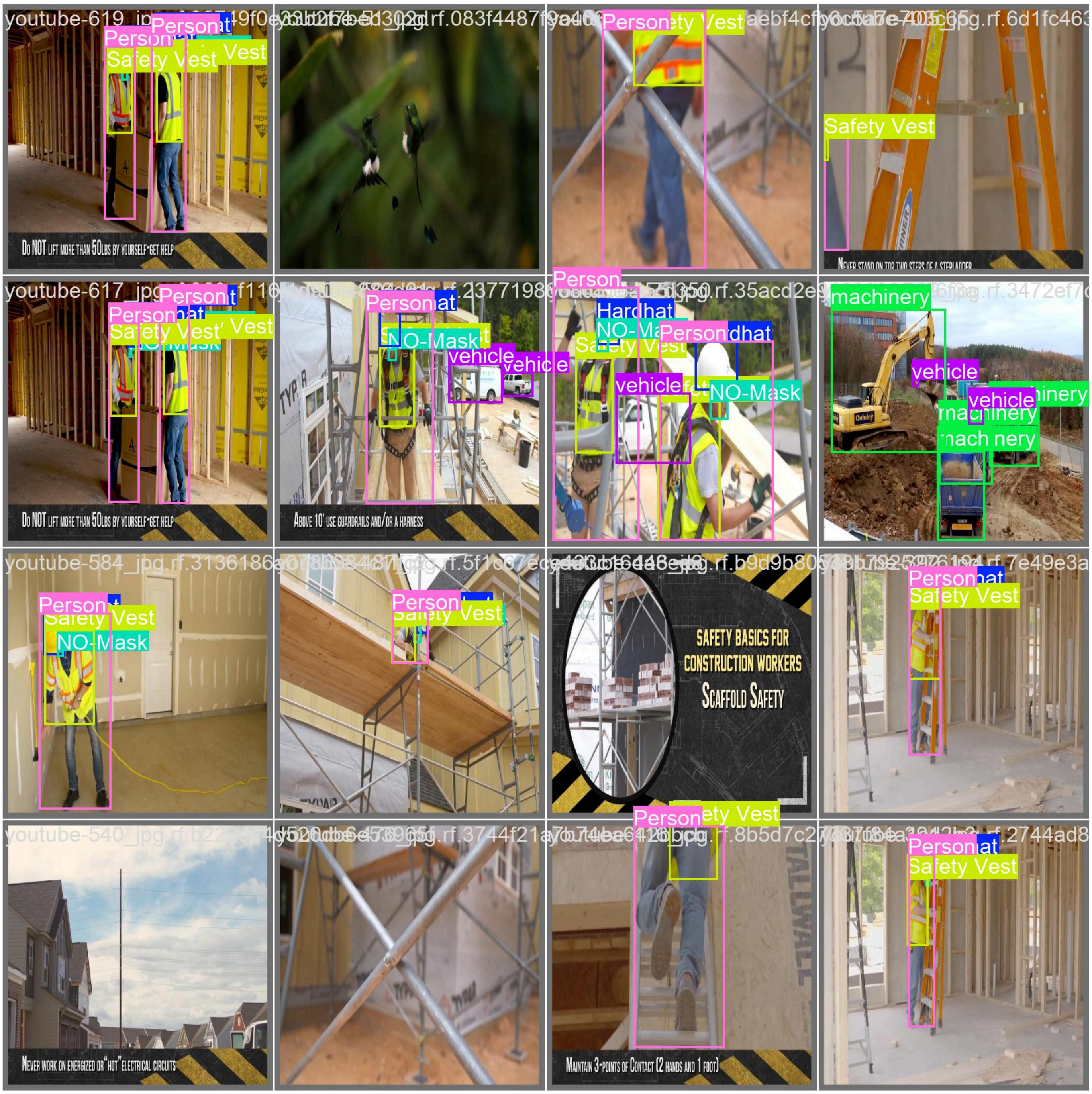
YOLO Model Fine-Tuning and Inference

Overview

This repository contains a Python-based framework for fine-tuning and performing inference with YOLO (You Only Look Once) models. The script provides functionalities such as dataset preparation, downloading model weights, displaying images, and computing dataset statistics.



