МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: «**Розроблення підходу до автоматизації розгортання веб-додатків**»

Виконав: здобувач 6 курсу групи Іт-62

Спеціальності 126 «Інформаційні системи та технології»

Фесюк Б. В.

Керівник: Чаплига В.М.

Рецензент:

ДУБЛЯНИ-2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Освітній ступінь «Магістр» за спеціальністю –
126 – «Інформаційні системи та технології»

“ЗАТВЕРДЖУЮ”
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
“____”_____2024_ р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту
Фесюка Богдана Васильовича

1. Тема роботи: **«Розроблення підходу до автоматизації розгортання веб-додатків».**

Керівник роботи Чаплига Вячеслав Михайлович, д.т.н., професор.

Затверджені наказом по університету від «12» ___09___ 2024 р. № __616/к-с _.

2. Строк подання студентом роботи: 15.12.2024 року.

3. Початкові дані до роботи: Нормативні документи та міжнародні стандарти щодо автоматизації розгортання веб-додатків, завдання на розробку підходу до автоматизації розгортання веб-додатків.

4. Зміст пояснювальної записки:

Вступ

Розділ 1 Аналіз впливу нейронних мереж на розробку сучасних веб сервісів з розпізнаванням зображень та їх автоматизацію.

Розділ 2 Дослідження підходів до розгортання веб-додатку з розпізнаванням зображень

Розділ 3 Розроблення підходу до автоматизації розгортання веб-додатків з використанням хмарних сервісів

Розділ 4 Охорона праці та безпека в надзвичайних ситуаціях

Розділ 5 Розрахунок економічної ефективності автоматизації розгортання та конфігурування веб-додатків

Висновки та пропозиції

Список використаної літератури

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 5	*Чаплига В.М., професор кафедри інформаційних технологій*		
4	*Городецький І.М., доцент кафедри фізики, інженерної механіки та безпеки виробництва*		

7. Дата видачі завдання «_____» _____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Приміт-ка
1	*Написання Вступу, першого розділу та означення головних завдань роботи*	03.09 - 10.09.24	
2	*Виконання другого розділу та формування початкових даних*	10.09 - 16.09.24	
3	*Виконання третього розділу та узагальнення отриманих результатів роботи*	17.09 - 23.09.24	
4.	*Написання розділу: «Охорона праці»*	23.09 - 31.09.24	
5	*Вартісна оцінка ефективності пропозицій роботи*	01.10 - 31.10.24	
6	*Завершення роботи*	01.11 - 30.11.24	
7	*Виправлення зауважень та перевірка на плагіат*	01.12 – 10.12.24	

Студент _____ Фесюк Б. В.

Керівник роботи _____ Чаплига В. М.

АНОТАЦІЯ

УДК 635.1

Розроблення підходу до автоматизації розгортання веб-додатків.

Фесюк Б. В. Кафедра ІТ – Дубляни, Львівський НУП, 2024.

Кваліфікаційна робота: 79 с. текст. част., 27 рис., 10 табл., 32 джерела.

Автоматизація розгортання веб-додатків стає все більш важливою через зростання складності сучасних програм, підвищення вимог до швидкості розробки, а також необхідність забезпечення стабільності та надійності в умовах конкурентного ринку. Ручне розгортання більше не відповідає сучасним вимогам через його тривалість, підвищений ризик помилок і недостатню масштабованість.

В роботі досліджуються та розробляються підходи до автоматизації технологічного процесу розгортання та управління конфігурацією веб додатку.

Об'єкт дослідження: процеси розгортання веб-додатків в умовах підвищених вимог до швидкості розробки сучасного програмного забезпечення.

Предмет дослідження: підходи до до автоматизації розгортання веб-додатків на основі використання хмарних сервісів.

Мета дослідження: дослідження та розробка підходів до автоматизації розгортання веб-додатків з розпізнаванням зображень на основі нейронних мереж та використання хмарних сервісів.

Наукова новизна роботи полягає.

Практичне значення одержаних у кваліфікаційній роботі результатів полягає у можливості їх використання для автоматизації і, відповідно, пришвидшення та покращення якості процесів розгортання веб-додатків з обробкою зображень.

Апробація результатів роботи. Основні теоретичні та практичні результати виконаної магістерської кваліфікаційної роботи доповідались та отримали схвалення на наукових семінарах кафедри ІТ, а також на Міжнародному студентському науковому форумі у жовтні 2024 року, ЛНУП, Дубляни.

Публікації здобувача за темою кваліфікаційної роботи.

Фесюк Б. В. Сучасні підходи до автоматизації розгортання веб-додатків. Студентська молодь і науковий прогрес: тези доп. Міжнар. студ. наук. форуму, 02–04 жовт. 2024 р. [Електронний ресурс]. Львів, 2024. С. 336.

Структура та обсяг кваліфікаційної роботи. Кваліфікаційна робота містить вступ, п’ять розділів, висновки та пропозиції, список використаної літератури та додатки.

Ключові слова: веб-додаток, розгортання, автоматизація, інформаційна технологія, нейромережа, зображення, машинне навчання, хмарний сервіс.

SUMMARY

UDC 635.1

Development of an approach to automating web application deployment.

Fesyuk B. V. IT Department - Dublyany, Lviv National University of Applied Sciences, 2024.

Qualification work: 79 p. text. part, 27 fig., 10 tab., 32 sources.

Automation of web application deployment is becoming increasingly important due to the increasing complexity of modern programs, increasing requirements for development speed, as well as the need to ensure stability and reliability in a competitive market. Manual deployment no longer meets modern requirements due to its duration, increased risk of errors and insufficient scalability. Therefore, automation of web application deployment is an integral part of modern software development. It provides process acceleration, stability, reliability, error reduction and improved productivity of development teams, which makes it a key element for the successful functioning and development of digital products.

The work investigates and develops approaches to automating the technological process of deploying and managing the configuration of a web application.

Object of research: web application deployment processes in conditions of increased requirements for the speed of development of modern software.

Subject of research: approaches to automating the deployment of web applications based on the use of cloud services.

Purpose of research: research and development of approaches to automating the deployment of web applications with image recognition based on neural networks and cloud services.

The scientific novelty of the work lies in the further development of methods for deploying web applications with image recognition and classification based on neural networks and cloud services.

The practical significance of the results obtained in the qualification work lies in the possibility of their use for automation and, accordingly, acceleration and improvement of the quality of the processes of deploying web applications with image processing.

Approbation of the results of the work. The main theoretical and practical results of the completed master's qualification work were reported and approved at the scientific seminars of the IT Department, as well as at the International Student Scientific Forum in October 2024, LNUP, Dublyany.

Publications of the applicant on the topic of the qualification work.

Fesyuk B. V. Modern approaches to the automation of web application deployment. Student youth and scientific progress: abstracts of the International Student Scientific Forum, 02–04 Oct. 2024 [Electronic resource]. Lviv, 2024. P. 336.

Structure and scope of the qualification work. The qualification work contains an introduction, five chapters, conclusions and proposals, a list of used literature and appendices.

Keywords: web application, deployment, automation, information technology, neural network, image, machine learning, cloud service.

ЗМІСТ

ВСТУП

Актуальність. Сучасні веб-додатки часто мають складну архітектуру, яка включає мікросервіси, бази даних, кешування, зовнішні API тощо. Ручне розгортання таких систем може займати багато часу. Автоматизація дозволяє значно зменшити час від моменту завершення розробки до розгортання на сервер, що є ключовим для компаній, які працюють за методологіями Agile або DevOps.

Автоматизоване розгортання є ключовою частиною процесів CI/CD (Continuous Integration/Continuous Deployment). Завдяки автоматизації веб-додатки можуть бути протестовані, побудовані та розгорнуті автоматично після внесення змін у код, що скорочує цикл розробки і підвищує якість продукту. Тому автоматизація розгортання веб-додатків є актуальною і невід'ємною частиною сучасної розробки програмного забезпечення. Вона забезпечує прискорення процесів, стабільність, надійність, зменшення помилок та покращення продуктивності команд розробників, що робить її ключовим елементом для успішного функціонування та розвитку цифрових продуктів.

Об'єкт дослідження: процеси розгортання веб-додатків в умовах підвищених вимог до швидкості розробки сучасного програмного забезпечення.

Предмет дослідження: підходи до до автоматизації розгортання веб-додатків на основі використання хмарних сервісів.

Мета дослідження: дослідження та розробка підходів до автоматизації розгортання веб-додатків з розпізнаванням зображень на основі нейронних мереж та хмарних сервісів.

Для досягнення визначеної мети були поставлені та вирішені наступні завдання:

- проаналізовано вплив нейронних мереж на розробку сучасних веб сервісів з розпізнаванням і класифікацією зображень та на їх автоматизацію;

- проаналізовано типіи нейронних мереж та архітектури нейронної мережі додатку з розпізнаванням і класифікацією зображень;
- досліджени підходіи до розгортання веб-додатку з розпізнаванням зображень;
- розроблено підхід до автоматизації розгортання веб-додатку з розпізнаванням зображень
- здійснено аналіз результатів автоматизації розгортання веб-додатку з розпізнаванням зображень;
- розглянуто питання охорони праці та безпеки в надзвичайних ситуаціях;
- проведено розрахунок економічної ефективності автоматизованого розгортання веб-додатку.

Наукова новизна роботи полягає в подальшому розвитку методів розгортання веб-додатків з розпізнаванням та класифікацією зображень на основі нейронних мереж та з використанням хмарних сервісів.

Практичне значення одержаних у кваліфікаційній роботі результатів полягає у можливості їх використання для автоматизації і, відповідно, пришвидшенні та покращенні якості процесів розгортання веб-додатків з обробкою зображень.

Апробація результатів роботи. Основні теоретичні та практичні результати виконаної магістерської кваліфікаційної роботи доповідались та отримали схвалення на наукових семінарах кафедри ІТ, а також на Міжнародному студентському науковому форумі у жовтні 2024 року, ЛНУП, Дубляни.

Публікації здобувача за темою кваліфікаційної роботи.

Фесюк Б. В. Сучасні підходи до автоматизації розгортання веб-додатків. Студентська молодь і науковий прогрес: тези доп. Міжнар. студ. наук. форуму, 02–04 жовт. 2024 р. [Електронний ресурс]. Львів, 2024. С. 336.

Структура та обсяг кваліфікаційної роботи. Кваліфікаційна робота містить вступ, п’ять розділів, висновки та пропозиції, список використаної літератури та додатки.

РОЗДІЛ 1. АНАЛІЗ ВПЛИВУ НЕЙРОННИХ МЕРЕЖ НА РОЗРОБКУ СУЧАСНИХ ВЕБ СЕРВІСІВ З РОЗПІЗНАВАННЯМ ЗОБРАЖЕНЬ ТА ЇХ АВТОМАТИЗАЦІЮ

1.1. Аналіз розвитку нейронних мереж та їх використання у веб розробці

В сучасних веб додатках взаємодія з користувачем є основним елементом, на який звертають увагу розробники. Задля її покращення потрібно постійно змінювати додаток, що веде за собою збільшення складності самого продукту. Штучний інтелект (AI) відіграє все більш важливу роль у цьому процесі, оскільки він дозволяє розробникам створювати програми, які можуть навчатися, адаптуватися та приймати рішення самостійно.

Тому надзвичайно важливим є впровадження функціональності, яку може дати розвиток штучного інтелекту в розробку веб додатків. Одним із таких механізмів є нейронні мережі, які по своїй суті імітують роботу людського мозку.

Нейронні мережі — це в основному масивні паралельні моделі обчислення, що імітують роботу людського мозку. Нейронна мережа складається з великої кількості простих процесорів, пов'язаних між собою через зважені зв'язки. За аналогією, вузли обробки можна назвати «нейронами». Вихід кожного вузла залежить лише від інформації, яка доступна локально на вузлі та зберігаються всередині або надходять через зважені зв'язки. Кожен блок отримує вхідні дані від багатьох інших вузлів передачі його вихід на ще один вузол.[3]

Нейронні мережі в своїй основі спираються на навчальні дані, щоб навчатися та підвищувати свою точність з часом. Однак, коли ці алгоритми навчання налаштовані на точність, вони стають потужними інструментами в штучному інтелекті, що дозволяє класифікувати та кластеризувати дані з високою швидкістю. Завдання з розпізнавання мовлення або розпізнавання зображень можуть тривати хвилини чи години порівняно з ручною ідентифікацією експертів-людей. Однією з найвідоміших нейронних мереж є пошуковий алгоритм Google. [4]

Якщо говорити про сучасні веб додатки, то сьогодні в процесі розробки дедалі частіше застосовуються нейронні мережі. Така колаборація веб сервісів та нейронних мереж в розробці додатків має велику кількість переваг для бізнесу, а саме:

- Зменшення використання ресурсів (грошей та часу).

- Підвищення якості та точності результатів обчислень за рахунок самостійного збору і аналізу нейронніими мережами необхідних даних та наданні якісних і точних результатів.

- Підвищення ефективності бізнес операцій за рахунок виконання ШІ рутинної роботи розробників.

- Зростання безпеки продукту, що використовується, за рахунок машинного навчання, яке допомагає швидко виявляти потенційні випадки шахрайства.

- Підвищення якості класифікація клієнтів, наприклад, в веб магазинах, де клієнти обслуговуються онлайн.

- Структурований підхід розробки, коли на початковому етапі команда розробників вибирає та структурує підхід до доступу та управління даними, оскільки будь-яка організація використовує декілька різних джерел та типів інформації, які забезпечують її роботу.

- Впровадження розділеного тестування (split testing) для досягнення найкращого результату.

- Швидкий аналіз даних та детальна звітність

Як видно, застосування нейронних мереж в веб розробці суттєво покращує ефективність всього продукту. Проте варто також зазначити, що окрім нейронних мереж у веб розробці застосовуються та інші галузі штучного інтелекту такі як: машинне навчання, глибинне навчання тому важливо розуміти різницю кожної із технологій, щоб обрати правильний підхід для власного продукту.

Штучний інтелект, машинне навчання, глибинне навчання та нейронні мережі можна розглядати як низку систем ШІ від найбільшої до найменшої, кожна

з яких охоплює наступну.

1.2. Аналіз типів нейронних мереж та їх пристосованості до розгортання додатків

Нейронні мережі також не є однорідними і є велика кількість різних мереж, які виконують різноманітні функції. Проте усі нейронні мережі можна умовно поділити на шість типів, кожен з яких має свої переваги та недоліки

- Штучні нейронні мережі (ANN)
- Згорткові нейронні мережі (CNN)
- Повторювані (рекурентні) нейронні мережі (RNN)
- Перцептрон
- Мережі довготривалої короткочасної пам'яті
- Радіальна базова функціональна нейронна мережа

Перцептрон є фундаментальним типом нейронної мережі, який використовується для завдань бінарної класифікації. Він складається з одного шару штучних нейронів (також відомих як перцептрони), які приймають вхідні значення, застосовують ваги та генерують вихідні дані. Перцептрон найчастіше використовують для лінійно роздільних даних, де він навчається класифікувати вхідні дані на дві категорії на основі межі рішення [9]. Перцептрон використовується у розпізнаванні образів, класифікації зображень і лінійній регресії. Однак перцептрон має обмеження в обробці складних даних, які не є лінійно роздільними.Також персептрони можна використовувати для розв'язання задач лінійної регресії, де метою є прогнозування безперервного результату на основі вхідних характеристик. Основними недоліками перцетронів є як вже зазначалось вижче це їх обмеження у обробці лише лінійних роздільних данних та брак глибини навчання через те, що перцептрони є однорівневими і не можуть вивчати складні ієрархічні представлення

Штучні нейронні мережі (ANN) це група з кількох перцептронів/нейронів на

кожному із шарів. ANN також відома як нейронна мережа прямого зв'язку, оскільки вхідні дані обробляються лише в прямому напрямку. Штучна нейронна мережа складається з 3 шарів: вхідний, прихований і вихідний рівні. Вхідний рівень приймає вхідні дані, прихований рівень обробляє вхідні дані, а вихідний рівень створює результат. По суті, кожен шар намагається вивчити певні ваги. Зазвичай штучні нейронні мережі використовують для роботи із табличними даними, текстом та зображенням. Основною перевагою використання штучних нейронних мереж є їх здатність вивчати будь-яку нелінійну функцію це допомагає мережі вивчати будь- який складний зв'язок між входом і виходом. Основні недоліки цієї нейронної мережі зустрічаються в роботі із зображеннями, річ у тім, що під час вирішення проблеми класифікації зображень за допомогою мережі першим кроком є перетворення 2- вимірного зображення в 1-вимірний вектор перед навчанням моделі. Це має два недоліки: кількість параметрів, які можна навчити, різко зростає зі збільшенням розміру зображення, а також штучна нейронна мережа втрачає просторові характеристики зображення, які стосуються розміщення пікселів у зображенні. Також ще одним із важливих недоліків штучної нейронної мережі є її неможливість збирати послідовну інформацію як вхідні дані, що є необхідною умовою для роботи із послідовними даними [9].

Різницю між рекурентною нейронною мережею та штучною нейронною мережею можна зобразити наступним чином (див. рис.1.1).

