

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу  
Кафедра математичних методів системного аналізу**

До захисту допущено:

Завідувач кафедри

\_\_\_\_\_ Оксана ТИМОЩУК

«\_\_\_» \_\_\_\_\_ 21\_\_ р.

**Дипломна робота**

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Системи і методи штучного  
інтелекту»**

**спеціальності 122 «Комп'ютерні науки»**

**на тему: «Модель класифікації пошкоджень на склі автомобіля за  
допомогою нейронних мереж»**

Виконав: студент IV курсу, групи КА-76

Славінський Всеволод Олександрович

\_\_\_\_\_

Керівник: д.т.н., доцент

Недашківська Надія Іванівна

\_\_\_\_\_

Консультант з нормконтролю: к.т.н., доцент

Коваленко Анатолій Єпіфанович

\_\_\_\_\_

Консультант з економічного розділу: к.е.н., доцент

Рощина Надія Василівна

\_\_\_\_\_

Рецензент: к.т.н., доцент кафедри СП ІПСА

НТУУ «КПІ ім. Ігоря Сікорського»

Безносик Олександр Юрійович

\_\_\_\_\_

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

Студент (-ка) \_\_\_\_\_

Київ – 2021 року

**Національний технічний університет України**  
**«Київський політехнічний інститут імені Ігоря Сікорського»**  
**Інститут прикладного системного аналізу**  
**Кафедра математичних методів системного аналізу**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп’ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_ Оксана ТИМОЩУК

«26» травня 2021 р.

**ЗАВДАННЯ**

**на дипломну роботу студенту**

**Славінський Всеволод Олександрович**

1. Тема роботи «Модель класифікації пошкоджень на склі автомобіля за допомогою нейронних мереж», керівник роботи доцент, д.т.н. Недашківська Надія Іванівна, затверджені наказом по університету від «26» травня 2021 р. № 1344-с
2. Термін подання студентом роботи 8.06.2021.
3. Вихідні дані до роботи – набір зображень із пошкодженнями лобового скла автомобіля.
4. Зміст роботи – 1. Дослідження предметної області задачі класифікації пошкоджень на склі автомобіля; 2. Математичні основи згорткових нейронних мереж 3. Побудова системи для класифікації зображень пошкоджень на склі автомобіля моделями нейронних мереж;
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

## 6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В.		

## 7. Дата видачі завдання 02.02.2021

## Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Підготовка даних до роботи	13.02.2021	Виконано
2.	Вивчення літератури за темою роботи	11.03.2021	Виконано
3.	Підготовка першого розділу	26.03.2021	Виконано
4.	Розробка моделей нейронних мереж	02.04.2021	Виконано
5.	Підготовка другого розділу	22.04.2021	Виконано
6.	Розробка веб клієнта	18.05.2021	Виконано
7.	Підготовка третього розділу	24.05.2021	Виконано
8.	Підготовка економічної частини	27.05.2021	Виконано
9.	Підготовка презентації доповіді	29.05.2021	Виконано
10.	Оформлення дипломної роботи	02.06.2021	Виконано

Студент

Всеволод СЛАВІНСЬКИЙ

Керівник

Надія НЕДАШКІВСЬКА

## РЕФЕРАТ

Дипломна робота: 122 с., 33 рис., 12 табл., 2 додатки, 17 джерел.

НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ВЕБ-ДОДАТОК, СТРАХУВАННЯ.

Об'єкт дослідження – класифікація пошкоджень на склі автомобілів.

Інколи люди потрапляють в автокатастрофи і щоб уникнути великих втрат за ремонт автомобіля, доволі часто машини страхують. Після кожного випадку, страховій компанії необхідно визначити, які саме пошкодження присутні на авто. Зазвичай, пошкодження присутні на лобовому або задньому склі.

Однак, зрозуміти яке це саме пошкодження та що з ним необхідно зробити – зовсім не тривіальна задача і потребує кваліфікаційних працівників.

Мета роботи – створити просте і гнучке рішення на основі веб додатка, яке б дозволяло дуже просто створювати підтвердження для страхових компаній, щодо виду пошкодження.

Ключовою особливістю програми є можливість автоматичного визначення категорії пошкодження для того, щоб додаток міг працювати незалежно від людей, які є носіями знань. Це значно прискорить та здешевить роботу великій кількості страхових компаній та їх посередникам.

## ABSTRACT

Bachelor thesis: 122 p., 33 fig., 12 tabl., 2 append., 17 sources.

NEURAL NETWORKS, CLASSIFICATION, CONVOLUTIONAL NEURAL NETWORKS, WEB APPLICATION, INSURANCE.

The object of research is the classification of damage to car windows.

Sometimes people get into car accidents and to avoid large losses for car repairs, cars are often insured. After each case, the insurance company needs to determine what kind of damage is present on the car. Usually, damage is present on the windshield or rear window.

However, understanding what exactly this damage is and what needs to be done with it is not a trivial task at all and requires skilled workers.

The purpose of the work is to create a simple and flexible solution based on a web application, which would make it very easy to create confirmation for insurance companies regarding the type of damage.

A key feature of the program is the ability to automatically determine the category of damage so that the application can work independently of people who are carriers of knowledge. This will significantly speed up and reduce the cost of a large number of insurance companies and their intermediaries.

## ЗМІСТ

ВСТУП .....	9
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАДАЧІ КЛАСИФІКАЦІЇ ПОШКОДЖЕНЬ НА СКЛІ АВТОМОБІЛЯ .....	11
1.1 Актуальність роботи.....	11
1.2 Опис даних .....	11
1.3 Аналіз даних.....	13
1.4 Постановка задачі .....	16
1.5 Формальна постановка задачі .....	16
1.6 Висновки до розділу 1 .....	17
РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ.....	19
2.1 Огляд згорткових нейронних мереж .....	19
2.1.1 Визначення операції згортки.....	20
2.2 Опис архітектури згорткової нейронної мережі .....	23
2.2.1 Input layer: .....	23
2.2.2 Convolution layer: .....	24
2.2.3 ReLu layer (активація):.....	24
2.2.4 Pooling layer (об'єднання): .....	24
2.2.5 Fully connected layer .....	25
2.3 Види шарів у згортковій архітектурі .....	25
2.3.1 Convolution .....	25
2.3.2 Шари активації .....	26
2.3.3 Pooling.....	28
2.3.4 Fully connected layer .....	29
2.3.5 Dropout.....	30
2.4 Порівняльний аналіз оптимізаторів.....	30
2.4.1 Градієнтний спуск .....	31
2.4.2 Стохастичний градієнтний спуск .....	32
2.4.3 Градієнтний спуск з моментом .....	33
2.4.4 Adagrad .....	33
2.4.5 AdaDelta.....	35
2.4.6 Adam .....	35

2.5	Preprocessing Image Data .....	36
2.5.1	Нормалізація .....	37
2.5.2	Аугментація .....	37
2.6	Відомі архітектури згорткових мереж.....	38
2.6.1	MobileNet.....	38
2.6.2	EfficientNet .....	40
2.6.3	VGG .....	40
2.6.4	Inception.....	42
2.6.5	ResNet .....	44
2.7	Висновки до розділу 2.....	46
РОЗДІЛ 3. ПОБУДОВА СИСТЕМИ ДЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ ПОШКОДЖЕНЬ НА СКЛІ АВТОМОБІЛЯ МОДЕЛЯМИ НЕЙРОННИХ МЕРЕЖ.....		47
3.1	Архітектура системи та вибір мови програмування і фреймворків....	47
3.1.1	Моделі нейронних мереж .....	48
3.1.2	Клієнт.....	50
3.1.3	Сервер.....	51
3.2	Порівняльний аналіз навчених моделей .....	52
3.2.1	Inception .....	53
3.2.2	EfficientNet .....	54
3.2.3	ResNet .....	55
3.2.4	MobileNet.....	57
3.2.5	NasNet .....	58
3.2.6	Порівняльна таблиця.....	60
3.3	Приклад роботи веб додатку .....	60
3.4	Висновки до розділу 3 .....	62
РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ .....		64
4.1	Постановка задачі проектування .....	64
4.2	Обґрунтування функцій програмного продукту .....	65
4.3	Обґрунтування системи параметрів ПП.....	68
4.4	Аналіз експертного оцінювання параметрів.....	71
4.5	Аналіз рівня якості варіантів реалізації функцій .....	76

4.6	Економічний аналіз варіантів розробки ПП .....	78
4.7	Вибір кращого варіанту ПП техніко-економічного рівня .....	84
4.8	Висновки до розділу 4.....	85
ВИСНОВКИ.....		86
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ .....		88
СПИСОК НАУКОВИХ ДОСЯГНЕНЬ.....		91
ДОДАТОК А ЛІСТІНГ ПРОГРАМИ .....		92
ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....		112



## ВСТУП

Один з пріоритетних напрямків страхування займає автострахування. Більшість випадків які припадають на автострахування потрапляють під категорії визначення пошкоджень які присутні на лобовому склі.

Страхові компанії дуже ретельно підходять до визначення категорії цього пошкодження і необхідного дії яку потрібно прийняти. Існує досить великий список категорій, який відповідає зразковому візуальному відображенню на склі а також іншим характеристикам.

За правилами цих пошкоджень існує 2 основних способи впливу при прийнятті рішень, такі як заміна лобового скла повністю або зробити ремонт за допомогою страхових партнерів.

Завдання полягає в тому щоб створити просте і гнучке рішення на основі веб і мобільного додатків, яке б дозволяло дуже просто створювати підтвердження для страхових компаній

Важливість роботи полягає в створенні моделі нейронної мережі, яка допоможе з класифікацією пошкоджень на склі автомобіля. Модель, отримана в результаті, значно полегшить роботу страховим компаніям у визначенні пошкоджень, а також зробить систему, яка не буде залежати від людського фактору, що позитивно вплине на страхових компаніях, а також на посередників

Об'єктом дослідження є аналіз зображень з пошкодженнями на склі автомобіля, які були отримані за допомогою реальних користувачів вже працюючого додатка, який було написано для однієї з компаній-клієнтів, а також зображення отриманні в результаті алгоритмів аргументації.

Предметом дослідження визначено згорткові нейронні мережі для класифікації зображень .

Пояснювальна записка складається з чотирьох розділів. У першому розділі проводиться постановка задачі, уточнення її актуальності, а також проводиться огляд об'єкта дослідження. У другому розділі розглянуто ряд

математичних моделей зготкових нейронних мереж, за допомогою яких відбувається класифікація. Третій розділ описує архітектуру розробленої програми, також у ньому аналізуються результати роботи алгоритму. Четвертий розділ являє собою економічну частину, в якій розробляється кошторис витрат на розробку.

## РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАДАЧІ КЛАСИФІКАЦІЇ ПОШКОДЖЕНЬ НА СКЛІ АВТОМОБІЛЯ

### 1.1 Актуальність роботи

Задача є актуальною, тому що страхова сфера є досить складною і з великою кількістю різноманітних нюансів. Також, страхові компанії досить закриті і користуються застарілим програмним забезпеченням або паперовим видом оформлення страхових випадків.

Однією з ключових особливостей програми є можливість автоматичного визначення категорії пошкодження для того, щоб додаток міг працювати незалежно від людей, які є носіями знань, а це принесе велику гнучкість в майбутньому масштабуванні на велику кількість страхових компаній та на посередників. Саме це ми намагаємося досягти використовуючи глибоке навчання і згорткові нейронні мережі.

В майбутньому після досягнення необхідної кількості метрик і даних додаток зможе досить точно визначати і прогнозувати нові метрики, які зможуть більш ефективно управляти страховими випадками.

### 1.2 Опис даних

Набір даних був сформований реальними користувачами додатка, розробленого працівниками компанії WebLegends у рамках контракту з компанією, що займається випадками автострахування, та попередньо відредагований мною.

Зображення не знаходяться у відкритому доступі та повністю належать компанії-розробнику. Самі зображення являють собою фото зроблені за допомогою смартфонів. На зображеннях знаходяться автомобільні засоби сфотографовані з різних сторін, а також ліцензійні номери цих авто. На більшості зображень присутнє скло з тим чи іншим пошкодженням. Щоб нейронна мережа була точнішою, я обрізав зображення, залишивши лише самі

пошкодження (Рисунок 1.1). Таким чином була прибрана уся чутлива інформація і залишилося лише те, що дійсно необхідно для нейронної мережі.

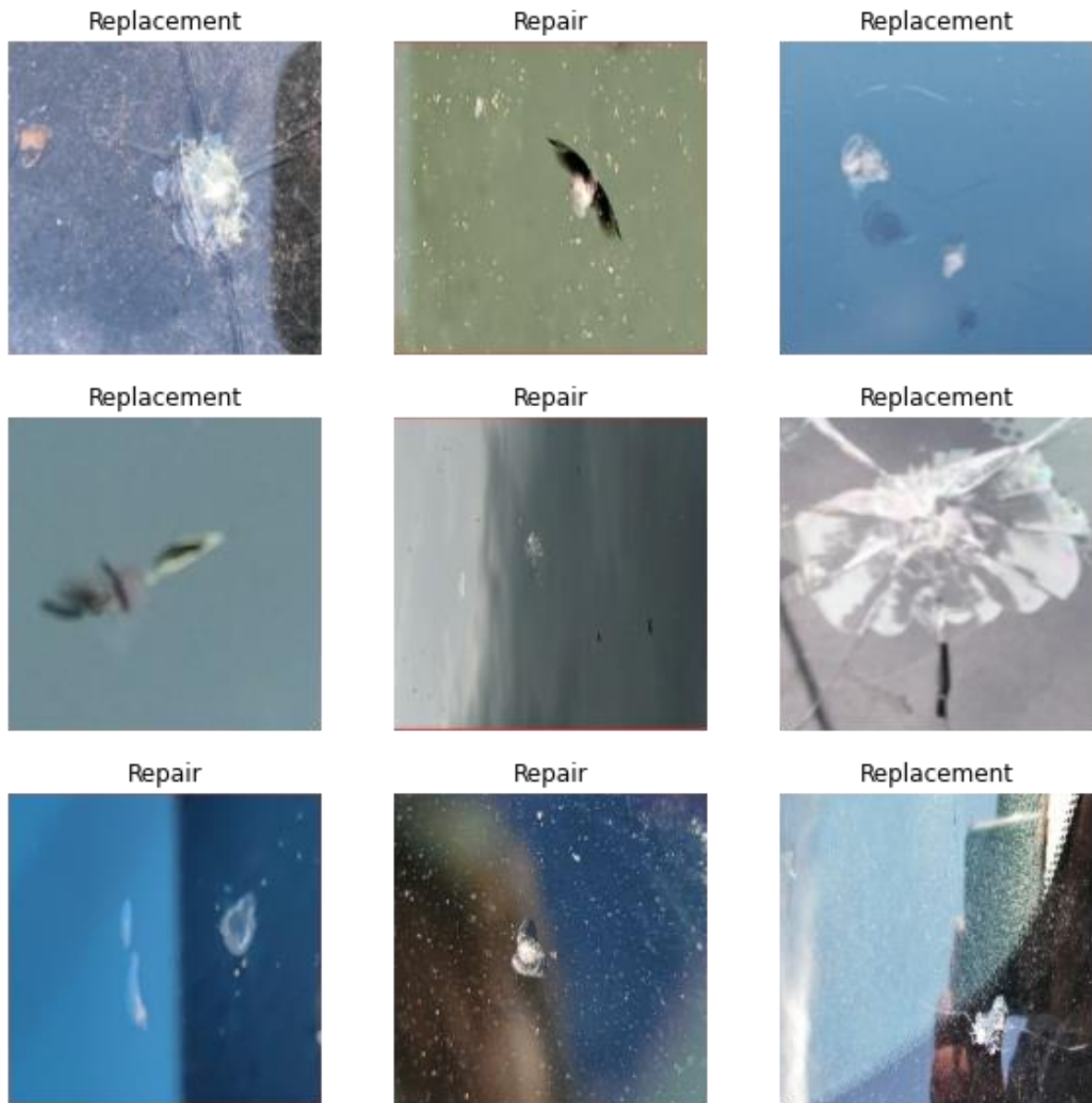


Рисунок 1.1 Зображення з набору даних з підписами

Усі зображення зберігаються у форматах .jpg, .jpeg та .png та мають розмір 224x224 пікселя та три канали R, G, B. На рівні датасету зображення не поділені на тренувальні, валідаційні та тестові – розділення буде здійснене в момент завантаження даних до середи програмування.

Дані про розмічені області зберігаються у файлі dataset\_metadata.json у форматі зображеному на рисунку 1.2

```

{
  "category": "5",
  "coordinates": {
    "height": 224,
    "width": 224,
    "x": 302.19718309859155,
    "y": 514.8387096774194
  },
  "fileName": "20190102--201128_RG1_f8d990e7-7d77-46da-b9a4-2cc5d8b83948_IMG_20181107_163034.jpg",
  "label": "replacement",
  "type": "rectangle"
},
{
  "category": "2",
  "coordinates": {
    "height": 223,
    "width": 224,
    "x": 324.9382716049383,
    "y": 485.3333333333333
  },
  "fileName": "20190102--201546_aztech_4f575c0c-5da9-4bf6-96fe-da0ac2b7022c_image.jpg",
  "label": "repair",
  "type": "rectangle"
},
}

```

Рисунок 1.2 Фрагмент файла dataset\_metadata.json

### 1.3 Аналіз даних

Загальна кількість зображень в наборі даних 9748. На кожному з зображень крупним планом зображена одна з двох категорій: «Repair» або «Replacement». При цьому 7163 з них належать категорії «Replacement» та 2585 до категорії «Repair» (Рисунок 1.3).



Рисунок 1.3 Розподіл категорій

Таку перевагу одної категорії над іншою дуже легко пояснити: у сучасному світі пошкоджене скло більш вірогідно потрібно буде повністю замінювати, оскільки ремонт лобового скла – це або дуже дорого, або ж зовсім неможливо. Тому оскільки дані, які належать до вибірки, абсолютно реальні, в цій перевазі нема зовсім нічого дивного.

Поглянемо детальніше як виглядають пошкодження кожного із класів.

Як бачимо, зображення з категорії Replacement представляють собою тріщини та дірки у склі (Рисунок 1.4). Однак є деякі тріщини, які майже не видно на зображенні через кут зору і можна лише здогадуватися де вони є.

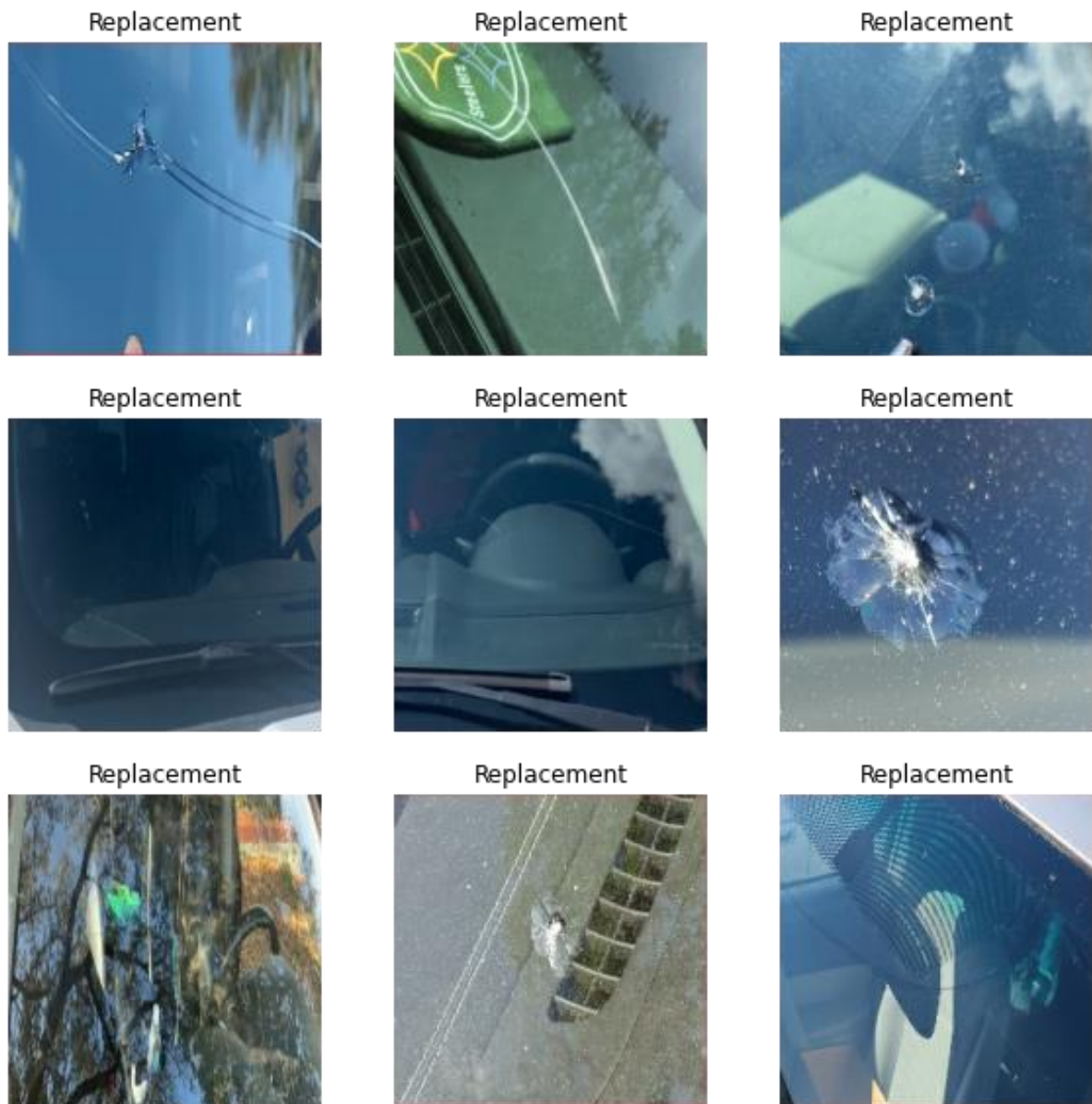


Рисунок 1.4 Зображення класа Replacement (повна заміна)

На відміну від минулої категорії, в категорії Repair майже усі зображення мають сколи (Рисунок 1.5).

