моделі. Навчання проходить більш стабільно при збалансованому наборі даних. Функція Dice показує якісні результати в задачах сегментації, але показує погану збіжність

моделі. Функція втрат, яка складається з комбінацій функцій, матиме наступний вигляд (3.4) (рисунок 3.11):

$$loss = Binary\ cross\ entropy + 0.3 * Dice \quad (3.4)$$

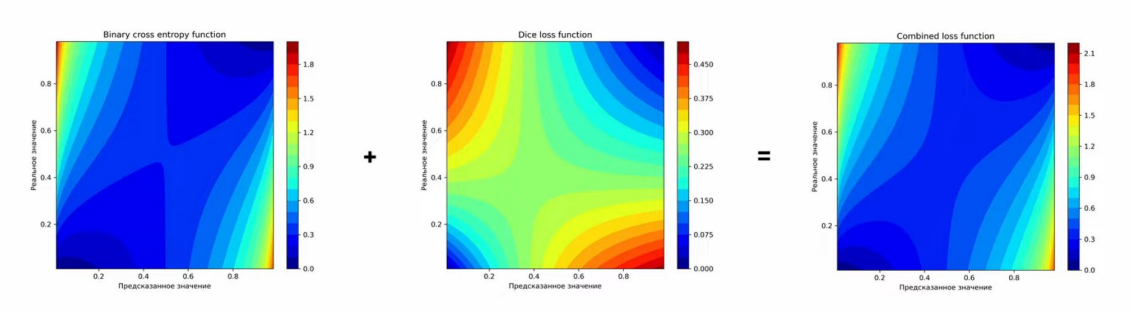


Рисунок 3.11 - граф комбінованої функції втрат

Така комбінація функцій, допоможе нівелювати погано збалансований набір вхідних даних, який був побудований для цієї моделі, і досягти хороших результатів навчання. Для того, щоб оцінити точність роботи нейронної мережі використовується коефіцієнт Dice.

3.2.5 Реалізація методу

Для створення моделей у фреймворку TensorFlow використовуються два основних методи: послідовний (Sequential) і функціональний (Functional). Перший слід використовувати для моделей, у яких кожен шар має один вхідний тензор та один вихідний тензор (Рисунок 3.11). Зазвичай, це прості моделі без складної архітектури. Цей метод недоречно використовувати в таких випадках:

1. Якщо шар у моделі має кілька вхідних та/або вихідних тензорів.
2. Якщо модель отримує кілька вхідних значень або якщо вихід моделі повертає кілька вихідних значень.
3. Якщомодель має нелінійну топологію.
4. Якщо потрібно розділити один шар між кількома моделями одночасно.

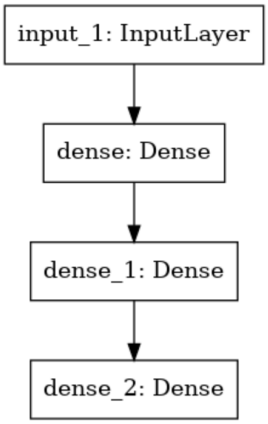


Рисунок 3.12 - Приклад послідовнної архітектури

Наступна методика, яка буде використана у цій роботі, є функціональний метод (Рисунок 3.12). Це дозволяє створити гнучкішу модель, ніж при використанні послідовного методу. Ідея функціонального стилю полягає в тому, що модель глибокого навчання може бути представлена у вигляді орієнтованого ациклічного графа його шарів. Таким чином, можливо самостійно визначити такі графіки, використовуючи функціональний стиль. Це робить недоліки попереднього методу можливими у цьому стилі.

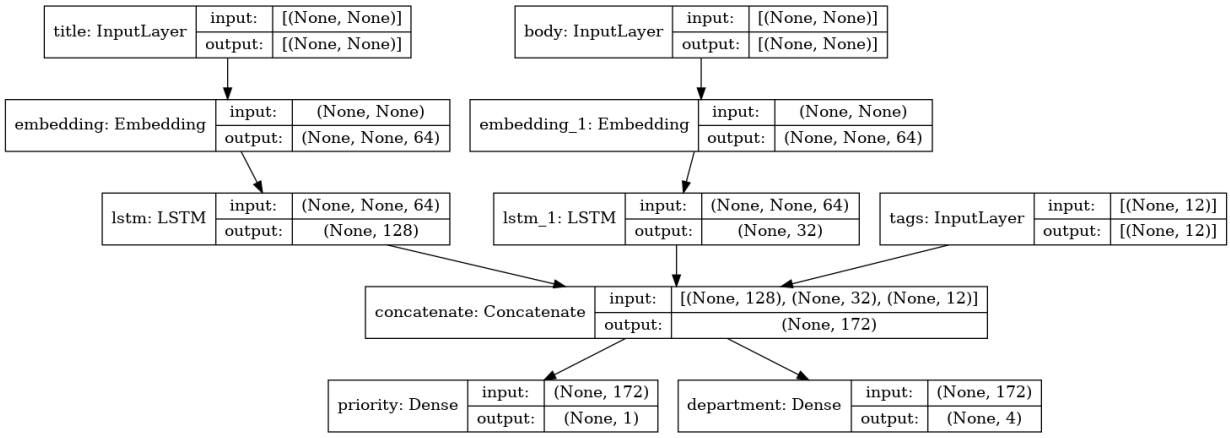


Рисунок 3.13 - Приклад функціональої архітектури

Бібліотека Keras, яка входить у склад Tensorflow, містить у собі необхідні модулі для реалізації нейромережі мовою Python. Функція input_layer задає вхідний шар нейронної мережі і встановлює розмір вхідних даних

```
def input_layer():
    return tf.keras.layers.Input(shape=SAMPLE_SIZE + (3,))
```

Функція downsample_block описує блоки, які формує енкодер. Вона задає метод ініціалізації вагових коефіцієнтів, включає згортковий шар, добавляє шар пакетної нормалізації і встановлює активаційну функцію LeakyReLU.

