Capstone Project Machine Learning Project Documentation

The Data-Bears Team Members:

1. *Ilyas Nayle*
2. *KAOUBARA DJONG-MON*
3. *DAGIMAWI MOGES WAKUMO*

EcoLens: A Visionary Solution for a Plastic-Free World

Deployment

Overview

The deployment phase of a machine learning project involves making a trained model available for use in the real-world. This process is critical as it enables the model to deliver practical value by making predictions on new or unseen data**.** Here we have developed a deep learning CNN model using TensorFlow for classifying that predicts where an object is plastic or non-plastic.

Model Serialization

Model Serialization is the process of saving a trained model so that it can be reused or deployed later without retraining. It is very critical for deploying models in production environments efficiently. The following explains how we did and what format we used for model serialization.

Process of Serialization:

- Choosing the serialization format: HDF5 Format (.h5): This is a common format used in TensorFlow and Keras. It is very useful because it saves the model architecture, weights, and optimizer configuration into a single file.
- Example:
    - # Define the path to save the model model_save_path = 'C:/Users/----/capstone/saved_models/plastic_classification_model.h5' # Create the directory if it doesn't exist import os os.makedirs(os.path.dirname(model_save_path), exist_ok=True) #

Save the model in HDF5 format model.save(model_save_path)
print("Model saved successfully.")

- In this example:
  - model_save_path specifies the path where the model will be saved.
  - os.makedirs ensures that the directory structure leading to model_save_path exists.
  - model.save(model_save_path) serializes the model in HDF5 format and saves it to the specified path.

Serialization in HDF5 (.h5) format is advantageous because it encapsulates all necessary components of the model into a single file, making it easy to load and deploy the model in different environments. This format is widely supported and efficient for storing deep learning models trained using frameworks like TensorFlow and Keras.

Model Serving

After training the model, our decision to save and deploy the model on-premises rather than the cloud services was decided by several strategic considerations which explained below:

1. Hardware and Resources Availability: We assessed our on-premises infrastructure to ensure it meets the computational demands of our trained machine learning model
2. Scalability: We carefully planned and configured our infrastructure to accommodate peak loads and ensure scalability.
3. Operational Efficiency: By deploying on-premises, we maintain operational control and flexibility over model updates, maintenance, and integration with existing IT systems
4. API Integration with Flask: We decided to integrate our serialized model into Flask to expose it via HTTP endpoints. Detailed explanation is given in the API Integration section.

API Integration

As part of our plan for deploying the model, we aim to integrate it into an API to facilitate easy access for applications and services within our organization.

Here’s our planned approach for API integration:

1. API Endpoint: We will expose the model through an HTTP endpoint /predict, which will accept POST requests for making predictions. This endpoint will be accessible within our organization's network.
2. Input format: API request will expect JSON payloads containing base64-encoded image data, which will for flexible and efficient transmission of image data to the model for classification.\

3. Response format: The API will return predictions in JSON format which will include the classified class and associated result.
4. Implementation strategy: We plan to implement the API using Flask, which will handle incoming HTTP requests, preprocess the data, invoke the model for prediction and format the response back to the client.
5. Deployment and Usage: Upon completion, the API will be deployed within our organization's infrastructure. Applications and services can then interact with the API by sending POST requests containing base64-encoded images. The API will process these requests, perform predictions using the model, and respond with prediction results, enabling seamless integration of machine learning capabilities into our application

This approach to API integration outlines our strategy for making our trained model accessible and scalable within our organization.

Security Considerations

Given that our model is deployed on-premises, it inherently benefits from certain security advantages compared to the cloud-based deployments. However, it is still important to implement robust security measures to safeguard data and ensure the integrity of the system. The following are what we consider for the security.

1. Network Security.
2. Authentication and Authorization
3. Data Encryption
4. Regular security updates

Monitoring and Logging

Monitoring and logging are essential components of maintaining and optimizing the performance, reliability, and security of our deployed model. Here's we will implement our on-premises deployment:

1. Performance Metrics: Monitoring inference time, resource utilization and throughput to optimize model performance and ensure efficient resource allocation.
2. Logging Strategy: Implementing logging for API requests, prediction results, and errors to track system interactions.
3. Maintenance and Optimization: Regularly review and optimize monitoring configurations, alert thresholds, and logging practices based on operational insights and evolving requirements. This ensures that our deployed model maintains optimal performance, reliability, and security over time.