Documentation of Dassault Fish Detection using YOLOv8

Karthik M Dani

2024-08-23

Dassault-ShrimpMonitoring

Objective:

To develop a deep learning-based system capable of identifying different fish species in images using the YOLOv8n model. The project involved collecting and preparing a dataset, organizing the data, and implementing YOLOv8n for object detection and classification.

Overview:

The project focused on automating the process of fish identification using computer vision and deep learning.

The core of the project was based on the YOLO (You Only Look Once) v8n model, which is well-suited for real-time object detection tasks.

The project was divided into several phases: data organization, labeling, splitting, resizing, and model training.

Project Breakdown:

1. Data Organization: Scripts: count_files.sh, organise.py, split_module.py, arrange.py Purpose: To organize and structure the dataset, ensuring that images and corresponding labels are correctly placed in their respective directories.

Details:

- `count_files.sh`: A Bash script to count files in directories, ensuring that all data is accounted for.
- `organise.py`: Moved images and labels to designated directories, split the dataset into training and validation sets, and removed empty folders.

- `split_module.py`: Further split the dataset into training and validation, ensuring a random and balanced distribution.
- `arrange.py`: Organized the final dataset into a clean structure by moving the train and validation folders into a cleaned_data directory.
- `visualize.py`: To ensure that the labels (bounding boxes) were correctly aligned with the images and to visualize them for validation. The script reads the normalized bounding box coordinates from the label files, converts them into absolute pixel values, and overlays them on the images using OpenCV. This step was crucial to confirm that the labels matched the objects in the images and that the bounding boxes were accurate.
- `resize.py`: To standardize image dimensions across the dataset. Resized all images to 640x640 pixels, ensuring consistency for input into the YOLOv8n model, preserving the aspect ratio and preventing distortion where necessary.

Model Training:

Model: YOLOv8n

To train a YOLOv8n model to detect and classify different fish species in the images.

The YOLOv8n model was chosen for its balance between speed and accuracy, particularly in real-time object detection scenarios.

The training process involved fine-tuning the model on the prepared dataset, adjusting hyperparameters, and validating the model's performance on the validation set.

Outcome:

Successfully developed a YOLOv8n-based system for fish identification with high accuracy.

The model demonstrated robust real-time detection capabilities, making it suitable for deployment in applications requiring quick and accurate fish species identification.

Learning and Skills:

- **Deep Learning**: Gained hands-on experience with object detection models, particularly YOLOv8n
- **Data Handling**: Developed expertise in organizing, processing, and visualizing large datasets.
- **Python and Bash Scripting**: Improved skills in Python for data processing and Bash for automating file management tasks.

- **OpenCV**: Enhanced proficiency in image processing and visualization using OpenCV.

Future Work:

- **Model Deployment**: Plan to deploy the trained model on an edge device for real-time fish identification in the field.
-

  Model Optimization: Further optimize the model for better performance in diverse environments, possibly by augmenting the dataset with more varied images.

More on YOLOv8 Architecture

1. Overview of YOLOv8

YOLO (You Only Look Once) is a real-time object detection system that performs detection tasks in a single pass through the network. YOLOv8 builds on previous versions with improvements in accuracy, speed, and efficiency.

2. YOLOv8 Architecture

YOLOv8 follows the general architecture of previous YOLO models but with several enhancements. Here's a detailed breakdown:

2.1 Backbone

- **Purpose**: Extracts features from the input image.
- **Common Backbone Architectures**:
  - YOLOv8 often uses a variant of the CSPNet (Cross-Stage Partial Networks) or efficient architectures like EfficientNet.
- **Details**:
  - **CSPDarknet53**: A modified Darknet architecture with Cross-Stage Partial connections to improve gradient flow and reduce computational cost.
  - **Layers**:
    - **Convolutional Layers**: Extract low-level features like edges and textures.
    - **Batch Normalization**: Normalizes activations to speed up training and stabilize learning.
    - **Activation Functions**: ReLU (Rectified Linear Unit) for introducing non-linearity.

2.2 Neck

- **Purpose**: Aggregates features from different stages of the backbone to produce feature pyramids.
- **Common Neck Architectures**: PANet (Path Aggregation Network), FPN (Feature Pyramid Network).
- **Details**:
  - **PANet**: Improves feature representation by aggregating features from different levels of the backbone.
  - **Layers**:
    * **Feature Fusion**: Combines high-level features with low-level features.
    * **Convolutional Layers**: For further processing and aggregation of features.

2.3 Head

- **Purpose**: Generates the final predictions, including class labels, bounding boxes, and confidence scores.
- **Common Heads**:
  - YOLOv8 head typically uses a combination of detection heads for different scales.
- **Details**:
  - **Bounding Box Prediction**: Predicts the coordinates of the bounding boxes.
  - **Class Prediction**: Predicts the probability distribution over the classes.
  - **Objectness Score**: Predicts the confidence that an object is present in the bounding box.

2.4 Output Layers

- **Purpose**: Outputs the final detections.
- **Details**:
  - **Detection Output**: Combines bounding box coordinates, class probabilities, and objectness scores.
  - **Non-Maximum Suppression (NMS)**: Post-processing step to remove duplicate bounding boxes and retain only the best detections.

3. Detailed Layer Breakdown

3.1 Convolutional Layer

- **Purpose**: Extract features by applying filters to the input image.
- **Details**:
  - **Kernel/Filter**: A small matrix (e.g., 3x3) that slides over the image.
  - **Stride**: Determines how the filter moves across the image.
  - **Padding**: Adds borders to the image to control the spatial dimensions.

3.2 Batch Normalization

- **Purpose**: Normalize the outputs of the previous layer.
- **Details**:
  - **Mean and Variance Calculation**: Compute the mean and variance for each feature map.
  - **Scaling and Shifting**: Apply learnable parameters to normalize activations.

3.3 Activation Function

- **Purpose**: Introduce non-linearity into the network.
- **Details**:
  - **ReLU (Rectified Linear Unit)**: Outputs zero for negative values and the input itself for positive values.
  - **Leaky ReLU**: Allows a small gradient when the input is negative.

3.4 Detection Head

- **Purpose**: Predicts the object's presence, bounding box, and class.
- **Details**:
  - **Anchor Boxes**: Predefined boxes used to predict bounding box coordinates.
  - **Regression**: Predicts offsets relative to anchor boxes.
  - **Classification**: Predicts the probability distribution over classes.

4. Training Process

- **Data Preparation**:
  - **Annotation**: Bounding boxes and class labels for each object in the images.
  - **Normalization**: Rescale bounding box coordinates to a range [0, 1].
- **Loss Function**:
  - **Bounding Box Loss**: Measures the difference between predicted and true bounding box coordinates.
  - **Objectness Loss**: Measures the confidence of object presence.
  - **Classification Loss**: Measures the difference between predicted and true class probabilities.
- **Optimizer**:
  - **Common Optimizers**: Adam, SGD (Stochastic Gradient Descent).
  - **Learning Rate Scheduling**: Adjusts the learning rate during training to improve convergence.

5. Inference

- **Forward Pass**:
  - **Feature Extraction**: Backbone extracts features.

  - **Feature Aggregation**: Neck combines features from different levels.
  - **Prediction Generation**: Head predicts bounding boxes, objectness scores, and class probabilities.
- **Post-processing**:
  - **Non-Maximum Suppression (NMS)**: Removes duplicate boxes based on Intersection over Union (IoU) thresholds.