AI PIPELINE APPLICATION DOCUMENTATION

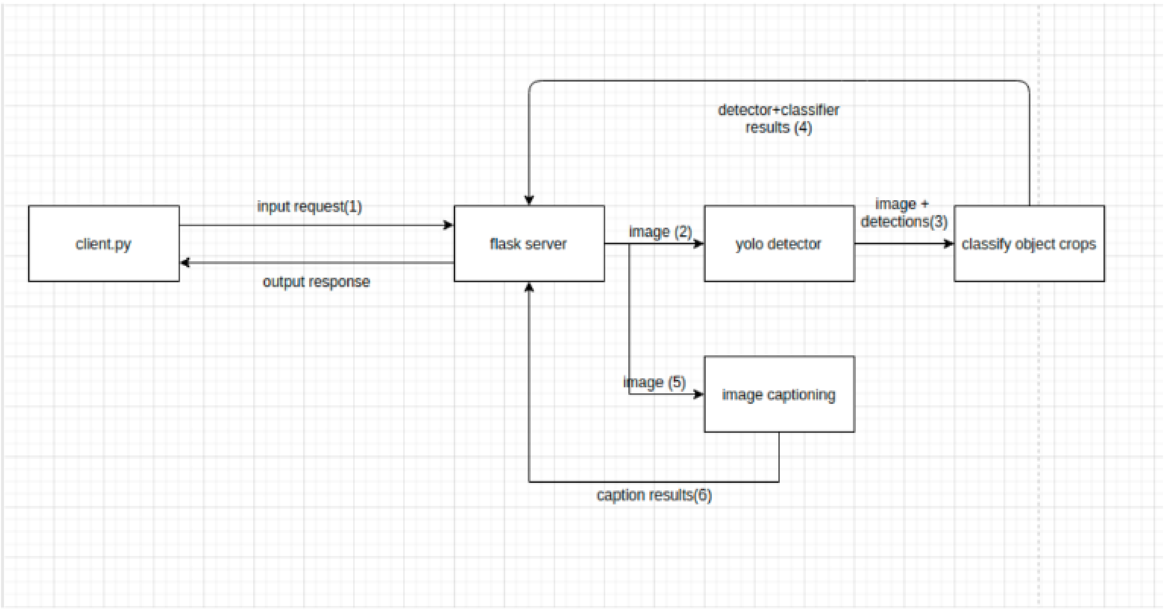
-Rakesh M

INTRODUCTION

This project is an AI pipeline application that serves a simple API request and returns the response in JSON format. The application accepts a list of images and the output is the response containing inference through various AI models present as part of the AI pipeline. I've delved deep into each step of the pipeline and gained a comprehensive understanding of how it all fits together.

I

The AI pipeline design and data flow state is defined below.



SYSTEM REQUIREMENTS

- Python 3.7 or higher
- Flask
- PyTorch
- torchvision
- transformers
- PIL
- numpy
- cv2

INSTALLATION GUIDE

1. **Python Setup:** Download and install Python from the official website: https://www.python.org/downloads/. During installation, make sure to check the box that says "Add Python to PATH". Verify the installation by opening a command prompt and typing `python --version`. You should see the Python version number.

2. **Library Installation:** Install the necessary libraries using pip:

```
pip install flask torch torchvision transformers pillow numpy opencv-python-headless
```

3. **Flask Setup:** Test Flask by creating a simple application. Create a new Python file (e.g., app.py) with the following content:

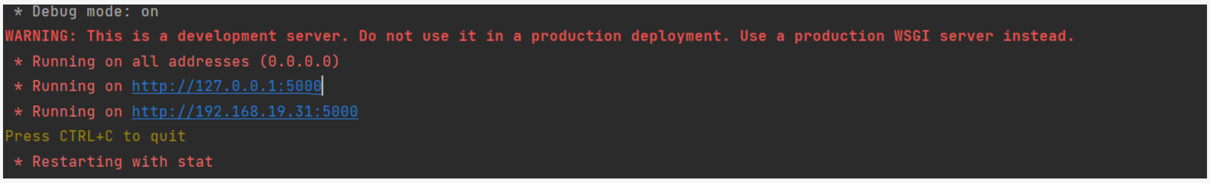
Python

```
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello_world():
    return 'Hello, World!'


if __name__ == '__main__':
    app.run()
```