Вона має наступну сигнатуру:

1. Ініціалізатор (initializer) – вхідний тензор, який використовується для ініціалізації наступних шарів та автоматичної установки розміру всіх наступних вхідних та вихідних тензорів.
2. Фільтри (filters) - кількість фільтрів у згортковій нейронній мережі. Також цей параметр відповідає розмір четвертої осі вихідного тензора.
3. Розмір (size) – обсяг фільтра в пікселях. За замовчуванням цей параметр дорівнює 3.
4. Батч нормалізація (batch_norm) - логічна змінна, що визначає, чи слід виконувати нормалізацію вихідного тензора. За замовчуванням цей параметр має значення False.
5. Кроки (strides) – кількість кроків (пікселів) між обчисленнями фільтра згортки. Це кортеж із двох цілих чисел або одного цілого числа, що вказує кроки по горизонталі та вертикалі.

Функція upsample_block помагає формувати декодер нейронної мережі, тільки замість функції згортки виконується протилежна їй функція transpose. Також в цьому блоці передбачена можливість додавання dropout шарів.

1. Ініціалізатор (initializer) – вхідний тензор, який використовується для ініціалізації наступних шарів та автоматичної установки розміру всіх наступних вхідних та вихідних тензорів.

2. Фільтри (filters) - кількість фільтрів у згортковій нейронній мережі. Також цей параметр відповідає розмір четвертої осі вихідного тензора.

3. Розмір (size) – обсяг фільтра в пікселях. За замовчуванням цей параметр дорівнює 3.dropout – логічна змінна, що вказує, чи використовувати шар dropout, щоб запобігти переобладнанню моделі. За замовчуванням значення false.

4. Кроки (strides) – кількість кроків (пікселів) між обчисленнями фільтра згортки. Це кортеж із двох цілих чисел або одного цілого числа, що вказує кроки по горизонталі та вертикалі

5. transpose — логічна змінна, яка визначає, як зображення буде збільшено вздовж перших двох осей (висота і ширина). За замовчуванням цей параметр має значення False.

Остання функція output_layer задає вихідний шар, розмірність якого задає кількість класів об’єктів на зображенні і використовує сигмоїдну активаційну функцію. Після цього, за допомогою цих функцій, створюється масив downsample_stack, який представляє енкодер, і upsample_stack, який представляє декодер. Така структура даних дозволяє реалізувати skip connections

За допомогою наступних циклів (Рисунок 3.13), з’єднуються блоки енкодера та декодера і реалізуються міжшарові з’єднання за допомогою операції конкантенація.

```
for block in downsample_stack:
    x = block(x)
    downsample_skips.append(x)

downsample_skips = reversed(downsample_skips[:-1])

for up_block, down_block in zip(upsample_stack, downsample_skips):
    x = up_block(x)
    x = tf.keras.layers.Concatenate()([x, down_block])

out_layer = out_layer(x)

unet_like = tf.keras.Model(inputs=inp_layer, outputs=out_layer)
```

Рисунок 3.13 код циклів, за допомогою яких з’єднуються локи енкодера та декодера

В результаті створюється модель, в якій вказано вхідні і вихідні шари.

Код програмної реалізації зображений у Додатку Г.

3.3 Експериментальні дослідження методу

В результаті тривалих експериментів та пошуку оптимальної архітектури моделі, була розроблена модель та натренована на датасеті на 30 епохах, такої кількості епох тренування достатньо оскільки розмір датасету невеликий. На Рисунок 3.14 зображено графік точності моделі.

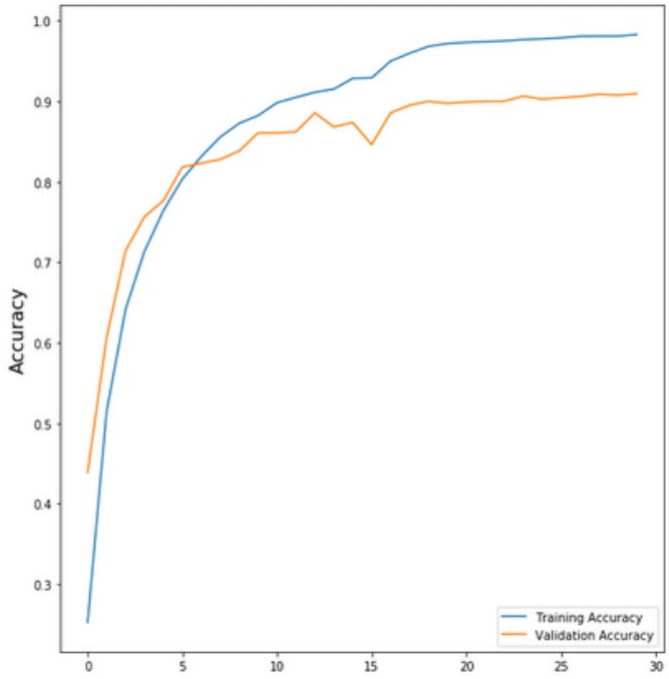


Рисунок 3.14 - графік точності моделі. Жовті лінія показує значення тестування моделі, синя лінія – тренування

В результаті навчання моделі, точність, яка визначається коефіцієнтом Dice, досягла відмітки 98%, а функція втрат 0.01 – що вважаєть хорошим результатом. Модель добре навчена, а отже результат сегментації буде чітким на тестових зображеннях.

