In [ ]:
~~~python
from copy import copy

import numpy as np

from ultralytics.data import build_dataloader, build_yolo_dataset
from ultralytics.engine.trainer import BaseTrainer
from ultralytics.models import yolo
from ultralytics.nn.tasks import DetectionModel
from ultralytics.utils import LOGGER, RANK
from ultralytics.utils.plotting import plot_images, plot_labels, plot_results
from ultralytics.utils.torch_utils import de_parallel, torch_distributed_zero_first


class DetectionTrainer(BaseTrainer):
    """
    A class extending the BaseTrainer class for training based on a detection model.

    Example:
        ```python
        from ultralytics.models.yolo.detect import DetectionTrainer

        args = dict(model='yolov8n.pt', data='coco8.yaml', epochs=3)
        trainer = DetectionTrainer(overrides=args)
        trainer.train()
        ~~~
    """

    def build_dataset(self, img_path, mode='train', batch=None):
        """
        Build YOLO Dataset.

        Args:
            img_path (str): Path to the folder containing images.
            mode (str): `train` mode or `val` mode, users are able to customize different augmentations for each mode.
            batch (int, optional): Size of batches, this is for `rect`. Defaults to None.
        """
        gs = max(int(de_parallel(self.model).stride.max() if self.model else 0), 32)
        return build_yolo_dataset(self.args, img_path, batch, self.data, mode=mode, rect=mode == 'val', stride=gs)

    def get_dataloader(self, dataset_path, batch_size=16, rank=0, mode='train'):
        """Construct and return dataloader."""
        assert mode in ['train', 'val']
        with torch_distributed_zero_first(rank): # init dataset *.cache only once if DDP
            dataset = self.build_dataset(dataset_path, mode, batch_size)
        shuffle = mode == 'train'
        if getattr(dataset, 'rect', False) and shuffle:
            LOGGER.warning("WARNING ⚠️ 'rect=True' is incompatible with DataLoader shuffle, setting shuffle=False")
            shuffle = False
        workers = self.args.workers if mode == 'train' else self.args.workers * 2
        return build_dataloader(dataset, batch_size, workers, shuffle, rank) # return dataloader

    def preprocess_batch(self, batch):
        """Preprocesses a batch of images by scaling and converting to float."""
        batch['img'] = batch['img'].to(self.device, non_blocking=True).float() / 255
        return batch

    def set_model_attributes(self):
        """Nl = de_parallel(self.model).model[-1].nl # number of detection layers (to scale hyps)."""
        # self.args.box *= 3 / nl # scale to layers
        # self.args.cls *= self.data["nc"] / 80 * 3 / nl # scale to classes and layers
        # self.args.cls *= (self.args.imgsz / 640) ** 2 * 3 / nl # scale to image size and layers
        self.model.nc = self.data['nc'] # attach number of classes to model
        self.model.names = self.data['names'] # attach class names to model
        self.model.args = self.args # attach hyperparameters to model
        # TODO: self.model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device) * nc

    def get_model(self, cfg=None, weights=None, verbose=True):
        """Return a YOLO detection model."""
        model = DetectionModel(cfg, nc=self.data['nc'], verbose=verbose and RANK == -1)
        if weights:
            model.load(weights)
        return model

    def get_validator(self):
        """Returns a DetectionValidator for YOLO model validation."""
        self.loss_names = 'box_loss', 'cls_loss', 'dfl_loss'
        return yolo.detect.DetectionValidator(self.test_loader, save_dir=self.save_dir, args=copy(self.args))

    def label_loss_items(self, loss_items=None, prefix='train'):
        """
        Returns a loss dict with labelled training loss items tensor.

        Not needed for classification but necessary for segmentation & detection
        """
```
```

```
        keys = [f'{prefix}/{x}' for x in self.loss_names]
        if loss_items is not None:
            loss_items = [round(float(x), 5) for x in loss_items]  # convert tensors to 5 decimal place floats
            return dict(zip(keys, loss_items))
        else:
            return keys

    def progress_string(self):
        """Returns a formatted string of training progress with epoch, GPU memory, loss, instances and size."""
        return ('\n' + '%11s' *
                (4 + len(self.loss_names))) % ('Epoch', 'GPU_mem', *self.loss_names, 'Instances', 'Size')

    def plot_training_samples(self, batch, ni):
        """Plots training samples with their annotations."""
        plot_images(images=batch['img'],
                    batch_idx=batch['batch_idx'],
                    cls=batch['cls'].squeeze(-1),
                    bboxes=batch['bboxes'],
                    paths=batch['im_file'],
                    fname=self.save_dir / f'train_batch{ni}.jpg',
                    on_plot=self.on_plot)

    def plot_metrics(self):
        """Plots metrics from a CSV file."""
        plot_results(file=self.csv, on_plot=self.on_plot)  # save results.png

    def plot_training_labels(self):
        """Create a labeled training plot of the YOLO model."""
        boxes = np.concatenate([lb['bboxes'] for lb in self.train_loader.dataset.labels], 0)
        cls = np.concatenate([lb['cls'] for lb in self.train_loader.dataset.labels], 0)
        plot_labels(boxes, cls.squeeze(), names=self.data['names'], save_dir=self.save_dir, on_plot=self.on_plot)
```

In [1]:

```
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.yaml')  # build a new model from YAML
model = YOLO('yolov8n.pt')  # load a pretrained model (recommended for training)
model = YOLO('yolov8n.yaml').load('yolov8n.pt')  # build from YAML and transfer weights

# Train the model
results = model.train(data='C:/Users/muzam/Desktop/Files/IIT/Deep Learning/Final Project/ultralytics-main/data.yaml', epochs=2
```

```
                   all       1392       1518      0.903      0.802      0.889      0.606



25 epochs completed in 7.015 hours.
Optimizer stripped from runs\detect\train4\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\train4\weights\best.pt, 6.2MB

Validating runs\detect\train4\weights\best.pt...
Ultralytics YOLOv8.0.221 🚀 Python-3.9.13 torch-1.13.0+cpu CPU (AMD Ryzen 7 4800H with Radeon Graphics)
YOLOv8n summary (fused): 168 layers, 3006038 parameters, 0 gradients, 8.1 GFLOPs

                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 44/44 [01:22

                   all       1392       1518      0.902      0.802      0.889      0.606
                pistol       1392        669      0.887      0.786      0.871      0.685
                 Knife       1392        849      0.918      0.818      0.908      0.527
Speed: 1.2ms preprocess, 53.1ms inference, 0.0ms loss, 0.4ms postprocess per image
Results saved to runs\detect\train4
```