9/12/2024

# PROGRAMMING ASSIGNMENT-02

## INTELIGENT SYSTEM

ITBIN-2110-0096 – C.Sajantha
INTAKE-10

# Contents

# Image Recognition System Report

## Introduction

This report details the development of an image recognition system using TensorFlow for model training and FastAPI for serving the model via a web application. The project includes data collection, model architecture, training process, evaluation, and deployment strategies.

## Overview

The image recognition system aims to classify images into predefined categories. The system leverages TensorFlow for model training and FastAPI to create a web API for model inference. The deployment of the application is handled using Railway.app, and a CI/CD pipeline is set up for automated testing and deployment.

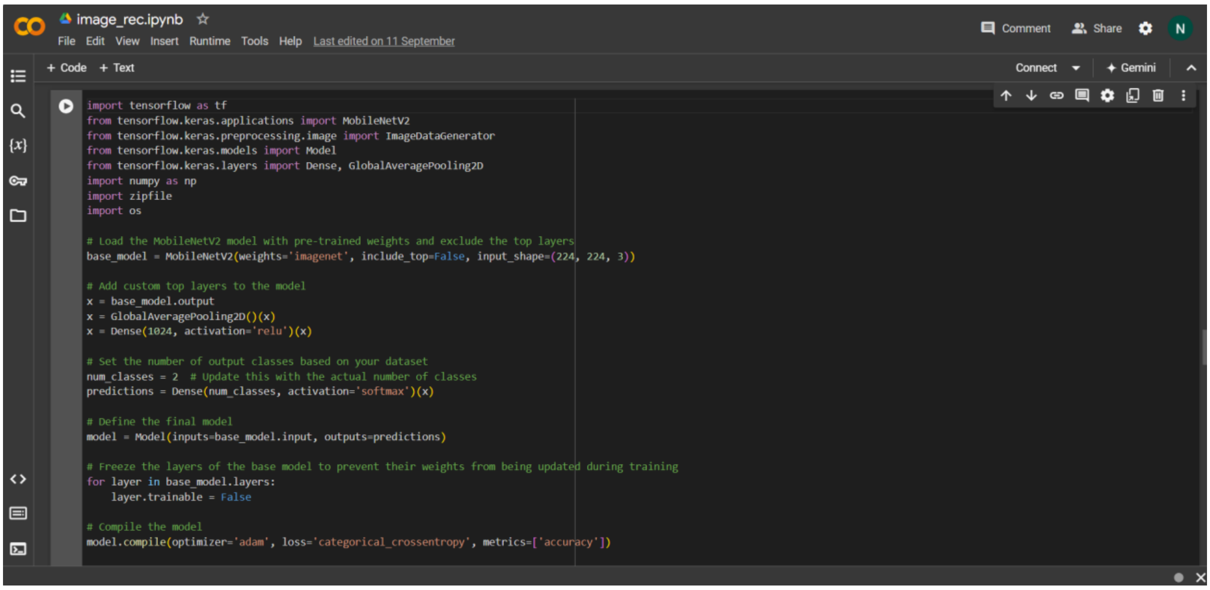
## Model Training

### 3.1 Data Collection

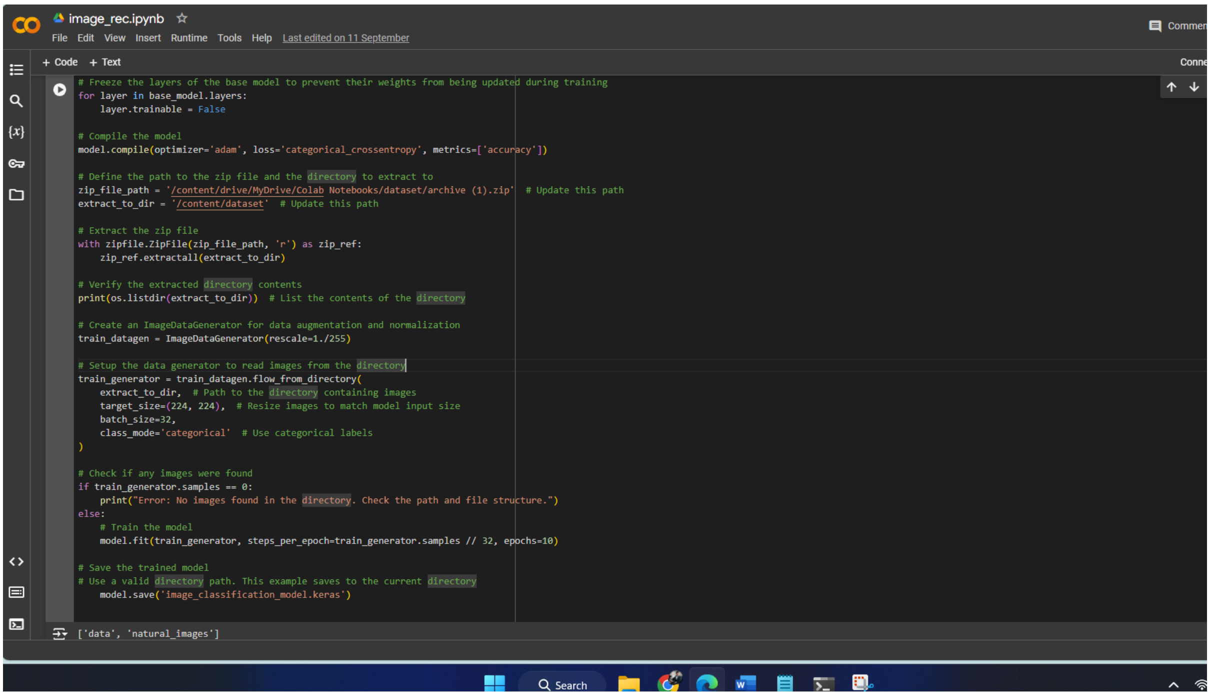
Data was collected from publicly available image datasets. The dataset consists of various classes including airplanes, cars, cats, dogs, flowers, fruits, motorbikes, and people. Images were labeled and organized into respective directories.

### 3.2 Model Architecture

The model architecture used is a convolutional neural network (CNN), which is well-suited for image classification tasks. The CNN comprises several convolutional layers followed by pooling layers, and ends with fully connected layers.



```python
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
import numpy as np
import zipfile
import os

# Load the MobileNetV2 model with pre-trained weights and exclude the top layers
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Add custom top layers to the model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)

# Set the number of output classes based on your dataset
num_classes = 2  # Update this with the actual number of classes
predictions = Dense(num_classes, activation='softmax')(x)

# Define the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of the base model to prevent their weights from being updated during training
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```