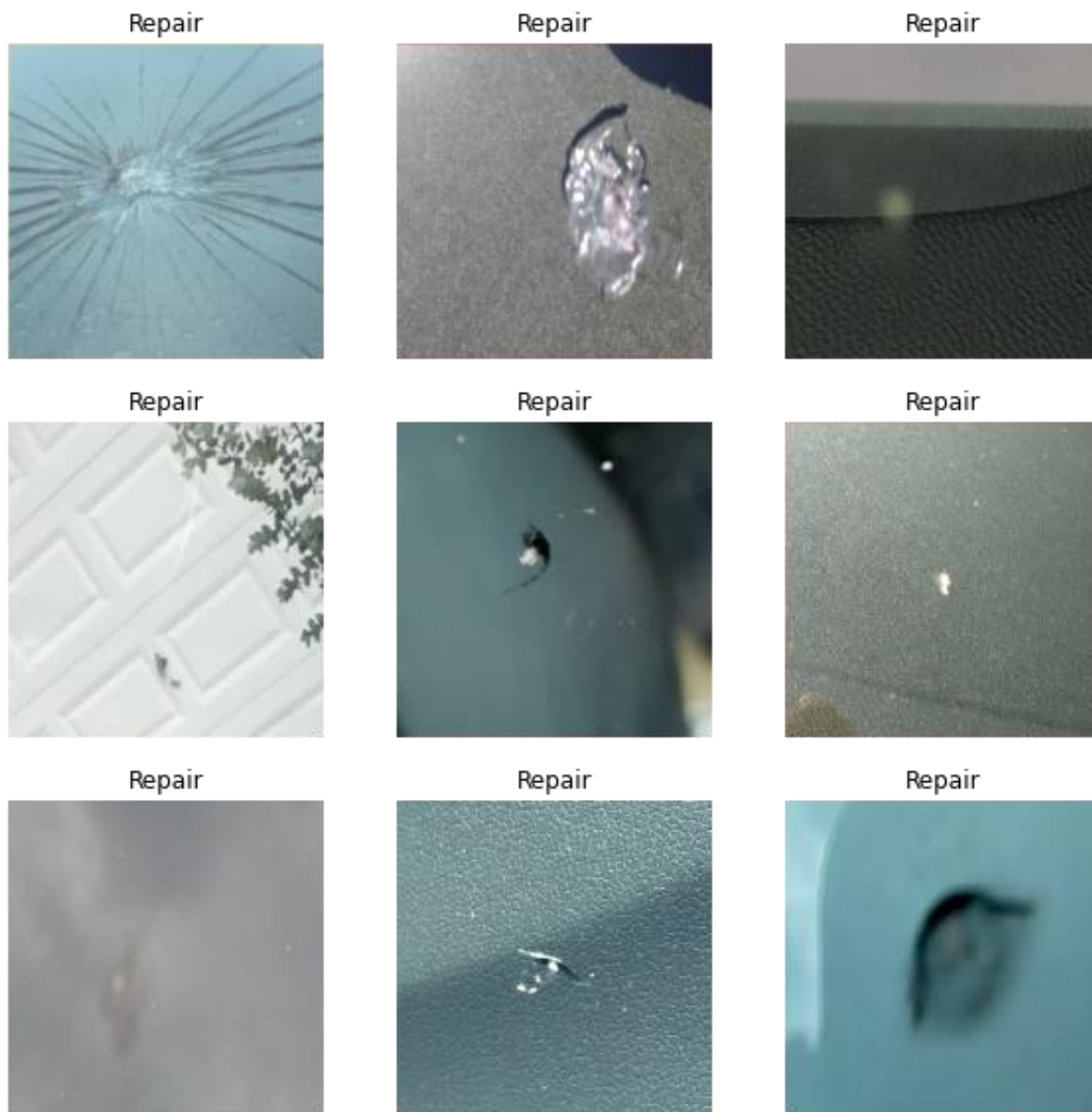


Рисунок 1.5 Зображення класа Repair (ремонт)

Звичайній людині доволі складно з впевненістю сказати де яка категорія, але цьому розподілу можна довіряти, оскільки його робили спеціально навчені люди. Що ж, подивимося як з цим впораються алгоритми машинного навчання.

## 1.4 Постановка задачі

Спочатку необхідно визначитися з типом задачі, яку ми розв'язуємо. Потрібно віднести кожне з зображень до одного з двох класів: Repair або Replacement. Тобто перед нами стоїть задача класифікації зображення.

Задача класифікації зображень – це задача, у якій міститься множина об'єктів (фотографій, малюнків, тощо), розподілених на класи за деякою логікою. Задана скінченна множина об'єктів, про яких відомо до яких класів вони належать. Ця множина називається вибіркою. У свою чергу ця вибірка розподіляється на 3 множини: тренувальна, валідаційна та тестова вибірки [1].

Тренувальна вибірка – вибірка, завдяки якій відбувається налаштування (оптимізація) ваг і коефіцієнтів моделі.

Валідаційна вибірка – вибірка для підбору параметрів, ознак та прийняття рішень, що стосуються навчання моделі (такі, як перенавчання)

Тестова вибірка – вибірка для оцінки якості навчання моделі. Тестова вибірка не повинна залежати від тренувальної [2].

Необхідно побудувати алгоритм, який буде здатний класифікувати довільний об'єкт із початкової множини.

Класифікувати об'єкт – указати номер або назву класу, до якого відноситься даний об'єкт.

В машинному навчанні задачі класифікації зображень вирішується, як правило, за допомогою методів штучної згорткової нейронної мережі у вигляді навчання з учителем [1].

## 1.5 Формальна постановка задачі

Нехай  $X$  – множина опису об'єктів,  $Y$  – скінченна множина номерів (імен) класів. Існує невідома цільова залежність-відображення  $y^*: X \rightarrow Y$ , значення якої відомі тільки на об'єктах скінченної вибірки  $X^m =$



$\{(x_1, y_1), \dots, (x_m, y_m)\}$ . Необхідно побудувати алгоритм:  $a: X \rightarrow Y$ , здатний класифікувати довільний об'єкт  $x \in X$ .

Отже, поставимо задачу.

Дано: розміщені зображення з пошкодженнями на склі автомобільних засобів.

Опис системи: необхідно розробити систему, яка отримувала б фотографію на вхід, після цього користувач міг би виділити зону, на якій є пошкодження, а система відповіла, до якої з категорій це пошкодження належить.

Вимоги до системи: система повинна працювати на сервері, щоб користувачу не було потреби завантажувати будь які додаткові модулі та бібліотеки для роботи системи. Окрім цього, важливо, щоб різні компоненти не залежали один від одного, щоб можна було розробляти їх паралельно. Також система повинна мати зрозумілий програмний інтерфейс, щоб у майбутньому її можна було легко інтегрувати в іншу систему, що вимагає розпізнання пошкоджень.

Результат роботи: результатом роботи буде виведена на екран назва категорії.

## 1.6 Висновки до розділу 1

В даному розділі спершу була розглядана актуальність задачі класифікації зображень з пошкодженнями на склі авто.

Було проведено опис даних, з поясненням їх походження та методом збереження, а також з. поясненням форматів файлів., типів кодування, проведено аналіз предметної області.

Провели попередній аналіз вихідних даних, подивилися на те, як виглядають екземпляри кожного з класів.

В кінці розділу розглянули поняття класифікації зображень, поняття тренувальної, валідаційної та тестової вибірок, а також поставили задачу, вимоги та обмеження до системи, яка розроблюється.

## РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

### 2.1 Огляд згорткових нейронних мереж

Нейронні мережі складаються з окремих одиниць, які називаються нейронами. Нейрони розташовані в ряді груп - шарів (див. Рисунок 2.1). Нейрони в кожному шарі з'єднані з нейронами наступного шару. Дані надходять від вхідного шару до вихідного шару вздовж цих сполук. Кожен окремий вузол виконує простий математичний розрахунок. Потім він передає свої дані у всі вузли, до яких він підключений.

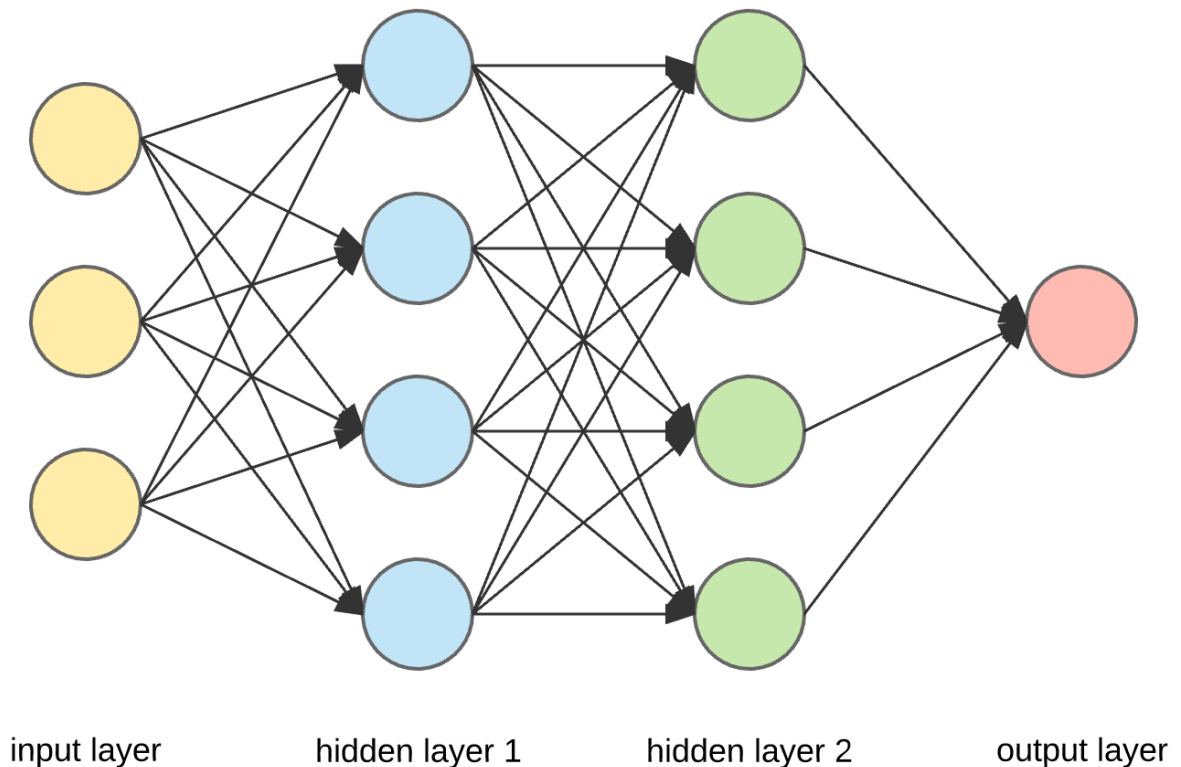


Рисунок 2.1 Архітектура нейронної мережі. [3]

На Рисунку 2.1 зображена архітектура нейронної мережі, що складається з декількох шарів: вхідного, вихідного та прихованих шарів. Кількість прихованих шарів визначають глибину нейронної мережі. Вхідні дані поступають до вхідного шару і потім поширюються через приховані шари до вихідного. В кожному прихованому шарі виконується лінійне перетворення вхідних даних, далі до них застосовується функція активації, яка додає

нелінійність до моделі. Аналогічно дані поширюються через усі приховані шари. У вихідному шарі визначається прогнозне значення  $y^*$  цільової змінної.

Нехай  $i$  – номер шару нейронної мережі, а  $j$  – номер нейрону прихованого шару. Тоді значення, яке надходить на вхід функції активації представляється наступним чином:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]} \quad (2.1)$$

де  $w, b, z$  – відповідно ваги, зміщення (bias) та результат лінійного перетворення.

Значення функції активації обчислюється наступним чином:

$$o_j^{[i]} = g(z_j^{[i]}) \quad (2.2)$$

де  $o_j^{[i]}$  – результат застосування функції активації;

$g$  – функція активації. [5]

### 2.1.1 Визначення операції згортки

Згорткові нейронні мережі (CNN) - це спеціальна архітектура штучних нейронних мереж, запропонована Яном ЛеКуном у 1988 р. Цей алгоритм може приймати вхідне зображення, призначати важливість різним аспектам / об'єктам на зображенні та мати можливість диференціювати один об'єкт від іншого.

Попередня обробка, необхідна в CNN, набагато нижча порівняно з іншими алгоритмами класифікації. Хоча в примітивних методах фільтри розробляються вручну, при достатній підготовці, CNN має можливість вивчати ці фільтри / характеристики.

Архітектура CNN аналогічна структурі зв'язку нейронів в мозку людини і натхненна організацією зорової кори. Окремі нейрони реагують на подразники лише в обмеженій області зорового поля, відомому як рецептивне поле. Колекція таких полів накладається на всю зорову зону.[4]

Один із найпопулярніших напрямків використання цієї архітектури - класифікація зображень. Наприклад, Facebook використовує CNN для алгоритмів автоматичного позначення, Amazon - для генерації рекомендацій щодо продуктів, а Google - для пошуку серед фотографій користувачів. [3]

Операція згортки визначається наступним чином [6]:

$$s(t) = (x * w)(t) \quad (2.3)$$

де  $x$  – вхідне значення,  $w$  – ядро згортки. Результат операції згортки також називають картою ознак [5].

Якщо розглядати дану операцію з точки зору нейронних мереж, то функція  $x$  – є вхідним значенням, а  $w$  – ядром. Результат операції іноді називають картою характеристик.[5]

#### 2.1.1.1 Представлення зображення у нейронній мережі

Розглянемо використання CNN для класифікації зображень більш детально. Основним завданням класифікації зображень є прийняття вхідного зображення та наступне визначення його класу. Це навичка, яку люди вивчають з народження і здатні легко визначити, що зображення на картині - це слон. Але комп'ютер бачить зображення зовсім інакше:

Замість зображення комп'ютер бачить масив пікселів (Рисунок 2.2). Наприклад, якщо розмір зображення становить 300 x 300. У цьому випадку розмір масиву буде 300x300x3 (Рисунок 2.3). Де 300 - це ширина, наступні 300 - це висота, а 3 - значення каналів RGB. Кожному з цих номерів комп'ютеру

присвоюється значення від 0 до 255. Це значення описує інтенсивність пікселя в кожній точці. [3]



Рисунок 2.2 Як комп'ютер бачить зображення [3]

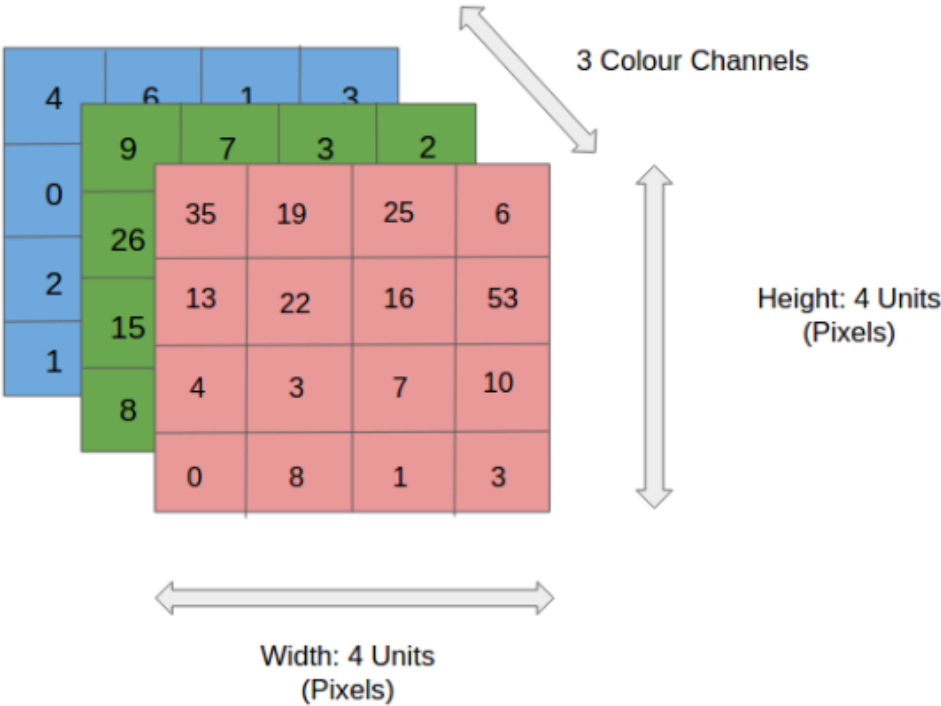


Рисунок 2.3 Матриця пікселів 4x4 в представленні комп'ютера [4]

Ви можете собі уявити, наскільки обчислювальними будуть речі, як тільки зображення досягнуть розмірів, скажімо, 8K (7680 × 4320). Роль конволюційних шарів полягає у зменшенні зображень у форму, яку легше обробити, не втрачаючи особливостей, які є критично важливими для отримання хорошого прогнозу. Це важливо, коли ми хочемо розробити

архітектуру, яка не тільки добре володіє функціями навчання, але й масштабована до масивних наборів даних.[4]

## 2.2 Опис архітектури згорткової нейронної мережі

Архітектури CNN складаються з окремих шарів (Рисунок 2.4). У всіх випадках шари приймають як вхідний 3D-об'єм, перетворюють цей об'єм за допомогою диференціальних рівнянь і виводять 3D-об'єм. Деякі шари вимагають налаштування гіперпараметрів, а інші - ні.

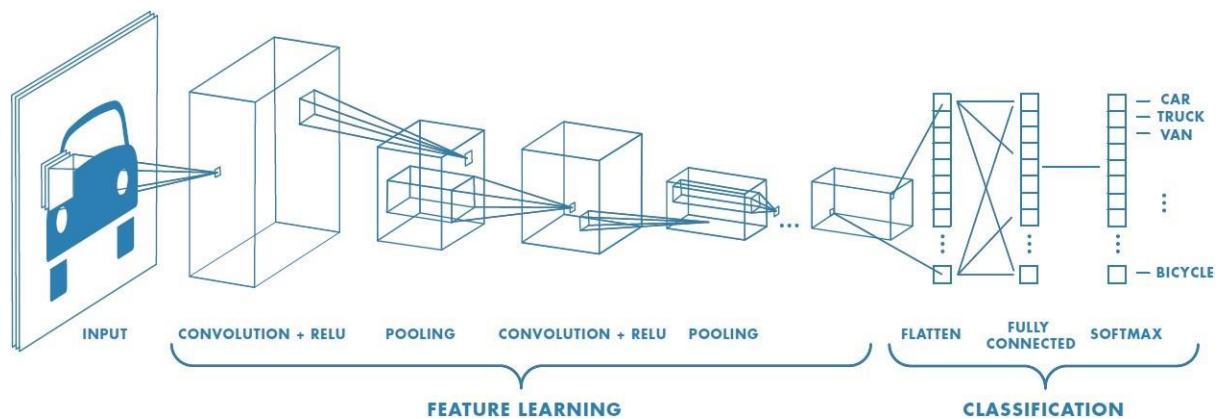


Рисунок 2.4 Описана вище архітектура CNN з підписаними назвами шарів [4]

### 2.2.1 Input layer:

- Вихідні значення пікселів зображення, представленого у вигляді 3D-матриці
- Розміри  $H \times W \times D$ , де глибина відповідає кількості кольорових каналів на зображенні.

### 2.2.2 Convolution layer:

- Конволюційні шари обчислюють вихідні дані вузлів, підключених до локальних областей вхідної матриці.
- Точкові добутки обчислюються між набором вагових коефіцієнтів (який зазвичай називають фільтрами) та значеннями, пов'язаними з локальною областю введення.

### 2.2.3 ReLu layer (активація):

- Вихідний обсяг конволюційних шарів подається на поелементну функцію активації, як правило, ReLu.
- Шар ReLu визначатиме, чи буде вхідний вузол "працювати" з урахуванням вхідних даних.
- Якщо вузол "спрацює", це буде сигналізувати, що фільтри шару згортки виявили візуальну особливість.
- Функція ReLu застосовуватиме функцію  $\max(0, x)$  з пороговим значенням 0.

### 2.2.4 Pooling layer (об'єднання):

- Для зменшення ширини та висоти вихідного обсягу застосовується об'єднання пікселів за допомогою згортки.



### 2.2.5 Fully connected layer

- Як і у звичайних нейронних мереж, кожен вузол у цьому рівні пов'язаний з кожним вузлом в обсязі об'єктів, що подаються далі.
- Імовірності класів обчислюються і виводяться в 3D-масиві з розмірами:  $[1 \times 1 \times K]$ , де  $K$  - кількість класів.

Далі детальніше пройдемося по кожному з шарів по черзі.[6]

## 2.3 Види шарів у згортковій архітектурі

### 2.3.1 Convolution

Шар згортки завжди йде першим. У нього вводиться зображення (матриця зі значеннями пікселів). Зчитування вхідної матриці пікселів починається у верхньому лівому куті зображення. Далі програмне забезпечення вибирає там меншу матрицю, яка називається фільтром (також називають нейроном або ядром). Потім за допомогою фільтра відбувається згортка, тобто рух вздовж вхідного зображення.

Завдання фільтра – помножити його значення на початкові значення пікселів. Всі ці множення підсумовуються. В результаті виходить одне число. (Рисунок 2.5) Оскільки фільтр зчитував зображення лише у верхньому лівому куті, він рухається все далі вправо на 1 одиницю, виконуючи таку саму операцію. Після роботи фільтра по всіх положеннях, виходиться матриця, але менша за вхідну. [3]

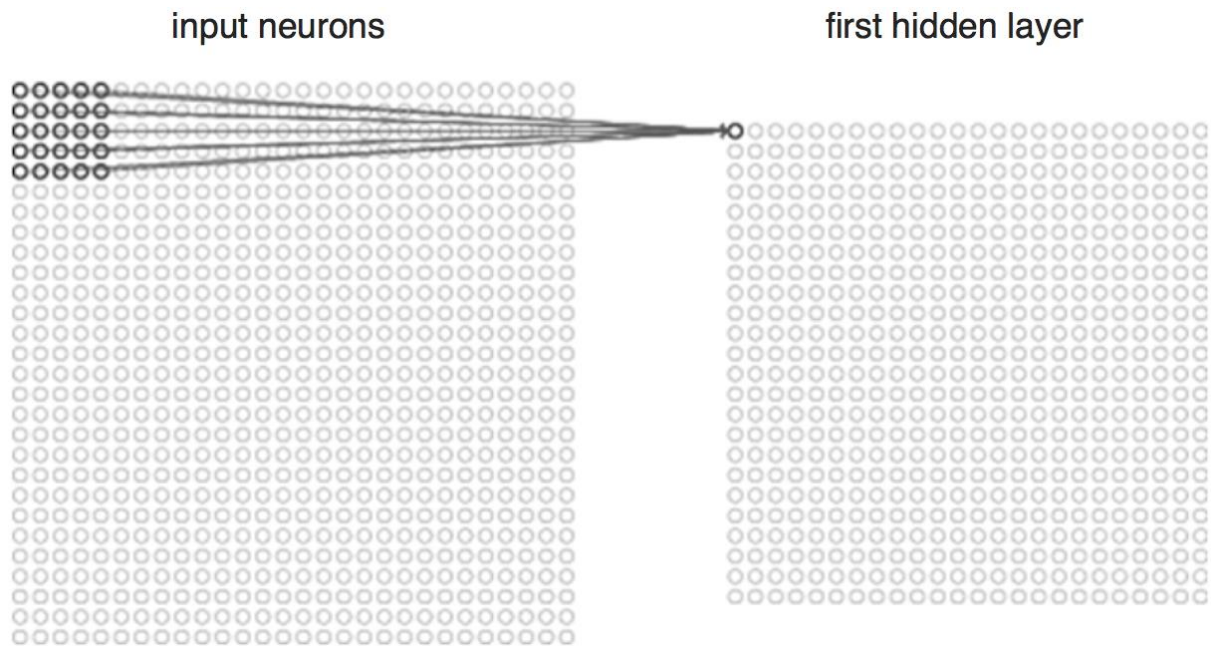


Рисунок 2.5 Приклад роботи шару згортки [3]

Згорткові шари – це будівельні блоки CNN. Ці шари зроблені з багатьох фільтрів, які визначаються їх шириною, висотою та глибиною. На відміну від щільних шарів регулярних нейронних мереж, згорткові шари побудовані з нейронів у 3-вимірах. Завдяки цій характеристиці, згорткові нейронні мережі є розумним рішенням для класифікації зображень. [6]

### 2.3.2 Шари активації

Після згортки вхідне зображення було перетворено в набір карт ознак зображення. Кожна карта ознаки відповідає візуальному об'єкту, який видно у певних місцях на вхідному зображенні. Розмір цього набору ознак дорівнює кількості вузлів (тобто фільтрів) у згортковому шарі.

