

Designing and Implementing a User-Friendly Plant Disease Classification System in the AWS Ecosystem: Insights and Practical Approaches

Module Assignment for

CS5024 - Theory and Practice of Advanced AI Ecosystems

Student Name: **Daniel Maguire**

Student ID: **23222425**

Revision Timestamp: 02/05/2024 22:56:02

Abstract

In this project I have developed an end-to-end system for classifying various plant diseases using Amazon Web Services (AWS). AWS was chosen as the ecosystem of choice as it is widely supported and is used to solve problems at scale in industry. Within the system a user can upload images of plants through a web interface, which are then processed and analysed using a machine learning model deployed on AWS SageMaker which attempts to determine the plant disease. The results of the classification along with confidence scores are saved in Amazon DynamoDB.

I have taken advantage of S3 for file storage (image storage, model backup and storage), EC2 for hosting and preprocessing, SageMaker for model deployment and DynamoDB for data storage. This allowed me to build a modular and scalable infrastructure for the plant disease classification app. The web application which the user will interact with was developed using the Flask framework which integrates nicely with the AWS services used. The plant disease classification model was trained on a dataset of plant images with various diseases and demonstrates high accuracy in identifying various diseases for the trained plant species.

Contents

Abstract.....	2
Introduction	3
AI Ecosystem Architecture Used	4
Web Interface (EC2 Instance).....	4
Image Storage (S3):	5
Image Preprocessing (EC2: Flask Application):.....	5
Model Inference (Flask Application → SageMaker Endpoint):.....	5
Result Storage (DynamoDB):.....	6
Model Description	6
The Deployment Process	7
Performance Results	8
Scalability Considerations	9
Data Storage Scalability.....	9
Compute Scalability	9
Model Inference Scalability.....	10
Database Scalability	10
Conclusion	10
References	10

Introduction

The main goal of this project is to design a user-friendly web platform where an end user can upload plant images and receive automated disease classification. The system relies on a trained machine learning model deployed on AWS SageMaker, which is a managed platform for creating, training and deploying machine learning models at scale.

AWS offers a range of cloud services that support the creation, deployment and expansion of machine learning applications. By harnessing the capabilities of AWS we can develop a plant disease classification application which can handle large amounts of plant images and providing accurate classifications, in real time.

Various AWS services are integrated into the plant disease classification system to allow for optimal performance. The web application, constructed using the Flask framework manages image preprocessing tasks within its codebase. This approach allows for flexibility and customisation in the process to prepare images for input into the machine learning model. The Flask web app offers a user interface to upload images and view the results of their classification. It smoothly integrates with AWS services providing an end to end user journey. Amazon SageMaker oversees deploying and scaling the trained model to efficiently analyse images. The classified results along with confidence scores are stored in Amazon DynamoDB, a scalable NoSQL database. Amazon S3 is utilised for storing plant uploaded images as well as model backups.

The importance of this process lies in its ability to simplify and automate the plant disease classification process. By utilising machine learning within the AWS environment, we can offer a tool for farmers, botanists, plant enthusiasts and researchers to quickly identify diseases of various common crops with considerable accuracy. This could have profound impact in areas like agriculture, where precise plant disease identification is essential for managing crops and controlling diseases as well as in biodiversity preservation efforts where swift identification of plant diseases can support monitoring and protecting botanically weak areas.

The efficiency and ease of utilising the AWS cloud computing platform for deploying machine learning applications at scale are also all key considerations when deciding the hosting platform of such a system. The scalability and dependability provided by AWS make it an excellent option for constructing advanced cloud hosted platforms at scale.

We will explore the intricacies of the system architecture, data storage, image preprocessing, model training and deployment processes, as well as web app development. We will examine the architecture of the chosen model, and will also discuss the scalability considerations and how we can improve and expand the plant disease classification system going forward.

