Real Time Object Recognition with Voice-Guided Navigation for the Visually Impaired using YOLO

This Document should contain the timely implementation updates and the Technologies you are using for this project

Total requirements
Once again I wanna make clear about the requirements
1. Multilingual
2. Settings page user details has to be displayed
3. Voice guidance i.e it has to give voice command of detected object
4. Navigation it should provide warning like u r left there is knife so it would be better if u turn right
5. Currency value has to be recognised
6. Not coco data set it should be like real time object detection mean whatever in the live video objects are there all has to be recognised (data set should be modified)
7. Like in super markets cost has to be guided through voice
8. It should respond to the voice command like how many people around me like that

Object Detection Voice Guide

Code overview:

This program detects objects in real-time using a pre-trained SSD (Single Shot MultiBox Detector) MobileNet model, calculates their approximate distance from the camera based on known object dimensions, and provides real-time voice feedback about the object's location and distance. It uses OpenCV for image processing, `pyttsx3` for voice synthesis, and threading to ensure smooth audio output.

Libraries Used

1. **OpenCV (cv2):**
    - Open Source Computer Vision Library.

    - Used for capturing video from the webcam, processing images, and drawing bounding boxes around detected objects.
2. **NumPy:**
    - Fundamental Python library for numerical operations.
    - Used to reshape arrays and handle bounding box data.
3. **pyttsx3:**
    - Python Text-to-Speech library.
    - Enables voice feedback for detected objects' names, distances, and positions.
4. **Threading:**
    - A Python module for multithreading.
    - Used to run the voice feedback function in a separate thread to ensure the program runs smoothly without lag.
5. **Queue:**
    - A thread-safe way to share data between threads.
    - Used to queue objects and their details for voice feedback.
6. **Time:**
    - Used to add short delays in the voice feedback loop.

Pre-trained Model

The program uses **SSD MobileNet V3** (Single Shot Detector with MobileNet backbone), which is a lightweight object detection model trained on the COCO dataset.

1. **Model Files:**
    - `ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt` - Configuration file for the SSD MobileNet model.
    - `frozen_inference_graph.pb` - Pre-trained weights of the SSD MobileNet model.
2. **COCO Dataset (Common Objects in Context):**
    - A dataset containing 91 commonly found object classes such as person, car, bicycle, chair, etc.
    - Class names are stored in the `coco.names` file.

Supporting File: average_sizes.txt

- Contains real-world average sizes (width in meters) of objects (e.g., cars, humans, chairs).
- Used for calculating approximate distances based on the object's bounding box width.

Example Content:

Copy code
```
person,0.5
car,1.7
bottle,0.075
```

Code Explanation

1. Loading Class Names and Average Sizes

- Reads class names from `coco.names` and real-world average object sizes from `average_sizes.txt` into a dictionary.

python
Copy code
```python
with open(classFile, 'rt') as f:
    classNames = f.read().rstrip('\n').split('\n')

with open(average_sizes_file, 'rt') as f:
    for line in f:
        obj, size = line.strip().split(',')
        average_sizes[obj.strip()] = float(size.strip())
```

2. Model Initialization

- Loads the SSD MobileNet V3 model and sets input parameters:
    - Input size: (320x320)
    - Normalization: Rescales input by dividing pixel values by 127.5.

python
Copy code

```
net = cv2.dnn_DetectionModel(weightsPath, configPath)
net.setInputSize(320, 320)
net.setInputScale(1.0 / 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
```

---

3. Voice Feedback Using pyttsx3

- A separate thread handles text-to-speech processing to avoid slowing down the main loop.
- The `Queue` stores object details (label, distance, and position).

python
Copy code

```
def speak(q):
    engine = pyttsx3.init()
    engine.setProperty('rate', 235)
    engine.setProperty('volume', 1.0)
    while True:
        if not q.empty():
            label, distance, position = q.get()
            rounded_distance = round(distance * 2) / 2
            engine.say(f"{label.upper()} is {rounded_distance}
meters to your {position}")
            engine.runAndWait()
            with queue.mutex:
                queue.queue.clear()
        else:
            time.sleep(0.1)
```

---

4. Distance Estimation

- Uses the formula for calculating distance based on the **focal length**, **real-world width**, and the object's bounding box width:

Distance=Real Width×Focal LengthWidth in Image\text{Distance} = \frac{\text{Real Width} \times \text{Focal Length}}{\text{Width in Image}}Distance=Width in ImageReal Width×Focal Length

- Focal length is an approximate constant, and real-world sizes are read from `average_sizes.txt`.

python
Copy code

```
def calculate_distance(object_width, real_width):
    return (real_width * focal_length) / (object_width + 1e-6)
```

---

5. Object Position

- Determines whether the object is to the **LEFT**, **FORWARD**, or **RIGHT** based on its horizontal position in the frame.

python
Copy code

```
def get_position(frame_width, box):
    if box[0] < frame_width // 3:
        return "LEFT"
    elif box[0] < 2 * (frame_width // 3):
        return "FORWARD"
    else:
        return "RIGHT"
```

---

6. Object Detection Loop

- Captures video from the webcam.
- Performs object detection using `net.detect()`.
- Filters results using **Non-Maximum Suppression (NMS)** to remove redundant bounding boxes.
- Calculates the object's distance and determines its position.
- Adds this information to the voice feedback queue.

python

```
classIds, confs, bbox = net.detect(img, confThreshold=thres)
indices = cv2.dnn.NMSBoxes(bbox, confs, thres, nms_threshold)

if len(indices) > 0:
    for i in indices.flatten():
        # Draw bounding boxes and calculate distance/position
        label = classNames[class_id - 1].lower()
        distance = calculate_distance(w, average_sizes[label])
        position = get_position(frame_width, (x, y, x + w, y +
h))
        queue.put((label, distance, position))
```

---

7. Displaying Results

- Draws bounding boxes and distance information on the video feed.
- Press `q` to exit the program.

python

```
cv2.rectangle(img, (x, y), (x + w, y + h), color=(0, 255, 0),
thickness=2)
cv2.putText(img, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.6, (0, 255, 0), 2)
cv2.imshow("Object Detection with Distance and Voice Guide",
img)
```

Approach

1. **Object Detection:**
    - Uses SSD MobileNet for detecting objects in the video feed.
    - Applies NMS to improve detection accuracy.
2. **Distance Estimation:**

    - Calculates the distance to the detected object using the focal length and real-world dimensions.
3. **Voice Feedback:**
    - Provides real-time voice alerts for object names, distances, and positions using `pyttsx3`.
4. **Real-Time Visualization:**
    - Draws bounding boxes and overlays distance data on the video feed.

How to Run the Code

1. Ensure the following files are present in the working directory:
    - `coco.names`
    - `ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt`
    - `frozen_inference_graph.pb`
    - `average_sizes.txt`

Install required libraries:
bash
Copy code
```
pip install opencv-python numpy pyttsx3
```

2.

Run the script:
```
Python obj_det_voice_guide.py
```

3.
4. Use the **webcam** to detect objects and listen to voice feedback.

Conclusion

This program integrates object detection, distance estimation, and voice feedback into a seamless pipeline. It is useful for applications like assistive technologies for visually impaired individuals, smart surveillance systems, and real-time navigation aids.

Person Count Detector

code overview:

This program uses a pre-trained **Faster R-CNN (ResNet-50)** object detection model to detect and count persons in real-time from video input (webcam or a video file). It overlays bounding boxes and labels on detected persons and provides text-to-speech (TTS) announcements of the count using `pyttsx3`. The model is part of the **Torchvision library** and trained on the COCO dataset.

Libraries Used

1. **OpenCV (cv2):**
    - Open-source library for real-time computer vision.
    - Used for capturing video frames, drawing bounding boxes, and displaying results.
2. **NumPy:**
    - Fundamental Python library for numerical computations.
    - Used for converting model predictions (PyTorch tensors) to arrays.
3. **PyTorch:**
    - Deep learning framework used for loading and running the Faster R-CNN model.
4. **Torchvision:**
    - A PyTorch library for computer vision tasks.
    - Provides access to pre-trained Faster R-CNN ResNet-50 models.
5. **pyttsx3:**
    - Python Text-to-Speech library.
    - Used to announce the total count of detected persons.

Model Used

Faster R-CNN (ResNet-50)

- **Faster R-CNN** is a two-stage object detection model that:
  - Generates region proposals using a Region Proposal Network (RPN).
  - Classifies objects and refines bounding boxes for detected regions.
- **ResNet-50 Backbone:**
  - A ResNet-50 network is used as the feature extractor.
- **Pre-trained on COCO Dataset:**
  - The COCO dataset includes 91 object categories, including "person".

