Train YOLOv5 Guide

How to train Yolov5

To train YOLOv5, you'll need to follow a few steps to set up your environment and configure the training process. Here's a basic guide to help you get started:

1. Install Dependencies

First, you need to install Python and other dependencies. It is recommended to use Python 3.7 or later.

Install dependencies:

```bash

pip install torch torchvision torchaudio
pip install numpy opencv-python matplotlib
```

Then, clone the YOLOv5 repository from GitHub and install the required packages.

```bash

git clone https://github.com/ultralytics/yolov5.git
cd yolov5
pip install -r requirements.txt
```

2. Prepare Dataset

YOLOv5 works with the **COCO** format or **YOLO format** datasets. Ensure your dataset is in the proper format (you can use tools like LabelImg or Roboflow to annotate images and create a dataset).

Here's an example of how the dataset structure should look:

```kotlin

dataset/
    ├── images/
    │   ├── train/
    │   └── val/
    └── labels/
```

```
        ├── train/
        └── val/
```

Each image should have a corresponding `.txt` label file. For example, `image1.jpg` would have a `image1.txt` file containing the object's bounding box and class information.

3. Modify the YAML File for Your Dataset

YOLOv5 uses a YAML file to define the dataset structure. Here's an example of a dataset YAML configuration:

yaml

```yaml
train: path_to_train_images  # Path to train images
val: path_to_val_images      # Path to validation images

nc: 2  # Number of classes
names: ['class1', 'class2']  # List of class names
```

4. Configure Hyperparameters (Optional)

You can modify the hyperparameters (like learning rate, batch size, etc.) in the training configuration file. These are located under `yolov5/data/hyps/hyp.scratch.yaml`.

5. Start Training

Run the training command by specifying the dataset YAML file and other parameters (like batch size, epochs, etc.). Example:

bash

```bash
python train.py --img 640 --batch 16 --epochs 50 --data path_to_your_dataset.yaml --
weights yolov5s.pt --cache
```

This command trains the model with the following arguments:

- `--img`: Image size (e.g., 640).
- `--batch`: Batch size.
- `--epochs`: Number of epochs to train.
- `--data`: Path to the dataset YAML file.
- `--weights`: Starting weights (you can use a pretrained model, such as `yolov5s.pt`).

- `--cache`: Cache images for faster training.

6. Monitor Training

You can monitor the training process by looking at the output in the terminal. YOLOv5 also saves training logs, and you can view them with TensorBoard:

```bash
tensorboard --logdir=runs/train
```

7. Evaluate Model

After training, you can evaluate the model's performance on a test set:

```bash
python val.py --weights runs/train/exp/weights/best.pt --data
path_to_your_dataset.yaml --img 640
```

8. Inference with Trained Model

Once the model is trained, you can use it for inference on new images:

```bash
python detect.py --weights runs/train/exp/weights/best.pt --source
path_to_image_or_video
```

This will run the detection on the input source and save the results in the `runs/detect` directory.

Let me know if you need any more details or have specific questions about any of the steps!

what is COCO format or YOLO format datasets

The **COCO** (Common Objects in Context) format and **YOLO** format are two common annotation formats used for object detection tasks. Both formats store information about

images and the objects within those images, including their class, position (bounding box), and sometimes additional attributes.

1. COCO Format

The **COCO format** is a widely-used dataset format, particularly for object detection, segmentation, and keypoint tasks. It organizes annotations into a JSON file, which contains details about images, categories (classes), and annotations (bounding boxes, segmentation masks, etc.).

COCO Format Structure:

A COCO dataset typically has the following structure:

```arduino
annotations/
  └── instances_train.json (annotations for training set)
  └── instances_val.json (annotations for validation set)
images/
  ├── 000000000001.jpg
  ├── 000000000002.jpg
  └── ...
```

The `instances_train.json` file contains the annotations in JSON format, which include the following fields:

- **images**: A list of image metadata (ID, filename, width, height).
- **annotations**: A list of annotations, where each annotation contains:
  - `image_id`: The ID of the corresponding image.
  - `category_id`: The ID of the object category (class).
  - `bbox`: The bounding box `[x, y, width, height]`.
  - `area`: The area of the bounding box (optional).
  - `iscrowd`: Whether the object is a "crowd" (optional).
- **categories**: A list of object categories, with each category having a unique `id` and `name`.

