dnt API Manual

None

None

None

Table of Contents

1. dnt API Manual

This site is generated with MkDocs and `mkdocstrings` from the Python source in `src/dnt`.

Use the navigation to browse package modules and APIs.

2. Quickstart

1. Install the package in your environment.
2. Run one of the examples in `examples/`.
3. Open the API reference for detailed class and function docs.

3. API Reference

3.1 API Reference

The pages in this section are rendered from module docstrings and object signatures.

3.2 dnt Package

DNT package for detection and tracking.

This package provides detection and tracking functionality.

3.3 Detection

3.3.1 Detection

Detection module for object detection using YOLO.

Detector

```
Detector(
    model: DetectorModel = DetectorModel.YOLO26x,
    weights: str | None = None,
    conf: float = 0.25,
    nms: float = 0.7,
    max_det: int = 300,
    device: str = "auto",
    half: bool = False,
)
```

A wrapper around Ultralytics detection models for running object detection on videos and selected frames.

This class loads a YOLO (v8, v11, 26) or RT-DETR model from a local `models/` directory (or from a user-supplied .pt file) and provides convenience methods to:

- detect objects frame by frame in a video and return results as a pandas DataFrame,
- run detection only on specified frame indices,
- process a batch of videos and save per-video detection text files, and
- query basic video properties (FPS, frame count).

The detector automatically chooses an inference device (`cuda` , `xpu` , `mps` , or `cpu`) when `device="auto"` , and it can optionally enable half-precision inference on GPU.

Parameters:

Name	Type	Description	Default
`model`	`DetectorModel`	Built-in model weights to use (for example `DetectorModel.YOLO26x`). Default is `DetectorModel.YOLO26x` .	`YOLO26x`
`weights`	`str`	Optional custom model weights to load. If relative, path is resolved under `<module_dir>/models/` . Default is None.	`None`
`conf`	`float`	Confidence threshold for detections. Default is `0.25` .	`0.25`
`nms`	`float`	IoU / non-maximum suppression threshold. Default is `0.7` .	`0.7`
`max_det`	`int`	Maximum number of detections per frame. Default is `300` .	`300`
`device`	`(auto, cuda, xpu, cpu, mps)`	Inference device to use. If `"auto"` , the detector will pick an available accelerator first (`cuda` → `xpu` → `mps`) and fall back to CPU. Default is `"auto"` .	`"auto"`
`enable_half`	`bool`	Whether to enable half-precision inference. This is only effective on GPU (CUDA). Default is `False` .	*required*

Notes

- The class expects model weight files to be located under `<module_dir>/models/` when using the built-in weight names.
- Returned detection tables typically contain the columns: `frame, res, x, y, w, h, conf, class` .

Initialize a Detector for Ultralytics YOLO/RT-DETR models.

Parameters:

Name	Type	Description	Default
`model`	`DetectorModel`	Built-in model to use. Default is "yolo26x".	`YOLO26x`
`weights`	`str`	Customized model weights to load. Default is None, which means using the built-in weights in `model` choice.	`None`
`conf`	`float`	Confidence threshold. Default is 0.25.	`0.25`
`nms`	`float`	IoU/NMS threshold. Default is 0.7.	`0.7`
`max_det`	`int`	Maximum detections per frame. Default is 300. In crowded scenes, you may want to increase this.	`300`
`device`	`(auto, cuda, xpu, cpu, mps)`	Inference device. Default is "auto".	`"auto"`
`half`	`bool`	Whether to use half precision (GPU only). Default is False.	`False`