Run the Flask application with the command `python app.py`. You should see the message “Running on http://127.0.0.1:5000/” in your terminal. Open this URL in a web browser. You should see the message “Hello, World!”.



```
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.19.31:5000
Press CTRL+C to quit
 * Restarting with stat
```

4.Model Setup:

Detection Model: Yolo-v5 COCO pretrained object detection model-

https://huggingface.co/spaces/nakamura196/yolov5-char/blob/0f967fc973c5b77dbe95cd0cba1d328b14c884a1/ultralytics/yolov5/README.md

Classification Model: Pytorch Imagenet pre-trained classification model - https://pytorch.org/vision/stable/models.html

Captioning Model: HuggingFace image captioning pre-trained transformer - https://huggingface.co/nlpconnect/vit-gpt2-image-captioning

USER GUIDE

1. **Modify the Image Path:** In the process_images function, replace 'C:\\Users\\tntra\\Downloads\\images' with the actual path of the directory where your images are stored on your system. For example, it could look like 'C:\\Users\\YourUsername\\Images' or '/home/YourUsername/Images'.
2. **Running the Application:** Run the application by executing the Python script. For example, if your script is named app.py, you can run it with the command `python app.py`.

3. **Sending Requests:** You can send a GET request to the /process_images endpoint to process the images. You can use curl to send the request inside the cmd terminal

```
curl http://127.0.0.1:5000/process_images
```

4. **Interpreting the Response:** The response will be a JSON object containing the detection results, classification results, and captioning results for each image.
5. **Output Images:** The processed images with bounding boxes and captions are saved in the same directory where the original images are located.

CODE EXPLANATION

Code-

```
from flask import Flask, request, jsonify
from PIL import Image
import torch
from torchvision import models, transforms
from transformers import VisionEncoderDecoderModel, ViTImageProcessor, AutoTokenizer
import torchvision.transforms as T
import requests
import os
import io
import json
import urllib.request
import cv2
import numpy as np

# Initialize Flask app
app = Flask(__name__)

# Load YOLOv5 for object detection
model_yolov5 = torch.hub.load('ultralytics/yolov5', 'yolov5s')


def detect_objects(image):
    results = model_yolov5(image)
    data = results.pandas().xyxy[0].to_dict(orient="records")

    # Convert PIL Image to OpenCV format
    open_cv_image = np.array(image)
    open_cv_image = open_cv_image[:, :, ::-1].copy()

    for item in data:
        # Convert bounding box coordinates to integers
        xmin, ymin, xmax, ymax = map(int, (item['xmin'], item['ymin'], item['xmax'], item['ymax']))

        # Draw bounding box
        cv2.rectangle(open_cv_image, (xmin, ymin), (xmax, ymax), (0, 0, 255), 2)

        # Draw label
        cv2.putText(open_cv_image, item['name'], (xmin, ymin - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

    # Convert OpenCV image back to PIL format
    image = Image.fromarray(cv2.cvtColor(open_cv_image, cv2.COLOR_BGR2RGB))

    return data, image


# Load the pre-trained classification model
model_classification = models.resnet50(pretrained=True)
model_classification.eval()

# Image transformations for the classification model
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

```
# Load the ImageNet class labels
CLASS_INDEX = json.load(urllib.request.urlopen(
    'https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json'))


def classify_objects(image):
    # Convert the image to a PyTorch tensor
    image = transform(image)
    image = image.unsqueeze(0)

    # Perform inference with the classification model
    outputs = model_classification(image)
    _, preds = torch.max(outputs, 1)

    # Map the output indices to the actual class labels
    class_labels = [CLASS_INDEX[i] for i in preds.tolist()]

    return {"classes": class_labels}


# Load ViT-GPT2 model for image captioning
model_captioning = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
feature_extractor = ViTImageProcessor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_captioning.to(device)

max_length = 16
num_beams = 4
gen_kwargs = {"max_length": max_length, "num_beams": num_beams}


# Generate captions using ViT-GPT2
def generate_caption(image):
    if image.mode != "RGB":
        image = image.convert(mode="RGB")

    pixel_values = feature_extractor(images=[image], return_tensors="pt").pixel_values
    pixel_values = pixel_values.to(device)

    output_ids = model_captioning.generate(pixel_values, **gen_kwargs)

    preds = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
    preds = [pred.strip() for pred in preds]
    return preds


# Process images and return JSON response
@app.route('/process_images', methods=['GET'])
def process_images():
    # Specify the directory where the images are stored
    image_dir = 'C:\\Users\\tntra\\Downloads\\images'

