Deploying a Deep Learning Model with Flask and Docker

Siddharth Mehta (UBIT: smehta28)
Devanshi Parmar (UBIT: dparmar)

Introduction

Deploying deep learning models as web applications enables easy access and real-world usability for users. This article provides a comprehensive guide on deploying a PyTorch-based image classification model using Flask and Docker. We will explore the step-by-step process of model serialization, setting up a web server with Flask, containerizing the application using Docker, automating deployments with GitHub Actions, and deploying the final application on cloud platforms like Render.

Understanding the Need for Deployment

Deep learning models, when developed, typically remain confined within research environments such as Jupyter Notebooks or local servers. However, to make these models accessible to users, we need to deploy them as web applications. Deploying a model involves:

- Exposing a user-friendly interface for image uploads.
- Running inference on uploaded images.
- Displaying the results with a confidence score.
- Ensuring the application is accessible from anywhere.

By leveraging Flask, a lightweight web framework, and Docker, a containerization tool, we can create a robust and scalable deployment setup.

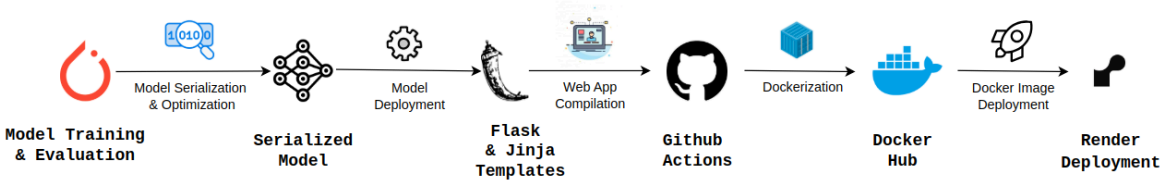
Project Overview

This project focuses on deploying a ResNet18 deep learning model trained to classify images into three categories: Dog, Food, and Vehicle. The model is serialized and integrated with a Flask-based web application that allows users to upload images and receive predictions. The final application is containerized using Docker for seamless deployment.

Key Features:

- Web-based user interface for image classification
- PyTorch ResNet18 model for accurate predictions
- Visualization of class probabilities for better interpretability
- Containerized with Docker for portability
- Automated CI/CD pipeline with GitHub Actions
- Deployment on cloud platforms like Render

Architecture



Steps Involved in Deployment

Step 1: Model Serialization

Before deployment, the deep learning model needs to be converted into a format that can be loaded and executed efficiently. Serialization is the process of saving the model in a compact format. This allows for easier loading and inference without retraining.

PyTorch provides utilities for model serialization, enabling efficient inference in production environments. The serialized model is stored in a dedicated folder for easy retrieval by the web application.

Step 2: Setting Up the Flask Web Application

Flask is a lightweight Python web framework that enables rapid web application development. In this project, Flask is used to create a web interface where users can upload images for classification.

The Flask application handles multiple tasks, including:

- Receiving image uploads from users.
- Preprocessing the images to match the model's expected input format.
- Running inference using the pre-trained deep learning model.
- Displaying classification results along with a probability distribution graph.
- Managing errors and providing a smooth user experience.

Step 3: Containerization with Docker

Docker provides a way to package applications and their dependencies into standardized units called containers. This ensures that the application runs consistently across different environments.

The Docker setup includes:

- Installing required dependencies.
- Copying application files, including the Flask app and serialized model.
- Configuring the runtime environment.
- Exposing necessary ports for web accessibility.

Using Docker, the application can be deployed without worrying about system dependencies, making it highly portable and scalable.

Step 4: Automating Deployments with GitHub Actions

Continuous Integration and Continuous Deployment (CI/CD) pipelines automate the process of building, testing, and deploying applications. GitHub Actions is used in this project to:

- Build the Docker image.
- Push the image to Docker Hub.
- Trigger deployment on a cloud platform using webhooks.

By automating deployments, updates to the model or web application can be seamlessly integrated and deployed without manual intervention.

Step 5: Deploying on a Cloud Platform

For real-world usage, the application is deployed on cloud platforms such as **Render, Fly.io, or Google Cloud Run**. The deployment process involves:

- Connecting the GitHub repository to the cloud platform.
- Configuring environment variables, such as the model file path.
- Enabling automatic deployments for continuous updates.
- Testing the live application to ensure smooth functionality.