Code Explanation

1. Initialize Pre-trained Model

- The **Faster R-CNN model** is loaded with weights pre-trained on the COCO dataset.
- `model.eval()` sets the model to evaluation mode, disabling layers like dropout or batch normalization that behave differently during training.

python
Copy code
```
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()
```

---

2. Text-to-Speech Announcement

- `pyttsx3` is initialized to provide voice feedback for the count of detected persons.
- The `announce_count` function generates a message based on the detected count and plays it.

python
Copy code
```
def announce_count(count):
    if count > 0:
        message = f"There are {count} person{'s' if count > 1
else ''} detected."
    else:
```

```
        message = "No persons detected."
    engine.say(message)
    engine.runAndWait()
```

---

3. Detect Persons in Video Frames

- Frames are preprocessed into tensors using Torchvision's `ToTensor` transform.
- The model outputs predictions, including bounding boxes, labels, and confidence scores.
- Persons (`label == 1` for COCO) with a confidence score above 0.6 are considered valid detections.

python
Copy code

```
def detect_persons(frame, prev_count):
    transform = transforms.ToTensor()
    frame_tensor = transform(frame).unsqueeze(0)

    # Perform detection
    with torch.no_grad():
        predictions = model(frame_tensor)[0]

    # Extract bounding boxes, labels, and scores
    boxes = predictions['boxes'].numpy()
    labels = predictions['labels'].numpy()
    scores = predictions['scores'].numpy()
```

---

4. Draw Bounding Boxes and Labels

- For each valid detection, a bounding box is drawn on the frame using OpenCV's `cv2.rectangle`.
- A label indicating the person index (e.g., "Person 1") is displayed above the bounding box.

python

Copy code

```
for i, label in enumerate(labels):
    if label == PERSON_CLASS_ID and scores[i] > 0.6:
        person_count += 1
        box = boxes[i].astype(int)
        cv2.rectangle(frame, (box[0], box[1]), (box[2], box[3]),
(0, 255, 0), 2)
        label_text = f"Person {person_count}"
        cv2.putText(frame, label_text, (box[0], box[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
```

---

5. Announce Person Count

- The count is displayed on the frame.
- If the count changes from the previous frame, it triggers an announcement via `announce_count`.

python
Copy code

```
cv2.putText(frame, f"Total Persons: {person_count}", (10, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
if person_count != prev_count:
    announce_count(person_count)
```

---

6. Video Capture and Real-Time Detection

- Captures frames from a webcam (`cv2.VideoCapture(0)`) or video file.
- The `detect_persons` function is called for each frame, and results are displayed in a window.
- Pressing the `q` key exits the loop.

python
Copy code

```
while True:
    ret, frame = cap.read()
```

```
    if not ret:
        print("Failed to grab frame.")
        break

    frame, prev_count = detect_persons(frame, prev_count)
    cv2.imshow("Person Detector", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Approach

1. **Load Pre-trained Model:**
    - Faster R-CNN with ResNet-50 backbone is loaded, which can detect 80 object categories.
2. **Preprocess Input Frames:**
    - Each video frame is converted to a tensor for model inference.
3. **Run Object Detection:**
    - The model outputs bounding boxes, labels, and scores.
    - Only "person" class labels (ID = 1) with scores above a threshold are processed.
4. **Display Results:**
    - Bounding boxes and labels are drawn on detected persons.
    - The total count of persons is displayed on the frame.
5. **Text-to-Speech Feedback:**
    - Announces the count only when it changes, ensuring minimal interruptions.

How to Run the Code

Prerequisites:

Install the required libraries:
bash

Copy code

```
pip install opencv-python numpy torch torchvision pyttsx3
```

1.
2. Ensure your system supports PyTorch's pre-trained models. Check the installation guide: PyTorch Installation.

Execution:

1. Save the code in a file (e.g., `person_detection.py`).

Run the script:

```
Python per_count.py
```

2.
3. For real-time detection, connect a webcam or use a video file (`test_video2.mp4`).

Applications

- **Surveillance Systems:** Monitor and count people in crowded areas.
- **Event Management:** Keep track of attendees.
- **Assistive Technology:** Provide real-time feedback for visually impaired individuals.

Conclusion: This project integrates state-of-the-art object detection with real-time text-to-speech feedback to count and announce detected persons. It demonstrates a practical application of machine learning in real-world scenarios using PyTorch and OpenCV.

Text to Multiple Language Convertor

Code Overview:

This app uses a pre-trained AI model to translate text or have a conversation. The user can input text, select an action (such as translating it to English, Chinese, or Japanese),

and the model will generate the output. It also converts the generated text into speech. The app uses libraries like **Hugging Face Transformers**, **Google Text-to-Speech (gTTS)**, and **Gradio** to provide an easy interface for users.

---

2. Libraries and Models Used

- `transformers`:
  - A library that provides access to pre-trained AI models for tasks like translation and text generation.
  - **Model Used**: `Qwen/Qwen2-1.5B-Instruct` – A large AI model that can generate text based on instructions.
- `gTTS` **(Google Text-to-Speech)**:
  - This library converts the generated text into speech in different languages (like English, Chinese, or Japanese).
- `gradio`:
  - A library for building user-friendly interfaces where users can interact with machine learning models easily.

---

3. How the App Works

1. **Loading the Model**:
   - The app loads a pre-trained AI model (`Qwen/Qwen2-1.5B-Instruct`) using the `transformers` library. This model is capable of handling tasks like translation and text generation.
2. **Handling Input**:
   - The user enters a text and selects an action (like translating to English, Chinese, Japanese, or just chatting).
   - Depending on the action selected, the app formats a prompt (like "Translate this text to English") and passes it to the AI model.
3. **Text Generation**:
   - The AI model processes the input text and generates the appropriate response (translated text or a chatbot response).
4. **Text-to-Speech**:
   - The generated text is converted into speech using the **Google Text-to-Speech (gTTS)** library, and an MP3 audio file is created.
5. **User Interface**:
   - The **Gradio** library is used to create a web interface, allowing users to input text, select actions, and see the output text along with audio.

---

4. Step-by-Step Explanation of the Code

Model and Tokenizer Loading:
python
Copy code
```
language_model =
AutoModelForCausalLM.from_pretrained(language_model_name)
tokenizer = AutoTokenizer.from_pretrained(language_model_name)
```

1.
    - The model (`Qwen/Qwen2-1.5B-Instruct`) and tokenizer are loaded from the Hugging Face library to handle the text input and output. The tokenizer prepares the input text for the model, while the model generates a response.

Processing User Input:
python
Copy code
```
def process_input(input_text, action):
    if action == "Translate to English":
        prompt = f"Please translate the following text into
English : {input_text}"
```

2.
    - Based on the action (like translating or chatting), a prompt is prepared, and the model processes this input to generate output text.

Text-to-Speech:
python
Copy code
```
def text_to_speech(text, lang):
    tts = gTTS(text=text, lang=lang)
    filename = "output_audio.mp3"
    tts.save(filename)
    return filename
```

3.
    - After generating the output text, the `gTTS` library converts the text into speech and saves it as an MP3 file.

Gradio Interface:
python
Copy code
```
iface = gr.Interface(fn=handle_interaction,
inputs=[gr.Textbox(), gr.Dropdown()], outputs=[gr.Textbox(),
gr.Audio()])
```

4.
    - The **Gradio** interface is created with text input and a dropdown for selecting actions (like translating or chatting). The output includes both text and audio.

Running the App:
python
Copy code
```
iface.launch(share=True)
```

5.
    - The `launch()` method runs the app, and users can interact with it through a web interface.

---

5. Outputs

- **Text Output**: The app shows the generated text based on the action (translation or conversation).
- **Audio Output**: The generated text is converted into speech, and the user can listen to it.

---

6. Example of How to Use It

1. Run the code.
2. Open the Gradio interface (a link will be shown in the console).
3. Type some text (e.g., "Hola, ¿cómo estás?").
4. Select an action (e.g., "Translate to English").
5. The app will show the translated text ("Hello, how are you?") and play the speech.

Conclusion