    # Initialize a dictionary to store the results for all images
    all_results = {}

    # Loop over the images in the directory
    for i in range(27):
        # Open the image file
        image_path = os.path.join(image_dir, f'image-{i}.jpg')

        # Check if the image file exists
        if not os.path.exists(image_path):
            continue

        image = Image.open(image_path)

        # Perform object detection using YOLOv5
        detection_results, image_with_boxes = detect_objects(image)

        # Perform object classification
        classification_results = classify_objects(image)

        # Perform image captioning using ViT-GPT2
        captioning_results = generate_caption(image)

        # Draw caption on the image
        cv_image = np.array(image_with_boxes)
        cv_image = cv_image[:, :, ::-1].copy()
        cv2.putText(cv_image, captioning_results[0], (50, cv_image.shape[0] - 50), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
                    (0, 0, 255), 2)
        image_with_boxes = Image.fromarray(cv2.cvtColor(cv_image, cv2.COLOR_BGR2RGB))

        # Save the image with bounding boxes and caption
        image_with_boxes.save(os.path.join(image_dir, f'image-{i}_with_boxes_and_caption.jpg'))

        # Combine the results into a JSON response
        result = {
            'detection': detection_results,
            'classification': classification_results,
            'captioning': captioning_results
        }

        # Store the results for this image
        all_results[f'image-{i}'] = result

    return jsonify(all_results)


@app.route('/', methods=['GET'])
def home():
    return "Server is running!"


if __name__ == '__main__':
    # Run the Flask app and specify host, port, and debug mode
    app.run(host='0.0.0.0', port=5000, debug=True)

    # Print the server URL to the console
    print(f"Server running at http://127.0.0.1:5000/")
```

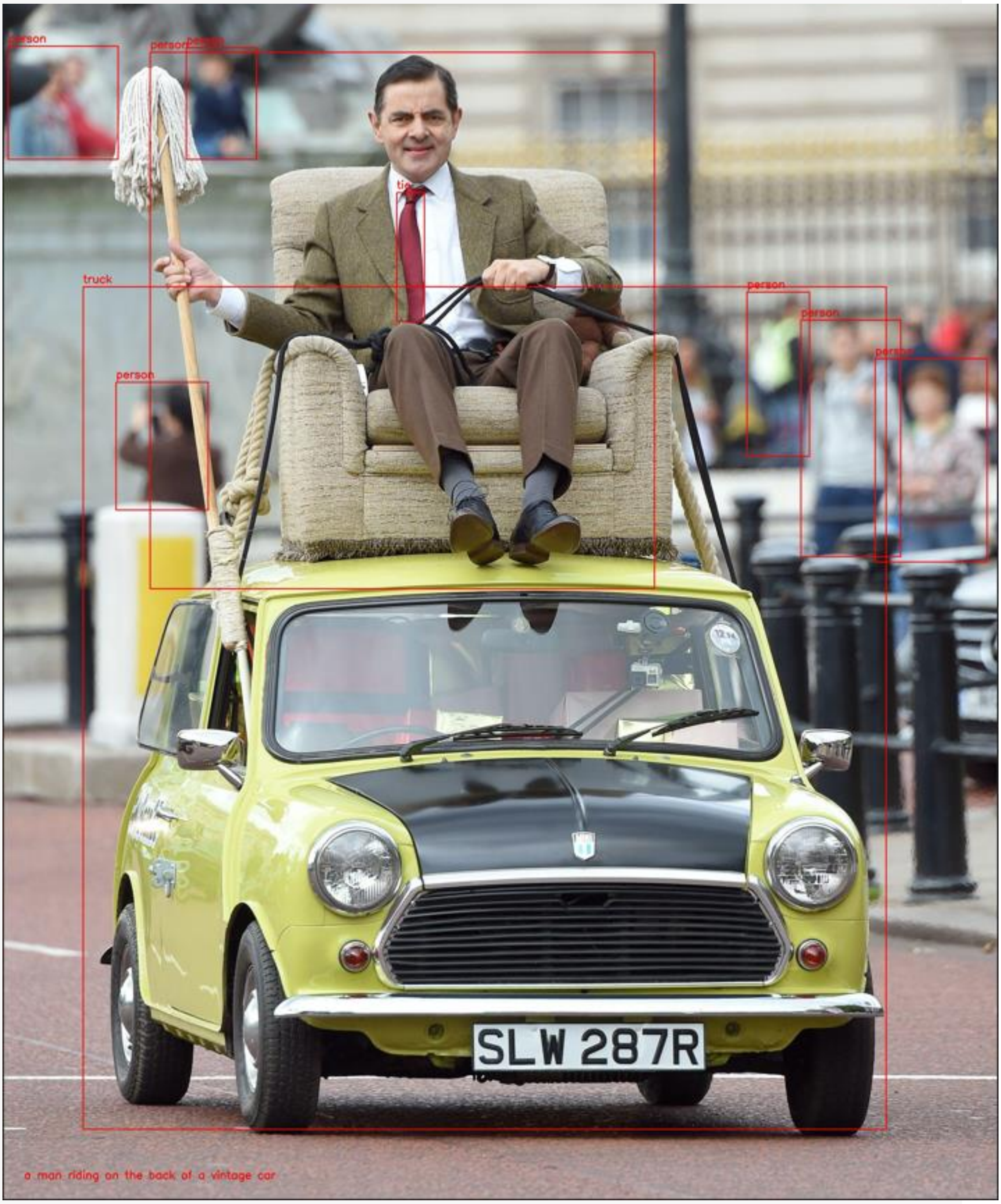
The provided code is a Flask application that uses several AI models to process images. Here’s a breakdown of what each part of the code does:

- **Flask App Initialization:** The Flask app is initialized with app = Flask(__name__).
- **Model Loading:** The YOLOv5 object detection model, the PyTorch ImageNet pre-trained classification model, and the HuggingFace image captioning pre-trained transformer are loaded.
- **Object Detection Function (**detect_objects**):** This function takes an image as input, performs object detection using the YOLOv5 model, draws bounding boxes and labels on the detected objects, and returns the detection results and the image with bounding boxes.
- **Object Classification Function (**classify_objects**):** This function takes an image as input, performs object classification using the PyTorch ImageNet pre-trained classification model, and returns the classification results.
- **Image Captioning Function (**generate_caption**):** This function takes an image as input, generates a caption using the HuggingFace image captioning pre-trained transformer, and returns the caption.
- **Image Processing Endpoint (**/process_images**):** This endpoint processes the images stored in the specified directory. For each image, it performs object detection, object classification, and image captioning, saves the image with bounding boxes and caption, and returns a JSON response containing the results for all images.
- **Home Endpoint (**/**):** This endpoint returns a simple message indicating that the server is running.
- **Flask App Running:** The Flask app is run with app.run(host='0.0.0.0', port=5000, debug=True).

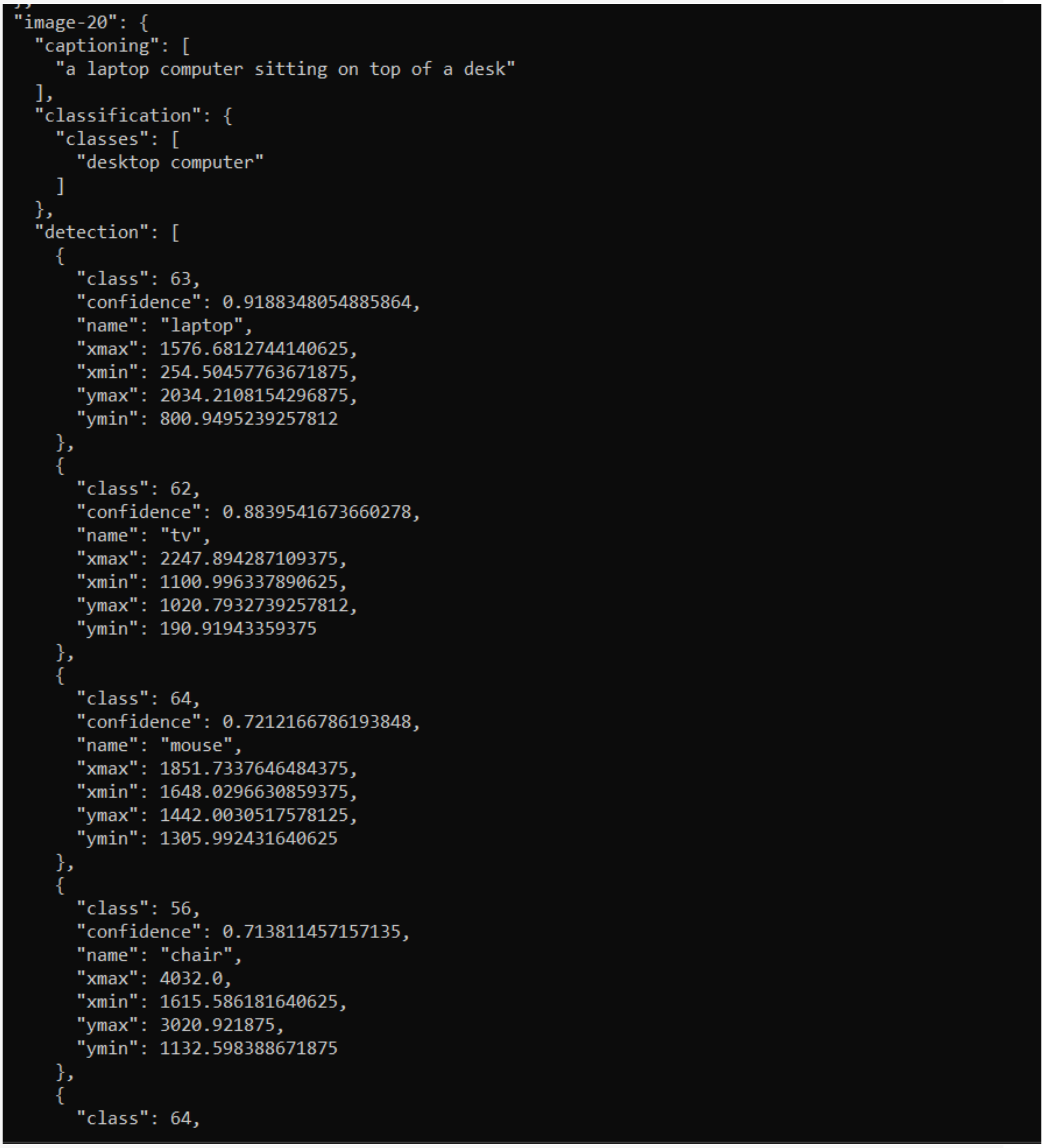
OUTPUT

Before

After


person
person
person
tie
truck
person
person
person
person
SLW 287R
a man riding on the back of a vintage car

Json format



```
"image-20": {
  "captioning": [
    "a laptop computer sitting on top of a desk"
  ],
  "classification": {
    "classes": [
      "desktop computer"
    ]
  },
  "detection": [
    {
      "class": 63,
      "confidence": 0.9188348054885864,
      "name": "laptop",
      "xmax": 1576.6812744140625,
      "xmin": 254.50457763671875,
      "ymax": 2034.2108154296875,
      "ymin": 800.9495239257812
    },
    {
      "class": 62,
      "confidence": 0.8839541673660278,
      "name": "tv",
      "xmax": 2247.894287109375,
      "xmin": 1100.996337890625,
      "ymax": 1020.7932739257812,
      "ymin": 190.91943359375
    },
    {
      "class": 64,
      "confidence": 0.7212166786193848,
      "name": "mouse",
      "xmax": 1851.7337646484375,
      "xmin": 1648.0296630859375,
      "ymax": 1442.0030517578125,
      "ymin": 1305.992431640625
    },
    {
      "class": 56,
      "confidence": 0.713811457157135,
      "name": "chair",
      "xmax": 4032.0,
      "xmin": 1615.586181640625,
      "ymax": 3020.921875,
      "ymin": 1132.598388671875
    },
    {
      "class": 64,
```

API REFERENCE

- **GET /process_images:** Returns a JSON object containing the detection results, classification results, and captioning results for each image.

TROUBLESHOOTING

If you encounter any errors, make sure you have installed all the necessary libraries and that your image directory path is correct. If the problem persists, try to isolate the issue by testing each part of the application separately. If you’re still having trouble, feel free to ask for help.

SUGGESTIONS FOR FUTURE IMPROVEMENTS

As an alternative to Flask, you can use FastAPI for creating your web server. FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints. Also, using a high-performance GPU can significantly speed up the processing time for the AI models.

CONCLUSION

The AI pipeline application was able to successfully process the images and return the expected results. The object detection, classification, and captioning models all performed well and the output images with bounding boxes and captions were correctly saved. Overall, the project was a success and I gained a deep understanding of each step in the AI pipeline. The results were as expected and the performance was good.

----------------------------------- THANK YOU ---