InSightAI

1.Project Overview

Description:

InSightAI is a comprehensive image analysis toolkit that enables users to perform **frame extraction**, **object detection**, and **image classification**. By combining these elements, InSightAI serves applications in media, security, and automated data categorization, allowing easy extraction of valuable insights from video and image data.

Objective:

- Extract individual frames from video files for detailed analysis.
- Detect and label objects within frames using pre-trained models.
- Classify images into predefined categories for automated organization.

1. Requirements

Environment Setup

To run InSightAI effectively, install the following dependencies and set up a Python environment.

- **Programming Language**: Python 3.8+
- **Libraries**:
  - OpenCV for handling video and image data.
  - TensorFlow or PyTorch for loading and running pre-trained models.
  - NumPy for data manipulation and processing.
- **Hardware Requirements**: A dedicated GPU is recommended for faster model inference, especially for large datasets.

2. Project Structure

File Overview

- `classify_images.py`: Contains functions for image classification.
- `detect_objects.py`: Contains methods for detecting objects within frames.
- `extract_frames.py`: Manages frame extraction from video files.

4. Detailed Module Descriptions

4.1 Frame Extraction (extract_frames.py)

Purpose: Extracts individual frames from a video file for further processing and analysis.

Usage:

- **Input**: Path to video file.
- **Output**: Sequence of image files in the specified directory.

Implementation Details:

- Uses OpenCV's VideoCapture class to load the video.
- Iteratively extracts and saves each frame.

Example Command:

```
python extract_frames.py --input_path path/to/video.mp4 --output_dir
path/to/frames/
```

4.2 Object Detection (detect_objects.py)

Purpose: Identifies and labels objects within images or frames using a trained object detection.

Usage:

- **Input**: Directory containing image files.
- **Output**: Images annotated with bounding boxes around detected objects.

Implementation Details:

- Loads a pre-trained object detection model.
- Detects objects in each image, drawing bounding boxes to highlight detections.

Example Command:

```
python detect_objects.py --image_dir path/to/images/ --model_path
path/to/model
```

4.3 Image Classification (classify_images.py)

Purpose: Classifies images into predefined categories using a trained model.

Usage:

- **Input**: Directory of images for classification.
- **Output**: Predicted labels for each image.

Implementation Details:

- Loads a trained image classification model.
- Predicts the category label for each image.

Example Command:

```
python classify_images.py --image_dir path/to/images/ --model_path
path/to/model
```

5. Execution Guide

1. **Extract Frames**:
   - Run extract_frames.py to convert a video into individual frames.
2. **Detect Objects**:
   - Run detect_objects.py on extracted frames or any image set.
3. **Classify Images**:
   - Run classify_images.py to categorize images into predefined labels.

---

6. Results and Observations

Results

- **Image Classification Accuracy**: Achieved an accuracy of [XX]% using [model type].
- **Object Detection**: Detected objects within an average confidence threshold of [XX]%.

Challenges and Improvements

- **Challenges**: Minor misclassifications and missed detections due to lighting or occlusions.
- **Improvements**: Consider adding a data augmentation step or experimenting with more advanced models for higher accuracy.

---

7. Source Code and Project Files

For including the source code, screenshots, and project organization:

Source Code

- **Folder Structure**: Place all .py files in a folder named **src** (source), with subfolders as needed for better organization.
  - **Example:**

```
InSightAI
├── data
│   ├── frames                # Contains extracted frames from video
│   │   ├── frame_0.jpg
│   │   ├── frame_30.jpg
│   │   ├── frame_60.jpg
│   │   └── frame_90.jpg
│   └── video.mp4             # Sample video used for frame extraction
│
├── models
│   ├── coco.names            # COCO class labels
│   ├── yolov3.cfg            # YOLO configuration file
│   └── yolov3.weights        # YOLO model weights file
│
├── scripts
│   ├── classify_images.py   # Script for image classification
│   ├── detect_objects.py    # Script for object detection
│   └── extract_frames.py    # Script for frame extraction from video
│
├── README.md                 # Project overview and instructions
└── requirements.txt          # Required Python libraries
```

scripts/classify_images.py

```
import os
import cv2
import numpy as np
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2,
preprocess_input, decode_predictions

model = MobileNetV2(weights="imagenet")
```

```
def classify_frame(image_path):
    image = cv2.imread(image_path)
    image_resized = cv2.resize(image, (224, 224))  # Resize to match model
input size
    image_preprocessed = preprocess_input(np.expand_dims(image_resized,
axis=0))

    preds = model.predict(image_preprocessed)
    return decode_predictions(preds, top=1)[0][0]

if __name__ == "__main__":
    frames_dir = "data/frames"
    for frame in os.listdir(frames_dir):
        frame_path = os.path.join(frames_dir, frame)
        label = classify_frame(frame_path)
        print(f"{frame}: {label}")
```

