МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНОВАЦІЙНИХ ОСВІТНІХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Жуков І.А.

"_____"_____2022 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ

"МАГІСТР"

Тема: Програмне забезпечення для складового аналізу побутових відходів

Виконавець: Столітенко Олександр Олександрович

Керівник: Антонов Володимир Констянтинович

Консультанти з окремих розділів пояснювальної записки:

Нормоконтролер: Андреєв Олександр Володимирович

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Інститут Заочного та дистанційного навчання Факультет

Кафедра Комп'ютерних систем та мереж

Напрям (спеціальність) 8.091501 "Комп'ютерні системи та мережі"
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Жуков І.А.

"_____"_____2022 р.

ЗАВДАННЯ

на виконання дипломної роботи

Столітенка Олександра Олександровича
(прізвище, ім'я, по батькові)

1. Тема дипломної роботи (проекту): Програмне забезпечення для складового аналізу побутових відходів

затверджена наказом ректора від "11" листопада 2011 р. № 2499/ст.

2. Термін виконання роботи (проекту): з 05 вересня 2022р. по 30 листопада 2022р.

3. Вихідні дані до роботи (проекту): програмне забезпечення для складового аналізу побутових відходів; дослідження методів розпізнавання об'єктів

4. Зміст пояснювальної записки: титульна сторінка, список термінів та скорочень, побудова згорткової нейронної мережі, класифікація відходів

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

Презентація *Power Point*

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Отримати завдання від керівника дипломної роботи (проекту).	05.09.22-06.09.22	виконано
2.	Виконати пошук літератури та посилань з теми “Нейронні мережі”.	07.09.22-14.09.22	виконано
3.	Зробити розділ 1 з теми “Теоретичні відомості”.	14.09.22-21.09.22	виконано
4.	Виконати пошук літератури та посилань з теми “Методологія”.	21.09.22-28.09.22	виконано
5.	Зробити розділ 2 з теми “ Методологія”.	28.09.22-05.10.22	виконано
6.	Виконати пошук літератури та посилань з теми “Результати роботи та аналіз”.	05.10.22-19.10.22	виконано
7.	Зробити розділ 3 з теми “Результати роботи та аналіз”.	19.10.22-02.11.22	виконано
8.	Оформити пояснювальну записку	02.11.22-09.11.22	виконано
9.	Підготовити графічний демонстраційний матеріал	09.11.22-30.11.22	виконано

7. Консультанти з окремих розділів

Розділ	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв

8. Дата видачі завдання: “ 05 ” вересня 2022 р.

Керівник дипломної роботи (проекту)_____. Антонов В. К.
(підпис керівника) (П.І.Б)

Завдання прийняв до виконання_____. Столітенко О. О.
(підпис випускника) (П.І.Б)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Програмне забезпечення для складового аналізу побутових відходів”: 90 с., 55 рис., 25 літературних джерел.

НЕЙРОННА МЕРЕЖА, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, ГЛИБИННЕ НАВЧАННЯ, РОЗПІЗНАВАННЯ ОБ’ЄКТІВ, *TENSORFLOW*, КЛАСИФІКАЦІЯ ВІДХОДІВ.

Об’єкт дослідження – програмне забезпечення для розпізнавання різних типів відходів.

Предмет дослідження – розробка згорткової моделі на мові програмування *Python* за допомогою бібліотеки *TensorFlow*.

Мета дипломної роботи – проаналізувати основні методи розпізнавання об’єктів та розробити програму для класифікації різних типів відходів.

Метод дослідження – принципова і програмна реалізація тренування і тестування моделі розпізнавання об’єктів.

Результати – у даній дипломній роботі розроблено згорткову модель для розпізнавання різних типів відходів.

Наукова новизна одержаних результатів – автоматизоване сортування різних типів відходів, що в свою чергу, спрощує процес сортування на спеціальних сортувальних станціях.

Програмне забезпечення для розпізнавання різних типів відходів, що було розроблене у магістерській роботі, дозволить розпізнавання об’єктів в реальному часі на конвеєрній стрічці.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

mAP - mean Average Precision

Tanh - hyperbolic tangent

ReLU - Rectified Linear Unit

CNN - Convolutional Neural Network

ELU - Exponential Linear Unit

ВСТУП

Актуальність теми

В Україні все більше підприємців запроваджують сортування сміття та привчають до цього своїх клієнтів. В магазинах все частіше можна побачити різні контейнери для сортування відходів. Серед великих торговельних мереж на цю стежку стали "Сільпо", *METRO*, "Нова пошта", *WOG* та інші. Зазвичай сортування відбувається на чотири фракції: скло, пластик, метал та папір. Потім відсортований матеріал потрапляє на станцію переробки та переробляється.

Зазвичай сортування відбувається працівниками станції вручну, але за допомогою глибинного навчання цей процес можна полегшити. Для того, щоб автоматизувати процес переробки, важливо запропонувати розумну структуру, яка зможе ефективно відслідковувати класифікацію відходів. За допомогою програмного забезпечення для виявлення об’єктів у відходах,

розпізнавання є цінною методологією, на відміну від традиційних стратегій переробки, через величезну кількість об’єктів, які розпізнаються за обмежений проміжок часу. Звичайний підхід залежить від доброї волі людини, яка схильна до невдачі при сортуванні. Техніки глибинного навчання ефективно застосовуються у різних галузях, наприклад, медична візуалізація, автономне водіння тощо. Застосування методів для сортування відходів може збільшити кількість переробленого матеріалу і таким чином спростити повсякденне життя для звичайних людей, а також підвищити ефективність індустрії.

Наукова новизна одержаних результатів

Автоматизоване сортування різних типів відходів, що в свою чергу, спрощує процес сортування на спеціальних сортувальних станціях.

Мета і завдання дослідження

Метою роботи є аналіз основних методів розпізнавання об’єктів та розробка програмного забезпечення для класифікації різних типів відходів.

Об’єкт дослідження – програмне забезпечення для розпізнавання різних типів відходів.

Предмет дослідження – розробка згорткової моделі на мові програмування *Python* за допомогою бібліотеки *TensorFlow*.

Методи дослідження – принципова і програмна реалізація тренування і тестування моделі розпізнавання об’єктів.

Практичне значення одержаних результатів

Програмне забезпечення для розпізнавання різних типів відходів, що було розроблене у магістерській роботі, дозволить розпізнавання об’єктів в реальному часі на конвеєрній стрічці.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Нейронні мережі

Основним компонентом машинного навчання (включаючи глибоке навчання) є штучний нейрон. За допомогою нейронів виконуються навчальні обчислення. У моделі штучного інтелекту величезна кількість нейронів працює разом, щоб виконувати складні числові/математичні обчислення. Формування мережі, яка формується цими нейронами, відоме як штучна нейронна мережа. Ідея штучних нейронів походить від нейронів, які біологічно присутні в сенсорній або нервовій системі людини. Подібно до нейронної мережі в тілі людини, штучна нейронна мережа поділена на шари.

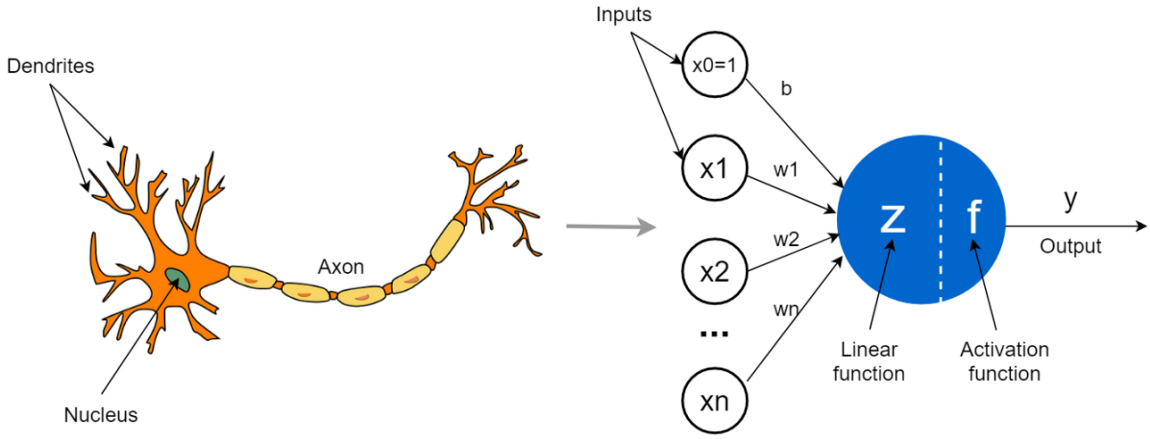

Рис. 1.1. Нейрон головного мозку людини та нейрон нейронної мережі

Дендрити є лише інформаційними терміналами нейронів у штучному нейроні. Через синапси та дендрити іншого нейрона вхідні дані обробляються аксоном, а їх вихідні дані передаються іншим нейронам. Вхідні сигнали, які

проходять по лінії входу, збільшуються на вагу лінії в обчислювальній моделі.

Зважений вхідний сигнал обробляється математичною функцією. Ця конкретна функція називається функцією активації *f*. Сигнал, який вже обробляється, знову передається нейронам наступного рівня для додаткової обробки. У цій моделі елемент навчання, як відомо, є вагою асоціації між нейронами. Значення цієї моделі змінюється протягом тривалості навчання з кінцевою метою, щоб помилка була зведена до нуля. В організмі людини сигнали, які передаються дендритами, додаються в тіло клітини, і якщо значення суми виходить за межі порогового значення, тоді в цей момент сигнали запускаються аксоном. У математичній або числовій моделі використовується аналогічна методологія. Порогове значення визначається функцією активації *f*. Сигмоїдна функція, як відомо, є стандартним рішенням функції активації. Сигмоїдна функція приймає значення підсумовування як вхідні дані та перетворює його на значення в діапазоні від 0 до 1.

1.1.1. Будова перцептрона

На рис. 1.2. показана детальна структура перцептрона. У деяких контекстах зміщення *b* позначається *w*0. Вхідні дані *x*0 завжди приймають значення 1. Отже, *b**1 = *b*.

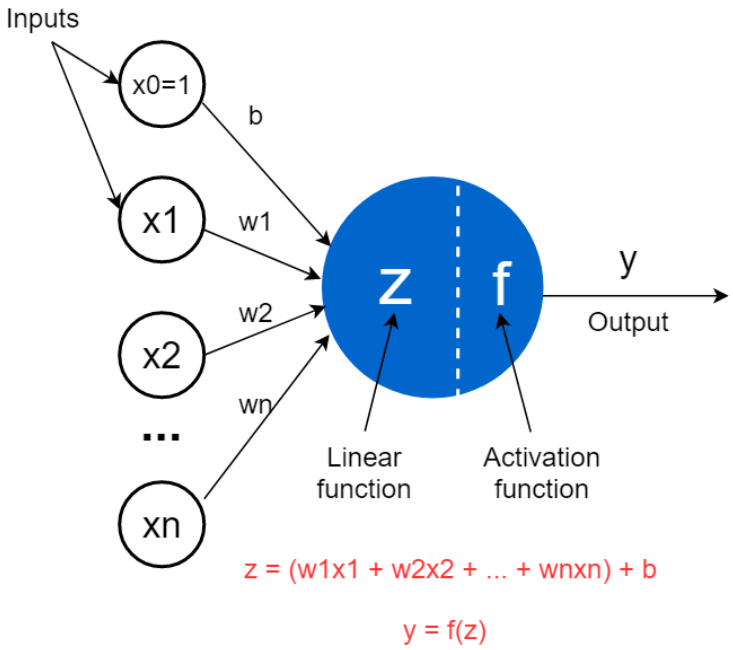

Рис. 1.2. Структура персептрона

Персептрон приймає вхідні дані, $x1$, $x2$, …, xn, множить їх на ваги, $w1$, w2, …, wn і додає член зсуву, b, потім обчислює лінійну функцію, z, до якої застосовується функція активації, f щоб отримати результат y.

Коли малюється персептрон зазвичай ігнорується одиниця зміщення для нашої зручності та спрощується діаграму наступним чином. Але в розрахунках все одно враховується одиниця зсуву.

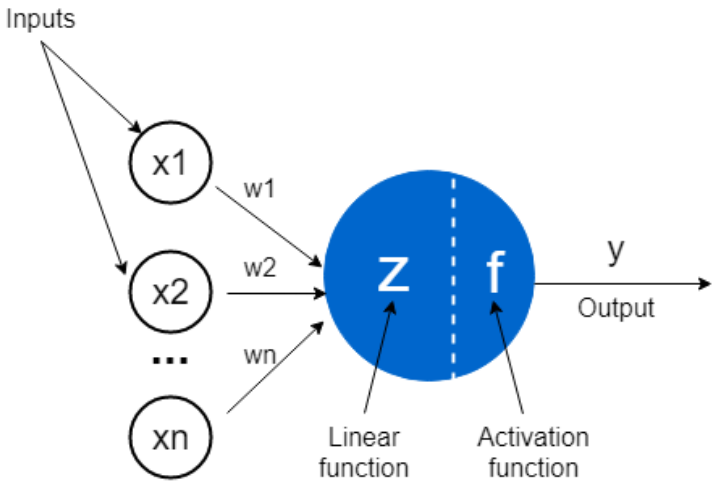

Рис. 1.3. Спрощена версія персептрона без одиниці зсуву

Зазвичай персептрон складається з двох математичних функцій.

Лінійна функція персептрона також називається лінійним компонентом перцептрона. Вона позначається літерою *z*. Результат є зваженою сумою вхідних даних плюс одиниця зміщення, і його можна обчислити наступним чином.

$$z = (w1x1 + w2x2 + \ldots + wnxn) + b$$

- *x*1, *x*2, …, *xn* – це вхідні дані, які приймають числові значення. Для одного нейрона може бути кілька (кінцевих) входів. Вони можуть бути необробленими вхідними даними або виходами інших перцептронів;
- *w*1, *w*2, …, *wn* — ваги, які приймають числові значення та контролюють рівень важливості кожного вхідного елемента. Чим вище значення, тим важливіший вхід;
- *w*1*x*1 + *w*2*x*2 + … + *wnxn* – зважена сума входів;
- *b* називається членом зсуву або одиницею зсуву, який також приймає числове значення. Він додається до зваженої суми вхідних даних. Метою включення зміщення є зміщення функції активації кожного персептрона, щоб не отримати нульове значення. Якщо всі входи *x*1, *x*2, …, *xn* дорівнюють 0, *z* дорівнює значенню зсуву.

Вагові коефіцієнти та зміщення називаються параметрами моделі нейронної мережі. Оптимальні значення цих параметрів знаходяться в процесі навчання (тренування) нейронної мережі.

Наведена вище функцію *z* може бути розглянута як лінійна регресійна модель, у якій ваги відомі як коефіцієнти, а член зміщення відомий як відрізок. Це лише термінологія, яка використовується для ідентифікації одного і того ж у різних контекстах.

Нелінійна (активаційна) функція персептрона.

Також називається нелінійним компонентом перцептрона. Позначається *f*. Цей компонент застосовують до *z*, щоб отримати результат y на основі типу функції активації, яку ми використовуємо.

$$y = f(z)$$

$$y = f((w1x1 + w2x2 + \ldots + wnxn) + b)$$

Функція *f* може мати різний тип функції активації.

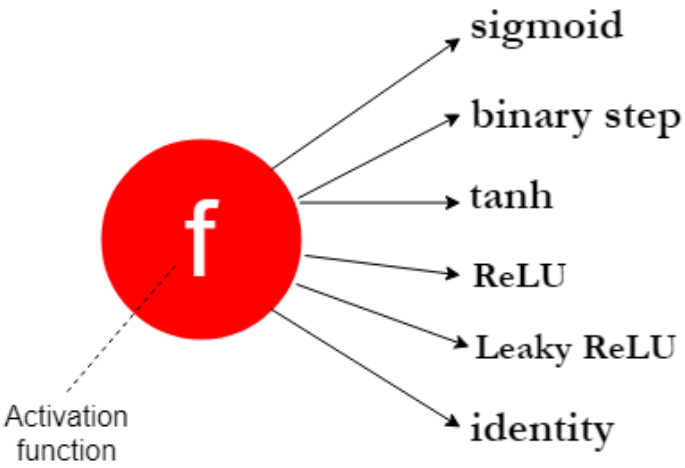

Рис. 1.4. Тип функції активації

1.1.2. Функції активації

Сигмоїдна функція.

Основна причина використання сигмоїдної функції, полягає в тому, що її результат між 0 та 1. Тому ця функція особливо використовується для моделей, де необхідно передбачити ймовірність як результат. Оскільки ймовірність будь-чого існує лише між діапазоном від 0 до 1, *sigmoid* є правильним вибором.

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

Рис. 1.5. Сигмоїдна функція активації

Binary step.

Двійкова покрокова (*binary step*) функція активації, яка також відома як функція активації порогового значення. Можна встановити будь-яке значення для порогу, наприклад значення 0.

Нейрон або перцептрон спрацьовує (або активується), лише коли значення *z* перевищує порогове значення, 0. Іншими словами, нейрон видає 1 (спрацьовує або активується), якщо значення *z* перевищує порогове значення 0. В іншому випадку він виводить 0.

Таким чином, тип функції активації визначає, як нейрон активується або спрацьовує, а член зміщення *b* контролює легкість спрацьовування.

$$\text{binary step}(z) = \begin{cases} \mathbf{1} & \text{if } z > 0 \\ \mathbf{0} & \text{if } z \leq 0 \end{cases}$$

Рис. 1.6. Двійкова покрокова функція активації

Функція активації *Tanh* або гіперболічного тангенса

Діапазон функції *tanh* — від (-1 до 1). *tanh* також сигмоподібний.

Перевагою є те, що від'ємні вхідні дані будуть відображені як сильно від'ємні, а нульові вхідні дані будуть відображені біля нуля на графіку тангенсу.

Функція *tanh* в основному використовується для класифікації між двома класами.

$$f(x_i) = \frac{exp\ (x_i)}{\sum_j exp\ (x_j)}$$

Рис. 1.7. Функція активації гіперболічного тангенса

Функція *ReLU.*

ReLU – випрямлена лінійна одиниця (*Rectified Linear Unit*).

Одна з найпопулярніших систем автофокусування в моделях *DL* — функція *Rectified Linear Unit (ReLU)* — це автофокус із швидким навчанням, який обіцяє забезпечити найсучаснішу продуктивність із неймовірними результатами. Порівняно з іншими автофокусами, такими як сигмоїда та функції *tanh*, функція *ReLU* пропонує набагато кращу продуктивність і узагальнення в глибокому навчанні. Ця функція є майже лінійною, яка зберігає властивості лінійних моделей, що полегшує їх оптимізацію за допомогою методів градієнтного спуску.

Функція *ReLU* виконує порогову операцію для кожного вхідного елемента, де всі значення, менші за нуль, встановлюються на нуль. Таким чином, *ReLU* представлено у вигляді:

$$f(x) = max\,(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$$

Рис. 1.8. Функція активації *ReLU*

Leaky ReLU.