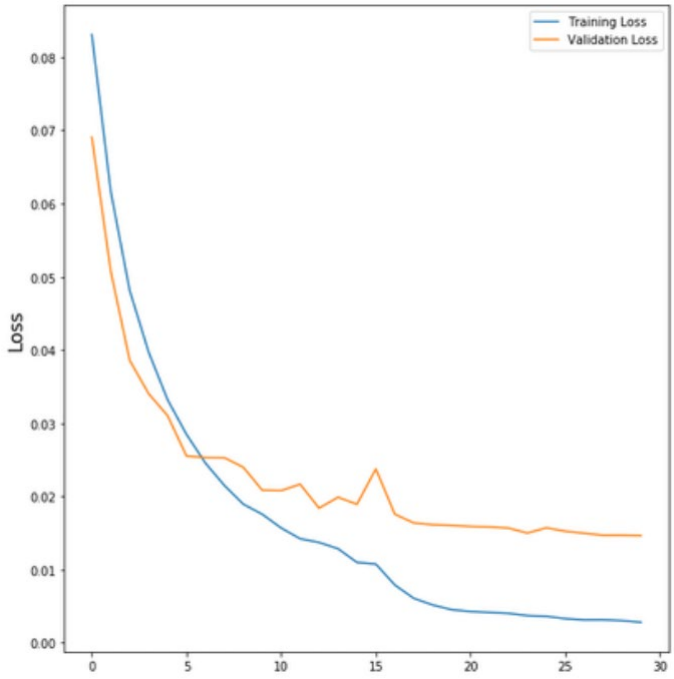


Рисунок 3.15 - графік втрат моделі. Жовті лінія показє значення тестування моделі, синя лінія – тренування.

Щодо результатів роботи нейромережі в режимі передбачень, було обрано 6 зображень з різним ступенем пошкодження на поверхнях деталей. Результати роботи зображено на рисунку 3.15.

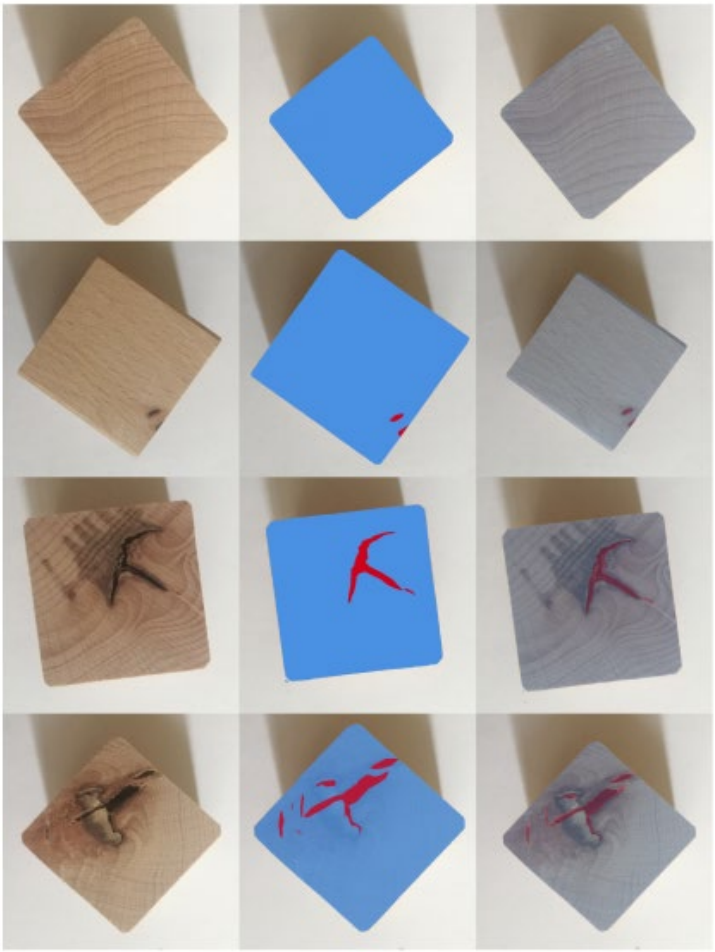


Рисунок 3.15 - приклад роботи моделі в режимі передачення. Зліва вхідне зображення, по центру маска зображення, справа – результат передбачення моделі.

З даних пезультатів можна зробити висновок, що навчена модель працює коректно, даже якщо на зображенні є тіні від деталей, нейронна мережа анотує дефекти, згідно тренувальних масок. Для покращення роботи методу на виробництві потрібно забезпечити хороше освітлення деталей, щоб зменшити ризики помилкового анотування власних тіней деталей, а також камера, яка фотографуватиме дерев’яні деталі, повина бути зафіксованою на певній висоті.

Висновки до розділу 3

В ході роботи дипломної роботи, у цьому розділі було виконано наступні завдання:

1. Створено структурні схеми системи розпізнавання дефектів дерев’яних виробів, які схематично показують рооту моделі під час тренування і в режимі передбачення.

2. Описано засоби розробки методу, побудовано архітектуру нейронної мережі та модифіковано за допомогоюкоефіцієнта Dice, яка вимірюватиме точність навченої моделі, і функцією втрат яка складається з комбінації функцій Binary cross entropy I Dice.

3. Описано процеси підготовки датасету, налаштування конвеєру даних та реалізації самого методу класифікації дерев’яних виробів, що дозволило побудувати алгоритм попередньої обробки даних.

4. Проаналізовано результати роботи методу, візуалізовано графіки навчання моделі та результати передбачення, які налядно демострують процес роботи моделі штучної нейронної мережі.