Example COCO JSON Annotation:

```json
```

```
{
  "images": [
    {
      "id": 1,
      "file_name": "000000000001.jpg",
      "width": 640,
      "height": 480
    }
  ],
  "annotations": [
    {
      "id": 1,
      "image_id": 1,
      "category_id": 1,
      "bbox": [100, 150, 200, 250],
      "area": 50000,
      "iscrowd": 0
    }
  ],
  "categories": [
    {
      "id": 1,
      "name": "person"
    },
    {
      "id": 2,
      "name": "car"
    }
  ]
}
```

In this example:

- The image `000000000001.jpg` contains a bounding box for the object labeled as "person".
- The bounding box is `[100, 150, 200, 250]`, where `(100, 150)` is the top-left corner and `(200, 250)` is the width and height.

2. YOLO Format

The **YOLO format** is simpler and is commonly used with the YOLO (You Only Look Once) object detection models. It uses a separate text file for each image where each line corresponds to an annotation for one object. The annotations include the class ID and the normalized coordinates of the bounding box (relative to the image size).

YOLO Format Structure:

For each image, there is a corresponding `.txt` file with the same name as the image file. Each line in the `.txt` file contains the following fields:

```php-template
<class_id> <x_center> <y_center> <width> <height>
```

Where:

- **class_id**: The ID of the object class (starting from 0).
- **x_center** and **y_center**: The normalized center of the bounding box (relative to the image width and height).
- **width** and **height**: The normalized width and height of the bounding box (relative to the image width and height).

The bounding box values in YOLO format are normalized, meaning they are between 0 and 1, regardless of the actual image dimensions.

Example YOLO Annotation:

For an image `image1.jpg`, the corresponding annotation file `image1.txt` might look like this:

```
0 0.5 0.5 0.4 0.6
1 0.7 0.3 0.2 0.3
```

In this example:

- The first line corresponds to an object of class `0` (e.g., "person"), with the bounding box centered at `(0.5, 0.5)` (50% of the image width and height), with width `0.4` and height `0.6` (40% and 60% of the image width and height).
- The second line corresponds to an object of class `1` (e.g., "car"), with the bounding box centered at `(0.7, 0.3)` and dimensions `0.2` and `0.3`.

Dataset Structure for YOLO Format:

```kotlin
dataset/
  ├── images/
  │   ├── train/
  │   └── val/
  └── labels/
      ├── train/
      └── val/
```

- **images/train/** and **images/val/**: Contain the image files for training and validation.
- **labels/train/** and **labels/val/**: Contain the corresponding `.txt` annotation files.

Key Differences:

- **COCO Format**: Uses a JSON file to store annotations and can handle complex tasks like segmentation, keypoints, and multiple annotations for an object. It also provides more detailed information (e.g., area, segmentation masks).
- **YOLO Format**: Is much simpler, with each image having a single `.txt` file containing the bounding box coordinates and class ID. It is easy to work with and is optimized for YOLO-based models.

Which Format to Use?

- If you are working with **YOLOv5** or any YOLO-based model, you will likely need to convert your dataset to **YOLO format**.
- If you are working with **COCO-based models** (like Mask R-CNN or other models trained on the COCO dataset), you should use the **COCO format**.

Let me know if you need any further clarification or help with converting between formats!

explain file structure of yolov5

The YOLOv5 repository has a specific directory structure designed to organize the model, dataset, training scripts, and other resources efficiently. Here's an overview of the typical file structure of YOLOv5:

1. Root Directory

```kotlin
yolov5/
├── data/
├── models/
├── utils/
├── runs/
├── weights/
├── datasets/
├── train.py
├── test.py
├── detect.py
├── val.py
├── export.py
├── requirements.txt
├── README.md
└── ... (other files)
```

2. Detailed Explanation of Each Directory/File

`data/`

This directory contains configuration files related to the datasets you will use for training and evaluation.