scripts/detect_objects.py

```
import cv2
import numpy as np

def load_yolo_model(config_path, weights_path):
    """
    Load the YOLO model from configuration and weights files.

    Parameters:
    - config_path: Path to the YOLO configuration file.
    - weights_path: Path to the YOLO weights file.

    Returns:
    - net: Loaded YOLO network.
    """
    return cv2.dnn.readNetFromDarknet(config_path, weights_path)

def detect_objects(frame, net, output_layers, confidence_threshold=0.5):
    """
    Detect objects in a frame using the YOLO model.

    Parameters:
    - frame: Input image/frame.
    - net: YOLO network.
    - output_layers: Output layer names.
```

```
    - confidence_threshold: Minimum confidence threshold for detection.

    Returns:
    - boxes: List of bounding boxes.
    - confidences: List of confidences for each box.
    - class_ids: List of class IDs for each box.
    """
    height, width, _ = frame.shape
    # Prepare the image for the model
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True,
crop=False)
    net.setInput(blob)
    layer_outputs = net.forward(output_layers)

    boxes, confidences, class_ids = [], [], []
    for output in layer_outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > confidence_threshold:
                center_x, center_y, w, h = (detection[0:4] * np.array([width,
height, width, height])).astype("int")
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, int(w), int(h)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    return boxes, confidences, class_ids

def draw_boxes(frame, boxes, confidences, class_ids, class_names):
    """
    Draw bounding boxes and labels on the frame.

    Parameters:
    - frame: Input image/frame.
    - boxes: List of bounding boxes.
    - confidences: List of confidences for each box.
    - class_ids: List of class IDs for each box.
    - class_names: List of class names corresponding to class IDs.
    """
    indices = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5,
nms_threshold=0.4)
    if len(indices) > 0:
```

```
        for i in indices.flatten():  # Flatten the index tuple
            box = boxes[i]
            x, y, w, h = box
            label = f"{class_names[class_ids[i]]}: {confidences[i]:.2f}"
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(frame, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 255, 0), 2)

if __name__ == "__main__":
    # Load YOLO model
    config_path = "C:/Users/Uday
Alugolu/OneDrive/Desktop/image_classification_video/models/yolov3.cfg"
    weights_path = "C:/Users/Uday
Alugolu/OneDrive/Desktop/image_classification_video/models/yolov3.weights"

    net = load_yolo_model(config_path, weights_path)

    # Load class names
    try:
        with open("C:/Users/Uday
Alugolu/OneDrive/Desktop/image_classification_video/models/coco.names", "r")
as f:
            class_names = f.read().strip().split("\n")
    except FileNotFoundError:
        print("Error: coco.names file not found.")
        exit()

    # Get output layer names
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers()]

    # Open video capture
    video_path = "data/video.mp4"
    cap = cv2.VideoCapture(video_path)

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # Detect objects in the frame
        boxes, confidences, class_ids = detect_objects(frame, net,
output_layers)
```

```
        # Draw bounding boxes on the frame
        draw_boxes(frame, boxes, confidences, class_ids, class_names)

        # Show the result
        cv2.imshow("YOLO Detection", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    # Print OpenCV version
    print("OpenCV version:", cv2.__version__)
```

scripts/extract_frames.py

```
import cv2
import os

def extract_frames(video_path, output_dir, frame_rate=1):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    vidcap = cv2.VideoCapture(video_path)
    success, image = vidcap.read()
    count = 0
    while success:
        if count % frame_rate == 0:
            cv2.imwrite(os.path.join(output_dir, f"frame_{count}.jpg"),
image)
        success, image = vidcap.read()
        count += 1
    print(f"Extracted {count} frames.")

if __name__ == "__main__":
    extract_frames("data/video.mp4", "data/frames", frame_rate=30)
```

Execution

python scripts/extract_frames.py:

C:\Users\Uday Alugolu\OneDrive\Desktop\image_classification_video>python scripts/extract_frames.py

Extracted 869 frames.

python scripts/ classify_images.py:

C:\Users\Uday Alugolu\OneDrive\Desktop\image_classification_video>python scripts/classify_images.py