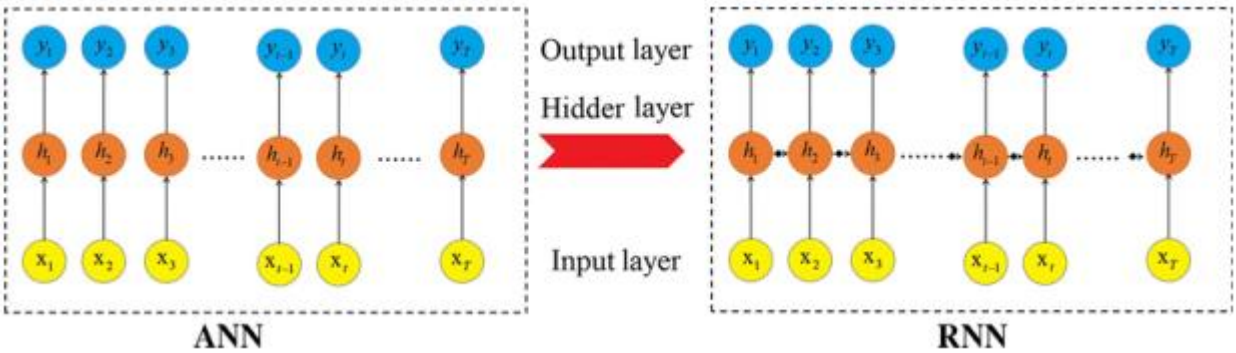


Рис.1.1 - Різниця між рекурентною та штучною нейронними мережами [10].

Згорткові нейронні мережі (CNN) зараз є найбільш популярними в спільноті глибокого навчання. Ці моделі CNN використовуються в різних програмах і доменах, і вони особливо поширені в проектах обробки зображень і відео.

Будівельними блоками CNN є фільтри, вони ж ядра. Ядра використовуються для отримання відповідних функцій із вхідних даних за допомогою операції згортки.

1.3. **Аналіз архітектури нейронної мережі додатку з розпізнаванням та класифікацією зображень**Додаток, автоматизоване розгортання якого буде представлене нижче в роботі, побудований на основі моделі, яка використовує згорткову нейронну мережу (convolutional neural network, CNN, ConvNet). Тому для розуміння функціонування додатку слід розуміти, як взагалі працюють згорткові нейронні мережі.

Куніхіко Фукусіма та Янн ЛеКун заклали основу дослідження згорткових нейронних мереж у своїй роботі в 1980 році «Зворотне поширення, застосоване до розпізнавання рукописного поштового індексу» в 1989 році відповідно. Ще більш відомим є те, що Янн Ле Кун успішно застосував зворотне поширення, щоб навчити нейронні мережі ідентифікувати та розпізнавати шаблони в серії рукописних поштових індексів. Він продовжував свої дослідження зі своєю командою протягом 1990-х років, завершившись проектом «LeNet-5», який застосовував ті самі принципи попередніх досліджень для розпізнавання документів [19]. З того часу з'явилася низка варіантів архітектури згорткових мереж CNN із впровадженням нових наборів даних, таких як MNIST і CIFAR-10. Деякі з цих інших архітектур можна відзначити: AlexNet; VGGNet; GoogLeNet; ResNet; ZFNet

Однак LeNet-5 відома як класична архітектура CNN. LeNet-5 - це одна з перших згорткових нейронних мереж, що складається з двох згорткових шарів для виділення ознак та трьох повністю пов'язаних рівнів для класифікації [12].

Архітектура LeNet-5 була представлена у 1998 році в дослідницькій статті під

назвою «Навчання на основі градієнтів, застосоване до розпізнавання документів» Янна Лекуна, Леона Ботту, Йошуа Бенгіо та Патріка Хаффнера. Це одна з найперших і найпростіших архітектур CNN.

Архітектуру LeNet-5 має наступні характеристики:

Складається з 7 шарів. Перший шар складається з вхідного зображення розміром 32x32. Він складається з 6 фільтрів розміром 5 x 5, що призводить до розміру 28x28x6. Другий рівень — це операція об'єднання, яка фільтрує розмір 2x2 і крок 2. Отже, розмір результуючого зображення становитиме 14x14x6.

Подібним чином третій шар також включає в себе операцію згортання з 16 фільтрами розміром 5x5, за якими слідує четвертий шар об'єднання з аналогічним розміром фільтра 2x2 і кроком 2. Таким чином, розмір результуючого зображення буде зменшено до 5x5x16.

Після того, як розмір зображення зменшено, п'ятий шар є повністю пов'язаним згортковим шаром із 120 фільтрами розміром 5x5 кожен. У цьому шарі кожна з 120 одиниць цього шару буде з'єднана з 400 (5x5x16) одиницями з попередніх шарів. Шостий шар також є повністю пов'язаним шаром із 84 одиницями.

Останній сьомий рівень буде вихідним рівнем softmax із «n» можливими класами залежно від кількості класів у наборі даних [13]

Умовно всі ці рівні можна зобразити наступною схемою.

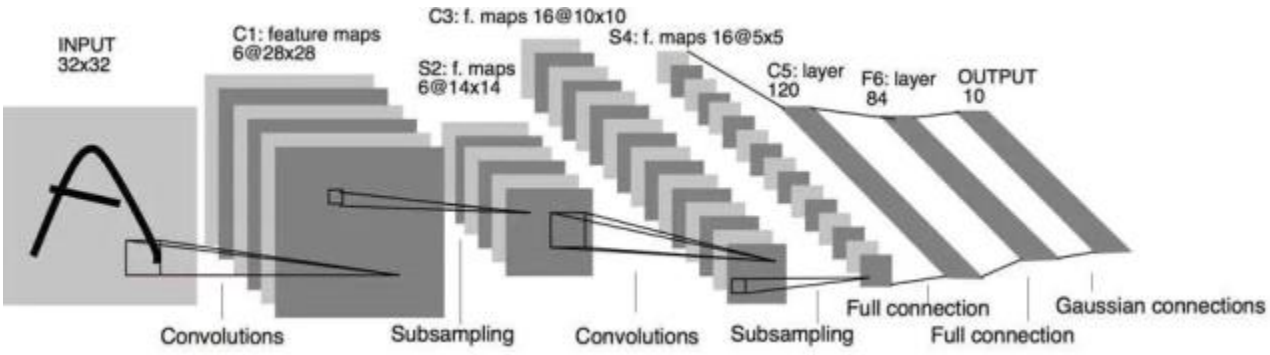


Рис.1.2 - Рівні LeNet-5 архітектури [14]

Згорткова нейронна мережа (CNN) — це алгоритм глибокого навчання, який може сприймати вхідне зображення, призначати важливість (вагові значення та зміщення) різним об'єктам на зображенні та мати можливість відрізнити одне зображення від іншого. Попередня обробка, яка є необхідною в CNN, набагато менша порівняно з іншими алгоритмами класифікації. У той час як у примітивних методах фільтри (характеристики) розробляються вручну, після достатнього навчання, CNN мають можливість одразу вивчати ці характеристики.

Архітектура згорткових моделей аналогічна структурі підключення нейронів у людському мозку та була натхненна організацією зорової кори. Окремі нейрони реагують на стимули лише в обмеженій області поля зору, відомої як рецептивне поле. Набір таких полів перекривається, щоб охопити всю візуальну область [15].

У згортковій нейронній мережі (CNN) є дві основні частини, а саме,

- Згортковий інструмент, який розділяє та ідентифікує різні особливості зображення для аналізу у процесі, відомому як вилучення ознак (FC). Мережа вилучення ознак складається з багатьох пар згорткових або пулінгових шарів.

- Повністю зв'язаний шар, який використовує вихід від процесу згортки та передбачає клас зображення на основі ознак, вилучених на попередніх етапах.

Модель вилучення ознак (FC) у CNN спрямована на зменшення кількості ознак у наборі даних. Кілька шарів CNN показано на діаграмі архітектури CNN.

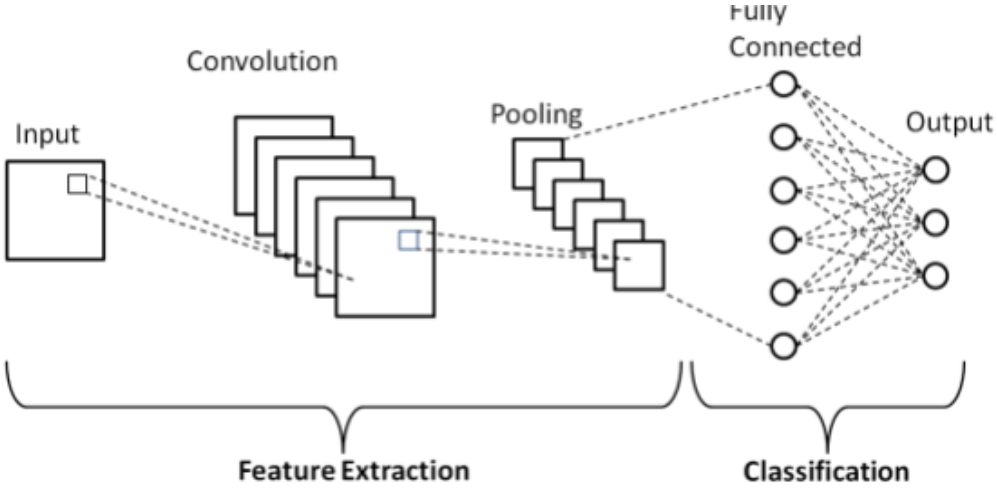


Рис.1.3 - Архітектура шарів CNN [16].

Шар об'єднання, коли подібно до згорткового рівня, рівень об'єднання також розгортає ядро або фільтр по вхідному зображенню. Але на відміну від згорткового рівня, рівень об'єднання зменшує кількість параметрів у вхідних даних і також призводить до втрати деякої інформації. Позитивним є те, що цей рівень зменшує складність і підвищує ефективність CNN.

Повністю зв'язаний шар, рівень FC - це місце, де відбувається класифікація зображень у CNN на основі ознак, витягнутих із попередніх шарів. Тут повне підключення означає, що всі входи або вузли з одного рівня підключені до кожного блоку активації або вузла наступного рівня [17].

Зазвичай, коли всі функції приєднуються до повністю зв'язаного рівня (FC), це може призвести до перенавчання на навчальному наборі даних. Перенавчання стає проблемою, коли модель настільки ефективно пристосовується до навчальних даних, що це негативно впливає на її продуктивність при використанні нових даних.

Для вирішення цієї проблеми використовується рівень відкидання, в якому під час процесу навчання випадковим чином відкидається частина нейронів з нейронної мережі, що призводить до зменшення розміру моделі. Під час відкидання 30% вузлів випадковим чином вилучаються з нейронної мережі.

Відкидання сприяє підвищенню продуктивності моделі машинного навчання, оскільки воно запобігає перенавчанню та спрощує структуру мережі, вилучаючи нейрони під час навчання.

Один із ключових параметрів у моделі CNN - це функція активації. Вони використовуються для вивчення та апроксимації різних неперервних і складних зв'язків мережі. Простими словами, це вирішує, яка інформація повинна активуватися в напрямку від входу до виходу мережі.

Це додає нелінійності до мережі. Існують різні популярні функції активації, такі як ReLU, Softmax, tanH і Sigmoid. Кожна з цих функцій має своє використання. Для бінарної класифікаційної моделі CNN функції sigmoid і softmax є

оптимальними, а для багатокласової класифікації зазвичай використовується softmax. Іншими словами, функції активації визначають, чи має нейрон бути активованим чи ні. Вони вирішують, чи вхідні дані є важливими для прогнозування за допомогою математичних операцій [18].

Усі рівні в CNN пов'язані не повністю, тому що це призведе до надмірно щільної мережі. Це також призвело б до збільшення втрат і вплинуло б на якість виведення, і це було б дорого з точки зору обчислень.

У двовимірній згортці є ядро або фільтр, відомий також як карта функцій. Це ядро проковзує по вхідній матриці зображення і виконується операція поелементного множення, перемножаючи кожен елемент ядра на відповідний елемент вхідної матриці зображення. Після цього ми сумуємо ці значення, отримуючи один вихідний піксель.

Цей процес повторюється, переміщаючи ядро з одного положення в інше, виконуючи операцію поелементного множення та сумування результатів до одного вихідного пікселя. Після того, як ми проковзали ядро по всьому зображенню, ми отримуємо нове двовимірне зображення з певним набором елементів, які виявило дане ядро. Ці елементи представляють собою результат зваженої суми після поелементного множення вхідних функцій, які розташовані приблизно в тому ж місці, що й вихідний піксель на вхідному шарі. Розмір ядра визначає, скільки функцій об'єднується в одну вихідну функцію, отже, розмір ядра визначає точне розташування функцій на вихідному рівні.

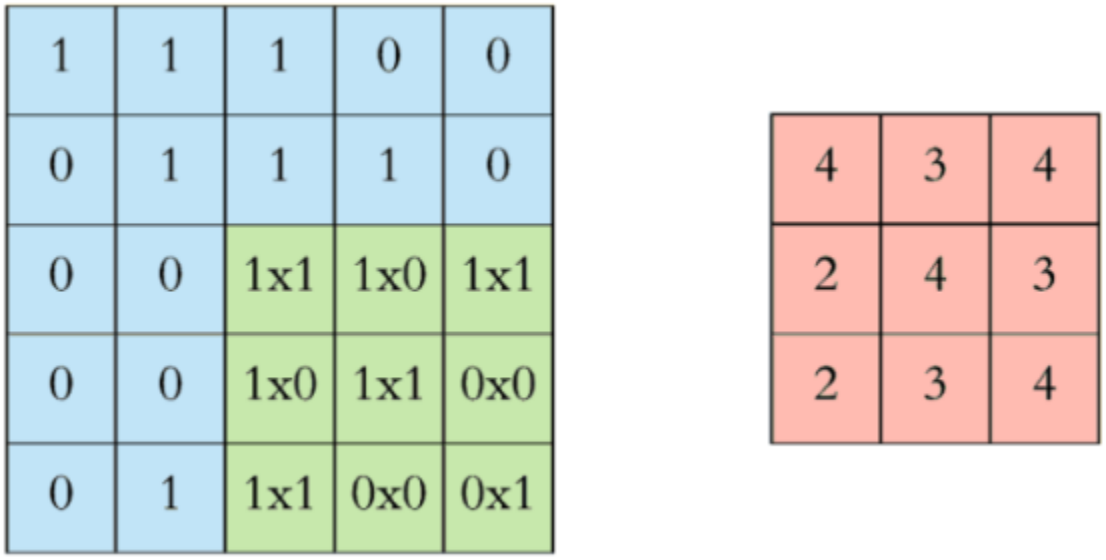


Рис.1.4 - Принцип роботи згортки [19]

На основі рисунку 1.4 є зображення розміром 5x5, що складається з 25 пікселів. Після застосування згортки отримано вихідний об'єкт розміром 3x3, що складається з 9 елементів. Таким чином, вихідна функція становить 3x3 відносно вхідної функції розміром 5x5.

Якщо уявити, що використовується стандартна повнозв'язана мережа з 10 нейронами для першого шару, кількість параметрів для обчислення становитиме 5x5x3x10 = 750 ваг, плюс 10 зміщень, загалом 760 параметрів лише для першого шару. Це значна кількість обчислень, особливо для зображення розміром 5x5 пікселів.

Розглядаючи випадок згортки, кількість параметрів становитиме 3x3x3x10 = 270 ваг та 10 зміщень, загалом 280 параметрів для обчислення. Це майже втричі менше, ніж у випадку стандартного повнозв'язаного шару, як у традиційній нейронній мережі. Тобто можна побачити ефективність використання згортки.

Основними термінами, які використовуються в згорткових нейронних мережах є наступні.

Padding (Заповнення). У деяких випадках під час ковзання ядра над вхідною матрицею деякі пікселі навколо краю обрізаються, оскільки деякі (ядро перевищує

межу, де знаходяться пікселі вхідної матриці, вздовж краю). Наприклад, є вхідна матриця 5X5 і розмір ядра 3X3 . Щоб вирішити цю проблему, нам потрібно додати кілька пікселів до вхідної матриці вздовж краю, щоб уникнути перевищення ядра. Цей прийом ми називаємо паддінгом.

Додавання цих фальшивих додаткових пікселів до краю вирішує проблему, ці пікселі зазвичай мають значення 0. Також важливо, що заповнюється все зображення, а не лише певна кількість сторін

Виконується паддінг, щоб вирішити наступне:

- Проблему скорочення вихідних даних, коли вихідні дані зменшуються під час проходження кожного фільтра, і ми втрачаємо багато даних

- Розкидання даних по краях [20].

Дійсні та однакові згортки. Дійсна згортка - це тип згортки, у якому не застосовується жодних відступів до вихідного зображення. Це може призвести до втрати даних по краях у випадках, коли фільтр не ідеально підходить до оригінального зображення.

Однакова згортка и передбачає застосування фільтрів, щоб розмір вхідних даних був таким самим, як розмір вихідних даних.

Крокування (Striding) - це кількість одиниць даних, які пропускає фільтр між кожною операцією. Пересуваючи ядро над вхідною матрицею можена вирішити, скільки пропускати данних, стрибати чи стрибати зліва направо, а також вгору та вниз під час ковзання. По суті, це означає, скільки стовпців ми хочемо пропустити під час ковзання ядра над вхідною матрицею.

У стандартному шарі згортки крок дорівнює 1, в основному включається кожен слайд або стовпець вхідної матриці. Крок 2 означає пропуск двох стовпців вхідної матриці після кожної операції згортки. Вибраний крок впливає на кінцевий розмір вихідної матриці. Крок 2 зменшує вихід приблизно в 2 рази, крок 3 зменшує вихід приблизно в 3 рази і так далі.

Мультиканальність і розмах (Dimension). Звичайне кольорове зображення

має кілька кольорових шарів, які називаються каналами. Існує три основні кольорові канали: червоний, зелений і синій канали «RGB». Збільшення кількості каналів збільшує розмір матриці вхідного шару в повністю зв'язаному шарі. Для прикладу візьмемо піксель розміром 6X6 з одним кольоровим каналом (шкала сірого), розміри зображення становитимуть 6X6 = 36, якщо це звичайне зображення RGB, розміри становитимуть 6x6x3 = 108.