- `data.yaml`: This is the main file for dataset configuration. It defines the paths to the training and validation data, as well as the number of classes and their names (for object detection). Example:

  ```yaml
  train: ../datasets/train/
  val: ../datasets/val/
  nc: 2
  names: ['person', 'car']
  ```

  - `train`: Path to the training dataset.
  - `val`: Path to the validation dataset.
  - `nc`: The number of classes.
  - `names`: A list of class names.

`models/`

This directory contains the model architecture configurations (YAML files) used for YOLOv5.

- `yolov5s.yaml`, `yolov5m.yaml`, `yolov5l.yaml`, `yolov5x.yaml`: These are the configuration files for different sizes of the YOLOv5 model (small, medium, large, and extra-large). They define the layers, number of filters, and other parameters for the specific model variant. Example (in `yolov5s.yaml`):

  ```yaml
  nc: 80
  depth_multiple: 0.33
  width_multiple: 0.50
  anchors:
    - [10,13, 16,30, 33,23]
    - [30,61, 62,45, 59,119]
    - [116,90, 156,198, 373,326]
  ```

`utils/`

This folder contains utility functions that are used throughout the YOLOv5 codebase.

- `datasets.py`: Functions for loading and processing datasets.
- `general.py`: General-purpose functions, such as for file handling and logging.
- `loss.py`: Functions to compute the loss during training.
- `metrics.py`: Functions to calculate evaluation metrics like mAP (mean Average Precision).

`runs/`

This directory is where the outputs from your training are saved. It stores the results of the training process, including weights and logs.

- `train/`: Contains the output from each training session, such as weights, training logs, and performance metrics.
- `detect/`: Stores the output of the inference (detection) process, including annotated images or video.

For example:

```lua
```

```
runs/
  ├── train/
  │   ├── exp/
  │   │   ├── weights/
  │   │   │   ├── last.pt
  │   │   │   └── best.pt
  │   │   ├── results.txt
  │   │   └── ... (other logs and outputs)
  └── detect/
      └── exp/
          └── images/
              ├── img1.jpg
              └── img2.jpg
```

`weights/`

This folder is where pretrained weights are stored.

- `yolov5s.pt`, `yolov5m.pt`, `yolov5l.pt`, `yolov5x.pt`: Pretrained weights for the various sizes of the YOLOv5 model. You can use these pretrained weights to either fine-tune the model on your own dataset or perform inference directly.

`datasets/`

This folder is typically used to store datasets, especially for testing or debugging. It is not used for actual training data, which is typically stored outside of the repository.

Main Scripts in Root Directory:

- `train.py`: The main script to train the YOLOv5 model. It allows you to specify the dataset, model architecture, hyperparameters, and other training options. Example usage:

  ```bash
  python train.py --data data.yaml --weights yolov5s.pt --epochs 50
  ```

- `test.py`: A script for evaluating the trained model on a test dataset. It computes the performance metrics. Example usage:

  ```bash
  python test.py --data data.yaml --weights runs/train/exp/weights/best.pt
  ```

- `detect.py`: A script to perform inference (detection) on images or videos using a trained model. You can use this to run detection on new images or videos. Example usage:

  ```bash
  python detect.py --weights runs/train/exp/weights/best.pt --source path_to_image_or_video
  ```

- `val.py`: A script for validating the model during training or on a separate validation set.
- `export.py`: A script to convert the model to other formats like TensorFlow, CoreML, or ONNX, for easier deployment.

Other Files:

- `requirements.txt`: A file listing all the Python dependencies required for YOLOv5 to run. It includes libraries such as `torch`, `opencv`, `numpy`, etc.
- `README.md`: The file that provides an overview of the YOLOv5 repository, installation instructions, and usage details.