AI Ecosystem Architecture Used

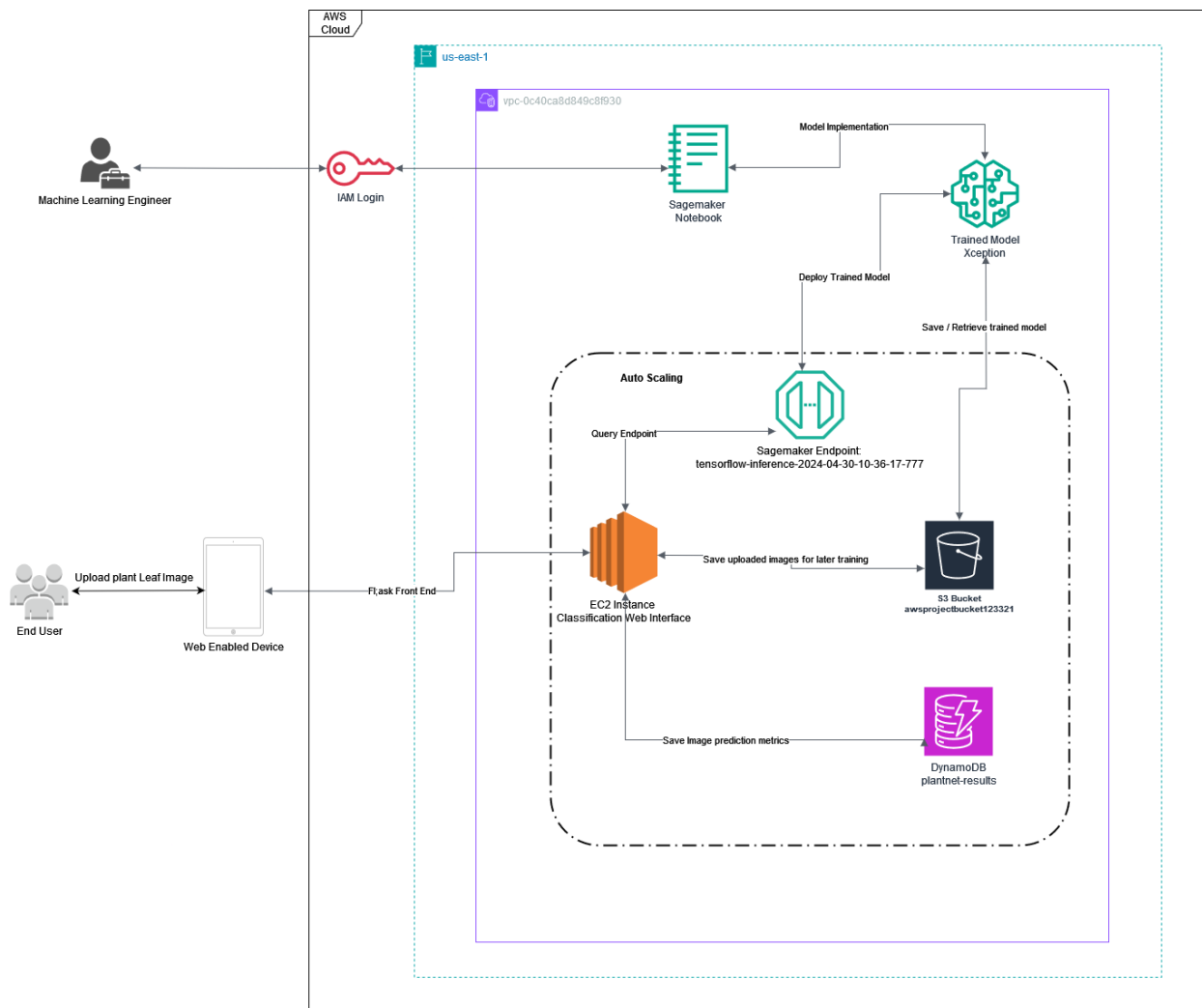


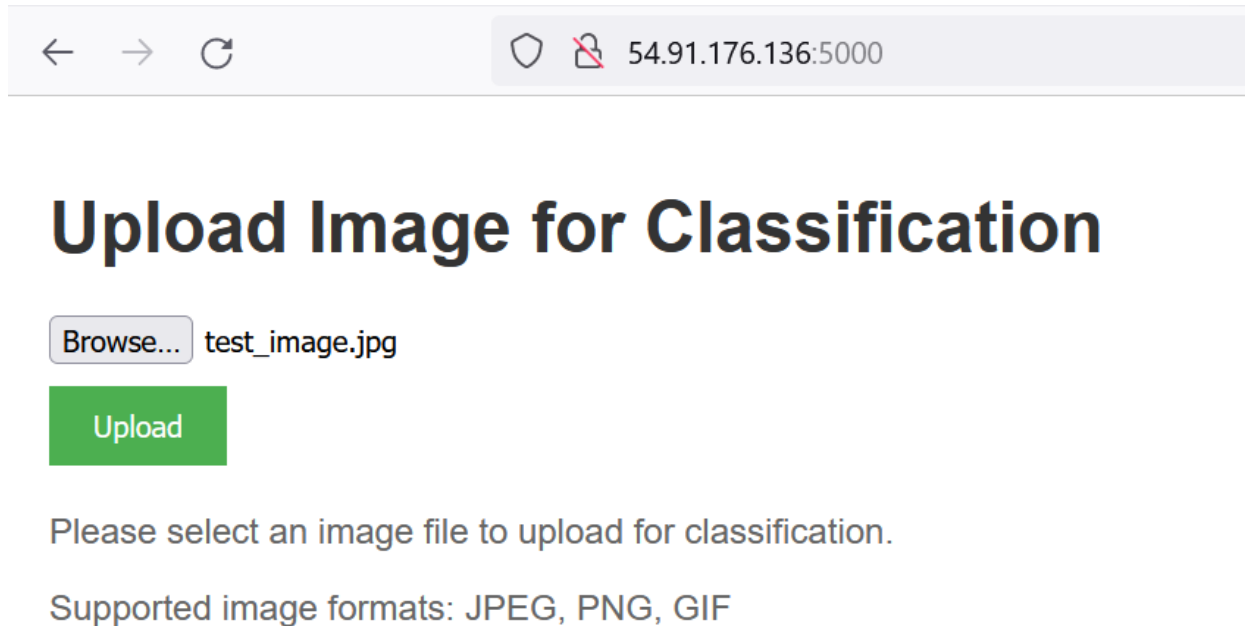
Figure 1: Architecture used to create Plant Classification Application

The plant disease classification application uses various components from the AWS ecosystem to create a comprehensive and scalable architecture. The architecture is designed to efficiently handle the upload, preprocessing, classification, and storage of plant images, uploaded by the user. Below we will look at the key components of the system architecture used in this project.

Web Interface (EC2 Instance)

The web application is hosted on an EC2 instance and acts as the user interface for the plant disease classification system. It is built using Flask, which allows developers to easily design and host web applications using Python. The Flask application handles image uploads and the display of classification results. It integrates with other AWS services to facilitate image storage (S3), preprocessing (EC2), and model inference (Sagemaker Endpoint). The user interface itself of the web application is built using HTML and CSS. Flask renders the HTML templates and populates them with

the classification results. See Figure 2 below for a clear understanding of what the UI landing page looks like.



← → ↻ 54.91.176.136:5000

Upload Image for Classification

Browse... test_image.jpg

Upload

Please select an image file to upload for classification.

Supported image formats: JPEG, PNG, GIF

Figure 2: Upload UI

Image Storage (S3):

Amazon S3 serves as our storage solution for the plant images that users upload on the web application. Once an image is uploaded it is securely saved in an S3 bucket offering durability, scalability and security for handling image data (AWS Documentation, 2024). These stored images can be accessed in future by the SageMaker model to improve the accuracy once they are screened by a human to be fit to be included in training data.

Image Preprocessing (EC2: Flask Application):

The Flask application handles image preprocessing tasks. Upon uploading an image, the application adjusts the size of the image to ready it for feeding into the machine learning model. In future I may move this logic to a Lambda function but due to time constraints I chose to include it directly within the Flask application.

Model Inference (Flask Application → SageMaker Endpoint):

When the image has been prepared for classification, the Flask app then invokes the SageMaker endpoint with the processed image as input which is then fed into the machine learning model. SageMaker carries out the analysis using the trained model. It then returns the classification outcome along with a confidence rating. The Flask app receives this analysis and displays it to the user on the results page (See Figure 3).