```python
# Freeze the layers of the base model to prevent their weights from being updated during training
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define the path to the zip file and the directory to extract to
zip_file_path = '/content/drive/MyDrive/Colab Notebooks/dataset/archive (1).zip'  # Update this path
extract_to_dir = '/content/dataset'  # Update this path

# Extract the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to_dir)

# Verify the extracted directory contents
print(os.listdir(extract_to_dir))  # List the contents of the directory

# Create an ImageDataGenerator for data augmentation and normalization
train_datagen = ImageDataGenerator(rescale=1./255)

# Setup the data generator to read images from the directory
train_generator = train_datagen.flow_from_directory(
    extract_to_dir,  # Path to the directory containing images
    target_size=(224, 224),  # Resize images to match model input size
    batch_size=32,
    class_mode='categorical'  # Use categorical labels
)

# Check if any images were found
if train_generator.samples == 0:
    print("Error: No images found in the directory. Check the path and file structure.")
else:
    # Train the model
    model.fit(train_generator, steps_per_epoch=train_generator.samples // 32, epochs=10)

# Save the trained model
# Use a valid directory path. This example saves to the current directory
    model.save('image_classification_model.keras')
```

```
['data', 'natural_images']
```

## 3.3 Training Process

The model was trained using a standard training procedure-

Data Augmentation: To improve generalization, data augmentation techniques such as rotation, flipping, and scaling were applied.

Training: The model was trained using a training set with a validation split to monitor overfitting.

Optimization: The Adam optimizer was used along with a categorical cross-entropy loss function.

Epochs: The training was carried out over a predefined number of epochs with early stopping based on validation accuracy.