This structure is designed to be modular, enabling easy experimentation with different configurations and models. It separates the training, evaluation, and inference processes while also keeping track of the outputs for each stage.

explain different parameters we can change

When training YOLOv5, you can customize several parameters to optimize the training process or adjust the model to suit your specific needs. Here's a detailed explanation of the different parameters you can change, which are passed either through the command line or adjusted in configuration files:

1. Training Parameters (Command-Line)

These parameters are passed when you run `train.py` via the command line:

1.1 General Parameters:

- `--img`:

    - **Description**: Image size (in pixels) used for training and inference. Typically set to 640 for a balanced trade-off between speed and accuracy.
    - **Example**: `--img 640`
    - **Default**: 640
- `--batch`:
    - **Description**: Batch size for training. It controls how many images are processed in parallel during each iteration. Larger batch sizes speed up training but require more GPU memory.
    - **Example**: `--batch 16`
    - **Default**: 16 (depends on available GPU memory)
- `--epochs`:
    - **Description**: Number of epochs to train the model. An epoch represents one full pass over the entire training dataset.
    - **Example**: `--epochs 50`
    - **Default**: 300 (can be changed depending on the dataset)
- `--data`:
    - **Description**: Path to the dataset YAML configuration file. This file defines the locations of your training and validation images, as well as the number of classes.
    - **Example**: `--data data.yaml`
- `--weights`:
    - **Description**: Pretrained model weights file or a custom checkpoint file to continue training from. Common options are `yolov5s.pt`, `yolov5m.pt`, etc.
    - **Example**: `--weights yolov5s.pt`
    - **Default**: Pretrained weights, usually `yolov5s.pt` for small models
- `--cache`:
    - **Description**: Cache images for faster training. It stores images in RAM for faster access during training.
    - **Example**: `--cache`
    - **Default**: Disabled
- `--image-weights`:

- **Description**: Use weighted image sampling to improve the training on imbalanced datasets by giving higher weights to certain images.
- **Example**: `--image-weights`
- **Default**: Disabled

- `--device`:
  - **Description**: Device on which to train the model. Can be `cpu`, `cuda` (for GPU), or specify multiple GPUs like `cuda:0,1`.
  - **Example**: `--device 0`
  - **Default**: `cuda` (if a compatible GPU is available)
- `--project`:
  - **Description**: The root directory to save the training results.
  - **Example**: `--project /path/to/save/`
  - **Default**: `runs/train`
- `--name`:
  - **Description**: The name of the experiment or training session. This is used to create a folder inside the `runs/train` directory to save the results.
  - **Example**: `--name my_model`
  - **Default**: `exp`

1.2 Hyperparameter Parameters:

You can customize the hyperparameters to fine-tune the training process for better performance. These can be defined either in the `hyp.scratch.yaml` file or as command-line arguments:

- `--lr0` (initial learning rate):
  - **Description**: The starting learning rate. This is the learning rate at the beginning of training and can affect how quickly the model converges.
  - **Example**: `--lr0 0.01`
  - **Default**: 0.01
- `--lrf` (learning rate final):
  - **Description**: The final learning rate after the scheduler decays it.

- **Example**: `--lrf 0.1`
- **Default**: 0.2

- `--momentum`:
  - **Description**: The momentum factor for the optimizer. Momentum helps smooth out the gradients and accelerates convergence.
  - **Example**: `--momentum 0.937`
  - **Default**: 0.937
- `--weight-decay`:
  - **Description**: Weight decay (L2 regularization) to prevent overfitting by penalizing large weights.
  - **Example**: `--weight-decay 0.0005`
  - **Default**: 0.0005
- `--hsv_h`, `--hsv_s`, `--hsv_v`:
  - **Description**: Hyperparameters for color augmentation. Controls the hue, saturation, and value changes applied to images during training.
  - **Example**: `--hsv_h 0.015` (randomly adjust hue by 1.5%)
  - **Default**: Small values for HSV augmentations (e.g., `0.015`)

1.3 Augmentation and Data Parameters:

These control how the images are preprocessed or augmented during training:

- `--augment`:
  - **Description**: Enable image augmentation to improve model generalization by applying transformations like rotation, scaling, and color adjustments.
  - **Example**: `--augment`
  - **Default**: Enabled
- `--flipud` (Vertical Flip Probability):
  - **Description**: The probability of flipping the images vertically during training.
  - **Example**: `--flipud 0.5`
  - **Default**: 0.0 (Disabled)
- `--fliplr` (Horizontal Flip Probability):

- **Description**: The probability of flipping images horizontally during training.
- **Example**: `--fliplr 0.5`
- **Default**: 0.5 (Enabled)

- `--mixup`:
  - **Description**: Mix images and labels during training to increase data variety. Mixup increases model robustness.
  - **Example**: `--mixup 0.2`
  - **Default**: 0.0 (Disabled)

1.4 Model Parameters:

These are model-specific parameters that can be adjusted by specifying a different model architecture or custom configurations.