Функції активації визначають релевантність даного вузла в нейронній мережі. Вузол, що відповідає прогнозуванню моделі, буде спрацьовувати після проходження функції активації. [6]

Функції активації мають достатньо довгу історію. Спочатку використовувалася функція сигмоїди, оскільки її похідна має діапазон значень від 0 до 1 та згладжену форму:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.4)$$

Також широко розповсюдженою була функція гіперболічного тангенсу, значення якого належали інтервалу  $[-1, 1]$ :

$$\tanh(x) = \frac{\operatorname{sh} x}{\operatorname{ch} x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} + 1}{e^{2x} - 1} \quad (2.5)$$

Але класичні функції активації замінила ReLU (Рисунок 2.6). Простота та ефективність ReLU призвели до появи її модифікацій, таких як Leaky ReLU та параметризованої ReLU [5].

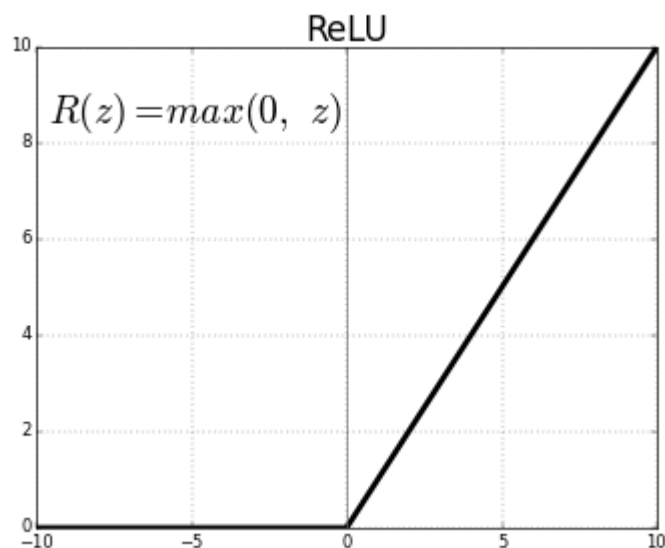


Рисунок 2.6 Графік функції ReLU

Однією з найбільших переваг ReLU перед іншими функціями активації є те, що він не активує всі нейрони одночасно. З зображення для функції ReLU вище ми помітимо, що воно перетворює всі негативні входи в нуль, і нейрон

не активується. Це робить його дуже обчислювальним, оскільки за раз активується мало нейронів.

Деяким недоліком ReLU є те, що він насичений в негативній області, тобто градієнт у цій області дорівнює нулю. З градієнтом, рівним нулю, під час зворотного розповсюдження всі ваги не оновлюватимуться, щоб виправити це, ми використовуємо Leaky ReLU.

Крім того, функції ReLU не мають нульового центру. Це означає, що для того, щоб він дійшов до своєї оптимальної точки, йому доведеться використовувати зигзагоподібний шлях, який може бути довшим. [7]

### 2.3.3 Pooling

Шар об'єднання можна побачити між шарами Convolution в архітектурі CNN. Цей рівень в основному зменшує кількість параметрів та обчислень у мережі, контролюючи надмірність шляхом поступового зменшення просторового розміру мережі.

Це має зменшити обчислювальну потужність, необхідну для обробки даних, завдяки зменшенню розмірності. Крім того, це корисно для вилучення домінантних ознак, які є інваріантними щодо обертання та позиції, таким чином підтримуючи процес ефективного навчання моделі.

Існує два типи об'єднання: Average Pooling та Max Pooling. Max Pooling повертає максимальне значення частини зображення, охопленої ядром. З іншого боку, Average Pooling повертає середнє значення всіх значень із частини зображення, охопленої ядром (Рисунок 2.7). [4]

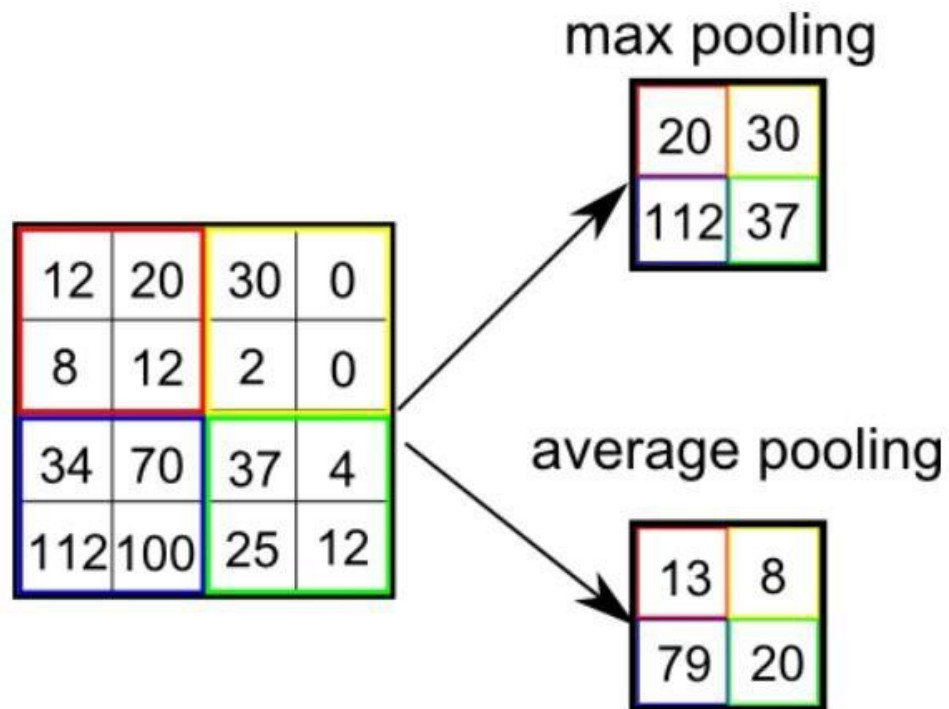


Рисунок 2.7 Приклад роботи шару об'єднання [4]

#### 2.3.4 Fully connected layer

У цьому шарі нейрони мають повний зв'язок з усіма активаціями попередніх шарів. Отже, їх активації можуть бути обчислені за допомогою матричного множення з подальшим зміщенням зміщення. Це остання фаза для мережі CNN. [7]

Метою повно зв'язного шару є прогнозування класів. Повністю зв'язаний шар прийме в якості вхідного сигналу сплюснений вектор вузлів, які були активовані в попередніх шарах згортки. Вибір правильної функції активації залежить від типу проблеми класифікації:

- Сигмоїд зазвичай використовують для двійкових задач класифікації, оскільки він є логістичною функцією
- Softmax гарантує, що сума значень у вихідному рівні складає 1, і може використовуватися як для двійкових, так і для багатокласних задач класифікації. [6]

### 2.3.5 Dropout

Також часто застосовується форма регуляризації, яка називається випадання. Випадання – це простий спосіб запобігти перенапруженню нейронних мереж.

Коли ми вводимо цю регуляризацію, ми випадковим чином відбираємо нейрони, які ігноруються під час навчання. Це тимчасово усуває внески в активацію нейронів, а також оновлення ваг (що відбуватиметься під час зворотного розмноження) також не застосовується (Рисунок 2.8). Це покращує узагальнення моделі та перешкоджає певним наборам ваг спеціалізуватися на конкретних особливостях, що може призвести до перенавчання, якщо не обмежено.

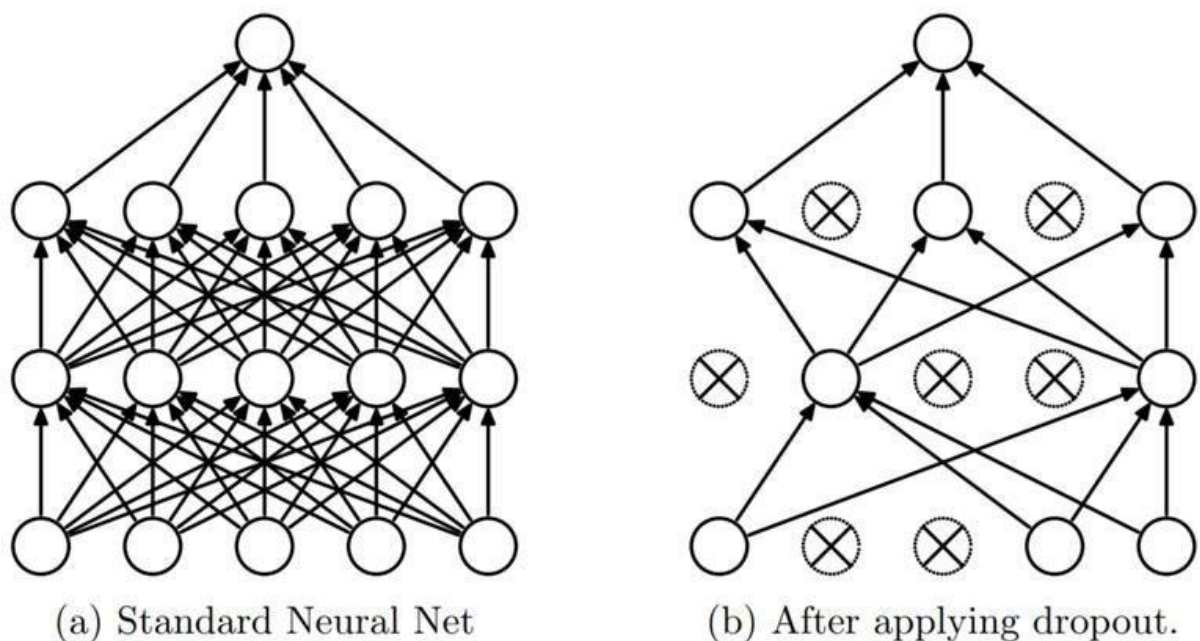


Рисунок 2.8 Приклад роботи шара випадання [8]

## 2.4 Порівняльний аналіз оптимізаторів

Оптимізатори – це алгоритми або методи, що використовуються для зміни атрибутів вашої нейронної мережі, таких як ваги та швидкість навчання, щоб зменшити втрати.

Алгоритми та стратегії оптимізації відповідають за зменшення втрат і забезпечують максимально точні результати роботи. [8]

#### 2.4.1 Градієнтний спуск

Градiєнтний спуск – це найосновніший, але найбільш часто використовуваний алгоритм оптимізації. Він широко використовується в алгоритмах лінійної регресії та класифікації. Зворотне поширення в нейронних мережах також використовує алгоритм градієнтного спуску.

Градiєнтний спуск – це алгоритм оптимізації першого порядку, який залежить від похідної першого порядку функції втрат. Він обчислює, яким чином слід змінити ваги, щоб функція могла досягти мінімумів.

Алгоритм:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \nabla J(\theta^{(t-1)}), \quad (2.6)$$

де  $\theta^{(t)}$  – вектор ваг нейронної мережі;

$\alpha$  – гіперпараметр, що визначає швидкість навчання;

$\nabla J$  – вектор градієнту функції втрат.

Переваги алгоритму градієнтного спуску:

- Простота обчислень.
- Легкий у реалізації.
- Легкий у розумінні.

Недоліки:

- Ваги змінюються після розрахунку градієнта для всього набору даних. Тому, якщо набір даних є занадто великим, то час роботи сильно зростає.
- Велика кількість пам'яті потрібна для обчислення градієнта для всього набору даних.

### 2.4.2 Стохастичний градієнтний спуск

Цей варіант градієнтного спуску намагається частіше оновлювати параметри моделі. У цьому параметри моделі змінюються після обчислення втрат на кожному навчальному прикладі. Отже, якщо набір даних містить 1000 рядків, SGD буде оновлювати параметри моделі 1000 разів за один цикл набору даних, а не один раз, як у випадку стандартного градієнтного спуску.

Алгоритм:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \nabla J(\theta^{(t-1)}; x_i; y_i), \quad (2.7)$$

де  $x_i, y_i$  – тренувальні приклади.

Оскільки параметри моделі часто оновлюються, параметри будуть мати велику дисперсію та коливання функцій втрат при різній інтенсивності.

Переваги:

- Часті оновлення параметрів моделі сходяться за менший час.
- Потрібно менше пам'яті, оскільки немає необхідності зберігати значення функцій втрат.
- Може отримати нові мінімуми.

Недоліки:

- Висока різниця в параметрах моделі.
- Може продовжувати виконуватися навіть після досягнення глобальних мінімумів.
- Для отримання збіжності, як у градієнтного спуску потрібно повільно зменшувати значення рівня навчання.



### 2.4.3 Градієнтний спуск з моментом

Градієнтний спуск з моментом був винайдений для зменшення великої дисперсії SGD та пом'якшення конвергенції. Включення моменту прискорює зближення до відповідного напрямку і зменшує коливання до нерелевантного напрямку. У цьому методі використовується ще один гіперпараметр, відомий як імпульс, що символізується " $\gamma$ ".

$$V^{(t)} = \gamma V^{(t-1)} + \alpha \nabla J(\theta^{(t-1)}), \quad (2.8)$$

де  $\gamma$  – гіперпараметр моделі, що відповідає моменту.

Правило оновлення ваг приймає наступний вигляд [8]:

$$\theta^{(t)} = \theta^{(t-1)} - V^{(t)} \quad (2.9)$$

Термін імпульсу  $\gamma$  зазвичай встановлюється на 0,9 або подібне значення.

Переваги:

- Зменшує коливання і велику дисперсію параметрів.
- Збігається швидше, ніж градієнтний спуск.

Недоліки:

- Додано ще один гіперпараметр, який потрібно обирати вручну.

### 2.4.4 Adagrad

Одним з недоліків усіх пояснених оптимізаторів є те, що швидкість навчання є постійною для всіх параметрів і для кожного циклу. Однак Adagrad змінює швидкість навчання « $\eta$ » для кожного параметра та на кожному кроці

часу «t». Це алгоритм оптимізації другого порядку. Він працює над похідною функції помилки.

$$g_i^{(t-1)} = \nabla J(\theta_i^{(t-1)}) \quad (2.10)$$

Тоді оновлення параметрів виконується за наступним правилом:

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \frac{\eta}{\sqrt{G_{ii}^{(t-1)} + \epsilon}} g_i^{(t-1)}, \quad (2.11)$$

де  $\eta$  – швидкість навчання, яка змінюється на основі минулих градієнтів в залежності від параметру  $\theta_i^{(t-1)}$  на основі минулих градієнтів;

$G_{ii}^{(t-1)}$  – сума квадратів градієнтів  $\theta_i^{(t-1)}$ ;

$\epsilon$  – доданок, який дозволяє уникати ділення на нуль і зазвичай обирається близьким до  $1e - 8$ ).

Переваги:

- Швидкість навчання змінюється для кожного навчального параметра.
- Не потрібно налаштовувати швидкість навчання вручну.
- Здатний тренуватися на обмежених даних.

Недоліки:

- Вимагає багато обчислень, оскільки потрібно обчислювати похідну другого порядку.
- Швидкість навчання завжди зменшується в результаті повільного навчання.

### 2.4.5 AdaDelta

Це розширення AdaGrad, яке прагне усунути проблему затухаючої швидкості навчання. Замість того, щоб акумулювати квадрати минулих градієнтів, AdaDelta обмежує вікно накопичених минулих градієнтів певним фіксованим розміром  $w$ . замість акумулювання квадратів минулих градієнтів. При цьому використовується зважене середнє, а не сума всіх градієнтів.

$$E[g^2]^{(t)} = \gamma E[g^2]^{(t-1)} + (1 - \gamma)[g^2]^{(t)}, \quad (2.12)$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{E[g^2]^{(t)} + \epsilon}} g^{(t)} \quad (2.13)$$

Переваги:

- Рівень навчання не занепадає, а навчання не припиняється.

Недоліки:

- Вимагає багато обчислень.

### 2.4.6 Adam

Адам (Адаптивна оцінка моменту) працює з імпульсами першого та другого порядку. Ідея цього алгоритму полягає в тому, щоб рухатися не дуже швидко до точки мінімуму функції втрат, оскільки можемо перестрибнути цей мінімум. На додаток до того, що Адам зберігає експоненційно зважене середнє квадратів минулих градієнтів, як і AdaDelta, Adam також зберігає експоненційно зважене середнє минулих значень градієнтів моменту.

$$\hat{m}^{(t)} = \frac{m^{(t)}}{1 - \beta_1^{(t)}}, \quad (2.14)$$

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1 - \beta_2^{(t)}}, \quad (2.15)$$

де  $\beta_1, \beta_2$  – гіперпараметри методу;

$m^{(t)}$  і  $Vv^{(t)}$  – це значення першого і другого моментів, які є середнім значенням і дисперсією градієнтів, відповідно.

Оновлення параметрів згідно з методом Adam наступне:

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{\hat{v}^{(t)}} + \epsilon} \hat{m}^{(t)} \quad (2.16)$$

Переваги:

- Метод працює швидко і швидко наближається до точки мінімуму функції втрат.
- Виправляє зникаючу швидкість навчання, велику дисперсію.

Недоліки:

- Вимагає багато обчислень.

## 2.5 Preprocessing Image Data

Згадаємо, як зображення інтерпретується комп'ютером. Комп'ютер переводить зображення у тривимірний масив значень від 0 до 255.

Перш ніж ми зможемо навчити модель розпізнавати та класифікувати зображення, дані потрібно певним чином підготувати. Залежно від завдання комп'ютерного зору, деякі кроки попередньої обробки можуть не потребувати реалізації, але майже завжди потрібно буде виконувати нормалізацію та аугментацію.

### 2.5.1 Нормалізація

Одною з найбільш поширених практик є нормалізація діапазону значень вхідних зображень перед подачею їх у модель. На цьому етапі попередньої обробки ми будемо масштабувати всі числові значення на наших зображеннях до значення в діапазоні від 0 до 1.

Це дуже важливо, оскільки деякі зображення можуть мати дуже високі значення пікселів, тоді як інші мають нижчі значення пікселів. Під час масштабування всіх зображень ви гарантуєте, що кожне зображення рівномірно сприяє функції втрати моделі.

Іншими словами, масштабування даних зображення гарантує, що всі зображення розглядаються однаково, коли модель тренується та оновлює свої ваги.

### 2.5.2 Аугментація

Класифікація зображень за об'єктами повинна бути статистично незмінною. Нагадаємо, що всі зображення представлені у вигляді тривимірних масивів значень пікселів. Однак слід врахувати шанс того, що не всі зображення будуть виглядати однаково.

Необхідно щоб алгоритм був незмінним, щоб покращити його здатність узагальнювати дані зображення. Це може бути зроблено шляхом введення збільшення кількості зображень в етапі попередньої обробки. Збільшення кількості вхідних зображень збільшує дисперсію навчальних даних різними способами, такими як: включення випадкового обертання, збільшення або зменшення яскравості, зміщення позицій об'єкта та горизонтальне або вертикальне гортання зображень. [6]

## 2.6 Відомі архітектури згорткових мереж

В наступних описах нейронних мереж будемо використовувати умовні позначення, описані на рисунку 2.9

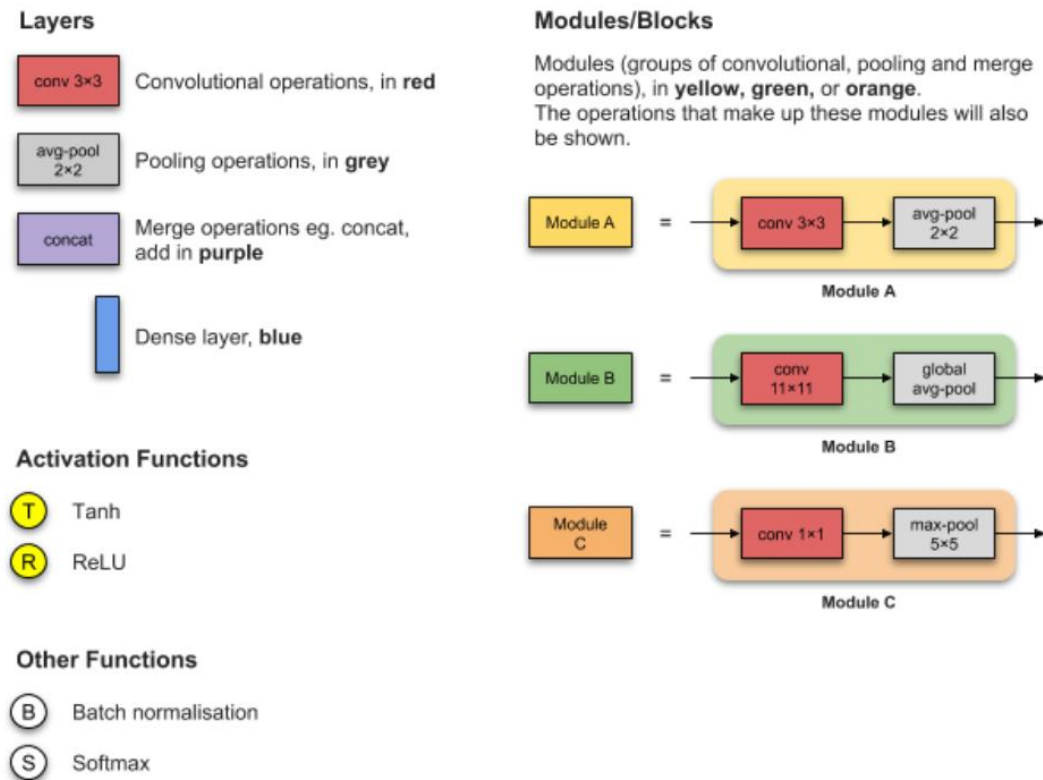


Рисунок 2.9 Умовні позначення шарів моделі [9]

### 2.6.1 MobileNet

Як підказує назва, модель MobileNet розроблена для використання в мобільних додатках, і це перша модель мобільного комп'ютерного зору TensorFlow.