```
['dog', 'fruit', 'cat', 'car', 'flower', 'airplane', 'motorbike', 'person']
Found 6899 images belonging to 8 classes.
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`,
  self._warn_if_super_not_called()
215/215 ━━━━━━━━━━━━━━━━━━━━ 325s 1s/step - accuracy: 0.9418 - loss: 0.1897
Epoch 2/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - accuracy: 0.9688 - loss: 0.2043
Epoch 3/10
/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.rep
  self.gen.throw(typ, value, traceback)
215/215 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - accuracy: 0.9929 - loss: 0.0259
Epoch 4/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 86us/step - accuracy: 1.0000 - loss: 3.5556e-05
Epoch 5/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 315s 1s/step - accuracy: 0.9966 - loss: 0.0109
Epoch 6/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 73us/step - accuracy: 1.0000 - loss: 0.0011
Epoch 7/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 317s 1s/step - accuracy: 0.9974 - loss: 0.0076
Epoch 8/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 2s 86us/step - accuracy: 1.0000 - loss: 1.8254e-07
Epoch 9/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - accuracy: 0.9982 - loss: 0.0061
Epoch 10/10
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 74us/step - accuracy: 0.9688 - loss: 0.0271
```

```
from google.colab import files
files.download('/content/image_classification_model.keras')
```

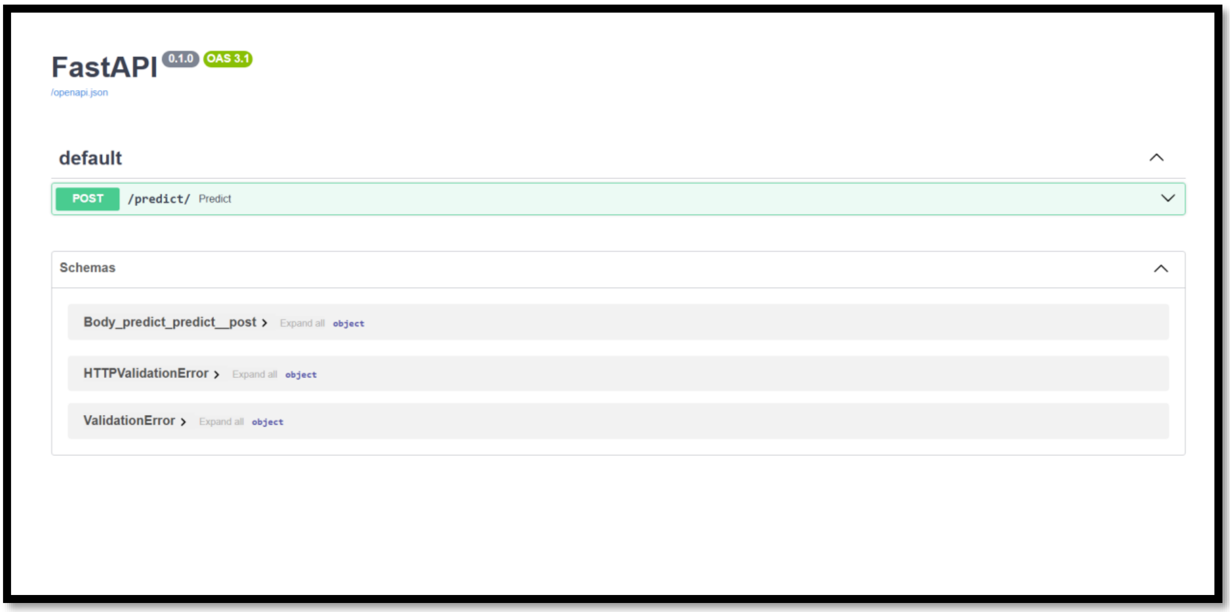
## 3.4 Evaluation

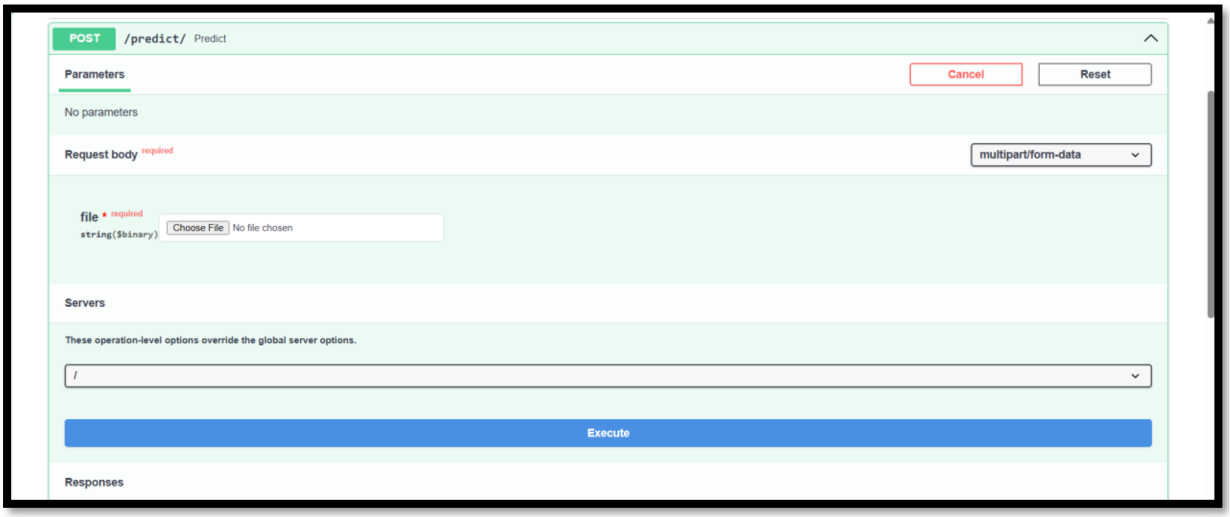
The model’s performance was evaluated using metrics such as accuracy, precision, recall, and F1-score on a test set. Confusion matrices were also analyzed to understand class-wise performance.

# FastAPI Application

## 4.1 Overview

The FastAPI application serves the trained model through a RESTful API. It allows users to upload images, which are then processed and classified by the model.







## 4.2 Application Structure

File Upload Endpoint: /predict/ accepts image files and returns classification results.

Preprocessing: Images are resized and normalized before being fed into the model.

Prediction: The model predicts the class of the image and returns the top prediction along with confidence scores.

## 4.3 API Endpoints

POST /predict/: Accepts an image file and returns the classification result. The response includes the predicted class, confidence score, and top predictions.

## 4.4 Running the Application Locally

The application can be run locally using uvicorn with the command:

uvicorn app:app --host 0.0.0.0 --port 8000

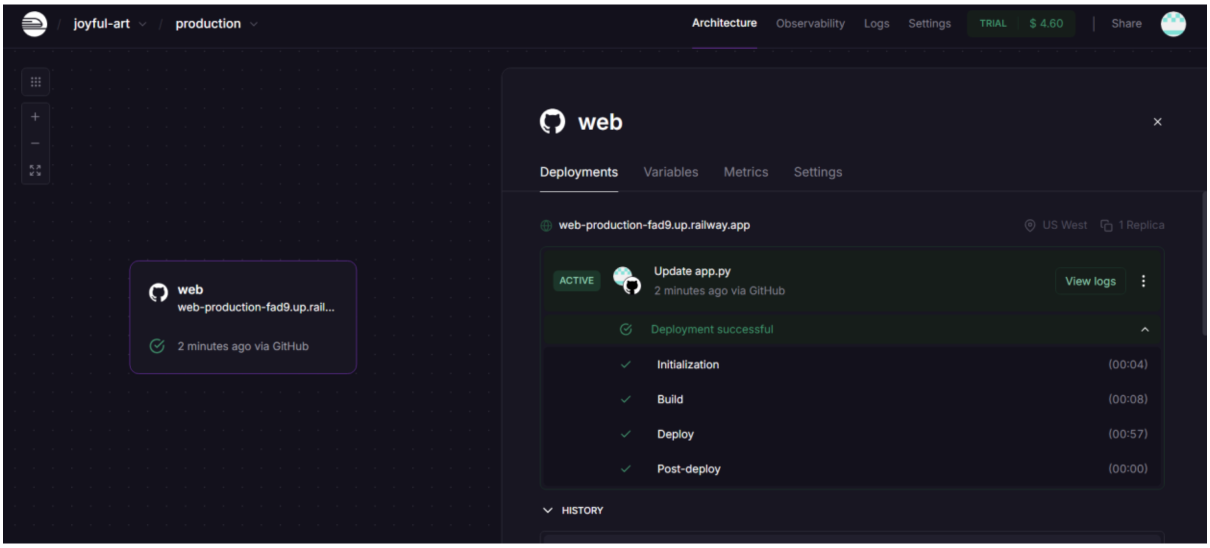
Ensure all dependencies are installed and the model file is available in the working directory.