Функція *Leaky ReLU* — це вдосконалена версія функції активації *ReLU*. Що стосується функції активації *ReLU*, градієнт дорівнює 0 для всіх значень вхідних даних, менших за нуль, що дезактивує нейрони в цій області та може спричинити проблему вмирання *ReLU*.

Для вирішення цієї проблеми визначено *Leaky ReLU*. Замість визначення функції активації *ReLU* як 0 для від'ємних значень *inputs(x),* визначається як надзвичайно мала лінійна складова *x*. Формула для цієї функції активації:

$$f(x)=\max(0.01*x , x)$$

Ця функція повертає *x*, якщо вона отримує будь-які позитивні вхідні дані, але для будь-якого від'ємного значення *x* вона повертає справді мале значення, яке дорівнює 0,01, помножене на *x*. Таким чином, він також дає вихід для від'ємних значень.

Функція *identity.*

Функція ідентичності, також звана відношенням ідентичності, картою ідентичності або перетворенням ідентичності, — це функція в математиці, яка завжди повертає те саме значення, яке використовувалося як аргумент. Можна коротко сказати, що це функція $y = x$ або функція $f(x) = x$.

1.2. Згорткові нейронні мережі

Основна функція згорткових нейронних мереж (*CNN*) полягає в тому, щоб групувати або класифікувати зображення, кластеризувати зображення за подібністю та виконувати виявлення об'єктів за допомогою штучних нейронних мереж. Згорткова нейронна мережа отримує інформацію про зображення та обробляє зображення як тензор, який є матрицею чисел з додатковими вимірами та виконує пошук. Тривимірні об'єкти є частинами екземплярів, які розпізнають зображення у вимірах. Згорткові нейронні мережі використовуються в різних програмах, включаючи розпізнавання обличчя та ідентифікація об'єктів. У складі нейронних мереж, згортковий рівень, шари субдискретизації та повністю зв'язаний шар є трьома відмінними типами шарів, які розглядаються як частини

CNN. *CNN* в основному використовується для розпізнавання зображень, оскільки цей метод має більше переваг у порівнянні з іншими стратегіями.

Існує три типи шарів у згорткових нейронних мережах:

1) Згортковий рівень (*Convolutional Layer*): у звичайній нейронній мережі кожен вхідний нейрон підключений до наступного прихованого шару. У *CNN*

лише невелика область нейронів вхідного шару підключається до прихованого шару нейронів.

2) Рівень об'єднання (*Pooling Layer*): шар об'єднання використовується для зменшення розмірності карти об'єктів. Усередині прихованого рівня *CNN* буде декілька рівнів активації та об'єднання.

3) Повністю зв'язаний рівень (*Fully-Connected layer*): повністю зв'язані рівні утворюють кілька останніх рівнів у мережі. Вхідні дані для повністю зв'язаного шару – це вихідні дані остаточного об'єднання або згорткового шару, які вирівнюються, а потім подаються в повністю зв'язаний шар.

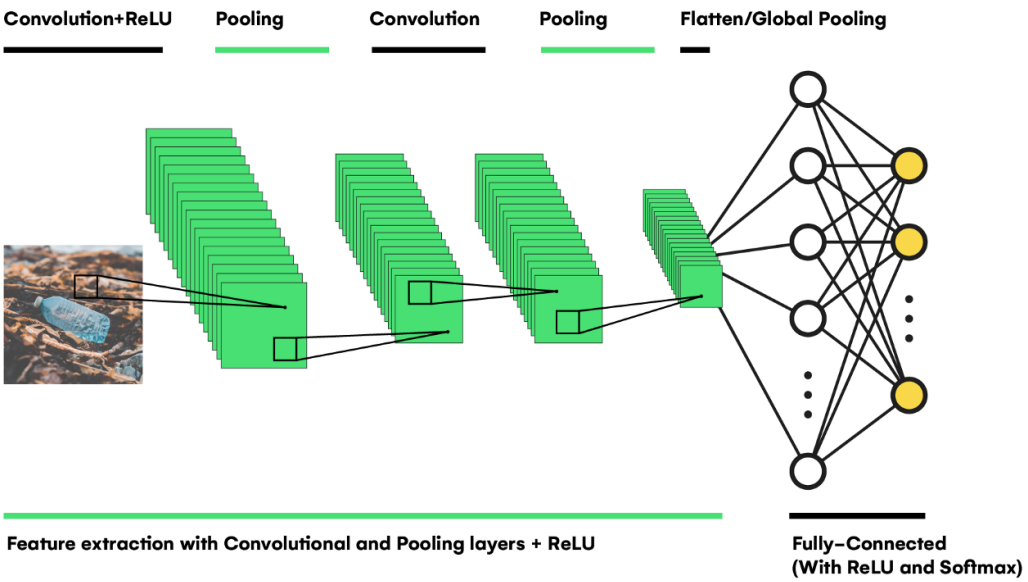

Рис. 1.9. Згорткова нейронна мережа

1.3. Робота Згорткової нейронної мережі

Спочатку вхідне зображення проходить через серію згорткових шарів, які потім виділяють особливості із зростаючою складністю за допомогою декількох подібностей (*kernels*) із різними значеннями на кожному шарі. Потім, у поєднанні зі згортковими шарами та для зменшення розмірності зображення, деякі об’єднані шари виконують згортку з певними варіаціями, такими як середній, максимальний або мінімальний елемент, замість класичної зваженої суми. Ці операції також сприяють збереженню інформації в мережі.

Після вилучення особливостей шар *Flatten* або *Global Pooling* перетворює вихідні дані останнього згорткового шару та подає повністю пов’язану (щільну) мережу, яка виконує класифікацію на основі вивчених шаблонів. Останній рівень усієї мережі містить стільки нейронів, скільки класів потрібно передбачити. За допомогою функції активації *Softmax* рівень може призначати ймовірність для кожного нейрона, який представляє, наскільки клас ознак присутній у вхідному зображенні.

1.4. Розпізнавання об’єктів

Виявлення об’єктів – це частина техніки комп’ютерного зору, де програмна основа може виявити, відслідковувати та визначити місцезнаходження об’єкта з будь-якого заданого відео чи зображення. Однією з особливостей виявлення об’єктів є те, що воно розпізнає клас об’єкта (наприклад, людина, стіл, стілець тощо) і координати розташування об’єкта на даному зображенні. Намалювавши рамку навколо об’єкта це вказує, що область вибрана. Обмежувальна рамка може знайти місце розташування об'єкта. Здатність розташувати об'єкт всередині зображення описує точність роботи алгоритму, який використовується для виявлення. Ідентифікація обличчя є одним із випадків виявлення об'єкта.

Здебільшого завдання з виявлення об’єктів виконується в три етапи:

- Маленькі прямокутні частини створюються на вхідному зображенні.
- Виділення ознак завершено для кожної частини прямокутної області для прогнозування чи містить прямокутна форма дійсний об’єкт.
- Нарешті, прямокутні рамки, які перекриваються, об’єднуються в єдине обмеження поле, що показує результат виявлення об'єкта.

1.5. Розпізнавання об’єктів за допомогою *TensorFlow API*

TensorFlow називають бібліотекою з відкритим вихідним кодом, розробленою *Google*, яка в основному використовується для числових обчислень і великомасштабних програм машинного навчання з кінцевою метою обробки отриманих даних разом із навчанням моделі, наданням прогнозів і уточненням майбутніх результатів.

Алгоритми моделей машинного навчання та моделей глибокого навчання об’єднуються разом у *TensorFlow*. Як фронт-енд він використовує *Python*, а у бек-енд продуктивно працює з *C++*.

TensorFlow дозволяє розробникам створювати діаграми/графіки обчислень. Кожен вузол на графіку визначає математичну дію, а кожне з’єднання представляє інформацію про дані. Розробники можуть повністю зосередитися на логіці програми, замість того, щоб керувати такими тонкощами, як вибір відповідних підходів для зв’язування виводу однієї функції з введенням інших.

На даний момент *TensorFlow* є найвідомішою програмною бібліотекою. Існує багато додатків із реального світу глибокого навчання, що роблять *TensorFlow* дуже відомим. Оскільки це бібліотека з відкритим кодом для машинного навчання (включаючи глибоке навчання), саме тому вона може відігравати дуже важливу роль у сфері текстових програм, голосового пошуку, виявлення зображень тощо. *TensorFlow* також використовується в системі

розпізнавання зображень *Facebook*, відомій як *DeepFace*. Також використовується *Siri* від *Apple* для розпізнавання голосу. Кожна програма *Google*, якою користуєтесь, використовує *TensorFlow* для покращення роботи.

TensorFlow Object Detection API — це структура з відкритим кодом, заснована на *TensorFlow*, яка може легко створювати, навчати та розгортати моделі виявлення об'єктів. На даний момент в їх структурі є багато попередньо підготовлених і навчених моделей, які називаються *Model Zoo. Model Zoo* складається з кількох попередньо підготовлених моделей, які навчаються на різних наборах даних, широко відомих як набір даних відкритих зображень (*Open Images Dataset*), набір даних *KITTI* та набір даних загальних об'єктів у контексті (*COCO*).

Існують різні моделі (наприклад, *ssd_mobilenet, ssd_resnet, ssd_inception, faster R-CNN inception, Faster R-CNN resnet*, тощо), які доступні в *Model Zoo*. Але головна відмінність серед цих різноманітних моделей є те, що вони мають відмінну архітектуру і дають різні показники точності разом із різною швидкістю виконання та точністю в налаштуванні обмежувальних рамок.

Тут середня точність (*mAP*) є результатом добутку точності та пам'яті розпізнавання обмежувальних прямокутників. Це гідна спільна міра, яка показує, як чутливість мережі розвиває об'єкти інтересу та наскільки правильно вона уникає помилкових тривог. Точність моделі зростає, оскільки оцінка *mAP* стає вищою, однак це відбувається за рахунок швидкості реалізації, якої потрібно триматися подалі. Механізм *API* виявлення об'єктів у *TensorFlow* показаний на рис 1.10.

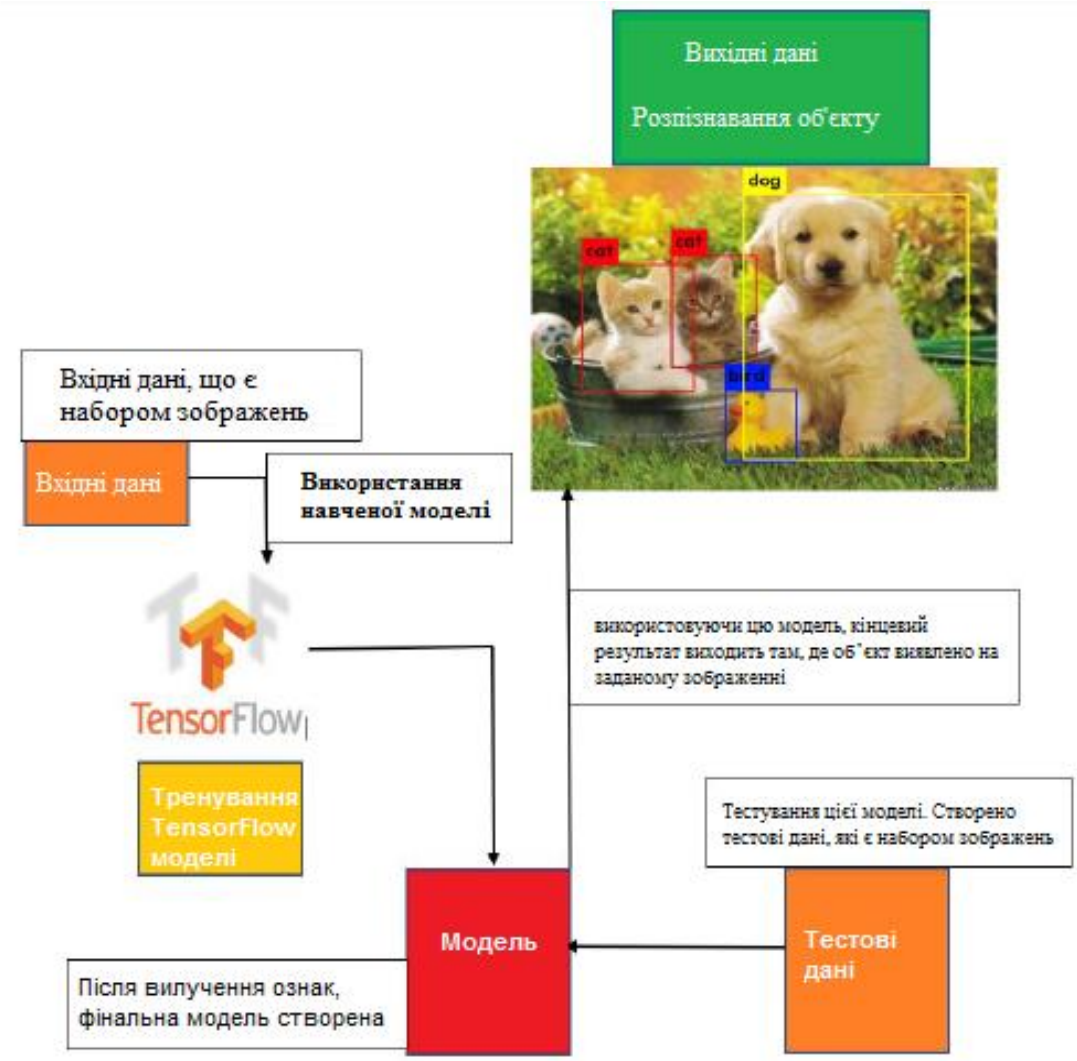

Рис. 1.10. Виявлення об’єктів у *TensorFlow*

1.6. Висновки до розділу

В першому розділі було розглянуто структуру нейронної мережі, будову персептрона, функції активації. Розглянуто структуру та принцип роботи *CNN*. Методи розпізнавання об’єктів за допомогою *TensorFlow API.*

РОЗДІЛ 2
МЕТОДОЛОГІЯ

2.1. Використані бібліотеки

2.1.1. *NumPy.*

Масив багатовимірної матриці, який підтримує математичні обчислення високого рівня. Операції над масивами, включеними в математику, такі як алгебраїчні, статистичні та тригонометричні моделі, можна виконувати за допомогою *NumPy*. Зображення перетворюється в матричну форму.

Матрична форма зображення використовується для інтерпретації та аналізу за допомогою Згорткової нейронної мережі. На етапах попередньої обробки, зображення збільшується до 224×224 пікселів. Після цього анотація зображення змінюється під стиль масиву *NumPy*. Нарешті точні позначки зображень включено в набір даних. На цьому також створений *SciPy*. Він пропонує більш примітне виконання, яке працює з масивами *NumPy* і потрібне для різних інженерних і логічних програм.

2.1.1.1. Створення масивів.

У *NumPy* є багато способів створити масив. Один із найпростіших — створити масив зі звичайних списків або кортежів *Python*, використовуючи функцію numpy.*array*().

Функція *array*() трансформує вкладені послідовності багатомірні масиви. Тип елементів масиву залежить від типу елементів вихідної послідовності (але можна і перевизначити його на момент створення).

```
>>> b = np.array([[1.5, 2, 3], [4, 5, 6]])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

Рис. 2.1. Використання функції *array*

Можна також перевизначити тип у момент створення:

```
>>> b = np.array([[1.5, 2, 3], [4, 5, 6]], dtype=np.complex)
>>> b
array([[ 1.5+0.j,  2.0+0.j,  3.0+0.j],
       [ 4.0+0.j,  5.0+0.j,  6.0+0.j]])
```

Рис. 2.2. Перевизначення типу у момент створення масиву

Функція *array*() не єдина функція створення масивів. Зазвичай елементи масиву спочатку невідомі, а масив, у якому зберігатимуться, вже потрібний. Тому є кілька функцій для того, щоб створювати масиви з якимось вихідним вмістом (за замовчуванням тип масива, що створюється — *float*64).

Функція *zeros*() створює масив із нулів, а функція *ones*() — масив із одиниць. Обидві функції приймають кортеж з розмірами та аргумент *dtype*.

```
>>> np.zeros((3, 5))
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> np.ones((2, 2, 2))
array([[[ 1.,  1.],
        [ 1.,  1.]],

       [[ 1.,  1.],
        [ 1.,  1.]]])
```

Рис. 2.3.Функції *zeros* та *ones*

Функція *eye*() створює одиничну матрицю (двовимірний масив).

```
>>> np.eye(5)
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

Рис. 2.4. Функція *eye*

Функція *empty*() створює масив без заповнення. Вихідний вміст випадковий і залежить від стану пам'яті на момент створення масиву (тобто від сміття, що в ній зберігається):

```
>>> np.empty((3, 3))
array([[  6.93920488e-310,   6.93920488e-310,   6.93920149e-310],
       [  6.93920058e-310,   6.93920058e-310,   6.93920058e-310],
       [  6.93920359e-310,   0.00000000e+000,   6.93920501e-310]])
>>> np.empty((3, 3))
array([[  6.93920488e-310,   6.93920488e-310,   6.93920147e-310],
       [  6.93920149e-310,   6.93920146e-310,   6.93920359e-310],
       [  6.93920359e-310,   0.00000000e+000,   3.95252517e-322]])
```

Рис. 2.5. Функція *empty*

Для створення послідовностей чисел, *NumPy* є функція *arange(),* аналогічна вбудованій в *Python range*(), тільки замість списки вона повертає масиви, і приймає не тільки цілі значення:

```
>>> np.arange(10, 30, 5)
array([10, 15, 20, 25])
>>> np.arange(0, 1, 0.1)
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
```

Рис. 2.6. Функція *arange*

2.1.1.2. Об'єднання та розподіл масивів.

Функції *hstack* та *vstack.*

Створюю два двовимірні масиви (рис. 2.7.).

a = np.array([(1, 2), (3, 4)])
b = np.array([(5, 6), (7, 8)])

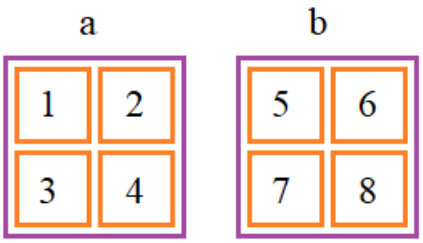

Рис. 2.7. Два двовимірні масиви

Їх можна поєднати як по горизонталі, так і по вертикалі, за допомогою функцій:

```
np.hstack([a, b])  # об’єднання по осі axis1 (розмірність 2x4)
np.vstack([a, b])  # об’єднання по осі axis0 (розмірність 4x2)
```

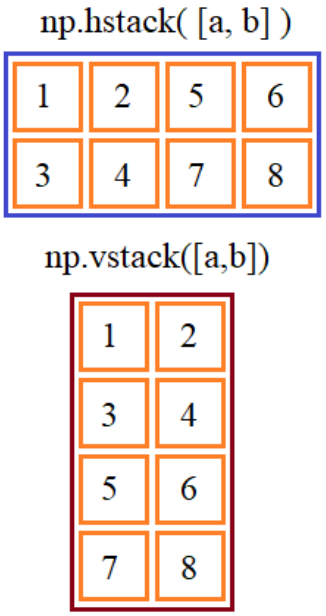

Рис. 2.8. Об’єднання масивів по горизонталі і по вертикалі

Масиви в *NumPy* можна як об'єднувати, а й розділяти. Для цього існують спеціальні функції *hsplit* та *vsplit*. Розглянемо їхню роботу на простих прикладах.

