Fish Disease Detection and Tracking with VoVNet and Detectron2

This project provides an end-to-end pipeline for instance segmentation and tracking of healthy and infected fish in underwater videos. It is built on the Detectron2 framework and features a custom VoVNet backbone with advanced attention mechanisms, sophisticated data augmentation techniques, and a robust object tracking system.

Features

- **Advanced Model Architecture**: Implements a custom **VoVNet** backbone featuring **OSA (Once-for-all) Modules** and integrated **CBAM (Convolutional Block Attention Module)** attention. This architecture is designed for efficient and powerful feature extraction, crucial for accurate detection.
- **State-of-the-Art Data Augmentation**:
  - **Copy-Paste Augmentation**: Dynamically pastes fish instances from various images onto training samples, significantly increasing data diversity and model robustness.
  - **Dark Channel Prior Defogging**: An image preprocessing step that algorithmically removes water turbidity and haze from frames, improving image clarity for the model.
- **Robust Object Tracking**: Utilizes the **Hungarian algorithm** with an **Intersection over Union (IoU)** cost matrix to track individual fish across frames, assigning stable IDs and maintaining trajectory paths.
- **End-to-End Workflow**: Provides scripts for every stage of the pipeline:
  1. **Data Preparation**: Convert LabelMe JSON annotations to the required COCO format.
  2. **Training**: Train the custom VoVNet model using a custom trainer.
  3. **Inference & Visualization**: Run inference on videos to detect, segment, and track fish with polished visualizations.

Project Scripts

- `labelme2coco.py` : A utility script to convert image annotations from LabelMe's format into the COCO format required by Detectron2. It automatically splits the data into training and validation sets.
- `train.py` : The main training script. It sets up the custom VoVNet model, registers the dataset, applies data augmentation (Defogging, Copy-Paste), and launches the Detectron2 training loop.
- `vidualize.py` : An advanced inference and visualization script. It performs object tracking with bounding box smoothing, confidence-based coloring, and applies image pre-processing to enhance video quality before detection.
- `video_process.py` : A straightforward inference script that runs detection and tracking on a video, visualizing segmentation masks, bounding boxes, and object paths.

Installation

This project relies on Detectron2. Please ensure your environment meets the prerequisites.

Prerequisites

- Linux or macOS
- Python ≥ 3.8
- PyTorch ≥ 1.8
- `torchvision` that corresponds to your PyTorch version.
- CUDA (if using GPU)

Steps

1. **Clone the Repository**

```
git clone https://gitlab.igem.org/2025/software-tools/ecust-china.git
cd https://gitlab.igem.org/2025/software-tools/ecust-china.git
```

2. **Install Detectron2** Follow the [official Detectron2 installation guide](). For a common setup with CUDA, you can run:

```
python -m pip install 'git+[https://github.com/facebookresearch/detectron2.git](https://
```

3. **Install Other Dependencies**

```
pip install opencv-python-headless scipy tqdm
```

Usage

Follow these steps to prepare your data, train the model, and run inference.

1. Dataset Preparation

If your data is annotated using LabelMe, you must first convert it to the COCO format.

1. Place your images and corresponding LabelMe `.json` files in a single directory.
2. Modify the paths in the `labelme2coco.py` script:

```
# Inside labelme2coco.py
LABELME_DIR       = r"path/to/your/labelme_data"
COCO_OUTPUT_DIR   = r"path/to/your/coco_output_directory"
```

3. Run the script:

```
python labelme2coco.py
```

This will create the output directory with `train` and `val` subfolders containing the images and `train_annotations.json` / `val_annotations.json` files.

2. Training the Model

The `train.py` script handles the entire training process.

1. **Configure Paths**: In `train.py`, update the `data_path` variable to point to the COCO directory you created.

```
# Inside train.py
def setup_datasets():
    data_path = 'path/to/your/coco_output_directory/'
    # ...
```

2. **Start Training**: Launch the training from your terminal. The script is configured to enable Copy-Paste and Defogging augmentations by default.

```
python train.py --config-file mask_rcnn_V_39_FPN_3x.yaml --num-gpus 1
```

   - `--config-file`: The base model configuration file (e.g., for a Mask R-CNN with a VoVNet-FPN backbone).
   - `--num-gpus`: The number of GPUs to use.

   The trained model weights (`model_final.pth`) and logs will be saved in the `result/` directory.

3. Inference and Tracking

Two scripts are provided for running inference on videos. `vidualize.py` is recommended for its superior visualization.

Option A: Advanced Visualization (`vidualize.py`)

This script provides smoothed bounding boxes and dynamic, confidence-based coloring for labels.

- **Run from the command line:**

```
python vidualize.py \
    --input /path/to/your/video.mp4 \
    --output /path/to/output/result.mp4 \
    --config-file mask_rcnn_V_39_FPN_3x.yaml \
    --weights result/model_final.pth
```

Option B: Standard Visualization (`video_process.py`)

This script provides standard visualization, overlaying segmentation masks, boxes, and tracking paths.

- **Run from the command line:**

```
python video_process.py \
    --input /path/to/your/video.mp4 \
    --output /path/to/output/result_standard.mp4 \
    --config-file mask_rcnn_V_39_FPN_3x.yaml \
    --weights result/model_final.pth
```

This README should provide a clear and comprehensive guide to your project.