- `--cfg`:
  - **Description**: Use a custom model configuration file instead of the default ones (e.g., yolov5s.yaml, yolov5m.yaml).
  - **Example**: `--cfg models/custom_model.yaml`

1.5 Evaluation Parameters:

These parameters are used during the evaluation phase of training.

- `--iou-thres`:
  - **Description**: Intersection over union (IoU) threshold used to calculate mAP during validation. It defines the minimum overlap required to consider a detection as correct.
  - **Example**: `--iou-thres 0.5`
  - **Default**: 0.5
- `--conf-thres`:
  - **Description**: Confidence threshold for the object detector. Detections below this confidence level are discarded.
  - **Example**: `--conf-thres 0.25`
  - **Default**: 0.001

---

2. Hyperparameter File (`hyp.scratch.yaml`)

This YAML file contains hyperparameters used during the training process, which are specific to the optimizer, learning rate, augmentation, etc. You can modify this file to change the default training behavior:

```yaml
# Hyperparameters (default values, adjustable)
lr0: 0.01  # Initial learning rate
lrf: 0.2   # Learning rate final
momentum: 0.937  # Momentum for SGD
weight_decay: 0.0005  # Weight decay
hsv_h: 0.015  # Hue augmentation
hsv_s: 0.7    # Saturation augmentation
hsv_v: 0.4    # Value augmentation
...
```

You can adjust these values according to the characteristics of your dataset and the specific model size you are using.

These are the most important parameters you can change to influence the performance of YOLOv5. By adjusting them, you can optimize the training process, improve model accuracy, or reduce overfitting, depending on your dataset and training goals.

explain different parameters

Sure! Here's an explanation of the different parameters you can change when training **YOLOv5**. These parameters help you control various aspects of training, including dataset configuration, model architecture, and training behavior.

1. Dataset and Configuration Parameters

- `--data`:
  - **Description**: Path to the dataset configuration YAML file.
  - **Example**: `--data data.yaml`
  - **Default**: `data.yaml`

    - **Usage**: The dataset YAML file should define paths to training and validation data, number of classes, and class names.
- `--batch-size`:
    - **Description**: Number of images processed in parallel during each training step.
    - **Example**: `--batch-size 16`
    - **Default**: 16 (depends on the GPU memory)
- `--img-size`:
    - **Description**: Size of the image used for training. The image is resized to the specified width and height.
    - **Example**: `--img-size 640`
    - **Default**: 640 (recommended for balanced performance)
- `--epochs`:
    - **Description**: Number of epochs for training. An epoch is one full pass through the training dataset.
    - **Example**: `--epochs 50`
    - **Default**: 300 (you can adjust this based on your dataset and requirements)
- `--workers`:
    - **Description**: Number of CPU workers used for loading the data in parallel.
    - **Example**: `--workers 8`
    - **Default**: 8
- `--image-weights`:
    - **Description**: Enable image-weighted sampling to give more importance to underrepresented images.
    - **Example**: `--image-weights`
    - **Default**: Disabled
- `--device`:
    - **Description**: Device to use for training (GPU or CPU). For GPU, specify the device ID (e.g., `0`, `cuda:0`, `cuda:1`), or use `cpu`.
    - **Example**: `--device cuda:0`

- **Default**: `cuda` (if GPU is available)

2. Model Parameters

- `--weights`:
  - **Description**: Path to the initial weights file (usually pre-trained). You can start from a pre-trained model or continue training from a saved checkpoint.
  - **Example**: `--weights yolov5s.pt`
  - **Default**: Pre-trained weights (e.g., `yolov5s.pt` for small model)
- `--cfg`:
  - **Description**: Path to a custom model architecture configuration file.
  - **Example**: `--cfg models/custom_model.yaml`
  - **Default**: Default model architecture configuration (e.g., `yolov5s.yaml`, `yolov5m.yaml`)
- `--img-size`:
  - **Description**: Image size for resizing input images. Standard values are usually 640x640, but you can experiment with different sizes.
  - **Example**: `--img-size 640`
  - **Default**: 640 (for fast training and inference)
- `--adam`:
  - **Description**: Use the Adam optimizer instead of SGD (Stochastic Gradient Descent).
  - **Example**: `--adam`
  - **Default**: Disabled (SGD by default)
- `--multi-scale`:
  - **Description**: Enables multi-scale training, where image sizes vary during training to increase generalization.
  - **Example**: `--multi-scale`
  - **Default**: Disabled