ВИСНОВКИ

У першому розділі кваліфікаційної роботи розглядаються основні терміни та методи в задачі класифікації дерев’яних деталей. Розглянуто основні терміни нейронної мережі, яка використовується для реалізації методу класифікації виробів з деревини. Він також окреслює основні проблеми, які можна вирішити за допомогою комп’ютерного зору, і підсумовує типи, необхідні для вирішення проблем сегментації дефектів дерева. Описано процес вибору мови програмування для завдання, а також розглянуто сильні та слабкі сторони різних фреймворків для створення нейронних мереж та їх навчання. Наведено переваги всіх можливих апаратних платформ для навчання нейронної мережі.

У другому розділі кваліфікаційної роботи описано запропонований метод класифікації виробів з деревини. Розглянуто архітектури нейронних мереж для виконання завдань семантичної сегментації та коротко підсумовано розвиток подібних архітектур із переліком переваг і недоліків кожної архітектури. Розглянуто метрики, які використано для оцінки продуктивності нейронних мереж. Також виконано порівняльний аналіз двох архітектур нейромереж, що дозволило встановити тип, необхідний для вирішення задачі сегментації дефектів дерев’яних виробів.

В третьому розділі дипломного проєкту практично реалізовано та проаналізовано засоби розробки методу класифікації дерев’яних виробів. Під час практичної реалізації створено датасет та створена архітектура нейронної мережі для розпізнавання дефектів дерев’яних виробів. Створено спеціалізовані метрики та функції втрат для тренування моделі спеціально для вирішення задачі семантичної сегментації зображень. У роботі детально описано налаштування конвеєру даних для роботи методу, а також здійснено програмну реалізацію нейронної мережі. Після проведеного аналізу збережених метрик та функції втрат моделі під час її тренування з побудовою графіків здійснено

кінцевий аналіз передбачень моделі на прикладі зображень із тестового набору даних.

В результаті виконання цього дипломного проекту була успішно створена та навчена модель семантичної сегментації дефектів деревини.

Проведений аналіз архітектур UNET і FCN показав, що UNET нейромережа є на 9% точніша від FCNN для зображень 256x256 пікселів. Виходячи із зазначеного показника, архітектуру UNET більш доцільно обирати для вирішення задачі семантичної сегментації зображень при дефектування дерев’яних виробів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ke, Z.N.; Zhao, Q.J.; Huang, C.H.; Ai, P.; Yi, J.G. Detection of wood surface defects based on particle swarm-genetic hybrid algorithm. In Proceedings of the 2016 International Conference on Audio, Language and Image Processing-Proceedings, Shanghai, China, 11–12 July 2016.

2. Hashim, U.R.; Hashim, S.Z.; Muda, A.K. Automated vision inspection of timber surface defect: A review. J. Teknol. 2015, 127–135.

3. Wang, L.; Qi, W.; Wu, J.; Hou, W. Recognizing the patterns of wood inner defects based on wavelet neuraIn Proceedings of thel networks. In Proceedings of the IEEE International Conference on Automation and Logistics, Jinan, China, 18–21 August 2007; pp. 1719–1724.

4. Du, X.; Li, J.; Feng, H.; Chen, S. Image reconstruction of internal defects in wood based on segmented propagation rays of stress waves. Appl. Sci. 2018, 8, 1778.

5. Peng, Z.; Yue, L.; Xiao, N. Simultaneous Wood Defect and Species Detection with 3D Laser Scanning Scheme. Int. J. Opt. 2016, 2016, 1–6.

6. Wang, Q.; Liu, X.E.; Yang, S. Predicting density and moisture content of Populus xiangchengensis and Phyllostachys edulis using the X-ray computed tomography technique. For. Prod. J. 2020, 70, 193–199.

7. França, F.J.N.; França, T.S.F.A.; Seale, R.D.; Shmulsky, R. Nondestructive evaluation of 2 by 8 and 2 by 10 southern pine dimensional lumber. For. Prod. J. 2020, 70, 79–87.

8. Fang, Y.; Lin, L.; Feng, H.; Lu, Z.; Emms, G.W. Review of the use of air-coupled ultrasonic technologies for nondestructive testing of wood and wood products. Comput. Electron. Agric. 2017, 137, 79–87.

9. Li, X.; Qian, W.; Cheng, L.; Chang, L. A Coupling Model Based on Grey Relational Analysis and Stepwise Discriminant Analysis for Wood Defect Area Identification by Stress Wave. BioResources 2020, 15, 1171–1186.

10. Jiang, S.; Zhou, Z.; Wang, K. Literature review and comparison of wood testing methods and research on application technology of computer 3D scanning detection. ACM Int. Conf. Proc. Ser. 2019, 418–422.

11. DeVallance, D.B.; Funck, J.W.; Reeb, J.E. Evaluation of laminated veneer lumber tensile strength using optical scanning and combined optical-ultrasonic techniques. Wood Fiber Sci. 2011, 43, 169–179.

12. Razmjooy, N.; Mousavi, B.S.; Soleymani, F. A real-time mathematical computer method for potato inspection using machine vision. Comput. Math. Appl. 2012.

13. Boardman, B.E.; Senft, J.F.; McCabe, G.P.; Ladisch, C.M. Colorimetric analysis in grading black walnut veneer. Wood Fiber Sci. 2007, 24, 99–107.

14. Momin, M.A.; Yamamoto, K.; Miyamoto, M.; Kondo, N.; Grift, T. Machine vision based soybean quality evaluation. Comput. Electron. Agric. 2017.

15. Dawood, T.; Zhu, Z.; Zayed, T. Machine vision-based model for spalling detection and quantification in subway networks. Autom. Constr. 2017.

16. Yang, F.; Wang, Y.; Wang, S.; Cheng, Y. Wood Veneer Defect Detection System Based on Machine Vision. Adv. Comput. Sci. Res. 2018, 86, 413–418.