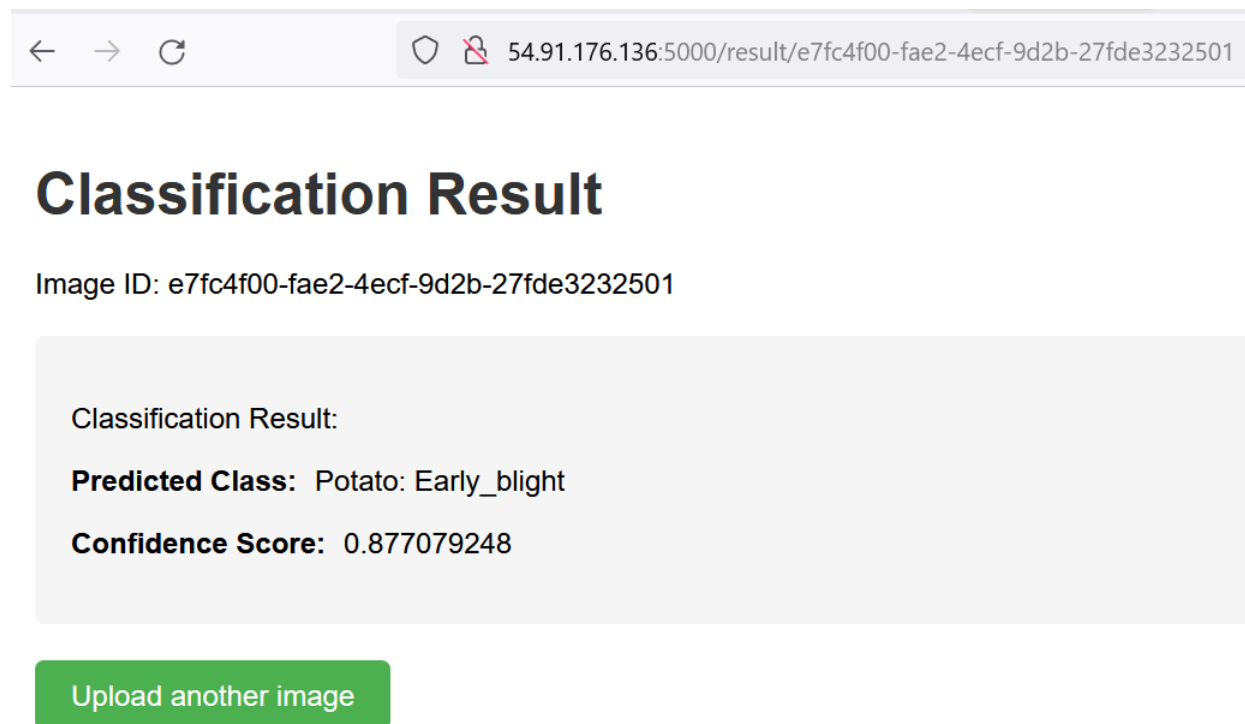


Figure 3: Classification Results UI

Result Storage (DynamoDB):

The classified results, along with the corresponding confidence scores are stored in DynamoDB. I chose DynamoDB as it is fast and scalable. The Flask application interacts with DynamoDB to store classification results. Each classified image is associated with a unique identifier, allowing for easy retrieval of the results. This stored metadata, along with the actual stored image in S3 can be used to improve the model accuracy in future.

Model Description

In the plant classification system, we employed a learning model based on the Xception architecture (Chollet, 2017). Xception is a type of convolution neural network (CNN) known for its high performance in tasks related to image classification. The model was trained on the Plant Village dataset which consists of various fruits and crops with various common diseases that affect them. Figure 4 shows a small sample of the type of images in the PlantVillage dataset, as well as their respective labels. To enhance the model's ability to generalise, the dataset underwent preprocessing and augmentation.

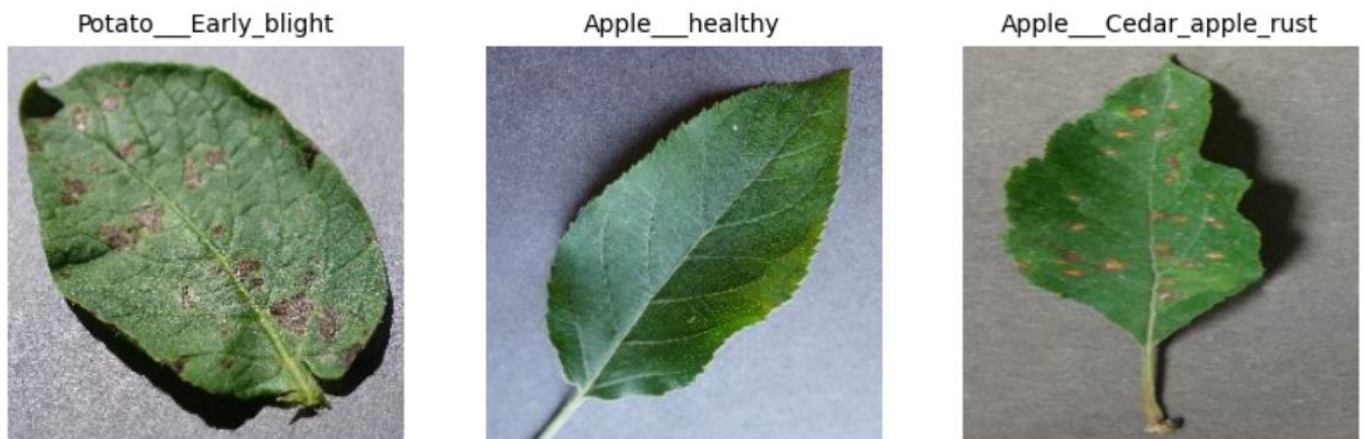


Figure 4: Examples from the PlantVillage dataset

Implementation of the Xception model was carried out using the TensorFlow and Keras libraries in Python. The architecture of the Xception model incorporates depthwise convolutions, which aid in reducing parameters and computational complexity when compared to standard convolution methods. Additionally residual connections and a global average pooling layer were included in conjunction with a fully connected layer to improve classification.