This app allows users to interact with an AI model for translation or chatting. It provides both text and speech outputs, making it a useful tool for language learning or casual conversations with an AI. The app combines advanced AI models, text-to-speech technology, and a simple user interface to create an interactive experience.

Currency Note Detector - Documentation Overview

Introduction

This application uses a pre-trained deep learning model to detect Indian currency notes in real-time through a webcam. The program identifies the currency note, displays the detected denomination on the video feed, and announces it via text-to-speech.

Approach Followed

1. **Pre-trained Model:**
    - A pre-trained Keras model (`currency_classifier.h5`) is used to classify images of currency notes into specific denominations.
    - The model outputs probabilities for each class, with the class representing the detected denomination.
2. **Image Preprocessing:**
    - The captured frame from the webcam is converted to grayscale and processed using edge detection (`Canny`).
    - Contours are identified to locate regions of interest (potential currency notes).
    - Each region is resized and normalized before being passed to the model for prediction.
3. **Prediction and Confidence Filtering:**
    - The model predicts the class of the detected note, representing the denomination.

- A confidence threshold (`CONFIDENCE_THRESHOLD = 0.7`) ensures only predictions with high certainty are processed further.

4. **Voice Feedback:**
   - The detected denomination is announced using the `pyttsx3` library for text-to-speech functionality.

Libraries Used

1. **TensorFlow/Keras:**
   - For loading and using the pre-trained model.
   - Functions: `load_model`, image preprocessing (normalization, resizing).
2. **NumPy:**
   - For handling numerical operations, array transformations, and predictions.
   - Functions: `np.array`, `np.expand_dims`, `np.max`, `np.argmax`.
3. **OpenCV:**
   - For real-time video processing, edge detection, contour extraction, and drawing bounding boxes/text on frames.
   - Functions: `cv2.VideoCapture`, `cv2.cvtColor`, `cv2.GaussianBlur`, `cv2.Canny`, `cv2.findContours`, `cv2.rectangle`, `cv2.putText`.
4. **pyttsx3:**
   - For converting text to speech, enabling voice feedback.
   - Functions: `engine.say`, `engine.runAndWait`.

Models Used

1. **Currency Classifier Model:**
   - The `currency_classifier.h5` model is a pre-trained neural network that identifies Indian currency notes.
   - Input: RGB image of dimensions `(224, 224)`.
   - Output: Probabilities for each class representing currency denominations (10, 20, 50, 100, 200, 500, 2000 INR).

Code Explanation

1. **Function Definitions:**
    - `get_currency_label(prediction)`: Maps model predictions (class IDs) to corresponding denominations based on a predefined dictionary.
    - `speak(text)`: Converts text to speech using `pyttsx3`.
    - `announce_count(count)`: Announces the detected denomination using voice feedback.
2. **Loading the Model:**
    - The pre-trained model is loaded without compilation (`compile=False`) to skip recompilation during usage.
3. **Webcam Initialization:**
    - `cv2.VideoCapture(0)` initializes the webcam for capturing real-time video feed.
4. **Image Processing:**
    - The video frames are converted to grayscale (`cv2.cvtColor`) and processed using Gaussian blur and Canny edge detection to find contours.
5. **Contour Detection:**
    - `cv2.findContours` identifies regions likely containing currency notes. The bounding box for each contour is extracted, and size filtering ensures irrelevant regions are ignored.
6. **Region of Interest (ROI) Preprocessing:**
    - Detected ROIs are resized to `(224, 224)` and normalized to scale pixel values between 0 and 1 before being fed into the model.
7. **Prediction and Confidence Validation:**
    - The model predicts the class and confidence of each detected ROI.
    - Only predictions with confidence > 70% are processed further.
8. **Bounding Box and Annotation:**
    - Bounding boxes are drawn around detected notes using `cv2.rectangle`.
    - The detected denomination is displayed above the bounding box with `cv2.putText`.
9. **Voice Feedback:**
    - The denomination is announced using the `speak` function if confidence criteria are met.
10. **Real-time Display and Exit:**
    - The processed video frame is displayed in a window (`cv2.imshow`).
    - The program exits when the 'q' key is pressed.

Execution Flow

1. The webcam starts and continuously captures frames.
2. Each frame is processed to identify regions resembling currency notes.
3. Detected regions are fed to the classifier model for denomination prediction.
4. High-confidence predictions trigger annotations on the frame and voice announcements.
5. The program loops until the user presses 'q'.

Features

- Real-time currency note detection and classification.
- Visual feedback: Annotates detected currency on the video feed.
- Voice feedback: Announces detected denomination audibly.