Нехай є одновимірний масив із 10 елементів:

```
a = np.arange(10)
```

І потрібно поділити його на дві рівні частини. Це реалізується за допомогою функції *hsplit*:

```
np.hsplit(a, 2)
```

Ця функція повертає список із двох масивів. Другий параметр 2 вказує кількість частин, куди ділиться вихідний масив. Причому поділ виконується по горизонталі. Якщо в прикладі вказати 3 частини, то виникне помилка.

Ці функції можна використовувати і з багатовимірними масивами.

Функція *array_split*.

Якщо потрібно виконати розбиття по довільній осі багатовимірного масиву використовується функція *array_split*.

Її робота аналогічна розглянутим вище функціям, лише додатково вказується вісь розбиття. Наприклад:

```
a = np.arange(18)
a.resize(3, 3, 2)

np.array_split(a, 2, axis=2)
np.array_split(a, 3, axis=0)
np.array_split(a, 3, axis=1)
```

2.1.2. Matplotlib.

Функції побудови графіків для мов програмування *Python* підтримуються *Matplotlib*. *Matplotlib* пропонує об’єктно-орієнтований інтерфейс програмування додатків. *Numpy* є одним із математичних числових розширень *Matplotlib*.

Matplotlib складається з багатьох модулів. Модулі наповнені різними класами та функціями, які ієрархічно пов'язані між собою.

Так як *Matplotlib* організована ієрархічно, а найпростішими для розуміння людиною є високорівневі функції, то знайомство з *Matplotlib* починають з самого високорівневого інтерфейсу *matplotlib.pyplot*. Так, щоб намалювати гістограму за допомогою цього модуля, потрібно викликати лише одну команду: *matplotlib.pyplot.hist(arr)*.

Користувачеві не потрібно думати, як саме бібліотека намалювала цю діаграму. Якби ми малювали гістрограму самостійно, то помітили б, що вона складається з фігур, що повертаються за формою - прямокутників. А щоб намалювати прямокутник, потрібно знати хоча б координату одного кута та ширину/довжину. Малювався б прямокутник лініями, з'єднуючи кутові точки прямокутника.

Цей приклад відображає ієрархічність малюнків, коли підсумкова діаграма (високий рівень) складається з простих геометричних фігур (нижчий, середній рівень), створених декількома універсальними методами малювання (низький рівень). Якби кожен малюнок треба було б створювати ось так, з нуля, це було б дуже довго та втомливо.

Інтерфейс *matplotlib.pyplot* є набором команд та функцій, які роблять синтаксис графічних *Matplotlib* команд схожим на команди, що використовуються у середовищі *Matlab*. Спочатку *Matplotlib* планувався як вільна альтернатива *Matlab*, де в одному середовищі були засоби як для малювання, так і для чисельного аналізу. Саме так у *Matplotlib* з'явився *pylab*, який поєднує модулі *pyplot* і *numpy* в один простір імен.

2.1.2.1. Ієрархічна структура малюнка.

Головною одиницею (об'єктом найвищого рівня) під час роботи з *Matplotlib* є малюнок (*Figure*). Будь-який малюнок у *Matplotlib* має вкладену

структуру. Користувацька робота передбачає операції з різними рівнями цієї структури (рис. 2.9.).

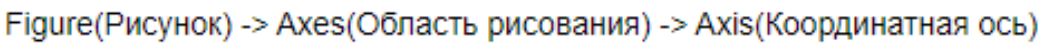

Рис. 2.9. Рівні в *Matplotlib*

- Малюнок (Figure) є об'єктом найвищого рівня, на якому розташовуються одна або кілька областей малювання (Axes), елементи малюнка Artisits (заголовки, легенда тощо) та основа-полотно (Canvas). На малюнку може бути кілька областей малювання Axes, але ця область малювання Axes може належати лише одному малюнку Figure;
- Область малювання (Axes) є об'єктом середнього рівня, який, мабуть, є головним об'єктом роботи з графікою matplotlib в об'єктно-орієнтованому стилі. Це те, що асоціюється зі словом "plot", це частина зображення з простором даних. Кожна область малювання Axes містить дві (або три у разі тривимірних даних) координатних осі (Axis об'єктів), які впорядковують відображення даних;
- Координатна вісь (Axis) є об'єктом середнього рівня, який визначає область зміни даних, на них наносяться поділи ticks та підписи до поділів ticklabels. Розташування поділів визначається об'єктом Locator, а підписи поділів обробляє об'єкт Formatter. Конфігурація координатних осей полягає у комбінуванні різних властивостей об'єктів Locator та Formatter;
- Елементи малюнка (Artists) є хіба що червоною лінією для всіх ієрархічних рівнів. Практично все, що відображається на малюнку, є елементом малюнка (Artist), навіть об'єкти Figure, Axes і Axis. Елементи малюнка Artists включають такі прості об'єкти як текст (Text), плоска лінія (Line2D), фігура (Patch) та інші.

Коли відображається малюнок (figure rendering), всі елементи малюнка Artists наносяться на основу-полотно (Canvas). Більшість їх пов'язується з областю малювання Axes. Також елемент малюнка не може спільно використовуватися декількома областями Axes або переміщуватися з однієї на іншу.

2.1.2.2. Pyplot.

Інтерфейс *pyplot* дозволяє користувачеві зосередитись на виборі готових рішень та налаштуванні базових параметрів малюнка.

Стандартний виклик *pyplot* в *Python*:

```
import matplotlib.pyplot as plt
```

Малюнки *matplotlib* створюються шляхом послідовного виклику команд: або в інтерактивному режимі (в консолі), або в скрипті (текстовий файл з *python*-кодом). Графічні елементи (точки, лінії, фігури тощо) нашаровуються одна на одну послідовно. У цьому наступні перекривають попередні, якщо вони займають загальні ділянки малюнку (регулюється параметром *zorder*).

У *matplotlib* працює правило "поточної області" ("*current axes*"), яке означає, що всі графічні елементи наносяться на поточну область малювання. Незважаючи на те, що областей малювання може бути декілька, одна з них завжди є поточною.

Найголовнішим об'єктом в *matplotlib* є малюнок *Figure*. Тому створення наукової графіки слід розпочинати саме зі створення малюнка. Створити малюнок у *matplotlib* означає задати форму, розміри та властивості основи-полотна (*canvas*), на якому буде створюватися майбутній графік.

Створити рисунок *figure* дозволяє метод *plt.figure*(). Після виклику будь-якої графічної команди, тобто функції, яка створює будь-який графічний об'єкт,

наприклад *plt.scatter*() або *plt.plot*(), завжди існує хоча б одна область для малювання (за замовчуванням прямокутної форми).

Щоб результат малювання, тобто поточний стан малюнка, відобразився на екрані, можна скористатися командою *plt.show*(). Буде показано всі малюнки (*figures*), які були створені.

```
import matplotlib.pyplot as plt

fig = plt.figure()   # створення объекта Figure
print (fig.axes)   # список поточних областей малювания пустий
print (type(fig))   # тип об’єкта Figure
plt.scatter(1.0, 1.0)   # scatter - метод для нанесення маркеру в точці (1.0, 1.0)

# після нанесения графічного элемента в виді маркеру
# список поточних областей складається із однієї області
print (fig.axes)

# дивитись преамбулу
save(name='pic_1_4_1', fmt='pdf')
save(name='pic_1_4_1', fmt='png')

plt.show()
```

```
[]
<class 'matplotlib.figure.Figure'>
[<matplotlib.axes._subplots.AxesSubplot object at 0x0000000007C2C2B0>]
```

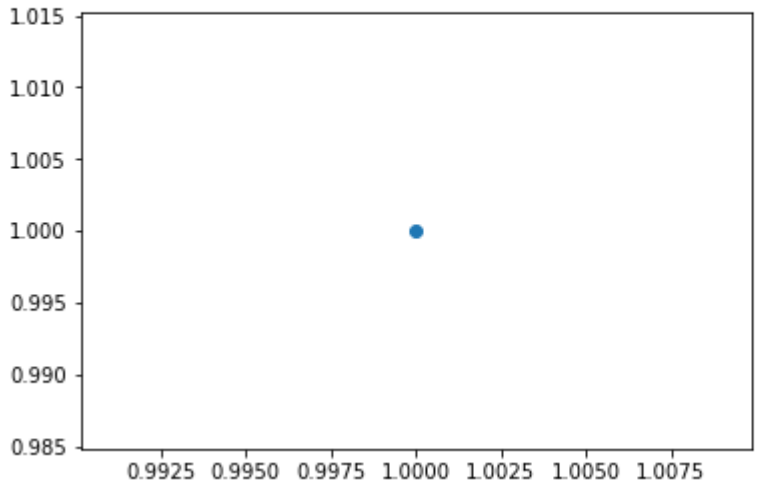

Рис. 2.10. Створення графічного об'єкта

2.1.3. *OS.*

OS є однією з бібліотек імпорту *Python*. Дозволяє системі взаємодіяти з основною операційною системою, на якій працює *Python*. За допомогою *os.path* вказує шлях для доступу та використовує різні методи: basename, dirname, exists, isdir, isfile, join, split.

Функція *basename* поверне назву файлу шляху.

```
import os

os.path.basename(r'C:\Python27\Tools\pynche\ChipViewer.py')
# ChipViewer.py
```

Це дуже корисна функція, особливо в випадках, коли потрібно використовувати ім'я файлу для найменування того чи іншого пов'язаного з роботою файлу, наприклад лог-файл. Така ситуація виникає часто під час роботи з файлами даних.

Функція dirname повертає лише частину каталогу шляху.

```
import os

print( os.path.dirname(r'C:\Python27\Tools\pynche\ChipViewer.py') )
# C:\\Python27\\Tools\\pynche
```

У цьому прикладі просто повертається шлях до каталогу. Це також корисно, коли потрібно зберегти інші файли поруч з тим, що ви обробляється зараз. Як і у випадку з лог-файлом, згаданим вище.

Функція exists показує, чи існує файл, чи ні. Потрібно лише вказати шлях.

```
import os

os.path.exists(r'C:\Python27\Tools\pynche\ChipViewer.py') # True
os.path.exists(r'C:\Python27\Tools\pynche\fake.py') # False
```

У першому прикладі, вказано функції exists справжній шлях, що вона вказує як *True*. Це говорить про те, що цей шлях існує. У другому прикладі вказано неправильний шлях, від чого функція вказує на це повідомленням *False*.

Методи *isdir* і *isfile* тісно пов'язані з методом *exists*, оскільки вони також тестують присутність або відсутність файлів чи папок на тих чи інших шляхах. Однак, *isdir* перевіряє тільки шляхи до папок, а *isfile*, відповідно, до файлів. Якщо вам потрібно перевірити шлях, і не важливо, чи це папка або файл, простіше буде скористатися методом *exists*.

```
import os

os.path.isfile(r'C:\Python27\Tools\pynche\ChipViewer.py') # True
os.path.isdir(r'C:\Python27\Tools\pynche\ChipViewer.py') # False
os.path.isdir(r'C:\Python27\Tools\pynche') # True
os.path.isfile(r'C:\Python27\Tools\pynche') # False
```

У першому прикладі вказано шлях до файлу щоб перевірити, чи є цей шлях насправді файлом. Потім, у другому прикладі, було пророблено те саме, але в контексті папки. Після цих двох прикладів трохи змінено умови, вказавши шлях до папки для обох функцій. Ці приклади наочно демонструють, як ці функції працюють.

Метод *join* дозволяє поєднати кілька шляхів за допомогою наданого роздільника. Наприклад, в *Windows*, у ролі роздільника виступає бекслеш (коса риска, що вказує назад), але в *Linux* функція роздільника присвоєна косій рисці, що вказує вперед (*forward slash*).

```
import os

print( os.path.join(r'C:\Python27\Tools\pynche', 'ChipViewer.py') )
# C:\\Python27\\Tools\\pynche\\ChipViewer.py
```

У цьому прикладі поєднано шлях каталогу та файлу разом, для отримання робочого шляху.

Метод *split* роз'єднує шлях на кортеж, який містить файл і каталог.

```
import os

print( os.path.split(r'C:\Python27\Tools\pynche\ChipViewer.py') )
# ('C:\\Python27\\Tools\\pynche', 'ChipViewer.py')
```

У цьому прикладі показано, що відбувається, коли вказується шлях до файлу. Тепер можна поглянути на те, що відбувається, якщо наприкінці шляху немає назви файлу.

```
import os

print( os.path.split(r'C:\Python27\Tools\pynche') )
# (‘C:\Python27\Tools’, ‘pynche’)
```

Як можна побачити, ця функція бере шлях і роз'єднує його таким чином, що підпапка стала другим елементом кортежу з рештою шляху в першому елементі. Насамкінець, звичайний випадок використання *split*.

```
import os

dirname, fname = os.path.split(r'C:\Python27\Tools\pynche\ChipViewer.py')
print(dirname)
# C:\\Python27\\Tools\\pynche
 print(fname)
# ChipViewer.py
```

У цьому прикладі вказано, як зробити множинне призначення. Коли ви роз'єднуєте шлях, він стає кортежем, що складається із двох частин. Після того, як випробувано дві змінні з лівої частини, перший елемент кортежу призначений до першої змінної, а другий елемент до другої змінної відповідно.

2.1.4. *TensorFlow*.

TensorFlow бібліотека розроблена *Google*, використовується для швидких математичних обчислень. Може безпосередньо створювати моделі глибокого навчання. Це один із видів математичної бібліотеки та програми штучного інтелекту та машинного навчання, як нейронні мережі. Існують різні типи моделей глибокого навчання, доступні та представлені за допомогою команди *TensorFlow pi*. Перед початком навчання моделі, *TensorFlow* допомагає в

доповненні даних. Це додатково використовується для підвищення ефективності алгоритму, а після цього завантажує попередньо навчені ваги мережі зображень.

TensorFlow забезпечує безліч різних наборів інструментів, які дозволяють писати код на бажаному рівні абстракції. Наприклад, можно написати код в *Core TensorFlow* (*C++*) і викликати цей метод з коду *Python.* Можна визначити архітектуру, на якій повинен працювати код (*CPU, GPU* тощо). Найнижчий рівень, на якому можна написати код, це *C++* або *Python*. Ці два рівні дозволяють писати числові програми для вирішення математичних операцій і рівнянь. Хоча це не рекомендується для побудови моделей машинного навчання, він пропонує широкий спектр бібліотек математики, які полегшують ваші завдання. Наступний рівень, на якому можна написати код, використовує специфічні абстрактні методи *TensorFlow*, які високо оптимізовані для компонентів моделі. Наприклад, за допомогою абстрактного методу *tf.layers* можна оперувати з шарами нейронної мережі. Можна побудувати модель і оцінити продуктивність моделі за допомогою методу *tf.metrics*. Найбільш широко використовуваним є *tf.estimator API*, який дозволяє легко створювати (тренувати і прогнозувати) готові моделі. *API estimator* є надзвичайно простим у використанні і добре оптимізований. Незважаючи на те, що вона забезпечує меншу гнучкість, вона має все необхідне для навчання та тестування вашої моделі.

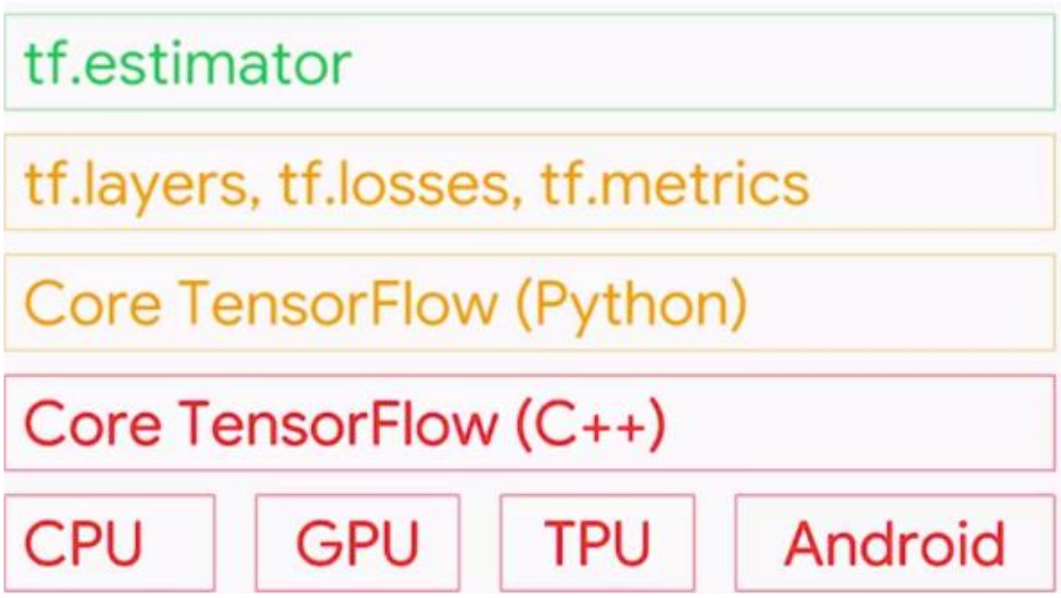

Рис. 2.11. Рівні абстракції TensorFlow

Основним типом даних у цій структурі є тензор. Тензор - це N-вимірний масив даних. Наприклад, можно викликати скаляр як тензор 0-розмірності. Вектор - це 1-мірний тензор, а матриця — двовимірний тензор.

2.1.4.1. Робота з *Graph* в *TensorFlow*.

TensorFlow використовує граф потоку даних для представлення обчислень у термінах залежностей між окремими операціями. Це призводить до низькорівневої моделі програмування, в якій спочатку визначається графік потоку даних, а потім створюється сеанс *TensorFlow* для запуску частин графіка через набір локальних і віддалених пристроїв.

Високорівневі *API*, такі як *tf.estimator.Estimator* і *Keras*, приховують деталі графів і сесій від кінцевого користувача.

Dataflow — це загальна модель програмування для паралельних обчислень. На графу потоку даних вузли являють собою одиниці обчислення, а ребра представляють дані, що споживаються або виробляються обчисленням. Наприклад, в графі *TensorFlow* операція *tf.matmul* відповідає одному вузлу з

двома вхідними ребрами (матрицями, які потрібно множити) і одним вихідним краєм (результатом множення).

Dataflow має ряд переваг, які *TensorFlow* використовує при виконанні програм:

- Паралелізм — використовуючи явні ребра для представлення залежностей між операціями, системі легко визначити операції, які можуть виконуватися паралельно.

- Розподілене виконання — використовуючи явні ребра для представлення значень, що протікають між операціями, *TensorFlow* може розділити вашу програму на декілька пристроїв, приєднаних до різних машин. *TensorFlow* вставляє необхідні комунікації і координацію між пристроями.

