Capstone Machine Learning Project Deployment

Project Title: Plant Disease Detection Using Deep Learning

Team Members

- Dawit Tadesse Hailu
- Chol Moses Malueth
- Fahadi Mugigayi

Deployment

1. Overview

The deployment phase involves making the trained convolutional neural network (CNN) for plant disease detection accessible for practical use. This phase includes model serialization, model serving, API integration, security considerations, and monitoring and logging. The goal is to create a simple and functional deployment for presentation purposes.

2. Model Serialization

The trained model is serialized using Keras to ensure it can be efficiently stored and easily loaded for deployment. The HDF5 format (`.h5`) is used for its compatibility with Keras and efficient storage capabilities. The size of our model amounted to a 2 MB, which is good in our case because it makes the loading process easier.

The model was saved using the model checkpointer:

```
# Define the ModelCheckpoint callback

checkpoint = ModelCheckpoint(filepath='model_checkpoint.h5',
                             save_best_only=True,
                             monitor='val_loss',
                             mode='min')
```

The model will be loaded using keras again right before being served:

```
from tensorflow.keras.models import load_model

model = load_model('plant_disease_model.h5')
```

3. Model Serving

A simple server is set up using Flask, a lightweight WSGI web application framework. This server loads the serialized model and serves it for making predictions. The flask server is hosted locally for purposes of demonstrating but can be hosted on aws or heroku after having proper configuration setup.

```
from flask import Flask, request, jsonify, render_template
```

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import numpy as np
import json
# Load labels from the JSON file
with open('labels.json', 'r') as f:
    labels = json.load(f)
app = Flask(__name__)
model = load_model('model_checkpoint.h5')
@app.route('/', methods=['GET'])
def hello_word():
    return render_template('index.html')
@app.route('/', methods=['POST'])
def predict():
    imagefile= request.files['imagefile']
    image_path = "./images/" + imagefile.filename
    imagefile.save(image_path)
    img = load_img(image_path, target_size=(224, 224))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = img / 255.0
    prediction = model.predict(img)
    predicted_class_index = np.argmax(prediction, axis=1)[0]
    predicted_class_label = labels[str(predicted_class_index)]
    print(prediction)
```

```
    print("Prediction made:", predicted_class_label)

    return render_template('index.html', prediction=predicted_class_label)
```

4. API Integration

The machine learning model is integrated into an API to facilitate easy access. The /predict endpoint accepts image files and returns predictions in JSON format.

Endpoint: /predict

Input Format: Multipart file upload (image)

Response Format: JSON with prediction results

A small frontend is made, which is used to upload the image and get a prediction on it. The following html outlines the simple interface:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutorial</title>
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWl
olflfl" crossorigin="anonymous">
    </head>

    <body>
        <h1 class="text-center">Plant Disease Classifier</h1>

        <form class="p-3 text-center" action='/', method="post" enctype="multipart/form-data">
            <input class="form-control" type="file" name="imagefile" >
            <input class="btn btn-primary mt-3"type="submit" value="Predict Image" >
        </form>

        {% if prediction %}
            <p class="text-center"> Image is a {{prediction}}</p>
        {% endif %}
    </body>
</html>
```

5. Security Considerations

Basic security measures can be implemented to ensure the server is secure. This includes an API key check for authorization. Although we have not tested it out, we have put the blueprint of how this can be achieved.

```
from flask import Flask, request, jsonify
```

```
from functools import wraps
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import numpy as np
app = Flask(__name__)
# Load the model
model = load_model('plant_disease_model.h5')
def require_api_key(func):
    @wraps(func)
    def decorated_function(*args, **kwargs):
        if request.headers.get('API-Key') == 'your_api_key_here':
            return func(*args, **kwargs)
        else:
            return jsonify({'message': 'Invalid API Key'}), 403
    return decorated_function
@app.route('/predict', methods=['POST'])
@require_api_key
def predict():
   imagefile= request.files['imagefile']
    image_path = "./images/" + imagefile.filename
    imagefile.save(image_path)
    img = load_img(image_path, target_size=(224, 224))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = img / 255.0
    prediction = model.predict(img)
```

```
    predicted_class_index = np.argmax(prediction, axis=1)[0]
    predicted_class_label = labels[str(predicted_class_index)]
    print(prediction)
    print("Prediction made:", predicted_class_label)
    return render_template('index.html', prediction=predicted_class_label)
if __name__ == '__main__':
    app.run(debug=True)
```

6. Monitoring and Logging

Logging can be set up using Python's logging module to track requests and errors. This ensures the deployed model's performance is monitored and issues are logged for debugging.

```
import logging
# Set up logging
logging.basicConfig(filename='app.log', level=logging.INFO)


@app.route('/predict', methods=['POST'])
def predict():
    Try:
        image = request.files['file']...
            …
            …
        prediction = model.predict(img)
        logging.info('Prediction made successfully')
        return jsonify({'prediction': str(prediction)})
    except Exception as e:
        logging.error('Error occurred: %s', str(e))
        return jsonify({'error': 'An error occurred'}), 500
```

```
if __name__ == '__main__':

    app.run(debug=True)
```

Metrics Tracked:

- Request success
- Errors

By following some of the above steps, we successfully deployed our plant disease detection model locally, and ensured that it is accessible, secure, and monitored for performance. This documentation serves as our guide to the deployment phase, providing a clear and concise overview of our process towards the presentation.