3. Learning Rate and Optimizer Parameters

- `--lr0`:

- **Description**: Initial learning rate (starting point for the learning rate).
- **Example**: `--lr0 0.01`
- **Default**: 0.01

- `--lrf`:
  - **Description**: Learning rate final (the learning rate at the end of the training process).
  - **Example**: `--lrf 0.1`
  - **Default**: 0.2
- `--momentum`:
  - **Description**: Momentum value for the optimizer, which helps to accelerate convergence.
  - **Example**: `--momentum 0.937`
  - **Default**: 0.937
- `--weight-decay`:
  - **Description**: Weight decay (L2 regularization) to reduce overfitting.
  - **Example**: `--weight-decay 0.0005`
  - **Default**: 0.0005
- `--scheduler`:
  - **Description**: Type of learning rate scheduler. You can use cosine annealing, step decay, etc.
  - **Example**: `--scheduler cosine`
  - **Default**: `linear` scheduler

4. Data Augmentation Parameters

- `--hsv-h`:
  - **Description**: The maximum hue change for the dataset (affects color variation).
  - **Example**: `--hsv-h 0.015`
  - **Default**: 0.015
- `--hsv-s`:
  - **Description**: The maximum saturation change.

- **Example**: `--hsv-s 0.7`
- **Default**: 0.7

- `--hsv-v`:
  - **Description**: The maximum value (brightness) change.
  - **Example**: `--hsv-v 0.4`
  - **Default**: 0.4

- `--flipud`:
  - **Description**: Whether to randomly flip images vertically (useful for datasets with symmetry).
  - **Example**: `--flipud`
  - **Default**: Disabled

- `--fliplr`:
  - **Description**: Whether to randomly flip images horizontally.
  - **Example**: `--fliplr`
  - **Default**: Enabled

- `--scale`:
  - **Description**: Scale images randomly within a specified range.
  - **Example**: `--scale 0.5`
  - **Default**: 0.5

- `--rotate`:
  - **Description**: Rotate images randomly to add variety.
  - **Example**: `--rotate 10`
  - **Default**: 10 degrees

5. Evaluation Parameters

- `--conf-thres`:
  - **Description**: Confidence threshold for detecting an object during inference.
  - **Example**: `--conf-thres 0.4`
  - **Default**: 0.5

- `--iou-thres`:
  - **Description**: IOU (Intersection Over Union) threshold for non-maximum suppression (NMS).
  - **Example**: `--iou-thres 0.45`
  - **Default**: 0.45
- `--save-json`:
  - **Description**: Whether to save results in JSON format (for COCO metrics).
  - **Example**: `--save-json`
  - **Default**: Disabled
- `--task`:
  - **Description**: Task type: `train`, `test`, `val`, or `detect`. Determines which action to take when running a script.
  - **Example**: `--task test`
  - **Default**: `train`

6. Miscellaneous Parameters

- `--exist-ok`:
  - **Description**: Allow overwriting existing results in the output directory.
  - **Example**: `--exist-ok`
  - **Default**: Disabled
- `--weights`:
  - **Description**: Specify the path to pre-trained weights.
  - **Example**: `--weights yolov5s.pt`
  - **Default**: `yolov5s.pt`

Summary of Common Parameters for YOLOv5:

- **Dataset & Configuration**: `--data`, `--img-size`, `--batch-size`, `--epochs`
- **Training Control**: `--weights`, `--lr0`, `--momentum`, `--device`

- **Augmentation**: `--hsv-h`, `--hsv-s`, `--flipud`, `--fliplr`
- **Evaluation**: `--conf-thres`, `--iou-thres`

By customizing these parameters, you can fine-tune the YOLOv5 training process to suit your specific use case, dataset, and hardware environment.