- Компіляція — компілятор XLA TensorFlow може використовувати інформацію у вашому графіку потоку даних для генерації більш швидкого коду, наприклад, шляхом злиття суміжних операцій.

- Переносимість — графік потоку даних не залежить від мови представлення коду вашої моделі. Ви можете побудувати графік потоку даних в Python, зберегти його в SavedModel і відновити в програмі C ++ для низьких затримок.

Вузли і ребра графа, що вказують, як окремі операції складаються разом, але не прописують, як вони повинні бути використані. Структура графіка схожа на код збірки: перевірка може передати деяку корисну інформацію, але вона не містить всього корисного контексту, який передає вихідний код.

TensorFlow забезпечує загальний механізм для зберігання колекцій метаданих в *tf.Graph*. Функція *tf.add_to_collection* дозволяє асоціювати список об'єктів з ключем (де *tf.GraphKeys* визначає деякі стандартні ключі), а *tf.get_collection* дозволяє шукати всі об'єкти, пов'язані з ключем. Багато частин бібліотеки *TensorFlow* використовують цю функцію: наприклад, коли створюється *tf.Variable*, вона додається за замовчуванням до колекцій, що

представляють "глобальні змінні" і "тренувальні змінні". Пізніше при створенні *tf.train.Saver* або *tf.train.Optimizer*, змінні в цих колекціях використовуються як аргументи за замовчуванням.

Більшість програм *TensorFlow* починаються з фази побудови графіка потоку даних. На цій фазі ви викликаєте функції *TensorFlow API*, які будують нові об'єкти *tf.Operation (node)* і *tf.Tensor (edge)* і додають їх до *tf.Graph. TensorFlow* надає граф за замовчуванням, який є неявним аргументом для всіх функцій *API* в одному контексті.

Виклик *tf.constant*(42.0) створює єдину *tf.Operation*, яка виробляє значення 42.0, додає її до графіка за замовчуванням, і повертає *tf.Tensor*, що представляє значення константи. Виклик *tf.matmul (x, y)* створює єдину операцію *tf.Operation*, яка помножує значення об'єктів *tf.Tensor x* і *y*, додає її до графіка за замовчуванням і повертає *tf.Tensor*, який представляє результат множення. Виконання *v = tf.Variable* (0) додає до графа *tf.Operation*, яке зберігатиме значення тензора, яке зберігається між викликами *tf.Session.run*. Об'єкт *tf.Variable* обертає цю операцію і може використовуватися як тензор, який зчитує поточне значення збереженого значення.

Об'єкт *tf.Variable* також має такі методи, як *tf.Variable.assign* і *tf.Variable.assign_add,* які створюють об'єкти *tf.Operation*, які при виконанні оновлюють збережені значення.

Виклик *tf.train.Optimizer.minimize* додасть операції і тензори до графіка за

замовчуванням, який обчислює градієнти, і повертає *tf.Operation*, який при запуску застосує ці градієнти до набору змінних.

2.1.4.2. Створення нейронної мережі за допомогою модуля *Layers*.

У машинному навчанні модель є функцією з параметрами, які можна вивчати, що відображає вхід на вихід. Оптимальні параметри отримують шляхом

навчання моделі за даними. Добре підготовлена модель забезпечує точне відображення від входу до потрібного виходу.

У TensorFlow існує два способи створення моделі машинного навчання:

- використовуючи API "Layers", де створюється модель за допомогою шарів;

- використовуючи API Core з операціями нижчого рівня, такими як tf.matMul(), tf.add () тощо.

Основними шарами для нейронної мережі для порівняння зображень є:

1) шари згортки які виділяють особливі шаблони на зображенні;

2) шари зменшення розміру для зменшення розмірності зображення та прискорення роботи;

3) шари нормалізації для пришвидшення роботи та стабільності моделі;

4) шари активації для нелінійності функції;

5) повнозв’язні шари для представлення закодованого вектору зображення.

Також може бути використана особливість шарів яка дозволяє використовувати однакові ваги для різних шарів.

2.1.5. *OpenCV*.

OpenCV використовується для аналізу широкого діапазону зображень і відео, подібно до розпізнавання обличчя та виявлення об’єктів, редагування зображення, роботизованого бачення, оптичної ідентифікації символів тощо. Обробка зображень здійснюється через *OpenCV*. Фокусується на комп’ютерному баченні в реальному часі.

2.1.5.1. Завантаження, відображення та збереження зображення.

```
import cv2

def loading_displaying_saving():
    img = cv2.imread('girl.jpg', cv2.IMREAD_GRAYSCALE)
    cv2.imshow('girl', img)
    cv2.waitKey(0)
    cv2.imwrite('graygirl.jpg', img)
```

Для завантаження зображення використовується функція *cv2.imread*(), де першим аргументом вказується шлях до зображення, а другим аргументом, який є необов'язковим, вказується, у якому кольоровому просторі необхідно представити зображення. Щоб рахувати зображення в *RGB* – *cv2.IMREAD_COLOR*, у відтінках сірого – *cv2.IMREAD_GRAYSCALE*. За умовчанням цей аргумент набуває значення *cv2.IMREAD_COLOR*. Ця функція повертає 2*D* (для зображення у відтінках сірого) або 3*D* (для кольорового зображення) масив *NumPy*. Форма масиву для кольорового зображення: висота * ширина * 3, де 3 це байти, по одному байту на кожну з компонентів. У відтінках сірого все трохи простіше: висота * ширина.

За допомогою функції *cv2.imshow*() відображається зображення на екрані. Як перший аргумент передається функції назва нашого вікна, а другим аргументом зображення, яке завантажено з диска, проте, якщо далі не вказано функцію *cv2.waitKey*(), то зображення миттєво закриється. Ця функція зупиняє виконання програми до натискання кнопки, яку потрібно передати першим аргументом. Для того, щоб будь-яка клавіша була зарахована, передається 0.

Рис. 2.12. Зліва представлено зображення у відтінках сірого, а праворуч у форматі *RGB*

За допомогою функції *cv2.imwrite*() записуємо зображення у файл у форматі *jpg* (дана бібліотека підтримує всі популярні формати зображень: *png*, *tiff*, *jpeg*, *bmp* тощо, тому можна зберегти зображення у будь-якому з цих форматів), де першим аргументом передається безпосередньо сама назва та розширення, а наступним параметром зображення, яке необхідно зберегти.

2.1.5.2. Доступ до пікселів та маніпулювання ними.

Для того, щоб дізнатися висоту, ширину та кількість каналів зображення можна використовувати атрибут *shape*:

```
print("Висота:"+str(img.shape[0]))
print("Ширина:" + str(img.shape[1]))
print("Кількість каналів:" + str(img.shape[2]))
```

2.1.5.3. Зміна розміру зображення.

Для зміни висоти і ширини зображення в *opencv* є функція *resize*():

```
def resizing():
    res_img = cv2.resize(img, (500, 900), cv2.INTER_NEAREST)
```

Дана функція першим аргументом приймає зображення, розмір якого потрібно змінити, другим — кортеж, який має містити ширину і висоту нового зображення, третім — метод інтерполяції (необов'язковий). Інтерполяція — це алгоритм, який знаходить невідомі проміжні значення за наявним набором відомих значень. Фактично це те, як будуть заповнюватися нові пікселі при модифікації розміру зображення. Наприклад, інтерполяція шляхом найближчого сусіда (*cv2.INTER_NEAREST*) просто береться для кожного пікселя підсумкового зображення один піксель вихідного, який найближчий до його положення — це найпростіший і найшвидший метод. Крім цього методу в *opencv* існують такі: *cv2.INTER_LINEAR*(використовується за замовчуванням), *cv2.INTER_ cv2.INTER_AREA, CUBIC* і *cv2.INTER_LANCZOS4*. Найкращим методом інтерполяції для стиснення зображення є *cv2.INTER_AREA*, для збільшення - *cv2.INTER_LINEAR*. Від цього методу залежить якість кінцевого зображення, але, як показує практика, якщо зменшити/збільшити зображення менше, ніж у 1.5 разів, то не важливо який метод інтерполяції було використано — якість буде схожа. Дане твердження можна перевірити практично, написавши наступний код:

```
res_img_nearest = cv2.resize(img, (int(w / 1.4), int(h / 1.4)),
                             cv2.INTER_NEAREST)
res_img_linear = cv2.resize(img, (int(w / 1.4), int(h / 1.4)),
                            cv2.INTER_LINEAR)
```

2.1.5.4. Зміщення зображення вздовж осей.

За допомогою функції *cv2.warpAffine*() можна переміщувати зображення вліво та вправо, вниз та вгору, а також будь-яку комбінацію з перерахованого:

```
def shifting():
    h, w = img.shape[:2]
    translation_matrix = np.float32([[1, 0, 200], [0, 1, 300]])
    dst = cv2.warpAffine(img, translation_matrix, (w, h))
    cv2.imshow('Изображение, сдвинутое вправо и вниз', dst)
    cv2.waitKey(0)
```

Спочатку в змінній *translation_matrix* створюється матриця перетворень як показано нижче:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Рис. 2.13. Матриця перетворень

Перший рядок матриці - [1, 0, *tx*], де *tx* - кількість пікселів, на які зрушується зображення вліво або вправо. Негативне значення *tx* зрушуватиме зображення вліво, позитивне вправо.

Другий рядок матриці - [0, 1, *ty*], де *ty* - кількість пікселів, на які зрушується зображення вгору або вниз. Негативне значення *ty* зрушуватиме зображення

вгору, позитивне — вниз. Важливо пам'ятати, що ця матриця визначається як масив з плаваючою точкою.

На наступному рядку і відбувається зсув зображення вздовж осей, за допомогою функції *cv2.warpAffine*(), яка першим аргументом приймає зображення, другим - матрицю, третім - розміри зображення. Якщо запустити цей код, видно наступне:

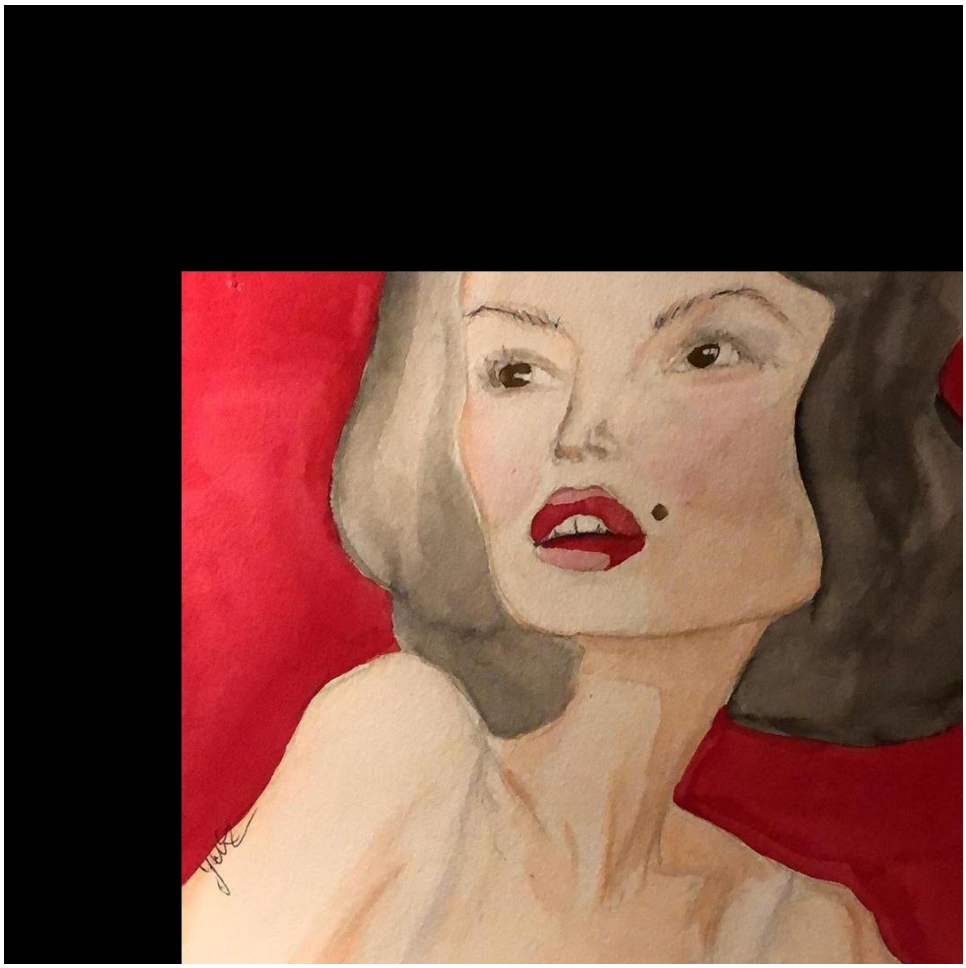

Рис. 2.14. Зсув зображення вздовж осей, за допомогою функції *cv2.warpAffine*

2.1.6. Scikit-learn.

Scikit-learn – один з найбільш широко використовуваних пакетів *Python* для *Data Science* та *Machine Learning*. Він дозволяє виконувати безліч операцій та надає безліч алгоритмів. *Scikit-learn* також пропонує чудову документацію про свої класи, методи та функції, а також опис використовуваних алгоритмів.

Буде використовуватися в проекті:

- *sklearn.preprocessing* – для попередньої обробки даних;
- *sklearn.model_selection* – вибір моделі та взаємодія з нею;
- *sklearn.linear_model* – модель лінійної регресії.

2.1.6.1. Попередня обробка даних.

Scikit-learn можна використовувати для підготовки даних до алгоритмів машинного навчання: стандартизації або нормалізації даних, кодування категоріальних змінних тощо.

Визначається масив *NumPy*:

```
>>> import numpy as np
>>> x = np.array([[0.1, 1.0, 22.8],
...               [0.5, 5.0, 41.2],
...               [1.2, 12.0, 2.8],
...               [0.8, 8.0, 14.0]])
>>> x
array([[ 0.1,  1. , 22.8],
       [ 0.5,  5. , 41.2],
       [ 1.2, 12. ,  2.8],
       [ 0.8,  8. , 14. ]])
```

Рис. 2.15. Визначення масиву *NumPy*

Наприклад, якщо потрібно перетворити дані таким чином, щоб середнє значення кожного стовпця (елемента) дорівнювало нулю, а стандартне відхилення – одиниці. У цьому випадку можна використовувати *sklearn.preprocessing.StandardScaler* (рис. 2.16.).

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> scaled_x = scaler.fit_transform(x)
>>> scaler.scale_
array([ 0.40311289,  4.03112887, 14.04421589])
>>> scaler.mean_
array([ 0.65,  6.5 , 20.2 ])
>>> scaler.var_
array([1.6250e-01, 1.6250e+01, 1.9724e+02])
>>> scaled_x
array([[-1.36438208, -1.36438208,  0.18512959],
       [-0.3721042 , -0.3721042 ,  1.4952775 ],
       [ 1.36438208,  1.36438208, -1.23894421],
       [ 0.3721042 ,  0.3721042 , -0.44146288]])
>>> scaled_x.mean().round(decimals=4)
0.0
>>> scaled_x.mean(axis=0)
array([ 1.66533454e-16, -1.38777878e-17,  1.52655666e-16])
>>> scaled_x.std(axis=0)
array([1., 1., 1.])
>>> scaler.inverse_transform(scaled_x)
array([[ 0.1,  1. , 22.8],
       [ 0.5,  5. , 41.2],
       [ 1.2, 12. ,  2.8],
       [ 0.8,  8. , 14. ]])
```

Рис. 2.16. Використання *sklearn.preprocessing.StandardScaler*

Іноді є деякі категоріальні дані, і потрібно перетворити їх на числа. Один із способів це зробити – використовувати клас *sklearn.preprocessing.OneHotEncoder*. Приклад з масивами ролей у компанії (рис. 2.17.).

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> roles = np.array([('Tom', 'manager'),
...                   ('Mary', 'developer'),
...                   ('Ann', 'recruiter'),
...                   ('Jim', 'developer')])
>>> roles
array([['Tom', 'manager'],
       ['Mary', 'developer'],
       ['Ann', 'recruiter'],
       ['Jim', 'developer']], dtype='<u9')>>> encoder = OneHotEnco
der()
>>> encoded_roles = encoder.fit_transform(roles[:, [1]])
>>> encoded_roles.toarray()
array([[0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])</u9')>
```

Рис. 2.17. Масив ролей у компанії

У наведеному вище прикладі перший стовпець об'єкта *encoded_roles* вказує, чи кожен співробітник є розробником. Другий та четвертий співробітник (Мері та Джим) - розробники. Другий стовпець пов'язаний із посадою менеджера. Тільки перший співробітник має цю посаду. Нарешті, третій стовпець відповідає посаді рекрутера, і це третій співробітник (Енн).

2.1.6.2. Вибір моделі.

Для навчання та тестування моделей машинного навчання, необхідно випадково розбивати дані на підмножини. Це включає як входи, так і їх відповідні виходи. Функція *sklearn.model_selection.train_test_split*() корисна у таких випадках (рис. 2.18).