MobileNet використовує відокремлені поглибленні шари згортки. Це значно зменшує кількість параметрів у порівнянні з мережею зі звичайними

шарами згортки з однаковою глибиною в сітках. Це призводить до полегшення глибоких нейронних мереж.

Згортка, що відокремлюється по глибині, складається з двох операцій.

1. Глибока згортка.
2. Точкова згортка.

MobileNet - це невелика модель з низькою затримкою та малою потужністю, параметризовані для задоволення обмежень ресурсів у різних випадках використання (Рисунок 2.10). Вони можуть використовуватися для класифікації, виявлення, вбудовування та сегментації.

MobileNet - це клас CNN, який був відкритий для Google, і тому це дає нам чудову відправну точку для навчання наших класифікаторів, які мають бути доволі малими, але водночас швидкими. [10]

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5×	Conv dw / s1	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 512$
		$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool $7 \times 7$	$7 \times 7 \times 1024$
FC / s1	$1024 \times 1000$	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Рисунок 2.10 Архітектура MobileNet [10]

### 2.6.2 EfficientNet

EfficientNet - це згорткова архітектура нейронної мережі та метод масштабування, який рівномірно масштабує всі розміри глибини/ширини/роздільної здатності, використовуючи складений коефіцієнт. (Рисунок 2.11) На відміну від звичайної практики, яка довільно масштабує ці фактори, метод масштабування EfficientNet рівномірно масштабує ширину, глибину та роздільну здатність мережі за допомогою набору фіксованих коефіцієнтів масштабування. [11]

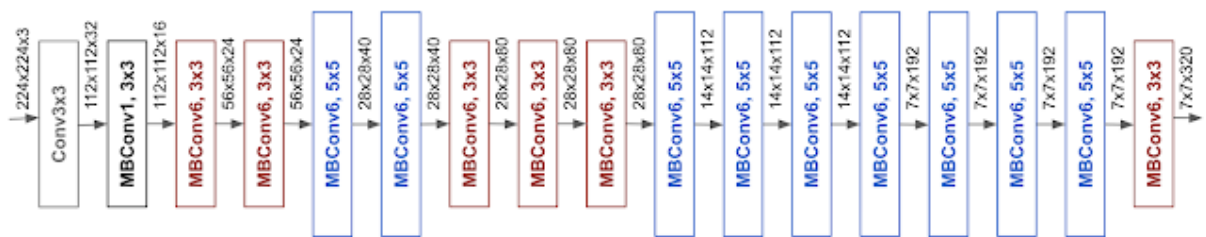


Рисунок 2.11 Архітектура EfficientNet [12]

Складений метод масштабування виправдовується інтуїцією того, що якщо вхідне зображення більше, то мережі потрібно більше шарів для збільшення сприйнятливого поля та більше каналів для захоплення більш дрібнозернистих візерунків на більшому зображенні.

### 2.6.3 VGG

Люди з Visual Geometry Group (VGG) винайшли VGG-16, який має 13 згорткових та 3 повністю зв'язані шари. Першою важливою відмінністю, яка стала фактичним стандартом, є використання великої кількості маленьких фільтрів. Зокрема, фільтри розміром  $3 \times 3$  та  $1 \times 1$  з кроком один (Рисунок



2.12), відрізняються від великих розмірів фільтрів у LeNet-5 та менших, але все ще відносно великих фільтрів та великого кроку чотирьох в AlexNet.

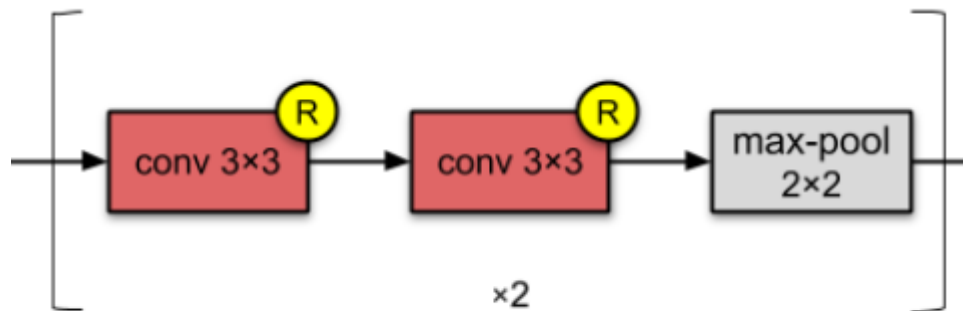


Рисунок 2.12 Група шарів згортки та об'єднання [9]

Шари Max Pooling використовуються після більшості, але не всіх згорткових шарів, виходячи з прикладу в AlexNet, але все об'єднання виконується з розміром  $2 \times 2$  і тим самим кроком, який теж став фактичним стандартом.

Зокрема, мережі VGG використовують приклади двох, трьох і навіть чотирьох згорткових шарів, складених разом перед використанням максимального шару об'єднання. Обґрунтуванням було те, що укладені згорткові шари з меншими фільтрами наближають ефект одного згорткового шару з фільтром більшого розміру, наприклад три складені згорткові шари з фільтрами  $3 \times 3$  наближаються до одного згорткового шару з фільтром  $7 \times 7$ .

Інша важлива відмінність - дуже велика кількість використовуваних фільтрів (Рисунок 2.13). Кількість фільтрів збільшується із глибиною моделі, хоча починається з відносно великої кількості 64 і збільшується через 128, 256 та 512 фільтрів в кінці частини видобування особливостей моделі.[13]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Рисунок 2.13 Архітектура мережVGG [13]

#### 2.6.4 Inception

Автори пропонують архітектуру, що називається початком. Ключовим нововведенням у цих моделях є початковий модуль. Це блок паралельних згорткових шарів з фільтрами різного розміру (наприклад,  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$ ) та шаром об'єднання  $3 \times 3$  max, результати якого потім об'єднуються (Рисунок 2.14).

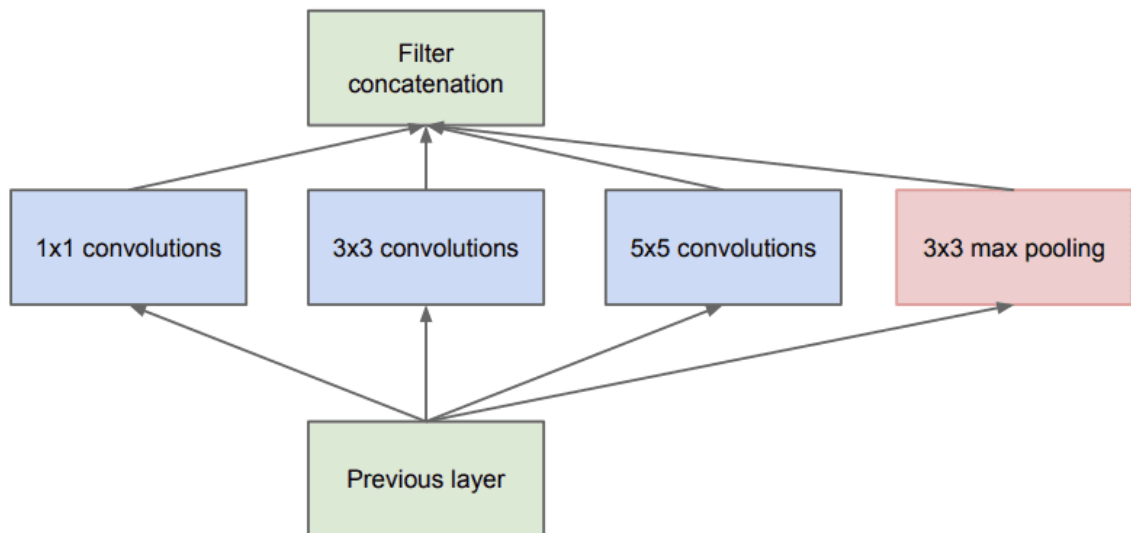


Рисунок 2.14 Модуль, запропонований в моделі Inception [13]

Другим важливим дизайнерським рішенням у моделі Inception було підключення виходу в різних точках моделі. Це було досягнуто шляхом створення невеликих вихідних мереж з основної мережі, які були навчені робити прогнози. Метою було надати додатковий сигнал помилки із завдання класифікації в різних точках глибинної моделі для вирішення проблеми зникаючих градієнтів. Потім ці невеликі вихідні мережі були видалені після навчання.

На Рисунку 2.15 наведена повернута версія (зліва направо для вводу-виводу) архітектури моделі Inception. Початок від входу ліворуч до класифікації виходів праворуч і два додаткові вихідні мережі, які використовувались лише під час навчання.[13]

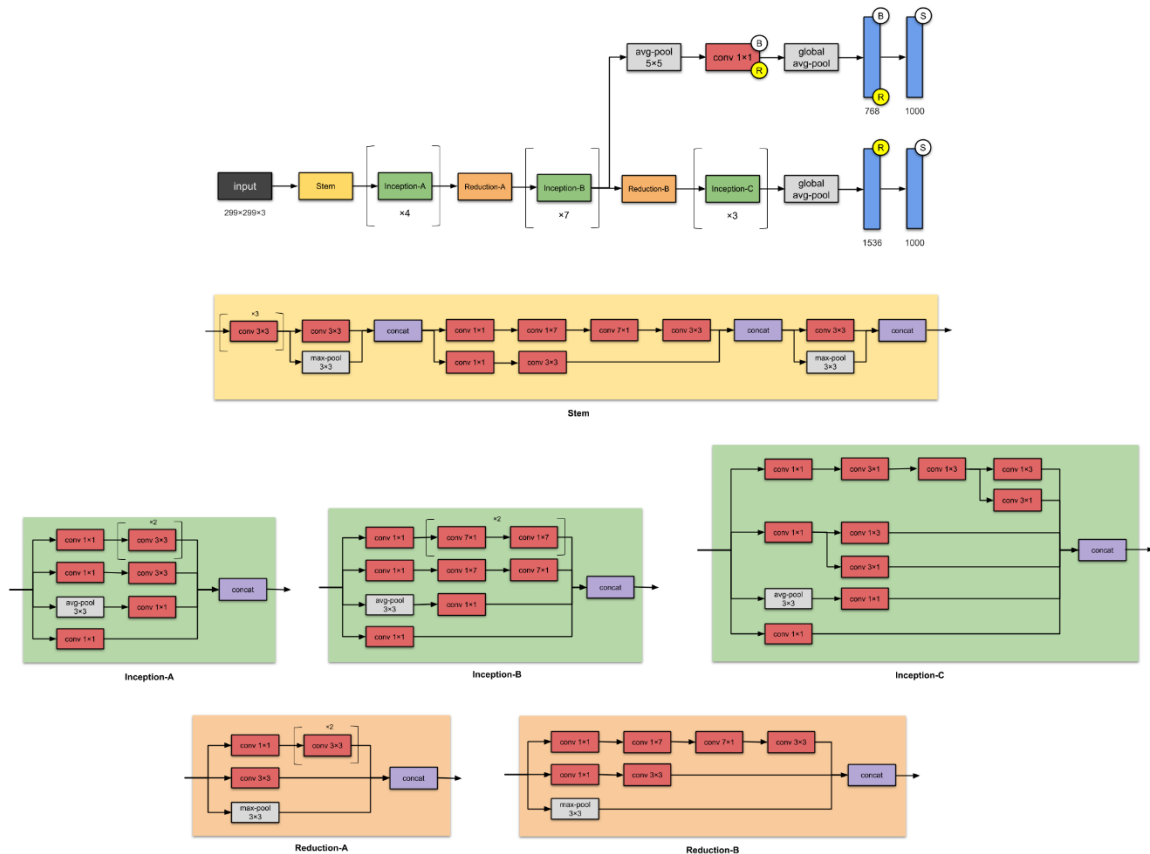


Рисунок 2.15 Архітектура моделі Inception [9]

### 2.6.5 ResNet

Коли кількість мережевих рівнів CNN зросла до певного числа, натомість продуктивність CNN знизилася. Причина полягає в тому, що глибші мережі набагато важче оптимізувати через горезвісну проблему зникнення / вибуху градієнтів.

Чи є спосіб збільшити кількість згорткових шарів, уникаючи проблеми вибуху / зникнення градієнта? У 2015 році для вирішення цієї проблеми було винайдено ResNet. [14]

ResNet представив концепцію, яка називається Залишкове навчання. Інтуїтивно висновок кожного шару згортки є принаймні таким же гарним, як і вхідний сигнал. тобто  $F(x) + x \geq x$  (Рисунок 2.16).

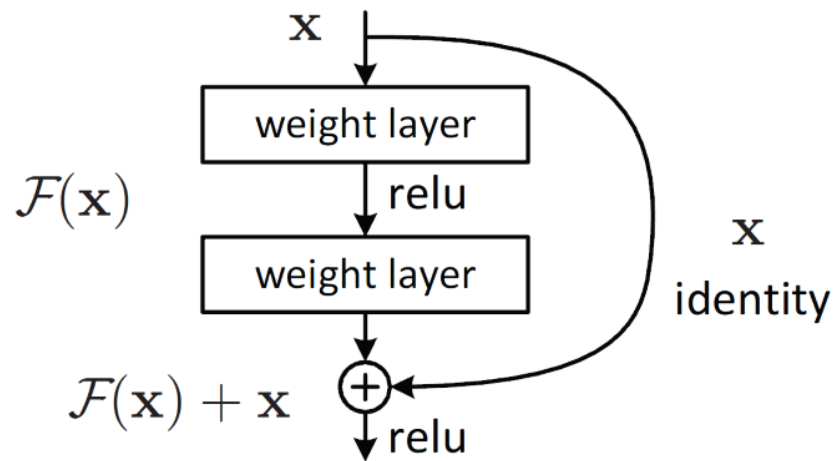


Рисунок 2.16 Залишкове навчання [14]

Доведено, що ця архітектура добре вирішує проблему градієнта. Найбільша кількість згорткових шарів ResNet може бути більше 1000 (Рисунок 2.17).

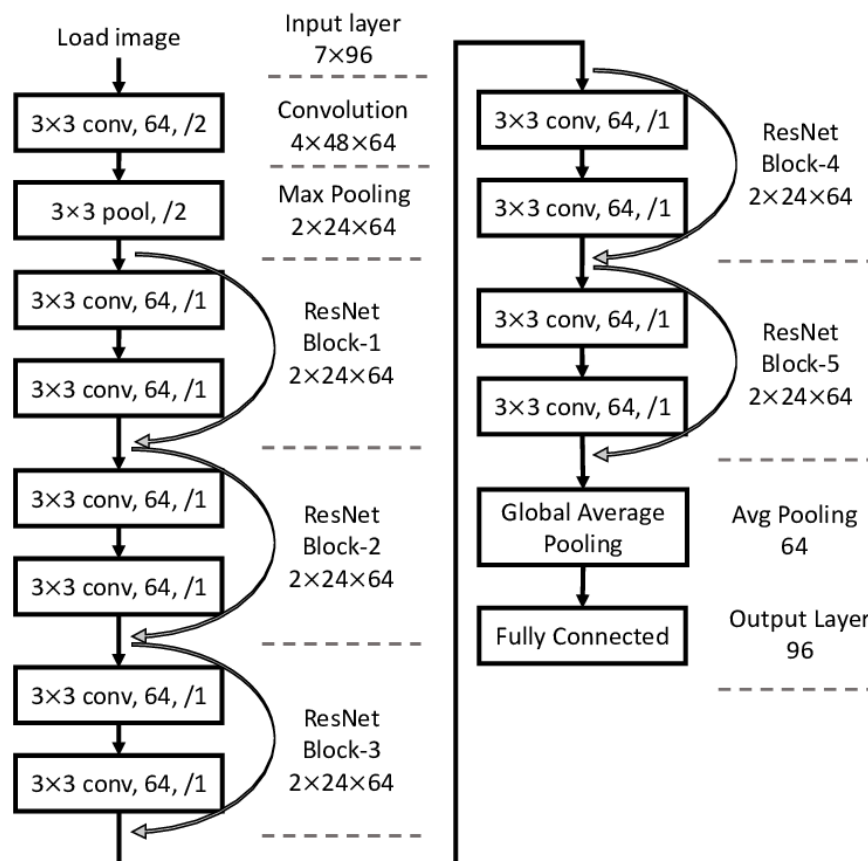


Рисунок 2.17 Архітектура ResNet [10]

## 2.7 Висновки до розділу 2

У другому розділі ми досить детально з'ясували основи нейронних мереж. Особлива увага була саме до згорткових нейронних мереж, тому що саме вони використовуються для задач класифікації зображень. Також з'ясували як саме комп'ютер інтерпретує вхідне зображення.

Було викладено базові поняття по архітектурі згорткових мереж, а також про шари згортки, об'єднання, активації, повно зв'язні шари і шари викидання.

Після цього були розглянуті базові принципи попередньої обробки зображень, які необхідно робити при роботі з CNN.

Також було викладено міркування стосовно оптимізаторів, які широко використовуються у глибокому навчанні. До кожного з них було наведено переваги та недоліки.

Наприкінці були наведені найбільш поширені архітектури згорткових нейронних мереж, а саме MobileNet, EfficientNet, VGG, Inception та ResNet. До кожного з них було наведено зображення з архітектурою, а також були пояснені деякі найбільш цікаві та незвичні модулі, завдяки яким ці мережі стали такими відомими.

## РОЗДІЛ 3. ПОБУДОВА СИСТЕМИ ДЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ ПОШКОДЖЕНЬ НА СКЛІ АВТОМОБІЛЯ МОДЕЛЯМИ НЕЙРОННИХ МЕРЕЖ

### 3.1 Архітектура системи та вибір мови програмування і фреймворків

Перед тим, як будувати архітектуру, слід спочатку згадати основні вимоги до системи:

- Необхідно, щоб до системи можна було звернутися будь з якого пристрою.
- Система не повинна вимагати завантаження будь яких додаткових модулів або бібліотек для своєї роботи.
- Різні компоненти не повинні залежати один від одного, щоб можна було розробляти їх паралельно.
- Система повинна мати зрозумілий користувальницький інтерфейс, оскільки основна аудиторія – це люди, які не дуже гарно працюють з комп'ютерами
- Система повинна мати зрозумілий програмний інтерфейс, щоб у майбутньому її можна було легко інтегрувати в іншу систему.

Кожне з вимог досить важливе і ми не можемо ігнорувати їх. Отже, щоб система задовольняла усім вимогам, було вирішено зробити веб додаток, який використовує попередньо навчені моделі згорткових нейронних мереж

Веб-додаток – це клієнт-серверний додаток, в якому клієнт взаємодіє з веб-сервером за допомогою браузера. Логіка веб-дodatка розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, а обмін інформацією відбувається по мережі. Одною з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є міжплатформовими сервісами.

Веб-додаток отримує запит від клієнта і виконує обчислення, після цього формує веб сторінку і відправляє її клієнтові мережею з використанням протоколу HTTP. [15]

Особливості такої моделі полягають в тому, що користувач відправляє певний запит на сервер, де той системно обробляється і кінцевий результат відсилається клієнту. Однією з основних переваг сервера є одночасне обслуговування відразу декількох клієнтів. Якщо одночасно надходить більше одного запиту, то такі запити встановлюються в певну чергу і сервер виконує їх по черзі.

Параметри, які реалізуються на стороні сервера:

- зберігання, захист і доступ до даних,
- обробка та валідація клієнтських запитів,
- відправка відповідей клієнту.

Параметри, які реалізуються на стороні клієнта:

- сторінка з інтерактивним графічним інтерфейсом,
- формулювання запиту до сервера і його подальша відправка,
- отримання відповіді від сервера та подальша її обробка. [16]

Тобто в нас є три глобальні компоненти – це клієнт, сервер та моделі CNN. Розглянемо їх детальніше.

### 3.1.1 Моделі нейронних мереж

Першим, та найважливішим компонентом, є саме згорткові нейронні мережі. Використані архітектури описані вище у розділі 2.6, тож нема потреби їх повторювати. Необхідно зазначити, що розробка велася у системі Google Colaboratory, професійне середовище програмування. Google Colaboratory – це безкоштовний хмарний сервіс на основі Jupyter Notebook. Основні переваги цього сервісу:

- не потребує довгої інсталяції та налаштування,
- дає доступ до графічних процесорів, які доволі коштовні, але набагато ефективніші,



- дозволяє легко інтегруватися з Google Drive, а це в свою чергу допомагає не бути прив'язаним до одного пристрою,
- має вбудовану історію змін, тож можна згадати, що було зроблено тиждень тому, навіть якщо це вже видалено.

#### 3.1.1.1 Мова програмування

Основною мовою для розробки нейронних мереж є Python. Python – це мова програмування високого рівня і саме тому досить простий для використання. Також оскільки Python написано на C++, він дуже швидкий, порівняно до інших мов програмування високого рівня.

#### 3.1.1.2 Фреймворк

Оскільки вже написано багато різних бібліотек для роботи з нейронними мережами, то сенсу писати усі функції, активатори та інше з нуля нема ніякого сенсу. Зручність та швидкість роботи з машинним навчанням сильно залежить від обраного фреймворка. На даний час, компанія Google є однією з передових у сфері нейронних мереж і нещодавно вони випустили нову версію свого фреймворка TensorFlow.

TensorFlow - це комплексна платформа з відкритим вихідним кодом для машинного навчання. Він має всеосяжну гнучку екосистему інструментів, бібліотек і ресурсів спільноти, яка дозволяє дослідникам просувати новітні досягнення в області машинного навчання, а розробникам легко створювати і розгортати додатки на основі машинного навчання.

TensorFlow 2.0 значно покращився, порівнюючи з попередньою версією. Було додано багато нових можливостей, які можуть дуже сильно покращити швидкість та якість роботи

### 3.1.2 Клієнт

Frontend (фронтенд або клієнтська частина програми) виконується в браузері користувача. Ця частина написана на мові програмування Javascript. Додаток може складатися тільки з клієнтської частини, а то й потрібно зберігати дані користувача довше однієї сесії. Це можуть бути, наприклад, фоторедактори або прості іграшки.

Single page application (SPA або односторінкове додаток). Більш цікавий варіант, коли використовуються і бекенд і фронтенд. За допомогою їх взаємодії можна створити додаток, яке буде працювати зовсім без перезавантажень сторінки в браузері. Або в спрощеному варіанті, коли переходи між розділами викликають перезавантаження, але будь-які дії в розділі обходяться без них. [17]

#### 3.1.2.1 Мова програмування

Оскільки клієнтська частина виконується в браузері, особливо варіантів немає, отже було використано Javascript. Альтернативою до нього є TypeScript, але Javascript більш гнучкий і з ним можна зробити додаток трохи швидше.