```
F:\Horizon lectures\Year 4 semester 1 Lecture notes\Intelligent system\img\New folder>uvicorn app:app --reload
INFO:     Will watch for changes in these directories: ['F:\\Horizon lectures\\Year 4 semester 1 Lecture notes\\Intellig
ent system\\img\\New folder']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [7152] using StatReload
2024-09-12 22:37:55.872505: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly di
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them off, se
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-09-12 22:37:59.224001: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly di
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them off, se
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Model file found locally.
2024-09-12 22:38:07.075249: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the app
ropriate compiler flags.
INFO:     Started server process [14124]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:61755 - "GET / HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:61759 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:61759 - "GET /openapi.json HTTP/1.1" 200 OK
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
INFO:     127.0.0.1:61761 - "POST /predict/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:62658 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:62658 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:62662 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:62662 - "GET /openapi.json HTTP/1.1" 200 OK
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
```

# Deployment Setup

## 5.1 Railway Deployment –https://web-production-874b0.up.railway.app/

Railway.app is used for deploying the FastAPI application. It supports automatic deployments and integrates well with GitHub repositories.



### 5.2 Deployment Steps

Create a Railway Project: Set up a new project on Railway.app.

Link Repository: Connect the GitHub repository containing the FastAPI application.

Configure Environment: Set environment variables and configure the build settings.

Deploy: Push changes to the GitHub repository to trigger the deployment.

## CI/CD Pipeline Documentation

### 6.1 CI/CD Pipeline Setup

A CI/CD pipeline is configured using GitHub Actions for continuous integration and deployment. The pipeline includes:

Build Jobs: Install dependencies and build the application.

Test Jobs: Run unit tests and integration tests.

Deploy Jobs: Deploy the application to Railway.app upon successful test completion.

### 6.2 Explanation of Steps

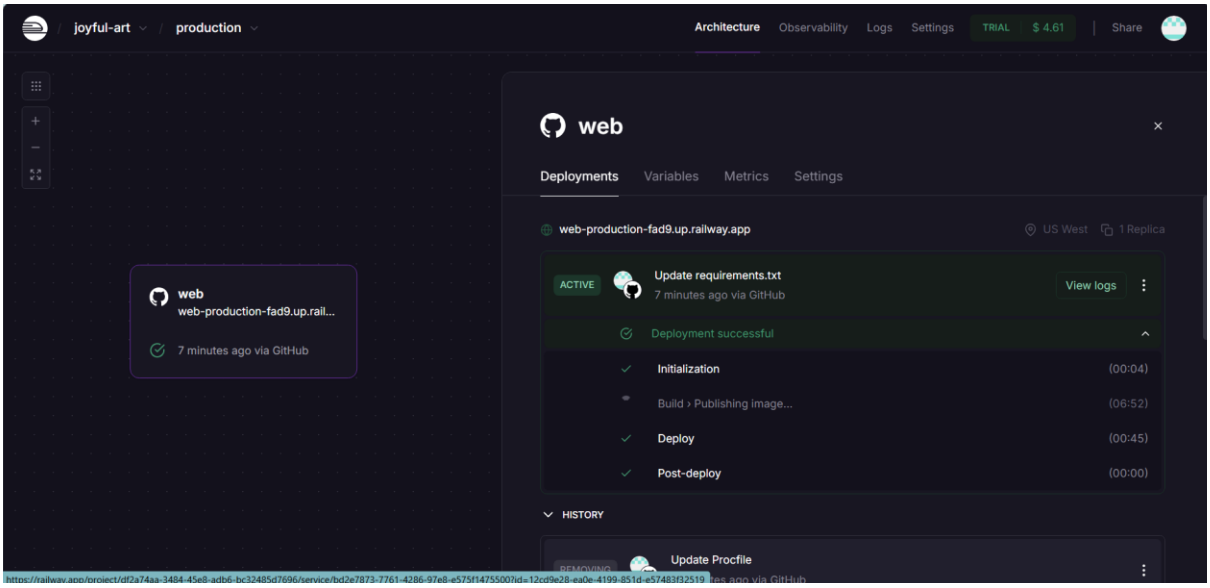
Build: Install necessary dependencies and set up the environment.

Test: Execute tests to ensure code quality and functionality.

Deploy: Deploy the application to the hosting platform, with automated triggers based on code changes.

## Online Testing

The deployed application is tested online using the provided Swagger UI on Railway.app. Functional testing includes verifying the image upload and prediction functionality, as well as performance testing to ensure response times are acceptable.




# Conclusion

## 8.1 Summary

The image recognition system developed leverages TensorFlow for model training and FastAPI for serving the model through a RESTful API. This solution provides a robust and scalable approach to image classification, offering users an easy way to interact with the model via a web interface. The deployment of the application is managed through Railway.app, which simplifies the hosting and scaling process. Additionally, a CI/CD pipeline using GitHub Actions ensures that the application is continuously integrated and deployed, maintaining high quality and consistency.

# Explanation of app.py Code

The provided app.py code illustrates how the image recognition system is implemented using FastAPI and TensorFlow. Here's a detailed breakdown of the key components:

1. Importing Libraries

python