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> x, y = np.arange(1, 21).reshape(-1, 2), np.arange(3, 40, 4)
>>> x
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12],
       [13, 14],
       [15, 16],
       [17, 18],
       [19, 20]])
>>> y
array([ 3,  7, 11, 15, 19, 23, 27, 31, 35, 39])
>>> x_train, x_test, y_train, y_test =\
...     train_test_split(x, y, test_size=0.4, random_state=0)
>>> x_train
array([[ 3,  4],
       [13, 14],
       [15, 16],
       [ 7,  8],
       [ 1,  2],
       [11, 12]])
>>> y_train
array([ 7, 27, 31, 15,  3, 23])
>>> x_test
array([[ 5,  6],
       [17, 18],
       [ 9, 10],
       [19, 20]])
>>> y_test
array([11, 35, 19, 39])
```

Рис. 2.18. Використання функції *sklearn.model_selection.train_test_split*

Крім звичайного поділу наборів даних, *scikit-learn* надає засоби для здійснення перехресної перевірки, налаштування гіперпараметрів моделей за допомогою пошуку по сітці, обчислення багатьох величин, що показують продуктивність моделі (наприклад, коефіцієнт детермінації, середньоквадратична помилка, показник відхилення з поясненням, матриця помилок, звіт про класифікацію, *f*-показники та багато іншого).

2.1.6.3. Отримання моделі лінійної регресії

```
from sklearn. linear_model import LinearRegression

# ініціалізація моделі лінійної регресії
model = LinearRegression()

# визначити змінні предиктора та відповіді
x, y = df[['x1', 'x2']], df.y

# застосувати модель регресії
model.fit(x, y)
```

Після цього можна використовувати наступний код для отримання коефіцієнтів регресії моделі разом зі значенням R-квадрату моделі:

```
#display regression coefficients and R-squared value of model
print(model. intercept_ , model. coef_ , model. score (X, y))

70.4828205704 [ 5.7945 -1.1576] 0.766742556527
```

Рис. 2.19. Отримання коефіцієнтів регресії моделі

Використовуючи цей висновок, ми можемо написати рівняння для обраної моделі регресії:

$$y = 70{,}48 + 5{,}79 * 1 - 1{,}16 * 2$$

Ми також можемо бачити, що *R* 2 моделі дорівнює 76,67.

Це означає, що 76,67% варіації змінної відгуку можна пояснити двома змінними-предикторами моделі.

2.2. Висновки до розділу

У другому розділі дипломної роботи було розглянуто основні використані бібліотеки при написанні програми та опис основних методів для взаємодії з ними.

РОЗДІЛ 3
РЕЗУЛЬТАТИ РОБОТИ ТА АНАЛІЗ

3.1. Написання коду програми

Робота відбувається в середовищі розробки *Visual Studio Code* за допомогою *Jupyter Notebook*.

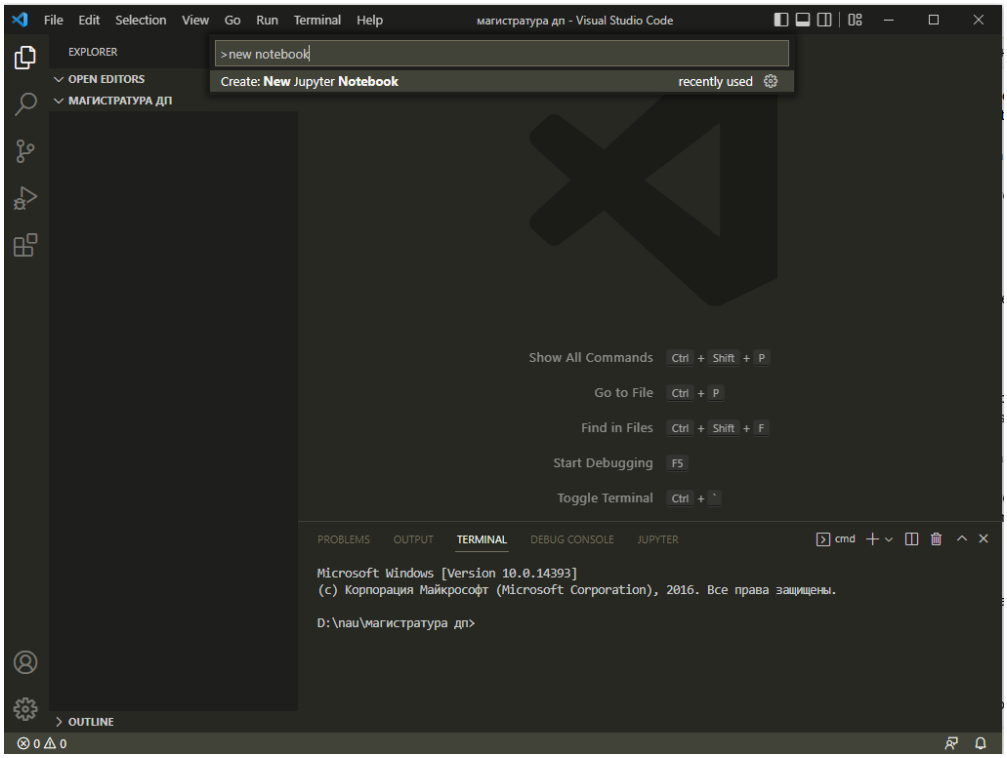

Рис. 3.1. Використання *Jupyter Notebook*

Спочатку імпортуються всі необхідні бібліотеки.

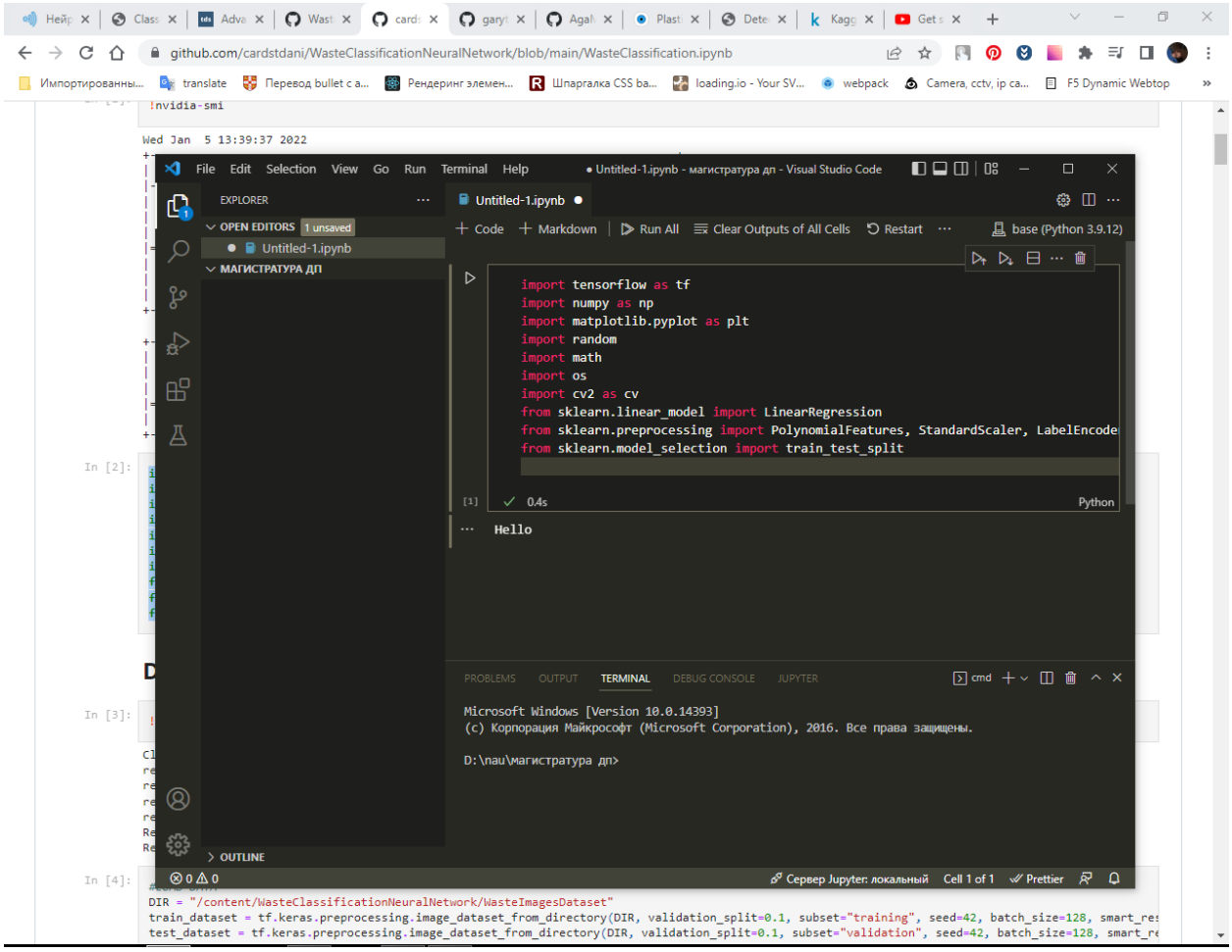

Рис. 3.2. Імпортування бібліотек

3.1.1. Збір та обробка даних

Підтягую дані з зображеннями з *git* репозиторію: https://github.com/cardstdani/WasteClassificationNeuralNetwork. Після стягування, доступ до даних можна отримати в папці з проектом.

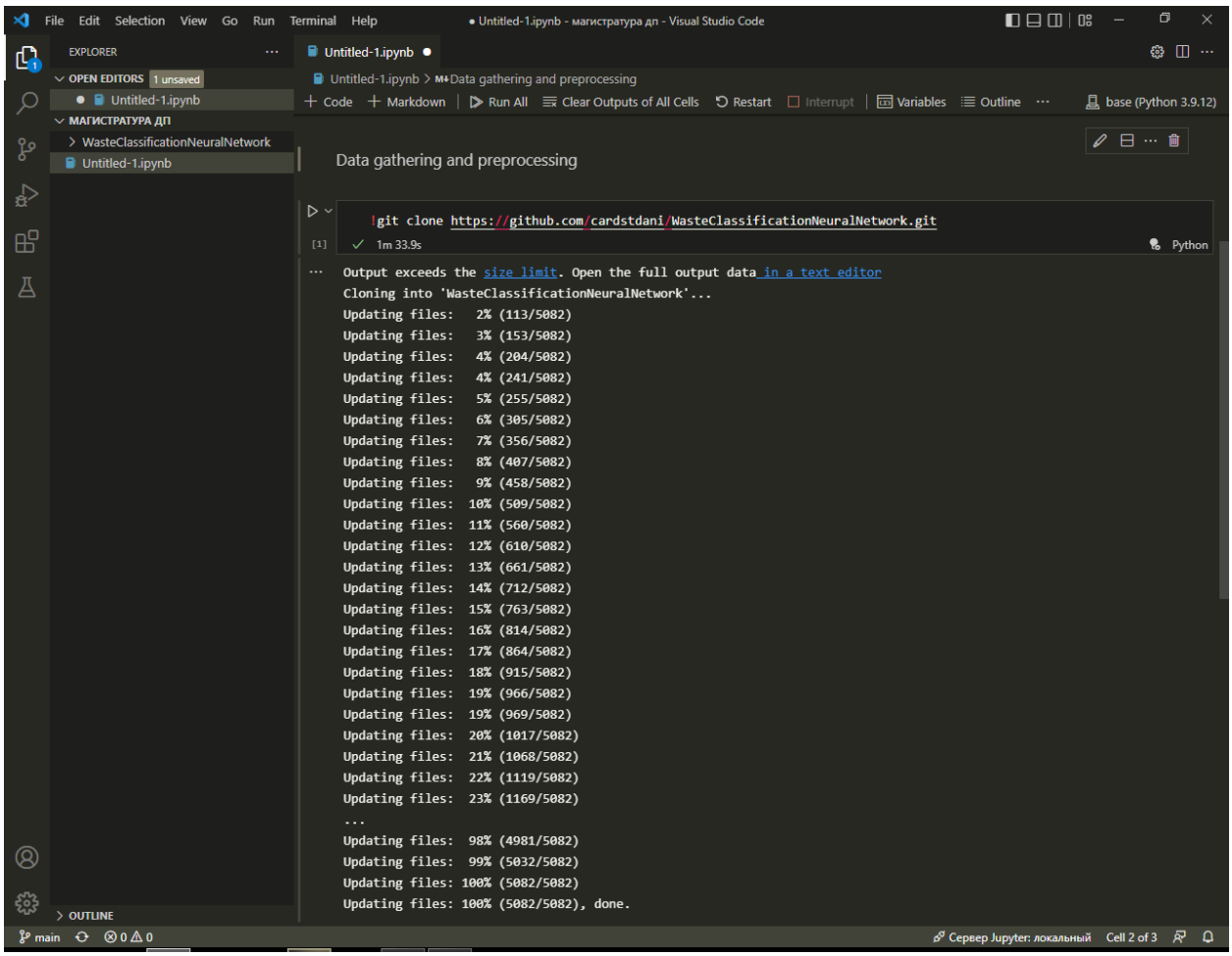

Рис. 3.3. Завантаження зображень з *git* репозиторію

Анотації цих зображень спеціально потрібні для навчання даних у моделі.

Щоб отримати анотації, призначені для маркування зображень використовується інструмент *LabelIMG*.

В інструменті *LabelImg* потрібно позначити відповідну область зображення класом, з яким воно пов’язано, наприклад картон, метал, скло тощо. Цей тип позначення зображення включає стратегію створення меж для об’єкта відходів на зображенні. 80% зображень для кожної категорії в усьому наборі зображень відходів використовуються для навчання. Інші 20% зображень призначені для фази тестування. Загальна мета цієї конкретної системи полягає в тому, щоб змусити структуру визначити, як виявляти об’єкти. Дані анотацій зображень, зроблених інструментом *LabelIMG*, зберігаються в форматі

розширення *XML*. Анотації до зображення включають розмір зображення, координати розташування, а також назви позначених об'єктів, які можна переглянути у файлі. *XML* потім можна перетворити на файл *CSV*. І цей файл *CSV* містить записи доступної інформації, яка включає назву записаного файлу, висоту та ширину, а також позначені класифікації та позиції координат (*xmin, ymin, xmax, ymax*), отримані із зображення. Цей перетворений файл *CSV* є одним із основних вхідних даних для навчання моделі.

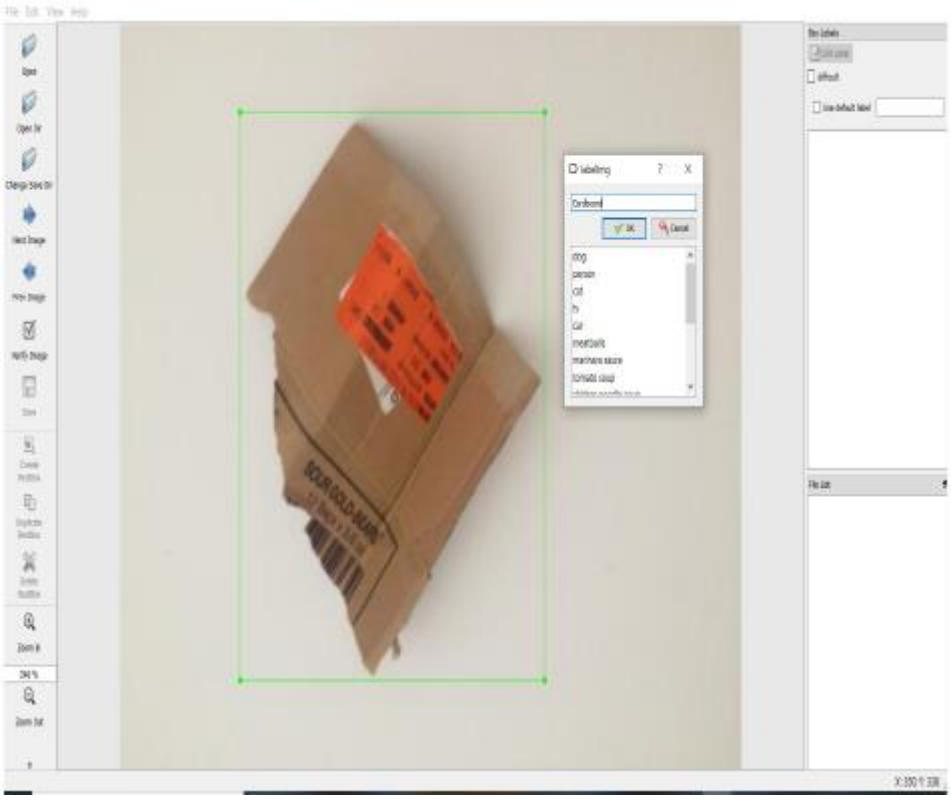

Рис. 3.4. Позначення окремого об’єкта із зображення

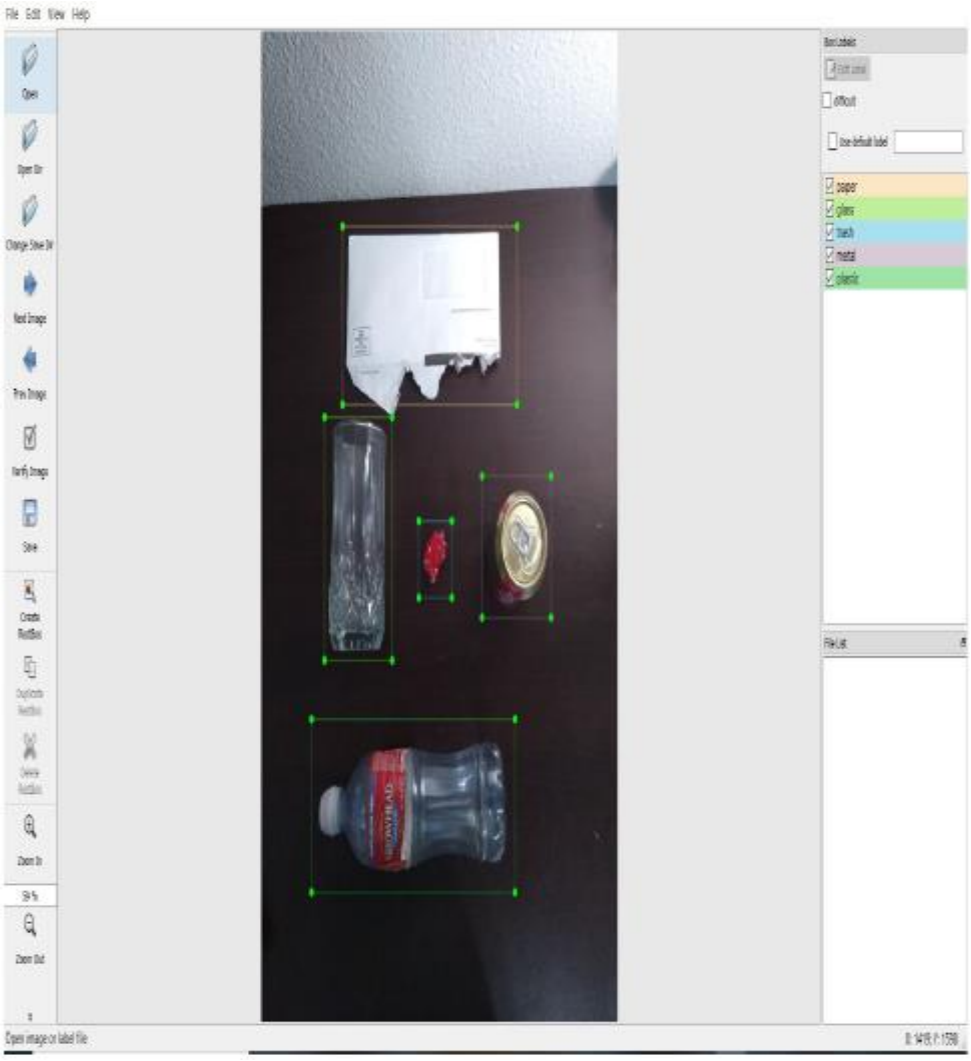

Рис. 3.5. Позначення декількох об’єктів із зображення

На рис. 3.4. і рис. 3.5. показано, як інструмент *LabelIMG* позначає один або декілька об’єктів із зображення.

Спочатку завантажую бібліотеку *LabelIMG* з *github* (рис. 3.6.).

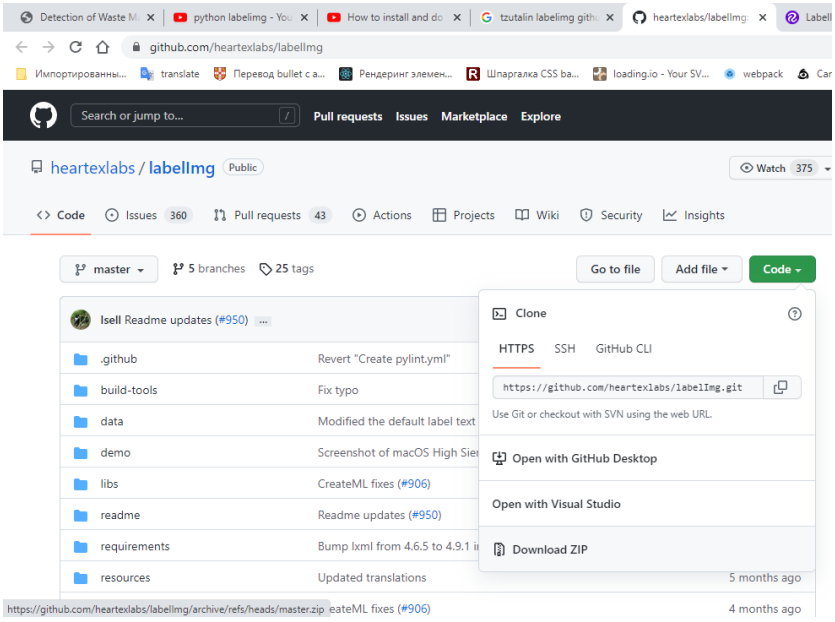

Рис. 3.6. Завантаження бібліотеки *LabelIMG*

Відкриваю термінал за допомогою *Anaconda Prompt* та встановлюю необхідні бібліотеки для роботи *LabelIMG*.