#### 3.1.2.2 Фреймворк

Цікавішим є вибір фреймворка, оскільки на мові Javascript дуже багато варіантів, як зробити веб сторінку. На поточний час найбільш популярними є три фреймворка: React, Angular та VueJs. React та Angular створені для доволі масштабних додатків зі складною бізнес логікою, тому було рішення зробити додаток на VueJs.

Vue - це прогресивна структура для побудови користувацьких інтерфейсів. На відміну від інших монолітних каркасів, Vue розроблений з нуля, задля швидкого застосування. Основна бібліотека орієнтована лише на шар представлення та її інтегрувати з іншими бібліотеками або існуючими проектами.

### 3.1.3 Сервер

#### 3.1.3.1 Мова програмування

Є багато мов для програмування бекенд частин веб додатку. Було вирішено працювати з Python, через дві причини. Перша – це тому, що моделі були створені за допомогою TensorFlow, який використовує Python і тому доволі логічно, що інтегрувати з бекенд частиною, що написана на Python, буде простіше усього. Другою причиною є вибір фреймворка. Інші мови програмування потребують доволі великої та складної архітектури серверної частини, але існує фреймворк, який дозволяє зробити дуже швидкий старт і не робити багато шарів абстракцій для коректної роботи.

#### 3.1.3.2 Фреймворк

Flask - це мікро-фреймворк, написаний на Python. Він класифікується як мікро-фреймворк, оскільки для нього не потрібні певні інструменти чи бібліотеки. Він не має рівня абстракції бази даних, перевірки форми або будь-яких інших компонентів, де вже існуючі сторонні бібліотеки забезпечують загальні функції. Однак Flask підтримує розширення, які можуть додавати функції програми так, ніби вони реалізовані в самому Flask. Саме тому він є гарним рішенням для інтеграції з TensorFlow моделями.

### 3.2 Порівняльний аналіз навчених моделей

Розглянемо різні моделі та проаналізуємо результати їх роботи, а після зробимо порівняльну таблицю, щоб зрозуміти, яка модель показала себе краще за всіх.

Для всіх моделей умови запуску були однакові:

- Дані аугментовані на випадковий поворот та віддзеркалення,
- Функція втрат – `SparseCategoricalCrossentropy`,
- Оптимізатор – Адам,
- Learning rate –  $3E-4$ ,
- Розмір батча – 64,
- Кількість епох – 80.

Хоча кількість епох була задана, як 80, була використана технологія `EarlyStopping`. Вона дозволяє стежити за обраною метрикою, і якщо ця метрика не покращується за декілька епох, `EarlyStopping` зупиняє навчання мережі. Цей підхід дозволяє уникнути проблеми перенавчання.

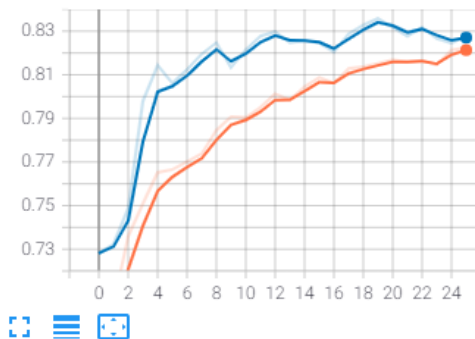
Для кожної з архітектур буде приведено два графіка. Перший – це графік точності, другий – графік втрат. На кожному з графіків дві лінії. Помаранчева лінія – це графік тренувальних даних. Синя лінія – це графік валідаційних даних.

### 3.2.1 Inception

Оскільки була обрана попередньо навчена мережа, валідаційні графіки трохи кращі, ніж навчальні. Графіки сходяться один до одного, що каже про гарне навчання мережі (Рисунок 3.1). Як бачимо результат доволі швидко покращився і вже через 4 епохи точність мережі покращилася на 8%.

epoch\_accuracy

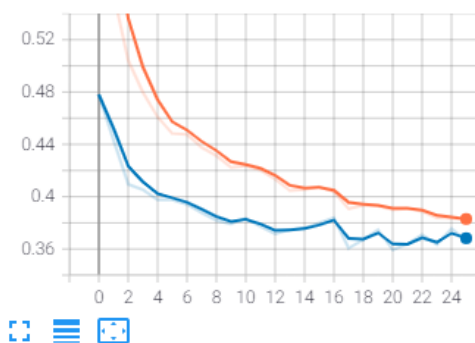
epoch\_accuracy  
tag: epoch\_accuracy



🔍 ☰ 🖨

epoch\_loss

epoch\_loss  
tag: epoch\_loss



🔍 ☰ 🖨

Рисунок 3.1 Графік навчання Inception

Навчання було припинене на 24й епосі, тому що починаючи з 16ї епохи валідаційна функція втрат не покращилася.

Результати на кінець навчання:

Таблиця 3.1 – Результати навчання Inception

	Train	Validation	Test
Accuracy	0.8225	0.8277	0.8441
Loss	0.3822	0.3662	0.3416

### 3.2.2 EfficientNet

Оскільки була обрана попередньо навчена мережа, валідаційні графіки трохи кращі, ніж навчальні. Графіки сходяться один до одного, що каже про гарне навчання мережі (Рисунок 3.2). Як бачимо результат доволі швидко покращився і вже через 4 епохи точність мережі покращилася на 5%.



Рисунок 3.2 Графік навчання EfficientNet

Навчання було припинене на 12й епосі, тому що починаючи з 8ї епохи валідаційна функція втрат не покращилася.

Результати на кінець навчання:

Таблиця 3.2 – Результати навчання EfficientNet

	Train	Validation	Test
Accuracy	0.8434	0.8226	0.8584
Loss	0.3503	0.8434	0.3357

### 3.2.3 ResNet

Оскільки була обрана попередньо навчена мережа, валідаційні графіки трохи кращі, ніж навчальні. Графіки сходяться один до одного, що каже про гарне навчання мережі (Рисунок 3.3). Серед усіх мереж, ця архітектура продемонструвала найсильніше та найстабільніше покращення метрик

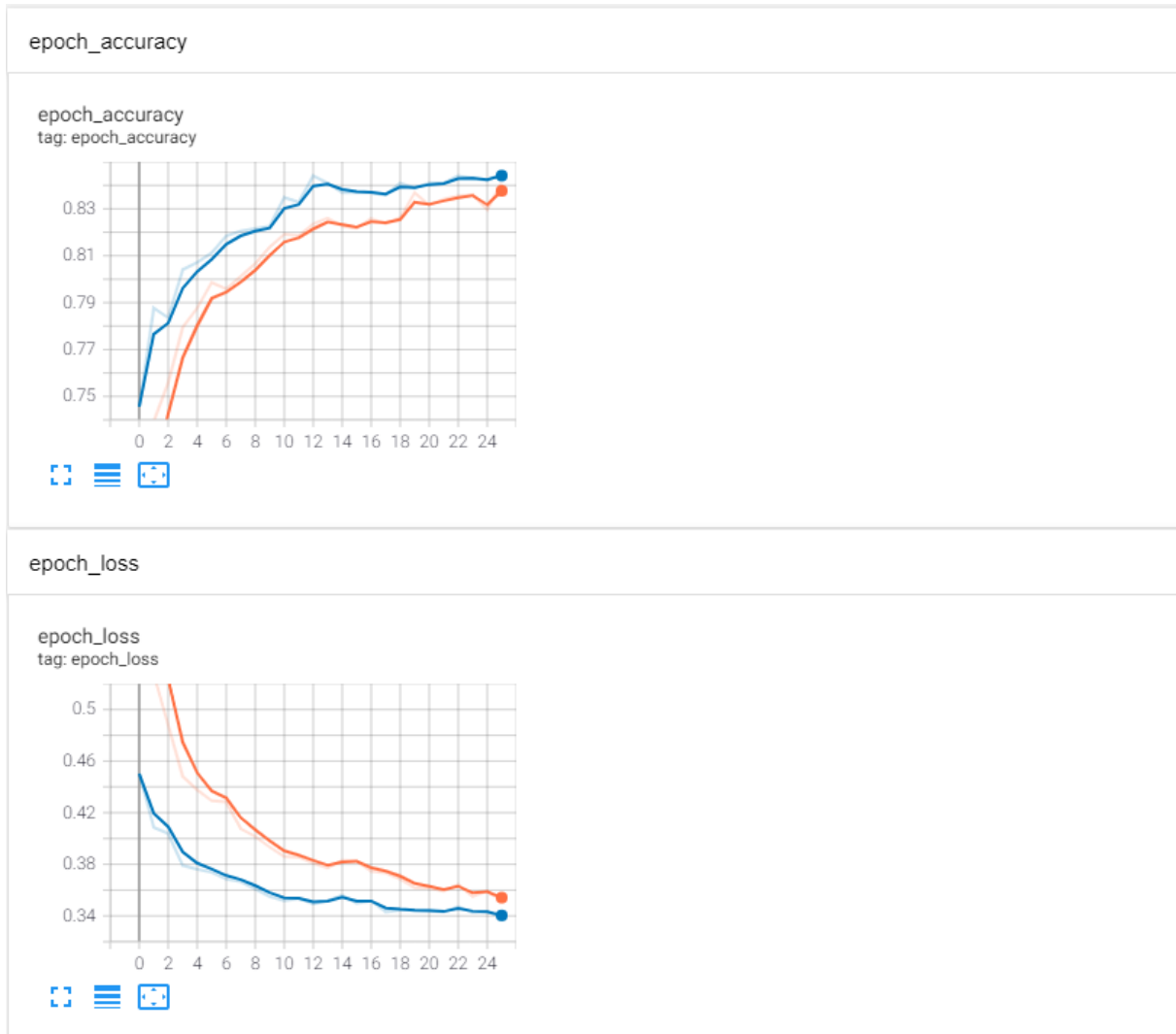


Рисунок 3.3 Графік навчання ResNet

Навчання було припинене на 24й епосі, тому що починаючи з 18ї епохи валідаційна функція втрат не покращилася.

Результати на кінець навчання:

Таблиця 3.3 – Результати навчання ResNet

	Train	Validation	Test
Accuracy	0.8410	0.8451	0.8471
Loss	0.3516	0.3387	0.3306



### 3.2.4 MobileNet

Оскільки була обрана попередньо навчена мережа, валідаційні графіки трохи кращі, ніж навчальні. Графіки сходяться один до одного, що каже про гарне навчання мережі (Рисунок 3.4). Серед усіх архітектур, ця показала найповільніше покращення метрик.

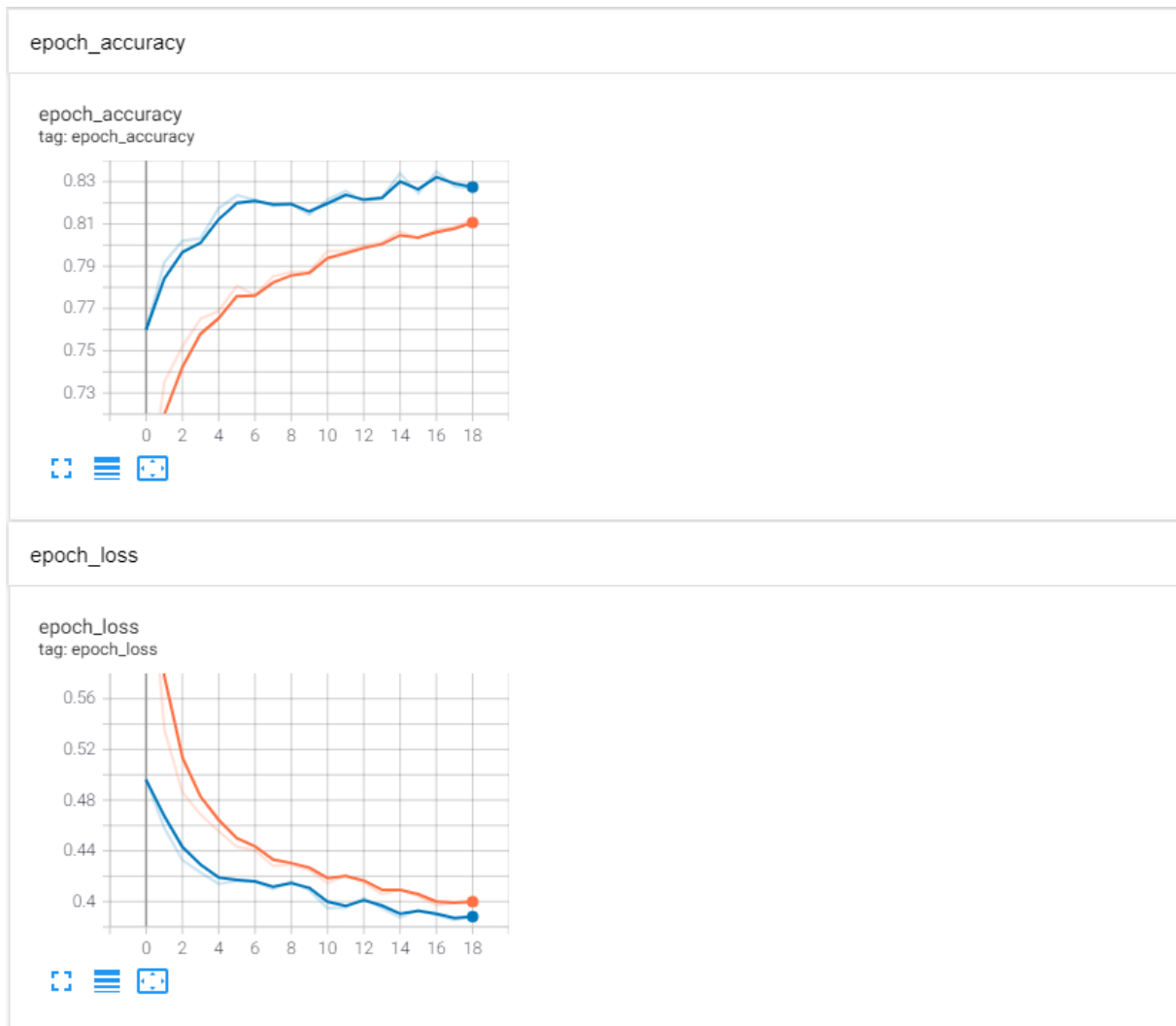


Рисунок 3.4 Графік навчання MobileNet

Навчання було припинене на 18й епосі, тому що починаючи з 14ї епохи валідаційна функція втрат не покращилася.

Результати на кінець навчання:

Таблиця 3.4 – Результати навчання MobileNet

	Train	Validation	Test
Accuracy	0.8120	0.3886	0.8482
Loss	0.4002	0.8267	0.3491

### 3.2.5 NasNet

Оскільки була обрана попередньо навчена мережа, валідаційні графіки трохи кращі, ніж навчальні. Графіки сходяться один до одного, що каже про гарне навчання мережі (Рисунок 3.5). Як бачимо результат доволі швидко покращився і вже через 4 епохи точність мережі покращилася на 6%.

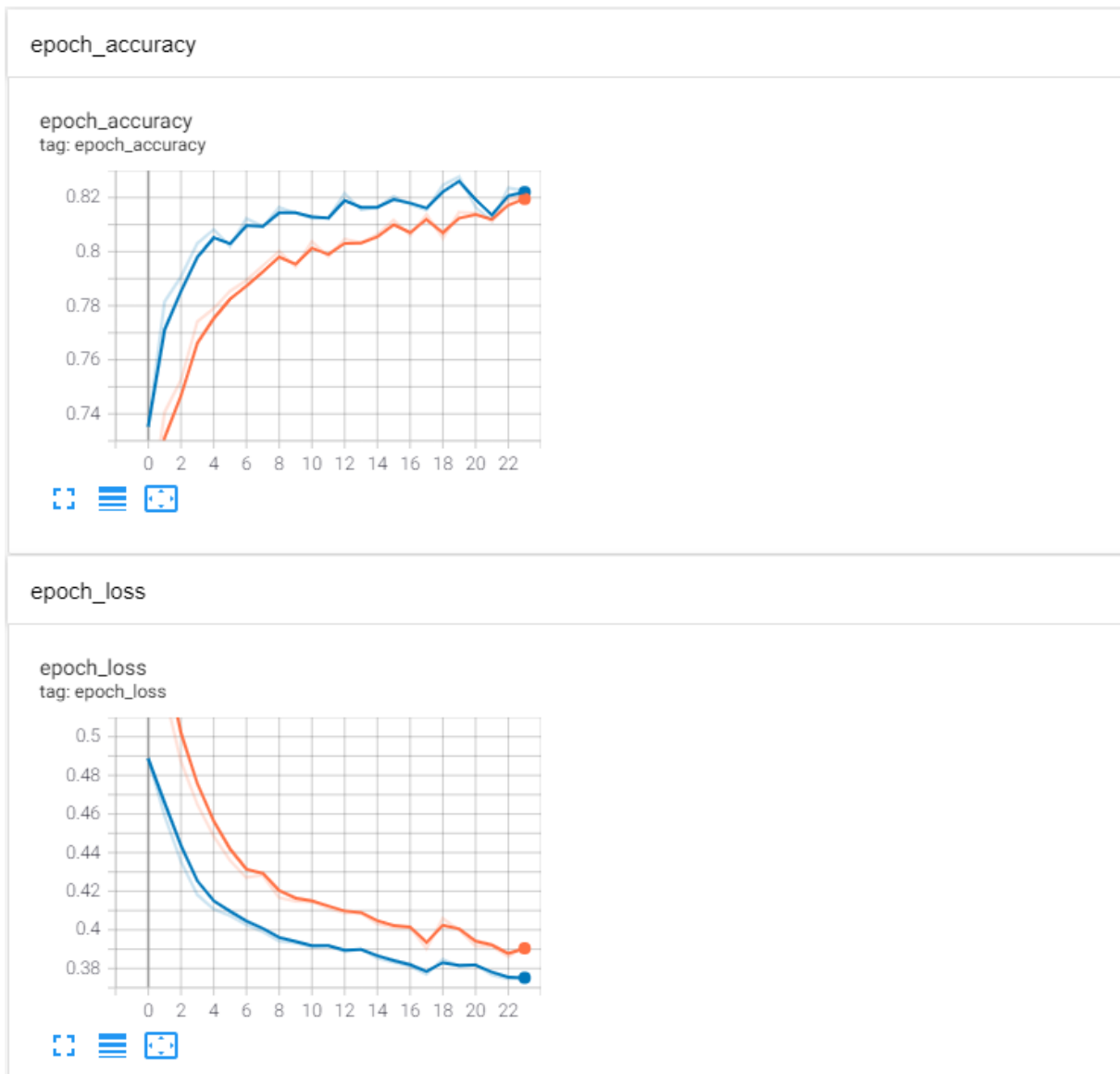


Рисунок 3.5 Графік навчання NasNet

Навчання було припинене на 22й епосі, тому що починаючи з 17ї епохи валідаційна функція втрат не покращилася.

Результати на кінець навчання:

Таблиця 3.5 – Результати навчання NasNet

	Train	Validation	Test
Accuracy	0.8205	0.8226	0.8461
Loss	0.3915	0.3750	0.3537

### 3.2.6 Порівняльна таблиця


Таблиця 3.6 – Порівняння різних архітектур

Архітектура	Test accuracy	Test loss
Inception	0.8441	0.3416
EfficientNet	0.8584	0.3357
ResNet	0.8471	0.3306
MobileNet	0.8482	0.3491
NasNet	0.8461	0.3537
<b>Краща</b>	<b>EfficientNet</b>	<b>ResNet</b>


### 3.3 Приклад роботи веб додатку

Ми оглянули навчені моделі і настав час подивитися, як працює веб-додаток. Отже, на даний час, є одна єдина сторінка, на якій можна обрати зображення, яке ми бажаємо прокласифікувати, потім обрати модель, за допомогою якої буде робитися класифікація і після цього отримати результат. Розглянемо трохи детальніше.


На рисунку 3.6 ми бачимо, як початковий стан роботи додатку. По-перше нам пропонують завантажити фото, яна якому є пошкодження. Після кліку на поле під назвою “Image input” відкриється діалогове вікно, де ми можемо обрати зображення. Після того, як зображення буде завантажено, нам пропонується виділити на ньому область з пошкодженням.


**Auto cracks classification**

1. Please choose photo, that contains some cracks on auto window


 Image input


2. Please select the damaged area in the photo.



preview area

3. Please choose model, that you want to use for classification

Inception\_v3 (pretrained)

☐ Smart decision 

CLASSIFY!

Рисунок 3.6 Робота веб додатка. Початковий стан

Останнім пунктом є вибір моделі для класифікації. При кліку відкриється випадаючий список з усіма описаними вище моделями. Також можна обрати пункт «Розумний вибір», яка получити результати усіх моделей, після цього візьме середнє арифметичне виведе той результат, який більша кількість моделей обрала.

Після усіх приготувань необхідно лише натиснути кнопку «Класифікувати!», яка відправить запит на сервер і отримає результат моделі (Рисунок 3.7).

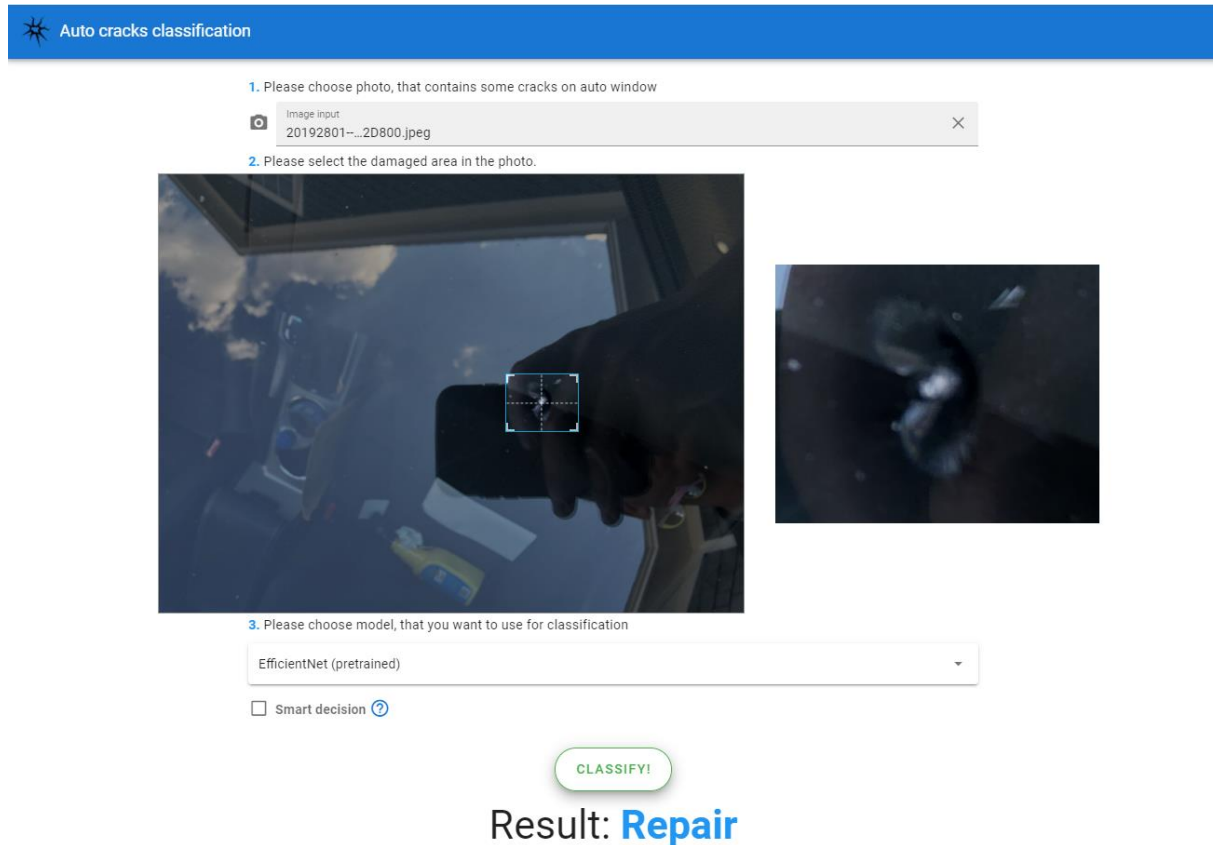


Рисунок 3.7 Робота веб додатка. Кінцевий вигляд

### 3.4 Висновки до розділу 3

В даному розділі було розглянуто архітектуру практичної частини, було наведене обґрунтування обраних технологій, що базується на основних вимогах системи, а також розглянуто технології, які використовуються на клієнті, принцип роботи сервера та взаємодію між ними.