Source code in `src/dnt/detect/yolo/detector.py`

```python
def __init__(
    self,
    model: DetectorModel = DetectorModel.YOLO26x,
    weights: str | None = None,
    conf: float = 0.25,
    nms: float = 0.7,
    max_det: int = 300,
    device: str = "auto",
    half: bool = False,
):
    """Initialize a Detector for Ultralytics YOLO/RT-DETR models.

    Parameters
    ----------
    model : DetectorModel, optional
        Built-in model to use. Default is "yolo26x".
    weights : str, optional
        Customized model weights to load.
        Default is None, which means using the built-in weights in `model` choice.
    conf : float, optional
        Confidence threshold. Default is 0.25.
    nms : float, optional
        IoU/NMS threshold. Default is 0.7.
    max_det : int, optional
        Maximum detections per frame.
        Default is 300. In crowded scenes, you may want to increase this.
    device : {"auto", "cuda", "xpu", "cpu", "mps"}, optional
        Inference device. Default is "auto".
    half : bool, optional
        Whether to use half precision (GPU only). Default is False.

    """
    # Load model
    cwd = Path(__file__).parent.absolute()
    model_dir = cwd / "models"
    if not model_dir.exists():
        os.makedirs(model_dir)

    if weights:
        model_path = Path(weights) if os.path.isabs(weights) else model_dir / weights
    else:
        model_path = model_dir / f"{model.value}"

    # actually load model
    if ("yolo" in str(weights).lower()) or (model in YOLO_MODELS):
        self.model = YOLO(str(model_path))
    elif ("rtdetr" in str(weights).lower()) or (model in RTDETR_MODELS):
        self.model = RTDETR(str(model_path))
    else:
        raise ValueError(
            f"Cannot infer model family from model={model} and weights={weights!r}. "
            "Use a known DetectorModel or provide weights containing 'yolo' or 'rtdetr'."
        )
    self.conf = conf
    self.nms = nms
    self.max_det = max_det

    # device selection
    requested_device = str(device).lower().strip()
    requested_backend = requested_device.split(":", maxsplit=1)[0]
    valid_devices = {"auto", "cuda", "xpu", "mps", "cpu"}
    if requested_backend not in valid_devices:
        raise ValueError(
            f"Invalid device={device!r}. Choose one of {sorted(valid_devices)} or backend:index like 'cuda:0'."
        )

    backend_available = {
        "cuda": torch.cuda.is_available(),
        "xpu": hasattr(torch, "xpu") and hasattr(torch.xpu, "is_available") and torch.xpu.is_available(),
        "mps": hasattr(torch.backends, "mps") and torch.backends.mps.is_available(),
        "cpu": True,
    }

    if requested_backend == "auto":
        auto_priority = ("cuda", "xpu", "mps", "cpu")
        self.device = next(d for d in auto_priority if backend_available[d])
    else:
        self.device = requested_device if backend_available[requested_backend] else "cpu"

    # half precision only makes sense on GPU
    self.half = half and (self.device == "cuda")
```

detect

```python
detect(
    input_video: str,
    iou_file: str | None = None,
    video_index: int | None = None,
    video_tot: int | None = None,
    start_frame: int | None = None,
    end_frame: int | None = None,
```

```
    verbose: bool = True,
    show: bool = False,
    disp_filename: bool = False,
) -> pd.DataFrame
```

Run object detection on a video and return per-frame detections.

Parameters:

Name	Type	Description	Default
`input_video`	`str`	Path to the input video file.	*required*
`iou_file`	`str`	If provided, detection results are written to this file (CSV without header).	`None`
`video_index`	`int`	Index of this video in a batch, used only for progress display.	`None`
`video_tot`	`int`	Total number of videos in the batch, used only for progress display.	`None`
`start_frame`	`int`	Frame index to start detection from. If None or out of range, starts at 0.	`None`
`end_frame`	`int`	Frame index to stop detection at. If None or out of range, uses the last frame.	`None`
`verbose`	`bool`	Whether to show a progress bar. Default is True.	`True`
`show`	`bool`	Whether to display the video frames with detections. Default is False.	`False`
`disp_filename`	`bool`	Whether to show the file name in the progress bar. Default is False.	`False`

Returns:

Type	Description
`DataFrame`	DataFrame with columns: `frame, res, x, y, w, h, conf, class`. If the video cannot be opened or no detections are found, an empty DataFrame with those columns is returned.

Source code in `src/dnt/detect/yolo/detector.py`