```
Anaconda Prompt (anaconda3)
(base) C:\Users\User>cd C:\Users\User\Downloads\Programs

(base) C:\Users\User\Downloads\Programs>conda activate label

(label) C:\Users\User\Downloads\Programs>conda install pyqt=5
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\User\anaconda3\envs\label

  added / updated specs:
    - pyqt=5


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2022.10.11 |       haa95532_0         125 KB
    openssl-1.1.1s             |       h2bbff1b_0         5.5 MB
    pyqt-5.9.2                 |   py36h6538335_2         3.3 MB
    sip-4.19.8                 |   py36h6538335_0         262 KB
    zlib-1.2.13                |       h8cc25b3_0         113 KB
    ------------------------------------------------------------
                                           Total:         9.3 MB

The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/win-64::ca-certificates-2022.10.11-haa95532_0 None
  icu                pkgs/main/win-64::icu-58.2-ha925a31_3 None
  jpeg               pkgs/main/win-64::jpeg-9e-h2bbff1b_0 None
  libpng             pkgs/main/win-64::libpng-1.6.37-h2a8f88b_0 None
  openssl            pkgs/main/win-64::openssl-1.1.1s-h2bbff1b_0 None
  pyqt               pkgs/main/win-64::pyqt-5.9.2-py36h6538335_2 None
  qt                 pkgs/main/win-64::qt-5.9.7-vc14h73c81de_0 None
  sip                pkgs/main/win-64::sip-4.19.8-py36h6538335_0 None
  sqlite             pkgs/main/win-64::sqlite-3.39.3-h2bbff1b_0 None
  zlib               pkgs/main/win-64::zlib-1.2.13-h8cc25b3_0 None
```

Рис. 3.7. Встановлення бібліотек для *LabelIMG*.

В терміналі ввожу команду *python labelImg.py* щоб відкрити програму (рис. 3.8).

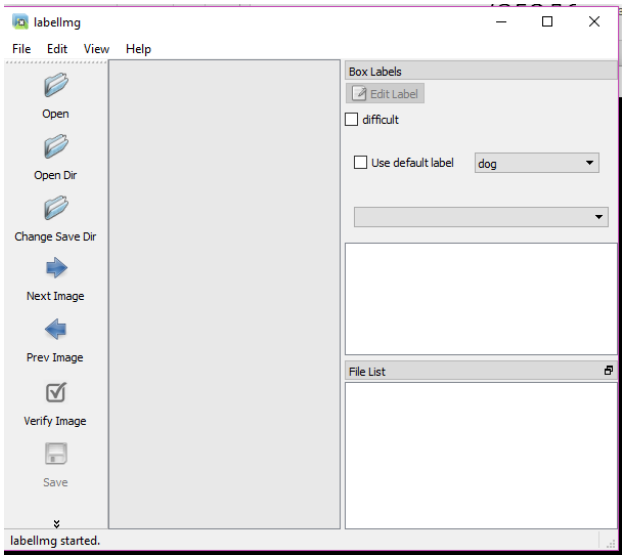

Рис. 3.8. Програма *LabelIMG*

Вибираю зображення із папки з набором даних. Спочатку із папки *Aluminium*.

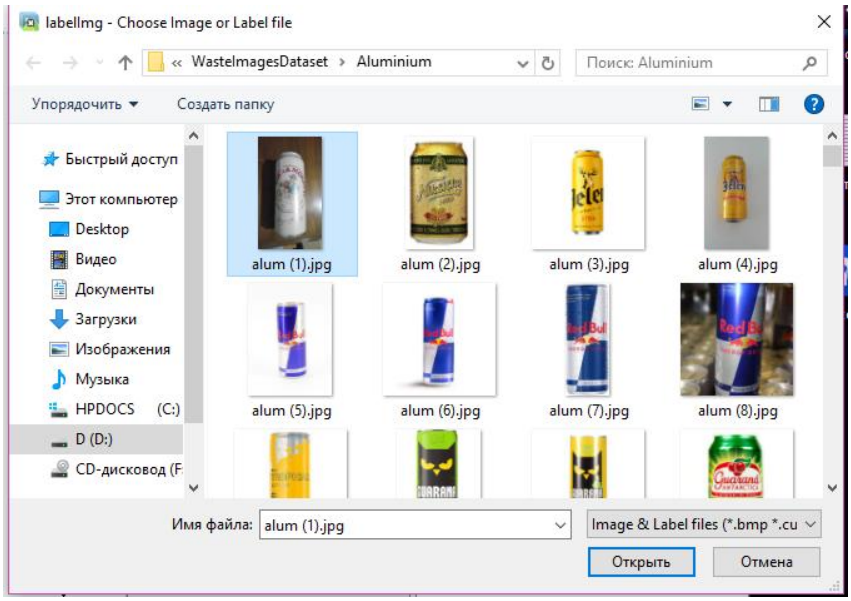

Рис. 3.9. Вибір зображення

Натискаю *Create RectBox* для створення рамки.

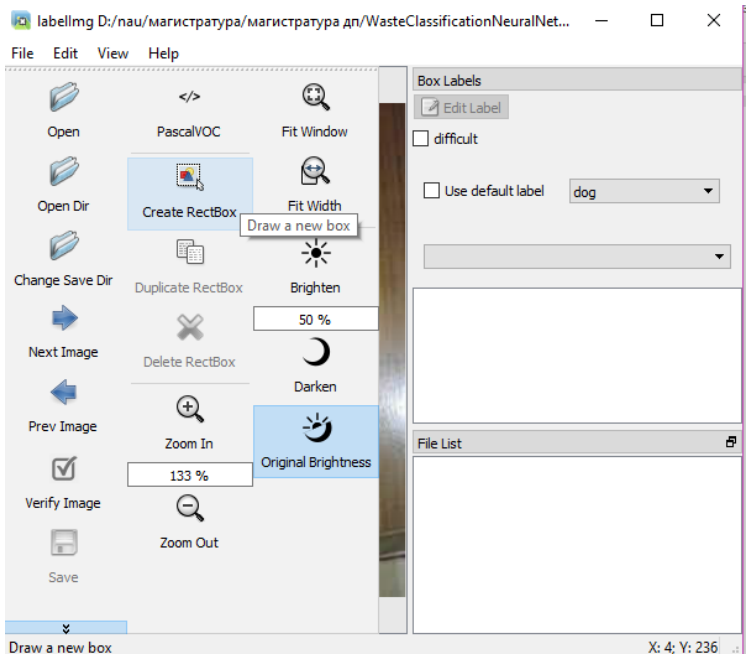

Рис. 3.10. *Create RectBox*

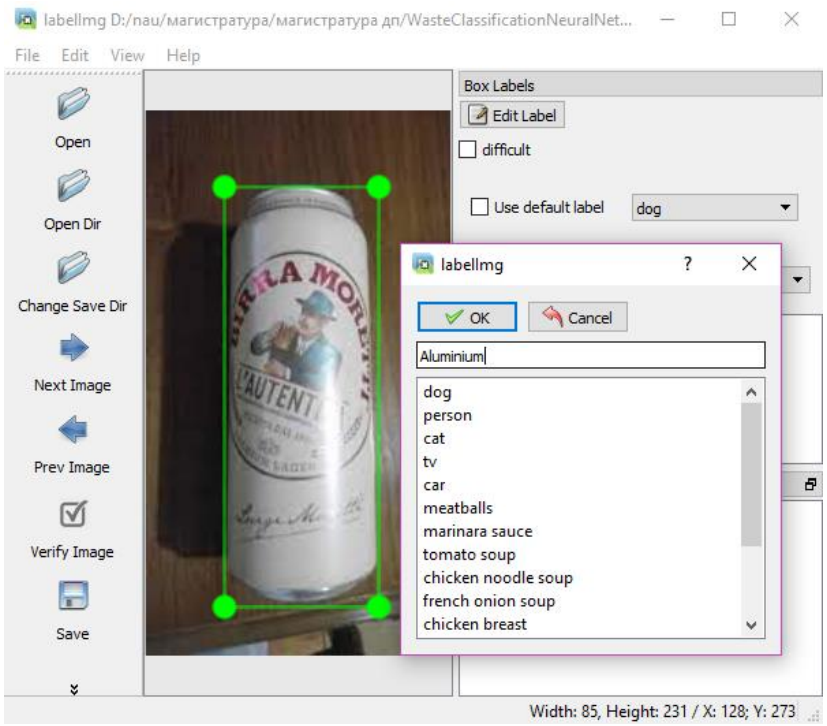

Рис. 3.11. Створення напису

Після цього натискаю *Save* та зберігаю дані в *xml* файлі в папці з проєктом. Таким же чином виконую анотацію інших зображень.

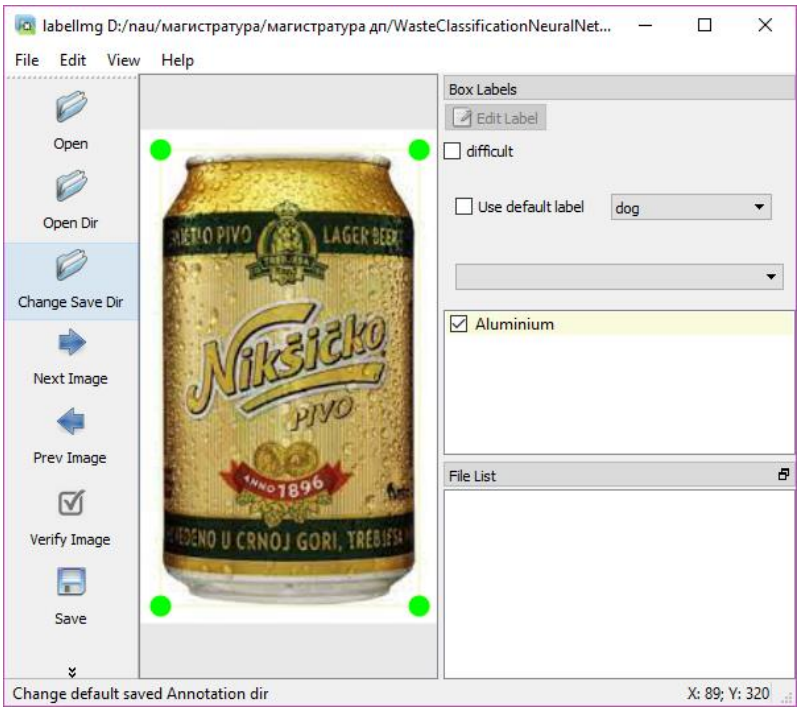

Рис. 3.12. Анотація інших зображень

Розділяю зображення з відповідними *xml* файлами, із кожної папки (*Aluminium, Carton, Glass, Organic Waste, Other Plastics, Paper and Cardboard, Plastic, Textiles, Wood*) – 80% у папку *train* і 20% у папку *test*.

Перетворюю всі *xml* файли в *csv* файл для кожної папки (*train* і *test*).

Для цього в папці з проєктом створюю файл *xml_to_csv.py* і записую наступний код:

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height',
                   'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df
```

```
def main():
    for directory in ['train', 'test']:
        image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
        print('Successfully converted xml to csv.')


main()
```

Створюю папки *data* і *training* і запускаю код за допомогою команди *python xml_to_csv.py* (рис. 3.13.).

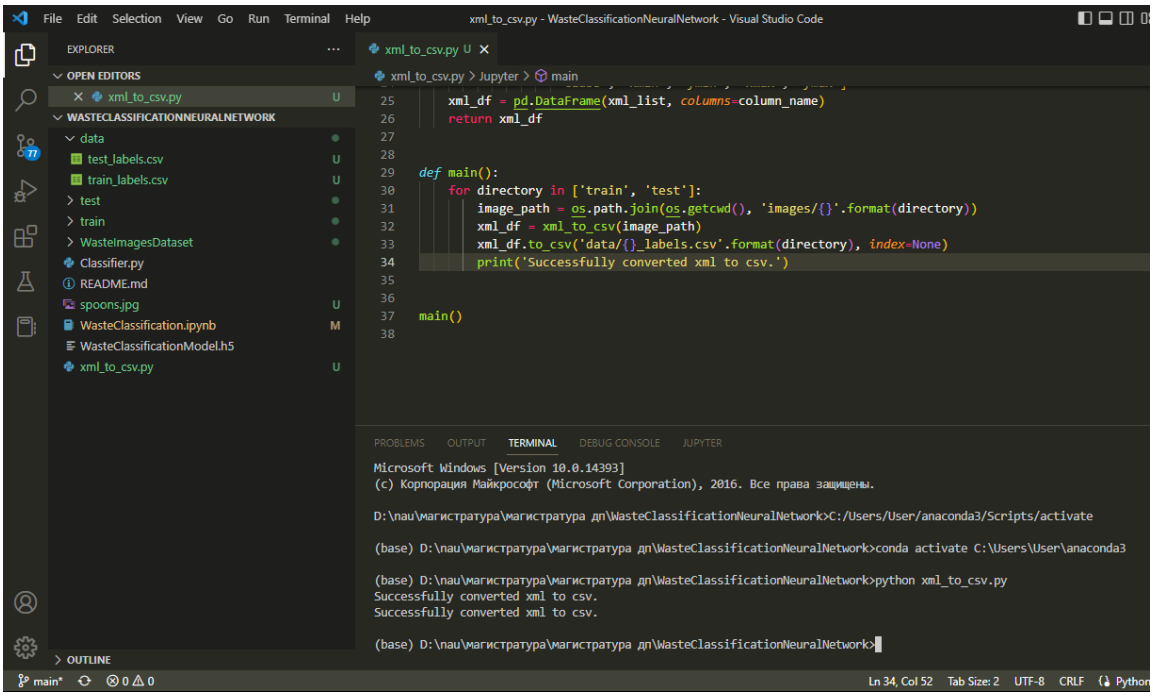

Рис. 3.13. Результат виконання коду *python xml_to_csv.py*

В результаті було створено два файли *csv* в папці *data.*

Далі створюю *tfrecord* файл. Формат файлу *TFRecord* — це власний бінарний формат зберігання *tensorflow*, який займає менше місця на диску. Це може мати значний вплив на продуктивність конвеєра імпорту, а також на час

навчання моделі. Для створення файлу *.record* створюю файл документа у робочому каталозі та записую наступний код у цей файл.

```
import os
import io
import pandas as pd
import tensorflow as tf

from PIL import Image
from models.research.object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict

flags = tf.compat.v1.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
flags.DEFINE_string('image_dir', '', 'Path to images')
FLAGS = flags.FLAGS

# пишу умову якщо row_label дорівнює одній із анотацій:
def class_text_to_int(row_label):
    if row_label == 'Aluminium':
        return 1
    if row_label == 'Carton':
        return 2
    if row_label == 'Glass':
        return 3
    if row_label == 'Organic Waste':
        return 4
    if row_label == 'Other Plastics':
```

```
            return 5
        if row_label == 'Paper and Cardboard':
            return 6
        if row_label == 'Plastic':
            return 7
        if row_label == 'Textiles':
            return 8
        if row_label == 'Wood':
            return 9
        else:
            return 0


    def split(df, group):
        data = namedtuple('data', ['filename', 'object'])
        gb = df.groupby(group)
        return [data(filename, gb.get_group(x)) for filename, x in
zip(gb.groups.keys(), gb.groups)]


    def create_tf_example(group, path):
        with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
            encoded_jpg = fid.read()
        encoded_jpg_io = io.BytesIO(encoded_jpg)
        image = Image.open(encoded_jpg_io)
        width, height = image.size

        filename = group.filename.encode('utf8')
        image_format = b'jpg'
```

```
xmins = []
xmaxs = []
ymins = []
ymaxs = []
classes_text = []
classes = []

for index, row in group.object.iterrows():
    xmins.append(row['xmin'] / width)
    xmaxs.append(row['xmax'] / width)
    ymins.append(row['ymin'] / height)
    ymaxs.append(row['ymax'] / height)
    classes_text.append(row['class'].encode('utf8'))
    classes.append(class_text_to_int(row['class']))

tf_example = tf.train.Example(features=tf.train.Features(feature={
    'image/height': dataset_util.int64_feature(height),
    'image/width': dataset_util.int64_feature(width),
    'image/filename': dataset_util.bytes_feature(filename),
    'image/source_id': dataset_util.bytes_feature(filename),
    'image/encoded': dataset_util.bytes_feature(encoded_jpg),
    'image/format': dataset_util.bytes_feature(image_format),
    'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
    'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
    'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
    'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
    'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
    'image/object/class/label': dataset_util.int64_list_feature(classes),
}))
```

```
    return tf_example


def main(_):
    writer = tf.io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(FLAGS.image_dir)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())

    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully created the TFRecords: {}'.format(output_path))


if __name__ == '__main__':
    tf.compat.v1.app.run()
```

Далі копіюю моделі *TensorFlow* із репозиторію з *github* за допомогою *git clone* https://github.com/tensorflow/models.git (рис. 3.14.).