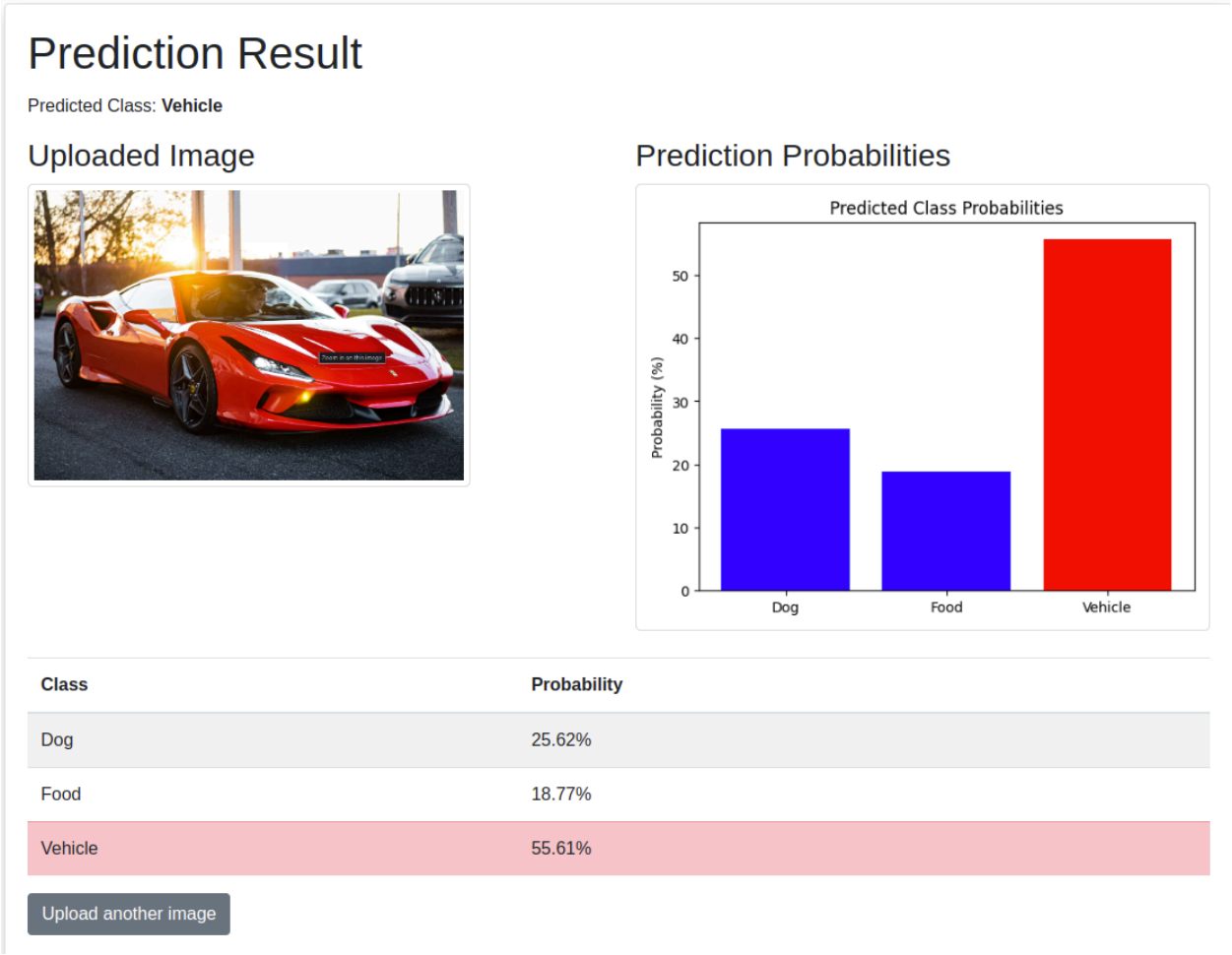
User Experience and Workflow

The final deployed application provides a simple and intuitive workflow for users:

1. **Uploading an Image**: Users can upload an image of an object they want to classify.
2. **Processing the Image**: The image is resized, normalized, and passed through the deep learning model.
3. **Generating Predictions**: The model returns probabilities for each class.

4. **Displaying Results**: The most likely class is highlighted, along with a bar chart showing probability scores.
5. **Repeating the Process**: Users can upload another image for classification.

The web interface is designed with a **clean layout**, displaying both the uploaded image and classification results in an easily understandable manner.



Challenges and Best Practices

Challenges Encountered

While deploying the model, several challenges were addressed:

- **Handling Image Variability**: The model must process various image sizes and formats correctly.
- **Optimizing Model Inference Speed**: Reducing latency ensures faster responses.
- **Ensuring Cross-Platform Compatibility**: Dockerized deployment prevents dependency issues.

- **Managing Deployment Costs**: Using free-tier cloud services helps in cost-effective deployment.

Best Practices for Deployment

- **Use Efficient Model Serialization**: Saves computation time during inference.
- **Implement Proper Error Handling**: Ensures smooth user experience.
- **Automate Deployments**: Reduces manual workload and speeds up updates.
- **Monitor Application Performance**: Helps detect issues and optimize response times.

Conclusion

Deploying deep learning models as web applications enhances their usability by making them accessible to a broader audience. This project demonstrates a complete pipeline for deploying a PyTorch-based image classification model using Flask and Docker, with automated deployments through GitHub Actions.

The combination of Flask for web serving, Docker for containerization, and GitHub Actions for CI/CD creates a scalable, reliable, and easily deployable solution. This setup can be extended further with additional features such as user authentication, API endpoints, or cloud-based storage.

By following the steps outlined in this guide, developers can efficiently deploy deep learning models and integrate them into real-world applications.

✅ Task for Audience

You can try using a different custom trained model to deploy with the pipeline architecture discussed. Reach out to us if you've any questions regarding the project.

References

- **ResNet Paper**: Deep Residual Learning for Image Recognition
- **PyTorch Documentation for ResNet18**: ResNet in torchvision.models
- **PyTorch Model Serialization**: torch.jit documentation
- **Flask Official Documentation**: https://flask.palletsprojects.com/
- **Flask File Upload Guide**: Handling File Uploads
- **Jinja Official Documentation**: https://jinja.palletsprojects.com/
- **Flask & Jinja Integration**: Flask Templates
- **Dynamic HTML Rendering with Jinja**: Jinja Templating Basics
- **Docker Official Documentation**: https://docs.docker.com/
- **Dockerfile Reference**: Dockerfile Guide
- **Docker Hub Guide**: Push Docker Image to Docker Hub

- **Dockerizing Flask Applications**: Containerize Flask App
- **GitHub Actions Documentation**: https://docs.github.com/en/actions
- **Automating Docker Builds with GitHub Actions**: Docker CI/CD Guide
- **GitHub Actions Workflow Syntax**: Workflow Configuration
- **Render Official Documentation**: https://render.com/docs
- **Using Docker on Render**: Deploying Docker Containers
- **Matplotlib for Graph Visualization**: https://matplotlib.org/stable/contents.html
- **Pillow (PIL) for Image Processing**: https://pillow.readthedocs.io/en/stable/