17. Park, J.K.; Kwon, B.K.; Park, J.H.; Kang, D.J. Machine learning-based imaging system for surface defect inspection. Int. J. Precis. Eng. Manuf. Green Technol. 2016, 3, 303–310.

18. Jung, S.Y.; Tsai, Y.H.; Chiu, W.Y.; Hu, J.S.; Sun, C.T. Defect detection on randomly textured surfaces by convolutional neural networks. In Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Auckland, New Zealand, 9 July 2018; pp. 1456–1461.

19. He, T.; Liu, Y.; Xu, C.; Zhou, X.; Hu, Z.; Fan, J. A fully convolutional neural network for wood defect location and identification. IEEE Access 2019, 7, 123453–123462.

20. Urbonas, A.; Raudonis, V.; Maskeliunas, R.; Damaševičius, R. Automated identification of wood veneer surface defects using faster region-based convolutional neural network with data augmentation and transfer learning. Appl. Sci. 2019, 9, 4898.

21. Kaiming, H.; Georgia, G.; Piotr, D.; Ross, G. Mask R-CNN. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.

22. Huang, H.; Wei, Z.; Yao, L. A novel approach to component assembly inspection based on mask R-CNN and support vector machines. Information 2019, 10, 282.

23. Kai, H.; Baojin, W.; Yi, S.; Jieru, G.; Yi, C. Defect Identification Method for Poplar Veneer Based on Progressive Growing Generated Adversarial Network and MASK R-CNN Model. Bioresources 2018, 15, 3041–3052.

24. Chiao, J.Y.; Chen, K.Y.; Liao, K.Y.K.; Hsieh, P.H.; Zhang, G.; Huang, T.C. Detection and classification the breast tumors using mask R-CNN on sonograms. Medicine 2019.

25. Pham, H.; Guan, M.Y.; Zoph, B.; Le, Q.V.; Dean, J. Efficient Neural Architecture Search via parameter Sharing. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.

26. Zoph, B.; Le, Q.V. Neural architecture search with reinforcement learning. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.

27. Liu, C.; Chen, L.C.; Schroff, F.; Adam, H.; Hua, W.; Yuille, A.L.; Fei-Fei, L. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 18–20 June 2019.

28. Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; Le, Q.V. Mnasnet: Platform-aware neural architecture search for mobile. In

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 18–20 June 2019.

29. Xie, S.; Kirillov, A.; Girshick, R.; He, K. Exploring randomly wired neural networks for image recognition. In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Korea, 27–28 October 2019.

30. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Regularized Evolution for Image Classifier Architecture Search. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019.

31. International Journal of Engineering Applied Sciences and Technology, 2020 Vol. 4, Issue 12, ISSN No. 2455-2143, Pages 310-316

32. Olaf Ronneberger, Philipp Fischer, Thomas Brox U-Net: Convolutional Networks for Biomedical Image Segmentation ArXiv, Germany, 18 May 2015. P. 1-9

33. Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, 10 Oct 2016, p. 1-14

34. Shruti Jadon, A survey of loss functions for semantic segmentation. IEEE, Via del Mar, Chile, 3 Sep 2020. P. 1-7.

35. Ozan Ozturk, Batuhan Sariturk, Dursun Zafer Seker Comparison of Fully Convolutional Networks (FCN) and UNET for Road Segmentation from High Resolution Imageries. International Journal of Environment and Geoinformatics (IJEGEO), 3 Dec 2020. P. 275-278

36. Nair Vinod, Hinton Geoffrey E. Rectified linear units improverestricted boltzmann machines // Proceedings of the 27th international conference on machine learning (ICML-10). — 2010. — P. 807–814.

37. Caesar H, Uijlings J, Ferrari V (2016) Region-based semantic segmentation with end-to-end training. In: ECCV

38. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR

39. Uijlings JRR, Van De Sande KEA, Gevers T, Smeulders AW (2013) Selective search for object recognition. Int J Comput Vis 104(2):154–171

40. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: NIPS

41. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: ICLR

42. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: CVPR

43. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: CVPR

44. Long J, Shelhamer E, Darrell T (2015) Fully convolutional networks for semantic segmentation. In: CVPR

45. Eigen D, Fergus R (2015) Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In: ICCV

46. Liu Y, Guo Y, Lew MS (2017) On the exploration of convolutional fusion networks for visual recognition. In: MMM

47. Keras documentation [електронний ресурс] - https://keras.io

48. Aurélien Géron Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

49. Rezatofighi, Hamid & Tsoi, Nathan & Gwak, JunYoung & Sadeghian, Amir & Reid, Ian & Savarese, Silvio. (2019). Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression.

50. Siddharth Sharma, Simone Sharma, activation functions in neural networks, International Journal of Engineering Applied Sciences and Technology, 2020 Vol. 4, Issue 12, ISSN No. 2455-2143, P. 310-316

51. A.D.Dongare, R.R.Kharde, Amit D.Kachare Introduction to Artificial Neural Network, International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 1, July 2012.P. 189-194.