Є деяка відмінність між поняттями фільтра та ядра. Фільтр — це набір ядер, кожне з яких є унікальним і працює з одним каналом вхідного рівня. Для кожного каналу на вхідному рівні є ядро, сукупність цих каналів створює фільтр.

Кожен фільтр створює один і тільки один вихідний канал. Принцип роботи полягає в тому, що під час ковзання фільтра по вхідному шару ми одночасно пересуваємо його ядра по всіх каналах, але використовуючи різні ядра для кожного шару.

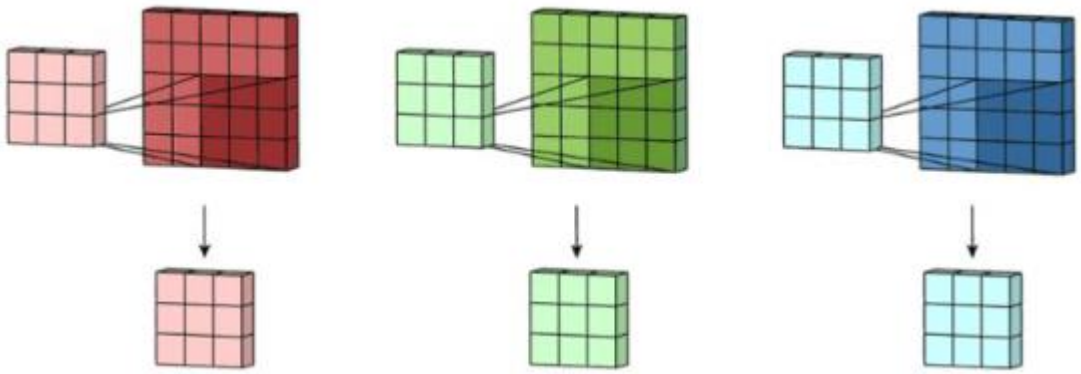


Рис.1.5 - Робота фільтра 3x3 з зображенням RGB

Ядра не мають однакової ваги, деякі ядра мають більшу вагу порівняно з іншими. Наприклад, червоний канал ядра з більшою вагою, ніж інші, більше реагує на відмінності в функціях червоного каналу, ніж інші.

Ми отримуємо один вихідний канал, підсумовуючи ваги ядер у кожному каналі. Потім додається зміщення, щоб отримати остаточне значення, до кожного

фільтра прикріплено одне зміщення, до нього застосовується нелінійність (наприклад, Relu). Цей процес повторюється для всіх інших фільтрів, і результати фільтрів об'єднуються разом, щоб отримати кінцевий результат, причому глибина (канали) кінцевого результату є загальною кількістю використаних фільтрів.

Об'єднання (Pooling). Max Pooling – це після того, як ми передаємо вихідний рівень згортки, кидаємо функцію нелінійності, є ще один шар, який називається шаром максимального об'єднання, цей шар просто зменшує розмірність зображення, зменшуючи кількість пікселів у виході з попереднього шару.

Максимальний шар об'єднання додається після шару згортки. У шарі максимального об'єднання ми визначаємо фільтр n X n і крок, ми беремо максимальне значення пікселів, охоплених розміром шару максимального об'єднання.

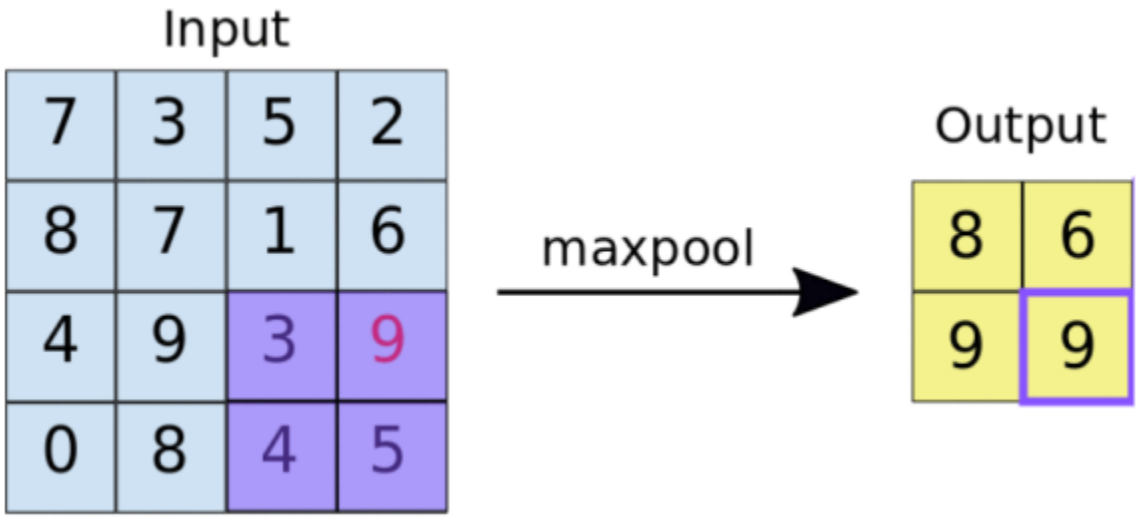


Рис.1.6. Максимальне об’єднання

Максимальне об’єднання використовується щоб зменшити обчислювальне навантаження.

Зменшення розміру зображення, яке буде використовуватися в повністю підключеному шарі, означає, що ми маємо менше параметрів, отже, менше обчислень. Зменшує перенавчання. Логіка, яка лежить в основі максимального

об’єднання, полягає в тому, що пікселі з найвищими значеннями є пікселями з найвидатнішими характеристиками, тому ми хочемо передати їх на наступний рівень конверсії або на повністю підключені рівні. Це допомагає моделі вивчити найвидатніші риси, отже, уникнути переобладнання.

Average pooling (середнє об'єднання) - це інша форма об'єднання, у якій береться саме середнє значення всіх значень пікселів у зоні, охопленій вибраним фільтром об'єднання, а не максимальне з них.

Результат максимального об'єднання або середнього об'єднання передається до наступної мережі або до повністю підключених рівнів.

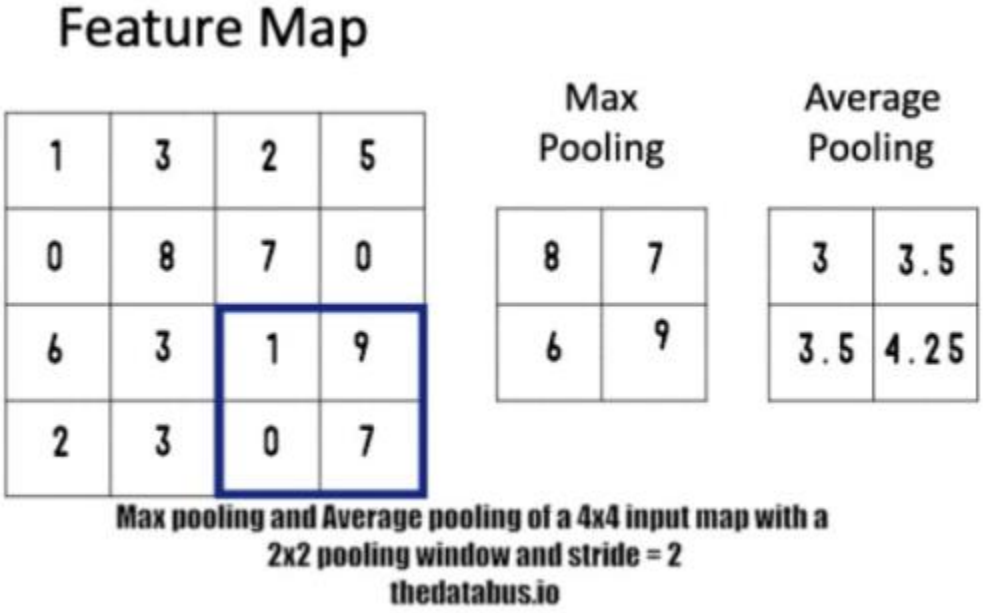


Рис.1.7 - Порівняння Максимального та середнього об’єднання

Для навчання моделі через використання згорткової нейронної мережі потрібно мати набір даних, який допоможе натренувати саму модель. Для цього завдання, наприклад, буде використовуватись датасет Cifar-IO. Набір даних CIFAR-10 складається з 60000 кольорових зображень 32x32 у 10 класах: літак, автомобіль, птах, кіт, олень, собака, жаба, кінь, корабель, вантажівка, по 6000 зображень на клас. Є 50000 навчальних і 10000 тестових зображень [19].

Набір даних розділено на п'ять навчальних партій і одну тестову, кожна з яких містить 10 000 зображень [21].

Для створення моделі нейронної мережі буде використовуватись фреймворк Keras поверх бібліотеки Tensorflow від Google. Основною структурою даних у Keras є модель, і основні два типи моделей у Keras - це Функціональна Модель API та Послідовна Модель.

Послідовна Модель - це модель із лінійним стеком шарів, яку дуже просто описати. У послідовній моделі два щільні шари визначаються моделлю. Це робить послідовну модель менш складною з точки зору кодування. Для визначення кожного шару достатньо лише одного рядка коду, такого як передбачення тренованої моделі, оцінка та розрахунок метрик і втрат, тренування та встановлення процесу навчання. Послідовна модель Keras проста у використанні, але вона обмежена тільки топологією моделі.

Основними перевагами імплементації Keras в продукті є:

- Легкість створення моделей та навчання.

- Легка інтеграція з основними бібліотеками для побудови нейронних мереж і може інтегруватися щонайменше з п'ятьма основними бібліотеками, такими як PlaidML, MXNet, Theano, CNTK і TensorFlow.

- Більша гнучкість тому, що також легко інтегрується з мовами глибокого навчання нижчого рівня, що дозволяє розробнику швидко реалізувати будь- що, що він побудував мовою базового рівня.

- Легке перетворення моделей у продукти і розробник може швидко перетворити свої моделі у продукти, оскільки Keras підтримує більший спектр платформ, ніж будь-які інші фреймворки глибокого навчання, включаючи Google Cloud. Це досягається за допомогою TensorFlow-Serving, у браузері через прискорені GPU JavaScript середовища виконання [22].

Тренування моделі програмно відбувається наступним чином.

Спочатку потрібно імортувати основні бібліотеки та завантажити датасет

Cifar- 10

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img , img_to_array
from tensorflow.keras.layers import Dense , Dropout , Conv2D , MaxPooling2D, Flatten , BatchNormalization


(x_train , y_train) , (x_test , y_test) = tf.keras.datasets.cifar10.load_data()

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 11s 0us/step
```

Рис.1.8 - Імпортування бібліотек та датасету

Cifar-10 містить 50000 зображень для тренування та 10 тисяч для тестування. Для того, щоб пришвидшити обчислення треба нормалізувати числа в матриці в діапазоні від 0 до 1, для цього слід поділити число на 255 через те, що використовується зображення RGB. Нормалізація зображення відбувається нас тупним чином.

```
def normalize(x):
    x = x.astype('float32')
    x = x/255.0
    return x

x_train = normalize(x_train)
x_test = normalize(x_test)
x_val = normalize(x_val)

y_train = tf.keras.utils.to_categorical(y_train , 10)
y_test = tf.keras.utils.to_categorical(y_test , 10)
y_val  = tf.keras.utils.to_categorical(y_val , 10)

datagen.fit(x_train)
```

Рис.1.9 - Нормалізація зображення для навчання моделі

Створення моделі є найвідповідальнішим процесом в побудові самої моделі, тому слід проаналізувати кожен із його елементів,

Рис.1.10 - Процес створення моделі.

де:

- Conv2D: Це шар згорткової мережі, який виконує згортку на вхідному зображенні.

- 64: Кількість фільтрів (ядер) у шарі, що визначає кількість вихідних каналів.

- (3, 3): Розмір ядра згортки, у цьому випадку 3x3 пікселі.

- activation='relu': Функція активації ReLU (Rectified Linear Unit), яка застосовується після виконання згортки для введення нелінійності.

- kernel_regularizer=tf.keras.regularizers.l2(weight_decay): регуляризатор ядра для контролю перенавчання; у цьому випадку, L2-регуляризатор із зазначеним коефіцієнтом ваги weight_decay.

- padding='same': додає краї вхідного зображення (нулі вкінці зображення), щоб зберегти розмір вихідного зображення.

- BatchNormalization це шар нормалізації, який застосовується після згорткового шару. Batch normalization допомагає збалансувати та стабілізувати ваги в нейронних мережах, покращуючи здатність моделі до навчання.

Саме навчання відбувається наступним чином:

```
lef results(model):
  epoch = 100

r = model.fit(datagen.flow(x_train , ytrain , batchsize = 32), epochs = epoch ,stepsjper_epoch=len(: acc = model.evaluate(x_test , ytest)
print("test set loss : " , acc[0])
print("test set accuracy acc[l]*106)

epoch_range I range(l, epoch+1)
pit.plot(epoch_range, r.history['accuracy'])
pit.plot(epochrange, r.history['valaccuracy'])
pit.title('Classification Accuracy')
pit.ylabel('Accuracy')
pit.xlabel('Epoch')
pit.legend(['Train', 'Vai'], loc='lower right')
pit.show()