```
from fastapi import FastAPI, File, UploadFile

import tensorflow as tf

from PIL import Image

import io
```

import numpy as np

import os

import gdown

FastAPI: Framework for building the web API.

TensorFlow: Library used for loading and interacting with the trained model.

PIL: Used for image processing.

io: Handles image file streams.

numpy: Used for numerical operations on image data.

os: Interacts with the operating system (e.g., checking file existence).

gdown: Library for downloading files from Google Drive.

## 2. Initializing FastAPI App

python

app = FastAPI()

Initializes the FastAPI application.

## 3. Model Path and Download Function

python

Copy code

```
model_path = 'image_classification_model.keras'

file_id = '1ANXn8Bz1rpEDXJkg0TLPiQOzFeJKq9il'


def download_model():
    if not os.path.exists(model_path):
        print("Model file not found locally. Downloading from Google Drive...")
        gdown.download(f'https://drive.google.com/uc?id={file_id}', model_path, quiet=False)
    else:
        print("Model file found locally.")
```

model_path: Path where the model file is stored.

file_id: Google Drive file ID for downloading the model.

download_model: Function that checks if the model file exists locally and downloads it if not.

### 3. Loading the Model

python

download_model()

model = tf.keras.models.load_model(model_path)

Downloads the model if necessary and loads it using TensorFlow.

### 5. Class Labels

```
class_labels = {
    0: 'airplane',
    1: 'car',
    2: 'cat',
    3: 'dog',
    4: 'flower',
    5: 'fruit',
    6: 'motorbike',
    7: 'person'
}
```

Defines the mapping of class indices to human-readable labels.

### 6. Image Preprocessing

python

Copy code

```
def preprocess_image(image: Image.Image):
    image = image.resize((224, 224))  # Resize to match your model input size
    image_array = np.array(image) / 255.0  # Normalize image
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension
    return image_array
```

preprocess_image: Function that resizes, normalizes, and adds a batch dimension to the input image.

## 7. Prediction Function

```
def predict_image(image_array: np.ndarray):
    predictions = model.predict(image_array)
    confidence_scores = predictions[0]
    predicted_class_index = int(np.argmax(confidence_scores))

    return {
        "predicted_class_index": predicted_class_index,
        "predicted_class_name": class_labels.get(predicted_class_index, "Unknown"),
        "confidence": float(confidence_scores[predicted_class_index]),
        "top_predictions": [
            {
                "class_index": predicted_class_index,
                "confidence": float(confidence_scores[predicted_class_index]),
                "class_name": class_labels.get(predicted_class_index, "Unknown")
            }
        ]
    }
```

predict_image: Function that makes predictions using the model and returns the top prediction along with confidence scores.

## 8. API Endpoint

```
@app.post("/predict/")
async def predict(file: UploadFile = File(...)):
    try:
        image = Image.open(io.BytesIO(await file.read())).convert("RGB")
        image_array = preprocess_image(image)
        prediction_details = predict_image(image_array)
        return prediction_details
    except Exception as e:
```

```
        return {"error": str(e)}
```

predict: FastAPI endpoint that accepts image files, processes them, and returns predictions. Handles exceptions and provides error messages if needed.

## 9. Running the Application Locally

python

Copy code

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Runs the FastAPI application locally on port 8000 using Uvicorn.