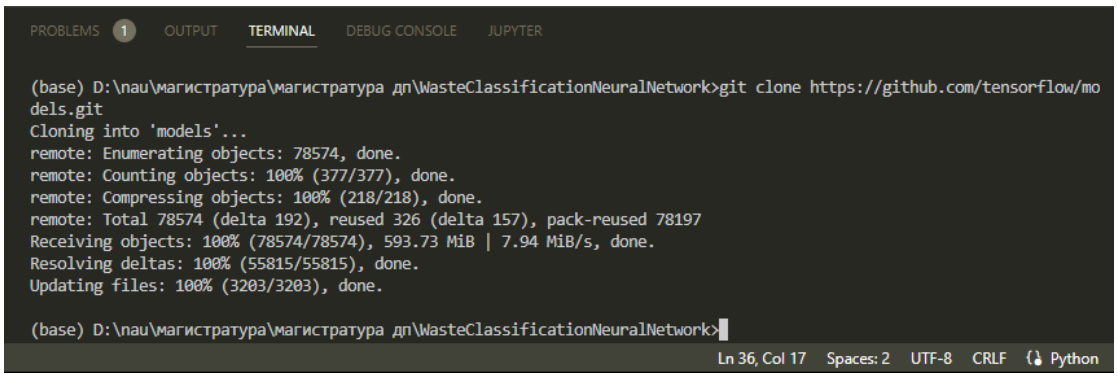

Рис. 3.14. Завантаження моделей з *git* репозиторію

Виконую серіалізацію за допомогою команди *protoc object_detection/protos/*.proto --python_out=.*

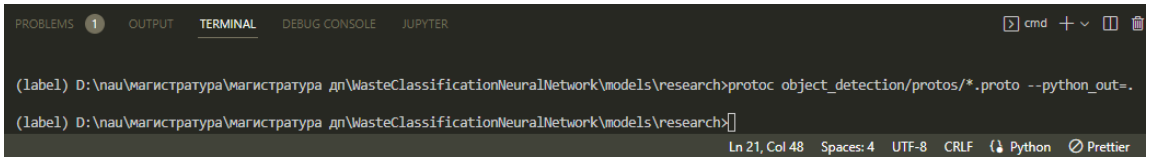

Рис. 3.15. Серіалізація файлів *object_detection*

Встановлюю бібліотеку *object_detection*, виконавши такі дії з каталогу *models\reserch\slim*:

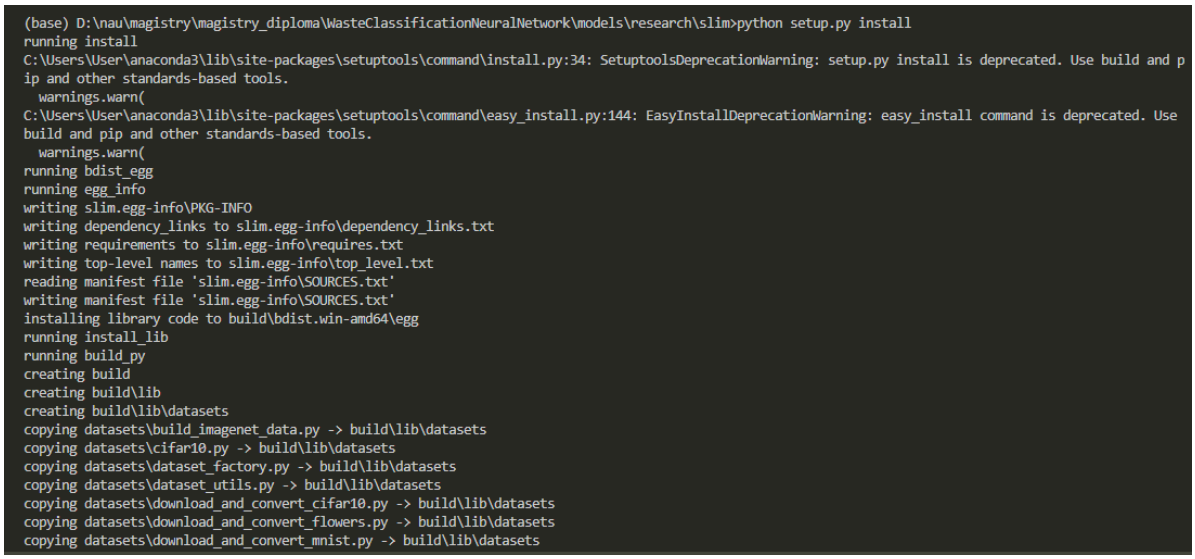

Рис. 3.16. Встановлення бібліотеки *object_detection*

Тепер можна запустити скрипт *generate_tfrecord.py*. Запускаю його двічі, один раз для *train TFRecord* і один раз для *test TFRecord*.

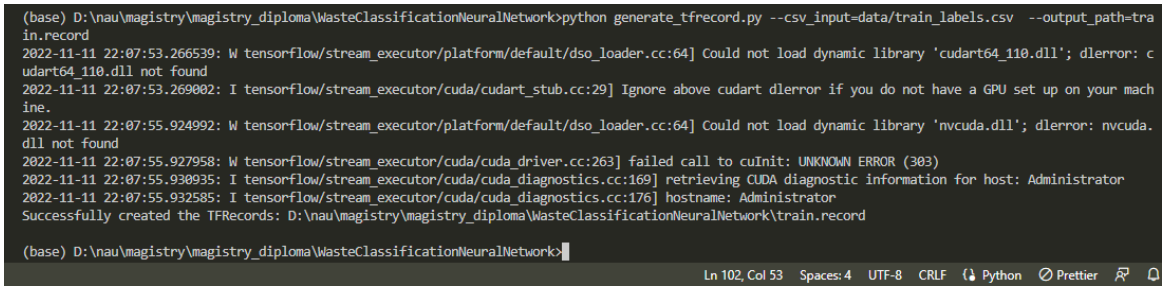

Рис. 3.17. Створення файлу *train TFRecord*

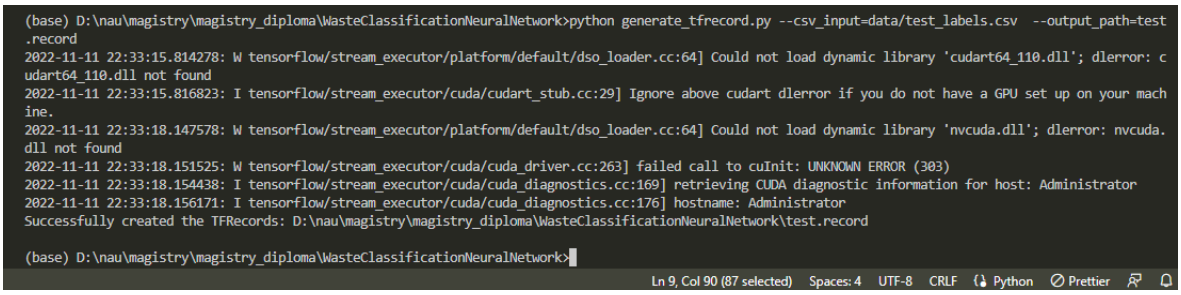

Рис. 3.18. Створення файлу *test TFRecord*

Як можна побачити на рис. 3.17. і рис. 3.18. файли були успішно створені.

Наступний код використовує *Keras API* бібліотеки, *Tensorflow* для попередньої обробки набору даних, розміщеного в папці, шляхом зміни розміру всіх зображень до стандартного розміру 256x256 і встановлення розміру пакета 128, тобто в процесі тренування дані проходитимуть через мережу у вигляді фрагментів по 128 зображень.

Розділення даних на два набори: тренування та тестування:

DIR = "/content/WasteClassificationNeuralNetwork/WasteImagesDataset"

train_dataset = tf.keras.preprocessing.image_dataset_from_directory(DIR, validation_split=0.1, subset="training", seed=42, batch_size=128, smart_resize=True, image_size=(256, 256))

test_dataset = tf.keras.preprocessing.image_dataset_from_directory(DIR, validation_split=0.1, subset="validation", seed=42, batch_size=128, smart_resize=True, image_size=(256, 256))

Крім того, можна зберегти кількість класів у змінній, витягнувши її з об’єкта набору даних тренування (у цьому випадку 9 класів) і використовувати *tf.data.AUTOTUNE* для оптимізації продуктивності як тренувальних, так і тестових об’єктів набору даних.

```
classes = train_dataset.class_names
numClasses = len(train_dataset.class_names)
print(classes)

AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

3.1.2. Побудова моделі та навчання

Оскільки маємо справу з відносно великим набором даних (понад 5000 зображень), під час навчання моделі на такому обсязі даних зручно використовувати звичайну техніку під назвою *Transfer Learning*. Це стосується заміни згорткової частини моделі на вже навчену. Тож перед навчанням модель зможе витягти корисні функції із вхідних зображень завдяки навченій згортковій частині. Крім того, доведеться тренувати лише кілька останніх щільних шарів, зменшуючи обчислювальну потужність і час. Доступно багато навчених згорткових моделей, але найпоширеніші включені в *Keras API*, який використовується для проєкту.

Використання *Keras API:*

```
baseModel = tf.keras.applications.MobileNetV3Large(input_shape=(256,
256,3), weights='imagenet', include_top=False, classes=numClasses)
for layers in baseModel.layers[:-6]:
  layers.trainable=False

last_output = baseModel.layers[-1].output
x = tf.keras.layers.Dropout(0.45) (last_output)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.BatchNormalization() (x)
x = tf.keras.layers.Dense(256, activation = tf.keras.activations.elu,
kernel_regularizer=tf.keras.regularizers.l1(0.045),
activity_regularizer=tf.keras.regularizers.l1(0.045),
kernel_initializer='he_normal')(x)
x = tf.keras.layers.Dropout(0.45) (x)
x = tf.keras.layers.Dense(numClasses, activation='softmax')(x)

model = tf.keras.Model(inputs=baseModel.input,outputs=x)
```

Як можна побачити в коді, модель побудована на основі попередньо навченої моделі *MobileNetV3Large* без оригінальних остаточних щільних шарів, які замінені прихованим шаром із 256 нейронів, який отримує дані від операції *Global Average Pooling*, шару *BatchNormalization* для виправлення внутрішнього коваріатного зсуву, функції активації *ELU* та кількох *Dropouts*. Після цього останній шар містить стільки нейронів, скільки вихідних класів визначається змінною *'numClasses'*. Щоб запобігти переобладнанню, головній причині відсутності узагальнення потрібно додати регулярізацію $L1$.

Окрім перенесення навчання, іноді доцільно розморозити кілька останніх згорткових шарів попередньо навченої згорткової моделі. Ця практика

називається *Fine Tuning* і покращує загальну продуктивність моделі. Тут лише останні шість шарів розморожені (піддаються навчанню).

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00125),
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

epochs = 50
lrCallback = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-3 * 10 **
(epoch / 30))
stepDecay = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 0.1 *
0.1**math.floor(epoch / 6))
history = model.fit(train_dataset, validation_data=test_dataset, epochs=epochs,
callbacks=[])
```

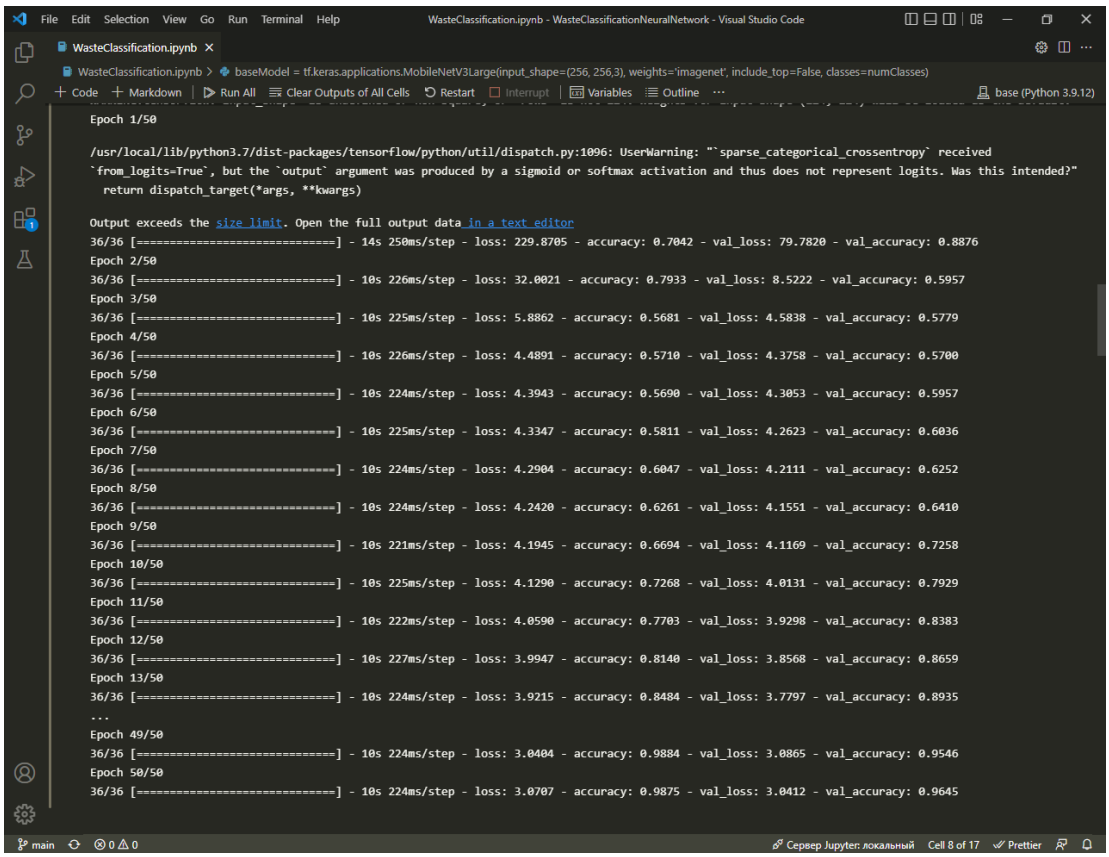

Рис. 3.19. Навчання моделі за допомогою *model.fit*()

Коли модель побудована, виконується її компіляція, призначаючи функцію втрат *SparseCategoricalCrossentropy*, алгоритм Адама для оптимізації всіх параметрів мережі та метрику точності. В результаті, модель навчається за допомогою функції *fit*() протягом 50 епох (часів, коли весь набір даних проходить через мережу).

Зберігаю побудовану модель:

```
model.save("/content/model.h5")
```

```
model.summary()
```

3.2. Оцінка моделі

Тривалість навчання залежить від апаратного забезпечення, складності моделі та розміру набору даних. В цьому випадку, з розміром набору даних приблизно 5000 зображень і моделлю з приблизно 4500000 параметрами, лише близько півтора мільйона з них можна навчити, завершення зайняло близько семи хвилин навчального процесу.

Побудова значення втрат за епохами за допомогою бібліотеки *Matplotlib:*

```
plt.plot(range(0, epochs), history.history["loss"], color="b", label="Loss")
plt.plot(range(0, epochs), history.history["val_loss"], color="r", label="Test Loss")
plt.legend()
plt.show()
```

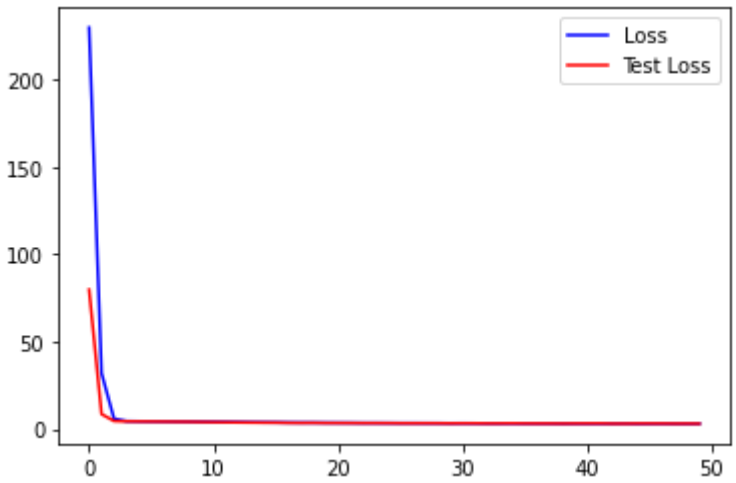

Рис. 3.20. Діаграма втрат

Побудова значення точності за епохами за допомогою бібліотеки *Matplotlib:*

```
plt.plot(range(0, epochs), history.history["accuracy"], color="b",
label="Accuracy")
plt.plot(range(0, epochs), history.history["val_accuracy"], color="r",
label="Test Accuracy")
plt.legend()
plt.show()
```

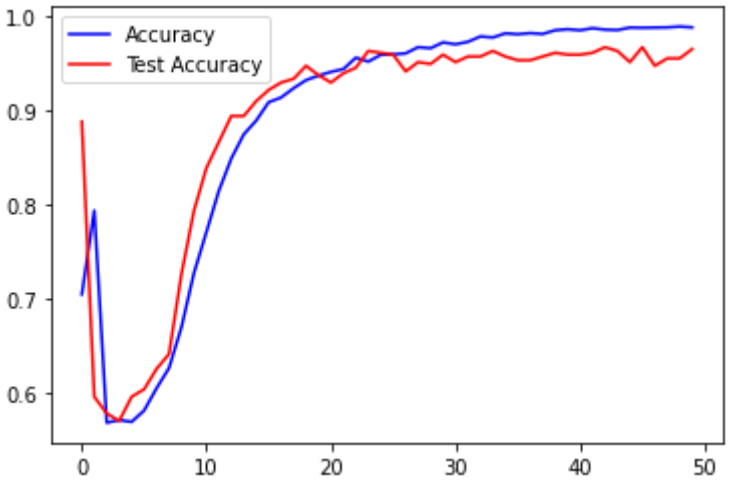

Рис. 3.21. Діаграма точності

Після завершення навчання можна побудувати значення втрат і точності за епохами на графіку за допомогою бібліотеки *Matplotlib*, щоб перевірити, як модель поводилася під час процесу. Як можна побачити на рис. 3.21., точність зростає протягом найперших епох подібним чином для значень тренування та тесту (по одному для кожного набору даних), доки не настане момент, коли точність тесту стане нижчою за точність тренування (синя лінія).

Це показник переоснащення. Чим більша різниця між двома значеннями тренінгу та тесту, тим більше переоснащення та менш узагальнена модель. Незважаючи на втрату узагальнення, різниця в цьому випадку становить не

більше 2 %, від 98,75 % точності під час навчання до 96,45 % під час тестування, що суттєво не впливає на результати моделі.

Побудова рейтингу навчання відносно втрат:

```
plt.xlim([0, 0.003])
learning_rates = 1e-3 * (10 ** (np.arange(epochs) / 30))
plt.plot(learning_rates, history.history['loss'], lw=3, color='#48e073')
plt.title('Learning rate vs. loss', size=20)
plt.xlabel('Learning rate', size=14)
plt.ylabel('Loss', size=14)
```

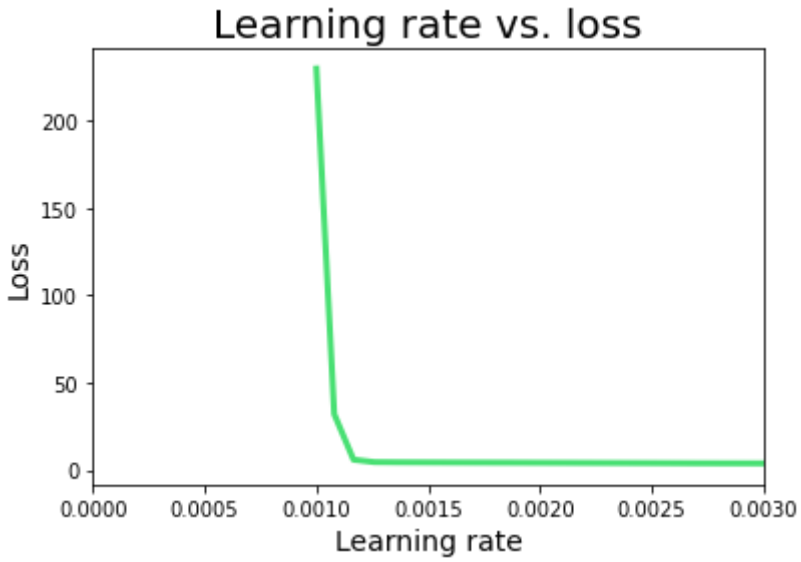

Рис. 3.22. Рейтинг навчання відносно втрат

Можна побачити кінцеві результати, побудувавши матрицю помилок та оцінивши модель за допомогою двох наборів даних. Отримана точність сягає 98%, але описана раніше проблема переоснащення може знизити це значення до 97% або навіть 96%.