# Plot training & validation loss values
pit.plot(epoc h_range,r.h istory['los s'])
pit.plot(epoch_range, r.history['val_loss'])
pit.title('Model loss')
pit.ylabel('Loss')
pit.xlabel('Epoch')
pit.legend(['Train', 'Vai'], loc='lower right')
pit.show()
```

Рис.1.11 - Навчання моделі нейронної мережі

Ефективність начання моделі має значення приблизно 87 відсотків при втратах тестування 40. Це можна побачити на наступній діаграмі

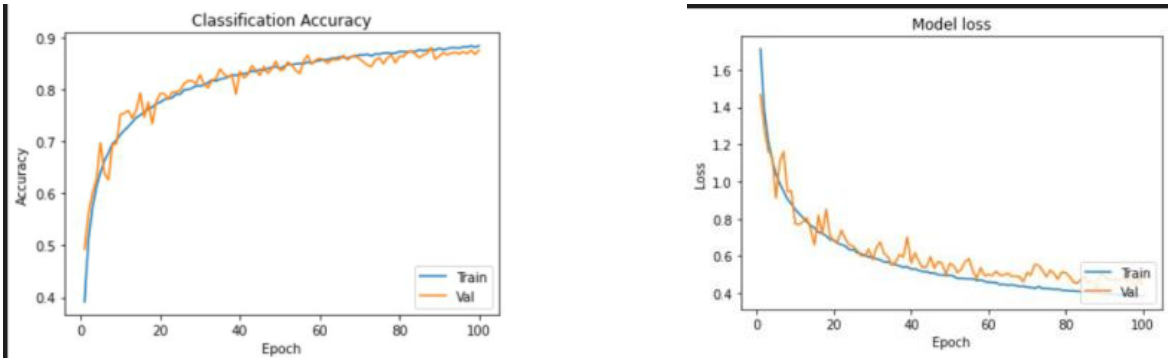


Рис.1.12 - Графік ефективності навчання моделі та втрат при тестуванні

Таким чином, на основі отриманих даних проведено навчання моделі, яка побудована з імплементацією датасету Cifar-10, для класифікації зображень на 10 класів. Ця модель буде важливою для функціонування всього додатку, адже після навчання вона зберігається, як окремий файл та може бути використаною для імплементації в великій кількості веб додатків.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ПІДХОДІВ ДО РОЗГОРТАННЯ ВЕБ-ДОДАТКУ З РОЗПІЗНАВАННЯМ ЗОБРАЖЕНЬ

2.1. Дослідження сучасних підходів до розгортання веб-додатків

З появою різноманітних технологій для створення веб-сервісів виникає необхідність у розгортанні системи для доступу кінцевих користувачів. Існує багато механізмів та технологій для вирішення цього завдання, проте вибір методу розгортання залежить від потреб та можливостей замовника та складності самого сервісу.

Необхідно врахувати, що кожен метод розгортання має свої переваги та недоліки, і ефективність деяких може обмежуватися для конкретних завдань. В практиці найчастіше використовують такі методи розгортання.

Метод розгортання "вручну" та за допомогою bash-скриптів є одним із найстаріших підходів із розгортання сервісів. Цей підхід дозволяє задіяти додаток на сервері за допомогою однієї інструкції командного рядка для запуску сервісу. Однак, це в основному застосовується до простих веб-сервісів. Для складніших додатків часто доводиться використовувати більше команд, наприклад, для запуску веб-додатку та декількох мікросервісів. Однак такий тип деплой може призвести до потенційних катастроф, які стають серйозною проблемою як для бізнесу, так і для розробників в бізнес умовах IT, де важлива швидкість та безпека, і ручне деплоймент невдовзі може стати непрактичним [23].

Для автоматизації ручних команд для деплою можна використовувати bash-скрипти для поетапного виконання. Також, об'єднуючи запуск кількох bash-файлів в один, можна спростити цей напівавтоматичний метод деплою, зменшити вплив людського фактора та прискорити процес.

Але перспективними і важливими аспектами розробки програмного забезпечення є сучасні підходи до автоматизації розгортання веб-додатків. Вони спрямовані на оптимізацію, прискорення та стандартизацію процесу доставки веб-

додатків від середовища розробки до середовища продуктивності (production), забезпечуючи стабільність, безпеку та масштабованість. Складові таких підходів систематизовані нами в табл. 2.1.

Таблиця 2.1.

Сучасні підходи до автоматизації розгортання веб-додатків

Складові підходів	**Інструменти та технології**
Контейнеризація (Containerization)	Docker – найбільш поширений інструмент контейнеризації, що забезпечує узгоджене середовище розробки та продуктивності, мінімізуючи ризик того, що програма працюватиме по-різному на різних системах.
Інфраструктура як код (IaC)	Terraform або AWS CloudFormation - дозволяють автоматизувати процес створення та управління інфраструктурою для веб-додатків.
Безперервна інтеграція та розгортання (CI/CD)	Jenkins, GitLab CI, CircleCI або GitHub Actions – поширені інструменти для реалізації CI/CD для сучасних веб-додатків, що постійно оновлюються.
Оркестрація контейнерів	Kubernetes –платформа, що забезпечує автоматичне розподілення великої кількості контейнерів по вузлах, їх масштабування, самовідновлення та балансування навантаження.
Моніторинг і логування	Prometheus, Grafana, а також Elasticsearch, Logstash та Kibana (ELK-стек) використовуються для моніторингу і аналізу роботи контейнерів та інфраструктури.
Автоматизація масштабування	Автоскейлінг в Kubernetes або у хмарних платформах автоматично керують навантаженням на додаток.
Управління паролями, ключами	HashiCorp Vault або AWS Secrets Manager дозволяють безпечно зберігати та керувати секретами.

Зазначимо, що сучасний підхід до автоматизації розгортання веб-додатків значно підвищує швидкість і якість розгортання додатків, дозволяючи швидко масштабувати інфраструктуру, ефективно управляти ресурсами та забезпечувати безперебійне функціонування систем. Це робить процес розгортання більш керованим, передбачуваним і безпечним.

2.2. Дослідження інструментів та технології автоматизації розгортання веб-додатків.

Розглянемо більш детально представлені інструменти та технології автоматизації розгортання веб-додатків.

Деплой за допомогою Docker та Docker Compose. Docker — це відкрита система для розробки, доставки та запуску додатків. Docker дозволяє відокремити ваші програми від інфраструктури, щоб швидко постачати програмне забезпечення [24]. Docker надає можливість запускати програми в ізольованому середовищі, яке називають контейнером. Це дозволяє використовувати багато контейнерів на хостовій машині, відрізняючись від віртуальних машин (ВМ), які інкапсулюють цілу операційну систему разом з виконуваним кодом поверх абстрактного рівня апаратних ресурсів.

Відмінність між Віртуальними машинами та Docker Контейнерами полягає в тому, що контейнери ділять хостову операційну систему, навпаки від Віртуальних машин, які працюють у власній операційній системі. Окрім різниці в віртуалізації, контейнери віртуалізують операційну систему, тоді як Віртуальна машина віртуалізує апаратний рівень. Також вони відрізняються енергозатратами пам'яті та часом запуску, де контейнери виявляються більш ефективними.

Контейнер упаковує службу або функціонал програми разом із всіма бібліотеками, файлами конфігурації, залежностями та іншими необхідними компонентами для роботи. Кожен контейнер використовує послуги однієї базової операційної системи. Docker-образи містять всі необхідні залежності для виконання коду всередині контейнера, тому контейнери, які переносяться між Docker- середовищами з однією ОС, працюють без змін.[25] Контейнери не залежать від середовища розгортання.

Docker базується на архітектурі клієнт-сервер. Docker-клієнт взаємодіє з Docker- демоном, який керує об'єктами Docker, такими як контейнери, імеджі, мережі. Docker- імедж — це шаблон для створення контейнера. Всі Docker-імеджі

зберігаються в реєстрі, до якого звертається демон для пошуку імеджу для створення контейнера. Принцип роботи Docker представлено на рис. 2.1.

Проблему обмеження використання лише одного контейнера для додатку вирішує docker-compose, надаючи можливість запускати мультиконтейнерні додатки.

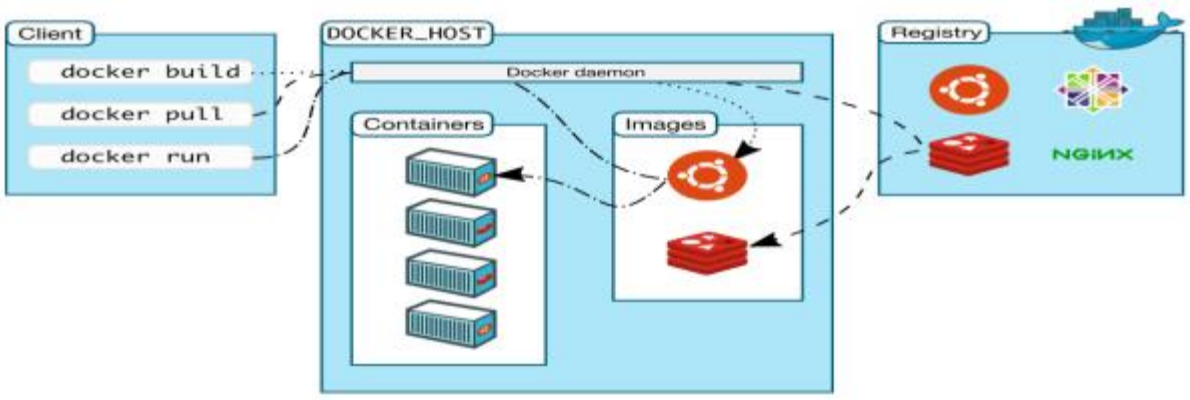


Рис. 2.1 – Схематичне зображення принципу роботи та архітектури Docker.

Для цього слід створити конфігураційний файл у форматі YAML, в якому визначаються всі необхідні сервіси для створення додатку. Docker використовується для керування окремими контейнерами, що входять до складу додатку, тоді як docker-compose використовується для одночасного управління кількома контейнерами. Різниця між docker та docker-compose проілюстрована на рис. 2.2.

Використання Docker для розгортання надає гнучкі можливості, зокрема:

- Можливість розгортання контейнерів у будь-якому середовищі з запущеним Docker.

- Масштабованість дозволяє швидке створення нових контейнерів залежно від потреб додатку та навантаження.

- Продуктивність розгортання програмного забезпечення в контейнерах виявляється набагато швидшою, ніж розгортання на тих самих серверах чи

віртуальних машинах, що може бути важливим для швидкого тестування програми та моніторингу її роботи [26].

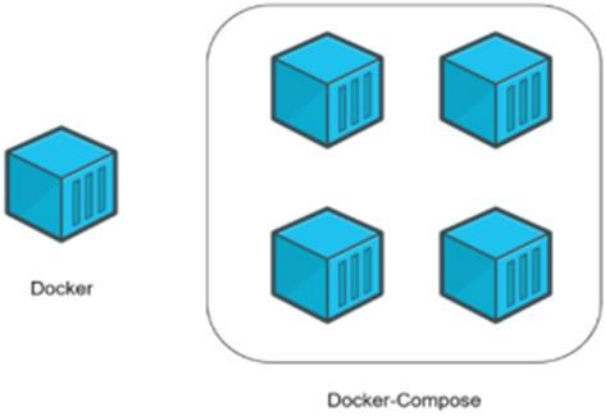


Рис. 2.2 - Різниця між докером та docker-compose

Використання сервісів CI/CD забезпечує автоматизоване впровадження додатків, що дозволяє зменшити кількість помилок на цьому етапі, які могли б виникнути внаслідок людського фактору.

CI/CD - це метод частої доставки програм клієнтам шляхом впровадження автоматизації на етапах розробки додатків. Основні концепції, пов'язані із CI/CD, це безперервна інтеграція, безперервна доставка та безперервне розгортання [27].

Безперервна інтеграція (англ. Continuous Integration) - це практика розроблення програмного забезпечення, яка полягає у виконанні частих автоматизованих збірок проекту для якнайшвидшого виявлення та розв'язання інтеграційних проблем [28]. Успішним впровадженням процесу CI є миттєве завантаження програмного коду розробника до спільного репозиторію, який керується системою контролю версій - СКВ (GitLab, GitHub).

Безперервна доставка (Continuous Delivery) являє собою програмний інженерний підхід, за яким команди виробляють програмне забезпечення в коротких циклах, гарантуючи, що програмне забезпечення може бути надійно

випущене в будь- який час [29]. Цей процес передбачає, що зміни, які були зроблені розробником, автоматично будуть пропущені через автотести, і при успішному їх проходженні вони будуть завантажені у репозиторій (наприклад, GitHub, GitLab чи інша СКВ). Після цього ці зміни можуть бути розгорнуті в робоче середовище [30].

Безперервне розгортання (англ. Continuous Deployment) передбачає автоматичне впровадження коду з репозиторію, що забезпечує оновлення версії програмного продукту.

В загальному, ці три вищезазначені практики часто називають "пайплайном CI/CD", який підтримується командами розробників та операцій, які працюють разом у гнучкий спосіб з дотриманням методології DevOps. Пайплайн можна розглядати як чіткий процес, який містить свої етапи. Згідно з документацією Red Hat [27], процес CI/CD можна розділити на такі етапи (див. рис. 2.3).

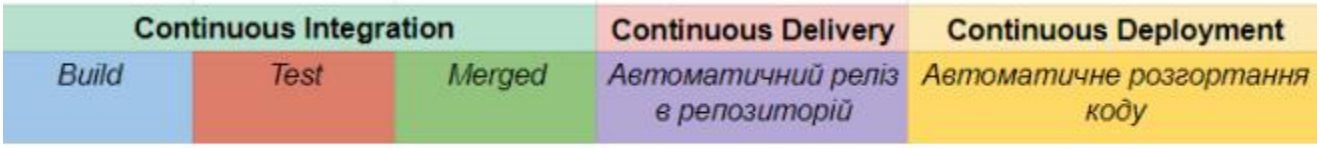


Рис. 2.3 - Послідовність CI/CD процесів

Застосування сервісів CI/CD порівняно з іншими способами має наступні переваги:

- Швидший вихід на ринок

Основною метою CI/CD пайплайну є доставка програмного забезпечення користувачам якнайшвидше та з більшою частотою.

- Зниження ризиків на етапі продакшену

Можливість швидко та регулярно тестувати програмне забезпечення дає широкі можливості на етапі тестування, що допомагає швидко реагувати на баги та виправляти їх, покращуючи якість коду.

- Ефективна інфраструктура

Впровадження підходу "інфраструктура-як-код" (IaC) дозволяє автоматизовано створювати середовище для розгортання програмного

забезпечення, замість того, щоб налаштовувати кожен сервер вручну.

- Вимірюваний прогрес

Використання сервісів CI/CD надає цілу низку метрик від часу збірки до покриття тестуванням, частоти дефектів до часу виправлення тесту. Ці дані допомагають визначити області, які можуть потребувати уваги, та швидко реагувати на зміни за потреби.[31].

2.3. Дослідження хмарних сервісів для розгортання веб-додатку.

В будь-якому із обраних методів розгортання найважливішим елементом є інфраструктура, де розгортається додаток, адже саме вона впливає на побудову архітектури розгортання, тому важливо розуміти різницю в різних тиипах інфраструктури.

Розгортання будь-якого програмного продукту здійснюється за допомогою використання ресурсів сервера. Однак спосіб надання цих ресурсів може варіюватися. Загалом можна виокремити дві моделі виділення серверних ресурсів. Це архітектура on-premise та архітектура хмарних обчислень.

Архітектура on-premise — це програмне та апаратне забезпечення інфраструктури, розгорнуте та запущене в межах конкретної організації. У цьому випадку організація має повний контроль над налаштуванням інфраструктури. Дані залишаються в приватній мережі, інформація доступна лише команді організації [32]. При такій побудові інфраструктури організація самостійно налаштовує власні сервери та дата-центри за потребою. Основною перевагою такої архітектури є захищеність даних, але конфігурування всієї системи безпеки та налаштування серверів вимагають ручної роботи.

Також слід відзначити, що для використання вже наявних сервісів можливо знадобиться зміна конфігурації для інтеграції цих сервісів у систему. Незважаючи на труднощі налаштування та проблеми із інтеграцією, такий тип побудови інфраструктури має свої переваги, зокрема:

- Надає фізичний контроль над усіма серверами.

- Забезпечує конфіденційність даних всередині інфраструктури.

- Для доступу до даних не потрібне Інтернет-з'єднання.

- Є більш рентабельним для компаній, які не так цікавляться неперервністю роботи програмного забезпечення.

Іншим типом виділення ресурсів сервера є хмарні обчислення. Хмарні обчислення (Cloud Computing) — це модель забезпечення доступу на вимогу через мережу до спільного пулу обчислювальних ресурсів, які підлягають налаштуванню (наприклад, серверів, засобів зберігання даних, прикладних програм та сервісів). Простіше кажучи, хмара — це місце, де можна орендувати віртуальні сервери чи інші ресурси для програмного забезпечення. Розгортання інфраструктури в хмарі має такі переваги.

- В хмарі оплачується лише за використані ресурси та тільки в період їх використання, це називається "pay-as-you-go". Ми не платимо за простій електроенергії.

- Хмара забезпечує гнучку масштабованість в залежності від потреб клієнтів. Можна швидко збільшити чи зменшити обчислювальну потужність та місткість сховища.

- Хмарні обчислення роблять резервне копіювання даних, аварійне відновлення та безперервність бізнесу простішими, менш дорогими і забезпечує надійність.

- Оскільки провайдери хмарних обчислень регулярно оновлюють своє обладнання, що збільшує продуктивність мережі та зменшує кількість затримок, надаючи більшу економію при масштабуванні.

- Багато постачальників хмарних послуг пропонують широкий набір політик, технологій та засобів керування, які зміцнюють безпеку взагалі, допомагаючи захистити дані користувачів, програми та інфраструктуру від потенційних загроз.

Хмари поділяють на приватні, публічні та гібридні. Вони можуть

відрізнятися в залежності від потреб та умов замовника та самого сервісу.

- Приватна хмара коли всі ресурси приватної хмари використовуються лише однією компанією або групою компаній.

- Публічні хмари належать і керуються сторонніми постачальниками хмарних послуг, які надають свої обчислювальні ресурси через Інтернет. Прикладами загальнодоступних хмар є Microsoft Azure, AWS, GCP. У цьому випадку всю інфраструктуру контролює та належить провайдеру хмари.

- Гібридні хмари об'єднують публічні та приватні хмари, забезпечуючи обмін даними та додатками між ними.

Хмарні послуги хоч і забезпечують гнучкість в побудові інфраструктури розгортання, однак варто зазначити, що відповідно до своєї моделі вони можуть надавати різні типи послуг. Існують три основні моделі надання хмарних сервісів, які забезпечують доступ до різних послуг.

- Інфраструктура як сервіс (IaaS - Infrastructure as a Service) - це базовий рівень хмарних обчислень і включає засоби зберігання, обчислень, резервування, відновлення після збоїв, роботи з базами даних і забезпечення безпеки, що надаються постачальником інфраструктури.

- Платформа як сервіс (PaaS - Platform as a Service) спрямована на розробників, які бажають витрачати більше часу на кодування, тестування та розгортання своїх програм.

- Програмне забезпечення як сервіс (SaaS - Software as a Service). Це найбільш поширена модель хмарних обчислень, яка надає готові рішення, такі як Gmail, Dropbox, Netflix.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ПІДХОДУ ДО АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ВЕБ-ДОДАТКУ З РОЗПІЗНАВАННЯМ ЗОБРАЖЕНЬ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1. Побудова інфраструктури розгортання веб-додатку

Побудова інфраструктури буде відбуватись відповідно до архітектури розгортання через використання інструменту Terraform, який використовує підхід Інфраструктури як код для побудови середовища. Цей інструмент визначається тим, що автоматично створює інфраструктуру відповідно до взаємозв'язків між ресурсами та станом інфраструктури на момент внесення змін до конфігурації. Тому для правильної побудови інфраструктури важливо урахувати всі зв'язки між різними ресурсами. Також необхідно налаштувати підключення до провайдера та вибрати потрібну версію провайдера. В нашому випадку цим провайдером буде Microsoft Azure.

Підключення провайдера відбувається наступним чином:

```
required_providers {
azurerm = {
source = "hashicorp/azurerm"
version = "~>2.30"
}
}
```

Terraform використовує декларативний підхід в побудові інфраструктури та спирається на поточний стан інфраструктури, тому перш ніж почати побудову інфраструктури слід ініціалізувати робочий репозиторій та зберегти поточний стан інфраструктури. Для ініціалізації використовується команда *terraform init* після виконання цієї команди записується поточний стан Terraform та створюються файли залежності, які потрібні для роботи Terraform (див. рис. 3.1).

Заключним етапом перед написанням конфігурації є підтвердження

підписки, що необхідно для зв'язку з акаунтом провайдера. Всі ресурси, які описані в конфігурації, будуть створюватися саме в цьому акаунті.

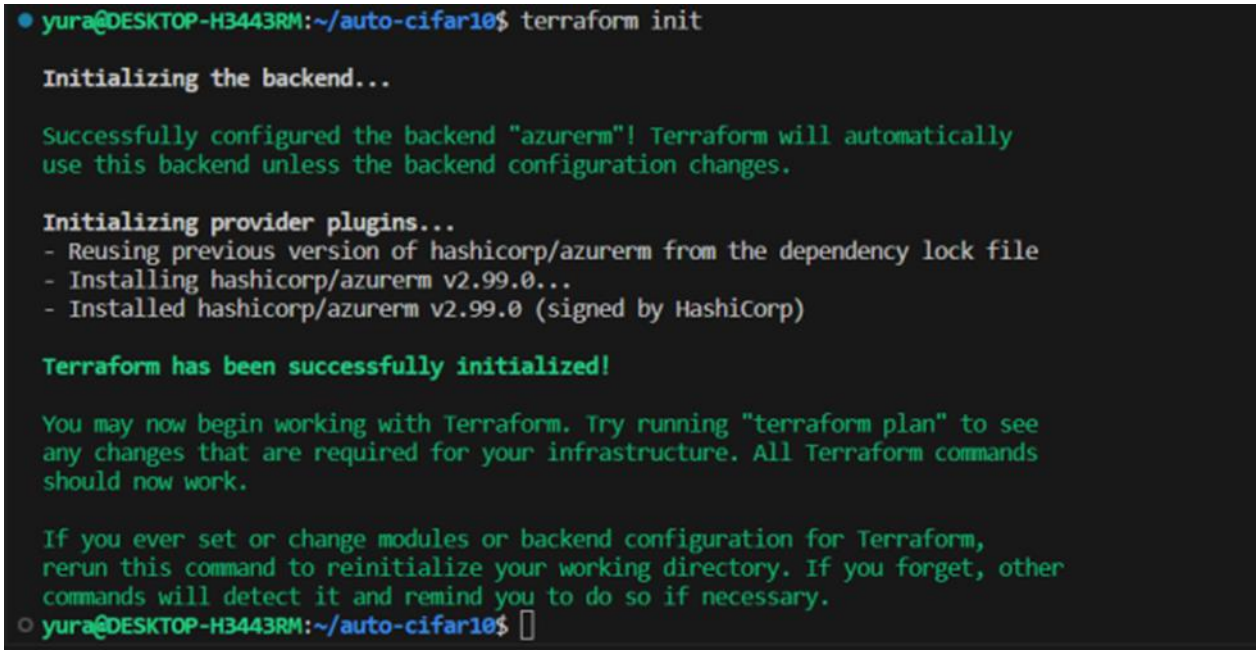


Рис. 3.1 – Ініціалізація робочого репозиторію

Для передачі даних про акаунт достатньо виконати команду az login. Після входу в акаунт та авторизації дані про підписку будуть внесені в конфігурацію Terraform, що спростить доступ до неї без необхідності повторного входу перед роботою з Terraform у вже ініціалізованій папці.

Після виконання підготовчих кроків можна перейти до конфігурування інфраструктури. Написання коду для побудови інфраструктури відбувається за принципом "від загального до конкретного", де спочатку створюються базові ресурси, які використовуються для створення більш складних.

До базових ресурсів входить ресурсна група, де описується інформація про регіон для розгортання інфраструктури. Вибір регіону визначає доступні типи ресурсів, оскільки в різних регіонах може бути різна наявність ресурсів.

Ресурсну групу можна уявити як директорію, в якій зберігаються всі ресурси, що дозволяє легко будувати між ними зв'язки. Далі слід налаштовувати віртуальну

мережу, яка взаємодіє з ресурсами. Спочатку створюється віртуальна мережа з вказанням діапазону IP-адрес. Налаштування віртуальної мережі також включає конфігурацію підмереж, кількість яких залежить від адресного простору віртуальної мережі.

Для забезпечення можливості надсилання запитів до віртуальної мережі з Інтернету слід налаштовувати правила вхідного трафіку в ресурсі, відомому як network security group. Цей ресурс дозволяє керувати вхідним та вихідним трафіком мережі, а правила для вхідного трафіку представлені в таблиці 3.1.

Таблиця 3.1.

Правила для групи безпеки віртуальної мережі

Протокол передачі даних	Доступ	Протокол управління передачею	Порт призначення	Пріоритет
SSH	Allow	TCP	22	120
HTTP	Allow	TCP	80	130
HTTPS	Allow	TCP	443	140

Пріоритет визначає порядок застосування правил. Дані правила діють лише всередині мережі створеної в MS Azure для комунікації між ресурсами.

Для того, щоб зробити віртуальні машини з скейл сету доступними ззовні потрібно надати їм публічну IP-адресу. Згідно із архітектурою розгортання публічна адреса буде спільною для всіх віртуальних машин в скейл сеті. Все це можливо через застосування ресурсу, який має назву балансувальник навантажень.

Для реалізації балансувальника навантажень в Terraform необхідно створити та встановити взаємозв'язки між кількома ресурсами. Перш за все, це включає сам балансувальник навантажень, який буде пов'язаний із іншими ресурсами, відповідальними за здійснення балансування навантаженнями.

Початковий крок - зв'язати балансувальник навантажень із раніше створеною

публічною адресою, щоб забезпечити його доступність в мережі. Окрім цього, потрібно створити пул бекенд адрес, який визначатиме адреси екземплярів віртуальних машин сету, які обслуговуються цим балансувальником.

Додатково необхідно налаштувати правило балансування навантаженнями в Terraform. Його конфігурація має вигляд, зазначений на рисунку 3.2.

```
// Creating loadbalancer rule which allow trafic to balancer from 80 port
resource "azurerm_lb_rule" "lbnatrule" {
    resource_group_name            = azurerm_resource_group.res-1.name
    loadbalancer_id                = azurerm_lb.lb-1.id
    name                           = "http"
    protocol                       = "Tcp"
    frontend_port                  = 80
    backend_port                   = 80
    backend_address_pool_ids        = [azurerm_lb_backend_address_pool.bpepool.id]
    frontend_ip_configuration_name = "PublicIPAddress"
    probe_id                       = azurerm_lb_probe.hth-1.id
}
```

Рис. 3.2 - Правило балансувальника навантажень

Правило балансування навантаження визначає спосіб розподілу вхідного трафіку між різними екземплярами з пулу адрес бекенд серверів. Його конфігурація включає визначення IP-адреси та порту інтерфейсу для зіставлення з різними серверними IP-адресами та портами. У цьому випадку, вхідний трафік з порту 80 буде розподілятись і оброблятись на всіх серверах бекенд, також на порті 80. Така конфігурація дозволяє балансеру обробляти http-трафік від віртуальних машин.

У конфігурації правил балансування навантаження можна помітити ресурс з ім'ям probe_id. Проба працездатності використовується для визначення стану кінцевої точки, і результат її роботи визначає, чи перенаправляти вхідний трафік між екземплярами бекенд пулу. Це дозволяє контролювати навантаження та виявляти можливі несправності в роботі програми на конкретному сервері. У випадку розгортання веб-сервісу проба буде перевіряти готовність серверів обробляти трафік, який надходить на порт 80.

Наступний етап передбачає створення скейл сету віртуальних машин. Саме на цих машинах буде відбуватись розгортання веб додатку тому, що використання скейл сету, а не просто віртуальної машини дає можливість масштабування. В скейл сеті можна збільшувати кількість машин, на яких працює додаток. Таке масштабування можна налаштувати просто збільшивши вручну кількість машин в самій конфігурації. Ця конфігурація виглядає наступним чином (див. рис. 3.3).

```
resource "azurerm_virtual_machine_scale_set" "vmss-1" {
  name                = "example-vmss"
  resource_group_name = azurerm_resource_group.res-1.name
  location            = azurerm_resource_group.res-1.location
  upgrade_policy_mode = "Manual"

  // capacity it is number of instaneces in our vmss
  sku {
   name     = "Standard_DS1_v2"
   tier     = "Standard"
   capacity = 1

  }
```

Рис. 3.3 - Конфігурація сету віртуальних машин

Тут можна напряму впливати на кількість віртуальних машин, змінюючи змінну *capacity*, як видно з рисунка 3.3 в сеті зараз є лише одна віртуальна машина. Такий спосіб масштабування є доволі простим, адже коли збільшується трафік, тоді просто вручну збільшують кількість віртуальних машин, але такий підхід є доволі довгим, адже потребує ручних змін, а у випадку, коли неочікувано потрібно збільшити місткість сету, тоді можуть виникати проблеми.

Тому в такому випадку можна використати такий ресурс як *azurerm_monitor_autoscale_setting*, який допоможе автоматизувати масштабування в залежності від навантаження, яке надходить на екземпляри віртуальних машин всередині скейл сету.

Для конфігурації цього ресурсу в Terraform необхідно передбачити певні вхідні параметри, які будуть ключовими для автоматизованого масштабування:

- мінімальна та максимальна кількість машин у мережі;
- кількість машин у мережі за замовчуванням;
- параметр для збору метрик;
- максимальні та мінімальні значення метрик;
- інтервал для розгортання.

Механізм автоматизованого масштабування в Terraform діє за принципом - спочатку кількість машин встановлюється на значення за замовчуванням. Далі регулярно проводиться збір та перевірка даних метрик, щоб визначити, чи не перевищують вони максимального значення. У разі перевищення максимального значення до існуючої кількості машин додається певна кількість, яка визначається в правилах. Параметр збору метрик може мати багато тригерів, але в даній роботі використовується метрика навантаженості центрального процесора. Правила збору метрики для збільшення кількості машин представлені на рис. 3.4.

```
rule {
  metric_trigger {
    metric_name        = "Percentage CPU"
    metric_resource_id = azurerm_virtual_machine_scale_set.vmss-1.id
    time_grain         = "PT1M"
    statistic          = "Average"
    time_window        = "PT5M"
    time_aggregation   = "Average"
    operator           = "GreaterThan"
    threshold          = 90
  }
  scale_action {
    direction = "Increase"
    type      = "ChangeCount"
    value     = "2"
    cooldown  = "PT1M"
  }
}
```

Рис. 3.4. Правила збору метрики для збільшення кількості машин

На даному рисунку можна побачити, що збільшення кількості віртуальних машин відбувається динамічно у зв'язку із високим пороговим значенням завантаженості центрального процесора. В результаті до скейл сету додаються ще два образи, щоб розподілити ресурси.

Також варто відзначити, що таке масштабування відбувається і в зворотньому напрямку, коли навантаження є невеликим і кількість машин зменшується для економії ресурсів (див. рис. 3.5).

```
rule {
  metric_trigger {
    metric_name        = "Percentage CPU"
    metric_resource_id = azurerm_virtual_machine_scale_set.vmss-1.id
    time_grain         = "PT1M"
    statistic          = "Average"
    time_window        = "PT5M"
    time_aggregation   = "Average"
    operator           = "LessThan"
    threshold          = 10
  }

  scale_action {
    direction = "Decrease"
    type      = "ChangeCount"
    value     = "2"
    cooldown  = "PT1M"
  }
}
```

Рис.3.5 - Правила збору метрики для зменшення кількості машин

У ситуації, коли використання процесору складає менше 10%, кількість машин у мережі буде зменшена на дві одиниці, щоб уникнути надмірного використання ресурсів для обслуговування сервісу. Важливо зауважити, що це правило не буде застосовано, якщо кількість машин у масштабованій мережі вже менша за три.

Terraform також надає можливість налаштувати автоматичне розгортання протягом певного періоду часу. Можна налаштувати так, що розгортання буде відбуватись щотижня. Також є можливість налаштувати сповіщення щодо зміни стану архітектури.

Етап створення публічної адреси є важливим не тільки для балансування навантаженням, але і для наступного етапу, управління конфігурацією. За замовчуванням в машинах скейл сету немає публічних адрес, але вони необхідні

для управління конфігурацією, тому важливо додати ще створення публічної IP-адреси для кожної із віртуальних машин.

Завершивши налаштування всіх етапів створення файлу конфігурації інфраструктури, для застосування цієї конфігурації та створення ресурсів, описаних у файлі на платформі хмарного провайдера Microsoft Azure, використана команда *terraform apply,* а результат її виконання показаний на рис. 3.6.

Рис. 3.6. Результат виконання команди terraform apply

Процес управління конфігурацією полягає у наступному. Як зазначалось вище, для того, щоб мати змогу через Ansible сконфігурувати процес розгортання на створеній інфраструктурі потрібно мати адресу хоста, на якому буде відбуватись процес розгортання. Для цього потрібно витягнути публічну адресу віртуальних машин, які були створені через Terraform.

Зробити це можна, розпарсивши ці адреси при створені інфраструктури. Для виконання цього завдання використовувався внутрішній командній рядок Azure, а саме, дана команда:

az vmss list-instance-public-ips --name example-vmss --resource-group azure-esource_group

Дана команда виводить всі публічні адреси скейл сету, а також виводиться ще інша додаткова інформація (див. рис. 3.7).

Тому для того, щоб з цього потоку даних отримати необхідну адресу буде використовуватись скрипт для парсингу *parser.py*, який виводить адреси в файл *ip.txt.*

Рис. 3.7. Результати виконання команди з отримання публічної адреси.

Однак перед тим, як записати ці адреси в файл host.txt потрібно також додати до кожного хоста наступні параметри:

- ansible_become=yes використовується для виконання всіх завдань у плейбуці в ролі користувача з привілеями, щоб мати можливість завантажувати пакети на хости.

- ansible_ssh_user=adminuser визначає ім'я користувача, за допомогою якого відбувається вхід на віддалені сервери.

- ansible_ssh_private_key_file=~/.ssh/id_rsa вказує шлях до приватного ssh-ключа, необхідного Ansible для зв'язку з хостами.

- ansible_python_interpreter=/usr/bin/python3 встановлює шлях до інтерпретатора Python, оскільки Ansible написаний на Python і вимагає наявності цього інтерпретатора для своєї роботи.

Для автентифікації окрім ssh-ключа також можна використовувати пароль.

Після конфігурації inventory файлу, можна переходити до процесу розгортання сервісу через ansible-playbook. Загальний опис цього процесу зображений на рисунку 3.8, але даний підхід є доволі узагальнюючим і не відображає всіх деталей.. На практиці кожен етап включає значну кількість кроків, проте за допомогою механізму Ansible розгортання стає більш простим завдяки його простому синтаксису YAML та подібності деяких завдань у playbook.

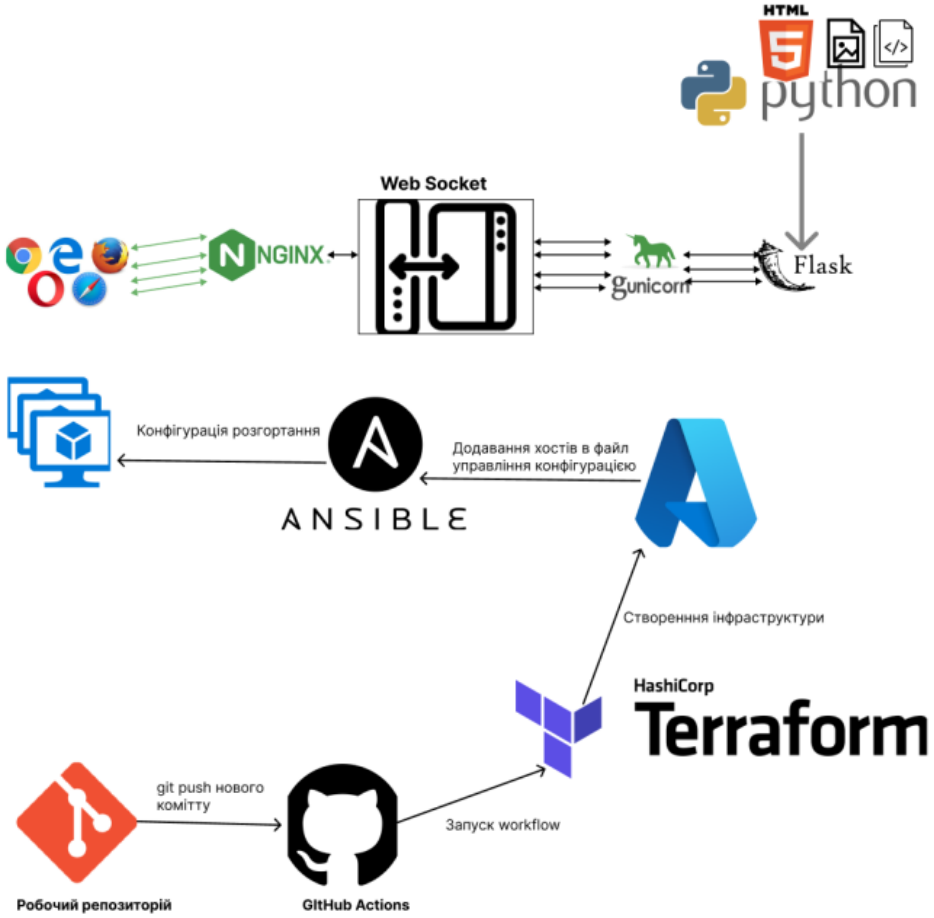


Рис. 3.8. Структурна схема та схема архітектури розгортання веб-сервісу

Отже, з початку налаштовується Python на машинах, щоб завантажити всі необхідні залежності для коректної роботи сервісу. Також на цьому етапі потрібно встановити веб-сервер Nginx, сервер додатків Gunicorn та віртуальне середовище Python-venv. Усі команди для розгортання додатку будуть виконуватися не локально на кожному сервісі, а у віртуальному середовищі Python. Це пояснюється тим, що використання venv надає ряд переваг, зокрема, відсутність необхідності слідкувати за версією Python на машині та уникнення проблем з сумісністю залежностей. Можна використовувати будь-яку версію Python для обраного середовища, що стає важливим, оскільки проєкт, який розгортається, був написаний кілька років тому, і можуть виникнути колізії з версією Python на локальному комп'ютері. Крім того, venv дозволяє завантажувати лише необхідні для проєкту пакети, використовуючи утиліту pip, не додаючи зайвого до проєкту.

Після встановлення необхідних пакетів завантажимо вихідний код веб-сервісу на віддалені сервери, використовуючи команду git clone, оскільки вихідний проєкт розміщений на GitHub.

Наступним етапом йде встановлення необхідних залежностей для запуску проєкту. Процес завантаження відбувається у віртуальному середовищі за допомогою утиліти pip. Усі необхідні пакети визначені в файлі requirements.txt, який стає доступним для машин після завантаження коду з GitHub. Важливою перевагою Ansible є те, що для кожного з вищезгаданих завдань існує окремий модуль, що спрощує процес конфігурування та робить його більш зрозумілим.

Далі здійснюється налаштування всіх необхідних сервісів для функціонування додатку. Основними сервісами є веб-сервер nginx та сервер додатків Gunicorn. Для запуску будь-якого сервісу необхідно створити файл конфігурації, який описує роботу кожного із сервісів.

Дані конфігураційні файли будуть створюватись на віддалених машинах. Також варто відзначити, що їхній вміст буде записуватись одразу в ansible playbook. Створення конфігураційного файлу для nginx має наступний вигляд (див. рис. 3.9).

