Adam Konrad

NVIDIA Jetson Developer Challenge 2017

OpenDVR

Makes your IP cameras smart

Overview

The typical IP camera system manages network and connected cameras. Such system usually just record video and provides a dashboard for access to a video archive for playback. These functionalities have already become a standard and bring little added value.

OpenDVR running on NVIDIA Jetson can provide affordable smart video analysis solutions. Utilization of deep learning-based video analysis provides high added value by sophisticated real-time data extraction like object detection, face detection and event detection.

Goals

1. **Network,** Device, and camera setup.
2. **Processing,** H264 video stream collection, processing, and storage.
3. **Deep learning,** intelligent video analysis using deep learning models, result retrieval, storage, and presentation.
4. **Access,** web-based, app-based access, and API access for 3rd party app integration.

Project status

1. **Network,** IP connection is made with a camera. Video stream and static image data is being captured.
2. **Processing,** Incoming H264 video frames and image data are passed to services for recording, streaming, or inference.
3. **Deep learning,** Captured image is used as input to FaceNet performing inference and outputs model data.
4. **Access,** Internal HTTP server provides access to data for clients.

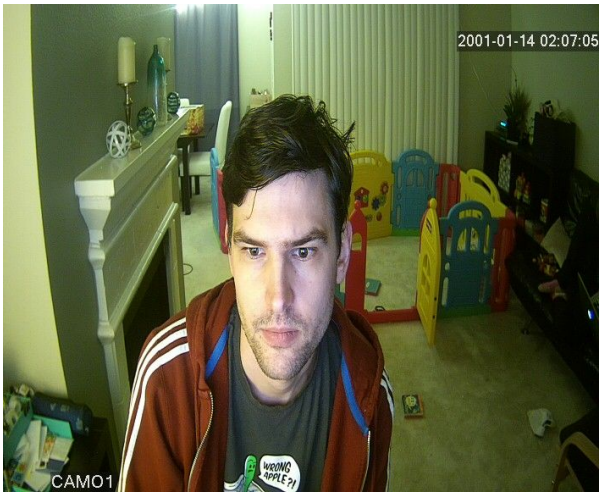
Result

OpenDVR successfully works and performs all previously described operations.

- Establishes IP Camera connection
- Retrieves configuration
- Streams video
- Records video
- Captures snapshots

Most importantly, it successfully performs inference on pre-trained FaceNet model. This allows extraction of facial recognition data from the IP camera video.

Output data is served from the internal web server inside the program to a client browser. Images below show the inference running in the program.



Applications

Deep learning video analysis brings the missing added value to the camera system. Image inference results can be used in security and home automation applications.

Example applications:

- Pedestrian detection in security perimeter (PedNet model)
- Parking lot detection and vehicle counting (ResNet model)
- Facial recognition in doorbell cameras (FaceNet model)

Data produced by the inference can be used by other systems to notify staff or owners about certain events.

Applications are also driving the choice of properly trained deep learning models.

Conclusion

OpenDVR successfully works and performs all previously described operations.

- Establishes IP Camera connection
- Retrieves configuration
- Streams video
- Records video
- Captures snapshots

Most importantly, it successfully performs inference on pre-trained FaceNet model. This allows extraction of facial recognition data from the IP camera video.

Internal web server provides data from inside the program to a client browser.

Future

What may be OpenDVR's future?

1. Deployment of more deep-learning models for exploration and testing
2. Currently the project uses pre-trained FaceNet model. Customized model training
3. Exploration of NVIDIA DeepStream SDK
4. Robust implementation of features
5. Better access to data from the web server

Implementation

Permanent project engineering goals

- Minimal dependencies
- Clear architecture
- Multithreading
- Automated tests
- Modern C++
- CMake build system

Architecture and program flow and related source files

The program starts, it detects and retrieves camera configuration. Relevant files are main.cpp, Camera.cpp, OnvifClient.cpp, RTSPClient.cpp, RTPClient.cpp.

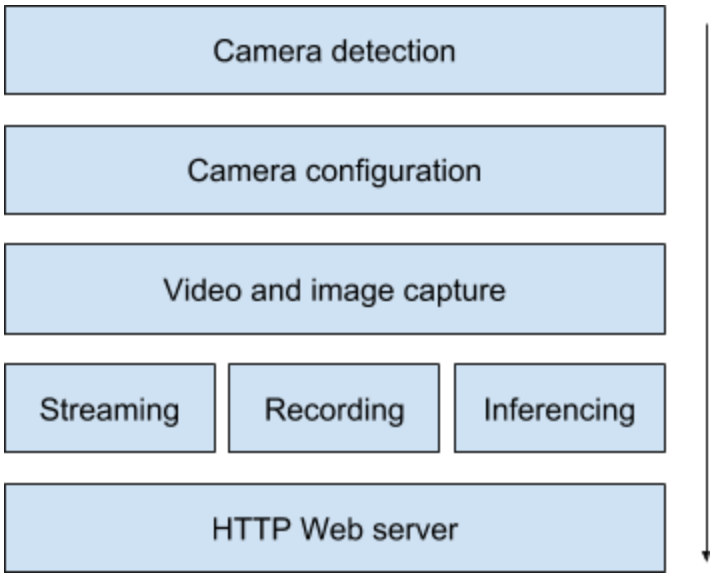
Based on configuration subscribes and runs services on camera stream, service interface is CameraService.hpp. This interface is implemented by individual services RecordMuxer.cpp for video recording, StreamMuxer.cpp for video streaming over HTTP, Inferencer.cpp deep learning controller. The inferencer internally uses detectNet.pp and tensorNet.cpp.

All services store current immutable chunk of data internally, represented by MediaContainer.cpp.

HTTP web server running in a separate thread accesses a specific service data and sends it down to the client. This is done in files http/Server.cpp, http/MediaCache.cpp and http/CameraController.cpp.

Program architecture diagram

Data flow is top to bottom.



Camera detection runs on the main thread. Once camera is detected new thread is launched for the camera.

Video capture is running from the camera thread. Image capture is launched from the camera thread and runs in it's own worker thread.

HTTP Web server is running in another separate thread and accesses data in the camera thread. Collection of this data produced by Streaming, Recording, and Inferencing services.

Build and testing

To build and test the project, your system must have latest versions of the following libraries installed (ideally from sources and with prefix /usr/local):

- LibAV
- cURL
- PugiXML
- MicroHTTPd
- CUDA
- Qt4

Project should successfully build using standard CMake process. The build process outputs two relevant binaries: src/opendvr and test/testopendvr

Symlink “networks” is needed in the “opendvr” binary directory, pointing to checked out repository https://github.com/dusty-nv/jetson-inference/tree/master/data/networks path.

To run the project, your IP camera must be accessible from the local network 192.168.1.0/24 and present at address 192.168.1.10. This address is currently hard coded as an input parameter in main.cpp.

Note: Specific camera settings depend on many factors, but the camera has to be on the same network like the Jetson.