Features

- Configurable training and inference settings
- Automatic dataset YAML file generation
- Image processing utilities (displaying and analyzing images)
- Dataset statistics computation
- Base model inference using pre-trained YOLO models
- Custom dataset support for training YOLO models
- Model evaluation and visualization tools

Dataset link: https://universe.roboflow.com/roboflow-universe-projects/construction-site-safety

Dependencies

Ensure you have the following Python libraries installed:

```
pip3 install -U -r requirements-dev.txt
```

Required Libraries:

- Python 3.x
- OpenCV (`cv2`)
- PyYAML (`yaml`)
- Pandas (`pandas`)
- Matplotlib (`matplotlib`)
- Seaborn (`seaborn`)
- Pillow (`PIL`)
- Ultralytics (`ultralytics`)
- Requests (`requests`)
- NumPy (`numpy`)

Configuration

The `CONFIG` class defines the core parameters:

- `DEBUG`: Enable debug mode (faster runs with fewer epochs)
- `DISPLAY_IMAGES`: Show images during processing
- `FRACTION`: Fraction of data used for debugging
- `CLASSES`: List of class names for detection
- `EPOCHS`: Number of training epochs
- `BATCH_SIZE`: Training batch size
- `BASE_MODEL`: YOLO model type (e.g., `yolov8s`)
- `CUSTOM_DATASET_DIR`: Path to dataset directory
- `OUTPUT_DIR`: Path for model outputs

- `IMAGE_SIZE`: Image resolution for model input

YOLO and Ultralytics

YOLO (You Only Look Once) is a cutting-edge object detection architecture known for its speed and accuracy. The idea is simple yet powerful—YOLO uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation, which makes it incredibly efficient for real-time detection.

Key Points:

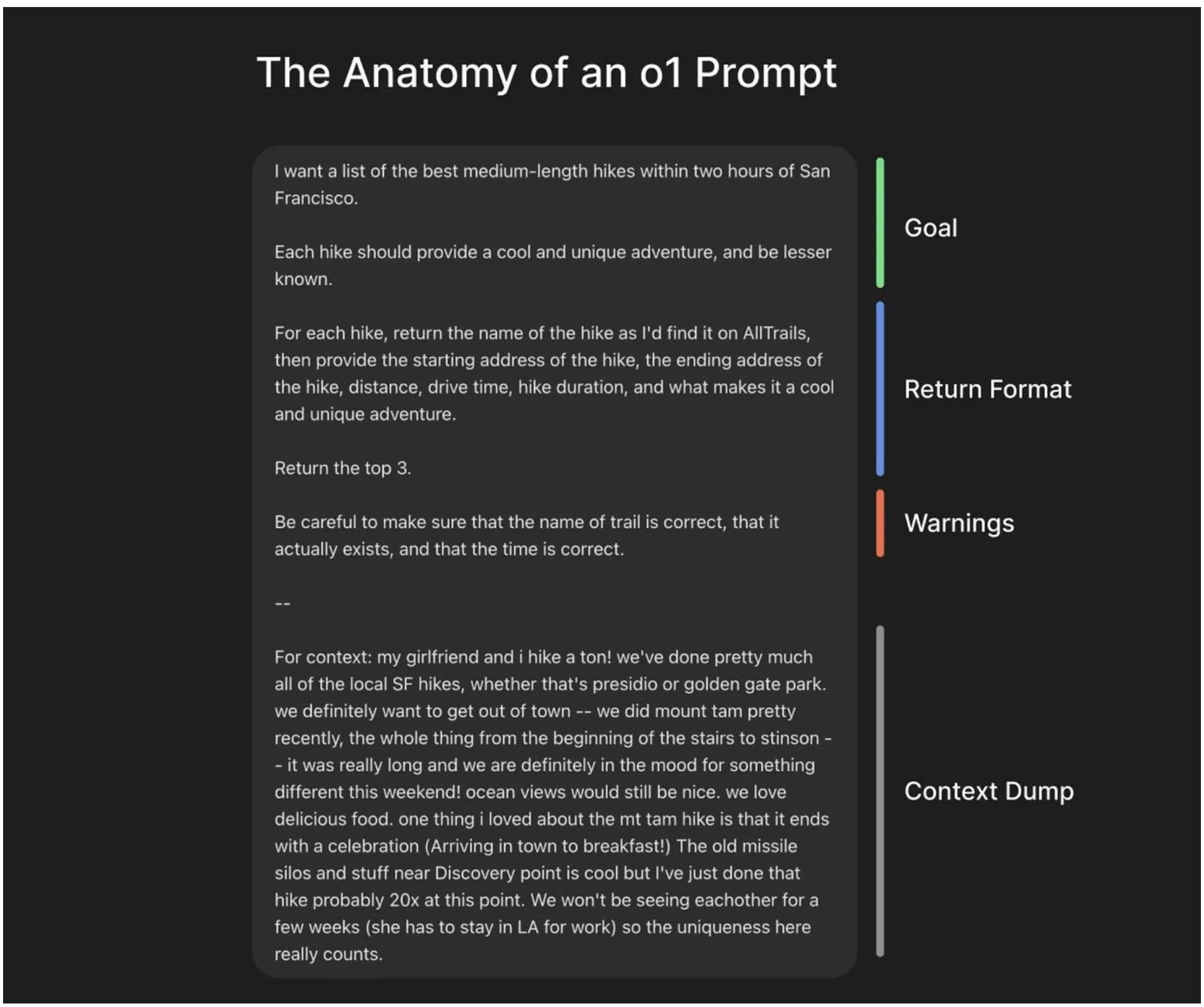
- **Pretrained Weights:**
  YOLO comes with various pretrained weights (like `yolov8n`, `yolov8s`, `yolov8m`, etc.) that have been trained on large, diverse datasets. These models provide a strong starting point, saving you time and computational resources compared to training from scratch.
- **Fine-Tuning:**
  Instead of building a model from zero, you can fine-tune these pretrained weights on your own custom datasets. This is particularly useful for specialized tasks where the objects of interest may not be covered by the original training data.
- **Custom Annotations:**
  Your dataset should include a labels folder containing annotations (bounding boxes around objects). These annotations are crucial for the model to learn how to detect and classify objects relevant to your specific application.
- **Ultralytics Library:**
  Ultralytics offers a user-friendly interface to work with YOLO models. It streamlines the process of training, fine-tuning, and performing inference. When fine-tuning, you load the pretrained weights and adjust them using your custom data and labels.
- **YAML Configuration File:**
  Fine-tuning with Ultralytics requires a YAML file that specifies your dataset configuration. This file should include:
  - `names`: A list of class names.
  - `nc`: The number of classes.
  - `train`: The path to the training images.
  - `val`: The path to the validation images.
  - `test`: (Optional) The path to the test images.