Для демонстрації практичної частини було обрано веб-додаток на основі клієнт-серверної архітектури, що використовує попередньо навчені моделі. Моделі були написані та навчені засобами фреймворку TensorFlow 2.0. Клієнтську частину було вирішено зробити за допомогою фреймворка швидкого застосування Vue Js. Серверну частину було вирішено зробити на основі фреймворку Flask, тому що за його допомогою легко інтегруватися із TensorFlow та іншими майбутніми модулями.

В наступному підпункті були описані початкові дані, а також описано прийом, який дозволяє уникнути перенавчання. і проведено порівняльний

аналіз навчених моделей, також було наведено графіки навчання, а саме порівняння валідаційних та тренувальних графіків втрат та точності. Після кожного підпункту з архітектурою моделі було наведено таблицю, де порівнюються тренувальні, валідаційні та тестові метрики наприкінці навчання. Після цього було наведено порівняльну таблицю з усіма моделями.

Останнім пунктом була показана та описана робота веб додатка.

## РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для класифікації пошкоджень на склі авто за допомогою згорткових нейронних мереж.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

### 4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи класифікації пошкоджень на склі автомобіля. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема



підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних для класифікації .

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.
- 

#### 4.2 Обґрунтування функцій програмного продукту

Головна функція  $F_0$  – розробка програмного продукту, який вирішує задачу класифікації пошкоджень на склі авто та будує його модель. Беручи за основу цю функцію, можна виділити наступні:

$F_1$  – вибір мови програмування;

$F_2$  – вибір фреймворку машинного навчання;

$F_3$  – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція  $F_1$ :

а) Python

б) C++

Функція  $F_2$ :

а) Tensorflow;

б) Pytorch

Функція  $F_3$ :

- a) Jupyter Notebook;
- б) Visual Studio.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

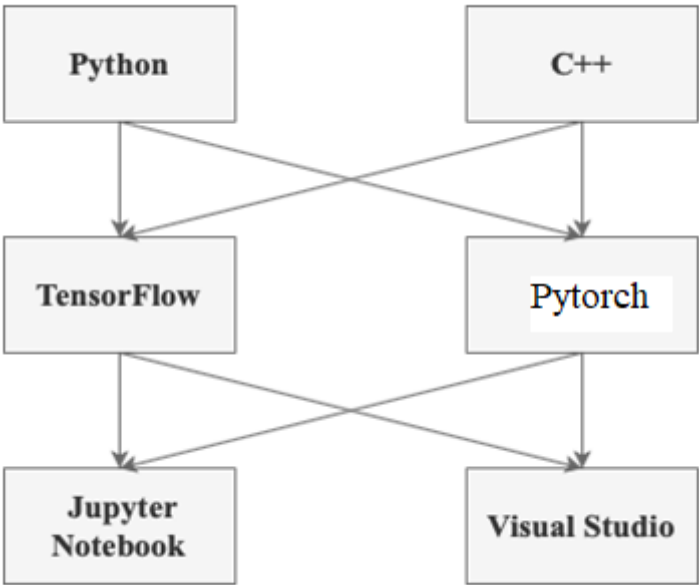


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
---------	---------------------	----------	----------

$F_1$	$A$	Швидка розробка програми, доступність бібліотек, кросплатформеність	Низька швидкість роботи, особливо, якщо потрібно обробляти велику кількість даних
	$B$	Код швидко виконується	Іде багато часу на розробку програми
$F_2$	$A$	Надійно працює з складними проектами	Не підтримується багатьма мовами
	$B$	Надійність	Додатковий час на інсталяцію та вивчення
$F_3$	$A$	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері	Відсутня можливість роботи без інтернету
	$B$	Багато інструментів, безпечна	Підтримує одночасно лише одну мову програмування

На основі цієї карти будуюмо позитивно-негативну матрицю варіантів основних функцій (Таблиця 4.1). Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція  $F_1$ :

Перевагу віддаємо швидкості вивчення, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант  $B$  має бути відкинутий.

Функція  $F_2$ :

Обидва варіанти можна використовувати в розробці.

Функція  $F_3$ :

Віддаємо перевагу варіанту А в разі вибору мови програмування Python.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1a - F_2a - F_3a$$

$$F_1a - F_2б - F_3a$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

#### 4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- $X1$  – швидкодія мови програмування;
- $X2$  – об’єм пам’яті для обчислень та збереження даних;
- $X3$  – час навчання даних;
- $X4$  – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об'єм пам'яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об'єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

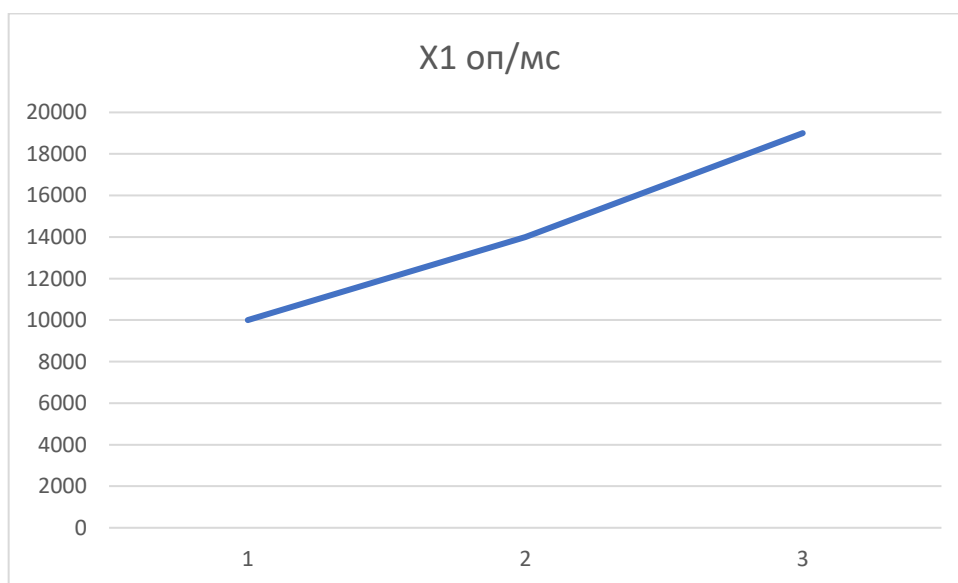


Рисунок 4.2 – X1, швидкодія мови програмування

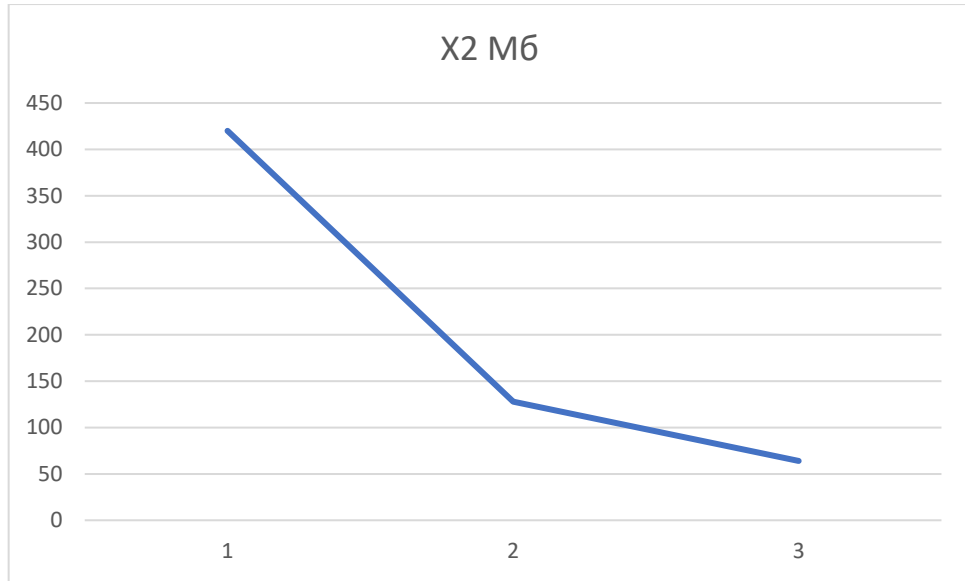


Рисунок 4.3 – X2, об'єм пам'яті

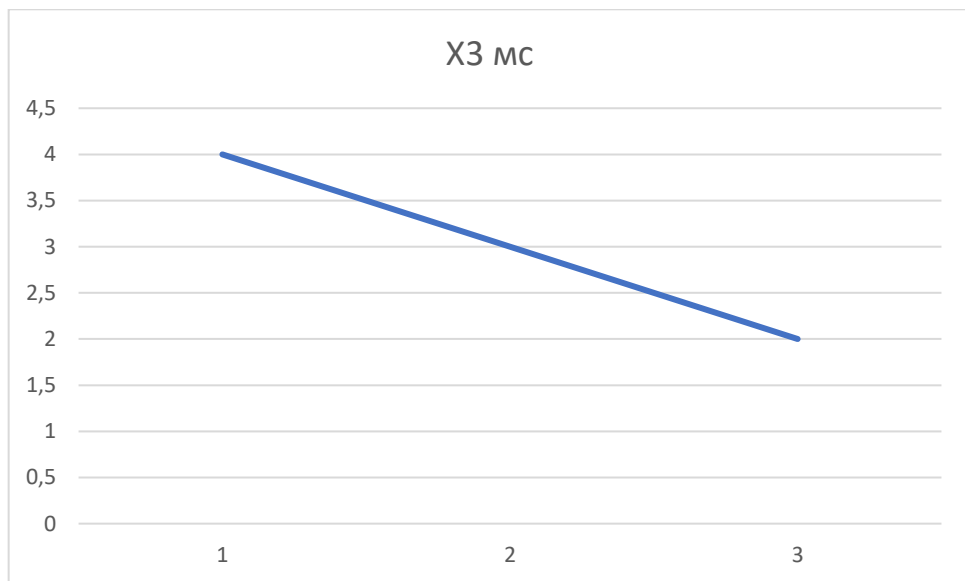


Рисунок 4.4 – X3, час попередньої обробки даних

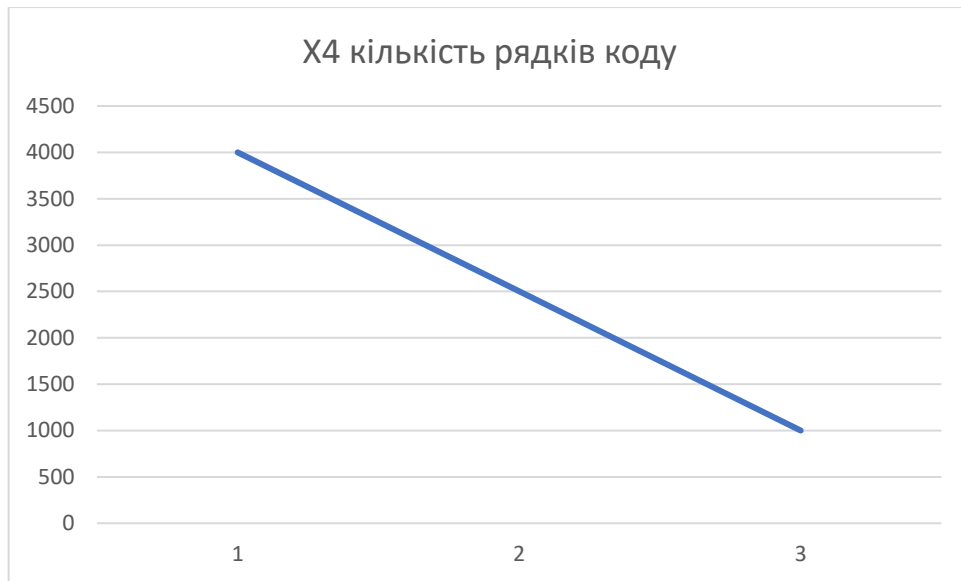


Рисунок 4.5 – X4, потенційний об'єм програмного коду

#### 4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів $R_i$	Відхилення $\Delta_i$	$\Delta_i^2$
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
X2	Об'єм пам'яті	Мб	2	1	3	1	2	1	2	12	-12,5	156,25
X3	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5,5	30,25
X4	Потенційний об'єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:



$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \quad (4.1)$$

де  $N$  – число експертів,

$n$  – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24,5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всіх параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 211. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0,86 > W_k = 0,67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	<	>	>	>	>	1,5
X1 і X3	<	>	<	=	<	<	>	<	0,5
X1 і X4	>	>	<	=	<	=	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	>	<	>	<	>	1,5

Числове значення, що визначає ступінь переваги  $i$ -го параметра над  $j$ -тим,  $a_{ij}$  визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю  $A = \|a_{ij}\|$ .

Для кожного параметра зробимо розрахунок вагомості  $K_{ei}$  за наступними формулами:

$$K_{Bi} = \frac{b_i}{\sum_{i=1}^n b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^N a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{Bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \quad (4.9)$$

$$b'_i = \sum_{i=1}^N a_{ij} b_j \quad (4.10)$$

.Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметрих <sub>i</sub>	Параметрих <sub>j</sub>				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	$b_i$	$K_{Bi}$	$b_i^1$	$K_{Bi}^1$	$b_i^2$	$K_{Bi}^2$
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23
X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32

X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2
Всього:					12,5	1	70	1	290,25	1

#### 4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів  $X2$  (Об'єм пам'яті),  $X3$  (час попередньої обробки даних) та  $X4$  (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра  $X1$  (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j}, \quad (4.11)$$

де  $n$  – кількість параметрів;

$K_{ei}$  – коефіцієнт вагомості  $i$ -го параметра;

$B_i$  – оцінка  $i$ -го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	10000	6	0,22	1,15
F2	A	X2	64	5	0,33	1,45
	Б	X2	128	2	0,3	0,8
F3	A	X3	1000	8	0,15	1,36

За даними з таблиці 5.7 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1,15 + 1,45 + 1,36 = 3,96,$$

$$K_{K2} = 1,15 + 0,8 + 1,36 = 3,31.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

#### 4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_0 = T_P \cdot K_{\Pi} \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де  $T_P$  – трудомісткість розробки ПП;

$K_{\Pi}$  – поправочний коефіцієнт;

$K_{СК}$  – коефіцієнт на складність вхідної інформації;

$K_M$  – коефіцієнт рівня мови програмування;

$K_{СТ}$  – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$  – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює:  $T_p = 90$  людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання:  $K_{\Pi} = 1.7$ . Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1:  $K_{СК} = 1$ . Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта  $K_{СТ} = 0.8$ . Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто  $T_p = 27$  людино-днів,  $K_{\Pi} = 0.9$ ,  $K_{СК} = 1$ ,  $K_{СТ} = 0.8$ :

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328.64 \text{ людино-годин.}$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 20000 грн., один аналітик в області даних з окладом 18000. Визначимо середню зарплату за годину за формулою:

$$C_{\text{ч}} = \frac{M}{T_m \cdot t} \text{ грн.}, \quad (4.14)$$

де  $M$  – місячний оклад працівників;

$T_m$  – кількість робочих днів тижень;

$t$  – кількість робочих годин в день.

$$C_{\text{ч}} = \frac{20000 + 20000 + 18000}{3 \cdot 21 \cdot 8} = 115,07 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{\text{зп}} = C_{\text{ч}} \cdot T_i \cdot K_d, \quad (4.16)$$

де  $C_{\text{ч}}$  – величина погодинної оплати праці програміста;

$T_i$  – трудомісткість відповідного завдання;

$K_d$  – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I.} \quad C_{\text{зп}} = 115,07 \cdot 1328,64 \cdot 1,2 = 183463,92 \text{ грн.}$$

$$\text{II.} \quad C_{\text{зп}} = 115,07 \cdot 1345,52 \cdot 1,2 = 185794,78 \text{ грн.}$$



Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 183463,92 \cdot 0,22 = 40362,06 \text{ грн.}$$

$$\text{II. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 185794,78 \cdot 0,22 = 40874,85 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ( $C_M$ )

Так як одна ЕОМ обслуговує одного програміста з окладом 20000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_G = 12 \cdot M \cdot K_3 = 12 \cdot 20000 \cdot 0,2 = 48000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} = C_G \cdot (1 + K_3) = 48000 \cdot (1 + 0.2) = 57600 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 57600 \cdot 0,22 = 12672 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 30% та вартості ЕОМ – 32000 грн.

$$C_A = K_{\text{ТМ}} \cdot K_A \cdot \text{Ц}_{\text{ПР}} = 1.15 \cdot 0.3 \cdot 32000 = 11040 \text{ грн.,}$$

де  $K_{\text{ТМ}}$  – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$K_A$ — річна норма амортизації;

$\Pi_{\text{ПР}}$ — договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{\text{ТМ}} \cdot \Pi_{\text{ПР}} \cdot K_P = 1.15 \cdot 32000 \cdot 0.05 = 1840 \text{ грн.},$$

де  $K_P$ — відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\begin{aligned} T_{\text{ЕФ}} &= (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.9 = \\ &= 1677.6 \text{ годин}, \end{aligned}$$

де  $D_K$ — календарна кількість днів у році;

$D_B, D_C$ — відповідно кількість вихідних та святкових днів;

$D_P$ — кількість днів планових ремонтів устаткування;

$t$ —кількість робочих годин в день;

$K_B$ — коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_C \cdot K_3 \cdot \Pi_{\text{ЕН}} = 1677.6 \cdot 0.3 \cdot 0.8 \cdot 3,51547 = 1415.41 \text{ грн.},$$

де  $N_C$ — середньо-споживча потужність приладу;

$K_3$ — коефіцієнтом зайнятості приладу;

$\text{Ц}_{\text{ЕН}}$  – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_{\text{Н}} = \text{Ц}_{\text{ПР}} \cdot 0.67 = 32000 \cdot 0.67 = 21440 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}, \quad (4.17)$$

$$C_{\text{ЕКС}} = 57600 + 12672 + 11040 + 1150 + 1415.41 + 21440 = 105317.41 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 105317.41 / 1677.6 = 62.77 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T, \quad (4.18)$$

$$\text{І.} \quad C_{\text{М}} = 62.77 \cdot 1328.64 = 83398.73 \text{ грн.}$$

$$\text{II. } C_M = 62,77 \cdot 1345,52 = 84458,29 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{3П} \cdot 0,67, \quad (4.19)$$

$$\text{I. } C_H = 183463,92 \cdot 0,67 = 122920,82 \text{ грн.}$$

$$\text{II. } C_H = 185794,78 \cdot 0,67 = 124482,50 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{3П} + C_{ВІД} + C_M + C_H, \quad (5.20)$$

$$\text{I. } C_{ПП} = 183463,92 + 40362,06 + 83398,73 + 122920,82 = 430145,53 \text{ грн.}$$

$$\text{II. } C_{ПП} = 185794,78 + 40874,85 + 84458,29 + 124482,50 = 435610,42 \text{ грн.}$$

#### 4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{TEPj} = K_{Kj} / C_{Фj}, \quad (5.21)$$

$$K_{TEP1} = 3,96 / 430145,53 = 9,2 \cdot 10^{-6},$$

$$K_{TEP2} = 3,31 / 435610,42 = 7,5 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня  $K_{\text{ТЕР1}} = 9,2 \cdot 10^{-6}$ .

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишилися після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості  $K_{\text{ТЕР}} = 9,2 \cdot 10^{-6}$ .

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- Використання моделей з великою ємністю
- Використання стандартного інтерфейсу візуалізації, швидкість розробки

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

#### 4.8 Висновки до розділу 4

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

## ВИСНОВКИ

В даній дипломній роботі були розглянуті методи класифікації пошкоджень на склі автомобіля, за допомогою згорткових нейронних мереж. В першому розділі була розглянута актуальність задачі та перспективи розробки. Також були описані та проаналізовані вхідні дані, а також наведено приклади до них. Була описана задача класифікації об'єктів, зокрема зображень, так викладені твердження, щодо цього типу задач. Окрім цього була проведена формальна та фактична постановка задачі і під кінець розділу були описані технічні вимоги до системи.

У другому розділі увага була сконцентрована на математичних основах роботи. Були наведені базові поняття нейронних мереж та згорткових нейронних мереж. Окрім цього було визначено, як комп'ютер «бачить» дані йому зображення. Після цього була детально описана структура нейронних мереж. Наведені приклади базових шарів згорткових мереж, описані найвідоміші оптимізатори і наведені деякі методи, що використовуються задля покращення роботи мережі. Під кінець розділу були показані та описані існуючі архітектури згорткових нейронних мереж, було наведено рисунки задля кращого розуміння їх архітектур.

Третій розділ було присвячено саме практичній частині роботи. Перш за все були згадані вимоги до системи та на їх основі були обрані програмні методи та засоби для роботи, зокрема мова програмування, фреймворк та середовище програмування. Після цього була наведена та описана обрана архітектура, було вказано особливості роботи та основні переваги і недоліки такої системи. Передостаннім пунктом було продемонстровано навчені нейронні мережі і було проведено порівняння їх результатів. В кінці розділу було показано роботу розробленого веб додатку.

В останньому розділі було розглянуто економічні показники для двох варіантів реалізації та обрано найоптимальніший із них.