Побудова матриці помилок:

```
def plot_confusion_matrix(cm, target_names, cmap=None):
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title('Confusion matrix')
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="black")

    plt.tight_layout()
    plt.ylabel('True label')
```

```
    plt.xlabel('Predicted label\naccuracy={:0.4f}%;
misclass={:0.4f}%'.format(accuracy, misclass))
    plt.show()

plt.figure(figsize=(10, 10))
true = []
predictions = []

"""
for images, labels in test_dataset.take(50):
  pred = model.predict(images)
  for i in range(32):
    try:
      ax = plt.subplot(4, 8, i + 1)
      plt.imshow(images[i].numpy().astype("uint8"))
      #print(classes[np.argmax(pred[i])], 100 * np.max(pred[i]), "real = " +
str(classes[labels[i]]))

      true.append(labels[i])
      predictions.append(np.argmax(pred[i]))

      plt.title(classes[labels[i]])
      plt.axis("off")
    except:
      print()

"""
path = "/content/WasteClassificationNeuralNetwork/WasteImagesDataset"
for i in os.listdir(path):
  folderPath = os.path.join(path, i)
```

```
  for j in os.listdir(folderPath)[:550]:
    fullPath = os.path.join(folderPath, j)
    try:
      img = tf.keras.preprocessing.image.load_img(fullPath, target_size=(256,
256))
      img_array = tf.keras.preprocessing.image.img_to_array(img)
      img_array = tf.expand_dims(img_array, 0)

      preds = model.predict(img_array)
      true.append(classes.index(i))
      predictions.append(np.argmax(preds))
    except:
      print("Error on image:", fullPath)

plot_confusion_matrix(tf.math.confusion_matrix(true, predictions), classes)
```

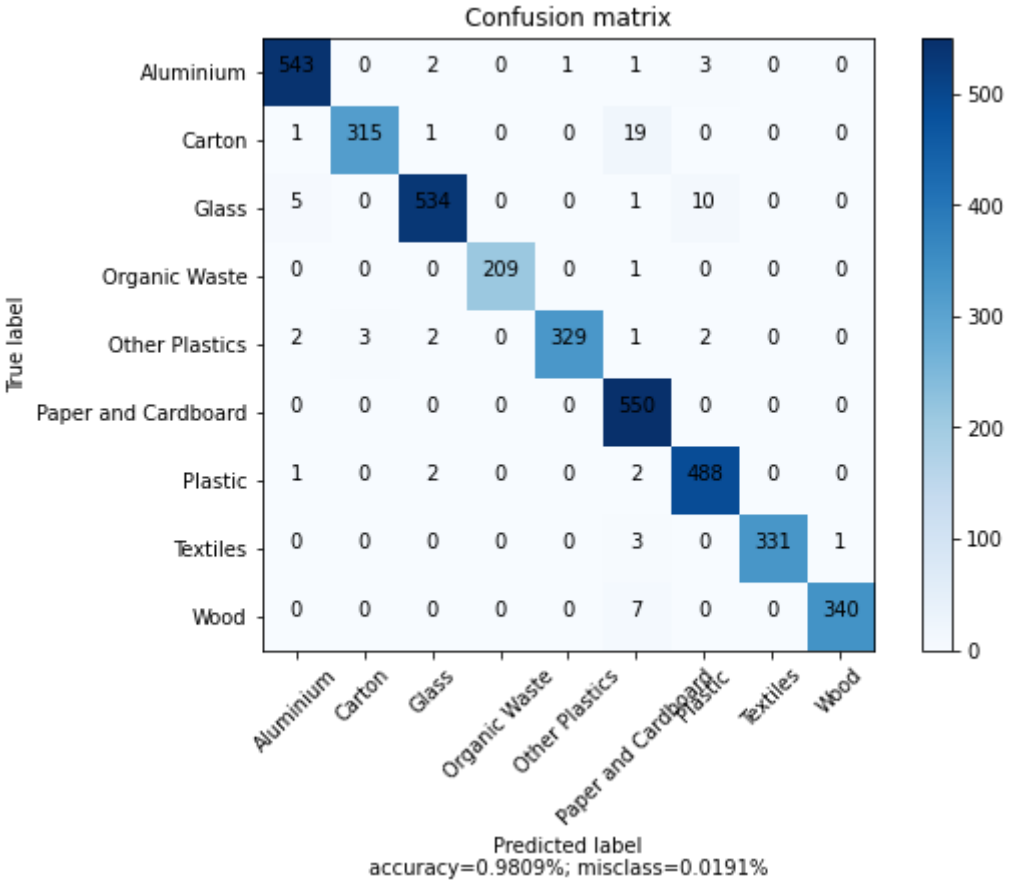

Рис. 3.23. Матриця помилок

Тим не менш, найкращий спосіб перевірити продуктивність моделі — розгорнути її у виробництві та оцінити за допомогою великої кількості «невидимих» даних.

Щоб зробити прогнози для одного зображення, що є основним використанням цієї моделі, можна перетворити вхідне зображення (рис. 3.24.) на масив значень, попередньо обробивши його за допомогою *API Keras* і скориставшись функцією *predict*(), щоб отримати прогнози з моделі.

```
import requests

img_data = requests.get("https://images.unsplash.com/photo-1591872203534-278fc084969e?ixlib=rb1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=1064&q=80").content
with open('img.jpg', 'wb') as handler:
    handler.write(img_data)

path = "/content/img.jpg"

img = tf.keras.preprocessing.image.load_img(path, target_size=(256, 256))
img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)

plt.imshow(img)
print(predictions[0]*100, "\n", classes)
print("Prediction: ", classes[np.argmax(predictions)], f"{predictions[0][np.argmax(predictions)]*100}%")
```

Рис. 3.24. Вхідне зображення

Наприклад, це зображення пластикових ложок дасть такий результат:

Рис. 3.25. Перетворення вхідного зображення на масив значень та використання функції *predict*()

Grad-CAM використовує градієнти будь-якого цільового поняття що вливаються в останній згортковий шар для створення грубої карти локалізації, що виділяє важливі області зображення для прогнозування.

Використання класу *Grad-CAM:*

```
from tensorflow.keras.models import Model
import tensorflow as tf
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import imutils

class GradCAM:
  def __init__(self, model, classIdx, layerName=None):
    self.model = model
    self.classIdx = classIdx
    self.layerName = layerName
    if self.layerName is None:
      self.layerName = self.find_target_layer()

  def find_target_layer(self):
    for layer in reversed(self.model.layers):
      if len(layer.output_shape) == 4:
        return layer.name
    raise ValueError("Could not find 4D layer. Cannot apply GradCAM.")

  def compute_heatmap(self, image, eps=1e-8):
    gradModel = Model(
                        inputs=[self.model.inputs],
                        outputs=[self.model.get_layer(self.layerName).output,
```

```
                                    self.model.output])
    with tf.GradientTape() as tape:
      inputs = tf.cast(image, tf.float32)
      (convOutputs, predictions) = gradModel(inputs)
      loss = predictions[:, self.classIdx]
    grads = tape.gradient(loss, convOutputs)
    castConvOutputs = tf.cast(convOutputs > 0, "float32")
    castGrads = tf.cast(grads > 0, "float32")
    guidedGrads = castConvOutputs * castGrads * grads
    convOutputs = convOutputs[0]
    guidedGrads = guidedGrads[0]
    weights = tf.reduce_mean(guidedGrads, axis=(0, 1))
    cam = tf.reduce_sum(tf.multiply(weights, convOutputs), axis=-1)
    (w, h) = (image.shape[2], image.shape[1])
    heatmap = cv2.resize(cam.numpy(), (w, h))
    numer = heatmap - np.min(heatmap)
    denom = (heatmap.max() - heatmap.min()) + eps
    heatmap = numer / denom
    heatmap = (heatmap * 255).astype("uint8")
    return heatmap


  def overlay_heatmap(self, heatmap, image, alpha=0.5,
colormap=cv2.COLORMAP_VIRIDIS):
    heatmap = cv2.applyColorMap(heatmap, colormap)
    output = cv2.addWeighted(image, alpha, heatmap, 1 - alpha, 0)
    return (heatmap, output)


path = "/content/img.jpg"
```

```
orig = cv2.imread(path)
resized = cv2.resize(orig, (256, 256))

image = tf.keras.preprocessing.image.load_img(path, target_size=(256, 256))
image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)

predictions = model.predict(image)
cam = GradCAM(model, np.argmax(predictions[0]), "expanded_conv_6/expand")
heatmap = cv2.resize(cam.compute_heatmap(image), (orig.shape[1], orig.shape[0]))

#heatmap = cv2.resize(heatmap, (orig.shape[1], orig.shape[0]))
(heatmap, output) = cam.overlay_heatmap(heatmap, orig, alpha=0.5)

cv2.rectangle(output, (0, 0), (340, 40), (0, 0, 0), -1)
cv2.putText(output, classes[np.argmax(predictions)], (10, 25),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

output = np.vstack([orig, heatmap, output])
output = imutils.resize(output, height=700)
cv2_imshow(output)
```

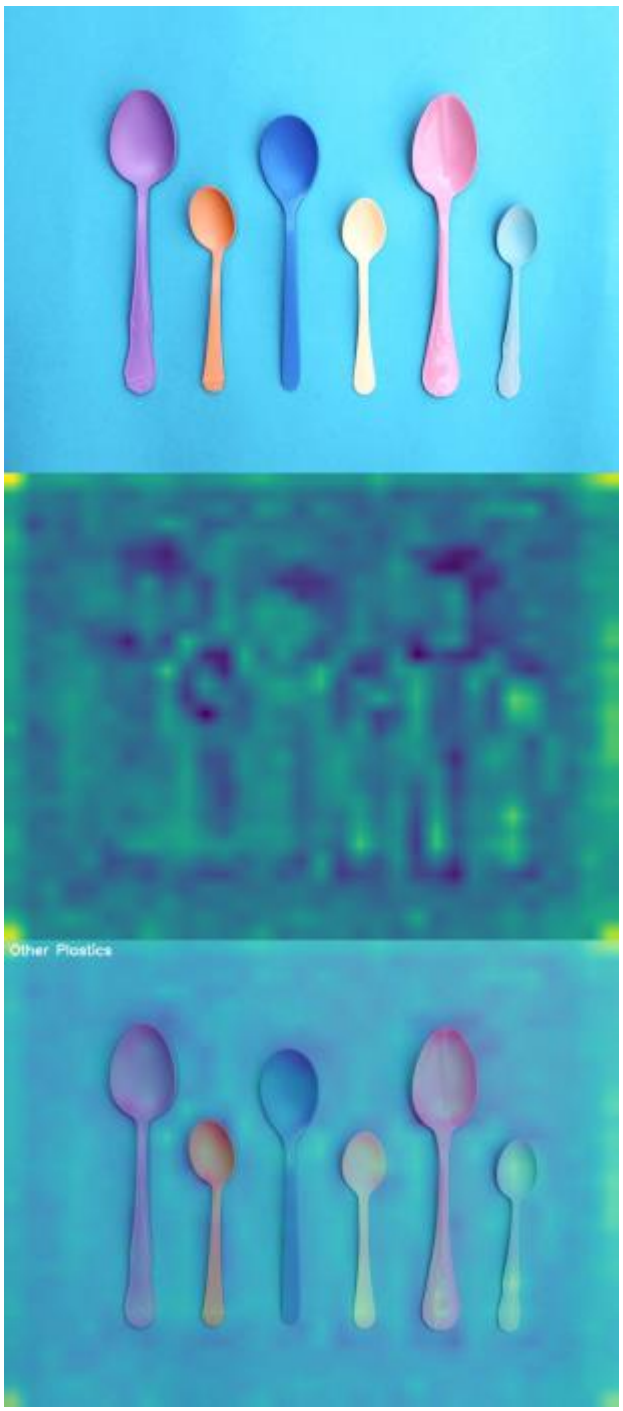

Рис. 3.26. Результат прогнозування

3.3. Висновки до розділу

Написавши код програми та побудувавши модель. Для того, щоб навчити модель з розміром набору даних приблизно 5000 зображень було витрачено близько семи хвилин навчального процесу. Було побудовано діаграми втрат і

точності. На діаграмі точності різниця становить не більше 2 %, від 98,75 % точності під час навчання до 96,45 % під час тестування, що суттєво не впливає на результати моделі. Було побудовано матрицю помилок, щоб оцінити з якими класами об’єктів моделі зручніше справлятися. Отримав прогнози для зображення з моделі. Використав клас *Grad-CAM*, що використовує градієнти будь-якого цільового поняття для створення карти локалізації, що виділяє важливі області зображення для прогнозування.

ВИСНОВКИ

У магістерській роботі розроблено програмне забезпечення для складового аналізу побутових відходів за допомогою бібліотеки *TensorFlow*. Метою даного проекту було проаналізівати основні методи розпізнавання об'єктів та розробити програмне забезпечення для класифікації різних типів відходів. В даному дипломному проекті створене програмне забезпечення для складового аналізу побутових відходів, що здатне розпізнати різні типи побутових відходів, такі як: алюміній, скло, пластик, дерево, картон, органіка і текстиль з точністю розпізнавання 96,45 %.

Побачивши можливості сучасних бібліотек, такі як *TensorFlow*, надають для побудови таких складних алгоритмів машинного навчання, зручно проаналізувати реальні переваги цієї моделі під час боротьби з екологічними загрозами. По-перше, це його вмісткість, яка займає лише 30 МБ дискового простору завдяки оптимізованій для мобільних пристроїв згортковій моделі, яку використовують під час виконання *Transfer Learning (MobileNetV3)*, що робить цю модель легкою для розгортання, що спонукає користувачів скористатися нею. Крім того, його результати на етапі оцінки та можливість узагальнення, незважаючи на невелике переоснащення, що спостерігається на кривих втрат і точності, роблять його достатньо надійним, щоб його можна було масштабувати до більшого набору даних.

Але щоб створити великий набір даних і масштабувати його з часом, потрібна потужна інфраструктура для зберігання даних. Крім того, співпраця багатьох людей ще більш необхідна для якісного збору та інтерпретації даних. Крім того, співпраця є однією з основних вимог для вдосконалення машинного навчання та технологій у цілому. Отже потрібно працювати разом, якщо хочемо мати збалансовану природу та світ, вільний від забруднення чи будь-яких небезпек.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Singh, S.; Mamatha, K.; Ragothaman, S.; Raj, K.D.; Anusha, N.; SusmiZacharia. Waste Segregation System Using Artificial Neural Networks. HELIX* 2017, 7, 2053–2058 с.

2. *Ranada, P. Why PH is the world's* 3*rd biggest dumper of plastics in the ocean. Rappler Blog. Retrieved October* 6, 2015, 2015.

3. *Sousa, J.; Rebelo, A.; Cardoso, J.S. Automation of Waste Sorting with Deep Learning*. 2019 *XVWorkshop de Visão Computacional (WVC). IEEE*, 2019, 43–48. с.

4. *Devi, R.S.; Vijaykumar, V.; Muthumeena, M. Waste Segregation using Deep Learning Algorithm*.

5. *Bircanog˘lu, C.; Atay, M.; Bes¸er, F.; Genç, Ö.; Kızrak, M.A. Recyclenet: Intelligent waste sorting using deep neural networks. 2018 Innovations in Intelligent Systems and Applications (INISTA). IEEE*, 2018, 1–7. с.

6. *Yang, M.; Thung, G. Classification of trash for recyclability status. CS229 Project Report* 2016, 2016.

7. *Lowe, D.G. Distinctive image features from scale-invariant keypoints. International journal of computer vision* 2004, 60, 91–110. с.

8. *Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. Communications of the ACM* 2017, 60, 84–90. с.

9. *Sakr, G.E.; Mokbel, M.; Darwich, A.; Khneisser, M.N.; Hadi, A. Comparing deep learning and support vector machines for autonomous waste sorting. 2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET). IEEE*, 2016, 207–212. с.

10. *Arebey, M.; Hannan, M.; Begum, R.; Basri, H. Solid waste bin level detection using gray level co-occurrence matrix feature extraction approach. Journal of environmental management* 2012, 104, 9–18. с.

11. *Costa, B.S.; Bernardes, A.C.; Pereira,J.V.; Zampa, V.H.; Pereira, V.A.; Matos, G.F.; Soares, E.A.; Soares, C.L.; Silva, A.F. Artificial intelligence in automated sorting in trash recycling. Anais do XV Encontro Nacional de Inteligência Artificial e Computacional. SBC*, 2018, 198-205. с.

12. *Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv*:1409.1556 2014.

13. *Donovan, J. Auto-trash sorts garbage automatically at the techcrunch disrupt hackathon. Techcrunch Disrupt Hackaton, San Francisco, CA, USA, Tech. Rep. Disrupt SF* 2016, 2016.

14. *Awe, O.; Mengistu, R.; Sreedhar, V. Smart trash net: Waste localization and classification. arXiv preprint* 2017.

15. *Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems,* 2015, 91–99. с.

16. *Dabholkar, A.; Muthiyan, B.; Srinivasan, S.; Ravi, S.; Jeon, H.; Gao, J. Smart illegal dumping detection. 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService). IEEE*, 2017, 255–260. с.

17. *Zhao, Z.Q.; Zheng, P.; Xu, S.t.; Wu, X. Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems* 2019, 30, 3212–3232. с.

18. *Çevik, E.; Zengin, K. Classification of Skin Lesions in Dermatoscopic Images with Deep Convolution Network. Avrupa Bilim ve Teknoloji Dergisi* 2019, 309–318. с.

19. *John, N.E.; Sreelakshmi, R.; Menon, S.R.; Santhosh, V. Artificial Neural Network based Intelligent Waste Segregator.*

20. *Albeahdili, H.M.; Han, T.; Islam, N.E. Hybrid algorithm for the optimization of training convolutional neural network. International Journal of Advanced Computer Science & Applications* 2015, 1, 79–85. с.

21. *Rosebrock, A. Histogram of Oriented Gradients and Object Detection. línea]. Disponible en:Blog*(http://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradientsobjectdetection) 2014.

22. *Girshick, R. Fast r-cnn. Proceedings of the IEEE international conference on computer vision*, 2015, 1440–1448. с.

23. *Bobulski, J.; Kubanek, M. Waste classification system using image processing and convolutional neural networks. International Work-Conference on Artificial Neural Networks. Springer*, 2019, 350–361. с.

24. Розширена класифікація відходів за допомогою машинного навчання // https://towardsdatascience.com/advanced-waste-classification-with-machine-learning-6445bff1304f

25. Персептрони в нейроних мережах // https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc