OpenFilter Onboarding Guide: First Two Pipelines + Visualization & JSON Outputs

This guide is a handoff-ready onboarding walkthrough for building two starter OpenFilter pipelines on a stock video. It also explains common points of confusion: why your output MP4 may not show bounding boxes, and how to save model results to JSON when you want structured output instead of overlays.

The steps below follow the standard pipeline wiring pattern in the OpenFilter docs (VideoIn → model filter → output filter connected by TCP ports).

However, as written in the original onboarding/Jira-style instructions, "VideoIn → Hub model → VideoOut" does not guarantee visible boxes in the output MP4. Some Hub models emit detections as metadata only, and VideoOut will write clean frames unless another filter draws overlays. This guide shows both the simplest pipelines and the correct visualization options, plus JSON-output options for both pipelines.

Goals & Deliverables

- Pipeline 1 (OCR): VideoIn → FilterOpticalCharacterRecognition → VideoOut
- Pipeline 2 (License Plates): VideoIn → FilterLicensePlateDetection → VideoOut
- Understand when you will or will not see bounding boxes in MP4 outputs
- Learn the two doc-supported ways to visualize boxes: Webvis or an annotation filter
- Learn how to log OCR outputs to JSON instead of relying on overlays
- Learn how to log license plate detections to JSON for inspection/downstream use

Minimum Setup

Create a folder and virtual environment:

```
mkdir openfilter_first_pipelines
cd openfilter_first_pipelines
python -m venv .ofenv
source .ofenv/bin/activate
```

Install OpenFilter with video support:

```
pip install "openfilter[video_in,video_out]"
```

Stock Video

Use the Plainsight tutorial sample video:

```
git clone https://github.com/PlainsightAI/openfilter-tutorial.git
cp openfilter-tutorial/example_video.mp4 ./example_video.mp4
```

Important: the tutorial clip is great for pipeline mechanics, but it may not contain license plates. If there are no plates, you will not see boxes or JSON detections. Use a known plate demo clip if you want guaranteed detections.

Install the Two Hub Models (Chosen for You)

Model A: Optical Character Recognition (OCR)

```
pip install filter-optical-character-recognition
```

```
from filter_optical_character_recognition.filter import FilterOpticalCharacterRecognition
```

Model B: License Plate Detection

```
pip install filter-license-plate-detection
```

```
from filter_license_plate_detection.filter import FilterLicensePlateDetection
```

This model requires a weights file named model.pth. Download it from the filter repo:

```
git clone https://github.com/PlainsightAI/filter-license-plate-detection.git
cd filter-license-plate-detection
make install
cp model.pth ../model.pth
cd ..
```

Core Wiring Rule (from OpenFilter docs)

Filters connect through TCP ports: upstream outputs → downstream sources.

```
outputs="tcp://*:5550"
sources="tcp://localhost:5550"
```

Pipeline 1 (OCR) — MP4 Output with Optional Overlays

Create pipeline1.py:

```
from openfilter.filter_runtime.filter import Filter
from openfilter.filter_runtime.filters.video_in import VideoIn
from openfilter.filter_runtime.filters.video_out import VideoOut
from filter_optical_character_recognition.filter import FilterOpticalCharacterRecognition

if __name__ == "__main__":
    Filter.run_multi([
        (VideoIn, dict(
            sources="file://./example_video.mp4",
            outputs="tcp://*:5550"
        )),

        (FilterOpticalCharacterRecognition, dict(
            sources="tcp://localhost:5550",
            outputs="tcp://*:5552",
            draw_visualization=True,
            visualization_topic="main",
            forward_ocr_texts=True,
        )),

        (VideoOut, dict(
            sources="tcp://localhost:5552",
            outputs="file://./output_1.mp4",
            fps=True,
        )),
    ])
```

Run it:

```
python pipeline1.py
```

Expected output: output_1.mp4. OCR overlays appear only on frames where readable text is detected. If the clip has little/no text, the MP4 may look unchanged — that's expected.

Pipeline 1 Option — Save OCR Results to JSON

If you want structured OCR outputs even when no overlays are visible, log results to JSON. The OCR filter forwards text into frame metadata when forward_ocr_texts=True.

Install the JSON sink filter:

```
pip install filter-json-sink
```

Create pipeline1_json.py:

```
from openfilter.filter_runtime.filter import Filter
from openfilter.filter_runtime.filters.video_in import VideoIn
from filter_optical_character_recognition.filter import FilterOpticalCharacterRecognition
from filter_json_sink.filter import FilterJsonSink

if __name__ == "__main__":
    Filter.run_multi([
        (VideoIn, dict(
            sources="file://./example_video.mp4",
            outputs="tcp://*:5550"
        )),

        (FilterOpticalCharacterRecognition, dict(
            sources="tcp://localhost:5550",
            outputs="tcp://*:5552",
            forward_ocr_texts=True,
        )),

        (FilterJsonSink, dict(
            sources="tcp://localhost:5552",
            output_path="./ocr_results.json",
            json_keys=["ocr_texts"],
            append=True,
        )),
    ])
```

Run it:

```
python pipeline1_json.py
```

You will get ocr_results.json containing per-frame OCR text results.

Pipeline 2 (License Plates) — Minimal MP4 Output

Create pipeline2.py:

```
from openfilter.filter_runtime.filter import Filter
from openfilter.filter_runtime.filters.video_in import VideoIn
from openfilter.filter_runtime.filters.video_out import VideoOut
from filter_license_plate_detection.filter import FilterLicensePlateDetection
```

```
if __name__ == "__main__":
    Filter.run_multi([
        (VideoIn, dict(
            sources="file://./example_video.mp4",
            outputs="tcp://*:5560"
        )),

        (FilterLicensePlateDetection, dict(
            sources="tcp://localhost:5560",
            outputs="tcp://*:5562",
            model_path="./model.pth",
            confidence_threshold=0.30,
            forward_detection_rois=True
        )),

        (VideoOut, dict(
            sources="tcp://localhost:5562",
            outputs="file://./output_2.mp4",
            fps=True,
        )),
    ])
```

Run it:

```
python pipeline2.py
```

Important: this minimal pipeline DOES NOT draw boxes into output_2.mp4. License Plate Detection emits metadata only; VideoOut writes clean frames.

Pipeline 2 Option — Save License Plate Detections to JSON

If you want to inspect detections even when no overlays are drawn, log the detection metadata to a JSON file using the same JSON sink pattern.

Install the JSON sink filter (if you haven't already):

```
pip install filter-json-sink
```

Create pipeline2_json.py:

```
from openfilter.filter_runtime.filter import Filter
from openfilter.filter_runtime.filters.video_in import VideoIn
from filter_license_plate_detection.filter import FilterLicensePlateDetection
from filter_json_sink.filter import FilterJsonSink

if __name__ == "__main__":
    Filter.run_multi([
        (VideoIn, dict(
            sources="file://./example_video.mp4",
            outputs="tcp://*:5560"
        )),

        (FilterLicensePlateDetection, dict(
            sources="tcp://localhost:5560",
            outputs="tcp://*:5562",
            model_path="./model.pth",
            confidence_threshold=0.30,
```

```
            forward_detection_rois=True
        )),

        (FilterJsonSink, dict(
            sources="tcp://localhost:5562",
            output_path="./license_plate_results.json",
            json_keys=["license_plate_detection"],
            append=True,
        )),
    ])
```

Run it:

```
python pipeline2_json.py
```

You will get license_plate_results.json with per-frame detection arrays (boxes, scores, labels). If the clip has no plates, this file will be empty or sparse.

Why Your Output Video May Not Show Bounding Boxes

- Some models draw overlays (OCR with draw_visualization=True); others do not.
- License Plate Detection emits metadata only. No drawing happens in that filter.
- VideoOut does not add overlays. It only writes the frames it receives.
- If your input clip has no plates, there will be no boxes or JSON detections.

This is why the Jira-style instruction “VideoIn → Hub model → VideoOut” runs fine but doesn’t guarantee visible boxes.

Option A — See Boxes Live with Webvis (Fastest Confirmation)

Install Webvis support:

```
pip install "openfilter[webvis,video_in]"
```

Create pipeline2_webvis.py:

```
from openfilter.filter_runtime.filter import Filter
from openfilter.filter_runtime.filters.video_in import VideoIn
from openfilter.filter_runtime.filters.webvis import Webvis
from filter_license_plate_detection.filter import FilterLicensePlateDetection

if __name__ == "__main__":
    Filter.run_multi([
        (VideoIn, dict(
            sources="file://./example_video.mp4!loop",
            outputs="tcp://*:5560"
        )),

        (FilterLicensePlateDetection, dict(
            sources="tcp://localhost:5560",
            outputs="tcp://*:5562",
            model_path="./model.pth",
            confidence_threshold=0.30,
            forward_detection_rois=True
        )),
```

```
        (Webvis, dict(
            sources="tcp://localhost:5562"
        )),
    ])
```

Run it and open http://localhost:8000

Option B — Write MP4s With Boxes (Demo Chain)

Install demo helpers:

```
pip install filter-license-annotation-demo filter-crop
```

Create pipeline2_boxes_to_mp4.py (same as previous doc version):

```
from openfilter.filter_runtime.filter import Filter
from openfilter.filter_runtime.filters.video_in import VideoIn
from openfilter.filter_runtime.filters.video_out import VideoOut

from filter_license_plate_detection.filter import FilterLicensePlateDetection
from filter_crop.filter import FilterCrop
from filter_optical_character_recognition.filter import FilterOpticalCharacterRecognition
from filter_license_annotation_demo.filter import FilterLicenseAnnotationDemo

if __name__ == "__main__":
    Filter.run_multi([
        (VideoIn, dict(
            sources="file://./example_video.mp4",
            outputs="tcp://*:5560"
        )),

        (FilterLicensePlateDetection, dict(
            sources="tcp://localhost:5560",
            outputs="tcp://*:5562",
            model_path="./model.pth",
            confidence_threshold=0.30,
            forward_detection_rois=True
        )),

        (FilterCrop, dict(
            sources="tcp://localhost:5562",
            outputs="tcp://*:5564",
            detection_key="license_plate_detection",
            detection_roi_field="box",
            output_prefix="cropped_",
            mutate_original_frames=False,
            topic_mode="main_only"
        )),

        (FilterOpticalCharacterRecognition, dict(
            sources="tcp://localhost:5564",
            outputs="tcp://*:5566",
            topic_pattern="license_plate",
            forward_ocr_texts=True
        )),
```

```
        (FilterLicenseAnnotationDemo, dict(
            sources="tcp://localhost:5566",
            outputs="tcp://*:5568",
            cropped_topic_suffix="license_plate"
        )),

        (VideoOut, dict(
            sources="tcp://localhost:5568",
            outputs="file://./output_2_with_boxes.mp4",
            fps=True
        )),
    ])
```

Quick Troubleshooting Checklist

- Confirm model.pth exists in your pipeline folder.
- Run Webvis to confirm detections.
- If no detections, switch to a known plate demo clip.
- Use JSON options to inspect outputs when overlays aren’t present.