Розроблений програмний продукт є рішенням реальної бізнес проблеми та є повністю функціонуючим та повноцінним продуктом. Веб додаток є в першу чергу рішенням страхових компаній, але може використовуватися і в персональних цілях.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Классификация. URL:  
<http://www.machinelearning.ru/wiki/index.php?title=%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F>
2. Как сформировать датасет для машинного обучения URL:  
<https://www.bigdataschool.ru/blog/dataset-data-preparation.html>
3. Image Classification with Convolutional Neural Networks URL:  
[https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8#:~:text=Convolutional%20neural%20networks%20\(CNN\)%20is,this%20architecture%20is%20image%20classification.&text=Instead%20of%20the%20image%2C%20the%20computer%20sees%20an%20array%20of%20pixels](https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8#:~:text=Convolutional%20neural%20networks%20(CNN)%20is,this%20architecture%20is%20image%20classification.&text=Instead%20of%20the%20image%2C%20the%20computer%20sees%20an%20array%20of%20pixels)
4. A Comprehensive Guide to Convolutional Neural Networks URL:  
<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
5. Гудфеллоу Я., Бенджио И., Курвилль А.. Глубокое обучение. пер. с англ. А. А. Слинкина. 2-е изд., испр. М.: ДМК Пресс, 2018, 652 с.
6. Convolutional Neural Networks — Image Classification w. Keras URL: <https://www.learndatasci.com/tutorials/convolutional-neural-networks-image-classification/>
7. Basic Overview of Convolutional Neural Network (CNN) URL:  
[https://medium.com/dataserries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17#:~:text=The%20activation%20function%20is%20a,neuron%20would%20fire%20or%20not.&text=We%20have%20different%20types%20of,Rectified%20Linear%20Unit%20\(ReLU\)](https://medium.com/dataserries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17#:~:text=The%20activation%20function%20is%20a,neuron%20would%20fire%20or%20not.&text=We%20have%20different%20types%20of,Rectified%20Linear%20Unit%20(ReLU))



8. Various Optimization Algorithms For Training Neural Network URL:  
<https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>
9. Illustrated: 10 CNN Architectures URL:  
<https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d>
10. Image Classification With MobileNet URL:  
<https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470>
11. EfficientNet URL:  
<https://paperswithcode.com/method/efficientnet#:~:text=EfficientNet%20is%20a%20convolutional%20neural,resolution%20using%20a%20compound%20coefficient>
12. EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling URL:  
<https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html>
13. Convolutional Neural Network Model Innovations for Image Classification URL: <https://machinelearningmastery.com/review-of-architectural-innovations-for-convolutional-neural-networks-for-image-classification/>
14. From AlexNet to NASNet: A Brief History and Introduction of Convolutional Neural Networks URL:  
<https://towardsdatascience.com/from-alexnet-to-nasnet-a-brief-history-and-introduction-of-convolutional-neural-networks-cf63bf3320e1>
15. Web application URL: [https://en.wikipedia.org/wiki/Web\\_application](https://en.wikipedia.org/wiki/Web_application)
16. Основные понятия и особенности клиент-серверной архитектуры URL: <https://testmatick.com/ru/osnovnye-ponyatiya-i-osobennosti-klient-servernoj-arhitektury/>

17. Как работают веб приложения URL:

<https://habr.com/ru/post/450282/>

## СПИСОК НАУКОВИХ ДОСЯГНЕНЬ

1. Славінський В.О. Система класифікації пошкоджень на склі автомобіля за допомогою згорткових нейронних мереж. Системні науки та кібернетика. №1. с.45 - 65.

## ДОДАТОК А ЛІСТІНГ ПРОГРАМИ

Головний файл з навчанням моделей

- tensorflowResearchPaper.py

```
# -*- coding: utf-8 -*-
"""TensorflowResearchPaper.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1xvWjRG-HlbvgFTuYqThovzApJ04nr8zr

## Imports
"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import os
import tensorflow as tf
import cv2
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
import tensorflow_datasets as tfds
from matplotlib import pyplot as plt
import PIL
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from datetime import datetime

# %matplotlib inline
# %load_ext tensorboard
from PIL import Image
from matplotlib import cm
import keras.backend as K

import sys,os
import json
import matplotlib.image as mpimg
from PIL import Image, ImageDraw

import tensorflow_hub as hub

"""

## Env setup (gdrive mount and dataset build)
### ***Skip, if local***"""
```

```

from google.colab import drive
drive.mount("/content/gdrive")

# Commented out IPython magic to ensure Python compatibility.
!dir
# %cd gdrive/MyDrive/FIG/Models/
#%cd /content/gdrive/MyDrive/Legendari/FIG/Dataset/my_dataset
!dir

# Commented out IPython magic to ensure Python compatibility.
# %mkdir prod
# %cd prod

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Select the Runtime > "Change runtime type" menu to enable a GPU accelera
tor, ')
    print('and then re-execute this cell.')
else:
    print(gpu_info)

"""## Dataset info

"""

data_dir = os.path.normpath("/content/gdrive/MyDrive/FIG/Dataset/build")
data_dir

(train_dataset, val_dataset, test_dataset), metadata = tfds.load(
    'auto_cracks_dataset',
    data_dir=data_dir,
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True
)

"""## Helper functions and constants"""

IMG_SIZE = 224
num_classes = metadata.features['label'].num_classes

get_label_name = metadata.features['label'].int2str
def show_imgs(ds, count = 3):
    itr = iter(ds)
    plt.figure(figsize=(10, 10))
    for i in range(count**2):
        image, label = next(itr)
        ax = plt.subplot(count, count, i + 1)
        plt.imshow(image)

```

```

plt.title(get_label_name(label))
plt.axis("off")

def show_imgs_only_one_class(ds, class_name, count = 3):
    itr = iter(ds)
    plt.figure(figsize=(10, 10))
    for i in range(count**2):
        image, label = next(itr)
        while get_label_name(label) != class_name:
            image, label = next(itr)

        ax = plt.subplot(count, count, i + 1)
        plt.imshow(image)
        plt.title(str(get_label_name(label)))
        plt.axis("off")

show_imgs(test_dataset)

show_imgs_only_one_class(val_dataset, 'Replacement')

show_imgs_only_one_class(val_dataset, 'Repair')

def get_callbacks(directory):
    callbacks = [
        ModelCheckpoint(
            filepath= os.path.join(directory, 'checkpoints'),
            verbose=1,
            save_best_only=True
        ),
        EarlyStopping(
            # Stop training when `val_loss` is no longer improving
            monitor="val_loss",
            # "no longer improving" being defined as "no better than 1e-2 less"
            min_delta=1e-2,
            # "no longer improving" being further defined as "for at least 2 epochs"
            patience=8,
            verbose=1,
        ),
        TensorBoard(
            log_dir=os.path.join(directory, 'tensorboard_log'),
            histogram_freq=1, # How often to log histogram visualizations
            embeddings_freq=0, # How often to log embedding visualizations
            update_freq="epoch",
        ) # How often to write logs (default: once per epoch)
    ]
    return callbacks

# Commented out IPython magic to ensure Python compatibility.
def train(model, train_ds, val_ds, test_ds, batch_size, epochs, add_info, show_Te
nsorBoard = True):

```

```

#directory = datetime.now().strftime("%Y-%m-%d %H:%M") + '_' + model.name
directory = model.name + '_' + add_info
callbacks = get_callbacks(directory)
history = model.fit(train_ds, epochs=epochs, batch_size=batch_size,
                    validation_data=val_ds, callbacks=callbacks)
model.save(os.path.join(directory, 'trained_model'))
print("Model saved at path ", os.path.join(directory, 'trained_model'))
if show_TensorBoard:
#         %tensorboard --logdir {os.path.join(directory, 'tensorboard_log')}

    loss, acc = model.evaluate(test_ds)
    print("Test Accuracy", acc)
    print("Test Loss", loss)
    print("_____")
    return os.path.join(directory, 'tensorboard_log')

def get_uncompile_model():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE
, IMG_SIZE, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))

    model.add(Flatten()) # this converts our 3D feature maps to 1D feature vecto
rs
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation= 'softmax'))
    return model

def get_uncompile_vgg_model():
    model = models.Sequential()
    model.add(Conv2D(16, kernel_size=(3,3), padding= 'same', activation= 'relu',
input_shape=(IMG_SIZE , IMG_SIZE, 3)))
    model.add(Conv2D(16, kernel_size=(3,3), padding= 'same',activation= 'relu'))
    model.add(MaxPooling2D(pool_size=(2,2), strides= (2,2)))

    model.add(Conv2D(32, kernel_size=(3,3), padding= 'same', activation= 'relu'))
    model.add(Conv2D(32, kernel_size=(3,3), padding= 'same', activation= 'relu'))
    model.add(MaxPooling2D(pool_size=(2,2), strides= (2,2)))

    model.add(Conv2D(64, kernel_size=(3,3), padding= 'same', activation= 'relu'))
    model.add(Conv2D(64, kernel_size=(3,3), padding= 'same', activation= 'relu'))
    model.add(Conv2D(64, kernel_size=(3,3), padding= 'same', activation= 'relu'))
    model.add(Conv2D(64, kernel_size=(3,3), padding= 'same', activation= 'relu'))

```

```

    model.add(MaxPooling2D(pool_size=(2,2), strides= (2,2)))

    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(MaxPooling2D(pool_size=(2,2), strides= (2,2)))

    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(Conv2D(128, kernel_size=(3,3), padding= 'same', activation= 'relu')
)
    model.add(MaxPooling2D(pool_size=(2,2), strides= (2,2)))

    model.add(Flatten())
    model.add(Dropout(0.6))
    model.add(Dense(128, activation= 'relu'))
    model.add(Dropout(0.6))
    model.add(Dense(64, activation= 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(32, activation= 'relu'))
    model.add(Dropout(0.4))
    model.add(Dense(num_classes, activation= 'softmax'))
    return model

"""## [Image Data Augmentation](https://www.tensorflow.org/tutorials/images/data_
augmentation)"""

AUTOTUNE = tf.data.AUTOTUNE
#NEW_IMG_SIZE = 180

def resize_and_rescale(image, label):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
    image = (image / 255.0)
    return image, label

def augment(image_label, seed):
    image, label = image_label

    image, label = resize_and_rescale(image, label)
    # Make a new seed
    new_seed = tf.random.experimental.stateless_split(seed, num=1)[0, :]

```



```

#randomly_change_image_contrast
image = tf.image.stateless_random_contrast(
    image, lower=0.3, upper=0.7, seed=new_seed)

#randomly_change_image_brightness
image = tf.image.stateless_random_brightness(
    image, max_delta=0.5, seed=new_seed)

#randomly_flip_image
image = tf.image.stateless_random_flip_left_right(
    image, seed=new_seed)

image = tf.image.stateless_random_flip_up_down(
    image, seed=new_seed)

image = tf.clip_by_value(image, 0, 1)
return image, label

resize_and_rescaleModel = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMG_SIZE, IMG_SIZE),
    layers.experimental.preprocessing.Rescaling(1./255)
])

def get_augmented_data(train_data, val_data, test_data, batch_size):
    counter = tf.data.experimental.Counter()
    train_ds = tf.data.Dataset.zip((train_data, (counter, counter)))
    train_ds = (
        train_ds
        .shuffle(1000)
        .map(augment, num_parallel_calls=AUTOTUNE)
        .batch(batch_size)
        .prefetch(AUTOTUNE)
    )
    val_ds = (
        val_data
        .map(resize_and_rescale, num_parallel_calls=AUTOTUNE)
        .batch(batch_size)
        .prefetch(AUTOTUNE)
    )
    test_ds = (
        test_data
        .map(resize_and_rescale, num_parallel_calls=AUTOTUNE)
        .batch(batch_size)
        .prefetch(AUTOTUNE)
    )
    return train_ds, val_ds, test_ds

def get_augmented_data_(train_data, val_data, test_data, batch_size):
    train_ds = train_data.map(lambda x, y: (resize_and_rescaleModel(x, training=True), y))

```

```

    val_ds = val_data.map(lambda x, y: (resize_and_rescaleModel(x, training=True),
y))
    test_ds = test_data.map(lambda x, y: (resize_and_rescaleModel(x, training=True)
, y))

    return train_ds, val_ds, test_ds

"""## Lets do it"""

epochs_count = 100
batch_size = 64

"""#### My VGG augmented"""

modelSimple = get_uncompile_vgg_model()
train_ds, val_ds, test_ds = get_augmented_data(train_data=train_dataset,
                                                val_data=val_dataset, test_data=te
st_dataset, batch_size=batch_size)

modelSimple.compile(optimizer=tf.optimizers.Adam(learning_rate=3E-4),
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                    metrics=['accuracy'])

tensorboard_path = train(modelSimple, train_ds=train_ds, val_ds=val_ds, test_ds=te
st_ds,
                        epochs=epochs_count, batch_size=batch_size, add_info='my_vg
g' )

train_ds, val_ds, test_ds = get_augmented_data(train_data=train_dataset,
                                                val_data=val_dataset, test_data=te
st_dataset, batch_size=batch_size)

#VGG-19
model3 = tf.keras.applications.VGG19(
    include_top=True, weights=None, input_tensor=None,
    input_shape=None, pooling=None, classes=num_classes ,
    classifier_activation='softmax'
)
# COMPILE
model3.compile(optimizer=tf.optimizers.Adam(learning_rate=3E-4),
               loss=tf.keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

model3_path = train(model3, train_ds=train_ds, val_ds=val_ds, test_ds=test_ds,
                    epochs=epochs_count, batch_size=batch_size, add_info='vgg'
)

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir {model3_path}

```

```

"""## Pretrained

"""

def get_pretrained_model(url, trainable=False):
    base_model = hub.KerasLayer(url, input_shape=(224, 224, 3))
    base_model.trainable = trainable

    model = keras.Sequential(
        [
            base_model,
            layers.Dropout(0.6),
            layers.Dense(64, activation="relu"),
            layers.Dropout(0.5),
            layers.Dense(66, activation="relu"),
            layers.Dropout(0.4),
            layers.Dense(num_classes, activation='softmax')
        ]
    )

    model.compile(
        optimizer=tf.optimizers.Adam(learning_rate=3E-4),
        loss=keras.losses.SparseCategoricalCrossentropy(),
        metrics=["accuracy"],
    )

    return model

# Inception_v3

url = "https://tfhub.dev/google/imagenet/inception_v3/feature_vector/5"

model = get_pretrained_model(url)

train_ds, val_ds, test_ds = get_augmented_data(train_data=train_dataset,
                                                val_data=val_dataset, test_data=test_dataset, batch_size=batch_size)

train(model, train_ds=train_ds, val_ds=val_ds, test_ds=test_ds,
      epochs=epochs_count, batch_size=batch_size, add_info='inception')

# EfficientNet

url = "https://tfhub.dev/tensorflow/efficientnet/b2/feature-vector/1"

```

```

model = get_pretrained_model(url)

train_ds, val_ds, test_ds = get_augmented_data(train_data=train_dataset,
                                                val_data=val_dataset, test_data=test_dataset, batch_size=batch_size)

train(model, train_ds=train_ds, val_ds=val_ds, test_ds=test_ds,
       epochs=epochs_count, batch_size=batch_size, add_info='efficientnet')

# Resnet_v2

url = "https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/5"

model = get_pretrained_model(url)

train_ds, val_ds, test_ds = get_augmented_data(train_data=train_dataset,
                                                val_data=val_dataset, test_data=test_dataset, batch_size=batch_size)

train(model, train_ds=train_ds, val_ds=val_ds, test_ds=test_ds,
       epochs=epochs_count, batch_size=batch_size, add_info='resnet')

# Mobilenet

url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"

model = get_pretrained_model(url)

train_ds, val_ds, test_ds = get_augmented_data(train_data=train_dataset,
                                                val_data=val_dataset, test_data=test_dataset, batch_size=batch_size)

train(model, train_ds=train_ds, val_ds=val_ds, test_ds=test_ds,
       epochs=epochs_count, batch_size=batch_size, add_info='mobilenet')

# Nasnet

url = "https://tfhub.dev/google/imagenet/nasnet_mobile/feature_vector/5"

model = get_pretrained_model(url)

train_ds, val_ds, test_ds = get_augmented_data(train_data=train_dataset,
                                                val_data=val_dataset, test_data=test_dataset, batch_size=batch_size)

train(model, train_ds=train_ds, val_ds=val_ds, test_ds=test_ds,
       epochs=epochs_count, batch_size=batch_size, add_info='nasnet')

```

```

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir sequential_2_my_vgg/tensorboard_log

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir sequential_1_inception/tensorboard_log

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir sequential_2_efficientnet/tensorboard_log

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir sequential_3_resnet/tensorboard_log

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir sequential_4_mobilenet/tensorboard_log

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir sequential_5_nasnet/tensorboard_log

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir vgg19_vgg/tensorboard_log

"""## Grid search for best model founding"""

optimisers = [
    (tf.keras.optimizers.Adam, 'adam'),
    (tf.keras.optimizers.Adamax, 'adamax'),
    (tf.keras.optimizers.Adagrad, 'adagrad'),
    (tf.keras.optimizers.Adadelta, 'adadelat'),
    (tf.keras.optimizers.SGD, 'sgd'),
    (tf.keras.optimizers.Nadam, 'nadam')
]
learning_rates = [1E-1, 1E-2, 1E-3, 1E-4, 1E-5, 1E-6]
batch_sizes = [256, 128, 64, 32]

for optimiser, optimiser_name in optimisers:
    for learning_rate in learning_rates:
        for batch in batch_sizes:
            K.clear_session()
            model = get_uncompile_model()
            train_ds, val_ds, test_ds = get_augmented_data(train_data=train_datas
et,
                                                    val_data=val_dataset,
test_data=test_dataset, batch_size=batch)

            model.compile(optimizer=optimiser(learning_rate=learning_rate),
                          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
True),
                          metrics=['accuracy']))

```

```

        add_info = optimiser_name + '_' + str(learning_rate) + '_' + str(batch)
        print(add_info)
        history = train(model, train_ds=train_ds, val_ds=val_ds, test_ds=test
_ds,
                        epochs=epochs_count, batch_size=batch, add_info=add_info)

def getFullPath(path):
    return os.path.join(os.getcwd(), path)

"""## Dataset processing and preparing functions"""

def crop_polygon(img, polygon):
    maskIm = Image.new('L', (img.shape[1], img.shape[0]), 0)
    ImageDraw.Draw(maskIm).polygon(polygon, outline=1, fill=1)
    mask = np.array(maskIm)

    # assemble new image (uint8: 0-255)
    newImArray = np.empty(img.shape, dtype='uint8')

    # colors (three first columns, RGB)
    newImArray[:, :, :3] = img[:, :, :3]

    # transparency (4th column)
    newImArray[:, :, 3] = mask*255
    return newImArray

def crop_box(img, xmin, ymin, xmax, ymax):
    crop_img = img[ymin:ymax, xmin:xmax]
    return crop_img

def load_image(path):
    img = mpimg.imread(path)

    if img.shape[2] == 3:
        newImArray = np.empty(
            (img.shape[0], img.shape[1], img.shape[2]+1),
            dtype='uint8')
        newImArray[:, :, :3] = img[:, :, :]
        newImArray[:, :, 3] = 255
        result = newImArray
    else:
        result = img

    return result

def show_image(img):
    imgplot = plt.imshow(img)
    plt.show()

```

```

def show_images_side_by_side(img1, img2):
    _, axarr = plt.subplots(1,2)
    axarr[0].imshow(img1)
    axarr[1].imshow(img2)
    plt.show()

def prepare_urls(path):
    for root, dirs, files in os.walk(path):
        for filename in files:
            pol_path = os.path.join(root, filename)
            with open(pol_path, 'r') as jsonFile:
                data = json.load(jsonFile)
            data['path'] = make_url_relative(data['path'])
            with open(pol_path, 'w') as jsonFile:
                json.dump(data, jsonFile)

def make_url_relative(url):
    return url[url.index('train'):]

def get_categories(labels_path):
    categories = {}
    for root, dirs, files in os.walk(labels_path):
        for filename in files:
            file = open(os.path.join(root, filename),)
            data = json.load(file)
            data = data['outputs']
            if data == {}:
                print(os.path.join(root, filename))
                category_set = {"NO_CATEGORY"}
            else:
                category_set = set([x['name'] for x in data['object']])
            for current in category_set:
                if current in categories:
                    categories[current] = categories[current] + 1
                else:
                    categories[current] = 1
    sorted_cat = sorted(categories.items(), key=lambda x: x[1], reverse=True)
    return sorted_cat

def get_categories_with_images(dataset_path):
    labels_path = os.path.join(dataset_path, 'labels')
    categories = {}
    for root, dirs, files in os.walk(labels_path):
        for filename in files:
            file = open(os.path.join(root, filename),)
            data = json.load(file)
            outputs = data['outputs']

```

```

        if outputs == {}:
            category_set = {"NO_CATEGORY"}
        else:
            category_set = set([x['name'] for x in outputs['object']])
        for current in category_set:
            if current in categories:
                categories[current] = categories[current] + 1
            else:
                categories[current] = 1
                print(current)
                show_image(load_image(os.path.join(dataset_path, data['path'])))
    ))

sorted_cat= sorted(categories.items(), key=lambda x: x[1], reverse=True)
return sorted_cat

def process_file_image(label_path):
    jsonFile = open(label_path,)
    pdata = json.load(jsonFile)
    image_path = os.path.join(os.getcwd(), pdata['path'])
    img = load_image(image_path)
    show_image(img)
    pdata = pdata['outputs']['object']
    for obj in pdata:
        print(obj)
        if 'polygon' in obj:
            polygon_data = obj['polygon']
            i = 1
            res = []
            while 'x'+str(i) in polygon_data:
                res.append((polygon_data['x'+str(i)], polygon_data['y'+str(i)]))
                i += 1
            cropped = crop_polygon(img, res)

            elif 'bndbox' in obj:
                box_data = obj['bndbox']
                cropped = crop_box(img, box_data['xmin'], box_data['ymin'], box_data['
xmax'], box_data['ymax'])

            show_image(cropped)

def process_file_bound(label_path):
    jsonFile = open(label_path,)
    pdata = json.load(jsonFile)
    image_path = os.path.join(os.getcwd(), pdata['path'])
    img = load_image(image_path)
    show_image(img)

    pdata = pdata['outputs']['object']
    for obj in pdata:
        print(obj)