```
- name: Create Nginx configuration file
  ansible.builtin.copy:
    content: |
      server {
        listen 80;
        server_name {{ host }};

        location / {
            proxy_pass http://unix:/tmp/{{ host }}.socket;
            proxy_set_header Host $host;
        }
      }
    dest: /etc/nginx/sites-available/{{ host }}
```

Рис. 3.9 - Конфігураційний файл nginx

Звідси видно, що контент буде записуватись на віддалену машину в той файл, який вказаний в шляху через змінну dest. Основні елементи цього файлу наступні.

- Процес слухання порту listen 80; — цей сервер слухатиме вхідні з'єднання на порті 80, що є портом для HTTP протоколу.
- Ім'я сервера server_name {{ host }}; — конфігурує ім'я сервера, яке визначається значенням змінної host. Це дозволяє налаштовувати різні віртуальні хости. Значення змінної хост є публічна IP адреса кожної віртуальної машини, тому при збільшені кількості віртуальних машин для кожної буде створюватись окрема директорія, яка буде містити конфігураційний файл nginx.
- Обробка запитів, розділ location / { ... } визначає обробку запитів, які відносяться до кореневого URL ("/"). У цьому випадку, всі такі запити будуть передаватись за допомогою проксі на Unix-сокет, який вказується шляхом /tmp/{{ host }}.socket.
- Перенаправлення-проксі proxy_pass http://unix:/tmp/{{ host }}.socket; вказує, що вхідні HTTP-запити повинні бути передані за допомогою проксі на Unix-сокет за вказаним шляхом.

- proxy_set_header Host $host; — додає заголовок Host у вихідний HTTP-запит. Це корисно для правильного розпізнавання імені хоста на віддаленому сервері.

Для того, щоб запустити сервер додатків Gunicorn для веб-сервера необхідно налаштувати файл конфігурації systemd, який використовується для визначення серверу додатків для конкретного хосту. Створення цього файлу в ansible playbook показано на рис. 3.10.

Цей файл складається з трьох частин [Unit], [Service], [Install]. В [Unit] відбувається лише опис служби, в даному випадку це Gunicorn сервер для хоста, який визначається змінною host.

В [Service] є набагато більше опцій, які мають наступне значення.

- Restart=on-failure вказує, що служба має перезапускатися в разі виявлення помилки або аварійного завершення роботи.

```
- name: Create Gunicorn systemd service unit file
  ansible.builtin.copy:
    content: |
      [Unit]
      Description=Gunicorn server for {{ host }}

      [Service]
      Restart=on-failure
      User=root
      WorkingDirectory=/home/{{ ansible_user }}/sites/{{ host }}
      EnvironmentFile=/home/{{ ansible_user }}/sites/{{ host }}/.env

      ExecStart=/home/{{ ansible_user }}/sites/{{ host }}/venv/bin/gunicorn \
          --bind unix:/tmp/{{ host }}.socket \
          app:app

      [Install]
      WantedBy=multi-user.target
    dest: /etc/systemd/system/gunicorn-{{ host }}.service
  notify:
    - Restart gunicorn
  tags:
    - gunicorn
```

Рис. 3.10 - Створення файлу конфігурації для Gunicorn в ansible playbook

- User=root задає користувача, від імені якого служба буде виконуватися, у цьому випадку це користувач "root".

- WorkingDirectory вказує робочий каталог, в якому буде виконуватися служба. У нашому випадку, це каталог /home/{{ ansible_user }}/sites/{{ host }}.

- EnvironmentFile це шлях до файлу середовища, де зберігаються змінні середовища для служби. Зазвичай використовується для збереження конфіденційних даних або параметрів конфігурації. Сам файл також записується вручну на віддалений сервер та має наступний вигляд

IS_DJANGO_DEBUG_FALSE=y

DEBUG=False

SITENAME={{ host }}

- ExecStart це команда для запуску служби. У даному випадку, це команда для запуску Gunicorn з певними параметрами, такими як прослуховування Unix- сокета та вказівка на Python-модуль app:app.

[Install] секція визначає, що служба має бути активована під час завантаження системи в режимі мульти-юзера.

На рис. 3.11 показано, що окрім створення файлу конфігурації для gunicorn є також інша опція notify, яка викликає обробник, так званий хендлер (handlers), який перевантажує сервіс.

```
handlers:
  - name: Restart nginx
    service:  name=nginx state=restarted

  - name: Restart gunicorn
    systemd:
      name=gunicorn-{{ host }}
      daemon_reload=yes
      enabled=yes
      state=restarted
```

Рис. 3.11 – Handlers для gunicorn та nginx

Хендлер потрібний лише для того, щоб застосувати створену конфігурацію. Він буде використовуватись кожного разу, коли конфігураційні файли для сервісів будуть зазнавати якихось змін для того, щоб застосувати ці зміни. Після всіх цих налаштувань можна запустити ansible playbook, через наступну команду: ansible-playbook -i host.txt.

Результат процесу управління конфігурацією показано на рис. 3.12.

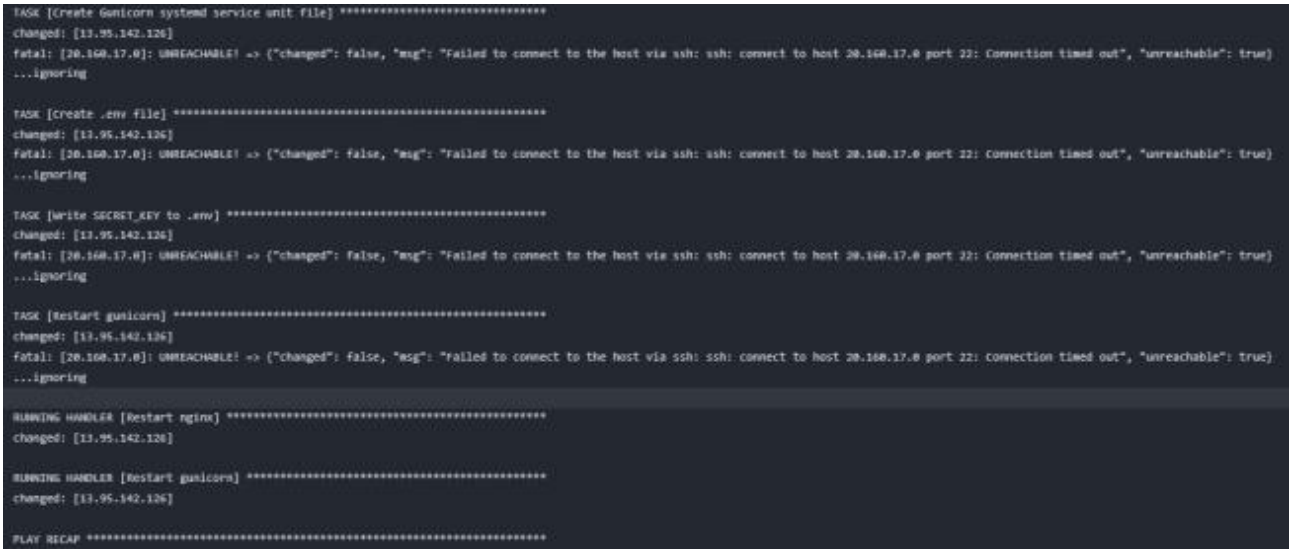


Рис. 3.12 – Результат застосування ansible-playbook

3.2. Розроблення підходу до автоматизації розгортання веб-додатку з опцією розпізнавання зображень

Автоматизація всього розгортання буде відбуватись через інструмент CI/CD Github Actions. Цей інструмент був вибраний через те, що його дуже легко інтегрувати в репозиторії Github, де знаходиться вихідний код програми. Як зазначалось в попередніх розділах весь процес розгортання в Github Actions складаються з Job (завдань, робіт), кожна з яких має власні кроки (steps). Процес розгортання описується в одному файлі з розширенням yml.

Відповідно до архітектури розгортання першим завданням (Job 1) має бути створення інфраструктури, однак, для початку потрібно зробити певні тести, щоб перевірити роботу додатку. Це будуть звичайні unit-тести, які перевірять основну

функціональність додатку, а саме додавання фотографії та успішне виконання класифікації цього зображення відповідно до 10 класів моделі. Тести будуть виконуватись поза віртуальними машинами, створення, яких буде в наступній роботі (Job). Спочатку буде відбуватись встановлення необхідних залежностей, потім відбувається завантаження вихідного коду програми, де міститься файл з тестами з Github, для чого потрібно задати Github токен, який дозволить використовувати цю систему контролю версій в workflow. Цей токен буде зберігатись в сховищі для секретних змінних, яке буде описано нижче. Після завантаження вихідного коду, відбувається запуск тестів. Варто відзначити, що всі кроки, сходинки (steps) йдуть послідовно один за одним і якщо в одному із step буде помилка, то Job не зможе виконатись. Виконання самої Job можна відслідкувати в графічному інтерфейсі і виглядає як показано на рис. 3.13.

Як видно з цього рисунку всі 4 тести пройшли успішно, тому джоба виконалась. Після перевірки основної функціональності системи далі необхідно побудувати інфраструктуру, на якій буде відбуватись розгортання. Для цього необхідно спершу налаштувати доступ до підписки хмарного провайдера.

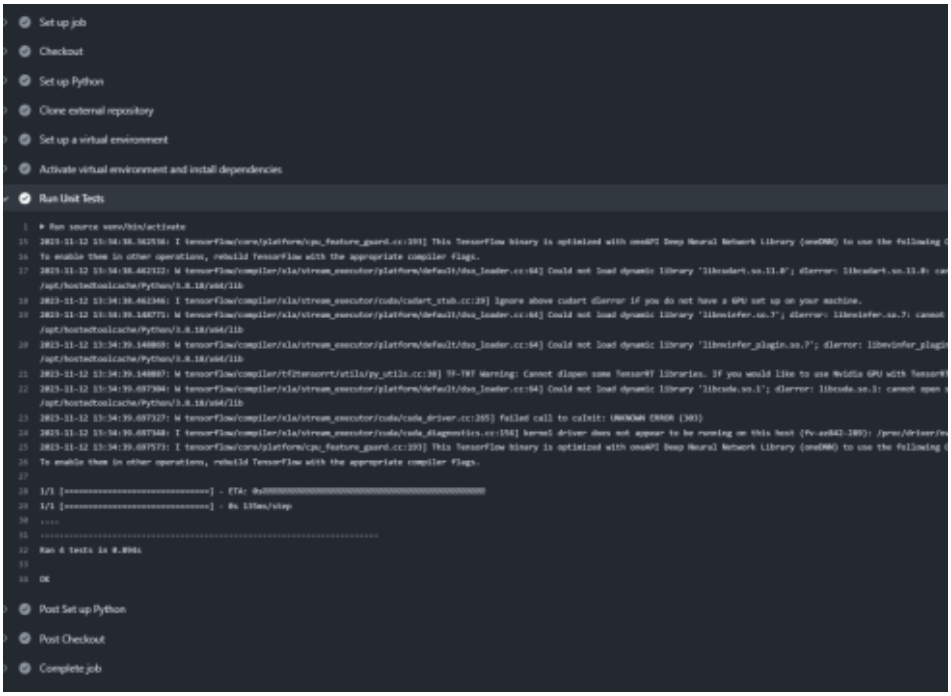


Рис. 3.13 - Виконання unit-тестів в workflow

В Github Actions для цього є зручний механізм змінних середовища, про який говорилось раніше, цей інструмент дозволяє зберігати змінні середовища, що є необхідними для автентифікації в Microsoft Azure, ці параметри зберігаються в сховищі для секретних змінних, куди можна записувати лише один раз і неможливо побачити вмісту змінної (див. рис. 3.14).

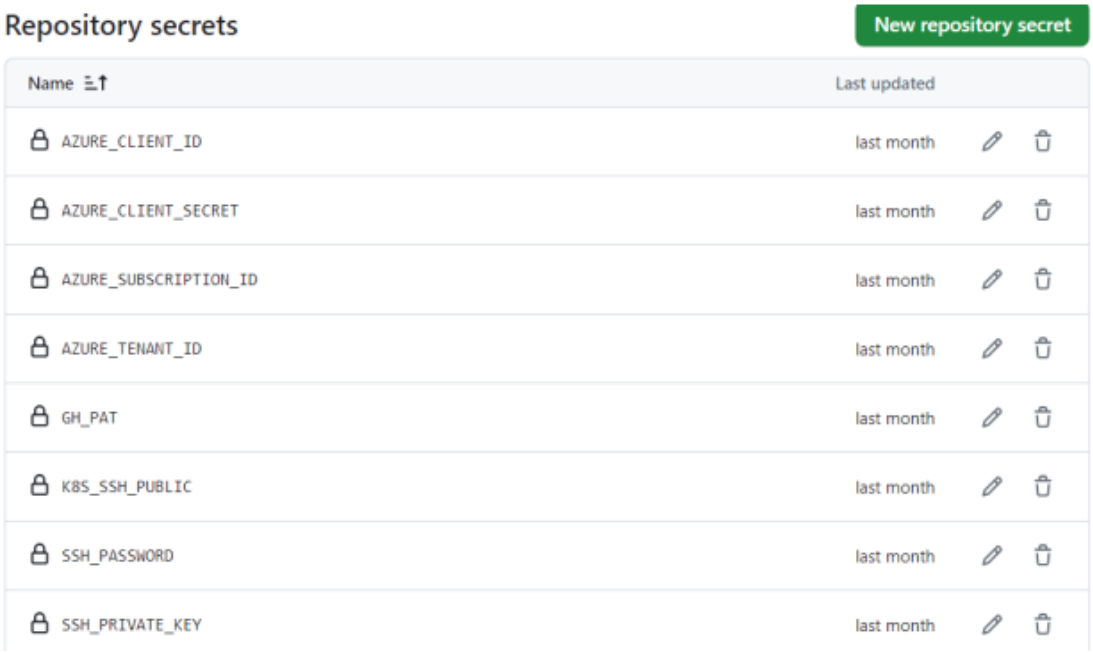


Рис. 3.14 - Сховище для секретних змінних.

Пізніше ці змінні через ініціалізацію можна буде використовувати в файлі main.yml, в якому проходить весь опис Github Actions workflow. Побудова інфраструктури відбувається через Terraform та складаається з двох джоб - terraform plan та terraform apply. Це зроблено через те, що після terraform plan відбувається зберігання стану інфраструктури, яка вже буде розгортатись через terraform apply. Вивід terraform plan зберігається в спеціальному ресурсі Microsoft Azure Storage Account, який використовується для зберігання даних. Всередині цього ресурсу перед розгортанням інфраструктури створюється контейнер з назвою tfstate, в якому зберігається поточний стан інфраструктури (див. рис. 3.15).

Після цього створюється джоба terraform apply, в якій береться поточний стан інфраструктури та відбувається застосування необхідних змін.

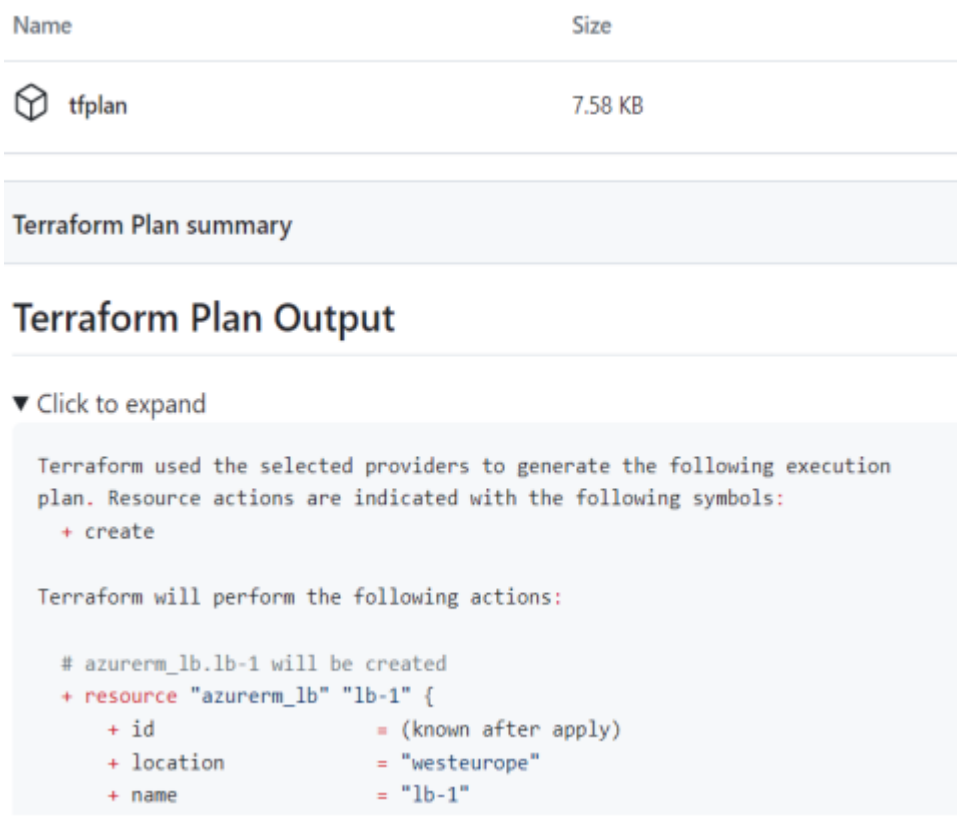


Рис. 3.15 - Додавання артефакту tfplan

Наступним етапом відповідно до архітектури розгортання має бути управління конфігурацією створеної інфраструктури та впровадження самого розгортання на створених машинах. Однак для цього слід створити inventory файл, в якому буде описуватись деталі про всі хости. При розгортанні вручну використовувались скрипти, які витягували IP-адресу кожного хоста та записували її в файл host.txt, проте у випадку, коли розгортання відбуватеться через CI/CD інструмент слід зробити так, щоб цей файл міг змінюватись динамічно, адже він буде знаходитись не локально на машині, а у репозиторії певної системи контролю версій, у даному випадку цією системою буде Github.

Для виконання цієї задачі в workflow створена окрема джоба parsing_ip (див. рис. 3.16).