  For example:

```yaml
# data.yaml
names:
    - Hardhat
    - Mask
```

```yaml
    - NO-Hardhat
    - NO-Mask
    - NO-Safety Vest
    - Person
    - Safety Cone
    - Safety Vest
    - machinery
    - vehicle
nc: 10
test: data/test/images
train: data/train/images # Optional
val: data/valid/images
```

- **Best Practices:**
  - **Documentation:** When generating code (especially with an LLM), always provide background context about what you're using and the expected output.
  - **File Paths:** Avoid using absolute paths in Python projects. Instead, save files relative to the project's root directory for better portability and a cleaner structure.



Code Example: Fine-Tuning with YOLO and Ultralytics

Below is a sample code snippet showing how to fine-tune a YOLO model using the Ultralytics library:

```python
from ultralytics import YOLO

# Load the pretrained YOLO model (e.g., yolov8s)
model = YOLO("models/yolov8s.pt")

# Fine-tune the model on your custom dataset
# Note: The 'data.yaml' file should include keys for names, nc, train, val, and
optionally test.
#       Set task="detect" for object detection tasks.
results = model.train(data="data.yaml", task="detect", epochs=50, batch=16)

# After training, you can run inference on new images
results = model("path/to/your/image.jpg")
```

This example demonstrates how to load a pretrained model and fine-tune it using a YAML file that holds the dataset configuration. The `data` parameter in the `model.train()` call is set to `"data.yaml"`, and the `task` parameter is set to `"detect"`, ensuring the model is properly set up for object detection tasks.

Usage

1. Install Dependencies

```
pip install -U -r requirements-dev.txt
```

2. Run the Model

```
python3 run.py
```

3. Modify Configuration

To change core settings, update the following values in the configuration file:

```python
# Set to True for development mode (reduced dataset, fewer epochs)
# Set to False for production mode (full dataset, full training)
DEBUG = True

# Set to True to view images and plots during processing
# Set to False for faster processing without visual output
DISPLAY_IMAGES = False

# Automatically adjusts dataset size based on DEBUG mode
FRACTION = 0.05 if DEBUG else 1.0  # Uses 5% of data when DEBUG is True
```

```
# Automatically adjusts training epochs based on DEBUG mode
EPOCHS = 3 if DEBUG else 50  # Uses 3 epochs when DEBUG is True
```

4. View MLflow UI

After training is completed, run the following command to launch the MLflow UI:

```
mlflow ui --backend-store-uri runs/mlflow
```

Directory Structure

```
.
├── data/
│   ├── train/
│   ├── valid/
│   ├── test/
├── models/
│   ├── yolov8s.pt
├── output/
│   ├── runs/
│   │   └── mlflow/
│   └── logs/
└── script.py
```