During training an Adam optimiser with a learning rate of 0.001 and a batch size of 32 was used. The optimisation objective employed was entropy loss function. Training spanned 3 epochs. The model reached a validation accuracy of 95%, demonstrating that the model can indeed classify diseases correctly with considerable accuracy (see Figure 5). The Xception model that was trained was prepared as a SageMaker model, uploaded to S3 and set up as an endpoint. This endpoint offers an API that the Flask web app calls upon to classify plant images.

The Deployment Process

Packaging the trained model: The trained Xception model, along with its dependencies and inference code, was packaged into a Docker container compatible with SageMaker.

Creating a SageMaker model: The packaged model was uploaded to an S3 bucket and registered as a SageMaker model.

Configuring the endpoint: An endpoint configuration was created, specifying the model and instance type.

Deploying the endpoint: The endpoint was deployed using the configured model and made available for inference requests. Figure 5 shows the deployed custom sagemaker endpoint ready to be used.



	tensorflow- inference-2024-04-30-10-36-17-777	arn:aws:sagemaker:us- east-1:654654338622:endpoint/tensorflow- inference-2024-04-30-10-36-17-777	4/30/2024, 11:36:18 AM	 InService	4/30/2024, 11:39:47 AM
---	--	--	---------------------------	---	---------------------------

Figure 5: Deployed Custom Sagemaker Endpoint

The plant disease classification model achieved a final validation accuracy of 0.9470 after 3 epochs. Figure 5 goes into more granular detail, along with the loss over multiple epochs.

Figure 5: Model performance over 3 epochs

[illegible]

Module Assignment for CS5024 - Theory and Practice of Advanced AI Ecosystems - 02/05/2024 22:56

Figure 7 displays the training and validation accuracy curves over the epochs. The model converges well, with the validation accuracy coming close to the training accuracy and reaching a stable plateau.

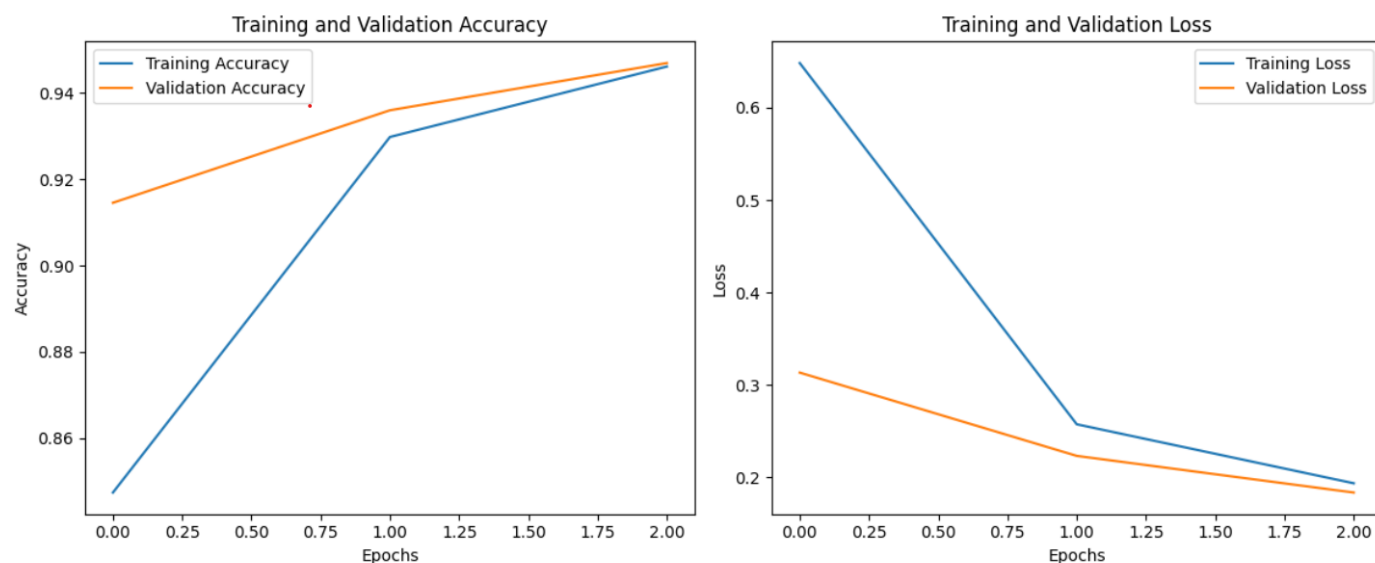


Figure 7: Model accuracy (left) and loss (right)