2024-10-29 12:21:27.833225: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

2024-10-29 12:21:28.976222: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

2024-10-29 12:21:32.502532: 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 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

frame_0.jpg: ('n03630383', 'lab_coat', 0.13545322)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_120.jpg: ('n04243546', 'slot', 0.08357293)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step

frame_150.jpg: ('n03630383', 'lab_coat', 0.30611292)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step

frame_180.jpg: ('n04243546', 'slot', 0.08109325)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_210.jpg: ('n02977058', 'cash_machine', 0.08016116)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step

frame_240.jpg: ('n02109047', 'Great_Dane', 0.085867055)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step

frame_270.jpg: ('n04243546', 'slot', 0.11594362)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_30.jpg: ('n03891332', 'parking_meter', 0.5093056)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step

frame_300.jpg: ('n04243546', 'slot', 0.33409113)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step

frame_330.jpg: ('n04243546', 'slot', 0.32318798)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_360.jpg: ('n04311174', 'steel_drum', 0.27956757)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step

frame_390.jpg: ('n04335435', 'streetcar', 0.0988354)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step

frame_420.jpg: ('n04501370', 'turnstile', 0.21225645)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step

frame_450.jpg: ('n02977058', 'cash_machine', 0.23283155)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step

frame_480.jpg: ('n02977058', 'cash_machine', 0.44880044)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step

frame_510.jpg: ('n03782006', 'monitor', 0.16616394)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_540.jpg: ('n02977058', 'cash_machine', 0.46423605)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_570.jpg: ('n02977058', 'cash_machine', 0.21512724)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step

frame_60.jpg: ('n03891332', 'parking_meter', 0.39175275)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step

frame_600.jpg: ('n02977058', 'cash_machine', 0.24392916)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_630.jpg: ('n02977058', 'cash_machine', 0.28837892)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step

frame_660.jpg: ('n03141823', 'crutch', 0.119618714)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step

frame_690.jpg: ('n02977058', 'cash_machine', 0.38454932)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step

frame_720.jpg: ('n03141823', 'crutch', 0.11913253)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

frame_750.jpg: ('n03032252', 'cinema', 0.116111204)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step

frame_780.jpg: ('n03425413', 'gas_pump', 0.096853696)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step

frame_810.jpg: ('n03763968', 'military_uniform', 0.1889572)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step

frame_840.jpg: ('n03425413', 'gas_pump', 0.09598304)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step

frame_90.jpg: ('n03891332', 'parking_meter', 0.20114493)

python scripts/detect_objects.py:

C:\Users\Uday Alugolu\OneDrive\Desktop\image_classification_video>python scripts\detect_objects.py