```
- name: Azure Login
  run: |
    az login --service-principal -u ${{ secrets.AZURE_CLIENT_ID }} -p ${{ secrets.AZURE_CLIENT_SECRET }}
    --tenant ${{ secrets.AZURE_TENANT_ID }}
  env:
    ARM_SUBSCRIPTION_ID: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
  shell: bash

- name: List VMSS Instance Public IPs
  run: |
    az vmss list-instance-public-ips --name example-vmss --resource-group azure-resource_group | python3 parser.py > ip.txt
  working-directory: ${{ github.workspace }}

- name: Append IPs to host.txt
  run: |
    python3 iphost.py >> host.txt
  working-directory: ${{ github.workspace }}

- name: Commit and Push Changes
  run: |
    git config --global user.email "gajdaryura@gmail.com"
    git config --global user.name "Yur4ik1234"
    git add ip.txt host.txt
    git commit -m "Update IPs"
    git push https://$GH_PAT@github.com/Yur4ik1234/auto-cifar10.git HEAD:master
  env:
    GH_PAT: ${{ secrets.GH_PAT }}
  working-directory: ${{ github.workspace }}
```

Рис.3.16 - Процес отримання публічних IP-адрес хостів в workflow

Скрипти, які використовуються для парсингу адрес не змінили лише помінявся процес їхнього використання через специфіку роботи Github Actions.

Для початку потрібно виконати автентифікацію до Microsoft Azure, вона відбувається через вбудовану команду Azure az login. Для успішної автентифікації потрібно використати змінні середовища. Після цього відбувається вже звичний процес парсиингу. Найголовнішою частиною є зберігання змін в файлі host.txt, цей процес відбувається через запушення нового коміту (commit – збереження змін) в репозиторій з вже доданими змінами. Відповідно після цього в файлі host.txt буде відображатись вже адреси нових хостів, які були створені в попередній джобі.

Наступним етапом є впровадження розгортання через застосування ansible playbook. Для цього буде використовуватись лише команда ansible playbook. Для автентифікації буде використовуватись ssh_password, що спростить виконання playbook в Github Actions, однак для того, щоб зробити такий тип автентифікації безпечнішими потрібно додати пароль в сховище секретних паролів. Тобто після

запису паролю в це сховище ніхто не зможе отримати інформацію про вміст даної змінної (див. рис. 3.17).

```yaml
ansible-deploy:
  name: 'Deploy Ansible Playbook'
  runs-on: ubuntu-latest
  needs: [parsing-ip]

  steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Run Ansible playbook
      run: |
        ansible-playbook -i host.txt -u adminuser --extra-vars "ansible_ssh_password=${{ secrets.SSH_PASSWORD }}"
        ansible/playbook.yaml --force
```

Рис. 3.17 - Застосування ansible playbook в workflow

Кожен з цих процесів відбувається поступово, як вже зазначалось вище. Також варто зазначити, що тригером, який запускає весь процес автоматизованого розгортання є виконання git push зі змінами в поточному репозиторії, після цього відбувається workflow, який описаний в файлі main.yml.

Загалом процес успішного виконання всього workflow виглядає наступним чином, що означає, що додаток був успішно розгорнутий на інфраструктурі (див. рис. 3.18).

Після успішного розгортання додатку він є доступним для використання в глобальній мережі через адресу, яка використовувалась для розгортання.

Якщо якісь зміни будуть внесені в код, тоді процес відбудеться ще раз, однак час виконання буде набагато меншим, адже для імплементації цих змін не потрібно буде виконувати завантаження всіх залежностей та побудови інфраструктури з самого початку.

Таким чином, реалізація процесу розгортання відбувалась через виконання процесу CI/CD з використанням інструменту Github Actions. Загалом процес автоматизованого розгортання розділений на 5 залежних один від одного

підпроцесів, кожен з яких виконує частину з розгортання та є початковим етапом для наступного процесу. Спочатку виконуються unit-тести, які перевіряють роботу самого веб-сервісу.

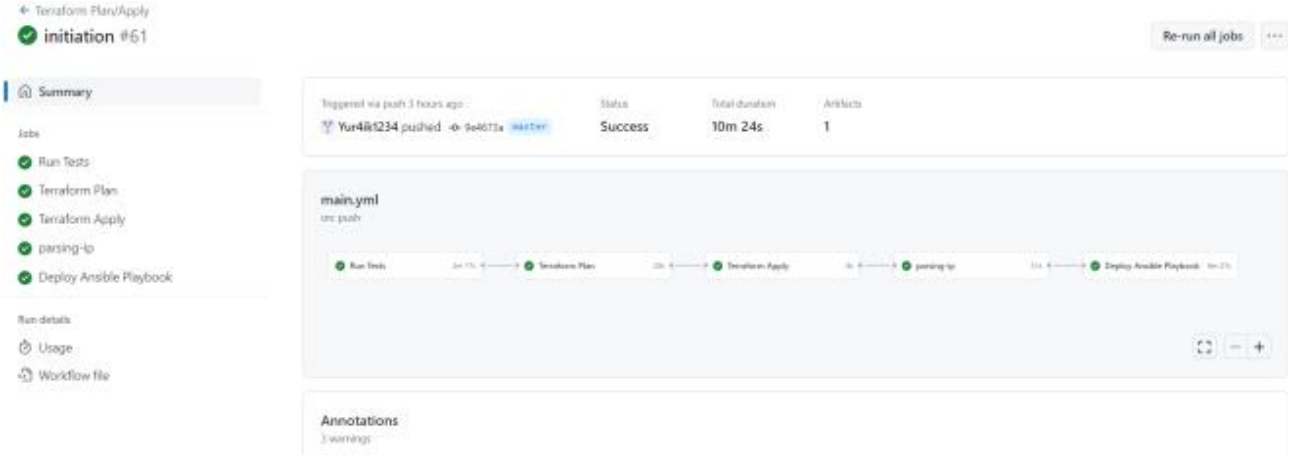


Рис. 3.18 – Підтвердження успішного виконання workflow

Після виконання тестів йде етап створення інфраструктури, на якій відбувається розгортання за допомогою Terraform прописаної інфраструктури в середовищі хмарного провайдера Microsoft Azure. Маючи готову інфраструктуру можна розпочинати розгортання самого сервісу за допомогою Ansible, однак проміжним кроком між цими двома підпроцесами є отримання публічної IP-адреси всіх хостів, на яких буде відбуватись розгортання, а вже після цього буде відбуватись розгортання самого додатку. Також варто зазначити, що вищеописаний процес розгортання не потребує запуску вручну, всі ці етапи відбуваються автоматично внаслідок реакції на зміну вихідного коду програми, що робить процес розгортання нової версії додатку доволі швидким та якісним.

3.3. Оцінка результатів автоматизованого розгортання веб-додатку з класифікацією зображень

Після завершення розгортання додатку можна перевірити як працює функціональність та чи відповідають результати класифікації зображень

ефективності моделі, яка отримана шляхом навчання на даних для тренування в датасеті Cifar-10, де є 500000 зображень.

Спочатку потрібно відкрити сам додаток, перейшовши на публічну IP-адресу. Інтерфейс розгорнутого додатку має наступний вигляд (див. рис. 3.19).

Рис. 3.19 - Стартова сторінка розгорнутого додатку

Тут є можливість завантажувати файли з локального пристрою, а також використовувати посилання на зображення. Для того, щоб використати перший метод потрібно просто завантажити з локального пристрою будь-яке зображення у форматі png, jpeg, jpg та отримати результати класифікації цього зображення відносно десяти класів (див. рис. 3.20).

Можна побачити, що зображення відповідає вимогам, які потрібні для класифікації. Але результати, на перший погляд, є доволі неточними, адже це зображення жаби і додаток мав би показати стовідсотковий результат, або близький

до того саме класу “жаба” в датасеті. Однак, варто зазначити, що ці результати відповідають нейронній моделі, яка використовується в додатку, адже максимальна точність класифікації, яку ця модель може показати складає 87 відсотків, тобто незважаючи на високу точність передбачення ця модель все одно має близько 13 відсоткову похибку, що впливає на кінцеві результати.

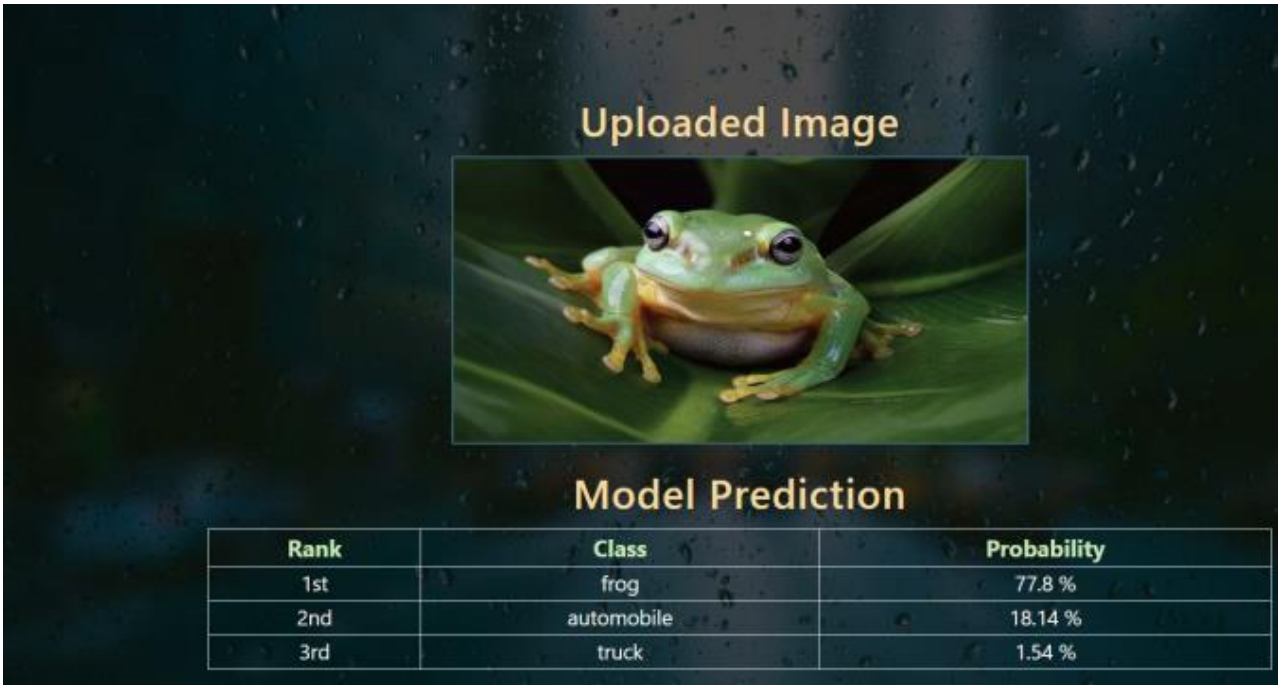


Рис. 3.20 – Результат роботи розробленого додатку при завантаженні зображення.

Тепер зробимо перевірку для класифікації зображення через посилання. Для цього використаємо зображення, на якому будуть зображені об'єкти із двох класів для того, щоб зрозуміти як модель розпізнає два різних об'єкти та якому вона надасть перевагу (див. рис. 3.21).

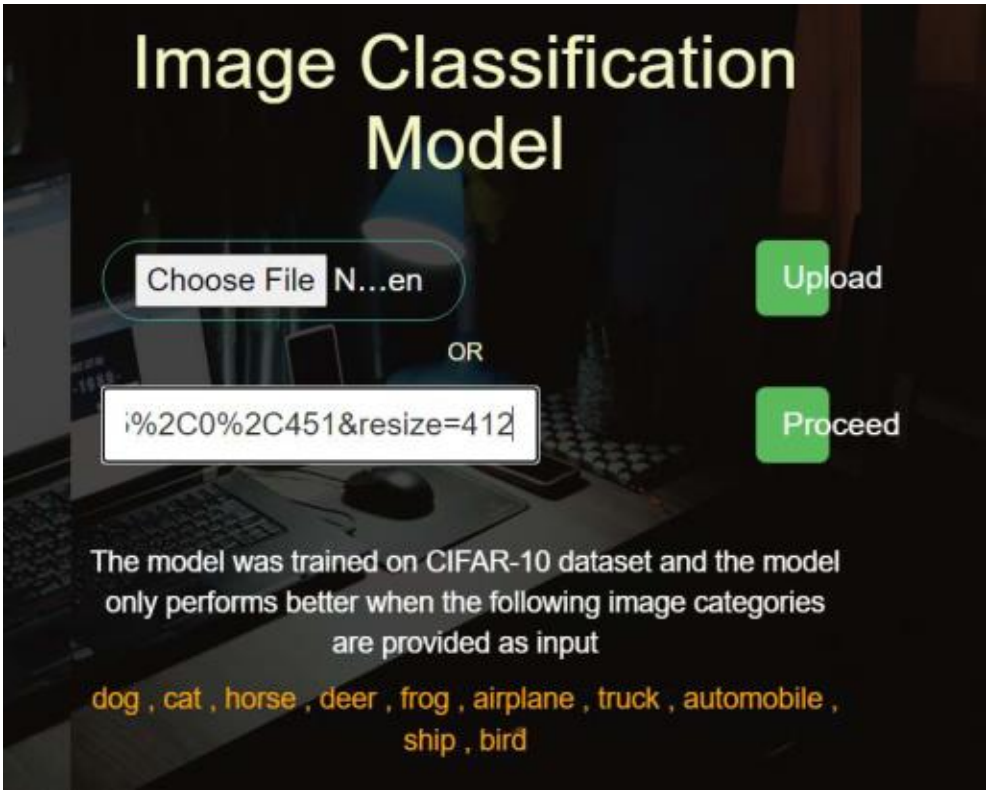


Рис. 3.21 - Використання функції класифікації зображень через посилання

Після натискання кнопки “Proceed” отримаємо наступний результат класифікації, як представлено на рис. 3.22.

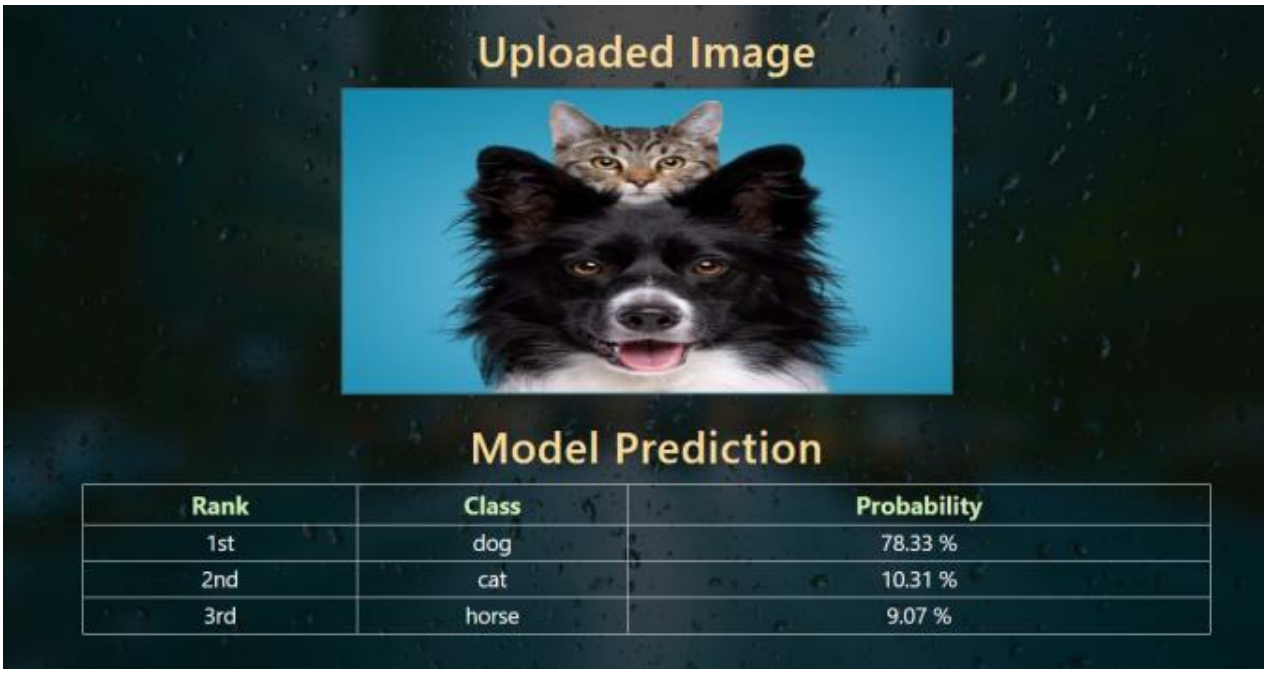


Рис. 3.22 - Результат роботи додатку через використання посилань на фото

Як видно з результатів, клас “Собака” має найбільший відсоток схожості. Це пояснюється розміщенням собаки на зображені, адже він є на передньому плані, що впливає на якість класифікації. Також можна дійти висновку, що дана модель добре визначає лише перший клас, а от із класом “Кіт” виникли проблеми через те, що модель спочатку сфокусувала свою увагу на найбільш помітних рисах, які є на зображені, а це риси собаки, а вже потім визначила інші частини зображення. Можна сказати, що для успішного використання цієї моделі необхідно, щоб були чітко видні основні риси кожного з класів, на основі яких вже приймає рішення, до якого класу слід класифікувати зображення.

З попередніх рисунків видно, що веб-сервіс видає тільки перші 3 класи із датасету, які згідно з моделлю є найбільш схожими до зображення, яке використовується для класифікації в сервісі. Однак для наочності таблицю результатів можна збільшити, для цього потрібно лише змінити певні параметри в програмі та додати відповідні зміни в html файл, змінивши таблицю. Після цих змін оновлений додаток матиме наступний вигляд (див. рис. 3.23).

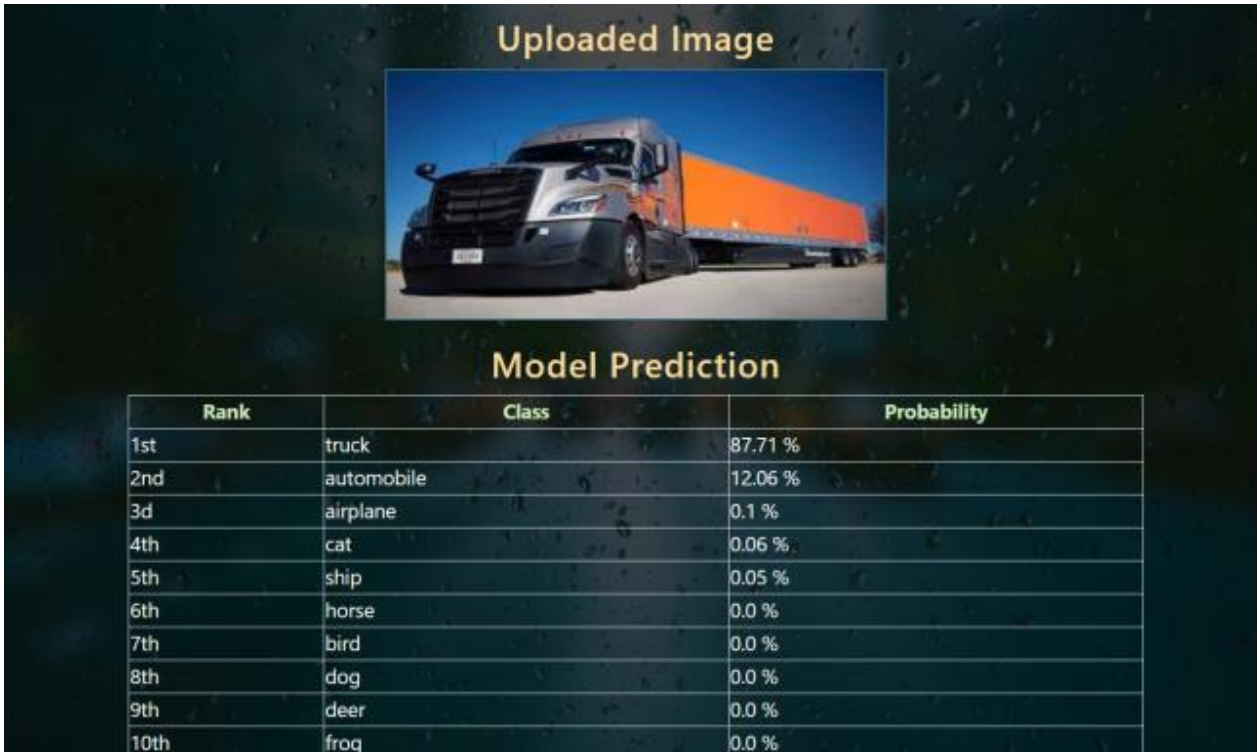
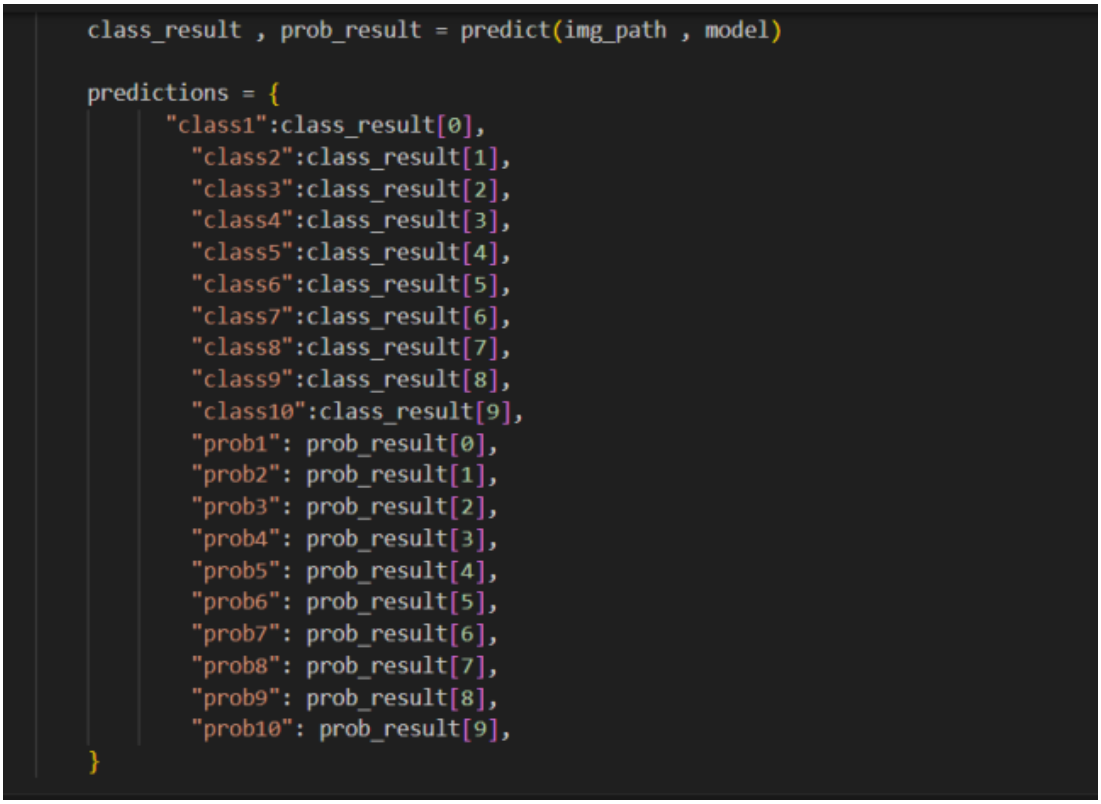


Рис. 3.23 - Повна таблиця результатів класифікації в додатку

Як видно останні чотири позиції взагалі не набрали жодного відсотка, що говорить про те, що згідно з моделлю дані класи не є схожими на запропоноване зображення. Також варто відзначити, що клас автомобіль займає другу сходинку, що є очікуваним через певну схожість із класом вантажівка. Це говорить про доволі ефективну роботу моделі, яка використовується в веб-сервісі.

Для впровадження даних змін необхідно в коді самої програми збільшити кількість класів класифікації з трьох до десяти (див. рис. 3.24).

Дана частина програми присутня в двох місцях, а саме при успішній класифікації за допомогою функції із завантаженням зображення із локального пристрою, а також через класифікацію зображення за допомогою посилання на саме зображення. Для двох цих функціональних можливостей визначення типу зображення використовується лише одна статична сторінка success.html, яка з'являєть лише у випадку, коли всі умови по типу зображення чи посилання відповідають вимогам для можливості отримати результат.