These metrics demonstrate a well-trained model and the effectiveness of Xception in accurately classifying plant disease images across a range of different species and diseases. There may be some misclassifications but I believe that more training data will help improve the model accuracy further.

Scalability Considerations

Data Storage Scalability

The plant disease classification system relies on Amazon S3 for storing uploaded plant images. As the number of users and uploaded images grows, the storage requirements will increase. S3 is designed to scale seamlessly, providing virtually unlimited storage capacity (Patrick Denny, 2024). It automatically distributes data across multiple servers and data centres, ensuring high availability and durability. To optimise costs and performance, S3 lifecycle policies can be implemented to transition older data to cheaper storage tiers or automatically delete unnecessary (AWS Documentation, 2024).

Compute Scalability

The Flask web application and the image preprocessing tasks are executed on compute resources. As the number of concurrent users and requests increases, the compute requirements will also grow. AWS provides several options for scaling compute resources. One approach is to use Amazon EC2 Auto Scaling, which automatically adjusts the number of EC2 instances based on demand (Patrick Denny, 2024). This ensures that the system can handle traffic spikes and maintain responsiveness. Additionally, containerisation technologies like Docker can be leveraged to package the application and its dependencies, enabling easy deployment and scaling across multiple instances.

Model Inference Scalability

The plant classification system uses AWS SageMaker to deploy the trained machine learning model for inference. SageMaker provides built-in scalability features to handle increased inference requests. The deployed model endpoint can be configured with auto-scaling policies to automatically adjust the number of instances based on the incoming traffic. SageMaker also supports deploying models across multiple instances and automatically distributes the inference requests among them (AWS Documentation, 2024). This ensures that the system can handle a high volume of inference requests without performance degradation.

Database Scalability

The classification results and metadata are stored in Amazon DynamoDB. DynamoDB is designed to handle massive amounts of data and high-velocity reads and writes. It automatically distributes data across multiple servers and scales horizontally to accommodate increased traffic. DynamoDB also offers features like auto-scaling and on-demand capacity, which dynamically adjust the read and write capacity based on the workload (Patrick Denny, 2024). This ensures that the database can handle a growing number of records and maintain consistent performance.

Conclusion

Throughout the project various difficulties were faced when preparing the custom Xception model for deployment. Yet through utilising the features of the AWS environment and adhering to the recommended methods I managed to overcome these obstacles and create a functional plant disease categorisation system. The system was created considering scalability, security and ease of use for the end user. Its modular layout enables the option to extend the architecture easily via other AWS services or external platforms in future. Also, in future, the application and model have the potential to be extended and further applied to various domains, such as agriculture, botany, and environmental conservation.

References

- AWS Documentation. (2024). *Amazon S3*. Retrieved from AWS Documentation:
<https://aws.amazon.com/s3/>
- AWS Documentation. (2024). *Amazon SageMaker - Machine Learning Platform*. Retrieved from AWS Documentation: <https://aws.amazon.com/sagemaker/>
- Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1251-1258.
- Patrick Denny. (2024). *Module 10: Automatic Scaling and Monitoring - Friday Week 8 Material*. Retrieved from CS5024 - Theory and Practice of Advanced AI Ecosystems:
<https://learn.ul.ie/d2l/le/lessons/17937/topics/650610>

Designing and Implementing a User-Friendly Plant Disease Classification System in the AWS Ecosystem: Insights and Practical Approaches

Patrick Denny. (2024). *Module 7: Storage - Friday Week 6 Material*. Retrieved from CS5024 - Theory and Practice of Advanced AI Ecosystems:
<https://learn.ul.ie/d2l/le/lessons/17937/topics/646049>

Patrick Denny. (2024). *Module 8: Databases - Tuesday Week 7 Material*. Retrieved from CS5024 - Theory and Practice of Advanced AI Ecosystems:
<https://learn.ul.ie/d2l/le/lessons/17937/topics/646579>