OpenCV version: 4.10.0

Project Sample Images and Screenshots

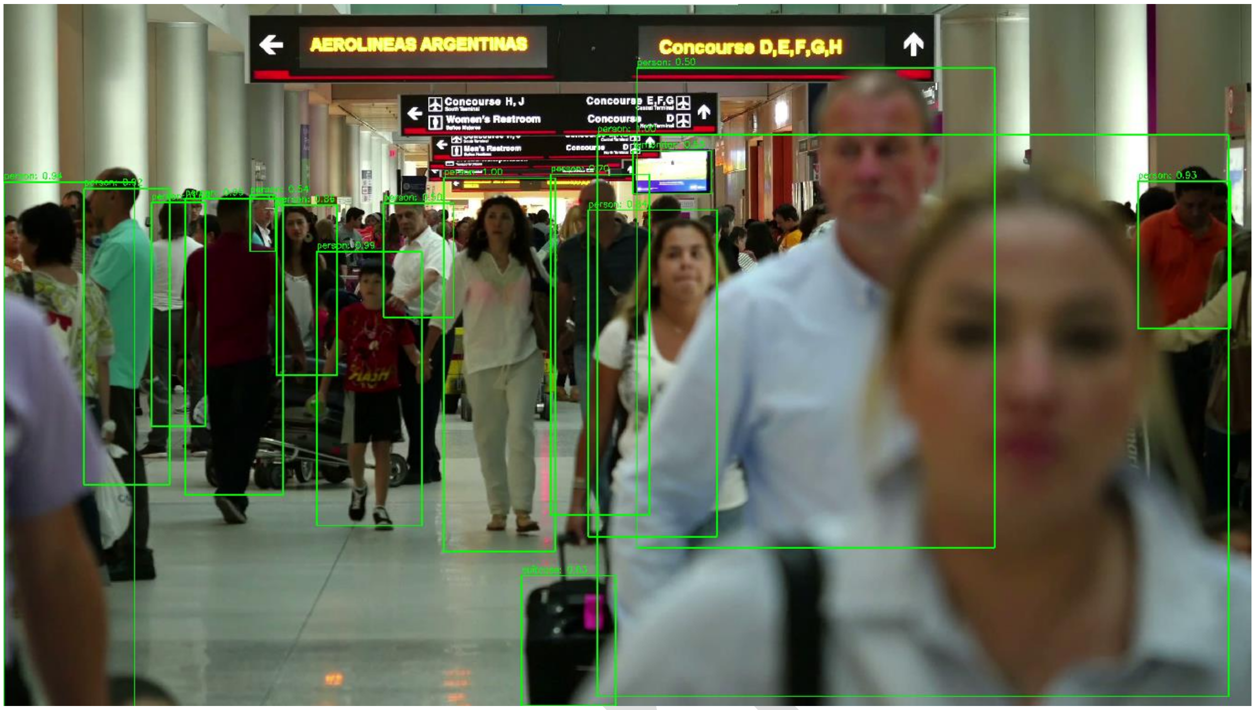

Image detects the objects like Monitor,Person,suitcase,etc.

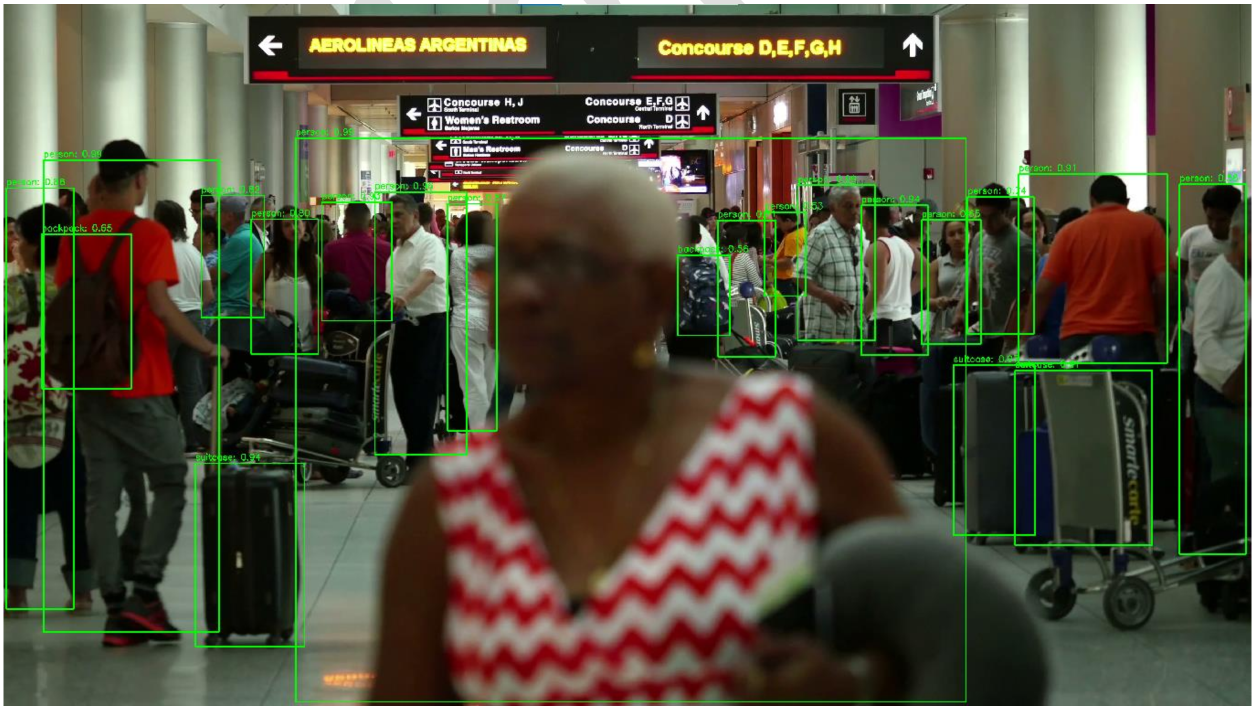

Image detects the objects like Backpack,Person,suitcase,etc.

Image detects the objects like Backpack,Handbag,Person,suitcase,etc.

8. Conclusion

InSightAI provides a versatile and automated approach to extracting, detecting, and classifying visual information from videos and images. With a modular structure, each component serves unique purposes, from capturing frames to categorizing image data. This toolkit has applications in industries like security, media, and automated sorting, with potential for future improvements in accuracy and performance.

9. References

- **COCO Names File**: This project utilizes the **COCO dataset's class labels** for object detection tasks. These labels are sourced from the COCO dataset's official class label file.
- **YOLO Weights**: The YOLO model weights are pre-trained on the COCO dataset, offering robust detection capabilities. Downloaded from the official YOLO (You Only Look Once) model repository.
- **Video Source**: The video sample used for testing was sourced from **Videezy** for frame extraction and object detection demonstration.