52. Пальчик В.О., Коваль В.С. Дефектування дерев’яних виробів з використанням згорткових нейронних мереж. V Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі». 10 листопада 2022. Тернопіль. Україна - с.30

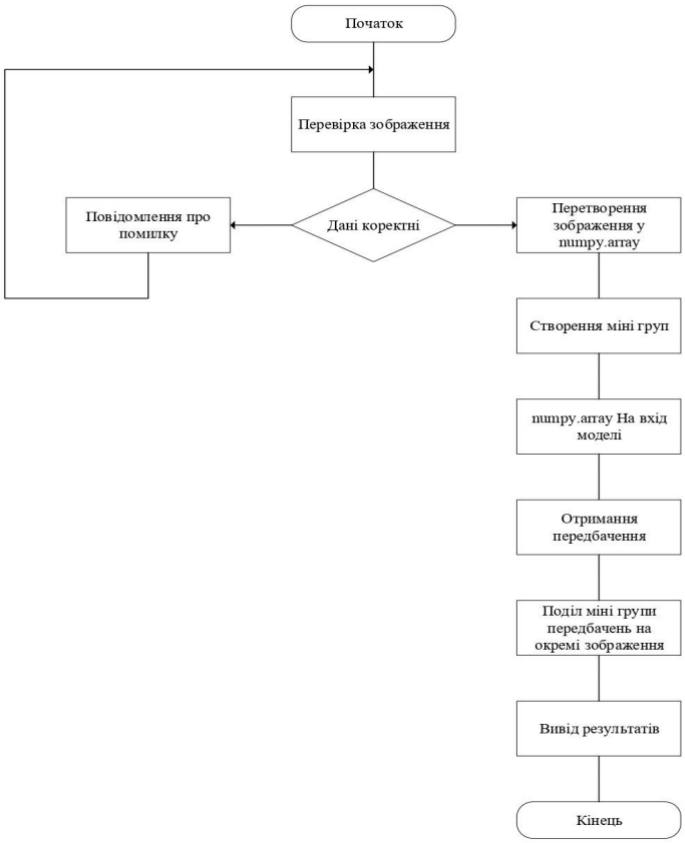
53. Пальчик В.О., Коваль В.С. Метод класифікаціх дерев’яних виробів на основі UNET штучних нейронних мереж. Школа-семінар молодих вчених і студентів "Комп’ютерні інформаційні технології (CIT’2022)", 29 листопада 2022р. Тернопіль. Україна.

54. Загальні рекомендації з підготовки, оформлення, захисту та оцінювання випускних кваліфікаційних робіт здобувачів вищої освіти першого «бакалаврського» і другого «магістерського» рівнів / За ред. доц. М.І. Шинкарика. Тернопіль: ТНЕУ, 2018. 67 с.

55. Комар М.П., Саченко А.О., Васильків Н.М. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за другим (магістерським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2021. 32 с."

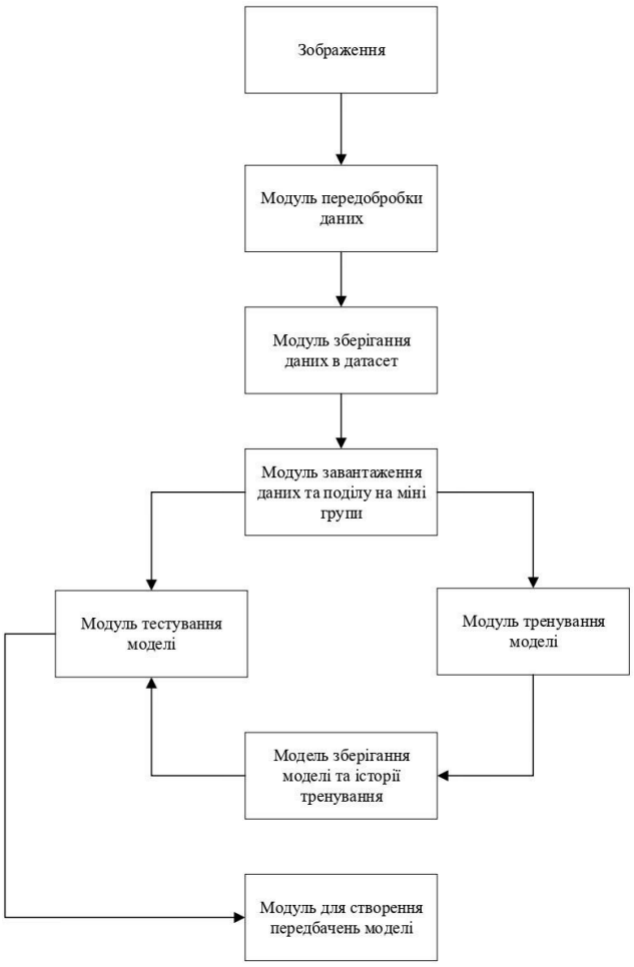
Додаток А

Алгоритм роботи методу класифікації дерев’яних виробів



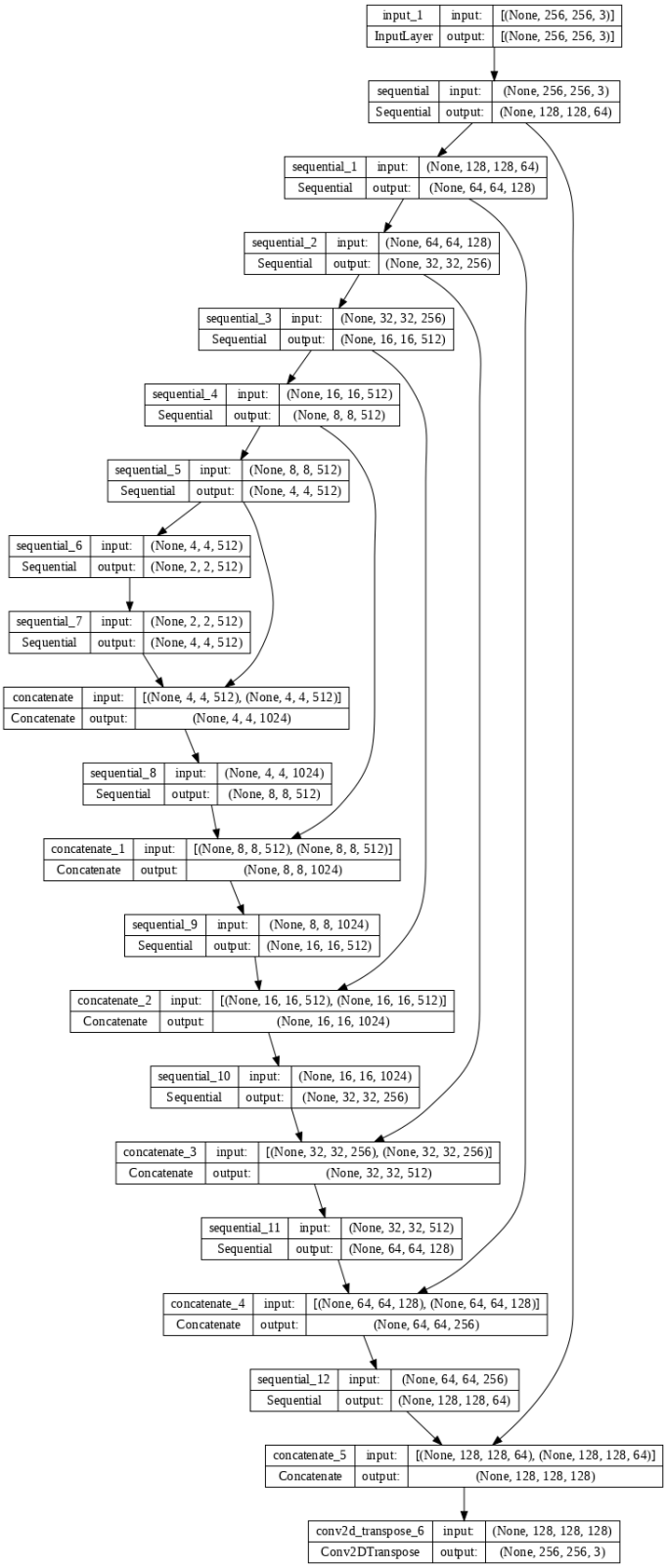
Додаток Б

Схема роботи методу класифікації дерев’яних виробів



Додаток В

Архітектура UNET для класифікації дерев’яних виробів



Додаток Г

Програмний код методу

```
!pip install tensorflow_addons

import os
import time
import glob
import shutil
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_addons as tfa

from skimage import measure
from skimage.io import imread, imsave, imshow
from skimage.transform import resize
from skimage.filters import gaussian
from skimage.morphology import dilation, disk
from skimage.draw import polygon, polygon_perimeter

print(f'Tensorflow version {tf.__version__}')
print(f'GPU is {"ON" if tf.config.list_physical_devices("GPU") else "O
FF" }')

CLASSES = 3

COLORS = ['black', 'blue', 'red']

SAMPLE_SIZE = (256, 256)

OUTPUT_SIZE = (1080, 1920)


def load_images(image, mask):
    image = tf.io.read_file(image)
    image = tf.io.decode_jpeg(image)
    image = tf.image.resize(image, OUTPUT_SIZE)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = image / 255.0

    mask = tf.io.read_file(mask)
    mask = tf.io.decode_png(mask)
    #mask = tf.image.rgb_to_grayscale(mask)
    mask = tf.image.resize(mask, OUTPUT_SIZE)
    mask = tf.image.convert_image_dtype(mask, tf.float32)
```

```
            masks = []

            for i in range(CLASSES):
                masks.append(tf.where(tf.equal(mask, float(i)), 1.0, 0.0))

            masks = tf.stack(masks, axis=2)
            masks = tf.reshape(masks, OUTPUT_SIZE + (CLASSES,))

            return image, masks

        def augmentate_images(image, masks):
            random_crop = tf.random.uniform((), 0.3, 1)
            image = tf.image.central_crop(image, random_crop)
            masks = tf.image.central_crop(masks, random_crop)

            random_flip = tf.random.uniform((), 0, 1)
            if random_flip >= 0.5:
                image = tf.image.flip_left_right(image)
                masks = tf.image.flip_left_right(masks)

            image = tf.image.resize(image, SAMPLE_SIZE)
            masks = tf.image.resize(masks, SAMPLE_SIZE)

            return image, masks


        os.mkdir('SemanticSegmentation/dataset/images_vk')
        os.mkdir('SemanticSegmentation/dataset/masks_vk')
        os.mkdir('SemanticSegmentation/videos/original_video_vk')
        os.mkdir('SemanticSegmentation/videos/res')
        os.mkdir('SemanticSegmentation/videos/res_incorrect')


        images = sorted(glob.glob('SemanticSegmentation/dataset/images_vk/*.jp
g'))
        masks = sorted(glob.glob('SemanticSegmentation/dataset/masks_vk/*.png'
))

        images_dataset = tf.data.Dataset.from_tensor_slices(images)
        masks_dataset = tf.data.Dataset.from_tensor_slices(masks)

        dataset = tf.data.Dataset.zip((images_dataset, masks_dataset))

        dataset = dataset.map(load_images, num_parallel_calls=tf.data.AUTOTUNE
)
        dataset = dataset.repeat(60)

        dataset = dataset.map(augmentate_images, num_parallel_calls=tf.data.AU
TOTUNE)
```

```
    images_and_masks = list(dataset.take(5))

    fig, ax = plt.subplots(nrows = 2, ncols = 5, figsize=(15, 5), dpi=125)

    for i, (image, masks) in enumerate(images_and_masks):
        ax[0, i].set_title('Image')
        ax[0, i].set_axis_off()
        ax[0, i].imshow(image)

        ax[1, i].set_title('Mask')
        ax[1, i].set_axis_off()
        ax[1, i].imshow(image/1.5)

        for channel in range(CLASSES):
            contours = measure.find_contours(np.array(masks[:,:,channel]))
            for contour in contours:
                ax[1, i].plot(contour[:, 1], contour[:, 0], linewidth=1, c
olor=COLORS[channel])

    plt.show()
    plt.close()



    train_dataset = dataset.take(2000).cache()
    test_dataset = dataset.skip(2000).take(100).cache()

    train_dataset = train_dataset.batch(16)
    test_dataset = test_dataset.batch(16)



    #10 Initing main blocks of model
    def input_layer():
        return tf.keras.layers.Input(shape=SAMPLE_SIZE + (3,))

    def downsample_block(filters, size, batch_norm=True):
        initializer = tf.keras.initializers.GlorotNormal()

        result = tf.keras.Sequential()

        result.add(
          tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',
                                 kernel_initializer=initializer, use_bias=
False))

        if batch_norm:
            result.add(tf.keras.layers.BatchNormalization())
```

```
            result.add(tf.keras.layers.LeakyReLU())
            return result

        def upsample_block(filters, size, dropout=False):
            initializer = tf.keras.initializers.GlorotNormal()

            result = tf.keras.Sequential()

            result.add(
                tf.keras.layers.Conv2DTranspose(filters, size, strides=2, padd
ing='same',
                                                kernel_initializer=initializer
, use_bias=False))

            result.add(tf.keras.layers.BatchNormalization())

            if dropout:
                result.add(tf.keras.layers.Dropout(0.25))

            result.add(tf.keras.layers.ReLU())
            return result

        def output_layer(size):
            initializer = tf.keras.initializers.GlorotNormal()
            return tf.keras.layers.Conv2DTranspose(CLASSES, size, strides=2, p
adding='same',
                                                   kernel_initializer=initiali
zer, activation='sigmoid')



        #11 Building UNET model for network
        inp_layer = input_layer()

        downsample_stack = [
            downsample_block(64, 4, batch_norm=False),
            downsample_block(128, 4),
            downsample_block(256, 4),
            downsample_block(512, 4),
            downsample_block(512, 4),
            downsample_block(512, 4),
            downsample_block(512, 4),
        ]

        upsample_stack = [
            upsample_block(512, 4, dropout=True),
            upsample_block(512, 4, dropout=True),
            upsample_block(512, 4, dropout=True),
            upsample_block(256, 4),
```

```
    upsample_block(128, 4),
    upsample_block(64, 4)
]

out_layer = output_layer(4)

# skip connections
x = inp_layer

downsample_skips = []

for block in downsample_stack:
    x = block(x)
    downsample_skips.append(x)

downsample_skips = reversed(downsample_skips[:-1])

for up_block, down_block in zip(upsample_stack, downsample_skips):
    x = up_block(x)
    x = tf.keras.layers.Concatenate()([x, down_block])

out_layer = out_layer(x)

UNET _like = tf.keras.Model(inputs=inp_layer, outputs=out_layer)

tf.keras.utils.plot_model(UNET _like, show_shapes=True, dpi=72)


#12 Creating metrix for img
def dice_mc_metric(a, b):
    a = tf.unstack(a, axis=3)
    b = tf.unstack(b, axis=3)

    dice_summ = 0

    for i, (aa, bb) in enumerate(zip(a, b)):
        numenator = 2 * tf.math.reduce_sum(aa * bb) + 1
        denomerator = tf.math.reduce_sum(aa + bb) + 1
        dice_summ += numenator / denomerator

    avg_dice = dice_summ / CLASSES

    return avg_dice

def dice_mc_loss(a, b):
    return 1 - dice_mc_metric(a, b)

def dice_bce_mc_loss(a, b):
```

```
        return 0.3 * dice_mc_loss(a, b) + tf.keras.losses.binary_crossentr
opy(a, b)


    #13 Building model
    UNET
_like.compile(optimizer='adam', loss=[dice_bce_mc_loss], metrics=[dice_mc_me
tric])



    #14 Creating/saving network for processes
    history_dice = UNET
_like.fit(train_dataset, validation_data=test_dataset, epochs=50, initial_ep
och=0)

    UNET _like.save_weights('SemanticSegmentation/networks/UNET _like')


    #15  Loading network for process images
    UNET _like.load_weights('SemanticSegmentation/networks/UNET _like')


    #16 Processing images
    rgb_colors = [
        (0,   0,   0),
        (0, 255, 0),
        (255, 0, 0)
    ]

    frames = sorted(glob.glob('SemanticSegmentation/videos/original_video_
vk/*.jpg'))

    i = 0

    for filename in frames:
        frame = imread(filename)
        sample = resize(frame, SAMPLE_SIZE)

        predict = UNET
_like.predict(sample.reshape((1,) +  SAMPLE_SIZE + (3,)))
        predict = predict.reshape(SAMPLE_SIZE + (CLASSES,))

        scale = frame.shape[0] / SAMPLE_SIZE[0], frame.shape[1] / SAMPLE_S
IZE[1]

        frame = (frame / 1.5).astype(np.uint8)

        isOk = 1
```

```
            for channel in range(1, CLASSES):
                contour_overlay = np.zeros((frame.shape[0], frame.shape[1]))
                contours = measure.find_contours(np.array(predict[:,:,channel]
))

                try:
                    for contour in contours:
                        rr, cc = polygon_perimeter(contour[:, 0] * scale[0],
                                                   contour[:, 1] * scale[1],
                                                   shape=contour_overlay.shape
)

                        contour_overlay[rr, cc] = 1

                    contour_overlay = dilation(contour_overlay, disk(1))
                    frame[contour_overlay == 1] = rgb_colors[channel]

                    # 2 => red (incorrect)
                    if(channel == 2):
                      isOk = 0
                except:
                    pass

            if(isOk == 1):
              imsave(f'SemanticSegmentation/videos/res/{os.path.basename(filen
ame)}', frame)
            else:
              imsave(f'SemanticSegmentation/videos/res_incorrect/{os.path.base
name(filename)}', frame)
```