```
class_result , prob_result = predict(img_path , model)

predictions = {
      "class1":class_result[0],
        "class2":class_result[1],
        "class3":class_result[2],
        "class4":class_result[3],
        "class5":class_result[4],
        "class6":class_result[5],
        "class7":class_result[6],
        "class8":class_result[7],
        "class9":class_result[8],
        "class10":class_result[9],
        "prob1": prob_result[0],
        "prob2": prob_result[1],
        "prob3": prob_result[2],
        "prob4": prob_result[3],
        "prob5": prob_result[4],
        "prob6": prob_result[5],
        "prob7": prob_result[6],
        "prob8": prob_result[7],
        "prob9": prob_result[8],
        "prob10": prob_result[9],
}
```

Рис. 3.24. Зміна кількості класів класифікації

Тому для того, щоб застосувати всі ці зміни, які були виконанні для класів класифікації необхідно змінити сам статичний файл, який відповідає за правильне відображення всіх позицій. Як видно з попередніх рисунків відображення результатів відбувається через показ всіх позицій у таблиці, тому якщо ми додали ще сім позицій, тоді необхідно застосувати ці зміни та відобразити їх в таблиці.

Таблиця є частиною HTML-сторінки success.html, тому для правильного відображення результатів необхідно просто додати ще сім нових стрічок для таблиці, які будуть використовувати вже створені в коді програми раніше класи класифікації. Після цих змін стає можливим процес повного відображення результатів класифікації.

Зазначимо, що зміна коду програми та статичних файлів, які використовуються додатком приведе до того, що почне відбуватись процес автоматизованого розгортання сервісу одразу після того як ці зміни будуть збережені та відправлені на віддалений сервер в систему контролю версій та стануть останніми змінами для сервісу, це значно спрощує процес розгортання і дозволяє одразу впровадити зміни і швидко побачити результат.

Також відзначимо, що ефективність нейронної моделі, яка використовується в додатку складає приблизно 87 відсотків. Даний результат є доволі високим, незважаючи на явні мінуси даної моделі, що для її застосування потрібні якісні і чіткі фото та проблеми із класифікацією зображень, на яких є більше, ніж один клас із датасету Cifar-10. Така висока ефективність стала можливою через довгий процес навчання. Ciffar- 10 має 50 тисяч зображень для навчання, що робить процес навчання доволі тривалим.

В машинному навчанні є такий термін *епоха*, він позначає один повний прохід через усі тренувальні дані під час навчання моделі. Під час кожної епохи модель навчається на усіх доступних тренувальних даних, а потім оцінюється на тестовому наборі, щоб визначити її точність та визначити, чи відбувається перенавчання. В Cifar-10 міститься 10 тисяч тестових зображень. Варто відзначити,

що кількість епох сильно впливає на ефективність моделі, якщо вона буде занадто малою, тоді модель буде недонавчатись і це зменшує її ефективність, проте якщо кількість епох є занадто великою тоді модель перенавчається, що теж є проблемою оскільки модель навчається не тільки властивостям тренувального набору даних, але і його шуму чи випадковості.

В моделі, яка використовувалась в додатку, використовується 100 епох, що є найбільш оптимальним рішенням для такого типу моделі. Проте наскільки сильно вплине на роботу додатку та якість класифікації якщо кількість епох зменшити до мінімуму. В такому випадку якість класифікації мала б сильно впасти. Так, після зменшення кількості епох до 2, точність представленої моделі зменшується до 53 відсотків з 87 при 100 епохах.

Аналіз результатів розгортання веб-додатку показує, що розгортання веб-сервісу відбулось успішно та свої функції, а саме класифікацію зображень через завантаження із локального джерела та класифікацію з використанням посилання на зображення, яке міститься в глобальній мережі, веб-сервіс виконує відповідно до визначених вимог.

ВИСНОВКИ

В роботі розроблено підхід до процесу автоматизації розгортання веб-додатку на основі описаної архітектури. Реалізація розгортання додатку з розпізнаванням зображень відбувається через виконання процесу CI/CD з використанням інструменту Github Actions, який запускає розгортання внаслідок реакції на зміну коду веб сервісу в системі контролю версій. Для побудови інфраструктури розгортання використовувались ресурси хмарного провайдера, що забезпечило гнучкість та швидкість розгортання, які є необхідними для додатків, які застосовують в своїй роботі моделі нейронних мереж.

Отримані експериментальні дані показують, що процес розгортання пройшов успішно, адже після його завершення веб сервіс став доступним у глобальній мережі та виконує всі свої функції, а саме класифікацію зображень.

Результати розгортання також показують наскільки важливим є час навчання моделі, адже при різному періоді навчання веб сервіс працював по різному. Точність класифікації зображень для випадку із довшим періодом навчання є більшою, ніж із меншим часом навчання.

Ефективність обраного підходу розгортання підтверджується на практиці адже виконуються всі задачі, які постають в процесі автоматизації розгортання. Даний метод з використанням CI/CD інструментів та підходу Infrastructure as Code (IaC) для побудови інфраструктури дає багато можливостей впливати на процес розгортання, зокрема впроваджувати швидке масштабування, що може бути критично важливо для сервісів, клієнтська база, яких швидко зростає.

Варто відзначити, що розроблений в роботі підхід хоч і виконує поставлені задачі, але для використання його у великих проєктах слід також налаштувати моніторингові системи, які будуть збирати метрики про стан всієї системи, а для малих проєктів можна налаштувати збір метрик через Terraform.

Такий підхід був частково представлений в даній роботі, але він потребуватиме більш детального аналізу, який буде враховувати потреби та особливості системи, яка розгортається.

В роботі здійснено економічний аналіз двох методів розгортання веб-додатків, а саме розгортання із використанням інфраструктури побудованої на ресурсах хмарного провайдера, а також розгортання із використанням інфраструктури на основі фізичного обладнання. Показано, що в короткотривалій перспективі використання ресурсів хмарного провайдера значно зменшує величину капітальних витрат, що дає високу гнучкість в процесі розробки, адже підхід Pay-as-you-go, дозволяє не переплачувати за ресурси, які не використовуються.

Розраховані терміни окупності для двох типів розгортання є однаковими, однак, ціна на той самий продукт є значно вищою для методу із використанням фізичного обладнання. Тому використання хмарних сервісів може забезпечити набагато більшу клієнтську базу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. The essential deployment metamodel: a systematic review of deployment automation technologies, Michael Wursterl, Uwe Breitenbucher Michael Falkenthal1,Christoph Krieger, Frank Leymann,2019

2. Comparison of Approaches to Service Deployment, Vanish Talwar, Qinyi Wu, Calton Pu, Wenchang Yanf, Gueyoung Jung, Dejan Milojicic HP Labs, Georgia Tech 2005

3. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=04d0b695 2a4 f0c7203577afc9476c2fcab2cba06 IV ANN METHODOLOGY

4. https://www.ibm.com/topics/neural-networks What is a neural network?

5. https://www.ibm.com/topics/neural-networks History of neural networks

6. https://avada-media.ua/en/services/neyronnyye-seti/

7. https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural- networks/

8. https://www.xenonstack.com/blog/deep-learning-vs-ml-vs-neural-network

9. https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3- types-of-neural-networks-in-deep-learning/

10. [https://www.researchgate.net/figure/Comparison-between-artificial-neural-network-ANN-and-recurrent-neural-network-RNN_fig5_344946727

11. https://www.ibm.com/topics/convolutional-neural-networks Types of convolutional neural networks

12. N. Strisciuglio, M. Lopez-Antequera, N.Petkov. A Push-Pull Layer Improves Robustness of Convolutional Neural Networks 4.1:стор 3. 2019

13. https://www.upgrad.com/blog/basic-cnn-architecture/ LeNet-5 CNN Architecture

14. https://medium.datadriveninvestor.com/five-powerful-cnn-architectures-b939c9ddd57b

15. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural- networks-the-eli5-way-3bd2b1164a53

16. https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow- a3f5d6746697

17. https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural- network

18. https://www.upgrad.com/blog/basic-cnn-architecture/ Basic Architecture

19. https://rstudio-conf-2020.github.io/dl-keras-tf/04-computer-vision-cnns.html#14

20. https://medium.com/@princekrampah/introduction-to-convolutional-neural- networks-13b3965e1b8c

21. https://www.cs.toronto.edu/~kriz/cifar.html#:~:text=The%20CIFAR%2D10%20 dataset,batch%2C%20each%20with%2010000%20images

22. https://www.upgrad.com/blog/introduction-to-deep-learning-neural-networks- with-keras/

23. https://blog.inedo.com/devops/manual-deployment-disasters/

24. https://docs.docker.com/get-started/overview/

25. https://www.techtarget.com/searchitoperations/definition/Docker

26. https://www.microfocus.com/documentation/enterprise-developer/ed40pu5/ETS- help/GUID-F5BDACC7-6F0E-4EBB-9F62-E0046D8CCF1B.html

27. https://www.redhat.com/en/topics/devops/what-is-ci-cd

28. https: //uk.wikipedia.org/wiki/Неперервна_інтеграція

29. https: //uk.wikipedia.org/wiki/Неперервна_доставка

30. http://elartu.tntu.edu.ua/bitstream/lib/34296/2/VIII_NTK_2020_Zakharkiv_N_M -Problems_of_continuous_144.pdf.

31. https://www.jetbrains.com/teamcity/ci-cd-guide/benefits-of-ci-cd/

32. https://www.scaleyourapp.com/what-is-on-premises-or-on-prem-everything-you- should-know/