```



```

if 'polygon' in obj:
    polygon_data = obj['polygon']
    i = 1
    polygon_coordinates = []
    while 'x'+str(i) in polygon_data:
        polygon_coordinates.append((polygon_data['x'+str(i)], polygon_data['y'+str(i)]))
        i += 1

    xmin = round(min(polygon_coordinates, key = lambda t: t[0])[0])
    xmax = round(max(polygon_coordinates, key = lambda t: t[0])[0])
    ymin = round(min(polygon_coordinates, key = lambda t: t[1])[1])
    ymax = round(max(polygon_coordinates, key = lambda t: t[1])[1])
    print("xmin", xmin)
    print("xmax", xmax)
    print("ymin", ymin)
    print("ymax", ymax)

    croppedPolygon = crop_polygon(img, polygon_coordinates)
    croppedBox = crop_box(img, xmin, ymin, xmax, ymax)

elif 'bndbox' in obj:
    box_data = obj['bndbox']
    xmin = box_data['xmin']
    xmax = box_data['xmax']
    ymin = box_data['ymin']
    ymax = box_data['ymax']

    polygon_coordinates = [(xmin, ymin), (xmin, ymax), (xmax, ymax), (xmax, ymin)]

    croppedPolygon = crop_polygon(img, polygon_coordinates)
    croppedBox = crop_box(img, xmin, ymin, xmax, ymax)

    print("Polygon")
    show_image(croppedPolygon)
    print("Box")
    show_image(croppedBox)

lbox = r'C:\Users\VsevolodSlavinskyi\Desktop\Data\Diploma\Project\Data\new_data\labels\repair\20190102--200419_SR5_94a2febe-b7d0-408c-9fad-b8c460edf2f5_398D3840-8778-49BF-8B82-45A6073944E2.json'
lpolygon = r'C:\Users\VsevolodSlavinskyi\Desktop\Data\Diploma\Project\Data\new_data\labels\replacement\20190502--191849_Orlando_Tech_f3f76d40-80d9-4324-87fc-305c89d2e9b8_7709FF88-EECE-40E7-B64B-8D256FB9D0DA.json'
process_file_new(lbox)
process_file_new(lpolygon)

labels_path = r'C:\Users\VsevolodSlavinskyi\Desktop\Data\Diploma\Project\Data\auto_cracks_dataset\auto_cracks\labels'
dataset_path = r'C:\Users\VsevolodSlavinskyi\Desktop\Data\Diploma\Project\Data\auto_cracks_dataset\auto_cracks'

```

```

cats = get_categories_with_images(dataset_path)

cats

old_data_path = r'C:\Users\VsevolodSlavinskyi\Desktop\Data\Diploma\FIG-
Recognition-Engine\my_dataset\batch_data_full_224'

with open(os.path.join(old_data_path, 'dataset_metadata.json'), 'r') as jsonFile:
    data = json.load(jsonFile)
    print(len(data), type(data))
    non_empty_data = [x for x in data if x['coordinates']['height'] != 0]
    print(len(non_empty_data), type(non_empty_data))

item = non_empty_data[0]
item

for item in non_empty_data[:10]:
    print(item['label'])
    img = load_image(os.path.join(old_data_path, 'train', item['fileName']))
    coord = item['coordinates']
    croppedBox = crop_box(img, round(coord['x']), round(coord['y']), round(coord['x'
]) + coord['width'], round(coord['y']) + coord['height'])

    show_images_side_by_side(img, croppedBox)
    print('_____')

```

Головна сторінка веб додатку

- MainPage.vue

```

<template>
  <v-container>
    <v-row class="mt-2" align="center" no-gutters justify="center">
      <v-col cols="9">
        <div class="subtitle-1 mb-1">
          <strong class="blue--text">1.</strong> Please choose photo, that
            contains some cracks on auto window
        </div>
        <v-file-input
          label="Image input"
          filled
          accept="image/*"
          dense
          prepend-icon="mdi-camera"
          @change="photoUploaded"
        >
      </v-file-input>
    </v-col>
  </v-container>

```

```

<v-col class="mt-n5" cols="9">
  <div class="subtitle-1 ">
    <strong class="blue--text">2.</strong> Please select the damaged area
    in the photo.
  </div>
</v-col>
<v-col cols="12" md="8" justify="center" align="center">
  <clipper-basic ref="clipper" class="my-
clipper" :src="photoUrl" :ratio="1" preview="my-preview">
    <div
      class="placeholder d-flex flex-column justify-space-between align-
center"
      slot="placeholder"
    >
      <v-img
        :aspect-ratio="1 / 1"
        width="400"
        src="../../assets/no_image.png"
      ></v-img>
    </div>
  </clipper-basic>
</v-col>

<v-col cols="12" md="4" align="left" justify="left">
  <div>

    <clipper-preview name="my-preview" class="my-clipper">
      <div class="placeholder d-flex flex-column justify-space-between align-
center" slot="placeholder">preview area</div>
    </clipper-preview>
  </div>
</v-col>

<v-col cols="9">
  <div class="subtitle-1 mb-2">
    <strong class="blue--text">3.</strong> Please choose model, that you
    want to use for classification
  </div>

  <v-autocomplete
    solo
    :disabled="smart"
    v-model="selectedModel"
    :items="models"
  ></v-autocomplete>

  <v-checkbox class="mt-n4" v-model="smart">
    <template v-slot:label>
      <strong>Smart decision </strong>
      <v-tooltip right>

```

```

<template v-slot:activator="{ on, attrs }">

  <v-icon
    class="ml-1"
    color="primary"

    v-bind="attrs"
    v-on="on">mdi-help-circle-outline</v-icon>

</template>
<span style="width: 100px">Smart decision will try to classify image with all models and will give that result, that most of models choose</span>
</v-tooltip>
</template>
</v-checkbox>

</v-col>

<v-col class="mt-3 mb-4" cols="12" align="center" justify="center">
  <v-btn
    color="green"
    elevation="9"
    rounded
    outlined
    :loading="loading"
    :disabled="loading"
    x-large
    @click="getCategory()"
  >
    Classify!
  </v-btn>
</v-col>

<v-col v-if="predictedCategory" align="center" justify="center" md="12">
  <div class="text-h3 mb-2">
    Result: <strong class="blue--text">{{ predictedCategory }}</strong>
  </div>
</v-col>
</v-row>
</v-container>
</template>

<script>
import axios from 'axios';

export default {
  name: "MainPage",

  data: () => ({
    photoUrl: "",
    width: 1500,

```

```

models: [
  "Inception_v3",
  "EfficientNet",
  "Resnet_v2",
  "Mobilenet",
  "Nasnet",
],
selectedModel: "Inception_v3",
predictedCategory: "",
smart: false,
loading: false,
resultImgURL: ''
}),
methods: {
  photoUploaded(file) {
    console.log(file);
    this.predictedCategory = '';
    if (file) {
      this.photoUrl = URL.createObjectURL(file);
    }
  },

  getResult() {
    const canvas = this.$refs.clipper.clip(); // call component's clip method
    this.resultImgURL = canvas.toDataURL("image/jpeg", 1); // canvas->image
  },

  getCategory(){
    let path = `http://localhost:5000/classify`;
    if(this.smart){
      path += 'Smart'
    }
    this.getResult();
    this.predictedCategory = '';

    this.loading = true;
    let data = {
      model: this.selectedModel,
      image: this.resultImgURL
    }
    axios.post(path, data)
      .then((response) => {
        this.loading = false;

        this.predictedCategory = response.data;
        console.log(response);
      }).catch((error) => {
        console.log(error);
      });
  }
};

```

```

    }

    },
    computed: {},
  };
</script>
<style scoped>
.my-clipper {
  width: 100%;
  max-width: 700px;
}
div.v-input__icon button.v-icon {
  font-size: 144px !important;
}
</style>

```

Головний файл з серверної частини

- Classify.py

```

import tensorflow as tf
import numpy as np
import os
import matplotlib.image as mpimg
from matplotlib import pyplot as plt
from PIL import Image
import base64
import io

labels = ('Repair', 'Replacement')
IMG_SIZE = 224
#resnet = tf.keras.models.Load_model('Models/prod/resnet')

def base64ImageToNumpy(base64str):
    imgdata = base64.b64decode(base64str)
    image = Image.open(io.BytesIO(imgdata))
    image_np = np.array(image)
    return image_np

def resize_and_rescale(image):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
    image = (image / 255.0)
    image = tf.reshape(image, shape=[-1, IMG_SIZE, IMG_SIZE, 3 ])
    return image

def classify_image(modelName, base64Image):

```

```

img = base64ImageToNumpy(base64Image)

modelUrl = os.path.join(os.getcwd(), 'app/static/Models', modelName)
model = tf.keras.models.load_model(modelUrl)

reshaped = resize_and_rescale(img)

prediction = model.predict(reshaped)
category = labels[np.argmax(prediction)]
return category

def classify_smart_image(modelName, base64Image):
    img = base64ImageToNumpy(base64Image)
    reshaped = resize_and_rescale(img)

    modelsUrl = os.path.join(os.getcwd(), 'app/static/Models')
    modelDirs = [ name for name in os.listdir(modelsUrl) if os.path.isdir(os.path
.join(modelsUrl, name)) ]

    preds = np.zeros((1, 2))
    print(modelDirs)
    for modelDir in modelDirs:
        model = tf.keras.models.load_model(os.path.join(modelsUrl, modelDir))
        print(modelDir)
        prediction = model.predict(reshaped)
        print(prediction[0])
        preds += prediction
    print(preds)

    category = labels[np.argmax(preds)]
    return category

```

## ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

## Слайд 1

# Моделі нейронних мереж для класифікації пошкоджень на склі автомобіля

Виконав: Славінський  
Всеволод

Науковий керівник:  
Недашківська Надія  
Іванівна

## Слайд 2

## Постановка задачі

Завдання полягає в тому щоб створити просте і гнучке рішення на основі веб і мобільного додатків, яке б дозволяло дуже просто створювати підтвердження для страхових компаній

Важливість роботи полягає в створенні моделі нейронної мережі, яка допоможе з класифікацією пошкоджень на склі автомобіля. Модель, отримана в результаті, значно полегшить роботу страховим компаніям у визначенні пошкоджень, а також зробить систему, яка не буде залежати від людського фактору, що позитивно вплине на страхових компаніях, а також на посередників

Опис системи: необхідно розробити систему, яка отримувала б фотографію на вхід, після цього користувач міг би виділити зону, на якій є пошкодження, а система відповіла, до якої з категорій це пошкодження належить.



## Слайд 3

## Актуальність роботи

Задача є актуальною, тому що страхова сфера в США є досить складною і з великою кількістю різноманітних нюансів. Також, страхові компанії досить закриті і користуються застарілим програмним забезпеченням або паперовим видом оформлення страхових випадків.

## Слайд 4

## Предмет та об'єкт досліджень

- Об'єктом дослідження є аналіз зображень з пошкодженнями на склі автомобіля, які були отримані за допомогою реальних користувачів вже працюючого додатка, який було написано для однієї з компаній-клієнтів, а також зображення отриманні в результаті алгоритмів аргументації.
- Предметом дослідження визначено згорткові нейронні мережі для класифікації зображень .

## Слайд 5



## Початкові дані

Набір даних був сформований реальними користувачами додатка, розробленого працівниками компанії WebLegends у рамках контракту з компанією, що займається випадками автострахування, та попередньо відредагований мною.

## Слайд 6

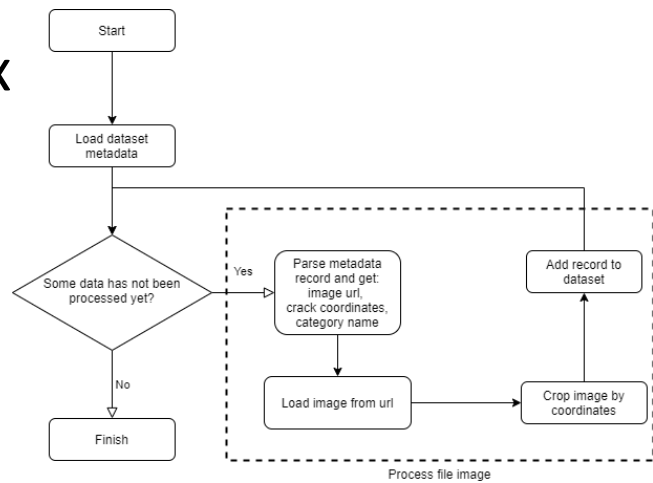
## Функції для обробки даних

- `def load_image(path)` – завантажує зображення для подальшої обробки
- `def show_image(img)` – виводить на екран задане зображення
- `def crop_polygon(img, polygon)` – вирізає прямокутник на зображенні по координатах
- `def process_file_image(label_path)` – головний цикл, в якому відбувається парсинг та обробка даних

## Слайд 7

## Схема процесінгу даних

- Вхідні дані – необроблені картинки та файл dataset\_metadata, де зберігаються відомості про кожне з зображень
- Вихідні дані – оброблений датасет, готовий до роботи



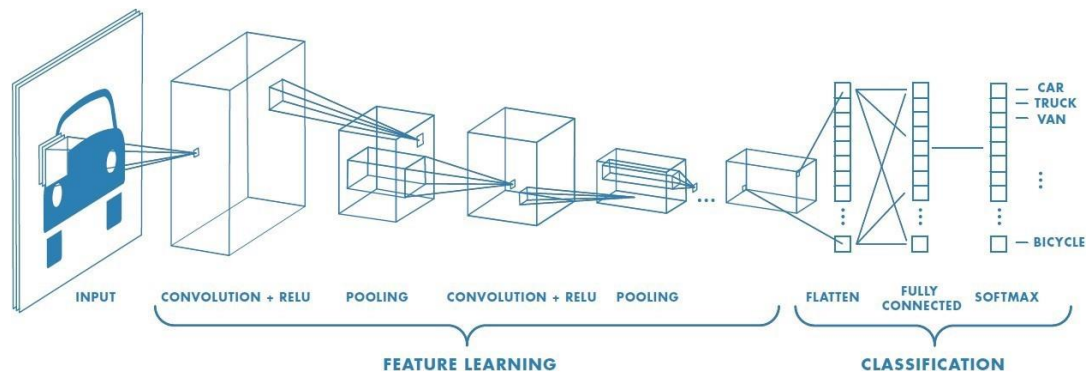
## Слайд 8

## Вихідні дані (датасет, готовий до роботи)



## Слайд 9

## Згорткові нейронні мережі



## Слайд 10

## Вимоги до системи

Необхідно, щоб до системи можна було звернутися з будь-якого пристрою.

Система не повинна вимагати завантаження будь-яких додаткових модулів або бібліотек для своєї роботи.

Різні компоненти не повинні залежати один від одного, щоб можна було розробляти їх паралельно.

Система повинна мати зрозумілий користувацький інтерфейс, оскільки основна аудиторія – це люди, які не дуже гарно працюють з комп'ютерами

Система повинна мати зрозумілий програмний інтерфейс, щоб у майбутньому її можна було легко інтегрувати в іншу систему.

## Слайд 11

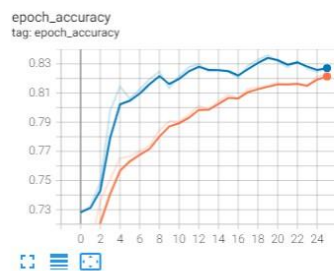
## Архітектура системи

Кожне з вимог досить важливе і ми не можемо ігнорувати їх. Отже, щоб система задовольняла усім вимогам, було вирішено зробити веб додаток, який використовує попередньо навчені моделі згорткових нейронних мереж

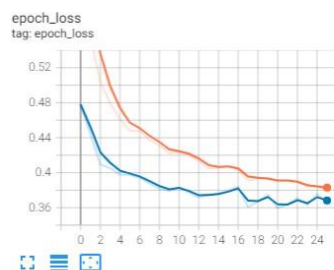
Особливості такої моделі полягають в тому, що користувач відправляє певний запит на сервер, де той системно обробляється і кінцевий результат відсилається клієнту.

## Слайд 12

epoch\_accuracy



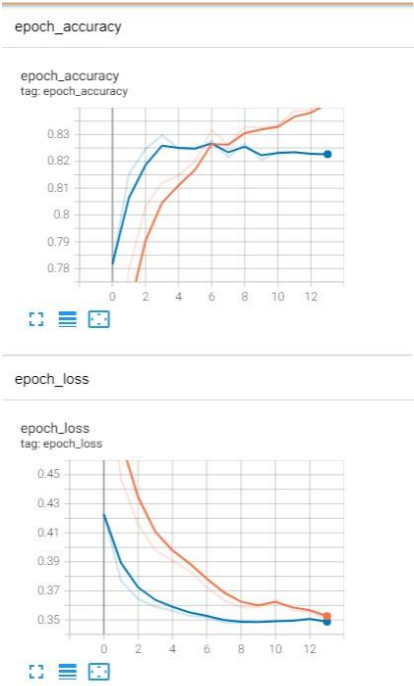
epoch\_loss



## Inception v3

	Train	Validation	Test
Accuracy	0.8225	0.8277	0.8441
Loss	0.3822	0.3662	0.3416

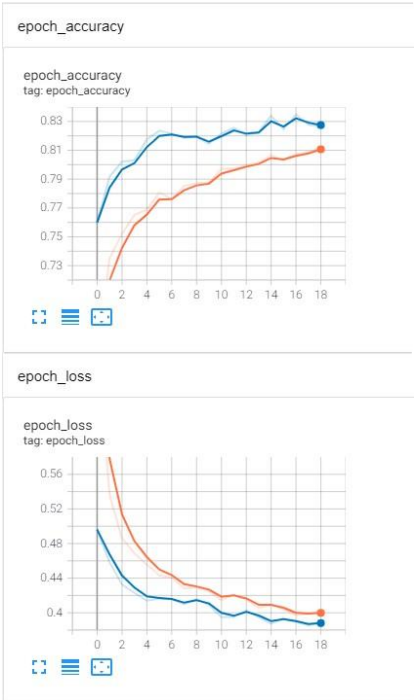
Слайд 13



# EfficientNet

	Train	Validation	Test
Accuracy	0.8434	0.8226	0.8584
Loss	0.3503	0.8434	0.3357

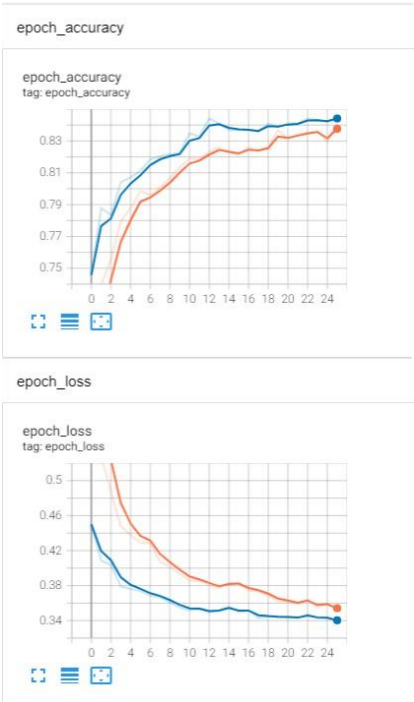
Слайд 14



# MobileNet

	Train	Validation	Test
Accuracy	0.8120	0.3886	0.8482
Loss	0.4002	0.8267	0.3491

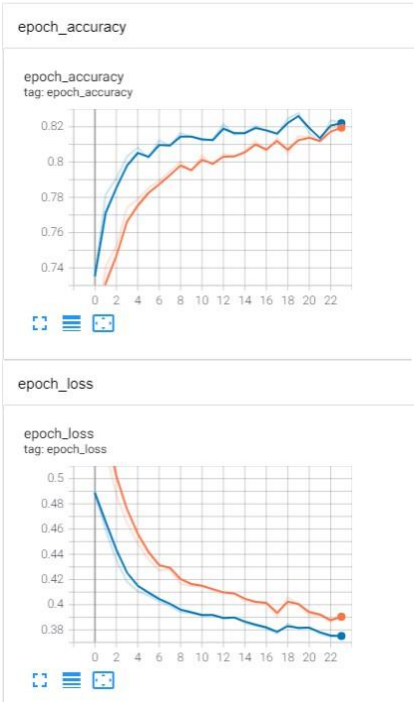
Слайд 15



# ResNet

	Train	Validation	Test
Accuracy	0.8410	0.8451	0.8471
Loss	0.3516	0.3387	0.3306

Слайд 16



# NasNet

	Train	Validation	Test
Accuracy	0.8205	0.8226	0.8461
Loss	0.3915	0.3750	0.3537

Слайд 17

## Порівняння архітектур

Архітектура	Test accuracy	Test loss
Inception	0.8441	0.3416
EfficientNet	0.8584	0.3357
ResNet	0.8471	0.3306
MobileNet	0.8482	0.3491
NasNet	0.8461	0.3537
<b>Краща</b>	<b>EfficientNet</b>	<b>ResNet</b>

Слайд 18

## Веб додаток

Auto cracks classification

1. Please choose photo, that contains some cracks on auto window

Image input

2. Please select the damaged area in the photo.

no content yet

preview area

3. Please choose model, that you want to use for classification

Inception\_v3 (pretrained)

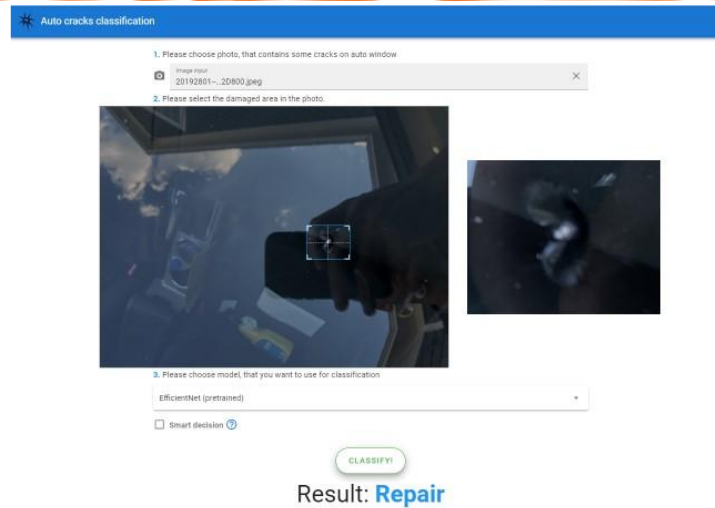
☐ Smart decision

CLASSIFY



Слайд 19

## Результати роботи додатку



Слайд 20

## Висновки

Виконано попередній аналіз та обробку вхідних даних для подальшого використання у роботі

Побудовано моделі нейронних мереж для класифікації пошкоджень на склі авто.

Виконано порівняльний аналіз найбільш популярних архітектур нейронних мереж.

Вивчено теоретичні основи згорткових нейронних мереж.

Зроблено веб додаток для зручної роботи з моделями нейронних мереж

Обрані найкращі моделі за метриками якості для вирішення практичної задачі класифікації пошкоджень скла.

## Слайд 21

## Перспективи подальших досліджень

---

Спробувати нові, більш складні архітектури мереж

---

Додати розпізнавання знаходження пошкоджень

---

Навчити мережу на більшій кількості категорій

---

Додати нові підкатегорії

---

Опублікувати веб додаток у мережі

---

Дослідити роботу автокодувальників

## Слайд 22

Дякую за увагу!

